

# ALLA SCOPERTA DI PYTHON

INTRODUZIONE ALLA PROGRAMMAZIONE IN PYTHON

# INTRODUZIONE ALL'INFORMATICA: SISTEMA BINARIO - COS'È?

- Il sistema binario è un metodo di rappresentazione numerica
- Noi siamo abituati al sistema decimale che usa... dieci simboli!



- Il sistema binario usa invece un alfabeto di soli 2 simboli: 0 ed 1



# INTRODUZIONE ALL'INFORMATICA: SISTEMA BINARIO - PERCHÈ?

- Il sistema binario permette l'introduzione di una **logica di semplice implementazione** per un computer

0 → Falso - 0V - Vuoto - Spento ...

1 → Vero - 5V - Pieno - Acceso ...

- Vedremo che grazie ad un **preciso insieme di regole** («algebra di Boole») un computer può **eseguire qualsiasi istruzione** usando solo 0 e 1



# INTRODUZIONE ALL'INFORMATICA: SISTEMA BINARIO - PERCHÈ?

- Il sistema binario permette l'introduzione di una **logica di semplice implementazione** per un computer

0 → Falso - 0V - Vuoto - Spento ...

1 → Vero - 5V - Pieno - Acceso ...

- Vedremo che grazie ad un **preciso insieme di regole** («algebra di Boole») un computer può **eseguire qualsiasi istruzione** usando solo 0 e 1

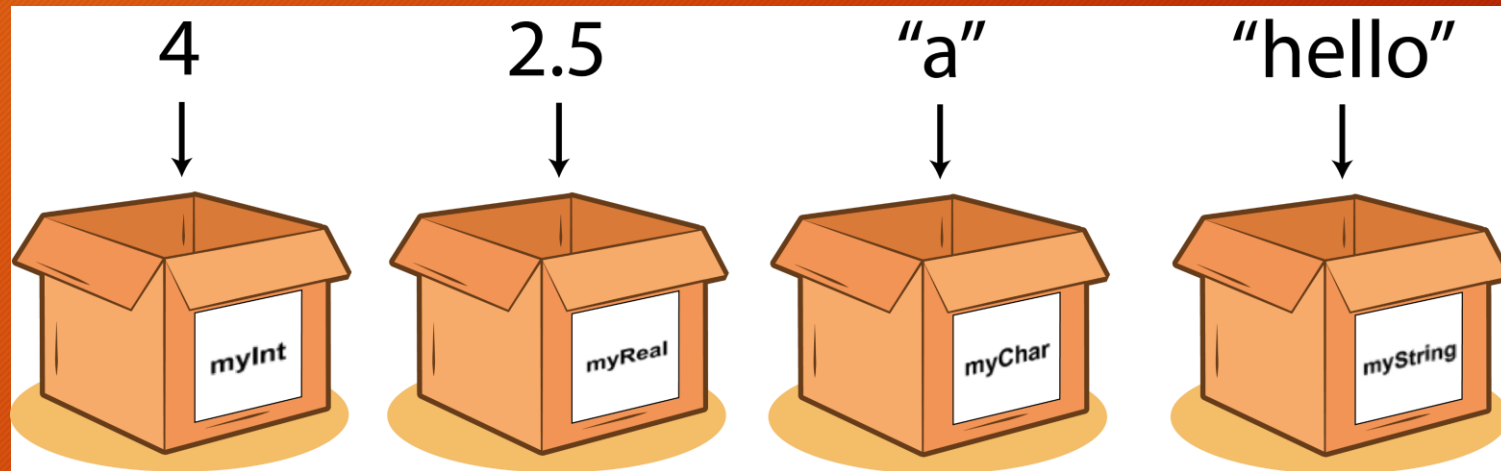
# INTRODUZIONE ALL'INFORMATICA: SISTEMA BINARIO - RAPPRESENTAZIONE NUMERICA

| Binary (Base-2) Value |   |   |   |   |   |   |   |   |   |   |       |     |      |
|-----------------------|---|---|---|---|---|---|---|---|---|---|-------|-----|------|
| 1                     | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |       |     |      |
|                       |   |   |   |   |   |   |   |   |   |   | 1024  | × 1 | 1024 |
|                       |   |   |   |   |   |   |   |   |   |   | 512   | × 1 | 512  |
|                       |   |   |   |   |   |   |   |   |   |   | 256   | × 1 | 256  |
|                       |   |   |   |   |   |   |   |   |   |   | 128   | × 1 | 128  |
|                       |   |   |   |   |   |   |   |   |   |   | 64    | × 1 | 64   |
|                       |   |   |   |   |   |   |   |   |   |   | 32    | × 0 | 0    |
|                       |   |   |   |   |   |   |   |   |   |   | 16    | × 1 | 16   |
|                       |   |   |   |   |   |   |   |   |   |   | 8     | × 1 | 8    |
|                       |   |   |   |   |   |   |   |   |   |   | 4     | × 1 | 4    |
|                       |   |   |   |   |   |   |   |   |   |   | 2     | × 0 | 0    |
|                       |   |   |   |   |   |   |   |   |   |   | 1     | × 0 | 0    |
|                       |   |   |   |   |   |   |   |   |   |   | Total |     | 2012 |

- In generale la base ci indica come interpretare le cifre!

