****

**DA HELLO WORLD A GPT**

*Ai Nerd,*

*che possa ispirare qualcuno a fare qualcosa di buono e di utile*

***PREFAZIONE***

Cresciuto a pane e Holly e Benji, quando ero molto piccolo, ho gli stessi anni di Macintosh e Windows se proprio lo volete sapere, mi sono appassionato all’informatica. All’epoca se conoscevi il Pascal eri un mezzo genio. I computer non erano assolutamente diffusi come lo sono oggi. I primi che servivano a qualcosa sono stati i Pentium. Proprio nel 93 (grandissima annata per i computer), nasce il Pentium, viene eseguita per la prima volta la navigazione su internet da NCSA, il CERN rilascia il world wide web a livello globale ed è il primo anno che nel mondo vengono venduti più computer che automobili. E quell’anno feci il grande passo.

Negli anni precedenti i computer esistevano per lo più solo nelle grandi aziende. L’Italia, a quei tempi, ancora non aveva “fatto scappare” tutte le aziende all’estero, anzi eravamo competitivi. Penaste che lo streaming lo hanno inventato Olivetti e Rai nel ’94 (Skydata). Mi sono avvicinato sempre di più ai computer attraverso MS-DOS, per il tempo era roba da Nerd. E così è cominciato tutto. Negli anni successivi non ho avuto la fortuna di essere stimolato nell’informatica, ma mi sono dedicato al calcio, fino ad arrivare ad ottimi livelli. Ho trascurato i PC e a più riprese mi sono riavvicinato, sempre quando un amico o il capo avevano dei problemi. Un volta, mentre stavo provando qualche magheggio, feci saltare la rete aziendale; dove lavoro siamo una sessantina di uffici 😊. Chiamai subito l’allora Telecom per ripristinare il gusto inventandogli una balla, e fortunatamente il giorno successivo era di nuovo tutto a posto. Sentivo di esserci portato. Molti anni dopo, era il 2009 o il 2010 non ricordo bene l’anno esatto, tramite un sintetizzatore vocale, realizzai una sorta di Google assistant, lo chiamai Jarvis, in onore del GRANDISSIMO TONY STARK che mi ha ispirato tantissimo. Girava sul mio sito, cliccando su un pulsante produceva del testo estratto da un DB con delle frasi campionate ad hoc in base a delle richieste. Era molto basilare: poteva salutarti, imparare il tuo nome, aprire una pagina web o YouTube. Ma era una figata. Successivamente ho iniziato a sviluppare giochi in javascript, siti ecc., ma non ho mai avuto il coraggio di avvicinarmi ad un’azienda, nonostante non fossi uno ripreso dalla piena. Oggi a distanza di mille anni mi occupo di tutt’altro nella vita, ma ho preso la decisione di prendere quella benedetta laurea. Probabilmente l’informatica non sarà mai il mio lavoro primario, ma continuerà ad essere un grande passione. Così, nonostante oggi non serva quasi più aiutare le persone a capire la programmazione, perché basta utilizzare ChatGPT, Claude, Gemini o qualunque altro modello avanzato di AI che è in grado di scrivere decine e decine di righe dettandogli le vostre richieste, provo a scrivere questo libro per aiutare gli altri alla vecchia maniera, a scrivere codice e soprattutto a capire cosa ci sta dietro. Prendete ad esempio una fotografia scattata con un cellulare. Facciamo un portrait (ritratto). Molte persone non sanno cosa avvenga mediante il software quando viene sfocato il soggetto. Male. Nella fotografia tradizionale utilizzavi un obiettivo, nel caso delle foto in primo piano un teleobiettivo, ruotavi la ghiera per mettere a fuoco la persona e successivamente facevi lavorare i diaframmi, ovvero ruotavi l’altra ghiera quella della luminosità dell’obiettivo per aumentare o diminuire il passaggio di luce, premevi il pulsante della profondità di campo per vedere se il risultato era soddisfacente. Infine, premevi il pulsante di scatto. La ghiera dei diaframmi controlla le lamelle del diaframma all'interno dell'obiettivo. Ruotando la ghiera, le lamelle si aprono o si chiudono, modificando l'ampiezza dell'apertura attraverso cui passa la luce. Quando le lamelle del diaframma si aprono di più, ad esempio f/2.8, permettono il passaggio di una maggiore quantità di luce. Questo riduce la profondità di campo, creando una zona più stretta di messa a fuoco. Il soggetto in primo piano sarà di conseguenza più nitido mentre lo sfondo appare sfocato. Questo succede quando premete il pulsante bianco del vostro smartphone. Prima era un’arte, ora è un pulsante. Ma ora lo sapete.

E questo libro ha lo scopo di farvi conoscere cosa c’è dietro tetris, piuttosto che ChatGPT. Tutto qui.

***CONVEZNIONI USTAE IN QUESTO LIBRO***

Con questo carattere verranno indicate tutte le istruzioni di codice: #esempio di codice python

Tutti gli esempi sono stati eseguiti con pycharme 2024.1.2 e python 3.9

E’ stata utilizzata una macchina windows 11, ma potete utilizzare tranquillamente Mac oppure Linux (Ubuntu) che è la distribuzione raccomanda.

Quando ho deciso di scrivere questo libro ero con un amico, il quale lavora con la borsa e mi aveva chiesto un particolare script. Mi ha chiesto di spiegarli il codice e non sono stato in grado di farglielo capire. Ho provato con carta e penna, e niente. Ho provato a chiedere aiuto a due modelli AI, nel caso specifico ho provato sia con Gemini, sia con ChatGPT. Ho chiesto di provare a spiegarlo in maniera schematica ed alla fine il mio amico è riuscito a comprendere lo script borsistico. Pertanto, anche questo libro seguirà la stessa convenzione.

Gli esempi verranno mostrati con la seguente struttura:

*-codice*

*-spiegazione del codice*

*-esecuzione del codice*

*-output*

*-riepilogo*

Tutti i capitoli hanno questa struttura:

*-introduzione*

*-esempio con spiegazioni*

*-conclusione*

*-esercizi*

Ho cercato di mantenere questa struttura per linearità, ma soprattutto per spiegare con massima chiarezza, talvolta anche ripetendomi, il codice. E gli esercizi a fine capitolo sono importanti per mettere in pratica le competenze acquisite. TUTTI gli esercizi hanno le soluzioni nel repository del libro.

Molti degli esempi di questo libro li trovate al link di supporto github:

[*https://github.com/iacopoooo/Python\_book*](https://github.com/iacopoooo/Python_book)

Bando alle ciance, iniziamo!

***Introduzione a Python***

* Storia di Python
* Installazione e configurazione
* **Hello world**

** ***Fondamenti di Programmazione in Python***

* Variabili e Tipi di Dati

** ***Strutture Dati in Python***

* Liste
* Tuple
* Dizionari
* Insiemi

** ***Strutture di Controllo***

* Condizionali
* Cicli

** ***Funzioni***

* Definizione e utilizzo
* Parametri e argomenti
* Funzioni Lambda
* Scope e durata delle variabili

** ***Moduli e Pacchetti***

* Importazione di moduli
* Creazione di moduli
* Utilizzo dei pacchetti standard

** ***Directory e File System***

* Gestione dei file
* Operazioni su directory
* Utilizzo dei moduli os e shutil

*** Gestione delle Eccezioni***

* Tipi di eccezioni
* Gestione delle eccezioni
* Creazione di eccezioni personalizzate

** ***Programmazione Orientata agli Oggetti in Python***

* Classi e oggetti
* Ereditarietà
* Polimorfismo
* Incapsulamento

** ***Programmazione Avanzata***

* Decoratori
* Generatori
* Threading in Python

** ***Esempio di programmi avanzati***

* ***Interfaccia Utente e Visualizzazione dei Dati***
  + Matplot
  + Tkinter
* ***Sviluppo Web e Gestione dei Dati***
  + SQLite
  + Programmazione di Rete
  + Django
  + Flask
* ***Creare programmi eseguibili con python***
  + PyInstaller
  + Tetris
  + Clacolatrice Hystory
* ***AI***
  + Scikit-learn
  + LSTM (Long Short-Term Memory)
  + Transformer
* ***Il caffè che vi avevo promesso***

***1. Introduzione a Python***

**Cos'è Python?**

**Prima di tutto python è una bomba. Già, python è un linguaggio potentissimo con cui potete fare tutto; sì, potete fare anche il caffè. E più avanti vi mostrerò come.**

**Sui libri, sul web o se chidete ad un AI cosa sia python probabilmente vi risponderà questo:**

Python è un linguaggio di programmazione ad alto livello, orientato agli oggetti, interpretato, e con tipizzazione dinamica. Che vuol dire tutto questo? In altre parole significa che supporta sia la programmazione procedurale (le funzioni), sia il paradigma degli oggetti (ereditarietà e compagnia bella, ma ci arriveremo più avnti). Significa che lo potete utilizzare su qualunque sistema o piattaforma purchè abbia installato l’interprete python che vedremo a breve. Significa che è semplice, che servono meno linee di codice rispetto ad altri linguaggi.  Significa che l'interprete riconosce automaticamente le variabili dal suo contenuto durante l'assegnazione (runtime). Nota: in altri lnguaggi ad esempio il C, questo non accade, ma dovete dichiarare le variabili esplicitmente prima di utilizzarle. Siginifica che la sintassi è comprensibilissima. Insomma è uno dei lingaggi piu potenti e popolari al mondo.

Se uno non avesse mai programmato e dovesse scegliere solo tre linguaggi certamente gli consiglerei, assembly, C, python. Con questi tre vai ovunque e oltre.

Creato da Guido van Rossum è conosciuto per la sua leggibilità del codice e per la sintassi che permette agli sviluppatori di esprimere concetti in meno linee di codice rispetto ad altri linguaggi, come C++ o Java. Python è utilizzato in molti ambiti, come lo sviluppo web, data science, intelligenza artificiale, automazione, e altro ancora.

**Storia di Python** Python ha avuto un inizio modesto negli anni '80, quando Guido van Rossum iniziò a lavorare su un progetto chiamato ABC, che influenzò fortemente la sintassi di Python. Il linguaggio è stato rilasciato al pubblico nel 1991 come Python 0.9.0. Nel corso degli anni, Python è cresciuto costantemente, diventando uno dei linguaggi più popolari al mondo. Le versioni principali del linguaggio includono Python 2, rilasciato nel 2000, e Python 3, rilasciato nel 2008. Python 3 ha introdotto molte migliorie e modifiche rispetto a Python 2, e attualmente è la versione standard utilizzata nella maggior parte dei progetti.

**Perché Python?**

Ecco alcuni motivi per cui Python è tanto popolare:

***Facilità di apprendimento:*** Python ha una curva di apprendimento più ripida rispetto ad altri linguaggi di programmazione. **La sintassi intuitiva di Python** facilita l’apprendimento e l’uso del linguaggio, rendendolo ideale per chi si avvicina anche per la prima volta alla programmazione.

***Ampia comunità:*** Essendo uno dei linguaggi più popolari, Python ha una vasta comunità di sviluppatori e una ricca documentazione. (Link ufficiale: https://www.python.org)

***Versatilità:*** Può essere utilizzato per una varietà di applicazioni, dai semplici script a sistemi complessi.

***Librerie e Framework***: Python ha un vasto ecosistema di librerie e framework che possono essere utilizzati per accelerare lo sviluppo di applicazioni.

Per approfondimenti potete consultare i seguenti link:

-<https://docs.python.org/3/tutorial/index.html>

Tutorial dalla documentazione ufficiale

-https://www.programmareinpython.it/corsi-e-lezioni-python-dal-nostro-canale-youtube/

Qui troverete sia i video sia i testi

Sono due risorse che vi consiglio per l’apprendimento

**Installazione e Setup:**

L'installazione di Python è il primo passo per iniziare a programmare. Python è disponibile per tutti i principali sistemi operativi (Windows, MacOS, e Linux). Ecco come installarlo. Vai al sito ufficiale di Python (python.org) e scarica l'ultima versione di Python per il tuo sistema operativo.

Installazione: Segui le istruzioni per installare Python sul tuo computer. Durante l'installazione, assicurati di spuntare l'opzione per aggiungere Python al PATH del sistema (potrebbe essere difficoltoso per chi non ha esperienza aggiungerla successivamente), così da poter eseguire Python da qualsiasi prompt dei comandi.

* **Windows**: Scarica l'installer dal sito ufficiale di Python (python.org) e segui le istruzioni per l'installazione. Assicurati di selezionare l'opzione per aggiungere Python al PATH durante l'installazione.
* **MacOS**: Python è preinstallato su MacOS, ma si consiglia di installare la versione più recente tramite Homebrew. Usa il comando brew install python.
* **Linux**: Su molte distribuzioni Linux, Python è già preinstallato. Per installare la versione più recente, puoi usare il gestore di pacchetti della tua distribuzione (ad esempio, sudo apt-get install python3 su Ubuntu).

**IDE e editor di testo.** Per scrivere codice Python, puoi utilizzare una varietà di ambienti di sviluppo integrati (IDE) e editor di testo. Ecco alcuni dei più popolari:

* **IDLE**: L'IDE predefinito che viene fornito con l'installazione di Python, semplice e adatto per i principianti.
* **PyCharm**: Un potente IDE per Python con molte funzionalità avanzate per lo sviluppo professionale.
* **Visual Studio Code**: Un editor di testo gratuito e open-source con estensioni per il supporto a Python.
* **Jupyter Notebook**: Utilizzato principalmente per il data science, permette di creare e condividere documenti che contengono codice live, equazioni, visualizzazioni e testo esplicativo.
* **Sublime Text**: Un editor di testo leggero e altamente personalizzabile.

Di seguito vi mostro alcune screen dei seguenti ***PyCharm, Visual Studio Code e l’ide predefinito.***

Per scrivere codice Python, puoi usare:IDLE, L'editor integrato che viene fornito con Python, buono per iniziare senza complicazioni.

Immagine che contiene testo, schermata, schermo, software

Descrizione generata automaticamente

Fig.1 EDITOR DI PYTHON

Visual Studio Code: Un editor molto popolare, con supporto per Python tramite estensioni, che fornisce funzionalità come il debugging, il controllo del codice in tempo reale, e la gestione dei repository Git.

Link: <https://code.visualstudio.com/>

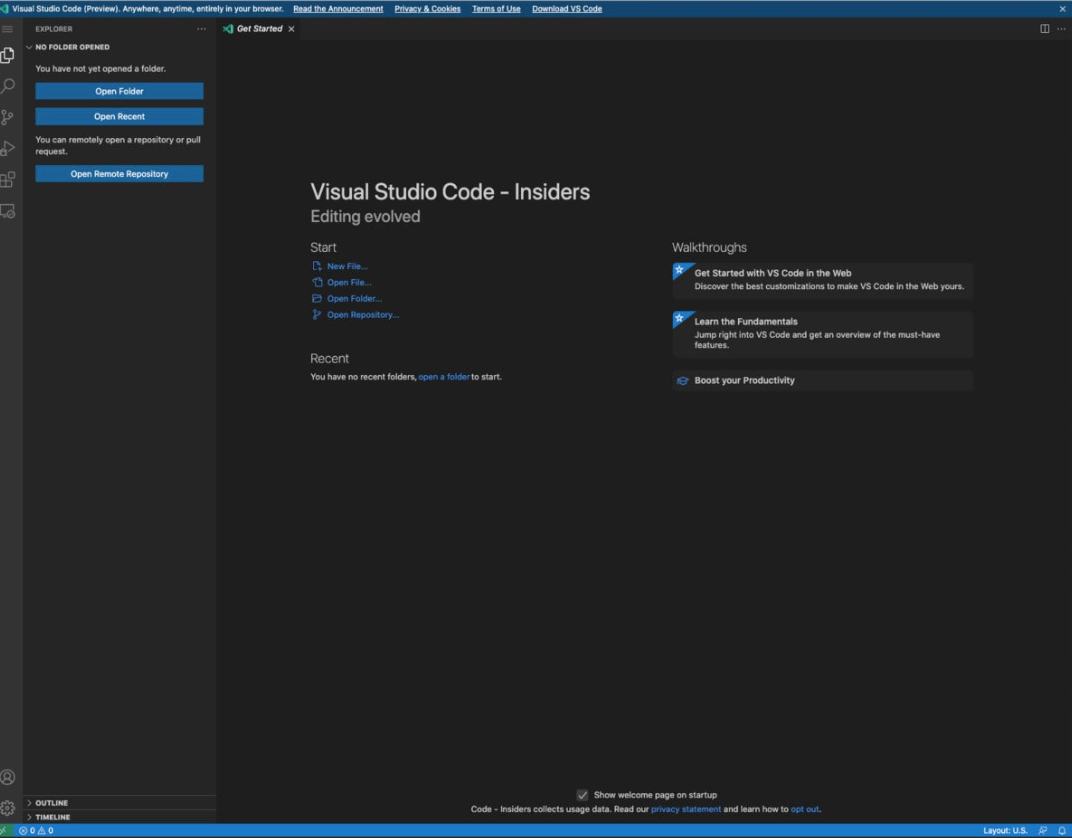


Fig.2 EDITOR VISUAL STUDIO CODE

PyCharm: Un IDE sviluppato da JetBrains, specifico per Python, che offre una suite completa di strumenti per lo sviluppo professionale.

Link: <https://www.jetbrains.com/pycharm/> vi consiglio pycharme community edition (versione gratuita)

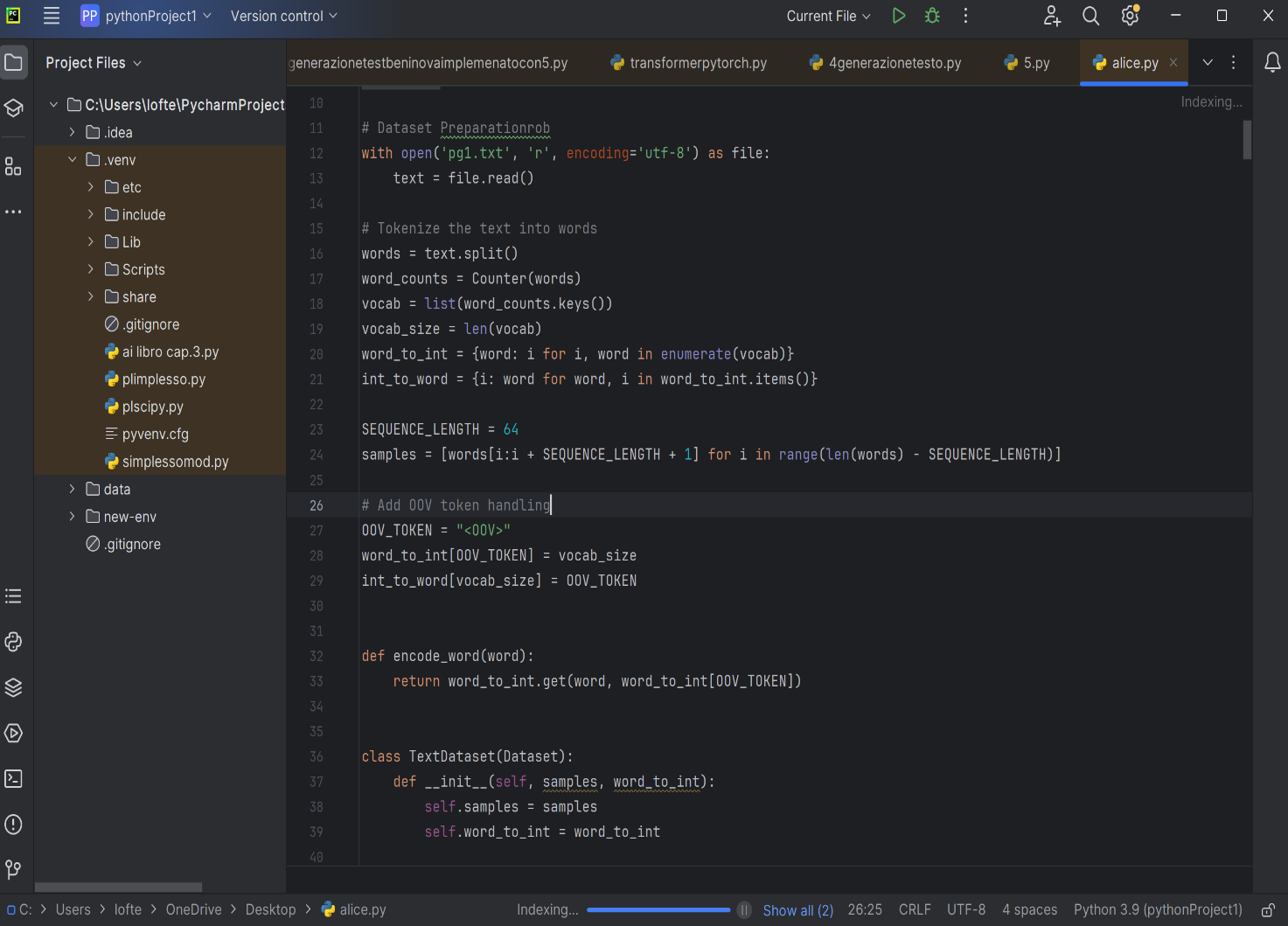


Fig.3 EDITO PYCHARME

Nota: per tutti gli esempi di questo libro abbiamo utilizzato *pycharme*

Primi Passi

Una volta installato Python e scelto il tuo ambiente di sviluppo, è il momento di scrivere il tuo primo programma. Il classico esempio "Hello, World!" è un ottimo punto di partenza. Ecco un esempio di codice Python per familiarizzare un po’:

1) Apri il tuo editor di testo o IDE.

2) Scrivi il seguente codice:

# Questo è un commento in Python  
  
“””

Questo è un  
commento  
Su più  
righe  
“””  
  
print("Ciao a tutti")  
  
#STAMPA Ciao, a tutti

3) Salva il file con estensione .py, ad esempio hello.py.

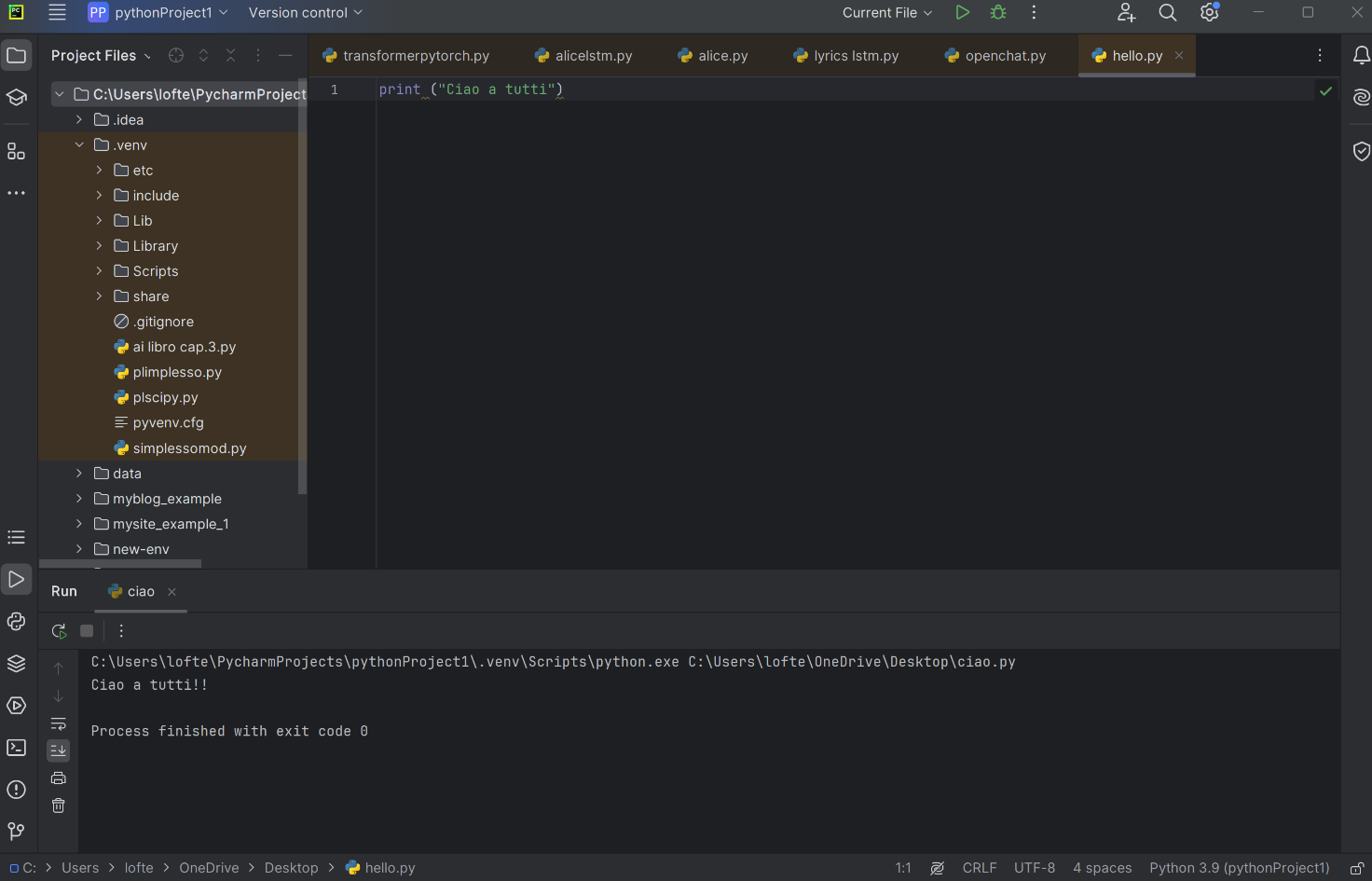
4) Esegui il programma:

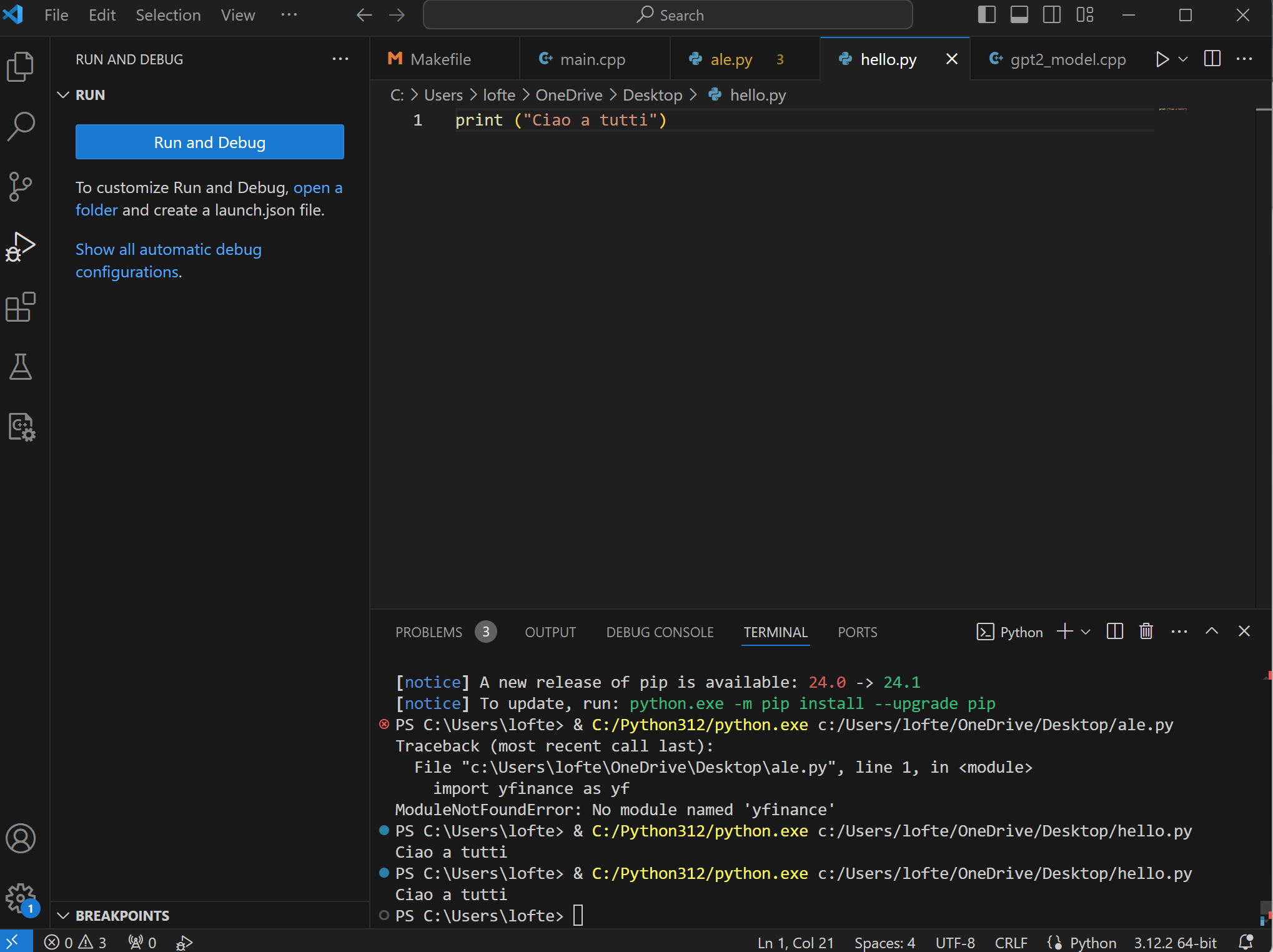
tramite un terminale o prompt dei comandi, naviga alla directory dove hai salvato il file. Digita python hello.py e premi Invio.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Tramite gli editor puoi premere sul pulsante *RUN*





L’output del programma sarà semplicemente la scritta a video “Ciao a tutti”.

Questa semplice riga di codice dimostra come eseguire una stampa a schermo, una delle funzioni più basilari di Python.

Conclusione della Sezione

Prova a scrivere una frase su due righe:

Link delle soluzioni si github: <https://github.com/iacopoooo/Python_book>

Nota: Ogni esempio o immagine sono numerati in base al capitolo

***2. Fondamenti di Programmazione in Python***

Variabili e Tipi di Dati

In Python, una variabile è un contenitore per memorizzare dati. Non è necessario dichiarare il tipo di una variabile in anticipo, cosa che rende Python un linguaggio con tipizzazione dinamica.

* **Numeri**: interi (int), numeri a virgola mobile (float), numeri complessi (complex).
* **Stringhe**: testo racchiuso tra virgolette singole o doppie.
* **Booleani**: True o False.
* **Liste**: sequenze ordinate di elementi, racchiuse tra parentesi quadre.
* **Tuple**: sequenze ordinate e immutabili di elementi, racchiuse tra parentesi tonde.
* **Set**: collezioni non ordinate di elementi unici, racchiuse tra parentesi graffe.
* **Dizionari**: collezioni di coppie chiave-valore, racchiuse tra parentesi graffe.

Esempio di codice:

# abbiamo dichiarato le variabili  
nome = "Mario"

eta = 30

altezza = 1.75

italiano = True

**Operatori** Gli operatori sono simboli che eseguono operazioni su variabili e valori. Alcuni operatori comuni in Python includono:

* **Operatori aritmetici**: +, -, \*, /, // (divisione intera), % (modulo), \*\* (esponenziazione).
* **Operatori di confronto**: ==, !=, >, <, >=, <=.
* **Operatori logici**: and, or, not.
* **Operatori di assegnazione**: =, +=, -=, \*=, /=, %=, \*\*=, //=.
* **Operatori di appartenenza**: in, not in.

Strutture di Controllo

Le strutture di controllo permettono di dirigere il flusso di esecuzione del programma.

Condizionali **(if, else, elif):** Permettono di eseguire blocchi di codice basati su condizioni specifiche.

Cicli **(for, while):** Utilizzati per ripetere l'esecuzione di un blocco di codice.

Esempi di codice:

# Condizioni

print("condizioni:")

if eta > 18:

print("Sei maggiorenne")

else:

print("Non sei maggiorenne")

# Cicli

print ("\nciclo for:")

for i in range(5):

print(i)

j = 0  
print("\nciclo while:")

while j < 5:

print(j)

j += 1

Vi starete chiedendo il perché di alcune istruzioni scritte con il rientro. Beh ve lo spiego subito.

**Indentazione**: python si serve dell’ *indentazione*, che non è altro che il rientro, per definire i blocchi di codice. E’ l’equivalente di altri linguaggi delle graffe.

Pseudocodice in C:

if (eta > 18) {

printf("Sei maggiorenne.\n");

} else {

printf("Sei minorenne.\n");

}

Codice equivalente Python

print("condizioni:")

if eta > 18:

print("Sei maggiorenne")

else:

print("Non sei maggiorenne")

Al link github avete questi esercizi.

Qui di seguito ricapitoliamo quanto appreso fin ora con alcuni esempi spiegati in dettaglio:

#### Strutture di Controllo

**Condizioni (if, else, elif)** Le strutture condizionali permettono al programma di eseguire diverse sezioni di codice in base a determinate condizioni.

x = 10

if x > 0:

print("x è positivo")

elif x == 0:

print("x è zero")

else:

print("x è negativo")

### Spiegazione del codice

1. **Assegnazione di una variabile**:

x = 10

In questa linea, stiamo assegnando il valore 10 alla variabile x. Questo significa che ogni volta che useremo x nel nostro programma, ci stiamo riferendo al numero 10.

1. **Struttura di controllo condizionale if**:

if x > 0:

print("x è positivo")

Qui, stiamo verificando se il valore di x è maggiore di 0. Se questa condizione è vera, il programma eseguirà il codice indentato sotto l'istruzione if. In questo caso, stamperà:

x è positivo

Poiché x è 10, questa condizione è vera, quindi questa parte del codice verrà eseguita.

1. **Struttura di controllo condizionale elif**:

elif x == 0:

print("x è zero")

elif sta per "else if". Questa parte del codice verrà eseguita se la condizione if precedente è falsa e se la condizione elif è vera. In questo caso, stiamo controllando se x è uguale a 0. Se x fosse 0, il programma stamperebbe:

x è zero

Poiché x è 10, questa condizione è falsa, quindi questa parte del codice non verrà eseguita.

1. **Struttura di controllo condizionale else**:

else:

print("x è negativo")

L'istruzione else copre tutti gli altri casi che non sono stati catturati dalle condizioni if ed elif precedenti. Se x non è maggiore di 0 e non è uguale a 0, allora x deve essere negativo. In questo caso, il programma stamperebbe:

x è negativo

Poiché x è 10, questa condizione non verrà eseguita.

### Riepilogo

Il programma verifica il valore della variabile x e stampa un messaggio in base al suo valore:

* Se x è maggiore di 0, stampa "x è positivo".
* Se x è uguale a 0, stampa "x è zero".
* Se x è minore di 0, stampa "x è negativo".

Dato che nel codice fornito x è assegnato a 10, l'output sarà:

x è positivo

Questo semplice esempio di struttura condizionale mostra come utilizzare if, elif e else in Python per eseguire diverse operazioni basate sul valore di una variabile.

**Cicli (for, while)** I cicli permettono di eseguire ripetutamente una sezione di codice fino a quando una condizione è vera.

* **Ciclo for**: utilizzato per iterare su una sequenza (lista, tupla, stringa, ecc.).

for i in range(5):

print(i)

* **Ciclo while**: esegue il blocco di codice finché la condizione è vera.

count = 0

while count < 5:

print(count)

count += 1

### Spiegazione del codice

1. **Inizializzazione di una variabile**:

count = 0

Qui, stiamo assegnando il valore 0 alla variabile count. Questo significa che ogni volta che useremo count nel nostro programma, ci stiamo riferendo al numero 0.

1. **Struttura di controllo ciclica while**:

while count < 5:

Il ciclo while continua a eseguire il blocco di codice indentato al suo interno finché la condizione specificata (count < 5) è vera. In questo caso, il ciclo continuerà finché il valore di count è inferiore a 5 .

1. **Stampa del valore di count**:

print(count)

All'interno del ciclo while, il programma stampa il valore attuale di count.

1. **Incremento della variabile count**:

count += 1

Questa linea di codice incrementa il valore di count di 1. L'operatore += è una scorciatoia per count = count + 1.

### Esecuzione del codice passo dopo passo

Vediamo cosa succede ad ogni iterazione del ciclo while:

* **Prima iterazione**:
  + count è 0, che è inferiore a 5.
  + Stampa 0.
  + Incrementa count a 1.
* **Seconda iterazione**:
  + count è 1, che è inferiore a 5.
  + Stampa 1.
  + Incrementa count a 2.
* **Terza iterazione**:
  + count è 2, che è inferiore a 5.
  + Stampa 2.
  + Incrementa count a 3.
* **Quarta iterazione**:
  + count è 3, che è inferiore a 5.
  + Stampa 3.
  + Incrementa count a 4.
* **Quinta iterazione**:
  + count è 4, che è inferiore a 5.
  + Stampa 4.
  + Incrementa count a 5.

