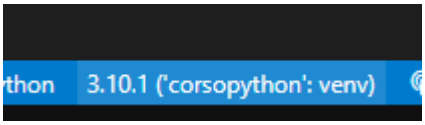
Corso Python

Capitolo 1 – Installazione

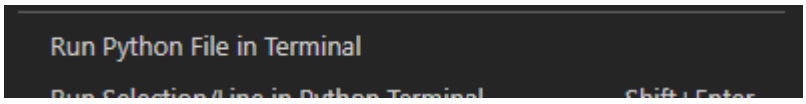
1) Installazione python
2) Spunta su path
3) Creazione di virtual environment per gestire il proprio ambiente in modo segregato

In terminale: `python -m venv nome_virtual_environment`

Selezionare il **VE** appena creato:



Run dello script avviando il terminale:



Capitolo 2 - Sintassi base

Commento:

#testo_da_commentare

Commento multi-riga:

selezione parte da commentare + ctrl + ù

Indentazione:

L’indentazione è essenziale all’esecuzione del codice

Es:

```
# Stampa messaggio
if 10 < 5:
    print('Ciao')
print('Ciao 2')
```

Figura 1 Verrà eseguito solo Ciao 2, visto che è fuori dall'if falso

---

Capitolo 3 – Variabili

N.B. : In python una variabile **non può essere soltanto dichiarata**, ma deve essere direttamente definita.

X = 6 // Ok

X // Errore

Le variabili non possono:

- Iniziare con un numero
- Contenere spazi
- Contenere trattini medi

E’ possibile usare varie tipologie di **case**:

- **camelCase**
- **PascalCase**
- **snake_case**

Definire più **variabili contemporaneamente**:

x, y, z = 23, 13, 42

x = y = z = 23

Unpack di una collection (la “destrutturazione” di JavaScript):

```
citta = ["roma","milano","napoli"]
x, y, z = citta
```

Capitolo 4 - Tipi di dato

- Vedere la tipologia di dato:

```
type(x)
```

- Non c'è bisogno di specificare il tipo di dato
- E' possibile modificare un dato in run

Esistono **9 tipologie** di dato:

1) **Stringa**
2) **Interi**
3) **Float**
4) **Boolean** (attenzione, deve essere definito in maiuscolo False/True)

5) **List** (simili agli array, ma diversi) ***
6) **Tuple** x = ("roma", "milano", "poggibonsi") ***
7) **Dictionary** x = {"key1": value1, "key2": value2} ***
8) **Set** x = {value1, value2, value3} ***
9) **Range** x = range(6)

***Collezioni di dati**

Capitolo 5 – Casting

Cosa è ?

Convertire il tipo di dato di un valore di una variabile

In **Py** (a differenza di JS) non vi è una conversione automatica, in corso, che permette di concatenare tipologie di dato differenti.

Es.

```
x = "La mia età è di "
y = 30
print(x + y)
# Errore
```

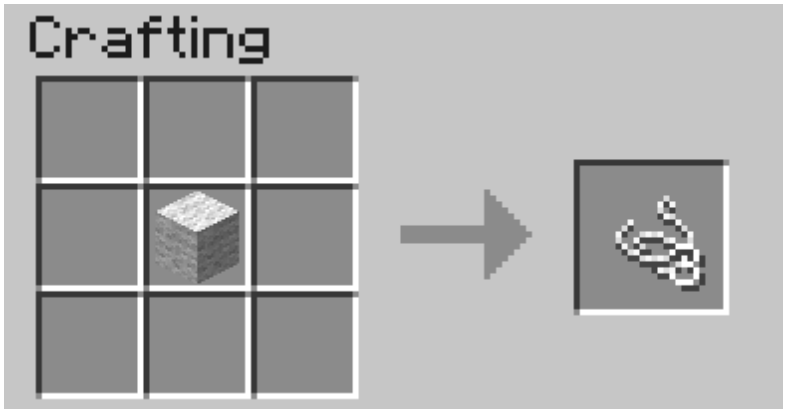
Es. (con **casting**):

```
x = "La mia età è di "
y = str(30)
print(x + y)
# Corretto
```

Tipi di casting:

- int(valore)
- float(valore)
- str(valore)

Capitolo 6 – Stringhe



Sintassi:

```
x = 'stringa'

y = "stringa"

z = """stringa riga 1
stringa riga 2
stringa riga 3
stringa riga 4
"""
```

Sono utilizzabili le **singole quotes** o le **double quotes**

Stringhe multilinea: uso delle **triple quotes**.

Le **stringhe sono trattate come degli array**, ovvero ogni carattere è considerabile un item di una lista.

Esempio di uso del metodo **len()** e dell'**indice** della stringa.

```
stringa = 'Questa è una stringa di test'

# Metodo len() per vedere lunghezza stringa
print(len(stringa))
# Indice singolo carattere
print(stringa[3])
```

Risposta in terminale:

```
28
s
```

Stampare una **sotto-stringa:**

print(stringa[:4])

-->|

print(stringa[4:])

|-->

print(stringa[1:4])

|-->|

N.B. : è possibile usare anche indici negativi

Metodi utili con le **stringhe**:

```
# Stringa in upperCase
print(stringa.upper())

# Stringa in lowerCase
print(stringa.lower())

# Stringa a cui ono rimossi spazi iniziali e finali
print(stringa.strip())

# Stringa in cui le o sono rimpiazzate con le w
print(stringa.replace('a', 'w'))
```

Concatenazione di 2 stringhe:

```
x = 'Ciao sono '
y = 'Zeobasio'

print(x + y)
```

// Ciao sono Zeobaconio

Concatenazione di stringhe con altri tipi di dato:

```
# Concatenazione
stringaConcatenata = 'Ciao, sono Agamennone e ho {} anni'

print(stringaConcatenata.format(15))
```

// Output

```
Ciao, sono Agamennone e ho 15 anni
```

// è possibile usare un **template con {}** ed il metodo **.format()** per concatenare più tipi di dato

Nota: è possibile inserire nel template {} l'indice a cui poi faranno riferimento i valori nel metodo format.

