Fondamenti di Informatica

AA 2019/2020

Eng. Ph.D. Michela Paolucci

DISIT Lab http://www.disit.dinfo.unifi.it
Department of Information Engineering, DINFO
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570
michela.paolucci@unifi.it

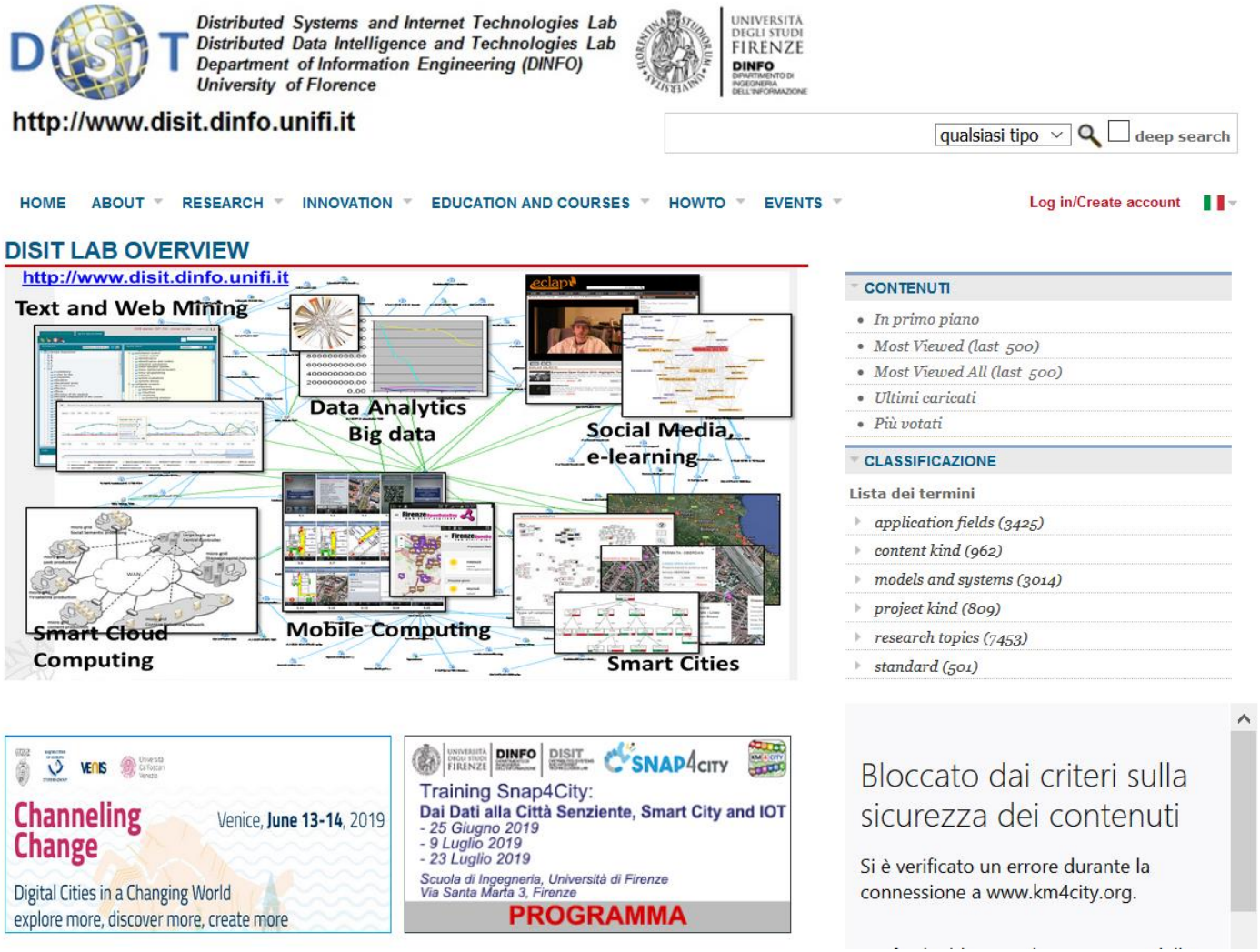
Disit Lab

Researchers: 20

Main research topics:

- Big Data, Smart Cities and Distributed Systems
- Smart City models and applications
- Collective Awarness Platforms
- Creative industry, creativity, elearning
- Smart cloud
- Smart manufacturing: Industry 4.0, Factory 4.0, e-factory
- Smart Retail: user behavior analysis, engagment, ..
- Autonomous drivers: operators, high speed trains, driverless

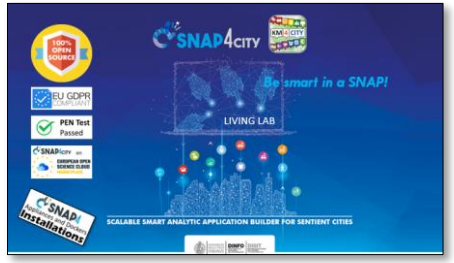
Disit Lab, http://www.disit.org/drupal/?q=it



Con chi lavoriamo

Progetti

- **Snap4city:** y EC project selected by **SELECT4Cities** pre-commercial procurement for the R&D of large-scale IoE/IoTplatforms for urban Innovation https://www.snap4city.org
- **Traffair:** **EC project as CEF**
- **MOSAIC, regional project with ALSTOM, Tages, LiberoLogico**
- **Weee Life:** **project of the EC with regione Toscana**
- **Feedback project:** co-funded by the Tuscany Region.
- **JOIN project:** co-Founded by the Tuscany Region
- **REPLICATE H2020, SCC1, EC flagship,** http://replicate-project.eu/
- **5G – Prato Wind Open Fiber**
- **RESOLUTE H2020, EC,** http://www.resolute-eu.org
- **Sii-Mobility SCN MIUR,** http://www.sii-mobility.org
- **Km4City:** http://www.km4city.org
- **Coll@bora Social Innovation, MIUR,** http://www.disit.org/5479
- **Trace-it**
- **Raiss**

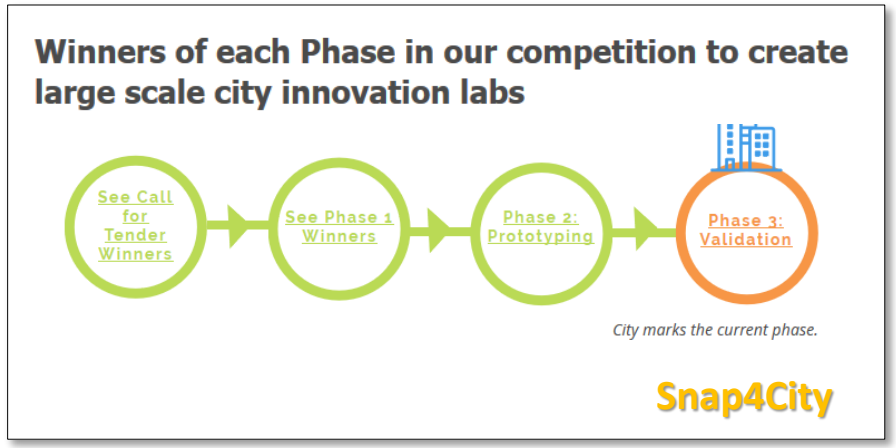


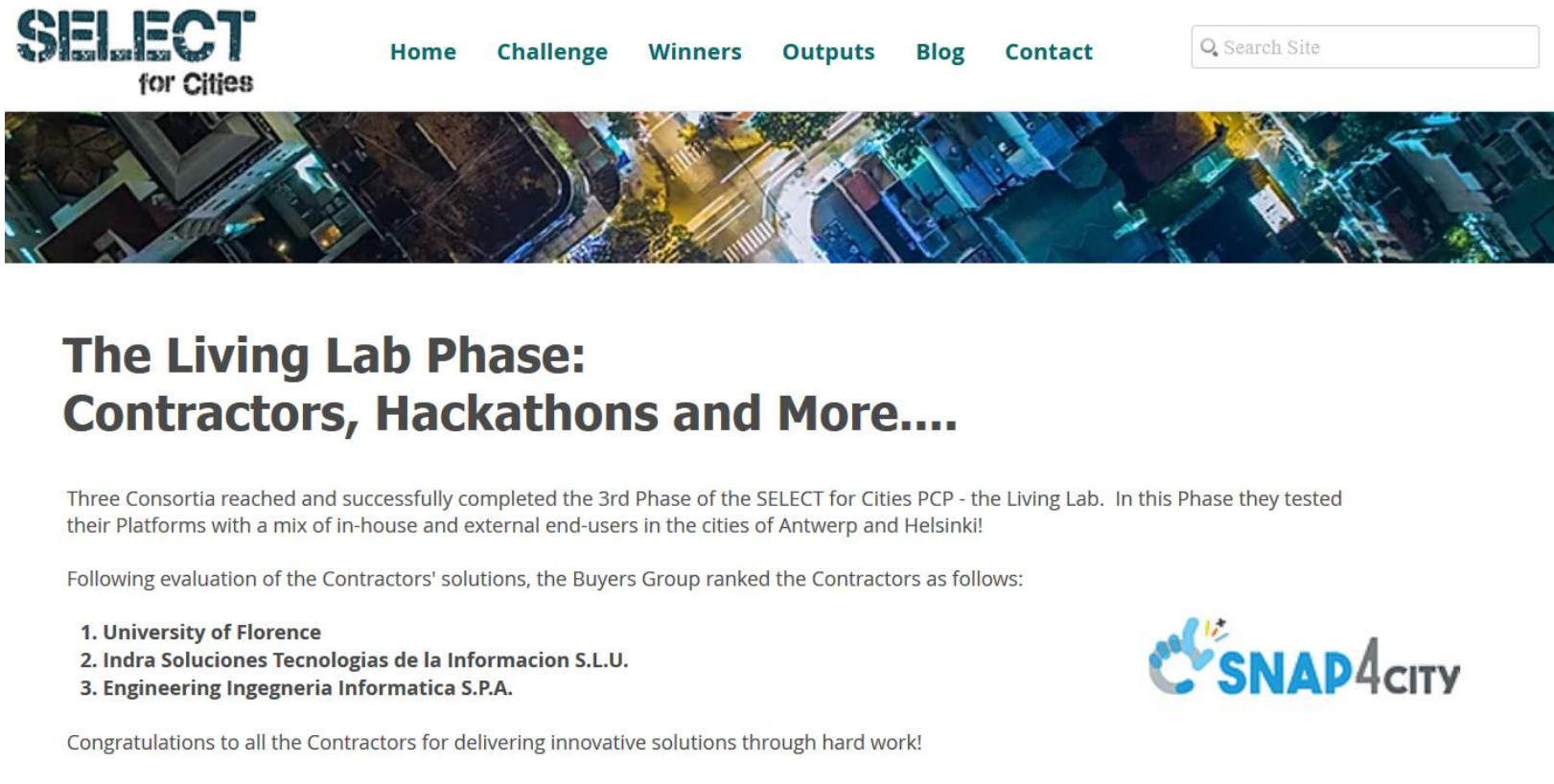
Snap4City

Snap4City has been proposed in response to **Select4Cities** challenges and PCP – Pre-Commercial Procurement Project (http://www.select4cities.eu/).

Smart Cities involved:

- Antwerp
- Copenaghen
- Helsinki





100% OPEN SOURCE

EU GDPR COMPLIANT

PEN Test Passed

SNAP4CITY on EUROPEAN OPEN SCIENCE CLOUD MARKETPLACE

SNAP4 Appliances and Dockers Installations

SNAP4CITY

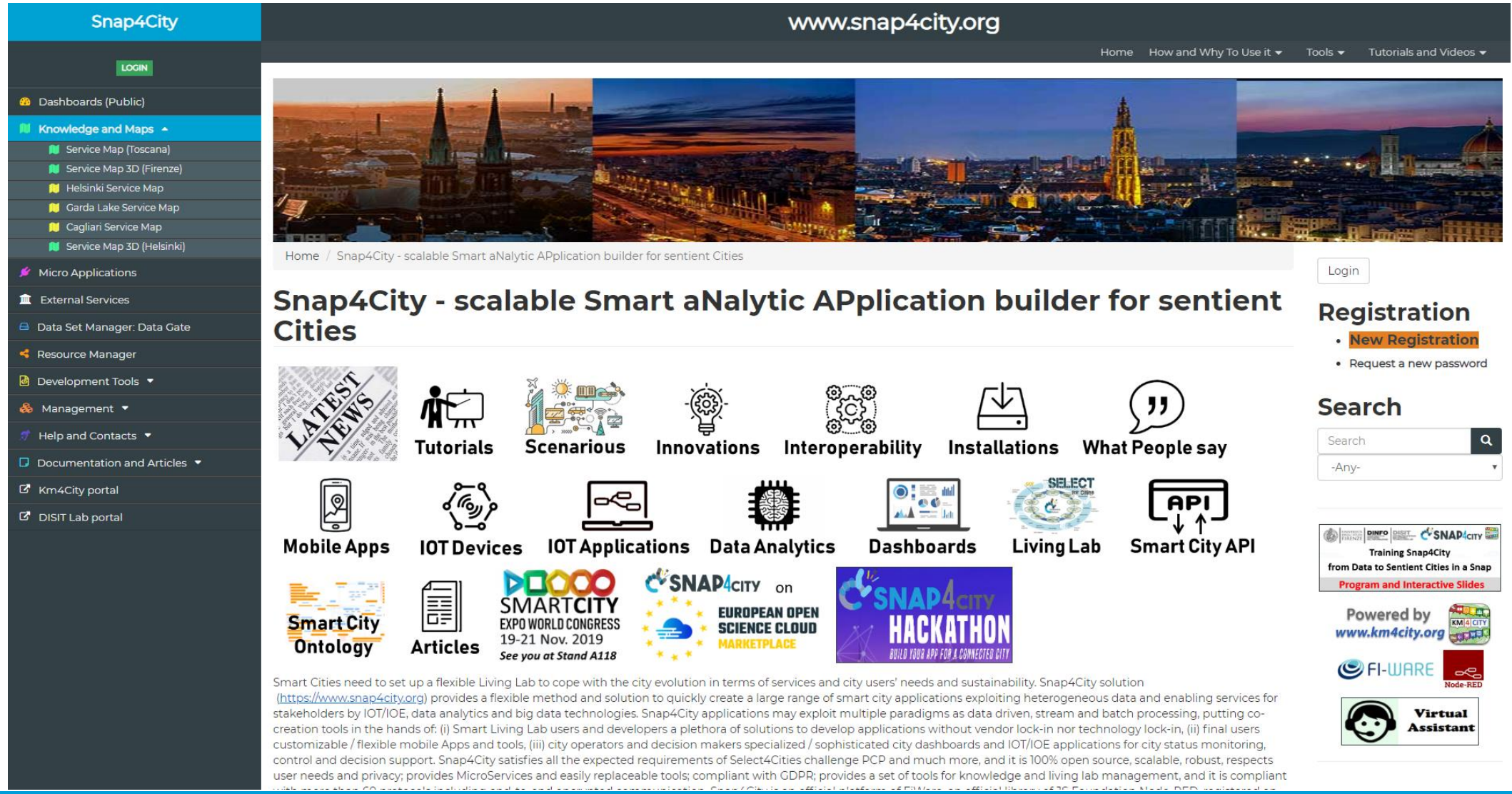
KM4CITY

Be smart in a SNAP!

SCALABLE SMART ANALYTIC APPLICATION BUILDER FOR SENTIENT CITIES

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DEPARTMENT OF INFORMATION ENGINEERING | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

https://www.snap4city.org



REPLICATE

REnaissance of PLaces with Innovative Citizenship And TEchnology

Smart Florence Plan

Demonstrate Smart City technologies in energy, transport and ICT in districts in:

- ***San Sebastian, Florence and Bristol,***
- ***follower cities of Essen, Nilufer and Lausanne***

Cities are the customer: considering local specificities

Solutions must be replicable, interoperable and scalable.

- *Integrated Infrastructure: deployment of ICT architecture, from internet of things to applications*
- *Low energy districts*
- *Urban mobility: sustainable and smart urban services*

Horizon 2020
European Union Funding
for Research & Innovation

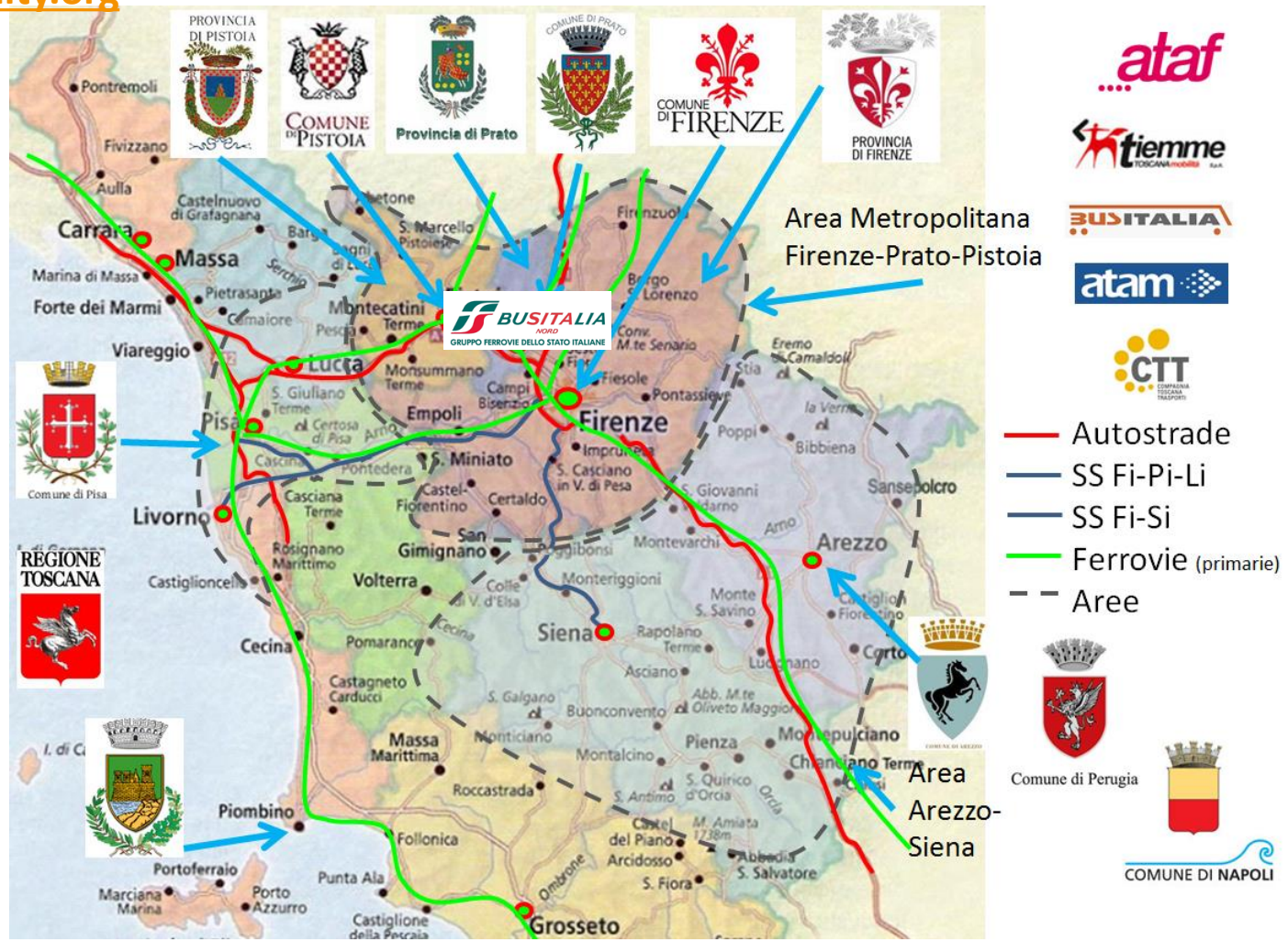
1 (coordinator) FOMENTO DE SAN SEBASTIAN FSS SPAIN
2 AYUNTAMIENTO DE SAN SEBASTIAN SAN SEBASTIAN SPAIN
3 COMUNE DI FLORENCE FLORENCE ITALY
4 BRISTOL COUNCIL BRISTOL UNITED KINGDOM
5 STADT ESSEN ESSEN GERMANY
6 NILUFER BELEDIYESI NILUFER TURKEY
7 VILLE DE LAUSANNE LAUSANNE SWITZERLAND
8 IKUSI ANGEL IGLESIAS, S.A. IKUSI SPAIN
9 ENDESA ENERGÍA, S.A. ENDESA SPAIN
10 EUROHELP CONSULTING, S.L. EUROHELP SPAIN
11 ILUMINACION INTELIGENTE LUIX, S.L. LUIX SPAIN
12 FUNDACION TECNALIA RESEARCH & INNOVATION TECNALIA SPAIN
13 EUSKALTEL, S.A. EUSKALTEL SPAIN
14 COMPAÑÍA DEL TRANVÍA DE SAN SEBASTIÁN DBUS SPAIN
15 CONSIGLIO NAZIONALE DELLE RICERCHE CNR ITALY
16 ENEL DISTRIBUZIONE, SPA ENEL ITALY
17 MATHEMA, SRL MATHEMA ITALY
18 SPES CONSULTING SPES ITALY
19 TELECOM ITALIA, SPA TELECOM ITALY
20 UNIVERSITA DEGLI STUDI DI FLORENCE UNIFI ITALY: DINFO.DISIT, DIEF
21 THALES ITALIA, SPA THALES ITALY
22 ZABALA INNOVATION CONSULTING ZABALA SPAIN
23 TECHNOMAR TECHNOMAR GERMANY
24 UNIVERSITY OF BRISTOL UOB UNITED KINGDOM
25 UNIVERSITY OF OXFORD UOXF UNITED KINGDOM
26 BRISTOL IS OPEN, LTD BIO UNITED KINGDOM
27 ZEETTA NETWORKS ZEETTA UNITED KINGDOM
28 KNOWLE WEST MEDIA CENTRE, LGB KWMC UNITED KINGDOM
29 TOSHIBA RESEARCH EUROPE, LTD TREL UNITED KINGDOM
30 ROUTE MONKEY, LTD ROUTE MONKEY UNITED KINGDOM
31 ESOTERIX SYSTMES, LTD ESOTERIX UNITED KINGDOM
32 NEC LABORATORIES EUROPE, LTD NEC UNITED KINGDOM
33 COMMONWHEELS CAR CLUB CIC CO-WHEELS UNITED KINGDOM
34 UNIVERSITY OF THE WEST OF ENGLAND UWE UNITED KINGDOM
35 ESADE BUSINESS SCHOOL ESADE SPAIN
36 SISTELEC SOLUCIONES DE TELECOMUNICACION, S.L. SISTELEC SPAIN

REPLICATE

Sii-Mobility

http://www.Sii-Mobility.org

- Experimentations and validation in Tuscany
- Integration with present central station and subsystems
- DISIT lab, Università di Firenze, is the coordinator



ECM; **Swarco Mizar**; Inventi In20; Geoin; QuestIT; Softec; T.I.M.E.; LiberoLogico; **MIDRA (autostrade, motorola)**; ATAF; Tiemme; CTT Nord; BUSITALIA; A.T.A.M.; Effective Knowledge; eWings; Argos Engineering; Elfi; Calamai & Agresti; Project; Negentis

Horizon 2020
European Union Funding
for Research & Innovation

http://www.resolute-eu.org

- **Develop European Resilience Management Guidelines (ERMG)**
  - Develop a conceptual framework for creating/ maintaining Urban Transport Systems
- **Enhance resilience through improved support of human decision making processes, particularly by training professionals and civil users on the ERMG and the RESOLUTE system**
- **Operationalize and validate the ERMG by implementing the RESOLUTE Collaborative Resilience Assessment and Management Support Systems (CRAMSS) for Urban Transport Systems addressing Road and Urban Rail Infrastructures**
  - **Pilots in Florence and Athens**
- **Adoption of the ERMG at EU and Associated Countries level**

University of Florence: DISIT lab DINFO (Proj coordinator), DISIA and DST	UNIFI	IT
THALES	THALES	IT
ATTIKOMetro	ATTIKO	GR
Comune di Firenze	CDF	IT
Centre for Research and Technology Hellas	CERTH	GR
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	FHG	DE
HUMANIST	HUMANIST	FR
SWARCO Mizar	SWMIZ	IT
Associação para o Desenvolvimento da Investigação no Instituto Superior de Gestão	ADI-ISG	PT
Consorzio Milano Ricerche	CMR	IT

http://servicemap.km4city.org

What is enabling and providing smart services

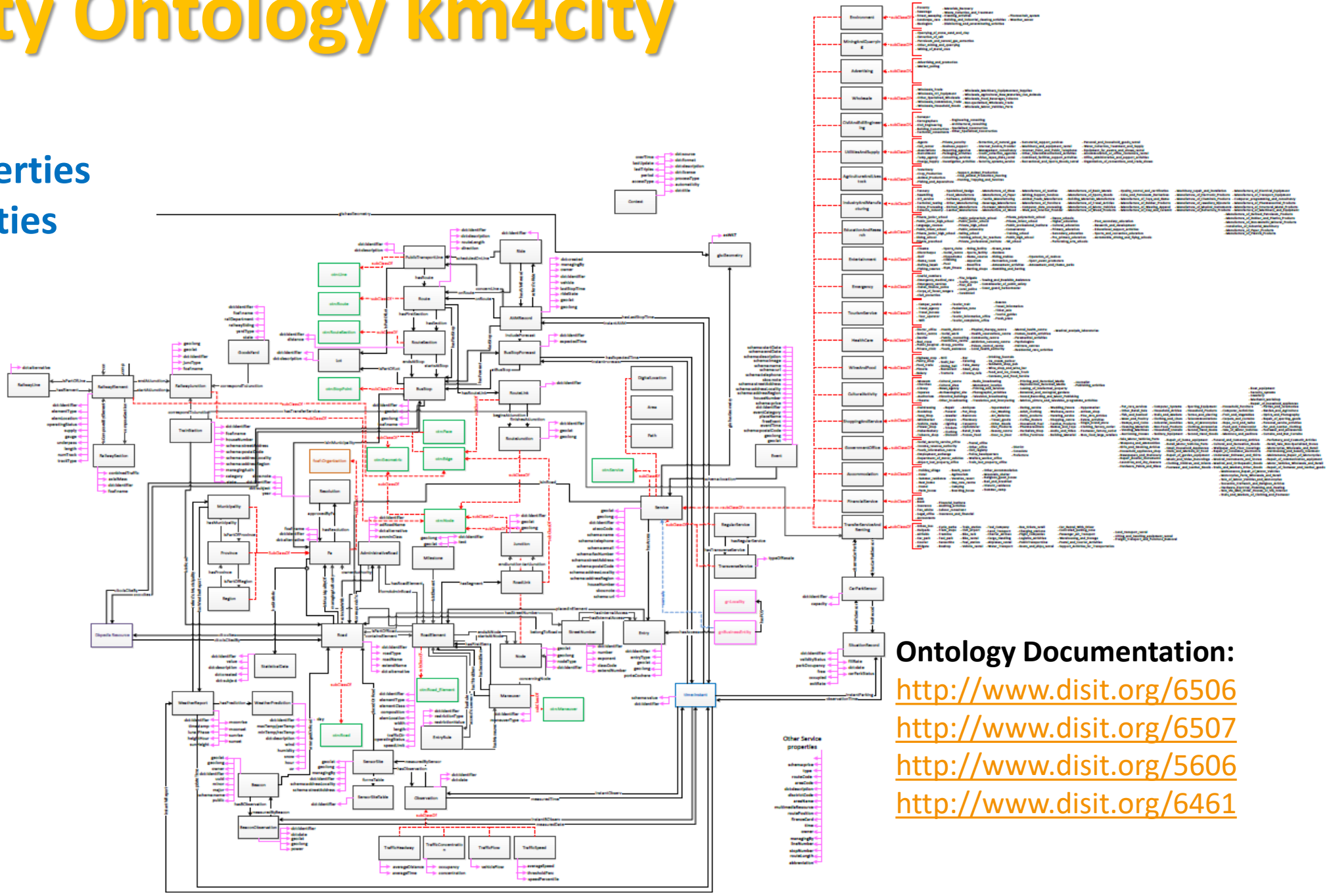
- **Smart Parking, in Tuscany**
- **Smart First Aid in Tuscany**
- **Smart Fuel pricing in Tuscany**
- **Smart search for POI and public transport srv.**
- **Public Transportation in Tuscany**
- **Routing in Tuscany, multimodal routing**
- **Social Media Monitoring and acting**
- **Traffic events and Resilience in Florence**
- **Bike Sharing in Pisa and Siena**
- **Recharge stations for e-vehicles**
- **Entertainment Events in Florence**
- **Traffic Sensors in Tuscany**
- **Weather forecast/condition in Tuscany**
- **Pollution and Pollination in Tuscany**
- **Origin Destination Matrices, trajectories**
- **People Monitoring Assessment in the City, in Florence via WiFi**
- **People Monitoring, in Tuscany via App**

All Point of Interests, cultural activities, IOT, …

Over than 1.2 Million of complex events per day!

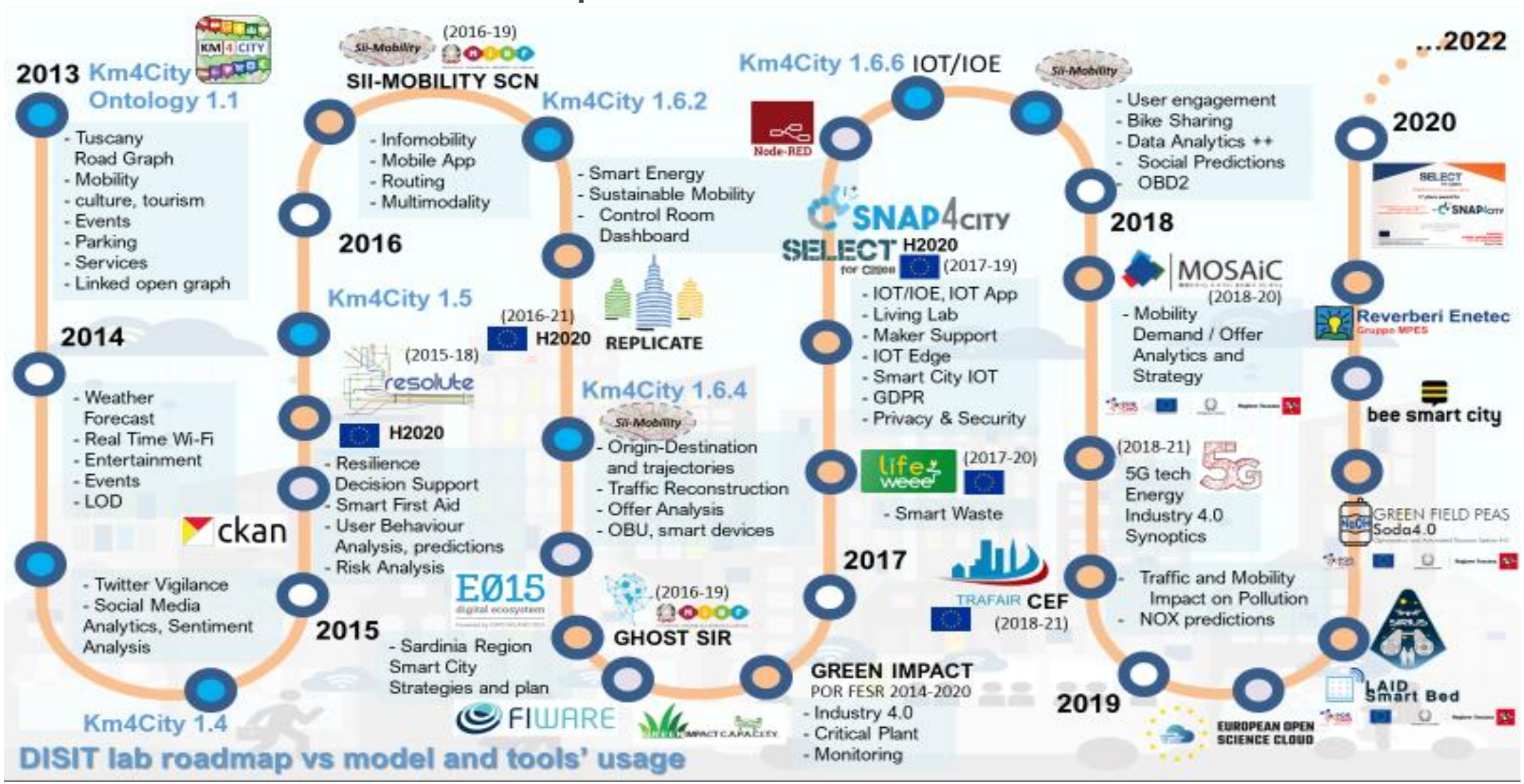
Smart-city Ontology km4city

>84 Classes
>100 ObjectProperties
>100 DataProperties



Ontology Documentation:
http://www.disit.org/6506
http://www.disit.org/6507
http://www.disit.org/5606
http://www.disit.org/6461

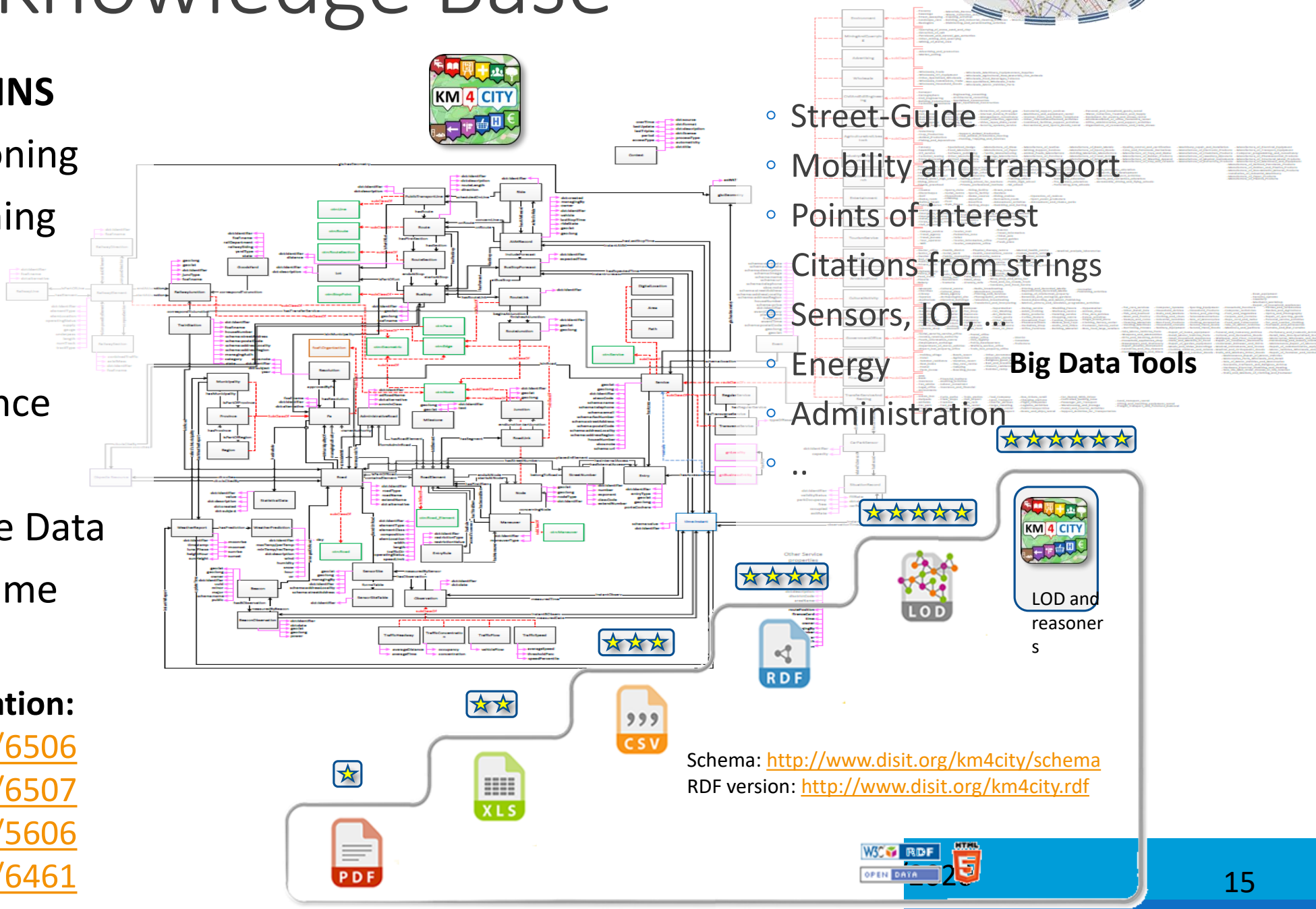
Disit Lab Roadmap



Km4City: Knowledge Base

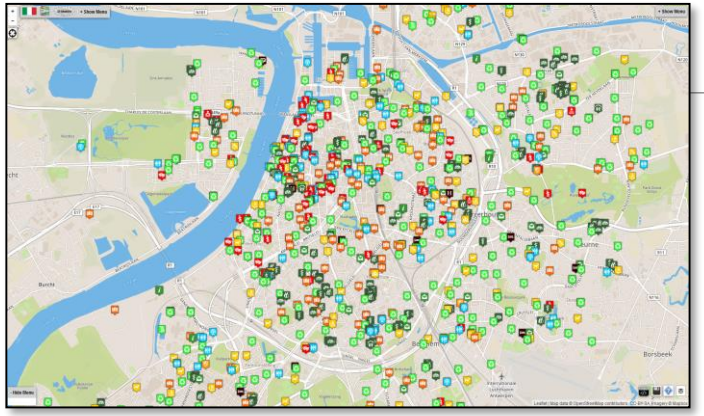
- **Multiple DOMAINS**
- Geospatial reasoning
- Temporal reasoning
- Metadata
- Statistics
- Risk and Resilience
- Licensing
- Open and Private Data
- Static and Real time
- IOT/IOE

Ontology Documentation:
http://www.disit.org/6506
http://www.disit.org/6507
http://www.disit.org/5606
http://www.disit.org/6461

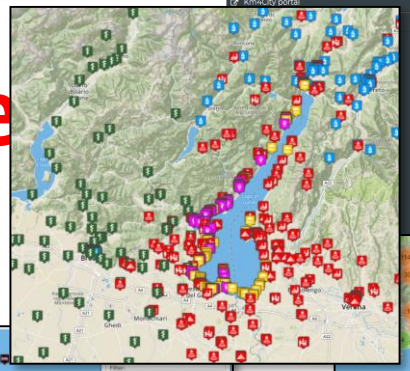


Km4City in …

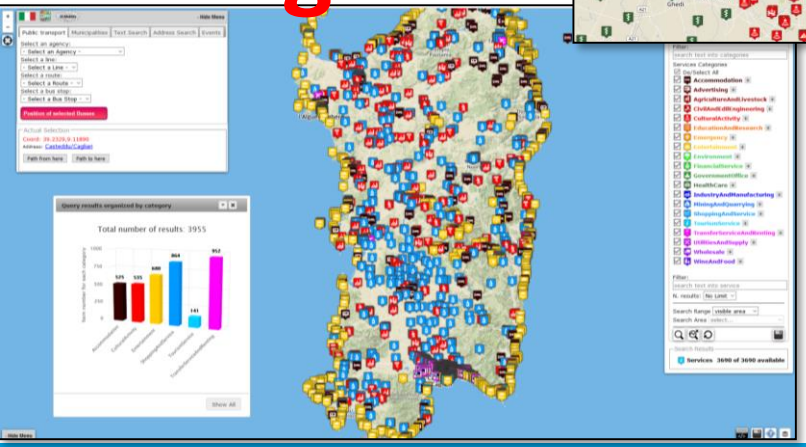
Antwerp



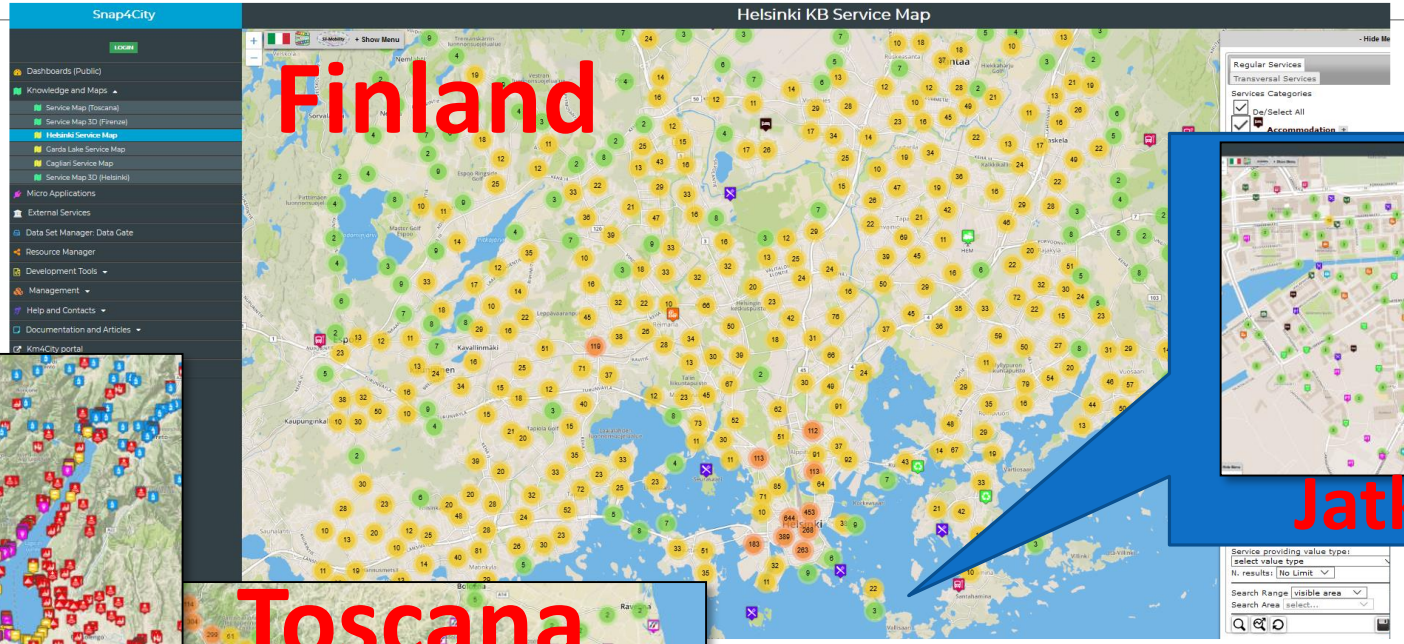
Garda Lake

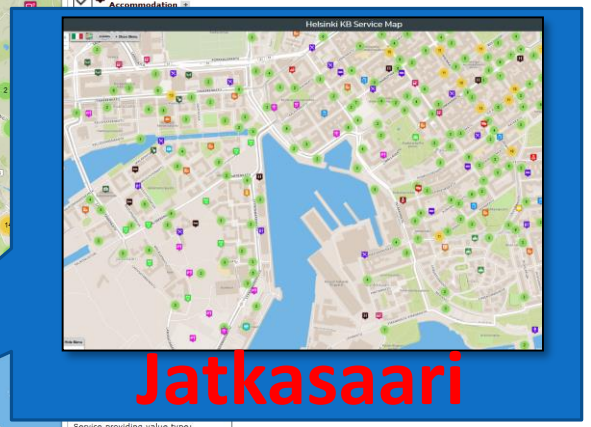


Sardegna



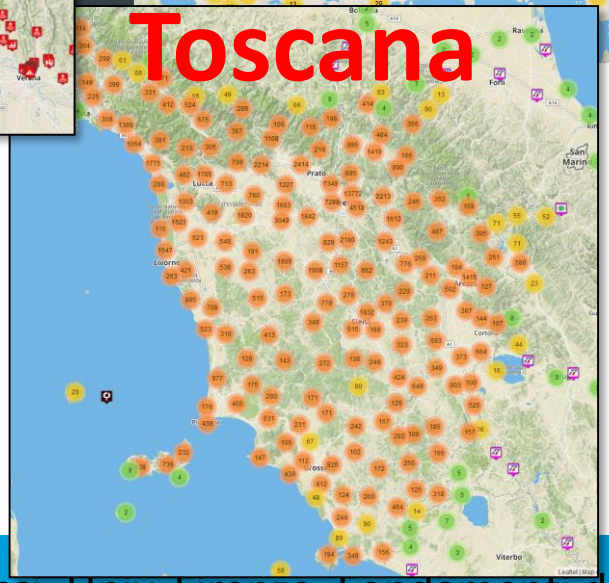
Finland



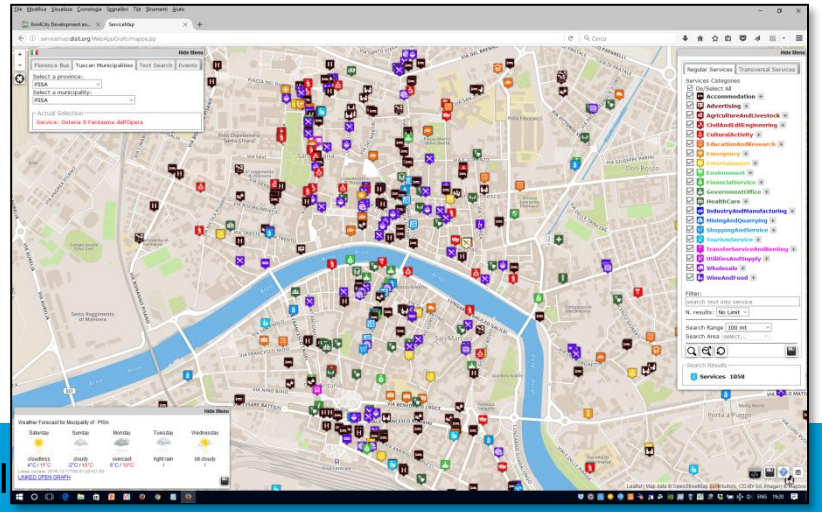


Jatkasaari

Toscana



Pisa



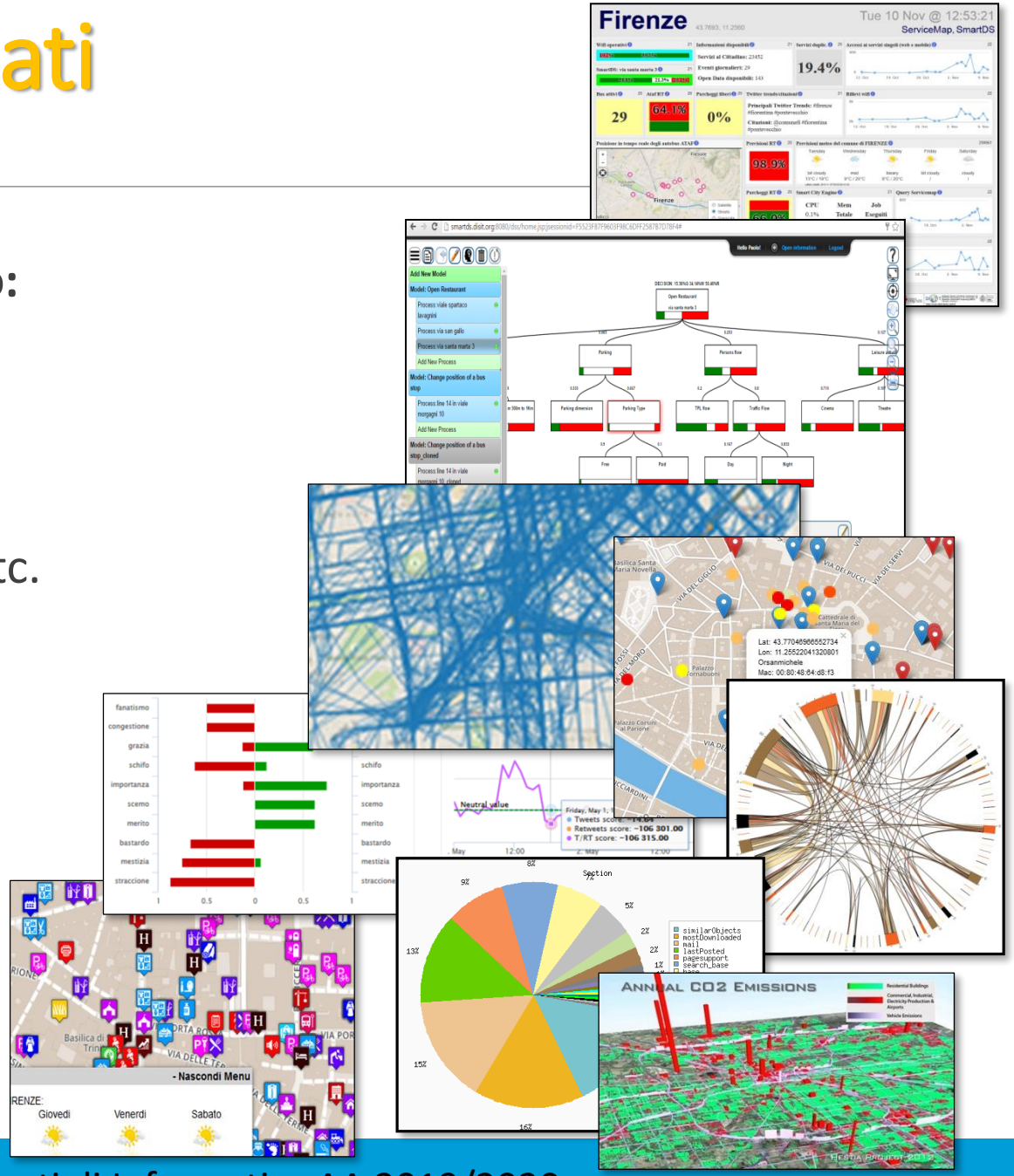
Decisioni supportate dai dati
periodiche ed in tempo reale

Condivisione e Integrazione Dati multi-dominio:
semantica e bigdata

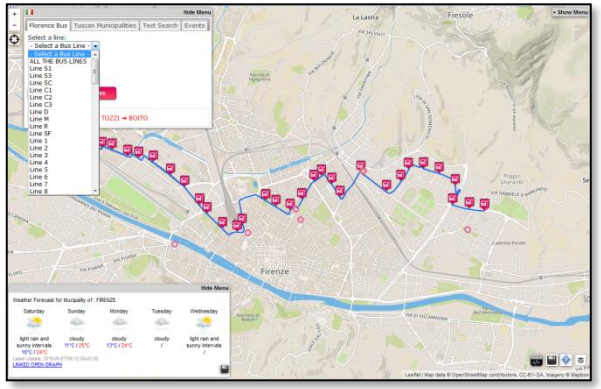
Dati →Smart City Engine → Control Room

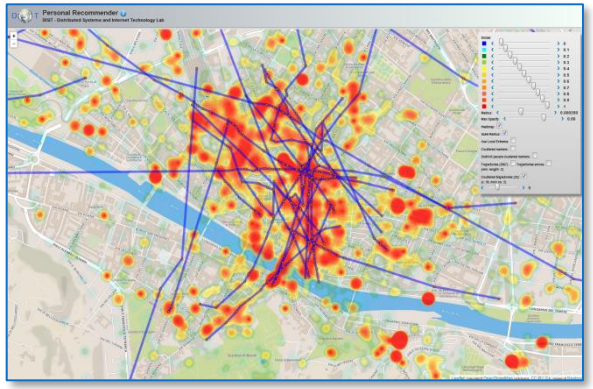
analisi: monitoraggio, flussi e comportamenti, sondaggi, mining, correlazioni, cause – effetti, etc.

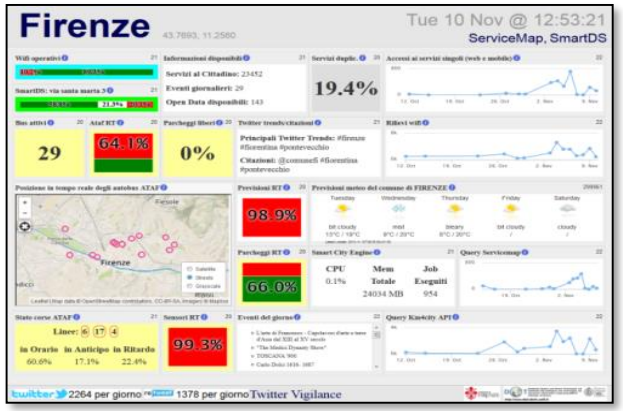
- Per il miglioramento di servizi correnti
- Per reagire ad eventi, incremento della resilienza,
- Per la creazione servizi innovativi
- …

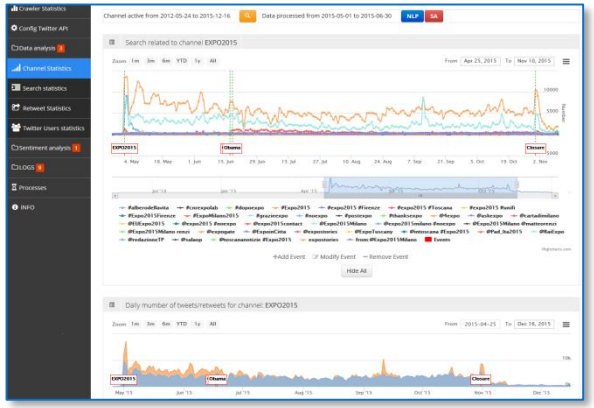


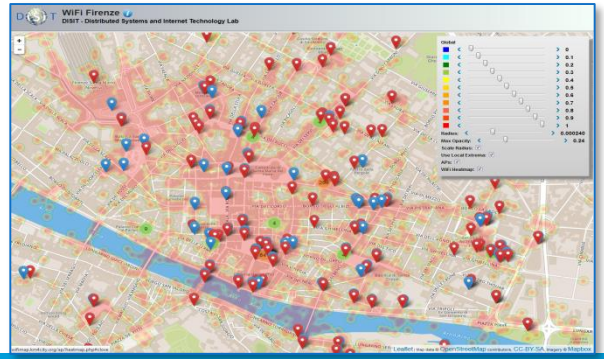
Smart City Dashboards

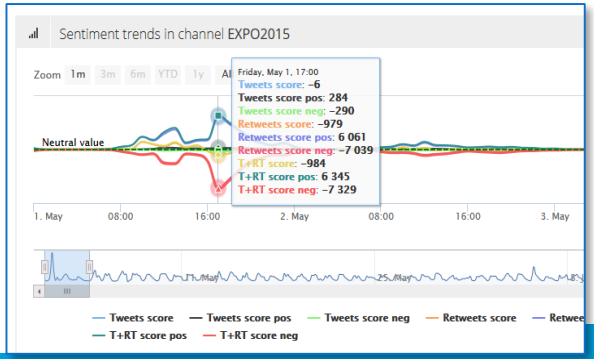








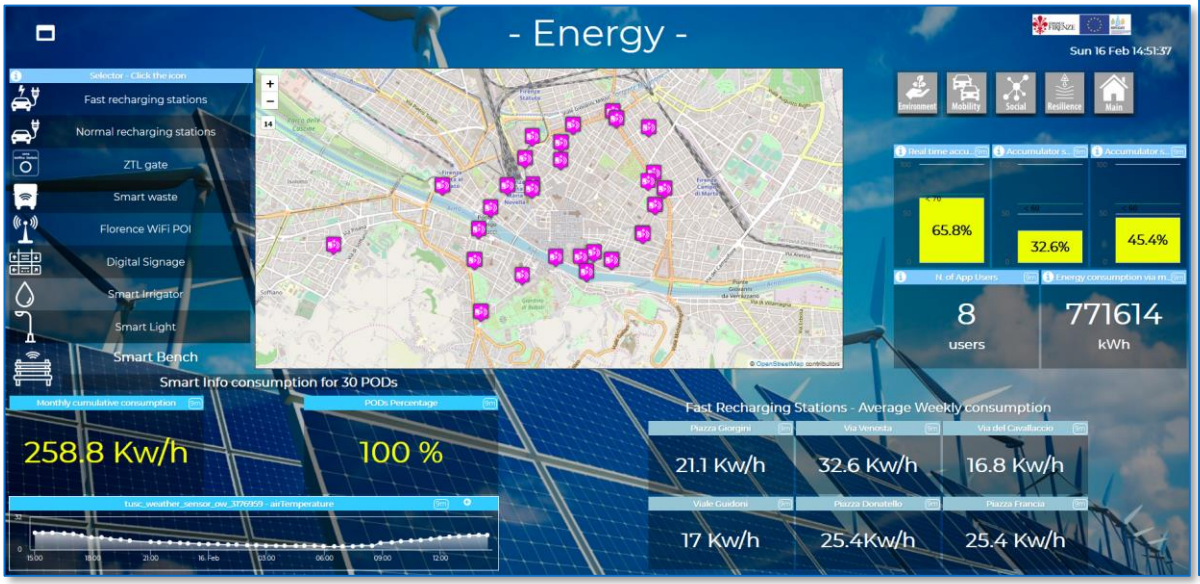




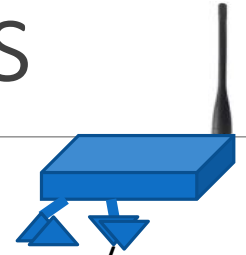
Smart City Dashboard

https://www.snap4city.org/dashboardSmartCity/view/index.php?iddasboard=OTM5

https://main.snap4city.org/view/index.php?iddasboard=OTQy



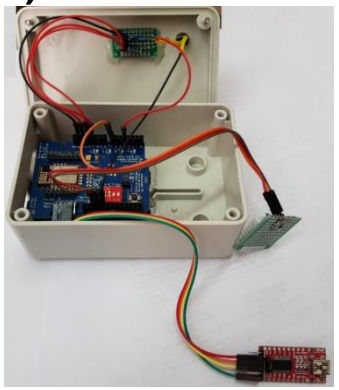
IOT Devices | IOT Edge Devices



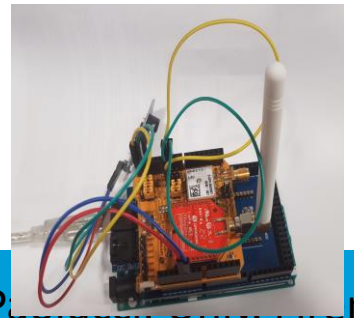
Sensors/
Actuators

SigFOX
Any and
Arduino

Arduino, Wi-
Fi, NGSI



LoraWAN +
Arduino +
I2C, NGSI

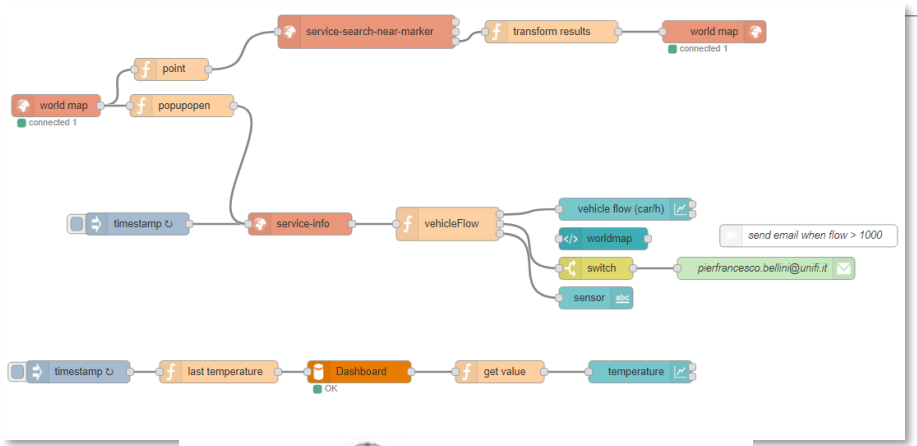


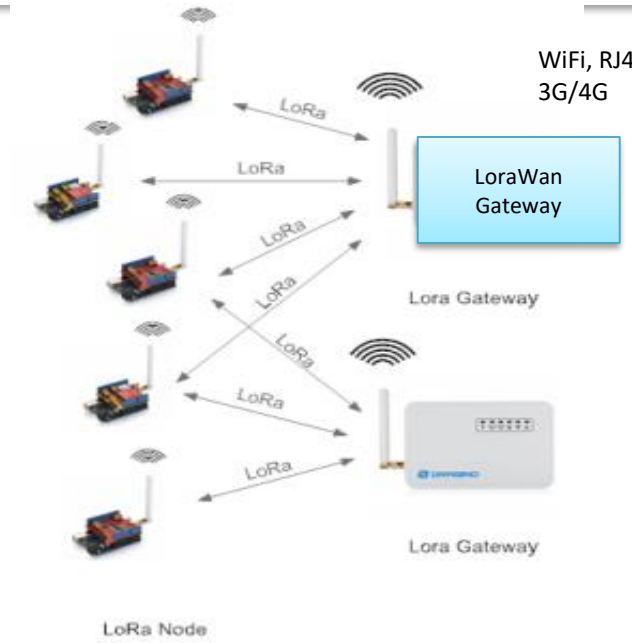
IOT Edge
NodeRED:
Raspberry

Snap4All IOT
Button ESP

IOT Edge
NodeRED:
Android,
LINUX,
Windows

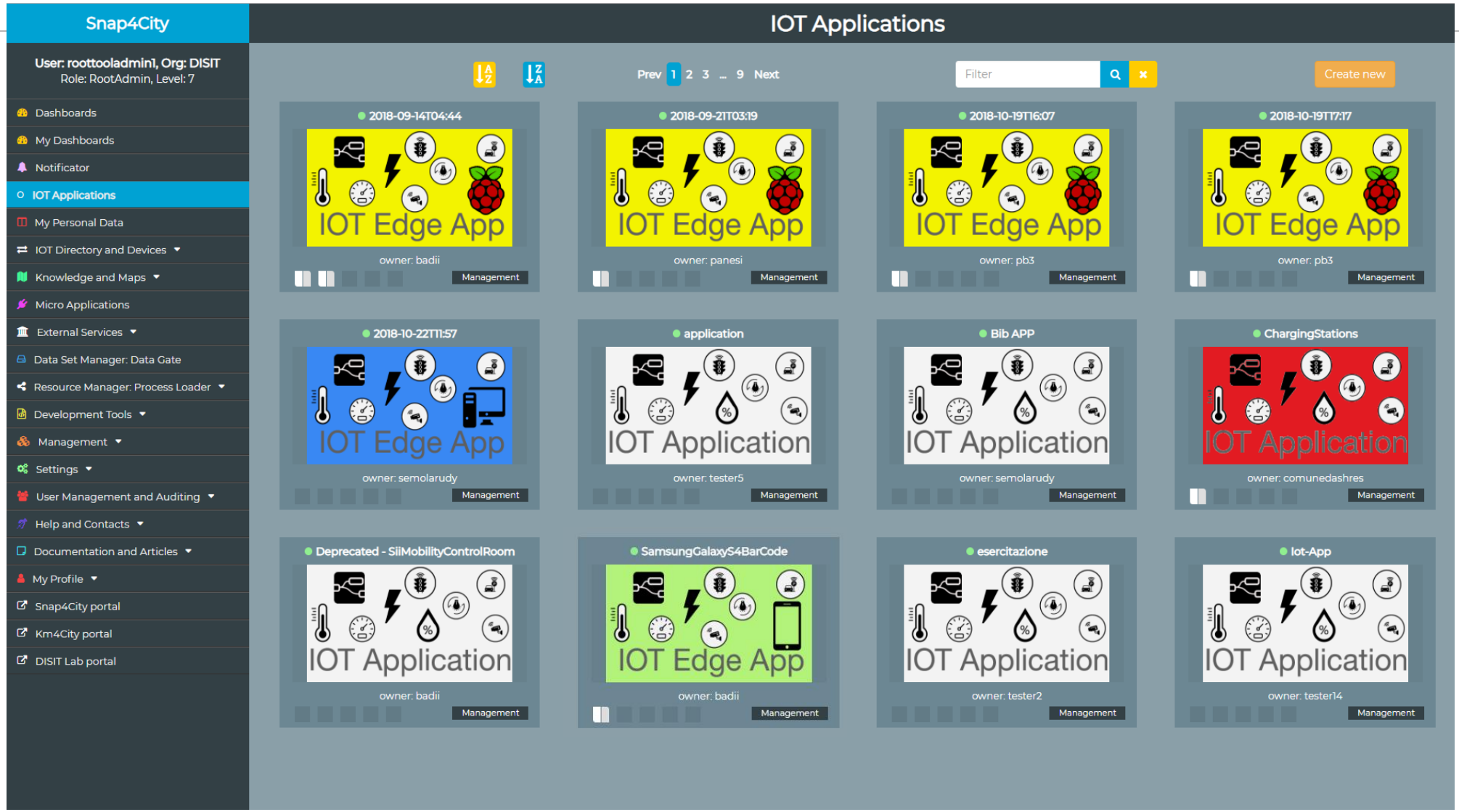
Internet of Things





WiFi, RJ45
3G/4G

IoT Applications



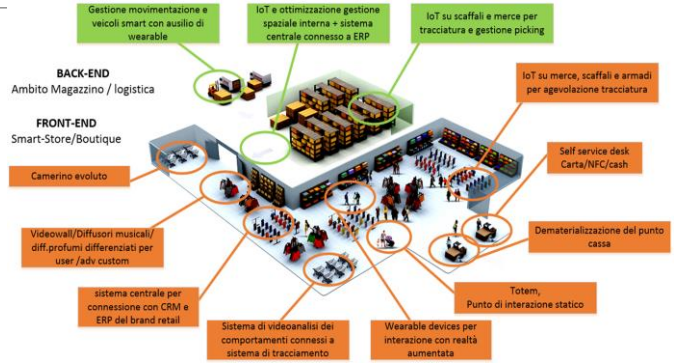
Smart Retail and Human monitoring/engaging

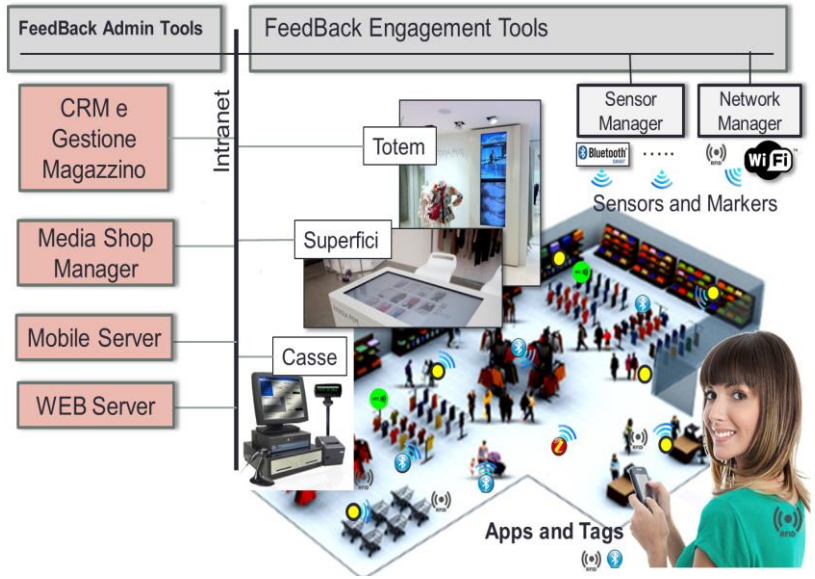
Feedback Project, from Feb 2017

- Flexible Advanced Engagement Exploiting User Profiles and Product/Production Knowledge
- VAR, Patrizia Pepe (Tessilform), DISIT, Effective Knowledge, SICE
- Keywords: retail, GDO, …

Goals and drivers:

- adaptive user engagement, customer experience
- Advanced user profiling, user behavior analysis
- Predictive models for engagement
- IOT and instrumentation
- Integrated incity customer experience





2014 2020 POR CreO PROGRAMMA OPERATIVO REGIONALE CRESCITA e OCCUPAZIONE
Fondo europeo di sviluppo regionale (Fesr)
Le ali alle tue idee

Smart Manufactory

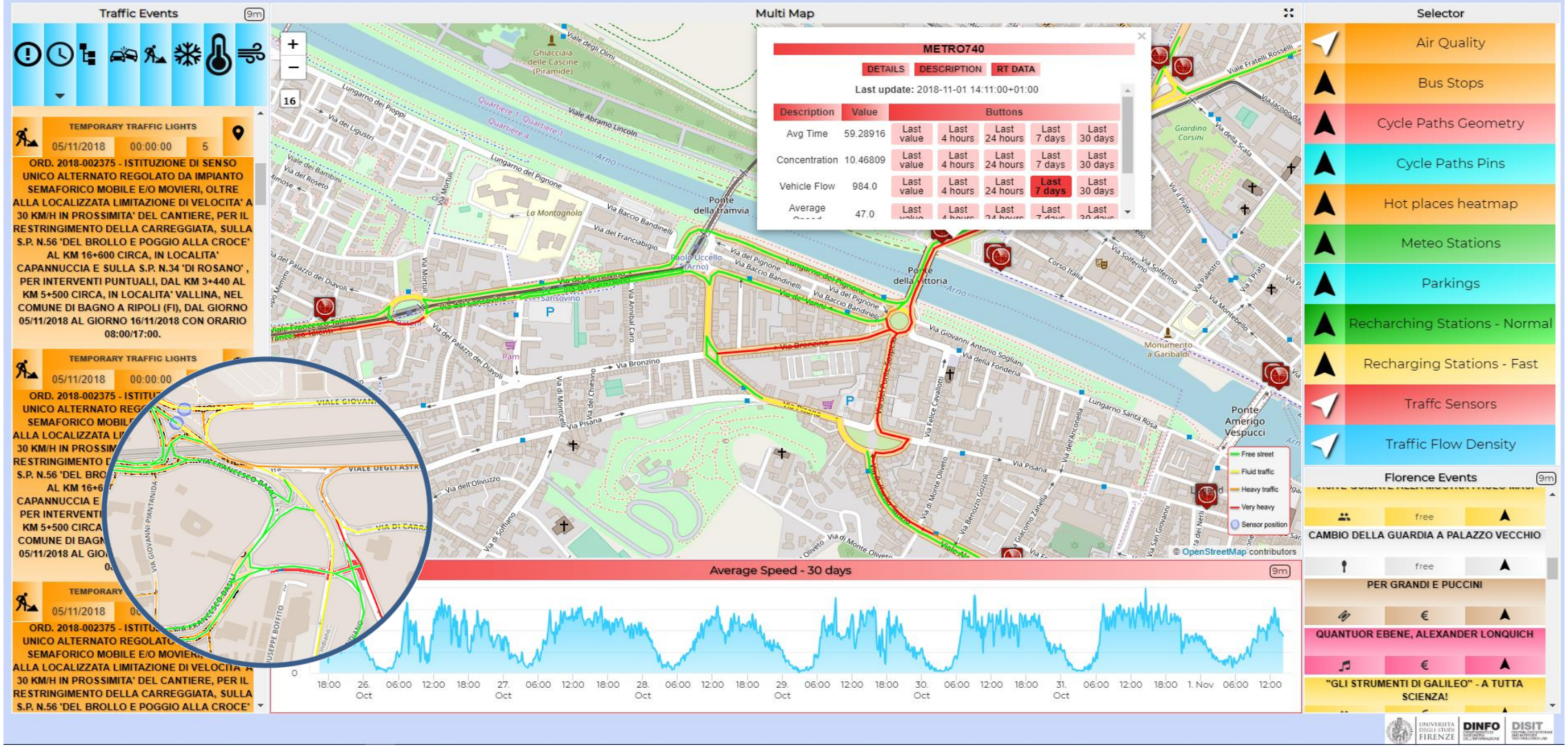
Riduzione costi e incremento efficienza

- **Automazione** della manutenzione e della produzione
- **Navigazione** indoor – Outdoor integrata
- **Ottimizzazione** flussi per utensili, pezzi e materiali

Progetti Regionali con PMI e GI

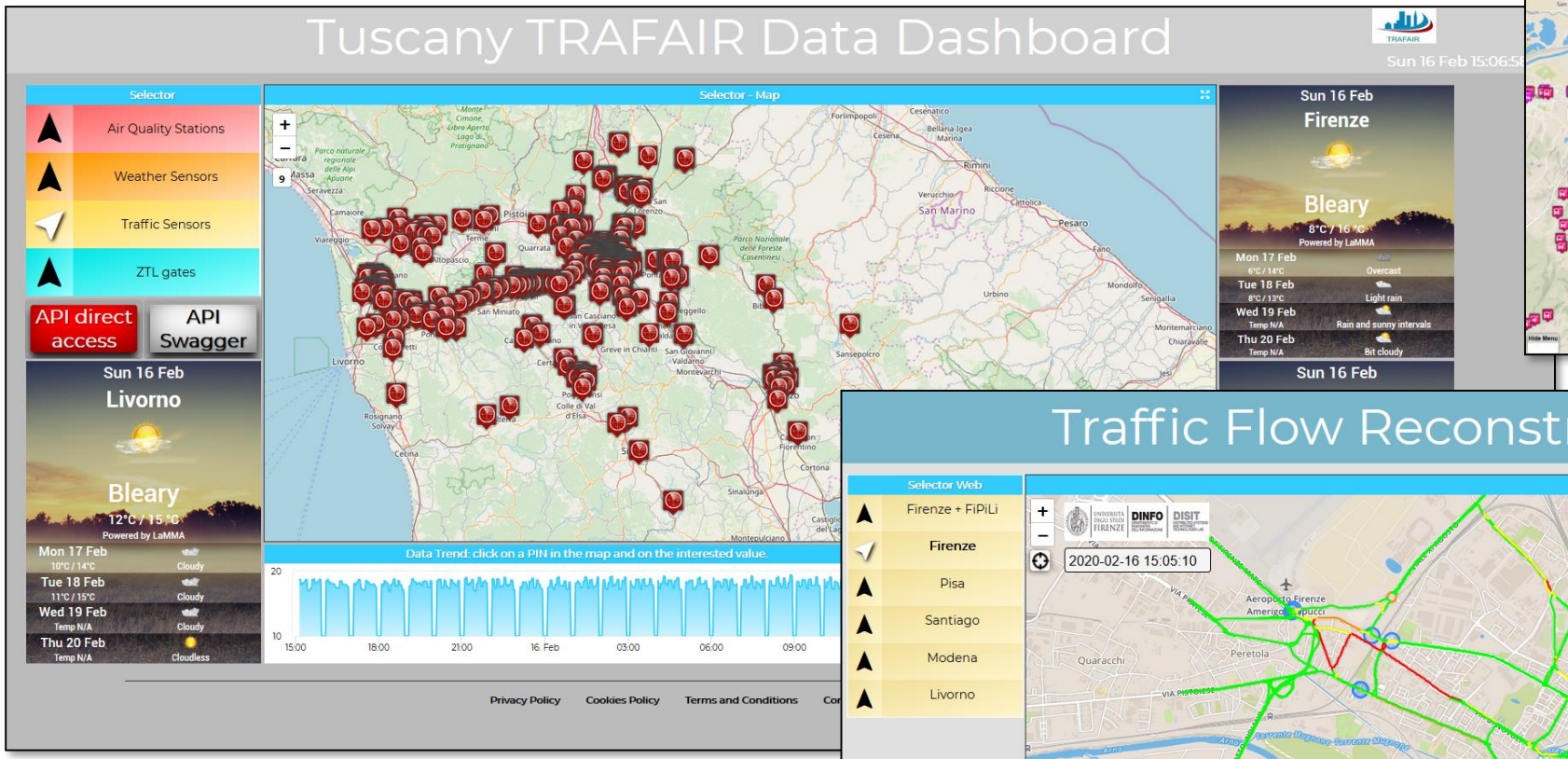
- **Frontman (Novicrom)**
- **Green Capacity (ALTAIR)**
- **Smart bed (Materassificio Montalese)**