# INTRODUZIONE ALL'INFORMATICA: DATI E VARIABILI

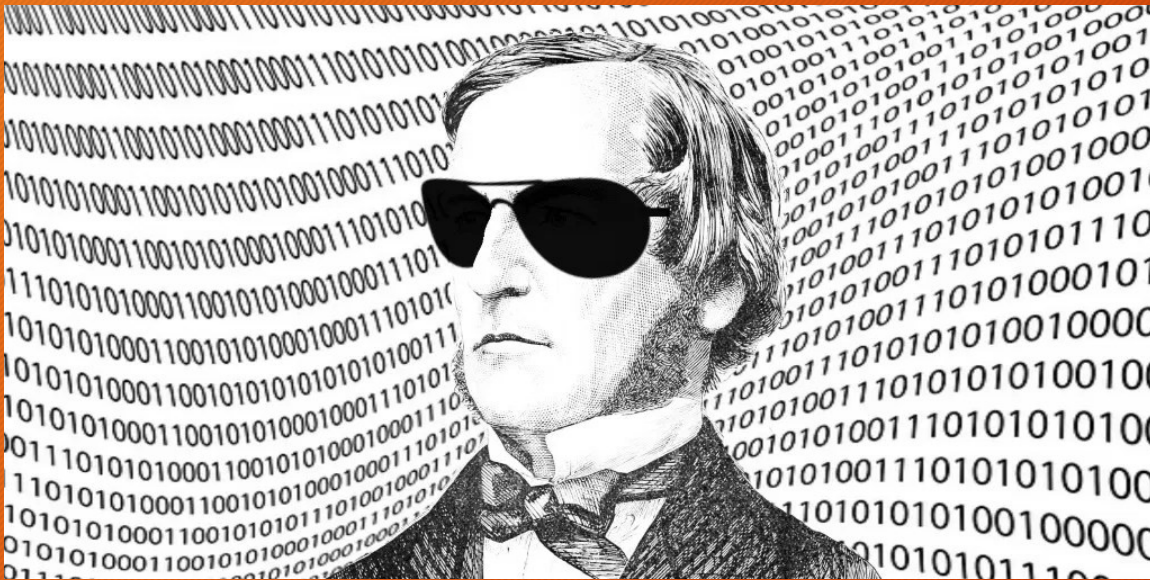
Spostandoci nell'aspetto pratico,  
quando **programmiamo** il numero (in generale il “dato”)  
sarà conservato in una **variabile**



**Ricordate: per il computer il “dato” è SEMPRE un insieme di 0 e 1!**



# INTRODUZIONE ALL'INFORMATICA: ALGEBRA BOOLEANA



Nell'algebra booleana le variabili  
possono assumere unicamente  
i valori 0 o 1

A livello logico questi corrispondono  
Ai valori **FALSO** o **VERO**

Potendo operare su VERO o FALSO un computer può  
**ESEGUIRE UN PROGRAMMA** seguendo un **ALGORITMO**

# ALGEBRA BOOLEANA: OPERATORI BOOLEANI

Gli operatori booleani permettono di elaborare delle espressioni che contengono al loro interno delle variabili booleane  
Sono gli "strumenti" che usa il computer per prendere una decisione!

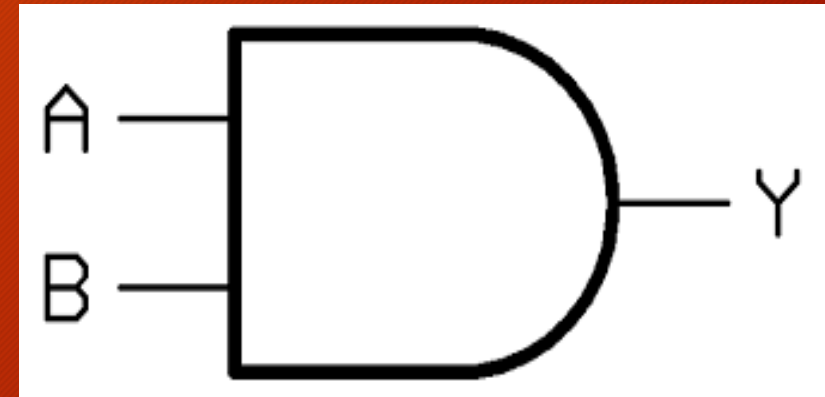
Si può dimostrare che bastano solo TRE operatori per potere elaborare QUALUNQUE espressione logica!