Alla fine della quinta iterazione, count diventa 5, quindi la condizione count < 5 non è più vera, e il ciclo while termina.

### Output del codice

L'output del programma sarà:

0

1

2

3

4

### Riepilogo

Il programma utilizza un ciclo while per stampare i numeri da 0 a 4. Ogni iterazione del ciclo:

* Controlla se count è inferiore a 5.
* Stampa il valore attuale di count.
* Incrementa count di 1.

Il ciclo si interrompe quando count raggiunge 5, poiché la condizione count < 5 non è più soddisfatta.

#### 5. Funzioni

**Definizione di funzioni** Le funzioni in Python sono definite usando la parola chiave def, seguita dal nome della funzione e da parentesi tonde che possono contenere parametri.

Qui di seguito trovate il link alle principali funzioni della libreria standard di python: <https://docs.python.org/3/library/functions.html>

**print()**: Stampa l'output su console.

print("Hello, World!")

**len()**: Restituisce la lunghezza (numero di elementi) di un oggetto.

lista = [1, 2, 3, 4]

print(len(lista)) # Output: 4

**range()**: Genera una sequenza di numeri, spesso utilizzata nei cicli for.

for i in range(5):

print(i) # stampa i numeri da 0 a 4

**type()**: Restituisce il tipo di un oggetto.

variabile = 5.10

print(type(variabile)) # Output: <class 'float'>

**int(), float(), str()**: Converte un valore in intero, float o stringa rispettivamente.

num = "10"

print(int(num)) # Output: 10

print(float(num)) # Output: 10.0

print(str(num)) # Output: "10"

**input()**: Permette di prendere input dall'utente.

nome = input("Come ti chiami? ")

print("Ciao, " + nome)

#Output: Come ti chiami?

#Aspetta l’input dell’utente adesempio: Iacopo

# ritorna: Ciao, Iacopo

**sum()**: Restituisce la somma di un iterabile.

numeri = [1, 2, 3, 4]

print(sum(numeri)) # Output: 10

**max() e min()**: Restituiscono rispettivamente il valore massimo e minimo di un iterabile.

numeri = [1, 2, 3, 4]

print(max(numeri)) # Output: 4

print(min(numeri)) # Output: 1

**sorted()**: Restituisce una lista ordinata di un iterabile.

numeri = [3, 1, 4, 1, 5]

print(sorted(numeri)) # Output: [1, 1, 3, 4, 5]

**open()**: Apre un file e restituisce un oggetto file.

with open('your\_file.txt', 'r') as file:

contenuto = file.read()

print(contenuto)# Stampa il contenuto del file your\_file.txt

Esempio di utilizzo di una funzione in python:

def saluta():

print("Ciao!")

### Spiegazione del codice

1. **Definizione di una funzione**:

def saluta():

Questa linea di codice definisce una nuova funzione chiamata saluta. La parola chiave def viene utilizzata per dichiarare una funzione in Python. Il nome della funzione è saluta, e le parentesi () indicano che non ci sono parametri in ingresso per questa funzione. Il carattere : indica l'inizio del blocco di codice che costituisce il corpo della funzione.

1. **Corpo della funzione**:

print("Ciao!")

Questa linea di codice è indentata sotto la definizione della funzione, il che significa che fa parte del corpo della funzione saluta. Il corpo della funzione contiene il codice che viene eseguito quando la funzione viene chiamata. In questo caso, il corpo della funzione consiste in una singola istruzione print che stampa la stringa "Ciao!" sulla console.

### Esempio di chiamata della funzione

Per vedere come funziona la funzione, dobbiamo chiamarla nel nostro programma. Ecco come farlo:

saluta()

Quando eseguiamo questa chiamata, il programma eseguirà il corpo della funzione saluta, che stamperà "Ciao!" sulla console.

### Output del codice

Quando la funzione saluta viene chiamata, l'output sarà:

Ciao!

### Riepilogo

* **Definizione della funzione**: La funzione saluta è definita utilizzando la parola chiave def seguita dal nome della funzione e da parentesi vuote, dato che non prende alcun argomento.
* **Corpo della funzione**: Contiene una singola istruzione print che stampa "Ciao!" sulla console.
* **Chiamata della funzione**: Quando la funzione saluta viene chiamata, l'istruzione print all'interno del corpo della funzione viene eseguita, risultando nell'output "Ciao!".

Questo semplice esempio mostra come definire e chiamare una funzione in Python, permettendo di raggruppare il codice in blocchi riutilizzabili.

Ora aggiungiamo un parametro alla funzione:

def saluta(nome):

"""

Questa funzione prende un *nome* come *parametro* e stampa un saluto personalizzato.

Parametri:

nome (str): Il nome della persona da salutare.

"""

print(f"Ciao, {nome}!")

# Esempio di utilizzo

saluta("Giovanna")# stampa Ciao, Giovanna

saluta("Amilcare") # stampa Ciao, Amilcare

“””Giovanna e Amilcare sono gli argomenti della funzione”””

**Nota Bene:**

* **Parametro**: Nella funzione saluta(nome), nome è il parametro. Esso agisce come una variabile locale all'interno della funzione, che sarà utilizzata per riferirsi al valore passato durante la chiamata della funzione.
* **Argomento**: Quando chiamiamo la funzione saluta("Giovanna o Amilcare"), "Giovanna e Amilcare" sono l'argomento. È il valore che viene passato al parametro nome.

**Più semplicemente nella definizione indichiamo i PARAMETRI, nella chiamata passiamo gli ARGOMENTI**

**Parametri e valori di ritorno** Le funzioni possono accettare parametri e restituire valori.

def somma(a, b): # a, b sono parametri

return a + b

risultato = somma(3, 4)# 3, 4 sono argomenti

print(risultato) # Output: 7

### Spiegazione del codice

1. **Definizione di una funzione con parametri**:

def somma(a, b):

Questa linea di codice definisce una nuova funzione chiamata somma. La parola chiave def viene utilizzata per dichiarare una funzione in Python. Il nome della funzione è somma, e le parentesi () contengono due parametri a e b. Questi parametri permettono di passare valori alla funzione quando viene chiamata.

1. **Corpo della funzione**:

return a + b

Questa linea di codice è indentata sotto la definizione della funzione, indicando che fa parte del corpo della funzione somma. L'istruzione return restituisce la somma dei valori a e b. Quando la funzione viene chiamata, il risultato dell'operazione a + b verrà restituito al punto in cui la funzione è stata chiamata.

1. **Chiamata della funzione e assegnazione del risultato**:

risultato = somma(3, 4)

Questa linea di codice chiama la funzione somma con gli argomenti 3 e 4. I valori 3 e 4 vengono passati ai parametri a e b rispettivamente. La funzione calcola a + b, che è 3 + 4, e restituisce 7. Questo valore restituito viene assegnato alla variabile risultato.

1. **Stampa del risultato**:

print(risultato) # Output: 7

Questa linea di codice stampa il valore della variabile risultato sulla console. Poiché risultato contiene 7, l'output sarà 7.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. La funzione somma viene definita con due parametri a e b.
2. La funzione viene chiamata con i valori 3 e 4, quindi a è 3 e b è 4.
3. La funzione esegue return a + b, che calcola 3 + 4 e restituisce 7.
4. Il valore restituito 7 viene assegnato alla variabile risultato.
5. Il valore di risultato viene stampato sulla console, producendo l'output 7.

### Output del codice

Quando il codice viene eseguito, l'output sarà:

7

### Riepilogo

* **Definizione della funzione**: La funzione somma è definita con due parametri a e b.
* **Corpo della funzione**: Contiene un'istruzione return che restituisce la somma di a e b.
* **Chiamata della funzione**: La funzione somma viene chiamata con i valori 3 e 4, e il risultato 7 viene assegnato alla variabile risultato.
* **Stampa del risultato**: La variabile risultato viene stampata sulla console, mostrando 7.

Questo esempio mostra come definire una funzione con parametri, restituire un valore e utilizzare il risultato della funzione in Python.

**Funzioni lambda** Le funzioni lambda sono piccole funzioni anonime definite con la parola chiave lambda.

doppio = lambda x: x \* 2

print(doppio(5)) # Output: 10

### Spiegazione del codice

1. **Definizione di una funzione lambda**:

doppio = lambda x: x \* 2

In questa linea di codice, stiamo definendo una funzione lambda e assegnandola alla variabile doppio. Le funzioni lambda in Python sono funzioni anonime, cioè funzioni senza un nome, definite utilizzando la parola chiave lambda. La sintassi generale di una funzione lambda è:

lambda argomento1, argomento2, ... : espressione

In questo caso, lambda x: x \* 2 definisce una funzione che prende un argomento x e restituisce x \* 2. Questa funzione viene assegnata alla variabile doppio.

1. **Chiamata della funzione lambda e stampa del risultato**:

print(doppio(5)) # Output: 10

Questa linea di codice chiama la funzione doppio con l'argomento 5. La funzione doppio esegue l'espressione x \* 2 con x uguale a 5, quindi calcola 5 \* 2 e restituisce 10. Il valore restituito 10 viene quindi passato alla funzione print, che lo stampa sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. La funzione lambda lambda x: x \* 2 viene definita e assegnata alla variabile doppio.
2. La funzione doppio viene chiamata con il valore 5, quindi x è uguale a 5.
3. La funzione lambda esegue l'operazione 5 \* 2 e restituisce 10.
4. Il valore restituito 10 viene stampato sulla console.

### Output del codice

Quando il codice viene eseguito, l'output sarà:

10

### Riepilogo

* **Definizione della funzione lambda**: La funzione lambda lambda x: x \* 2 è definita e assegnata alla variabile doppio. Questa funzione prende un argomento x e restituisce il doppio di x.
* **Chiamata della funzione lambda**: La funzione doppio viene chiamata con l'argomento 5. La funzione calcola 5 \* 2 e restituisce 10.
* **Stampa del risultato**: Il valore 10 viene stampato sulla console.

Le funzioni lambda sono utili per definire funzioni semplici in modo conciso e possono essere utilizzate ovunque sia necessaria una funzione.

Esercizi di Pratica

Scrivi una funzione che calcoli e restituisca la media di una lista di numeri.

Crea un modulo che contenga funzioni per gestire operazioni comuni su liste, come trovare il minimo, il massimo e la media dei valori.

Nota: l’esercizio 1 lo trovate nel fle del captolo, mentre l’esercizio 2 lo trovate nei file esercizio2.py e modulo\_lista.py. Inoltre quando in python create un modulo personalizzato o lo richiamate in uno script verrà creato un file come quello rappresentato nell’immagine denominata screenshot.png. Questo tipo di file viene generato automaticamente da Python quando un modulo viene importato, per velocizzare le successive esecuzioni.

Conclusione

L'apprendimento di come definire e usare le funzioni e i moduli è cruciale per scrivere codice Python pulito e manutenibile. Nel prossimo modulo, possiamo esplorare la gestione delle eccezioni e la programmazione di interfacce grafiche con Tkinter, espandendo ulteriormente le tue competenze in Python.

**Directory e File System**

In questa sezione vi mostro rapidamente attraverso alcuni semplici esempi come maneggiare i file in python:

Ecco alcuni semplici esempi su come gestire file, cartelle e il file system in Python utilizzando il modulo os e il modulo shutil.

Il modulo os in Python fornisce un modo per interagire con il sistema operativo del computer (windows, linux, mac). Questo modulo consente di eseguire una varietà di operazioni legate al file system, alla gestione dei processi, alla manipolazione dei file, dei percorsi e molto altro ancora.

Il modulo shutil in Python fornisce funzioni di alto livello per copiare e rimuovere file e directory .

*GESTIONE DEI FILE E OPERAZIONI CON LE DIRECTORY*

**1. Creare una cartella**

import os

# Creare una nuova cartella

os.mkdir('nuova\_cartella')

**2. Verificare se una cartella esiste**

import os

# Verificare se una cartella esiste

if os.path.exists('nuova\_cartella'):

print("La cartella esiste")

else:

print("La cartella non esiste")

**3. Rinominare una cartella**

import os

# Rinominare una cartella

os.rename('nuova\_cartella', 'cartella\_rinominata')

**4. Eliminare una cartella**

import os

# Eliminare una cartella vuota

os.rmdir('cartella\_rinominata')

**5. Creare un file e scrivere in esso**

# Creare e aprire un file per scrivere

with open('file\_di\_prova.txt', 'w') as file:

file.write('Questo è un file di prova.\n')

file.write('Scrittura di più righe.\n')

**6. Leggere da un file**

# Aprire un file per leggere

with open('file\_di\_prova.txt', 'r') as file:

contenuto = file.read()

print(contenuto)

**7. Aggiungere dati a un file**

# Aprire un file per aggiungere

with open('file\_di\_prova.txt', 'a') as file:

file.write('Aggiunta di un\'altra riga.\n')

**8. Copiare un file**

import shutil

# Copiare un file

shutil.copy('file\_di\_prova.txt', 'file\_copiato.txt')

**9. Spostare un file**

import shutil

# Spostare un file

shutil.move('file\_copiato.txt', 'cartella\_destinazione/file\_copiato.txt')

**10. Eliminare un file**

import os

# Eliminare un file

os.remove('file\_di\_prova.txt')

**11. Elencare file in una cartella**

import os

# Elencare i file in una cartella

file\_nella\_cartella = os.listdir('.')

print(file\_nella\_cartella)

**12. Creare una struttura di directory**

import os

# Creare una struttura di directory

os.makedirs('struttura/cartella1/cartella2')

**13. Rimuovere una struttura di directory**

import shutil

# Rimuovere una struttura di directory

shutil.rmtree('struttura')

Per ora abbiamo lavoro con file .txt, ma vi starete chiedendo: “se devo lavorare con un file pdf o exel?”

In Python avete delle librerie specifiche per fare questo; per i pdf una delle più popolari è **PyPDF2**, e per i file exel è **openpyxl**.

Prima di tutto installate le librerie dal terminale con il seguente comando: *pip install PyPDF2* (per pdf) e *pip install openpyxl* (per exel)

Nota:Se state lavorando con pycharme (vedi Fig.2) o visual studio, dovrete inserire il comando nel terminale del programma, se invece state lavorando con l’editor predefinito di python dal terminale (Fig.1) vi dovrete recare nella cartella dove avete installato python ed eseguire il comando.

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

Fig.1 terminale stadard di windows per chi lavora con l’editor predefinito di python

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Fig. 2 il terminale di pycharme

In questo snippet avete una panoramica di come lavorare con i pdf:

import PyPDF2  
  
# Aprire un file PDF

with open(your\_document.pdf', 'rb') as file:

reader = PyPDF2.PdfFileReader(file)  
  
 # Ottenere il numero di pagine

numero\_pagine = reader.numPages

print(f"Numero di pagine: {numero\_pagine}")  
  
 # Estrarre il testo dalla prima pagina

prima\_pagina = reader.getPage(0)

testo = prima\_pagina.extractText()

print(testo)

E questo è un esempio di exel:

import openpyxl  
  
# Caricare il file Excel

workbook = openpyxl.load\_workbook('file.xlsx')  
  
# Selezionare un foglio

foglio = workbook['Foglio1']  
  
# Leggere una cella

valore = foglio['A1'].value

print(f"Il valore della cella A1 è: {valore}")  
  
# Iterare su tutte le righe e colonne

for riga in foglio.iter\_rows(values\_only=True):

print(riga)

Se invece state lavorando con i csv (altra esetnsione dei file exel) dovrete usare il modulo csv che è già integrato nella libreria stabdard di python; questo significa che non dovrete installare niente tramite il comando pip.

import csv  
  
# Leggere e stampare il contenuto del file CSV

with open('esempio.csv', mode='r') as file:

reader = csv.reader(file)  
  
 # Iterare attraverso le righe del CSV

for row in reader:

print(row) #stampa il contenuto del csv

Approfondimenti per leggere e scrivere i file:

Lavorare con file in Python è semplice grazie alle funzioni integrate open, read, write, e close. Ecco come leggere e scrivere file di *testo*:

* **Scrivere un file**

# Aprire un file in modalità scrittura

with open('example.txt', 'w') as file:

file.write('Questo è un esempio di scrittura su file.')

### Spiegazione del codice

1. **Apertura del file con open**:

with open('example.txt', 'w') as file:

* + **with Statement**: Il with statement in Python viene utilizzato per gestire le risorse come i file. Garantisce che il file venga chiuso correttamente dopo che il blocco di codice interno viene eseguito, anche se si verifica un'eccezione.
  + **open Function**: La funzione open viene utilizzata per aprire un file. Prende due argomenti principali: il nome del file ('example.txt') e la modalità di apertura ('w' per scrittura).
    - 'w' (write mode): Apre il file per la scrittura. Se il file esiste già, viene troncato. Se il file non esiste, viene creato un nuovo file.
  + **Assegnazione a file**: Il file aperto viene assegnato alla variabile file, che rappresenta l'oggetto file su cui possiamo eseguire operazioni di scrittura.

1. **Scrittura nel file**:

file.write('Questo è un esempio di scrittura su file.')

* + **write Method**: Il metodo write dell'oggetto file scrive una stringa nel file. In questo caso, la stringa 'Questo è un esempio di scrittura su file.' viene scritta nel file example.txt.
  + **Buffering**: La scrittura su file potrebbe non avvenire immediatamente, ma essere memorizzata temporaneamente in un buffer. Il with statement assicura che tutto il contenuto del buffer venga scritto su disco quando il blocco viene eseguito.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Apertura del file**:
   * Il file example.txt viene aperto in modalità scrittura. Se il file esiste già, il suo contenuto viene cancellato. Se il file non esiste, viene creato un nuovo file.
   * L'oggetto file viene assegnato alla variabile file.
2. **Scrittura nel file**:
   * La stringa 'Questo è un esempio di scrittura su file.' viene scritta nel file example.txt utilizzando il metodo write.
3. **Chiusura del file**:
   * Quando il blocco with termina, il file viene automaticamente chiuso, assicurando che tutti i dati vengano scritti correttamente su disco.

### Output del codice

Non c'è un output visibile sulla console perché il codice scrive una stringa su un file. Tuttavia, se apriamo il file example.txt, vedremo che contiene il seguente testo:

Questo è un esempio di scrittura su file.

### Riepilogo

* **Apertura del file con open**: Il file example.txt viene aperto in modalità scrittura ('w'), creando il file se non esiste o cancellando il contenuto se esiste già.
* **Scrittura nel file**: La stringa 'Questo è un esempio di scrittura su file.' viene scritta nel file utilizzando il metodo write.
* **Gestione delle risorse con with**: Il with statement garantisce che il file venga chiuso correttamente dopo la scrittura, assicurando che tutti i dati vengano scritti su disco.

In questo esempio, abbiamo visto come scrivere una stringa su un file utilizzando Python, aprendo il file in modalità scrittura e gestendo correttamente le risorse con il with statement.

* **Leggere un file**

# Aprire un file in modalità lettura

with open('example.txt', 'r') as file:

content = file.read()

print(content)

### Spiegazione del codice

1. **Apertura del file in modalità lettura**:

with open('example.txt', 'r') as file:

* + **with Statement**: Il with statement in Python viene utilizzato per gestire le risorse come i file. Garantisce che il file venga chiuso correttamente dopo che il blocco di codice interno viene eseguito, anche se si verifica un'eccezione.
  + **open Function**: La funzione open viene utilizzata per aprire un file. Prende due argomenti principali: il nome del file ('example.txt') e la modalità di apertura ('r' per lettura).
    - 'r' (read mode): Apre il file per la lettura. Se il file non esiste, viene generata un'eccezione FileNotFoundError.
  + **Assegnazione a file**: Il file aperto viene assegnato alla variabile file, che rappresenta l'oggetto file su cui possiamo eseguire operazioni di lettura.

1. **Lettura del contenuto del file**:

content = file.read()

* + **read Method**: Il metodo read dell'oggetto file legge tutto il contenuto del file e lo restituisce come una stringa. Questo contenuto viene assegnato alla variabile content.

1. **Stampa del contenuto del file**:

print(content)

* + **print Function**: La funzione print stampa il valore della variabile content sulla console. In questo caso, stampa il contenuto del file example.txt.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Apertura del file**:
   * Il file example.txt viene aperto in modalità lettura. Se il file non esiste, viene generata un'eccezione FileNotFoundError.
   * L'oggetto file viene assegnato alla variabile file.
2. **Lettura del contenuto del file**:
   * Il metodo read legge tutto il contenuto del file e lo assegna alla variabile content.
3. **Stampa del contenuto del file**:
   * La funzione print stampa il contenuto della variabile content sulla console.

### Output del codice

L'output sarà il contenuto del file example.txt. Se il file contiene la stringa:

Questo è un esempio di scrittura su file.

Allora l'output sarà:

Questo è un esempio di scrittura su file.

### Riepilogo

* **Apertura del file in modalità lettura**: Il file example.txt viene aperto in modalità lettura ('r').
* **Lettura del contenuto del file**: Il metodo read legge tutto il contenuto del file e lo assegna alla variabile content.
* **Stampa del contenuto del file**: La funzione print stampa il contenuto della variabile content sulla console.

In questo esempio, abbiamo visto come leggere il contenuto di un file utilizzando Python, aprendo il file in modalità lettura e gestendo correttamente le risorse con il with statement.

**Gestione dei file CSV e JSON** Python offre moduli specifici per lavorare con file CSV e JSON, che sono formati comuni per la memorizzazione e lo scambio di dati.

* **CSV**

import csv

# Scrivere un file CSV

with open('example.csv', mode='w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['Nome', 'Età', 'Città'])

writer.writerow(['Alice', 30, 'Roma'])

writer.writerow(['Bob', 25, 'Milano'])

# Leggere un file CSV

with open('example.csv', mode='r') as file:

reader = csv.reader(file)

for row in reader:

print(row)

### Spiegazione del codice

1. **Importazione del modulo csv**:

import csv

Questa linea di codice importa il modulo csv, che fornisce funzionalità per leggere e scrivere file CSV (Comma Separated Values) in Python.

1. **Scrittura di un file CSV**:

with open('example.csv', mode='w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['Nome', 'Età', 'Città'])

writer.writerow(['Alice', 30, 'Roma'])

writer.writerow(['Bob', 25, 'Milano'])

* + **Apertura del file**:

with open('example.csv', mode='w', newline='') as file:

Il file example.csv viene aperto in modalità scrittura ('w'). L'argomento newline='' assicura che le nuove linee vengano gestite correttamente su tutte le piattaforme.

* + **Creazione del writer**:

writer = csv.writer(file)

Il metodo csv.writer crea un oggetto writer che verrà utilizzato per scrivere righe nel file CSV.

* + **Scrittura delle righe**:

writer.writerow(['Nome', 'Età', 'Città'])

writer.writerow(['Alice', 30, 'Roma'])

writer.writerow(['Bob', 25, 'Milano'])

Il metodo writer.writerow scrive una riga nel file CSV. Le righe scritte contengono i dati delle colonne.

1. **Lettura di un file CSV**:

with open('example.csv', mode='r') as file:

reader = csv.reader(file)

for row in reader:

print(row)

* + **Apertura del file**:

with open('example.csv', mode='r') as file:

Il file example.csv viene aperto in modalità lettura ('r').

* + **Creazione del reader**:

reader = csv.reader(file)

Il metodo csv.reader crea un oggetto reader che verrà utilizzato per leggere le righe dal file CSV.

* + **Iterazione sulle righe**:

for row in reader:

print(row)

Un ciclo for viene utilizzato per iterare su ogni riga nel file CSV. Ogni riga viene stampata sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Scrittura di un file CSV**:
   * Il file example.csv viene aperto in modalità scrittura.
   * Viene creato un oggetto writer.
   * Vengono scritte tre righe nel file CSV:

Nome,Età,Città

Alice,30,Roma

Bob,25,Milano

* + Il file viene chiuso automaticamente al termine del blocco with.

1. **Lettura di un file CSV**:
   * Il file example.csv viene aperto in modalità lettura.
   * Viene creato un oggetto reader.
   * Ogni riga del file CSV viene letta e stampata sulla console.

### Output del codice

L'output sarà:

css

['Nome', 'Età', 'Città']

['Alice', 30, 'Roma']

['Bob', 25, 'Milano']

### Riepilogo

* **Scrittura di un file CSV**:
  + Il file example.csv viene aperto in modalità scrittura.
  + Le righe vengono scritte nel file CSV utilizzando csv.writer e writer.writerow.
* **Lettura di un file CSV**:
  + Il file example.csv viene aperto in modalità lettura.
  + Le righe vengono lette dal file CSV utilizzando csv.reader e iterate con un ciclo for.

In questo esempio, abbiamo visto come scrivere e leggere un file CSV utilizzando il modulo csv di Python, gestendo correttamente l'apertura e la chiusura del file con il with statement.

Inizio modulo

Fine modulo

* **JSON**

import json

data = {

'name': 'Alice',

'age': 30,

'city': 'Roma'

}

# Scrivere un file JSON

with open('example.json', 'w') as file:

json.dump(data, file)

# Leggere un file JSON

with open('example.json', 'r') as file:

data = json.load(file)

print(data)

### Spiegazione del codice

1. **Importazione del modulo json**:

import json

Questa linea di codice importa il modulo json, che fornisce funzionalità per lavorare con dati JSON (JavaScript Object Notation) in Python.

1. **Definizione dei dati**:

data = {

'name': 'Alice',

'age': 30,

'city': 'Roma'

}

Questa linea di codice definisce un dizionario Python chiamato data. Il dizionario contiene tre coppie chiave-valore: name (nome), age (età), e city (città).

1. **Scrittura di un file JSON**:

with open('example.json', 'w') as file:

json.dump(data, file)

* + **Apertura del file**:

with open('example.json', 'w') as file:

Il file example.json viene aperto in modalità scrittura ('w'). Se il file non esiste, viene creato un nuovo file. Se il file esiste già, il suo contenuto viene cancellato.

* + **Scrittura dei dati**:

json.dump(data, file)

La funzione json.dump scrive il contenuto del dizionario data nel file example.json in formato JSON.

1. **Lettura di un file JSON**:

with open('example.json', 'r') as file:

data = json.load(file)

print(data)

* + **Apertura del file**:

with open('example.json', 'r') as file:

Il file example.json viene aperto in modalità lettura ('r').

* + **Lettura dei dati**:

data = json.load(file)

La funzione json.load legge il contenuto del file example.json e lo converte in un dizionario Python. Questo dizionario viene assegnato alla variabile data.

* + **Stampa dei dati**:

print(data)

La funzione print stampa il contenuto della variabile data sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Scrittura di un file JSON**:
   * Il file example.json viene aperto in modalità scrittura.
   * Il contenuto del dizionario data viene scritto nel file example.json in formato JSON.
   * Il file viene chiuso automaticamente al termine del blocco with.
2. **Lettura di un file JSON**:
   * Il file example.json viene aperto in modalità lettura.
   * Il contenuto del file example.json viene letto e convertito in un dizionario Python.
   * Il contenuto del dizionario viene stampato sulla console.

### Output del codice

L'output sarà:

arduino

{'name': 'Alice', 'age': 30, 'city': 'Roma'}

### Riepilogo

* **Scrittura di un file JSON**:
  + Il file example.json viene aperto in modalità scrittura.
  + I dati vengono scritti nel file in formato JSON utilizzando json.dump.
* **Lettura di un file JSON**:
  + Il file example.json viene aperto in modalità lettura.
  + I dati vengono letti dal file e convertiti in un dizionario Python utilizzando json.load.
  + Il contenuto del dizionario viene stampato sulla console.

In questo esempio, abbiamo visto come scrivere e leggere un file JSON utilizzando il modulo json di Python, gestendo correttamente l'apertura e la chiusura del file con il with statement.

Questi sono esempi molto semplici su come lavorare con i file pdf , csv ed exel. Per maggiori informazioni nella community troverete centinaia di tutorial che potrebbero fare al caso vostro.

Questi esempi coprono le operazioni di base per la gestione di file e cartelle in Python

Conclusione

Ora avete una panoramica abbastanza ampia su come gestire e lavorare con i file in python.

***Gestione delle Eccezioni in Python***

Cosa Sono le Eccezioni?

Le eccezioni sono eventi che possono modificare il flusso normale del programma, generalmente a seguito di errori.

Quando si verifica un errore, Python interrompe l'esecuzione normale e genera un'eccezione. Se non viene gestita, l'eccezione causa il termine del programma con un messaggio di errore. Tuttavia, puoi scrivere del codice per catturare e gestire le eccezioni, consentendo al programma di continuare a funzionare in modo controllato.

Python ha diversi tipi di eccezioni incorporati come ValueError, TypeError, IndexError, e molti altri.

Gestire Eccezioni con try e except

Il blocco **try** ti permette di testare un blocco di codice per errori. Il blocco **except** ti permette di gestire l'errore.

Esempio: Gestione di Eccezioni

try:

numero = int(input("Inserisci un numero: "))

except ValueError:

print("Errore: inserisci un valore numerico.")

else:

print(f"Hai inserito il numero {numero}")

Usare **finally**

Il blocco finally è un posto dove puoi mettere il codice che vuoi eseguire sempre, indipendentemente dal risultato dei blocchi try e except.

Esempio con finally

try:

f = open("file\_non\_esistente.txt")

f.read()

except FileNotFoundError:

print("Errore: File non trovato.")

finally:

print("Questa parte viene sempre eseguita.")

Ora vi mostro un esempio completo con l’utilizzo try, except, else, e finally per illustrarti come gestire le eccezioni in Python:

print("Esempio con la gestione delle eccezioni")

def dividi(numeratore, denominatore):

try:

risultato = numeratore / denominatore

except ZeroDivisionError:

print("Errore: divisione per zero.")

except TypeError:

print("Errore: entrambi i valori devono essere numeri.")

else:

print(f"Risultato: {risultato}")

finally:

print("Esecuzione completata.")

# Esempi di utilizzo della funzione dividi

dividi(10, 2) # Caso senza eccezioni

dividi(10, 0) # Caso con ZeroDivisionError

dividi(10, 'a') # Caso con TypeError  
  
print("\n==============================\n")  
  
#uso senza la gestione delle eccezioni

print("Esempio senza gestire le eccezioni")

def dividi(numeratore, denominatore):  
  
 risultato = numeratore / denominatore

return risultato

print(dividi(10, 2)) #cosi funziona

#print(dividi(10, 0)) # se decommentate genererà errroe

**Spiegazione del Codice:**

Definizione di funzione

def dividi(numeratore, denominatore):

Definiamo una funzione dividi che prende due argomenti: numeratore e denominatore.

Blocco try:

try:

risultato = numeratore / denominatore

Nel blocco try, eseguiamo l'operazione di divisione. Questo è il codice che potrebbe generare un'eccezione.

Blocchi except:

except ZeroDivisionError:

print("Errore: divisione per zero.")

except TypeError:

print("Errore: entrambi i valori devono essere numeri.")

Ecco cosa avviene nei due blocchi except:

* Il primo cattura l'eccezione ZeroDivisionError, che si verifica se il denominatore è zero.
* Il secondo cattura l'eccezione TypeError, che si verifica se uno dei due valori non è un numero (ad esempio, se denominatore è una stringa).

Blocco else:

else:

print(f"Risultato: {risultato}")

Il blocco else viene eseguito solo se il blocco try non genera alcuna eccezione. In questo caso, stampa il risultato della divisione.

Blocco finally:

finally:

print("Esecuzione completata.")

Il blocco finally viene sempre eseguito, indipendentemente dal fatto che si verifichi un'eccezione o meno. Qui, stampa "Esecuzione completata."

Esecuzione del codice:

dividi(10, 2):

* Il blocco try esegue la divisione senza errori.
* Il blocco else viene eseguito, stampando il risultato della divisione.
* Il blocco finally viene eseguito, stampando "Esecuzione completata."

dividi(10, 0):

* Il blocco try genera un ZeroDivisionError.
* Il blocco except ZeroDivisionError viene eseguito, stampando l'errore.
* Il blocco finally viene eseguito, stampando "Esecuzione completata."

dividi(10, 'a'):

* Il blocco try genera un TypeError perché non si può dividere un numero per una stringa.
* Il blocco except TypeError viene eseguito, stampando l'errore.
* Il blocco finally viene eseguito, stampando "Esecuzione completata."

Questo è un esempio completo in python per la gestione delle eccezioni egarantisce che il programma possa continuare a funzionare in modo da evitare di interrompere il codice gestendo gli errori.

Nel file github pythonbook7eccezioni.py le ultime due righe di codice sono commentate; se provate a de commentarle vedrete come si comporterebbe il codice senza la gestione delle eccezioni.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Fig.1 ZeroDivisionError

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Fig.2 TypeError

Esercizi di Pratica

Per consolidare la comprensione dei concetti, è utile provare alcuni esercizi pratici

Calcolatrice Semplice: Scrivi una funzione che accetti due numeri e un'operazione (somma, sottrazione, moltiplicazione, divisione) e restituisca il risultato.

Verifica Maggiorenne: Scrivi un programma che chieda l'età dell'utente e stampi se è maggiorenne o meno.

Stampa i Numeri Pari: Scrivi un ciclo che stampi tutti i numeri pari da 0 a 10.

Di seguito vi riporto i miei primi esempi di codice, ma vi consiglio di provare voi stesse a completare gli esercizi che vi ho proposto

Crea uno script che gestisce due tipi di errori comuni: la divisione per zero e l'input non numerico durante la conversione a intero.

Calcolatrice Semplice: Scrivi una funzione che accetti due numeri e un’operazione (somma, sottrazione, moltiplicazione, divisione) e restituisca il risultato.

Di seguito ti mostro il mio primo script python per realizzare una calcolatrice… era uno schifo e c’erano alcuni errori:

#semlice calcolatrice

num1  
num2  
  
def calc()  
 print (“scegli + somma, - sottrazione, x moltiplicazione, : sottrazione”)  
 if(+)  
 return (num1+num2)  
 else if(-)  
 return (num1-num2)  
 else if (x)  
 return (num1\*num2)  
 else  
 return   
 num1/num2

Questo codice presentava alcuni errori di sintassi e logici che ho corretto con un po’ di pazienza e di documentazione.

Di seguito, ti mostro una versione corretta e funzionante di una calcolatrice che gestisce le quattro operazioni di base: somma, sottrazione, moltiplicazione e divisione.

def calcolatrice():

num1 = float(input(“Inserisci il primo numero: “))

num2 = float(input(“Inserisci il secondo numero: “))

operazione = input(“Scegli l’operazione (+, -, \*, /): “)  
  
 if operazione == ‘+’:

return num1 + num2

elif operazione == ‘-‘:

return num1 – num2

elif operazione == ‘\*’:

return num1 \* num2

elif operazione == ‘/’:

if num2 != 0:

return num1 / num2

else:

return “Errore: divisione per zero”

else:

return “Operazione non valida”  
  
# Chiamata della funzione

risultato = calcolatrice()

print(“Risultato:”, risultato)

Correzioni e miglioramenti apportati:

Input degli utenti: Aggiunta la funzione input() per permettere agli utenti di inserire i numeri e scegliere l’operazione desiderata.

Conversione dei numeri: Utilizzo di float() per convertire l’input dell’utente in numeri decimali, permettendo operazioni con numeri interi e float.

Controllo dell’operazione: Utilizzo di if, elif, e else per gestire le diverse operazioni.

Gestione errori per la divisione: Aggiunto un controllo per evitare la divisione per zero, che causerebbe un errore nel programma.

Questo script ora funzionerà correttamente come una calcolatrice semplice, chiedendo all’utente i numeri e l’operazione e stampando il risultato finale.

Nota: se tu utilizzassi la shell di pycharme o ufficiale di python e scrivessi ad esempio 2+2 e successivamente premessi invio avresti direttamente l’output corretto, 4; questo perché la shell esegue direttamente le istruzioni.

Verifica Maggiorenne: Scrivi un programma che chieda l’età dell’utente e stampi se è maggiorenne o meno:

Quando scrissi per la prima volta un piccolo snippet condizionale era il seguente:

#maggiorene  
  
print (‘verifica maggiorenne\n’)  
  
eta=int(input(“Inserisci la tua : “))

if (eta>=18)

print(“si maggiorenne”)

else

(“sei minorenne”)

Anche in questo caso il presentava errori da correggere. E’ normale la prima volta commettere errori. In Python, è importante prestare attenzione all’indentazione1 e alla sintassi corretta delle istruzioni if e else. Ecco una versione corretta del tuo codice:

print(‘Verifica maggiorenne\n’)  
  
# L’input dell’età viene convertito direttamente in intero

eta = int(input(“Inserisci la tua età: “))  
  
# Controllo dell’età con una struttura if-else correttamente indentata

feta >= 18:

print(“Sei maggiorenne.”)

else:

print(“Sei minorenne.”)

