Python

Python è un linguaggio di programmazione ad alto livello, interpretato, orientato agli oggetti e con semantica dinamica.

Variabili

```
x = 5
y = "Hello, World!"

print(x) # stampa 5
print(y) # stampa Hello, World!
```

Tipi di dati

```
x = 5 # int
y = "Hello, World!" # str
z = 20.5 # float
w = True # bool
```

Input

```
x = input("Enter your name: ")
print("Hello, " + x) # Hello, [input]

x = int(input("Enter a number: ")) # converte l'input in un intero
print(x)
```

Stringhe

```
a = "Hello, World!"
print(a[1]) # stampa la seconda lettera
print(a[2:5]) # stampa dal terzo al quinto carattere
print(len(a)) # lunghezza della stringa
print(a.lower()) # tutto minuscolo
print(a.upper()) # tutto maiuscolo
print(a.replace("H", "J")) # sostituisce H con J
print(a.split(",")) # divide la stringa in una lista
print(a.strip()) # rimuove gli spazi bianchi all'inizio e alla fine

txt = "C'era una volta"
x = "olt" in txt # True se "olt" è presente in txt

a = "Hello"
b = "World"
c = a + b # HelloWorld

age = 36
txt = f"My name is John, and I am {age}" # {} è un segnaposto
```

Funzioni

```
def function():
  print("Hello, World!")

function()


def function2(name):
  print("Hello, " + name)

function2("John")


def function3(x):
  return 5 * x

print(function3(3)) # 15
```

Cicli

```
for x in range(6): # da 0 a 6
  print(x)

for x in range(2, 6): # da 2 a 6
    print(x)

for x in range(2, 30, 3): # da 2 a 30 con step 3
    print(x)

for x in range(len(fruits)): # per ogni elemento della lista
    print(fruits[x])

while x < 6:
    print(x)
    x += 1

while True: # ciclo infinito
    print(x)
    x += 1
    if x == 6:
        break # esce dal ciclo
    if x == 3:
        continue # salta alla prossima iterazione
```

Condizioni

```
if 5 > 2:
  print("Five is greater than two!")

if 2 < 5:
    print("Two is less than five!")
elif 2 == 5: # else if
    print("Two is equal to five!")
else:
    print("Two is greater than five!")

switch(x):
    case 1: # if x == 1
        print("One")
        break # esce dallo switch altrimenti eseguirebbe anche il caso
successivo
    case 2: # if x == 2
        print("Two")
        break
    default: # else
        print("Something else")
```

Strutture Dati

Liste (Array)

Una lista è una collezione **ordinata** e **cambiabile** di **elementi**. In Python, le liste sono scritte tra parentesi quadre. Sono praticamente degli array.

```
lista = ["apple", "banana", "cherry"]

for x in lista: # per ogni elemento della lista
  print(x)

if "apple" in lista: # se l'elemento è presente nella lista
    print("Yes, 'apple' is in the fruits list")

lista.append("orange") # aggiunge un elemento alla lista
lista.remove("banana") # rimuove un elemento dalla lista
lista.pop(1) # rimuove l'elemento in posizione 1
lista.pop() # rimuove l'ultimo elemento
lista.clear() # svuota la lista
```

Set

Un set è una collezione **non ordinata** e non indicizzata di **elementi unici**. In Python, i set sono scritti tra parentesi graffe.

```
set = {"apple", "banana", "cherry"}

for x in set: # per ogni elemento del set
  print(x)

set.add("orange") # aggiunge un elemento al set
set.remove("banana") # rimuove un elemento dal set
set.clear() # svuota il set
```

