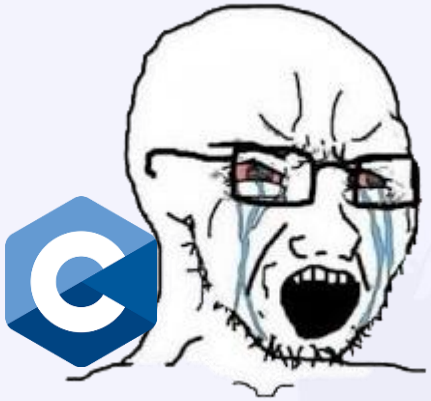
Introduciamo

(vi piacerà)

(fidatevi)

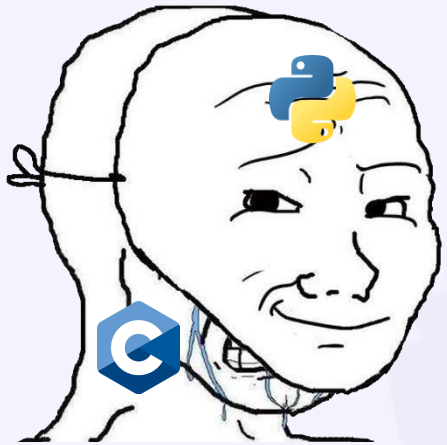
Ecco perché vi piacerà:



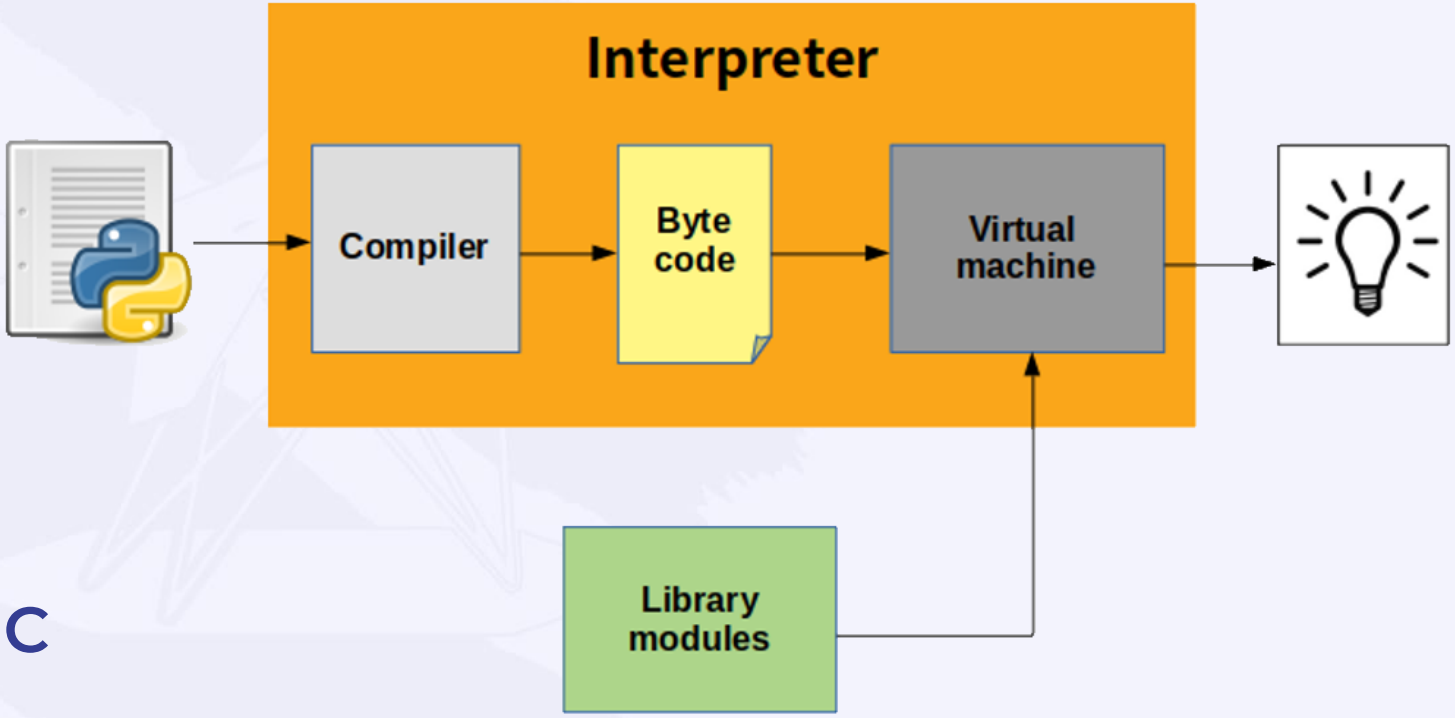
```c
#include <stdio.h>

int main(int args, char *argv){
    printf("Coding zero");
}
```

```python
print("Coding zero")
```

Ma facciamo un passo indietro...



Python viene **compilato**
da un **interprete scritto** in **C**



Variabili

primitive

```
# nome = valore

intero = 3
decimale = 7.3
carattere = "C"
stringa = "Testo"
```

In python non è necessario specificare il tipo

```
intero : int = 3
decimale : float = 7.3
carattere : str = "C"
stringa : str = "Testo"
```

Tuttavia farlo può evitare errori di assegnazione (in python: *TypeError*)

```
bytes_var = b"hello"
bytearray_var = bytearray(b"hello")
memoryview_var = memoryview(b"hello")
```

Esistono tipi più tecnici, utili nei protocolli di rete e nell'analisi dei dati

Variabili

collezioni

```python
list_var = [1, 2, 3, 4]          # list
tuple_var = (1, 2, 3, 4)         # tuple
set_var = {1, 2, 3, 4}           # set
frozenset_var = frozenset({1, 2, 3, 4})  # frozenset
dict_var = {"chiave": "valore", "età": 25}  # dict
range_var = range(10)            # range
```

Le collezioni (collections) contengono una lista di variabili sequenziali o associate tramite chiave.