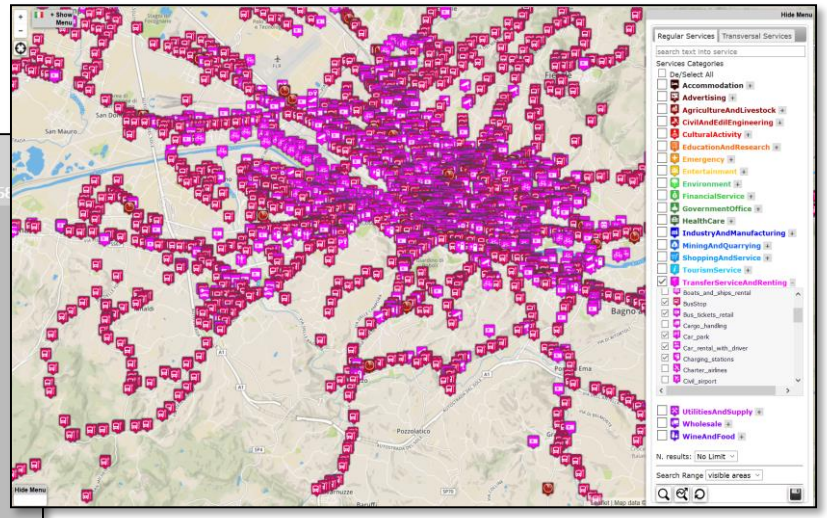
Toscana Traffico

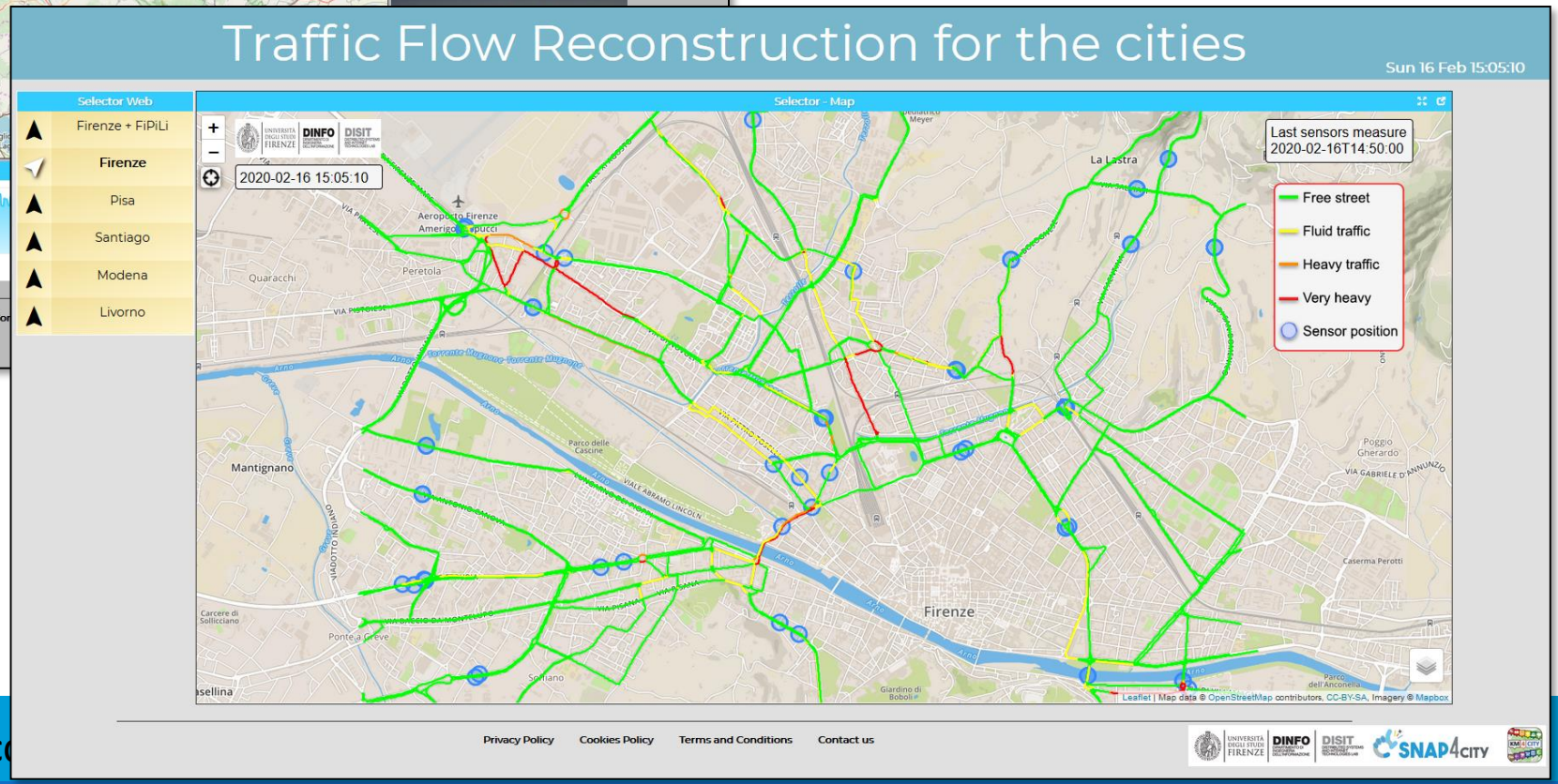


https://main.snap4city.org/view/index.php?iddasboard=MTE5MQ==

Traffic Flow Tools





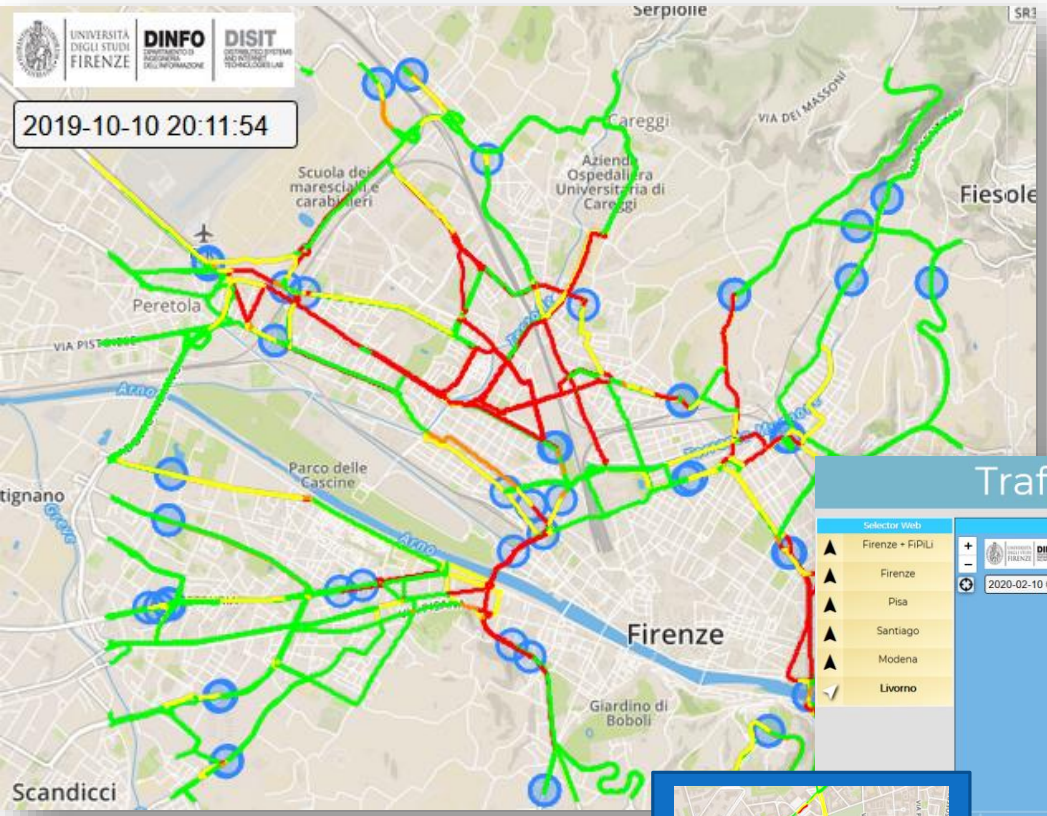


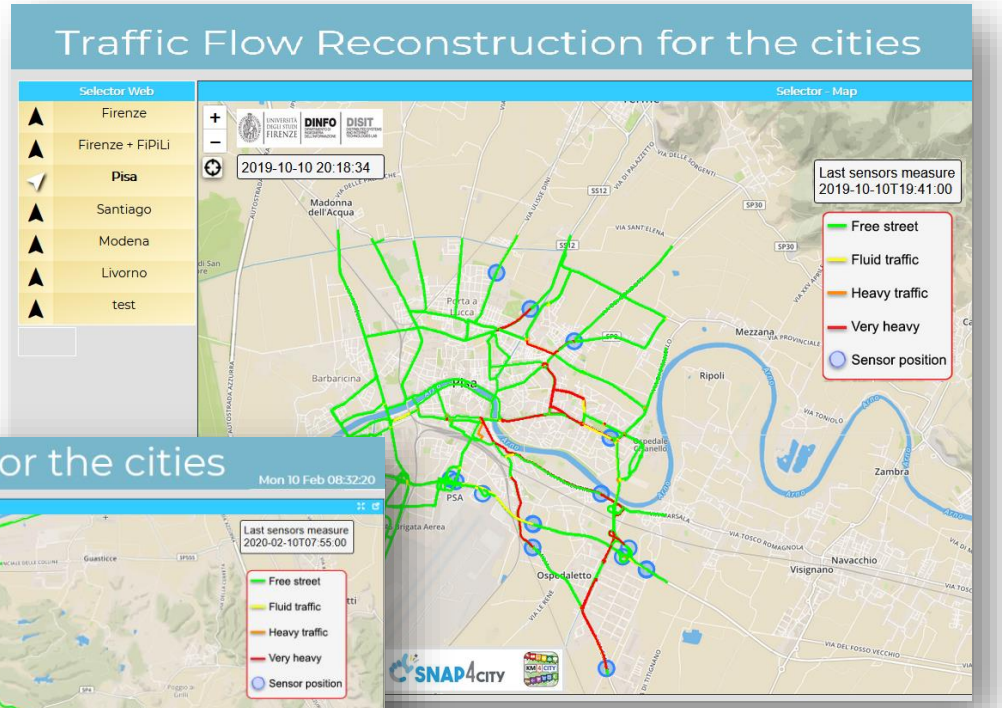
Disit RT Traffic Model Dashboards: Firenze, Pisa, Livorno

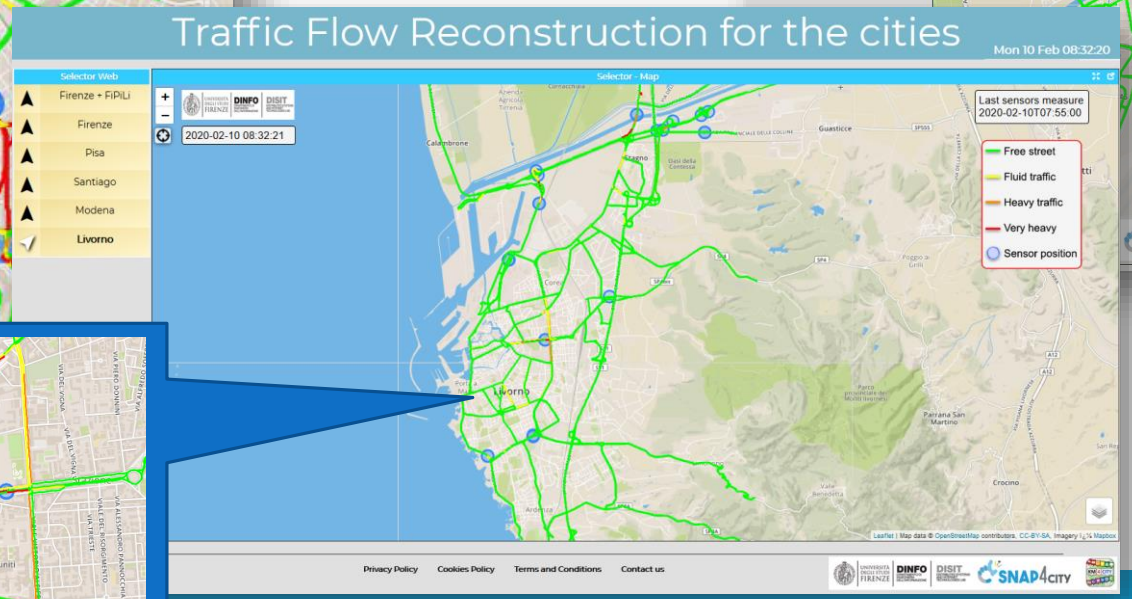
https://www.snap4city.org/dashboardSmartCity/view/index.php?iddasboard=MTc5NQ==

Pisa

Firenze





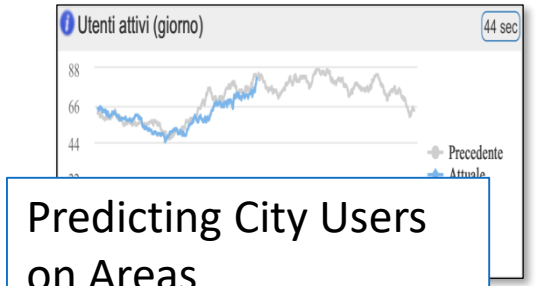


Livorno

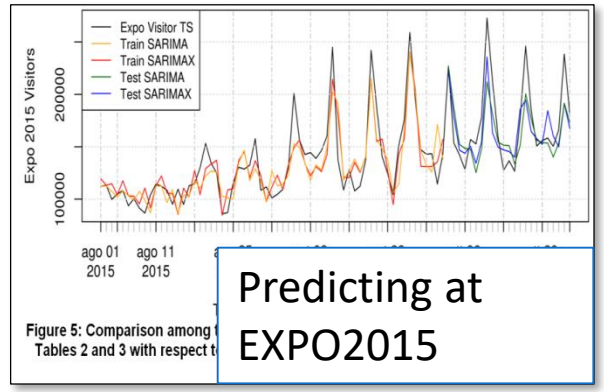
Analysing: Predicting, detection

Aiming at managing

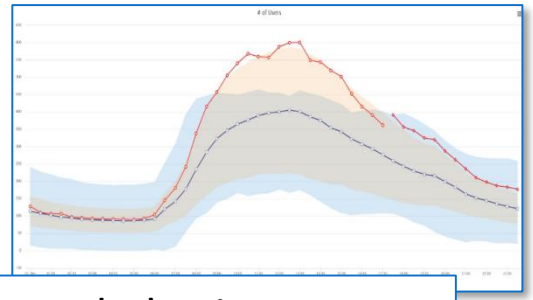
- Appreciation
- User relationships
- quality of service
- workload
- early warning/detection
- Dysfunction
- Habitudes



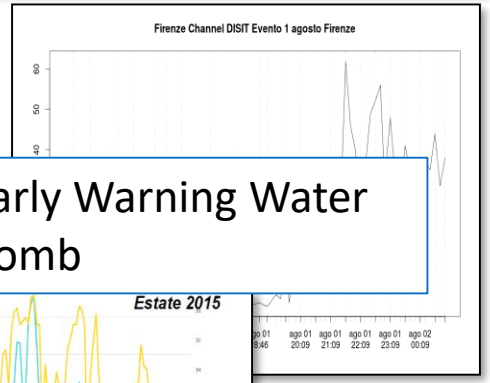
Predicting City Users on Areas



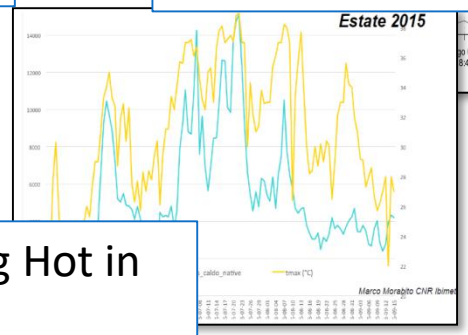
Predicting at EXPO2015



User behaviuor

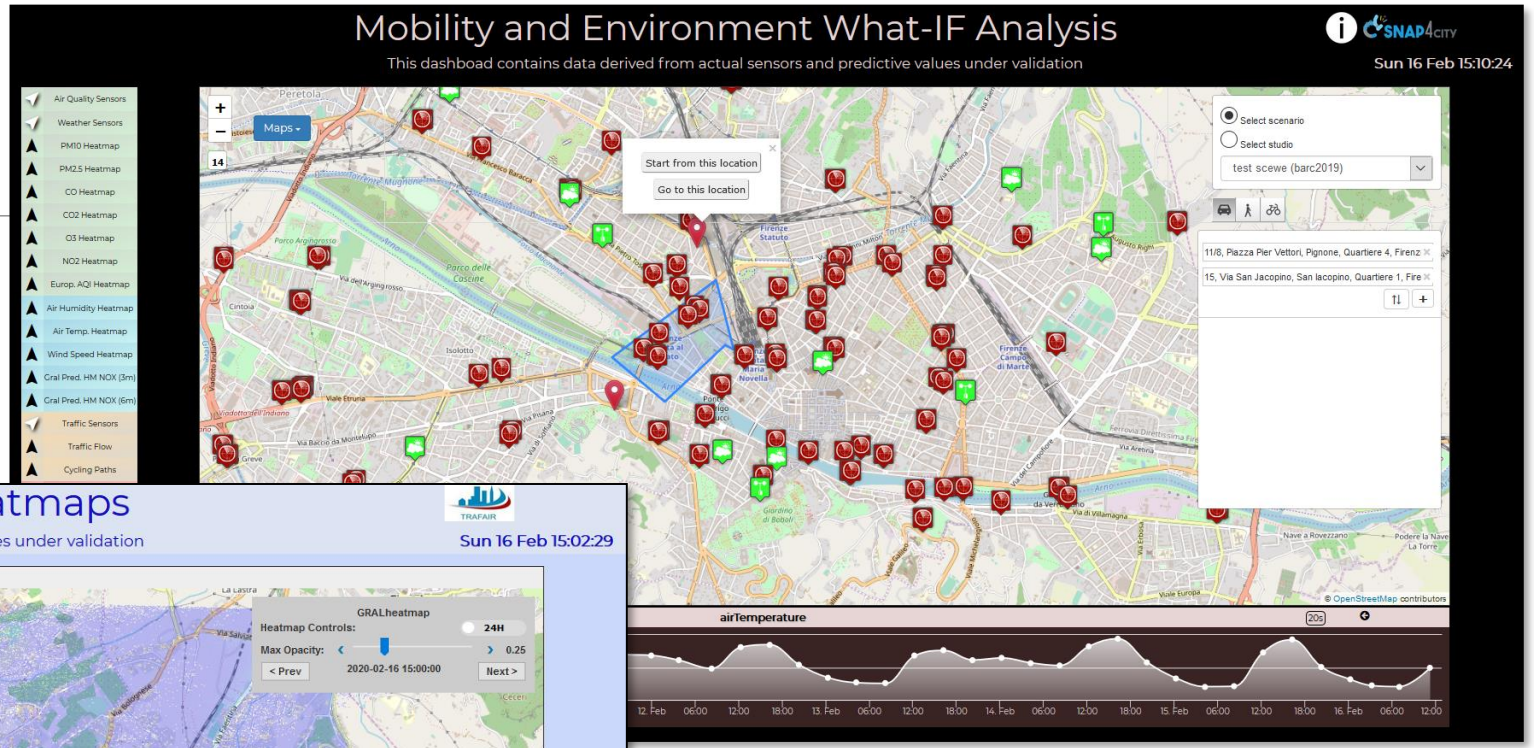


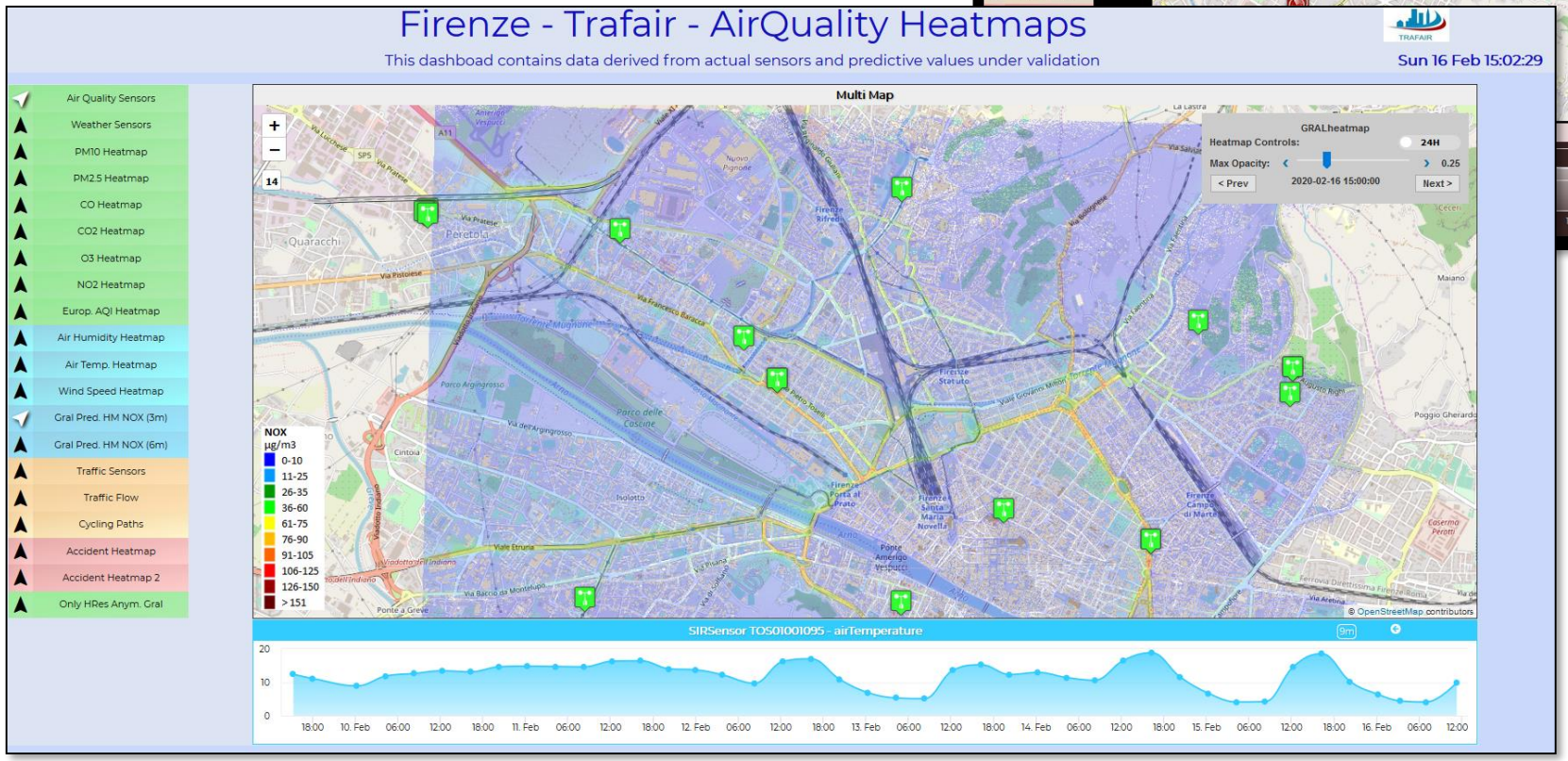
Early Warning Water Bomb



Early Warning Hot in Tuscany

… Predictions





Smart Cloud - Computing

Projects: http://www.disit.org/5501

- ICARO: http://www.disit.org/5482
- Social Museum and Smart Tourism

Tools: http://www.disit.org/5489

- Smart Cloud Engine and reasoner http://www.disit.org/6544
- Cloud ontology and tools: http://www.disit.org/5604
  - Configuration analysis and checker
  - Service Level Analyzer and control
- Cloud Simulation, ICLOS
- Cloud Monitoring, SM

http://www.cloudicaro.it

Regione Toscana

http://www.disit.org/6588

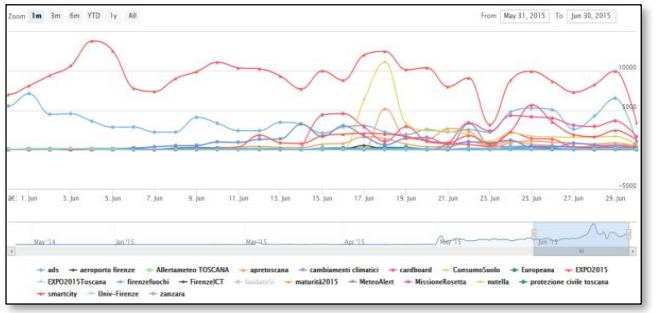
Text and Web Mining

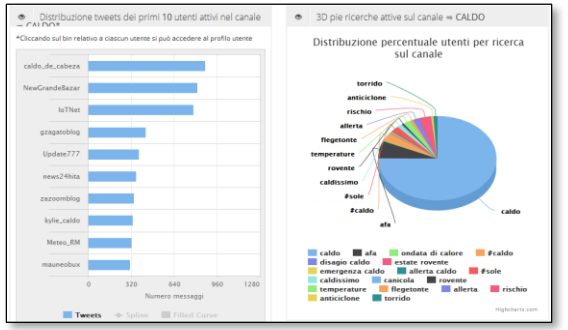
Projects: http://www.disit.org/5501

- OSIM: http://www.disit.org/5482
- SACVAR: http://www.disit.org/5604
- Blog/Twitter Vigilance

Tools: http://www.disit.org/5489

- Text and web mining, Natural Language Processing
- Service localization
- Web Crawling
- Competence analysis
- Blog Vigiliance, sentiment analysis

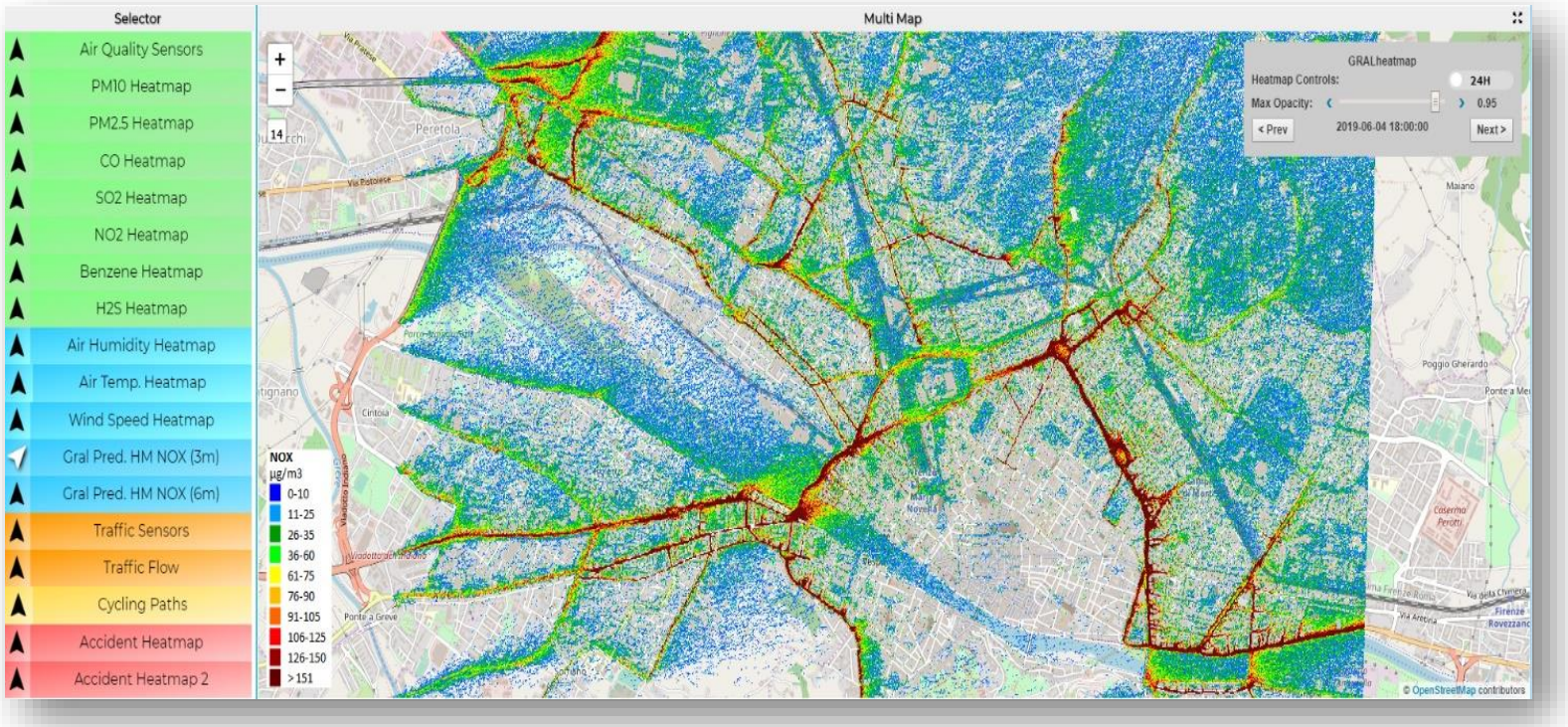




GRAL pred. Dashboard: Firenze

La Dashboard descrive la mappa della concentrazione di NOx previsto nella città di Firenze sia a 3 che a 6 metri (risoluzione 4x4 metri)

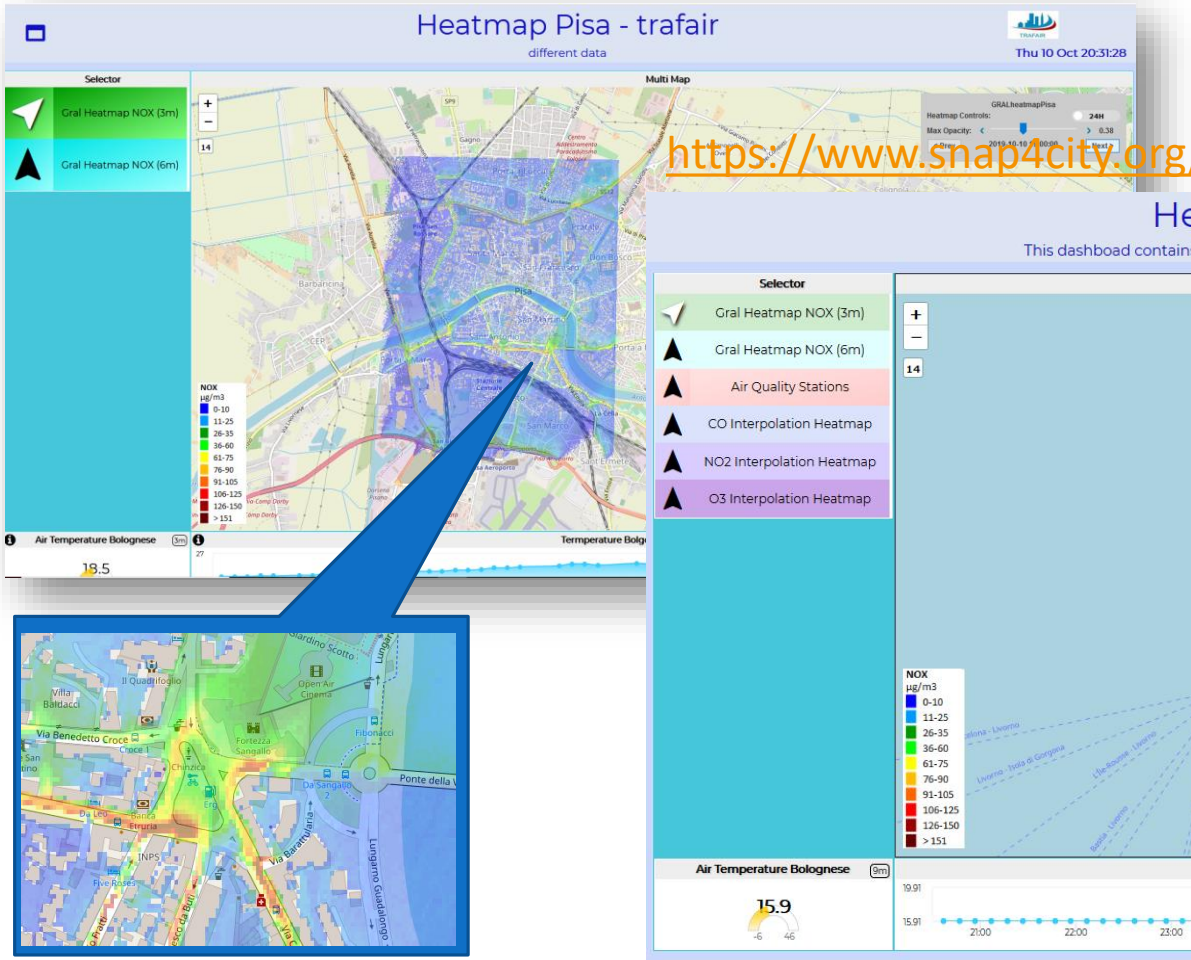
Dalla dashboard è possibile vedere animazioni 24 ore su 24 e relative ai due giorni successivi (48 ore)



https://www.snap4city.org/dashboardSmartCity/view/index.php?iddashboard=MTUzMg==

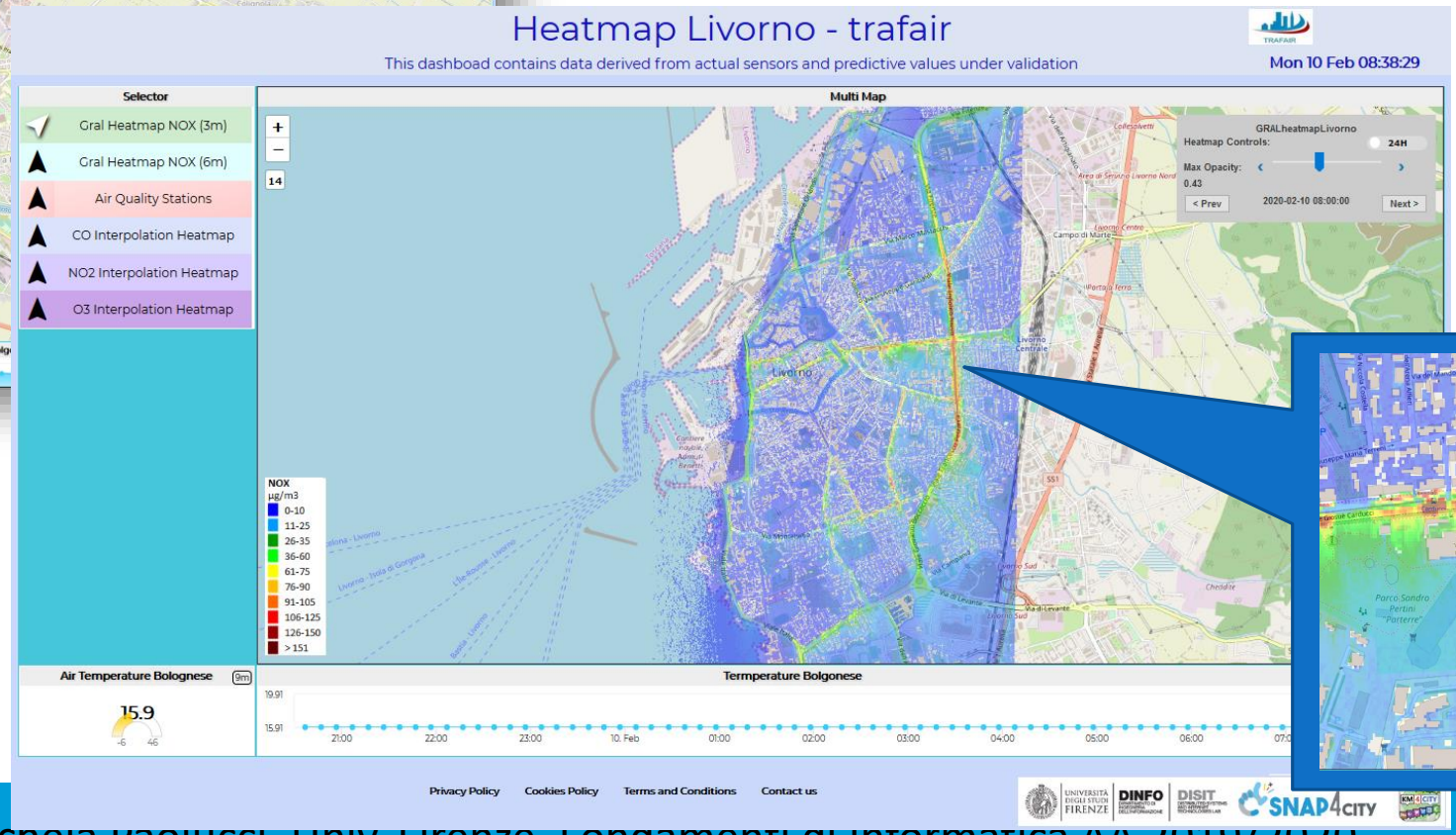
GRAL prediction DashBoards: Firenze, Livorno, Pisa

https://www.snap4city.org/dashboardSmartCity/view/index.php?iddasboard=MTc2Nw==



Pisa

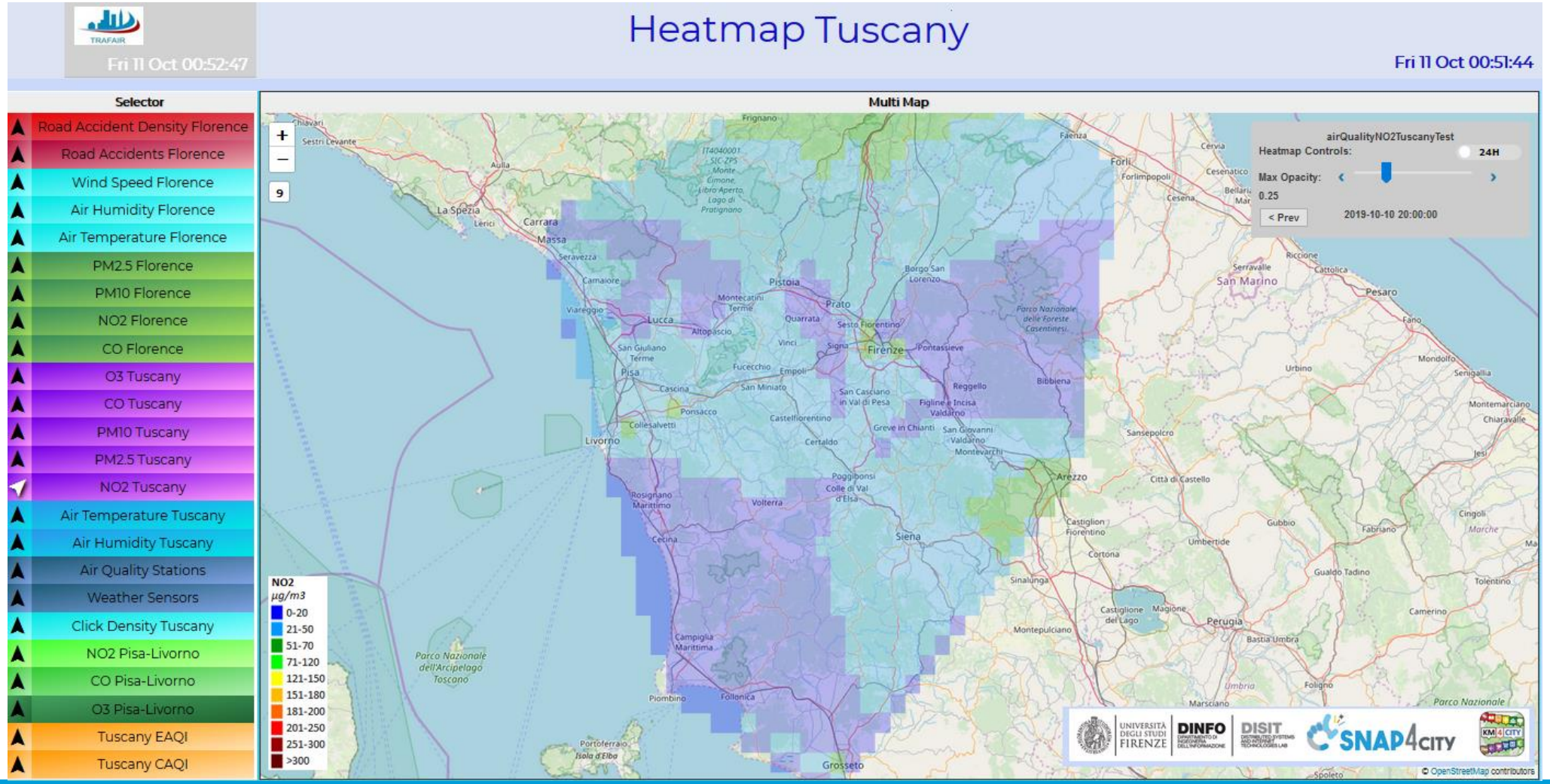
https://www.snap4city.org/dashboardSmartCity/view/index.php?iddasboard=MTgzMw==



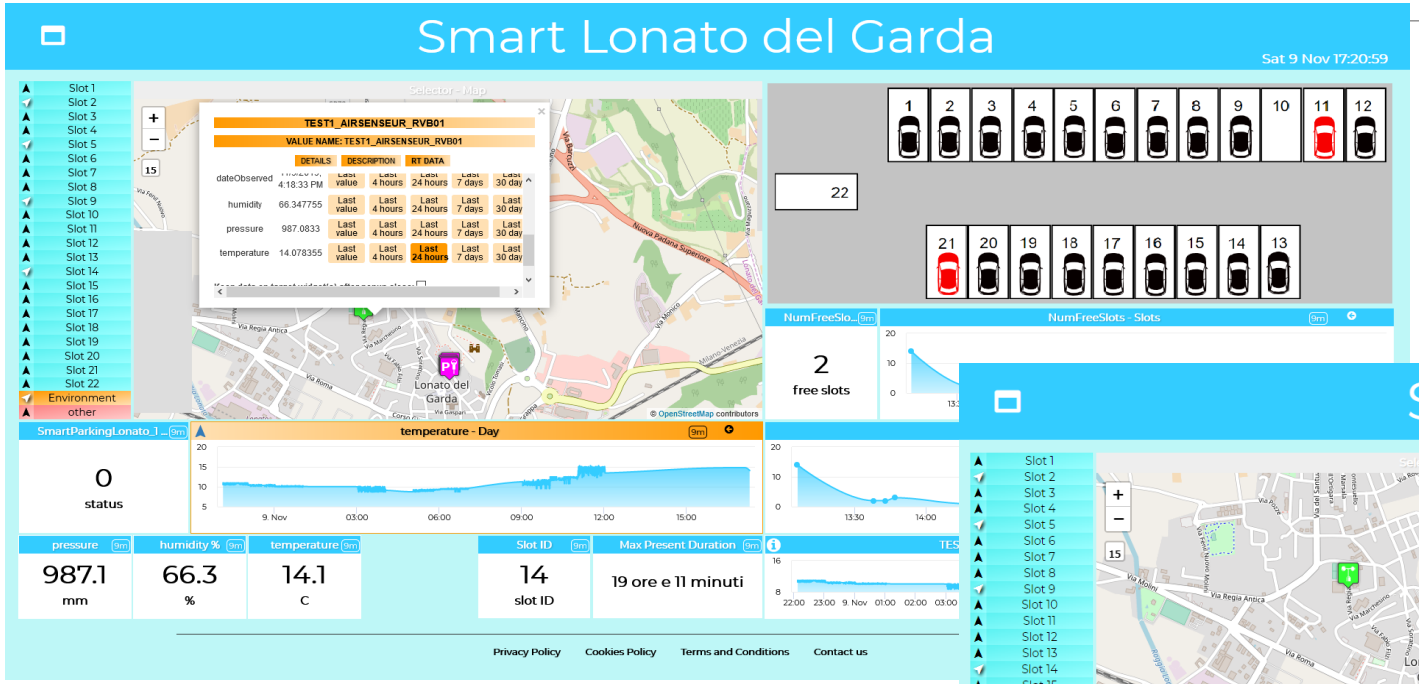
Livorno

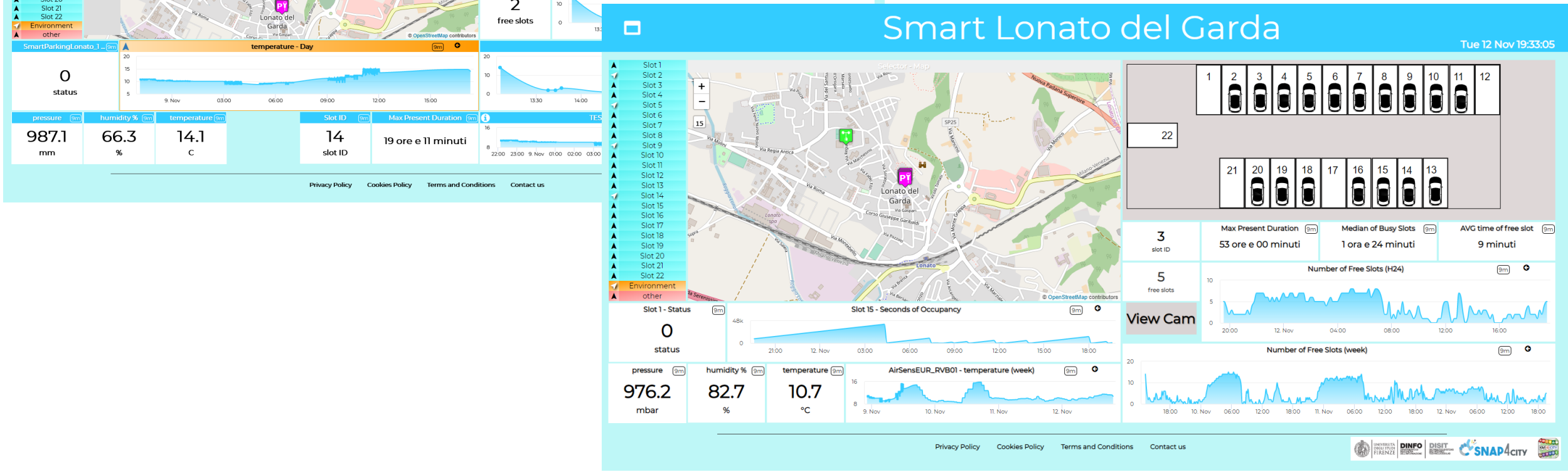
Tuscany Air Quality Interpolation Maps

https://www.snap4city.org/dashboardSmartCity/view/index.php?iddasboard=MTUxNQ==

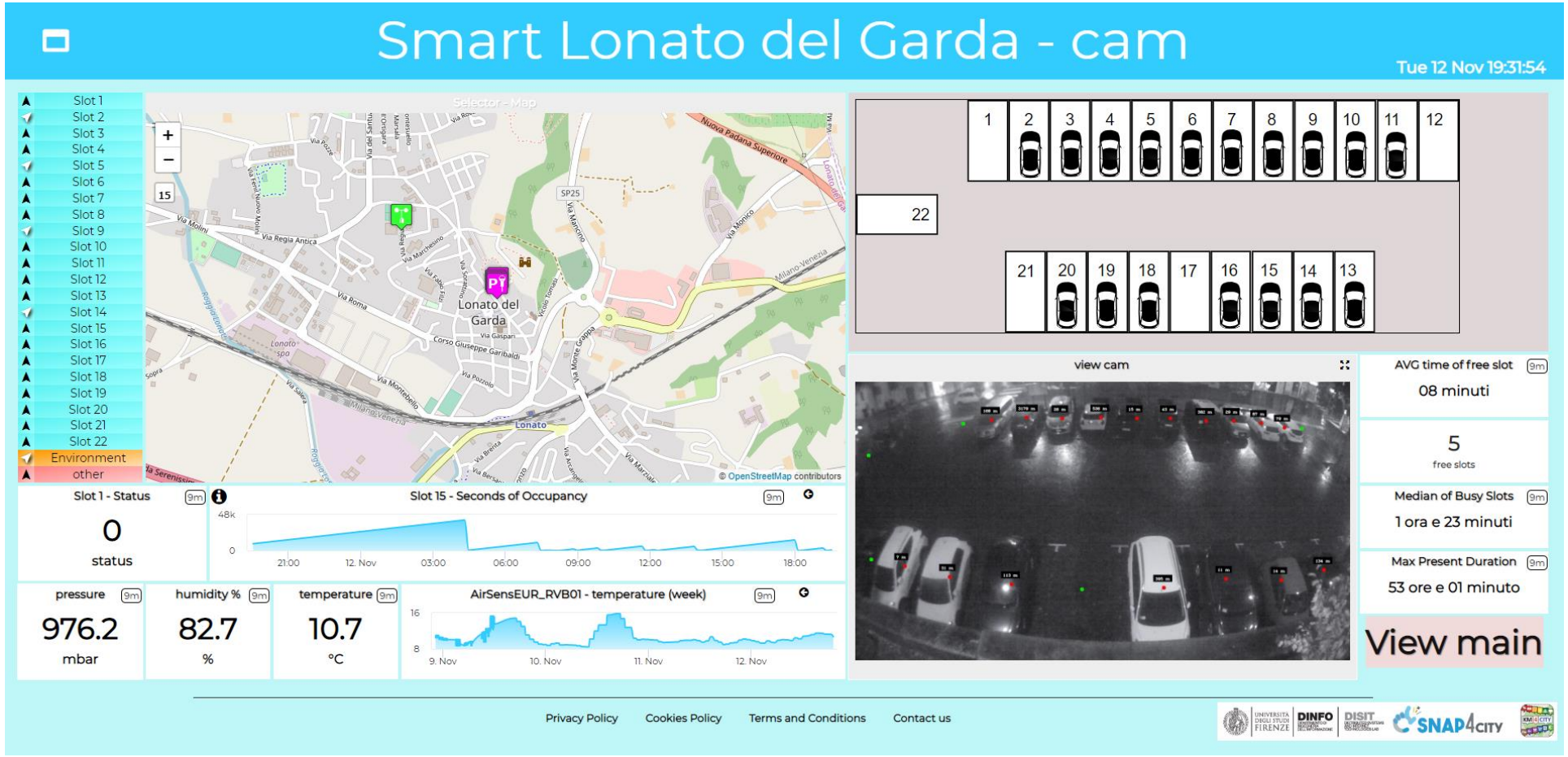


Smart Parking

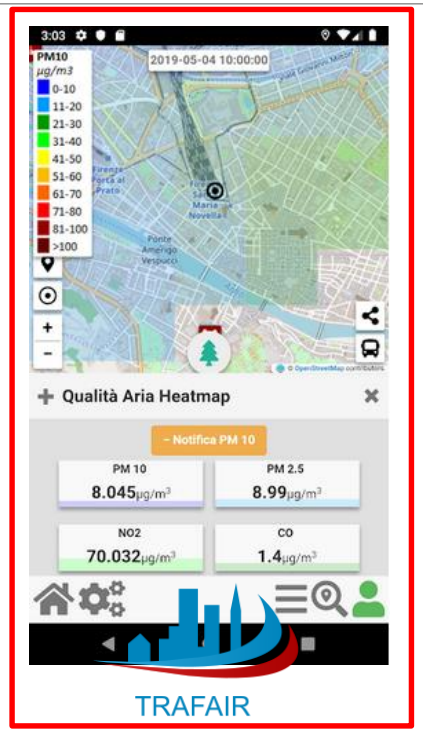


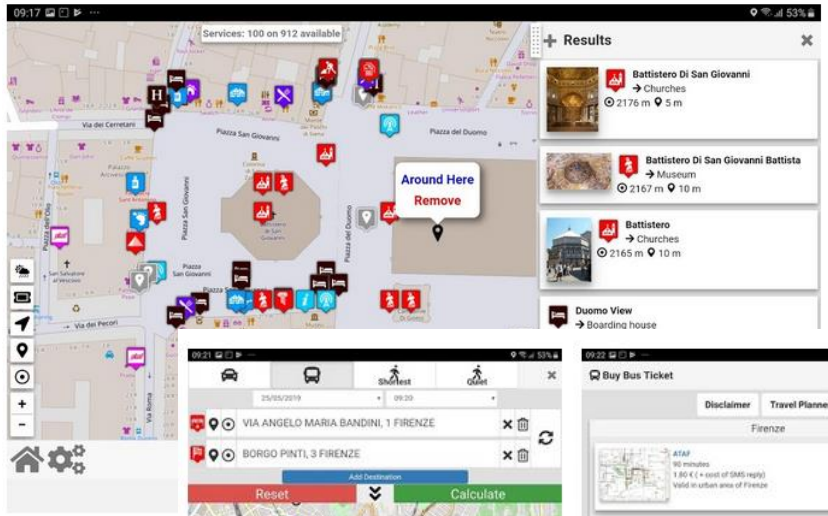


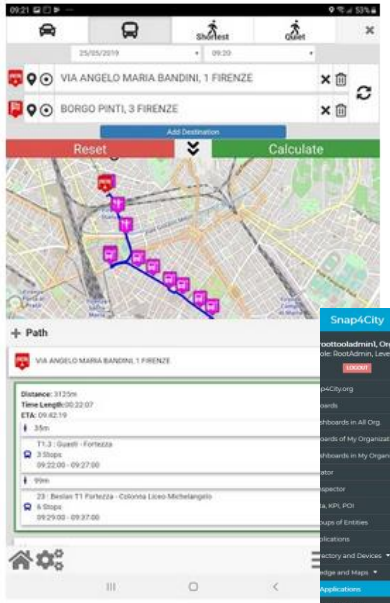
Smart Parking

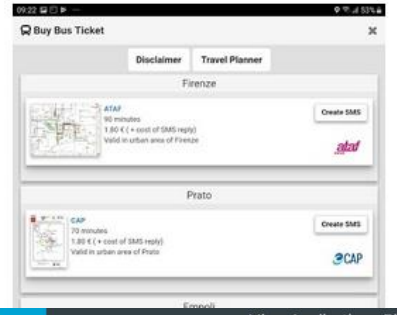


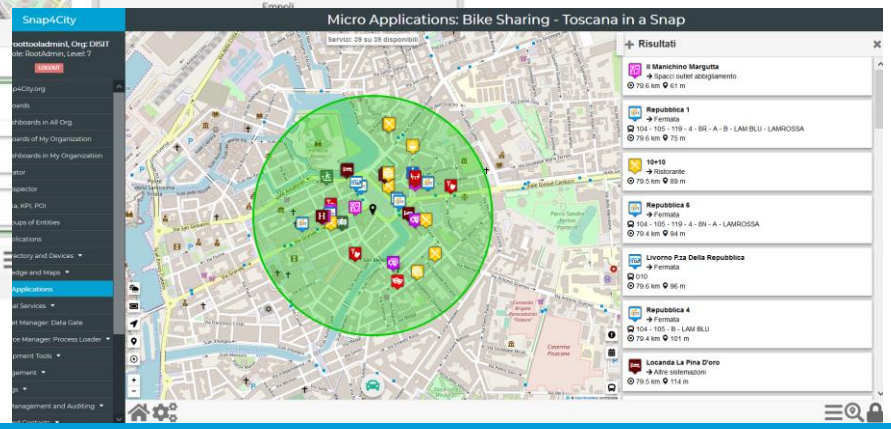
Per cittadini/turisti/studenti e pendolari Mobile App: *Toscana in a Snap*











TRAFAIR

Snap4City è un assistente personale su tutte le città e le aree della Toscana (Livorno, Firenze, Pisa, etc.):

- ambiente, mobilità e trasporti, cultura e turismo, intrattenimento, ma anche ospedali, meteo, orari del trasporto pubblico (13 operatori Toscani), pianificazione viaggi, qualità dell'aria, previsioni del tempo, percorsi multimodali, parcheggi, predizione sui posti liberi in parcheggio, car sharing, piste ciclabili, notizie ed eventi, biglietterie, acquisto biglietti a tempo, stazioni di rifornimento, noleggio, stato della pollinazione, ecc.

https://play.google.com/store/apps/details?id=org.disit.snap4city.mobileApp.tuscany&hl=it

Snap4City Mobile App: *Toscana in a Snap4*

https://play.google.com/store/apps/details?id=org.disit.snap4city.mobileApp.tuscany&hl=it

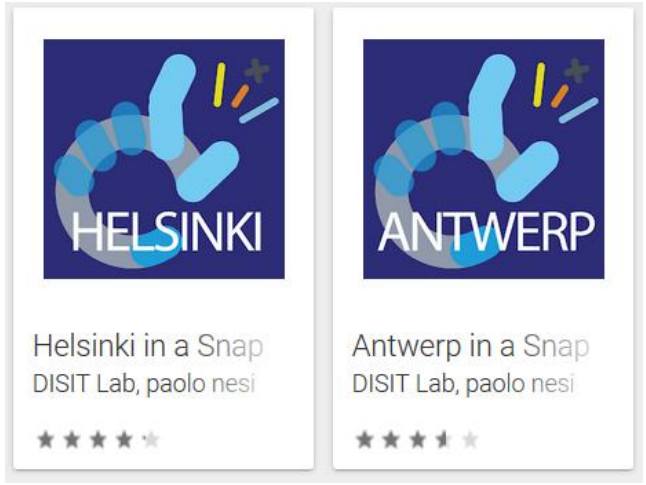
I dati disponibili sono elencati sul portale Snap4city (https://www.snap4city.org) e possono dipendere dalla tua posizione in Toscana. La maggior parte dei dati sono dati aperti, altri tipi di dati possono essere privati e accessibili solo per un periodo di tempo limitato che può dipendere dalle sperimentazioni in atto

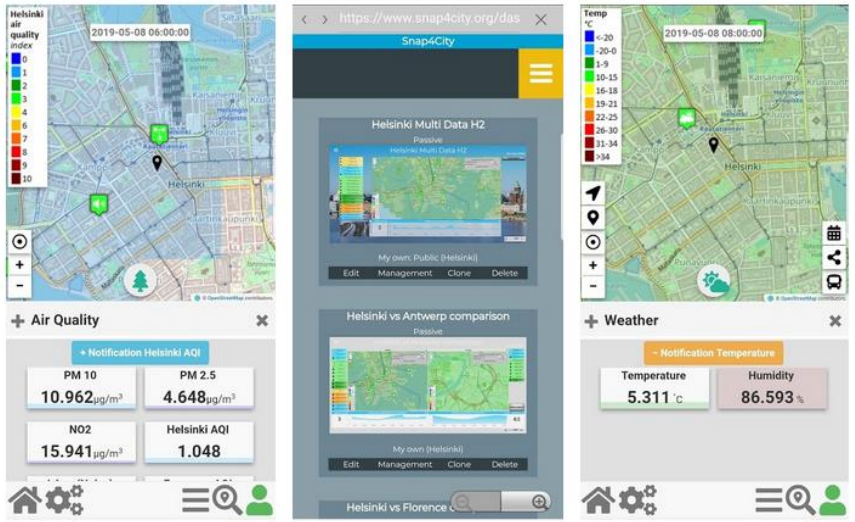
Registrandosi su *Toscana in a Snap4*, tale registrazione sarà valida anche per il portale Snap4City su cui è possibile:

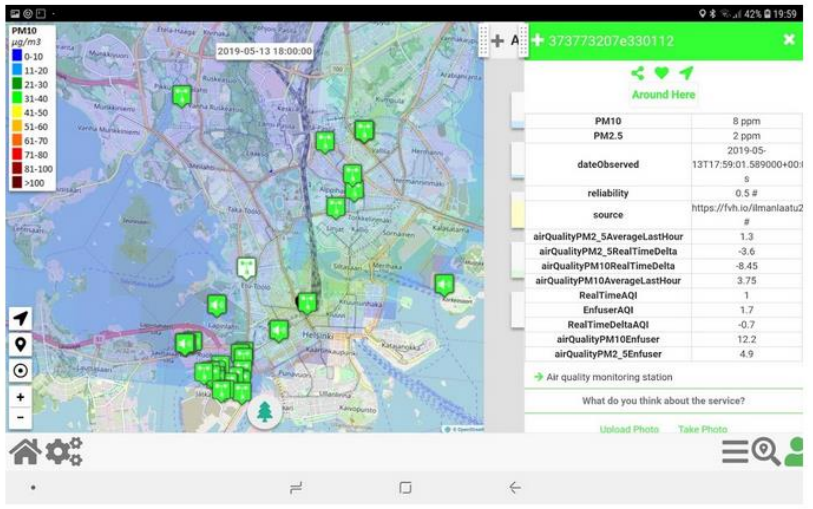
- Gestire il proprio profilo in base al regolamento GDPR della Commissione europea
- Avere accesso a molti altri servizi dal tuo portatile e da altri dispositivi
- Connettere i propri dispositivi (e.s. sensori qualità aria, temperature, etc.) e creare le proprie IoTApp o Dashboards

Snap4City Mobile App: Non solo Toscana…

Mobile App simili sono state realizzate e sono accessibili anche per Anversa e Helsinki





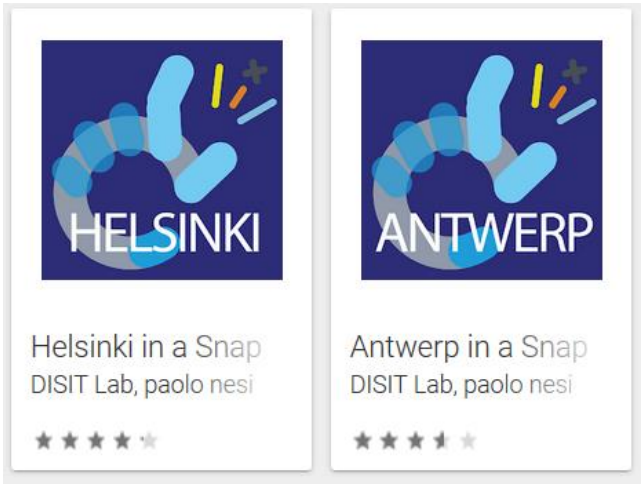


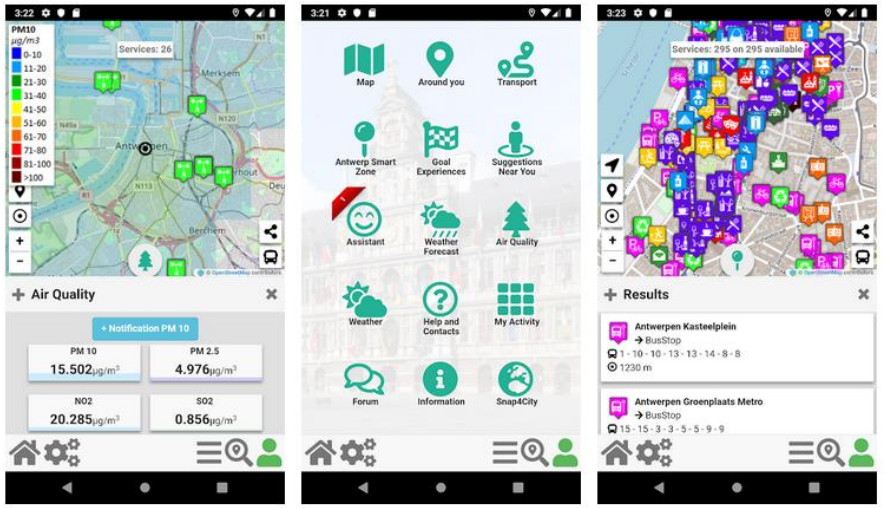
https://play.google.com/store/apps/details?id=org.disit.snap4city.mobileApp.helsinki&hl=it

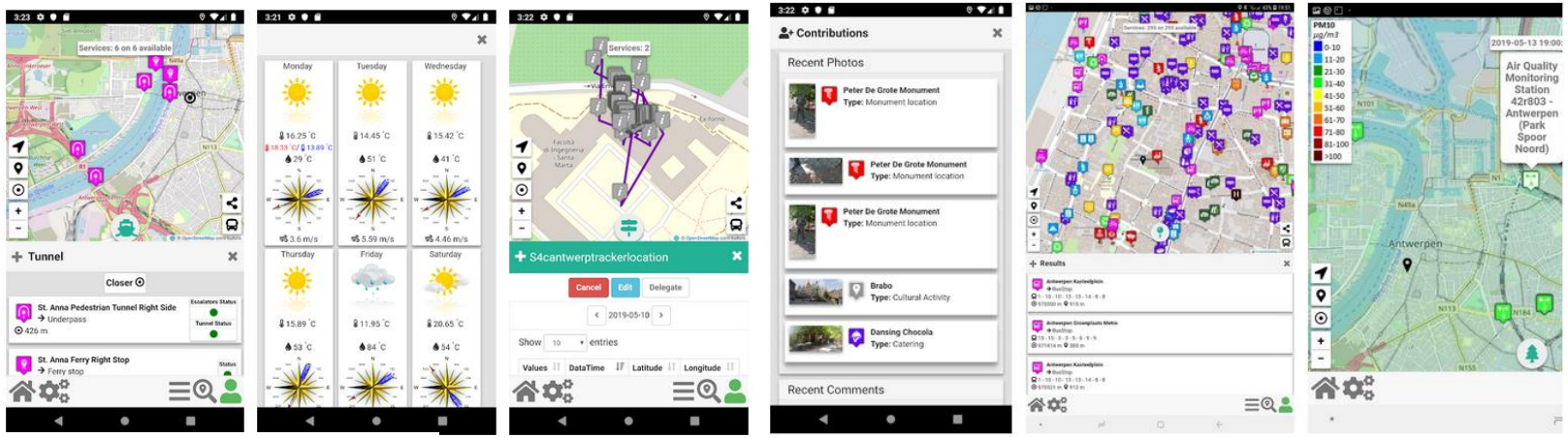
Helsinki

Snap4City Mobile App: Non solo Toscana…

Mobile App simili sono state realizzate e sono accessibili anche in Anversa e Helsinki





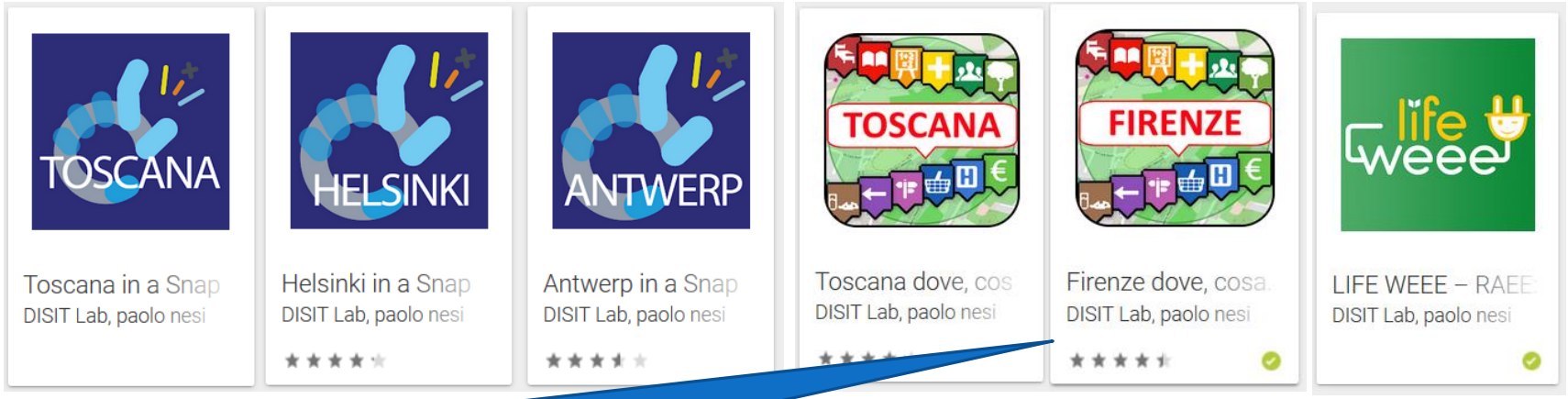


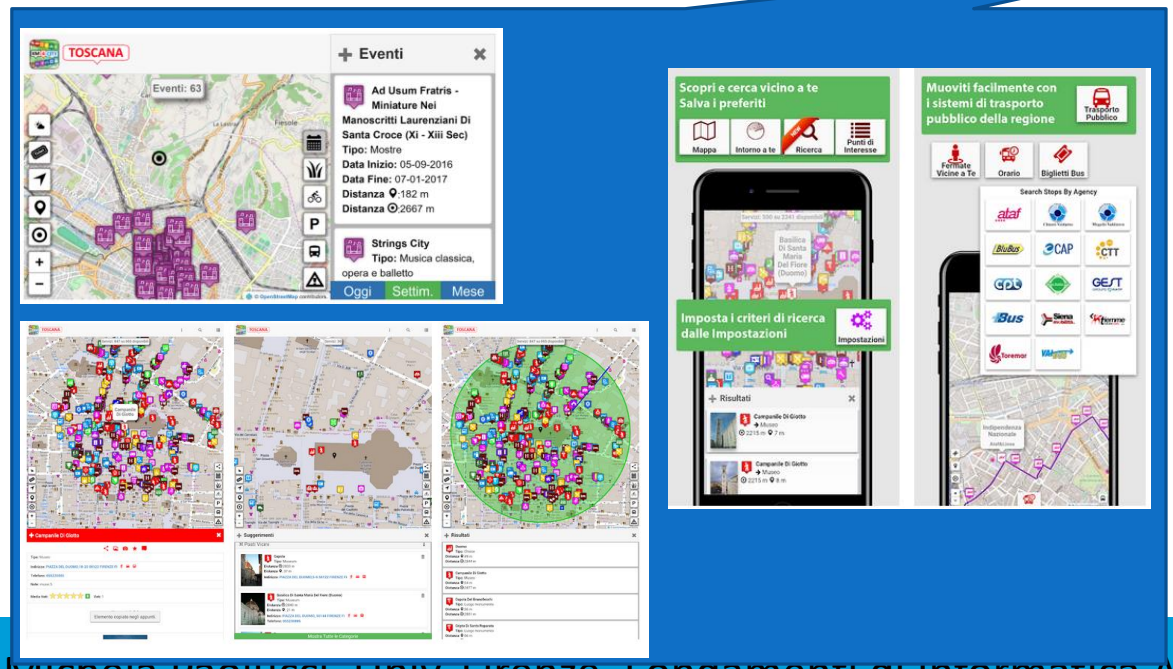
https://play.google.com/store/apps/details?id=org.disit.snap4city.mobileApp.antwerp&hl=it

Antwerp

Snap4City Mobile App: Non solo Toscana…

Mobile App simili sono state realizzate e sono accessibili anche in Anversa e Helsinki

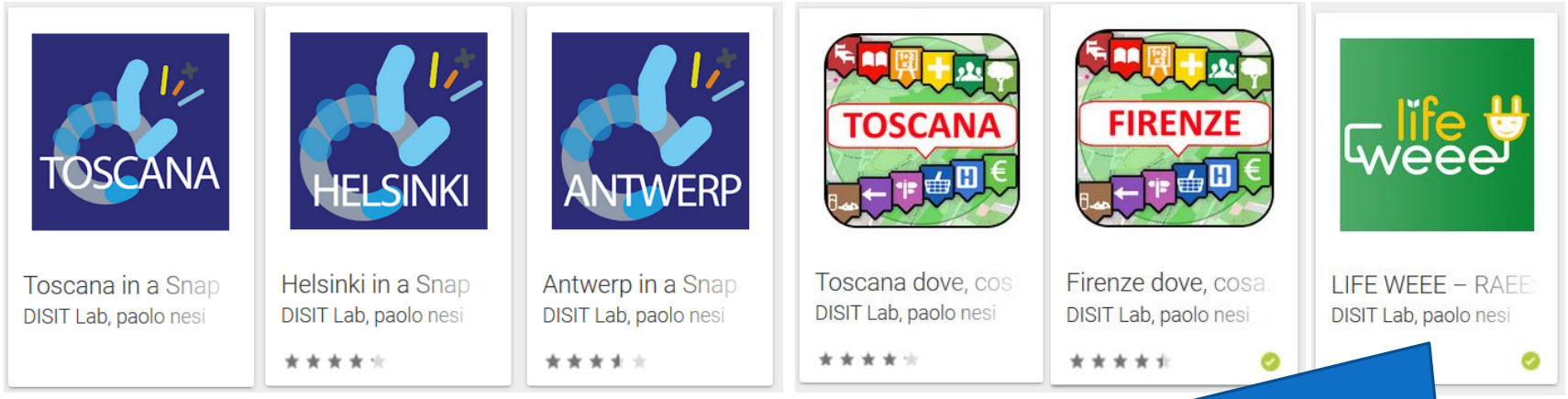




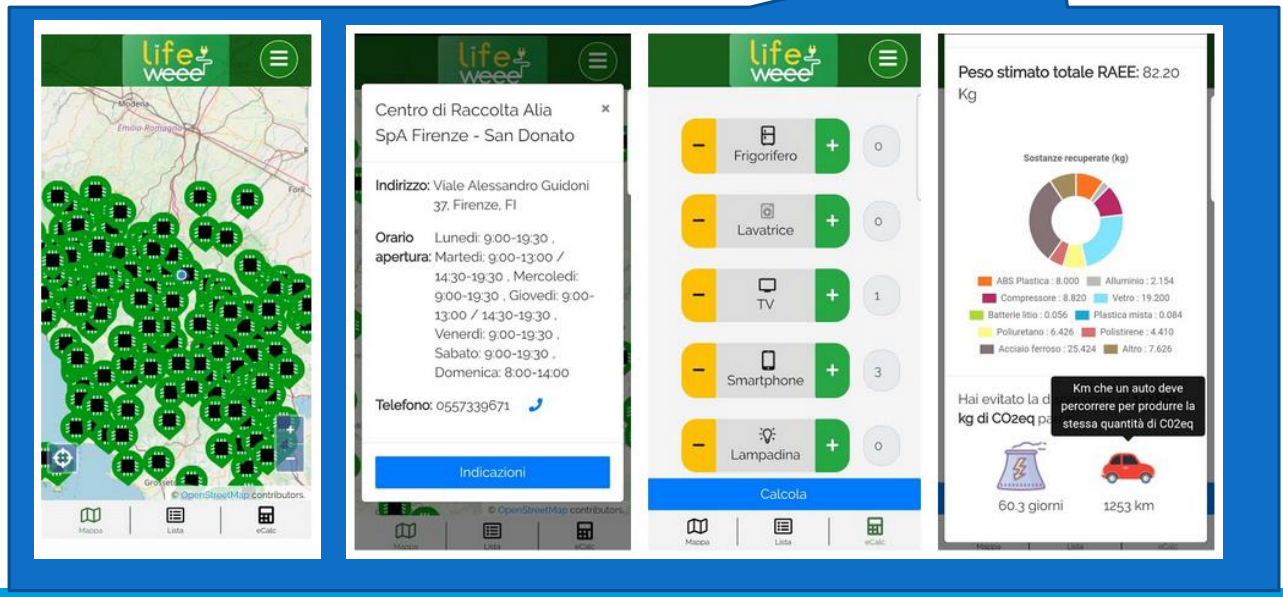
- I servizi geolocalizzati sono sui trasporti e mobilità, beni culturali, ospedali, meteo, servizi, autobus e tempi di attesa, banche, digital location, energia, etc. con immagini, audio e documenti, ODB2.
  Soluzioni in testing: MOSAIC guida connessa, KitBike Sii-Mobility

Snap4City Mobile App: Non solo Toscana…

Mobile App simili sono state realizzate e sono accessibili anche in Anversa e Helsinki

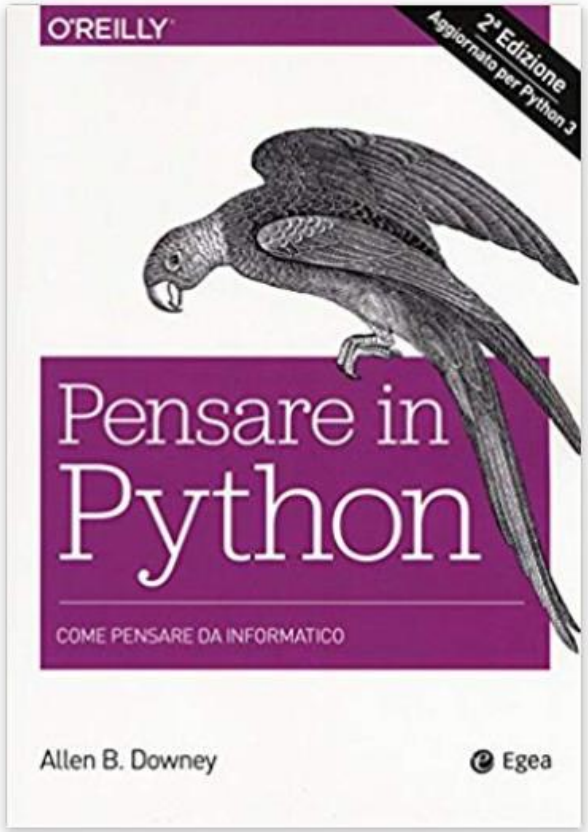


Il progetto LIFE WEEE: per incoraggiare cittadini e imprese a gestire con maggiore attenzione la raccolta dei rifiuti elettrici ed elettronici (RAEE)



Introduzione al Corso

Libro di Testo



- **Titolo del Libro: Pensare in Python**
- **Autore : Allen Downey**
- **Editore: EGEA**
- **Data di Pubblicazione: 2018**
- **Genere: libro. elaborazione dati**
- **Argomento : Python, linguaggio**
- **ISBN-10: 8823822645**
- **ISBN-13: 9788823822641**

Bibliografia –

- Concetti di informatica e fondamenti di Python, Cay Horstmann, Rance D. Necaise, Apogeo
- Pensare in Python - Come pensare da Informatico, Allen Downey, Green Tea Press
- Pensare da informatico - Imparare con Python, Allen Downey Jeffrey Elkner Chris Meyers
- …

Programma del Corso

Anno Accademico 2019-20

Laurea Triennale (DM 270/04) - INGEGNERIA GESTIONALE

Programma del corso - Cognomi A-L

- Introduzione al linguaggio python
  - Tipi, variabili e costanti
  - Operatori ed espressioni
  - Istruzioni
- Rappresentazione dei dati
  - Numeri
  - Interi
  - Caratteri e stringhe

Elementi di sintassi di un linguaggio

Esecuzione di programmi e ambienti: notebooks, IDE, console

- Il linguaggio python
  - tipi mutabili e immutabili
  - operatori ed espressioni
  - istruzioni
  - funzioni
  - cicli while e for
  - esecuzione condizionale
- Strutture dati e algoritmi elementari: Liste, Dizionari, Insiemi, iterazioni su strutture dati

Costo di esecuzione e complessità

Il modello di costo

Cenni sulla complessità di un algoritmo:

- Algoritmi di ordinamento su vettori
  - Sequential-sort
- Cenni sugli alberi
  - Alberi
  - Alberi binari di ricerca: i) Visita in forma ricorsiva; ii) Ricerca; iii) Inserimento ordinato

Cenni su analisi dei dati, lettura e scrittura di file in forma tabulare, grafici.

