PARTE 2

Introduzione a Python

Nicolò Bartolini

Argomenti

Cos’è Python

Installazione degli strumenti

Sintassi di base di Python

Variabili e funzioni

→ **Cos’è Python**

Installazione degli strumenti

Sintassi di base di Python

Variabili e funzioni

Cos'è Python

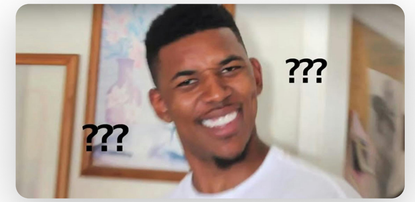
il bro n. 2

Python è un linguaggio di programmazione ad **alto livello**, dinamico, interpretato e **facile da imparare**. È stato creato da Guido van Rossum e rilasciato per la prima volta nel **1991**.

Python si distingue per la sua **leggibilità** e la sua **pulizia sintattica**, consentendo agli sviluppatori di esprimere concetti in **meno righe di codice** rispetto a linguaggi più verbosi (i.e.: C++, Java).

Caratteristiche principali

- **Sintassi semplice** che facilita la lettura e la scrittura del codice.
- **Tipizzazione dinamica**: non è necessario dichiarare il tipo delle variabili.
- **Gestione automatica della memoria**: include un garbage collector.
- **Estensibilità**: supporta moduli e librerie, che incoraggiano la modularità del programma e la riutilizzazione del codice.



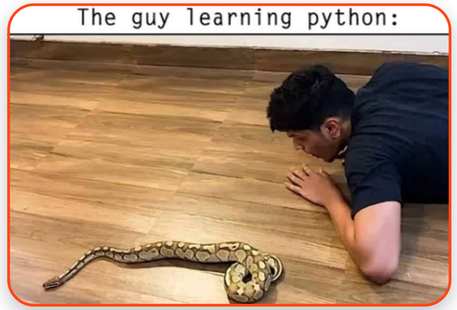
ci arriveremo...

FUN FACT: **Python prende il nome dalla serie TV "Monty Python's Flying Circus", uno dei programmi preferiti del bro. Van Rossum voleva un nome breve, unico e leggermente misterioso, così ha scelto "Python" per riflettere l'ironia e la giocosità del suo approccio alla programmazione.**

Logo di Python

Cosa si può fare con Python

Slide un po' noiosa.
Aggiungo memino



Python è incredibilmente versatile e può essere utilizzato in una vasta gamma di applicazioni. Di seguito solo alcune tra le possibilità di Python:

- **Sviluppo web**: Python può essere utilizzato per la realizzazione del **backend** di **siti web** attraverso framework web avanzati come Flask, FastAPI o Django.
- **Analisi dei dati, statistica e Machine Learning**: Python è il linguaggio **leader** nel settore dell'analisi dei dati e dell'apprendimento automatico, grazie a librerie come Pandas, NumPy, Scikit-Learn e TensorFlow.
- **Automazione**: la semplicità e la versatilità della sintassi di Python lo rendono ottimo per la realizzazione di script che automatizzano **compiti complessi e routine**.
- **Sviluppo di giochi**: seppur non molto utilizzato in questo ambito, Python offre anche la possibilità di realizzare **giochi in 2D** tramite la libreria Pygame.
- **Scienza e calcolo numerico**: grazie a librerie come SciPy e Matplotlib, Python è molto utilizzato nell'ambito scientifico per la simulazione e l'analisi di esperimenti scientifici.
- **Biologia**: Python è anche utilizzato per analizzare dati genomici e proteomici, modellare sistemi biologici e simulare interazioni molecolari, con librerie popolari come BioPython che forniscono strumenti specifici per questi compiti.
- **Scienze forestali**: Python aiuta nella modellazione della crescita forestale, nella valutazione della biomassa e nell'analisi spaziale dei dati forestali grazie a librerie come GDAL o Geopandas.

Cosa è scritto in Python (di famoso)

Nonostante Python sia un linguaggio principalmente utilizzato nell'ambito della **ricerca scientifica**, viene comunque implementato ampiamente anche in molti software che utilizziamo tutti i giorni.

Instagram utilizza Python sul suo lato server per delle funzioni ausiliarie.