Principali differenze

Tra tuple, list, set e dict

Tuple

- **Ordinata**: Gli elementi hanno un ordine definito e possono essere accessibili tramite indice.
- **Immutabile**: Una volta creata, non puoi modificarne il contenuto
- **Sintassi**: (1, 2, 3)

Dict

- **Mutabile**: Puoi aggiungere, modificare o rimuovere coppie chiave-valore.
- **Coppie chiave-valore**: Associa chiavi uniche a valori.
- **Sintassi:** {"chiave1": "valore1", "chiave2": "valore2"}

List

- **Ordinata**: Gli elementi hanno un ordine definito e possono essere accessibili tramite indice.
- **Mutabile**: Puoi aggiungere, rimuovere o modificare elementi.
- **Sintassi**: [1, 2, 3]

Set

- **Non ordinata**: Gli elementi NON hanno un ordine definito
- **Mutabile**: Puoi aggiungere e modificare elementi (ma niente duplicati)
- **Sintassi**: {1, 2, 3}

Cicli condizionali e iterativi

Costrutti if - else - elif

```
if condizione:
    # Codice se vera
elif altra condizione:
    # Codice se questa è vera
else:
    # Altrimenti
```

Ciclo for

```
for i in range(1, 100):
    # codice
```

```
array = [2,5,1,2,7,5]
for i in array:
    print(i)
```

```
for index, value in enumerate(array):
    print(index, value)
```

Ciclo while

```
while condizione:
    # codice_da_ripetere
```

```
while True:
    if condizione:
        break
```

Funzioni

```
def function(args):
    #code
```

In **ptyhon** non è necessario specificare **nè il tipo di ritorno** della funzione **nè il tipo degli argomenti.**

```
def somma(a: int, b: int) -> int:
    return a + b
```

Ma anche in questo caso è possibile farlo per **evitare di cadere in errore** con codici troppo complessi

```
variabile = "globale"

def funzioneLocale():
    return variabile + " non
sono stata cambiata"

def funzioneGlobale():
    global variabile
    return variabile + " sono
stata cambiata"
```

Di default le funzioni in python operano con variabili locali.
Se viene specificata la keyword global tuttavia, le modifiche sulla variabile avranno effetti globali

Classi

```python
class Rettangolo:
    def __init__(self, lato1, lato2):
        self.lato1 = lato1
        self.lato2 = lato2

    def perimetro(self):
        return (2*self.lato1) + (2*self.lato2)

    def area(self):
        return self.lato1 * self.lato2
```

Python introduce anche elementi di OOP. Con l'uso delle classi possiamo rappresentare entità reali o astratte tramite metodi e attributi per rendere il codice scalabile e riutilizzabile.

Qualche funzione built-in

```
# 1. print()
# Stampa a schermo qualsiasi valore passato come argomento.
print()

# 2. len()
# Restituisce la lunghezza (numero di elementi) di una sequenza, come una lista o una stringa.
len()

# 3. type()
# Restituisce il tipo di un oggetto.
type()

# 4. int()
# Converte una stringa o un numero in un intero.
int()

# 5. str()
# Converte un valore in una stringa.
str()
```

```
# 6. sum()
# Calcola la somma di tutti gli elementi in un iterabile (come una lista o una tupla).
sum()

# 7. max()
# Restituisce il valore massimo in un iterabile.
max()

# 8. min()
# Restituisce il valore minimo in un iterabile.
min()

# 9. sorted()
# Restituisce una nuova lista ordinata dagli elementi di un iterabile.
sorted()

# 10. range()
# Genera una sequenza di numeri, utile per iterazioni.
range()
```

Per vedere tutte le funzioni built in e le potenzialità di python, clicca qui

Alcune delle librerie più note

```python
# 1. numpy.array()
# Crea un array multidimensionale, utile per il calcolo numerico.
import numpy as np
np.array()

# 2. pandas.DataFrame()
# Crea un DataFrame, una struttura dati bidimensionale simile a una
tabella.
import pandas as pd
pd.DataFrame()

# 3. matplotlib.pyplot.plot()
# Crea un grafico a linee, molto usato per la visualizzazione dei dati.
import matplotlib.pyplot as plt
plt.plot()

# 4. seaborn.heatmap()
# Crea una heatmap (mappa di calore) basata su dati bidimensionali.
import seaborn as sns
sns.heatmap()

# 5. requests.get()
# Invia una richiesta HTTP GET per recuperare dati da un URL.
import requests
requests.get()
```

Alcune delle librerie più note

```python
# 6. tensorflow.keras.Model()
# Definisce un modello di rete neurale in TensorFlow.
import tensorflow as tf
tf.keras.Model()

# 7. sklearn.linear_model.LinearRegression()
# Crea un modello di regressione lineare, usato nell'analisi dei dati e
nel machine learning.
from sklearn.linear_model import LinearRegression
LinearRegression()

# 8. os.listdir()
# Restituisce una lista dei nomi di file in una directory.
import os
os.listdir()

# 9. sys.exit()
# Termina l'esecuzione di uno script Python.
import sys
sys.exit()

# 10. json.loads()
# Converte una stringa JSON in un oggetto Python (come un dizionario).
import json
json.loads()
```

E ora... Live Coding

Pong

Calcolatrice grafica