**AND - OR - NOT**



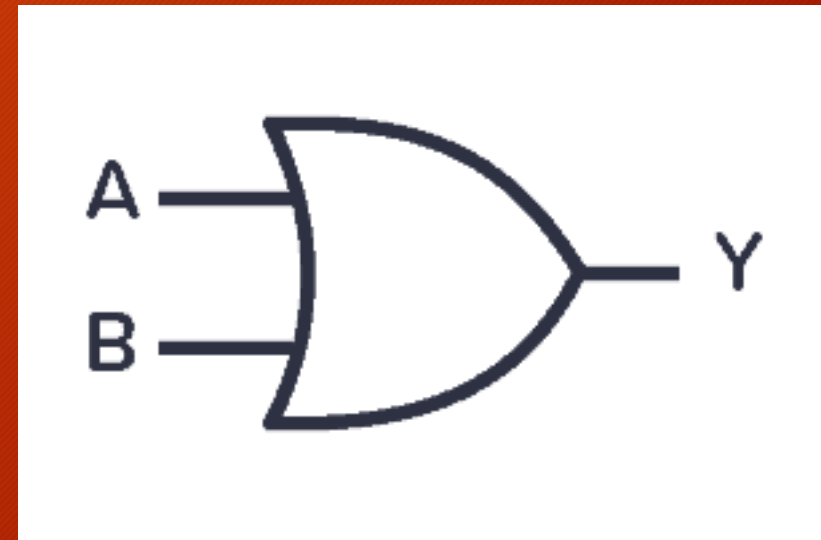
# TABELLA DI VERITA' - AND

| A<br><i>(primo operando)</i> | B<br><i>(secondo operando)</i> | Y<br><i>(uscita)</i> |
|------------------------------|--------------------------------|----------------------|
| VERO                         | VERO                           | VERO                 |
| VERO                         | FALSO                          | FALSO                |
| FALSO                        | VERO                           | FALSO                |
| FALSO                        | FALSO                          | FALSO                |



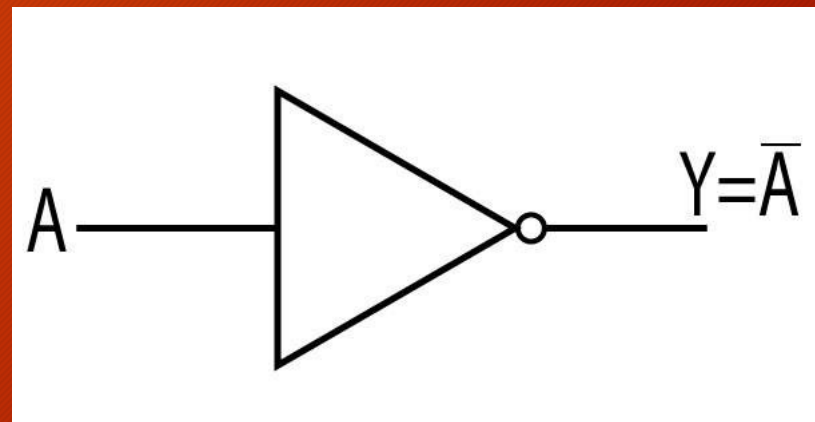
# TABELLA DI VERITA' - OR

| A<br><i>(primo operando)</i> | B<br><i>(secondo operando)</i> | Y<br><i>(uscita)</i> |
|------------------------------|--------------------------------|----------------------|
| VERO                         | VERO                           | VERO                 |
| VERO                         | FALSO                          | VERO                 |
| FALSO                        | VERO                           | VERO                 |
| FALSO                        | FALSO                          | FALSO                |



# TABELLA DI VERITA' - NOT

| A<br>(operando) | Y<br>(uscita) |
|-----------------|---------------|
| VERO            | FALSO         |
| FALSO           | VERO          |





# METTERE TUTTO INSIEME: UN ALGORITMO

Questi operatori sono resi disponibili  
da ogni linguaggio di programmazione

Permettono, insieme al costrutto IF/ELSE di creare un  
**FLUSSO DI ESECUZIONE**

Questo **FLUSSO DI ESECUZIONE** implementa un **ALGORITMO**

**UN ALGORITMO BEN PROGETTATO RAGGIUNGE L'OBIETTIVO  
PER CUI ABBIAMO DECISO DI SCRIVERE UN PROGRAMMA!**

# PERCHÉ PYTHON

- Python è un linguaggio moderno che sfrutta strumenti informatici che semplificano lo sviluppo:
  - Interpretato (non “compilato”)
  - “Garbage Collection” - (gestione automatica della memoria)
  - Orientato agli oggetti - OOP
  - Tipizzazione dinamica

# PYTHON 2 VS PYTHON 3

- Python viene attualmente mantenuto su **due versioni**:
- **Python 2**: La versione “storica” mantenuta per motivi di **retrocompatibilità** - si prevede di **deprecarla**!
- **Python 3**: La versione di riferimento, quella su cui conviene allinearsi
- Un programma scritto per una versione **NON** funzionerà sull'interprete dell'altra!
- Fate attenzione perché alcuni **moduli** (vedremo più avanti...) possono richiedere **una versione specifica**!