Pagina del Corso

http://www.disit.org/drupal/?q=node/7020

Qui trovate:

- AVVISI
- Slide del corso
- Approfondimenti

Modalità d’esame – alcune linee guida -

L'esame si compone di una prova scritta e una orale.

La prova scritta è svolta su carta A4.

Si accede alla prova orale solo se la parte di programmazione è corretta e funzionante

La prova orale può essere sostenuta a partire dalla settimana seguente alla prova scritta, non oltre la prova scritta successiva.

La prova orale inizia con la discussione dell'elaborato, e prosegue con l'approfondimento di tutti i contenuti del programma del corso.

Orario del Corso e Ricevimento

Docente: PAOLUCCI MICHELA

Ingegneria FIRENZE - A.A. 2019/2020 - 2° periodo

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
08:15						
09:15						
10:15						
11:15	FonDiInf(A-L) (Auditorium B - C.D.M.)					
12:15	FonDiInf(A-L) (Auditorium B - C.D.M.)					
14:00					FonDiInf(A-L) (Auditorium B - C.D.M.)	
15:00					FonDiInf(A-L) (Auditorium B - C.D.M.)	
16:00					FonDiInf(A-L) (Auditorium B - C.D.M.)	
17:00						
18:00						

Legenda: C.D.M.: Centro Didattico Morgagni
FonDiInf(A-L): FONDAMENTI DI INFORMATICA (A-L)

- Il ricevimento si svolge su appuntamento contattando la docente via e-mail:
  - michela.paolucci@unifi.it

Programma del corso - Cognomi A-L

- Introduzione
  - Introduzione al linguaggio python
    - Tipi, variabili e costanti
    - Operatori ed espressioni
    - Istruzioni
  - Rappresentazione dei dati
    - Numeri
    - Interi
    - Caratteri e stringhe

Elementi di sintassi di un linguaggio

Algoritmo (1)

- Il concetto di Algoritmo è uno dei concetti basilari della matematica
- Un algoritmo descrive un procedimento per risolvere una classe di Problemi in un numero finito di passi descritti in modo rigoroso
- Formalizzare un algoritmo significa trascrivere l'algoritmo da una scrittura informale ad una scrittura prettamente formale e, a volte, anche matematica
- Un algoritmo è un elenco finito di istruzioni, di passi, che devono essere eseguiti per arrivare alla soluzione finale del problema

Algoritmo (2)

- Esempi di algoritmi:
  - Per fare una torta c'è bisogno di una ricetta che deve essere seguita punto per punto
  - Per entrare in una Banca è necessario seguire una serie di linee guida
  - etc.
- Gli Algoritmi fanno uso di Dati in ingresso e sono in grado di produrre dei risultati che costituiscono la soluzione del problema in questione
- Se l'Algoritmo non produce risultati, serve a poco

Algoritmo (3)

- Algoritmo formalizzato con la notazione matematica:

  ‘Calcolo delle radici di un polinomio di 2° grado’

Dato il polinomio $ax^2+bx+c=0$, si possono calcolare le radici x_1 e x_2 con il seguente algoritmo:

$$\Delta = b^2 - 4*a*c$$

$$x_{1,2} = (-b \pm \sqrt{\Delta}) / 2*a \qquad \text{supponendo il } \Delta >= 0 ;$$

Algoritmo (4)

- I dati costituiscono le informazioni che vengono fornite dall'esterno, dall'utente
- L'algoritmo acquisisce dei dati dall'esterno e comunica i risultati all'ambiente esterno
- I risultati sono il prodotto dell'algoritmo
- L'algoritmo descrive come i risultati vengono prodotti in base ai dati
- L'algoritmo è composto di passi e di relazioni tra passi
- L'algoritmo può essere eseguito dall'uomo o da macchine automatiche, oppure dall'uomo con l'ausilio di macchine automatiche
- L'algoritmo deve essere comprensibile per l'esecutore (uomo o macchina)
- Lo schema di esecuzione degli algoritmi è costituito da alcune regole che indicano l'ordine di esecuzione delle istruzioni che lo costituiscono

Algoritmo (5)

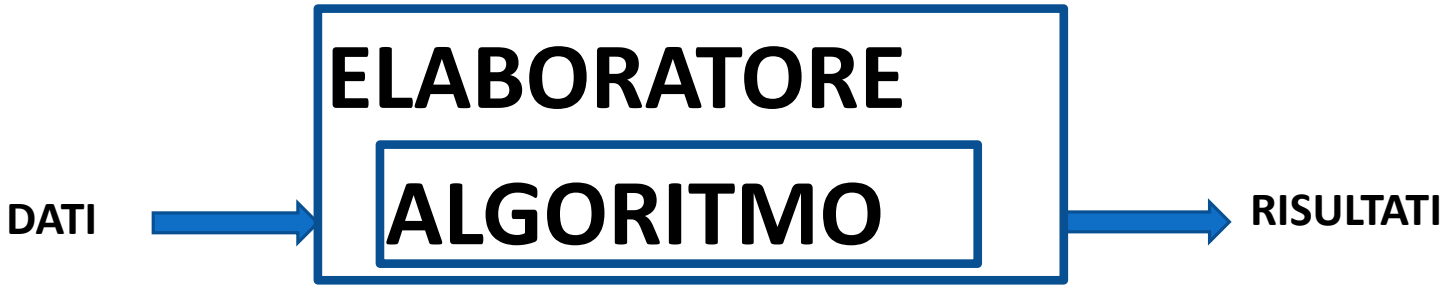
- Esempi di Algoritmi:
  - Andare a fare la spesa
  - Preparare una torta
  - Entrare in banca
  - Andare a iscriversi all'Università
  - Data una serie di numeri trovare il maggiore
  - Dati due numeri trovare il MCD
  - Dati due numeri trovare il mcm
  - …

Elaboratore (1)

- L'Elaboratore è quella 'macchina' in grado di eseguire gli algoritmi forniti dall'uomo sulla base di dati diversi ma in modo automatico
- Una volta fornito un algoritmo dall'esterno (realizzato dall'uomo), spesso l'elaboratore è in grado di eseguirlo più velocemente dell'uomo
- Sull'elaboratore l'uomo può eseguire programmi diversi tra loro per tipologia e formulazione MA è necessario che venga usato un linguaggio di programmazione compatibile con l'elaboratore stesso
- L'elaboratore è in grado di operare solo in base a istruzioni redatte in un formato per lui eseguibile

Elaboratore (2)

- Un algoritmo deve essere descritto attraverso un linguaggio generalizzato costituito da strutture linguistiche definite
- Solitamente le proposizioni che vengono usate per gli algoritmi contengono una descrizione di:
  - i) operazioni che devono essere eseguite
  - ii) oggetti sui quali si devono effettuare le operazioni per ottenere i risultati voluti
- L'elaboratore è flessibile perché permette di eseguire algoritmi diversi semplicemente cambiando il programma (algoritmo)



Elaboratore (3)

- Un algoritmo, per essere eseguibile, deve essere comprensibile per chi lo esegue
- Gli elaboratori sono in grado di eseguire istruzioni SE queste sono opportunamente codificate
- Un insieme di istruzioni ordinate secondo un certo schema, che corrisponde all'implementazione di un algoritmo, può essere chiamato **Programma**
- Con programmi diversi verranno usati dati diversi e/o uguali in ingresso e producono risultati diversi e/o uguali ma sempre coerenti con il programma
- L'**Elaboratore** è una apparecchiatura AUTOMATICA, DIGITALE o ELETTRONICA capace di effettuare trasformazioni su dei dati in ingresso

Informatica (1)

- ‘Scienza che studia l’informazione e, più specificamente, l’elaborazione dei dati e il loro trattamento automatico per mezzo di computer’, Garzanti Linguistica

- ‘Scienza pratica che si occupa del trattamento dell’informazione mediante procedure automatizzabili’, Wikipedia

- ‘Scienza che studia l’elaborazione delle informazioni e le sue applicazioni; più precisamente l’i. si occupa della **rappresentazione**, dell’organizzazione e del trattamento automatico della informazione. Il termine i. deriva dal fr. *informatique* (composto di INFORMATion e automatIQUE, «informazione automatica») e fu coniato da P. Dreyfus nel 1962.’, Treccani

Informatica (2)

- L'informatica può essere anche definita come lo studio degli algoritmi che descrivono e trasformano l'informazione

- Per passare da un problema a un programma è necessario applicare una serie di procedimenti, principalmente:
  - Analisi
  - Programmazione

  L'insieme di queste attività serve per risolvere un problema tramite l'elaboratore

Problema → **ALGORITMO** → **Programma**

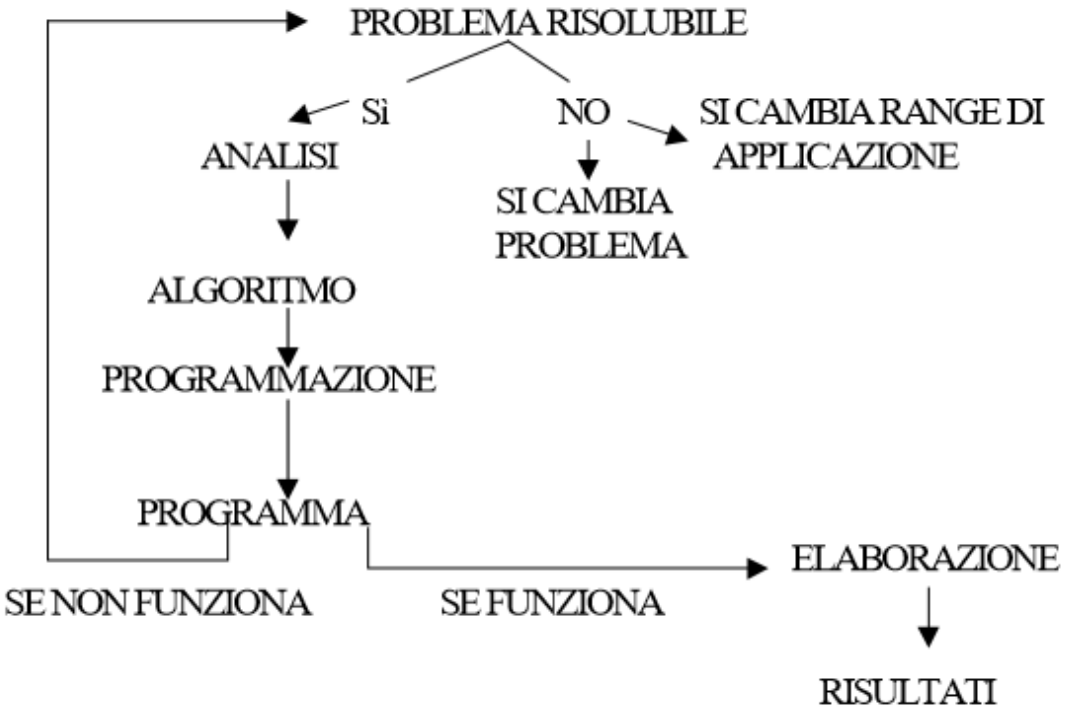
Informatica (3)

- Lo scopo dell'Analisi è quello di definire un algoritmo, cioè definire un elenco finito di istruzioni che determinano una serie di operazioni per ottenere una soluzione al problema

- Prima ancora di formulare l'algoritmo, si deve analizzare il problema e definire:
  - Se è risolvibile o meno
  - Se i dati rientrano nel dominio stabilito

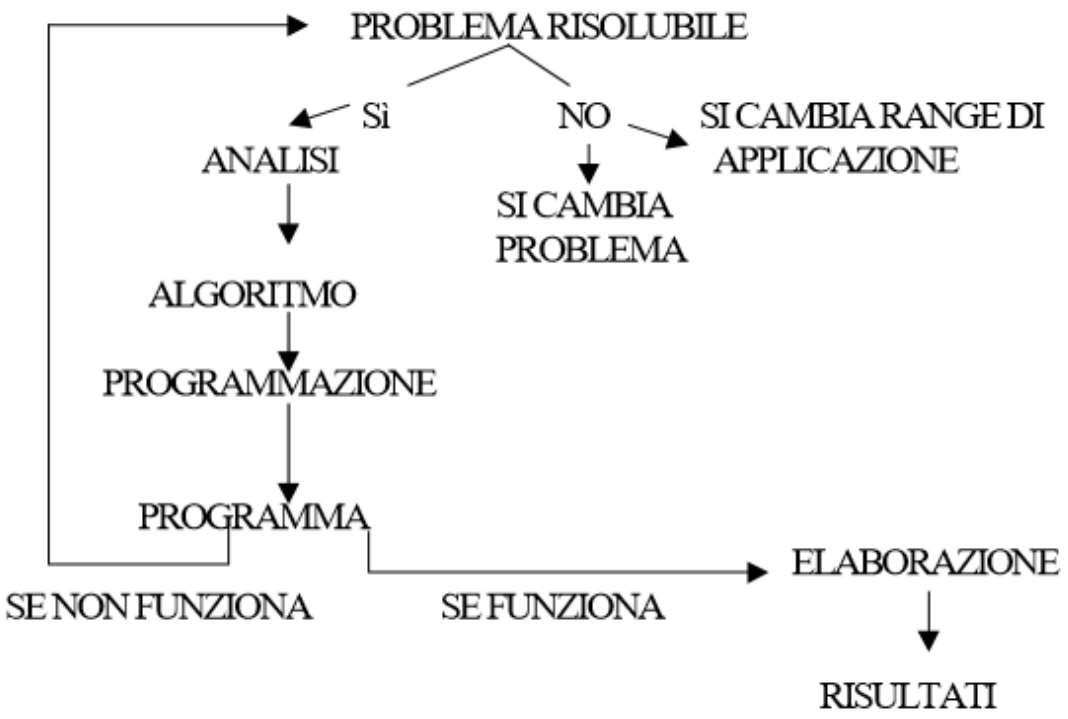
  Tutto in base ai presupposti e ai dati iniziali

Informatica (4)



- Se il problema è risolvibile, si passa alle operazioni di analisi e alla definizione dell'algoritmo
- SE NON è risolvibile, ci sono due alternative:
  - Si cambia completamente il problema
  - Si riduce il range di applicazione e si controlla nuovamente la risolubilità del problema

Informatica (4)



- Una volta definito l'Algoritmo si passa alla Programmazione
- La programmazione ha come scopo la definizione di un programma ottimale
- Non è detto che il programma funzioni, in questo caso è necessario:
  - tornare al problema iniziale
  - modificarlo se necessario
  - ripetere il resto delle operazioni

Programma (1)

- Un Programma è una traduzione dell'algoritmo in un linguaggio comprensibile dall'elaboratore (Calcolatore) e deve essere in grado di:
  - Produrre il risultato voluto
  - Oppure dichiarare che la soluzione non è stata trovata
- Un programma è registrato nella memoria di un Elaboratore
- Cambiando il programma, si può mettere l'elaboratore in grado di risolvere problemi di natura diversa
- E' assolutamente necessario che la SINTASSI di ciascuna riga di codice sia assolutamente corretta
- Partiamo dalle basi… Come **rappresentare** i dati?

Rappresentazione (1)

- Il problema della rappresentazione consiste nel dare ad un algoritmo una forma tale che possa essere eseguita da un elaboratore

- In questo ambito assume un ruolo fondamentale l'uso di un linguaggio di programmazione

- In questo corso sarà affrontato il problema avendo come riferimento il linguaggio Python

Rappresentazione (2)

- E’ necessario capire come riuscire a formalizzare in maniera logica (Algoritmo) i concetti che vogliamo trasformare in programma;
- Questa operazione è una delle più importanti e complessa da effettuare
- Da questa fase dipende poi tutta la scrittura del programma
- Due strumenti fondamentali per formalizzare i problemi, per poi poterli trasformare in programmi leggibili e risolvibili dall’elaboratore sono:
  - Algebra di Boole (utile per le operazioni sui dati)
  - Diagrammi di Flusso o Flow Chart (utili per rappresentare in maniera ordinata i nostri processi logici).

Algebra di Boole (1)

- I fondamenti dell'Algebra Booleana sono stati delineati dal matematico inglese George Boole, nato a Lincoln nel 1815 e morto nel 1864, in un lavoro pubblicato nel 1847 riguardante l'analisi della logica e in particolare l'algebra della logica.

- L'Algebra di Boole è stata fondamentale nel campo della elettronica digitale, in particolare nella progettazione e realizzazione dei circuiti elettronici

- L'Algebra di Boole è fondamentale nell'informatica

Algebra di Boole (2)

- L'Algebra di Boole non è legata esclusivamente alla programmazione, ma fa parte della vita quotidiana
- Pensiamo ad esempio a quando vogliamo uscire di casa:
  - ***se piove non possiamo uscire;***
- Questo processo mentale, ovvero quello di verificare se piova o meno, può assumere, di fatto, due stati:
  - o è vero (e quindi non usciamo)
  - o è falso (e quindi usciamo).

- Oppure quando andiamo al cinema:
  - ***Se compriamo il biglietto, ci fanno entrare;***
  - Vero: ho comprato il biglietto ed entro
  - Falso: niente biglietto e niente ingresso al cinema

Algebra di Boole (3)

- Si può capire che se questo tipo di logica risulta utile nella vita quotidiana, lo è ancora di più per un computer

- L'Algebra di Boole, a differenza di quella tradizionale, è un'algebra **binaria**, ovvero le variabili possono assumere soltanto due stati

- Esempi:
  - 1/0;
  - v/f (vero, falso);
  - l/h (low, high);
  - t/f (true, false);
  - on/off; (acceso/spento)

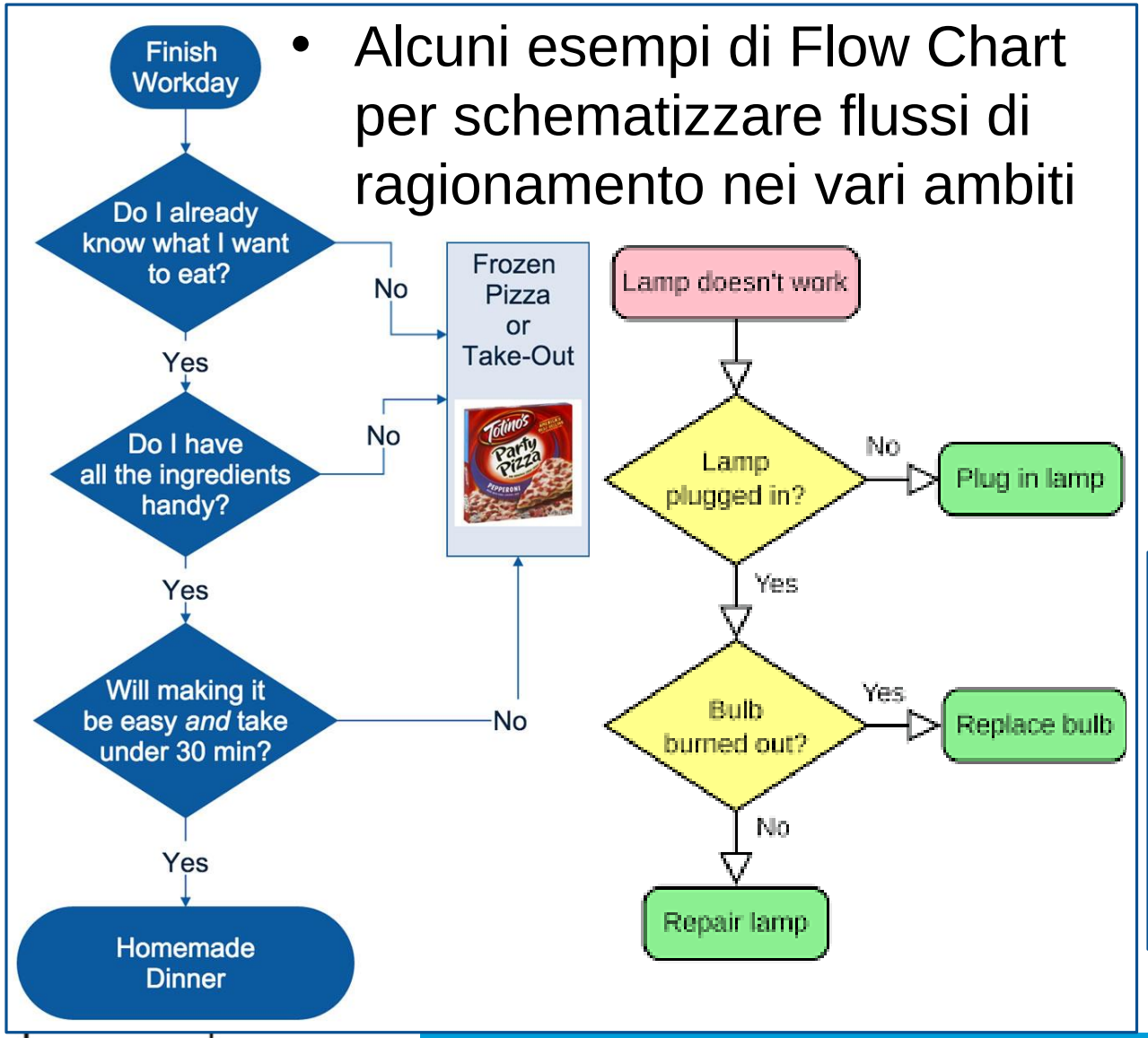
Introduzione ai Flow Chart (1)

- I **Flow Chart** sono dei disegni/schemi che rappresentano graficamente un ragionamento rappresentandolo come un algoritmo

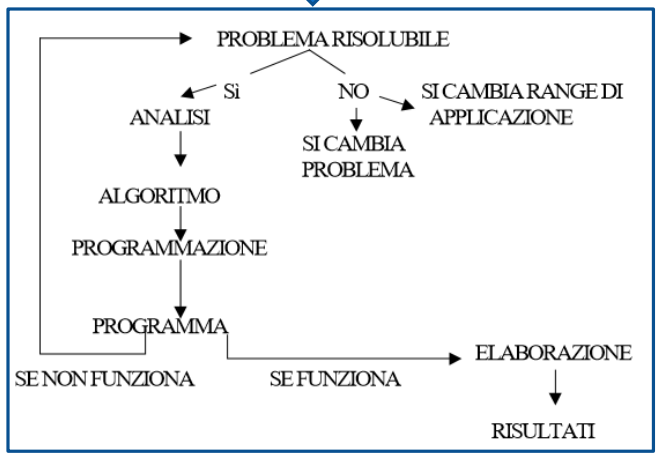
- Una rappresentazione schematica del problema permette:
  - una comprensione immediata del funzionamento del percorso logico che sta alla base della risoluzione del problema
  - un controllo accurato sulla funzionalità e la casistica del ragionamento

Introduzione ai Flow Chart (2)

- Alcuni esempi di Flow Chart per schematizzare flussi di ragionamento nei vari ambiti



- Ricordano lo schema visto per la formalizzazione di un algoritmo e di un programma



… Tornando al Python

- Il nostro scopo è quello di analizzare, formalizzare e risolvere problemi avendo come riferimento il linguaggio Python
- Pyhton è stato sviluppato agli inizi degli anni novanta da Guido Van Rossum, con lo scopo di poter sviluppare e modificare velocemente i programmi
- Python si è poi diffuso in modo rilevante anche per applicazioni professionali, scientifiche, accademiche
- E' necessario quindi capire quali sono le prime basi di partenza per poi poter usare anche Algebra di Boole e Flow Chart per formalizzare i problemi
- Come possiamo formalizzare i nostri problemi?
- Quali sono gli strumenti che Python ci fornisce?
- Come si possono rappresentare i dati che poi useremo per formalizzare i problemi?

Programma del corso - Cognomi A-L

- Introduzione al linguaggio python
  - Tipi, variabili e costanti
  - Operatori ed espressioni
  - Istruzioni
- Rappresentazione dei dati
  - Numeri
  - Interi
  - Caratteri e stringhe

Elementi di sintassi di un linguaggio

Il linguaggio di programmazione Python (1)

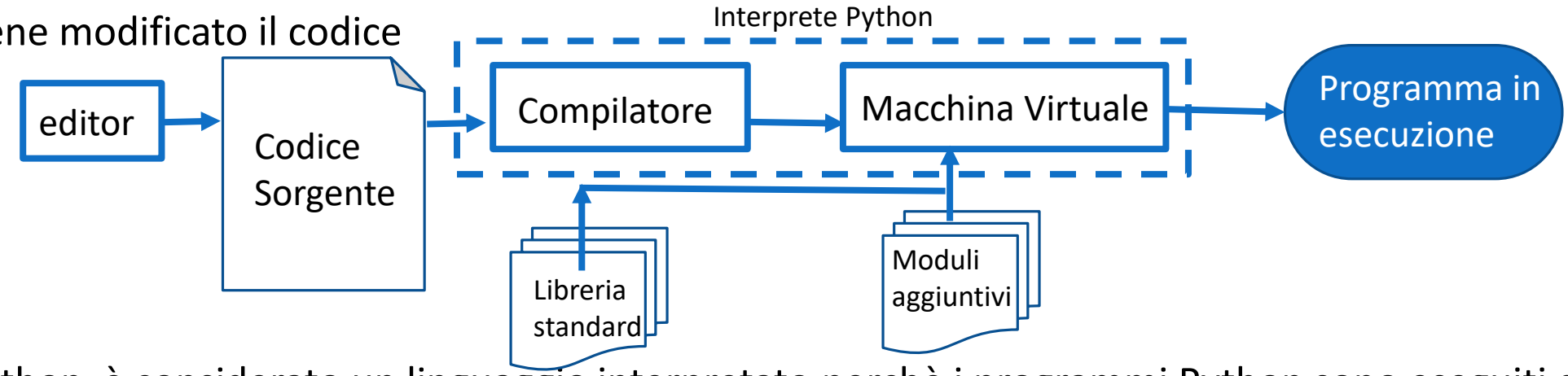
- Il linguaggio di programmazione che imparerai ` e il Python. Python è un esempio di linguaggio di alto livello; altri linguaggi di alto livello sono: il C, il C++, il Perl ed il Java
- Esistono anche linguaggi di basso livello, talvolta chiamati “linguaggi macchina” o “linguaggi assembly”
- «In modo non del tutto corretto si può affermare che i computer possono eseguire soltanto programmi scritti in linguaggi di basso livello»: i programmi scritti in un linguaggio di alto livello devono essere elaborati prima di poter essere eseguiti
- I programmi ad alto livello sono più veloci da scrivere, più corti e facilmente leggibili, ed è più probabile che siano corretti. In secondo luogo i linguaggi di alto livello sono portabili: portabilità significa che possono essere eseguiti su tipi di computer diversi con poche o addirittura nessuna modifica. I programmi scritti in linguaggi di basso livello possono essere eseguiti solo su un tipo di computer e devono essere riscritti per essere trasportati su un altro sistema.

Il linguaggio di programmazione Python (2)

- I programmi di alto livello vengono trasformati in programmi di basso livello eseguibili dal computer tramite due tipi di elaborazione: l'interpretazione e la compilazione
- Un interprete legge il programma di alto livello e lo esegue, trasformando ogni riga di istruzioni in un'azione. L'interprete elabora il programma un po' alla volta, alternando la lettura delle istruzioni all'esecuzione dei comandi che le istruzioni descrivono

Il linguaggio di programmazione Python (3)

- Un compilatore legge il programma di alto livello e lo traduce completamente in basso livello, prima che il programma possa essere eseguito
- In questo caso il programma di alto livello viene chiamato codice sorgente, ed il programma tradotto codice oggetto o eseguibile. Dopo che un programma è stato compilato può essere eseguito ripetutamente senza che si rendano necessarie ulteriori compilazioni finchè non ne viene modificato il codice



- Python è considerato un linguaggio interpretato perchè i programmi Python sono eseguiti da un interprete
- Ci sono due modi di usare l'interprete: a linea di comando o in modo script

Il linguaggio di programmazione Python (4)

- Ci sono due modi di usare l'interprete: a linea di comando o in modo script
- In modo "linea di comando" si scrivono i programmi Python una riga alla volta: dopo avere scritto una riga di codice alla pressione di Invio (o Enter, a seconda della tastiera) l'interprete la analizza subito ed elabora immediatamente il risultato, eventualmente stampandolo a video

```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>>
```

```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(1+1)
2
>>>
```

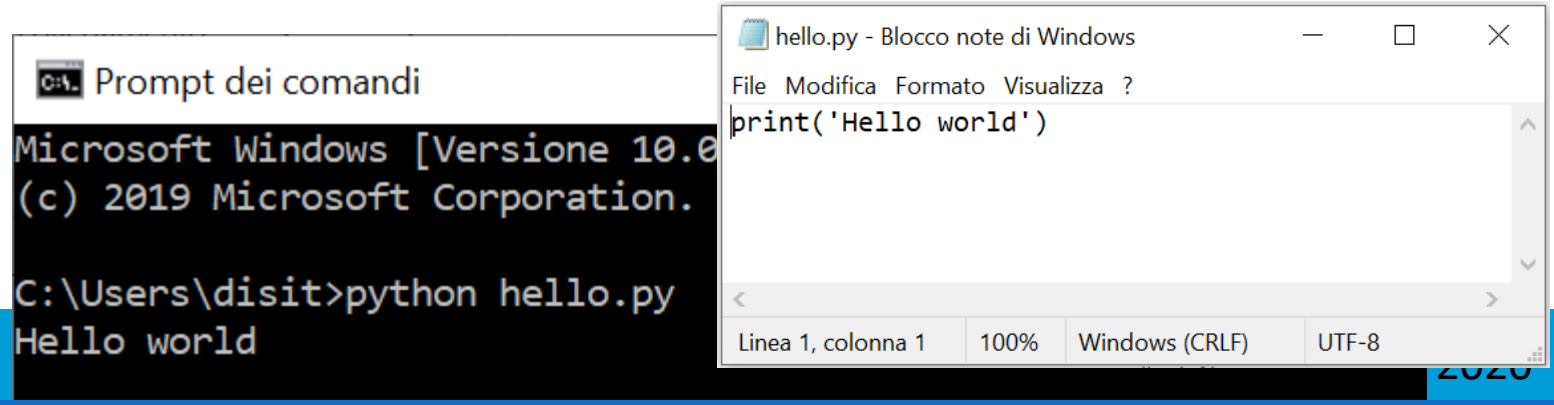
- La prima linea di questo esempio è il comando che fa partire l'interprete Python in ambiente Windows e può cambiare leggermente a seconda del sistema operativo utilizzato. Le due righe successive sono semplici informazioni di copyright del programma

Il linguaggio di programmazione Python (5)

- La terza riga inizia con >>>: questa è l'indicazione (chiamata "prompt") che l'interprete usa per indicare la sua disponibilità ad accettare comandi
- Il comando inserito serve a stampare la frase 'Hello world'

```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>>
```

- In alternativa alla riga di comando si può scrivere un programma in un file (detto script) ed usare l'interprete per eseguire il contenuto del file
- Nell'esempio seguente è stato usato un editor di testi per creare un file chiamato hello.py:



Il linguaggio di programmazione Python (6)

- lavorare da linea di comando è conveniente per lo sviluppo e per il test del programma perchè si possono inserire ed eseguire immediatamente singole righe di codice
- Quando si ha un programma funzionante lo si dovrebbe salvare in uno script per poterlo eseguire o modificare in futuro senza doverlo riscrivere da capo ogni volta
- Tutto ciò che viene scritto in modalità “linea di comando” è irrimediabilmente perso nel momento in cui usciamo dall’ambiente Python

Cos'è un Programma? (1)

- Un programma è una sequenza di istruzioni che specificano come effettuare una elaborazione
- L'elaborazione può essere sia di tipo matematico (per esempio la soluzione di un sistema di equazioni o il calcolo delle radici di un polinomio) che simbolico (per esempio la ricerca e sostituzione di un testo in un documento)
- I dettagli sono diversi per ciascun linguaggio di programmazione, ma un piccolo gruppo di istruzioni è praticamente comune a tutti:
  - **input**: ricezione di dati da tastiera, da file o da altro dispositivo
  - **output:** scrittura di dati su video, su file o trasmissione ad altro dispositivo
  - **matematiche**: esecuzione di semplici operazioni matematiche, quali l'addizione e la sottrazione
  - **condizionali**: controllo di alcune condizioni ed esecuzione della sequenza di istruzioni appropriata
  - **ripetizione**: ripetizione di un'azione, di solito con qualche variazione

Cos'è un Programma? (2)

- Ogni programma che hai usato per quanto complesso possa sembrare è costituito da istruzioni che assomigliano a queste
- Si può affermare che la programmazione altro non è che la suddivisione di un compito grande e complesso in una serie di sotto-compiti via via più piccoli, finché questi sono sufficientemente semplici da essere eseguiti da una di queste istruzioni fondamentali

Linguaggi formali e naturali (1)

- I linguaggi naturali sono le lingue parlate, tipo l'inglese, l'italiano, lo spagnolo. Non sono stati "progettati" da qualcuno e anche se è stato imposto un certo ordine nel loro sviluppo si sono evoluti naturalmente
- I linguaggi formali sono linguaggi progettati per specifiche applicazioni
- Ad esempio, la notazione matematica è un linguaggio formale particolarmente indicato ad esprimere relazioni tra numeri e simboli:
  - i chimici usano un linguaggio formale per rappresentare la struttura delle molecole
  - i linguaggi di programmazione sono linguaggi formali che sono stati progettati per esprimere elaborazioni
- I linguaggi formali tendono ad essere piuttosto rigidi per quanto riguarda la sintassi:
  - 3 + 3 = 6 è una dichiarazione matematica sintatticamente corretta, mentre 3 = ÷6$ non lo è
  - H_2O, è un simbolo chimico sintatticamente corretto contrariamente a $_2Zz$.

Linguaggi formali e naturali (2)

- Le regole sintattiche si possono dividere in due categorie: la prima riguarda i **token**, la seconda la **struttura**
- I **token** sono gli elementi di base del linguaggio (quali possono essere le parole in letteratura, i numeri in matematica e gli elementi chimici in chimica)
  - Uno dei problemi con 3 = ÷6$ è che $ non è un token valido in matematica
  - $_2$Zz non è valido perché nessun elemento chimico è identificato dal simbolo Zz
- Il secondo tipo di regola riguarda la **struttura** della dichiarazione, cioè il modo in cui i token sono disposti
  - La dichiarazione 3 = ÷6$ è strutturalmente non valida perché un segno ÷ non può essere posto immediatamente dopo un segno =
  - Allo stesso modo l'indice nelle formule chimiche deve essere indicato dopo il simbolo dell'elemento chimico, non prima, e quindi l'espressione $_2$Zz non è valida

Linguaggi formali e naturali (3)

- Quando si legge una frase in italiano o una dichiarazione in un linguaggio formale si deve capire quale sia la struttura della dichiarazione
- Questo processo (chiamato parsing) in un linguaggio naturale viene realizzato in modo inconscio e spesso non ci si rende conto della sua intrinseca complessità
- Per esempio, quando si sente la frase “La scarpa ` e caduta”, si capisci che “la scarpa” è il soggetto e che “` e caduta” ` e il verbo
- Una volta analizzata la frase, si può capire cosa essa significa (cioè la semantica della frase). Partendo dal presupposto che tu sappia cosa sia una “scarpa” e cosa significhi “cadere” riesci a comprendere il significato generale della frase

Linguaggi formali e naturali (4)

- Anche se i linguaggi formali e quelli naturali condividono molte caratteristiche (token, struttura, sintassi e semantica) ci sono tuttavia molte differenze:
  - **Ambiguità**: i linguaggi naturali ne sono pieni ed il significato viene ottenuto anche grazie ad indizi ricavati dal contesto. I linguaggi formali sono progettati per essere completamente non ambigui, ciò significa che ciascuna dichiarazione ha esattamente un significato, indipendente dal contesto
  - **Ridondanza**: per evitare l'ambiguità e ridurre le incomprensioni i linguaggi naturali impiegano molta ridondanza. I linguaggi formali sono meno ridondanti e più concisi
  - **Letteralità**: i linguaggi naturali fanno uso di paragoni e metafore, e possiamo parlare in termini astratti intuendo immediatamente che ciò che sentiamo ha un significato simbolico. I linguaggi formali invece esprimono esattamente ciò che dicono

Linguaggi formali e naturali (5)

- In un certo senso la differenza tra linguaggi naturali e formali è come quella esistente tra poesia e prosa, ma in misura decisamente più evidente:
  - **Poesia**: le parole sono usate tanto per il loro suono che per il loro significato, e la poesia nel suo complesso crea un effetto o una risposta emotiva. L'ambiguità è non solo frequente, ma spesso addirittura cercata
  - **Prosa**: il significato delle parole è estremamente importante, con la struttura che contribuisce a fornire maggior significato. La prosa può essere soggetta ad analisi più facilmente della poesia, ma può risultare ancora ambigua
  - **Programmi**: il significato di un programma per computer è non ambiguo e assolutamente letterale, può essere compreso nella sua interezza con l'analisi dei token e della struttura.

Python: Tipi di dati

Valori e tipi (1)

- Per formalizzare un problema e scrivere un programma capace di risolverlo, è necessario gestire una serie di dati
- I calcolatori elaborano valori che rappresentano informazioni e possono essere di tipi diversi
- In un programma Python, ciascun dato è di uno specifico tipo
- Il TIPO: è caratterizzato dai valori che può rappresentare e dalle operazioni che possono essere effettuate su tali valori
- Un tipo di dato che viene messo a disposizione dal linguaggio stesso viene chiamato ***tipo di dato primitivo***
- Python fornisce supporto per parecchi tipi di dati: numeri, stringhe di testo, file, contenitori e molti altri

Valori e tipi (2)

- Vediamone alcuni:
  - int (interi)
  - float (numeri a virgola mobile o floating point)
  - str (stringhe)
- Si vedranno in futuro altri tipi di dati…
- I programmatori possono definire ulteriori tipi di dato (che vedremo)

NOTA: Una stringa è una sequenza di caratteri come ‘Hello World!’

Valori e tipi (3)

- Alcuni esempi:
  - **int** (interi):
    - Devo tenere in considerazione il numero di ruote di una macchina 4 ➡
    - Devo chiedere l'età ad un cliente ➡ 30
    - Devo sapere la temperatura in °C in montagna ➡ -6
  - **float**
    - Devo scrivere il risultato di una divisione per due di un numero dispari ➡ 5/2 = 2.5
  - **str**
    - Devo scrivere il nome di una persona ➡ 'Marta'
    - Devo far comparire un messaggio in una pagina web
      - 'Hello World!'

Valori e tipi (3)

- Se non si è sicuri del tipo di un valore usato, è possibile chiederlo all'interprete:

```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(1+1)
2
>>> type('Hello World')
<class 'str'>
>>> type(2)
<class 'int'>
>>> type(2.4)
<class 'float'>
>>> type('18')
<class 'str'>
>>>
```

Valori e tipi (4)

- Un valore è una delle cose fondamentali manipolate da un programmatore, come lo sono una lettera dell'alfabeto nella scrittura o un numero in matematica
- I valori che abbiamo visto finora sono ad esempio "Hello, World!" e 2, quest'ultimo visto anche come risultato ottenuto quando abbiamo sommato 1+1

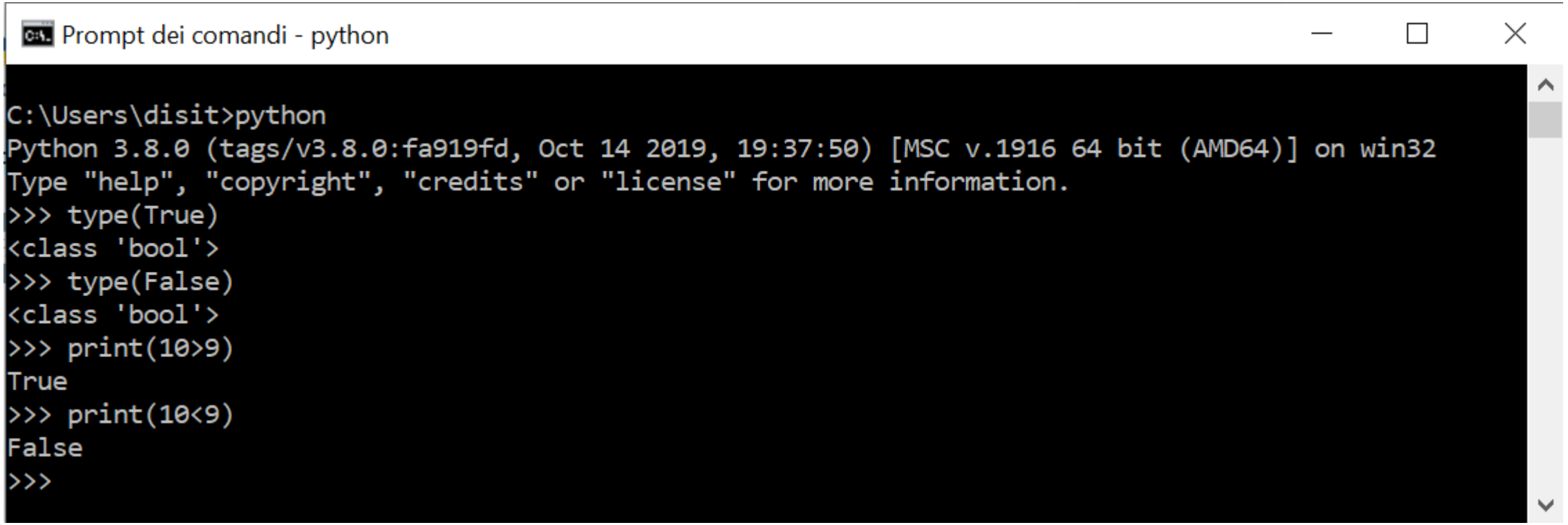
```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>>
```

```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(1+1)
2
>>> _
```

- Questi valori appartengono a tipi diversi: 2 è un intero, e "Hello, World!" ` e una stringa, così chiamata perché contiene una serie (o "stringa") di caratteri

Boolean

- Sono stati introdotti con L'Algebra di Boole
- In Python vengono espressi tramite True o False



Variabili e Costanti

- **COSTANTE**: è un valore che NON varia nel corso della computazione
  - Si usa quando nel formalizzare e risolvere il problema si ha la necessità che ad un elemento usato nella computazione sia associato (‘assegnato’) sempre lo stesso valore
  - Si usa quando NON deve essere possibile modificare tale elemento neanche per errore nel contesto della computazione in esame
- **VARIABILE**: al contrario della costante, è un valore che VARIA nel corso della computazione
- In Python, a livello sintattico, NON c’è distinzione tra COSTANTE e VARIABILE, è il programmatore che DEVE sapere cosa sta usando
- Solitamente si usano lettere MAIUSCOLE per dare i nomi alle costanti

Costanti: esempi

- Esempi
  - il numero di ruote di una macchina avrà sempre valore 4
  - Il mio codice fiscale NON potrà cambiare
  - Il mese di Aprile avrà sempre 30 giorni

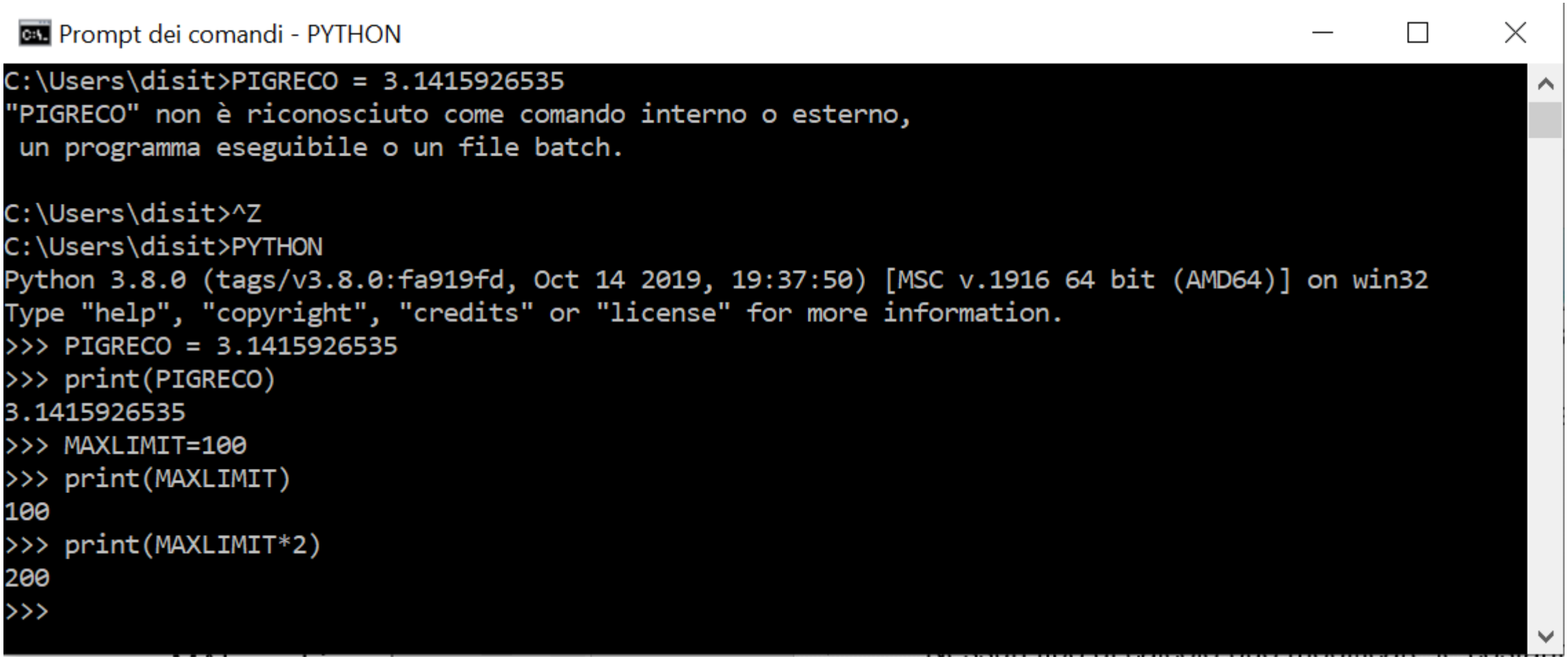
  - Se voglio calcolare la circonferenza e l'area di un cerchio supponendo di usare le prime 10 cifre del pigreco:
    - Associo alla costante '**pigreco**' tali cifre che NON dovranno MAI cambiare nel corso del mio programma
    - Poi effettuo i vari calcoli usando la costante '**pigreco**'

Uso delle Costanti in Python (1)

- Ad esempio, Se nel mio programma scrivo:
  MAXLIMIT =100
  PIGRECO = 3.1415926535
- Se gestisco MAXLIMIT e PIGRECO come costanti, allora significa che:
  - Al nome PIGRECO associo il valore 3.1415926535, ovvero uso solo le prime 10 cifre dopo la virgola (approssimazione)
  - Tutte le volte che il programmatore scrive 'PIGRECO' nel programma, automaticamente il compilatore sostituirà il valore assegnato che in questo caso è: 3.1415926535
  - Tutte le volte che il programmatore scrive 'MAXLIMIT' nel programma, automaticamente il compilatore sostituirà il valore assegnato che in questo caso è: 100

Uso delle Costanti in Python (2)

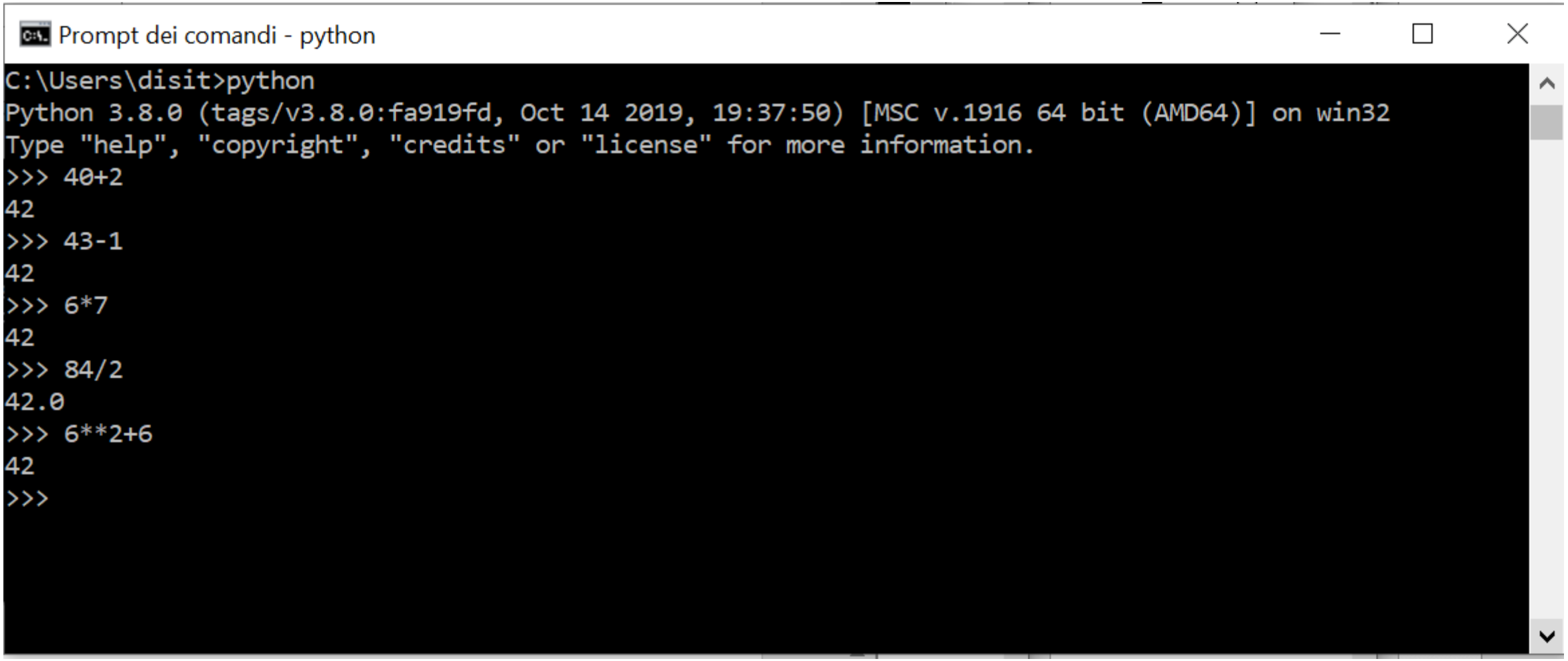
- Vediamo cosa succede da riga di comando…



Operatori Aritmetici (1)

- Python dispone di una serie di operatori
- Gli Operatori sono simboli speciali che rappresentano i calcoli fondamentali come addizione, moltiplicazione, etc.
- Gli operatori +, -, *, / eseguono nell'ordine le operazioni aritmetiche di addizione, sottrazione e moltiplicazione, divisione
- L'operatore **, esegue l'elevamento a potenza, ovvero calcola la potenza di un numero

Operatori Aritmetici (2)



Uso delle Costanti e degli Operatori in Python (1)

- Supponiamo di voler calcolare la somma del numero di ruote di 2 macchine e 3 moto. E' possibile:
  - definire come costanti il numero di ruote di ogni tipo di veicolo (che ovviamente NON devono cambiare nel corso della computazione)
  - effettuare il calcolo

RMACCHINA = 4

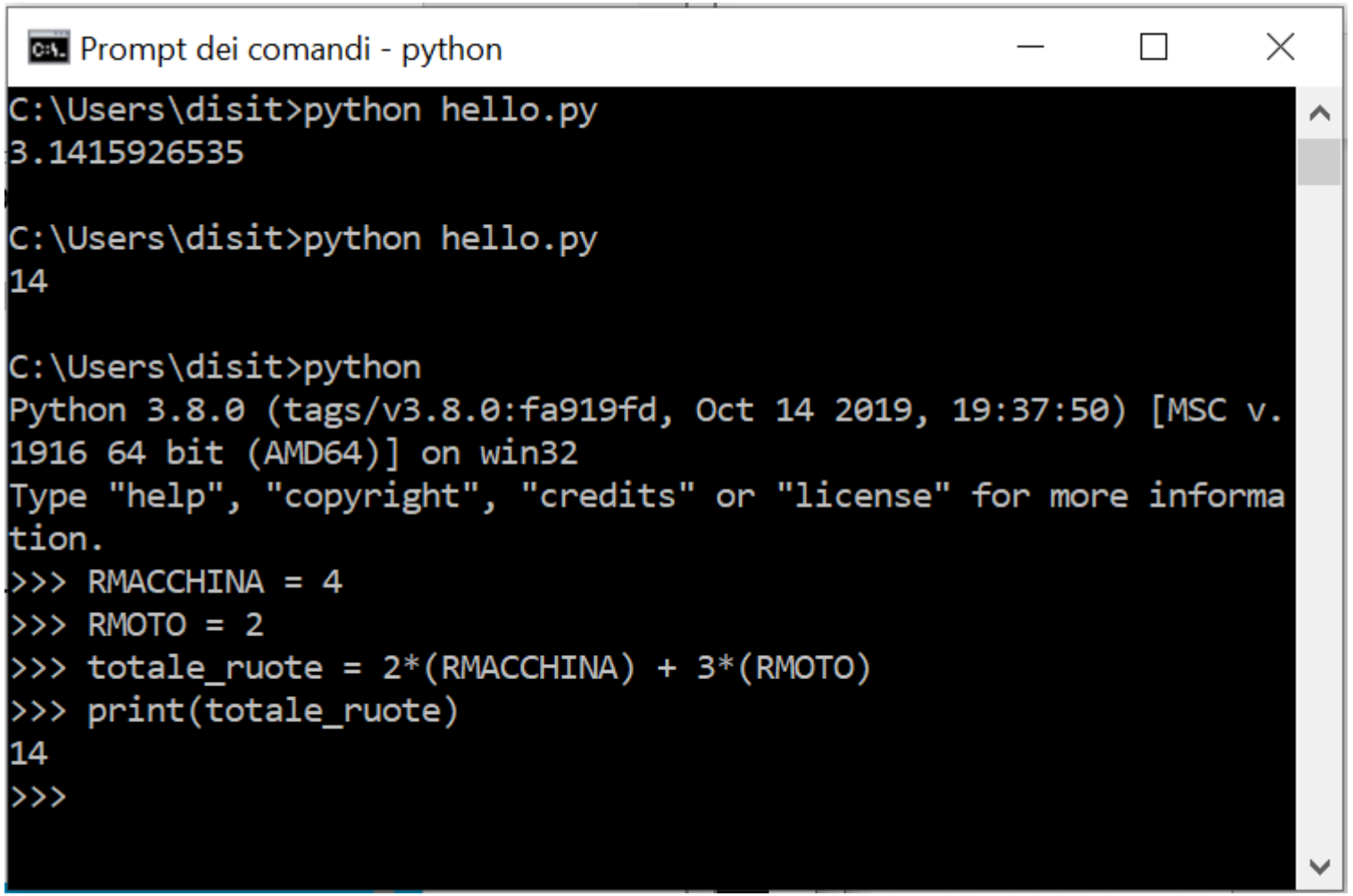
RMOTO = 2

totale_ruote = 2*(**RMACCHINA**) + 3*(**RMOTO**)

- Il compilatore sostituisce i valori impostati all'inizio dal programmatore
- Quindi, verrà eseguita la seguente espressione:
  - totale_ruote = 2*(4) + 3*(2) = 14

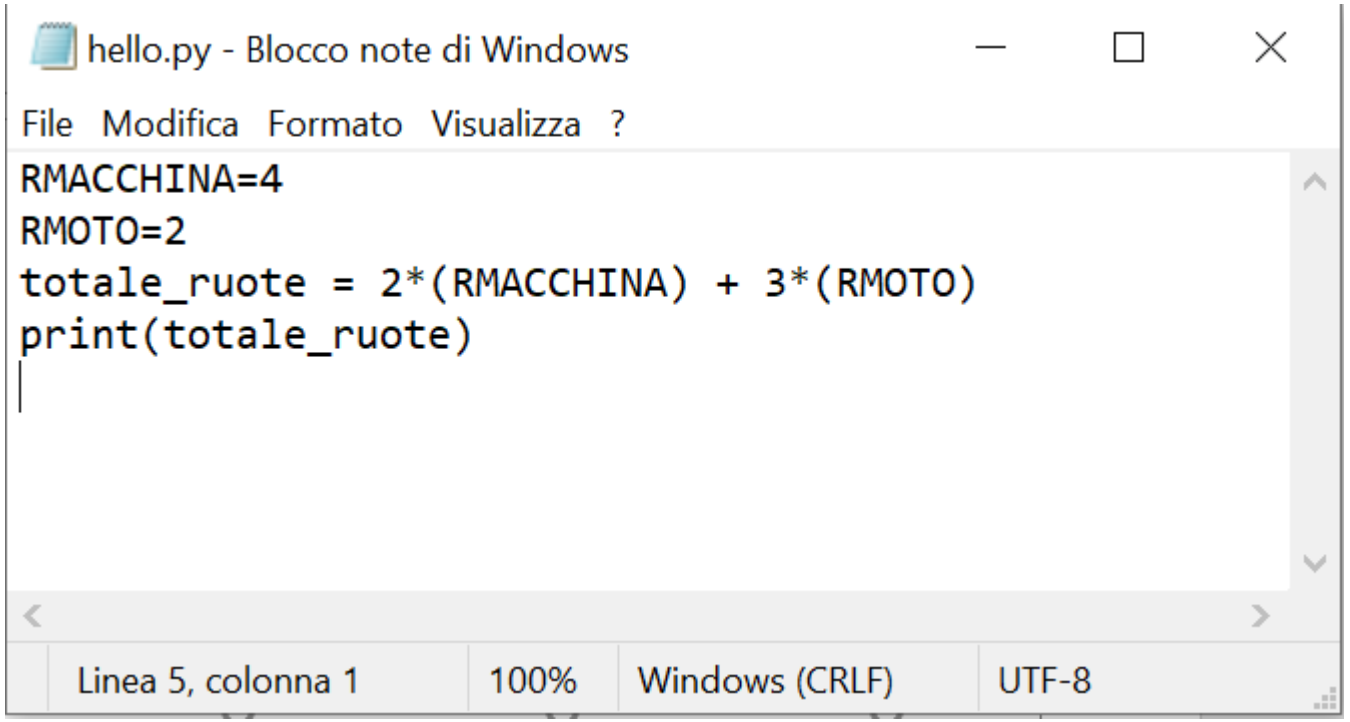
Uso delle Costanti in Python (3) – riga di comando

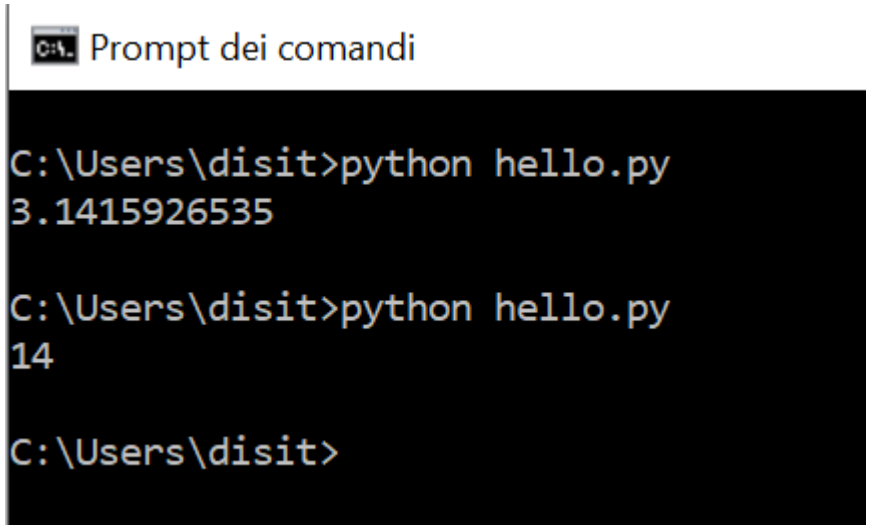
- Da riga di comando si scrive una riga per volta
- Per stampare il risultato su Console, si usa la print(«ciò che si vuole stampare»)
- print() è una funzione Python
- Il concetto di funzione verrà approfondito più avanti nel corso



Uso delle Costanti in Python (4) – file.py

- Si scrive il programma in un file di testo (estensione py)
- Esempio nome file ‘hello.py’
- Da riga di comando, si esegue il programma (richiamando il nome del file), in questo modo il compilatore legge ed esegue tutte le righe di codice contenute nel file





Concetto di Variabile (1)

- Quando si scrive un programma si ha la necessità di usare anche altre tipologie di elementi oltre alle COSTANTI. Ad esempio cosa succede se:
  - Si deve scrivere la data attuale su una pagina web?
  - Si devono visualizzare le previsioni del tempo per domani?
  - Si vuole scrivere l'algoritmo per calcolare il codice fiscale di una persona?
- In questi casi avrò bisogno di ricorrere al concetto di **Variabile**

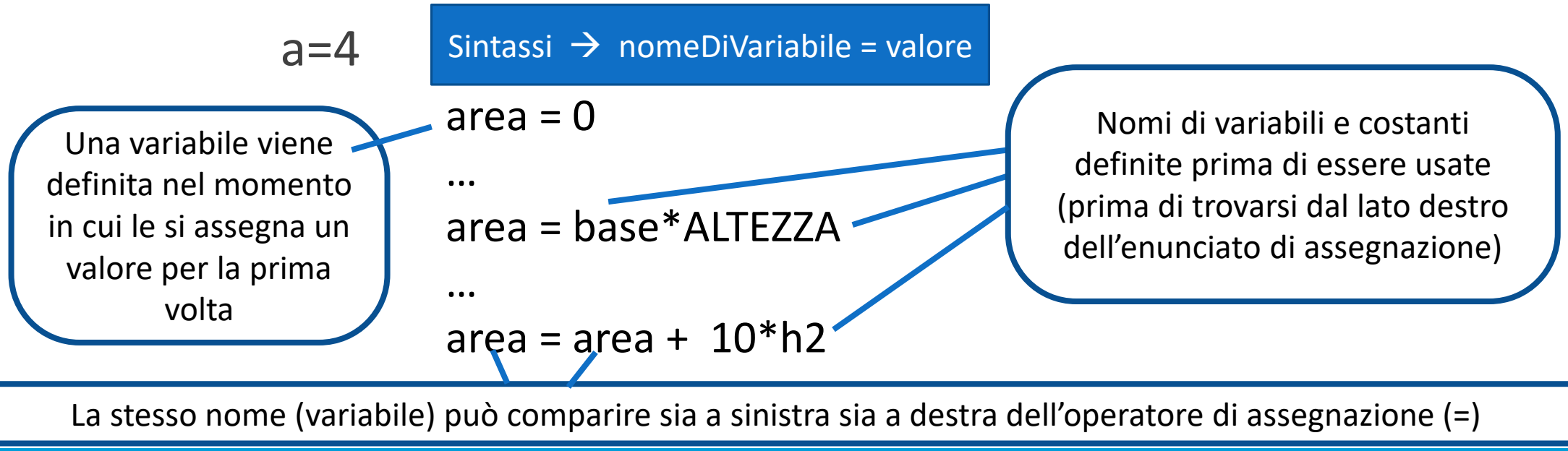
Valori e tipi (2)

- In Python una variabile può memorizzare valori di qualsiasi tipo: il tipo di dato è associato al valore NON alla variabile. Ad esempio la variabile seguente viene inizializzata con un valore di tipo int:
- altezza = 100

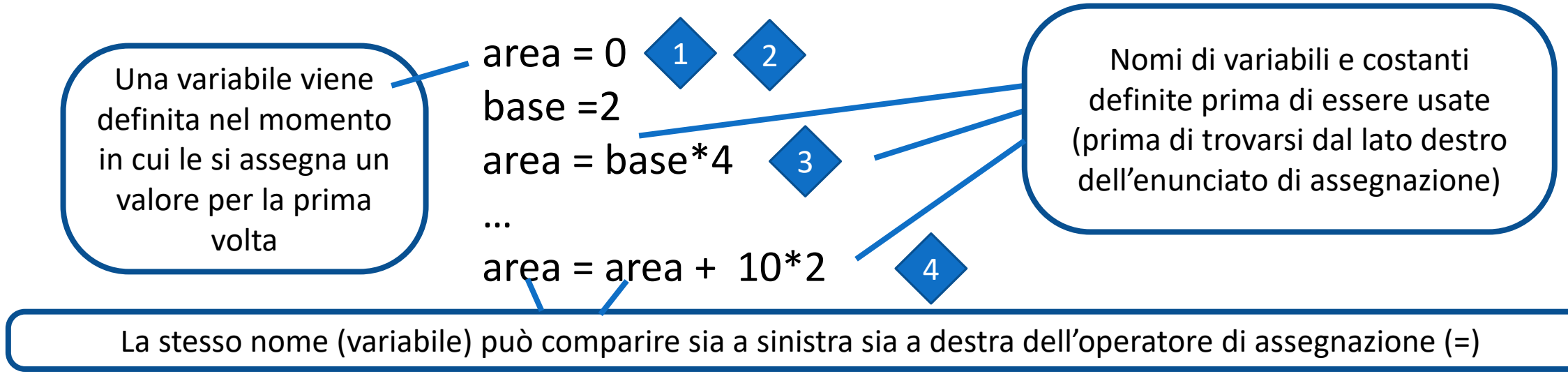
- In seguito la stessa variabile può contenere un valore di tipo diverso ad esempio un float (numero a virgola mobile):
- altezza = 98.5
- Oppure potrebbe contenere una stringa:
- Altezza = ‘1 metro e 78 centimentri’

Variabili e Costanti: enunciato di assegnazione

- Variabili e costanti sono una posizione nella memoria del computer, in cui si possono archiviare informazioni durante l'esecuzione di un programma
- Si può pensare ad una variabile come ad un posto auto in un parcheggio: il posto auto ha una informazione di identificazione (Es: 'A34') e può contenere un veicolo
- Allo stesso modo una variabile ha un nome (es: 'area') e può contenere un valore (ad esempio il numero 4). Per memorizzare un valore, si usa l'enunciato di **assegnazione** nel modo seguente:



Variabili e Costanti: enunciato di assegnazione (2)



1 **area =** ☐ Si tratta della prima assegnazione: la variabile viene creata

2 **area =** **0** La variabile viene inizializzata al valore intero 0

3 **area =** **6** La seconda assegnazione sovrascrive il valore memorizzato

Nomi delle Variabili (1)

- Il **nome** di una variabile deve rispettare le seguenti regole:
  - È una qualunque sequenza di caratteri alfabetici, numerici
  - Non si possono usare simboli o spazio, è possibile usare il segno di underscore '_'
  - Il primo carattere non può essere un numero
- Python è CASE SENSITIVE
  - QUINDI le variabili ‘area’ e ‘Area’ SONO DIVERSE!!