Correzioni e chiarimenti:

Indentazione: Python usa l’indentazione per definire i blocchi di codice. Nel tuo script originale, l’indentazione non era corretta per le istruzioni if e else. Ho sistemato l’indentazione in modo che print per la condizione if e else sia correttamente allineato sotto il blocco condizionale.

Sintassi delle condizioni: La sintassi di if e else deve essere precisa. Dopo la condizione in if, deve esserci il carattere due punti (😊, che mancava nel codice originale.

Messaggi di stampa: Ho corretto il testo stampato per essere più chiaro e direttamente correlato alla condizione verificata (maggiorenne o minorenne).

Questo script ora funziona correttamente e può essere utilizzato per verificare se l’utente è maggiorenne o minorenne basandosi sull’età inserita.

Stampa i Numeri Pari: Scrivi un ciclo che stampi tutti i numeri pari da 0 a 10:

La prima volta scrissi un codice del genere:

numero=0  
  
for numero < 10

numero + 2

print(numero)

else print (“qiesti sono i numeri fino a 10 pari”)

Anche in questo caso feci degli errori. In Python, un ciclo for è generalmente utilizzato con un iterabile, mentre per i cicli basati su una condizione è più comune utilizzare while. Qui sotto trovi una versione corretta dello script che utilizza un ciclo while per stampare tutti i numeri pari fino a 10 con alcuni miglioramenti:

numero = 0  
  
while numero <= 10:

print(numero)

numero += 2  
  
print(“Questi sono i numeri pari fino a 10.”)

Correzioni e spiegazioni:

Uso del ciclo while: Per iterare basandosi su una condizione, è appropriato usare while. La condizione numero <= 10 garantisce che il ciclo continui fino a quando il numero non supera 10.

Incremento: Ho modificato numero + 2 in numero += 2, che è il modo corretto per incrementare il valore di numero di 2. La forma += è un operatore di assegnazione che aggiorna il valore di numero senza bisogno di riassegnarlo esplicitamente.

Stampa all’interno del ciclo: Ho spostato il print(numero) all’interno del ciclo while per stampare ogni numero pari man mano che viene calcolato.

Messaggio finale: Il messaggio “Questi sono i numeri pari fino a 10.” Viene stampato una sola volta alla fine del ciclo, per segnalare che il ciclo ha completato la sua esecuzione.

Questo script adesso funziona correttamente e stamperà i numeri pari da 0 a 10, seguito da un messaggio finale.

Volendo utilizzare il ciclo for per lo stesso esercizio, anche se ripeto, iterare usando una condizione è sempre più appropriato while, il codice sarebbe leggermente diverso e più diretto, ma utilizzando la funzione range(). La funzione range() può essere usata per generare una sequenza di numeri, e può essere configurata per iniziare da un certo numero, finire prima di un altro, e incrementare di un valore specifico.

Ecco un esempio di un ciclo for per stampare tutti i numeri pari fino a 10:

for numero in range(0, 11, 2):

print(numero)  
  
print(“Questi sono i numeri pari fino a 10.”)

Spiegazione:

range(0, 11, 2): Questa chiamata a range() inizia da 0, va fino a (ma non include) 11, e incrementa di 2 ad ogni passo. I valori che produce sono 0, 2, 4, 6, 8, e 10, che sono tutti i numeri pari tra 0 e 10 inclusi.

Ciclo for: Il ciclo itera su ogni elemento prodotto da range(), e in questo caso, ogni elemento è un numero pari che viene poi stampato.

Messaggio finale: Come nel caso del ciclo while, il messaggio finale viene stampato una volta che il ciclo for ha terminato, per indicare che tutti i numeri pari fino a 10 sono stati stampati.

Nota: se avessi voluto iterare semplicemente i numeri fino a 5, in molti altri linguaggi avrei potuto:

codice c:

#include <stdio.h>  
  
int main() {

for (int i = 0; i < 5; i++) {

printf(“%d\n”, i);

}

return 0;  
}

codice python:

for i in range(1, 6):

print(i)

# stampa

1

2

3

4

5

#se avessi scritto:

for i in range(5):

print(i)

#la stampa sarebbe stata

0

1

2

3

4

Vi starete chiedendo allora quando si usa while e quando for. Utilizza while per **iterazione basata su una condizione, utilizza for Iterazione su una sequenza e Range di numeri. Di seguito un semplice esempio dell’uso for.**

Stampiamo la lista di alcuni nomi:

# Lista di nomi

nomi = [“Amilcare”, “Giovanni”, “Cristina”, “Elena”, “Sophia”]#anticipazione del prossimo capitolo  
  
# Ciclo for per stampare ogni nome nella lista

for nome in nomi:

print(nome)

Soluzione esempio 4: Gestione delle Eccezioni in Python

Questo script gestisce due tipi di errori comuni: la divisione per zero e l'input non numerico durante la conversione a intero.

def dividi\_numeri():  
 try:  
 num1 = int(input("Inserisci il primo numero: "))  
 num2 = int(input("Inserisci il secondo numero: "))  
 risultato = num1 / num2  
 print(f"Il risultato della divisione è: {risultato}")  
 except ValueError:  
 print("Errore: Per favore inserisci solo numeri interi.")  
 except ZeroDivisionError:  
 print("Errore: La divisione per zero non è permessa.")  
 finally:  
 print("Operazione tentata di divisione.")  
  
# Chiamata alla funzione per testare la gestione delle eccezioni  
dividi\_numeri()

In tutto il capitolo abbiamo utilizzato **eccezioni predefinite** di python *ValueError, TypeError* ecc.

Per concludere il capitolo vi mostro un esemio che gestisce gli errori come nell’esercizio 2, ma creando un una *gestione personalizzata delle eccezioni*:

def dividi\_numeri():  
 try:  
 num1 = input("Inserisci il primo numero: ")  
 num2 = input("Inserisci il secondo numero: ")  
  
 # Controllo se l'input non è un numero intero  
 if not num1.isdigit() or not num2.isdigit():  
 raise Exception("Input non valido, puoi inserire solo numeri interi e non lettere o caratteri speciali.")  
  
 num1 = int(num1)  
 num2 = int(num2)  
  
 risultato = num1 / num2  
 print(f"Il risultato della divisione è: {risultato}")  
 except Exception as e:  
 print(f"Errore: {e}")  
 finally:  
 print("Operazione tentata di divisione.")  
  
  
# Chiamata alla funzione per testare la gestione delle eccezioni  
dividi\_numeri()

Spiegazione:

**if not num1.isdigit() or not num2.isdigit():**

* **num1.isdigit()**: Questo metodo verifica se la stringa num1 è composta solo da cifre. Restituisce True se tutti i caratteri nella stringa sono cifre e la stringa non è vuota, altrimenti restituisce False.
* **not num1.isdigit()**: Utilizzando not invertiamo il risultato del metodo isdigit(). Quindi, not num1.isdigit() sarà True se num1 non è composto solo da cifre.
* **or not num2.isdigit()**: Stessa logica si applica a num2. La condizione complessiva utilizza l'operatore logico or, il che significa che se almeno uno dei due numeri (num1 o num2) non è composto solo da cifre, la condizione sarà True.

**raise Exception("Input non valido, puoi inserire solo numeri interi.")**

* **raise**: L'istruzione raise viene utilizzata per sollevare un'eccezione. Quando Python incontra un'istruzione raise, interrompe l'esecuzione normale del programma e avvia il processo di gestione delle eccezioni.
* **Exception("Input non valido, puoi inserire solo numeri interi.")**: Viene sollevata una nuova eccezione di tipo Exception con il messaggio specificato. Questo messaggio verrà mostrato all'utente per indicare che l'input non è valido perché non è un numero intero.

Spiegazione:

Lista di Nomi: Innanzitutto, creiamo una lista chiamata nomi che contiene cinque nomi di persone.

Ciclo for: Usiamo un ciclo for per iterare su ogni elemento nella lista nomi. In ogni iterazione, la variabile nome assume il valore dell'elemento corrente della lista (i nomi uno dopo l'altro).

Stampa: Dentro il ciclo, usiamo la funzione print() per stampare il nome corrente.

Questo script stampa i nome della lista.

Conclusione

Questa sezione ti ha introdotto ai concetti fondamentali della programmazione in Python. Assicurati di comprendere bene questi concetti, poiché sono la base su cui costruirai le tue future conoscenze di programmazione. Fai più esperimenti possibile, è il modo migliore per imparare rapidamente. Nel prossimo modulo, esploreremo le strutture dati come liste, tuple, dizionari e set.

***Strutture Dati in Python***

Liste

Le liste sono collezioni ordinate e modificabili di elementi. Possono contenere elementi di diversi tipi, incluso il mix di tipi nella stessa lista.

Esempio di utilizzo delle liste:

# Creazione di una lista

frutti = ["mela", "banana", "ciliegia"]  
  
# Aggiunta di un elemento

frutti.append("arancia")  
  
# Accesso a un elemento

#Assegnazione di un valore ad una variabile  
primo\_frutto = frutti[0]

print(primo\_frutto) # Output: mela

#stessa cosa ma con

#Accesso diretto a un elemento della lista

print(frutti[1]) # stampa banana  
  
# Rimozione di un elemento

frutti.remove("banana")

**Codice passo passo:**

# Creazione di una lista

frutti = ["mela", "banana", "ciliegia"]

# Aggiungere un elemento.

frutti.append("arancia") #frutti diventa ["mela", "banana", "ciliegia", "arancia"]

# Accedere agli elementi. Assegnazione di un valore ad una variabile

primo\_frutto = frutti[0]

print(primo\_frutto) # Output: mela

#Accesso diretto a un elemento della lista

print(frutti[1]) # Output: banana

# Rimuovere un elemento

frutti.remove("mela")# frutti diventa ["banana", "ciliegia", "arancia"]

#Accesso diretto a un elemento della lista

print(frutti[1]) # Output: ciliegia

Gli array iniziano dall’indice 0.

0 mela, 1 banana,2 ciliegia

L’elemento dell’array 1 banana è stato rimosso, per cui ora il codice stamperà ciliegia

Con l’assegnazione come abbiamo fatto prima con *primo\_frutto,* il codice stamperà sempre la variabile assegnata, indipendentemente dall’eliminazione dell’elemento dall’array

### Spiegazione del codice

1. **Creazione di una lista**:

frutti = ["mela", "banana", "ciliegia"]

Questa linea di codice crea una lista chiamata frutti che contiene tre elementi: "mela", "banana", e "ciliegia". Le liste in Python sono collezioni ordinate di elementi che possono essere modificati (mutabili).

1. **Aggiungere un elemento alla lista**:

frutti.append("arancia")

La funzione append viene utilizzata per aggiungere un nuovo elemento alla fine della lista frutti. In questo caso, l'elemento "arancia" viene aggiunto alla lista. Dopo questa operazione, la lista frutti sarà ["mela", "banana", "ciliegia", "arancia"].

1. **Rimuovere un elemento dalla lista**:

frutti.remove("banana")

La funzione remove viene utilizzata per rimuovere il primo elemento della lista che corrisponde al valore specificato. In questo caso, l'elemento "banana" viene rimosso dalla lista frutti. Dopo questa operazione, la lista frutti sarà ["mela", "ciliegia", "arancia"].

1. **Accedere agli elementi della lista**:

primo\_frutto = frutti[0]

Questa linea di codice accede al primo elemento della lista frutti utilizzando l'indice 0. In Python, gli indici delle liste iniziano da 0. Quindi, frutti[0] restituisce il primo elemento della lista, che è "mela". Questo valore viene assegnato alla variabile primo\_frutto.

1. **Stampa del primo elemento della lista, indipendentemente dall’eliminazione di un elemento dall’array**:

print(primo\_frutto) # Output: mela

print (frutti[1]) # Output: ciliega

La funzione print stampa il valore assegnato alla variabile primo\_frutto nella console. Poiché primo\_frutto contiene "mela", l'output sarà "mela", indipendentemente dalla sua eliminazione o meno. Frutti[1] stampa ciliegia in quanto abbiamo rimosso mela dall’array e quindi sono cambiati gli indici.

### Output del codice

L'output sarà:

mela

ciliegia

### Riepilogo

* **Creazione di una lista**: La lista frutti viene creata con gli elementi ["mela", "banana", "ciliegia"].
* **Aggiunta di un elemento**: L'elemento "arancia" viene aggiunto alla fine della lista frutti.
* **Rimozione di un elemento**: L'elemento "banana" viene rimosso dalla lista frutti.
* **Accesso al primo elemento**: Il primo elemento della lista frutti viene assegnato alla variabile primo\_frutto.
* **Stampa del primo elemento**: La funzione print stampa il valore di primo\_frutto, che è "mela".

In questo esempio, abbiamo visto come creare una lista, aggiungere e rimuovere elementi dalla lista, accedere agli elementi della lista e stampare un elemento della lista utilizzando Python.

**Tuple** Le tuple sono simili alle liste, ma sono immutabili, il che significa che non possono essere modificate dopo la loro creazione.

# Creazione di una tupla

numeri = (1, 2, 3, 4, 5)

# Accedere agli elementi

secondo\_numero = numeri[1]

print(secondo\_numero) # Output: 2

# Le tuple non possono essere modificate

# numeri[0] = 10 # Questo causerà un errore

### Spiegazione del codice

1. **Creazione di una tupla**:

numeri = (1, 2, 3, 4, 5)

Questa linea di codice crea una tupla chiamata numeri che contiene cinque elementi: 1, 2, 3, 4, e 5. Le tuple in Python sono collezioni ordinate di elementi che sono immutabili (cioè, non possono essere modificate dopo la loro creazione).

1. **Accedere agli elementi della tupla**:

secondo\_numero = numeri[1]

Questa linea di codice accede al secondo elemento della tupla numeri utilizzando l'indice 1. In Python, gli indici delle tuple (e delle liste) iniziano da 0. Quindi, numeri[1] restituisce il secondo elemento della tupla, che è 2. Questo valore viene assegnato alla variabile secondo\_numero.

1. **Stampa del secondo elemento della tupla**:

print(secondo\_numero) # Output: 2

La funzione print stampa il valore della variabile secondo\_numero sulla console. Poiché secondo\_numero contiene 2, l'output sarà 2.

1. **Immutabilità delle tuple**:

# Le tuple non possono essere modificate

# numeri[0] = 10 # Questo causerà un errore

Le tuple in Python sono immutabili, il che significa che non possono essere modificate dopo la loro creazione. Il commento # numeri[0] = 10 è un esempio di un'operazione che causerà un errore. Se si tenta di eseguire questa linea di codice, Python genererà un TypeError perché le tuple non supportano l'assegnazione di elementi.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Creazione della tupla numeri**: La tupla numeri viene creata con i valori (1, 2, 3, 4, 5).
2. **Accesso al secondo elemento della tupla**: Il secondo elemento della tupla numeri, che è 2, viene assegnato alla variabile secondo\_numero.
3. **Stampa del secondo elemento della tupla**: La funzione print stampa il valore di secondo\_numero, che è 2.
4. **Immutabilità delle tuple**: Il commento mostra che le tuple non possono essere modificate e un tentativo di farlo causerà un errore.

### Output del codice

L'output sarà:

2

### Riepilogo

* **Creazione di una tupla**: La tupla numeri viene creata con gli elementi (1, 2, 3, 4, 5).
* **Accesso al secondo elemento**: Il secondo elemento della tupla numeri viene assegnato alla variabile secondo\_numero.
* **Stampa del secondo elemento**: La funzione print stampa il valore di secondo\_numero, che è 2.
* **Immutabilità delle tuple**: Le tuple in Python sono immutabili, quindi non possono essere modificate dopo la loro creazione. Un tentativo di modificare una tupla causerà un TypeError.

In questo esempio, abbiamo visto come creare una tupla, accedere agli elementi della tupla e comprendere l'immutabilità delle tuple in Python.

**Dizionari** I dizionari sono collezioni non ordinate di coppie chiave-valore. Ogni chiave deve essere unica.

# Creazione di un dizionario

studente = {

"nome": "Alice",

"età": 25,

"città": "Roma"

}

# Aggiungere una nuova coppia chiave-valore

studente["università"] = "La Sapienza"

# Rimuovere una coppia chiave-valore

del studente["età"]

# Accedere ai valori

nome\_studente = studente["nome"]

print(nome\_studente) # Output: Alice

### Spiegazione del codice

1. **Creazione di un dizionario**:

studente = {

"nome": "Alice",

"età": 25,

"città": "Roma"

}

Questa linea di codice crea un dizionario chiamato studente che contiene tre coppie chiave-valore: "nome": "Alice", "età": 25, e "città": "Roma". Un dizionario in Python è una collezione non ordinata di coppie chiave-valore, dove ogni chiave deve essere unica.

1. **Aggiungere una nuova coppia chiave-valore**:

studente["università"] = "La Sapienza"

Questa linea di codice aggiunge una nuova coppia chiave-valore al dizionario studente. La nuova chiave è "università" e il valore associato è "La Sapienza". Dopo questa operazione, il dizionario studente sarà:

{

"nome": "Alice",

"età": 25,

"città": "Roma",

"università": "La Sapienza"

}

1. **Rimuovere una coppia chiave-valore**:

del studente["età"]

Questa linea di codice rimuove la coppia chiave-valore con chiave "età" dal dizionario studente. Dopo questa operazione, il dizionario studente sarà:

{

"nome": "Alice",

"città": "Roma",

"università": "La Sapienza"

}

1. **Accedere ai valori del dizionario**:

nome\_studente = studente["nome"]

Questa linea di codice accede al valore associato alla chiave "nome" nel dizionario studente. Il valore è "Alice", che viene assegnato alla variabile nome\_studente.

1. **Stampa del valore**:

print(nome\_studente) # Output: Alice

La funzione print stampa il valore della variabile nome\_studente sulla console. Poiché nome\_studente contiene "Alice", l'output sarà Alice.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Creazione del dizionario studente**: Il dizionario studente viene creato con i valori {"nome": "Alice", "età": 25, "città": "Roma"}.
2. **Aggiunta di una nuova coppia chiave-valore**: La coppia "università": "La Sapienza" viene aggiunta al dizionario studente, che diventa:

{

"nome": "Alice",

"età": 25,

"città": "Roma",

"università": "La Sapienza"

}

1. **Rimozione di una coppia chiave-valore**: La coppia con chiave "età" viene rimossa dal dizionario studente, che diventa:

{

"nome": "Alice",

"città": "Roma",

"università": "La Sapienza"

}

1. **Accesso al valore associato alla chiave "nome"**: Il valore associato alla chiave "nome" nel dizionario studente viene assegnato alla variabile nome\_studente.
2. **Stampa del valore**: La funzione print stampa il valore di nome\_studente, che è "Alice".

### Output del codice

L'output sarà:

Alice

### Riepilogo

* **Creazione di un dizionario**: Il dizionario studente viene creato con gli elementi {"nome": "Alice", "età": 25, "città": "Roma"}.
* **Aggiunta di una nuova coppia chiave-valore**: La coppia "università": "La Sapienza" viene aggiunta al dizionario studente.
* **Rimozione di una coppia chiave-valore**: La coppia con chiave "età" viene rimossa dal dizionario studente.
* **Accesso al valore**: Il valore associato alla chiave "nome" nel dizionario studente viene assegnato alla variabile nome\_studente.
* **Stampa del valore**: La funzione print stampa il valore di nome\_studente, che è "Alice".

In questo esempio, abbiamo visto come creare un dizionario, aggiungere e rimuovere coppie chiave-valore dal dizionario, accedere ai valori del dizionario e stampare un valore del dizionario utilizzando Python.

**Set** I set sono collezioni non ordinate di elementi unici. Sono utili per eliminare i duplicati e per le operazioni di insiemi matematici.

# Creazione di un set

numeri = {1, 2, 3, 4, 5, 5, 6}

# Aggiungere un elemento

numeri.add(7)

# Rimuovere un elemento

numeri.remove(4)

# Verificare se un elemento è nel set

print(3 in numeri) # Output: True

### Spiegazione del codice

1. **Creazione di un set**:

numeri = {1, 2, 3, 4, 5, 5, 6}

Questa linea di codice crea un set chiamato numeri che contiene i valori {1, 2, 3, 4, 5, 5, 6}. Un set in Python è una collezione non ordinata di elementi unici. Notare che, anche se il valore 5 appare due volte nella definizione del set, nel set risultante ci sarà una sola occorrenza di 5. Quindi, il set numeri sarà {1, 2, 3, 4, 5, 6}.

1. **Aggiungere un elemento al set**:

numeri.add(7)

La funzione add viene utilizzata per aggiungere un nuovo elemento al set numeri. In questo caso, l'elemento 7 viene aggiunto al set. Dopo questa operazione, il set numeri sarà {1, 2, 3, 4, 5, 6, 7}.

1. **Rimuovere un elemento dal set**:

numeri.remove(4)

La funzione remove viene utilizzata per rimuovere un elemento specifico dal set numeri. In questo caso, l'elemento 4 viene rimosso dal set. Dopo questa operazione, il set numeri sarà {1, 2, 3, 5, 6, 7}.

1. **Verificare se un elemento è nel set**:

print(3 in numeri) # Output: True

L'operatore in viene utilizzato per verificare se un elemento è presente nel set. In questo caso, stiamo verificando se il valore 3 è nel set numeri. Poiché 3 è effettivamente presente nel set, l'operazione 3 in numeri restituirà True. La funzione print stampa questo valore sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Creazione del set numeri**: Il set numeri viene creato con i valori {1, 2, 3, 4, 5, 5, 6}. Poiché i set non permettono duplicati, il set risultante sarà {1, 2, 3, 4, 5, 6}.
2. **Aggiunta di un elemento al set**: L'elemento 7 viene aggiunto al set numeri, che diventa {1, 2, 3, 4, 5, 6, 7}.
3. **Rimozione di un elemento dal set**: L'elemento 4 viene rimosso dal set numeri, che diventa {1, 2, 3, 5, 6, 7}.
4. **Verifica della presenza di un elemento nel set**: L'operazione 3 in numeri verifica se 3 è presente nel set numeri. Poiché 3 è presente, l'operazione restituisce True.

### Output del codice

L'output sarà:

graphql

True

### Riepilogo

* **Creazione di un set**: Il set numeri viene creato con gli elementi {1, 2, 3, 4, 5, 5, 6}, e i duplicati vengono rimossi automaticamente, risultando nel set {1, 2, 3, 4, 5, 6}.
* **Aggiunta di un elemento**: L'elemento 7 viene aggiunto al set numeri, che diventa {1, 2, 3, 4, 5, 6, 7}.
* **Rimozione di un elemento**: L'elemento 4 viene rimosso dal set numeri, che diventa {1, 2, 3, 5, 6, 7}.
* **Verifica della presenza di un elemento**: L'operazione 3 in numeri verifica se 3 è presente nel set numeri e restituisce True.

In questo esempio, abbiamo visto come creare un set, aggiungere e rimuovere elementi dal set, e verificare la presenza di un elemento nel set utilizzando Python.

Esercizi di Pratica

- Creare una lista di numeri e calcola la loro somma.

- Utilizzare una tupla per immagazzinare dati che non dovrebbero cambiare, come i giorni della settimana.

- Creare un dizionario per un piccolo database di contatti, dove le chiavi sono i nomi delle persone e i valori sono i loro numeri di telefono.

- Utilizzare un set per eliminare duplicati da una lista di elementi.

Prova a svolgere questi esercizi. Sbaglia più che puoi, consulta la documentazione e migliora il tuo codice.

Era il mio secondo giorno di python e stavo migliorando poco a poco. Qui di seguito vi riporto lo script del primo esercizio

#stavo migliorando  
  
#lista numero  
  
numeri=[1,3,4,8,10]  
  
print(numeri[1]+numeri[2]+numeri[3]+numeri[4]+numeri[0])

Lo script Python per calcolare la somma dei numeri in una lista è corretto e funziona come previsto. Tuttavia, se l’obiettivo è sommare tutti gli elementi di una lista, esiste un modo più efficiente e meno soggetto ad errori usando la funzione built-in sum(). Questo metodo è più breve e più facilmente adattabile, specialmente se la dimensione della lista cambia. Ecco come dovrebbe apparire lo script migliorato:

# script migliorato

# Lista di numeri

numeri = [1, 3, 4, 8, 10]  
  
# Calcolo della somma degli elementi della lista

somma = sum(numeri)  
  
# Stampa del risultato

print(somma)

Spiegazione:

sum(numeri): Questa funzione prende una lista di numeri e restituisce la loro somma totale.

Stampa del risultato: Il risultato della somma viene poi stampato.

Questo approccio non solo riduce il codice necessario, ma è anche più robusto contro modifiche future della lista (come aggiungere o rimuovere elementi).

Utilizzare un set per eliminare duplicati da una lista di elementi:

Per eliminare i duplicati da una lista in Python, puoi sfruttare le proprietà dei set, che non permettono elementi duplicati. Convertendo una lista in un set, qualsiasi elemento duplicato verrà automaticamente rimosso. Dopo aver convertito la lista in un set, se necessario, puoi convertirla di nuovo in una lista. Ecco il codice di seguito:

#stavo diventando bravo :)  
  
# Lista originale con duplicati

lista\_con\_duplicati = [1, 2, 2, 3, 4, 4, 4, 5, 6, 6, 6, 7]  
  
# Conversione della lista in un set per rimuovere i duplicati

set\_senza\_duplicati = set(lista\_con\_duplicati)  
  
# (Opzionale) Conversione del set di nuovo in una lista

lista\_senza\_duplicati = list(set\_senza\_duplicati)

# Stampa della lista senza duplicati

print(lista\_senza\_duplicati)

Spiegazione:

Creazione della lista: Definisci una lista che include alcuni numeri duplicati.

Conversione in set: Converti la lista in un set, set\_senza\_duplicati. In questa fase, tutti i duplicati vengono automaticamente rimossi perché i set non possono contenere elementi duplicati.

Riconversione in lista: Se hai bisogno di lavorare ulteriormente con una lista (dato che i set non mantengono un ordine specifico e non supportano l'indicizzazione), puoi convertire il set di nuovo in una lista.

Stampa del risultato: Mostra la lista risultante, che ora è priva di duplicati.

Questo metodo è molto efficiente e viene spesso utilizzato per pulire i dati dalle ripetizioni in modo semplice e rapido.

Ulteriore esempio.

Script Python per Rimuovere un Elemento Specifico da un Set:

# Creazione del set con un possibile errore di duplicazione

animali = {"gatto", "cane", "uccello", "gatto"} # "gatto" duplicato non ha effetto  
  
# Stampa del set originale per vedere l'effetto dei duplicati

print("Set originale:", animali)  
  
# Rimozione di "gatto" dal set

animali.remove("gatto")  
  
# Stampa del set dopo la rimozione

print("Set dopo la rimozione di 'gatto':", animali)

Spiegazioni:

Definizione del Set: I set sono definiti tra parentesi graffe {}. Gli elementi duplicati, come "gatto" nel nostro esempio, vengono automaticamente rimossi quando il set viene creato.

Stampa del Set Originale: Prima di modificare il set, stampiamo il suo contenuto iniziale per mostrare che non contiene duplicati nonostante "gatto" sia stato menzionato due volte.

Rimozione di un Elemento: Usando il metodo remove(), puoi eliminare un elemento specifico dal set. Se l'elemento non è presente nel set, remove() lancerà un KeyError. Se vuoi evitare l'errore, puoi usare il metodo discard() che non genera errori se l'elemento non è presente.

Stampa dopo la Rimozione: Dopo la rimozione, stampiamo il set per mostrare il risultato finale.

Questo script dimostra come gestire i set in Python, mostrando che i duplicati vengono gestiti automaticamente e come rimuovere in modo sicuro gli elementi.

Non vi darò suggerimenti; provate a svolgere gli esercizi rimanenti.

In github, comunque, troverete anche le soluzioni di questi ultimi due esercizi.

Conclusione

Questi concetti di strutture dati sono essenziali per scrivere programmi efficienti e organizzati in Python. Nel prossimo modulo, esploreremo concetti più avanzati come la programmazione orientata agli oggetti e l'uso di moduli e pacchetti.

#### **Moduli e Pacchetti**

I moduli sono file contenenti definizioni e istruzioni Python. I moduli sono usati per suddividere il codice in file gestibili e riutilizzabili. Python ha molti moduli preinstallati che contengono funzioni utili.

<https://pypi.org/>

**Importare moduli** Un modulo è un file contenente definizioni e istruzioni Python. Può essere importato in altri script.

import math

print(math.sqrt(16)) # Output: 4.0

### Spiegazione del codice

1. **Importazione del modulo math**:

**link**: https://docs.python.org/3/library/math.html

import math

Questa linea di codice importa il modulo math, che è una libreria standard di Python che fornisce funzioni matematiche, tra cui funzioni per il calcolo di radici quadrate, esponenziali, logaritmi, trigonometria e altro ancora.

1. **Utilizzo della funzione sqrt**:

print(math.sqrt(16)) # Output: 4.0

Qui utilizziamo la funzione sqrt (abbreviazione di "square root") del modulo math per calcolare la radice quadrata di 16. La funzione math.sqrt prende un singolo argomento e restituisce la radice quadrata di quel numero.

* + **math.sqrt(16)**: Questa chiamata di funzione calcola la radice quadrata di 16, che è 4.0.
  + **print**: La funzione print stampa l'output della funzione math.sqrt(16) sulla console. Poiché math.sqrt(16) restituisce 4.0, il valore stampato sarà 4.0.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo math viene importato, rendendo disponibili tutte le funzioni matematiche definite in questo modulo.
2. La funzione math.sqrt(16) viene chiamata per calcolare la radice quadrata di 16.
3. Il risultato della funzione math.sqrt(16), che è 4.0, viene passato alla funzione print.
4. La funzione print stampa 4.0 sulla console.

### Output del codice

Quando il codice viene eseguito, l'output sarà:

4.0

### Riepilogo

* **Importazione del modulo**: Il modulo math viene importato utilizzando import math.
* **Utilizzo della funzione sqrt**: La funzione math.sqrt(16) calcola la radice quadrata di 16, che è 4.0.
* **Stampa del risultato**: La funzione print stampa 4.0 sulla console.

Il modulo math è molto utile per eseguire operazioni matematiche complesse in Python. La funzione sqrt è solo una delle tante funzioni disponibili in questo modulo.

**Creare e usare moduli** Puoi creare i tuoi moduli salvando le funzioni in un file .py e importandolo in altri script.

# saluti.py

def saluta():

print("Ciao!")

# main.py

import saluti

saluti.saluta() # Output: Ciao!

### Spiegazione del codice

#### File: saluti.py

1. **Definizione della funzione saluta**:

def saluta():

print("Ciao!")

Questa linea di codice definisce una funzione chiamata saluta nel file saluti.py. La parola chiave def viene utilizzata per dichiarare una funzione in Python. Il nome della funzione è saluta, e le parentesi () indicano che non ci sono parametri in ingresso per questa funzione. Il carattere : indica l'inizio del blocco di codice che costituisce il corpo della funzione.

Il corpo della funzione contiene una singola istruzione print che stampa la stringa "Ciao!" sulla console quando la funzione viene chiamata.

#### File: main.py

1. **Importazione del modulo saluti**:

import saluti

Questa linea di codice importa il modulo saluti. Quando importiamo un modulo in Python, il codice nel modulo importato viene eseguito e le funzioni e le variabili definite nel modulo diventano accessibili nel file che lo importa.

1. **Chiamata della funzione saluta dal modulo saluti**:

saluti.saluta() # Output: Ciao!

In questa linea, stiamo chiamando la funzione saluta definita nel modulo saluti. Utilizziamo la notazione punto (.) per accedere alla funzione saluta all'interno del modulo saluti. Quando saluti.saluta() viene eseguito, la funzione saluta stampa "Ciao!" sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Esecuzione di saluti.py**:
   * Il file saluti.py viene eseguito una volta quando il modulo viene importato in main.py.
   * La funzione saluta viene definita ma non viene eseguita a questo punto.
2. **Esecuzione di main.py**:
   * Il modulo saluti viene importato, rendendo la funzione saluta disponibile in main.py.
   * La funzione saluta viene chiamata utilizzando saluti.saluta().
   * La funzione saluta stampa "Ciao!" sulla console.

### Output del codice

Quando il codice viene eseguito, l'output sarà:

Ciao!

### Riepilogo

* **Definizione della funzione**: La funzione saluta è definita nel file saluti.py. Quando viene chiamata, stampa "Ciao!" sulla console.
* **Importazione del modulo**: Il modulo saluti viene importato in main.py utilizzando import saluti.
* **Chiamata della funzione**: La funzione saluta viene chiamata nel file main.py utilizzando saluti.saluta(), che stampa "Ciao!" sulla console.

Questo esempio mostra come definire una funzione in un modulo separato e come importarla e utilizzarla in un altro script Python.

**Panoramica sui principali pacchetti standard** Python include una libreria standard molto vasta, che comprende numerosi moduli utili:

* **os**: interfaccia con le funzionalità del sistema operativo.
* **sys**: parametri e funzioni specifiche del sistema.
* **math**: funzioni matematiche.
* **datetime**: manipolazione di date e ore.
* **json**: parsing e generazione di JSON.

Questa sezione copre i fondamenti essenziali di Python, permettendo ai lettori di acquisire una solida base su cui costruire competenze più avanzate.

### Modulo os

Il modulo os fornisce un modo per interagire con il sistema operativo. Offre numerose funzioni per la gestione dei file e delle directory.

**Esempi:**

* **Ottenere il percorso della directory corrente**

import os

current\_directory = os.getcwd()

print("Directory corrente:", current\_directory)

### Spiegazione del codice

1. **Importazione del modulo os**:

import os

Questa linea di codice importa il modulo os, che è una libreria standard di Python che fornisce una varietà di funzioni per interagire con il sistema operativo. Il modulo os permette di eseguire operazioni come la gestione di file e directory, l'ottenimento di informazioni sull'ambiente del sistema e altro ancora.

1. **Ottenimento della directory corrente**:

current\_directory = os.getcwd()

La funzione os.getcwd() restituisce il percorso assoluto della directory corrente di lavoro (current working directory). Questo significa che il valore restituito è la directory in cui il programma Python viene eseguito. Il percorso restituito viene assegnato alla variabile current\_directory.

1. **Stampa della directory corrente**:

print("Directory corrente:", current\_directory)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Directory corrente:" seguito dal valore della variabile current\_directory. Questo output mostrerà all'utente il percorso della directory corrente.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo os viene importato, rendendo disponibili tutte le funzioni definite in questo modulo.
2. La funzione os.getcwd() viene chiamata per ottenere il percorso della directory corrente di lavoro.
3. Il percorso restituito da os.getcwd() viene assegnato alla variabile current\_directory.
4. La funzione print stampa il messaggio "Directory corrente:" seguito dal valore di current\_directory sulla console.

### Output del codice

Quando il codice viene eseguito, l'output sarà simile a questo (a seconda della directory in cui viene eseguito il programma):

javascript

Directory corrente: /percorso/alla/directory/corrente

### Riepilogo

* **Importazione del modulo os**: Il modulo os viene importato utilizzando import os.
* **Ottenimento della directory corrente**: La funzione os.getcwd() viene utilizzata per ottenere il percorso della directory corrente di lavoro. Questo percorso viene assegnato alla variabile current\_directory.
* **Stampa della directory corrente**: La funzione print stampa il messaggio "Directory corrente:" seguito dal percorso della directory corrente.

Il modulo os è molto utile per eseguire operazioni che richiedono l'interazione con il sistema operativo, come la gestione di file e directory. In questo esempio, abbiamo visto come ottenere e stampare la directory corrente di lavoro.

Inizio modulo

Fine modulo

* **Creare una nuova directory**

import os

os.mkdir('nuova\_cartella')

### Spiegazione del codice

1. **Importazione del modulo os**:

import os

Questa linea di codice importa il modulo os, che è una libreria standard di Python che fornisce una varietà di funzioni per interagire con il sistema operativo. Il modulo os permette di eseguire operazioni come la gestione di file e directory, l'ottenimento di informazioni sull'ambiente del sistema e altro ancora.

1. **Creazione di una nuova directory**:

os.mkdir('nuova\_cartella')

La funzione os.mkdir() crea una nuova directory con il nome specificato. In questo caso, il nome della nuova directory è 'nuova\_cartella'. La funzione prende un argomento, che è il percorso della nuova directory da creare. Se il percorso non è specificato come assoluto, verrà creato come percorso relativo alla directory corrente di lavoro.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo os viene importato, rendendo disponibili tutte le funzioni definite in questo modulo.
2. La funzione os.mkdir('nuova\_cartella') viene chiamata per creare una nuova directory chiamata 'nuova\_cartella' nella directory corrente di lavoro.