Escape dei caratteri

È possibile inserire quotes usando come escape il backslash (****)

Es.

```
stringa = 'L\'amore è un apostrofo rosa'
```

Capitolo 7 - Booleani

Figura 2 Meme stupidi ma molto cutie e dove trovarli...

```
x = False
y = True
```

N.B. : Devono essere definiti con la prima lettera maiuscola

E' possibile valutare il valore delle variabili usando **bool()**

Esempio di valori che danno sempre False

```
# FALSE:
#     bool(False)
#     bool(None)
#     bool(0)
#     bool("")
#     bool(())
#     bool([])
#     bool({})
```

Capitolo 8 – Operatori aritmetici

Operatori aritmetici:

1. **+** - Addizione
2. **-** - Sottrazione
3. ***** - Moltiplicazione
4. **/** - Divisione
5. **%** - Modulo
6. **//** - Floor division (divisione arrotondata per difetto)

Operatori di assegnamento:

1. **+=**
2. **-=**
3. ***=**
4. **/=**
5. **%=**
6. ****=**
7. **//=**

…

Es.

```
x = 15
x += 2
print(x)
```

// 17

Metodi utili:

1. **.min()** – minimo tra dei numeri
2. **.max()** – massimo tra dei numeri
3. **.abs()** – valore |assoluto| di un numero
4. **.pow()** – potenza di un numero

Es.

X = min(5,6,7)

// 5

Capitolo 9 - If

Operatori di comparazione:

- **==**
- **!=**
- **<=**
- **>=**
- **<**
- **>**

Elif ed **else:**
Componenti della sintassi che considerano altre condizioni

Esempio completo:

```
x = 5

if x > 10:
    print('5 è maggiore di 10')
elif x < 10:
    print('5 è minore di 10')
else:
    print('5 NON è minore di 10')
// 5 è minore di 10
```

Operatori logici:

AND, OR e NOT

Es.

```
if x < 10 and x > 2:
    print('Si, x è compreso')
```

N.B. : è possible riscrivere l'if sopra con un metodo più "algebrico"! 😮

```
if 2 < x < 10:
    print('Si, x è compreso!')
```

Es. 2

```
if not(x == 2):
    print('X è diverso da 2')
```

Es. di shorthand:

```
if x < 10: print('X è minore di 10')
else: print('X non è minore di 10')
```

If annidati: ovvero un if dentro un if

Es.

```
if x % 2 != 0:
    print('X è dispari')
    if (x < 10):
        print('ed è pure inferiore a 10')
else:
    print('X è pari')
```

X è dispari

ed è pure inferiore a 10

// Output

Capitolo 10 - Ciclo While

Sintassi:

```
i = 0
while i < 6:
    print(i)
    i += 1
```

// Output: 0,1,2,3,4,5

Es.

```
i = 0

while i < 6:
    o = str(i)
    print('Ciao, sono arrivato al numero ' + o)
    i += 1
```

Continue e break: ad una condizione, **continuerà** con il ciclo while, sennò **romperà** il ciclo

```
i = 0
while i < 6:
    i += 1
    if (i == 3):
        continue
    print(i)
```

o

```
i = 0
while i < 6:
    i += 1
    if (i == 3):
        break
    print(i)
```

N.B. : è possibile integrare un **else** alla fine di un ciclo while

```
i = 0
while i < 6:
    i += 1
    if (i == 3):
        continue
    print(i)
else:
    print('Il ciclo ha finito')
```

Capitolo 11 - Ciclo for

Figura 3 The Ouroboros is a mystical symbol representing a dragon devouring its own tail. It represents the eternal cycle of life or wholeness and infinity.

Sintassi:

```
collezione_di_elementi = ['elemento1', 'elemento2', 'elemento3']

for elemento in collezione_di_elementi:
    print(elemento)
```

Es. con stringa

```
stringa = 'Anguria'

for lettera in stringa:
    print(lettera)
```

// Output: **A,n,g,u,r,i,a**

Es. 2 con range

```
for x in range(6):
    print(x)
```

// Output: **1,2,3,4,5**

E' possibile usare sia **if annidati** che **continue** o **break** all'interno del ciclo for

---

Capitolo 12 - Introduzione alle Collezioni di dati

Cosa sono?

Tipologie di dati più complessi di quelli primitivi

- **List**
- **Tuple**
- **Set**
- **Dictionary**

Termini chiave:

- **Ordinato:** la collezione ha ordine ben definito e l'aggiunta di elementi non incide sull'ordine stesso

Es.:

citta = ["elemento1", "elemento2", "elemento3"]

- **Indicizzato:** possiamo accedere agli elementi tramite indice

Es.:

citta[0] // "elemento1"

- **Modificabile:** possiamo aggiungere, eliminare e cambiare elementi una volta creata la collezione
- **Immutabile:** gli elmenti non possono essere modificabili
- **Permette duplicati:** è possibile avere elementi con lo stesso valore

	Ordinata	Indicizzata	Modificabile	Immutabile	Perm. duplicati
Lista	X	X	X		X
Tuple	X	X		X	X
Set				X	
Dictionary*	X	X	X		

*Dalla versione 3.7