# IL PRIMO PROGRAMMA - CIAO MONDO!

```
print("Hello, World!")
```

Questo programma stampa a video la frase “Hello, World!”

# CONFRONTIAMOLO CON IL C...

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");

    return 0;
}
```

# STRUTTURA DI UN PROGRAMMA PYTHON

- Possiamo identificare 3 macro aree:
- **MODULI:** Permettono di integrare delle funzionalità (magari scritte da altri programmatori!)
- **ISTRUZIONI DI CONTROLLO:** Permettono di definire il flusso di esecuzione
- **OGGETTI:** Tutto è un oggetto in Python! L'interprete si occuperà automaticamente di assegnare e liberare la memoria per le variabili che decidiamo di usare.



# VARIABILI IN PYTHON - ASSEGNAZIONE

- In Python non dobbiamo dichiarare le variabili, solo assegnarle
- “=” è l’operatore di assegnazione

```
robot = "Terminator"  
print(robot)
```

# VARIABILI IN PYTHON - ASSEGNAZIONE

- Possiamo anche essere più creativi con l'operatore "=" ...

```
robot = arnold = "Terminator"  
print(arnold)  
  
kill_count = 100  
  
kill_count += 1
```



Quanto vale kill\_count  
all'ultima istruzione?

# OPERATORI ARITMETICI

- Gli operatori aritmetici sono intuitivi e combinabili con l'assegnazione:

```
escaped = 1  
kill_count = kill_count - escaped  
double_kill = kill_count * 2  
half_kill = kill_count / 2
```



# CONCATENAZIONE DI STRINGHE

- Cosa succede se proviamo ad usare il + su una stringa di caratteri?

```
basic_red = "Mountain"  
basic_blue = "Island"  
dual_land = basic_red + " " + basic_blue  
print(dual_land) # "Mountain Island"
```

(La terza stringa è lo spazio vuoto)

L'operazione di “legare” una stringa alla successiva si definisce CONCATENAZIONE

# NUMERI INTERI E NUMERI REALI

- Usando il punto possiamo rappresentare anche numeri reali:

```
PI = 3.1415  
almost_PI = int(PI)
```



“Forzare” una variabile a “diventare” intera  
(in generale, convertirla in un ALTRO TIPO) si chiama “CASTING”

# MODULI

- Come già introdotto, un modulo permette di INCLUDERE altre funzionalità sia dalla LIBRERIA STANDARD (fornita direttamente da Python) sia dalla COMUNITÀ DI SVILUPPO od anche MODULI PROPRI:

La parola chiave di riferimento è **IMPORT**:

```
import time  
import serial
```

Moduli standard

```
import my_awesome_module  
import some_dude_module
```

Moduli non standard

I moduli vanno  
importati  
all'INIZIO  
del codice!



# PYTHON DATA COLLECTIONS - LISTE

- A volte può essere utile riunire più oggetti in uno "stesso contenitore" in Python questo si può fare con le LISTE:

```
boring_fruits = ["Apple", "Banana", "Pear"]  
cool_fruits = ["Watermelon", "Cherry", "Peach"]
```

- L'accesso a gli elementi si effettua tramite INDICE:

```
boring_fruits[0]
```

(N.B. - Si inizia a contare da 0!)



# PYTHON DATA COLLECTIONS - DIZIONARI

- Può essere comodo a volte associare due elementi fra di loro
- Questo si può fare in Python con un **DIZIONARIO**
- In questo modo possiamo accedere con una **CHIAVE** e ritornare un **VALORE**

```
mana_base = {  
    "Red": "Mountain",  
    "Blue": "Island",  
    "White": "Plain",  
    "Green": "Forest",  
    "Black": "Swamp"  
}  
  