### Considerazioni

* Se una directory con il nome 'nuova\_cartella' esiste già, la chiamata a os.mkdir('nuova\_cartella') genererà un'eccezione FileExistsError.
* Assicurati di avere i permessi di scrittura nella directory in cui stai cercando di creare la nuova directory.

### Esempio di utilizzo

Supponiamo di avere la seguente struttura di directory:

bash

/progetto

main.py

Se eseguiamo il codice os.mkdir('nuova\_cartella') all'interno di main.py, dopo l'esecuzione la struttura sarà:

bash

/progetto

main.py

nuova\_cartella/

### Riepilogo

* **Importazione del modulo os**: Il modulo os viene importato utilizzando import os.
* **Creazione di una nuova directory**: La funzione os.mkdir('nuova\_cartella') viene utilizzata per creare una nuova directory chiamata 'nuova\_cartella' nella directory corrente di lavoro.

Il modulo os è molto utile per eseguire operazioni che richiedono l'interazione con il sistema operativo, come la gestione di file e directory. In questo esempio, abbiamo visto come creare una nuova directory utilizzando os.mkdir().

* **Elencare i file in una directory**

import os

files = os.listdir('.')

print("File nella directory corrente:", files)

### Spiegazione del codice

1. **Importazione del modulo os**:

import os

Questa linea di codice importa il modulo os, che è una libreria standard di Python che fornisce una varietà di funzioni per interagire con il sistema operativo. Il modulo os permette di eseguire operazioni come la gestione di file e directory, l'ottenimento di informazioni sull'ambiente del sistema e altro ancora.

1. **Elencare i file nella directory corrente**:

files = os.listdir('.')

La funzione os.listdir() restituisce un elenco contenente i nomi delle voci nella directory specificata dal percorso. Se il percorso specificato è '.' (un singolo punto), la funzione elencherà i file e le directory presenti nella directory corrente di lavoro. L'elenco restituito viene assegnato alla variabile files.

1. **Stampa dei file nella directory corrente**:

print("File nella directory corrente:", files)

Questa linea di codice utilizza la funzione print per stampare il messaggio "File nella directory corrente:" seguito dall'elenco dei file e delle directory presenti nella directory corrente di lavoro, contenuto nella variabile files.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo os viene importato, rendendo disponibili tutte le funzioni definite in questo modulo.
2. La funzione os.listdir('.') viene chiamata per ottenere un elenco dei file e delle directory nella directory corrente di lavoro.
3. L'elenco restituito da os.listdir('.') viene assegnato alla variabile files.
4. La funzione print stampa il messaggio "File nella directory corrente:" seguito dall'elenco dei file e delle directory sulla console.

### Output del codice

L'output dipende dal contenuto della directory corrente di lavoro. Ad esempio, se nella directory corrente ci sono tre file (file1.txt, file2.txt, file3.txt) e una directory (nuova\_cartella), l'output sarà:

arduino

File nella directory corrente: ['file1.txt', 'file2.txt', 'file3.txt', 'nuova\_cartella']

### Riepilogo

* **Importazione del modulo os**: Il modulo os viene importato utilizzando import os.
* **Elencare i file nella directory corrente**: La funzione os.listdir('.') viene utilizzata per ottenere un elenco dei file e delle directory presenti nella directory corrente di lavoro. Questo elenco viene assegnato alla variabile files.
* **Stampa dei file nella directory corrente**: La funzione print stampa il messaggio "File nella directory corrente:" seguito dall'elenco dei file e delle directory presenti nella directory corrente di lavoro.

Il modulo os è molto utile per eseguire operazioni che richiedono l'interazione con il sistema operativo, come la gestione di file e directory. In questo esempio, abbiamo visto come elencare i file e le directory presenti nella directory corrente di lavoro utilizzando os.listdir().

Inizio modulo

Fine modulo

### Modulo sys

Il modulo sys fornisce l'accesso a variabili e funzioni che interagiscono strettamente con l'interprete Python.

**Esempi:**

* **Ottenere la versione di Python**

import sys

print("Versione di Python:", sys.version)

### Spiegazione del codice

1. **Importazione del modulo sys**:

import sys

Questa linea di codice importa il modulo sys, che è una libreria standard di Python che fornisce accesso a variabili e funzioni che interagiscono strettamente con l'interprete Python. Il modulo sys permette di ottenere informazioni sull'ambiente di esecuzione di Python, tra cui la versione dell'interprete, i percorsi di ricerca dei moduli, gli argomenti della riga di comando, e altro ancora.

1. **Stampa della versione di Python**:

print("Versione di Python:", sys.version)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Versione di Python:" seguito dal valore di sys.version. La variabile sys.version contiene una stringa che descrive la versione dell'interprete Python attualmente in esecuzione, includendo il numero di versione e altre informazioni aggiuntive come il tipo di build e la data di compilazione.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo sys viene importato, rendendo disponibili tutte le funzioni e le variabili definite in questo modulo.
2. La funzione print viene chiamata con due argomenti:
   * Il primo argomento è la stringa "Versione di Python:".
   * Il secondo argomento è sys.version, che contiene la stringa descrittiva della versione dell'interprete Python.
3. La funzione print stampa questi argomenti sulla console.

### Output del codice

L'output dipende dalla versione dell'interprete Python in uso. Ad esempio, se stai utilizzando Python 3.8.10, l'output sarà simile a questo:

csharp

Versione di Python: 3.8.10 (default, May 3 2021, 08:55:58)

[GCC 7.5.0]

### Riepilogo

* **Importazione del modulo sys**: Il modulo sys viene importato utilizzando import sys.
* **Stampa della versione di Python**: La funzione print viene utilizzata per stampare il messaggio "Versione di Python:" seguito dal valore di sys.version, che contiene informazioni dettagliate sulla versione dell'interprete Python attualmente in esecuzione.

Il modulo sys è molto utile per ottenere informazioni sull'ambiente di esecuzione di Python e per interagire con l'interprete Python in vari modi. In questo esempio, abbiamo visto come ottenere e stampare la versione dell'interprete Python utilizzando sys.version.

* **Terminare lo script**

print("Script in esecuzione")

sys.exit()

print("Questo non sarà stampato")

### Spiegazione del codice

1. **Stampa del messaggio "Script in esecuzione"**:

print("Script in esecuzione")

Questa linea di codice utilizza la funzione print per stampare il messaggio "Script in esecuzione" sulla console. Quando questa riga viene eseguita, il messaggio viene visualizzato sullo schermo.

1. **Terminazione dello script con sys.exit()**:

sys.exit()

Questa linea di codice utilizza la funzione sys.exit() del modulo sys per terminare l'esecuzione dello script. Quando sys.exit() viene chiamata, l'interprete Python interrompe immediatamente l'esecuzione dello script e chiude il programma. Qualsiasi codice che segue la chiamata a sys.exit() non verrà eseguito.

1. **Stampa del messaggio "Questo non sarà stampato"**:

print("Questo non sarà stampato")

Questa linea di codice tenta di stampare il messaggio "Questo non sarà stampato" sulla console. Tuttavia, poiché sys.exit() è stato chiamato prima di questa riga, l'esecuzione dello script viene terminata e questa linea di codice non viene mai eseguita.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. La funzione print("Script in esecuzione") viene eseguita, stampando il messaggio "Script in esecuzione" sulla console.
2. La funzione sys.exit() viene chiamata, causando la terminazione immediata dello script. L'interprete Python chiude il programma.
3. Poiché lo script è terminato, la linea print("Questo non sarà stampato") non viene mai eseguita.

### Output del codice

L'output sarà:

Script in esecuzione

Il messaggio "Questo non sarà stampato" non viene visualizzato perché l'esecuzione dello script è stata terminata prima che questa linea di codice potesse essere eseguita.

### Riepilogo

* **Stampa del messaggio**: La funzione print("Script in esecuzione") stampa il messaggio "Script in esecuzione" sulla console.
* **Terminazione dello script**: La funzione sys.exit() termina immediatamente l'esecuzione dello script. Qualsiasi codice che segue questa chiamata non viene eseguito.
* **Codice non eseguito**: La linea print("Questo non sarà stampato") non viene eseguita perché lo script è stato terminato prima di raggiungere questa linea.

L'utilizzo di sys.exit() è utile quando si desidera terminare uno script Python prematuramente, ad esempio in risposta a un errore critico o una condizione che richiede l'arresto del programma.

Inizio modulo

Fine modulo

* **Ottenere gli argomenti della riga di comando**

import sys

print("Argomenti della riga di comando:", sys.argv)

### Spiegazione del codice

1. **Importazione del modulo sys**:

import sys

Questa linea di codice importa il modulo sys, che è una libreria standard di Python che fornisce accesso a variabili e funzioni che interagiscono strettamente con l'interprete Python. Il modulo sys permette di ottenere informazioni sull'ambiente di esecuzione di Python, tra cui gli argomenti passati alla riga di comando.

1. **Stampa degli argomenti della riga di comando**:

print("Argomenti della riga di comando:", sys.argv)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Argomenti della riga di comando:" seguito dal valore di sys.argv. La variabile sys.argv è una lista che contiene gli argomenti passati alla riga di comando quando lo script Python è stato eseguito.

### Dettagli su sys.argv

* **sys.argv**:
  + sys.argv è una lista in cui il primo elemento (sys.argv[0]) è il nome dello script Python stesso.
  + Gli elementi successivi di sys.argv sono gli argomenti aggiuntivi forniti sulla riga di comando.

### Esempio di esecuzione

Supponiamo di avere uno script chiamato example.py contenente il seguente codice:

import sys

print("Argomenti della riga di comando:", sys.argv)

Se eseguiamo questo script dalla riga di comando con alcuni argomenti, ad esempio:

sh

python example.py arg1 arg2 arg3

### Output del codice

L'output sarà:

less

Argomenti della riga di comando: ['example.py', 'arg1', 'arg2', 'arg3']

* sys.argv[0] è 'example.py', il nome dello script.
* sys.argv[1] è 'arg1'.
* sys.argv[2] è 'arg2'.
* sys.argv[3] è 'arg3'.

### Riepilogo

* **Importazione del modulo sys**: Il modulo sys viene importato utilizzando import sys.
* **Stampa degli argomenti della riga di comando**: La funzione print viene utilizzata per stampare il messaggio "Argomenti della riga di comando:" seguito dal valore di sys.argv, che è una lista contenente il nome dello script e gli argomenti della riga di comando.

L'utilizzo di sys.argv è utile quando si desidera accedere agli argomenti della riga di comando all'interno di uno script Python, permettendo di personalizzare il comportamento dello script in base agli input forniti dall'utente.

### Modulo math

Il modulo math fornisce funzioni matematiche definite dallo standard C.

**Esempi:**

* **Calcolare la radice quadrata**

import math

print("Radice quadrata di 16:", math.sqrt(16))

### Spiegazione del codice

1. **Importazione del modulo math**:

import math

Questa linea di codice importa il modulo math, che è una libreria standard di Python che fornisce una varietà di funzioni matematiche. Il modulo math include funzioni per eseguire operazioni matematiche comuni come il calcolo di radici quadrate, esponenziali, logaritmi, funzioni trigonometriche e altro ancora.

1. **Calcolo della radice quadrata**:

print("Radice quadrata di 16:", math.sqrt(16))

In questa linea, stiamo utilizzando la funzione math.sqrt() del modulo math per calcolare la radice quadrata di 16. La funzione math.sqrt() prende un singolo argomento (un numero) e restituisce la radice quadrata di quel numero.

* + **math.sqrt(16)**: Questa chiamata di funzione calcola la radice quadrata di 16, che è 4.0.

1. **Stampa del risultato**: La funzione print() stampa il messaggio "Radice quadrata di 16:" seguito dal risultato della funzione math.sqrt(16) sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo math viene importato, rendendo disponibili tutte le funzioni definite in questo modulo.
2. La funzione math.sqrt(16) viene chiamata per calcolare la radice quadrata di 16.
3. Il risultato della funzione math.sqrt(16), che è 4.0, viene passato alla funzione print().
4. La funzione print() stampa il messaggio "Radice quadrata di 16:" seguito dal risultato 4.0 sulla console.

### Output del codice

Quando il codice viene eseguito, l'output sarà:

yaml

Radice quadrata di 16: 4.0

### Riepilogo

* **Importazione del modulo math**: Il modulo math viene importato utilizzando import math.
* **Calcolo della radice quadrata**: La funzione math.sqrt(16) viene utilizzata per calcolare la radice quadrata di 16, che è 4.0.
* **Stampa del risultato**: La funzione print() stampa il messaggio "Radice quadrata di 16:" seguito dal risultato della funzione math.sqrt(16) sulla console.

Il modulo math è molto utile per eseguire operazioni matematiche complesse in Python. In questo esempio, abbiamo visto come calcolare la radice quadrata di un numero utilizzando math.sqrt().

Inizio modulo

Fine modulo

* **Calcolare il fattoriale**

import math

print("Fattoriale di 5:", math.factorial(5))

### Spiegazione del codice

1. **Importazione del modulo math**:

import math

Questa linea di codice importa il modulo math, che è una libreria standard di Python che fornisce una varietà di funzioni matematiche. Il modulo math include funzioni per eseguire operazioni matematiche comuni come il calcolo di radici quadrate, esponenziali, logaritmi, funzioni trigonometriche e altro ancora, incluso il calcolo del fattoriale.

1. **Calcolo del fattoriale**:

print("Fattoriale di 5:", math.factorial(5))

In questa linea, stiamo utilizzando la funzione math.factorial() del modulo math per calcolare il fattoriale di 5. La funzione math.factorial() prende un singolo argomento (un numero intero) e restituisce il fattoriale di quel numero.

* + **math.factorial(5)**: Questa chiamata di funzione calcola il fattoriale di 5, che è 5!. Il fattoriale di un numero n (indicato come n!) è il prodotto di tutti i numeri interi positivi da 1 a n. Quindi, 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120.

1. **Stampa del risultato**: La funzione print() stampa il messaggio "Fattoriale di 5:" seguito dal risultato della funzione math.factorial(5) sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo math viene importato, rendendo disponibili tutte le funzioni definite in questo modulo.
2. La funzione math.factorial(5) viene chiamata per calcolare il fattoriale di 5.
3. Il risultato della funzione math.factorial(5), che è 120, viene passato alla funzione print().
4. La funzione print() stampa il messaggio "Fattoriale di 5:" seguito dal risultato 120 sulla console.

### Output del codice

Quando il codice viene eseguito, l'output sarà:

yaml

Fattoriale di 5: 120

### Riepilogo

* **Importazione del modulo math**: Il modulo math viene importato utilizzando import math.
* **Calcolo del fattoriale**: La funzione math.factorial(5) viene utilizzata per calcolare il fattoriale di 5, che è 120.
* **Stampa del risultato**: La funzione print() stampa il messaggio "Fattoriale di 5:" seguito dal risultato della funzione math.factorial(5) sulla console.

Il modulo math è molto utile per eseguire operazioni matematiche complesse in Python. In questo esempio, abbiamo visto come calcolare il fattoriale di un numero utilizzando math.factorial().

Inizio modulo

Fine modulo

* **Calcolare il seno di un angolo (in radianti)**

print("Seno di π/2:", math.sin(math.pi / 2))

### Spiegazione del codice

1. **Importazione del modulo math**:

import math

Questa linea di codice importa il modulo math, che è una libreria standard di Python che fornisce una varietà di funzioni matematiche. Il modulo math include funzioni per eseguire operazioni matematiche comuni come il calcolo di radici quadrate, esponenziali, logaritmi, funzioni trigonometriche e altro ancora.

1. **Calcolo del seno di π/2**:

print("Seno di π/2:", math.sin(math.pi / 2))

In questa linea, stiamo utilizzando la funzione math.sin() del modulo math per calcolare il seno di un angolo. L'argomento passato a math.sin() è l'angolo in radianti. Per calcolare il seno di π/2 (pi greco diviso 2), utilizziamo math.pi che fornisce il valore di pi greco.

* + **math.pi**: Questa costante del modulo math rappresenta il valore di pi greco (circa 3.14159).
  + **math.pi / 2**: Questo calcola il valore di π/2.
  + **math.sin(math.pi / 2)**: Questa chiamata di funzione calcola il seno di π/2 radianti. Il seno di π/2 è 1.

1. **Stampa del risultato**: La funzione print() stampa il messaggio "Seno di π/2:" seguito dal risultato della funzione math.sin(math.pi / 2) sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo math viene importato, rendendo disponibili tutte le funzioni e le costanti definite in questo modulo.
2. La funzione math.pi / 2 calcola il valore di π/2.
3. La funzione math.sin(math.pi / 2) viene chiamata per calcolare il seno di π/2 radianti.
4. Il risultato della funzione math.sin(math.pi / 2), che è 1.0, viene passato alla funzione print().
5. La funzione print() stampa il messaggio "Seno di π/2:" seguito dal risultato 1.0 sulla console.

### Output del codice

Quando il codice viene eseguito, l'output sarà:

Seno di π/2: 1.0

### Riepilogo

* **Importazione del modulo math**: Il modulo math viene importato utilizzando import math.
* **Calcolo del seno di π/2**: La funzione math.sin(math.pi / 2) viene utilizzata per calcolare il seno di π/2 radianti, che è 1.0.
* **Stampa del risultato**: La funzione print() stampa il messaggio "Seno di π/2:" seguito dal risultato della funzione math.sin(math.pi / 2) sulla console.

Il modulo math è molto utile per eseguire operazioni matematiche complesse in Python. In questo esempio, abbiamo visto come calcolare il seno di un angolo in radianti utilizzando math.sin().

### Modulo datetime

Il modulo datetime fornisce classi per manipolare date e orari.

**Esempi:**

* **Ottenere la data e l'ora corrente**

from datetime import datetime

now = datetime.now()

print("Data e ora corrente:", now)

### Spiegazione del codice

1. **Importazione del modulo datetime**:

from datetime import datetime

Questa linea di codice importa la classe datetime dal modulo datetime, che è una libreria standard di Python per lavorare con date e orari. La classe datetime fornisce vari metodi per creare, manipolare e formattare date e orari.

1. **Ottenimento della data e ora corrente**:

now = datetime.now()

La funzione datetime.now() restituisce l'oggetto datetime che rappresenta la data e l'ora corrente. Il valore restituito include anno, mese, giorno, ora, minuto, secondo e microsecondo. Questo valore viene assegnato alla variabile now.

1. **Stampa della data e ora corrente**:

print("Data e ora corrente:", now)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Data e ora corrente:" seguito dal valore della variabile now. Questo mostrerà la data e l'ora corrente sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. La classe datetime viene importata dal modulo datetime.
2. La funzione datetime.now() viene chiamata per ottenere l'oggetto datetime che rappresenta la data e l'ora corrente.
3. L'oggetto datetime ottenuto viene assegnato alla variabile now.
4. La funzione print stampa il messaggio "Data e ora corrente:" seguito dal valore di now sulla console.

### Output del codice

L'output dipende dal momento in cui il codice viene eseguito. Ecco un esempio di output:

yaml

Data e ora corrente: 2024-06-10 14:30:45.123456

* **2024-06-10**: Rappresenta la data (anno-mese-giorno).
* **14:30:45.123456**: Rappresenta l'ora (ora:minuto

.microsecondo).

### Riepilogo

* **Importazione della classe datetime**: La classe datetime viene importata dal modulo datetime utilizzando from datetime import datetime.
* **Ottenimento della data e ora corrente**: La funzione datetime.now() viene utilizzata per ottenere la data e l'ora corrente. Il valore ottenuto viene assegnato alla variabile now.
* **Stampa della data e ora corrente**: La funzione print stampa il messaggio "Data e ora corrente:" seguito dal valore della variabile now sulla console.

Il modulo datetime è molto utile per lavorare con date e orari in Python. In questo esempio, abbiamo visto come ottenere e stampare la data e l'ora corrente utilizzando datetime.now().

* **Formattare una data**

from datetime import datetime

formatted\_date = now.strftime("%Y-%m-%d %H:%M:%S")

print("Data formattata:", formatted\_date)

### Spiegazione del codice

1. **Importazione del modulo datetime**:

from datetime import datetime

Questa linea di codice importa la classe datetime dal modulo datetime, che è una libreria standard di Python per lavorare con date e orari. La classe datetime fornisce vari metodi per creare, manipolare e formattare date e orari.

1. **Ottenimento della data e ora corrente**:

now = datetime.now()

La funzione datetime.now() restituisce l'oggetto datetime che rappresenta la data e l'ora corrente. Il valore restituito include anno, mese, giorno, ora, minuto, secondo e microsecondo. Questo valore viene assegnato alla variabile now.

1. **Formattazione della data e ora**:

formatted\_date = now.strftime("%Y-%m-%d %H:%M:%S")

La funzione strftime() è un metodo dell'oggetto datetime che converte l'oggetto datetime in una stringa formattata. Il formato della stringa è specificato utilizzando dei direttivi:

* + **%Y**: Anno con quattro cifre.
  + **%m**: Mese con due cifre (01-12).
  + **%d**: Giorno del mese con due cifre (01-31).
  + **%H**: Ora in formato 24 ore con due cifre (00-23).
  + **%M**: Minuti con due cifre (00-59).
  + **%S**: Secondi con due cifre (00-59).

Il risultato della formattazione viene assegnato alla variabile formatted\_date.

1. **Stampa della data formattata**:

print("Data formattata:", formatted\_date)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Data formattata:" seguito dal valore della variabile formatted\_date. Questo mostrerà la data e l'ora corrente sulla console in un formato leggibile.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. La classe datetime viene importata dal modulo datetime.
2. La funzione datetime.now() viene chiamata per ottenere l'oggetto datetime che rappresenta la data e l'ora corrente.
3. La funzione strftime("%Y-%m-%d %H:%M:%S") viene chiamata sull'oggetto datetime per formattare la data e l'ora corrente in una stringa leggibile.
4. La stringa formattata viene assegnata alla variabile formatted\_date.
5. La funzione print stampa il messaggio "Data formattata:" seguito dal valore di formatted\_date sulla console.

### Output del codice

L'output dipende dal momento in cui il codice viene eseguito. Ecco un esempio di output:

yaml

Data formattata: 2024-06-10 14:30:45

* **2024-06-10**: Rappresenta la data (anno-mese-giorno).
* **14:30:45**: Rappresenta l'ora (ora:minuto

).

### Riepilogo

* **Importazione della classe datetime**: La classe datetime viene importata dal modulo datetime utilizzando from datetime import datetime.
* **Ottenimento della data e ora corrente**: La funzione datetime.now() viene utilizzata per ottenere la data e l'ora corrente. Il valore ottenuto viene assegnato alla variabile now.
* **Formattazione della data e ora**: La funzione strftime("%Y-%m-%d %H:%M:%S") viene utilizzata per formattare la data e l'ora corrente in una stringa leggibile. Il risultato viene assegnato alla variabile formatted\_date.
* **Stampa della data formattata**: La funzione print stampa il messaggio "Data formattata:" seguito dal valore della variabile formatted\_date sulla console.

In questo esempio, abbiamo visto come formattare una data e un'ora corrente utilizzando il metodo strftime() dell'oggetto datetime.

* **Calcolare la differenza tra due date**

from datetime import timedelta

future\_date = now + timedelta(days=5)

print("Data futura:", future\_date)

### Spiegazione del codice

1. **Importazione delle classi datetime e timedelta**:

from datetime import datetime, timedelta

Questa linea di codice importa la classe datetime e la classe timedelta dal modulo datetime. La classe datetime viene utilizzata per lavorare con date e orari, mentre la classe timedelta viene utilizzata per rappresentare la differenza tra due date o per manipolare date.

1. **Ottenimento della data e ora corrente**:

now = datetime.now()

La funzione datetime.now() restituisce un oggetto datetime che rappresenta la data e l'ora corrente. Il valore restituito include anno, mese, giorno, ora, minuto, secondo e microsecondo. Questo valore viene assegnato alla variabile now.

1. **Calcolo della data futura**:

future\_date = now + timedelta(days=5)

La classe timedelta viene utilizzata per creare un oggetto che rappresenta un intervallo di tempo. In questo caso, timedelta(days=5) crea un intervallo di tempo di 5 giorni. Quando sommiamo questo intervallo di tempo all'oggetto datetime corrente (now), otteniamo una nuova data che è 5 giorni nel futuro. Questo valore viene assegnato alla variabile future\_date.

1. **Stampa della data futura**:

print("Data futura:", future\_date)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Data futura:" seguito dal valore della variabile future\_date. Questo mostrerà la data che è 5 giorni nel futuro rispetto alla data e ora corrente.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Le classi datetime e timedelta vengono importate dal modulo datetime.
2. La funzione datetime.now() viene chiamata per ottenere l'oggetto datetime che rappresenta la data e l'ora corrente.
3. La classe timedelta(days=5) viene utilizzata per creare un intervallo di tempo di 5 giorni.
4. L'intervallo di tempo viene aggiunto all'oggetto datetime corrente (now) per calcolare la data futura. Il risultato viene assegnato alla variabile future\_date.
5. La funzione print stampa il messaggio "Data futura:" seguito dal valore di future\_date sulla console.

### Output del codice

L'output dipende dal momento in cui il codice viene eseguito. Ecco un esempio di output, supponendo che la data e l'ora corrente sia il 10 giugno 2024:

yaml

Data futura: 2024-06-15 14:30:45.123456

* **2024-06-15**: Rappresenta la data futura (anno-mese-giorno), che è 5 giorni dopo la data corrente.
* **14:30:45.123456**: Rappresenta l'ora futura (ora:minuto

.microsecondo), che è la stessa dell'ora corrente.

### Riepilogo

* **Importazione delle classi datetime e timedelta**: Le classi datetime e timedelta vengono importate dal modulo datetime utilizzando from datetime import datetime, timedelta.
* **Ottenimento della data e ora corrente**: La funzione datetime.now() viene utilizzata per ottenere la data e l'ora corrente. Il valore ottenuto viene assegnato alla variabile now.
* **Calcolo della data futura**: La classe timedelta(days=5) viene utilizzata per creare un intervallo di tempo di 5 giorni, che viene aggiunto alla data e ora corrente per ottenere una data futura. Il risultato viene assegnato alla variabile future\_date.
* **Stampa della data futura**: La funzione print stampa il messaggio "Data futura:" seguito dal valore della variabile future\_date sulla console.

In questo esempio, abbiamo visto come utilizzare la classe timedelta per calcolare una data futura aggiungendo un intervallo di tempo a una data corrente.

### Modulo json

Il modulo json permette di lavorare con dati JSON, un formato di testo per la serializzazione di dati strutturati.

**Esempi:**

* **Convertire un dizionario in una stringa JSON**

import json

data = {

'name': 'Alice',

'age': 30,

'city': 'New York'

}

json\_string = json.dumps(data)

print("Stringa JSON:", json\_string)

### Spiegazione del codice

1. **Importazione del modulo json**:

import json

Questa linea di codice importa il modulo json, che è una libreria standard di Python che fornisce funzioni per lavorare con dati JSON (JavaScript Object Notation). Il JSON è un formato di testo utilizzato per rappresentare strutture di dati semplici e oggetti, ed è comunemente usato per lo scambio di dati tra client e server nelle applicazioni web.

1. **Definizione di un dizionario Python**:

data = {

'name': 'Alice',

'age': 30,

'city': 'New York'

}

Questa linea di codice definisce un dizionario Python chiamato data. Un dizionario in Python è una collezione non ordinata di coppie chiave-valore, dove ogni chiave è unica. In questo caso, il dizionario data contiene informazioni su una persona: il nome ('name'), l'età ('age'), e la città ('city').

1. **Conversione del dizionario in una stringa JSON**:

json\_string = json.dumps(data)

La funzione json.dumps() converte un oggetto Python (come un dizionario) in una stringa JSON. Il dizionario data viene passato come argomento alla funzione json.dumps(), che restituisce una stringa JSON che rappresenta il dizionario. Questa stringa JSON viene assegnata alla variabile json\_string.

1. **Stampa della stringa JSON**:

print("Stringa JSON:", json\_string)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Stringa JSON:" seguito dal valore della variabile json\_string. Questo mostrerà la rappresentazione JSON del dizionario sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo json viene importato, rendendo disponibili tutte le funzioni definite in questo modulo.
2. Un dizionario Python chiamato data viene definito con le informazioni di una persona.
3. La funzione json.dumps(data) viene chiamata per convertire il dizionario data in una stringa JSON. Il risultato viene assegnato alla variabile json\_string.
4. La funzione print stampa il messaggio "Stringa JSON:" seguito dal valore di json\_string sulla console.

### Output del codice

L'output sarà:

css

Stringa JSON: {"name": "Alice", "age": 30, "city": "New York"}

### Riepilogo

* **Importazione del modulo json**: Il modulo json viene importato utilizzando import json.
* **Definizione di un dizionario Python**: Un dizionario chiamato data viene definito con le informazioni di una persona.
* **Conversione del dizionario in una stringa JSON**: La funzione json.dumps(data) viene utilizzata per convertire il dizionario data in una stringa JSON. Il risultato viene assegnato alla variabile json\_string.
* **Stampa della stringa JSON**: La funzione print stampa il messaggio "Stringa JSON:" seguito dal valore della variabile json\_string sulla console.

Il modulo json è molto utile per lavorare con dati JSON in Python. In questo esempio, abbiamo visto come convertire un dizionario Python in una stringa JSON utilizzando json.dumps().

* **Convertire una stringa JSON in un dizionario**

import json

json\_data = '{"name": "Alice", "age": 30, "city": "New York"}'

parsed\_data = json.loads(json\_data)

print("Dati dal JSON:", parsed\_data)

### Spiegazione del codice

1. **Importazione del modulo json**:

import json

Questa linea di codice importa il modulo json, che è una libreria standard di Python che fornisce funzioni per lavorare con dati JSON (JavaScript Object Notation). Il JSON è un formato di testo utilizzato per rappresentare strutture di dati semplici e oggetti, ed è comunemente usato per lo scambio di dati tra client e server nelle applicazioni web.

1. **Definizione di una stringa JSON**:

json\_data = '{"name": "Alice", "age": 30, "city": "New York"}'

Questa linea di codice definisce una stringa JSON chiamata json\_data. La stringa JSON rappresenta un oggetto con tre proprietà: name (nome), age (età), e city (città). La stringa JSON è formattata come un oggetto JSON, con chiavi e valori racchiusi tra virgolette doppie.

1. **Parsing della stringa JSON in un dizionario Python**:

parsed\_data = json.loads(json\_data)

La funzione json.loads() converte una stringa JSON in un oggetto Python. In questo caso, la stringa JSON json\_data viene passata come argomento alla funzione json.loads(), che restituisce un dizionario Python che rappresenta l'oggetto JSON. Questo dizionario viene assegnato alla variabile parsed\_data.

1. **Stampa dei dati dal JSON**:

print("Dati dal JSON:", parsed\_data)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Dati dal JSON:" seguito dal valore della variabile parsed\_data. Questo mostrerà il dizionario Python ottenuto dalla conversione della stringa JSON sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo json viene importato, rendendo disponibili tutte le funzioni definite in questo modulo.
2. Una stringa JSON chiamata json\_data viene definita con le informazioni di una persona.
3. La funzione json.loads(json\_data) viene chiamata per convertire la stringa JSON json\_data in un dizionario Python. Il risultato viene assegnato alla variabile parsed\_data.
4. La funzione print stampa il messaggio "Dati dal JSON:" seguito dal valore di parsed\_data sulla console.

### Output del codice

L'output sarà:

css

Dati dal JSON: {'name': 'Alice', 'age': 30, 'city': 'New York'}

### Riepilogo

* **Importazione del modulo json**: Il modulo json viene importato utilizzando import json.
* **Definizione di una stringa JSON**: Una stringa JSON chiamata json\_data viene definita con le informazioni di una persona.
* **Parsing della stringa JSON in un dizionario Python**: La funzione json.loads(json\_data) viene utilizzata per convertire la stringa JSON json\_data in un dizionario Python. Il risultato viene assegnato alla variabile parsed\_data.
* **Stampa dei dati dal JSON**: La funzione print stampa il messaggio "Dati dal JSON:" seguito dal valore della variabile parsed\_data sulla console.

Il modulo json è molto utile per lavorare con dati JSON in Python. In questo esempio, abbiamo visto come convertire una stringa JSON in un dizionario Python utilizzando json.loads().

Inizio modulo

Fine modulo

* **Leggere e scrivere file JSON**

import json

# Scrivere dati su un file JSON

with open('data.json', 'w') as json\_file:

json.dump(data, json\_file)

# Leggere dati da un file JSON

with open('data.json', 'r') as json\_file:

loaded\_data = json.load(json\_file)

print("Dati letti dal file JSON:", loaded\_data)

### Spiegazione del codice

1. **Importazione del modulo json**:

import json

Questa linea di codice importa il modulo json, che è una libreria standard di Python per lavorare con dati JSON (JavaScript Object Notation).

1. **Definizione dei dati**:

data = {

'name': 'Alice',

'age': 30,

'city': 'New York'

}

Questa linea definisce un dizionario Python chiamato data, contenente informazioni su una persona.

1. **Scrivere dati su un file JSON**:

with open('data.json', 'w') as json\_file:

json.dump(data, json\_file)

Queste linee di codice aprono un file chiamato data.json in modalità scrittura ('w'). Il blocco with open(...) as ...: garantisce che il file venga chiuso correttamente dopo che l'operazione di scrittura è completata. La funzione json.dump(data, json\_file) scrive il contenuto del dizionario data nel file data.json in formato JSON.

1. **Leggere dati da un file JSON**:

with open('data.json', 'r') as json\_file:

loaded\_data = json.load(json\_file)

Queste linee di codice aprono il file data.json in modalità lettura ('r'). Il blocco with open(...) as ...: garantisce che il file venga chiuso correttamente dopo che l'operazione di lettura è completata. La funzione json.load(json\_file) legge il contenuto del file data.json e lo converte in un dizionario Python, che viene assegnato alla variabile loaded\_data.

1. **Stampa dei dati letti dal file JSON**:

print("Dati letti dal file JSON:", loaded\_data)

Questa linea di codice utilizza la funzione print per stampare il messaggio "Dati letti dal file JSON:" seguito dal valore della variabile loaded\_data. Questo mostrerà il contenuto del file JSON letto sulla console.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. Il modulo json viene importato.
2. Un dizionario Python chiamato data viene definito con le informazioni di una persona.
3. Il file data.json viene aperto in modalità scrittura e il contenuto del dizionario data viene scritto nel file in formato JSON.
4. Il file data.json viene aperto in modalità lettura e il contenuto del file viene letto e convertito in un dizionario Python, assegnato alla variabile loaded\_data.
5. La funzione print stampa il messaggio "Dati letti dal file JSON:" seguito dal valore di loaded\_data sulla console.

### Output del codice

L'output sarà:

css

Dati letti dal file JSON: {'name': 'Alice', 'age': 30, 'city': 'New York'}

### Riepilogo

* **Importazione del modulo json**: Il modulo json viene importato utilizzando import json.
* **Definizione dei dati**: Un dizionario Python chiamato data viene definito con le informazioni di una persona.
* **Scrivere dati su un file JSON**: Il contenuto del dizionario data viene scritto nel file data.json in formato JSON utilizzando json.dump().
* **Leggere dati da un file JSON**: Il contenuto del file data.json viene letto e convertito in un dizionario Python utilizzando json.load().
* **Stampa dei dati letti dal file JSON**: La funzione print stampa il messaggio "Dati letti dal file JSON:" seguito dal valore della variabile loaded\_data sulla console.

In questo esempio, abbiamo visto come scrivere e leggere dati JSON su e da un file utilizzando le funzioni json.dump() e json.load().

Questi esempi coprono alcune delle funzionalità principali di ciascun modulo, permettendo di comprenderne l'uso pratico e le potenzialità.

 **Modulo os**

* Documentazione ufficiale: [os — Miscellaneous operating system interfaces](https://docs.python.org/3/library/os.html)
* Link: https://docs.python.org/3/library/os.html

 **Modulo sys**

* Documentazione ufficiale: [sys — System-specific parameters and functions](https://docs.python.org/3/library/sys.html)
* Link: https://docs.python.org/3/library/sys.html

 **Modulo math**

* Documentazione ufficiale: [math — Mathematical functions](https://docs.python.org/3/library/math.html)
* Link: https://docs.python.org/3/library/math.html

 **Modulo datetime**

* Documentazione ufficiale: [datetime — Basic date and time types](https://docs.python.org/3/library/datetime.html)
* Link: https://docs.python.org/3/library/datetime.html

 **Modulo json**

* Documentazione ufficiale: [json — JSON encoder and decoder](https://docs.python.org/3/library/json.html)
* Link: https://docs.python.org/3/library/json.html

***Programmazione Orientata agli Oggetti in Python***

Ora possiamo passare alla sezione \_\_\_ del nostro percorso di Python, che si concentrerà sulla programmazione orientata agli oggetti (OOP). La OOP è un paradigma di programmazione che utilizza "oggetti" – strutture dati che contengono sia dati (attributi) che procedure (metodi) – per rendere il codice più riutilizzabile, organizzato e facile da gestire.