- Esempi corretti:
  - a1
  - totale_ruote
  - totaleRuote
  - area
  - circonferenza

- Esempi errati:
  - $a1
  - totale ruote
  - 2r
  - ruot@

Nomi delle Variabili (2)

- In Python ci sono alcune ‘parole riservate’ che definiscono le regole del linguaggio stesso
- Tali parole non possono essere usate come nomi di variabili e sono le seguenti:

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

- Come buona prassi è bene dare i nomi significativi alle variabili, in modo da documentare a cosa servano

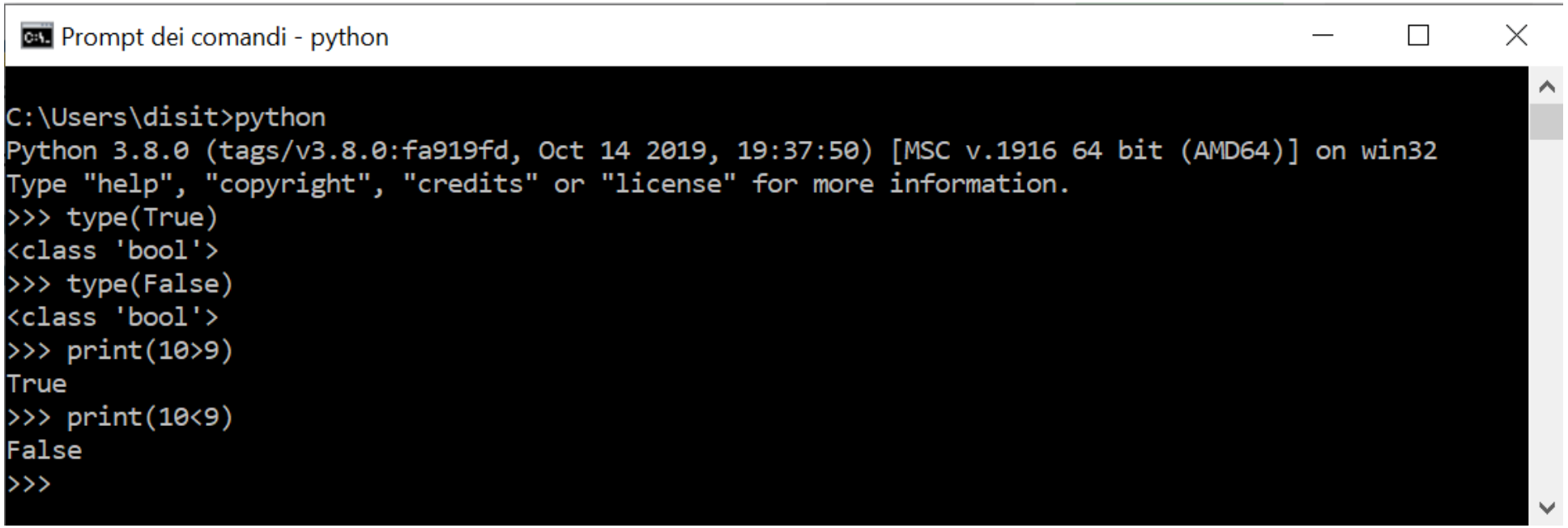
Concetto di Istruzione

- ‘In informatica, elemento della programmazione con cui si richiede al computer, attraverso un codice prestabilito, l’esecuzione di una determinata operazione (leggere dati in ingresso, effettuare un calcolo, una selezione, ecc.)’, Treccani

- ‘Con il termine istruzione, in informatica, si intende il comando impartito ad un esecutore (processore) utilizzando un linguaggio ad esso comprensibile’, wikipedia

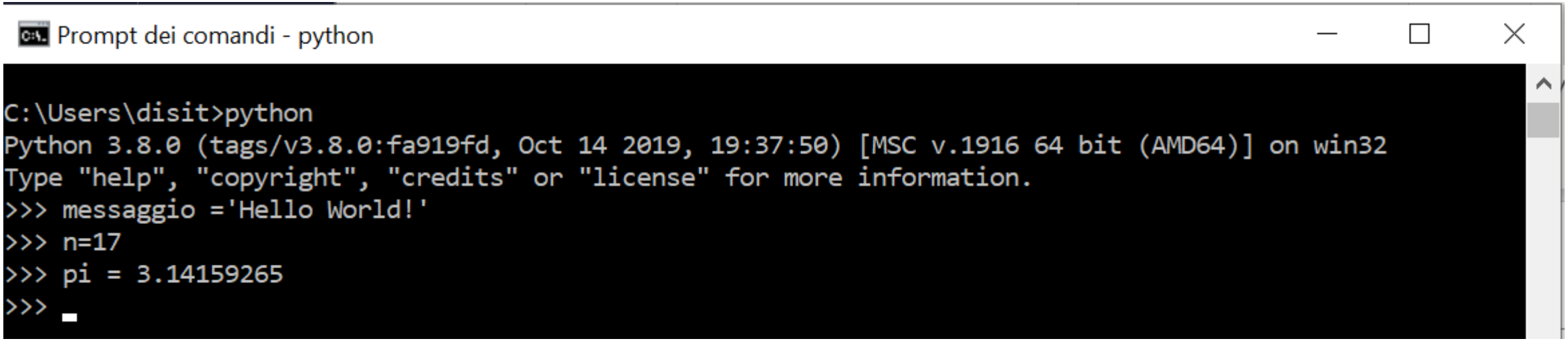
Istruzione

- Un'istruzione è un'operazione che l'interprete Python può eseguire
- Quando si scrive una istruzione sulla riga di comando, Python la esegue e se previsto stampa il risultato a video
- Uno script di solito contiene una sequenza di istruzioni: se sono presenti più istruzioni i loro risultati appariranno via via che le singole istruzioni saranno eseguite



Istruzioni di Assegnazione (1)

- Una istruzione di assegnazione creare una nuova variabile assegnandole il valore



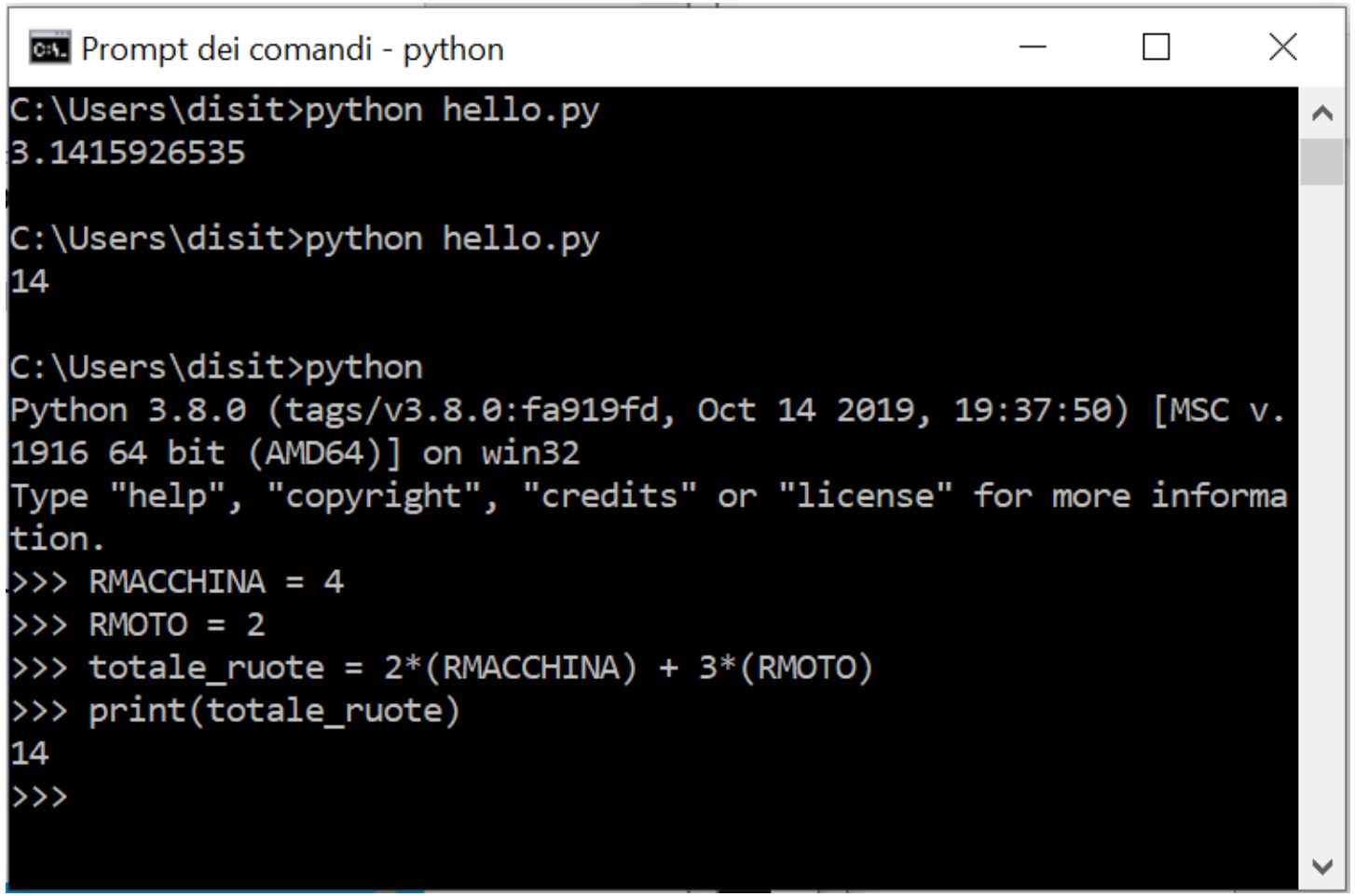
- L'esempio effettua tre assegnazioni:
  - La prima assegna una stringa ad una nuova variabile di nome messaggio
  - La seconda assegna il numero intero 17 alla variabile n
  - La terza assegna il valore decimale approssimato di pigreco alla variabile pi

DIAGRAMMA DI STATO

messaggio	→	Hello World!
n	→	17
pi	→	3.14159265

Istruzioni di Assegnazione (2)

Altri esempi…



Espressioni e Istruzioni

- Un'espressione è una combinazione di valori, variabili e operatori
- Un valore è considerato già di per sé un'espressione, come pure una variabile, per cui quelle che seguono sono tutte delle espressioni valide (supponendo che alla variabile x sia già stato assegnato un valore):
  - 17
  - x
  - X+17
- Un'istruzione è una porzione di codice che l'interprete Python può eseguire
- Abbiamo già visto due tipi di istruzioni: istruzioni di stampa (print) e istruzioni di assegnazione
- Tecnicamente, un'espressione è anche un'istruzione, ma è forse più semplice considerarle come cose distinte
- La differenza fondamentale è che l'espressione contiene un valore, l'istruzione no.

Modalità interattiva e modalità script (1)

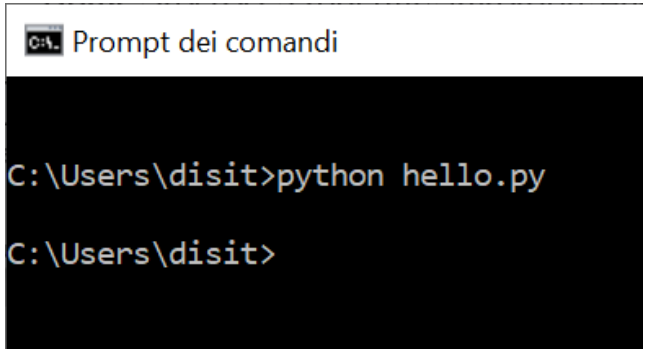
- Uno dei vantaggi di lavorare con un linguaggio interpretato è quello di poter provare pezzi di codice in modalità interattiva prima di inserirli in uno script
- Ma tra le due modalità, ci sono delle differenze che possono disorientare
- Per esempio, usando Python come una calcolatrice, potreste scrivere:

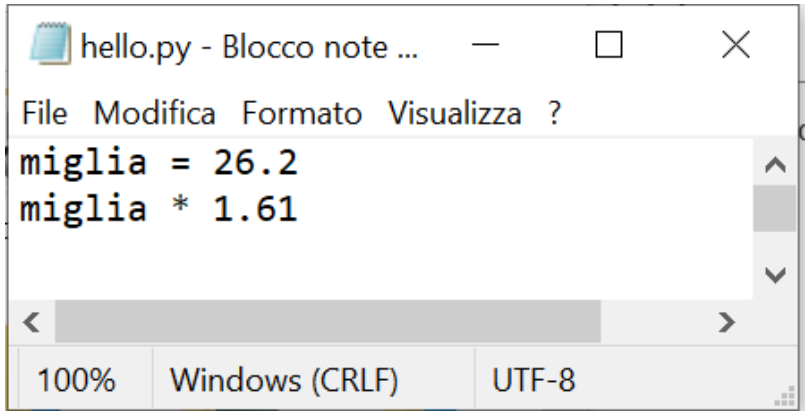
```
>>> miglia = 26.2
>>> miglia * 1.61
42.182
>>>
```

- La prima riga assegna un valore a miglia, e non ha alcun effetto visibile. La seconda riga è un'espressione, e l'interprete la valuta e ne mostra il risultato. Vediamo così che una maratona misura circa 42 chilometri
- Ma se scrivete lo stesso codice in uno script e lo eseguite, non otterrete alcun riscontro. In modalità script, un'espressione, di per sé, non ha effetti visibili

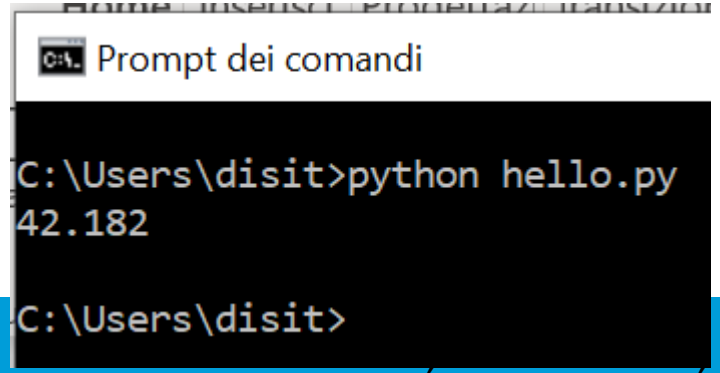
Modalità interattiva e modalità script (2)

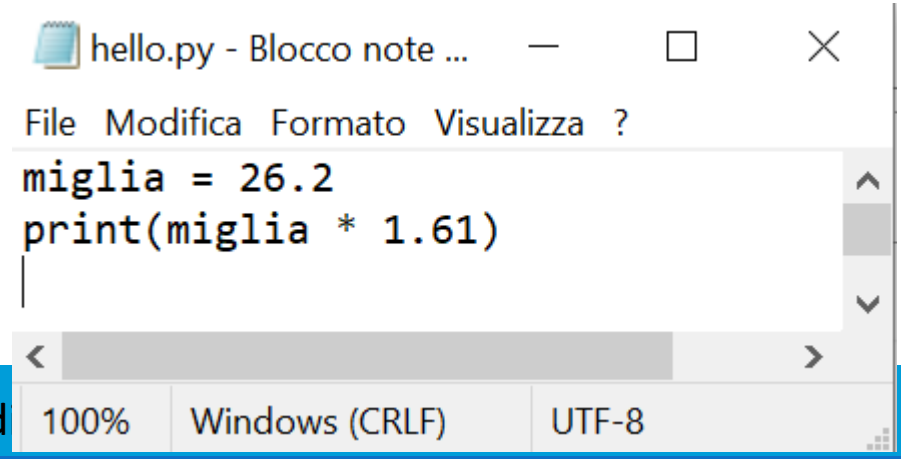
- Se si scrive lo stesso codice precedente (usato nella modalità interattiva) in uno script e si esegue, non si ottiene alcun riscontro
- In modalità script, un'espressione, di per sé, non ha effetti visibili





- In realtà Python valuta l'espressione, ma non ne mostra il risultato finché non gli si dice esplicitamente di farlo:





Modalità interattiva e modalità script (3)

- Uno script di solito contiene una sequenza di istruzioni
- Se ci sono più istruzioni, i risultati compaiono man mano che le istruzioni vengono eseguite
- Per esempio lo script:

```
print(1)
x = 2
print(x)
```

- visualizza questo:

```
1
2
```

- mentre l'istruzione di assegnazione (x=2) non produce alcun output sullo schermo.

Ordine delle operazioni

- Quando un'espressione contiene più operatori, l'ordine in cui viene eseguito il calcolo dipende dalle regole di precedenza
- Python segue le stesse regole di precedenza usate in matematica
- L'acronimo PEMDAS è un modo utile per ricordare le regole:
  - **P**arentesi: hanno il più alto livello di precedenza e possono essere usate per far valutare l'espressione in qualsiasi ordine volete. Dato che le espressioni tra parentesi sono valutate per prime, 2 * (3-1) fa 4, e (1+1)**(5-2) fa 8. Si possono usare le parentesi anche solo per rendere più leggibile un'espressione, come in (minuti * 100) / 60; in questo caso non influiscono sul risultato.
  - **E**levamento a potenza: ha la priorità successiva, così 2**1+1 fa 3, e non 4, e 3*1**3 fa 3, e non 27
  - **M**oltiplicazione e **D**ivisione hanno la stessa priorità, superiore ad **A**ddizione e **S**ottrazione, anch'esse aventi la stessa priorità. 2*3-1 fa 5, e non 4, e 6+4/2 fa 8, e non 5.
  - Gli operatori con la stessa priorità sono valutati da sinistra verso destra (eccetto la potenza), così nell'espressione gradi / 2 * pi, la divisione viene calcolata per prima e il risultato viene moltiplicato per pi. Per dividere per 2π, dovete usare le parentesi o scrivere gradi / 2 / pi.

Operazioni sulle stringhe (1)

- In genere non potete effettuare operazioni matematiche sulle stringhe, anche se il loro contenuto sembra essere un numero, quindi gli esempi che seguono non sono validi:
  - '2'-'1’
  - 'uova'/'facili’
  - 'terzo'*'una magia’
- L’operatore + invece funziona con le stringhe, anche se non si comporta nel modo consueto: in questo caso esegue il concatenamento, cioè unisce le stringhe collegandole ai due estremi
- Per esempio:

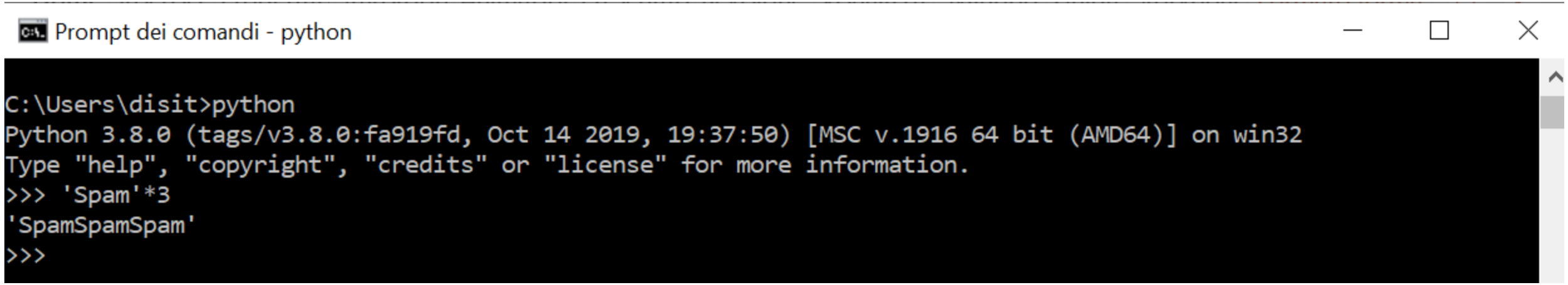
```
primo = 'bagno’
secondo = 'schiuma’
print(primo + secondo)
```

- Il risultato a video di questo programma è:

```
bagnoschiuma
```

Operazioni sulle stringhe (2)

- Anche l'operatore * funziona sulle stringhe: ne esegue la ripetizione. Per esempio:



- Uno degli operandi deve essere una stringa, l'altro un numero intero
- Questo utilizzo di + e * è coerente per analogia con l'addizione e la moltiplicazione in matematica
- Così come 4*3 è equivalente a 4+4+4, ci aspettiamo che 'Spam'*3 sia lo stesso di 'Spam'+'Spam'+'Spam', ed effettivamente è così

Commenti

- Man mano che il programma cresce di dimensioni e diventa più complesso, diventa anche sempre più difficile da leggere
- I linguaggi formali sono ricchi di significato, e può risultare difficile capire a prima vista cosa fa un pezzo di codice o perché è stato scritto in un certo modo
- Per questa ragione, è buona abitudine aggiungere delle note ai vostri programmi, per spiegare in linguaggio naturale cosa sta facendo il programma nelle sue varie parti
- Queste note sono chiamate commenti, e sono demarcate dal simbolo #:

```
# calcola la percentuale di ora trascorsa
percentuale = (minuti * 100) / 60
```

- Qualsiasi cosa scritta dopo il simbolo # e fino alla fine della riga, viene trascurata e non ha alcun effetto sull'esecuzione del programma. I commenti più utili sono quelli che documentano caratteristiche del codice di non immediata comprensione. È ragionevole supporre che chi legge il codice possa capire cosa esso faccia; è molto più utile spiegare perché.
- Dei buoni nomi di variabile possono ridurre la necessità di commenti, ma nomi lunghi possono complicare la lettura, pertanto va trovato un giusto compromesso

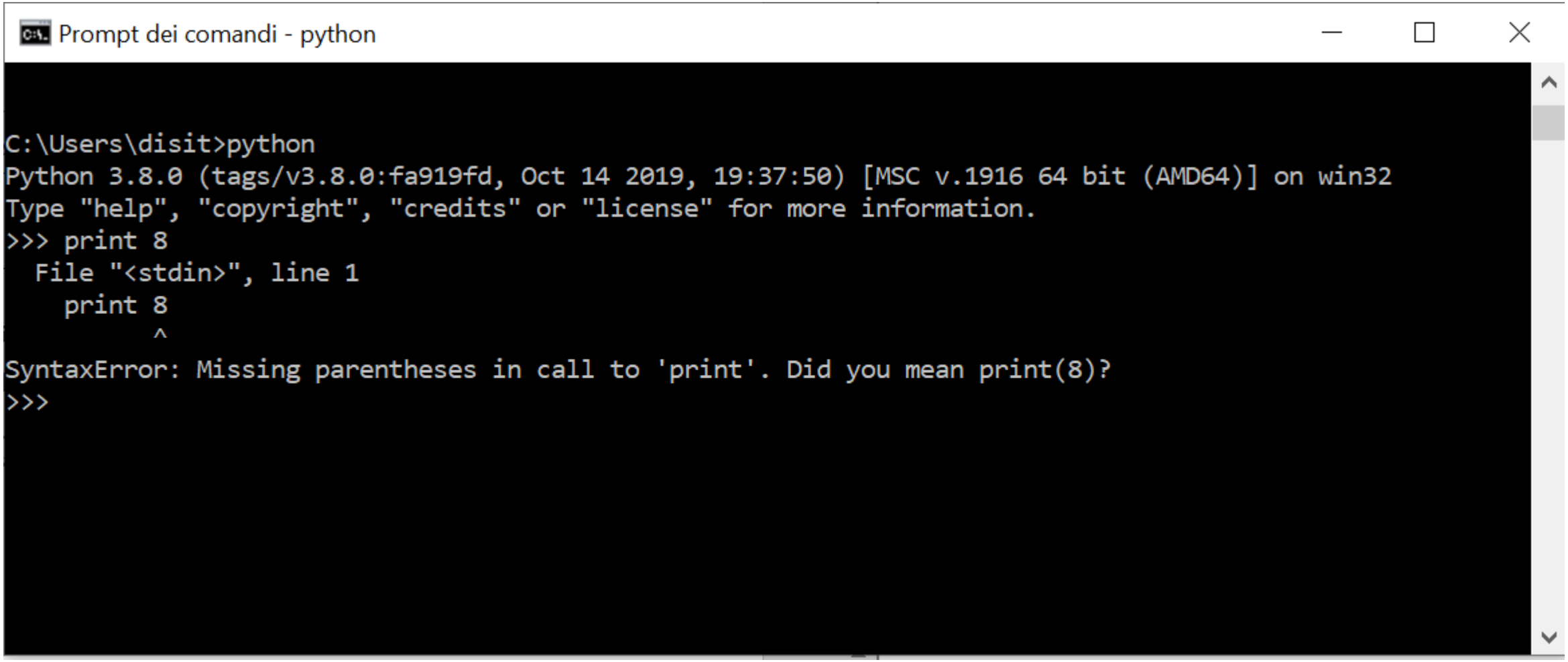
Cos'è il debug? (1)

- La programmazione è un processo complesso e dato che è fatto da esseri umani spesso comporta errori
- Gli errori di programmazione sono chiamati bug ed il processo della loro ricerca e correzione è chiamato debug
- Sono tre i tipi di errore nei quali si incorre durante la programmazione:
  - gli errori di sintassi
  - gli errori in esecuzione
  - gli errori di semantica

Errori di sintassi (1)

- Python può eseguire un programma solo se il programma è sintatticamente corretto, altrimenti l'elaborazione fallisce e l'interprete ritorna un messaggio d'errore
- La sintassi si riferisce alla struttura di un programma e alle regole concernenti la sua struttura
- In italiano, per fare un esempio, una frase deve iniziare con una lettera maiuscola e terminare con un punto
  - […]. questa frase contiene un errore di sintassi. E anche questa
- Per la maggior parte dei lettori qualche errore di sintassi non è un problema significativo.
- Python non è così permissivo: se c'è un singolo errore di sintassi da qualche parte nel programma Python stamperà un messaggio d'errore e ne interromperà l'esecuzione, rendendo impossibile proseguire

Errori di sintassi (2)



Errori in esecuzione

- Il secondo tipo di errore è l'errore in esecuzione (o "runtime"), così chiamato perché l'errore non appare finché il programma non è eseguito
- Questi errori sono anche chiamati eccezioni perché indicano che è accaduto qualcosa di eccezionale nel corso dell'esecuzione (per esempio si è cercato di dividere un numero per zero)
- Gli errori in esecuzione sono rari nei semplici programmi iniziali, li troveremo più probabilmente, in futuro

Errori di semantica

- Se c'è un errore di semantica il programma verrà eseguito senza problemi nel senso che il computer non genererà messaggi d'errore durante l'esecuzione, ma il risultato non sarà ciò che ci si aspettava
- Sarà qualcosa di diverso, e questo qualcosa è esattamente ciò che è stato detto di fare al computer
- Il problema sta nel fatto che il programma che è stato scritto non è quello che si desiderava scrivere:
  - il significato del programma (la sua semantica) è sbagliato
  - L'identificazione degli errori di semantica è un processo complesso perché richiede di lavorare in modo inconsueto, guardando i risultati dell'esecuzione e cercando di capire cosa il programma ha fatto di sbagliato per ottenerli.

Debug sperimentale (1)

- Una delle più importanti abilità di un programmatore è la capacità di effettuare il debug (o "rimozione degli errori")
- Sebbene questo possa essere un processo frustrante è anche una delle parti più intellettualmente vivaci, stimolanti ed interessanti della programmazione.
- In un certo senso il debug può essere paragonato al lavoro investigativo.
- Il programmatore è messo di fronte agli indizi e deve ricostruire i processi e gli eventi che hanno portato ai risultati che hai ottenuto.

Debug sperimentale (2)

- Il debug è una scienza sperimentale: dopo che il programmatore si è fatto una idea di ciò che può essere andato storto, deve modificare il programma e lo provarlo (testarlo) ancora.
- Se l'ipotesi era corretta allora può il programmatore predire il risultato della modifica e avvicinarsi di un ulteriore passo all'avere un programma funzionante
- Se l'ipotesi era sbagliata deve cercarne un'altra
- Per qualcuno la programmazione e il debug sono la stessa cosa, intendendo con questo che la programmazione è un processo di rimozione di errori finchè il programma fa ciò che ci si aspetta. L'idea ` e che si dovrebbe partire da un programma che fa qualcosa e facendo piccole modifiche ed eliminando gli errori man mano che si procede si dovrebbe avere in ogni momento un programma funzionante sempre più completo

Operatori ed Espressioni

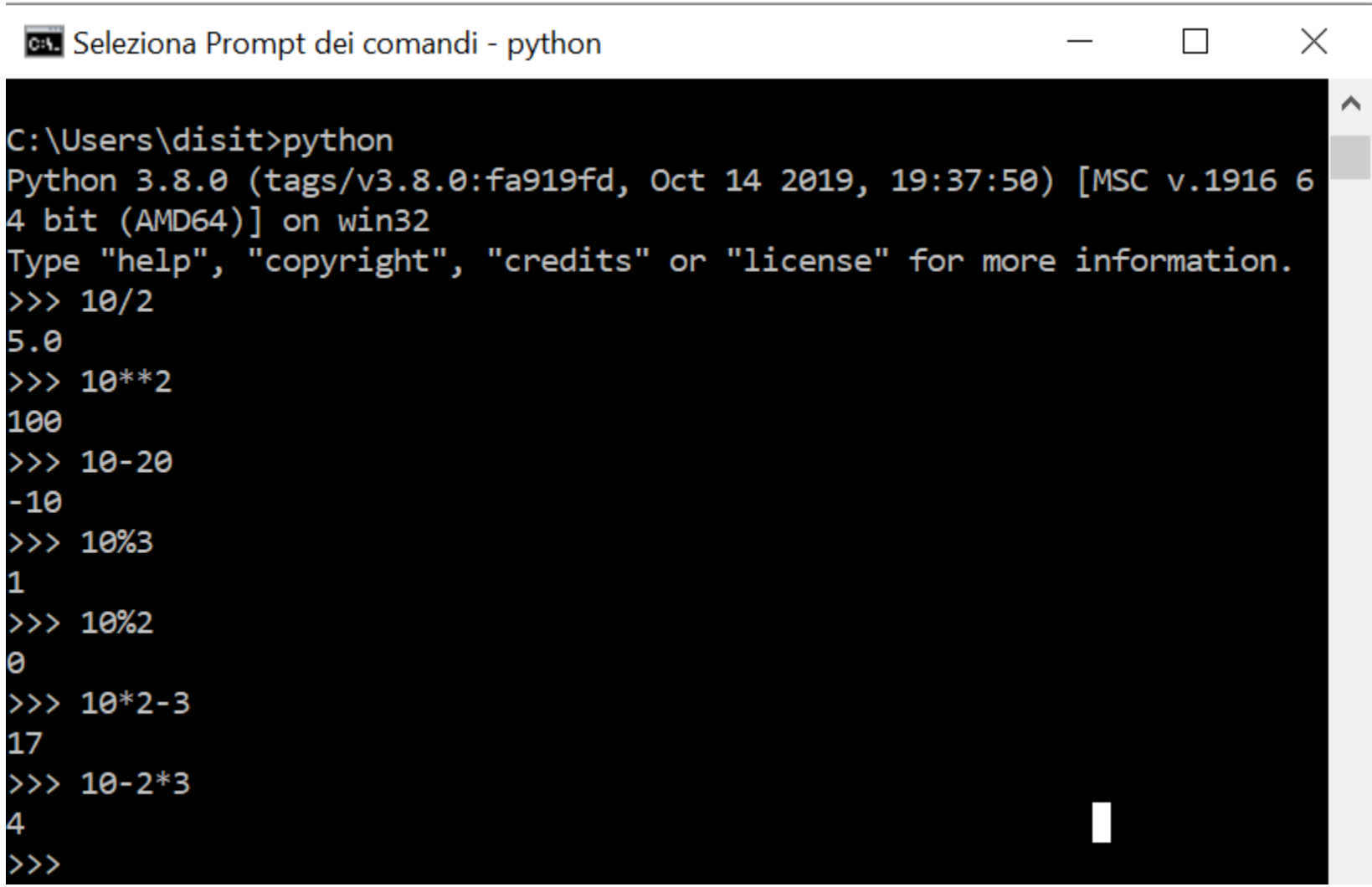
- Una espressione è la combinazione di costanti e riferimenti a variabili tramite operatori
- Gli operatori sono classificati per 'tipo di operazione':

Aritmetici	Divisione / Modulo % Somma + Differenza - Prodotto *
Relazionali	Minore < Maggiore > Maggiore o uguale >= Minore o uguale <= Uguaglianza == Diverso !=
Logici	Congiunzione && Negazione ! Disgiunzione ||

Operatori Aritmetici (1)

- L'operatore somma, sottrazione, elevazione a potenza, modulo

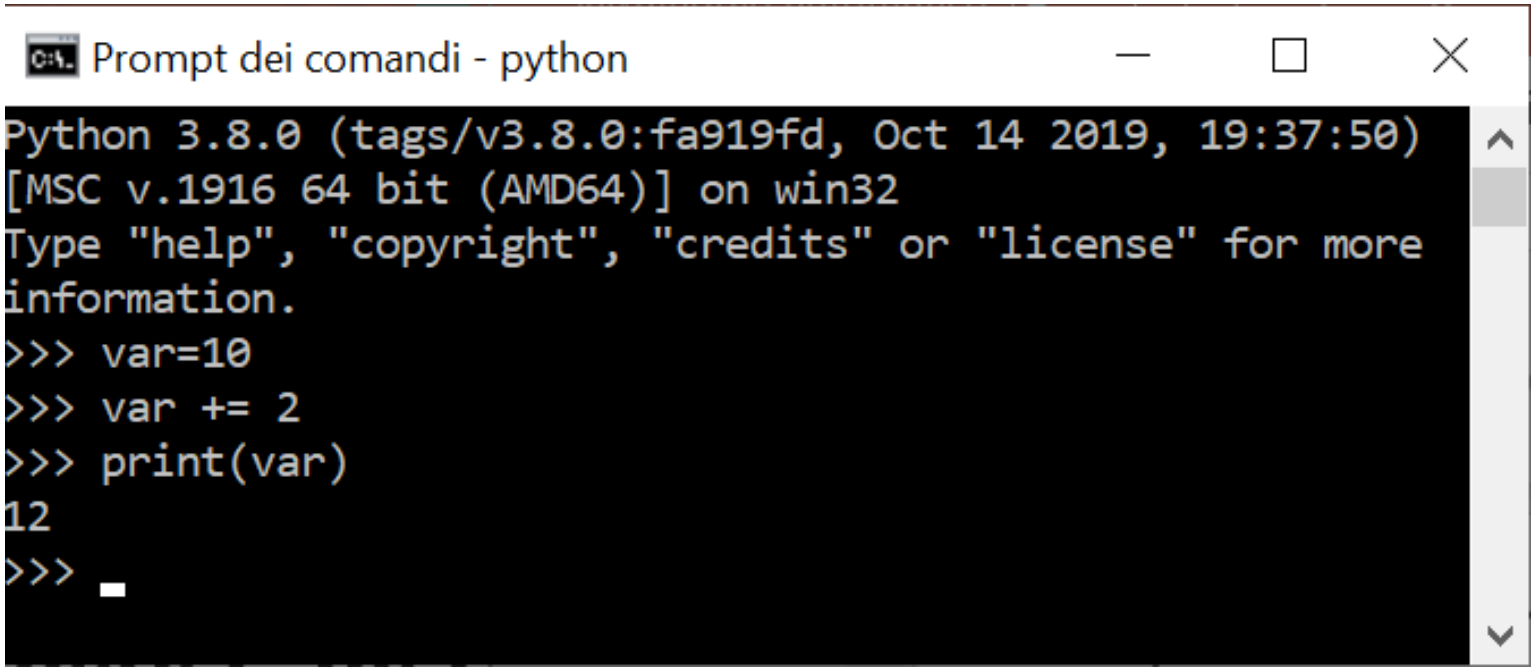
Aritmetici	Divisione	/
	Modulo	%
	Somma	+
	Differenza	-
	Prodotto	*
	Elevamento a potenza	**
	Incremento	+=
	Decremento	-=



Operatori Aritmetici (2)

- Operatore di assegnazione semplice (=)
- Gli operatori di assegnazione composti (+=, -=, *=, /=)

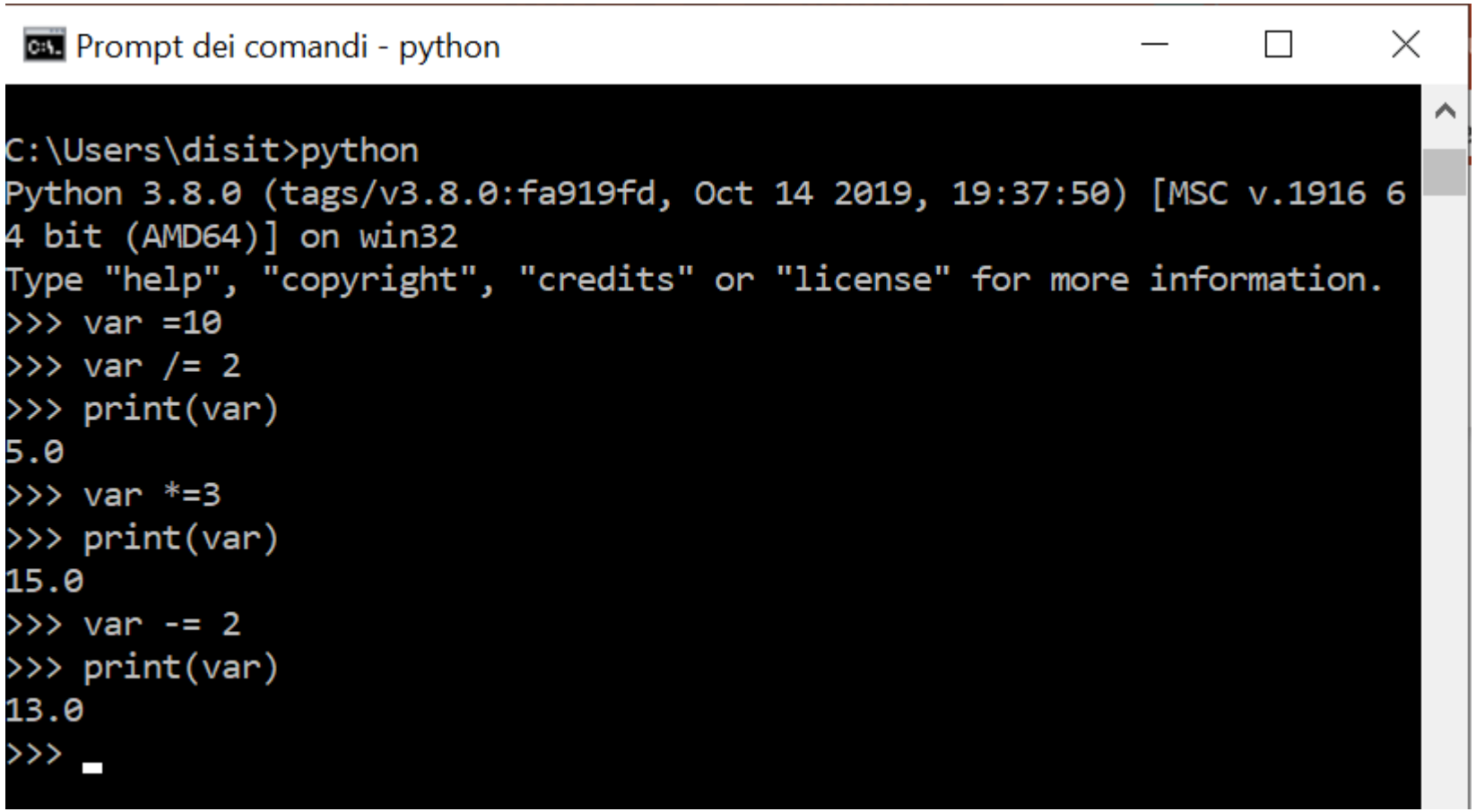
Aritmetici	Divisione	/	
	Modulo	%	
	Somma	+	
	Differenza	-	
	Prodotto	*	
	Elevamento a potenza	**	
	Assegnazione composti	+=	*=
		-=	/=



Operatori Aritmetici (3)

- Operatore di assegnazione semplice (=)
- Gli operatori di assegnazione composti (+=, -=, *=, /=)

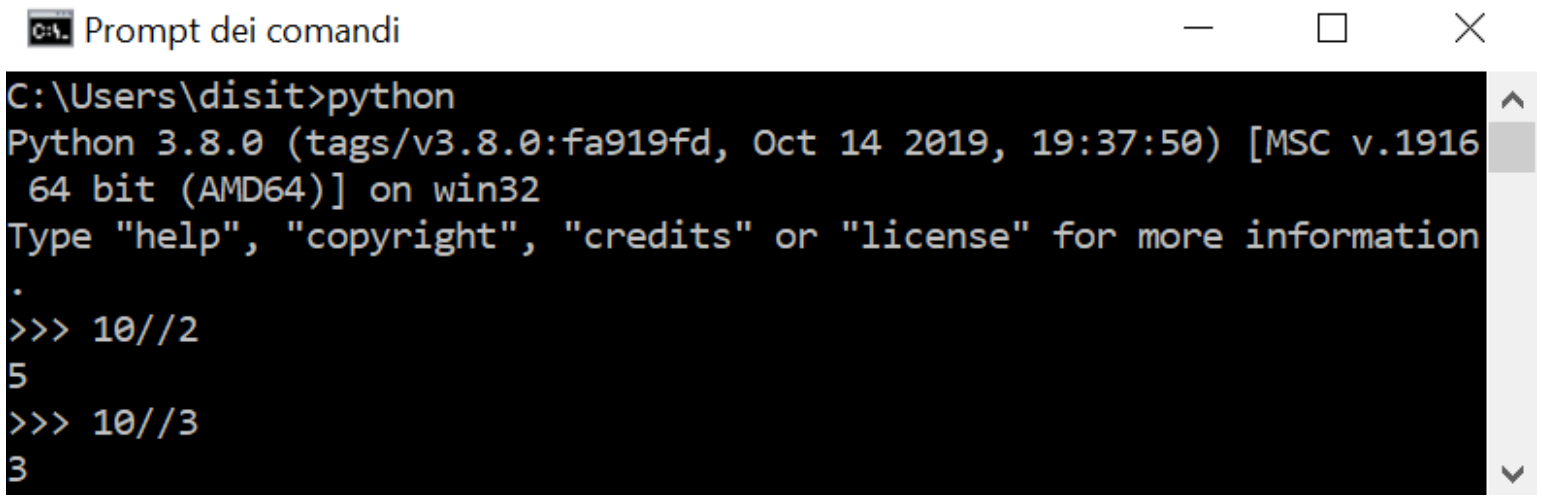
Aritmetici	Divisione	/
	Modulo	%
	Somma	+
	Differenza	-
	Prodotto	*
	Elevamento a potenza	**
	Assegnazione composti	+= *= -= /=
	Parte intera	//



Operatori Aritmetici (4)

- Operatore di assegnazione semplice (=)
- Gli operatori di assegnazione composti (+=, -=, *=, /=)

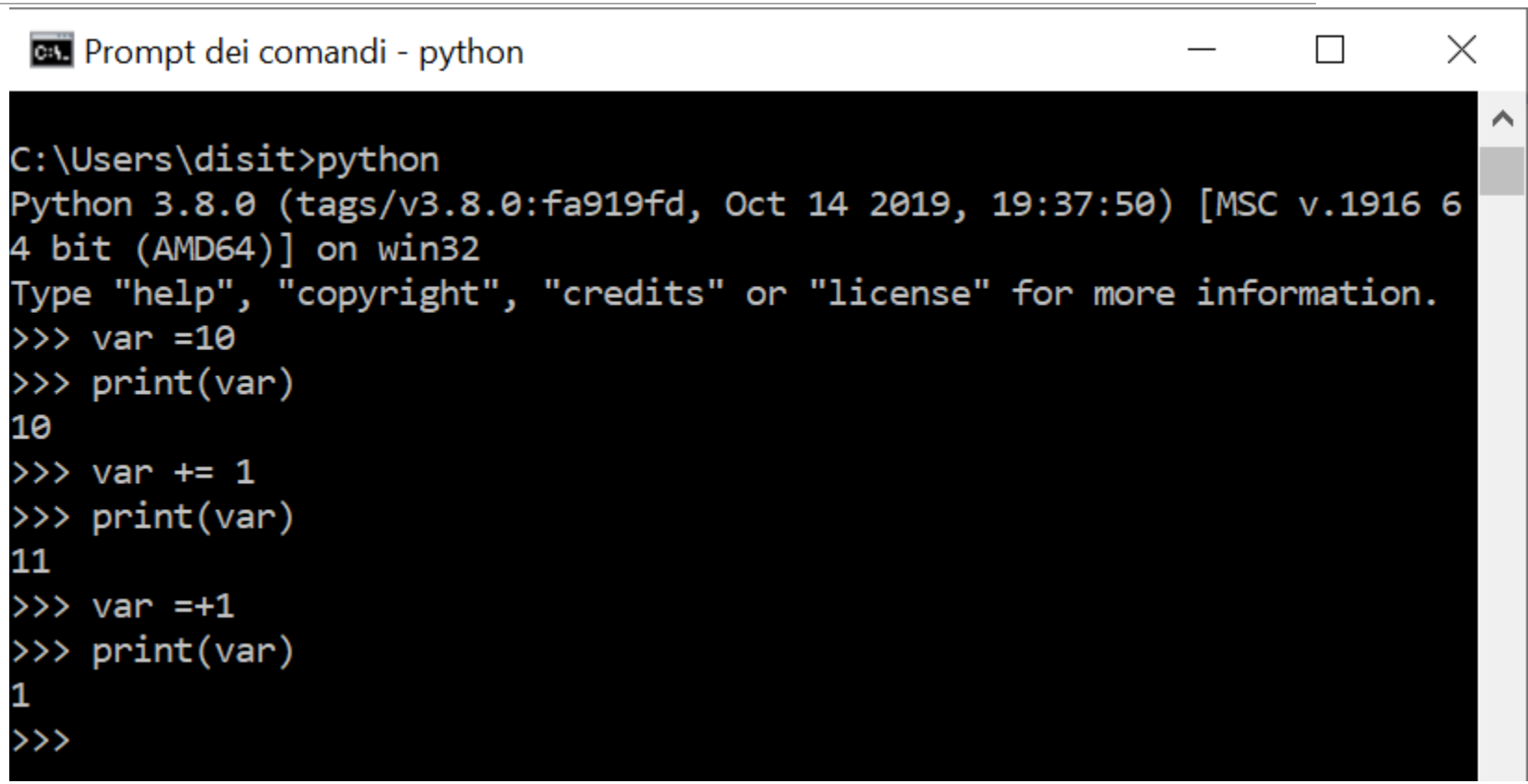
Aritmetici	Divisione	/
	Modulo	%
	Somma	+
	Differenza	-
	Prodotto	*
	Elevamento a potenza	**
	Assegnazione composti	+= *= -= /=
	Parte intera	//



Operatori Aritmetici (5)

- ATTENZIONE!!!!
  - var +=1 incrementa var di una unità
  - var = +1 è una operazione di ASSEGNAZIONE

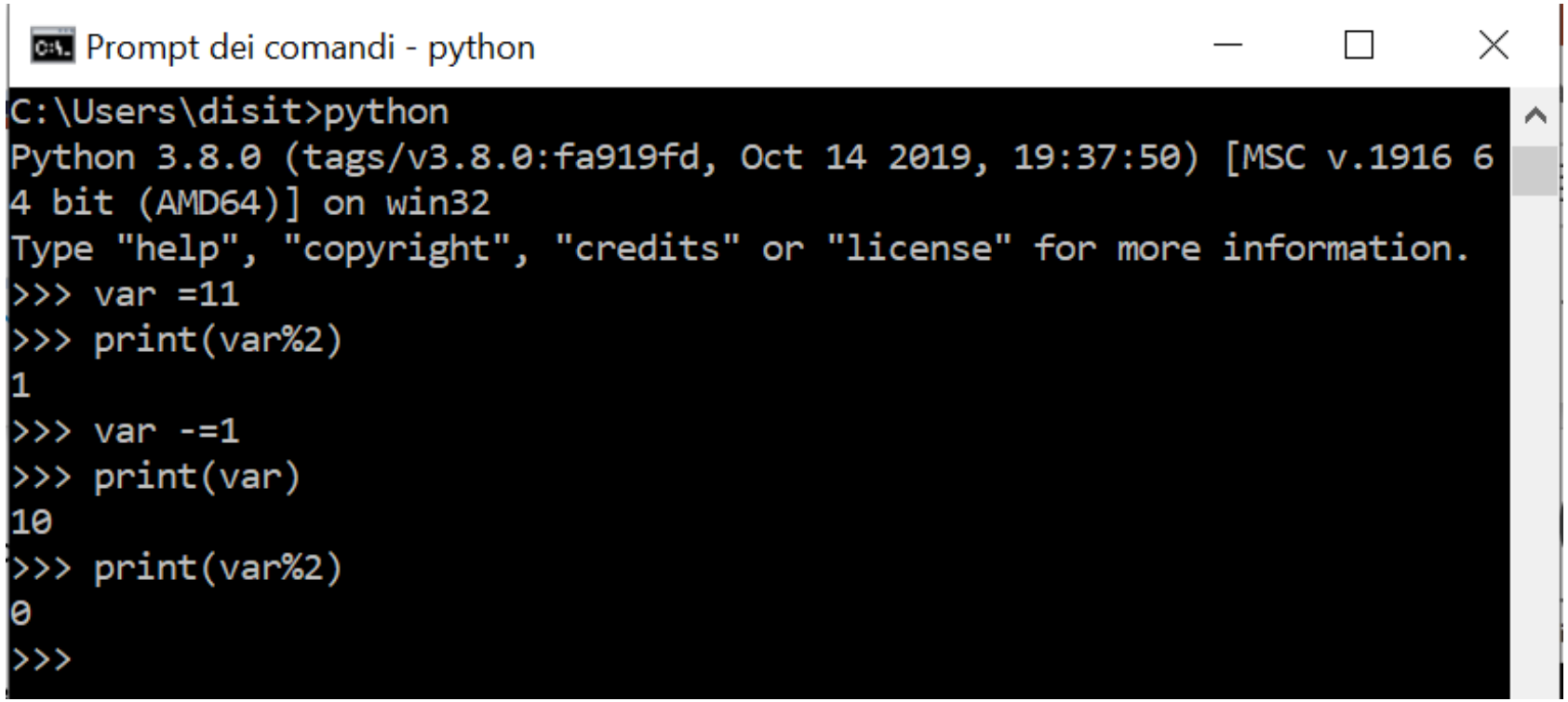
Aritmetici	Divisione	/
	Modulo	%
	Somma	+
	Differenza	-
	Prodotto	*
	Elevamento a potenza	**
	Incremento	+=
	Decremento	-=



Operatori Aritmetici (6)

- L'operatore modulo (%), in Python così come in altri linguaggi, fornisce il resto della divisione intera

Aritmetici	Divisione	/
	Modulo	%
	Somma	+
	Differenza	-
	Prodotto	*
	Elevamento a potenza	**
	Incremento	+=
	Decremento	-=



Operatori Relazionali (1)

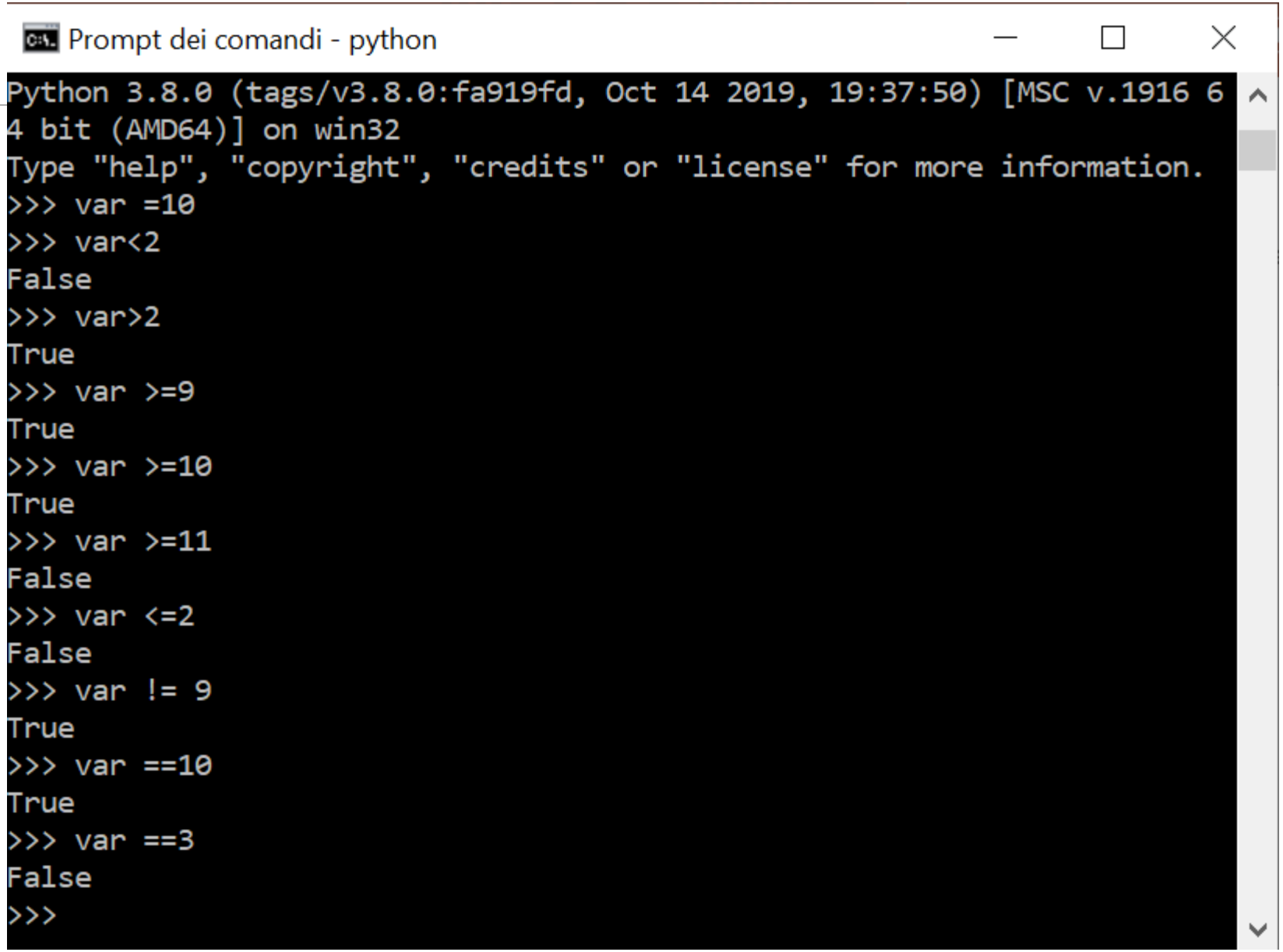
- Gli operatori relazionali e di uguaglianza (o disuguaglianza) confrontano il primo operando con il secondo per testare la validità della relazione in esame
- Gli operatori relazioni sono operatori **binari**:

- Il Risultato di una espressione relazionale è:
  - False se è falsa
  - True se la relazione testata è vera

Relazionali	Minore	<
	Maggiore	>
	Maggiore o uguale	>=
	Minore o uguale	<=
	Uguaglianza	==
	Diverso	!=

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Operatori Relazionali (2)



Operatore di Uguaglianza (==) e Operatore di assegnazione (=) (1)

- Attenzione!! E' necessario fare distinzione tra il concetto di assegnazione (in Python si usa '=') e l'operatore relazionale di Uguaglianza (in Python '==')
- **#Concetto di assegnazione**
- a = 5+3; # assegnazione l'elaboratore calcola il valore #dell'espressione che sta a destra del simbolo #'=' e ne assegna il valore alla variabile a
- #NOTA: L'operatore di assegnazione opera sempre da destra verso sinistra
- **# Operatore Relazionale di uguaglianza**

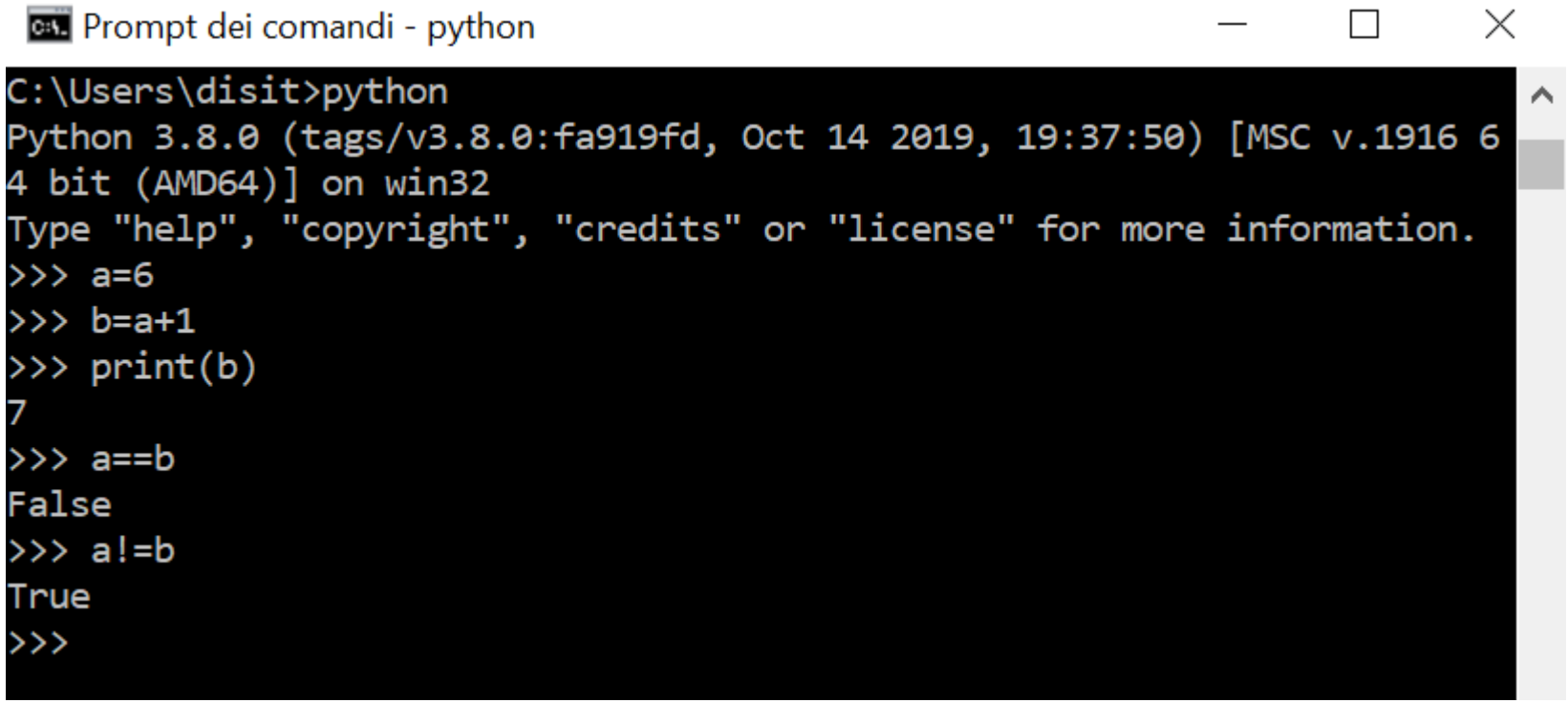
<valore1> <operatore di relazione> <valore2>

Quando si scrive: … a==b … l'elaboratore effettua un confronto tra le due variabili a e b e rende come risultato:

- False, Se l'affermazione è falsa, ovvero se a è diversa da b
- True, se l'affermazione è vera

Esempi di operatori Relazionali

- Operatore di Uguaglianza e Diverso:
  - a=6
  - b=a+1 #assegnazione b ha valore 7
  - #Se scrivo a==b, ho come risultato False
  - #Se scrivo a!=b, ho come risultato True,



Operatori Logici (1)

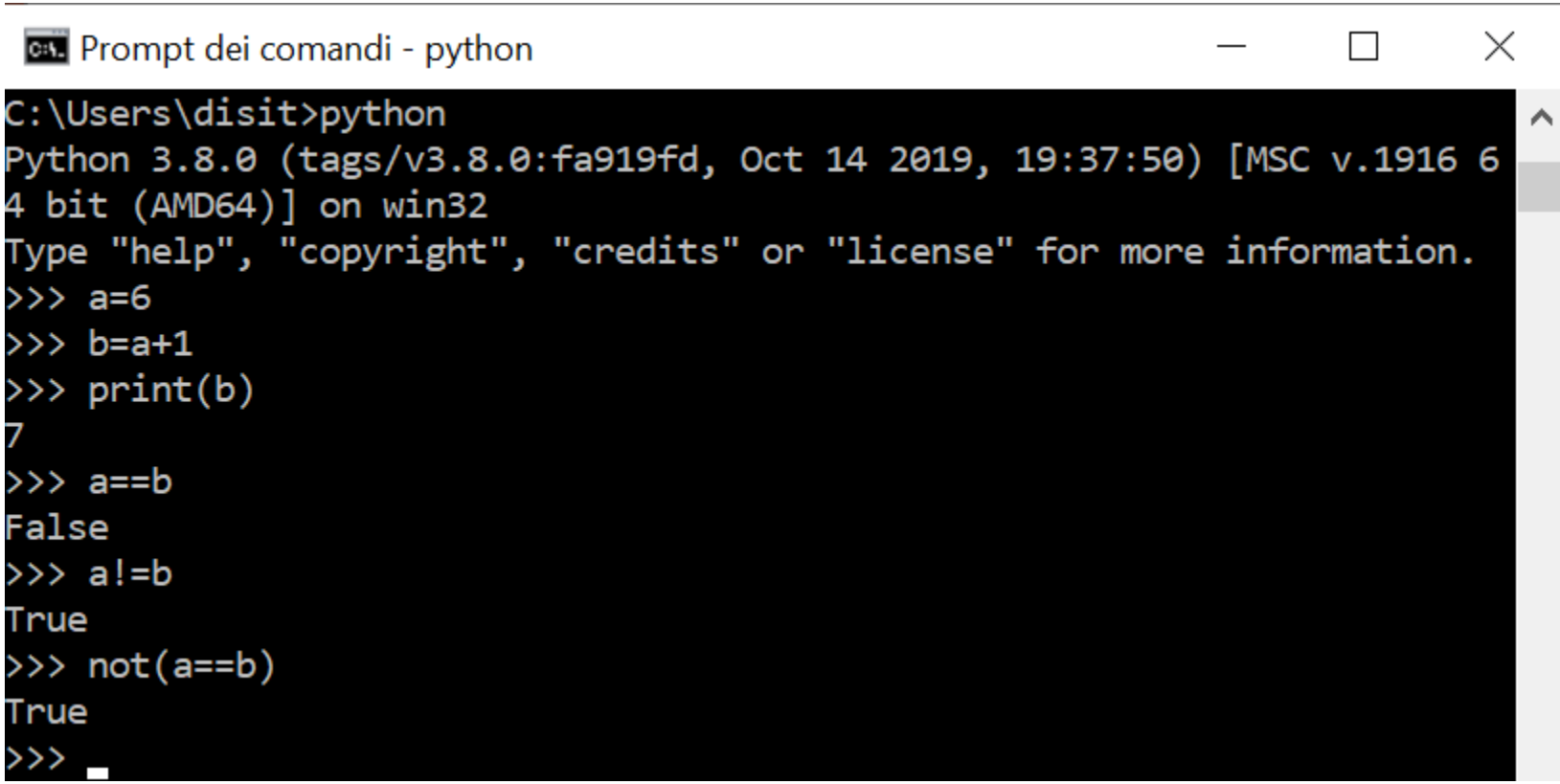
- Sono introdotti dall'Algebra di Boole
- I tre operatori di base sono:
  - AND (in Python **and**), OR (**or**) e NOT (**not**)

Logici	Congiunzione	and
	Negazione	not
	Disgiunzione	or

Operatori: NOT

- **NOT** A è una funzione logica che inverte il valore logico del suo operando
- È un operatore **unario**
- In Python, L'operatore NOT restituisce:
  - False, se NOT A è falso
  - True, se NOT A è vero
- In Python NOT A si scrive:
  - **not A**

A	NOT *A*
F	*V*
V	*F*



Operatori: AND

- A **AND** B è una funzione logica che è vera solo se entrambi gli operandi A e B sono veri
- L'operatore AND è binario
- In Python, L'operatore AND rende:
  - False, se l'espressione valutata (si veda tabella) è falsa
  - True, se l'espressione valutata (tabella) è vera
- In Python, A AND B, si scrive: A and B
- Ovvero: <espressione> and <espressione>
- A and B rende:
  - True, se entrambe le espressioni valutate sono vere
  - False, in caso contrario, ovvero se almeno una delle due espressioni valutate è falsa

A	*B*	*A* **AND** *B*
F	*F*	*F*
F	*V*	*F*
V	*F*	*F*
V	*V*	*V*

Operatori: AND - esempi

- a=6
- b=a+1 #assegnazione b ha valore 7
- c=7

- #… ((c==b) and (a<b))… rende True… ((true) and (true))
- #… ((c!=b) and (a<b))… rende False… ((false) and (true))
- #… ((c==b) and (c>b)) … rende False… ((true) and (false))
- #… ((c==b) and (c<b)) … rende False… ((true) and(false))
- #… ((a<=b) and (c==b)) … rende Frue… ((true) and (true))
- #… a and b… rende True perché sia a che b sono diverse da 0

Operatori: OR

- A **OR** B è una funzione logica che è vera solo se almeno uno dei due operandi A o B è vero
- In Python, L'operatore OR rende:
  - False, se l'espressione valutata (si veda tabella) è falsa
  - True, se l'espressione valutata (tabella) è vera
- In Python, A OR B, si scrive **A or B**
- Ovvero: <espressione1> or <espressione2>
- In Python, A or B rende:
  - il valore booleano True, se almeno uno dei due operandi è true
  - il valore booleano False, se entrambi gli operandi sono false

A	*B*	*A* **OR** *B*
F	*F*	*F*
F	*V*	***V***
V	*F*	***V***
V	***V***	***V***

Operatori: OR - esempi

a=6

b=a+1 #assegnazione b ha valore 7

c=7

#… ((c==b) or (a<b))… rende true… ((true) or (true))

#… ((c!=b) or (a<b))… rende true… ((false) or (true))

#… ((c==b) or (c>b)) … rende true… ((true) or (false))

#… ((c==b) or (c<b)) … rende true… ((true) or (false))

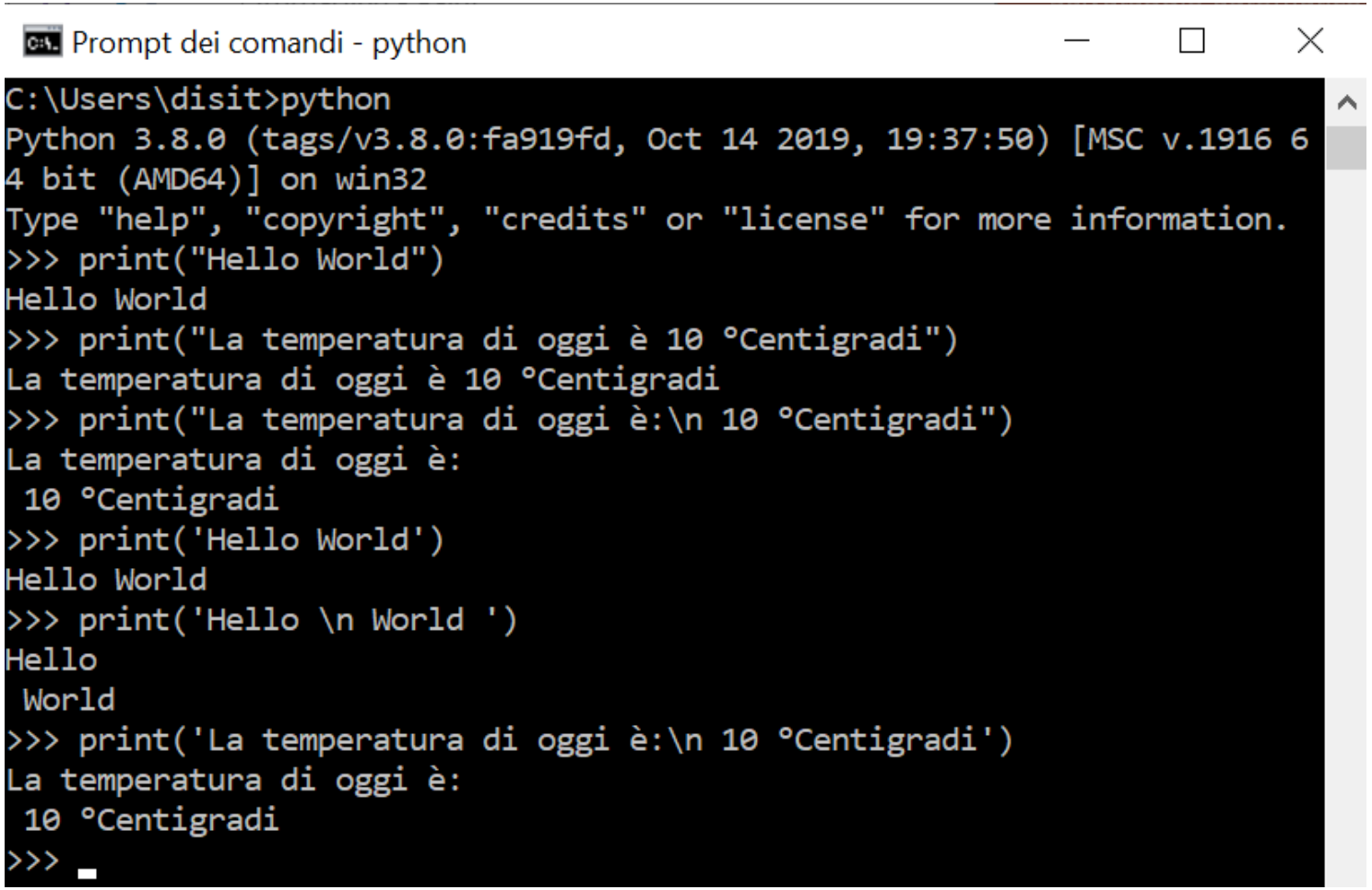
#… ((a<=b) or (c==b)) … rende true… ((true) or (true))

#… (a>b) or (c<b) … rende false perché nessuna delle //due espressioni valutate è vera

#… a or b… rende true perché sia a che b sono diverse da 0

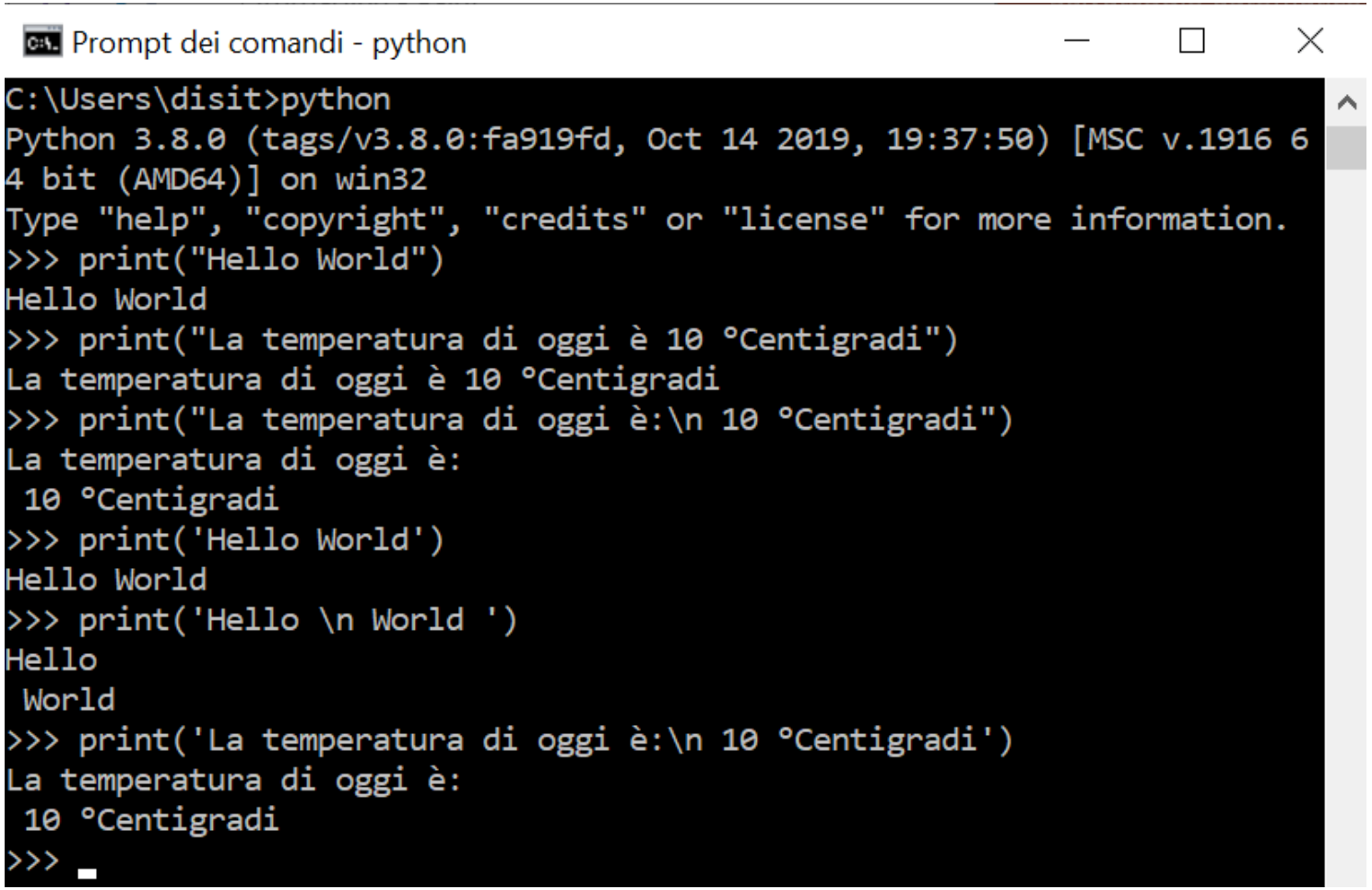
Stampare: Uso di print (1)

- Si è visto che per stampare su console si usa print()
- Vediamo alcuni esempi e usi di print per poi approfondire l'argomento via via che verranno trattati tipi di dati e strutture più complesse
- Stampare stringhe:
  - print("Hello World")
  - print("Hello World')
  - print("La temperatura di oggi è 10 °Centigradi")



Stampare: Uso di print (2)

- Si possono usare sia i doppi che i singoli apici
- Stampare stringhe:
  - print("Hello World")
  - print('Hello \n World')
  - print("La temperatura di oggi è 10 °Centigradi")
  - print("La temperatura di oggi è:\n 10 °Centigradi")
- Ci sono alcuni simboli speciali che permetto di formattare la stampa in uscita
  - Ad esempio \n manda a capo



Operatore di formato per stringhe (1)

Sintassi:

stringaContenteInformazioniDiFormato % (valore1, valore2, …, valoreN)

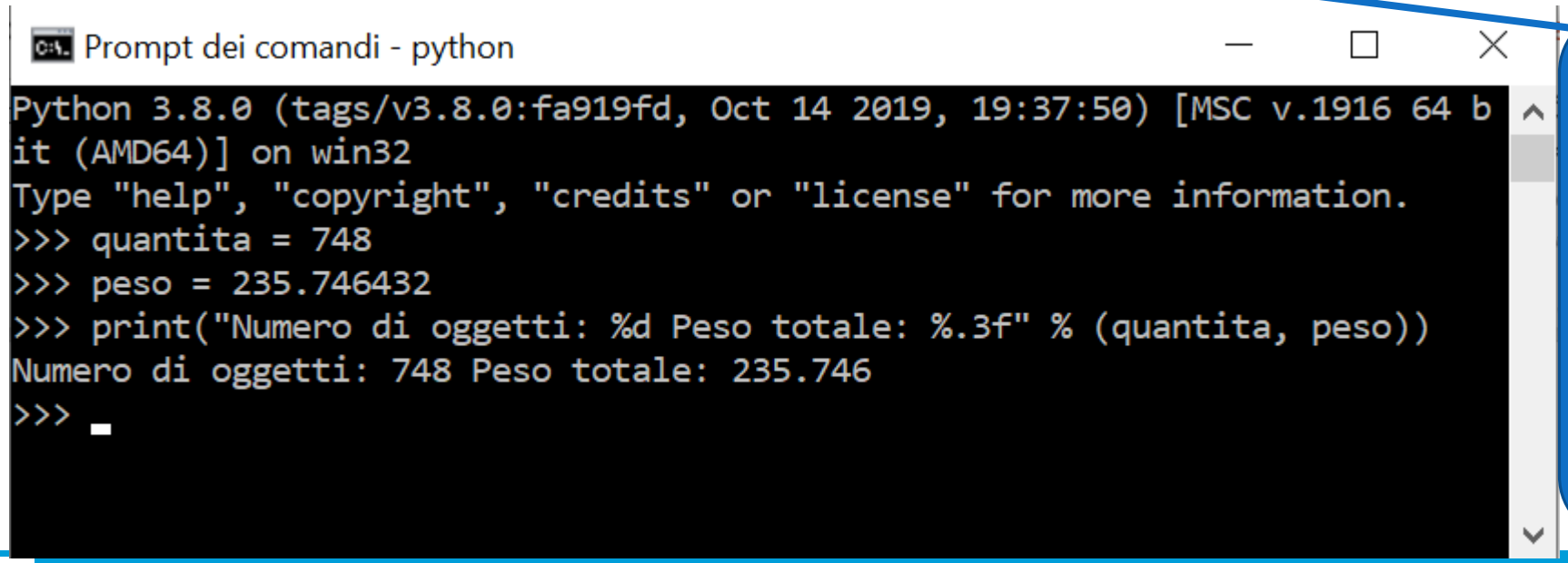
- **NOTA:** Se il valore a sinistra del simbolo % è una stringa, allora il simbolo % diviene operatore di formato

Esempio:

#cifre decimali da considerare

```
print("Numero di oggetti: %d Peso totale: %.3f" % (quantita, peso))
```

Stringa con informazioni di formato, ovvero contiene uno o più indicatori di formato (%d %.3f)



Nella stringa con le info di formato ci sono due indicatori, quindi qui si inseriscono le due variabili a cui gli indicatotri fanno riferimento

Operatore di formato per stringhe (2)

Esempio:

#cifre decimali da considerare

```
print("Numero di oggetti: %d Peso totale: %.3f" % (quantita, peso))
```

- Si è visto che è possibile specificare non solo il formato ma anche il numero di cifre decimali da considerare
- La sequenza di caratteri %10.2f è un indicatore di formato (format specifier), ovvero descrive il formato che va imposto ad un valore
- La lettera f finale specifica che l'indicatore di formato è relativo ad un valore in virgola mobile
- La lettera d specifica che l'indicatore di formato è relativo ad un valore di tipo intero
- Esistono alte lettere (si veda tabella successiva) che indicano ulteriori formati, ad esempio s sta per le stringhe

Operatore di formato per stringhe (3)

https://docs.python.it/html/lib/typesseq-strings.html

Conversione	Significato
d	Numero intero decimale con segno.
i	Numero intero decimale con segno.
o	Ottale senza segno.
u	Decimale senza segno.
x	Esadecimale senza segno (minuscolo).
X	Esadecimale senza segno (maiuscolo).
e	Numero in virgola mobile, in formato esponenziale (minuscolo).
E	Numero in virgola mobile, in formato esponenziale (maiuscolo).
f	Decimale in virgola mobile.
F	Decimale in virgola mobile.
g	Lo stesso di "`e`" se l'esponente è più grande di -4 o minore della precisione, "`f`" altrimenti.
G	Lo stesso di "`E`" se l'esponente è più grande di -4 o minore della precisione, "`F`" altrimenti.
c	Carattere singolo (accetta interi o stringhe di singoli caratteri).
r	Stringa (converte ogni oggetto Python usando `repr()`).
s	Stringa (converte ogni oggetto Python usando `str()`).
%	Nessun argomento viene convertito, riporta un carattere "`%`" nel risultato.

