

```
In [ ]: from notebook.services.config import ConfigManager
cm = ConfigManager()
cm.update('livereveal', {
    'width': 1440,
    'height': 900,
    'scroll': True,
})
```



Python from zero to hero

Lezione 0

Docente: Luca Zomparelli

Cenni storici

Ideatore di Python Guido Van Rossum (Olanda 1956)



Guido Van Rossum crea all'inizio degli anni 90 Python

- Guido ha lavorato alla Google fino al 2012 poi alla Dropbox e nel 2020 nella Developer Division alla Microsoft
- Obbiettivi di Python: semplice, Open Source, linguaggio naturale, tempi di sviluppo brevi
- Dittatore benevolo della comunità, quest'anno ha lasciato la guida

Il nome

Python deriva dalla passione di Guido per i Monti Python



Perchè imparare Python?

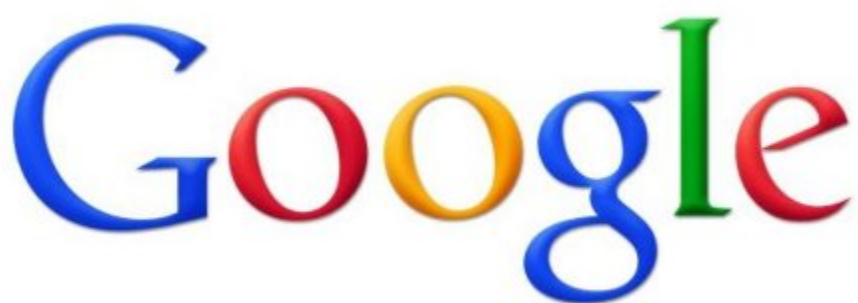
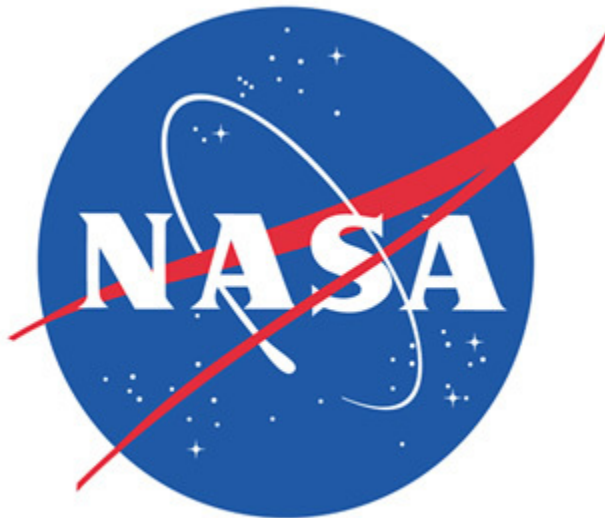


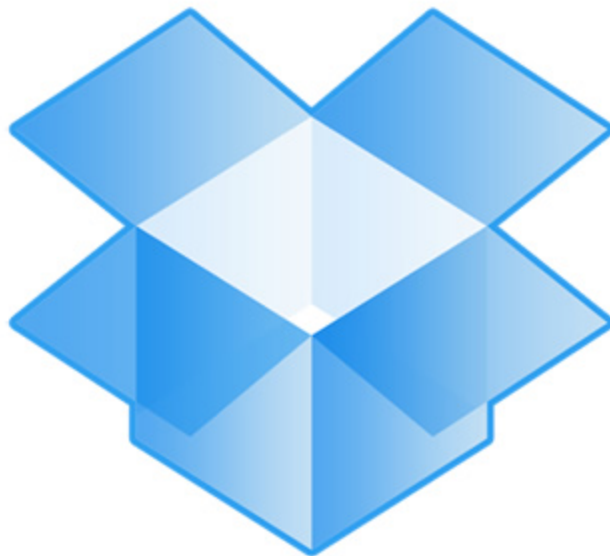
- E' Open Source
- Multi piattaforma
- Sintassi semplice
- Linguaggio di alto livello
- È indirizzato agli oggetti
- Estensibile / incorporabile
- Comunità vastissima
- Numerosissime librerie
- Varietà di utilizzi
- Elevate prestazioni
- È usato da aziende importanti

Chi usa Python



Alcuni degli utilizzatori principali





Dropbox

| ... e molte altre.