Classi e Oggetti

La programmazione orientata agli oggetti in Python si basa su "classi" che definiscono gli oggetti. Una classe è come un modello per creare oggetti; ogni oggetto creato da una classe è chiamato "istanza" della classe.

**Classi**

Le classi sono una caratteristica fondamentale della programmazione orientata agli oggetti (OOP). Servono come modello per creare oggetti (istanze). Le classi definiscono attributi (dati) e metodi (funzionalità) che le istanze create dalla classe avranno. In pratica, una classe è una sorta di "schema" o "progetto" per creare oggetti specifici.

**Istanze**

Le istanze sono oggetti concreti creati da una classe. Ogni istanza della classe ha il proprio set di dati (attributi) e può utilizzare i metodi definiti nella classe. Creare un'istanza significa essenzialmente creare un oggetto basato sulla classe.

**Attributi**

Gli attributi sono variabili che appartengono a una classe o a un'istanza di una classe. Gli attributi di classe sono condivisi da tutte le istanze della classe, mentre gli attributi di istanza sono specifici per ogni istanza. Gli attributi vengono utilizzati per memorizzare lo stato o le proprietà dell'oggetto.

**Metodi**

I metodi sono funzioni definite all'interno di una classe che descrivono i comportamenti che un oggetto della classe può eseguire. I metodi possono manipolare gli attributi dell'oggetto e spesso operano su di essi. I metodi di istanza richiedono un parametro self, che si riferisce all'istanza corrente dell'oggetto.

**Riassumendo:**

* **Classi**: Servono come modelli o progetti per creare oggetti.
* **Istanze**: Sono oggetti concreti creati da una classe, con il proprio stato e comportamento.
* **Attributi**: Sono variabili che memorizzano lo stato o le proprietà di un oggetto.
* **Metodi**: Sono funzioni definite all'interno di una classe che descrivono i comportamenti degli oggetti.

**Classi e oggetti** Le classi sono modelli per creare oggetti. Un oggetto è un'istanza di una classe.

Nel seguente esempio vi mostro due script completi con la creazione di classi, istanze, attributi e metodi:

print ("========= Esempio 1: =========")

#solo a scopo informativo

class Automobile:  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 print("Chiamato \_\_new\_\_")  
 instance = super().\_\_new\_\_(cls) # Creazione dell'istanza  
 return instance  
  
 def \_\_init\_\_(self, marca, modello, anno):  
 print("Chiamato \_\_init\_\_")  
 self.marca = marca  
 self.modello = modello  
 self.anno = anno  
  
 def descrizione(self):  
 return f"{self.marca} {self.modello} del {self.anno}"  
  
 def accendi(self):  
 return f"{self.marca} {self.modello} è accesa."  
  
# Creazione di un'istanza della classe Automobile  
auto1 = Automobile("Fiat", "Punto", 2020)  
#auto2 = Automobile("Tesla", "Model S", 2022)  
"""se decommentassi auto2, nell'output otterrei:   
Chiamato \_\_new\_\_  
Chiamato \_\_init\_\_  
Chiamato \_\_new\_\_  
Chiamato \_\_init\_\_,   
questo perchè ho due istanze auto1 e auto2"""  
# Utilizzo dei metodi  
print(auto1.descrizione())  
print(auto1.accendi())  
  
print("========= Fine esempio 1 ========= \n")  
#per semplicità viene omesso il costruttore new  
#pertanto tutti le classi python iniziano con init direttamente

print("========= Esempio 2: =========")

#Qui di seguito un esempio completo commentato con classe, metodi, istanze e attributi

print("USATE QUESTO")

class Automobile: # classe  
 def \_\_init\_\_(self, marca, modello, anno): # metodo  
 self.marca = marca # attributo  
 self.modello = modello # attributo  
 self.anno = anno # attributo  
  
 def descrizione(self): # metodo  
 print(f"{self.marca} {self.modello} del {self.anno}") # metodo  
 #notare la print direttamente nel metodo  
  
 def accendi(self): # metodo  
 return f"{self.marca} {self.modello} è accesa." # metodo  
#qui ho utilizzato return, per cui quando chiamo il metodo devo utilizzare print  
  
# Creazione di istanze della classe Automobile  
auto1 = Automobile("Fiat", "Punto", 2020) # istanza  
auto2 = Automobile("Tesla", "Model S", 2022) # istanza  
  
# Utilizzo dei metodi  
  
#print è nella creazione del metodo  
auto1.descrizione() # metodo  
  
#qui non c'era è allora devo inserire print  
print(auto2.accendi()) # metodo  
  
print("========= Fine esempio 2 =========")  
  
"""  
La maggior parte delle classi in Python rappresenta tipi mutabili (cioè, i loro attributi possono cambiare dopo la creazione dell'istanza). \_\_init\_\_ è perfetto per questo scopo.  
\_\_new\_\_ è più adatto ai tipi immutabili, come le tuple, gli interi, le stringhe, ecc., dove l'oggetto deve essere completamente costruito prima che qualsiasi cosa possa essere fatta su di esso.  
In altre parole nel 99% dei casi utilizzerete init  
"""

**Nota:** La f nella stringa f"{self.marca} {self.modello} del {self.anno}" indica che si tratta di una **f-string** (o formatted string literal), una funzionalità introdotta in Python 3.6. Le f-string rendono il processo di formattazione delle stringhe più leggibile e conciso rispetto ai metodi tradizionali come la concatenazione di stringhe o l'uso del metodo format.

Le due classi hanno lo stesso compito. La prima parte del codice (il listato con \_new\_) viene utilizzata per metodi statici. Qui di seuito vi spiego cosa avviene:

- \_\_new\_\_ è un metodo speciale che viene chiamato prima di \_\_init\_\_.

- cls rappresenta la classe stessa (Automobile in questo caso).

- super().\_\_new\_\_(cls) crea effettivamente una nuova istanza della classe.

- Questo metodo ritorna l'istanza creata.

Il resto del codice è invariato.

Il metodo \_\_init\_\_ è uno speciale metodo in Python, noto come "costruttore". Viene chiamato automaticamente quando viene creata una nuova istanza di una classe. Il suo scopo principale è inizializzare gli attributi dell'oggetto.

- Il metodo \_\_init\_\_ prende come argomento *self*, che rappresenta l'istanza dell'oggetto stesso. È tramite *self* che si possono definire gli attributi dell'istanza.

- Oltre a self, il metodo \_\_init\_\_ può accettare altri parametri che vengono utilizzati per inizializzare gli attributi dell'istanza.

- A differenza di alcuni altri linguaggi di programmazione, in Python, il vero costruttore è il metodo \_\_new\_\_, che è responsabile della creazione di una nuova istanza. Tuttavia, nella maggior parte dei casi, non è necessario sovrascrivere \_\_new\_\_.

- Il metodo \_\_init\_\_ viene chiamato subito dopo la creazione dell'istanza per inizializzarla.

**Creazione di una nuova istanza**:

* Quando crei una nuova istanza della classe Automobile con auto1 = Automobile("Fiat", "Punto", 2020), Python chiama automaticamente il metodo \_\_init\_\_ della classe Automobile.

**Inizializzazione degli attributi**:

* All'interno di \_\_init\_\_, self.marca viene impostato su "Fiat", self.modello su "Punto", self.anno su 2020, e self.accensione su False.

**Accesso agli attributi**:

* Una volta che l'oggetto è stato creato e inizializzato, puoi accedere ai suoi attributi e metodi. Ad esempio, auto1.descrizione() restituirà "Fiat Punto del 2020".

Nell’esempio precedente:

* Quando auto1 e auto2 vengono creati, il metodo \_\_init\_\_ viene chiamato automaticamente, inizializzando i rispettivi attributi.
* Successivamente, i metodi descrizione e accendi possono essere chiamati sulle istanze per interagire con i loro attributi.

Il metodo \_\_init\_\_ è fondamentale per inizializzare gli oggetti in Python. Ti permette di impostare lo stato iniziale degli attributi e di eseguire qualsiasi configurazione necessaria ogni volta che viene creata una nuova istanza della classe.

Il metodo \_\_new\_\_ in Python è responsabile della creazione di una nuova istanza della classe. Viene chiamato prima del metodo \_\_init\_\_. Solitamente, non è necessario sovrascrivere \_\_new\_\_, a meno che non si desideri un controllo avanzato sulla creazione delle istanze.

La maggior parte delle classi in Python rappresenta tipi *mutabili* (cioè, i loro attributi possono cambiare dopo la creazione dell'istanza). **\_\_init\_\_ è perfetto per questo scopo**. \_\_new\_\_ è più adatto ai tipi *immutabili*, come le tuple, gli interi, le stringhe, ecc., dove l'oggetto deve essere completamente costruito prima che qualsiasi cosa possa essere fatta su di esso. **In altre parole nel 99% dei casi utilizzerete init**

Di seguito approfondiamo passo passo la creazione e l’utilizzo di una classe. Questa volta utilizzeremo la classe *Studente*

Esempio: Definizione di una Classe

class Studente:  
 def \_\_init\_\_(self, nome, età):  
 self.nome = nome  
 self.età = età  
  
 def saluta(self):  
 print(f"Ciao, mi chiamo {self.nome} e ho {self.età} anni")  
  
  
# Creare un oggetto  
alice = Studente("Alice", 25)  
alice.saluta() # Output: Ciao, mi chiamo Alice e ho 25 anni

### Spiegazione del codice

1. **Definizione della classe Studente**:

class Studente:

Questa linea di codice definisce una nuova classe chiamata Studente. Le classi in Python sono modelli per creare oggetti che raggruppano dati e comportamenti.

1. **Metodo \_\_init\_\_ (costruttore)**:

def \_\_init\_\_(self, nome, età):

self.nome = nome

self.età = età

Il metodo \_\_init\_\_ è un costruttore speciale che viene chiamato quando viene creato un nuovo oggetto della classe. Questo metodo inizializza l'oggetto con i dati forniti. In questo caso, il costruttore prende due argomenti, nome ed età, e li assegna agli attributi dell'oggetto self.nome e self.età.

1. **Metodo saluta**:

def saluta(self):

print(f"Ciao, mi chiamo {self.nome} e ho {self.età} anni")

Questo metodo è una funzione che appartiene alla classe Studente. Il metodo saluta stampa un messaggio che include il nome e l'età dello studente utilizzando gli attributi self.nome e self.età.

1. **Creazione di un oggetto**:

alice = Studente("Alice", 25)

Questa linea di codice crea un nuovo oggetto alice della classe Studente. Viene chiamato il costruttore \_\_init\_\_ con i parametri nome="Alice" ed età=25, e questi valori vengono assegnati agli attributi self.nome e self.età dell'oggetto alice.

1. **Chiamata del metodo saluta**:

alice.saluta() # Output: Ciao, mi chiamo Alice e ho 25 anni

Questa linea di codice chiama il metodo saluta sull'oggetto alice. Il metodo saluta stampa il messaggio "Ciao, mi chiamo Alice e ho 25 anni" utilizzando gli attributi dell'oggetto alice.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Definizione della classe Studente**:
   * La classe Studente viene definita con un costruttore \_\_init\_\_ e un metodo saluta.
2. **Creazione di un oggetto alice**:
   * L'oggetto alice della classe Studente viene creato con nome="Alice" ed età=25. Questi valori vengono assegnati agli attributi self.nome e self.età dell'oggetto alice.
3. **Chiamata del metodo saluta sull'oggetto alice**:
   * Il metodo saluta stampa il messaggio "Ciao, mi chiamo Alice e ho 25 anni" sulla console utilizzando gli attributi self.nome e self.età dell'oggetto alice.

### Output del codice

L'output sarà:

Ciao, mi chiamo Alice e ho 25 anni

### Riepilogo

* **Definizione della classe**: La classe Studente viene definita con un costruttore \_\_init\_\_ che inizializza gli attributi nome ed età, e un metodo saluta che stampa un messaggio di saluto.
* **Creazione di un oggetto**: Un oggetto alice della classe Studente viene creato con il nome "Alice" e l'età di 25 anni.
* **Chiamata del metodo**: Il metodo saluta dell'oggetto alice viene chiamato per stampare un messaggio di saluto che include il nome e l'età di Alice.

In questo esempio, abbiamo visto come definire una classe con un costruttore e un metodo, come creare un oggetto di quella classe e come chiamare un metodo dell'oggetto utilizzando Python.

**Ereditarietà**

L'ereditarietà è un modo per formare nuove classi usando classi che sono state già definite. Le nuove classi possono sovrascrivere o estendere i comportamenti delle classi esistenti. In altre parole l’ereditarietà permette di creare una nuova classe che eredita i metodi e le proprietà di un'altra classe.

Qui di seguito vengono mostrati brevemente due metodi per la creazione di una classe, con costruttore e senza costrottore:

CON COSTRUTTORE:

class Cane(Animale):  
 def \_\_init\_\_(self, nome, razza):  
 super().\_\_init\_\_(nome, "Cane")  
 self.razza = razza

SENZA COSTRUTTORE:

class Cane(Animale):  
 def set\_razza(self, razza):# NOTARE L’USO DI SET  
 self.razza = razza

Nel primo caso

**Inizializzazione diretta degli attributi**:

* Il costruttore (\_\_init\_\_) permette di inizializzare direttamente gli attributi nome e razza quando viene creata un'istanza della classe Cane.
* La specie è impostata direttamente a "Cane" nel costruttore.

**Chiamata al costruttore della classe base**:

* super().\_\_init\_\_(nome, "Cane") chiama il costruttore della classe base Animale per inizializzare nome e specie.

**Semplificazione**:

* Tutte le informazioni necessarie per creare un'istanza di Cane sono fornite direttamente durante l'istanziazione.

**Chiarezza**:

* Il codice è più conciso e chiaro, poiché tutti gli attributi vengono impostati in un unico punto.

Nel secondo caso:

**Inizializzazione separata degli attributi**:

* Gli attributi nome e specie sono inizializzati tramite il costruttore della classe base Animale.
* razza è impostato separatamente utilizzando il metodo set\_razza. Senza produrrebbe errore.

**Mantenimento del costruttore della classe base**:

* Non c'è un costruttore personalizzato, quindi viene utilizzato direttamente quello della classe base Animale. Questo potrebbe portare a comportamenti imprevisti.

**Flessibilità**:

* La razza può essere impostata o modificata successivamente alla creazione dell'istanza, non al momento dell'istanziazione.

**Maggiore verbosità**:

* L'inizializzazione richiede più passaggi, ovvero la creazione dell'istanza seguita dalla chiamata a set\_razza.

In altre parole è consigliabile utilizzare il COSTRUTTORE.

Esempio: Ereditarietà in Python

class Persona:

def \_\_init\_\_(self, nome, età):

self.nome = nome

self.età = età

def saluta(self):

print(f"Ciao, mi chiamo {self.nome}")

class Studente(Persona):

def \_\_init\_\_(self, nome, età, università):

super().\_\_init\_\_(nome, età)

self.università = università

def saluta(self):

print(f"Ciao, mi chiamo {self.nome} e studio a {self.università}")

# Creare un oggetto

mario = Studente("Mario", 22, "La Sapienza")

mario.saluta() # Output: Ciao, mi chiamo Mario e studio a La Sapienza

### Spiegazione del codice

1. **Definizione della classe Persona**:

class Persona:

def \_\_init\_\_(self, nome, età):

self.nome = nome

self.età = età

def saluta(self):

print(f"Ciao, mi chiamo {self.nome}")

Questa parte del codice definisce una classe Persona con un costruttore \_\_init\_\_ che inizializza gli attributi nome ed età, e un metodo saluta che stampa un messaggio di saluto con il nome della persona.

1. **Definizione della classe Studente che eredita da Persona**:

class Studente(Persona):

def \_\_init\_\_(self, nome, età, università):

super().\_\_init\_\_(nome, età)

self.università = università

def saluta(self):

print(f"Ciao, mi chiamo {self.nome} e studio a {self.università}")

Questa parte del codice definisce una classe Studente che eredita dalla classe Persona. La classe Studente aggiunge un nuovo attributo università e sovrascrive il metodo saluta.

* + **Costruttore \_\_init\_\_**: Il costruttore della classe Studente chiama il costruttore della classe base Persona utilizzando super().\_\_init\_\_(nome, età), che inizializza gli attributi nome ed età. Poi, aggiunge un nuovo attributo università.
  + **Metodo saluta**: Il metodo saluta della classe Studente sovrascrive il metodo saluta della classe base Persona. Questo metodo stampa un messaggio che include il nome e l'università dello studente.

1. **Creazione di un oggetto Studente**:

mario = Studente("Mario", 22, "La Sapienza")

mario.saluta() # Output: Ciao, mi chiamo Mario e studio a La Sapienza

Questa parte del codice crea un oggetto mario della classe Studente con i parametri nome="Mario", età=22, e università="La Sapienza". Quando si chiama il metodo saluta sull'oggetto mario, viene eseguito il metodo saluta della classe Studente, che stampa il messaggio "Ciao, mi chiamo Mario e studio a La Sapienza".

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Definizione della classe Persona**:
   * La classe Persona viene definita con un costruttore \_\_init\_\_ che inizializza nome ed età, e un metodo saluta che stampa un messaggio di saluto.
2. **Definizione della classe Studente**:
   * La classe Studente eredita da Persona, aggiunge un nuovo attributo università, e sovrascrive il metodo saluta.
3. **Creazione dell'oggetto mario**:
   * L'oggetto mario viene creato con il nome "Mario", l'età 22, e l'università "La Sapienza".
   * Il costruttore della classe Studente chiama il costruttore della classe base Persona per inizializzare nome ed età, e poi aggiunge l'attributo università.
4. **Chiamata del metodo saluta**:
   * Quando si chiama mario.saluta(), viene eseguito il metodo saluta della classe Studente, che stampa "Ciao, mi chiamo Mario e studio a La Sapienza".

### Output del codice

L'output sarà:

css

Ciao, mi chiamo Mario e studio a La Sapienza

### Riepilogo

* **Definizione della classe Persona**: La classe Persona ha un costruttore che inizializza nome ed età, e un metodo saluta che stampa un messaggio di saluto.
* **Definizione della classe Studente**: La classe Studente eredita da Persona, aggiunge l'attributo università, e sovrascrive il metodo saluta.
* **Creazione di un oggetto Studente**: L'oggetto mario viene creato con il nome "Mario", l'età 22, e l'università "La Sapienza".
* **Chiamata del metodo saluta**: Il metodo saluta della classe Studente stampa un messaggio che include il nome e l'università dello studente.

In questo esempio, abbiamo visto come utilizzare l'ereditarietà in Python per creare una classe che estende le funzionalità di un'altra classe e come sovrascrivere metodi nella classe derivata.

**Polimorfismo**

Il polimorfismo permette di utilizzare metodi con lo stesso nome in classi diverse. L'azione specifica che un metodo esegue può dipendere dalla classe di un oggetto, quindi il polimorfismo permette a metodi di diverse classi di avere lo stesso nome ma comportamenti differenti.

Esempio: Polimorfismo in Python

class Cane:

def parla(self):

print("Bau")

class Gatto:

def parla(self):

print("Miao")

# Esempio di polimorfismo

animali = [Cane(), Gatto()]

for animale in animali:

animale.parla()

### Spiegazione del codice

1. **Definizione della classe Cane**:

class Cane:

def parla(self):

print("Bau")

Questa parte del codice definisce una classe chiamata Cane. La classe contiene un metodo parla che stampa "Bau" quando viene chiamato. Questo metodo rappresenta il modo in cui un cane "parla".

1. **Definizione della classe Gatto**:

class Gatto:

def parla(self):

print("Miao")

Questa parte del codice definisce una classe chiamata Gatto. La classe contiene un metodo parla che stampa "Miao" quando viene chiamato. Questo metodo rappresenta il modo in cui un gatto "parla".

1. **Esempio di polimorfismo**:

animali = [Cane(), Gatto()]

for animale in animali:

animale.parla()

* + **Creazione di una lista di animali**:

animali = [Cane(), Gatto()]

Questa linea di codice crea una lista chiamata animali che contiene due oggetti: un oggetto Cane e un oggetto Gatto.

* + **Iterazione attraverso la lista di animali**:

for animale in animali:

animale.parla()

Questo ciclo for itera attraverso ogni elemento della lista animali. Per ogni animale nella lista, chiama il metodo parla. Grazie al polimorfismo, Python chiama il metodo appropriato per il tipo specifico di animale (cioè, Cane o Gatto).

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Definizione delle classi Cane e Gatto**:
   * La classe Cane viene definita con un metodo parla che stampa "Bau".
   * La classe Gatto viene definita con un metodo parla che stampa "Miao".
2. **Creazione di una lista di animali**:
   * Viene creata una lista animali contenente un oggetto Cane e un oggetto Gatto.
3. **Iterazione attraverso la lista di animali**:
   * Il ciclo for itera attraverso ogni oggetto nella lista animali.
   * Per l'oggetto Cane, viene chiamato il metodo parla, che stampa "Bau".
   * Per l'oggetto Gatto, viene chiamato il metodo parla, che stampa "Miao".

### Output del codice

L'output sarà:

Bau

Miao

### Riepilogo

* **Definizione delle classi Cane e Gatto**: Le classi Cane e Gatto hanno entrambe un metodo parla che stampa un suono specifico ("Bau" per Cane e "Miao" per Gatto).
* **Creazione di una lista di animali**: Viene creata una lista animali che contiene un oggetto Cane e un oggetto Gatto.
* **Esempio di polimorfismo**: Il ciclo for itera attraverso la lista animali e chiama il metodo parla su ogni oggetto. Grazie al polimorfismo, Python chiama il metodo appropriato per il tipo specifico di animale.

In questo esempio, abbiamo visto come utilizzare il polimorfismo in Python per chiamare metodi specifici su oggetti di diverse classi, dimostrando come lo stesso metodo (parla) possa avere comportamenti diversi a seconda dell'oggetto su cui viene chiamato.

Incapsulamento

L'incapsulamento è il concetto di nascondere gli attributi di una classe, prevenendo l'accesso diretto agli stessi dall'esterno e facendo in modo che possano essere modificati solo tramite metodi definiti.

Esempio incapsulamento

class ContoBancario:

def \_\_init\_\_(self, saldo):

self.\_\_saldo = saldo

def deposita(self, importo):

if importo > 0:

self.\_\_saldo += importo

def ottieni\_saldo(self):

return self.\_\_saldo

# Creare un oggetto

conto = ContoBancario(1000)

conto.deposita(500)

print(conto.ottieni\_saldo()) # Output: 1500

### Spiegazione del codice

1. **Definizione della classe ContoBancario**:

class ContoBancario:

Questa linea di codice definisce una nuova classe chiamata ContoBancario. Le classi in Python sono modelli per creare oggetti che raggruppano dati e comportamenti.

1. **Costruttore \_\_init\_\_**:

def \_\_init\_\_(self, saldo):

self.\_\_saldo = saldo

Il metodo \_\_init\_\_ è un costruttore speciale che viene chiamato quando viene creato un nuovo oggetto della classe. Questo metodo inizializza l'oggetto con il saldo fornito. L'attributo \_\_saldo è privato (indicata da due underscore \_\_) e non può essere direttamente accessibile dall'esterno della classe.

1. **Metodo deposita**:

def deposita(self, importo):

if importo > 0:

self.\_\_saldo += importo

Questo metodo consente di depositare un importo sul conto bancario. Se l'importo è maggiore di zero, viene aggiunto al saldo corrente.

1. **Metodo ottieni\_saldo**:

def ottieni\_saldo(self):

return self.\_\_saldo

Questo metodo restituisce il saldo corrente del conto bancario. Poiché \_\_saldo è un attributo privato, questo metodo fornisce un modo per accedervi dall'esterno della classe.

1. **Creazione e utilizzo di un oggetto ContoBancario**:

conto = ContoBancario(1000)

conto.deposita(500)

print(conto.ottieni\_saldo()) # Output: 1500

* + **Creazione di un oggetto ContoBancario**:

conto = ContoBancario(1000)

Questa linea crea un nuovo oggetto conto della classe ContoBancario con un saldo iniziale di 1000.

* + **Deposito di un importo**:

conto.deposita(500)

Questa linea deposita 500 sul conto bancario. Il saldo diventa 1500.

* + **Ottenimento e stampa del saldo**:

print(conto.ottieni\_saldo()) # Output: 1500

Questa linea chiama il metodo ottieni\_saldo per ottenere il saldo corrente del conto e lo stampa. Il saldo è 1500.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Definizione della classe ContoBancario**:
   * La classe ContoBancario viene definita con un costruttore \_\_init\_\_ per inizializzare il saldo e due metodi: deposita per depositare importi e ottieni\_saldo per ottenere il saldo corrente.
2. **Creazione dell'oggetto conto**:
   * Viene creato un oggetto conto della classe ContoBancario con un saldo iniziale di 1000.
3. **Deposito di un importo**:
   * Viene depositato un importo di 500 sul conto, aggiornando il saldo a 1500.
4. **Ottenimento e stampa del saldo**:
   * Il metodo ottieni\_saldo viene chiamato per ottenere il saldo corrente, che è 1500, e viene stampato sulla console.

### Output del codice

L'output sarà:

yaml

1500

### Riepilogo

* **Definizione della classe ContoBancario**: La classe ContoBancario ha un costruttore per inizializzare il saldo e due metodi: deposita e ottieni\_saldo.
* **Inizializzazione del saldo**: L'attributo \_\_saldo viene inizializzato nel costruttore e reso privato.
* **Deposito di un importo**: Il metodo deposita consente di aggiungere un importo al saldo se l'importo è positivo.
* **Ottenimento del saldo**: Il metodo ottieni\_saldo restituisce il saldo corrente.
* **Creazione e utilizzo di un oggetto ContoBancario**: Un oggetto conto viene creato, viene depositato un importo e il saldo viene stampato.

In questo esempio, abbiamo visto come creare una classe con attributi privati, come definire metodi per modificare e accedere a questi attributi e come utilizzare questi metodi per gestire un conto bancario.

*Esercizi di Pratica*

Per rinforzare la comprensione della programmazione orientata agli oggetti, puoi eseguire esercizi che ti permettano di creare classi con diverse caratteristiche, utilizzare l'ereditarietà per estendere le funzionalità di classi esistenti, e implementare metodi che dimostrano il polimorfismo. Ad esempio potreste creare la classe *automobile* e estenderla con la classe *coupe*

*Conclusione*

Questi concetti sono fondamentali per scrivere codice organizzato e scalabile in Python e sono ampiamente usati in molti progetti di software. Nel prossimo modulo, potremmo esplorare altri concetti avanzati come le eccezioni, la gestione dei file e l'interazione con database esterni.

Soluzione eserczio:

# Definizione della classe Automobile

class Automobile:

def \_\_init\_\_(self, marca, modello, anno, colore):

self.marca = marca # Attributo pubblico

self.modello = modello # Attributo pubblico

self.anno = anno # Attributo pubblico

self.\_\_colore = colore # Attributo privato

def descrizione(self):

"""

Restituisce una descrizione dell'automobile.

"""

return f"{self.marca} {self.modello}, anno {self.anno}, colore {self.\_\_colore}"

def cambia\_colore(self, nuovo\_colore):

"""

Cambia il colore dell'automobile.

"""

self.\_\_colore = nuovo\_colore

# Estensione della classe Automobile con la classe Coupé

class Coupe(Automobile):

def \_\_init\_\_(self, marca, modello, anno, colore, tipo\_tetto):

super().\_\_init\_\_(marca, modello, anno, colore) # Chiamata al costruttore della superclasse

self.tipo\_tetto = tipo\_tetto # Attributo pubblico specifico di Coupé

def descrizione(self):

"""

Restituisce una descrizione del coupé, includendo il tipo di tetto.

"""

return f"{super().descrizione()}, tipo tetto: {self.tipo\_tetto}"

# Esempio di utilizzo delle classi

auto = Automobile("Toyota", "Corolla", 2020, "Rosso")

print(auto.descrizione()) # Output: Toyota Corolla, anno 2020, colore Rosso

coupé = Coupe("BMW", "Serie 4", 2021, "Blu", "Tetto rigido")

print(coupé.descrizione()) # Output: BMW Serie 4, anno 2021, colore Blu, tipo tetto: Tetto rigido

# Cambiare il colore dell'automobile

auto.cambia\_colore("Verde")

print(auto.descrizione()) # Output: Toyota Corolla, anno 2020, colore Verde

# Cambiare il colore del coupé

coupé.cambia\_colore("Nero")

print(coupé.descrizione()) # Output: BMW Serie 4, anno 2021, colore Nero, tipo tetto: Tetto rigido

Miglior pratica:

# Definizione della classe Automobile

class Automobile:

def \_\_init\_\_(self, marca, modello, anno, colore):

self.\_\_marca = marca # Attributo privato

self.\_\_modello = modello # Attributo privato

self.\_\_anno = anno # Attributo privato

self.\_\_colore = colore # Attributo privato

def get\_marca(self):

return self.\_\_marca

def get\_modello(self):

return self.\_\_modello

def get\_anno(self):

return self.\_\_anno

def get\_colore(self):

return self.\_\_colore

def set\_colore(self, nuovo\_colore):

self.\_\_colore = nuovo\_colore

def descrizione(self):

"""

Restituisce una descrizione dell'automobile.

"""

return f"{self.\_\_marca} {self.\_\_modello}, anno {self.\_\_anno}, colore {self.\_\_colore}"

# Estensione della classe Automobile con la classe Coupé

class Coupe(Automobile):

def \_\_init\_\_(self, marca, modello, anno, colore, tipo\_tetto):

super().\_\_init\_\_(marca, modello, anno, colore) # Chiamata al costruttore della superclasse

self.\_\_tipo\_tetto = tipo\_tetto # Attributo privato specifico di Coupé

def get\_tipo\_tetto(self):

return self.\_\_tipo\_tetto

def set\_tipo\_tetto(self, tipo\_tetto):

self.\_\_tipo\_tetto = tipo\_tetto

def descrizione(self):

"""

Restituisce una descrizione del coupé, includendo il tipo di tetto.

"""

return f"{super().descrizione()}, tipo tetto: {self.\_\_tipo\_tetto}"

# Esempio di utilizzo delle classi

auto = Automobile("Toyota", "Corolla", 2020, "Rosso")

print(auto.descrizione()) # Output: Toyota Corolla, anno 2020, colore Rosso

coupé = Coupe("BMW", "Serie 4", 2021, "Blu", "Tetto rigido")

print(coupé.descrizione()) # Output: BMW Serie 4, anno 2021, colore Blu, tipo tetto: Tetto rigido

# Cambiare il colore dell'automobile

auto.set\_colore("Verde")

print(auto.descrizione()) # Output: Toyota Corolla, anno 2020, colore Verde

# Cambiare il colore del coupé

coupé.set\_colore("Nero")

print(coupé.descrizione()) # Output: BMW Serie 4, anno 2021, colore Nero, tipo tetto: Tetto rigido

# Cambiare il tipo di tetto del coupé

coupé.set\_tipo\_tetto("Tetto apribile")

print(coupé.descrizione()) # Output: BMW Serie 4, anno 2021, colore Nero, tipo tetto: Tetto apribile

### Spiegazione

1. **Attributi Privati**:
   * Tutti gli attributi sono dichiarati privati con il prefisso \_\_ (doppio underscore).
   * Questo protegge gli attributi da accessi diretti dall'esterno della classe.
2. **Metodi Getter**:
   * Vengono creati metodi pubblici per ottenere i valori degli attributi privati (get\_marca, get\_modello, get\_anno, get\_colore).
3. **Metodi Setter**:
   * Vengono creati metodi pubblici per modificare i valori degli attributi privati (set\_colore, set\_tipo\_tetto).
4. **Descrizione**:
   * Il metodo descrizione utilizza i getter per ottenere i valori degli attributi e restituire una descrizione completa dell'oggetto.

### Vantaggi

* **Incapsulamento**: Protegge gli attributi interni della classe da modifiche accidentali o non autorizzate.
* **Controllo**: Permette di aggiungere logica di validazione nei metodi setter, se necessario.
* **Flessibilità**: Consente di modificare l'implementazione interna senza influenzare il codice esterno che utilizza la classe.

In questo modo, si garantisce che l'accesso e la modifica degli attributi avvengano solo attraverso i metodi definiti, migliorando la sicurezza e la manutenibilità del codice.

Entrambi gli approcci hanno i loro vantaggi e possono essere utilizzati a seconda delle necessità del progetto e delle preferenze personali. Tuttavia, preferire l'incapsulamento con **attributi privati** (secondo esempio)offre una maggiore protezione dei dati e controllo su come gli attributi vengono manipolati. In altre parole è buona prativa dichiarare gli attributi PRIVATI ed accedervi tramite metodi PUBBLICI.

#### **Decoratori e Generatori**

**Decoratori** I decoratori sono funzioni che prendono un'altra funzione e ne estendono il comportamento senza modificarla direttamente.

def decoratore(f):

def wrapper():

print("Qualcosa prima della funzione")

f()

print("Qualcosa dopo la funzione")

return wrapper

@decoratore

def saluta():

print("Ciao!")

saluta()

### Spiegazione del codice

1. **Definizione del decoratore**:

def decoratore(f):

def wrapper():

print("Qualcosa prima della funzione")

f()

print("Qualcosa dopo la funzione")

return wrapper

* + **Decoratore decoratore**: Un decoratore è una funzione che prende un'altra funzione come argomento, la "avvolge" in un'altra funzione (il wrapper) e la restituisce. Questo permette di aggiungere funzionalità extra alla funzione originale.
  + **Funzione wrapper**: La funzione wrapper è definita all'interno del decoratore. Questa funzione stampa un messaggio prima e dopo aver chiamato la funzione f. La funzione f rappresenta la funzione originale che viene passata al decoratore.
  + **Restituzione del wrapper**: La funzione decoratore restituisce la funzione wrapper.

1. **Decorazione della funzione saluta**:

@decoratore

def saluta():

print("Ciao!")

* + **Sintassi @decoratore**: Questa sintassi applica il decoratore decoratore alla funzione saluta. È equivalente a scrivere:

def saluta():

print("Ciao!")

saluta = decoratore(saluta)

La funzione saluta viene passata al decoratore decoratore, che restituisce la funzione wrapper. Ora, ogni volta che chiamiamo saluta, in realtà viene chiamata la funzione wrapper.

1. **Chiamata della funzione saluta**:

saluta()

Quando chiamiamo saluta, viene eseguita la funzione wrapper, che include il comportamento aggiuntivo definito nel decoratore.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Definizione del decoratore**:
   * Viene definito il decoratore decoratore, che contiene la funzione wrapper.
2. **Decorazione della funzione saluta**:
   * La funzione saluta viene decorata con @decoratore, quindi saluta è ora una funzione wrapper.
3. **Chiamata della funzione saluta**:
   * Quando chiamiamo saluta(), viene eseguita la funzione wrapper.
   * La funzione wrapper stampa "Qualcosa prima della funzione".
   * La funzione wrapper chiama la funzione originale f, che stampa "Ciao!".
   * La funzione wrapper stampa "Qualcosa dopo la funzione".

### Output del codice

L'output sarà:

Qualcosa prima della funzione

Ciao!

Qualcosa dopo la funzione

### Riepilogo

* **Definizione del decoratore**: Un decoratore è una funzione che avvolge un'altra funzione, permettendo di aggiungere comportamenti extra.
* **Funzione wrapper**: La funzione wrapper definisce cosa succede prima e dopo la chiamata alla funzione originale.
* **Applicazione del decoratore**: Il decoratore viene applicato alla funzione saluta usando la sintassi @decoratore.
* **Chiamata della funzione decorata**: Quando la funzione decorata viene chiamata, viene eseguita la funzione wrapper, che include il comportamento aggiuntivo.

In questo esempio, abbiamo visto come definire e utilizzare un decoratore per aggiungere funzionalità extra a una funzione esistente in Python.