Operatore di formato per stringhe (4)

- Oltre alla possibilità di specificare non solo il formato e il numero di cifre decimali da considerare, èanche possibile specificare la larghezza del campo:

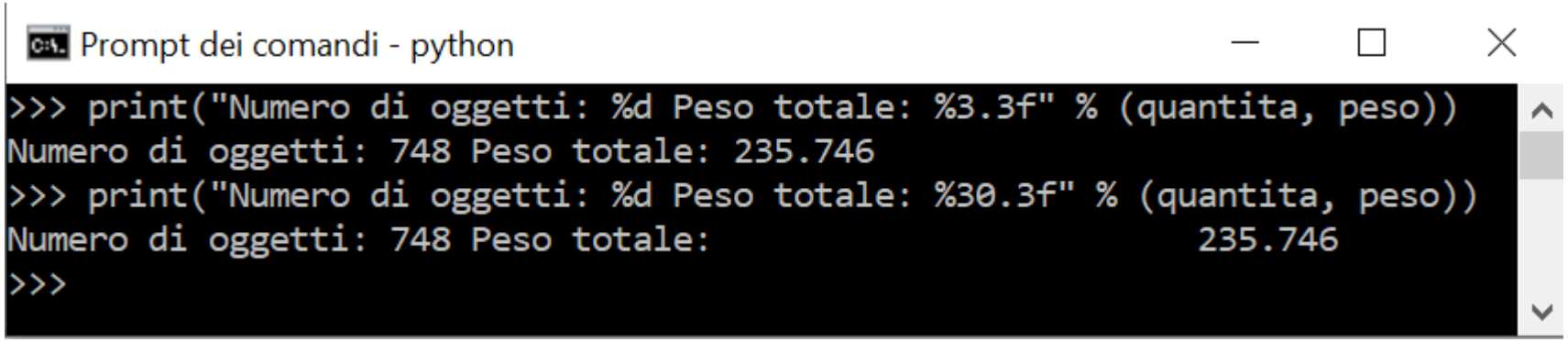
peso = 235.746432

print("Numero di oggetti: %d Peso totale: %10.3f" % (quantita, peso))

peso ->

			2	3	5	.	7	4	6

- In questo caso vengono usati 10 caratteri in totale, i primi 3 sono spazi vuoti



Operatore di formato per stringhe (5)

Alcuni esempi…..

Indicatore	Esempio di visualizzazione	Descrizione
"%d"	24	Numeri interi
"%5d"	␣␣␣24	Aggiunti spazi a sx in modo che ampiezza totale = 5
"%05d"	00024	Aggiunti 0 a sx in modo che ampiezza totale = 5
"%f"	1.248160	Numeri in virgola mobile
"%.2f"	1.24	Visualizza due cifre dopo il punto decimale
"%7.2f"	␣␣␣1.24	Aggiunti spazi a sx in modo che ampiezza totale = 7
"%s"	Hello	Stringhe
"%9s"	␣␣␣␣Hello	Ampiezza tot. 9, stringa allineata a destra
"%-9s"	Hello␣␣␣␣	Ampiezza tot. 9, stringa allineata a sinistra
"%d%%"	24%	Per visualizzare un segno di percentuale si usa %%
"%+6d"	␣␣␣+24	Usando il segno +, i numeri pos. Si visualizzano con +

Operatore di formato per stringhe (6)

Prompt dei comandi - python

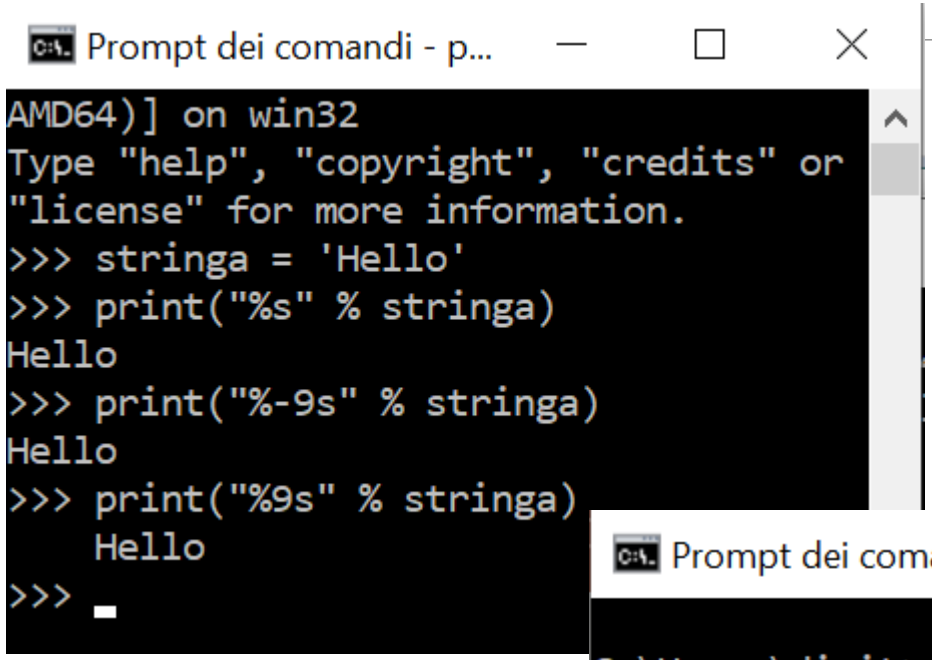
```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50)
[MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>> quantita = 24
>>> print("Ecco quanto vale quantità: %d" % quantita)
Ecco quanto vale quantità: 24
>>> print("Ecco quanto vale quantità: %5d" % quantita)
Ecco quanto vale quantità:    24
>>> print("Ecco quanto vale quantità: %05d" % quantita)
Ecco quanto vale quantità: 00024
>>>
```

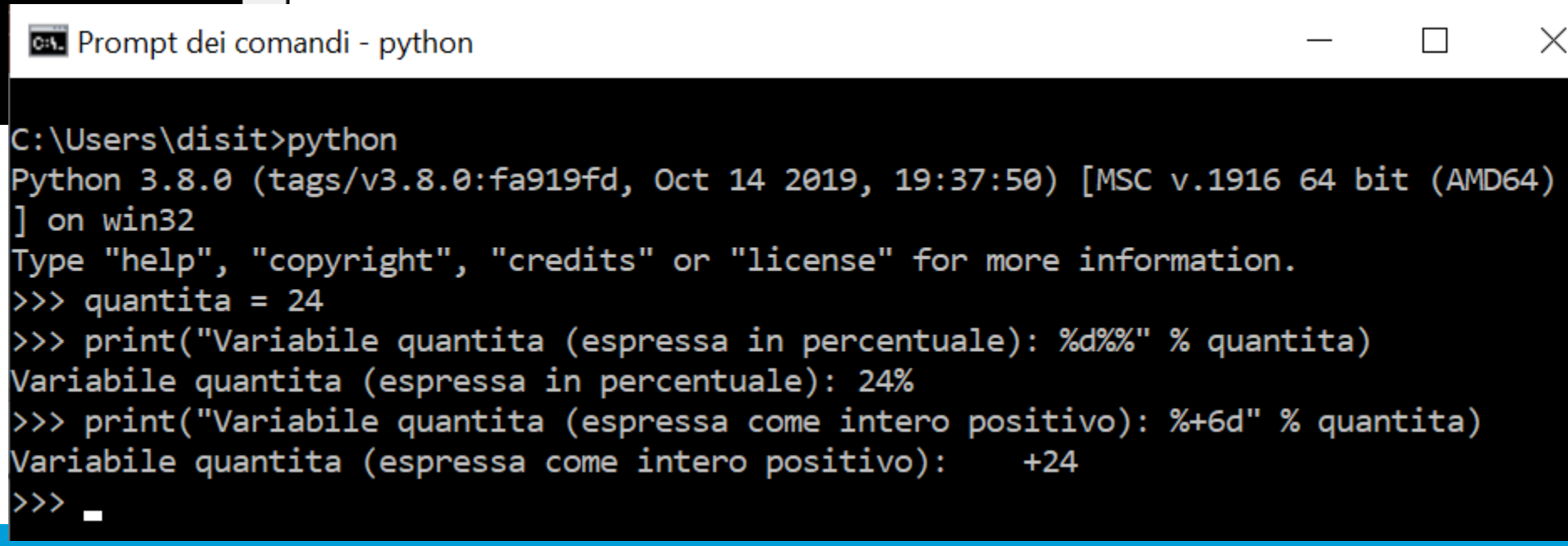
Prompt dei comandi - python

```
C:\Users\disit>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> numero = 1.24816
>>> print("Ecco quanto vale la variabile numero: %f!" % numero)
Ecco quanto vale la variabile numero: 1.248160!
>>> print("Ecco quanto vale la variabile numero: %.2f!" % numero)
Ecco quanto vale la variabile numero: 1.25!
>>> print("Ecco quanto vale la variabile numero: %7.2f!" % numero)
Ecco quanto vale la variabile numero:    1.25!
>>>
```

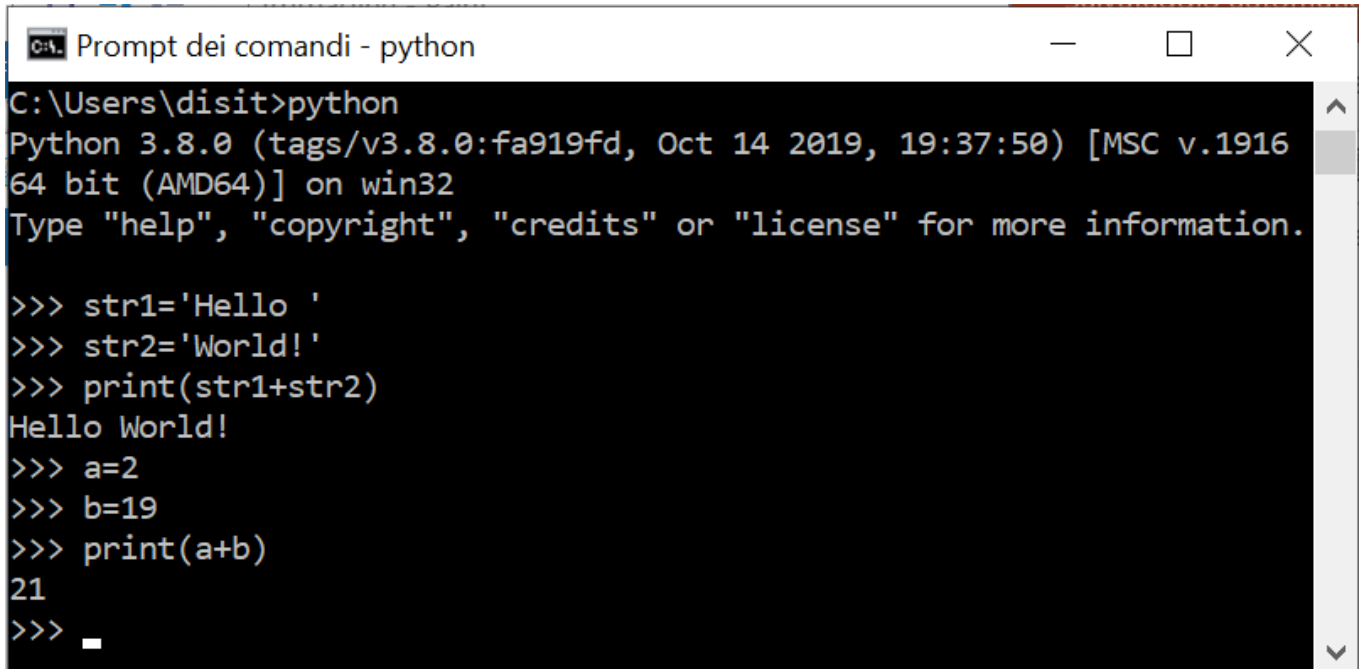
Operatore di formato per stringhe (7)





Stampare stringhe e numeri (1)

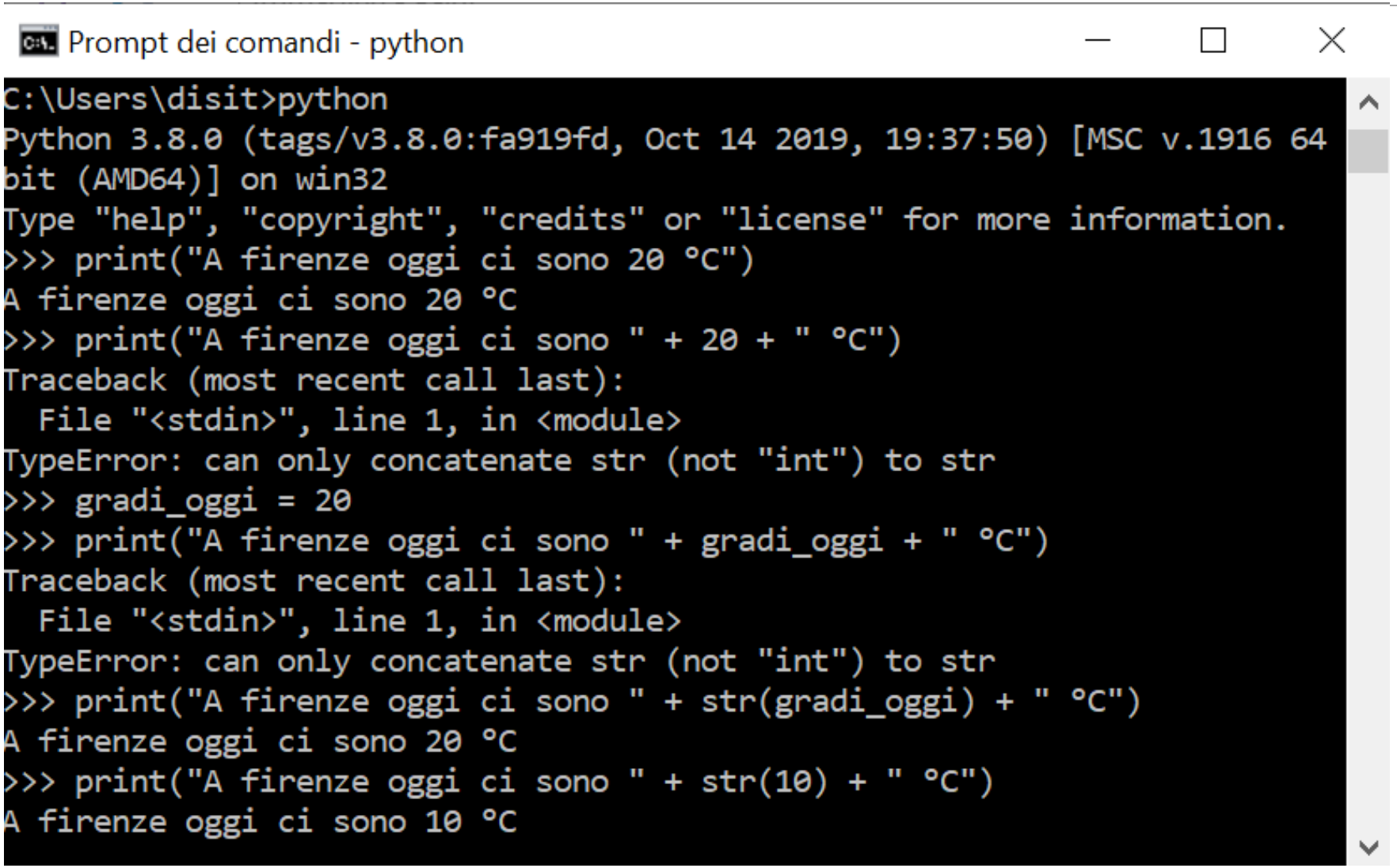
- Quando si vuole stampare più di una stringa alla volta si dice che si effettua una operazione di concatenazione
- L'operatore di concatenazione in Python è '+'
- Se si vogliono stampare due stringhe:
  - str1= 'Hello '
  - str2 ='World!'
  - print(str1+str2)
- Se si vogliono stampare due numeri:
  - a=2
  - b=19
  - print(a+b) # in questo caso il + è
    # l'operatore di addizione!!
- Cosa succede se si vogliono stampare insieme stringhe e numeri?



Stampare stringhe e numeri (2)

- Cosa succede se si vogliono stampare insieme stringhe e numeri?
- Python da' errore se si tenta di concatenare tipi di dati diversi da stringhe
- Se si scrive ad esempio:
  - print("A firenze oggi ci sono " + 20 + " °C")
  - Si avrà il seguente errore:
  - … TypeError: can only concatenate str (not "int") to str
- Per risolvere il problema si puo' fare una operazione di conversion di tipo (CAST), ovvero si dice a Python di trattare le variabili di tipo intero (o i numeri) come se fossero stringhe:
  - print("A firenze oggi ci sono " + str(20) + " °C")
- Oppure:
  - gradi_oggi = 20
  - print("A firenze oggi ci sono " + str(gradi_oggi) + " °C")
- In questo modo python riconosce lo stesso tipo stringa e riesce a concatenare e stampare ciò che ci si aspetta

Stampare stringhe e numeri (3)



Conversione tra numeri e stringhe (1)

- Si è visto che a volte si rende necessario convertire in stringa un valore numerico
- Non è possibile infatti concatenare una stringa con numero
- La funzione str() converte in stringa un numero intero o un numero in virgola mobile (il concetto di funzione verrà approfondito più avanti):

  str(1432) converte il valore numerico 1432 nella stringa ‘1432’

- Quindi è possibile scrivere la seguente riga di codice, senza che Python dia errore:

```
print("Anno: "+str(1432))
```

- La funzione str() può essere usata anche per convertire in stringa un valore in virgola mobile:

```
print("Anno: "+str(14.032))
```

Conversione tra numeri e stringhe (1)

- Viceversa, per trasformare i numeri interi e i numeri in virgola mobile in stringhe, si usano rispettivamente le funzioni int() e float()

id = int('1734')

price = float('17.34')

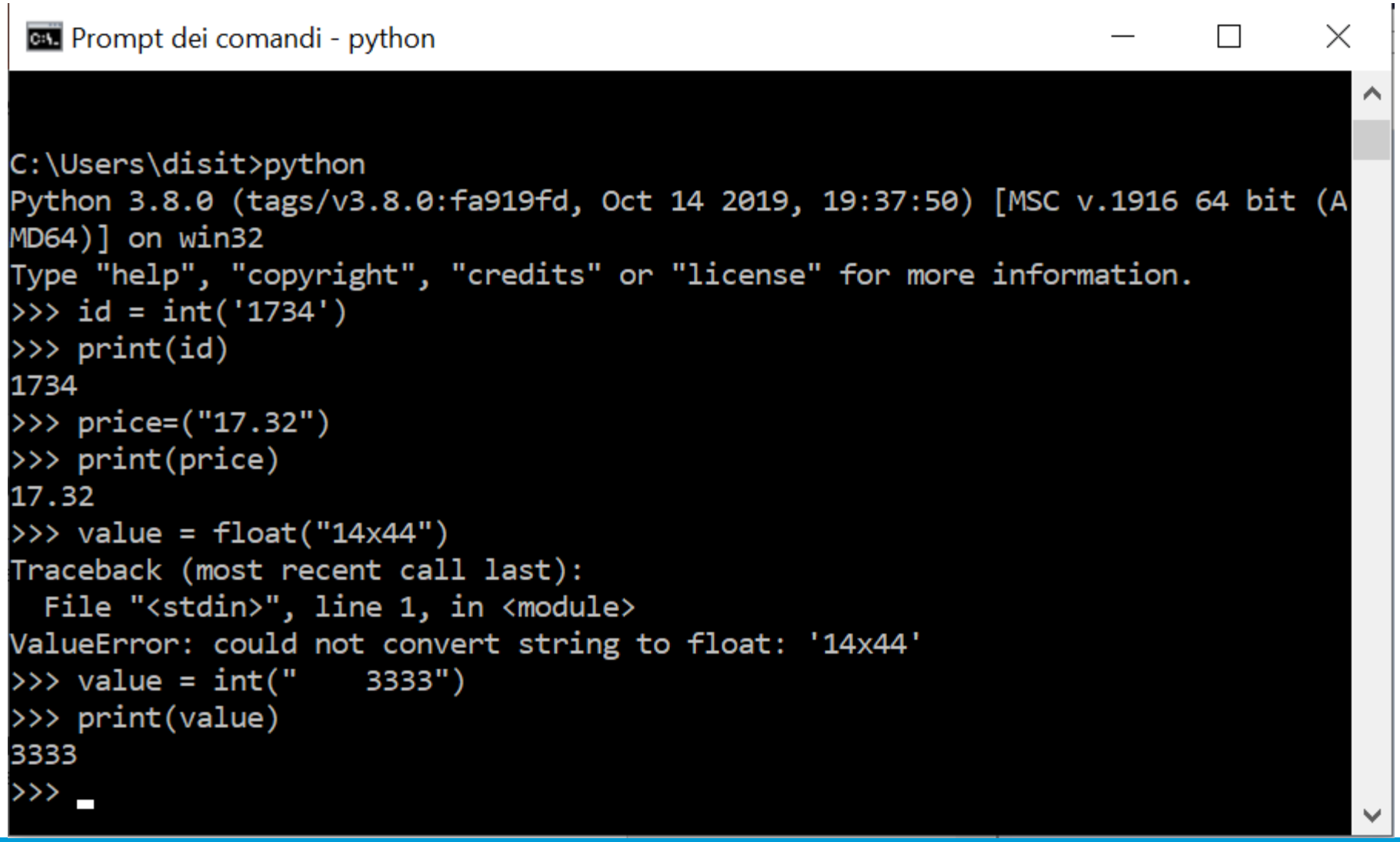
- Questa conversione è particolarmente utile quando le stringhe vengono fornite dall'utente del programma come dati di ingresso e devono poi essere convertite in numeri

value = float('17x56')

- Genera invece un errore perché la lettera x non puo' far parte di un letterale in virgola mobile
- Eventuali spazi presenti vengono ignorati:

value = int(' 1756') # assegna il valore intero 1756 alla variabile value

Conversione tra numeri e stringhe (2)



Concatenazione stringhe e numeri

Tipi in Phyton:

- Numeri
- Stringhe
- Etc.

Per concatenare stringhe e numeri è necessario trasformare i numeri in stringhe perché i tipi devo essere uguali

Si usa a funzione str():

- var ="le variabili sono: "+str(x)+" e "+str(y)

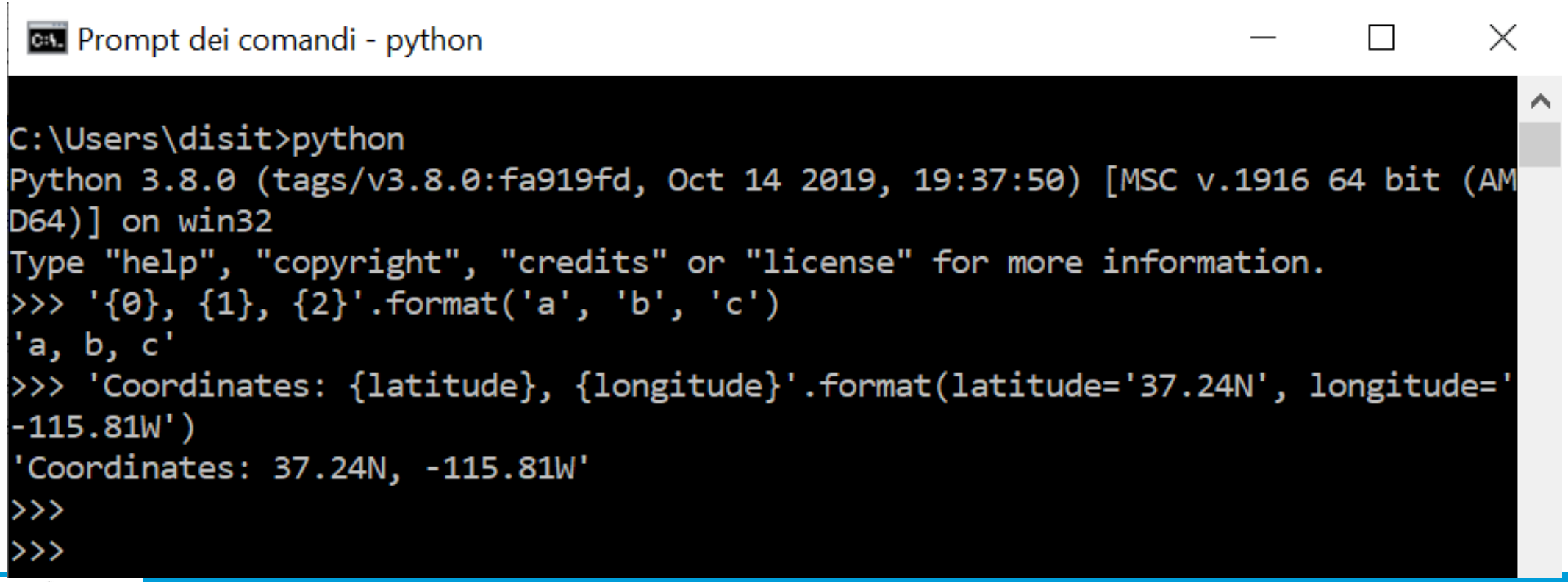
main.py saved

```
 9   x = 9
10   y = 10
11   var ="le variabili sono: x che vale "+str
     (x)+" e y che vale "+str(y)
12   print(var)
```

```
le variabili sono: x che vale 9 e y che vale
10
```

Stampa a Console

- Si possono stampare stringhe anche in modo più raffinato …
- Ad esempio **ricorrendo a format**
  - '{0},{1},{2}'.format('a',2,'Hello!') #in questo caso format concatena anche tipi diversi… si capirà meglio in futuro



Python: Compilatori e Console

Esecuzione di programmi e ambienti: notebooks, IDE, console

- Il linguaggio python
  - o tipi mutabili e immutabili
  - o operatori ed espressioni
  - o istruzioni
  - o funzioni
  - o cicli while e for
  - o esecuzione condizionale
- Strutture dati e algoritmi elementari: Liste, Dizionari, Insiemi, iterazioni su strutture dati

Costo di esecuzione e complessità

Il modello di costo

Cenni sulla complessità di un algoritmo:

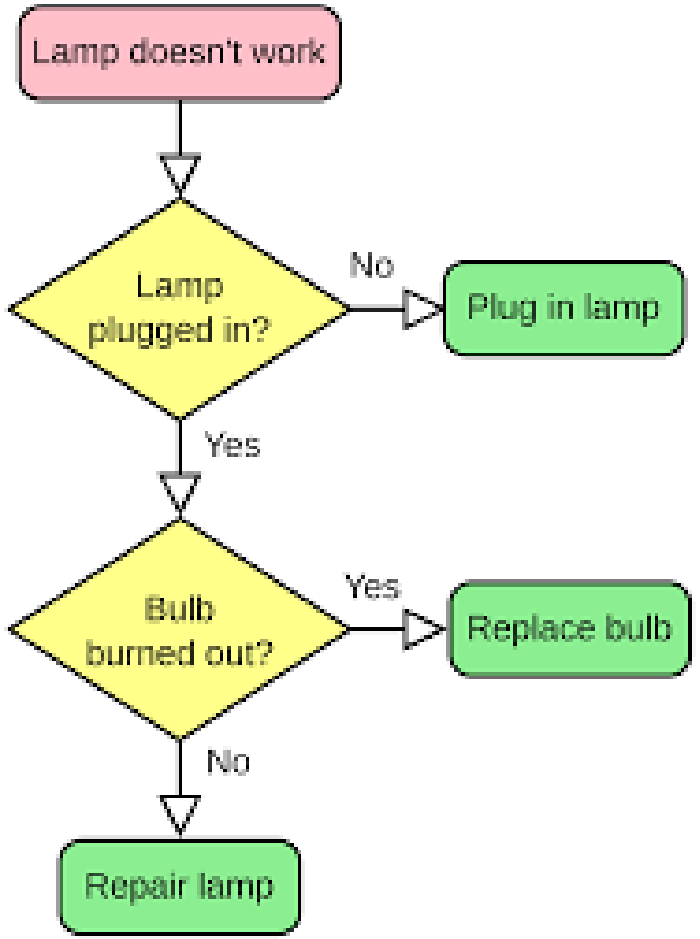
- Algoritmi di ordinamento su vettori
  - o Sequential-sort
- Cenni sugli alberi
  - o Alberi
  - o Alberi binari di ricerca: i) Visita in forma ricorsiva; ii) Ricerca; iii) Inserimento ordinato

Cenni su analisi dei dati, lettura e scrittura di file in forma tabulare, grafici.

Uso dei Flow Chart

Flow Chart

- I Flow Chart servono per schematizzare flussi di ragionamento nei vari ambiti
- Servono come strumento per formalizzare/visualizzare gli algoritmi
- Gli algoritmi per poter essere 'processabili' da un elaboratore devono essere tradotti in un programma
- I passi finiti degli algoritmi sono tradotti in Istruzioni in un programma



Riprendiamo il concetto di …. Istruzione

- ‘In informatica, elemento della programmazione con cui si richiede al computer, attraverso un codice prestabilito, l’esecuzione di una determinata operazione (leggere dati in ingresso, effettuare un calcolo, una selezione, ecc.)’, Treccani

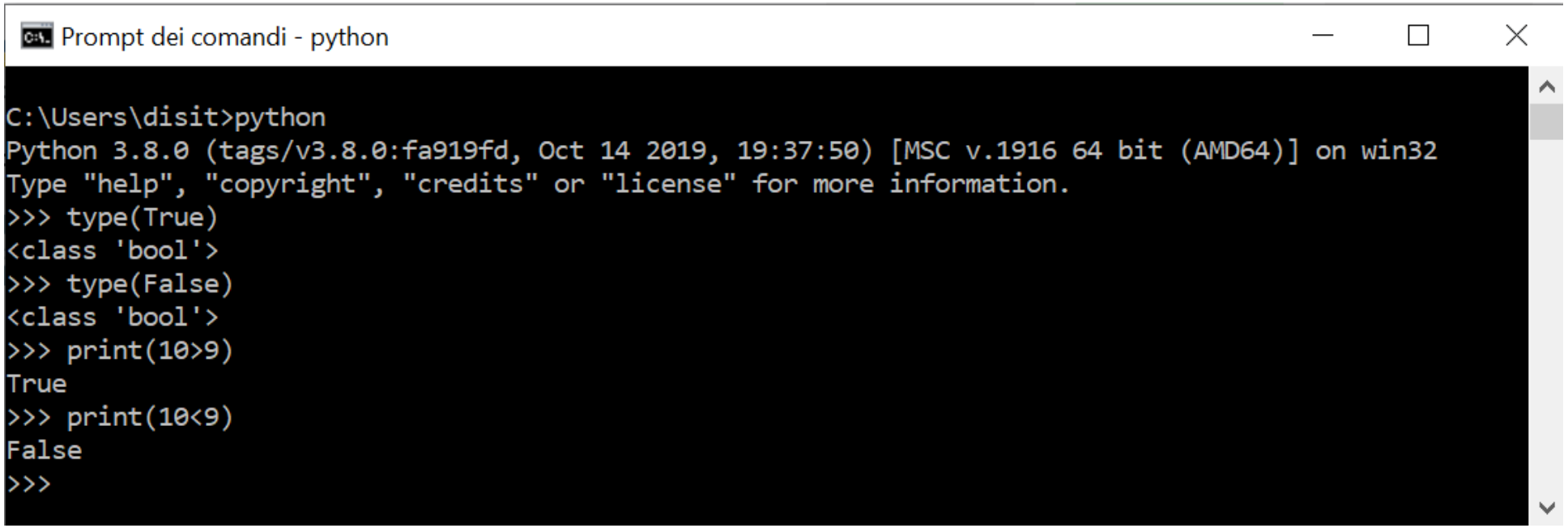
- ‘Con il termine istruzione, in informatica, si intende il comando impartito ad un esecutore (processore) utilizzando un linguaggio ad esso comprensibile’, wikipedia

Istruzioni e Flow Chart

- Le ISTRUZIONI servono per 'dirigere' il flusso della esecuzione di un programma:
- Il ruolo delle istruzioni è quello di determinare la sequenza con cui le istruzioni devono essere eseguite
- Uno strumento fondamentale per descrivere le istruzioni è il flow chart (diagramma di flusso)
- FLOW CHART: sono diagrammi che servono per evidenziare il flusso delle istruzioni

Istruzione

- Un'istruzione è un'operazione che l'interprete Python può eseguire
- Quando si scrive una istruzione sulla riga di comando, Python la esegue e se previsto stampa il risultato a video
- Uno script di solito contiene una sequenza di istruzioni: se sono presenti più istruzioni i loro risultati appariranno via via che le singole istruzioni saranno eseguite



Blocco di Istruzioni

- ‘Un sottoinsieme auto consistente di istruzioni adiacenti la cui esecuzione complessiva equivale all'esecuzione di una singola macroistruzione complessa viene denominato **blocco di istruzioni**.’ Wikipedia

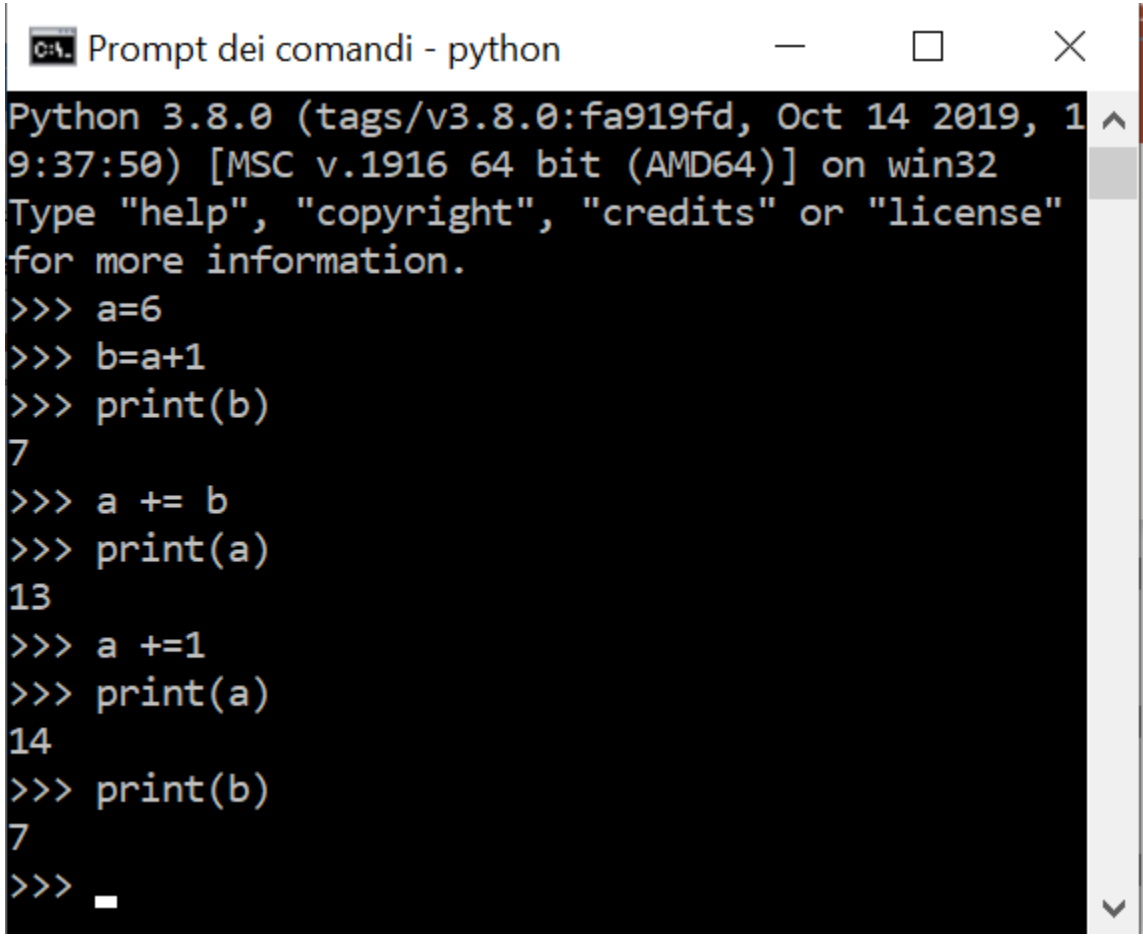
- Un **blocco di istruzioni** può essere considerato l'equivalente di una istruzione
  - Ogni istruzione può essere scomposta in istruzioni più semplici finché non si arriva ad una istruzione che non può essere ulteriormente scomposta (approccio top-down).
  - Ogni istruzione può essere raggruppata all'interno di un'istruzione più complessa (compound), fino ad arrivare al livello dell'intero programma (approccio bottom-up).

ISTRUZIONI e Flow Chart (1)

- Le ISTRUZIONI servono per 'dirigere' il flusso della esecuzione di un programma
- Il ruolo delle istruzioni è quello di determinare la sequenza con cui le istruzioni devono essere eseguite
- Uno strumento fondamentale per descrivere le istruzioni è il flow chart (diagramma di flusso)
- FLOW CHART: sono diagrammi che servono per evidenziare il flusso delle istruzioni

ISTRUZIONI e Flow Chart (2)

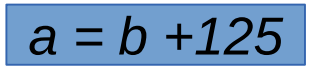
- La più semplice istruzione è una espressione (istruzione di assegnazione):
  - a = b + 125
- Altri esempi:
  - a=3 #assegnazione
  - a/3 #espressione (divisione)
  - 10%3 #espressione (modulo) -> rende 1 perché 10 NON è divisibile per 3
  - 10%5 #espressione (modulo) -> rende 0 perché 10 è divisibile per 5



- #NOTA: Si ricorda che l'assegnazione è a sua volta una espressione
- a=6 #assegnazione: a ha valore 6
- b=a+1 #assegnazione: b ha valore 7
- a += b #assegnazione: a vale 13, b vale 7
- a += 1 #assegnazione: a vale 14, b vale 7

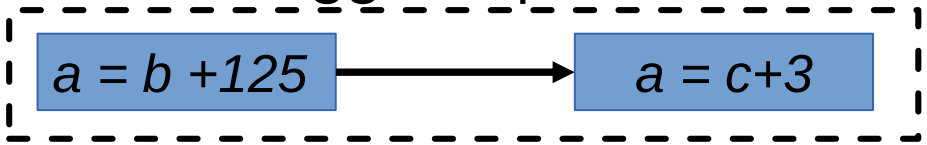
Espressioni e Flow chart (1)

- Nel rappresentare i flow chart si fa uso di:
  - **Rettangoli**: entro cui si inseriscono le espressioni

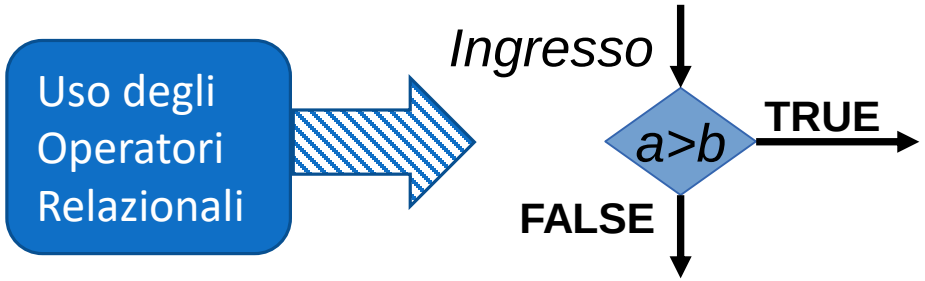


  - **Linee continue e frecce**: per collegare le varie componenti

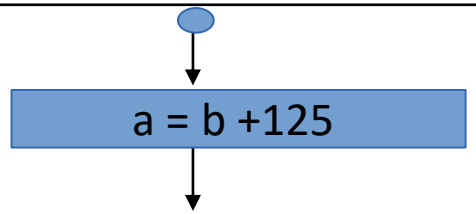
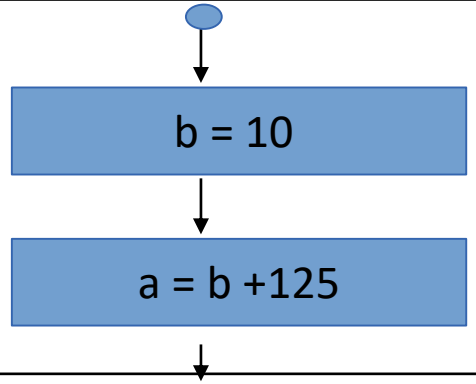
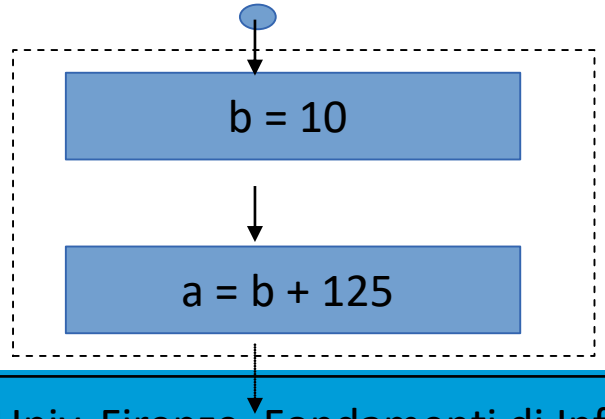
  - **Linee tratteggiate**: per effettuare raggruppamenti (compound o blocchi di istruzioni)



  - **Rombi**: per esprimere le condizioni (che vedremo)



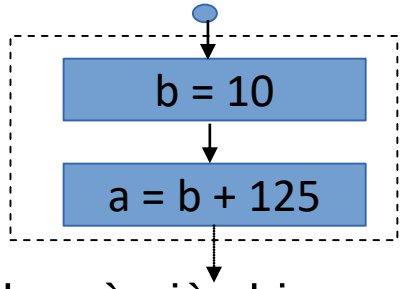
Espressioni e Flow chart (2)

Linguaggio Python	Flow chart	NOTE
CASO A) a= b+125		Istruzione semplice
CASO B) b =10 a = b + 125		Sequenza di 2 istruzioni
CASO C) b =10 a = b + 125 ATTENZIONE alla **indentazione**! Il TAB (Spazio) serve per il compound		Istruzione Compound

Concetto di compound

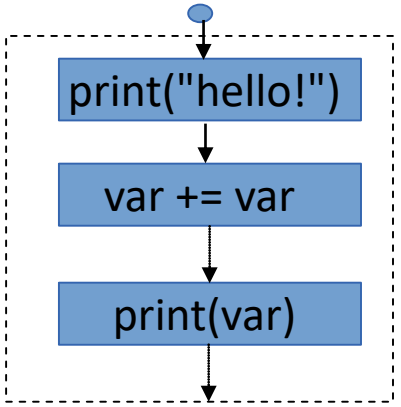
- L'esecuzione del compoud consiste nella esecuzione delle istruzioni in base alla tabulazione

```
if(var==6):
    b=10
    a = b + 125
```



- L'utilità del compound sarà più chiara successivamente

```
if(var==6):
    print("hello!")
    var += var
    print(var)
```



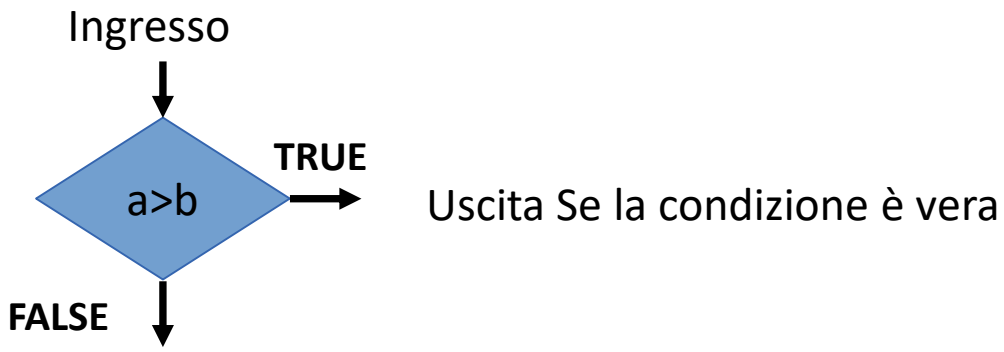
- Il vantaggio è quello di raccogliere una serie di istruzioni in blocchi in modo che possano essere trattati in maniera unitaria

- **Il Tab (o gli spazi) condizionano la SINTASSI vincolando l'ordine di associazione dei termini**

Condizione – istruzione if (1)

- Le istruzioni condizionali permettono di decidere direzioni diverse nel flusso di esecuzione in base al valore restituito da una espressione

- Istruzione **if**
  - Condiziona l'esecuzione di una istruzione detta CORPO, al risultato restituito da una espressione detta GUARDIA



Uscita Se la condizione è vera

Uscita Se la condizione è falsa

Condizione – istruzione if (2)

- Le istruzioni condizionali permettono di decidere direzioni diverse nel flusso di esecuzione in base al valore restituito da una espressione
- Istruzione if
  - Condiziona l'esecuzione di una istruzione detta CORPO, al risultato restituito da una espressione detta GUARDIA
  - Sintassi:

if(GUARDIA):
    CORPO

- Esempio:

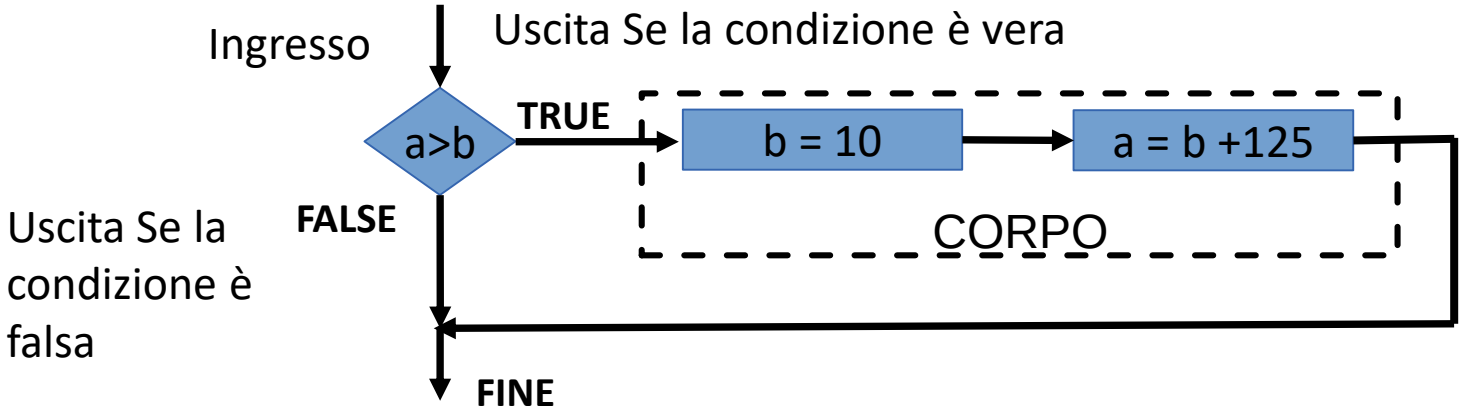
```
if(a>b):                # (a>b) → guardia
    b =10               # istruzioni che si trovano dentro il compound (TAB)
    a = b + 125
```

Condizione – istruzione if (3)

```
if(GUARDIA):
    CORPO
```

- Esempio:

```
if(a>b):              # (a>b) → guardia
    b =10             # istruzioni che si trovano dentro il compound
    a = b + 125
```



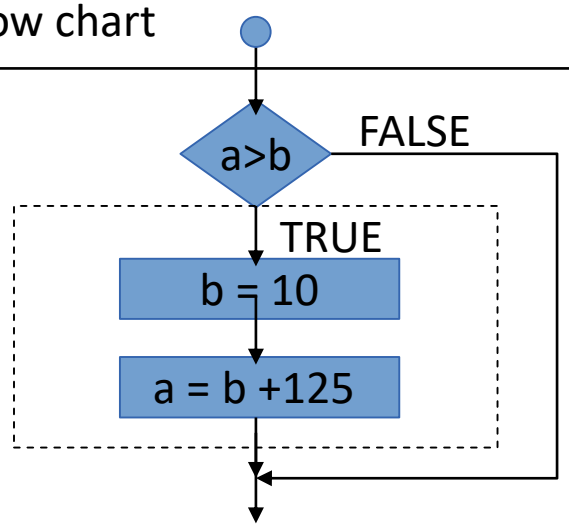
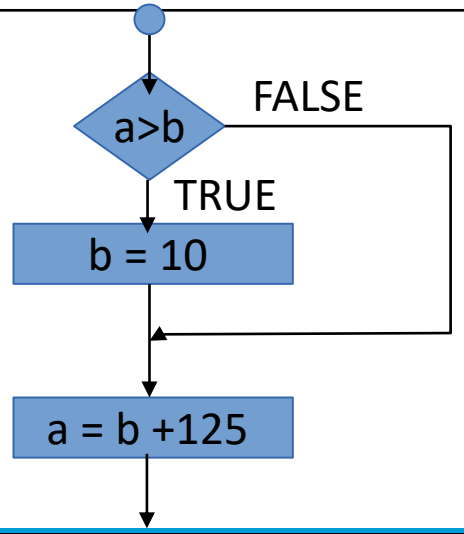
Condizione – istruzione if (4)

- Esempio:

```
if(a>b):              # (a>b) → guardia
    b =10             # istruzioni che si trovano dentro il compound
    a = b + 125
```

- Sequenza di azioni:
  - Si valuta l'espressione (a>b), ovvero la guardia in cui compare l'operatore Aritmetico 'maggiore di'
  - SE la guardia è vera (True), allora viene eseguito il corpo
  - SE la guardia è falsa (false), allora il corpo NON viene valutato dell'elaboratore

Condizione - istruzione if e Flow Chart

Linguaggio Python	Flow chart	Note
if(a>b): b =10 a = b + 125		Presenza di compound (si raggruppano due istruzioni)
if(a>b): b =10 a = b + 125		In questo caso una sola istruzione viene eseguita in base al verificarsi o meno della guarda dell'if

Esempi di istruzione if (1)

```
a = 6

b = a+1          #b ha valore 7, a ha valore 6

if (a<b):        # (a>b) → guardia

  b = 10         # in questo caso è VERO che a>b, quindi l'elaboratore
                 # esegue il corpo

  a = b + 125

  print("SE a è minore di b, Stampo il valore di a = b+125, con b=10,
        che risulta: "+str(a)+". \nIl valore di b ADESSO è: "+str(b))
print("\n\nDA qui PASSO SEMPRE!")
```

```
SE a è minore di b, Stampo il valore di a = b+125, con b=10, che risulta: 135.
Il valore di b ADESSO è: 10
```

CONSOLE

```
DA qui PASSO SEMPRE!
```

Indentazione (1)

- E’ importante scrivere programmi:
  - Sintatticamente e semanticamente validi: interpretabili dal compilatore (aspetto che sarà approfondito)
  - Leggibili: è necessario fare in modo che il codice che si sta scrivendo sia comprensibile da chi legge il programma (noi stessi, un nostro collega, etc.)
- Se si scrive un codice con una struttura chiara sarà più semplice risolvere/trovare eventuali errori logici che si stanno facendo
- Una delle regole di base per scrivere un programma leggibile è l’indentazione affiancata dall’uso dei commenti
- ATTENZIONE! Abbiamo visto che L’indentazione in Python determina il flusso di esecuzione delle istruzioni!!

Indentazione (2)

- L'indentazione consiste nell'inserire spazi o tabulazioni (che solitamente vengono ignorati dal compilatore) per mettere in luce eventuali gerarchie dei cicli o delle funzioni
- Ad esempio, torniamo alla nostra condizione if:

```
if(a>b):                    # (a>b) → guardia
    b =10                   # le istruzioni che sono indentate di un livello dopo l'if, costituiscono il corpo
    a = b + 125
c=10                        #fuori dal compound perchè è allo stesso LIVELLO dell'if
```

- Si noti che la sintassi è:

```
if(guardia):
        corpo
```

- Per la comprensibilità del codice, i commenti aiutano a capire la logica di funzionamento

Indentazione (3)

- Senza la tabulazione Python non da' errore ma il programma scritto è poco chiaro (si pensi a programmi complessi)

main.py saving...

```
1   if(10>2):print('Hello!')
2
3
```

```
Hello!
>
```

- Se si indenta in modo errato, Python da' errore!

main.py saved

```
1   if(10>2):
2   print('Hello!')
3
4
```

```
  File "main.py", line 2
    print('Hello!')
    ^
IndentationError: expected an indented block
>
```

- Ecco la sintassi chiara e corretta!

main.py saving...

```
1   if(10>2):
2     print('Hello!')
3
```

```
Hello!
>
```

Commenti (1)

- I Commenti in Python possono essere:
  - Su una riga di codice #commento
  - Su più righe:

```
"""
Questo
è un commento su più linee
"""
```

  - Oppure:

```
'''
Anche
questoo!
'''
```

  - Per i commenti su più righe servono tre ' a inizio e fine commento (oppure tre ")
  - NON vanno bene caratteri diversi da ' e " , ad esempio * o .
  - NON vanno bene ' e " ripetuti di un numero diverso da tre!

Commenti (2)

```
1   """
2   Questo
3   è un commento su più linee
4   """
5
6   if(10>2): #questo è un commento su una riga o inline
7       print('Hello!')
8
9   '''
10  Anche
11  questoo!
12  '''
13  ...
14  Questo No! Python darà errore
15  ...
16
17  ""
18  Questo
19  ... Neanche!!
20  """
```

OK!

OK!

OK!

Carattere Sbagliato (anche se ripetuto tre)

ERRORE!

Carattere giusto ma ripetuto solo due volte!!!

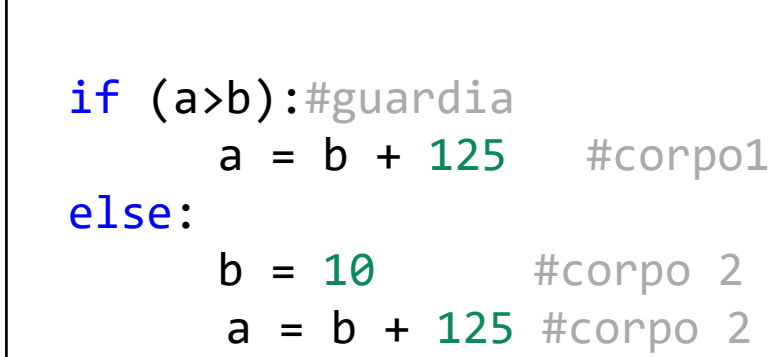
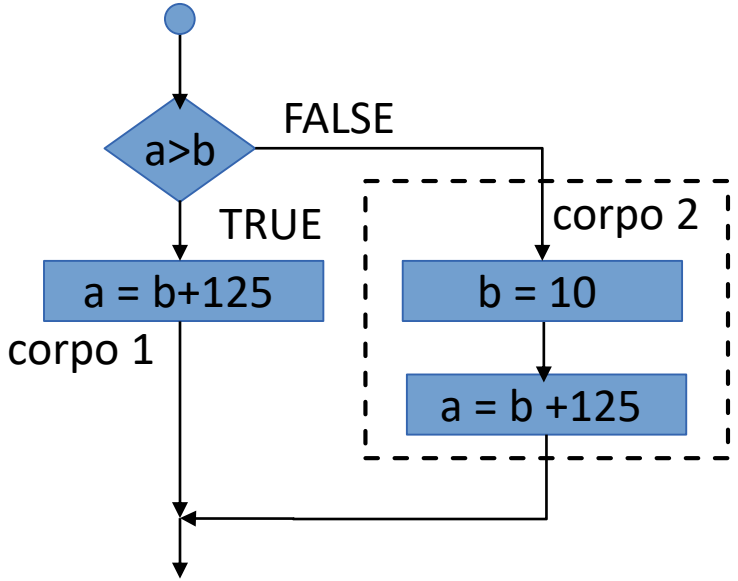
ERRORE!

Condizione – istruzione if else (1)

- L'istruzione **if**, può essere estesa tramite l'uso di **if else** per avere due corpi alternativi che vengono valutati o meno in base al valore della guardia
- Esempio:

```
a=5
b=9
if (a>b):#guardia
      a = b + 125   #corpo1
else:
      b = 10 #corpo 2 alternativo
      a = b + 125
```

Condizione – istruzione if else (2)

Linguaggio Python	Flow chart
	

```
if (a>b):#guardia
        a = b + 125    #corpo1
else:
        b = 10         #corpo 2
        a = b + 125 #corpo 2
```

Condizione – istruzione if else (3)

- L'istruzione **if**, può essere estesa tramite l'uso di **if else** per avere due corpi alternativi che vengono valutati o meno in base al valore della guardia
- Esempio:

```
a=5
b=9
if (a>b):#guardia
      a = b + 125   #corpo1
      print('passo dal corpo1')
else:
      b = 10 #corpo 2 alternativo
      a = b + 125
      print('passo dal corpo2')

print('\nPasso sempre da questa riga di codice')
```

Condizione – istruzione if else (4)

In questo caso la variabile intera a ha valore 20 e la variabile intera b ha valore 9, quindi è vero che a>b. Come conseguenza, si passa dal corpo1:

main.py saving...

```
1   a=20
2   b=9
3   if (a>b):#guardia
4       a = b + 125    #corpo1
5       print('passo dal corpo1')
6   else:
7       b = 10              #corpo 2 alternativo
8       a = b + 125
9       print('passo dal corpo2')
10
11  print('\nPasso sempre da questa riga di codice')
```

```
passo dal corpo1

Passo sempre da questa riga di codice
>
```

Condizione – istruzione if else (5)

In questo caso la variabile intera a ha valore 5 e la variabile intera b ha valore 9, quindi NON è vero che a>b. Come conseguenza, si passa dal corpo2:

main.py saved

```
1   a=5
2   b=9
3   if (a>b):#guardia
4       a = b + 125    #corpo1
5       print('passo dal corpo1')
6   else:
7       b = 10              #corpo 2 alternativo
8       a = b + 125
9       print('passo dal corpo2')
10
11  print('\nPasso sempre da questa riga di codice')
```

```
passo dal corpo2

Passo sempre da questa riga di codice
```

Python: Compilatori e Console

Esecuzione di programmi e ambienti: notebooks, IDE, console
- Il linguaggio python
  - o tipi mutabili e immutabili
  - o operatori ed espressioni
  - o istruzioni
  - o funzioni
  - o cicli while e for
  - o esecuzione condizionale
- Strutture dati e algoritmi elementari: Liste, Dizionari, Insiemi, iterazioni su strutture dati

Costo di esecuzione e complessità
Il modello di costo
Cenni sulla complessità di un algoritmo:
- Algoritmi di ordinamento su vettori
  - o Sequential-sort
- Cenni sugli alberi
  - o Alberi
  - o Alberi binari di ricerca: i) Visita in forma ricorsiva; ii) Ricerca; iii) Inserimento ordinato

Cenni su analisi dei dati, lettura e scrittura di file in forma tabulare, grafici.

Condizione – iterazione (1)

- Le istruzioni di iterazione permettono di eseguire ripetitivamente un corpo di istruzioni finché non si verifica una certa condizione sui valori delle variabili del programma

- Le istruzioni di iterazione sono le seguenti:
  - for
  - while

Condizione – iterazione: for (1)

```
for count in range(0,10):      #assegno ad il valore iniziale 0
    print(count)
```

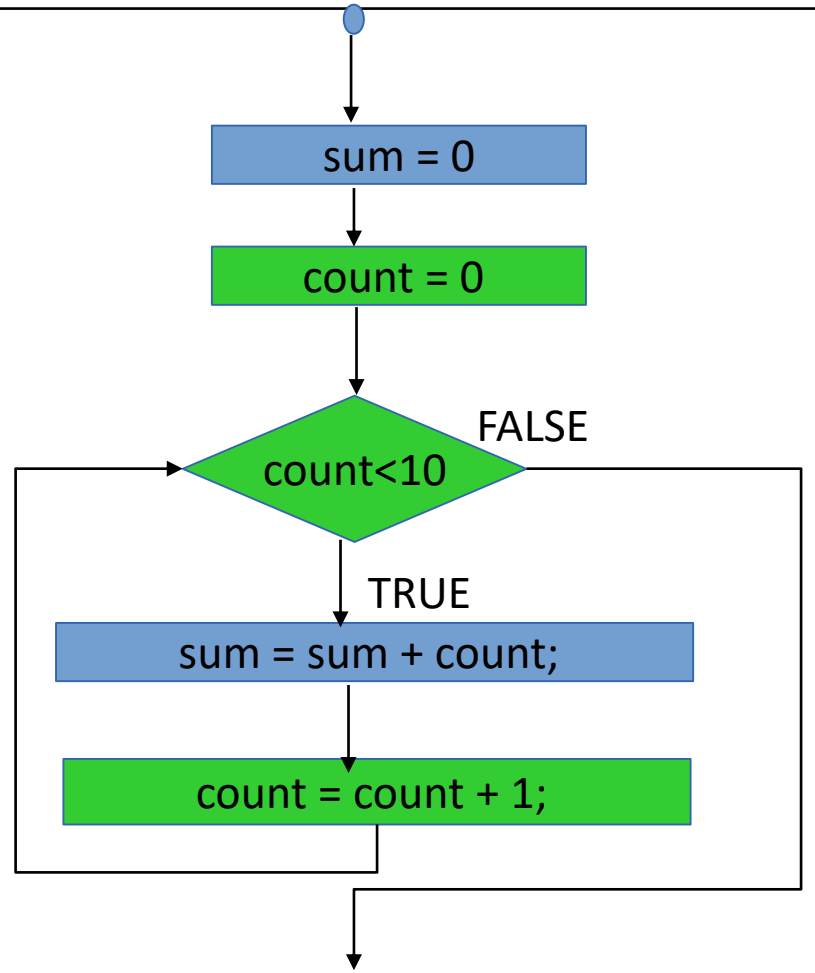
	Quando viene valutata	Descrizione
condizione_inizio ciclo (es: count = 0)	Prima di eseguire il ciclo (1 sola volta)	Usata per l'**inizializzazione** dell'indice del ciclo
Guardia (es: count <10)	Viene controllata prima della esecuzione di OGNI iterazione (es: 11 volte…)	usata per verificare l'**uscita** dal ciclo for
Incremento (es: count = count +1)	Incremento effettuato DOPO OGNI iterazione	Usato per **incrementare** l'indice del ciclo

Condizione – iterazione: for (2)

Linguaggio Python	Flow chart

```
sum = 0
count = 0
for count in range(0,10):
    sum = sum + count
```

#Con:
count =0 inizializzazione – si valuta a //INIZIO ciclo
count <10 guardia
count = count + 1 incremento



Condizione – iterazione: for (1)

Inizio ciclo
Indice a=0

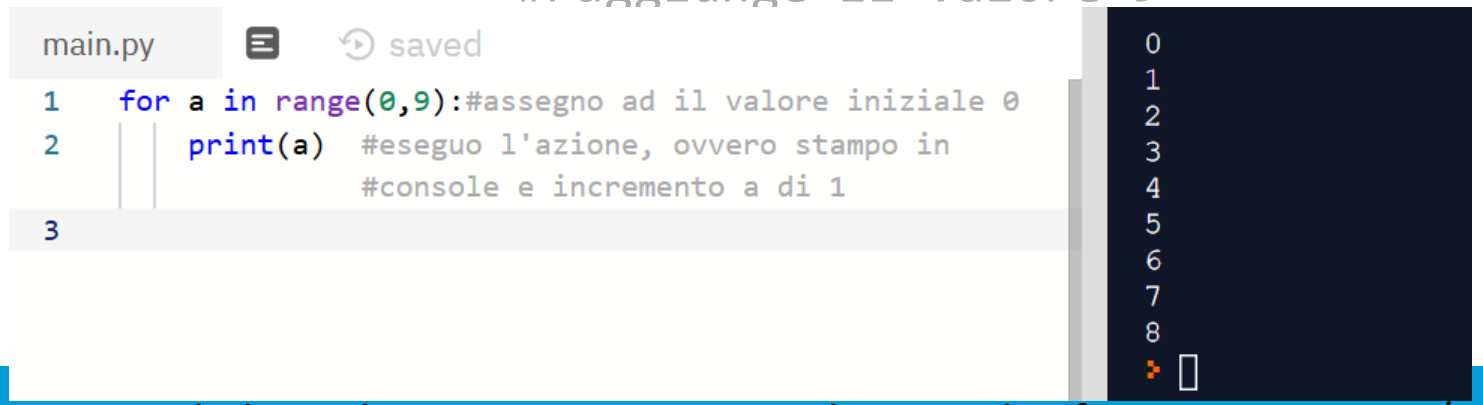
Fine ciclo
Indice a=8

```
for a in range(0,9):
    print(a) #corpo
```

Inizio ciclo

1. a=0, quindi entro nel corpo perché a è compreso fra 0 e 9, eseguo il corpo (stampo a ovvero 0 a console), incremento a di 1
2. a=1, quindi entro nel corpo perché a è compreso fra 0 e 9, eseguo il corpo (stampo a ovvero 1 a console), incremento a di 1
3. ..
8. a=9, quindi NON entro nel corpo perché a è compreso fra 0 e 9

```
for a in range(0,9):        #assegno ad il valore iniziale 0
    print(a)                #eseguo l'azione, ovvero stampo in #console e
                            #incremento a di 1. Lo faccio finché a non
                            #raggiunge il valore 9
```



Condizione – iterazione: for (2)

```
sum = 0

for count in range(0,10):

  print("entro nel ciclo for e count vale:"+str(count)+ "e sum adesso
            vale "+ str(sum)+"\n")

  sum = sum + count


print("\nAlla fine del ciclo for, sum avra' valore:"+str(sum)+"\n");
```

RISULTATO ????

Condizione – iterazione: for (3)

```
1    sum = 0;
2    for count in range(0,10):
3      print("entro nel ciclo for e count
       vale:"+str(count)+ "e sum adesso vale
       "+ str(sum)+"\n")
4      sum = sum + count
5
6    print("\nAlla fine del ciclo for, sum
     avra' valore:"+str(sum)+"\n");
```

```
entro nel ciclo for e count vale:0e sum adesso vale 0

entro nel ciclo for e count vale:1e sum adesso vale 0

entro nel ciclo for e count vale:2e sum adesso vale 1

entro nel ciclo for e count vale:3e sum adesso vale 3

entro nel ciclo for e count vale:4e sum adesso vale 6

entro nel ciclo for e count vale:5e sum adesso vale 10

entro nel ciclo for e count vale:6e sum adesso vale 15

entro nel ciclo for e count vale:7e sum adesso vale 21

entro nel ciclo for e count vale:8e sum adesso vale 28

entro nel ciclo for e count vale:9e sum adesso vale 36


Alla fine del ciclo for, sum avra' valore:45

>
```

Condizione – iterazione: for (4)

```
a=5
b=9

for a in range(0,9):
    print(a)

for a in range(9):
    print(a)
for a in range(5,9):
    print(a)
```

CHE DIFFERENZA C'E'
FRA QUESTI CICLI ????

