SuperDuper Py 02

Ciao youtubers e benvenuti in SuperDuper PY una serie di tutorial pensati per chi è all'inizio con Python e sta cercando un qualcosa che non sia il classico tutorial tipo addominali in 7 minuti o che spieghi ogni singolo comando senza dare esempi o idee.

In questo tutorial, andremo avanti con i concetti fondamentali di Python, andando a fondo con le operazioni sulle e vedremo come implementare una calcolatrice che ci permette di fare varie operazioni.

Una **stringa** è una sequenza di caratteri. In informatica e, in particolare, nella programmazione, una stringa rappresenta informazioni basate su testo. Un carattere in questo contesto è qualsiasi simbolo che può essere rappresentato, che può includere lettere, numeri, segni di punteggiatura, spazi e altri simboli.

Le stringhe sono uno dei tipi di dati fondamentali o primitivi come li abbiamo chiamati nel tutorial precedente. Mentre numeri come interi e numeri in virgola rappresentano una quantità, le stringhe rappresentano un testo.

1. **Virgolette singole**:

   ```
   stringa1 = 'Ciao, come stai?'
   ```

2. **Virgolette doppie**:

   ```
   stringa2 = "Benvenuto in Python!"
   ```

3. **Triple virgolette (singole o doppie)**: Queste sono spesso utilizzate per stringhe su più righe o per docstrings (documentazione all'interno del codice).

   ```
   stringa3 = '''Questo è un esempio
   di una stringa
   su più righe.'''
   stringa4 = """Un altro esempio
   di una stringa
   su più righe."""
   ```

Il fatto di poter scegliere il tipo di virgolette ci fa particolarmente comodo quando il testo contiene un apice stringa = "E' una bella giornata!" o se vogliamo inserire un testo virgolettato altro_esempio = 'Lui ha detto: "Ciao!"'.

Possiamo eseguire varie operazioni sulle stringhe. Come abbiamo visto le possiamo sommare o concatenare insieme semplicemente scrivendo nuova stringa = stringa1 + stringa2.

Ci sono varie operazioni di base come lower, upper e title che ci permettono di rendere tutto minuscolo, maiuscolo o ti mettere maiuscole tutte le lettere.

```
>>> titolo = "the quick brown fox jumps over the lazy dog"
>>> titolo.upper()
'THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG'
>>> titolo.title()
'The Quick Brown Fox Jumps Over The Lazy Dog'

>>> print(titolo)
the quick brown fox jumps over the lazy dog
```

Se dopo aver fatto queste operazioni provo a stampare la variabile titolo però, mi posso accorgere che non è cambiata. Questa è un'altra proprietà delle stringhe e si chiama immutabilità. Significa in pratica che posso fare qualunque operazione senza modificare il contenuto originale. per cambiarlo devo scrivere:

```
titolo = titolo.tile()
```

split(): ci torna utile tutte le volte che dobbiamo dividere una stringa, fra parentesi possiamo inserire il carattere che vogliamo usare per tagliare la frase, oppure lasciarlo vuoto che significa che deve tagliare la frase ogni volta che trova uno spazio.

```
frase = "Ciao, come stai?"
parole = frase.split()
print(parole)  # ['Ciao,', 'come', 'stai?']
```

Se faccio il type(parole) mi accorgo che ha creato una lista ovvero un tipo di variabile che può contenere più variabili. ogni variabile ha un indice e posso accedere ad ogni variabile semplicemente scrivendo la posizione quindi parole[0] mi darà ciao, parole[1] mi restituirà come e così via.

Possiamo fare varie operazioni sulle liste, per aggiungere una parola possiamo usare .append("bene") e vediamo che ha aggiunto la nostra nuova parola in coda. Per eliminare l'ultima cosa inserita possiamo usare pop().

Una cosa figa è che se ad esempio scriviamo parole[-1] ci restituisce l'ultima variabile che in questo caso è stai? perchè abbiamo cancellato "bene" usando pop().

Altre operazioni sono reverse che inverte il contenuto della lista, o sort che in questo caso mette le parole in ordine alfabetico. Count conta quanti valori ci sono count("come") restituisce quanti come sono presenti nella lista e ovviamente posso cambiare il contenuto della lista scrivendo parole[0] = Hello. E ovviamente se supero il numero massimo di elementi della lista scrivendo parola[20] ottengo un errore quindi anche qui, inutile dirlo siccome in caso di errore, il programma si interrompe, userò un try/except e se ci fate caso, quando ottengo un errore, l'interprete di python mi dice di che tipo è l'errore, in questo caso indexError, quindi lo posso usare per gestire l'errore scrivendo nell'exception indexError e nel caso in cui l'errore sia altro far stampare il tipo di errore.