Dizionari

Un dizionario è una collezione **non ordinata** e **cambiabile** di **elementi**. In Python, i dizionari sono scritti tra parentesi graffe e hanno chiave e valore.

```
dizionario = {
  "id": "ID001",
  "pacco": "Pacco001",
  "indirizzo": "Via Roma 1",
  "consegnato": False
}
```

```
print(dizionario["id"]) # ID001

for x in dizionario: # per ogni chiave del dizionario
  print(x)

for x in dizionario: # per ogni valore del dizionario
    print(dizionario[x])

for x, y in dizionario.items(): # per chiave e valore
    print(x, y)

if "id" in dizionario: # se la chiave è presente nel dizionario
    print("Yes, 'id' is one of the keys in the thisdict dictionary")

dizionario["colore"] = "rosso" # aggiunge un elemento al dizionario
dizionario.pop("model") # rimuove un elemento dal dizionario
dizionario.popitem() # rimuove l'ultimo elemento
dizionario.clear() # svuota il dizionario
```

Tuple

Una tupla è una collezione **ordinata** e **non modificabile** di **elementi**. In Python, le tuple sono scritte tra parentesi tonde. Raramente usate, soprattutto per "etichette"

```
tupla = ("apple", "banana", "cherry")

for x in tupla: # per ogni elemento della tupla
  print(x)

if "apple" in tupla: # se l'elemento è presente nella tupla
    print("Yes, 'apple' is in the fruits tuple")

tupla.count("apple") # conta quante volte è presente "apple"
tupla.index("apple") # restituisce l'indice di "apple"
```

Operatori

```
x = 5
y = 3

print(x + y) # 8

x += 3 # x = x + 3
x -= 3 # x = x - 3
x *= 3 # x = x * 3
x /= 3 # x = x / 3
x %= 3 # x = x % 3
x **= 3 # x = x ** 3 (elevato a 3)
```

```
x == y # uguale
x != y # diverso
x > y # maggiore
x < y # minore
x >= y # maggiore o uguale
x <= y # minore o uguale

x and y # True se entrambi sono True
x or y # True se uno dei due è True
not x # True se x è False
```

Lambda

Le funzioni lambda sono funzioni anonime che possono avere un numero qualsiasi di argomenti, ma possono avere solo una espressione.

Sono utili per **funzioni brevi** che si possono scrivere in una sola riga.

```
x = lambda a : a + 10
print(x(5)) # 15

y = lambda a, b : a * b if a > b else a + b
# se a è maggiore di b, restituisce a * b, altrimenti a + b
# attenzione a non abusarne, possono rendere il codice difficile da leggere
```

Classe

```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x) # 5


class Person:
  def __init__(self, name, age): # costruttore
    self.name = name
    self.age = age

  def __str__(self): # metodo per stampare l'oggetto
    return f"{self.name}, {self.age}"

  def aumentaEta(self): # metodo
    self.age += 1

  def cambiaNome(self, nome): # metodo
    self.name = nome
```

```
p1 = Person("John", 36)
```

Ereditarietà

```
# Student eredita da Person dell'esempio precedente
class Student(Person): # Student eredita da Person
  def __init__(self, name, age, year): # costruttore
    super().__init__(name, age) # richiama il costruttore di Person
    self.graduationyear = year

  def __str__(self): # sovrascrive il metodo __str__
    return f"{self.name}, {self.age}, {self.graduationyear}"

x = Student("Mike", 22, 2021)
print(x) # Mike, 22, 2021
```

Moduli

Un modulo è un file contenente funzioni, classi e variabili che possono essere riutilizzate in altri file Python.

```
# mymodule.py

def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

```
# main.py
import mymodule

mymodule.greeting("Jonathan")
print(mymodule.person1["age"])
```

Esempio di main

```
# import
import mymodule

# classi
class MyClass:
  x = 5

# funzioni
def add(a, b):
  return a + b

# main
def main():
  print("Hello, World!")
  print(MyClass.x)
  mymodule.greeting("Jonathan")
  print(add(5, 3))

# esecuzione (in fondo a tutto)
if __name__ == "__main__":
  main()
```

output:

```
$ python main.py # esecuzione

Hello, World!
5
Hello, Jonathan
8
```