Netflix impiega Python in ambito di sicurezza, monitoraggio e decisione algoritmica.

NETFLIX

Spotify utilizza Python sul suo lato server e per effettuare data analytics.

Reddit è, invece, scritto interamente in Python, linguaggio scelto nel 2005 per scalabilità e manutenibilità.

Cos’è Python

→ **Installazione degli strumenti**

Sintassi di base di Python

Variabili e funzioni

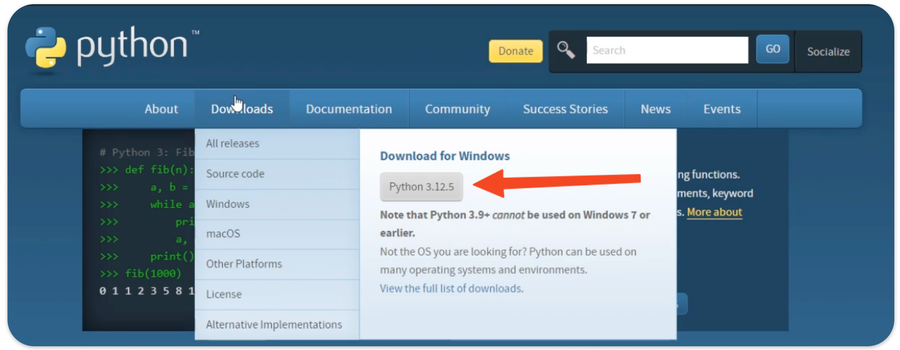
Installazione di Python

Installare Python significa semplicemente installare l'**interprete Python**, ovvero quel programma che andrà a leggere il nostro **codice sorgente** e a interpretarlo in linguaggio macchina, in modo da permettere al computer di eseguirne le istruzioni.

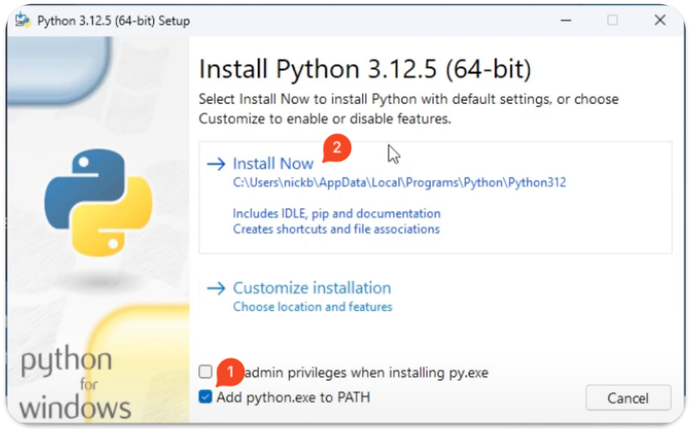
L'installazione si effettua secondo i seguenti passi:

Dettagli aggiuntivi su questo processo possono essere visionati a partire dal minuto 01:04:30 della Lezione 1, accessibile al seguente link: https://www.youtube.com/watch?v=b335d6ol_Yg

① Andare sul sito python.org



② Avviare l'installer appena scaricato, selezionare l'ultima spunta e cliccare su "Install Now"



③ Avviare il "prompt dei comandi" di Windows e digitare e inviare il comando "`python`".

Se l'installazione è andata a buon fine, questo comando avvierà l'IDLE di Python.

④ Scrivere e inviare "`exit()`", poi digitare e lanciare il comando: `python -m pip install --upgrade pip`
Questo aggiornerà il gestore dei pacchetti di Python all'ultima versione.

Cos'è un IDE

Integrated
Development
Environment

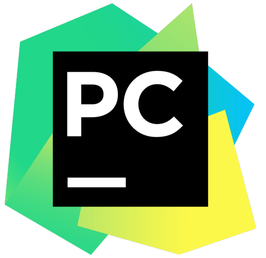
Un **IDE** è un **software** che aiuta i programmatori a **scrivere**, **testare** e **debuggare** il loro codice in modo molto più efficiente.
Di per sé, il codice sorgente di un qualsiasi linguaggio di programmazione, potrebbe essere scritto attraverso qualunque strumento, potenzialmente anche tramite Microsoft Word, purché il file che si sta scrivendo abbia l'**estensione corretta** (.py per Python).
Tuttavia, la programmazione è un processo molto differente rispetto alla scrittura di testi e, per tale motivo, sono stati inventati gli IDE.