print(mana_base["Blue"])
```



Chiave



Valore

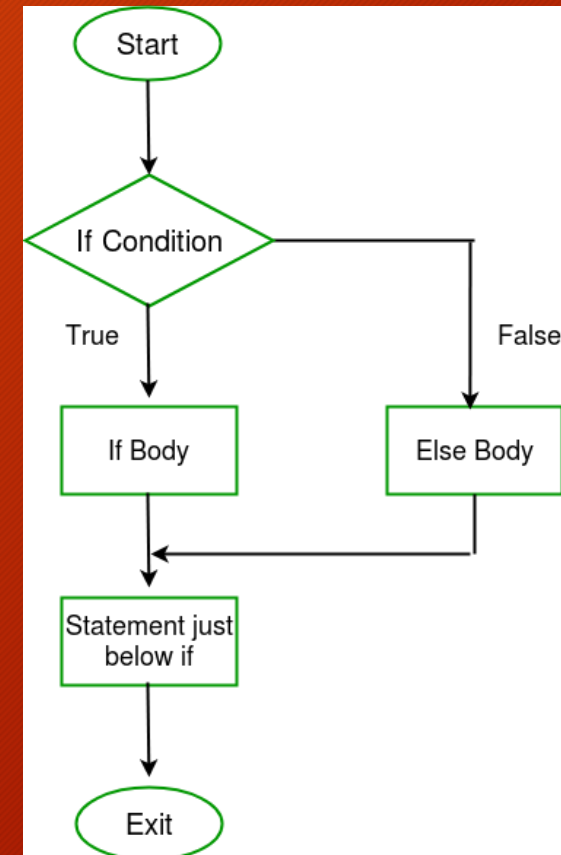


N.B. - Ad una chiave possiamo associare qualsiasi OGGETTO!  
Anche una LISTA o un DIZIONARIO (dizionari di dizionari!)

# STRUTTURE DI CONTROLLO - IF / ELIF / ELSE

- Il costrutto IF / ELIF / ELSE permette di **CONTROLLARE LA DIREZIONE** del **FLUSSO DI ESECUZIONE**:

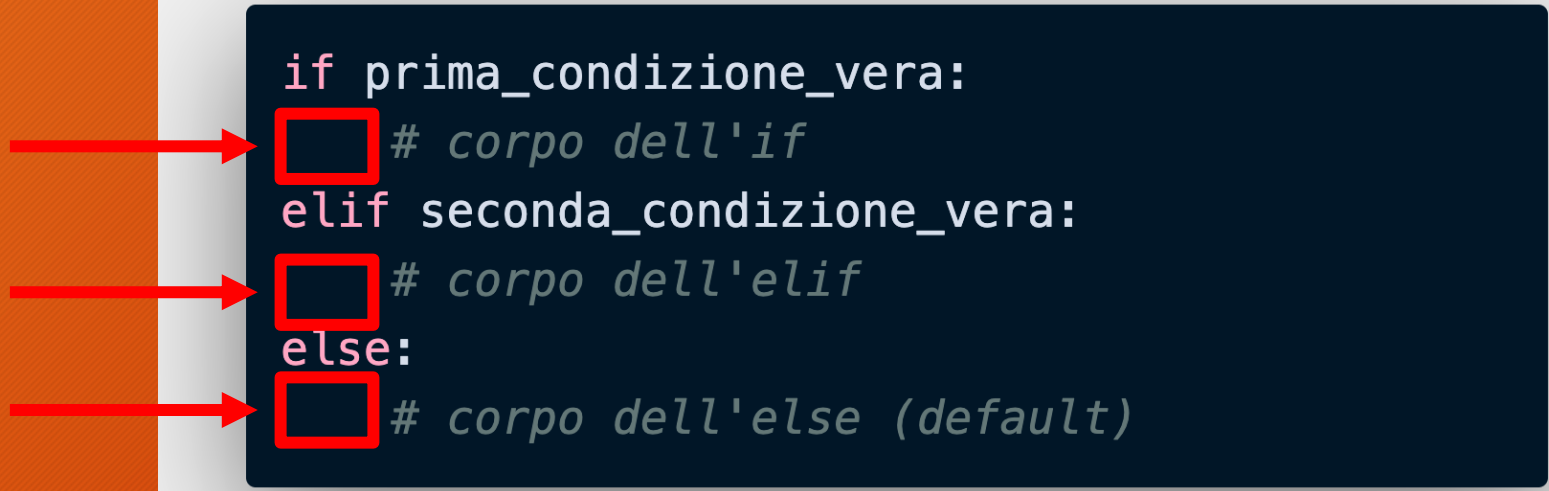
```
if prima_condizione_vera:  
    # corpo dell'if  
elif seconda_condizione_vera:  
    # corpo dell'elif  
else:  
    # corpo dell'else (default)
```





# STRUTTURE DI CONTROLLO - IF / ELIF / ELSE

A livello SINTATTICO è importante ricordare che il BLOCCO è identificato dall'IDENTAZIONE!  
In questo modo Python “forza” lo sviluppatore ad usare sempre l'indentazione corretta!



```
if prima_condizione_vera:
    # corpo dell'if
elif seconda_condizione_vera:
    # corpo dell'elif
else:
    # corpo dell'else (default)
```

The diagram illustrates the correct indentation for Python control structures. It shows a code block with three lines: an `if` statement, an `elif` statement, and an `else` statement. Each of the three lines is followed by a comment indicating the body of the respective statement. The body of each statement is enclosed in a red box, and a red arrow points from the left to the box, highlighting the indentation. The comments are: `# corpo dell'if`, `# corpo dell'elif`, and `# corpo dell'else (default)`.

# STRUTTURE DI CONTROLLO - IF / ELIF / ELSE

In caso di IF annidati, ogni livello deve avere la sua rispettiva indentazione:

Primo livello di indentazione

Secondo livello di indentazione