Versioni di Python

Versione 2.7 o 3.x



Quale scegliere?

Noi useremo la 3.10

Quali librerie ci sono per Python

Alcune fra le principali librerie



Il linguaggio



Filosofia di Python

- Programmazione indirizzata agli oggetti
- Ambiente aperto (si può accedere a tutto)
- Sintassi semplice e ordinata
- Ricco di convenzioni e con poche restrizioni (spoiler)
- Batterie incluse (<https://docs.python.org/3/library/index.html>)
- Moduli per tutto super ottimizzati

Python interattivo

Molto comodo per la sperimentazione

- IDLE
- Riga di comando
- Jupyter

Per questa presentazione ho utilizzato Jupyter, ma voi userete IDLE, per semplicità.

Partiamo da un classico



`print()`

Questa funzione è la differenza più evidente fra la versione 2.7 e le 3.x.

Alcuni esempi

```
In [1]: print("Test di output!")
```

Test di output!

```
In [2]: print("Testiamo", "un output", "multiplo", "con i numeri", 3, " e ", 2.7)
```

Testiamo un output multiplo con i numeri 3 e 2.7

```
In [3]: print('Prima riga', end='')  
print(' e ancora la prima.')
```

Prima riga e ancora la prima.

Variabili

In Python, non serve dichiarare le variabili. Ogni volta che si assegna un valore ad una "etichetta" viene automaticamente creata una variabile del tipo adeguato:

```
In [4]: numero = 3  
decimale = 7.5  
frase = "Ciao Mondo!!"  
  
print(type(numero).__name__, numero)  
print(type(decimale).__name__, decimale)  
print(type(frase).__name__, frase)  
  
int 3  
float 7.5  
str Ciao Mondo!!
```

Riutilizzo delle variabili

Se ad un'etichetta viene riassegnato un nuovo valore, il tipo può cambiare (sconsigliato)



```
In [5]: numero = "Questo non è un numero"  
frase = 42  
  
print(type(numero).__name__, numero)  
print(type(frase).__name__, frase)  
  
str Questo non è un numero  
int 42
```

In Python TUTTO è un oggetto



Vediamo alcuni esempi

```
In [6]: a = int('10')  
print(a.bit_length())  
a = int('65000')  
print(a.bit_length())  
  
4  
16
```

```
In [7]: b = "facciamo una prova."  
print(b.count('a'))  
print(b.split())  
print(b.upper())  
print(b.title())
```

```
4
['facciamo', 'una', 'prova.']
FACCIAMO UNA PROVA.
Facciamo Una Prova.
```

Aggiungiamo un po' di interattività

Chiedere input all'utente

```
In [8]: # -*- coding: utf-8 -*-
# -*- coding: cp1252 -*-
nome = input("Come ti chiami? ")

print("Ciao " + nome + ", Python è cool!")
```

```
Come ti chiami? Luca
Ciao Luca, Python è cool!
```

Formattazione stringhe

Esistono alcune modalità per la formattazione delle stringhe

```
In [9]: modo1 = "Numero di %s, %d" % ("prova1", 1)
modo2 = "Numero di {}, {}".format("prova2", 2)
modo3 = "Numero di {1}, {0}".format(3, "prova3")

numero, testo = 4, "prova4"
modo4 = f"Numero di {testo}, {numero}"

print(modo1)
print(modo2)
print(modo3)
print(modo4)
```

```
Numero di prova1, 1
Numero di prova2, 2
Numero di prova3, 3
Numero di prova4, 4
```

Due note sulle virgolette per le stringhe

Si possono utilizzare diversi tipi di virgolette:

- 'primo'
- "secondo"
- '''terzo'''
- """quarto"""

```
In [11]: print('Primo testo con l\'apostrofo\n')
print("Secondo testo con l'apostrofo\n")
print("""Terzo test
con degli invii a capo.
""")
print(''''Quarto test
con degli invii a capo.'''')
```


Primo testo con l'apostrofo

Secondo testo con l'apostrofo

Terzo test
con degli invii a capo.