Gli IDE forniscono moltissime funzionalità che rendono la scrittura del codice molto più semplice, rapida e immediata. Tra questi strumenti abbiamo:

- **Editor di testo specializzato**: gli IDE possiedono un editor di testo ottimizzato per la scrittura di codice, con un font monospaced, contatore delle righe di codice e colorazione automatica delle parole chiave.
- **Autocompletamento**: scrivendo del codice nell'editor di testo di un IDE, compaiono automaticamente dei possibili completamenti per quello che si sta scrivendo, velocizzando la scrittura generale.
- **Gestore dei file**: gran parte degli IDE fornisce un gestore dei file che permette di navigare rapidamente tra i molteplici file che compongono i progetti più complessi.
- **Strumenti di esecuzione e building**: gli IDE velocizzano anche il processo di esecuzione del codice sorgente grazie a strumenti che lo eseguono senza dover scrivere alcun comando.

Gli strumenti più utilizzati per Python

I due strumenti più utilizzati per **costruire applicazioni in Python** sono **Visual Studio Code** (**VSCode**) e **JetBrains Pycharm**.



ATTENZIONE: **VSCode non è precisamente un IDE. La "I" di IDE sta per "Integrated" e vuole indicare che gli IDE sono integrati alla perfezione con il linguaggio per cui sono stati realizzati. VSCode, invece, può essere utilizzato con tutti i linguaggi, rendendolo, di fatto, un potente editor di testo.**

Memino per compensare la pesantezza della slide precedente

- **Visual Studio Code** (**VSCode**): grazie ai numerosissimi **plug-in** disponibili, è un editor di testo utilizzabile per pressoché qualsiasi linguaggio di programmazione. **Leggero** e totalmente **configurabile**, è ampiamente utilizzato in moltissimi contesti.
- **PyCharm**: sviluppato da **JetBrains**, è un vero e proprio **IDE** specifico per Python. Supporta numerosissime **funzionalità avanzate** e permette di gestire progetti estremamente complessi con facilità. È la **punta di diamante** degli IDE per lo sviluppo in Python.
  (Quelli di JetBrains sono dei dragoni, ogni loro IDE è clamorosamente perfetto, anche quello per Java)

Cosa utilizzeremo

Nonostante io abbia appena acclamato e lodato PyCharm, utilizzeremo, comunque, **Visual Studio Code**.

Il motivo è semplice:
PyCharm può risultare **overwhelming** per chi è alle prime armi a causa delle sue funzionalità avanzate.
VSCode, di contro, è leggero, semplice e versatile, caratteristiche che lo rendono **adatto** in questa situazione.

Possiamo, quindi, procedere all'installazione e alla configurazione iniziale di Visual Studio Code, iniziando dallo scaricare l'installer dal seguente link:

https://code.visualstudio.com/download

Cos’è Python

Installazione degli strumenti

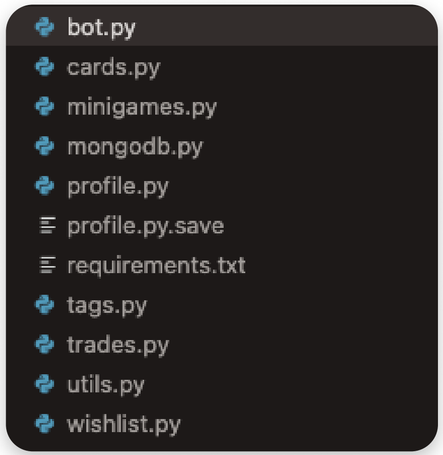
→ **Sintassi di base di Python**

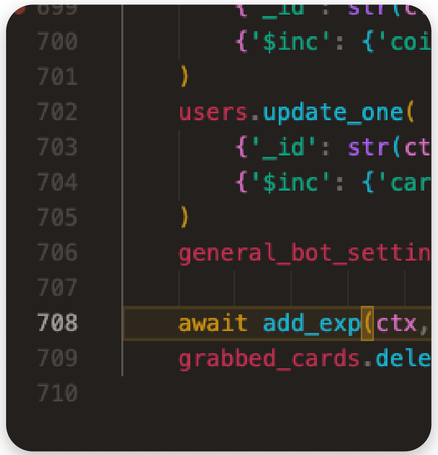
Variabili e funzioni

Struttura di base di un programma Python

Quando si scrive un programma in Python, si parte da un **codice sorgente** costituito da uno o più **file**. Di solito, i file di un programma Python hanno l'**estensione** `.py`. Tuttavia, questa estensione non è obbligatoria. L'importante è che il codice sia in **formato testuale** (anche un file `.txt` va bene) e che segua le **regole** del linguaggio di programmazione utilizzato.
Nonostante non sia obbligatoria, l'estensione .py viene **sempre** utilizzata perché indica immediatamente, senza dover aprire il file, che il contenuto sarà codice Python. Inoltre, una volta aperto il file, aiuta l'IDE (VSCode o chi per lui) a riconoscere il linguaggio di programmazione, attivando gli strumenti adatti.

Per eseguire un programma Python, basta eseguire il comando `python <nome_file>.py` e osservare l'esecuzione sequenziale delle istruzioni nel codice sorgente.
I codici Python, come quelli di altri linguaggi, possono diventare complessi e contorti. È consigliabile seguire una **convenzione di strutturazione**: posizionare le **importazioni** all'inizio, definire le **funzioni** e scrivere il **codice procedurale alla fine**. Questo approccio aiuta a mantenere l'**ordine** nel codice e renderlo più comprensibile.





Elementi di base di Python

Ogni linguaggio possiede delle **regole sintattiche** alla base di tutto il proprio funzionamento. Queste regole sono fondamentali e, dal momento che Python è un **linguaggio formale**, il non rispettarle causa degli **errori di sintassi**. Oltre alle regole sintattiche basilari, i linguaggi di programmazione associano un **significato** ad ogni **simbolo**. Queste due caratteristiche, insieme, permettono di scrivere del codice che "funziona". In questa slide e nelle successive introduciamo gli elementi sintattici fondamentali di Python, che poi approfondiremo andando avanti, arricchendoli con le annesse regole sintattiche e i significati.

- **Valori**: sono i veri e propri **dati** con cui il linguaggio effettua elaborazioni. Ogni valore ha un **tipo**.
- **Variabili**: sono utilizzate per **memorizzare i valori** e vengono identificate attraverso uno specifico **nome**.
- **Operatori**: sono utilizzati per effettuare **operazioni** (assegnamento, operazioni matematiche o confronti).
- **Funzioni**: blocchi di codice che possono essere **riutilizzati**.
- **Classi**: per ora evitiamo anche la loro introduzione, ci arriveremo molto più avanti.
- **Errori**: per ora possiamo vederli come dei testi mostrati nell'output che **spiegano un problema incontrato dall'interprete** e indicano da **dove è stato generato**.
- **Commento**: una o più righe di codice che vengono **ignorate** dall'interprete e sono utilizzate per **annotare** il codice o fornire **spiegazioni**.
- **Istruzione** (**statement**): una riga di codice che fa qualcosa (es.: `x = 5`, `print("ciao")`, `x = 2 + 2`).

DISCLAIMER:

Da qui in poi, le slide, salvo casi particolari, saranno solo un leggero supporto agli esempi di codice vero e proprio che vedremo insieme.

Commenti

Righe di codice ignorate dall’interprete che vengono utilizzate per scrivere annotazioni in linguaggio naturale o fornire spiegazioni del codice. Molto utili e fondamentali per aiutare voi stessi del futuro o altri programmatori a comprendere appieno il codice scritto.

```
# Questo è un commento a linea singola (single-line comment)
# Questo è un altro commento a linea singola
# Questi commenti non vengono interpretati
print("ciao") # Questo è un altro commento a linea singola, posizionato dopo il codice

"""
Questo è un commento a linea multipla (multi-line comment)
Tutto ciò che è racchiuso tra queste due virgolette è un commento a linea multipla
Anche questo non viene interpretato
print("ciao") non verrà eseguito perché è all'interno di un commento
"""
```