Condizione – iterazione: for (5)

main.py saving...

```
1   a=5
2   b=9
3
4   for a in range(0,9):
5       print(a)
6
7
8
```

```
0
1
2
3
4
5
6
7
8
```

main.py saved

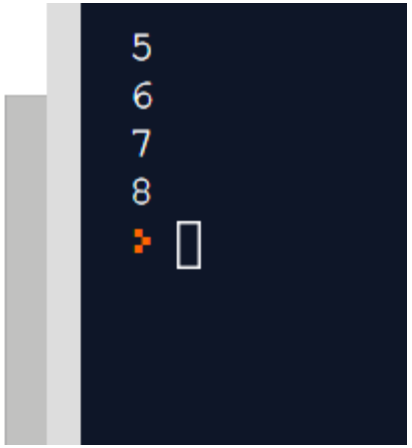
```
1   a=5
2   b=9
3
4   for a in range(9):
5       print(a)
6
7
8
9
```

```
0
1
2
3
4
5
6
7
8
```

Condizione – iterazione: for (5)

main.py saved

```
1    a=5
2    b=9
3
4    for a in range(5,9):
5        print(a)
6
```

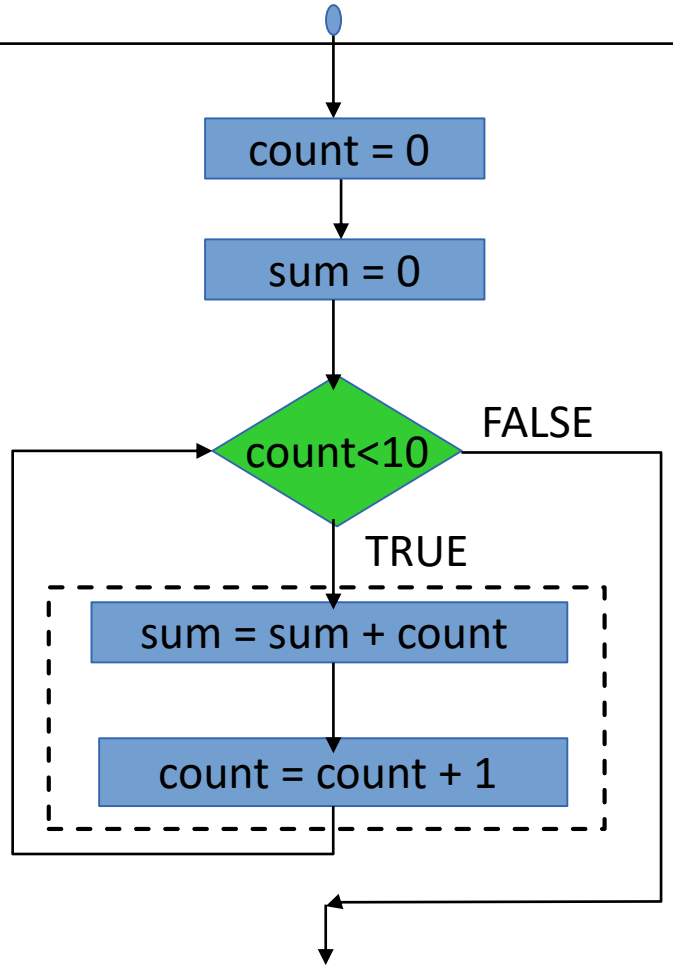


Condizione – iterazione: while (1)

```
count=0
sum=0
while count < 10:
        sum = sum + count
        count = count +1
```

	Quando viene valutata	Descrizione
Guardia (es: count <10)	Prima della esecuzione di OGNI iterazione (es: 11 volte…)	usato per verificare l'uscita dal ciclo while

Condizione – iterazione: while (2)

Linguaggio Python	Flow chart
`` count=0 sum=0 while count < 10: sum = sum + count count = count +1 ```	

Con:
count =0 inizializzazione
count <10 guardia
count = count + 1 incremento

Condizione – iterazione: while (3)

```
count=0

sum=0

while count < 10:

      print("entro nel ciclo while. La variabile count vale:«
       +str(count)+" e sum adesso vale: "+str(sum)+"\n")

      sum = sum + count

      count = count + 1


print("\nAlla fine del ciclo While, sum avrà valore:"+str(sum)+"\n")
```

Condizione – iterazione: while (4)

main.py saving...

```
1   count=0
2   sum=0
3   while count < 10:
4       print("entro nel ciclo while. La
        variabile count vale: "
5       +str(count)+" e sum adesso vale: "
        +str(sum)+"\n")
6       sum = sum + count
7       count = count + 1
8
9   print("\nAlla fine del ciclo While,
    sum avrà valore:"+str(sum)+"\n")
10
```

```
entro nel ciclo while. La variabile count vale: 0 e sum adesso vale: 0

entro nel ciclo while. La variabile count vale: 1 e sum adesso vale: 0

entro nel ciclo while. La variabile count vale: 2 e sum adesso vale: 1

entro nel ciclo while. La variabile count vale: 3 e sum adesso vale: 3

entro nel ciclo while. La variabile count vale: 4 e sum adesso vale: 6

entro nel ciclo while. La variabile count vale: 5 e sum adesso vale: 10

entro nel ciclo while. La variabile count vale: 6 e sum adesso vale: 15

entro nel ciclo while. La variabile count vale: 7 e sum adesso vale: 21

entro nel ciclo while. La variabile count vale: 8 e sum adesso vale: 28

entro nel ciclo while. La variabile count vale: 9 e sum adesso vale: 36


Alla fine del ciclo While, sum avrà valore:45

>
```

Esempio: calcolo del fattoriale (1)

- N! = N*(N-1)*(N-2)...*3*2*1
- Implementazione grazie all'uso del ciclo for

```
n=0
fact=1
N = 10
print('Calcolo di '+str(N)+'!')
for n in range(2,N+1):
    fact = fact *n #alla fine si avrà N!
    print("Fattoriale di "+str(n)+" = "+str(fact));

print("Alla fine si ha il fattoriale voluto: "+ str(N)+"!="+str(fact))
```

Esempio: calcolo del fattoriale (2)

- N! = N*(N-1)*(N-2)...*3*2*1
- Implementazione grazie all'uso del ciclo for

main.py saved

```
1   n=0
2   fact=1
3   N = 10
4   print('Calcolo di '+str(N)+'!')
5   for n in range(2,N+1):
6       fact = fact *n; #alla fine si avrà N!
7       print("Fattoriale di "+str(n)+" = "+str(fact));
8
9   print("Alla fine si ha il fattoriale voluto: "+ str
    (N)+"!="+str(fact))
10
```

```
Calcolo di 10!
Fattoriale di 2 = 2
Fattoriale di 3 = 6
Fattoriale di 4 = 24
Fattoriale di 5 = 120
Fattoriale di 6 = 720
Fattoriale di 7 = 5040
Fattoriale di 8 = 40320
Fattoriale di 9 = 362880
Fattoriale di 10 = 3628800
Alla fine si ha il fattoriale voluto: 10!=3628800
```

Esempio: Interazione da console (1)

- print() serve per scrivere su console
- Come fare per leggere da console?
- Si usa input()
- La 'funzione' input viene usata per permettere all'utente di immettere dati da tastiera dei valori che verranno poi utilizzati dal programma
- input accetta un singolo argomento opzionale: una stringa che viene mostrata a video prima di leggere il valore digitato
- Una volta che l'utente ha digitato un valore e premuto il tasto Invio, input restituisce il valore come stringa, come mostra il seguente esempio:

```
>>> nome = input("Inserisci il tuo nome:")
Inserisci il tuo nome:Michela
>>> print("Ciao "+nome+"!")
Ciao Michela!
>>>
```

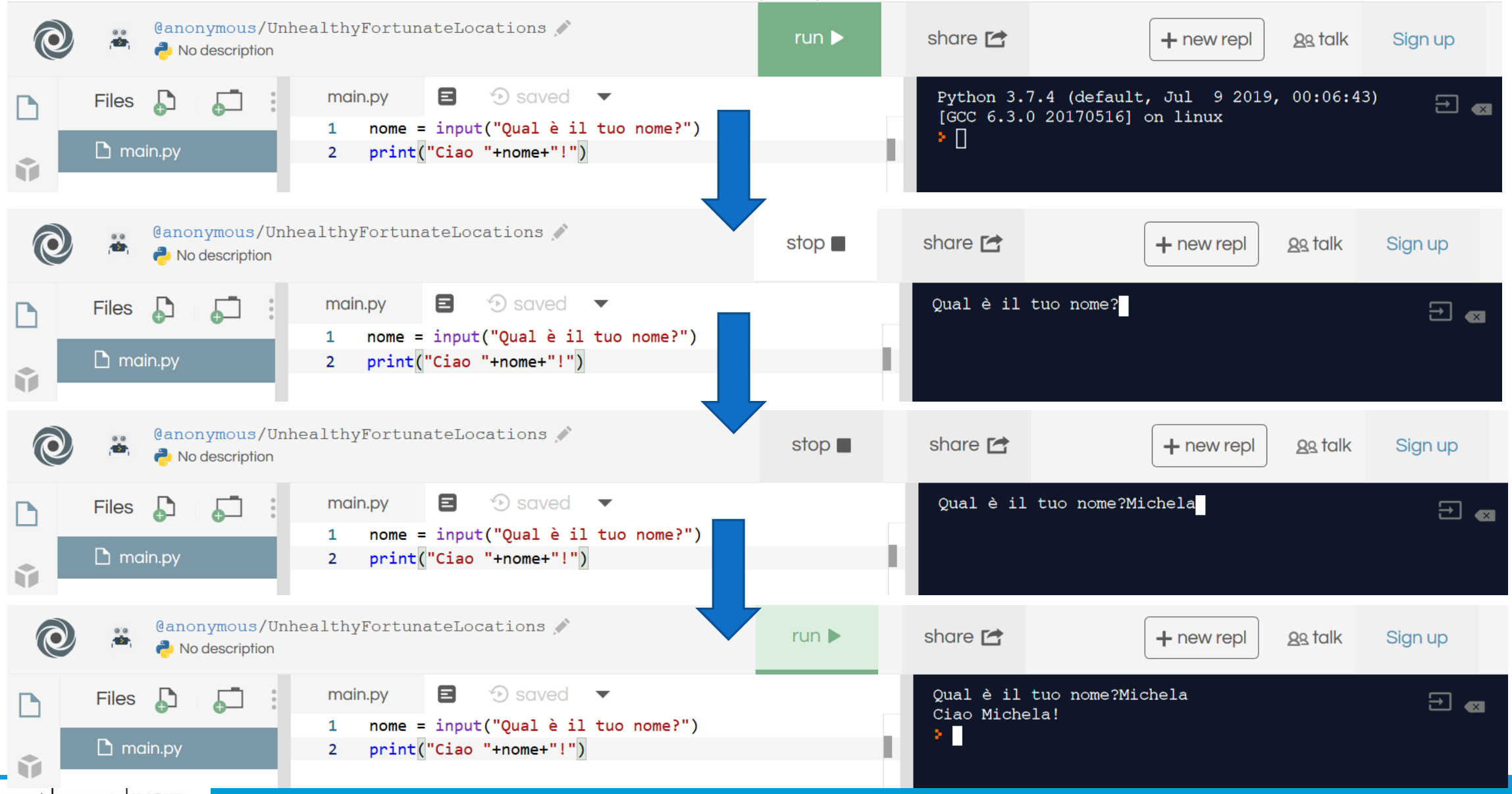
Inserimento da console (1)

- Python fornisce un insieme di funzioni predefinite che permettono di inserire dati da tastiera
- La funzione più semplice è: input()
- Quando questa funzione viene chiamata, il programma si ferma ed attende che l'operatore inserisca qualcosa, confermando poi l'inserimento con Invio (o Enter)
- A quel punto il programma riprende e input ritorna (restituisce) ciò che l'operatore ha inserito sotto forma di stringa

```
nome = input("Qual è il tuo nome?")
print("Ciao "+nome+"!")
```

Inserimento da console (2)

```
nome = input("Qual è il tuo nome?")
print("Ciao "+nome+"!")
```



Esempio: Interazione da console (2)

```
anni= 0
my_age = 8
print("Hello!")
c = input("Vuoi interagire? scrivi y se si: ")
if c == 'y': #operatore di uguaglianza
    anni = input("Stiamo parlando\nQuanti anni hai?")
    a = int(anni)
    if a==my_age:
        print("Anche io ho "+str(anni)+" anni!\n") #corpo1
    elif a<my_age: #corpo2:
        print("Io sono più grande di te! Ho "+str(my_age)+str(" anni!"))
    else: #corpo2:
        print("Io sono più piccola di te! Ho "+str(my_age)+str(" anni!"))
else:
    print("ADDIO!\n")
```

- Alla domanda 'Quanti anni hai', l'utente immette un intero (ciò che ci si aspetta)
- Il programma tramite la input associa la stringa 8 alla variabile anni
- anni è pero' una stringa
- Per fare un confronto tra la stringa anni e la variabile intera my_age è necessario usare il cast
- E quindi ha senso scrivere:

```
if a==my_age:
  print('Ciao')
```

- Il programma infatti stampa ….

Caso 1: stessa età

main.py saved

```
1   anni= 0
2   my_age = 8
3   print("Hello!")
4   c = input("Vuoi interagire? scrivi y se si: ")
5   if c == 'y': #operatore di uguaglianza
6       anni = input("Stiamo parlando\nQuanti anni hai?")
7       a = int(anni)
8       if a==my_age:
9           print("Anche io ho "+str(anni)+" anni!\n")
            #corpo1
10      elif a<my_age: #corpo2:
11          print("Io sono più grande di te! Ho "+str
            (my_age)+str(" anni!"))
12      else: #corpo2:
13          print("Io sono più piccola di te! Ho "+str
            (my_age)+str(" anni!"))
14  else:
15      print("ADDIO!\n")
```

```
Hello!
Vuoi interagire? scrivi y se si: y
Stiamo parlando
Quanti anni hai?8
Anche io ho 8 anni!
```

Caso 2: la programmatrice ha età maggiore dell'utente che interagisce in console

main.py saved

```
1   anni= 0
2   my_age = 8
3   print("Hello!")
4   c = input("Vuoi interagire? scrivi y se si: ")
5   if c == 'y': #operatore di uguaglianza
6       anni = input("Stiamo parlando\nQuanti anni hai?")
7       a = int(anni)
8       if a==my_age:
9           print("Anche io ho "+str(anni)+" anni!\n")
            #corpo1
10      elif a<my_age: #corpo2:
11          print("Io sono più grande di te! Ho "+str
            (my_age)+str(" anni!"))
12      else: #corpo2:
13          print("Io sono più piccola di te! Ho "+str
            (my_age)+str(" anni!"))
14  else:
15      print("ADDIO!\n")
```

```
Hello!
Vuoi interagire? scrivi y se si: y
Stiamo parlando
Quanti anni hai?10
Io sono più piccola di te! Ho 8 anni!
```

Caso 3: la programmatrice ha età maggiore dell'utente che interagisce in console

main.py saving...

```
1   anni= 0
2   my_age = 8
3   print("Hello!")
4   c = input("Vuoi interagire? scrivi y se si: ")
5   if c == 'y': #operatore di uguaglianza
6       anni = input("Stiamo parlando\nQuanti anni hai?")
7       a = int(anni)
8       if a==my_age:
9           print("Anche io ho "+str(anni)+" anni!\n")
            #corpo1
10      elif a<my_age: #corpo2:
11          print("Io sono più grande di te! Ho "+str
            (my_age)+str(" anni!"))
12      else: #corpo2:
13          print("Io sono più piccola di te! Ho "+str
            (my_age)+str(" anni!"))
14  else:
15      print("ADDIO!\n")
16
```

```
Hello!
Vuoi interagire? scrivi y se si: y
Stiamo parlando
Quanti anni hai?6
Io sono più grande di te! Ho 8 anni!
>
```

Caso 4: nessuna interazione

main.py saved

```
1   anni= 0
2   my_age = 8
3   print("Hello!")
4   c = input("Vuoi interagire? scrivi y se si: ")
5   if c == 'y': #operatore di uguaglianza
6       anni = input("Stiamo parlando\nQuanti anni hai?")
7       a = int(anni)
8       if a==my_age:
9           print("Anche io ho "+str(anni)+" anni!\n")
            #corpo1
10      elif a<my_age: #corpo2:
11          print("Io sono più grande di te! Ho "+str
            (my_age)+str(" anni!"))
12      else: #corpo2:
13          print("Io sono più piccola di te! Ho "+str
            (my_age)+str(" anni!"))
14  else:
15      print("ADDIO!\n")
```

```
Hello!
Vuoi interagire? scrivi y se si: noooooooooooo
ADDIO!

>
```

Esempio: somma dei primi N numeri pari (1)

```
sum= 0
N=4
print("Somma dei primi "+str(N)+" numeri pari:")
for i in range(1,N+1):
      sum += i*2
      print("aggiungo: "+str(i*2));
      print("------somma parziale: "+str(sum));

print("\nSomma: "+str(sum));
```

Adesso sto calcolando la somma dei primi 4 numeri pari, basta cambiare questa riga di codice per fare la somma dei primi 10 interi, 20, etc.

Non voglio sommare 0 (non avrebbe senso)
Voglio sommare anche i=10
Quindi si scrive range (1,N+1)

Esempio: somma dei primi N numeri pari (2)

main.py saving...

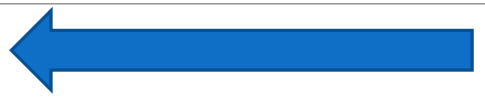
```
1   sum= 0
2   N=4
3   print("Somma dei primi "+str(N)+" numeri pari:")
4   for i in range(1,N+1):
5       sum += i*2
6       print("aggiungo: "+str(i*2));
7       print("------somma parziale: "+str(sum));
8
9   print("\nSomma: "+str(sum));
10
```

```
Somma dei primi 4 numeri pari:
aggiungo: 2
------somma parziale: 2
aggiungo: 4
------somma parziale: 6
aggiungo: 6
------somma parziale: 12
aggiungo: 8
------somma parziale: 20

Somma: 20
```

Python: Console

Esecuzione di programmi e ambienti: notebooks, IDE, console



- Il linguaggio python
  - o tipi mutabili e immutabili
  - o operatori ed espressioni
  - o istruzioni
  - o funzioni
  - o cicli while e for
  - o esecuzione condizionale
- Strutture dati e algoritmi elementari: Liste, Dizionari, Insiemi, iterazioni su strutture dati

Costo di esecuzione e complessità

Il modello di costo

Cenni sulla complessità di un algoritmo:

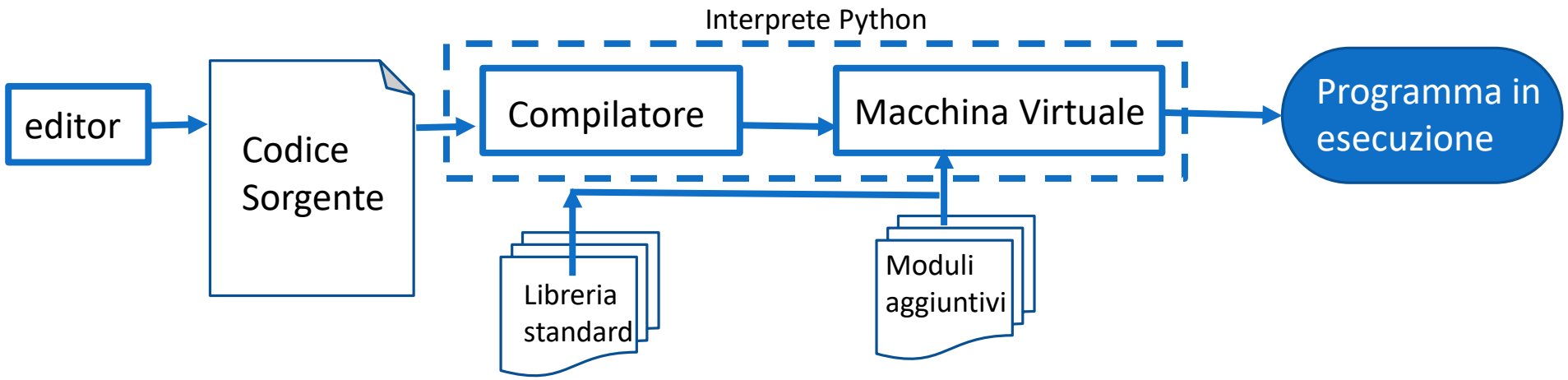
- Algoritmi di ordinamento su vettori
  - o Sequential-sort
- Cenni sugli alberi
  - o Alberi
  - o Alberi binari di ricerca: i) Visita in forma ricorsiva; ii) Ricerca; iii) Inserimento ordinato

Cenni su analisi dei dati, lettura e scrittura di file in forma tabulare, grafici.

Python: Console
Scrivere programmi da riga di comando

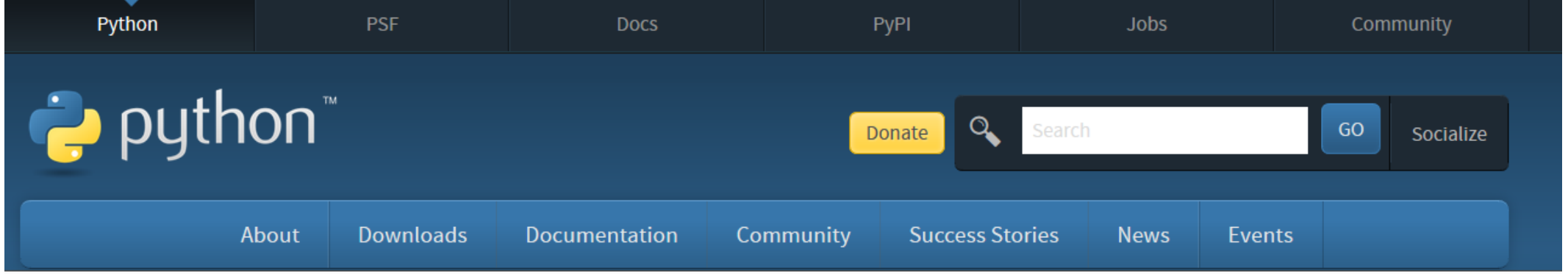
Compilatori

- Per far girare codice Python e scrivere programmi in Python, esistono varie possibilità
- All'inizio può essere utile ricorrere ai compilatori online, ce ne sono diversi ma ovviamente necessitano di una connessione
- Da riga di comando, è possibile scaricare e installare Python (useremo la versione 3.8) e poi aprire la Console
- Per i programmi più complessi si userà una IDE, in particolare vedremo Spyder



Scaricare Python (versione 3.8) (1)

- https://www.python.it/download/



https://www.python.org/downloads/

Python >>> Downloads >>> Windows

https://www.python.org/downloads/windows/

Python Releases for Windows

- Latest Python 3 Release - Python 3.8.0
- Latest Python 2 Release - Python 2.7.17

Stable Releases

- Python 2.7.17 - Oct. 19, 2019
  - Download Windows debug information files
  - Download Windows debug information files for 64-bit binaries
  - Download Windows help file
  - Download Windows x86-64 MSI installer
  - Download Windows x86 MSI installer
- Python 3.7.5 - Oct. 15, 2019

  Note that Python 3.7.5 *cannot* be used on Windows XP or earlier.

  - Download Windows help file
  - Download Windows x86-64 embeddable zip file

Pre-releases

- Python 3.5.8rc2 - Oct. 12, 2019
  - No files for this release.
- Python 2.7.17rc1 - Oct. 9, 2019
  - Download Windows debug information files
  - Download Windows debug information files for 64-bit binaries
  - Download Windows help file
  - Download Windows x86-64 MSI installer
  - Download Windows x86 MSI installer
- Python 3.7.5rc1 - Oct. 2, 2019
  - Download Windows help file
  - Download Windows x86-64 embeddable zip file

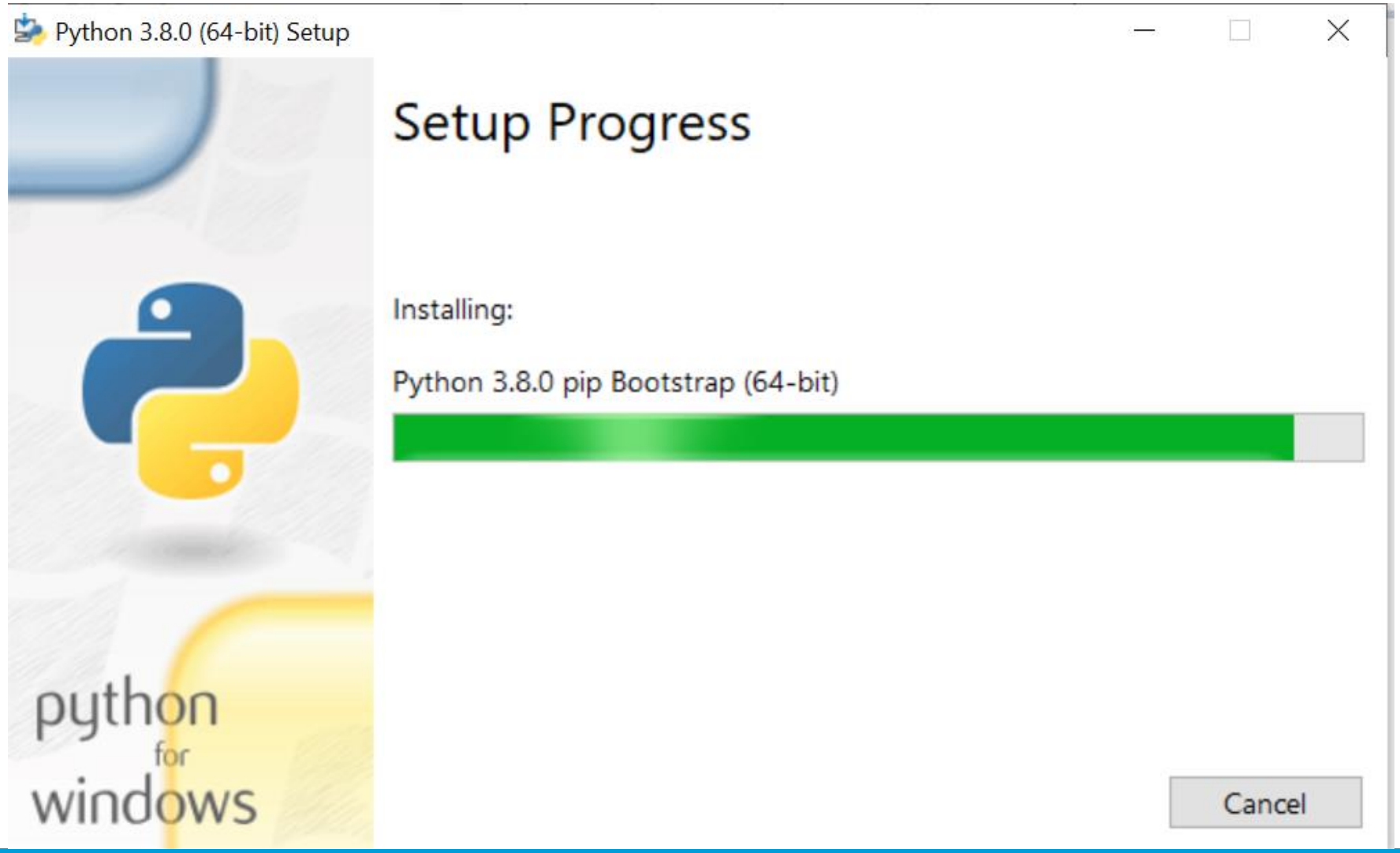
Scaricare Python (versione 3.8) (3)

Files

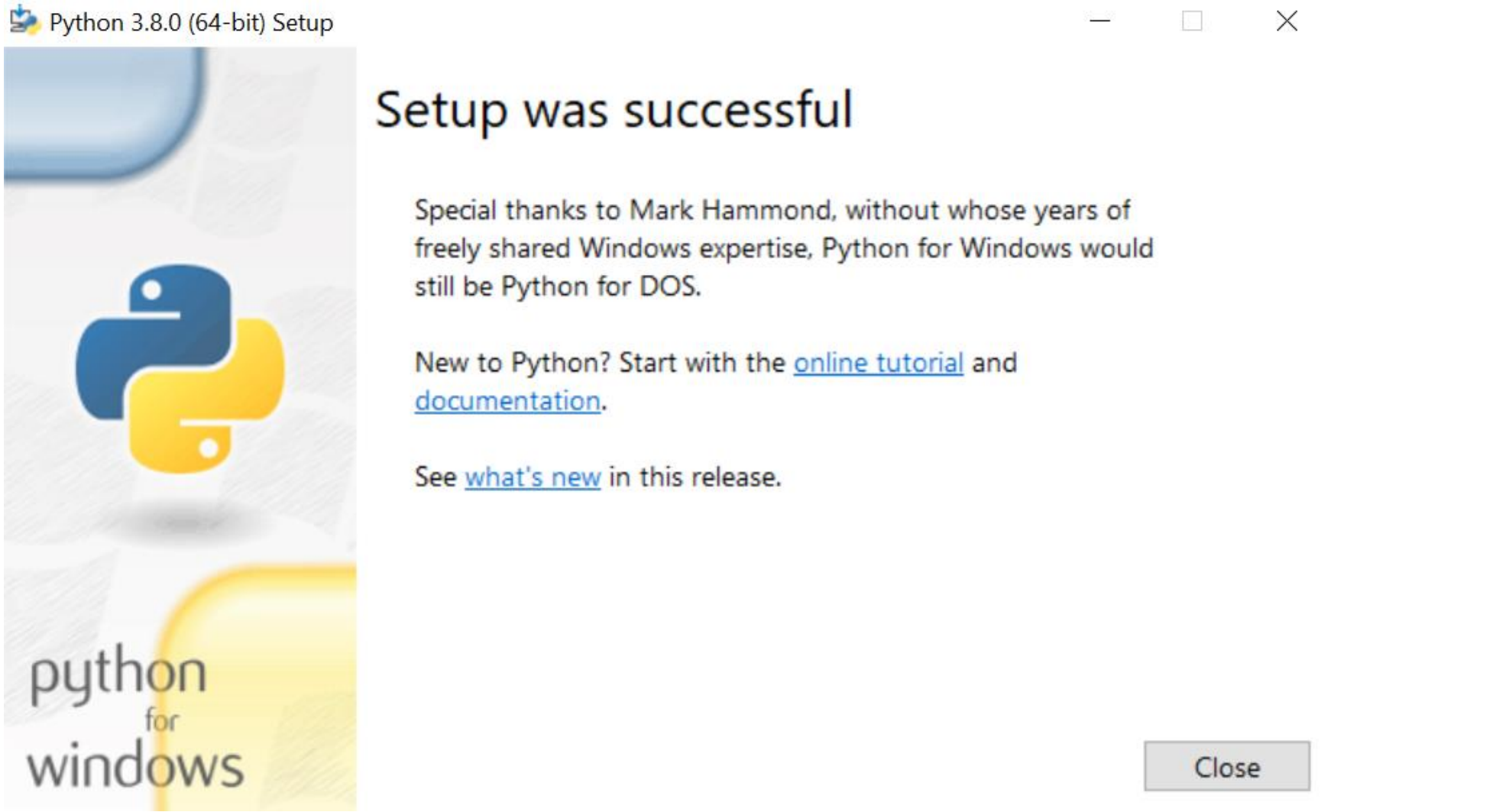
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		e18a9d1a0a6d858b9787e03fc6fdaa20	23949883	SIG
XZ compressed source tarball	Source release		dbac8df9d8b9edc678d0f4cacdb7dbb0	17829824	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	f5f9ae9f416170c6355cab7256bb75b5	29005746	SIG
Windows help file	Windows		1c33359821033ddb3353c8e5b6e7e003	8457529	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	99cca948512b53fb165084787143ef19	8084795	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	29ea87f24c32f5e924b7d63f8a08ee8d	27505064	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	f93f7ba8cd48066c59827752e531924b	1363336	SIG
Windows x86 embeddable zip file	Windows		2ec3abf05f3f1046e0dbd1ca5c74ce88	7213298	SIG
Windows x86 executable installer	Windows		412a649d36626d33b8ca5593cf18318c	26406312	SIG
Windows x86 web-based installer	Windows		50d484ff0b08722b3cf51f9305f49fdc	1325368	SIG

Scaricare Python (versione 3.8) (4)

Scaricare Python (versione 3.8) (5)

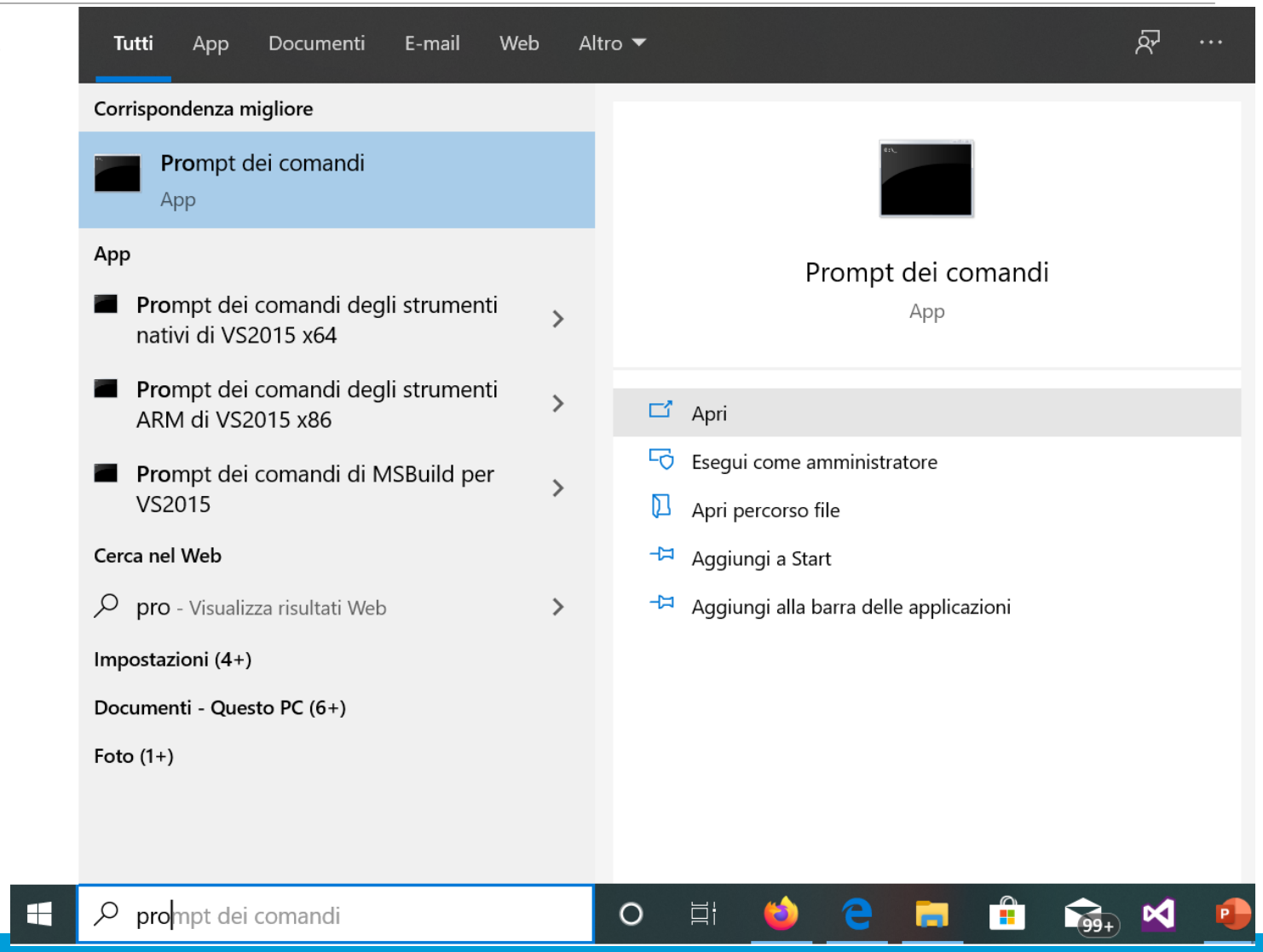


Scaricare Python (versione 3.8) (6)



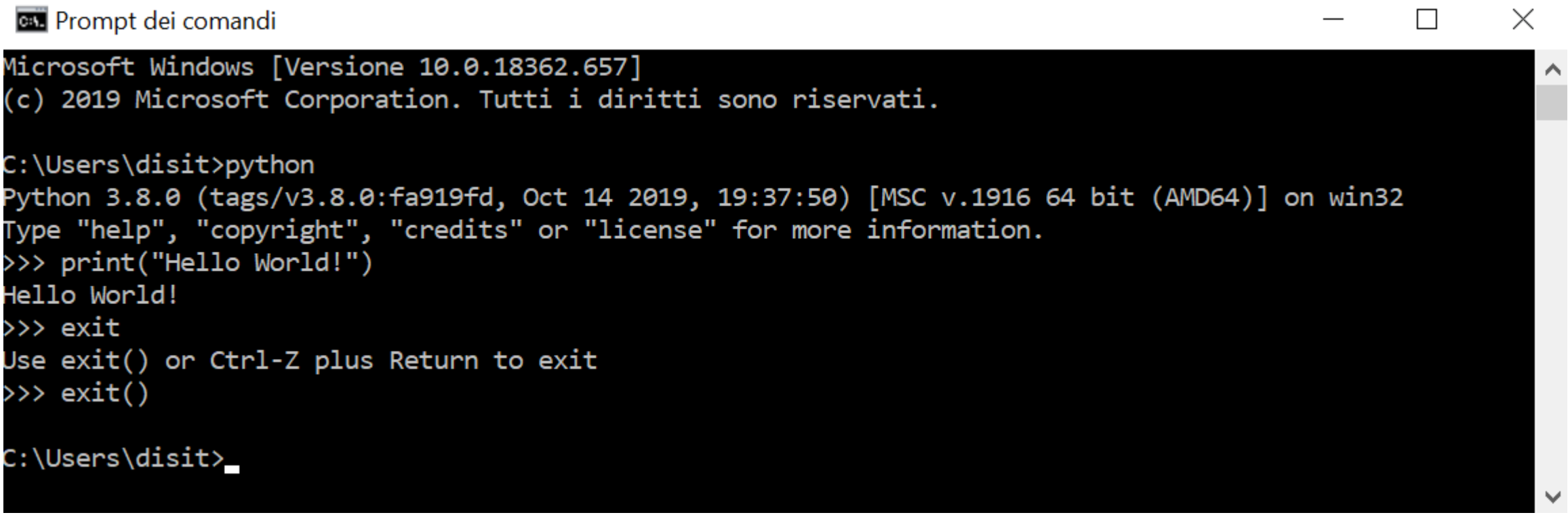
Lanciare Python (versione 3.8) (1)

- Aprire un promtp dei comandi
- Andare nella directory in cui è stato istallato python



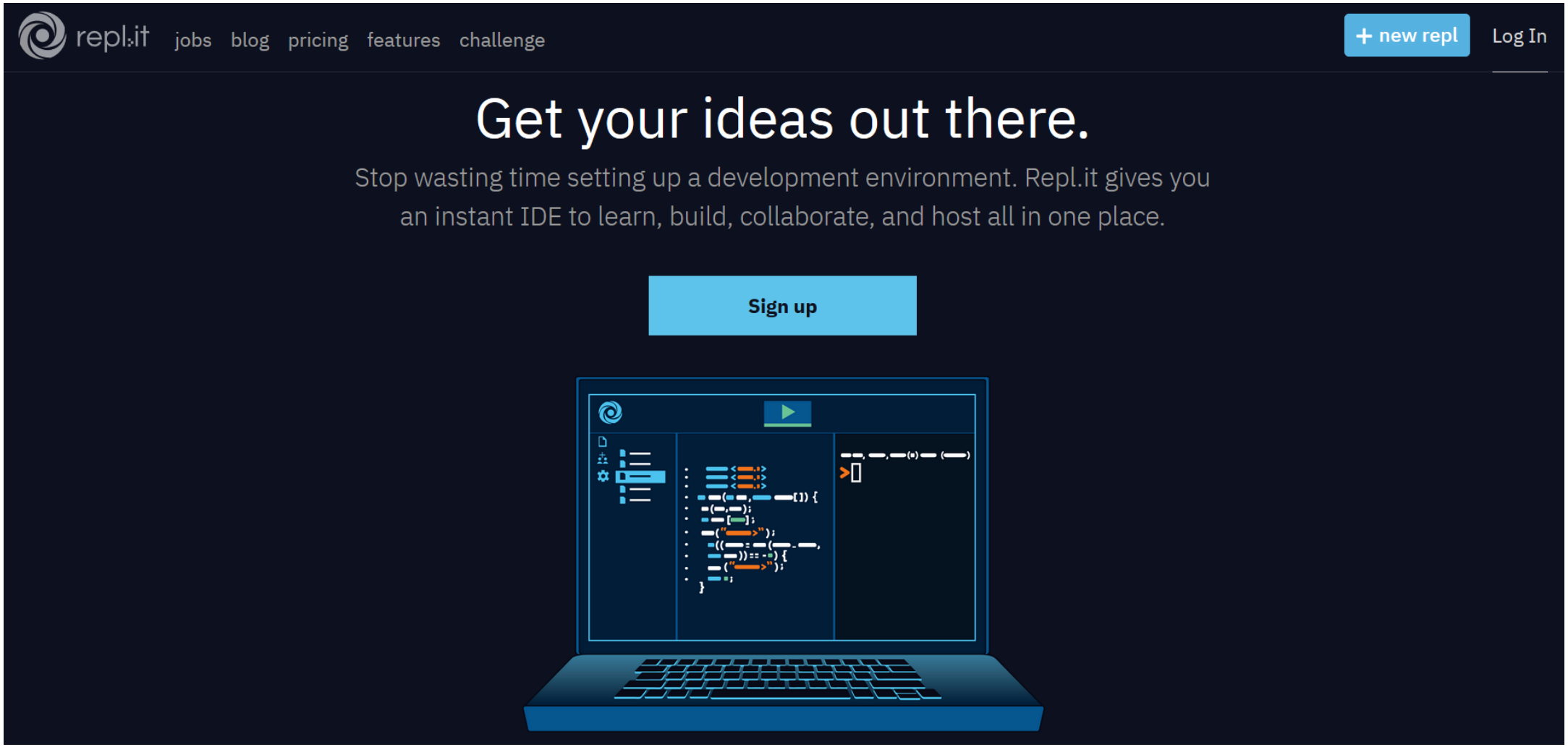
Scaricare Python (versione 3.8) (3)

- Aprire un promtp
- Per aprire la console Python, digitare: python
- Per chiudere la console Python, digitare: exit() Oppure la combinazione dei tasti è: Ctrl+Z



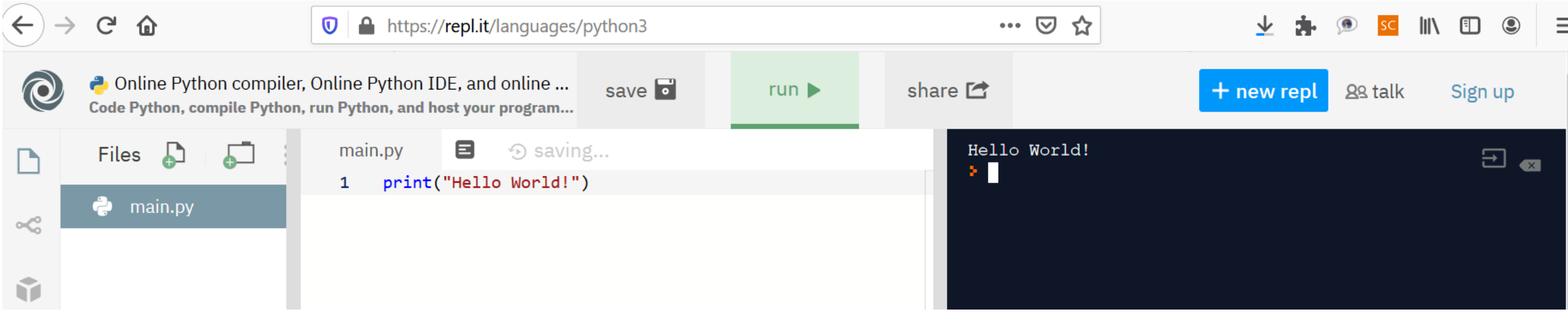
Python: Compilatori Online

Compilatori online: Repl.it (1)



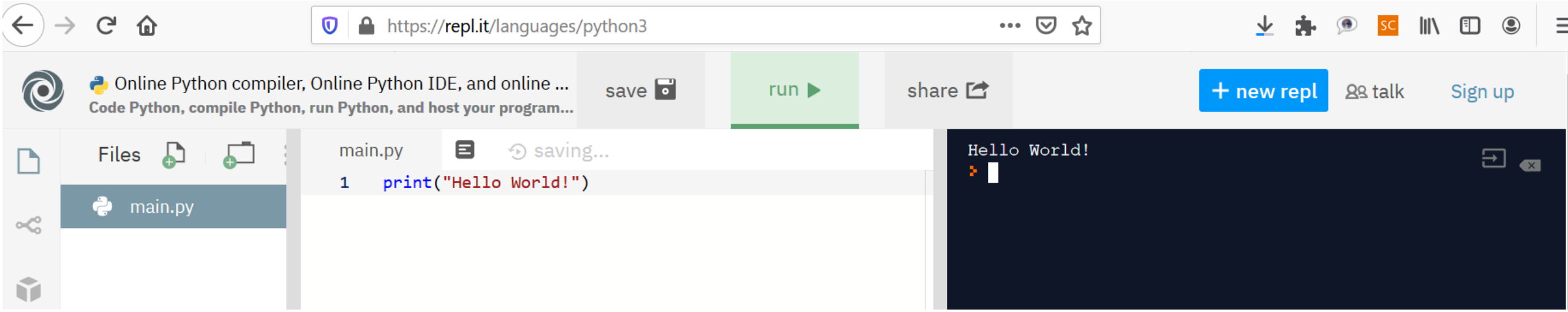
Compilatori online: Repl.it (2)

- https://repl.it/languages/python3
- nel riquadro centrale è possibile scrivere codice Python
- Cliccando sul bottone RUN, è possibile far girare il programma e vedere i risultati prodotti in Console (riquadro a destra)



Compilatori online: Repl.it (2)

- https://repl.it/languages/python3
- nel riquadro centrale è possibile scrivere codice Python
- Cliccando sul bottone RUN, è possibile far girare il programma e vedere i risultati prodotti in Console (riquadro a destra)



Fine delle introduzione al Python

Programma del corso - Cognomi A-L

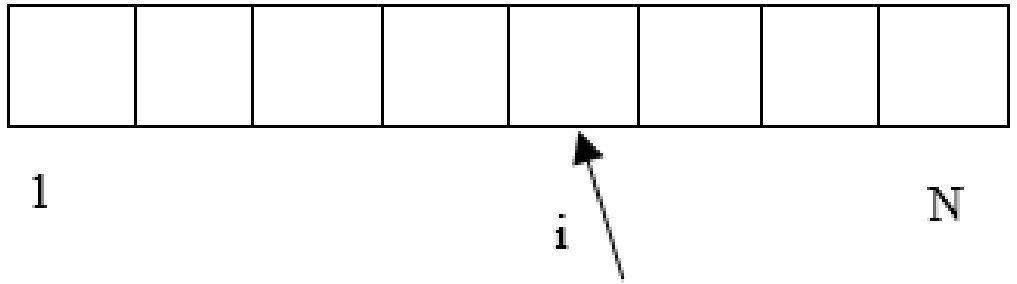
- Introduzione al linguaggio python
  - Tipi, variabili e costanti
  - Operatori ed espressioni
  - Istruzioni
- Rappresentazione dei dati
  - Numeri
  - Interi
  - Caratteri e stringhe

Elementi di sintassi di un linguaggio

Strutture astratte

Il vettore (1)

- Il vettore è un insieme ordinato di elementi dello stesso tipo
- Il vettore ha elementi ordinati nel senso che è possibile riferirsi alle singole componenti indicandole per mezzo di un indice (indice i in figura)
- L'indice identifica in modo ordinato gli elementi del vettore, dal primo all'ultimo secondo un criterio di ordinamento (per esempio in ordine crescente)



Il vettore (2)

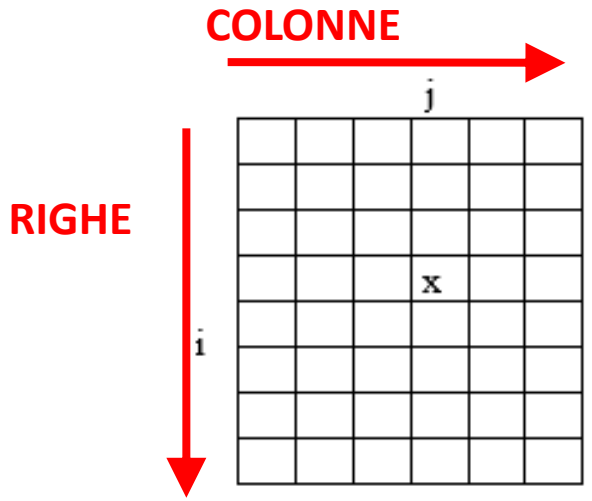
- Il vettore ha una dimensione pari al numero di componenti/elementi di cui è composto
- La dimensione è FISSA, ovvero NON può essere variata dopo la sua realizzazione
- I vettori sono tipicamente strutture statiche
- Gli elementi sono tutti dello stesso tipo, quindi il vettore è una struttura omogenea
- Ad esempio si possono avere vettori di:
  - numeri interi
  - reali
  - Booleani
  - etc.

Il vettore (3)

- Il concetto di vettore è di fondamentale importanza applicativa in ogni disciplina scientifica
- L'idea di **aggregare informazioni omogenee** in una unica entità e di poterle reperire successivamente in base alla loro posizione, porta spesso a **formulare problemi in modo compatto ed elegante**
- Il vettore in ambito informatico risulta una struttura ideale per rappresentare tabelle contenenti informazioni, soprattutto quando queste sono di **natura statica**
- Un vettore V che ha dimensione N, viene rappresentato indicando i suoi elementi con V_i o V(i) ma viene anche considerato nella sua interezza tramite V
- V_i o V(i) rappresenta l'elemento i-esimo del vettore V

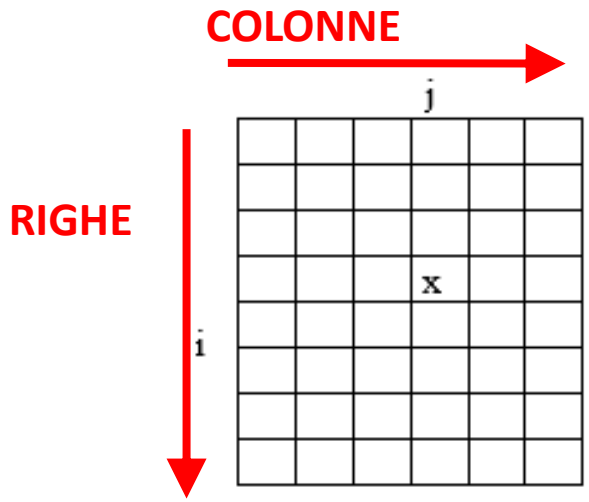
La matrice (1)

- La matrice è la naturale estensione bidimensionale del vettore
- La matrice ha elementi ordinati, ovvero è possibile riferirsi alle singole componenti indicandole per mezzo di due indici partendo dal primo all'ultimo secondo un criterio di ordinamento che tiene conto di entrambe le dimensioni
- Sia A(i,j) l'elemento della matrice A con:
  - i, indica la riga partendo dall'alto
  - j, indica la colonna a partire da sinistra



La matrice (2)

- Le matrici devono essere composte da **elementi dello stesso tipo**, pertanto sono strutture **omogenee**
- Le matrici, come i vettori, sono **strutture statiche**, ovvero non è possibile cambiare le dimensioni della matrice dopo la sua creazione iniziale
- La matrice può avere un numero di colonne diverso dal numero di righe (matrice rettangolare)
- Si chiamano matrici quadrate quelle che hanno il numero di colonne uguale al numero di righe



Il Record (1)

- Un record è un insieme **ordinato** ed **eterogeneo** di elementi
- Il singolo elemento del record può essere chiamato campo o componente
- In un record si possono avere componenti di vario tipo: interi, numeri naturali, vettori di interi, etc.
- Il record è tipicamente una struttura statica nel senso che non è possibile variarne il numero di componenti dopo la sua creazione iniziale
- Esempio:
  - cognome: Mario
  - nome: Rossi
  - via: Via Ponte Verde
  - civico: 45

Il Record (2)

- Tenendo presente l'esempio:
  - cognome: Mario
  - nome: Rossi
  - via: Via Ponte Verde
  - civico: 45
- Ogni campo del record viene identificato dal nome del campo:
  - cognome è il nome del primo campo, e *Rossi* è il relativo contenuto informativo (valore)
  - cognome è una stringa mentre civico è un intero
- Il record si può rappresentare nel modo seguente:
  - {cognome, nome, via, civico}
- Tipicamente il record stesso o meglio la struttura che lo definisce, ha un nome

La Tabella (1)

- Le tabelle sono una generalizzazione del concetto di matrice
- La tabella, a differenza della matrice, può contenere dati di tipo diverso. Ad esempio: numeri, commenti, descrizioni, etc.
- Tipicamente ad ogni colonna della tabella è associato un tipo e quindi un significato
- La tabella è **un insieme ordinato ed omogeneo di record**
- Tipicamente la tabella è una struttura dati statica, ovvero le sue dimensioni NON variano dopo la sua realizzazione iniziale

La Tabella (2)

- Un record della tabella può essere identificato in due modi:
  - Per indice posizionale
  - Per chiave
- L'**indice posizionale** è l'indice che identifica il singolo record partendo dal primo fino all'ultimo. In questo senso la tabella può essere vista come un insieme di record

Nome	Cognome	Età	Telefono	Indice
Carlo	Zolli	14	0123413412	1
Pino	Polli	24	0234242444	2
Gino	Arbi	4	0345543535	3
Lino	Corbi	56	0455436266	4
….	….	….	….	…

Tabella con accesso per chiave (1)

- Per **chiave** si intende che il singolo record viene identificato per mezzo del contenuto stesso di un campo del record
- Ad esempio nella tabella sottostante potrebbe essere usato il numero di telefono come chiave, oppure l'età, il nome, etc.
- Individuare un record per chiave significa identificare la posizione del record nella tabella
- Esempio: il record di Gino è in posizione di indice 3, l'indice di ordinamento va da 1 a N

Nome	Cognome	Età	Telefono	Indice
Carlo	Zolli	14	0123413412	1
Pino	Polli	24	0234242444	2
Gino	Arbi	4	0345543535	3
Lino	Corbi	56	0455436266	4
….	….	….	….	…

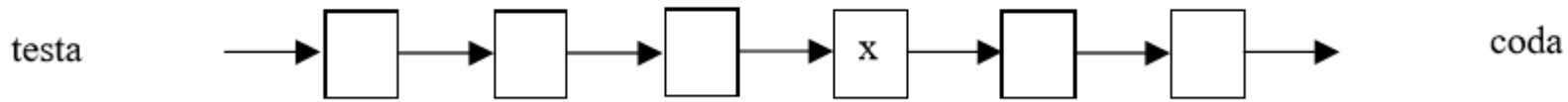
Tabella con accesso per chiave (2)

- La chiave per la quale si effettua una ricerca al fine di individuare il singolo record dovrebbe essere sufficientemente significativa da identificare in modo univoco il record
- Questo potrebbe NON essere possibile (es: se si ordina per cognome, potrebbero esserci due o più cognomi uguali)
- Per risolvere il problema si possono assegnare delle chiavi univoche oppure comporre campi dei record in modo da definire delle chiavi più significative (es: nome + cognome)

Nome	Cognome	Età	Telefono	Indice
Carlo	Zolli	14	0123413412	1
Pino	Polli	24	0234242444	2
Gino	Arbi	4	0345543535	3
Lino	Corbi	56	0455436266	4
….	….	….	….	…

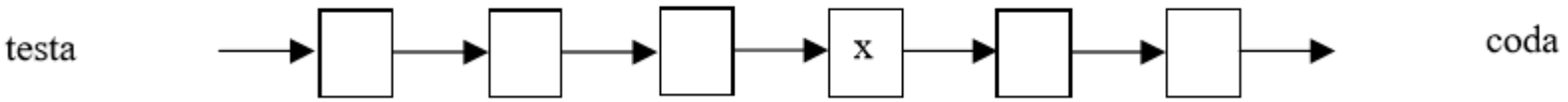
La Lista (1)

- Le liste sono strutture dati che possono essere omogenee e non omogenee, lineari e non lineari
- Si dicono **liste lineari** quelle organizzate come un insieme ordinato di elementi
- Ogni elemento ha un successivo e un predecessore ad esclusione:
  - del primo elemento (**testa**) che NON ha un predecessore
  - dell'ultimo elemento (**coda**) che non ha un successivo
- Per accedere all'elemento finale è necessario partire dal primo elemento della lista (testa) e scorrere su tutti gli elementi successivi fino all'ultimo (coda)
- Questo tipo di organizzazione per l'accesso alle componenti viene detto **sequenziale**



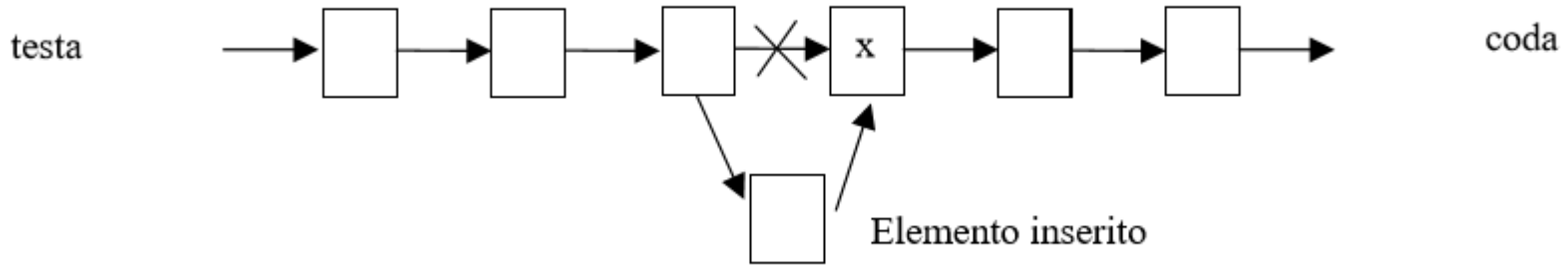
La Lista (2)

- Le liste sono strutture dati ad accesso **sequenziale**
- La lista può contenere tipicamente un **numero variabile di elementi nel tempo**, ovvero la sua dimensione in termini di elementi può variare dopo la sua creazione iniziale, quindi deve essere considerata una struttura **dinamica**
- Al momento della sua creazione, la lista è tipicamente **vuota**, in seguito vengono inseriti gli elementi connettendoli secondo regole di ordinamento o convenienza
- Su una lista si possono effettuare operazioni di:
  - Inserimento, cancellazione, visita, modifica, creazione, concatenazione di due liste, etc.
  - Una lista si crea inserendo il **primo elemento**



Inserimento in una lista (1)

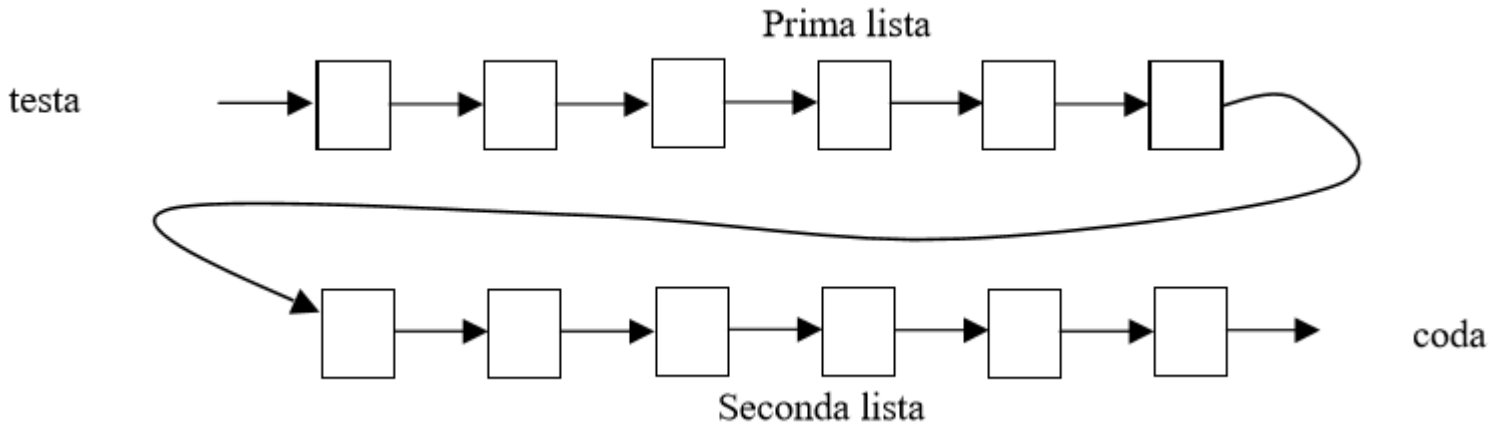
- L'inserimento in testa e in coda è semplice da comprendere: si aggiunge un ulteriore elemento e lo si collega alla lista
- L'inserimento di un elemento all'interno di una lista deve essere effettuato 'aprendo' la lista e 'ristabilendo' i legami tra gli elementi
- Si possono inserire anche elementi in testa e in coda con altre modalità…
- Se NON vi sono elementi, allora la lista è vuota
- Una lista con un solo elemento può diventare vuota se questo unico elemento viene cancellato



Concatenazione di liste

- Due liste possono essere concatenate nel senso che una lista può essere congiunta con un'altra in modo da formare un'unica lista con tutti gli elementi

- Questo avviene effettuando il collegamento tra la coda della prima lista e la testa della seconda lista



Differenza tra lista e vettore

- Le maggiori differenze tra liste e vettori sono le seguenti:
  - **Il vettore ha un accesso diretto ai singoli elementi MENTRE la lista ha un accesso sequenziale**

- La lista può avere un numero variabile di elementi nel tempo ed è quindi una struttura **dinamica**, al contrario del vettore che è una struttura **statica**

- La lista può anche contenere elementi di tipo diverso

- La lista può anche essere NON lineare

Descrizione di liste lineari e non lineari

- Gli elementi di una lista B possono essere elencati nel modo seguente:
  - B (3,5,6,77,88,92) -> lista lineare
- In questo caso la lista ha 6 elementi
- Su una lista sono possibili le operazioni di: ricerca, cancellazione, concatenazione, visita, etc.

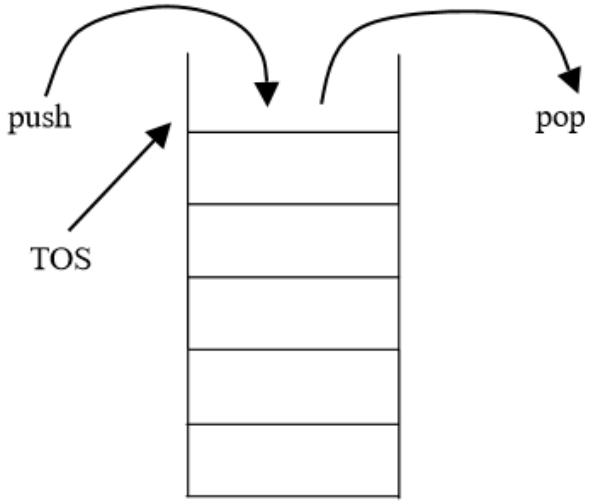
- Si dicono liste non lineari quelle liste che contengono fra i loro elementi altre liste:
  - G (B, A, F, (2, 5,7), G, (G,T))

- La lista G contiene alcuni elementi singoli ma anche altre liste

- Alberi e Grafi sono liste NON lineari

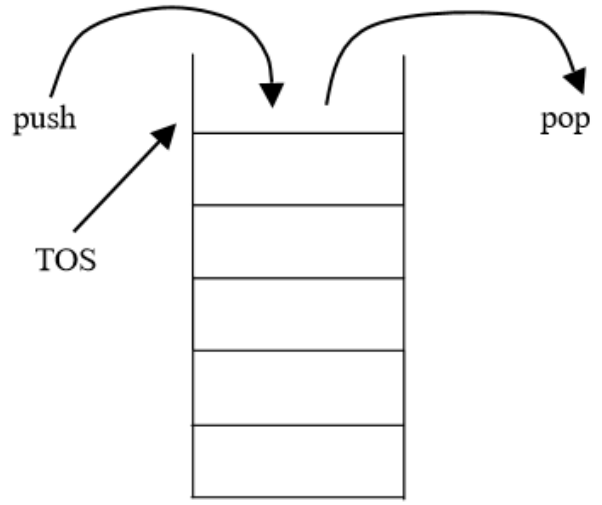
La pila (o stack) (1)

- La pila (o stack) è una struttura lineare dove le operazioni di inserimento ed estrazione dei dati avvengono sempre dalla stessa parte
- La pila è una struttura informativa di tipo:
  - **FILO (First Input Last Output)**, se il primo a entrare è l'ultimo ad uscire
  - **LIFO** (**Last Input First Output**) se l'ultimo a entrare è anche il primo a uscire.
- Esempio: tubetto di pasticche delle vitamine: per prendere l'ultima pasticca (ovvero la prima che è stata inserita) è necessario togliere tutte le pasticche precedenti



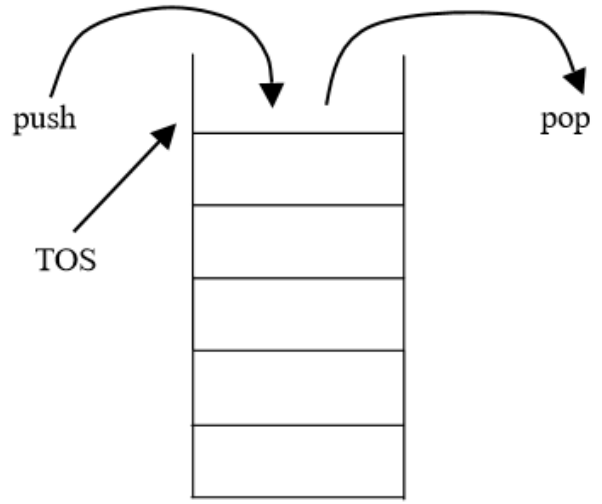
La pila (o stack) (2)

- L'operazione di inserimento è detta PUSH
- L'operazione di estrazione è detta POP
- Per la gestione di una pila è necessario conoscere solo il punto in cui viene fatta l'inserzione o l'estrazione, detto Top Of the Stack, TOS



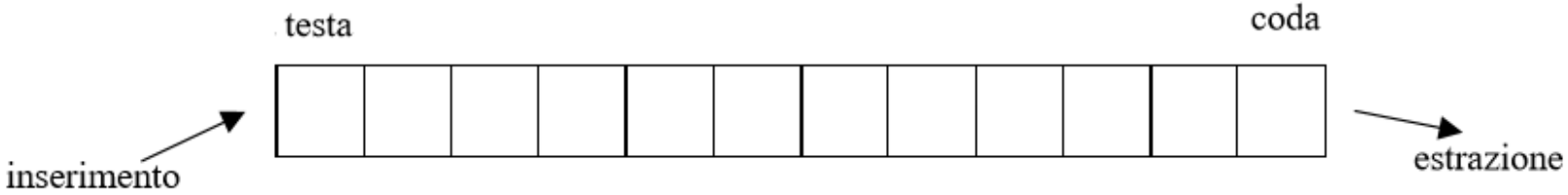
La pila (o stack) (3)

- Lo **stack** è tipicamente una **struttura dinamica** perché le sue dimensioni possono non essere note al momento della sua realizzazione
- Le operazioni sulla pila fanno tutte riferimento al TOS (Top Of the Stack), quindi è necessario monitorare tale elemento, in pratica è necessario tenere sotto controllo quando lo stack si svuota
- Se NON si effettua tale controllo, potrebbe infatti accadere di effettuare una operazione di POP (estrazione) senza che vi sia un elemento nello stack
- Le operazioni di base sono pertanto:
  - Creazione della pila, inserimento (push), estrazione (pop), verifica di stack vuoto PRIMA di fare ogni pop



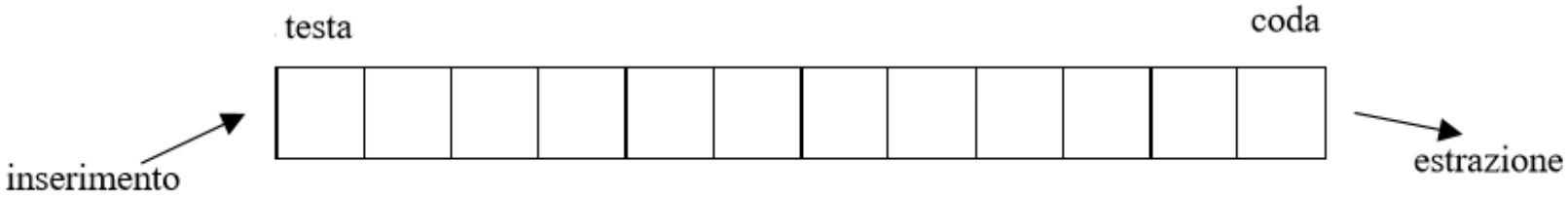
La coda (o buffer) (1)

- La **coda** (o buffer) è una struttura lineare dove le operazioni di inserimento vengono effettuate da una parte, mentre quelle di estrazione da un'altra
- Tipicamente la parte in cui si fanno le estrazioni viene chiamata **testa** mentre quella in cui si effettuano gli inserimenti viene chiamata **coda** (si dice infatti 'metti in coda')
- La coda è tipicamente una struttura dinamica, ovvero le sue dimensioni possono non essere note al momento della sua realizzazione
- La creazione della coda si effettua inserendo il primo elemento, mentre la coda si svuota togliendo l'ultimo elemento rimasto
- Le operazioni di base sono:
  - Creazione, inserimento, estrazione, verifica di coda vuota, svuotamento



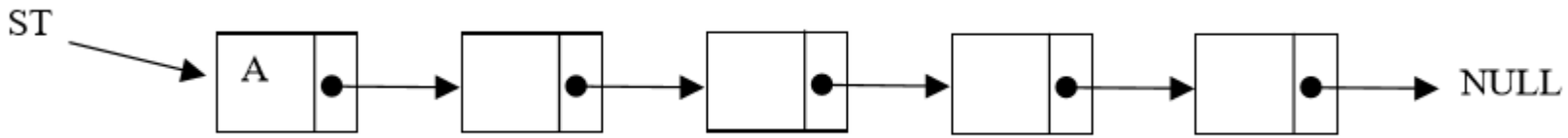
La coda (o buffer) (2)

- Il meccanismo di gestione della coda si basa sulla politica:
  - FIFO (First Input First Output), il primo ad entrare è il primo ad 'essere servito' cioè ad uscire dalla coda
- Lo stesso meccanismo può essere espresso anche con la politica LILO (Last Input Last Output), l'ultimo ad entrare è anche l'ultimo ad uscire
- Le operazioni di base per una coda sono:
  - creazione, inserimento, estrazione, verifica di coda vuota, svuotamento



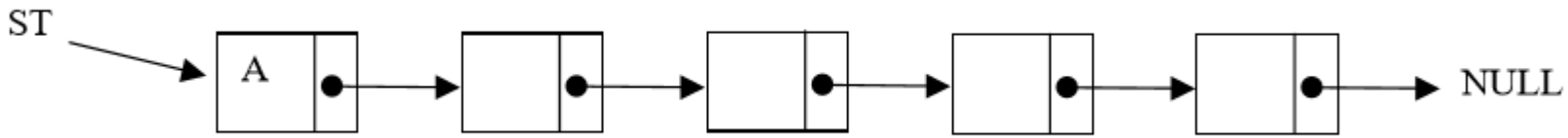
Lista concatenata semplice, catena semplice (1)

- La **lista concatenata** o catena semplice è una struttura interna ordinata per la quale elementi consecutivi NON sono disposti in memorie fisiche in celle consecutive
- La sequenza, e quindi l'ordine degli elementi in una lista, viene data in base ai legami che vengono definiti tra gli elementi stessi della lista
- I singoli elementi della lista possono essere dei record
- Ogni record contiene: un contenuto informativo (Non obbligatorio), un campo chiave che distingue un record dagli altri e un campo che contiene il riferimento all'elemento successivo



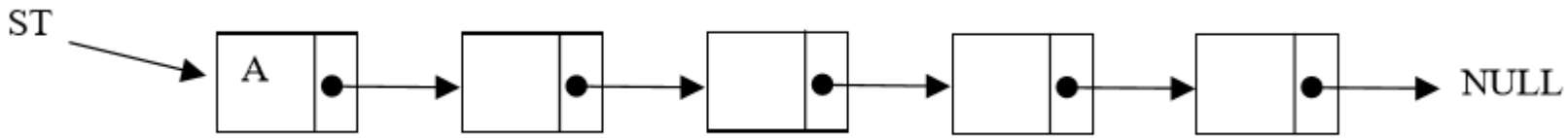
Lista concatenata semplice, catena semplice (2)

- Questa struttura dati è molto più flessibile rispetto al vettore, poiché:
  - è possibile inserire elementi anche all'interno della lista (senza dover riorganizzare la lista) e non solo alle estremità come nella coda
- Ogni elemento della lista contiene una parte dati e una parte per referenziare l'elemento successivo (**Puntatore**)
- Se si conosce un elemento (es: A in figura), si può fare riferimento all'elemento successivo nella lista (seguendo il puntatore)



Lista concatenata semplice, catena semplice (3)

- Se si conosce un elemento (es: A in figura), si può fare riferimento all'elemento successivo nella lista (seguendo il puntatore)
- Il primo elemento viene identificato da un puntatore (in figura ST)
- L'ultimo elemento non riferisce a nessun elemento, pertanto il suo puntatore viene impostato ad un valore prefissato e nullo, pari a NULL



Lista concatenata semplice, catena semplice (4)

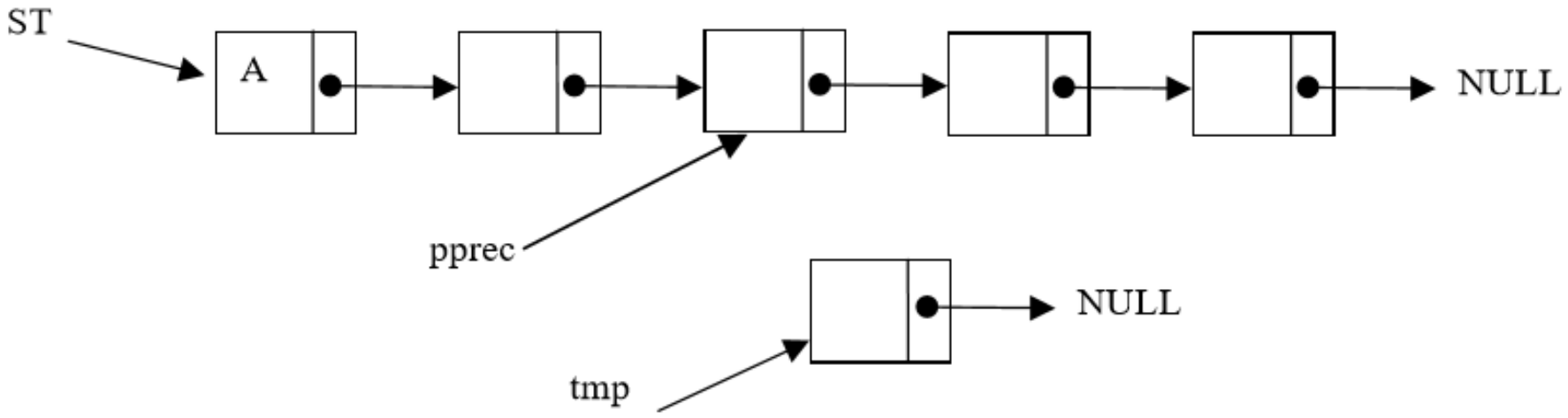
- La **lista** è una struttura dati sequenziale, pertanto non è possibile raggiungere il singolo elemento se non si conosce il suo INDIRIZZO di memoria tramite un PUNTATORE
- Confronto con il vettore:
  - La lista necessita di un maggior numero di byte di memoria per la presenza dei puntatori
  - La lista è una struttura dati più flessibile poichè le seguenti operazioni sono di più semplice realizzazione:
    - Cancellazione: cancellaz. di un elemento e aggiornamento del puntatore
    - Inserimento: inserimento di un nuovo elemento riorganizzando i puntatori degli elementi in cui il nuovo è inserito