Possiamo cambiare qualcosa nel testo con replace().

len(): Mostra come trovare la lunghezza di una stringa. Lo posso usare come type, quindi se scrive len(parole) mi dirà che è lunga 3.

```
nome = "Alice"
lunghezza = len(nome)
print(lunghezza)  # 5
```

Posso scrivere un loop in cui chiedo all'utente di scrivere un nome di almeno cinque lettere.

```
isWannaContinue = True
while isWannaContinue:
   nome = input("Inserisci un nome di almeno 5 lettere: ")
   if len(nome) >= 5:
       print(f"Il nome inserito è lungo {len(nome)} lettere.")
       isWannaContinue = False
   else:
       print(f"Il nome inserito è troppo corto. contiene solo
{len(nome)} lettere.")
```

Come vedete quello che faccio è sempre una comparazione rispetto a qualcosa, quindi controllo se un qualcosa è vero o falso o se è maggiore o minore di una certa quantità.

Ci sono vari tipi di comparatori in Python come `==`, `!=`, `<`, `>`, `<=`, `>=`. Ogni volta che faccio una comparazione ottengo un valore che è sempre o True o False.

Se scrivo ad esempio print(type(5==0)) ottengo False e posso cambiare operatore per accorgermi che mi darà un risultato diverso in base al tipo di comparazione.

Detto questo possiamo passare al primo esercizio di oggi dove creeremo una calcolatrice che permette all'utente di scrivere un calcolo tipo 1 + 2.

```
      question = input("Inserisci il calcolo: ")
      if "+" in question:
         splitQuestion = question.split("+")
         if len(splitQuestion) == 2:
             try:
                 returnString = f"{float(splitQuestion[0].strip()) +
float(splitQuestion[1].strip())}"
                 print(f"Il risultato di {question} = {returnString}")
             except ValueError:
                 print(f"L'input non è un numero valido. {question}")
```

Ovviamente posso inserire tutte le operazioni possibili e creare un loop, che è proprio quello che dovete fare, quindi vi invito a mettere in pausa il video e scrivere questo programma.

Sulla base delle informazioni che vi ho dato fin'ora, il risultato dovrebbe essere più o meno simile al mio. Se non lo è, non preoccupatevi, potete trovare il codice sulla pagina github, ho messo il link in descrizione.

```
isWannaContinue = True
while isWannaContinue:
   question = input("Inserisci il calcolo: ")
   if question.lower() == "exit":
       isWannaContinue = False
       break
   elif "+" in question:
       splitQuestion = question.split("+")
       if len(splitQuestion) == 2:
           try:
               returnString = f"{float(splitQuestion[0].strip()) +
float(splitQuestion[1].strip())}"
               print(f"Il risultato di {question} = {returnString}")
           except ValueError:
               print(f"L'input non è un numero valido. {question}")
   elif "-" in question:
       splitQuestion = question.split("-")
       if len(splitQuestion) == 2:
           try:
               returnString = f"{float(splitQuestion[0].strip()) -
float(splitQuestion[1].strip())}"
               print(f"Il risultato di {question} = {returnString}")
           except ValueError:
               print(f"L'input non è un numero valido. {question}")

   elif "*" in question:
       splitQuestion = question.split("*")
       if len(splitQuestion) == 2:
           try:
               returnString = f"{float(splitQuestion[0].strip()) *
float(splitQuestion[1].strip())}"
               print(f"Il risultato di {question} = {returnString}")
           except ValueError:
               print(f"L'input non è un numero valido. {question}")

   elif "/" in question:
```

```
        splitQuestion = question.split("/")
        if len(splitQuestion) == 2:
            try:
                returnString = f"{float(splitQuestion[0].strip()) /
float(splitQuestion[1].strip())}"
                print(f"Il risultato di {question} = {returnString}")
            except ValueError:
                print(f"L'input non è un numero valido. {question}")

    elif "^" in question:
        splitQuestion = question.split("^")
        if len(splitQuestion) == 2:
            try:
                returnString = f"{float(splitQuestion[0].strip()) **
float(splitQuestion[1].strip())}"
                print(f"Il risultato di {question} = {returnString}")
            except ValueError:
                print(f"L'input non è un numero valido. {question}")
```

L'idea è che per ogni operazione si ripete grossomodo la stessa operazione. Controllo che operatore ha fornito l'utente, se è fra i casi che ho previsto, splitto la frase e provo a eseguire l'operazione, se non ho problemi stampo il risultato, in caso contrario avverto l'utente che c'è un problema.