Tipi di dati

I **tipi di dati** sono la tipologia di ogni valore. Esistono **cinque** tipi di dati **principali** in Python, e alcuni secondari che scopriremo andando avanti.

```
5 # Questo è un numero intero (int)

3.14 # Questo è un numero decimale (float)

"ciao" # Questa è una stringa di testo (str)

True # Questo è un booleano vero (bool)
False # Questo è un booleano falso (bool)

None # Questo è None
```

Per **verificare** il tipo di dato di un valore, Python offre la funzione built-in `type()`. Questa funzione può essere printata e vedremo il tipo del valore.

```
print(type(5))
# Stampa <class 'int'>
print(type(3.14))
# Stampa <class 'float'>
print(type("ciao"))
# Stampa <class 'str'>
print(type(True))
# Stampa <class 'bool'>
print(type(None))
# Stampa <class 'NoneType'>
```

Operatori

Gli **operatori** consentono assegnazioni, operazioni matematiche e confronti e **restituiscono un valore**. Nei linguaggi di programmazione esistono operatori **unari**, **binari** o **ternari**, a seconda del **numero di operandi** con cui lavorano (es.: l'addizione è binaria poiché coinvolge due operandi). Python ha pochi operatori unari, molti binari e un operatore ternario, che esamineremo più avanti poiché non è un operatore tradizionale.

```python
# --== Operatori binari (binary operators) ==--
# Operatore "+"
5 + 3 # L'operatore "+" restituisce il risultato dell'addizione
5 - 3 # L'operatore "-" restituisce il risultato della sottrazione
5 * 3 # L'operatore "*" restituisce il risultato della moltiplicazione
5 / 3 # L'operatore "/" restituisce il risultato della divisione
5 % 3 # L'operatore "%" restituisce il resto della divisione
5 ** 3 # L'operatore "**" restituisce il risultato della potenza
```

```python
# --== Operatori di assegnamento (assignment operators) ==--
# Operatore "="
a = 5 # L'operatore "=" assegna il valore 5 alla variabile "a"
# Operatore "+="
a += 5 # L'operatore "+=" aggiunge 5 al valore della variabile "a"
# Esistono altri operatori di assegnamento combinati con gli altri
# operatori binari, ma li omettiamo per brevità (es.: operatore "/=")
```

```python
# --== Operatori di confronto (comparison operators) ==--
5 == 3 # Restituisce True se i valori sono uguali
5 != 3 # Restituisce True se i valori non sono uguali
5 > 3 # Restituisce True se il primo valore è maggiore del secondo
5 < 3 # Restituisce True se il primo valore è minore del secondo
5 >= 3 # Restituisce True se il primo valore è maggiore o uguale del secondo
5 <= 3 # Restituisce True se il primo valore è minore o uguale del secondo
```

```python
# --== Operatori unari (unary operators) ==--
# Operatore "-" unario
-5 # L'operatore "-" unario restituisce il numero opposto
# Operatore "~" unario
~5 # L'operatore "~" unario inverte il bit a destra
# Operatore "+" unario
+5 # L'operatore "+" unario restituisce lo stesso numero
# Operatore "not" unario
not True # L'operatore "not" unario restituisce False
not False # L'operatore "not" unario restituisce True
```

```python
# --== Operatori logici (logical operators) ==--
# Operatore "and"
True and False # Restituisce False
True and True # Restituisce True
# Operatore "or"
True or False # Restituisce True
True or True # Restituisce True
False or False # Restituisce False
# Operatore "not"
not True # Restituisce False
not False # Restituisce True
```