**Generatori** I generatori sono un modo semplice per creare iteratori utilizzando la parola chiave yield.

def conta\_fino\_a(n):

i = 1

while i <= n:

yield i

i += 1

# Usare il generatore

for numero in conta\_fino\_a(5):

print(numero)

### Spiegazione del codice

1. **Definizione della funzione conta\_fino\_a**:

def conta\_fino\_a(n):

i = 1

while i <= n:

yield i

i += 1

* + **Generatore conta\_fino\_a**: Un generatore è una funzione che contiene uno o più yield anziché return. Quando viene chiamata, restituisce un oggetto iteratore che può essere utilizzato per iterare sui valori generati.
  + **Inizializzazione**: La variabile i viene inizializzata a 1.
  + **Ciclo while**: Un ciclo while viene utilizzato per generare valori da 1 a n. Fino a quando i è minore o uguale a n, il ciclo continua.
  + **Yield**: L'istruzione yield restituisce il valore corrente di i e sospende l'esecuzione della funzione. La prossima volta che l'iteratore viene chiamato, l'esecuzione riprende subito dopo il yield.

1. **Uso del generatore**:

for numero in conta\_fino\_a(5):

print(numero)

* + **Ciclo for**: Un ciclo for viene utilizzato per iterare sui valori generati dalla funzione conta\_fino\_a(5).
  + **Stampa dei valori**: Ogni valore generato dalla funzione conta\_fino\_a viene assegnato alla variabile numero e stampato.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Definizione della funzione conta\_fino\_a**:
   * La funzione conta\_fino\_a viene definita con un ciclo while che genera valori da 1 a n.
2. **Chiamata della funzione generatrice**:
   * La funzione conta\_fino\_a(5) viene chiamata, restituendo un generatore che produce i valori 1, 2, 3, 4 e 5.
3. **Iterazione sui valori generati**:
   * Il ciclo for itera sui valori generati da conta\_fino\_a(5).
   * Per ogni valore generato, il valore viene stampato.

### Output del codice

L'output sarà:

1

2

3

4

5

### Riepilogo

* **Definizione del generatore**: La funzione conta\_fino\_a è un generatore che utilizza yield per produrre valori da 1 a n.
* **Inizializzazione e ciclo while**: La variabile i viene inizializzata a 1 e incrementata in ogni iterazione del ciclo while fino a quando i è minore o uguale a n.
* **Uso del generatore**: Un ciclo for viene utilizzato per iterare sui valori generati da conta\_fino\_a(5), stampando ogni valore.

In questo esempio, abbiamo visto come definire e utilizzare un generatore in Python per produrre una sequenza di valori. I generatori sono utili per lavorare con sequenze di dati grandi o infinite senza dover caricare tutti i dati in memoria contemporaneamente.

Inizio modulo

Fine modulo

Questa sezione copre concetti avanzati di Python, fornendo una solida base per scrivere codice più complesso e gestire scenari avanzati di programmazione.

***Threading in Python***

Cos'è il Threading?

Il threading permette a un programma di eseguire più operazioni contemporaneamente. È particolarmente utile per le operazioni I/O-bound o per creare interfacce utente che rimangono responsive mentre eseguono altri compiti in background.

#### Esempio: Uso di Thread

import threading

def stampa\_numeri():

for i in range(1, 6):

print(i)

# Creazione di due thread che eseguono la stessa funzione

t1 = threading.Thread(target=stampa\_numeri)

t2 = threading.Thread(target=stampa\_numeri)

t1.start()

t2.start()

t1.join()

t2.join()

### Spiegazione del codice

#### 1. Definizione della funzione stampa\_numeri

def stampa\_numeri():

for i in range(1, 6):

print(i)

* **Funzione stampa\_numeri:** Questa funzione stampa i numeri da 1 a 5.
* **Ciclo for:** Un ciclo for viene utilizzato per iterare attraverso i numeri da 1 a 5 e stamparli.

#### 2. Creazione e avvio dei thread

# Creazione di due thread che eseguono la stessa funzione

t1 = threading.Thread(target=stampa\_numeri)

t2 = threading.Thread(target=stampa\_numeri)

t1.start()

t2.start()

t1.join()

t2.join()

* **Creazione dei thread:** Vengono creati due oggetti Thread, t1 e t2, specificando la funzione stampa\_numeri come target.
* **Avvio dei thread:** I thread t1 e t2 vengono avviati con il metodo start().
* **Metodo join():** Viene utilizzato per aspettare che i thread t1 e t2 terminino l'esecuzione prima di continuare con il resto del programma.

### Esecuzione del codice

Vediamo come il codice si esegue passo dopo passo:

1. **Definizione della funzione stampa\_numeri:**
   * La funzione stampa\_numeri viene definita con un ciclo for che stampa i numeri da 1 a 5.
2. **Creazione e avvio dei thread:**
   * I thread t1 e t2 vengono creati e avviati, eseguendo contemporaneamente la funzione stampa\_numeri.
3. **Attesa del completamento dei thread:**
   * Il programma aspetta che entrambi i thread terminino l'esecuzione prima di procedere.

### Output del codice

L'output potrebbe essere:

1

1

2

2

3

3

4

4

5

5

Nota: L'ordine di esecuzione dei thread può variare, quindi l'output potrebbe non essere sempre lo stesso.

### Riepilogo

* **Definizione della funzione stampa\_numeri:** La funzione stampa i numeri da 1 a 5.
* **Creazione e avvio dei thread:** Due thread vengono creati e avviati per eseguire la stessa funzione.
* **Attesa del completamento dei thread:** Il metodo join() viene utilizzato per aspettare che i thread terminino l'esecuzione.

In questo esempio, abbiamo visto come utilizzare i thread in Python per eseguire operazioni simultanee. Il threading è utile per migliorare la reattività delle applicazioni e gestire operazioni I/O-bound in modo più efficiente.

***Esempio di programmi avanzati***

* *Interfaccia Utente e Visualizzazione dei Dati*

Da qui in poi ci divertiamo:

**Visualizzazione dei dati con Matplotlib** Matplotlib è una libreria per creare grafici e visualizzazioni dei dati.

* **Esempio di utilizzo di Matplotlib**

**Link:** <https://matplotlib.org/>

import matplotlib.pyplot as plt

# Dati da visualizzare

x = [1, 2, 3, 4, 5]

y = [2, 3, 5, 7, 11]

# Creare un grafico

plt.plot(x, y, marker='o')

plt.title('Grafico esempio')

plt.xlabel('X')

plt.ylabel('Y')

plt.show()

### Spiegazione del codice

1. **Importazione del modulo Matplotlib**:

import matplotlib.pyplot as plt

Questa linea di codice importa il modulo matplotlib.pyplot e lo assegna all'alias plt. Matplotlib è una libreria Python utilizzata per creare visualizzazioni di dati come grafici e diagrammi.

1. **Definizione dei dati**:

x = [1, 2, 3, 4, 5]

y = [2, 3, 5, 7, 11]

* + **Liste x e y**: Vengono definite due liste di dati: x rappresenta i valori sull'asse delle ascisse e y rappresenta i valori sull'asse delle ordinate.

1. **Creazione del grafico**:

plt.plot(x, y, marker='o')

plt.title('Grafico esempio')

plt.xlabel('X')

plt.ylabel('Y')

plt.show()

* + **plt.plot(x, y, marker='o')**: Questa funzione crea un grafico a linee con i dati forniti. Gli argomenti x e y specificano i punti da tracciare, e l'argomento marker='o' aggiunge un cerchio ('o') a ciascun punto dati.
  + **plt.title('Grafico esempio')**: Questa funzione imposta il titolo del grafico come 'Grafico esempio'.
  + **plt.xlabel('X')**: Questa funzione imposta l'etichetta dell'asse delle ascisse come 'X'.
  + **plt.ylabel('Y')**: Questa funzione imposta l'etichetta dell'asse delle ordinate come 'Y'.
  + **plt.show()**: Questa funzione visualizza il grafico. È importante chiamare plt.show() per rendere il grafico visibile.

### Esecuzione del codice

Quando esegui questo script, verrà visualizzata una finestra con un grafico a linee che mostra i punti definiti dalle liste x e y. Ogni punto è contrassegnato da un cerchio ('o'). Il grafico avrà un titolo e etichette sugli assi.

### Output del codice

L'output sarà un grafico a linee con i punti dati specificati:

* **Asse X**: [1, 2, 3, 4, 5]
* **Asse Y**: [2, 3, 5, 7, 11]
* **Titolo del grafico**: "Grafico esempio"
* **Etichetta asse X**: "X"
* **Etichetta asse Y**: "Y"

### Riepilogo

* **Importazione del modulo Matplotlib**: Il modulo matplotlib.pyplot viene importato per creare e visualizzare grafici.
* **Definizione dei dati**: Vengono definite due liste di dati per gli assi X e Y.
* **Creazione del grafico**: Un grafico a linee viene creato con i dati forniti e personalizzato con un titolo e etichette sugli assi.
* **Visualizzazione del grafico**: Il grafico viene visualizzato chiamando plt.show().

In questo esempio, abbiamo visto come utilizzare Matplotlib per creare e visualizzare un grafico a linee con Python. Matplotlib è una libreria potente e flessibile per la visualizzazione dei dati, che offre molte opzioni per personalizzare e creare grafici complessi.

***Introduzione alla Programmazione di Interfacce Grafiche con Tkinter***

Cosa è Tkinter?

Tkinter è il modulo standard di Python per la creazione di interfacce utente grafiche (GUI). È semplice da usare e integrato con l'ambiente standard Python.

*NOTA: da ora in avanti dovreste aver appreso un’ottima padronanza di python, di conseguenza le spiegazioni del codice saranno meno prolisse*

Creazione di una Finestra di Base

import tkinter as tk  
  
# Crea la finestra principale  
root = tk.Tk()  
  
# Imposta il titolo della finestra  
root.title("Finestra di Esempio")  
  
# Avvia il ciclo principale dell'evento  
root.mainloop()

Aggiungere Widget

Tkinter offre widget come pulsanti, etichette e caselle di testo per costruire GUI.

Esempio: Aggiunta di un Pulsante

import tkinter as tk  
  
def saluta():  
 print("Ciao mondo!")  
  
root = tk.Tk()  
root.title("Pulsante")  
  
# Crea un pulsante e lo aggiunge alla finestra  
pulsante = tk.Button(root, text="Clicca qui", command=saluta)  
pulsante.pack()  
  
root.mainloop()

Spiegazione del codice:

**Importazione della libreria principale**

*import tkinter as tk*

Questa riga importa la libreria tkinter e la abbrevia con l'alias tk; tk è l’abbreviazione convenzionale, ma avreste potuto abbreviarlo ad esmepio tkk oppure pippo, non avrebbe fatto differenza. Questo permette di accedere a tutte le funzionalità di tkinter utilizzando tk.

Nota: è importante cercare di rispettare gli standard e le convenzioni in modo da manutenere il codice e nel caso un vostro collega ci debba lavorare.

**Creazione della finestra principale**

*root = tk.Tk()* #nel caso l’avreste abbreviato pippo sarebbe stato pippo.Tk()

Questa riga crea la finestra principale dell'applicazione GUI e la memorizza nella variabile root. La finestra principale è il contenitore di base per tutti gli altri widget (come pulsanti, etichette, caselle di testo, ecc.).

**Impostazione del titolo della finestra**

*root.title("Finestra di Esempio")*

Questa riga imposta il titolo della finestra principale. Nel nostro caso su "Finestra di Esempio". Questo titolo verrà visualizzato nella barra del titolo della finestra.

**Impostazione di un etichetta**

*label = tk.Label(root, text="Ciao, mondo!") label.pack()*

Un'etichetta (Label) visualizzerà il testo "Ciao, mondo!".

**tk.Label**: Questo è il costruttore della classe Label di tkinter. Un'etichetta è un widget semplice che mostra del testo o un'immagine.

**root**: Questo è il contenitore padre del widget Label. In questo caso, root è la finestra principale creata precedentemente.

**text="Ciao, mondo!"**: Questo parametro specifica il testo che verrà visualizzato nell'etichetta. In questo caso, il testo è "Ciao, mondo!".

**label**: Questa è la variabile che memorizza l'oggetto Label appena creato.

**label.pack()** dispone i widget all'interno della finestra. Il ciclo principale continua a mantenere la finestra aperta e a gestire gli eventi fino a quando non viene chiusa.

**Inserimento di un pulsante**

*button = tk.Button(root, text="Cliccami", command=root.destroy) button.pack()*

Un pulsante (Button) che, quando cliccato, chiude la finestra (root.destroy).

**tk.Button**: Questo è il costruttore della classe Button di tkinter. Un pulsante è un widget interattivo che può essere cliccato dall'utente per eseguire un'azione.

**root**: Questo è il contenitore padre del widget Button. In questo caso, root è la finestra principale creata precedentemente.

**text="Cliccami"**: Questo parametro specifica il testo che verrà visualizzato sul pulsante. In questo caso, il testo è "Cliccami".

**command=root.**destroy: Questo parametro specifica la funzione che verrà chiamata quando il pulsante viene cliccato. In questo caso, root.destroy è una funzione che chiude la finestra principale, quindi cliccando il pulsante la finestra si chiuderà.

**button:** Questa è la variabile che memorizza l'oggetto Button appena creato.

**button.pack():** Questo metodo posiziona il widget Button nella finestra principale utilizzando il gestore di geometria pack. Il gestore pack organizza i widget in blocchi, uno dopo l'altro, nella finestra principale.

**Avvio del ciclo principale dell'evento**

*root.mainloop()*

Questa riga avvia il ciclo principale dell'evento. Il ciclo principale è responsabile di mantenere aperta la finestra e di rispondere agli eventi, come clic del mouse o pressioni di tasti. Finché il ciclo principale è in esecuzione, la finestra rimane aperta e l'applicazione continua a funzionare.

Esercizi di Pratica

Crea una GUI semplice con Tkinter che accetti l'input dell'utente e mostri una risposta in un'etichetta.

Scrivi uno script che gestisca diverse eccezioni, come divisione per zero e input non valido.

Conclusione

La comprensione di come gestire le eccezioni ti permette di scrivere programmi più sicuri e resistenti agli errori, mentre l'apprendimento di Tkinter apre la porta alla creazione di applicazioni desktop con interfacce utente grafiche. Nel prossimo modulo, potremmo esplorare altri argomenti avanzati come il threading o l'accesso a database.

Soluzione esempio 1: GUI Semplice con Tkinter

Questo script crea una finestra che permette all'utente di inserire del testo in una casella di input. Quando l'utente clicca su un pulsante, il testo inserito viene visualizzato in un'etichetta.

import tkinter as tk  
  
def mostra\_risposta():  
 # Recupera il testo dall'entry widget e aggiorna l'etichetta  
 risposta = entry.get()  
 label.config(text=f"Hai scritto: {risposta}")  
  
# Crea la finestra principale  
root = tk.Tk()  
root.title("Input dell'Utente")  
  
# Crea un widget di entry  
entry = tk.Entry(root, width=50)  
entry.pack(pady=20)  
  
# Crea un pulsante che quando cliccato eseguirà la funzione 'mostra\_risposta'  
button = tk.Button(root, text="Invia", command=mostra\_risposta)  
button.pack(pady=10)  
  
# Crea un'etichetta per mostrare la risposta  
label = tk.Label(root, text="")  
label.pack(pady=20)  
  
# Avvia il ciclo principale dell'evento  
root.mainloop()

***Accesso a Database con Python***

Uso di SQLite

SQLite è un database relazionale leggero che è incorporato in Python, ideale per applicazioni più piccole e prototipi.

Esempio: Creazione e Interrogazione di un Database

import sqlite3  
  
# Connessione al database SQLite  
conn = sqlite3.connect('esempio.db')  
c = conn.cursor()  
  
# Creazione di una tabella  
c.execute('''CREATE TABLE if not exists inventario (id INTEGER PRIMARY KEY, nome TEXT, quantità INTEGER)''')  
  
# Inserimento di dati  
c.execute("INSERT INTO inventario (nome, quantità) VALUES ('penna', 100)")  
  
# Interrogazione del database  
c.execute('SELECT \* FROM inventario')  
print(c.fetchall())  
  
conn.commit()

conn.close()

Spiegazione del codice:

*import sqlite3*

sqlite3 è un modulo integrato in Python che fornisce una interfaccia per lavorare con i database SQLite.

*conn = sqlite3.connect('esempio.db')*

Questo comando stabilisce una connessione a un database SQLite chiamato esempio.db. Se il database non esiste, SQLite lo crea automaticamente. conn è l'oggetto connessione che userai per interagire con il database.

*c = conn.cursor()*

Un cursore (cursor) è un oggetto utilizzato per eseguire comandi SQL e recuperare i risultati delle query.

*c.execute('''CREATE TABLE if not exists inventario (id INTEGER PRIMARY KEY, nome TEXT, quantità INTEGER)''')*

Questo comando SQL crea una tabella chiamata inventario con tre colonne:

id, nome, quantità. IF NOT EXISTS assicura che la tabella venga creata solo se non esiste già.

*c.execute("INSERT INTO inventario (nome, quantità) VALUES ('penna', 100)")*

Questo comando inserisce un nuovo record nella tabella inventario con nome uguale a 'penna' e quantità uguale a 100.

*c.execute('SELECT \* FROM inventario')*

*print(c.fetchall())*

Questo comando esegue una query per selezionare tutti i record dalla tabella inventario.

c.fetchall() recupera tutti i risultati della query e li restituisce come una lista di tuple.

*print(c.fetchall())* stampa i risultati sulla console.

*conn.commit()* salva tutte le modifiche fatte al database durante la sessione. Se non chiami commit(), le modifiche non verranno salvate.

*conn.close()* chiude la connessione al database. È una buona pratica chiudere sempre la connessione quando hai finito di lavorare con il database.

***Programmazione di Rete in Python***

Socket Programming

La programmazione di socket consente a Python di comunicare su reti TCP/IP. È utile per lo sviluppo di applicazioni web, client/server e altre applicazioni di rete.

Esempio: Creazione di un Server Semplice tcp/ip (lato server)

Create un file e chiamatelo server\_tcp.py o come preferite

#servertcp  
import socket  
  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind(('localhost', 12345))

s.listen(5)

print('Server in ascolto...')  
  
while True:

clientsocket, address = s.accept()

print(f"Connessione da {address} è stata stabilita!")

clientsocket.send(bytes("Benvenuto al server!", "utf-8"))

clientsocket.close()

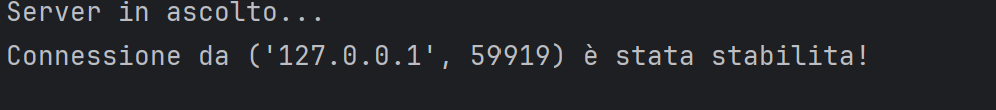
(Lato client) Create un altro file e chiamatelo client\_tcp.py o come preferite

#Clienttcp  
import socket  
  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

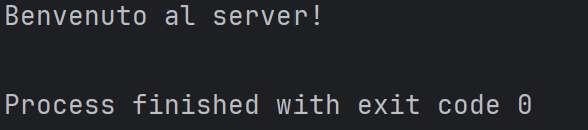
s.connect(('localhost', 12345))  
  
msg = s.recv(1024)

print(msg.decode("utf-8"))  
  
s.close()

Lanciate prima il file server\_tcp.py e successivamente il file client\_tcp.py, altrimenti causerà un errore. Di seguito vi mostro le screen del risultato



Output Server\_tcp.py



Output Client\_tcp.py

Spiegazione del codice server\_tcp.py:

*import socket*

Il modulo socket fornisce accesso alle interfacce di rete di basso livello. È necessario per creare e gestire le connessioni di rete

*s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)*

Questa linea crea un nuovo socket. socket.AF\_INET specifica che il socket utilizzerà l'indirizzo di famiglia IPv4, mentre socket.SOCK\_STREAM indica che il socket sarà di tipo TCP

*s.bind(('localhost', 12345))*

Questa linea associa il socket a un indirizzo e una porta specifici. In questo caso, il server sarà in ascolto su localhost (indirizzo IP 127.0.0.1) e sulla porta 12345 (potevate sceglierne anche un’ altra)

*s.listen(5)*

Questa linea mette il socket in modalità di ascolto, consentendo al server di accettare connessioni in entrata. Il parametro 5 specifica la dimensione della coda di backlog, cioè il numero massimo di connessioni non accettate che il sistema può gestire prima di rifiutare nuove connessioni.

*print('Server in ascolto...')*

Stampa un messaggio informativo che indica che il server è in ascolto delle connessioni in entrata.

*while True:*

*clientsocket, address = s.accept()*

*print(f"Connessione da {address} è stata stabilita!")*

*clientsocket.send(bytes("Benvenuto al server!", "utf-8"))*

*clientsocket.close()*

Questo è un ciclo perpetuo che mantiene il server in esecuzione in modo da accettare connessioni in entrata:

* s.accept() blocca l'esecuzione fino a quando non viene ricevuta una connessione. Quando una connessione viene accettata, restituisce un nuovo socket clientsocket per comunicare con il client e l'indirizzo del client address.
* print(f"Connessione da {address} è stata stabilita!") stampa un messaggio che indica che una connessione da un determinato indirizzo è stata stabilita.
* clientsocket.send(bytes("Benvenuto al server!", "utf-8")) invia un messaggio di benvenuto al client connesso. La funzione bytes() converte la stringa in una sequenza di byte utilizzando la codifica UTF-8.
* clientsocket.close() chiude il socket client, terminando la connessione.

Spiegazione del codice client\_tcp.py:

Le prime inee sono uguali a quelle del prijmmo script.

*s.recv(1024):*

Riceve fino a 1024 byte di dati dal server. La dimensione del buffer (1024 byte) specifica la quantità massima di dati da leggere alla volta.

*print(msg.decode("utf-8"))*

Decodifica i byte ricevuti in una stringa utilizzando la codifica UTF-8.

*print(...):*

Stampa il messaggio decodificato sulla console.

*s.close():*

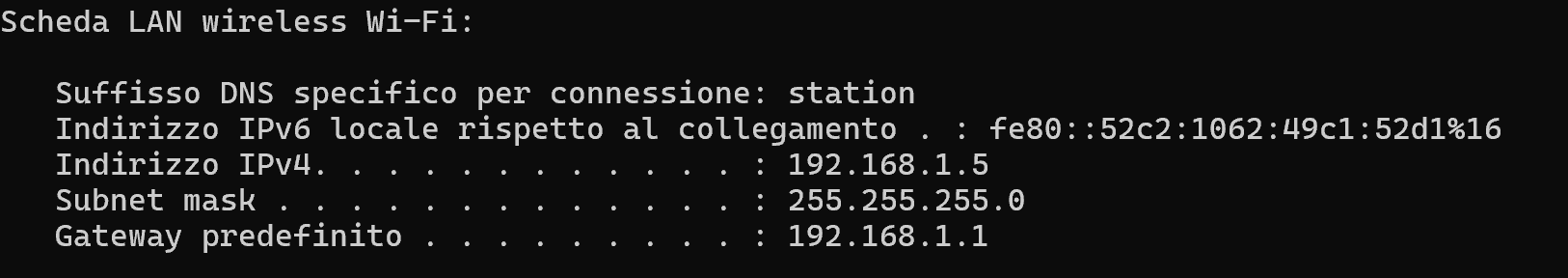
Chiude il socket, terminando la connessione con il server.

Per realizzare un server http invece dovresete creare un scrupt leggermente differente:

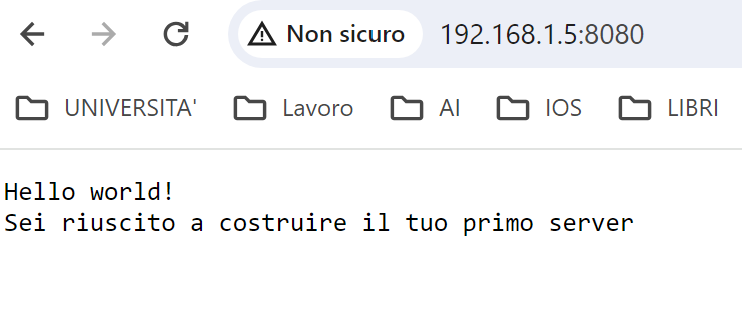
import socket  
  
# Creazione di un socket TCP  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
s.bind(('localhost', 8080))  
#per connettersi allo script da un altro dispositivo, ad esempio un cellulare  
#dovreste sostituire localhost, con l'indirizzo ip locale del tuo computer  
#apri il terminale winddows o mac/linux e digita ipconfig (su win) e ifconfig (su unix)  
#dovresti vedere qualcosa del genere 192.168.1.5 (ip locale)  
s.listen(5)  
print('Server HTTP in ascolto su porta 8080...')  
  
while True:  
 clientsocket, address = s.accept()  
 print(f"Connessione da {address} è stata stabilita!")  
 # Risposta HTTP  
 response = (  
 'HTTP/1.1 200 OK\r\n'  
 'Content-Type: text/plain\r\n'  
 'Content-Length: 57\r\n' # Lunghezza del contenuto del messaggio  
 'Connection: close\r\n'  
 '\r\n'  
 'Hello world!\nSei riuscito a costruire il tuo primo server in Python!'  
 )  
 clientsocket.sendall(response.encode('utf-8'))  
 clientsocket.close()  
  
#potresti modificare lo script per modificare il messaggio di benvenuto, mostrare un immagine o per scambiare file

Per vederlo in azione vi basterà aprire il vostro browser e digitare nella barra degli indirizzi 127.0.0.1:8080.

Se ad esempio voleste provare a connettervi al server da un altro dispositivo (da un cellulare), dovreste assicurarvi di essere connessi sulla stessa rete. Inoltre dovrete modificare ‘localhost’ con l’indirizzo locale del vostro pc su cui avvierete lo script; per fare ciò aprite il terminale Windows o Linux o mac e dicitare ipconfig (su windows) e ifconfig (sui sitemi unix) come nella slide mostrata:



E questo è l’ouput del programma:



Spiegazione del codice del server http:

*s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)*

socket.AF\_INET specifica che il socket utilizzerà IPv4. Socket.SOCK\_STREAM indica che il socket utilizzerà il protocollo TCP, che è affidabile e orientato alla connessione.

*s.bind(('localhost', 8080))*

Associa il socket all'indirizzo localhost (127.0.0.1) e alla porta 8080. Puoi sostituire localhost con l'indirizzo IP del tuo computer per consentire connessioni da altri dispositivi nella stessa rete.

*s.listen(5)*

Mette il socket in modalità di ascolto, pronto ad accettare fino a 5 connessioni in coda.

*clientsocket, address = s.accept()*

Accetta una connessione in arrivo e restituisce un nuovo socket (clientsocket) e l'indirizzo del client (address).

*response = (*

*'HTTP/1.1 200 OK\r\n'*

*'Content-Type: text/plain\r\n'*

*'Content-Length: 57\r\n'*

*'Connection: close\r\n'*

*'\r\n'*

*'Hello world!\nSei riuscito a costruire il tuo primo server in Python!'*

*)*

*clientsocket.sendall(response.encode('utf-8'))*

Prepara una risposta HTTP con le intestazioni e il corpo del messaggio. Invia la risposta al client.

*clientsocket.close()*

Chiude la connessione con il client.

**Connessione dal Client HTTP**

Per connettersi a questo server dal browser o da un altro dispositivo (ad esempio, un cellulare):

1. **Trova l'indirizzo IP locale del tuo computer:**
   * Su Windows, esegui ipconfig.
   * Su Mac/Linux, esegui ifconfig.
2. **Accedi all'indirizzo nel browser:**
   * Digita http://<IP\_DEL\_TUO\_COMPUTER>:8080 nel browser del dispositivo.

Questo codice implementa un semplice server HTTP che ascolta le connessioni sulla porta 8080. Quando un client si connette, il server risponde con un messaggio di testo e chiude la connessione. Per accedere al server da un dispositivo diverso (ad esempio un cellulare), è necessario sostituire *localhost* con l'indirizzo IP locale del computer che esegue il server.

Esercizi di Pratica

Scrivi uno script che utilizza threading per eseguire più operazioni di download in parallelo.

Crea una piccola applicazione che utilizza SQLite per gestire un inventario di prodotti.

Sviluppa un client e un server che comunicano tra loro utilizzando i socket.

Da qui in poi dovreste essere in grado di procedere da soli. Gli script mostrati nel capitolo sono un’ottima base per svolgere questi esercizi, dovete solo modificarli in parte.

Conclusione

Questi argomenti avanzati aprono molte possibilità per lo sviluppo di applicazioni Python potenti e efficienti. Il threading aiuta a gestire compiti paralleli, mentre l'accesso a database e la programmazione di rete sono essenziali per lo sviluppo di applicazioni robuste e interattive. Nel prossimo modulo, possiamo esplorare framework web come Django o Flask per costruire applicazioni web complete.

***Sviluppo Web con Python***

**Introduzione a Django**

Django è un framework web ad alto livello che incoraggia lo sviluppo rapido e pulito, con un design che promuove la riutilizzabilità e la manutenibilità del codice. Django utilizza il pattern di progettazione "model-view-template" (MVT).

Caratteristiche Principali di Django:

ORM (Object-Relational Mapping): Semplifica la manipolazione dei dati della database usando classi Python invece di SQL diretto.

Sistema di autenticazione integrato: Supporta l'autenticazione degli utenti, sessioni, permessi e più.

Sistema di template: Permette di definire l'HTML in modo dinamico con i dati provenienti dal server.

Admin interface: Un'interfaccia pronta per gestire i contenuti del sito.

*Esempio*: Creazione di un Progetto Django

Per iniziare con Django, dovresti installare il pacchetto django tramite pip, creare un progetto e avviare il server:

*bash*

pip install django

django-admin startproject mysite

cd mysite

python manage.py runserver

**Introduzione a Flask**

Flask è un micro-framework per Python basato su Werkzeug e Jinja2. È leggero e modulare, rendendolo molto adattabile a vari progetti, dall'API web semplici a complesse applicazioni web.

Caratteristiche Principali di Flask:

Leggerezza: Minimalismo con estendibilità.

Flessibilità: Usa quello che ti serve, estendilo come vuoi.

Sistema di template Jinja2: Potente e facile da usare per generare output HTML da dati.

*Esempio*: Creazione di un'applicazione Flask

Per creare una semplice applicazione web con Flask, prima installa Flask, poi crea un file Python:

*bash*

pip install flask

Ecco un esempio di codice per un'app Flask:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return "Ciao Mondo!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

Esercizi di Pratica:

Crea un'applicazione Django che gestisce un blog con la possibilità di aggiungere, modificare ed eliminare post.

Sviluppa un'applicazione Flask che accetta input dall'utente tramite un form e mostra i risultati in una nuova pagina.

Conclusione

Lo sviluppo web con Django e Flask offre potenti strumenti per creare applicazioni web interattive e dinamiche. Django è eccellente per progetti più grandi e strutturati, mentre Flask offre flessibilità e leggerezza per progetti più piccoli o personalizzati. Nel prossimo modulo, possiamo esplorare come integrare database avanzati, utilizzare API esterne, o ottimizzare le prestazioni del web server.

Crea un'applicazione Django che gestisce un blog con la possibilità di aggiungere, modificare ed eliminare post.

Sviluppa un'applicazione Flask che accetta input dall'utente tramite un form e mostra i risultati in una nuova pagina.

Speigazione esercizi svolti. Esercizio 1:

Applicazione Django per Gestire un Blog

Per creare un'applicazione blog con Django, segui questi passaggi:

Installazione e Configurazione di Django:

(da terminale)

pip install django

django-admin startproject myblog\_example

cd myblog\_example

python manage.py startapp blog

Fatto ciò navigate da esplora risorse fino a: C:\Users\lofte\PycharmProjects\pythonProject1\myblog\_example

Dovreste vedere qualcosa del genere:

Immagine che contiene testo, schermata, software, numero

Descrizione generata automaticamente

Aprite la cartella myblog\_example (quella sotto a blog) e tra i file dovreste avere setting.py. Apritelo con pycharme o il l’editor che utilizzate ed aggiungete la riga blog

# myblog\_example/settings.py  
  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'blog', # Aggiungi questa linea  
]

Aprite e modificate il file urls.py nella cartella myblog\_example per includere gli URL dell'app blog.

# myblog\_example/urls.py  
  
from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('', include('blog.urls')), # Assicurati che questa linea sia presente  
]

Recatevi all’interno della cartella blog. Se non esiste già, create e configurate urls.py nella cartella blog con le seguenti configurazioni.