```
if prima_condizione_vera:
    ☐ # corpo dell'if esterno
    if condizione_annidata:
        ☐ # corpo dell'if interno
    else:
        ☐ # corpo dell'else interno
else:
    ☐ # corpo dell'else esterno
```

# STRUTTURE DI CONTROLLO - OPERATORI BOOLEANI

- Gli operatori booleani che abbiamo già introdotto nella scorsa lezione possono essere usati **per costruire le condizioni** di cui vogliamo verificare il **valore di verità logico**:

```
if prima_cond or seconda_cond and  
    (terza_cond or not quarta_cond):  
    print("...Ma quante ne sono?")  
else:  
    print("...Ed è anche FALSO!")
```



# STRUTTURE DI CONTROLLO - OPERATORI BOOLEANI

- Gli operatori più utilizzati sono:

|                  |                 |
|------------------|-----------------|
| <b>a == b</b>    | Uguale          |
| <b>a != b</b>    | Diverso         |
| <b>a &gt; b</b>  | Maggiore        |
| <b>a &lt; b</b>  | Minore          |
| <b>a &gt;= b</b> | Maggiore uguale |
| <b>a &lt;= b</b> | Minore uguale   |
| <b>a and b</b>   | AND logica      |
| <b>a or b</b>    | OR logica       |
| <b>not a</b>     | NOT logica      |
| <b>True</b>      | VERO logico     |
| <b>False</b>     | FALSO logico    |

# STRUTTURE DI CONTROLLO - CICLO FOR

- Un ciclo in generale ripete (“itera”) una serie di istruzioni finché una certa condizione è vera
- Il ciclo FOR itera seguendo l’evoluzione di un INDICE
- Molto spesso si itera all’INTERNO DI UNA LISTA:

```
kill_list = [13, 68, 0, 55, 31]
for kill_count in kill_list:
    if kill_count > 25:
        print("BLOOD-SEEKER!")
```

Indice del FOR

Ad ogni iterazione  
l'indice SCORRE  
gli elementi della lista

# STRUTTURE DI CONTROLLO - CICLO WHILE

- Un ciclo WHILE invece ITERA finché (while...!) la **CONDIZIONE DI CICLO** rimane vera:

```
hp = 50
while hp > 0:
    if damage_is_taken:
        hp -= 1
print("You're DEAD!")
```

N.B. - Sia per il ciclo FOR che per il ciclo WHILE bisogna rispettare sempre le regole dell'IDENTAZIONE per il BLOCCO di codice!



# STRUTTURE DI CONTROLLO - CICLO WHILE

Alcune considerazioni:

- 1) Il ciclo WHILE ed il ciclo FOR sono INTERSCAMBIABILI!  
Usiamo quello più adatto al contesto ma uno PUÒ SEMPRE essere trasformato nell'ALTRO
- 2) È possibile instaurare dei LOOP INFINITI (solitamente con il WHILE), es. :

```
while True:  
    print("Non uscirò mai più da qui")
```

Un loop infinito può sia essere un **ERRORE** (implementazione non corretta) sia una **NECESSITA'**!

# FUNZIONI - INTRODUZIONE

Le **FUNZIONI** permettono di **INCAPSULARE** una parte di codice in modo da poter essere **RIUTILIZZABILE** (In tempi antichi venivano anche definite **SOTTOPROGRAMMI**)

La parola chiave in questo caso è **DEF**

```
def compute_advantage(player_1_score, player_2_score):  
    advantage = ""  
    if player_1_score > player_2_score:  
        advantage = "player_1"  
    elif player_2_score > player_1_score:  
        advantage = "player_2"  
    else:  
        advantage = "draw"  
    return advantage
```

# FUNZIONI - ANATOMIA DI UNA FUNZIONE

PARAMETRI DI INGRESSO

```
def compute_advantage(player_1_score, player_2_score):  
    advantage = ""  
    if player_1_score > player_2_score:  
        advantage = "player_1"  
    elif player_2_score > player_1_score:  
        advantage = "player_2"  
    else:  
        advantage = "draw"  
    return advantage
```

CORPO DELLA FUNZIONE

RITORNO DELLA FUNZIONE



# FUNZIONI - INVOCAZIONE DI UNA FUNZIONE

Una volta definita, la funzione può essere INVOCATA nel resto del codice:

```
player_1_score = 15  
player_2_score = 33  
advantage = compute_advantage(player_1_score, player_2_score)  
print(advantage)
```

# FUNZIONI - CONSIGLI DI STILE

## Qualche regoletta per scrivere delle FUNZIONI ELEGANTI

- Una funzione dovrebbe essere **ATOMICA**: meno fa meglio è!
- Se una funzione sembra fare più del dovuto la soluzione è semplice: la dividiamo in più funzioni!
- Una funzione non dovrebbe prendere **TROPPI** parametri di ingresso!
- Dal **NOME** della funzione dovremmo già intuire a cosa dovrebbe servire!



# FUNZIONI - SCOPE DELLE VARIABILI

Il concetto di SCOPE ( o VISIBILITÀ) è molto importante: le variabili definite in una funzione ESISTONO solo durante la sua esecuzione!