Cos’è Python

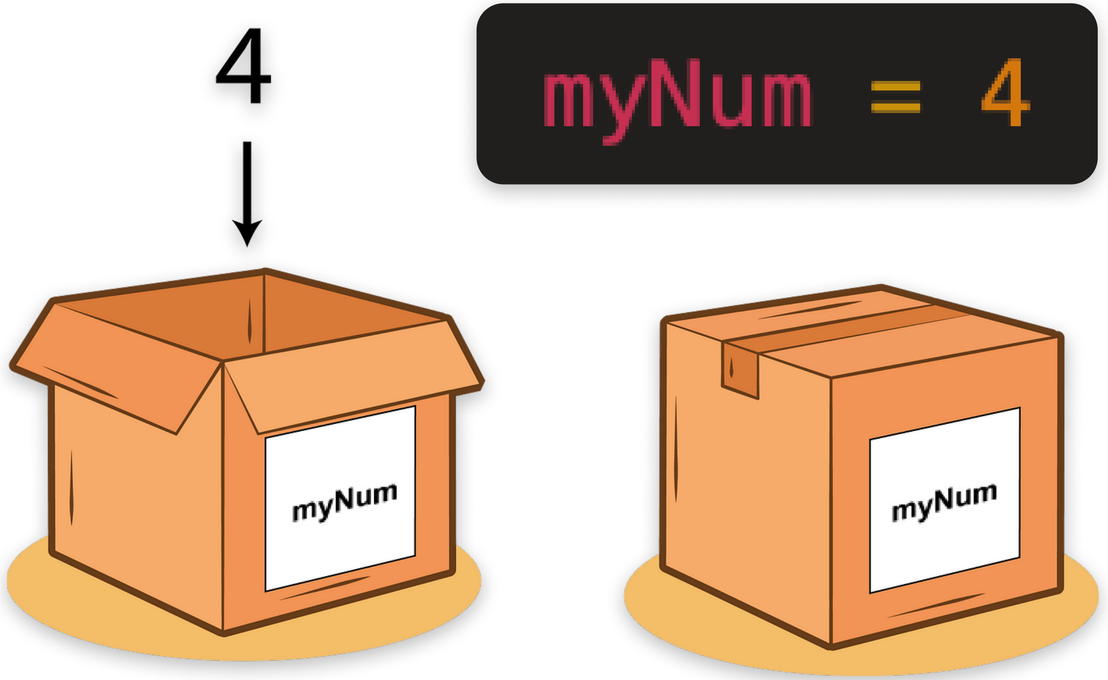
Installazione degli strumenti

Sintassi di base di Python

→ **Variabili e funzioni**

Variabili

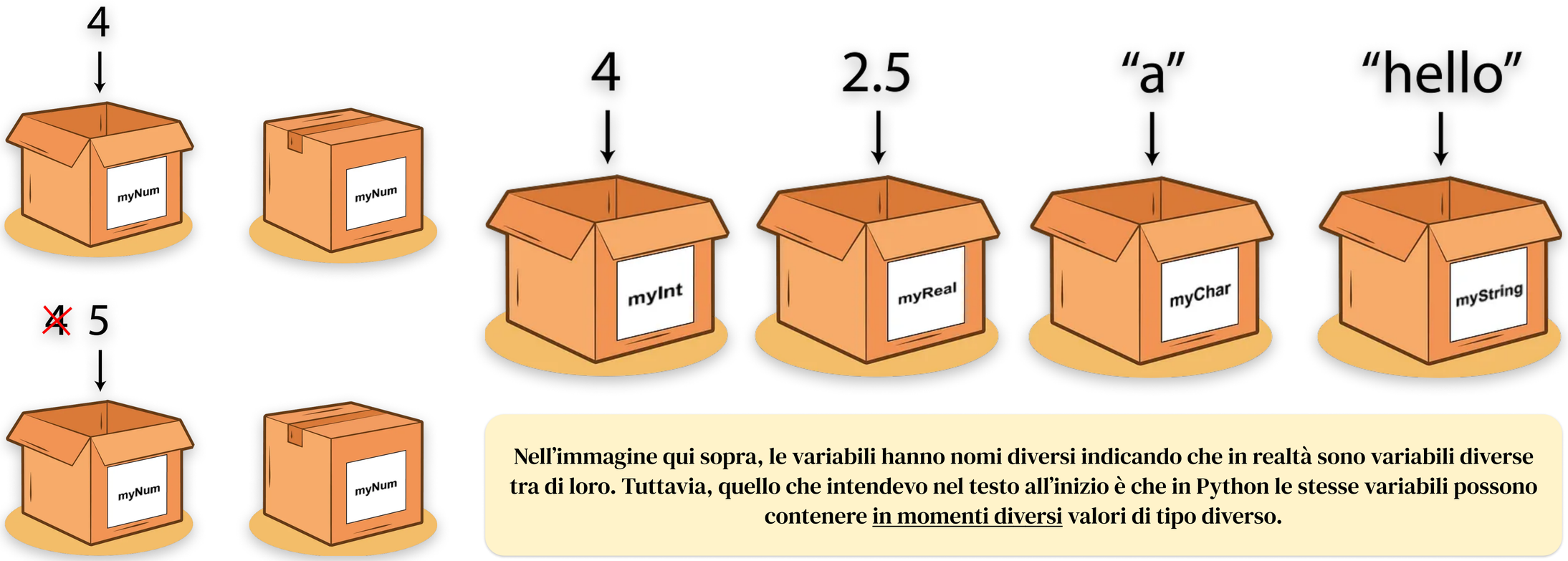
Le **variabili** sono uno dei concetti fondamentali dell'intera programmazione. Possiamo vedere le variabili con due "metafore".