# blog/urls.py  
  
from django.urls import path  
from . import views  
  
urlpatterns = [  
 path('', views.post\_list, name='post\_list'),  
 path('post/<int:pk>/', views.post\_detail, name='post\_detail'),  
 path('post/new/', views.post\_new, name='post\_new'),  
 path('post/<int:pk>/edit/', views.post\_edit, name='post\_edit'),  
 path('post/<int:pk>/delete/', views.post\_delete, name='post\_delete'),  
]

Controllate che views.py nella tua app blog contenga le viste appropriate. Se non le contenesse modificate il file come segue:

# blog/views.py  
  
from django.shortcuts import render, redirect, get\_object\_or\_404  
from .models import Post  
from .forms import PostForm  
  
def post\_list(request):  
 posts = Post.objects.all()  
 return render(request, 'blog/post\_list.html', {'posts': posts})  
  
def post\_detail(request, pk):  
 post = get\_object\_or\_404(Post, pk=pk)  
 return render(request, 'blog/post\_detail.html', {'post': post})  
  
def post\_new(request):  
 if request.method == "POST":  
 form = PostForm(request.POST)  
 if form.is\_valid():  
 post = form.save(commit=False)  
 post.save()  
 return redirect('post\_detail', pk=post.pk)  
 else:  
 form = PostForm()  
 return render(request, 'blog/post\_edit.html', {'form': form})  
  
def post\_edit(request, pk):  
 post = get\_object\_or\_404(Post, pk=pk)  
 if request.method == "POST":  
 form = PostForm(request.POST, instance=post)  
 if form.is\_valid():  
 post = form.save(commit=False)  
 post.save()  
 return redirect('post\_detail', pk=post.pk)  
 else:  
 form = PostForm(instance=post)  
 return render(request, 'blog/post\_edit.html', {'form': form})  
  
def post\_delete(request, pk):  
 post = get\_object\_or\_404(Post, pk=pk)  
 post.delete()  
 return redirect('post\_list')

In esplora risorse create la seguente struttura di cartelle (all’interno della cartella blog): templates/blog/

E create i seguenti file html con il vostro editor;assicuratevi che vengano inseriti all’interno di template/blog (se venissero creati in un'altra directory tagliateli ed incollateli lì):

**-post\_list.html**:

<h1>Blog Posts</h1>  
{% for post in posts %}  
 <h2><a href="{% url 'post\_detail' post.pk %}">{{ post.title }}</a></h2>  
 <p>{{ post.content|truncatewords:30 }}</p>  
{% endfor %}  
<a href="{% url 'post\_new' %}">New Post</a>

**-post\_detail.html**:

<h1>{{ post.title }}</h1>  
<p>{{ post.content }}</p>  
<a href="{% url 'post\_edit' post.pk %}">Edit</a>  
<form method="POST" action="{% url 'post\_delete' post.pk %}">  
 {% csrf\_token %}  
 <button type="submit">Delete</button>  
</form>  
<a href="{% url 'post\_list' %}">Back to List</a>

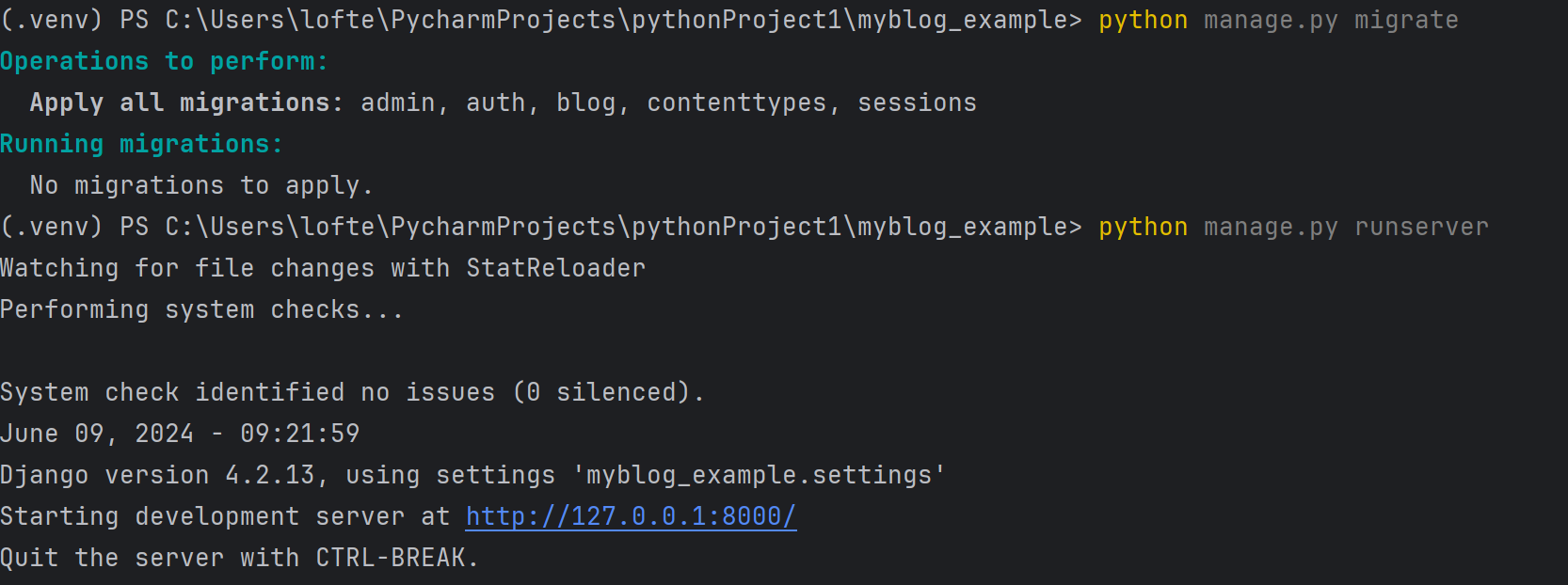
**-post\_edit.html**:

<h1>Edit Post</h1>  
<form method="POST">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Save</button>  
</form>  
<a href="{% url 'post\_list' %}">Back to List</a>

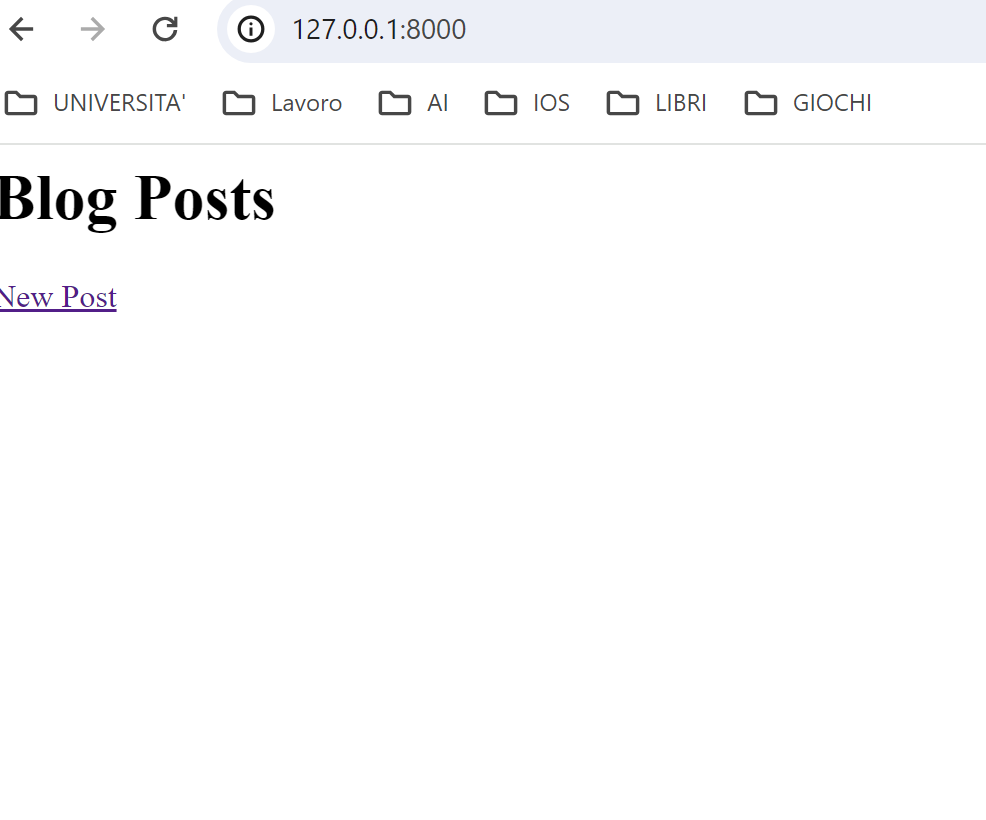
Per chi di voi conosce html sarà una passeggiata

**Recatevi da terminale in** path/to/your/project/myblog\_example e digitate i seguenti comandi per eseguira la migrazione del progetto*: python manage.py makemigrations python manage.py migrate*

Ora date il comando: *python manage.py runserver*

**

Ed infine cliccando su 127.0.01:8000 dovreste vedere il vostro blog in esecuzione:

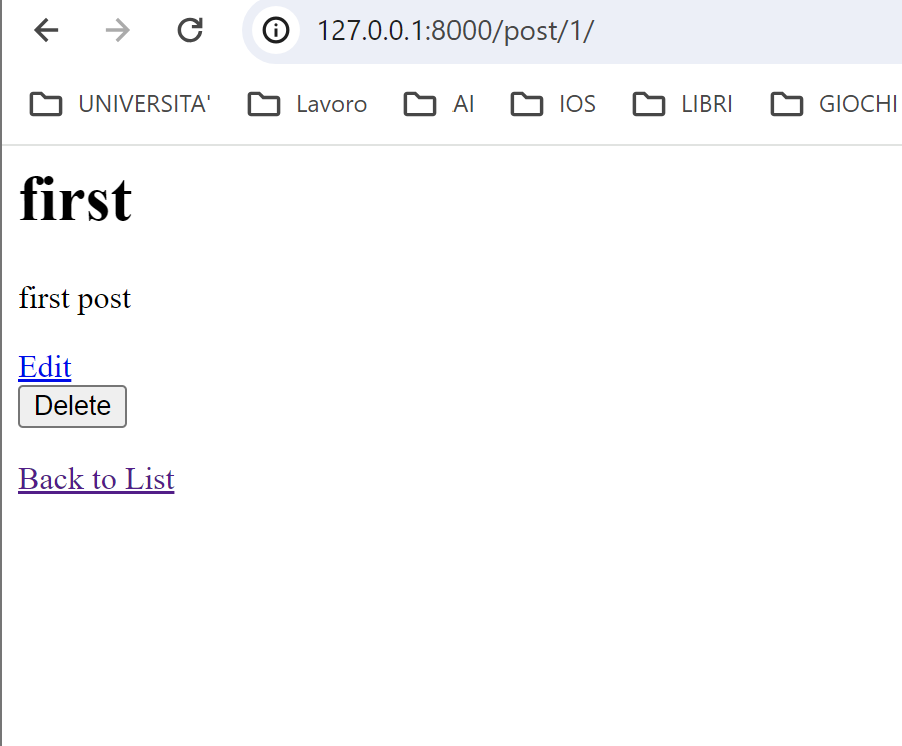


Pagina per scrivere un nuovo post:

Immagine che contiene testo, schermata, schermo, Carattere

Descrizione generata automaticamente

Pagina che elenca i post



Scrittura del primo post



Spiegazione Esercizio 2:

create uno script python e nominatelo ad esempio app.py:

from flask import Flask, render\_template, request, redirect, url\_for  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/', methods=['GET', 'POST'])  
def home():  
 if request.method == 'POST':  
 user\_input = request.form['user\_input']  
 return redirect(url\_for('result', input=user\_input))  
 return render\_template('home.html')  
  
@app.route('/result/<input>')  
def result(input):  
 return render\_template('result.html', input=input)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

Importazione dei moduli necessari:

**Flask**: il framework principale per creare l'applicazione web.

**render\_template**: una funzione per rendere i template HTML.

**request**: un oggetto che contiene i dati della richiesta HTTP.

**redirect**: una funzione per reindirizzare l'utente a un'altra pagina.

**url\_for**: una funzione per generare URL dinamici per le route definite nell'app.

*app = Flask(\_\_name\_\_)*

Crea un'istanza dell'applicazione Flask. Il parametro \_\_name\_\_ dice a Flask dove trovare le risorse come i template e i file statici.

*@app.route('/', methods=['GET', 'POST'])*

*def home():*

*if request.method == 'POST':*

*user\_input = request.form['user\_input']*

*return redirect(url\_for('result', input=user\_input))*

*return render\_template('home.html')*

**@app.route('/', methods=['GET', 'POST'])**: questo decoratore definisce una route per la homepage che accetta sia richieste GET che POST.

**home()**: la funzione che gestisce le richieste alla homepage.

* **request.method == 'POST'**: controlla se la richiesta è un POST (cioè se l'utente ha inviato il modulo).
* **request.form['user\_input']**: ottiene l'input dell'utente dal modulo HTML.
* **redirect(url\_for('result', input=user\_input))**: reindirizza l'utente alla pagina dei risultati, passando l'input dell'utente come parte dell'URL.
* **render\_template('home.html')**: rende il template HTML home.html per le richieste GET.

*@app.route('/result/<input>')*

*def result(input):*

*return render\_template('result.html', input=input)*

**@app.route('/result/<input>')**: questo decoratore definisce una route per la pagina dei risultati che accetta un parametro dinamico input.

**result(input)**: la funzione che gestisce le richieste alla pagina dei risultati.

* **render\_template('result.html', input=input)**: rende il template HTML result.html passando l'input dell'utente come variabile.

*if \_\_name\_\_ == '\_\_main\_\_':*

*app.run(debug=True)*

*Avvio l’applicazione.*

**if name == 'main'**: questo blocco garantisce che l'applicazione venga eseguita solo se il file viene eseguito direttamente (non importato come modulo).

**app.run(debug=True)**: avvia il server di sviluppo Flask in modalità debug. La modalità debug consente il riavvio automatico del server quando vengono apportate modifiche al codice e fornisce un'interfaccia di debug interattiva in caso di errori.

Ora create nella directory del porgetto app.py una cartella chiamata templates. Al suo interno create due file html:

home.html

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Home</title>  
</head>  
<body>  
 <h1>Enter something</h1>  
 <form method="POST">  
 <input type="text" name="user\_input">  
 <button type="submit">Submit</button>  
 </form>  
</body>  
</html>

e result.html

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Result</title>  
</head>  
<body>  
 <h1>You entered: {{ input }}</h1>  
 <a href="{{ url\_for('home') }}">Go back</a>  
</body>  
</html>

Ora avviate lo script app.py e cliccate su <http://127.0.0.1:5000> e dovreste vedere i seguenti screenshot:



Questo è l’output



In questo ultimo capitolo vi mostrerò alcuni script da cui potrete prendere spunto per lo sviluppo di applicazioni di vario tipo. Non mi soffermerò troppo nelle spiegazioni anche perché dovreste aver acquisito una buona familiarità con il linguaggio.

Ora vi mostro come creare una semplice calcolatrice che mantiene tutte le operazioni svolte. Successivamente andremo a creare l’applicazione desktop dal codice python. Si, Python offre una libreria che fa proprio questo: “*PyInstaller*”

Vi mostro il codice dell’app calc.py:

import tkinter as tk # Importa la libreria tkinter e la associa al nome tk  
  
def on\_click(event=None):  
 global expression # Dichiara l'uso della variabile globale 'expression'  
 text = event.widget.cget("text") # Ottiene il testo del widget che ha generato l'evento  
 if text == "=": # Se il testo è "=", significa che si vuole valutare l'espressione  
 try:  
 # Sostituisce 'x' con '\*' e '%' con '/100\*' per preparare l'espressione per la valutazione  
 expression\_to\_evaluate = expression.replace('x', '\*').replace('%', '/100\*')  
 result = str(eval(expression\_to\_evaluate)) # Valuta l'espressione e converte il risultato in stringa  
 entry.delete(0, tk.END) # Cancella il contenuto dell'entry  
 entry.insert(tk.END, result) # Inserisce il risultato nell'entry  
 save\_operation(f"{expression} = {result}") # Salva l'operazione nella cronologia  
 except Exception as e: # Gestisce eventuali errori durante la valutazione dell'espressione  
 entry.delete(0, tk.END) # Cancella il contenuto dell'entry  
 entry.insert(tk.END, "Error") # Inserisce "Error" nell'entry  
 expression = "" # Resetta l'espressione  
 elif text == "C": # Se il testo è "C", significa che si vuole cancellare l'espressione corrente  
 entry.delete(0, tk.END) # Cancella il contenuto dell'entry  
 expression = "" # Resetta l'espressione  
 elif text == "History": # Se il testo è "History", significa che si vuole mostrare la cronologia  
 show\_history() # Chiama la funzione per mostrare la cronologia  
 else: # Per tutti gli altri testi (numeri e operatori)  
 expression += str(text) # Aggiunge il testo all'espressione corrente  
 entry.delete(0, tk.END) # Cancella il contenuto dell'entry  
 entry.insert(tk.END, expression) # Inserisce l'espressione aggiornata nell'entry  
  
def save\_operation(operation):  
 # Apre il file "operations\_history.txt" in modalità append e scrive l'operazione  
 with open("operations\_history.txt", "a") as file:  
 file.write(operation + "\n")  
  
def load\_operations():  
 try:  
 # Prova a leggere tutte le linee dal file "operations\_history.txt"  
 with open("operations\_history.txt", "r") as file:  
 return file.readlines()  
 except FileNotFoundError:  
 return [] # Se il file non esiste, restituisce una lista vuota  
  
def show\_history():  
 history\_window = tk.Toplevel(root) # Crea una nuova finestra secondaria  
 history\_window.title("Cronologia Operazioni") # Imposta il titolo della finestra  
 history\_window.geometry("300x400") # Imposta le dimensioni della finestra  
 text\_widget = tk.Text(history\_window, font=('Arial', 12), wrap='word') # Crea un widget Text per mostrare la cronologia  
 text\_widget.pack(expand=True, fill='both') # Espande il widget per riempire la finestra  
 operations = load\_operations() # Carica le operazioni dalla cronologia  
 for operation in operations: # Inserisce ogni operazione nel widget Text  
 text\_widget.insert(tk.END, operation)  
 text\_widget.config(state='disabled') # Rende il widget Text di sola lettura  
  
expression = "" # Inizializza la variabile globale 'expression' come stringa vuota  
root = tk.Tk() # Crea la finestra principale dell'applicazione  
root.title("Calcolatrice") # Imposta il titolo della finestra principale  
root.geometry("600x800") # Imposta le dimensioni della finestra principale  
  
# Crea un widget Entry per mostrare le espressioni  
entry = tk.Entry(root, justify='right', font=('Arial', 24))  
entry.grid(row=0, column=0, columnspan=5, sticky='nsew') # Posiziona l'entry nella griglia  
  
# Configura il ridimensionamento delle righe e colonne della griglia  
root.grid\_rowconfigure(0, weight=1)  
for i in range(1, 6):  
 root.grid\_rowconfigure(i, weight=1)  
for i in range(5):  
 root.grid\_columnconfigure(i, weight=1)  
  
# Definisce i pulsanti della calcolatrice nell'ordine in cui appaiono  
buttons = [  
 '7', '8', '9', '+', '(',  
 '4', '5', '6', '-', ')',  
 '1', '2', '3', 'x', '%',  
 'C', '0', '.', '=', '/',  
 'History'  
]  
  
# Crea e posiziona i pulsanti nella griglia  
for i, button in enumerate(buttons):  
 btn = tk.Button(root, text=button, font=('Arial', 20), width=5) # Crea un pulsante  
 btn.grid(row=(i//5)+1, column=i%5, sticky='nsew') # Posiziona il pulsante nella griglia  
 btn.bind('<Button-1>', on\_click) # Associa la funzione on\_click all'evento di clic del pulsante  
  
root.mainloop() # Avvia il ciclo principale dell'applicazione tkinter

E’ già molto commentato il codice ed inoltre dovreste aver raggiunto un ottimo livello, ma in ogni caso vi aggiungo un’ulteriore spiegazione del codice:

Questo script implementa una semplice calcolatrice GUI usando la libreria tkinter di Python. Di seguito è riportata una spiegazione passo-passo del codice.

#### Importazione della Libreria

import tkinter as tk

* Importa il modulo tkinter e lo assegna al nome tk. tkinter è una libreria standard di Python utilizzata per creare interfacce grafiche.

#### Definizione della Funzione on\_click

def on\_click(event=None):

global expression

text = event.widget.cget("text")

if text == "=":

try:

expression\_to\_evaluate = expression.replace('x', '\*').replace('%', '/100\*')

result = str(eval(expression\_to\_evaluate))

entry.delete(0, tk.END)

entry.insert(tk.END, result)

save\_operation(f"{expression} = {result}")

except Exception as e:

entry.delete(0, tk.END)

entry.insert(tk.END, "Error")

expression = ""

elif text == "C":

entry.delete(0, tk.END)

expression = ""

elif text == "History":

show\_history()

else:

expression += str(text)

entry.delete(0, tk.END)

entry.insert(tk.END, expression)

* **Parametri**: La funzione prende un parametro opzionale event.
* **Variabile Globale**: expression è una stringa che mantiene l'espressione corrente.
* **Gestione dei Bottoni**:
  + **=**: Valuta l'espressione e mostra il risultato.
  + **C**: Cancella l'espressione corrente.
  + **History**: Mostra la cronologia delle operazioni.
  + **Altri**: Aggiunge il testo del bottone all'espressione.

#### Salvataggio delle Operazioni

def save\_operation(operation):

with open("operations\_history.txt", "a") as file:

file.write(operation + "\n")

* **Salva l'Operazione**: Appende l'operazione alla fine del file operations\_history.txt.

#### Caricamento delle Operazioni

def load\_operations():

try:

with open("operations\_history.txt", "r") as file:

return file.readlines()

except FileNotFoundError:

return []

* **Carica le Operazioni**: Legge tutte le operazioni dal file operations\_history.txt. Se il file non esiste, restituisce una lista vuota.

#### Mostrare la Cronologia

def show\_history():

history\_window = tk.Toplevel(root)

history\_window.title("Cronologia Operazioni")

history\_window.geometry("300x400")

text\_widget = tk.Text(history\_window, font=('Arial', 12), wrap='word')

text\_widget.pack(expand=True, fill='both')

operations = load\_operations()

for operation in operations:

text\_widget.insert(tk.END, operation)

text\_widget.config(state='disabled') # Make the text widget read-only

* **Finestra della Cronologia**: Crea una nuova finestra che mostra le operazioni salvate.
* **Toplevel**: Crea una finestra secondaria.
* **Text Widget**: Utilizzato per mostrare le operazioni.

#### Configurazione dell'Interfaccia Principale

expression = ""

root = tk.Tk()

root.title("Calcolatrice")

root.geometry("600x800")

* **Variabile Globale**: expression inizialmente vuota.
* **Finestra Principale**: Crea la finestra principale dell'applicazione.

#### Entry Widget

entry = tk.Entry(root, justify='right', font=('Arial', 24))

entry.grid(row=0, column=0, columnspan=5, sticky='nsew')

* **Entry Widget**: Campo di testo per mostrare l'espressione corrente.
* **Griglia**: Posiziona l'entry nella griglia.

#### Configurazione della Griglia

root.grid\_rowconfigure(0, weight=1)

for i in range(1, 6):

root.grid\_rowconfigure(i, weight=1)

for i in range(5):

root.grid\_columnconfigure(i, weight=1)

* **Configurazione delle Righe e Colonne**: Assicura che le righe e le colonne della griglia si ridimensionino correttamente.

#### Pulsanti della Calcolatrice

buttons = [

'7', '8', '9', '+', '(',

'4', '5', '6', '-', ')',

'1', '2', '3', 'x', '%',

'C', '0', '.', '=', '/',

'History'

]

* **Lista dei Pulsanti**: Definisce l'ordine e il testo dei pulsanti della calcolatrice.

#### Posizionamento dei Pulsanti

for i, button in enumerate(buttons):

btn = tk.Button(root, text=button, font=('Arial', 20), width=5)

btn.grid(row=(i//5)+1, column=i%5, sticky='nsew') # i//5+1 per le righe, i%5 per le colonne

btn.bind('<Button-1>', on\_click)

* **Creazione dei Pulsanti**: Crea e posiziona i pulsanti nella griglia.
* **Event Binding**: Associa la funzione on\_click a ogni pulsante.

#### Esecuzione del Ciclo Principale

root.mainloop()

* **Ciclo Principale**: Esegue l'applicazione e gestisce gli eventi finché la finestra non viene chiusa.

### Riassunto

Questo script crea una semplice calcolatrice GUI con funzionalità di base come calcoli aritmetici, pulizia dell'espressione corrente, visualizzazione della cronologia delle operazioni e gestione degli errori. Utilizza tkinter per l'interfaccia grafica, eval per valutare le espressioni (con alcune sostituzioni per supportare x come moltiplicazione e % come percentuale), e file di testo per salvare e caricare la cronologia delle operazioni.

**Ed ora vi mostro come creare la relativa applicazione per windows, linux e mac.**

Per creare un'applicazione desktop per Windows, Linux (Ubuntu) e Mac utilizzando il file Python calc.py che hai allegato, puoi seguire le istruzioni dettagliate qui sotto. Utilizzeremo PyInstaller per creare eseguibili stand-alone per ciascuna piattaforma.

**1. Installare PyInstaller**

Prima di tutto, installa PyInstaller sul tuo sistema. Apri il terminale o il prompt dei comandi e digita:

pip install pyinstaller

**2. Impacchettare lo Script Python**

**Per Windows:**

1. **Apri il Prompt dei Comandi**:
   * Naviga nella directory che contiene il tuo script Python, calc.py.
2. **Crea l'Eseguibile**:
   * Esegui il seguente comando (da terminale):

pyinstaller --onefile calc.py

1. **Trova l'Eseguibile**:
   * Dopo che il processo è completato, l'eseguibile sarà situato nella cartella dist all'interno della tua directory di progetto.

**Per Ubuntu (Linux):**

1. **Apri il Terminale**:
   * Naviga nella directory che contiene il tuo script Python, calc.py.
2. **Crea l'Eseguibile**:
   * Esegui il seguente comando:

pyinstaller --onefile calc.py

1. **Trova l'Eseguibile**:
   * Dopo che il processo è completato, l'eseguibile sarà situato nella cartella dist all'interno della tua directory di progetto.
2. **Rendi il File Eseguibile**:
   * Naviga nella cartella dist e rendi il file eseguibile eseguendo:

chmod +x calc

**Per Mac:**

1. **Apri il Terminale**:
   * Naviga nella directory che contiene il tuo script Python, calc.py.
2. **Crea l'Eseguibile**:
   * Esegui il seguente comando:

pyinstaller --onefile calc.py

1. **Trova l'Eseguibile**:
   * Dopo che il processo è completato, l'eseguibile sarà situato nella cartella dist all'interno della tua directory di progetto.

**3. Opzioni Aggiuntive per PyInstaller**

* **Aggiungi un'Icona**: Se desideri aggiungere un'icona personalizzata al tuo eseguibile, utilizza l'opzione --icon:

pyinstaller --onefile --icon=path/to/icon.ico calc.py

* **Includi File di Dati**: Se il tuo script dipende da file esterni, puoi includerli utilizzando l'opzione --add-data:

pyinstaller --onefile --add-data "src:data" calc.py

**Esempio Completo:**

Se desideri includere un'icona e alcuni file di dati, il comando completo potrebbe essere:

pyinstaller --onefile --icon=path/to/icon.ico --add-data "src:data" calc.py

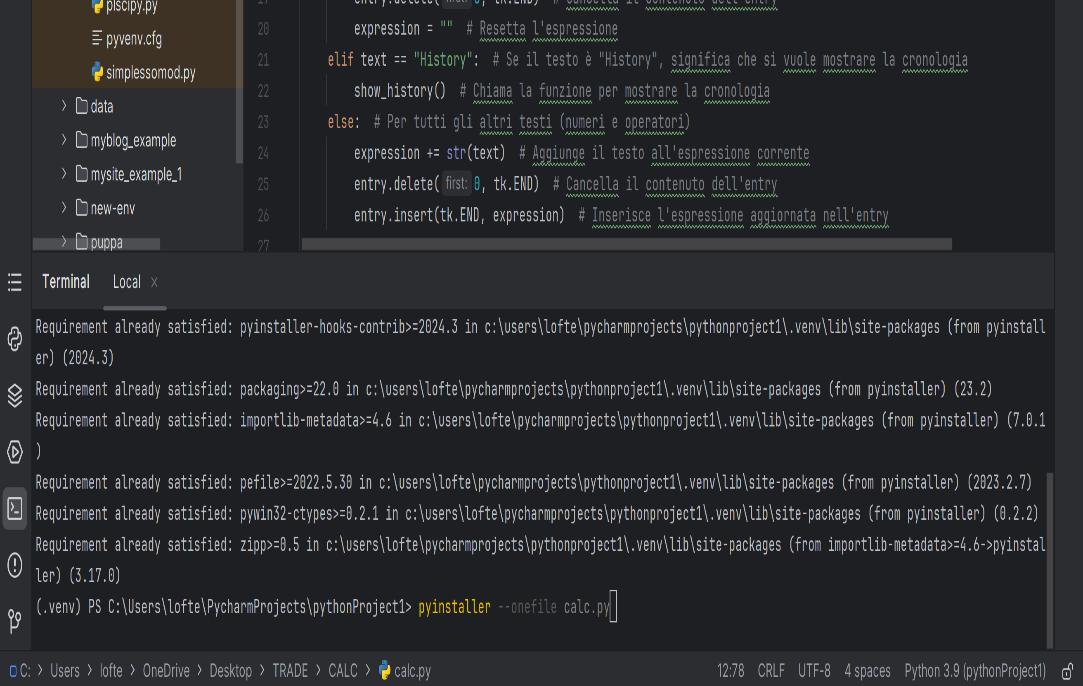
**4. Esecuzione dell'Eseguibile**

* **Windows**: Doppio clic sul file calc.exe nella cartella dist.
* **Ubuntu**: Apri il terminale, naviga nella cartella dist ed esegui ./calc.
* **Mac**: Apri il terminale, naviga nella cartella dist ed esegui ./calc.

Seguendo questi passaggi, dovresti essere in grado di creare eseguibili stand-alone per Windows, Linux e Mac a partire dal tuo script Python calc.py. Se hai bisogno di ulteriori dettagli o assistenza su qualche passaggio specifico, fammelo sapere!

Ecco alcune screen dei vari passaggi:





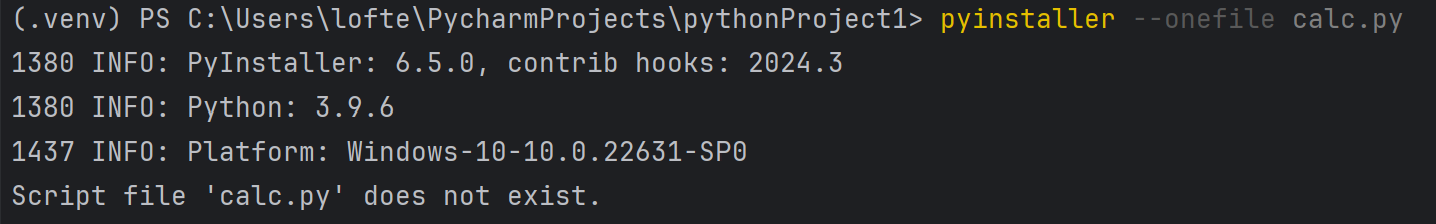
 Questo è un errore comune. Dovete recarvi nel percorso dove è presente il vostro file calc.py

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

se avete digitato il comando nel percorso corretto, nella cartella del vostro script dovreste avere i seguenti file:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente aprite dist: Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamente

ora potete avviare la vostra calcolatrice:

Immagine che contiene testo, schermata, numero, calendario

Descrizione generata automaticamente

Passiamo ad un altro esempio. Proviamo a creare un gioco.

Qui di seguito vi mostro come realizzare una copia del famoso gioco Tetris:

import pygame  
import random  
  
# Inizializzazione Pygame  
pygame.init()  
  
# Costanti  
WIDTH, HEIGHT = 300, 600  
GRID\_SIZE = 30  
FPS = 60  
BLACK = (0, 0, 0)  
FALL\_SPEED = 0.5 # Puoi regolare questo valore per cambiare la velocità di discesa  
POINTS\_PER\_LINE = 1 # Punteggio per ogni riga completata  
WINNING\_SCORE = 10 # Punteggio necessario per vincere  
  
# Inizializzazione finestra di gioco  
screen = pygame.display.set\_mode((WIDTH, HEIGHT))  
pygame.display.set\_caption("Tetris")  
  
clock = pygame.time.Clock()  
  
# Definizione dei pezzi di Tetris  
SHAPES = [  
 [[1, 1, 1, 1]],  
 [[1, 1], [1, 1]],  
 [[1, 1, 1], [0, 1, 0]],  
 [[1, 1, 1], [1, 0, 0]],  
 [[1, 1, 1], [0, 0, 1]],  
 [[1, 1, 0], [0, 1, 1]],  
 [[0, 1, 1], [1, 1, 0]]  
]  
  
# Associa un colore univoco a ciascun tipo di pezzo  
PIECE\_COLORS = {  
 0: (255, 0, 0), # Rosso  
 1: (0, 255, 0), # Verde  
 2: (0, 0, 255), # Blu  
 3: (255, 255, 0), # Giallo  
 4: (255, 165, 0), # Arancione  
 5: (128, 0, 128), # Viola  
 6: (0, 255, 255) # Ciano  
}  
  
# Inizializzazione griglia di gioco  
grid = [[0] \* (WIDTH // GRID\_SIZE) for \_ in range(HEIGHT // GRID\_SIZE)]  
  
#Disegna la Griglia:  
def draw\_grid():  
 for row in range(len(grid)):  
 for col in range(len(grid[row])):  
 if grid[row][col] != 0:  
 color = PIECE\_COLORS[grid[row][col] - 1] # -1 per ottenere l'ID corretto  
 pygame.draw.rect(screen, color, (col \* GRID\_SIZE, row \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE))  
 pygame.draw.rect(screen, BLACK, (col \* GRID\_SIZE, row \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE), 1)  
  
#Disegna un Pezzo:  
def draw\_piece(piece, offset, piece\_id):  
 for row in range(len(piece)):  
 for col in range(len(piece[row])):  
 if piece[row][col] == 1:  
 color = PIECE\_COLORS[piece\_id]  
 pygame.draw.rect(screen, color, ((offset[0] + col) \* GRID\_SIZE, (offset[1] + row) \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE))  
 pygame.draw.rect(screen, BLACK, ((offset[0] + col) \* GRID\_SIZE, (offset[1] + row) \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE), 1)  
  
#Controlla le Collisioni:  
def check\_collision(piece, offset):  
 for row in range(len(piece)):  
 for col in range(len(piece[row])):  
 if piece[row][col] == 1:  
 if offset[1] + row >= len(grid) or offset[0] + col < 0 or offset[0] + col >= len(grid[0]) or grid[offset[1] + row][offset[0] + col] != 0:  
 return True  
 return False  
  
#Unisci il Pezzo alla Griglia:  
def merge\_piece(piece, offset, piece\_id):  
 for row in range(len(piece)):  
 for col in range(len(piece[row])):  
 if piece[row][col] == 1:  
 grid[offset[1] + row][offset[0] + col] = piece\_id + 1 # +1 per ottenere l'ID corretto  
  
#Rimuovi le Righe Completate:  
def remove\_completed\_rows():  
 completed\_rows = [row for row in range(len(grid)) if all(grid[row])]  
 for row in completed\_rows:  
 del grid[row]  
 grid.insert(0, [0] \* len(grid[0]))  
 return len(completed\_rows)  
  
#Ruota un Pezzo:  
def rotate\_piece(piece):  
 return [[piece[j][i] for j in range(len(piece))] for i in range(len(piece[0]) - 1, -1, -1)]  
  
#Mostra un Messaggio:  
def display\_message(message):  
 font = pygame.font.Font(None, 36)  
 text = font.render(message, True, (255, 255, 255))  
 screen.blit(text, (WIDTH // 4, HEIGHT // 2))  
 pygame.display.flip()  
 pygame.time.wait(2000) # Mostra il messaggio per 2 secondi  
 pygame.quit()  
 quit()  
  
#Funzione Principale  
def main():  
 current\_piece = random.choice(SHAPES)  
 piece\_id = SHAPES.index(current\_piece) # Ottieni l'ID del pezzo corrente  
 piece\_offset = [len(grid[0]) // 2 - len(current\_piece[0]) // 2, 0]  
  
 fall\_time = 0  
 score = 0  
  
 while True:  
 dt = clock.tick(FPS) / 1000 # Ottieni il tempo trascorso in secondi  
  
 fall\_time += dt  
  
 if fall\_time >= FALL\_SPEED:  
 fall\_time = 0  
  
 new\_offset = [piece\_offset[0], piece\_offset[1] + 1]  
 if not check\_collision(current\_piece, new\_offset):  
 piece\_offset = new\_offset  
 else:  
 merge\_piece(current\_piece, piece\_offset, piece\_id)  
 lines\_cleared = remove\_completed\_rows()  
 score += lines\_cleared \* POINTS\_PER\_LINE  
  
 if score >= WINNING\_SCORE:  
 display\_message("Hai vinto!")  
  
 current\_piece = random.choice(SHAPES)  
 piece\_id = SHAPES.index(current\_piece)  
 piece\_offset = [len(grid[0]) // 2 - len(current\_piece[0]) // 2, 0]  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_LEFT:  
 new\_offset = [piece\_offset[0] - 1, piece\_offset[1]]  
 if not check\_collision(current\_piece, new\_offset):  
 piece\_offset = new\_offset  
 elif event.key == pygame.K\_RIGHT:  
 new\_offset = [piece\_offset[0] + 1, piece\_offset[1]]  
 if not check\_collision(current\_piece, new\_offset):  
 piece\_offset = new\_offset  
 elif event.key == pygame.K\_DOWN:  
 new\_offset = [piece\_offset[0], piece\_offset[1] + 1]  
 if not check\_collision(current\_piece, new\_offset):  
 piece\_offset = new\_offset  
 elif event.key == pygame.K\_UP:  
 rotated\_piece = rotate\_piece(current\_piece)  
 if not check\_collision(rotated\_piece, piece\_offset):  
 current\_piece = rotated\_piece  
  
 screen.fill(BLACK)  
 draw\_grid()  
 draw\_piece(current\_piece, piece\_offset, piece\_id)  
  
 # Mostra il punteggio  
 font = pygame.font.Font(None, 36)  
 score\_text = font.render(f"Punteggio: {score}", True, (255, 255, 255))  
 screen.blit(score\_text, (10, 10))  
  
 pygame.display.flip()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Spiegazione del codice:

**Definizione dei Pezzi di Tetris e dei Colori**

**SHAPES**

SHAPES = [

[[1, 1, 1, 1]],

[[1, 1], [1, 1]],

[[1, 1, 1], [0, 1, 0]],

[[1, 1, 1], [1, 0, 0]],

[[1, 1, 1], [0, 0, 1]],

[[1, 1, 0], [0, 1, 1]],

[[0, 1, 1], [1, 1, 0]]

]

* Questa lista contiene le varie forme dei pezzi di Tetris rappresentati come matrici di 1 e 0.
* Ogni sub-lista rappresenta una forma diversa:
  + [[1, 1, 1, 1]]: Linea orizzontale.
  + [[1, 1], [1, 1]]: Blocco quadrato.
  + [[1, 1, 1], [0, 1, 0]]: T.
  + [[1, 1, 1], [1, 0, 0]]: L rovesciata.
  + [[1, 1, 1], [0, 0, 1]]: L.
  + [[1, 1, 0], [0, 1, 1]]: S.
  + [[0, 1, 1], [1, 1, 0]]: Z.

**PIECE\_COLORS**

PIECE\_COLORS = {

0: (255, 0, 0), # Rosso

1: (0, 255, 0), # Verde

2: (0, 0, 255), # Blu

3: (255, 255, 0), # Giallo

4: (255, 165, 0), # Arancione

5: (128, 0, 128), # Viola

6: (0, 255, 255) # Ciano

}

* Un dizionario che associa un colore a ciascun tipo di pezzo. Gli indici corrispondono all'indice delle forme in SHAPES.

**Disegna la Griglia**

**draw\_grid**

def draw\_grid():

for row in range(len(grid)):

for col in range(len(grid[row])):

if grid[row][col] != 0:

color = PIECE\_COLORS[grid[row][col] - 1] # -1 per ottenere l'ID corretto

pygame.draw.rect(screen, color, (col \* GRID\_SIZE, row \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE))

pygame.draw.rect(screen, BLACK, (col \* GRID\_SIZE, row \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE), 1)

* Disegna la griglia di gioco e riempie i blocchi con i colori corretti se sono occupati.
* grid[row][col] != 0: Controlla se la cella della griglia è occupata.
* color = PIECE\_COLORS[grid[row][col] - 1]: Ottiene il colore corretto per il blocco.
* pygame.draw.rect(screen, color, ...): Disegna il blocco riempito con il colore.
* pygame.draw.rect(screen, BLACK, ..., 1): Disegna il bordo del blocco.