Inserimento in lista concatenata semplice (1)

- Le operazioni di inserimento in testa e in coda alla lista sono semplici
- L'inserimento di un elemento lungo la lista è invece più complesso, è necessario:
  - Interrompere la catena
  - Aggiungere un ulteriore elemento
  - Ristabilire i legami
  - Fare attenzione a non perdere parte della lista durante il procedimento
- Si procede in tre fasi

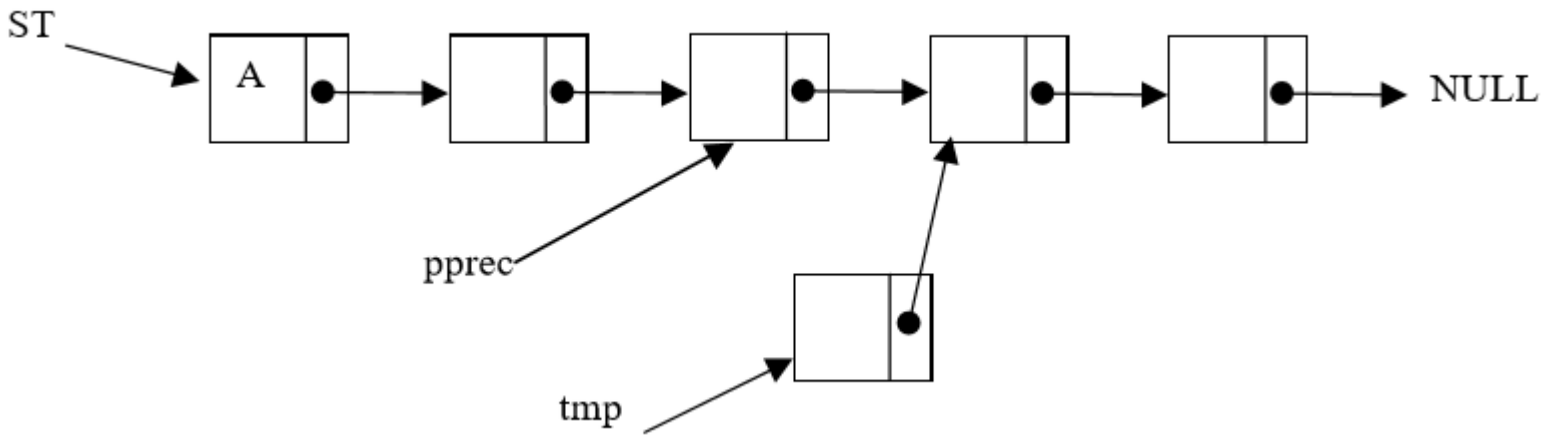
Inserimento in lista concatenata semplice – fase1

- Supponiamo di avere una lista e un nuovo elemento da inserire al suo interno (il uovo elemento è una lista di un solo elemento, puntata da tmp)
- Si deve avere un riferimento (puntatore) all'elemento successivamente al quale si vuole effettuare l'inserimento (pprec in figura)



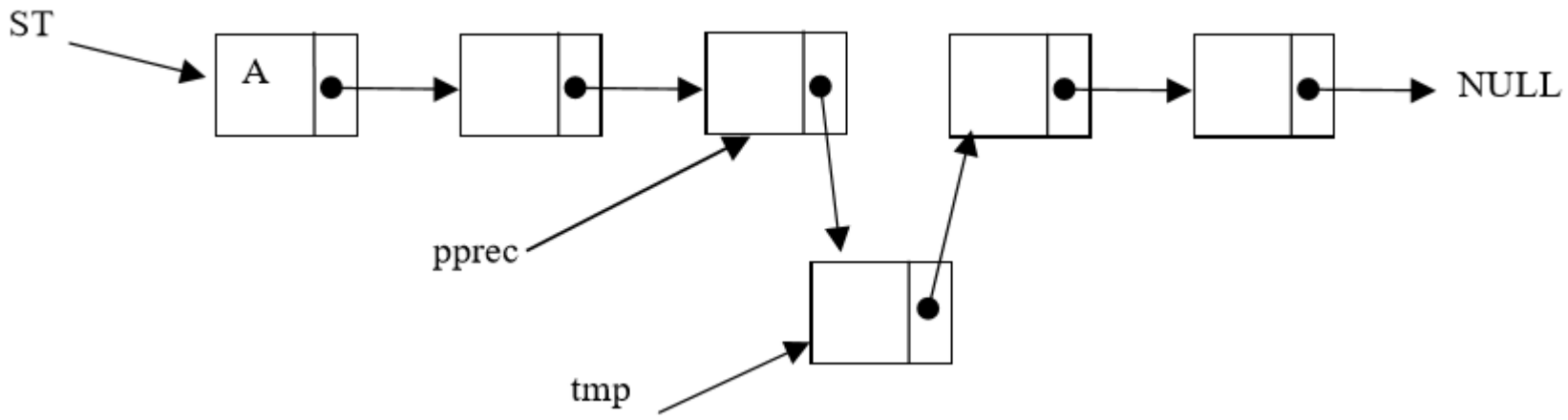
Inserimento in lista concatenata semplice – fase2

- L'elemento da inserire viene congiunto alla seconda parte della lista



Inserimento in lista concatenata semplice – fase3

- L'elemento precedente viene congiunto all'elemento da inserire assegnando il valore di tmp a valore del puntatore al successivo di pprec
- A questo punto 'operazione è conclusa e i due puntatori pprec e tmp non sono più necessari



Tornando alle strutture concrete…

Riprendiamo il concetto di stringa… (1)

- La stringa è un insieme ordinato di simboli estratti da un certo alfabeto
- La stringa ha simboli ordinati, nel senso che è possibile riferirsi ai singoli simboli indicandoli per mezzo della loro posizione nella stringa a partire dal primo simbolo all'ultimo secondo un criterio di ordinamento
- Nel caso degli elaboratori, l'alfabeto è quello dei caratteri rappresentabili ovvero quelli codificati tramite la tabella ASCII
- Esempi:
  - monica
  - sdfliufhwebla
  - 214324nps9
  - :{(/<#ib6)HAA*

Riprendiamo il concetto di stringa… (2)

- Il problema di rappresentare la scrittura nei calcolatori risale alle origini dell'informatica. La soluzione adottata più frequentemente, è la definizione di una *codifica di carattere* che consiste nell'associare ad ogni simbolo dell'alfabeto in uso una specifica sequenza di bit
- Un tipico esempio di *codifica*, risalente al 1967 e ancora in uso in ambiente informatico, è lo standard *ASCII – American Standard Code for Information Interchange*
- Il set di caratteri della codifica ASCII comprende tutte le lettere dell'alfabeto inglese, maiuscole e minuscole, le dieci cifre decimali, i simboli di interpunzione e a altri simboli grafici, alcuni caratteri di controllo (e.s. a-capo)
- 'altro aspetto che caratterizza una codifica è l'unità di codifica, ovvero l'unità atomica utilizzata per costruire le rappresentazioni dei caratteri; nel caso dell'ASCII consiste di un pacchetto di 7 bit. Poiché l'ASCII attribuisce ad ogni carattere un'unica unità di codifica, e che ad ogni unità atomica è associato un carattere distinto, l'alfabeto ASCII è composto esattamente da 128 caratteri
- Le varianti sono talmente numerose che i 128 byte della tabella estesa non sono purtroppo sufficienti a rappresentarle tutte, per questo motivo esistono diverse estensioni della tabella ASCII
- La tabella ASCII estesa tipicamente utilizzata in Italia è quella dell'Europa occidentale, creata per le lingue germaniche e neolatine (escluso il rumeno)

Riprendiamo il concetto di stringa… (3)

- Per cercare di ovviare al problema è stato creato un nuovo standard internazionale detto Unicode, definito dalla Unicode Consortium e dalla International Organization for Standardization (ISO 10646), che rappresenta i caratteri usando 2 byte (16 bit). Con 2 byte il numero di combinazioni possibili diventa 256x256 = 65.536, perciò Unicode supporta 65.536 diversi segni, al posto dei 256 del set ASCII.
- In python le stringhe sono sequenze di caratteri UNICODE

Tabella ASCII estesa

Dec	Sym	Dec	Char	Dec	Char	Dec	Char
0	NUL	32		64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	□

Dec	Char	Dec	Char	Dec	Char	Dec	Char
128	Ç	160	á	192	+	224	Ó
129	ü	161	í	193	-	225	ß
130	é	162	ó	194	-	226	Ô
131	â	163	ú	195	+	227	Ò
132	ä	164	ñ	196	-	228	õ
133	à	165	Ñ	197	+	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	+	232	Þ
137	ë	169	®	201	+	233	Ú
138	è	170	¬	202	-	234	Û
139	ï	171	½	203	-	235	Ù
140	î	172	¼	204	¦	236	ý
141	ì	173	¡	205	-	237	Ý
142	Ä	174	«	206	+	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	_	208	¶	240	Û
145	æ	177	_	209	Ð	241	±
146	Æ	178	_	210	Ê	242	_
147	ô	179	¦	211	Ë	243	¾
148	ö	180	¦	212	È	244	¶
149	ò	181	Á	213	i	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	¦	217	+	249	¨
154	Ü	186	¦	218	+	250	·
155	ø	187	+	219	_	251	¹
156	£	188	+	220	_	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	_
159	ƒ	191	+	223	_	255	

Riprendiamo il concetto di stringa… (4)

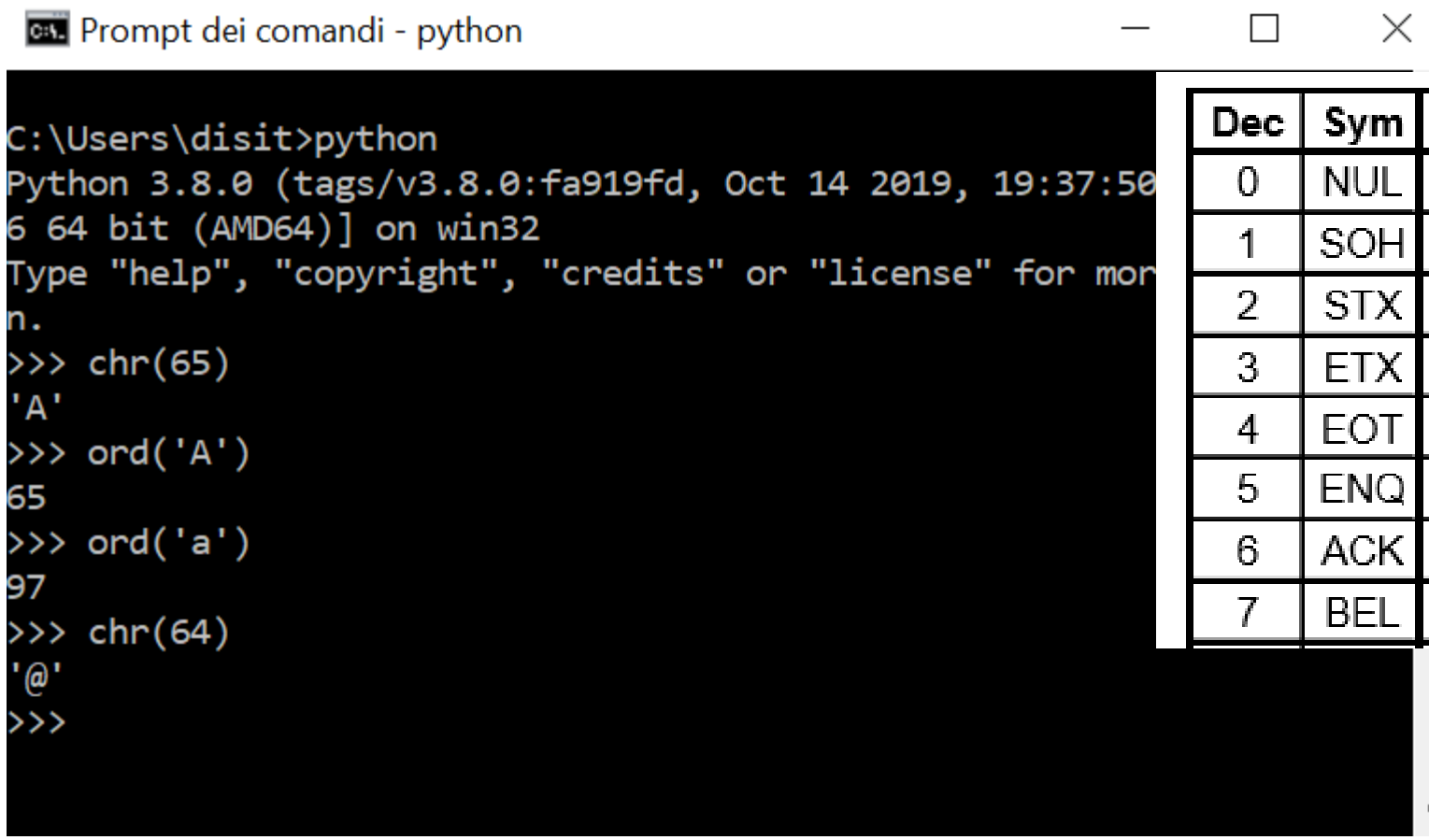
- Le tabelle dei codici Unicode sono disponibili sul sito http://www.unicode.org/charts.
- I primi caratteri di Unicode sono esattamente gli stessi della codifica ASCII, in modo da mantenere la compatibilità con il sistema preesistente
- In python le stringhe sono sequenze di caratteri UNICODE
- Cosa significa? Come possiamo verifcare?
- Esistono due funzioni: chr() e ord() che eseguono rispettivamente la conversione da numero a carattere e da carattere a numero basandosi esattamente sulla tabella UNICODE
- Facciamo degli esempi..
- Cosa ci si aspetta con i seguenti comandi?

chr(65)

ord('A')

Dec	Sym	Dec	Char	Dec	Char	Dec	Char
0	NUL	32		64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g

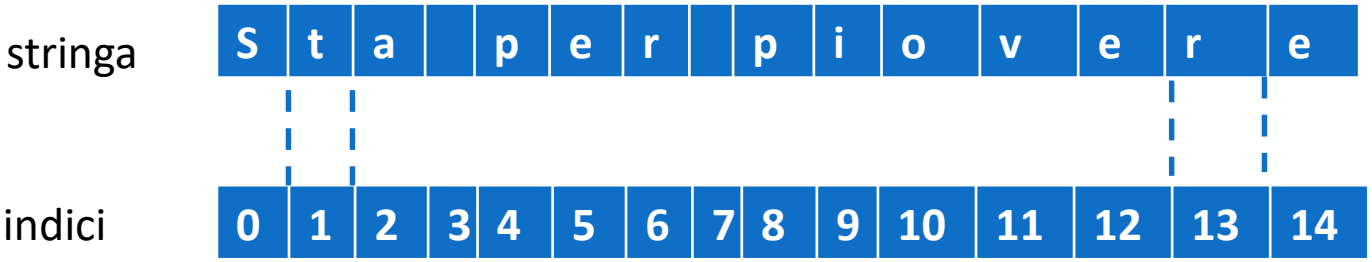
Riprendiamo il concetto di stringa… (5)



Dec	Sym	Dec	Char	Dec	Char	Dec	Char
0	NUL	32		64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g

Riprendiamo il concetto di stringa… (5)

- Le Stringhe sono caratteri UNICODE
- E' possibile accedere ai singoli caratteri contenuti in una stringa mediante la loro posizione al suo interno: tale posizione viene detta *indice* del carattere
- Il primo carattere ha indice 0, il secondo ha indice 1 e così via
- Si accede al singolo carattere con una speciale notazione a indice, racchiudendo la sua posizione tra parentesi quadre
- Se ad esempio si definisce:

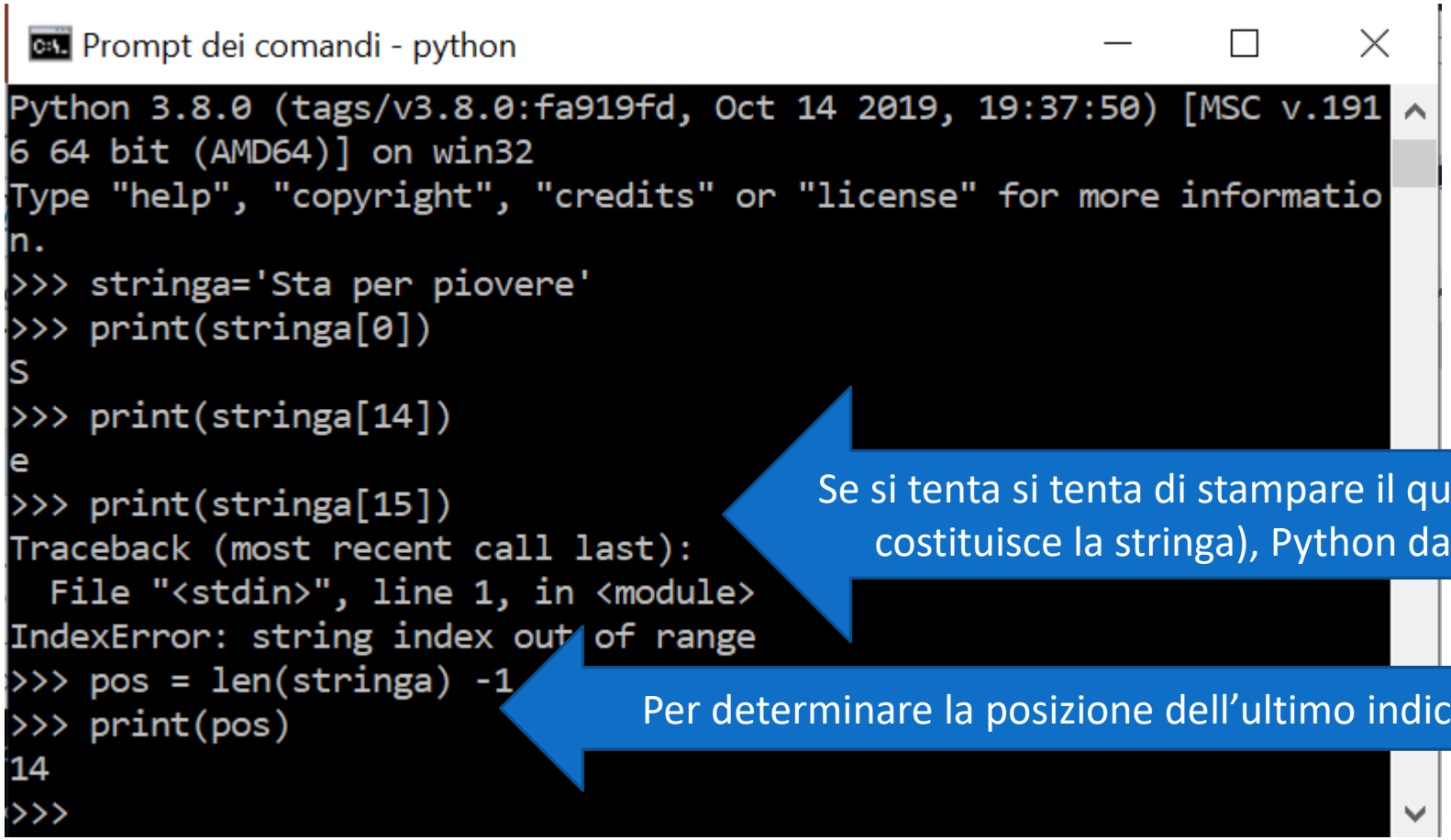


stringa = 'Sta per piovere'

Allora, gli enunciati seguenti stampano rispettivamente la prima e l'ultima lettera della stringa:

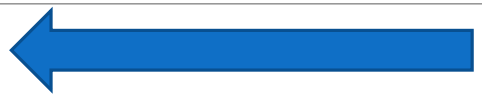
```
print(stringa[0])
print(stringa[14])
```

Riprendiamo il concetto di stringa… (6)



Python: Compilatori e Console

Esecuzione di programmi e ambienti: notebooks, IDE, console



- Il linguaggio python
  - tipi mutabili e immutabili
  - operatori ed espressioni
  - istruzioni
  - funzioni
  - cicli while e for
  - esecuzione condizionale
- Strutture dati e algoritmi elementari: Liste, Dizionari, Insiemi, iterazioni su strutture dati

Costo di esecuzione e complessità

Il modello di costo

Cenni sulla complessità di un algoritmo:

- Algoritmi di ordinamento su vettori
  - Sequential-sort
- Cenni sugli alberi
  - Alberi
  - Alberi binari di ricerca: i) Visita in forma ricorsiva; ii) Ricerca; iii) Inserimento ordinato

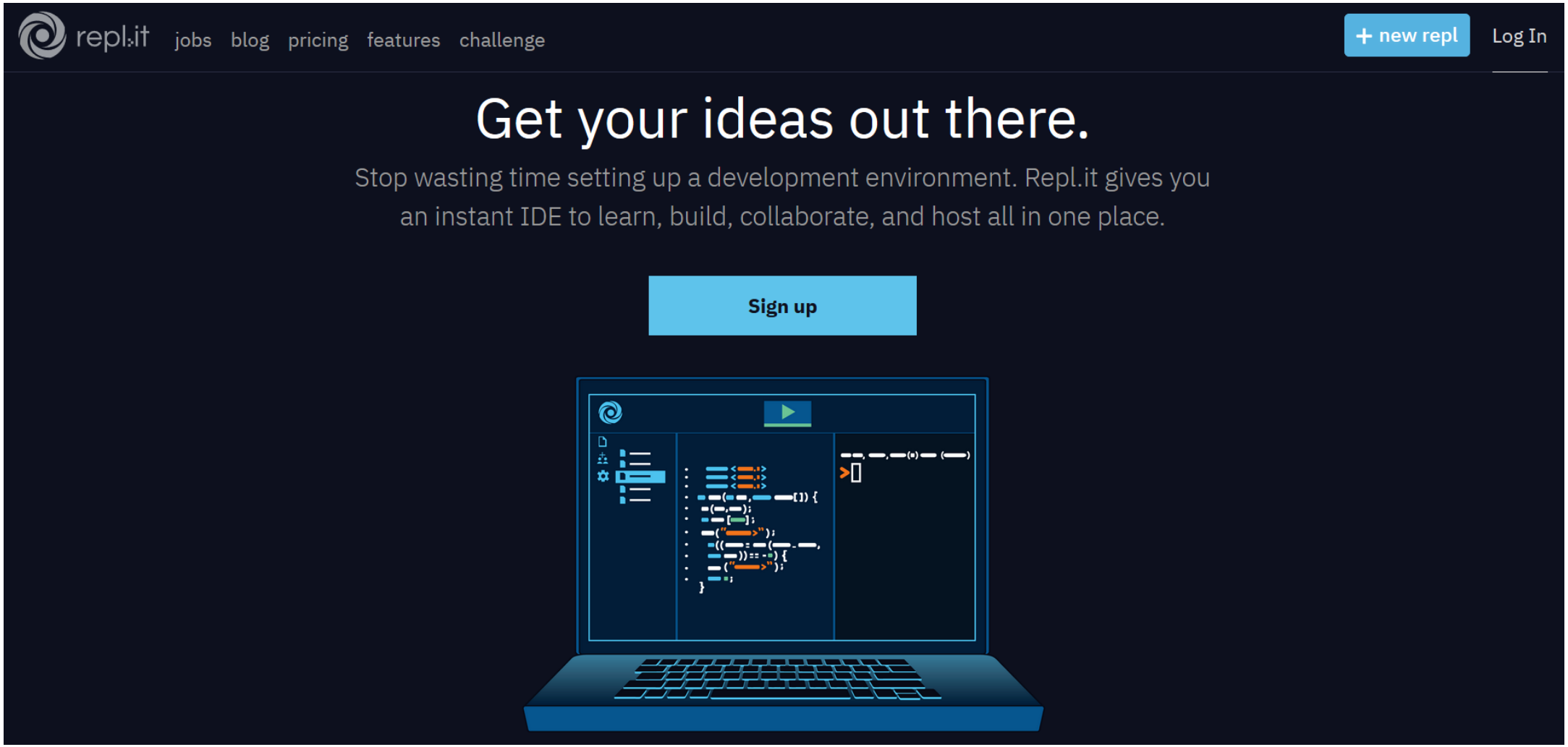
Cenni su analisi dei dati, lettura e scrittura di file in forma tabulare, grafici.

Python: Compilatori e Console

Compilatori

- Per far girare codice Python e scrivere programmi in Python, esistono varie possibilità
- All'inizio può essere utile ricorrere ai compilatori online, ce ne sono diversi ma ovviamente necessitano di una connessione
- Da riga di comando, è possibile scaricare e installare Python (useremo la versione 3.8) e poi aprire la Console
- Per i programmi più complessi si userà una IDE, in particolare vedremo Spyder

Compilatori online: Repl.it (1)

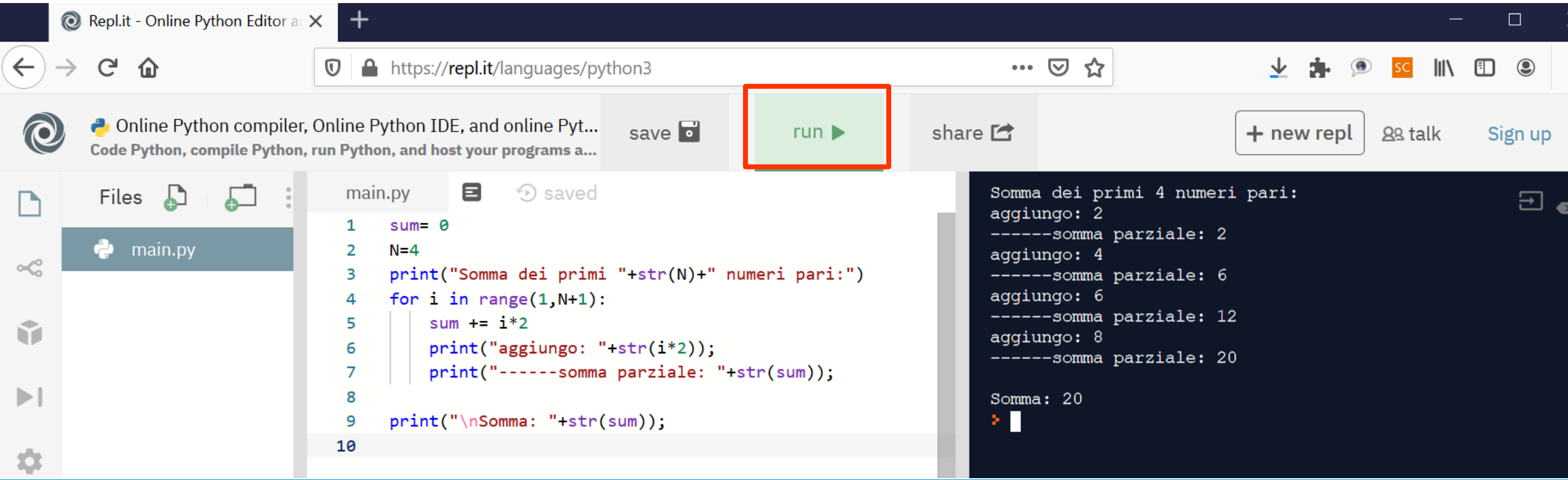


Compilatori online: Repl.it (2)

- https://repl.it/languages/python3
- nel riquadro centrale è possibile scrivere codice Python
- Cliccando sul bottone RUN, è possibile far girare il programma e vedere i risultati prodotti in Console (riquadro a destra)

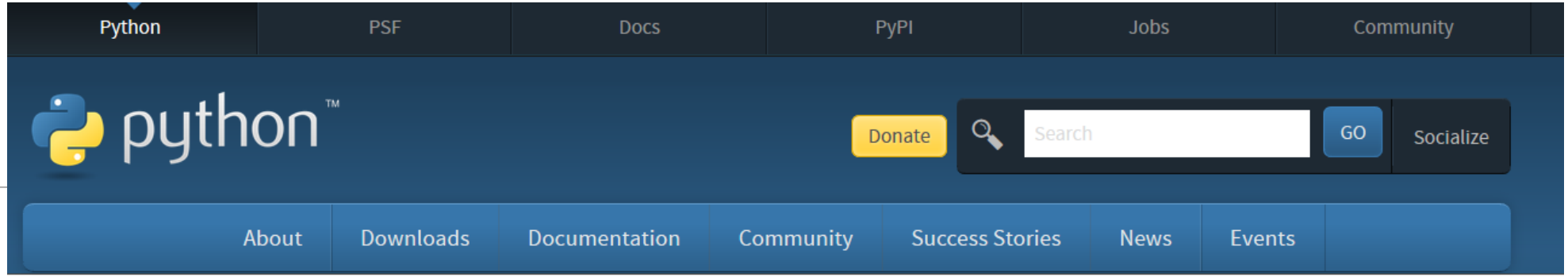
Compilatori online: Repl.it (2)

- https://repl.it/languages/python3
- nel riquadro centrale è possibile scrivere codice Python
- Cliccando sul bottone RUN, è possibile far girare il programma e vedere i risultati prodotti in Console (riquadro a destra)



Scaricare Python (versione 3.8) (1)

- https://www.python.it/download/



Python >>> Downloads >>> Windows

Python Releases for Windows

- Latest Python 3 Release - Python 3.8.0
- Latest Python 2 Release - Python 2.7.17

Stable Releases

- Python 2.7.17 - Oct. 19, 2019
  - Download Windows debug information files
  - Download Windows debug information files for 64-bit binaries
  - Download Windows help file
  - Download Windows x86-64 MSI installer
  - Download Windows x86 MSI installer
- Python 3.7.5 - Oct. 15, 2019

  Note that Python 3.7.5 *cannot* be used on Windows XP or earlier.

  - Download Windows help file
  - Download Windows x86-64 embeddable zip file

Pre-releases

- Python 3.5.8rc2 - Oct. 12, 2019
  - No files for this release.
- Python 2.7.17rc1 - Oct. 9, 2019
  - Download Windows debug information files
  - Download Windows debug information files for 64-bit binaries
  - Download Windows help file
  - Download Windows x86-64 MSI installer
  - Download Windows x86 MSI installer
- Python 3.7.5rc1 - Oct. 2, 2019
  - Download Windows help file
  - Download Windows x86-64 embeddable zip file

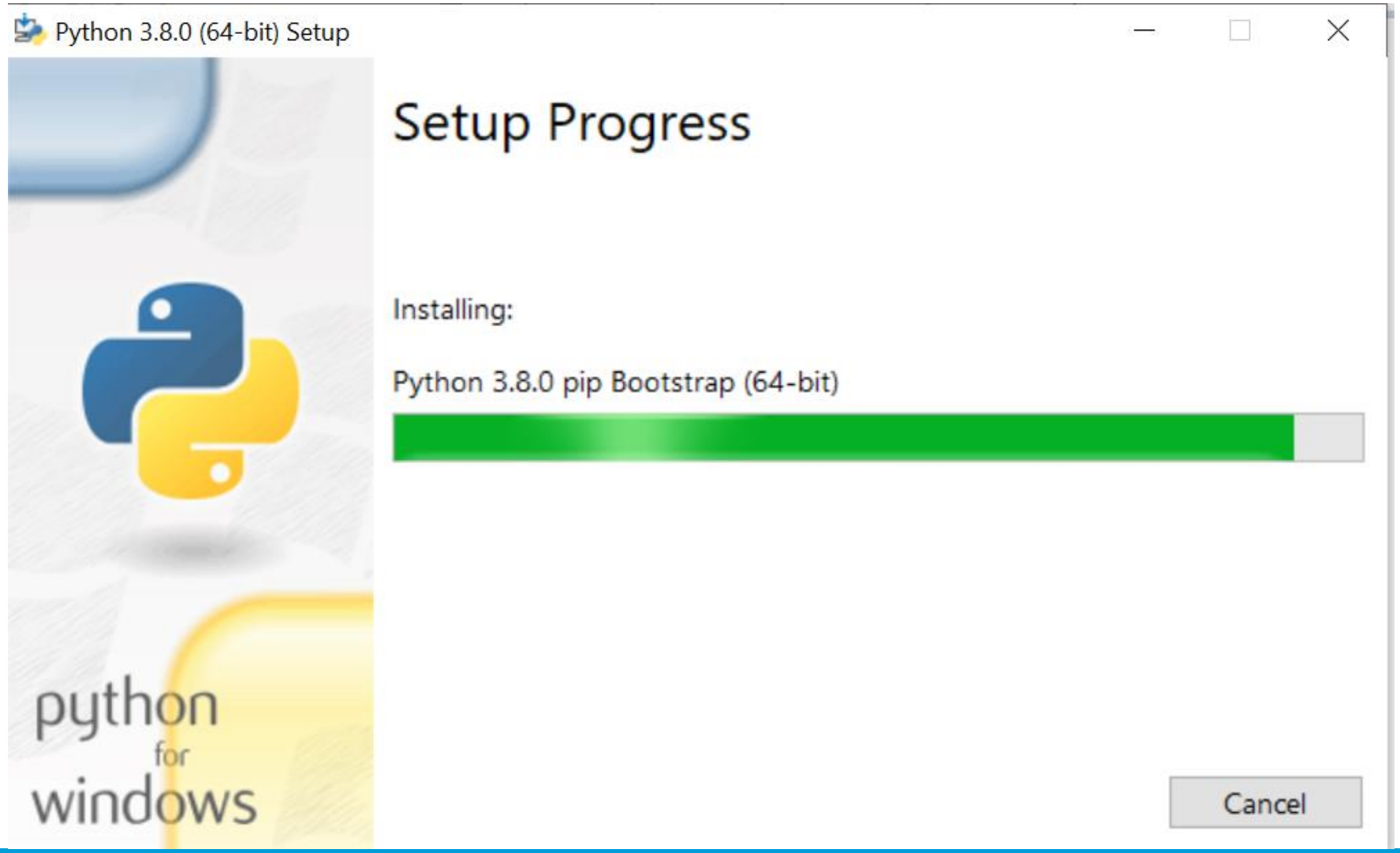
Scaricare Python (versione 3.8) (3)

Files

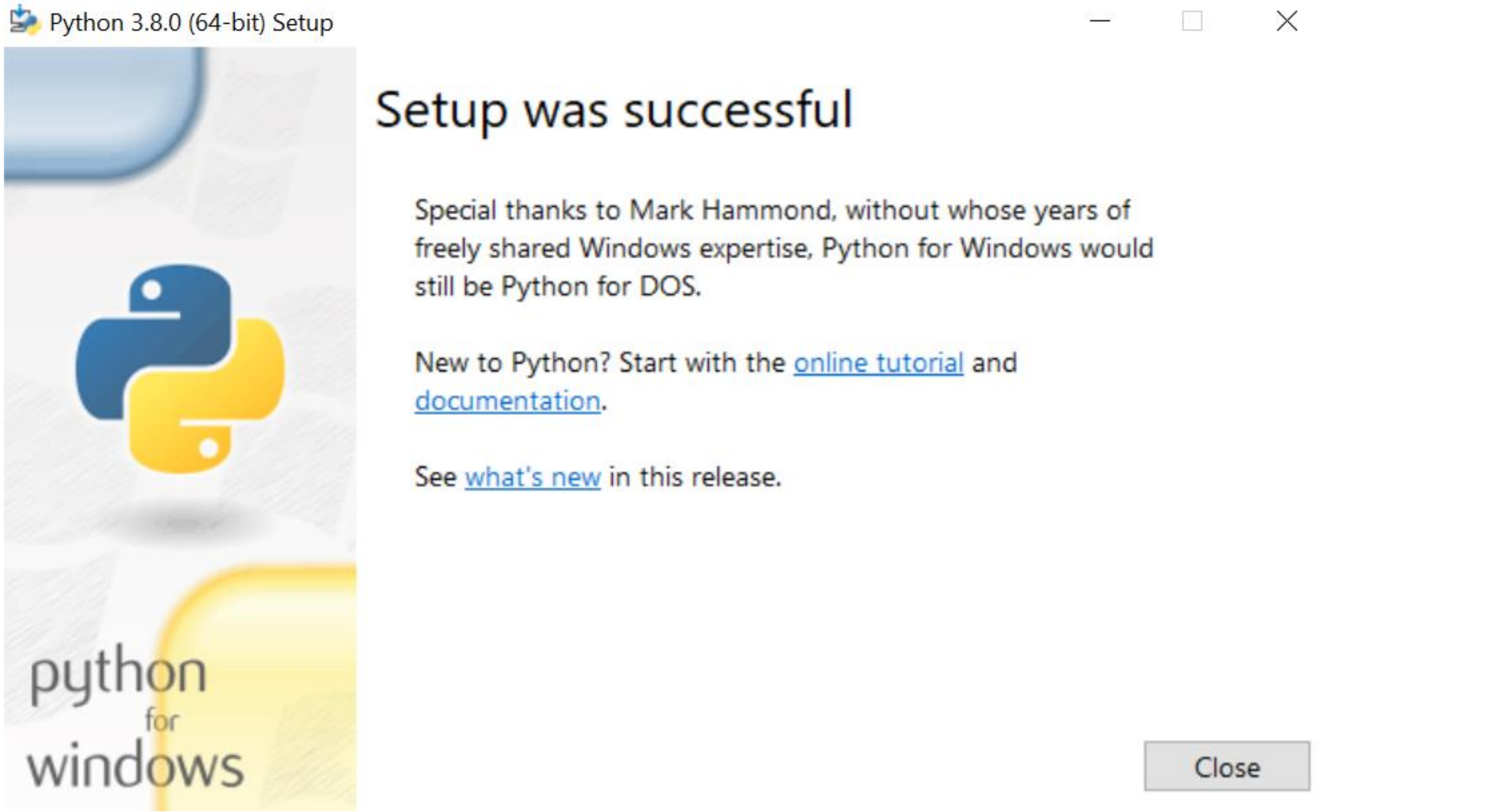
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		e18a9d1a0a6d858b9787e03fc6fdaa20	23949883	SIG
XZ compressed source tarball	Source release		dbac8df9d8b9edc678d0f4cacdb7dbb0	17829824	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	f5f9ae9f416170c6355cab7256bb75b5	29005746	SIG
Windows help file	Windows		1c33359821033ddb3353c8e5b6e7e003	8457529	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	99cca948512b53fb165084787143ef19	8084795	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	29ea87f24c32f5e924b7d63f8a08ee8d	27505064	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	f93f7ba8cd48066c59827752e531924b	1363336	SIG
Windows x86 embeddable zip file	Windows		2ec3abf05f3f1046e0dbd1ca5c74ce88	7213298	SIG
Windows x86 executable installer	Windows		412a649d36626d33b8ca5593cf18318c	26406312	SIG
Windows x86 web-based installer	Windows		50d484ff0b08722b3cf51f9305f49fdc	1325368	SIG

Scaricare Python (versione 3.8) (4)

Scaricare Python (versione 3.8) (5)

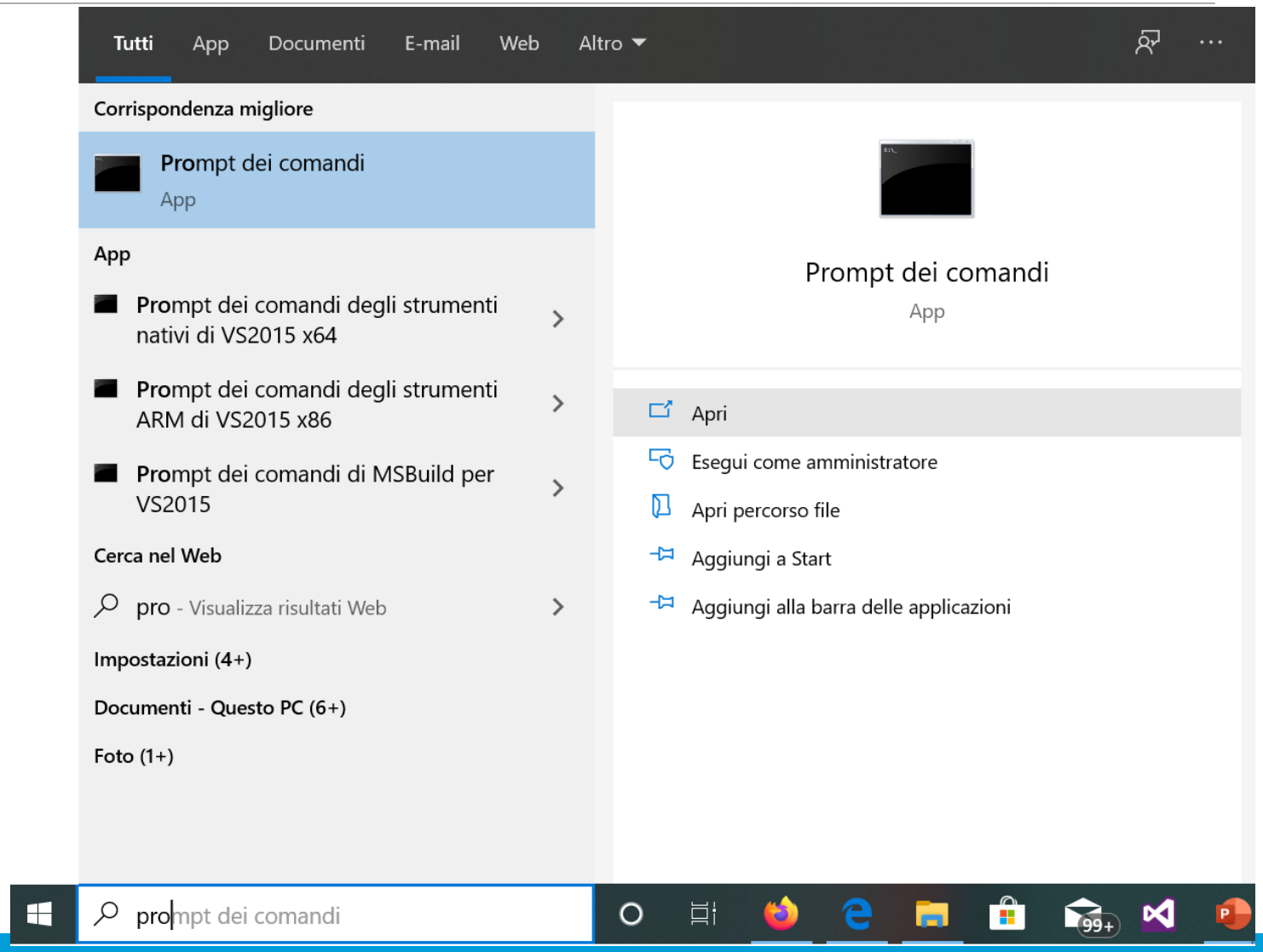


Scaricare Python (versione 3.8) (6)



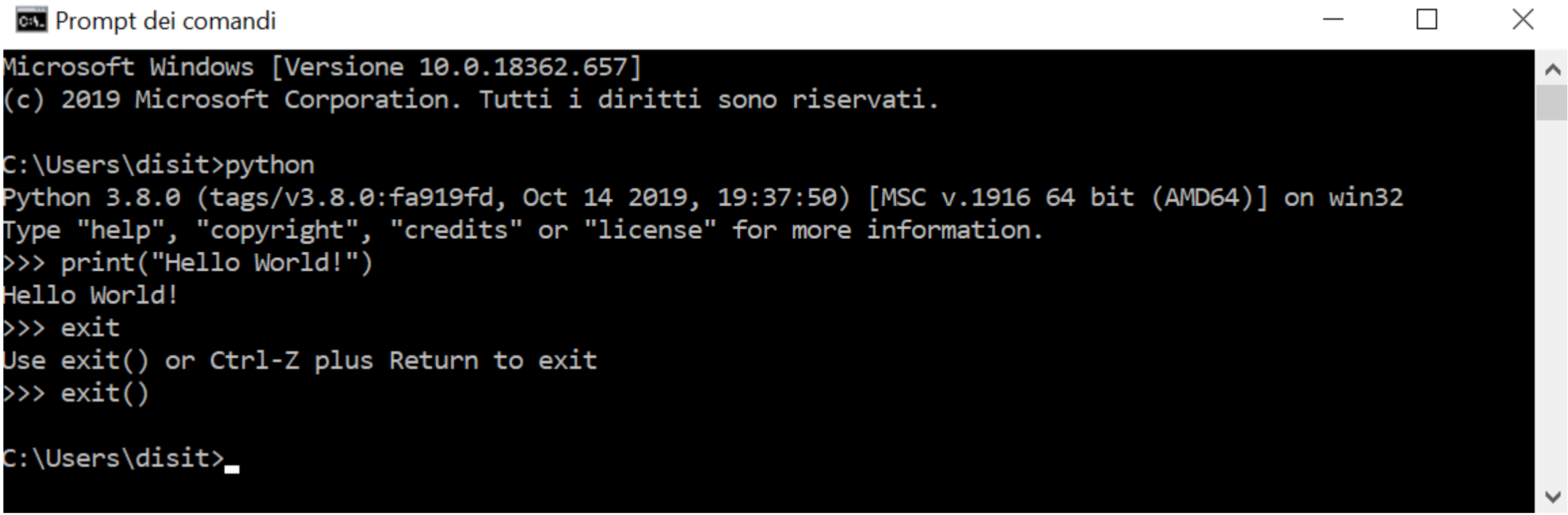
Lanciare Python (versione 3.8) (1)

- Aprire un promtp dei comandi
- Andare nella directory in cui è stato istallato python



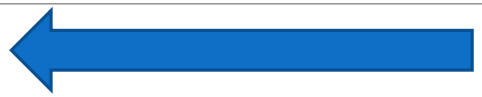
Scaricare Python (versione 3.8) (3)

- Aprire un promtp
- Per aprire la console Python, digitare: python
- Per chiudere la console Python, digitare: exit() Oppure la combinazione dei tasti è: Ctrl+Z



Esercitazioni …

Python: Compilatori e Console



Esecuzione di programmi e ambienti: notebooks, IDE, console

- Il linguaggio python
  - tipi mutabili e immutabili
  - operatori ed espressioni
  - istruzioni
  - funzioni
  - cicli while e for
  - esecuzione condizionale
- Strutture dati e algoritmi elementari: Liste, Dizionari, Insiemi, iterazioni su strutture dati

Costo di esecuzione e complessità

Il modello di costo

Cenni sulla complessità di un algoritmo:

- Algoritmi di ordinamento su vettori
  - Sequential-sort
- Cenni sugli alberi
  - Alberi
  - Alberi binari di ricerca: i) Visita in forma ricorsiva; ii) Ricerca; iii) Inserimento ordinato

Cenni su analisi dei dati, lettura e scrittura di file in forma tabulare, grafici.

Pyhton: Spyder

Spyder

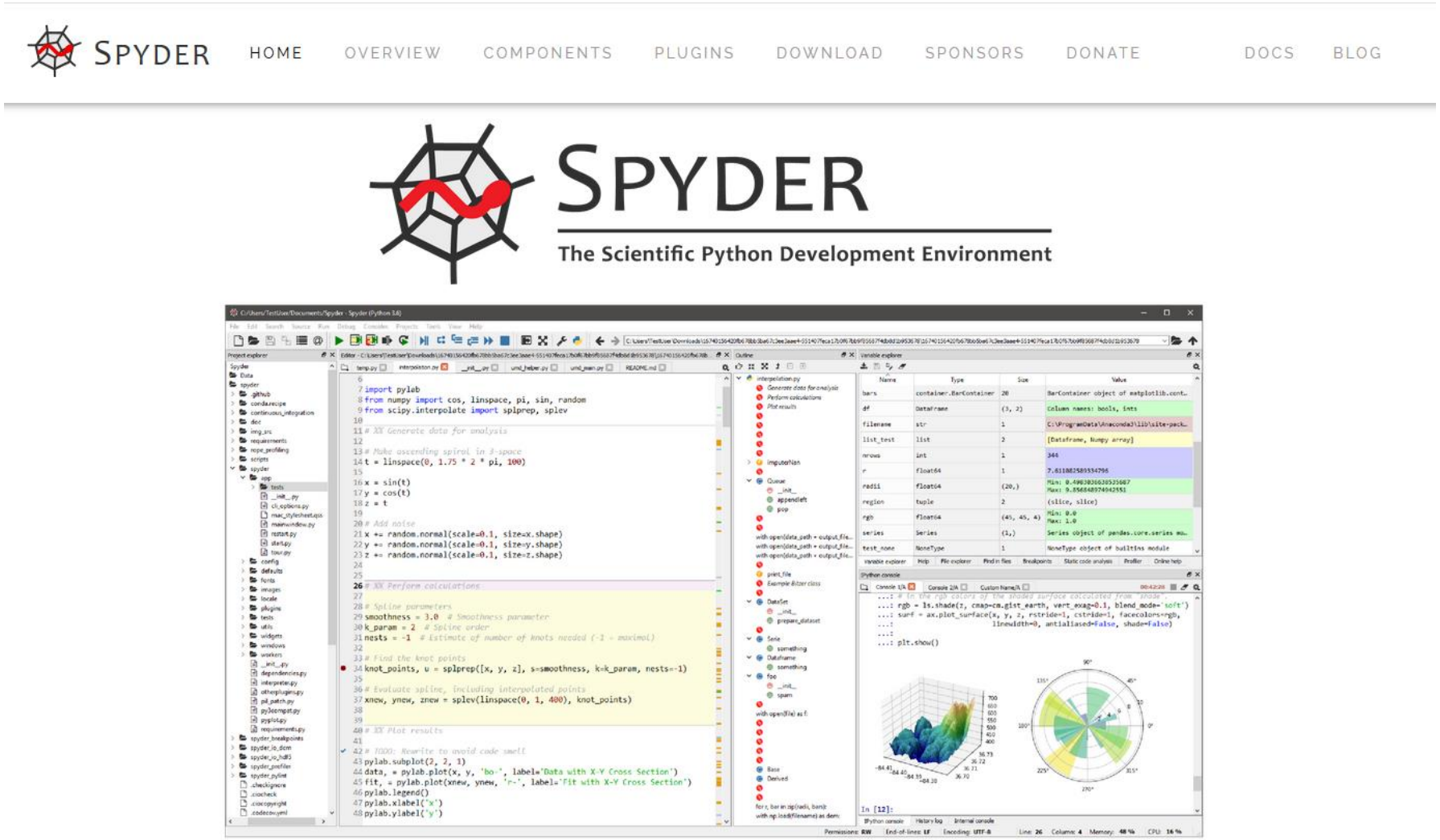
Open Source

https://www.spyder-ide.org/

Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.

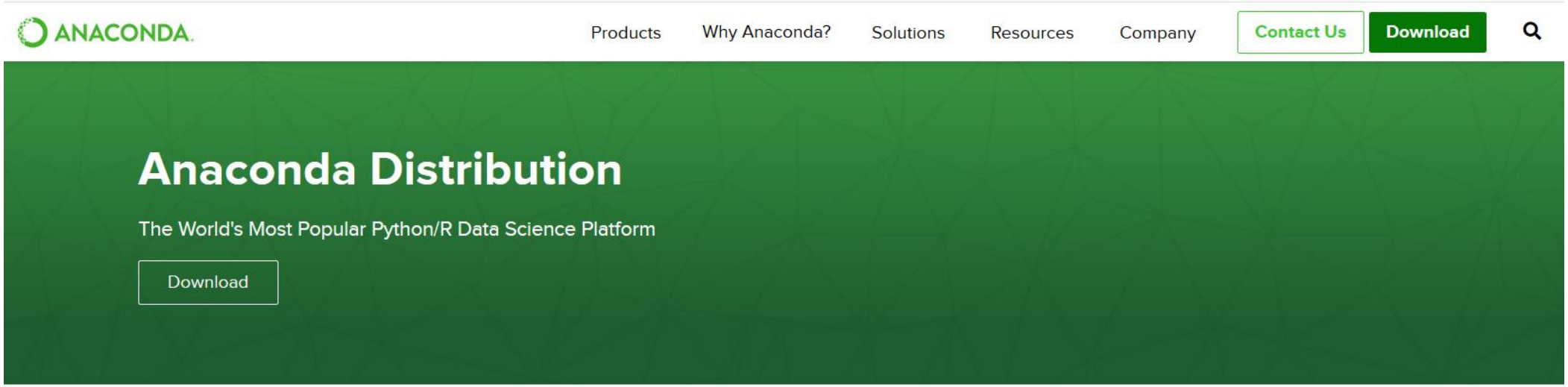
It offers a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

https://www.spyder-ide.org/



Dowload and install Spyder with Anaconda (1)

https://www.spyder-ide.org/

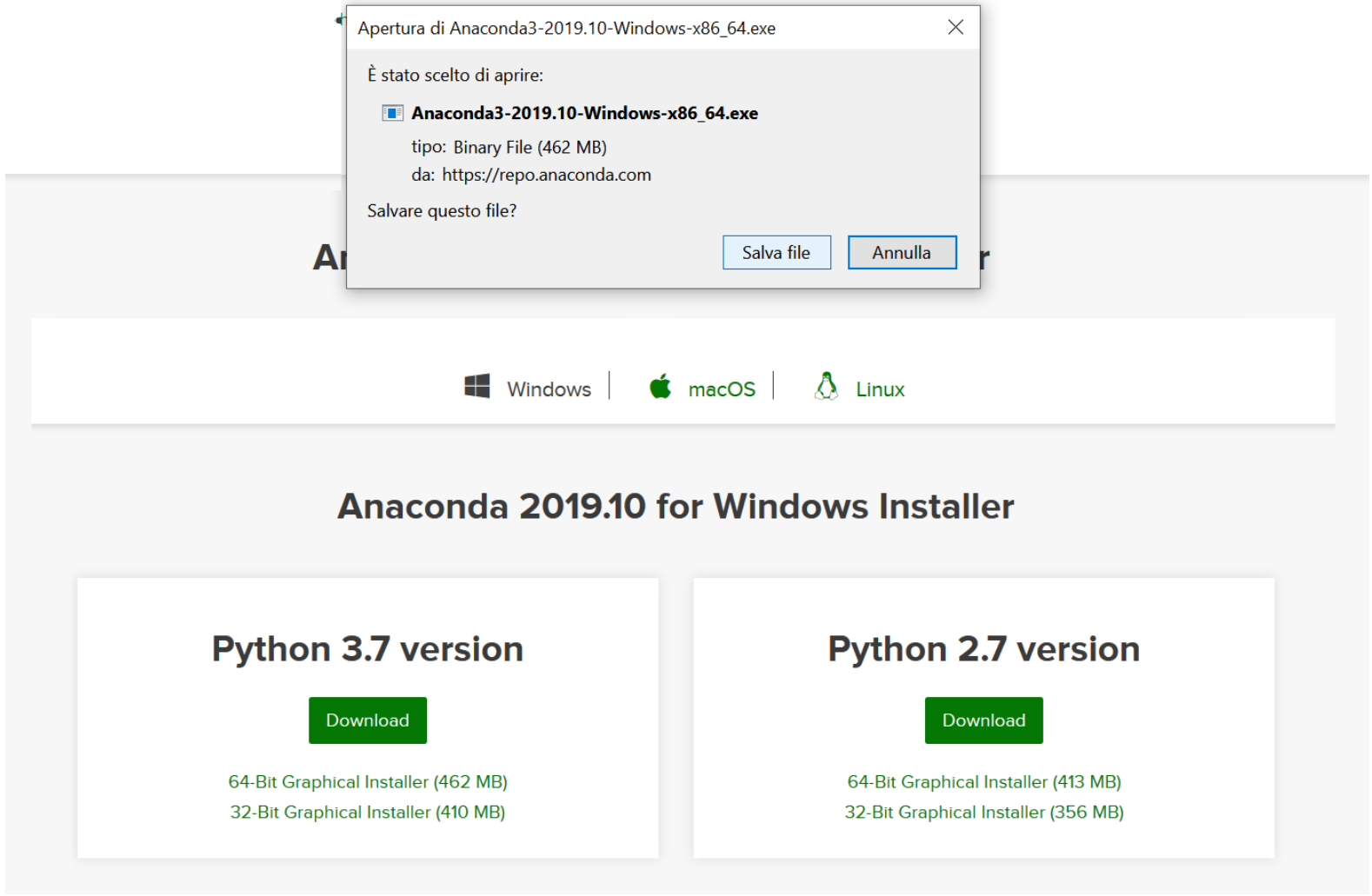


The open-source Anaconda Distribution is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 15 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:

- Quickly download 1,500+ Python/R data science packages
- Manage libraries, dependencies, and environments with Conda
- Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
- Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews

Dowload and install Spyder with Anaconda 2)

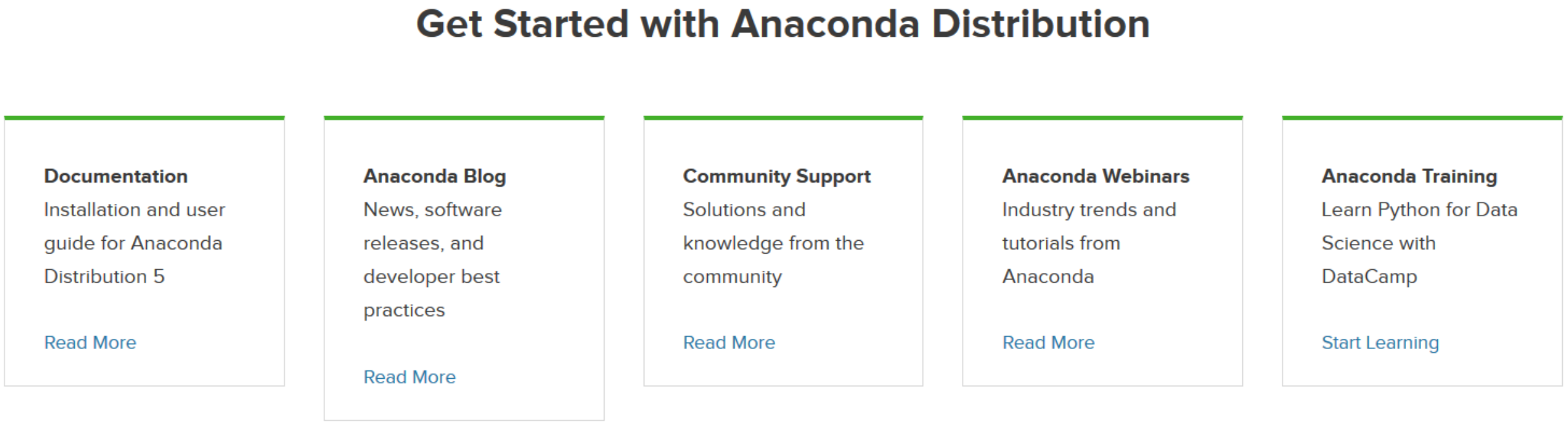
https://www.spyder-ide.org/



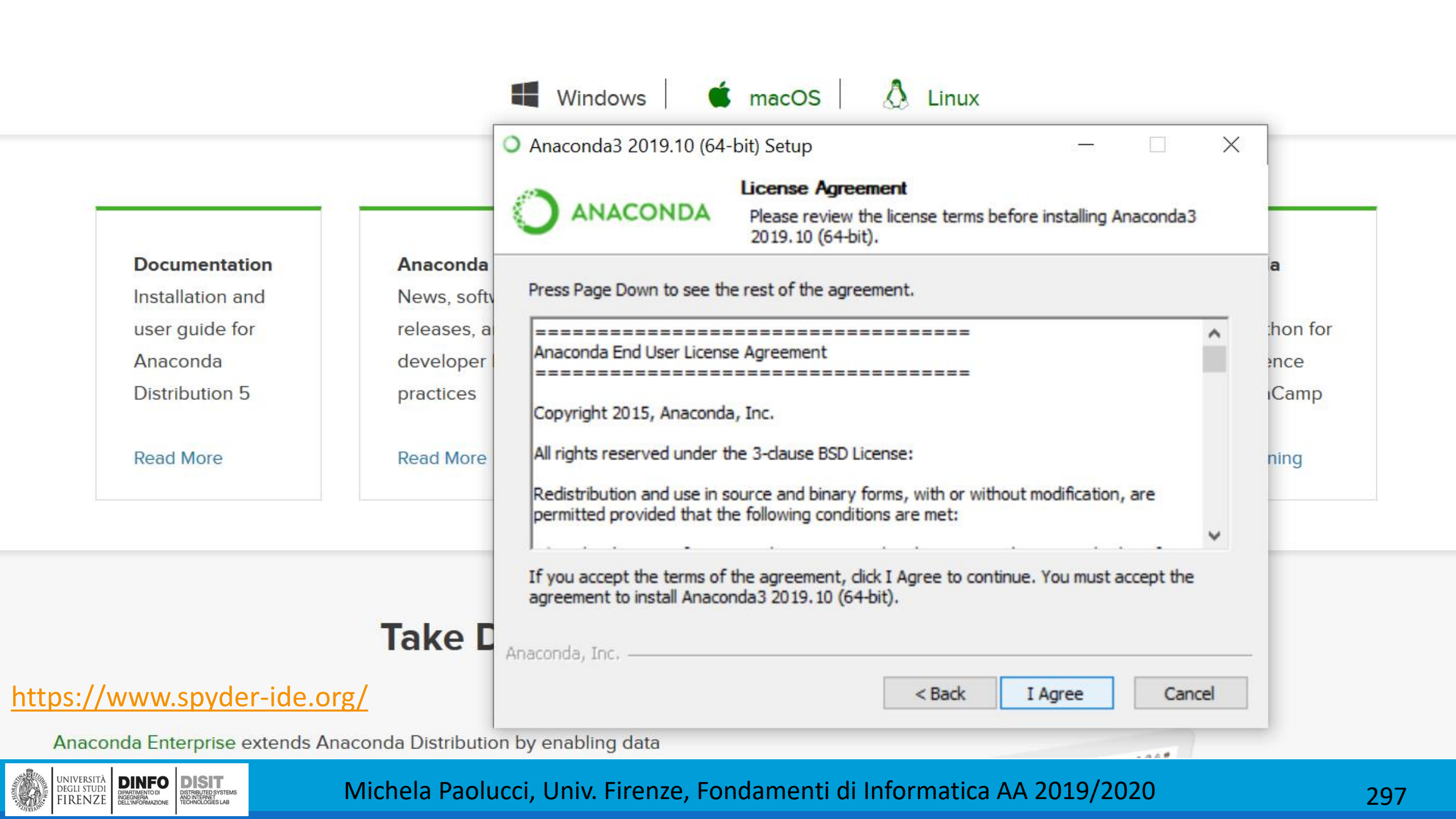
Download and install Spyder with Anaconda(3)

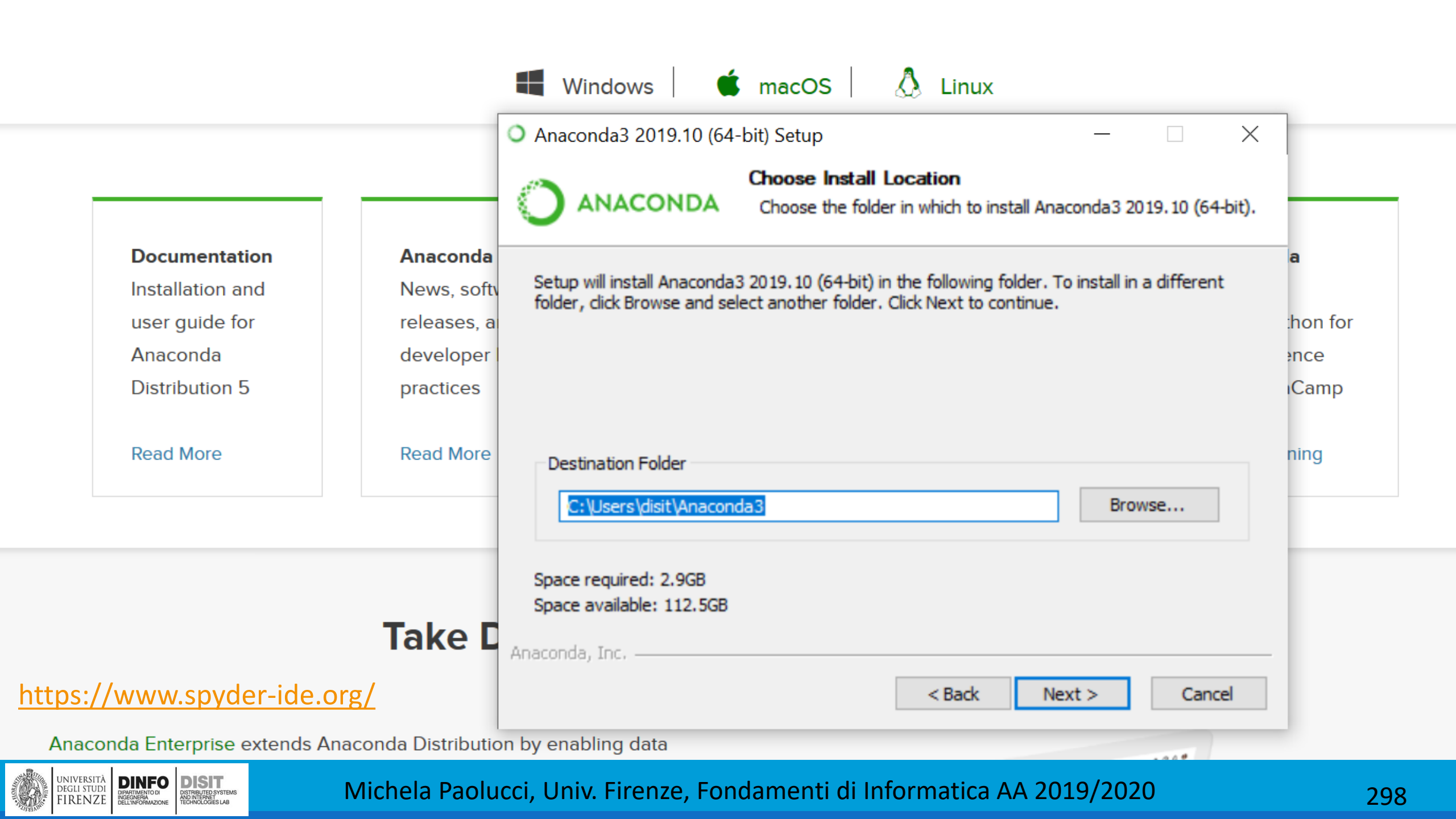
https://www.spyder-ide.org/

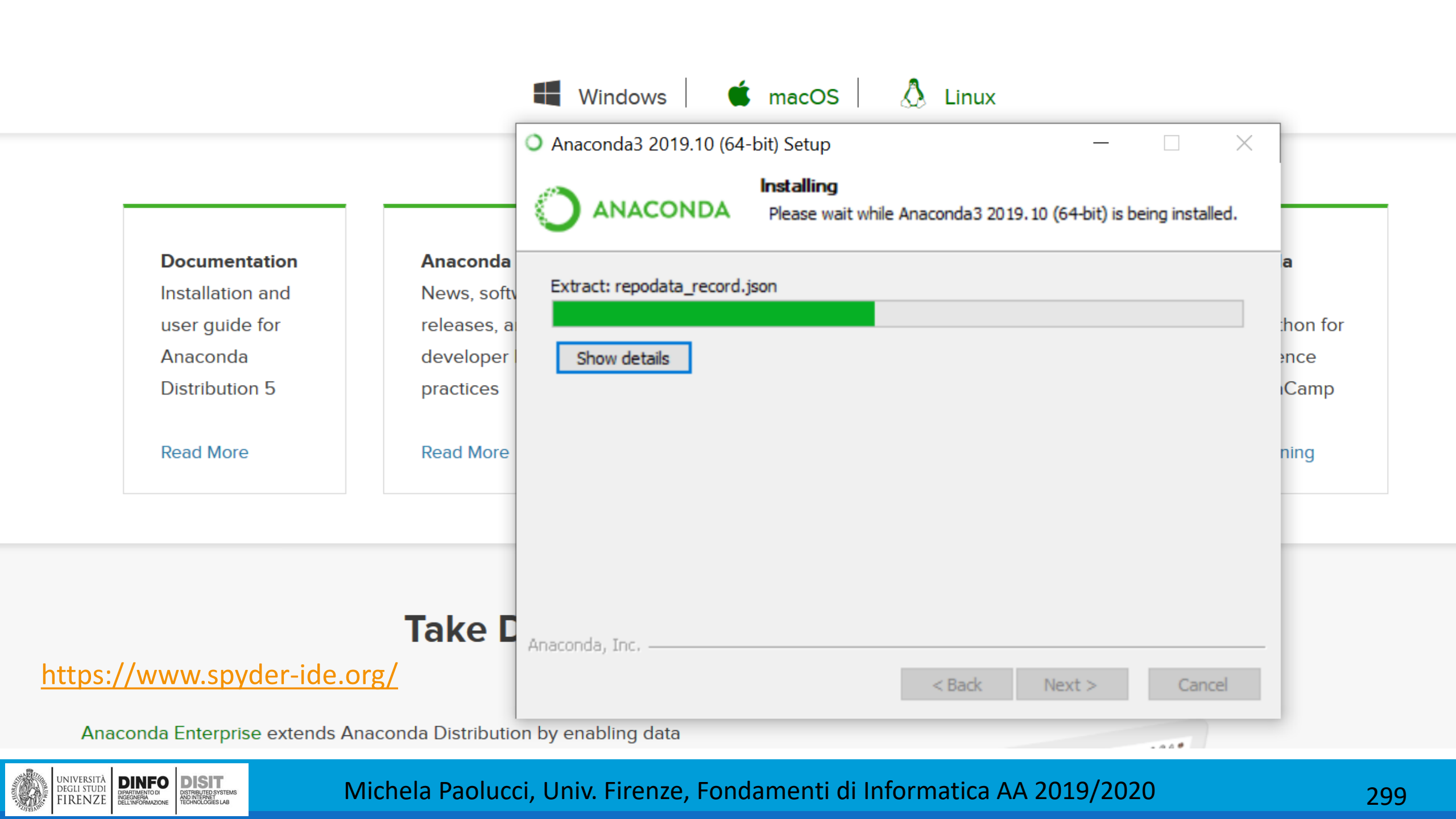
Guide




Windows
macOS
Linux
Anaconda3 2019.10 (64-bit) Setup
ANACONDA.
Welcome to Anaconda3 2019.10 (64-bit) Setup
Setup will guide you through the installation of Anaconda3 2019.10 (64-bit).
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel
Documentation
Installation and user guide for Anaconda Distribution 5
Read More
Anaconda
News, softv
releases, a
developer
practices
Read More
Take D
Anaconda Enterprise extends Anaconda Distribution by enabling data
https://www.spyder-ide.org/


Windows
macOS
Linux
Anaconda3 2019.10 (64-bit) Setup
ANACONDA
License Agreement
Please review the license terms before installing Anaconda3 2019.10 (64-bit).
Press Page Down to see the rest of the agreement.
Anaconda End User License Agreement
Copyright 2015, Anaconda, Inc.
All rights reserved under the 3-clause BSD License:
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2019.10 (64-bit).
Anaconda, Inc.
< Back
I Agree
Cancel
Documentation
Installation and user guide for Anaconda Distribution 5
Read More
Anaconda
News, softw releases, a developer practices
Read More
Take D
https://www.spyder-ide.org/
Anaconda Enterprise extends Anaconda Distribution by enabling data


Windows
macOS
Linux
Anaconda3 2019.10 (64-bit) Setup
ANACONDA
Choose Install Location
Choose the folder in which to install Anaconda3 2019.10 (64-bit).
Setup will install Anaconda3 2019.10 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Users\disit\Anaconda3
Browse...
Space required: 2.9GB
Space available: 112.5GB
Anaconda, Inc.
< Back
Next >
Cancel
Documentation
Installation and user guide for Anaconda Distribution 5
Read More
Anaconda
News, softv releases, a developer practices
Read More
Take D
https://www.spyder-ide.org/
Anaconda Enterprise extends Anaconda Distribution by enabling data


Windows
macOS
Linux
Anaconda3 2019.10 (64-bit) Setup
ANACONDA
Installing
Please wait while Anaconda3 2019.10 (64-bit) is being installed.
Extract: repodata_record.json
Show details
Anaconda, Inc.
< Back
Next >
Cancel
Documentation
Installation and
user guide for
Anaconda
Distribution 5
Read More
Anaconda
News, soft
releases, a
developer
practices
Read More
Take D
https://www.spyder-ide.org/
Anaconda Enterprise extends Anaconda Distribution by enabling data