Le variabili, concettualmente, rappresentano dei **nomi per dei valori**, che quindi inserite nel codice vengono "sostituite" con il valore al loro interno svolgendo lo stesso compito.

Riassegnamento e tipi di dati multipli

In Python, tutte le variabili possono essere **riassegnate** quando si vuole (la riassegnazione non implica la ricreazione) con anche **tipi di dati differenti** rispetto a quanto specificato in fase di dichiarazione.



Nell’immagine qui sopra, le variabili hanno nomi diversi indicando che in realtà sono variabili diverse tra di loro. Tuttavia, quello che intendevo nel testo all’inizio è che in Python le stesse variabili possono contenere in momenti diversi valori di tipo diverso.

Funzioni

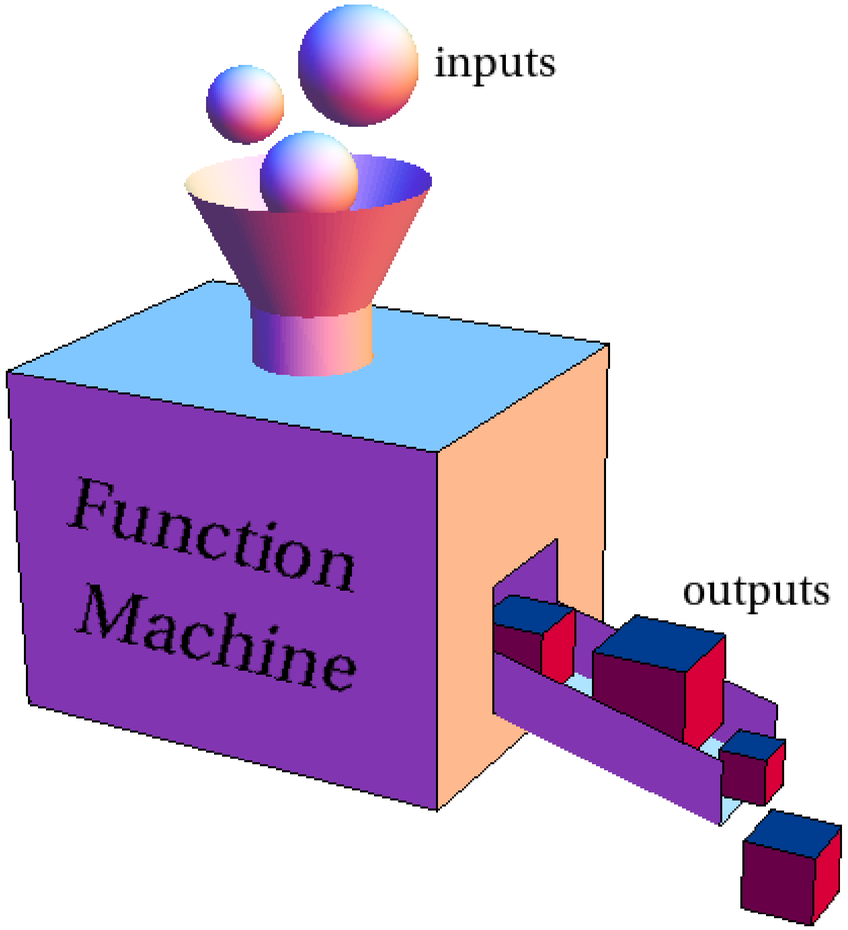
Le **funzioni** sono l'elemento fondamentale della programmazione **procedurale**. Possiamo vedere le funzioni come una scatola che prende qualcosa in input (argomenti o parametri), svolge delle elaborazioni al suo interno e **restituisce** qualcosa come risultato.