Quarto test
con degli invii a capo.

prefissi letterali

Sono alcune lettere da anteporre alle costanti stringa, per specificarne alcuni comportamenti o rappresentazioni. Vediamo alcuni esempi ...

binary

```
In [12]: # rappresentazione binaria
binaria = b"Prova di stringa binaria\n"
print(binaria)
```

b'Prova di stringa binaria\n'

```
In [13]: # Decodifica di una buffer
print(binaria.decode())
```

Prova di stringa binaria

```
In [14]: # Codifica di una stringa
stringa = "Prova di stringa binaria\n"
print(stringa.encode("utf-16"))
```

b'\xff\xfeP\x00r\x00o\x00v\x00a\x00 \x00d\x00i\x00 \x00s\x00t\x00r\x00i\x00n\x00g\x00a\x00 \x00b\x00i\x00n\x00a\x00r\x00i\x00a\x00\n\x00'

raw

```
In [28]: # Rappresentazione di stringa 'normale'
s1 = "Questa è una stringa \n ed è normale!"
print(s1)
```

Questa è una stringa
ed è normale!

```
In [31]: # Rappresentazione di stringa 'raw'
s2 = r"Questa è una stringa \n ed è raw!"
print(s2)
```

Questa è una stringa \n ed è raw!

Un pessimo esempio di stringa raw

```
In [32]: # Un pessimo esempio di stringa raw
s3 = r"C:\Users\Luca"
print(s3)
```

C:\Users\Luca

... e la portabilità?!



formatted

(... ancora sulla formattazione dalla versione 3.6)

```
In [15]: # inizializzo alcune variabili
variabile_numerica = 5
variabile_stringa = "prova5"

modo5 = f"Numero di {variabile_stringa}, {variabile_numerica}"

print(modo5)
```

Numero di prova5, 5

slice

Lo slice in Python consente di prendere porzioni di oggetti "lista", in maniera semplice e molto leggibile

```
In [16]: testo = "Questa è una frase da tagliare."
# Questa è una frase da tagliare.
# ^0
print(testo[0])
```

Q

```
In [17]: # Questa è una frase da tagliare.
#                                     ^-1
print(testo[-1])
```

.

```
In [18]: # Questa è una frase da tagliare.
#      ^3                               ^-4
print(testo[3:-4])
```

sta è una frase da tagli

```
In [19]: # Questa è una frase da tagliare.
#      ^3           ^12
print(testo[3:12])
```

sta è una

```
In [20]: # Questa è una frase da tagliare.
#      ^-14      ^24
print(testo[-14:24])
```

e da ta

Tuple

Oggetti immutabili (lo sono anche le stringhe)

```
In [21]: var_tupla = (1, 2, "terzo")
print(var_tupla)
```

```
var_tupla[1] = 77
```

```
(1, 2, 'terzo')
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In [21], line 3
      1 var_tupla = (1, 2, "terzo")
      2 print(var_tupla)
----> 3 var_tupla[1] = 77

TypeError: 'tuple' object does not support item assignment
```

```
In [22]: var_tupla = var_tupla[:1] + (77,) + var_tupla[2:]
print(var_tupla)

(1, 77, 'terzo')
```

Liste

```
In [23]: var_lista = ['a', 'b', 3]
print(var_lista)
var_lista[1] = "secondo"
print(var_lista)
var_lista.append('aggiunto')
print(var_lista)

['a', 'b', 3]
['a', 'secondo', 3]
['a', 'secondo', 3, 'aggiunto']
```

Dizionari

```
In [25]: var_voti = {'Luca': 7, 'Max': 8, 'Maria': 7}
print(var_voti)
print(f"Il voto di Luca è {var_voti['Luca']}")
var_voti['Silvia'] = 8.5
print(var_voti)

{'Luca': 7, 'Max': 8, 'Maria': 7}
Il voto di Luca è 7.
{'Luca': 7, 'Max': 8, 'Maria': 7, 'Silvia': 8.5}
```