```
def compute_advantage(player_1_score, player_2_score):  
    advantage = ""  
    if player_1_score > player_2_score:  
        advantage = "player_1"  
    elif player_2_score > player_1_score:  
        advantage = "player_2"  
    else:  
        advantage = "draw"  
    return advantage
```

Tutte le variabili che dichiariamo nel corpo della funzione verranno DEALLOCATE al suo termine!

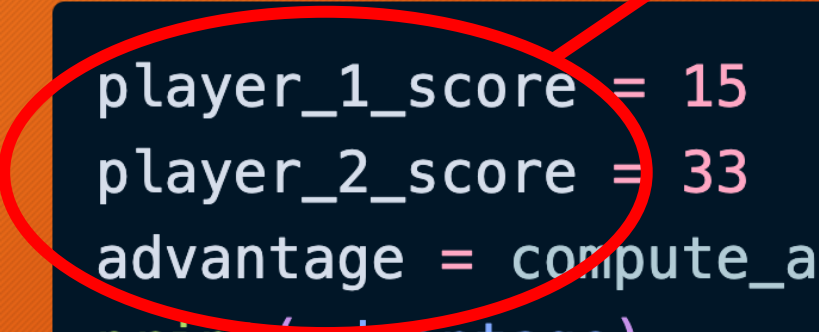
Questo vuol dire che FUORI dalla funzione quelle variabili NON SONO ACCESSIBILI!



# FUNZIONI - SCOPE DELLE VARIABILI

Questo permette di avere **VARIABILI** con lo **STESSO NOME** senza che ci sia nessun **CONTRASTO** fra di loro!

Queste variabili sono in un **LIVELLO DI SCOPE** diverso rispetto a quelle della funzione!



```
player_1_score = 15  
player_2_score = 33  
advantage = compute_advantage(player_1_score, player_2_score)  
print(advantage)
```

# FUNZIONI - VARIABILI GLOBALI

Per avere una variabile a SCOPE GLOBALE (cioè VISIBILE ed ACCESSIBILE per TUTTO il codice) si usa la parola chiave GLOBAL:

```
outside_variable = "global_value"
```

Una variabile dichiarata FUORI da una funzione è già GLOBALE!

```
def my_function():
```

```
    global outside_variable
```

```
    outside_variable = "changed_value"
```

Se voglio MODIFICARLA in una funzione devo DICHIARARLO usando la parola "global"

```
my_function()
```

```
print(outside_variable) # "changed_value"
```

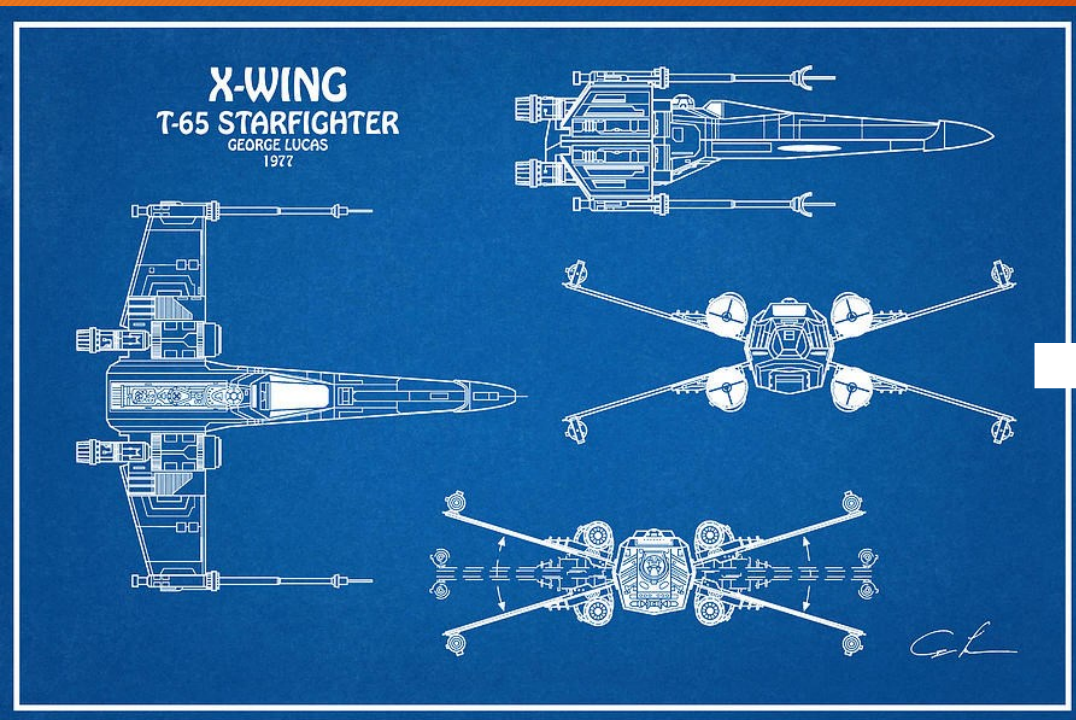
In questo modo la variabile sarà modificata anche FUORI dalla funzione

# CLASSI E OGGETTI

- Come abbiamo introdotto, Python è **orientato a gli oggetti**
- Un concetto fondamentale in questo **paradigma di programmazione** è la CLASSE
- La CLASSE è un concetto complesso che porta con sé una propria **filosofia di sviluppo**
- Un modo semplice per introdurlo è considerarlo come un TIPO PERSONALIZZATO



# CLASSI E OGGETTI



Classe



Oggetti “costruiti” dalla classe  
(ISTANZE di una una classe)

# CLASSI E OGGETTI

Vediamo un esempio se volessimo definire un “tipo di robot”