Windows | macOS | Linux

Documentation
Installation and user guide for Anaconda Distribution 5

Read More

Anacon
News, s releases develop practice

Read Mo

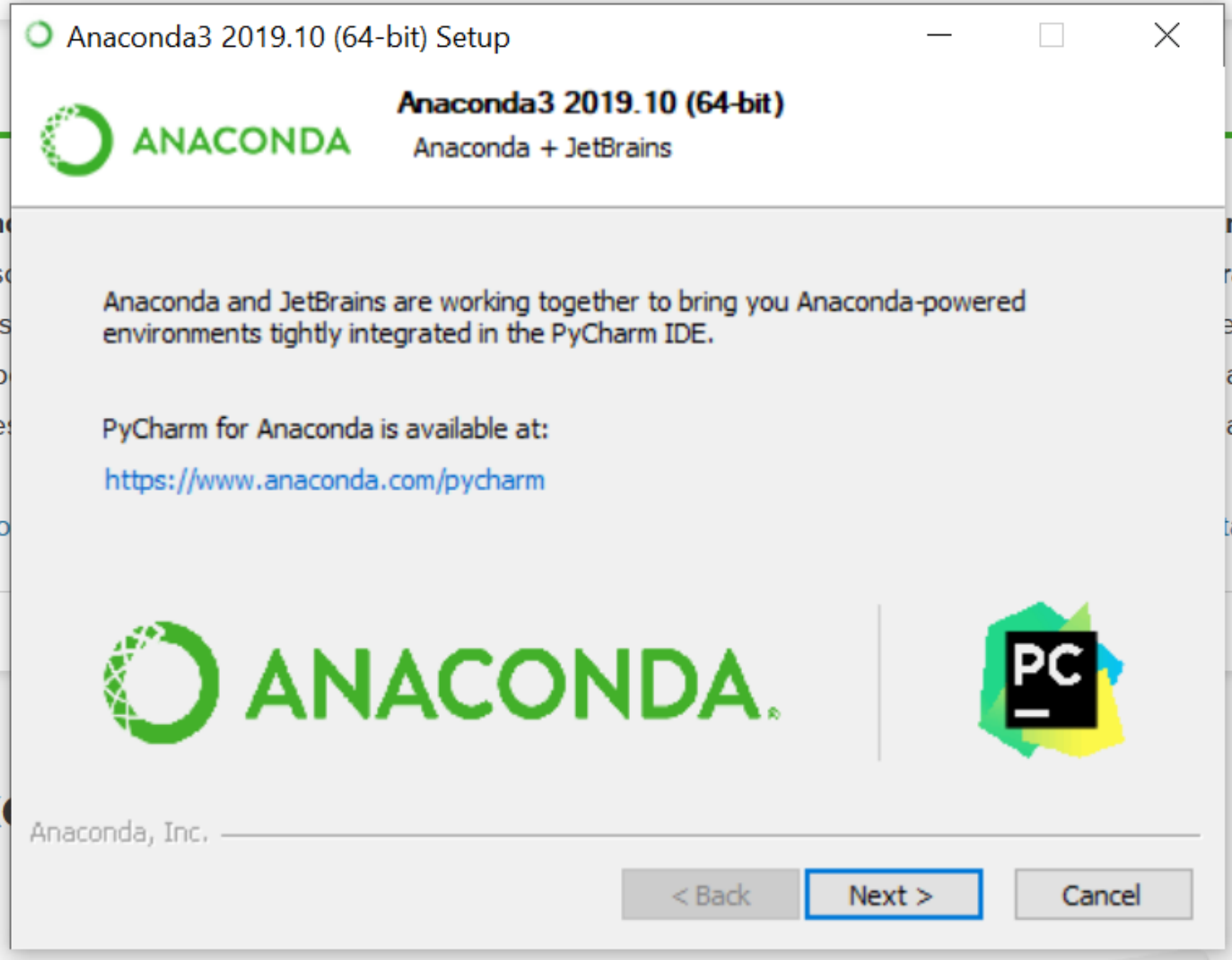
naconda
raining
earn Python for ata Science with ataCamp

tart Learning

Tak

https://www.spyder-ide.org/

Anaconda Enterprise extends Anaconda Distribution by enabling data science



Dowload and install Spyder with Anaconda

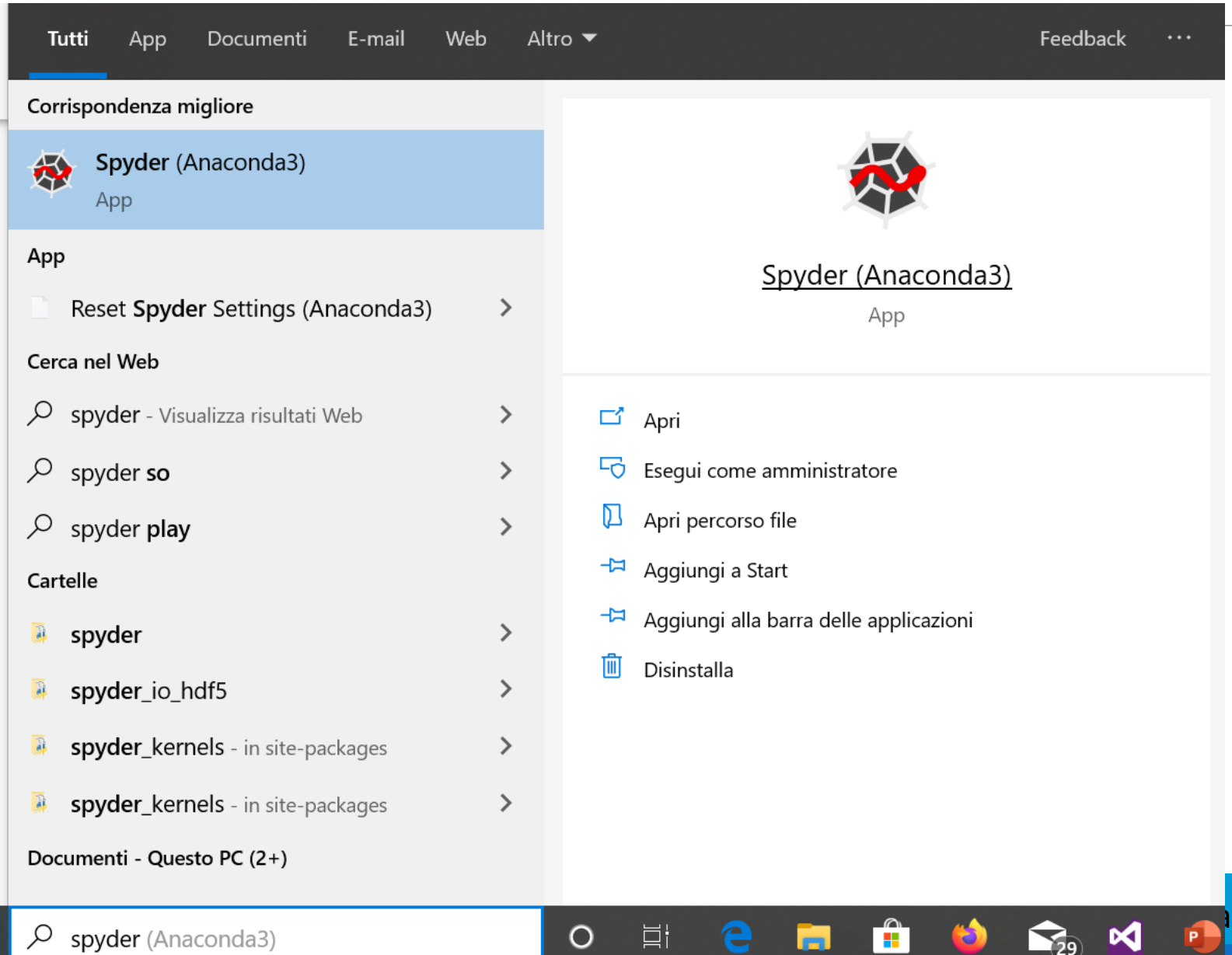
Getting start with Anaconda:

- https://docs.anaconda.com/anaconda/user-guide/getting-started/

Anaconda cloud;

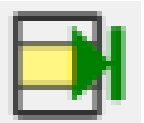
- https://anaconda.org/

Launch Spyder

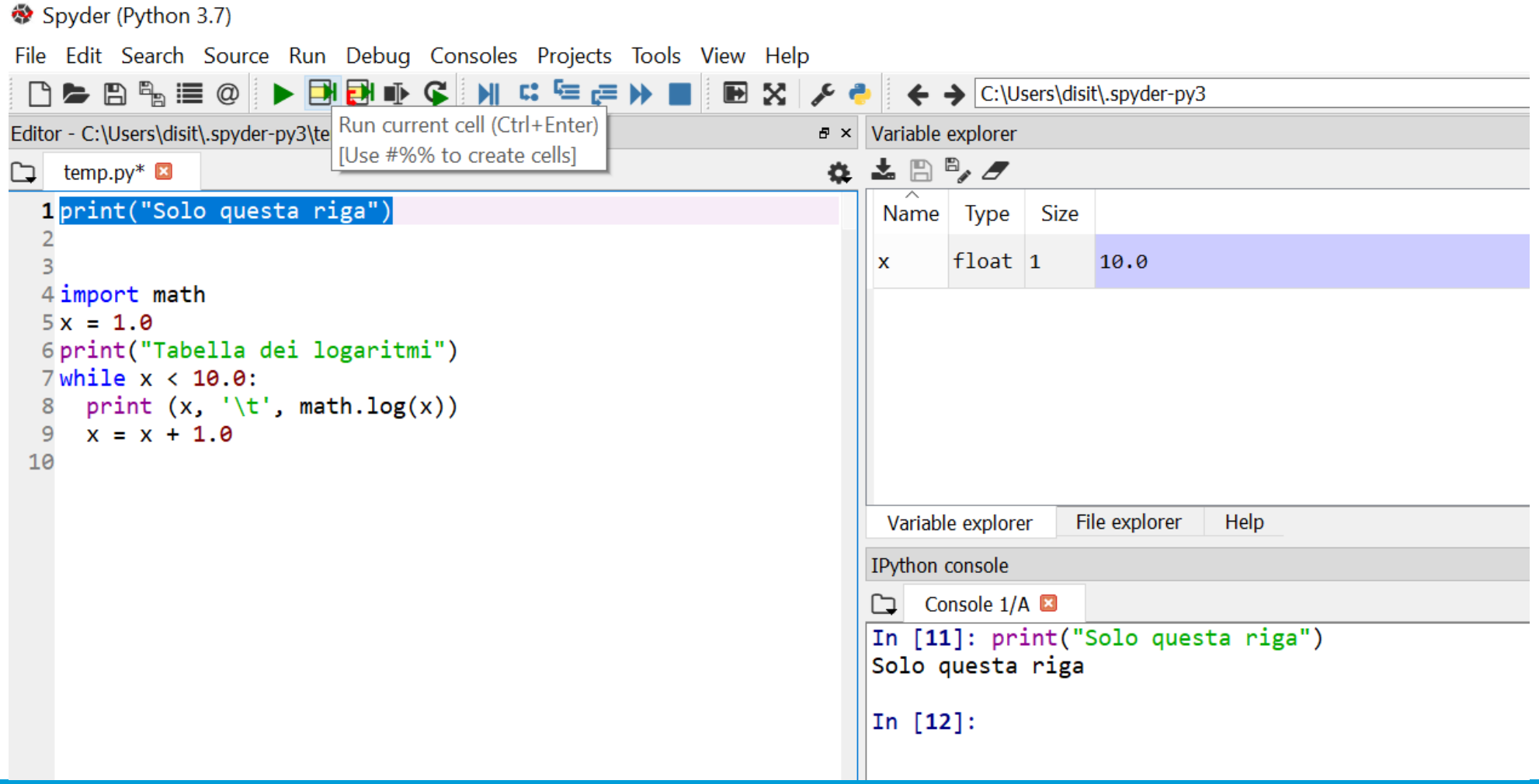


Run (standard): tutte righe di codice

Run: solo alcune righe di codice

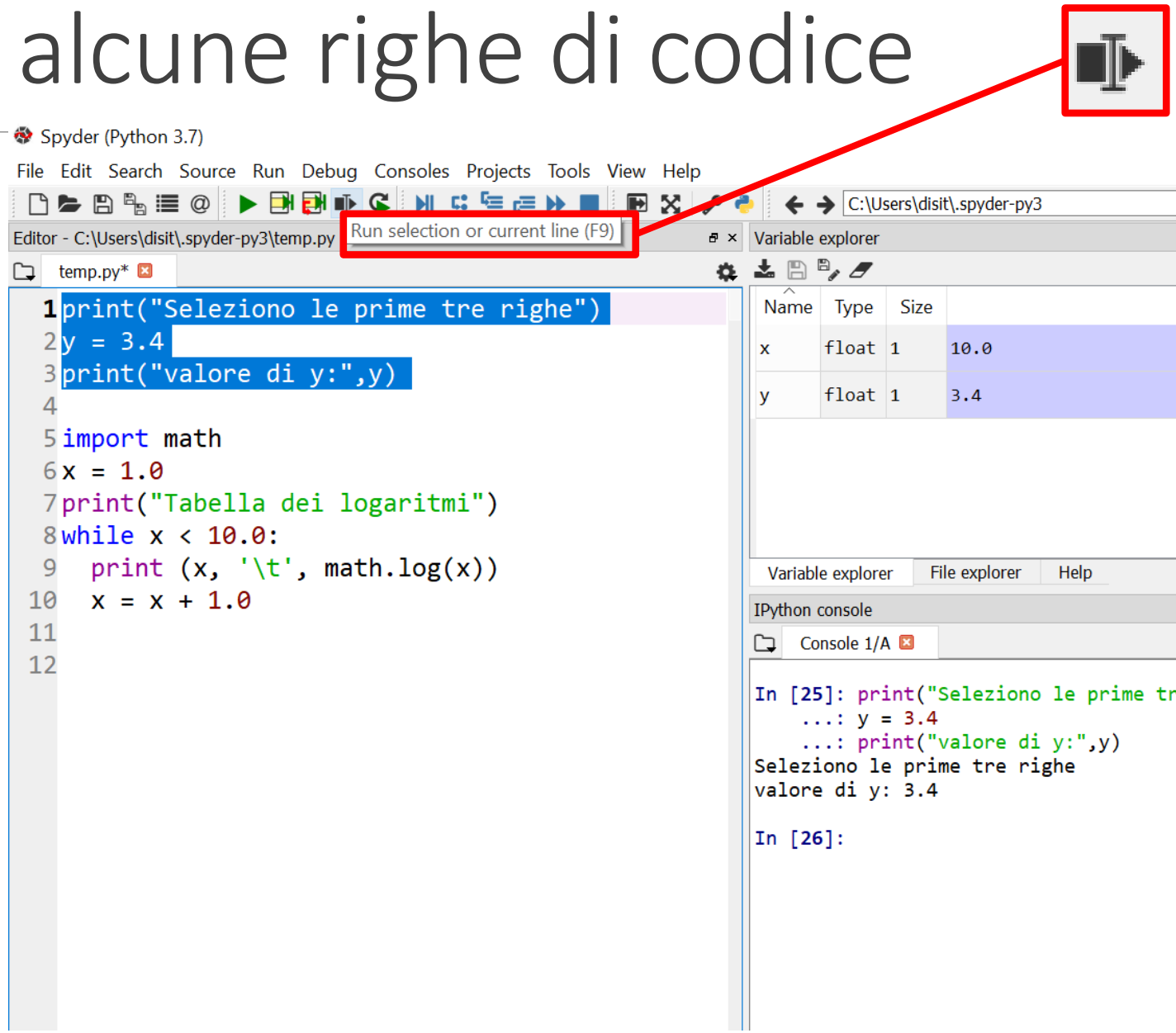


- Selezionare le righe di codice e cliccare su 'Run current cell'

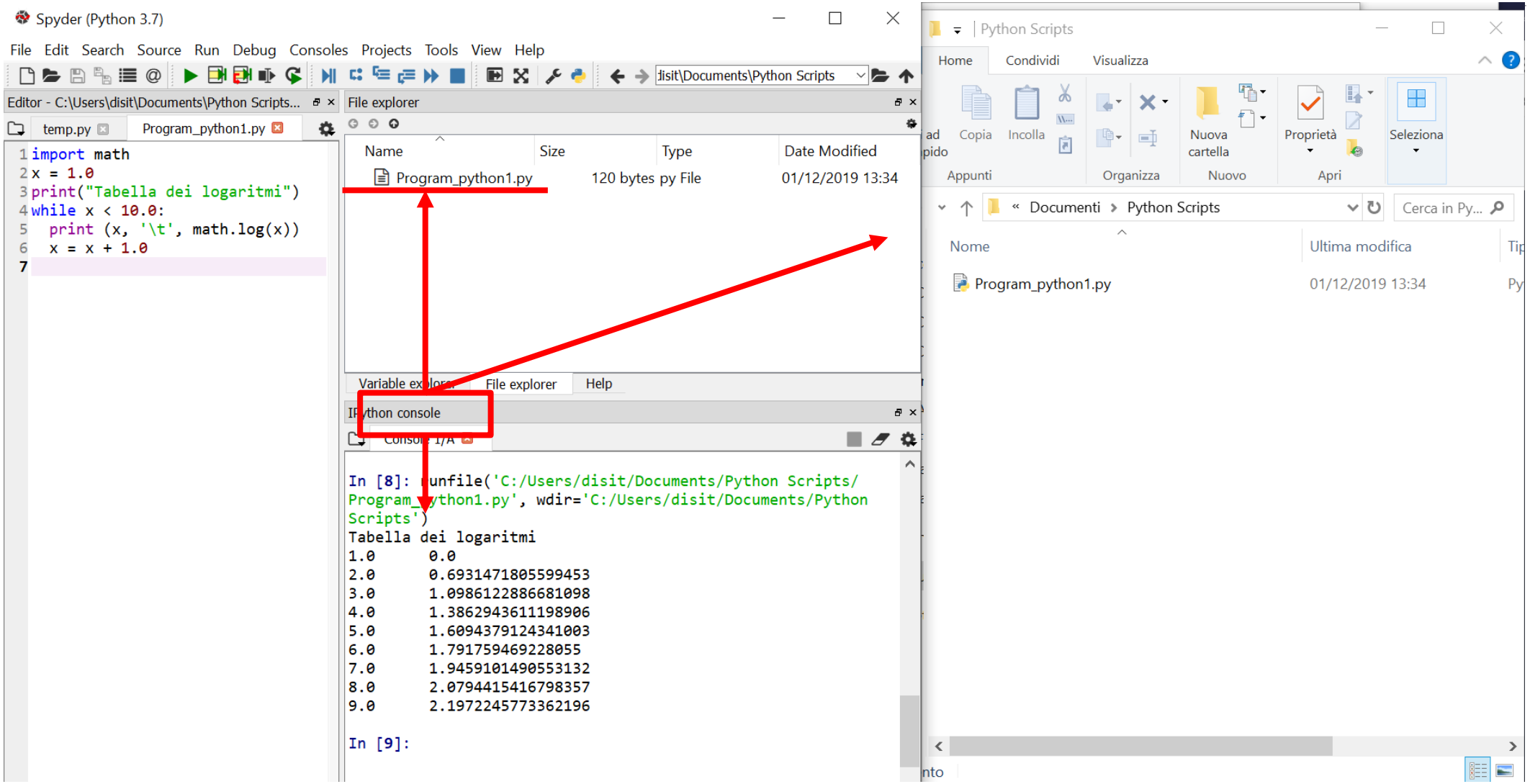


Run: solo alcune righe di codice

- selezionare le righe di codice e cliccare su 'Run selection or current line'



Run a program – file system



Run again last file

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\disit\Documents\Python Scripts

Editor - C:\Users\disit\Documents\Python Scripts\Program_python1.py

temp.py | Program_python1.py

```
1 import math
2 x = 1.0
3 print("Tabella dei logaritmi")
4 while x < 10.0:
5   print (x, '\t', math.log(x))
6   x = x + 1.0
7
8
```

Variable explorer

Name	Type	Size	Value
x	float	1	10.0
y	float	1	3.4

Variable explorer | File explorer | Help

IPython console

Console 1/A

```
6.0      1.791759469228055
7.0      1.9459101490553132
8.0      2.0794415416798357
9.0      2.1972245773362196

In [30]: runfile('C:/Users/disit/Documents/Python Scripts/Program_python1.py', wdir='C:/Users/disit/
Documents/Python Scripts')
Tabella dei logaritmi
1.0      0.0
2.0      0.6931471805599453
3.0      1.0986122886681098
4.0      1.3862943611198906
5.0      1.6094379124341003
6.0      1.791759469228055
7.0      1.9459101490553132
8.0      2.0794415416798357
9.0      2.1972245773362196

In [31]:
```

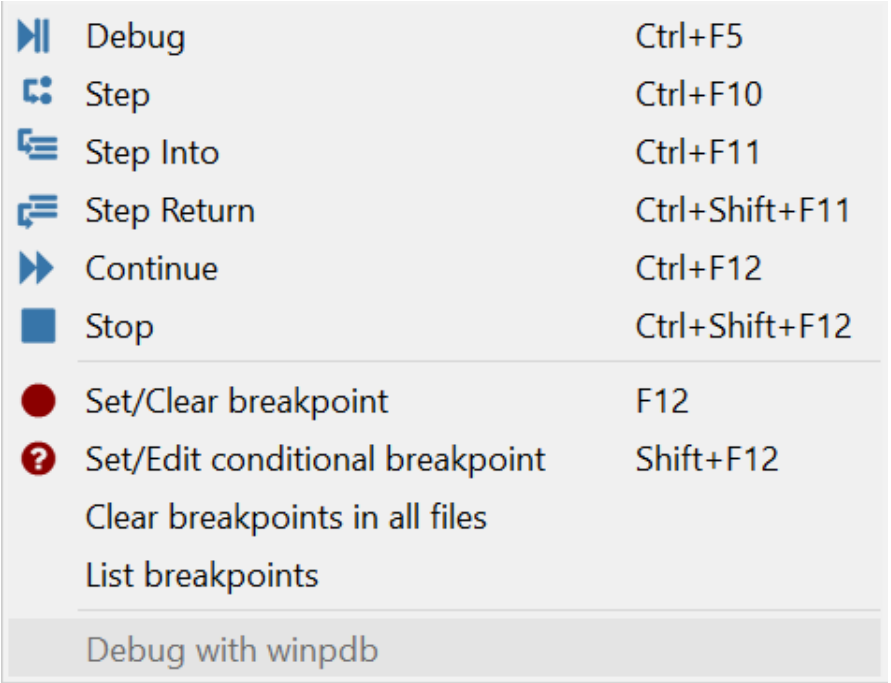
IPython console | History log

Debugger – Concetto di debug (1)

- Il Debugger serve come supporto per eliminare gli eventuali errori di programmazione
- Gli errori in un programma (che implementa un algoritmo) possono essere di tipo:
  - Sintattico: segnati dal Debugger
  - Semantico: NON sono segnalati dal Debugger. Il programma non fa quello che dovrebbe …
- Il debugger serve anche per verificare che tutti i passaggi dell'algoritmo (istruzioni) siano corretti
  - Il programmatore deve infatti sapere quali sono gli effetti di ogni istruzione che viene eseguita dal compilatore

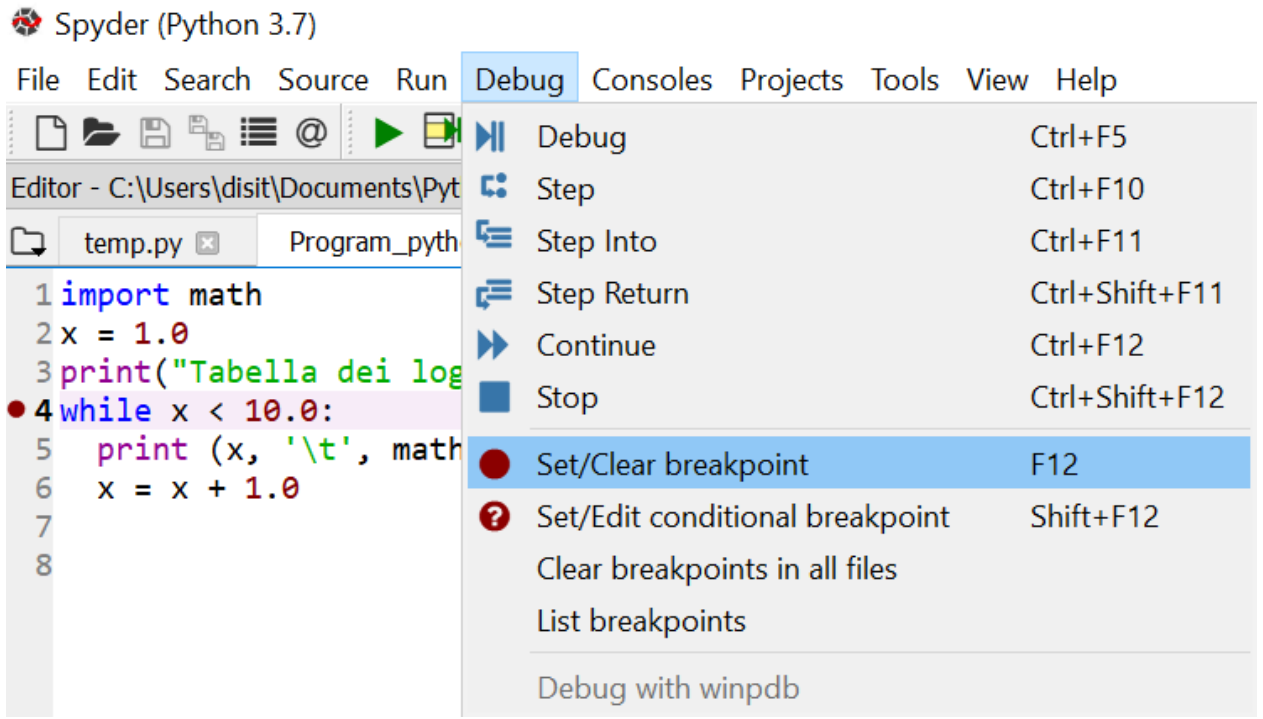
Debugger – Concetto di debug (2)

- L'ambiente di sviluppo (in questo caso Spider) fornisce degli strumenti per monitorare lo stato del programma
  - Ad esempio permette di vedere il valore delle variabili in un determinato 'momento di esecuzione'
- Una volta 'Avviato' il Debugger, è possibile controllare l'esecuzione del programma istruzione per istruzione, grazie all'uso degli step
  - Step -> singola riga
  - Step Into -> continua il debug entrando nel codice di una funzione (uso e definizione delle funzioni)
  - Step Return -> esce dalla funzione (opposto a step into), da approfondire in futuro…
  - Continue (fino alla fine o fino al prossimo 'breakpoint')

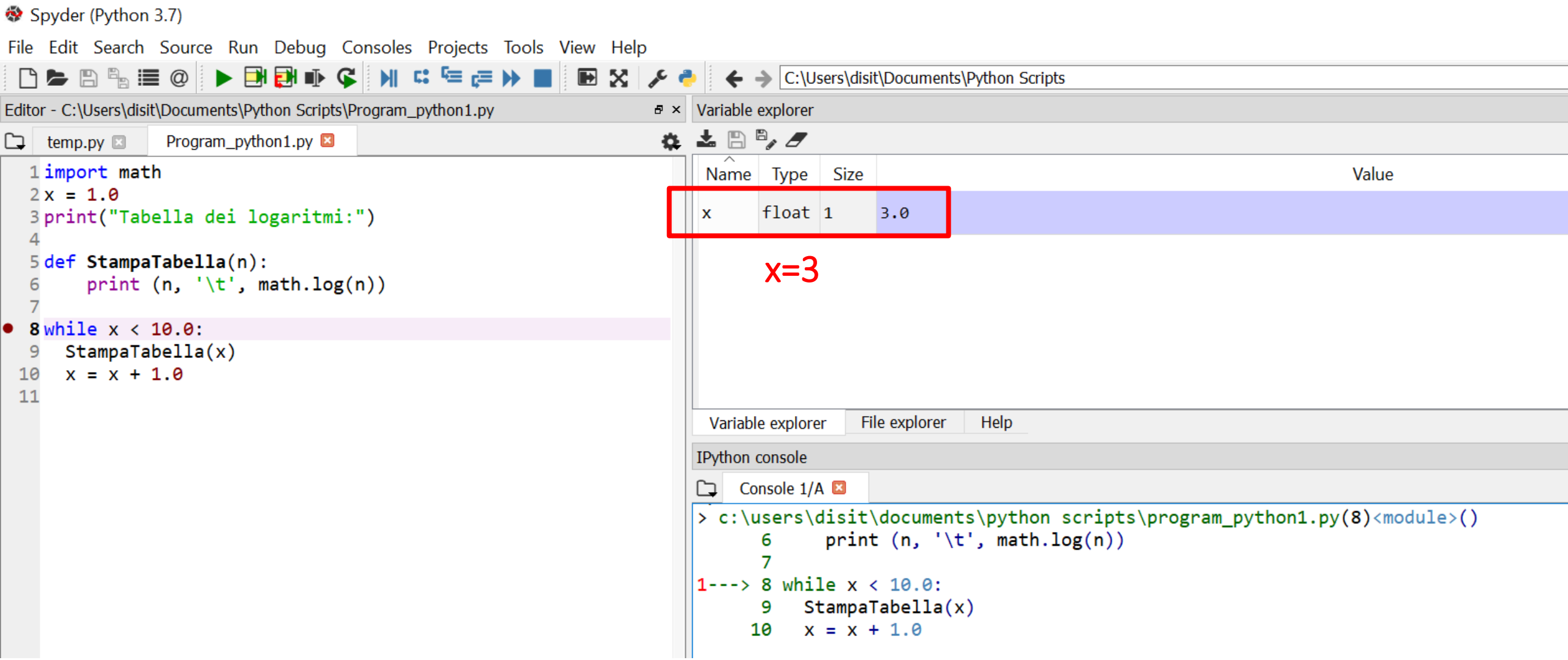


Concetto di Breakpoint

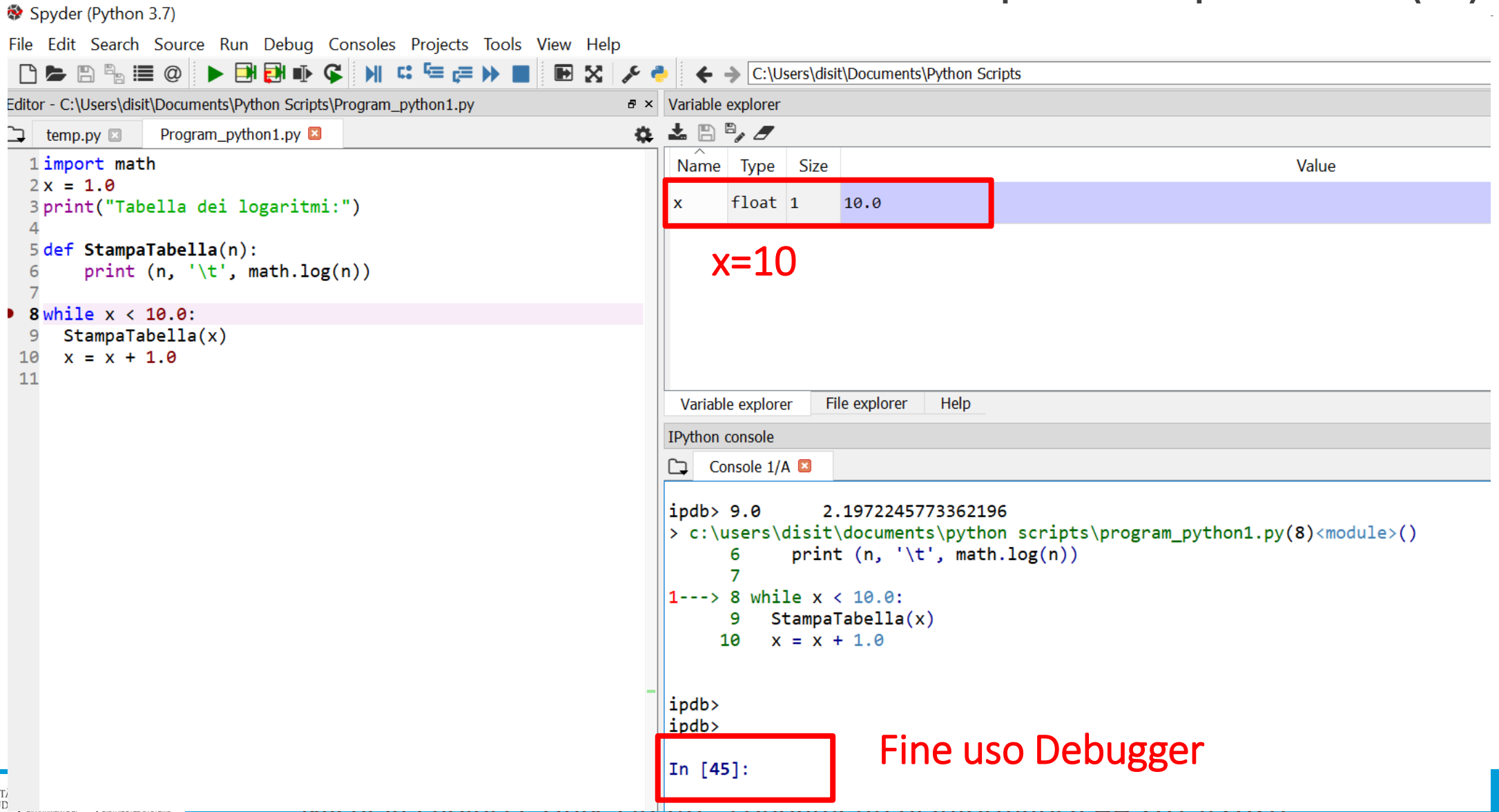
- I breakpoint (punti di interruzione) vengono inseriti in corrispondenza delle istruzioni del programma che hanno rilevanza nel monitoraggio
- Una volta bloccata l'esecuzione del programma è possibile:
  - Continuare fino al breakpoint successivo
  - Continuare passo passo, ovvero istruzione per istruzione (usando uno dei vari step)
- Inoltre è possibile:
  - Monitorare le variabili 'locali' presenti nel
    - programma via via che viene eseguito



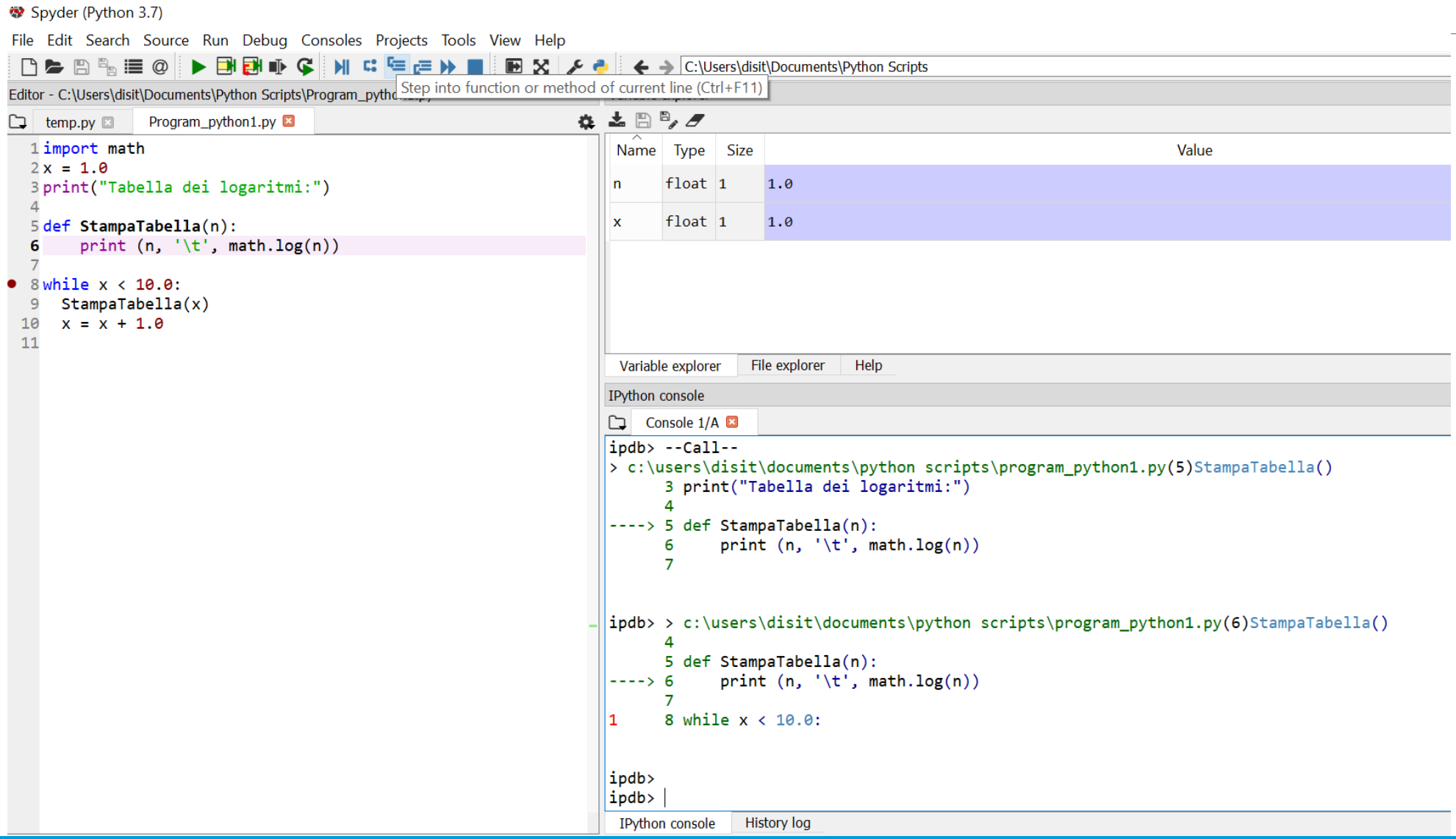
Vedere cosa si scrive in Console ‘passo passo’ (1)



Vedere cosa si scrive in Console ‘passo passo’ (2)



Esempio di ‘step into’



Rappresentazione Dati

Programma del corso - Cognomi A-L

- Introduzione al linguaggio python
  - Tipi, variabili e costanti
  - Operatori ed espressioni
  - Istruzioni
- Rappresentazione dei dati
  - Numeri
  - Interi
  - Caratteri e stringhe

Elementi di sintassi di un linguaggio

Numeri

- Un NUMERO è un ente dotato di un suo significato intrinseco, del quale è possibile dare rappresentazioni diverse
- ESEMPIO:
  - Il numero 'quattro' definisce un concetto che ha un proprio significato. E' possibile associare a tale concetto rappresentazioni diverse:
    - IV, 4, 100 (in binario)
- Le diverse rappresentazioni sono caratterizzate dalla combinazione di più convenzioni:
  - Posizionalità della codifica
  - Numero di cifre
  - Codifica del segno
  - Rappresentazione di parti frazionarie e valori razionali

Codifica posizionale

- I Numeri Naturali sono codificati attraverso una base finita di cifre elementari (0-9)
- Il peso delle cifre dipende dalla posizione in cui appaiono:
- $[a_N a_{N-1}\ a_1 a_0]_B ::= \sum_{n=0}^{N} a_n B^n$
- Con:
  - B = base di numerazione della codifica posizionale
  - a_N cifra più significativa (Most Significant Digit, MSD)
  - a_0 cifra meno significativa (Least Significant Digit, LSD)
- Una convenzione aggiuntiva definisce l'ordine con cui sono presentate le cifre:
  - **Big Endian**: in ultima posizione la cifra più significativa (leggendo da sx a dx, il MSD è a sinistra)
  - **Litte Endian**: in ultima posizione la cifra meno significativa (leggendo da sx a dx, il MSD è a destra)

Base di numerazione

- La base di numerazione che si usa nella vita quotidiana è la base 10
- I calcolatori usano la base 2
- La base 2 è la base minima che include cifre diverse, ovvero la base minima su cui è possibile contare in maniera posizionale
- E' utile studiare la conversione della base di rappresentazione

Operazioni con i binari

SOMMA

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0, con riporto di 1

PRODOTTO

0 * 0 = 0

0 * 1 = 0

1 * 0 = 0

1 * 1 = 1

SOTTRAZIONE

0 - 0 = 0

1 - 1 = 0

1 - 0 = 1

0 - 1 = 1, con prestito di 1

DIVISIONE

Dividendo/divisore = 1, SE ci sta

Dividendo/divisore = 0, SE non ci sta

Esempi di somma

Base 10	Base 2
	1 1 1
19 +	1 0 0 1 1 +
17 =	1 0 0 0 1 =
36	1 0 0 1 0 0

SOMMA

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0, con riporto di 1

Base 10	Base 2
	1 1 1 1 1 1
55 +	1 1 0 1 1 1 +
29 =	0 1 1 1 0 1 =
84	1 0 1 0 1 0 0

Base 10	Base 2
	1 1 1 1 1 1
103 +	1 1 0 0 1 1 1 +
41 =	0 1 0 1 0 0 1 =
144	1 0 0 1 0 0 0 0

Esempi di sottrazione

Base 10	Base 2
	0 1 0 1 0 1
84 -	**1** 0 **1** 0 **1** 0 0 -
29 =	0 0 1 1 1 0 1 =
55	0 1 1 0 1 1 1

SOTTRAZIONE

0 - 0 = 0

1 - 1 = 0

1 - 0 =1

0 - 1 = 1, con prestito di 1

Base 10	Base 2
	0 0 1
155 -	1 **1** **1** 0 0 1 1 -
23 =	0 0 1 0 1 1 1 =
92	1 0 1 1 1 0 0

Base 10	Base 2
	0 0 0 1 1 0
178 -	**1** 0 **1** **1** 0 0 **1** 0 -
95 =	0 1 0 1 1 1 1 1 =
83	0 1 0 1 0 0 1 1

Conversione della base di rappresentazione (1)

- Esistono due algoritmi di conversione della base di rappresentazione
  - Basati sull'uso dell'aritmetica di base
  - Basati sull'uso dell'aritmetica di arrivo
- CASO 1a): **Conversione in base di arrivo DA base 10 A base 2**, Algoritmo:
  1) Il numero iniziale viene sviluppato in forma polinomiale nella base di partenza
  2) I singoli coefficienti e potenze che ne derivano sono convertiti nella base di arrivo (occorre quindi conoscerne la conversione)
  3) La conversione è completata effettuando somme e prodotti nella base di arrivo
- Esempio:

$$[a_N a_{N-1}\ a_1 a_0]_B ::= \sum_{n=0}^{N} a_n B^n$$

- $[125]_{10} = [1*10*10+2*10+5]_{10} = [1*1010*1010+10*1010+0101]_2$

Conversione della base di rappresentazione (2)

- Esempio:
  - $[125]_{10}$= [1*10*10+2*10+5]$_{10}$ = //1) sviluppo in forma polinomiale
  - $[1*1010*1010+10*1010+0101]_2$ //2) coefficienti e potenze sono convertiti nella base di arrivo
- Con:
  - $[10]_{10}$= $[1010]_2$
  - $[2]_{10}$= $[10]_2$
  - $[5]_{10}$= $[0101]_2$

- 3) a questo punto è necessario fare la somma dei singoli addendi (dal punto 2)), si procede con il calcolo dei prodotti, ricordando le regole di base

Conversione della base di rappresentazione (3)

- Calcolo dei prodotti, ricordando le regole di base:

```
                1010*1010
          --------------------
                 0 0 0 0
               1 0 1 0
                0 0 0 0
                 1 0 1 0
          -----------------------------
                 1 1 0 0 1 0 0
```

- Ripetendo per i vari prodotti e facendo la somma finale si ottiene:

$[125]_{10} = [1100100 + 10100 + 0101]_2 = [1111101]_2$

Esempi conversione in base di arrivo da base 10 a base 2 (1)

$[134]_{10}=[10*10+2*10+4]_{10}=[1010*1010+0011*1010+0100]_2=$

1010*1010

```
    0000
   11010
  0000
 1010
1100100
```

0011*1010

```
    0000
   0011
  0000
 0011
0011110
```

Base 10	Base 2
	1 1 1 1
100 +	1 1 0 0 1 0 0 +
30 +	0 0 1 1 1 1 0 +
4 =	0 0 0 0 1 0 0 =
134	1 0 0 0 0 1 1 0

$[134]_{10}=[10000110]_2$

Esempi conversione in base di arrivo da base 10 a base 2 (2)

$[671]_{10}=[6*10^2+7*10+1]_{10}=[0110*1010*1010+0111*1010+0001]_2$

Sapendo che: $[1010*1010]_2= [1100100]_2$

$[600]_{10}$

```
   1100100*0110
   -------------
     0000000
    1100100
   1100100
   0000000
  -----------
  1001011000
```

```
    0111*1010
    ---------
     10000
    10111
   10000
   0111
  -------
  1000110
```

Base 10	Base 2
	1
600 +	1 0 0 1 0 1 1 0 0 0 +
70 +	0 0 0 1 0 0 0 1 1 0 +
1 =	0 0 0 0 0 0 0 0 0 1 =
671	1 0 1 0 0 1 1 1 1 1

$[671]_{10}=[1010011111]_2$

Conversione della base di rappresentazione (4)

- CASO 1b): **Conversione in base di arrivo DA base 2 A base 10:**

$[1100100]_2 = [1*2^6 + 1*2^5 + 1*2^2]_{10} = [64 + 32 + 4]_{10} = [100]_{10}$

NOTA: quando la conversione è da base 2 a base 10, conviene fare i conti in base di arrivo

Esercizi conversione in base di arrivo

da base 2 a base 10

- $[1100111]_2 = [1*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0]_{10} =$
  $= [64+32+4+2+1]_{10} = [103]_{10}$
- $[10010110]_2 = [1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0]_{10} =$
  $= [128+16+4+2]_{10} = [150]_{10}$
- $[10010]_2 = [1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0]_{10} = [16+2]_{10} = [18]_{10}$
- $[11111110]_2 = [1*2^8 + 1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0]_{10} =$
  $= [256+128+64+32+16+8+4+2]_{10} = [510]_{10}$

Conversione della base di rappresentazione (5)

- CASO 2): **Conversione in base di partenza DA base 10 A base 2**:
- Algoritmo dei resti successivi
- In questo caso conviene usarlo per passare dalla base 10 alla base 2
- La rappresentazione di un numero A in base 2 è determinata dalla equazione:

- $A = \sum_{k=0}^{\infty} a_k 2^k =$
  $= a_0 * 2^0 + \sum_{k=1}^{\infty} a_k 2^k =$ //sia k=z+1
  $= a_0 * 1 + \sum_{z=0}^{\infty} a_{z+12}{}^{z+1} =$ //sostituisco k=z
  $= a_0 + 2 \sum_{k=0}^{\infty} a_{k+1}\ 2^k$

Conversione della base di rappresentazione (6)

- Dalla equazione precedente si deduce che a_0 vale 1 Se e solo se A è dispari (A= a_0 + numero pari)
- Si può calcolare a_0 come il **resto** della divisione intera A/2:

$$A/2 = \sum_{k=0}^{\infty} a_{k+12}{}^{k}$$

- Per determinare a_1 e tutti gli altri coefficienti si procede analogamente. Il procedimento termina quando il resto della divisione per 2 è 0

Base10	Resto divisione per due
125	1
62	0
31	1
15	1
7	1
3	1
1	1
0	0

$[125]_{10}=[1111101]_2$

Conversione in base di partenza DA base 10 A base 2

356	0
178	0
89	1
44	0
22	0
11	1
5	1
2	0
1	1
0	0

$[356]_{10}=[101100100]_2$

1279	1
639	1
319	1
159	1
79	1
39	1
19	1
9	1
4	0
2	0
1	1
0	0

$[1279]_{10}=[10011111111]_2$

596	0
298	0
149	1
74	0
37	1
18	0
9	1
4	0
2	0
1	1
0	0

$[596]_{10}=[1001010100]_2$

Base esadecimale (1)

- I numeri in base due sono lunghi
- Memorizzare la rappresentazione in base 10 è semplice, mentre quella in base due è più complessa:
  - $[125]_{10} = [1111101]_2$
- Per ottenere una codifica più compatta si usa spesso la base 16
- In base 16:
  - Ci sono 16 cifre: [0-9] e A,B,C,D,E,F
  - Le cifra rappresentano i numeri in base 10, rispettivamente: 11,12,13,14,15
- La conversione da base 2 a base esadecimale si ottiene 'impaccando' i bit:
  - $[125]_{10} = [1111101]_2 = [01111101]_2 = [7D]_{16}$

Dec	Hex
0	0
1	1
…	…
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Base esadecimale (2)

- Partendo da:
  - $[125]_{10} = [1111101]_2$
- La conversione da base 2 a base esadecimale si ottiene 'impaccando' i bit:
  - $[125]_{10} = [01111101]_2 = [FD]_{16}$
- Con:
  - $[0111]_2 = [0*2^3+1*2^2+1*2^1+1*2^0]_{10} = [4+2+1]_{10} = [7]_{10} = [7]_{16}$
  - $[1101]_2 = [1*2^3+1*2^2+0*2^1+1*2^0]_{10} = [13]_{10} = [D]_{16}$

Esercizi conversione da base 10 a base esadecimale (1)

1) conversione da base 10 a base 2

$[497]_{10}=[4*10^2+9*10+7]_{10}=[0100*1010*1010+1001*1010+0111]_2$

Con: $[1010*1010]_2$= 1100100

1100100*0100

0 0 0 0 0 0 0

0 0 0 0 0 0 0

1 1 0 0 1 0 0

0 0 0 0 0 0 0

0 1 1 0 0 1 0 0 0 0

1001*1010

0 0 0 0

1 0 0 1

0 0 0 0

1 0 0 1

1 0 1 1 0 1 0

Base 10	Base 2
	1 1 1 1
400 +	0 1 1 0 0 1 0 0 0 0 +
90 +	0 0 0 1 0 1 1 0 1 0 +
7 =	0 0 0 0 0 0 0 1 1 1 =
497	0 1 1 1 1 1 0 0 0 1

Esercizi conversione da base 10 a base esadecimale (2)

2) si ‘impaccano’ i bit (a partire da destra)

$[497]_{10}=[4*10^2+9*10+7]_{10}=[0100*1010*1010+1001*1010+0111]_2$
$= [0111110001]_2 =$

$[0001]_2 = [2^0]_{10}= [1]_{10} = [1]_{16}$
$[1111]_2 = [2^3 + 2^2 + 2^1 + 2^0]_{10}= [15]_{10} = [F]_{16}$
$[0001]_2 = [2^0]_{10}= [1]_{10} = [1]_{16}$

$[497]_{10}= [0111110001]_2 = [1F1]_{16}$

Esercizi conversione da base 10 a base esadecimale (3)

1) conversione da base 10 a base 2

$[345]_{10}=[3*10^2+4*10+5]_{10}=[0100*1010*1010+1001*1010+0101]_2$

Con: $[1010*1010]_2= 1100100$

```
  1100100*0011
 ------------
  1 1 1 0 0 1 0 0
    1 1 1 0 0 1 0 0
    0 0 0 0 0 0 0
  0 0 0 0 0 0 0
 ------------------
  0 1 0 0 1 0 1 1 0 0
```

```
     0100*1010
    ----------
      0 0 0 0
    0 1 0 0
      0 0 0 0
  0 1 0 0
 ------------
  0 1 0 1 0 0 0
```

Base 10	Base 2
	1 1 1
300 +	0 1 0 0 1 0 1 1 0 0 +
40 +	0 0 0 0 1 0 1 0 0 0 +
5 =	0 0 0 0 0 0 0 1 0 1 =
345	0 1 0 1 0 1 1 0 0 1

Esercizi conversione da base 10 a base esadecimale (4)

2) si ‘impaccano’ i bit (a partire da destra)

$[345]_{10}=[3*10^2+4*10+5]_{10}=[0100*1010*1010+1001*1010+0101]_2$
$= [101011001]_2 =$

$[0001]_2 = [2^0]_{10}= [1]_{10} = [1]_{16}$
$[0101]_2 = [2^2 + 2^0]_{10}= [5]_{10} = [5]_{16}$
$[1001]_2 = [2^2 + 2^0]_{10}= [9]_{10} = [1]_{16}$

$[497]_{10}= [0111110001]_2 = [159]_{16}$

Parti Frazionarie (1)

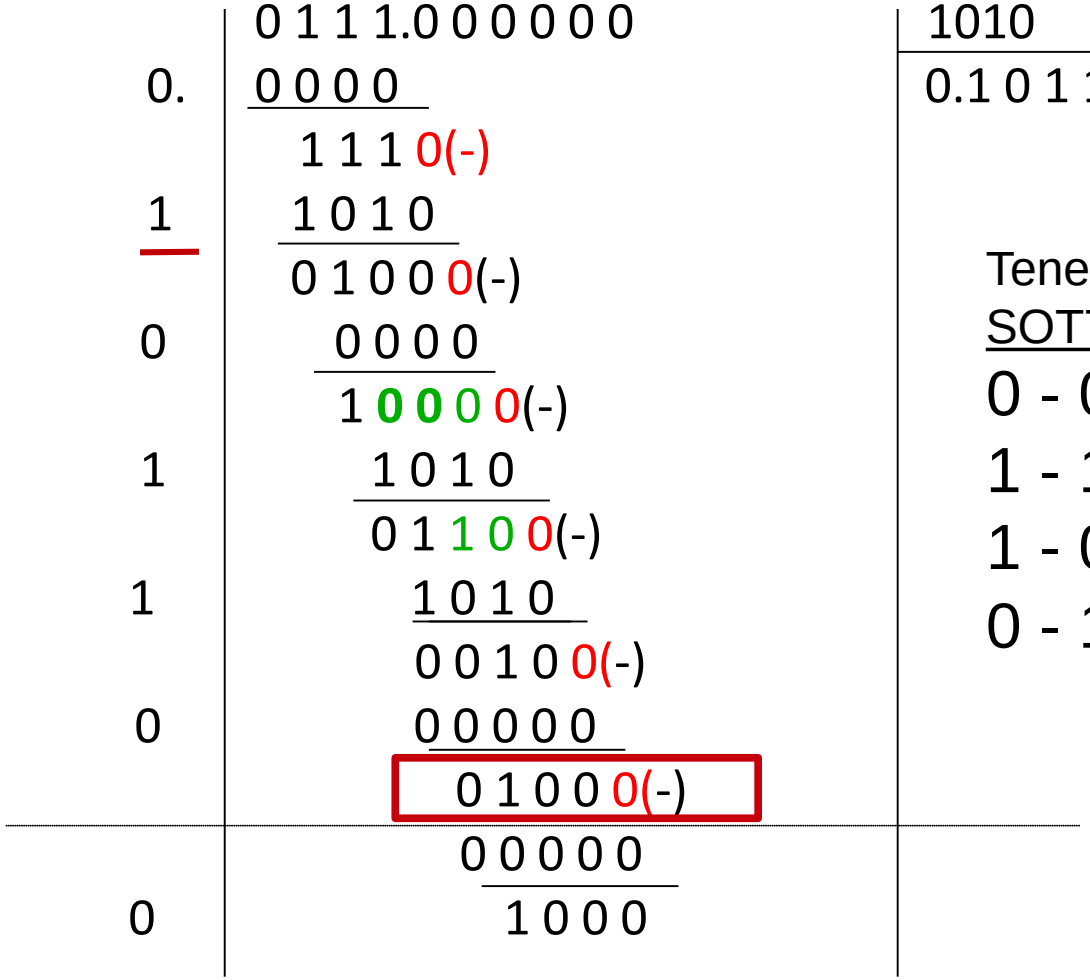
- Fino ad ora abbiamo visto le conversioni di numeri interi
- Una parte frazionaria è un numero razionale inferiore all'unità
- Come si è visto per i numeri interi, anche nel caso della parte frazionaria, la conversione può essere effettuata in due modalità:
  - Base di arrivo
  - Base di partenza

Parti Frazionarie - Conversione in base di arrivo (1)

- **Conversione in base di arrivo:**
  - (a) Si sviluppa la parte frazionaria in forma polinomiale nella base di partenza
  - (b) Si convertono i singoli termini nella base di arrivo
  - (c) Si eseguono le operazioni in base di arrivo
- Esempio:
  - $[0.7]_{10} \overset{(a)}{=} [7*10^{-1}]_{10} = [7/10]_{10} \overset{(b)}{=} [0111/1010]_2$

  - Si esegue poi la divisione tra binari (c)
    - Le regole per eseguire la divisione tra binari sono le stesse regole usate nei decimali

Parti Frazionarie - Conversione in base di arrivo (2)



1010

0.1 0 1 1 0

Tenere presenti le regole della SOTTRAZIONE

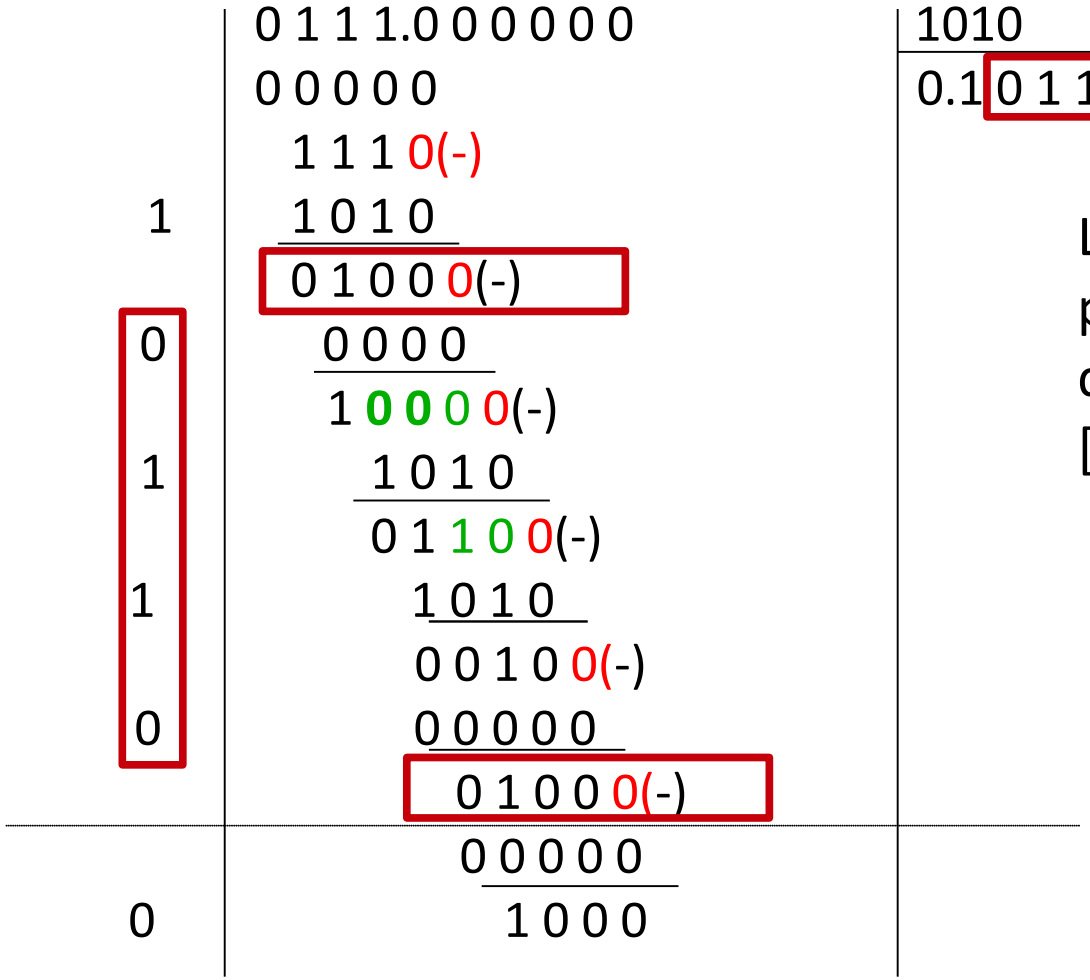
0 - 0 = 0

1 - 1 = 0

1 - 0 = 1

0 - 1 = 1, con prestito di 1

Parti Frazionarie - Conversione in base di arrivo (2)

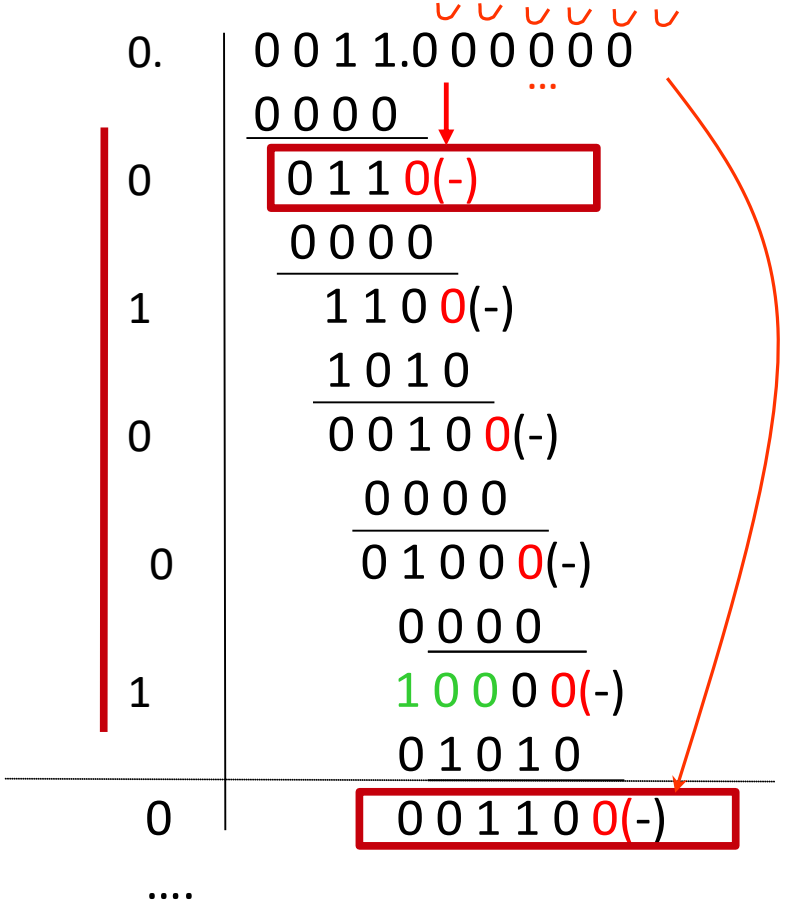


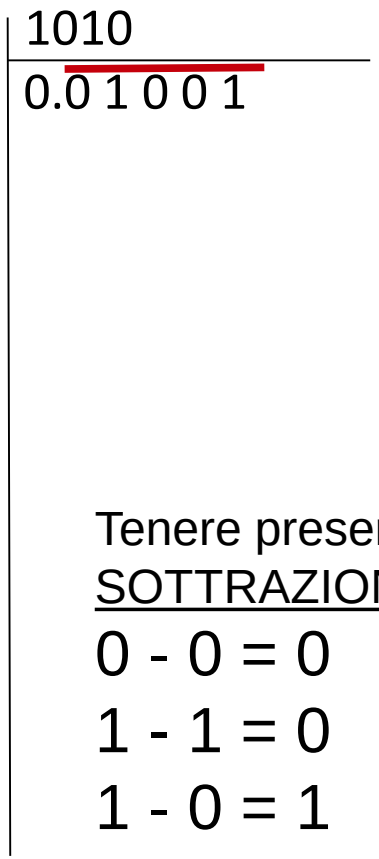
Le ultime 4 cifre si ripetono periodicamente, quindi abbiamo:

$[0.7]_{10} = [0.\overline{10110}]_2$

Esempio2 conversione parti frazionarie in base di arrivo

- $[0.3]_{10} = [3*10^{-1}]_{10} = [3/10]_{10} = [0011/1010]_2$





Le 5 cifre dopo la virgola si ripetono periodicamente, quindi abbiamo:
$[0.3]_{10} = [0.01001]_2$

Tenere presenti le regole della SOTTRAZIONE

0 - 0 = 0

1 - 1 = 0

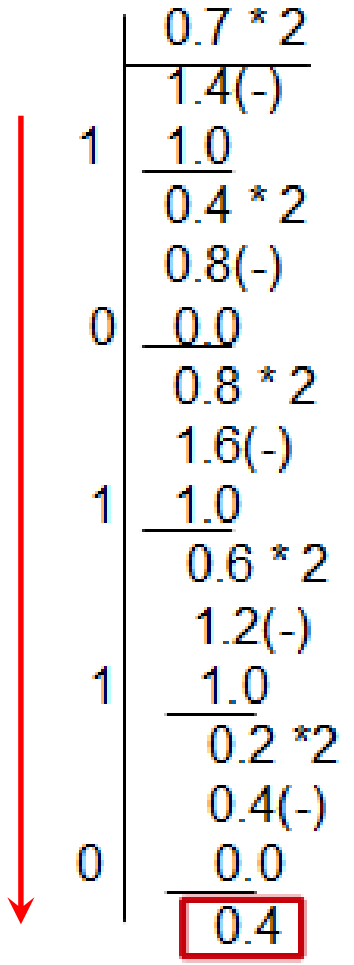
1 - 0 = 1

0 - 1 = 1, con prestito di 1

Parti Frazionarie - Conversione in base di partenza (1)

- Conversione in base di partenza:
  - (a) Si moltiplica per 2 la parte frazionaria
  - (b) Si sottrae dal risultato ottenuto in (a) la sua parte intera
  - (c) La sequenza delle parti intere sottratte fornisce la rappresentazione voluta

- Esempio:
  - $[0.7]_{10}$



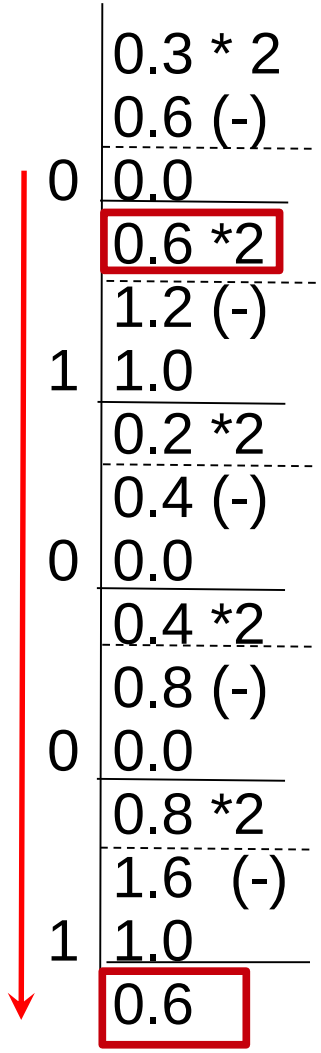
Parti Frazionarie - Conversione in base di partenza (2)

	0.7 * 2
	1.4(-)
1	1.0
	0.4 * 2
	0.8(-)
0	0.0
	0.8 * 2
	1.6(-)
1	1.0
	0.6 * 2
	1.2(-)
1	1.0
	0.2 *2
	0.4(-)
0	0.0
	0.4

Le ultime 4 cifre si ripetono periodicamente, quindi abbiamo:

$[0.7]_{10}=[0.\overline{10110}]_2$

Parte frazionaria – conversione in base di partenza



Le 5 cifre si ripetono periodicamente, quindi abbiamo:

$[0.7]_{10}=[0.01001]_2$

Interi senza segno (1)

- Un intero senza segno viene rappresentato in base 2 su N bit di memoria $\{a_i\}$ con i che va da 0 a N-1
  - **a** = $\sum_{i=0}^{N-1} a_i 2^i$
  - La rappresentazione codifica tutti e soli i numeri che vanno da 0 a 2^N-1 inclusi
- Esempio:
  - Se N=8, allora **a** appartiene all'intervallo [0,255]
- Il numero N di bit varia a seconda dell'architettura della macchina (solitamente a 32 o 64 bit)

Caratteri

- Un carattere è rappresentato in memoria su N=8 bit, che codificano un numero intero senza segno tra 0 e 255
- La corrispondenza tra i valori numerici compresi tra 0 e 255 e i caratteri è stabilita dalla **Tabella dei codici ASCII** che include:
  - Caratteri alfanumerici
    - Cifre decimali
    - Lettere minuscole e maiuscole
  - Simboli di interpunzione e parentesi (es: [ ] { })
  - Simboli aritmetici (es: * + - %)
  - Caratteri di vario genere (es: @ #)
  - Caratteri di controllo (es: a capo, tabulazione, etc.)

Tabella dei codici ANSI-ASCII

	032	@	064	`	096	€	0128		0160	À	0192	à	0224
!	033	A	065	a	097		0129	¡	0161	Á	0193	á	0225
"	034	B	066	b	098	‚	0130	¢	0162	Â	0194	â	0226
#	035	C	067	c	099	ƒ	0131	£	0163	Ã	0195	ã	0227
$	036	D	068	d	0100	„	0132	¤	0164	Ä	0196	ä	0228
%	037	E	069	e	0101	…	0133	¥	0165	Å	0197	å	0229
&	038	F	070	f	0102	†	0134	¦	0166	Æ	0198	æ	0230
'	039	G	071	g	0103	‡	0135	§	0167	Ç	0199	ç	0231
(	040	H	072	h	0104	ˆ	0136	¨	0168	È	0200	è	0232
)	041	I	073	i	0105	‰	0137	©	0169	É	0201	é	0233
*	042	J	074	j	0106	Š	0138	ª	0170	Ê	0202	ê	0234
+	043	K	075	k	0107	‹	0139	«	0171	Ë	0203	ë	0235
,	044	L	076	l	0108	Œ	0140	¬	0172	Ì	0204	ì	0236
-	045	M	077	m	0109		0141		0173	Í	0205	í	0237
.	046	N	078	n	0110	Ž	0142	®	0174	Î	0206	î	0238
/	047	O	079	o	0111		0143	¯	0175	Ï	0207	ï	0239
0	048	P	080	p	0112		0144	°	0176	Ð	0208	ð	0240
1	049	Q	081	q	0113	‘	0145	±	0177	Ñ	0209	ñ	0241
2	050	R	082	r	0114	’	0146	²	0178	Ò	0210	ò	0242
3	051	S	083	s	0115	“	0147	³	0179	Ó	0211	ó	0243
4	052	T	084	t	0116	”	0148	´	0180	Ô	0212	ô	0244
5	053	U	085	u	0117	•	0149	µ	0181	Õ	0213	õ	0245
6	054	V	086	v	0118	–	0150	¶	0182	Ö	0214	ö	0246
7	055	W	087	w	0119	—	0151	·	0183	×	0215	÷	0247
8	056	X	088	x	0120	˜	0152	¸	0184	Ø	0216	ø	0248
9	057	Y	089	y	0121	™	0153	¹	0185	Ù	0217	ù	0249
:	058	Z	090	z	0122	š	0154	º	0186	Ú	0218	ú	0250
;	059	[	091	{	0123	›	0155	»	0187	Û	0219	û	0251
<	060	\	092	|	0124	œ	0156	¼	0188	Ü	0220	ü	0252
=	061	]	093	}	0125		0157	½	0189	Ý	0221	ý	0253
>	062	^	094	~	0126	ž	0158	¾	0190	Þ	0222	þ	0254
?	063	_	095		0127	Ÿ	0159	¿	0191	ß	0223	ÿ	0255

- Esempi:

il carattere 0 è codificato dal numero 48

Le parentesi graffe dai numeri 123 e 124 …

Il numero 0 NON codifica alcun carattere, mentre invece viene usato come segno di terminazione nella codifica delle stringhe

Il complemento

- Dato un numero intero N rappresentato in base b con k cifre, Si definisce Cb, il complemento a b di N in base b, tale che: N + Cb(N) = bk
- Ad esempio, Con bk = 1 0 0 0 0 0 , Per ogni b (1 seguito da k cifre uguali a 0)

- ESEMPIO in base 10:

BASE 10: 3552 è il complemento a 10 di 6448;

Verifica (si effettua la somma):

3552 +
6448

10000 = 10^4, in questo caso si ha: **b=10, k=4**

Il complemento

Esempio in base 2:

- Dato un numero intero N rappresentato in base 2 con k cifre, Si definisce C2, il complemento a 2 di N in base 2, tale che: $N + C2(N) = 2^k$

$[01100]_2$ è il complemento a 2 di $[10100]_2$

Verifica (si effettua la somma)

```
  1 1 1
    0 1 1 0 0
    1 0 1 0 0
  ---------------
  1 0 0 0 0 0
```

$1\,0\,0\,0\,0\,0 = 2^5+0*2^4+...+0*2^0 = 2^5$, **con b=2, k=5 bit**

Algoritmo per calcolare il complemento a 2 di N

- Sia N numero binario

- Algoritmo:
  - Si copiano i bit del numero a partire dal meno significativo fino al primo 1 compreso
  - Si invertono tutti gli altri bit

- Esempio: Calcolare il C2(10100011100000)

10100011100000+

01011100100000 ← Risultato

100000000000000 ← Verifica sommando

Esempi complemento a 2 (1)

- C2[01101101] = ??, k=

2) Si inverte	1) si copia fino al primo 1 compreso
0 1 1 0 1 1 0	1
1 0 0 1 0 0 1	1

- C2[01101101] = 10010011
- Verifica:

Base 10	Base 2
	1 1 1 1 1 1 1
109 +	0 1 1 0 1 1 0 1 +
147 =	1 0 0 1 0 0 1 1 =
$2^8 = 256$	1 0 0 0 0 0 0 0 0

Esempi complemento a 2 (2)

- C2[1011001000] = ??, k=10

2) Si inverte	1) si copia fino al primo 1 compreso
1 0 1 1 0 0	1 0 0 0
0 1 0 0 1 1	1 0 0 0

- C2[1 0 1 1 0 0 1 0 0 0] = **0 1 0 0 1 1** 1 0 0 0
- Verifica:

Base 10	Base 2
	1 1 1 1 1 1
712 +	1 0 1 1 0 0 1 0 0 0 +
312 =	0 1 0 0 1 1 1 0 0 0 =
2^{10}=1024	1 0 0 0 0 0 0 0 0 0 0

Interi con segno (1)

- Gli interi con segno sono rappresentati su N bit usando una rappresentazione in complemento a 2
- Consiste nel rappresentare ogni numero negativo attraverso il complemento a 2 del corrispondente numero positivo
- Questa codifica può essere vista come una variante della codifica posizionale in cui:
  - Il bit più significativo (Most Significative Bit, MSB) ha peso negativo
  - Gli altri bit hanno peso positivo, come visto fino ad ora

Interi con segno (2)

- Quindi se k=8 → l'ottavo bit, quello più significativo (MSB), ovvero quello più a sinistra, rappresenta il segno del numero:
  - + se il MSB = 0
  - - se il MSB = 1

- La sequenza di bit a_{N-1}, a_{N-2}, …, a_1, a_0 codifica il valore:
  - $A = -a_{N-1} * 2^{(N-1)} + \sum_{n=0}^{N-2} a_n * 2^n$, con $a_n = \{0,1\}$

- I valori rappresentati sono i numeri compresi nell'intervallo $[-2^{(N-1)}; 2^{(N-1)}]$

Esempi Interi con segno

Calcolare –A, con A=$[110101]_2$

2) Si inverte | 1) si copia fino al primo 1 compreso

1 1 0 1 0 | 1

0 0 1 0 1 | 1 -A = $[001011]_2$

Calcolare –A, con A=$[0111111]_2 = [63]_{10}$

2) Si inverte 1) si copia fino al primo 1 compreso

0 1 1 1 1 1 1

1 0 0 0 0 0 1 -A = $[1000001]_2$

Bibliografia – Rappresentazione Dati

- Stefano Berretti, Laura Carnevali, Enrico Vicario. Fondamenti di programmazione. Linguaggio C, strutture dati, algoritmi elementari, C++. Editore: Esculapio. ISBN: 9788893850513

Fondamenti di Informatica

AA 2019/2020

Eng. Ph.D. Michela Paolucci

DISIT Lab http://www.disit.dinfo.unifi.it
Department of Information Engineering, DINFO
University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570
michela.paolucci@unifi.it