Set

```
In [26]: primo = {'Luca', 'Luigi', 'Maria'}
secondo = {'Andrea', 'Luigi', 'Silvia'}
print(primo)
print(secondo)
print(primo | secondo)
print(primo - secondo)
print(primo & secondo)

{'Maria', 'Luca', 'Luigi'}
{'Silvia', 'Andrea', 'Luigi'}
{'Luigi', 'Maria', 'Silvia', 'Luca', 'Andrea'}
{'Maria', 'Luca'}
{'Luigi'}
```

Controllo di flusso



Python, ovviamente, fornisce diverse dichiarazioni per controllare il flusso dei programmi, come molti altri linguaggi, anche se con alcuni "colpi di scena".

(ricordatevi di non incrociare mai i flussi!)

if

```
In [29]: numero_stringa = input("Inserisci un numero: ")
numero = int(numero_stringa)

if(numero > 10):
    print(f"{numero} è maggiore di 10.")
elif(numero < 10):
    print(f"{numero} è minore di 10.")
else:
    print(f"Hai inserito 10.")
```

```
Inserisci un numero: 10
Hai inserito 10.
```

Due note sulle valutazioni

```
In [30]: pippo = True
print("True è", "True" if pippo else "False")

pippo = False
print("False è", "True" if pippo else "False")
print("'prova' è", "True" if "prova" else "False")
print("' ' è", "True" if "" else "False")
print("[]" è", "True" if [] else "False")
print("[1, 2, 3] è", "True" if [1, 2, 3] else "False")
```

```
True è True
False è False
'prova' è True
' ' è False
[] è False
[1, 2, 3] è True
```

for

```
In [31]: for i in range(5):
        print(i)
```

```
0
1
2
3
4
```

break

```
In [32]: lista_test = ['Margherita', 'Tulipano', 'Orchidea', 'Rosa', 'Rododendro']
for fiore in lista_test:
```

```
if fiore == 'Rosa':
    print('Trovato!')
    break
else:
    print(fiore)
```

Margherita
Tulipano
Orchidea
Trovato!

continue

```
In [35]: for i in range(10):
        if i % 2 == 0:
            print(f'{i}: numero pari')
            continue
        print(f'{i}: numero dispari')
```

0: numero pari
1: numero dispari
2: numero pari
3: numero dispari
4: numero pari
5: numero dispari
6: numero pari
7: numero dispari
8: numero pari
9: numero dispari

else

```
In [ ]: cicli = input('Quanti cicli? ')
        for i in range(int(cicli)):
            if i > 5:
                break
            else:
                print(f'Ciclo {i}')
        else:
            print('Ho eseguito tutti i cicli.')
```

pass

```
In [ ]: if 3 > 2:
        pass
        else:
            print('Test')
```

while

```
In [38]: variabile_test = True
        i = 0
        while variabile_test:
            if i > 3:
                variabile_test = False
            else:
```

```
        i += 1
    print(f'i = {i}, variabile_test = {variabile_test}')

print('Ciclo terminato')

i = 1, variabile_test = True
i = 2, variabile_test = True
i = 3, variabile_test = True
i = 4, variabile_test = True
i = 4, variabile_test = False
Ciclo terminato
```

match (structural pattern matching)

dalla versione 3.10

```
In [5]: status = 404

match status:
    case 400:
        print("Bad request")

    case 404:
        print("Not found")

    case 418:
        print("I'm a teapot")

    case _:
        print("Unknown!!")
```

Not found

```
In [1]: status = 400

match status:
    case 400:
        print("Bad request")

    case 404:
        print("Not found")

    case 418:
        print("I'm a teapot")

    case _:
        print("Unknown!!")
```

Bad request

assegnazione "tricheco" (walrus)

dalla versione 3.10

```
In [2]: if(n := 10):
        print(f"Valore di {n}.")
```

Valore di 10.