**Disegna un Pezzo**

**draw\_piece**

def draw\_piece(piece, offset, piece\_id):

for row in range(len(piece)):

for col in range(len(piece[row])):

if piece[row][col] == 1:

color = PIECE\_COLORS[piece\_id]

pygame.draw.rect(screen, color, ((offset[0] + col) \* GRID\_SIZE, (offset[1] + row) \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE))

pygame.draw.rect(screen, BLACK, ((offset[0] + col) \* GRID\_SIZE, (offset[1] + row) \* GRID\_SIZE, GRID\_SIZE, GRID\_SIZE), 1)

* Disegna il pezzo corrente sullo schermo.
* piece è la matrice che rappresenta il pezzo.
* offset è una coppia di coordinate (x, y) che indica la posizione del pezzo nella griglia.
* piece\_id è l'indice del pezzo in SHAPES.
* color = PIECE\_COLORS[piece\_id]: Ottiene il colore del pezzo.
* pygame.draw.rect(screen, color, ...): Disegna il blocco del pezzo.

**Controlla le Collisioni**

**check\_collision**

def check\_collision(piece, offset):

for row in range(len(piece)):

for col in range(len(piece[row])):

if piece[row][col] == 1:

if offset[1] + row >= len(grid) or offset[0] + col < 0 or offset[0] + col >= len(grid[0]) or grid[offset[1] + row][offset[0] + col] != 0:

return True

return False

* Controlla se il pezzo corrente collide con la griglia o altri pezzi.
* Controlla se il pezzo supera i bordi della griglia o se collide con blocchi già presenti.

**Unisci il Pezzo alla Griglia**

**merge\_piece**

def merge\_piece(piece, offset, piece\_id):

for row in range(len(piece)):

for col in range(len(piece[row])):

if piece[row][col] == 1:

grid[offset[1] + row][offset[0] + col] = piece\_id + 1 # +1 per ottenere l'ID corretto

* Aggiunge il pezzo corrente alla griglia.
* grid[offset[1] + row][offset[0] + col] = piece\_id + 1: Imposta la cella della griglia con l'ID del pezzo.

**Rimuovi le Righe Completate**

**remove\_completed\_rows**

def remove\_completed\_rows():

completed\_rows = [row for row in range(len(grid)) if all(grid[row])]

for row in completed\_rows:

del grid[row]

grid.insert(0, [0] \* len(grid[0]))

return len(completed\_rows)

* Rimuove le righe complete dalla griglia e restituisce il numero di righe rimosse.
* completed\_rows = [row for row in range(len(grid)) if all(grid[row])]: Trova le righe complete.
* del grid[row]: Rimuove la riga completa.
* grid.insert(0, [0] \* len(grid[0])): Aggiunge una riga vuota in cima alla griglia.

**Funzione Principale**

**main**

def main():

current\_piece = random.choice(SHAPES)

piece\_id = SHAPES.index(current\_piece) # Ottieni l'ID del pezzo corrente

piece\_offset = [len(grid[0]) // 2 - len(current\_piece[0]) // 2, 0]

fall\_time = 0

score = 0

while True:

dt = clock.tick(FPS) / 1000 # Ottieni il tempo trascorso in secondi

fall\_time += dt

if fall\_time >= FALL\_SPEED:

fall\_time = 0

new\_offset = [piece\_offset[0], piece\_offset[1] + 1]

if not check\_collision(current\_piece, new\_offset):

piece\_offset = new\_offset

else:

merge\_piece(current\_piece, piece\_offset, piece\_id)

lines\_cleared = remove\_completed\_rows()

score += lines\_cleared \* POINTS\_PER\_LINE

if score >= WINNING\_SCORE:

display\_message("Hai vinto!")

current\_piece = random.choice(SHAPES)

piece\_id = SHAPES.index(current\_piece)

piece\_offset = [len(grid[0]) // 2 - len(current\_piece[0]) // 2, 0]

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

new\_offset = [piece\_offset[0] - 1, piece\_offset[1]]

if not check\_collision(current\_piece, new\_offset):

piece\_offset = new\_offset

elif event.key == pygame.K\_RIGHT:

new\_offset = [piece\_offset[0] + 1, piece\_offset[1]]

if not check\_collision(current\_piece, new\_offset):

piece\_offset = new\_offset

elif event.key == pygame.K\_DOWN:

new\_offset = [piece\_offset[0], piece\_offset[1] + 1]

if not check\_collision(current\_piece, new\_offset):

piece\_offset = new\_offset

elif event.key == pygame.K\_UP:

rotated\_piece = rotate\_piece(current\_piece)

if not check\_collision(rotated\_piece, piece\_offset):

current\_piece = rotated\_piece

screen.fill(BLACK)

draw\_grid()

draw\_piece(current\_piece, piece\_offset, piece\_id)

# Mostra il punteggio

font = pygame.font.Font(None, 36)

score\_text = font.render(f"Punteggio: {score}", True, (255, 255, 255))

screen.blit(score\_text, (10, 10))

pygame.display.flip()

if \_\_name\_\_ == "\_\_main\_\_":

main()

* **Inizializzazione del Gioco**:
  + current\_piece e piece\_id: Seleziona un pezzo casuale e ottiene il suo ID.
  + piece\_offset: Imposta la posizione iniziale del pezzo.
  + fall\_time e score: Inizializza il tempo di caduta e il punteggio.
* **Ciclo di Gioco**:
  + dt = clock.tick(FPS) / 1000: Calcola il tempo trascorso.
  + Aggiorna fall\_time e controlla se è il momento di far cadere il pezzo.
  + Controlla le collisioni e, se necessario, unisce il pezzo alla griglia.
  + Rimuove le righe completate e aggiorna il punteggio.
  + Se il punteggio raggiunge WINNING\_SCORE, mostra un messaggio di vittoria.
* **Gestione degli Eventi**:
  + Controlla l'input dell'utente per spostare e ruotare i pezzi.
* **Aggiornamento della Schermata**:
  + Riempie lo schermo di nero, disegna la griglia e il pezzo corrente, e mostra il punteggio.
* **Avvio del Gioco**:
  + Se il file viene eseguito direttamente, chiama la funzione main.

Questo script implementa un gioco base di Tetris con le funzioni di disegno, controllo delle collisioni, gestione del punteggio e controllo degli input dell'utente.

Con lo script successivo andiamo a recuperare gli articoli da **wikipedia**:

import requests  
  
def wikipedia\_search(query):  
 # Endpoint per la ricerca Wikipedia in italiano  
 search\_url = "https://it.wikipedia.org/w/api.php"  
 params = {  
 "action": "query",  
 "list": "search",  
 "srsearch": query,  
 "format": "json",  
 "utf8": "",  
 "srlimit": 3 # Limitiamo i risultati ai primi 3  
 }  
 response = requests.get(search\_url, params=params)  
 if response.status\_code == 200:  
 search\_results = response.json().get("query", {}).get("search", [])  
 return search\_results  
 else:  
 return None  
  
def show\_search\_results(results):  
 for i, result in enumerate(results, 1):  
 print(f"{i}: {result['title']}")  
  
def get\_article\_snippet(pageid):  
 # Endpoint per il contenuto della pagina Wikipedia in italiano  
 content\_url = "https://it.wikipedia.org/w/api.php"  
 params = {  
 "action": "query",  
 "pageids": pageid,  
 "prop": "extracts",  
 "exintro": True, # Ottiene solo l'introduzione/riassunto dell'articolo  
 "explaintext": True,  
 "format": "json"  
 }  
 response = requests.get(content\_url, params=params)  
 if response.status\_code == 200:  
 page = response.json().get("query", {}).get("pages", {}).get(str(pageid), {})  
 return page.get("extract", "")  
 else:  
 return None  
  
def main():  
 query = input("Inserisci la tua ricerca per Wikipedia: ")  
 results = wikipedia\_search(query)  
 if results:  
 show\_search\_results(results)  
 choice = int(input("Scegli un articolo per leggere il riassunto (1-3): "))  
 if choice in range(1, 4):  
 snippet = get\_article\_snippet(results[choice-1]["pageid"])  
 print(snippet)  
 else:  
 print("Scelta non valida.")  
 else:  
 print("Nessun risultato trovato.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Questo script è davvero molto semplice; qui di seguito vi fornisco una breve spiegazione:

**Funzione wikipedia\_search(query)**:

* **Scopo**: Effettua una ricerca su Wikipedia utilizzando l'API di Wikipedia in italiano.
* **Parametri**: query (stringa di ricerca).
* **Operazioni**:
  + Definisce l'URL dell'API di Wikipedia.
  + Imposta i parametri della richiesta, inclusa la query di ricerca e il formato di risposta.
  + Effettua una richiesta HTTP GET all'API di Wikipedia.
  + Se la risposta è valida (codice di stato 200), restituisce i primi 3 risultati della ricerca.
  + Se la risposta non è valida, restituisce None.

**Funzione show\_search\_results(results)**:

* **Scopo**: Visualizza i titoli degli articoli trovati nella ricerca.
* **Parametri**: results (lista dei risultati della ricerca).
* **Operazioni**:
  + Itera sui risultati e stampa il titolo di ogni articolo con un indice numerico.

**Funzione get\_article\_snippet(pageid)**:

* **Scopo**: Ottiene il riassunto/intestazione di un articolo di Wikipedia utilizzando l'API di Wikipedia in italiano.
* **Parametri**: pageid (ID della pagina dell'articolo).
* **Operazioni**:
  + Definisce l'URL dell'API di Wikipedia.
  + Imposta i parametri della richiesta, specificando di ottenere solo l'introduzione dell'articolo in formato testuale.
  + Effettua una richiesta HTTP GET all'API di Wikipedia.
  + Se la risposta è valida (codice di stato 200), restituisce il riassunto dell'articolo.
  + Se la risposta non è valida, restituisce None.

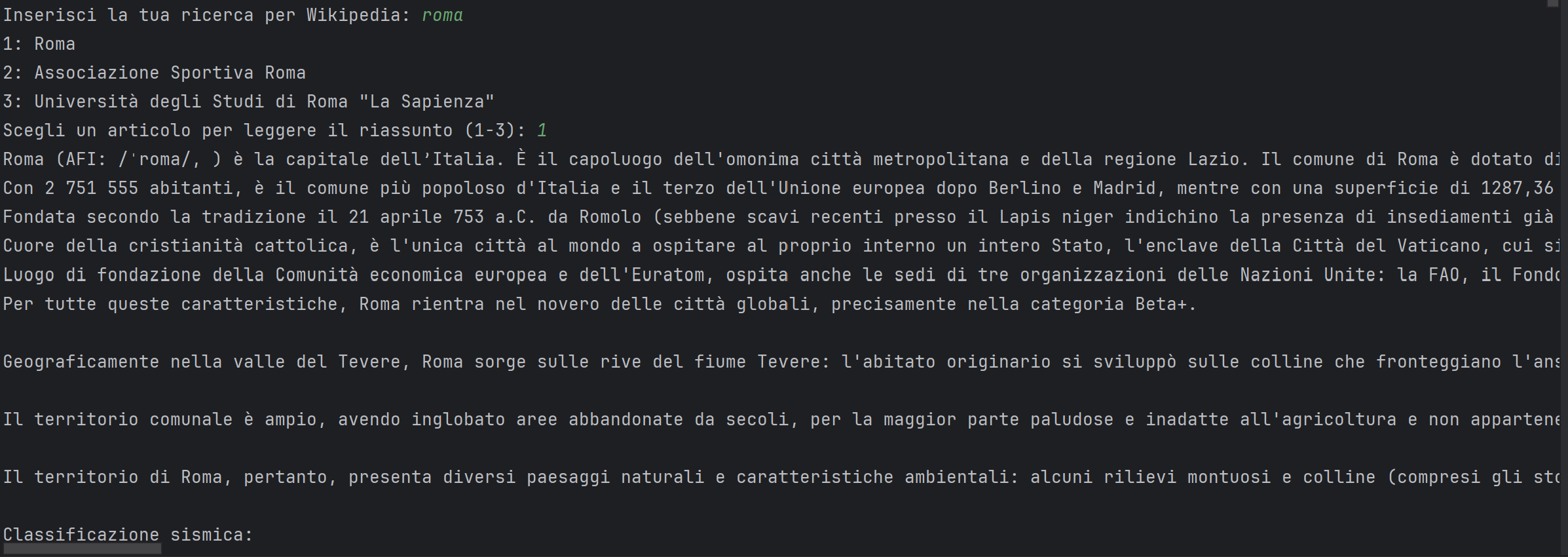
**Funzione main()**:

* **Scopo**: Gestisce l'interazione con l'utente e coordina le funzioni di ricerca e visualizzazione.
* **Operazioni**:
  + Chiede all'utente di inserire una query di ricerca.
  + Esegue la funzione wikipedia\_search(query) con la query fornita dall'utente.
  + Se ci sono risultati:
    - Visualizza i risultati della ricerca con show\_search\_results(results).
    - Chiede all'utente di scegliere un articolo tra i risultati (1-3).
    - Se la scelta è valida (1-3), esegue get\_article\_snippet(pageid) per ottenere e stampare il riassunto dell'articolo scelto.
    - Se la scelta non è valida, stampa un messaggio di errore.
  + Se non ci sono risultati, informa l'utente che non sono stati trovati risultati.

**Blocco if \_\_name\_\_ == "\_\_main\_\_":**:

* **Scopo**: Assicura che la funzione main() venga eseguita solo se lo script viene eseguito direttamente, non quando importato come modulo in un altro script.

Output dello script:



Da tenere presente che lo script limita i risultati ai primi tre più pertinenti. Potreste modificare lo script per aumentare i risultati o perfezionare lo script con altre migliorie.

**Introduzione a Scikit-Learn** Scikit-Learn è una libreria per il machine learning che fornisce strumenti semplici ed efficienti per l'analisi e la modellazione dei dati.

* **Esempio di utilizzo di Scikit-Learn**

**Link:** <https://scikit-learn.org/>

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Caricare il dataset Iris

iris = load\_iris()

X = iris.data

y = iris.target

# Suddividere i dati in training e test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Creare e addestrare il modello

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

# Fare previsioni e valutare il modello

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

### Spiegazione del codice

1. **Importazione dei moduli Scikit-learn**: (link: https://scikit-learn.org/stable/)

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

* + **load\_iris**: Funzione per caricare il dataset Iris.
  + **train\_test\_split**: Funzione per suddividere i dati in set di addestramento e test.
  + **RandomForestClassifier**: Classe per creare un modello di classificazione Random Forest.
  + **accuracy\_score**: Funzione per calcolare l'accuratezza del modello.

1. **Caricamento del dataset Iris**:

iris = load\_iris()

X = iris.data

y = iris.target

* + **load\_iris**: Carica il dataset Iris, che è un classico dataset per il machine learning. Contiene 150 campioni di iris, ciascuno con 4 caratteristiche (lunghezza e larghezza del sepalo e del petalo) e una delle 3 specie di iris come target.
  + **X**: Matrice delle caratteristiche (features).
  + **y**: Vettore dei target (etichette delle classi).

1. **Suddivisione dei dati in set di addestramento e test**:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

* + **train\_test\_split**: Suddivide i dati in set di addestramento (70%) e set di test (30%). Cambiando il valore di test\_size verranno modificati i dati, ad esempio 0.2, imposta 80% addestramento e 20% test.
  + **test\_size=0.3**: Indica che il 30% dei dati sarà utilizzato per il test.
  + **random\_state=42**: Imposta un seme per la generazione casuale, garantendo che la suddivisione sia riproducibile.

1. **Creazione e addestramento del modello**:

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

* + **RandomForestClassifier**: (classe) Crea un'istanza del modello di classificazione Random Forest. (Link: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>)
  + **Model**: istanza della classe **RandomForestClassifier**
  + **fit**: Metodo. Addestra il modello utilizzando i dati di addestramento. Nota: Model crea un oggetto a partire dalla classe **RandomForestClassifier, in altre parole istanzia (crea)**

1. **Previsioni e valutazione del modello**:

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

* + **predict**: Fa previsioni sui dati di test utilizzando il modello addestrato.
  + **accuracy\_score**: Calcola l'accuratezza del modello confrontando le previsioni (y\_pred) con le etichette effettive (y\_test).

### Esecuzione del codice

Quando esegui questo script, il modello Random Forest viene addestrato sui dati di addestramento e valutato sui dati di test. Viene stampata l'accuratezza del modello.

### Output del codice

L'output sarà simile a questo:

makefile

Accuracy: 1.0

L'accuratezza può variare leggermente a seconda della suddivisione dei dati, ma con il seme random\_state=42, dovresti ottenere risultati riproducibili.

### Riepilogo

* **Importazione dei moduli Scikit-learn**: Vengono importati i moduli necessari per caricare i dati, suddividerli, creare il modello e valutare le prestazioni.
* **Caricamento del dataset Iris**: Viene caricato il dataset Iris e le caratteristiche e le etichette vengono separate.
* **Suddivisione dei dati**: I dati vengono suddivisi in set di addestramento e test.
* **Creazione e addestramento del modello**: Viene creato e addestrato un modello di classificazione Random Forest.
* **Previsioni e valutazione**: Vengono fatte previsioni sui dati di test e l'accuratezza del modello viene calcolata e stampata.

In questo esempio, abbiamo visto come utilizzare Scikit-learn per caricare un dataset, suddividere i dati, addestrare un modello di classificazione e valutare le sue prestazioni. Scikit-learn è una libreria potente e flessibile per il machine learning in Python, che offre molte funzionalità per costruire e valutare modelli di apprendimento automatico.

Inizio modulo

Fine modulo

Ed infine due script per addestrare un modello di AI basato sull’architettura transformer ed uno su LSTM.

LSTM è memoria a lungo e breve termine (Long short-term memory - LSTM) è una variante di RNN che consente al modello di espandere la propria capacità di memoria per adattarsi ad un arco temporale più lungo.

Una rete neurale ricorrente (RNN) è un modello di deep learning addestrato per elaborare e convertire un input di dati sequenziale in un output di dati sequenziale specifico. I dati sequenziali sono dati, come parole, frasi o dati di serie temporali, in cui i componenti sequenziali sono correlati in base a regole semantiche e sintattiche complesse. Una RNN è un sistema software costituito da molti componenti interconnessi che imitano il modo in cui gli umani eseguono conversioni sequenziali di dati, come la traduzione di testo da una lingua all'altra. Le RNN vengono in gran parte sostituite dall'intelligenza artificiale (IA) basata su trasformatori e dai modelli linguistici di grandi dimensioni (LLM), che sono molto più efficienti nell'elaborazione sequenziale dei dati.

I modelli transformer o meglio l’architettura transformer di deep learning sviluppata da google nel 2017 e la sua caratteristica principale, per spiegarvelo in parole povere, utilizza un meccanismo multitesta (<https://arxiv.org/abs/1706.03762>) particolarmente adatti per il processamento del linguaggio naturale NPL in grado di comprendere e generalizzare contesti assai più complessi dei modelli LSTM. OpenAI per ChatGPT (GPT Generative Pre-trained Transformer)utilizza Pythorch con architettura transformer.

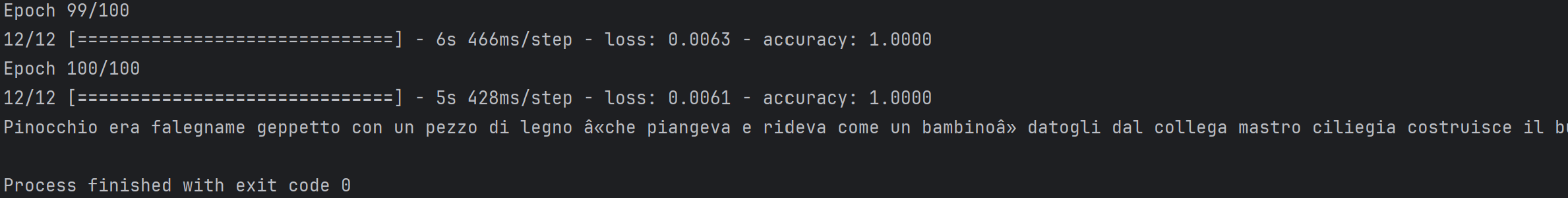
Mi fermi qui, perché avremmo necessità di un libro intero per parlare di modelli AI, la matematica che c’è dietro e tutte le tecniche di sviluppo.

Vi illustro brevemente due modelli semplicissimi per la generazione del linguaggio naturale:

***LSTM***

import tensorflow as tf  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.optimizers import Adam  
import numpy as np  
  
gpus = tf.config.experimental.list\_physical\_devices('GPU')  
if gpus:  
 try:  
 for gpu in gpus:  
 tf.config.experimental.set\_memory\_growth(gpu, True)  
 logical\_gpus = tf.config.experimental.list\_logical\_devices('GPU')  
 print(len(gpus), "Physical GPUs,", len(logical\_gpus), "Logical GPUs")  
 except RuntimeError as e:  
 print(e)  
  
  
tokenizer = Tokenizer()  
  
data = open('data/ppp.txt').read()  
  
corpus = data.lower().split("\n")  
  
tokenizer.fit\_on\_texts(corpus)  
total\_words = len(tokenizer.word\_index) + 1  
  
print(tokenizer.word\_index)  
print(total\_words)  
  
input\_sequences = []  
for line in corpus:  
 token\_list = tokenizer.texts\_to\_sequences([line])[0]  
 for i in range(1, len(token\_list)):  
 n\_gram\_sequence = token\_list[:i+1]  
 input\_sequences.append(n\_gram\_sequence)  
  
# pad sequences  
max\_sequence\_len = max([len(x) for x in input\_sequences])  
input\_sequences = np.array(pad\_sequences(input\_sequences, maxlen=max\_sequence\_len, padding='pre'))  
  
# create predictors and label  
xs, labels = input\_sequences[:,:-1],input\_sequences[:,-1]  
  
ys = tf.keras.utils.to\_categorical(labels, num\_classes=total\_words)  
  
model = Sequential()  
model.add(Embedding(total\_words, 240, input\_length=max\_sequence\_len-1))  
model.add((LSTM(150, return\_sequences=True)))  
model.add((LSTM(75)))  
model.add(Dense(total\_words, activation='softmax'))  
adam = Adam(learning\_rate=0.01)  
model.compile(loss='categorical\_crossentropy', optimizer=adam, metrics=['accuracy'])  
history = model.fit(xs, ys, epochs=100, verbose=1)  
model.save("lyricsirish.h5")  
  
seed\_text = "Pinocchio era"  
next\_words = 100  
  
for \_ in range(next\_words):  
 token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]  
 token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_len-1, padding='pre')  
 predicted\_probs = model.predict(token\_list, verbose=0)  
 predicted = np.argmax(predicted\_probs, axis=-1)[0]  
 output\_word = ""  
 for word, index in tokenizer.word\_index.items():  
 if index == predicted:  
 output\_word = word  
 break  
 seed\_text += " " + output\_word  
print(seed\_text)

**OUTPUT:**



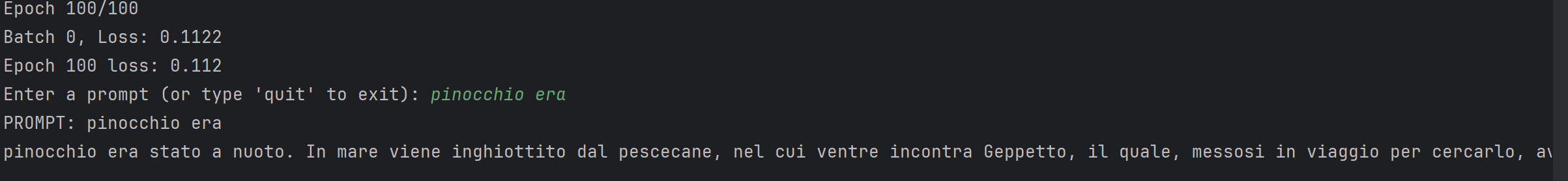
Spiegazione del codice:

***TRANSFORMER:***

import torch  
import torch.nn as nn  
import torch.optim as optim  
import math  
import torch.nn.functional as F  
from torch.utils.data import Dataset, DataLoader, random\_split  
from collections import Counter  
  
# Dataset Preparation  
with open('data/ppp.txt', 'r', encoding='utf-8') as file:  
 text = file.read()  
  
# Tokenize the text into words  
words = text.split()  
word\_counts = Counter(words)  
vocab = list(word\_counts.keys())  
vocab\_size = len(vocab)  
word\_to\_int = {word: i for i, word in enumerate(vocab)}  
int\_to\_word = {i: word for word, i in word\_to\_int.items()}  
  
SEQUENCE\_LENGTH = 64  
samples = [words[i:i + SEQUENCE\_LENGTH + 1] for i in range(len(words) - SEQUENCE\_LENGTH)]  
  
# Add OOV token handling  
OOV\_TOKEN = "<OOV>"  
word\_to\_int[OOV\_TOKEN] = vocab\_size  
int\_to\_word[vocab\_size] = OOV\_TOKEN  
  
  
def encode\_word(word):  
 return word\_to\_int.get(word, word\_to\_int[OOV\_TOKEN])  
  
  
class TextDataset(Dataset):  
 def \_\_init\_\_(self, samples, word\_to\_int):  
 self.samples = samples  
 self.word\_to\_int = word\_to\_int  
  
 def \_\_len\_\_(self):  
 return len(self.samples)  
  
 def \_\_getitem\_\_(self, idx):  
 sample = self.samples[idx]  
 input\_seq = torch.LongTensor([encode\_word(word) for word in sample[:-1]])  
 target\_seq = torch.LongTensor([encode\_word(word) for word in sample[1:]])  
 return input\_seq, target\_seq  
  
  
BATCH\_SIZE = 32  
dataset = TextDataset(samples, word\_to\_int)  
  
# Use only a subset of the dataset for quick testing  
dataset\_size = len(dataset)  
subset\_size = int(0.1 \* dataset\_size) # Use 10% of the dataset  
\_, subset = random\_split(dataset, [dataset\_size - subset\_size, subset\_size])  
dataloader = DataLoader(subset, batch\_size=BATCH\_SIZE, shuffle=True)  
  
print(f"Using a subset of size: {len(subset)} samples")  
print(subset[1])  
  
  
def generate\_square\_subsequent\_mask(sz):  
 mask = (torch.triu(torch.ones(sz, sz)) == 1).transpose(0, 1)  
 mask = mask.float().masked\_fill(mask == 0, float('-inf')).masked\_fill(mask == 1, float(0.0))  
 return mask  
  
  
class PositionalEncoding(nn.Module):  
 def \_\_init\_\_(self, max\_len, d\_model, dropout=0.1):  
 super(PositionalEncoding, self).\_\_init\_\_()  
 self.dropout = nn.Dropout(p=dropout)  
 pe = torch.zeros(max\_len, d\_model)  
 position = torch.arange(0, max\_len, dtype=torch.float).unsqueeze(1)  
 div\_term = torch.exp(torch.arange(0, d\_model, 2).float() \* (-math.log(10000.0) / d\_model))  
 pe[:, 0::2] = torch.sin(position \* div\_term)  
 pe[:, 1::2] = torch.cos(position \* div\_term)  
 pe = pe.unsqueeze(0)  
 self.register\_buffer('pe', pe)  
  
 def forward(self, x):  
 x = x + self.pe[:, :x.size(1)]  
 return self.dropout(x)  
  
  
class TextGen(nn.Module):  
 def \_\_init\_\_(self, vocab\_size, embed\_dim, num\_layers, num\_heads):  
 super(TextGen, self).\_\_init\_\_()  
 self.pos\_encoder = PositionalEncoding(max\_len=SEQUENCE\_LENGTH, d\_model=embed\_dim)  
 self.emb = nn.Embedding(vocab\_size + 1, embed\_dim) # +1 for OOV token  
 self.decoder\_layer = nn.TransformerDecoderLayer(d\_model=embed\_dim, nhead=num\_heads, batch\_first=True)  
 self.decoder = nn.TransformerDecoder(decoder\_layer=self.decoder\_layer, num\_layers=num\_layers)  
 self.linear = nn.Linear(embed\_dim, vocab\_size + 1)  
 self.dropout = nn.Dropout(0.2)  
  
 def forward(self, x):  
 emb = self.emb(x)  
 input\_mask = generate\_square\_subsequent\_mask(x.size(1)).to(x.device)  
 x = self.pos\_encoder(emb)  
 x = self.decoder(x, memory=x, tgt\_mask=input\_mask, memory\_mask=input\_mask)  
 x = self.dropout(x)  
 out = self.linear(x)  
 return out  
  
  
epochs = 100 # Reduce number of epochs for quick testing  
learning\_rate = 0.001  
device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
model = TextGen(vocab\_size=vocab\_size, embed\_dim=100, num\_layers=2, num\_heads=2).to(device)  
criterion = nn.CrossEntropyLoss()  
optimizer = optim.Adam(model.parameters(), lr=learning\_rate)  
print(model)  
  
total\_params = sum(p.numel() for p in model.parameters())  
print(f"{total\_params:,} total parameters.")  
total\_trainable\_params = sum(p.numel() for p in model.parameters() if p.requires\_grad)  
print(f"{total\_trainable\_params:,} training parameters.\n")  
  
# Print total sample and batch  
numero\_di\_campioni = len(subset)  
dimensione\_batch = BATCH\_SIZE  
numero\_batch = math.ceil(numero\_di\_campioni / dimensione\_batch)  
print(f"Numero totale di campioni nel dataset: {numero\_di\_campioni}")  
print(f"Dimensione del batch: {dimensione\_batch}")  
print(f"Numero totale di batch: {numero\_batch}")  
  
  
def train(model, epochs, dataloader, criterion, optimizer):  
 model.train()  
 for epoch in range(epochs):  
 running\_loss = 0  
 print(f"Epoch {epoch + 1}/{epochs}")  
 for batch\_idx, (input\_seq, target\_seq) in enumerate(dataloader):  
 input\_seq, target\_seq = input\_seq.to(device), target\_seq.to(device)  
  
 # Forward pass  
 outputs = model(input\_seq)  
  
 # Reshape target\_seq  
 target\_seq = target\_seq.contiguous().view(-1)  
 outputs = outputs.view(-1, vocab\_size + 1)  
  
 # Calculate loss  
 loss = criterion(outputs, target\_seq.view(-1))  
  
 # Backward pass and optimization  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 running\_loss += loss.item()  
  
 if batch\_idx % 10 == 0:  
 print(f"Batch {batch\_idx}, Loss: {loss.item():.4f}")  
  
 epoch\_loss = running\_loss / len(dataloader)  
 print(f"Epoch {epoch + 1} loss: {epoch\_loss:.3f}")  
  
  
# Run the training function  
train(model, epochs, dataloader, criterion, optimizer)  
def return\_int\_vector(text):  
 words = text.split()  
 input\_seq = torch.LongTensor([encode\_word(word) for word in words[-SEQUENCE\_LENGTH:]]).unsqueeze(0)  
 return input\_seq  
  
def sample\_next(predictions):  
 probabilities = F.softmax(predictions[:, -1, :], dim=-1).cpu()  
 next\_token = torch.argmax(probabilities)  
 return int(next\_token.cpu())  
  
def text\_generator(sentence, generate\_length):  
 model.eval()  
 sample = sentence  
 for i in range(generate\_length):  
 int\_vector = return\_int\_vector(sample)  
 if len(int\_vector) >= SEQUENCE\_LENGTH - 1:  
 break  
 input\_tensor = int\_vector.to(device)  
 with torch.no\_grad():  
 predictions = model(input\_tensor)  
 next\_token = sample\_next(predictions)  
 sample += ' ' + int\_to\_word[next\_token]  
 print(sample)  
 print('\n')

# Prompt  
while True:  
 user\_input = input("Enter a prompt (or type 'quit' to exit): ")  
 if user\_input.lower() == 'quit':  
 break  
 generate\_length = 100 # You can adjust this value as needed   
 print(f"PROMPT: {user\_input}")  
 text\_generator(user\_input, generate\_length)

**OUTPUT:**



Spiegazione del codice:

Con questi due esempio siamo riusciti a creare due piccoli modelli per la generazione del testo.

Nota: per il modello lstm ho utilizzato un seeb nel codice *seed\_text = "Pinocchio era"* , mentre per il modello transformer un pormpt per digitare il contesto.

E questo è tutto. Buon divertimento!

***LINK UTILI:***

### Python

* **Documentazione ufficiale di Python**: La documentazione ufficiale di Python è un'ottima risorsa per imparare e approfondire il linguaggio. Include tutorial, guide di riferimento, e documentazione dettagliata delle librerie standard. [Python 3.12.4 Documentation](https://docs.python.org/3.12/): https://docs.python.org/3.12/
* **Python Beginner's Guide**: Una guida per principianti per iniziare con Python. [Python Beginner's Guide](https://wiki.python.org/moin/BeginnersGuide): https://wiki.python.org/moin/BeginnersGuide

### Ambienti di Sviluppo

* **PyCharm**: PyCharm è un potente IDE per Python sviluppato da JetBrains. È disponibile in una versione gratuita (Community Edition) e una a pagamento (Professional Edition). [PyCharm](https://www.jetbrains.com/pycharm/): https://www.jetbrains.com/pycharm/
* **Visual Studio Code**: Un editor di codice gratuito e open source sviluppato da Microsoft, con supporto eccellente per Python attraverso estensioni. [Visual Studio Code](https://code.visualstudio.com/): https://code.visualstudio.com/

### Librerie Python Principali

* **math**: La libreria standard di Python per operazioni matematiche di base. [math — Mathematical functions](https://docs.python.org/3/library/math.html): https://docs.python.org/3/library/math.html
* **matplotlib**: Una libreria completa per la creazione di visualizzazioni statiche, animate e interattive in Python. [Matplotlib](https://matplotlib.org/): https://matplotlib.org/
* **TensorFlow**: Una libreria open source per il machine learning sviluppata da Google. [TensorFlow](https://www.tensorflow.org/): https://www.tensorflow.org/
* **os**: La libreria standard per interfacciarsi con il sistema operativo. [os — Miscellaneous operating system interfaces](https://docs.python.org/3/library/os.html): https://docs.python.org/3/library/os.html
* **scikit-learn**: Una libreria per il machine learning in Python, che offre semplici ed efficienti strumenti per l'analisi predittiva dei dati. Scikit-learn: https://scikit-learn.org/stable/
* **numpy**: Una libreria fondamentale per il calcolo scientifico con Python. Fornisce supporto per array di grandi dimensioni e matrici multidimensionali. [NumPy](https://numpy.org/): https://numpy.org/
* **pandas**: Una libreria per la manipolazione e l'analisi dei dati, che offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali. Pandas: https://pandas.pydata.org/
* **requests**: Una libreria semplice e elegante per effettuare richieste HTTP. Requests: https://docs.python-requests.org/en/master/
* **BeautifulSoup**: Una libreria per l'analisi di documenti HTML e XML. BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/
* **Flask**: Un microframework per lo sviluppo web leggero e semplice. Flask: https://flask.palletsprojects.com/
* **Django**: Un framework di alto livello per lo sviluppo web che incoraggia lo sviluppo rapido e un design pulito e pragmatico. [Django](https://www.djangoproject.com/): https://www.djangoproject.com/
* **PyTorch**: Una libreria open source per il deep learning sviluppata da Facebook's AI Research lab. [PyTorch](https://pytorch.org/): https://pytorch.org/
* **Keras**: Un'API di rete neurale di alto livello, scritta in Python e capace di funzionare sopra TensorFlow, CNTK, o Theano. [Keras](https://keras.io/): https://keras.io/
* **SciPy**: Una libreria usata per la matematica, la scienza e l'ingegneria. [SciPy](https://www.scipy.org/): https://www.scipy.org/
* **SymPy**: Una libreria per l'algebra simbolica. [SymPy](https://www.sympy.org/): https://www.sympy.org/
* **seaborn**: Una libreria per la visualizzazione dei dati basata su matplotlib, che fornisce un'interfaccia di alto livello per disegnare grafici statistici attraenti e informativi. Seaborn: https://seaborn.pydata.org/
* **Plotly**: Una libreria per la creazione di grafici interattivi e dashboard. Plotly: https://plotly.com/python/

Queste librerie coprono una vasta gamma di funzionalità e sono ampiamente utilizzate nella comunità Python per diversi tipi di applicazioni, dalla scienza dei dati allo sviluppo web e al machine learning.

***RINGRAZIAMENTI***

***INDICE:***