lez_00

January 21, 2026

1 Benvenuti in python!

1.1 Le variabili

Python è un linguaggio a **tipizzazione dinamica**, questo significa che non è necessario specificare di volta in volta il tipo della variabile che si vuole definire/utilizzare. Questo ha un vantaggio e uno svantaggio:

- La sintassi è relativamente più semplice rispetto ad un linguaggio a tipizzazione statica (C, Java, ...)
- L'interprete deve, ogni volta che vuole accedere ad una variabile, assicurarsi di quale sia il tipo della variabile, con la conseguenza di un'esecuzione più lenta

Vediamo ora come utilizzare le variabili in python, iniziando con l'operazione di assegnazione di un valore a una variabile:

[52]:
```
intero = 1
decimale = 1.5
carattere = 'A'
stringa = 'Python'
```

Per definire una variabile ed assegnarle un valore è sufficiente utilizzare la sintassi di base: *nome_variabile = valore_variabile*

Sebbene una variabile in python non sia "costretta" ad assumere uno ed un solo tipo di valore, in un dato momento all'interno del programma ciascuna variabile conterrà un valore di un certo tipo.

È possibile conoscere il tipo di variabile che stiamo maneggiando con la funzione *type()*

[25]:
```
print(type(intero))
print(type(decimale))
print(type(carattere))
print(type(stringa))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'str'>
```

Vediamo quindi come ***intero*** sia una variabile di tipo ***int***, ***decimale*** una variabile di tipo ***float*** e sia ***carattere*** che ***stringa*** siano due variabili di tipo ***str***

Data la natura dinamica della tipizzazione in python, nulla ci impedisce di inizializzare una variabile con un tipo, e poi assegnarle un valore di un tipo totalmente diverso:

[26]:
```
numero = 1
print(numero, type(numero))
numero = 'Ciao'
print(numero, type(numero))
```

```
1 <class 'int'>
Ciao <class 'str'>
```

Notare come l'output della funzione *type()* parli di *class 'int'* e *class 'str'*. Vedremo il concetto di classi e programmazione ad oggetti più avanti.

> **NOTA:** è buona norma scrivere i nomi delle variabili in lettere minuscole, separando eventuali parole con un underscore '_'. Inoltre è importante dare a queste dei nomi significativi, in modo da poter capire al volo a cosa servono. Nulla ci impedisce di chiamare tutte le variabili 'a', 'b', 'sergio' e 'pippo', ma la lettura del codice risulta più ostica in questo caso.

Iniziamo ora ad esplorare i diversi tipi di variabile utilizzabili in python e alcune operazioni elementari che possono essere eseguite con e su di essi.

1.2 Stringhe

Le stringhe sono banalmente dei valori alfanumerici che possiamo identificare come testo. Una stringa è definita tra doppi apici (" ") o singoli apici (' '). Anche un singolo carattere è una stringa in python:

[27]:
```
stringa1 = 'Hello'
stringa2 = "World!"
```

Concatenazione È possibile **concatenare** due stringhe con l'operatore '+':

[28]:
```
stringa_concatenata = stringa1 + stringa2
print(stringa_concatenata)
```

```
HelloWorld!
```

E ovviamente è possibile concatenare anche più di due stringhe:

[29]:
```
stringa_lunga = stringa1 + ' ' + stringa2 + '!!!!'
print(stringa_lunga)
```

```
Hello World!!!!
```

Spesso ci interessa sapere quanto è lunga una stringa, ed è possibile ottenere questa informazione con la funzione *len()*:

[30]:
```
print(len(stringa_concatenata))
print(len(stringa_lunga))
```

```
11
15
```

La funzione *len()* prende in considerazione anche gli spazi.

Accesso con indici Possiamo accedere al singolo elemento di una stringa utilizzando indici numerici tra parentesi quadre:

[31]:
```
titolo = 'Il buono, il brutto e il cattivo'
print(titolo[0]) #primo carattere
print(titolo[6]) #settimo carattere
print(titolo[len(titolo) - 1 ]) #ultimo carattere
print(titolo[-1]) #ultimo carattere
print(titolo[-2])
```

```
I
n
o
o
v
```

Come vediamo dal codice, l'indicizzazione parte sempre da **zero**, ed arriva ad **n - 1** dove n è il numero di caratteri che compongono la stringa, spazi compresi.

Possiamo anche accedere ad una *slice* o sottostringa definendo un range di caratteri che ci interessano. La sintassi di base per questa operazione è *nome_stringa[estremo_1:estremo_2]*:

[32]:
```
print(titolo[3:8])
print(titolo[:13])
print(titolo[3:])
```

```
buono
Il buono, il
buono, il brutto e il cattivo
```

Notiamo come l'estremo superiore venga ignorato:

[33]:
```
print(titolo[3:8])
print(titolo[8])
```

```
buono
,
```

Il carattere in posizione 8 sarebbe la virgola, ma il nostro range si ferma in posizione 7. Inoltre se non specifichiamo l'estremo inferiore, lo slice di stringa che otterremo inizierà dal primo carattere, se non specifichiamo l'estremo superiore invece, lo slice sarà dal carattere di partenza fino alla fine della stringa.