Funzioni

```
In [4]: def somma(a, b):  
        """Somma due numeri"""  
        return a + b, 5  
  
        print(somma(1, 2))  
        print(somma(7, 33))
```

```
(3, 5)  
(40, 5)
```

I parametri alle funzioni sono passati sempre per riferimento ...

... anche se non sembra!

Esempio 1

```
In [5]: uno, due = 1, 2  
        print(uno, due)  
  
        def cambia(primo, secondo):  
            """Scambia due valori"""  
            primo, secondo = secondo, primo  
            print(primo, secondo)  
  
        cambia(uno, due)  
        print(uno, due)
```

```
1 2  
2 1  
1 2
```

Esempio 2

```
In [6]: uno, due = [1], [2]  
        print(uno, due)  
  
        def cambia(primo, secondo):  
            """Scambia il primo elemento delle liste"""  
            primo[0], secondo[0] = secondo[0], primo[0]  
            print(primo, secondo)  
  
        cambia(uno, due)  
        print(uno, due)
```

```
[1] [2]  
[2] [1]  
[2] [1]
```

Classi

```
In [7]: class automobile():  
        """Rappresenta il concetto di auto"""  
        colore = 'Rosso'  
        alimentazione = 'Benzina'  
  
        def accendi(self):  
            """Accende l'auto"""
```

```

        self.accesa = True

    def spegni(self):
        """Spegne l'auto"""
        self.accesa = False

```

```

In [8]: macchina = automobile()
print(macchina.colore)
print(macchina.alimentazione)
print(macchina.accesa)

```

Rosso
Benzina

```

-----
AttributeError                                Traceback (most recent call last)
Cell In [8], line 4
      2 print(macchina.colore)
      3 print(macchina.alimentazione)
----> 4 print(macchina.accesa)

AttributeError: 'automobile' object has no attribute 'accesa'

```

Membri statici

```

In [9]: a = automobile()
b = automobile()
print(a.colore, b.colore)
automobile.colore = 'Bianco'
print(a.colore, b.colore)

a.colore = 'Verde'
print(a.colore, b.colore)

```

Rosso Rosso
Bianco Bianco
Verde Bianco

Una classe migliore

```

In [10]: class automobile():
    """Rappresenta il concetto di auto"""

    def __init__(self, colore, alimentazione):
        """Il costruttore"""
        self.colore = colore
        self.alimentazione = alimentazione
        self.accesa = False

    def accendi(self):
        """Accende l'auto"""
        self.accesa = True

    def spegni(self):
        """Spegne l'auto"""
        self.accesa = False

```

```

In [11]: macchina = automobile('verde', 'GPL')
print(macchina.colore)
print(macchina.alimentazione)
print(macchina.accesa)

```



```
macchina.accendi()  
print(macchina.accesa)
```

```
verde  
GPL  
False  
True
```

Metodi e attributi privati



Esempio di dichiarazione

```
In [12]: class automobile():  
        """Rappresenta il concetto di auto"""  
  
        def __init__(self):  
            """Il costruttore"""  
            self.__accesa = False  
  
        def accendi(self):  
            """Accende l'auto"""  
            self.__accesa = True  
  
        def spegni(self):  
            """Spegne l'auto"""  
            self.__accesa = False  
  
        def stato(self):  
            """Dice se è accesa"""  
            return self.__accesa
```

Esempi d'uso

L'elemento `__accesa` è raggiungibile da fuori?

```
In [13]: print(macchina.__accesa)
```

```
-----  
AttributeError                                Traceback (most recent call last)  
Cell In [13], line 1  
----> 1 print(macchina.__accesa)  
  
AttributeError: 'automobile' object has no attribute '__accesa'
```

Proviamo un uso migliore

```
In [14]: macchina = automobile()  
print(macchina.stato())  
  
macchina.accendi()  
print(macchina.stato())  
  
macchina.spegni()  
print(macchina.stato())
```

False

True

False

L'elemento `__accesa` è raggiungibile da fuori?

Ricco di convenzioni e con poche restrizioni

(spoiler)

```
In [15]: print(macchina._automobile__accesa)
```

False