Quello che abbiamo fatto è stato fondamentalmente riscrivere lo stesso codice più volte. Esiste però un modo più efficiente per scrivere il codice, ovvero usare le funzioni.

Una funzione, non è altro che un blocco di codice che fa qualcosa e che posso richiamare tutte le volte che ho bisogno che venga fatta quella cosa.

Osservando quello che ho scritto, mi posso accorgere che quello che cambia fondamentalmente nelle varie operazione è il carattere che uso per fare lo split posso scrivere un qualcosa del tipo:

```
def checkOperation(splitCharacter, question):
"""
Controlla l'operazione da fare.
:param splitCharacter: il carattere da usare per dividere la domanda.
:param question: la domanda da controllare.
:return: il risulta dell'operazione o un messaggio di errore.
"""

    splitQuestion = question.split(splitCharacter)
    if len(splitQuestion) <= 2:
        try:
            if splitCharacter == "+":
                                    return  f"{float(splitQuestion[0].strip())  +
float(splitQuestion[1].strip())}"
            elif splitCharacter == "-":
                                    return  f"{float(splitQuestion[0].strip())  -
float(splitQuestion[1].strip())}"
            elif splitCharacter == "*":
```

```
                                return  f"{float(splitQuestion[0].strip())  *
float(splitQuestion[1].strip())}"
            elif splitCharacter == "/":
                                return  f"{float(splitQuestion[0].strip())  /
float(splitQuestion[1].strip())}"
            elif splitCharacter == "^":
                               return  f"{float(splitQuestion[0].strip())  **
float(splitQuestion[1].strip())}"
            else:
                return "operazione non valida"
        except ValueError:
            return f"L'input non è un numero valido. {question}"
```

In questo modo il codice finale diventa:

```
isWannaContinue = True
while isWannaContinue:
    question = input("Inserisci il calcolo: ")
    if question.lower() == "exit":
        isWannaContinue = False
        break
    elif "+" in question:
        print(checkOperation("+", question))
    elif "-" in question:
        print(checkOperation("-", question))
    elif "*" in question:
        print(checkOperation("*", question))
    elif "/" in question:
        print(checkOperation("/", question))
    elif "^" in question:
        print(checkOperation("^", question))
```

E’ un po' difficile da capire questo concetto, però se ci pensate bene, mi permette di poter aggiungere molte più opzioni e controllo al codice che sto scrivendo. Potrei ad esempio voler prevedere il fatto che al momento utilizzo solo un tipo di divisione /, ma l’utente potrebbe voler usare anche // che restituisce sempre un intero, oppure potrebbe scrivere ** al posto di ^ e questo alla fine è quello che fa il programmatore. Aggiunge possibilità all’utente in modo da rendere il programma sempre più utile.

Un esercizio che potreste fare, è quello di prevedere che l’utente possa memorizzare una variabile e richiamarla durante un’operazione. Abbiamo visto le liste, quindi possiamo immaginare che la nostra memoria sia una lista tipo memories = [1,2,3,4].

Supponiamo che l’utente scriva “mem”, potreste fare in modo che se l’utente scrive x = 2 o pi = 3.14, nella parte di calcolo, non restituisca un errore nel caso in cui uno scriva 5 * pi.

Un’altro modo per memorizzare i dati oltre alle liste, sono i dizionari: memories = {}. La particolarità dei dizionari è che è possibile associare un nome a un valore. Vi fa venire in mente qualcosa? Posso fare in modo che se

l’utente scrive x = 2 venga creato un dizionario tipo memories = {x:2, pi:3.14}.

Bene siamo giunti al termine di questo tutorial. Oltre a ricordarvi che in descrizione trovate il link dove trovare i codici che abbiamo usato nel tutorial, se non lo avete fatto vi invito a iscrivervi al canale e a mettere un like.

GoodByte! Alla prossima