> **Nota**: le stringhe sono un tipo di variabili definite *immutable*, e non possono essere modificate. Se una funzione modifica una stringa, in realtà ci sta ritornando una nuova variabile modificata. Infatti, se proviamo a modificare il carattere di una stringa otterremo un errore:

[34]:
```
stringa = 'Ciao'
stringa[2] = 'b'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[34], line 2
      1 stringa = 'Ciao'
----> 2 stringa[2] = 'b'

TypeError: 'str' object does not support item assignment
```

1.2.1 Metodi generali per le stringhe

Sia *len()* che *print()* sono due **funzioni**. Quelli che vedremo ora invece sono **metodi**. La differenza la comprenderemo quando vedremo la programmazione ad oggetti, ma per ora basti sapere che le funzioni sono definite 'globalmente' mentre i metodi sono definiti sulle classi. Per il momento l'unica differenza che noteremo è nel modo in cui funzioni e metodi vengono chiamati: - **Sintassi funzione**: *nome_funzione(argomenti_funzione)* - **Sintassi metodi**: *nome_oggetto.nome_metodo(argomenti_metodo)*

Nel nostro caso gli **oggetti** saranno **variabili che contengono stringhe**. Quindi apparterranno alla **classe 'Str'**

1.2.2 Modificare il *case* di una stringa

Vediamo tre metodi per modificare il case di una stringa: - stringa.upper() : Rende maiuscoli tutti i caratteri della stringa; - stringa.lower() : Rende minuscoli tutti caratteri della stringa; - stringa.swapcase(): Trasforma le maiuscole in minuscole, e viceversa.

[51]:
```
my_string = 'Bella Ciao'
print(my_string)
print(my_string.upper())
print(my_string.lower())
print(my_string.swapcase())
```

```
Bella Ciao
BELLA CIAO
bella ciao
bELLA cIAO
```

1.2.3 Caratteri di formattazione e metodo strip()

Alcuni caratteri speciali ci consentono di dare un minimo di formattazione ad una stringa. Quelli che vedremo sono **n** o carattere di **new line**, necessario per andare a capo, e **t** o carattere di tabulazione, che inserisce un numero predeterminato di spazi ed è importante per stampare tabelle in modo ordinato:

[50]:
```
print('Stringa originale:')
print('Ciao, mi chiamo Lorenzo')
print('Stringa con carattere \\n')
print('Ciao,\nmi chiamo Lorenzo')
```

```
Stringa originale:
Ciao, mi chiamo Lorenzo
Stringa con carattere \n
Ciao,
mi chiamo Lorenzo
```

[49]:
```
print('Numeri non tabulati:')
print('1 44 150 93 1101\n49 193 19383 18 1945')
print('Numeri tabulati con \\t')
print('1\t44\t150\t93\t1101\n49\t193\t19383\t18\t1945')
```

```
Numeri non tabulati:
1 44 150 93 1101
49 193 19383 18 1945
Numeri tabulati con \t
1       44      150     93      1101
49      193     19383   18      1945
```

Notare come sia possibile utilizzare il carattere di **escape** '\', per stampare un carattere speciale senza che venga interpretato da *print()*

Quando leggiamo del testo in input da un file, capita che la formattazione preveda dei caratteri speciali all'inizio o alla fine di una riga, e questi possono rappresentare un problema in base al tipo di operazione che vogliamo svolgere su quel testo. Il metodo *strip()* ci consente di rimuovere questi caratteri di formattazione **dall'inizio e dalla fine** una stringa:

[48]:
```
mystring = '\n\nHello\n\t\n'
print('Prima di strip()')
print(mystring)
print('Dopo strip()')
print(mystring.strip())
```

```
Prima di strip()


Hello


Dopo strip()
Hello
```

1.2.4 Cercare e contare occorrenze in una stringa

I metodi *count()* e *find()* ci consentono rispettivamente di contare le occorrenze di un carattere o di una sottostringa e di trovare una parola o un carattere all'interno di una stringa:

[47]:
```
stringa = 'Io sono Albus Silente'
print('Sto contando le "i"')
print(stringa.count('i'))
print('Sto contando le "I"')
print(stringa.count('I'))
print('Sto contando le "o"')
print(stringa.count('o'))
```

```
Sto contando le "i"
1
Sto contando le "I"
1
Sto contando le "o"
3
```

[46]:
```
stringa.find('Albus')
```

[46]:
```
-1
```

Il metodo find() restituisce l'indice del primo carattere dell'occorrenza trovata