A livello tecnico, senza entrare nei dettagli, una funzione è un **blocco di codice** che può essere riutilizzato.

> **Un blocco di codice, in Python, è un insieme di righe di codice indentate con una o più tabulazioni all'inizio.**

Una funzione viene creata con la seguente **sintassi**:

```
85  def nome_funzione(lista, argomenti):
86      # Blocco di codice in cui si fa qualcosa
87      return qualcosa
```



Utilizzo delle funzioni

Una volta dichiarata una funzione, gli **argomenti** definiti nella sua intestazione (**signature**, **firma**), diventano delle variabili che possono essere utilizzate al suo interno, ma che sono temporaneamente **vuote**.

Non è obbligatorio inserire degli argomenti. Infatti possono non essere necessari per funzioni che fanno qualcosa che non deve lavorare con dei dati che gli passiamo noi.

Una funzione può essere chiamata scrivendo il suo **nome** seguito dalle **parentesi** con gli **argomenti** che vogliamo passare alla funzione.

Le funzioni sono molto utili per non riscrivere 10mila volte le stesse cose. Supponiamo, ad esempio, di avere del codice che, presi due numeri, li somma, li moltiplica e poi somma il risultato di quelle due operazioni. Supponiamo che dobbiamo utilizzare questo codice in diverse parti del nostro programma. Allora possiamo scrivere una funzione che svolge queste operazioni in modo da riutilizzarla tutte le volte che ci serve.

```
89  def somma_moltiplica_e_somma(a, b):
90      somma = a + b # Qui sommiamo i due numeri
91      moltiplicazione = a * b # Qui moltipliciamo i due numeri
92      risultato = somma + moltiplicazione # Qui sommiamo i due risultati
93      return risultato # Volendo si poteva anche scrivere return somma + moltiplicazione oppure return (a + b) + (a * b)
```

Utilizzo delle funzioni

Una volta create, le funzioni possono essere **chiamate** e possiamo concettualmente pensare che al posto della chiamata venga sostituito il codice presente nella funzione.

```
 95   x = 2
 96   y = 3
 97   print(somma_moltiplica_e_somma(x, y))
 98   # IL CODICE QUI SOPRA È EQUIVALENTE A:
 99   somma = x + y
100   moltiplicazione = x * y
101   risultato = somma + moltiplicazione
102   print(risultato)
```

Notiamo che possiamo passare come argomento alla funzione **altre variabili** che abbiamo già definito. Potremmo anche passare i **valori**, facendo: `somma_moltiplica_e_somma(2, 3)` Quindi, non è necessario che le variabili passate come argomenti abbiano lo stesso nome delle variabili definite durante la creazione della funzione.

Esistono diverse **tipologie di argomenti**, ma li vedremo con degli esempi insieme. Per ora, direi che abbiamo introdotto i concetti fondamentali di Python. Tuttavia, il codice che possiamo scrivere è ancora troppo lineare, per questo vedremo, successivamente, la **programmazione strutturata**.

Bonus: convenzioni

Quando si scrive del codice, esistono delle **convenzioni** (regole non scritte) che andrebbero rispettate per scrivere del **codice pulito**, **coerente** e più **leggibile**. Ogni linguaggio di programmazione ha le sue convenzioni, in Python abbiamo le seguenti:

Nomi:

- **Variabili e funzioni**: `snake_case` (es. `mia_funzione`)
- **Classi**: PascalCase (es. `MiaClasse`)
- **Costanti**: `SCREAMING_SNAKE_CASE` (es. `PI_GRECO`)

Commenti:

- *Brevi*: due spazi dopo il codice, # seguito da uno spazio
- *Descrittivi*: sopra il codice, chiari e dettagliati

Spaziatura:

- Una linea vuota tra funzioni; due linee vuote dopo importazioni
- Niente spazi tra parentesi e contenuto

Indentazione:

- 4 spazi per livello