```
class Robot:
    def __init__(self, robot_color, robot_size):
        self.color = robot_color
        self.size = robot_size

red_big_robot = Robot("red", "XL")
blue_small_robot = Robot("blue", "S")

print(red_big_robot.color) # "red"
print(blue_small_robot.size) # "S"
```

# CLASSI E OGGETTI

```
class Robot:
```

```
    def __init__(self, robot_color, robot_size):
```

```
        self.color = robot_color
```

```
        self.size = robot_size
```

```
red_big_robot = Robot("red", "XL")
```

```
blue_small_robot = Robot("blue", "S")
```

```
print(red_big_robot.color) # "red"
```

```
print(blue_small_robot.size) # "S"
```


Questa è una **FUNZIONE SPECIALE**  
che si chiama **COSTRUTTORE**:  
serve a specificare quali sono i **PARAMETRI**  
di un oggetto **COSTRUITO** tramite quella classe

La sua forma sarà **SEMPRE**:  
`__init__(self, parametro_1, parametro_2 ...)`



# CLASSI E OGGETTI

```
class Robot:
    def __init__(self, robot_color, robot_size):
        self.color = robot_color
        self.size = robot_size
```



Qui stiamo dicendo al **COSTRUTTORE**  
come **RIEMPIRE** i parametri dell'oggetto  
che vogliamo definire

```
red_big_robot = Robot("red", "XL")
blue_small_robot = Robot("blue", "S")

print(red_big_robot.color) # "red"
print(blue_small_robot.size) # "S"
```

# CLASSI E OGGETTI

```
class Robot:
    def __init__(self, robot_color, robot_size):
        self.color = robot_color
        self.size = robot_size
```

```
red_big_robot = Robot("red", "XL")
blue_small_robot = Robot("blue", "S")
```

```
print(red_big_robot.color) # "red"
print(blue_small_robot.size) # "S"
```

Qui stiamo usando il **COSTRUTTORE**  
(che si invoca con il **NOME DELLA CLASSE**)  
per **COSTRUIRE** due oggetti diversi

# CLASSI E OGGETTI

```
class Robot:
    def __init__(self, robot_color, robot_size):
        self.color = robot_color
        self.size = robot_size
```

```
red_big_robot = Robot("red", "XL")
blue_small_robot = Robot("blue", "S")
```

```
print(red_big_robot.color) # "red"
print(blue_small_robot.size) # "S"
```

Qui stiamo ACCEDENDO ai PARAMETRI  
dei singoli oggetti costruiti prima  
TRAMITE L'OPERATORE . (punto)



# CLASSI E OGGETTI

Una classe può contenere al suo interno anche delle FUNZIONI associate a quel particolare OGGETTO:  
Queste funzioni si chiamano METODI DELL'OGGETTO

```
class Robot:
    def __init__(self, robot_color, robot_size):
        self.color = robot_color
        self.size = robot_size
```

```
def print_color(self):
    print("My color is " + self.color)
```

```
def print_size(self):
    print("My size is " + self.size)
```

```
green_medium_robot = Robot("green", "M")
```

```
green_medium_robot.print_color() # "My color is green"
```

```
green_medium_robot.print_size() # "My size is M"
```

I metodi devono avere sempre  
ALMENO un parametro di INGRESSO: "self"  
Ricordate:

"self" serve per riferirsi all'OGGETTO costruito DALLA CLASSE

# CLASSI E OGGETTI

```
class Robot:
    def __init__(self, robot_color, robot_size):
        self.color = robot_color
        self.size = robot_size
```

```
def print_color(self):
    print("My color is " + self.color)
```

Metodo che stampa il colore del robot

```
def print_size(self):
    print("My size is " + self.size)
```

Metodo che stampa la grandezza del robot

```
green_medium_robot = Robot("green", "M")
```

```
green_medium_robot.print_color() # "My color is green"
```

Invocazione del primo metodo

```
green_medium_robot.print_size() # "My size is M"
```

Invocazione del secondo metodo