[45]:
```
print(stringa[stringa.find('Albus')])
```

```
o
```

1.2.5 Convertire una variabile in una stringa

Se cerchiamo di concatenare una stringa con una variabile non-stringa (come ad esempio un intero), andremo incontro ad un errore (TypeError):

[44]:
```
print('Abito in via Rossi numero ' + 23)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 print(                                + 23)

TypeError: can only concatenate str (not "int") to str
```

Questo perché in generale non è possibile combinare due variabili di tipo diverso. Tuttavia la funzione *str()* ci consente di convertire un intero e altri tipi di variabili in una stringa:

[43]:
```
numero = 42
stringa_numerica = str(numero)
print('La risposta alla vita, all\'universo e a tutto quanto è ' + str(numero))
print('La risposta alla vita, all\'universo e a tutto quanto è ' +␣
 ↪stringa_numerica)
print(type(numero))
print(type(stringa_numerica))
```

```
La risposta alla vita, all'universo e a tutto quanto è 42
La risposta alla vita, all'universo e a tutto quanto è 42
<class 'int'>
<class 'str'>
```

1.2.6 Il metodo format()

Con il metodo *format()* è possibile creare un "modello" di stringa per poi riempirlo con dei valori. Vengono lasciati degli spazi vuoti, delimitati da parentesi graffe { }, che poi il metodo riempirà con le variabili che gli passeremo:

[41]:
```
modello = '{} abita in via {} numero {}'
nome = 'Mario Rossi'
via = 'Bianchi'
civico = 23

print(modello.format(nome, via, civico))
```

```
Mario Rossi abita in via Bianchi numero 23
```

È anche possibile specificare un nome per la variabile che inseriremo tra le parentesi graffe >NOTA: il nome della variabile inserita tra parentesi graffe non deve necessariamente coincidere con il nome della variabile che passeremo al metodo format(). L'esempio che segue dovrebbe chiarire questo aspetto

[39]:
```
modello2 = 'Il {animal} ha {number} zampe'
animale = 'Gatto'
zampe = 4
animale2 = 'Ragno'
zampe2 = 8

print(modello2.format(number = zampe, animal = animale))
print(modello2.format(animal = animale2, number = zampe2))
```

```
Il Gatto ha 4 zampe
Il Ragno ha 8 zampe
```

1.2.7 Le *f* string

Un altro metodo per formattare delle stringhe con delle variabili è utilizzare le cosiddette *f string*. Una *f string* viene dichiarata utilizzando il carattere " *f* " prima di aprire gli apici della stringa. Le posizioni che verranno occupate dalle variabili sono nuovamente indicate con parentesi graffe. Vediamo un esempio:

[38]:
```
name = 'Lorenzo'
age = 27
print(f'Mi chiamo {name} ed ho {age} anni')
```

```
Mi chiamo Lorenzo ed ho 27 anni
```

1.2.8 Prendere una stringa in input

Possiamo chiedere una stringa all'utente utilizzando la funzione *input()*. Questa funzione prende come argomento una stringa, che verrà stampata e mostrata all'utente per chiedergli cosa inserire. Anche se l'utente digitasse dei numeri, questi saranno processati come se fossero una stringa.

[37]:
```
cell = input('Inserisci il tuo numero di telefono: ')
print('Il numero inserito è: {}'.format(cell))
```

```
Inserisci il tuo numero di telefono:  123456
```

```
Il numero inserito è: 123456
```

1.2.9 Split() e Join()

split() e *join()* ci consentono di passare da una stringa ad una lista (vettore in python) e viceversa. Il metodo *split()* si applica ad una stringa e "taglia" in corrispondenza del carattere fornito come argomento:

[36]:
```
stringa = ('Cane,Gatto,Topo')
lista = stringa.split(',')
print(lista)
print(type(lista))
```

```
['Cane', 'Gatto', 'Topo']
<class 'list'>
```

Il metodo *join()* invece ci permette di creare una stringa contenente tutti gli elementi di una lista, separati da un carattere al quale applichiamo il metodo, mentre la lista da unire viene passata come argomento:

[35]:
```
lista = ['Giallo', 'Verde', 'Azzurro']
stringa = ','.join(lista)
print(stringa)
print(type(stringa))
```

```
Giallo,Verde,Azzurro
<class 'str'>
```

1.3 *Esercizi*

Per risolvere alcuni di questi esercizi sarà necessario un po' di Google-Fu, in quanto non tutti i metodi o le funzioni necessari a risolverli sono stati presentati durante la lezione.

1. Definire una stringa con parole separate da virgole e non da spazi, per poi sostituire le virgole con gli spazi IN UN SOLO PASSAGGIO. Cercare online il metodo o la funzione più appropriata per farlo. Esempio: "Ciao,sono,io" -> "Ciao sono io" > **NOTA:** si impara più dai fallimenti che dai successi
2. Chiedere due stringhe in input all'utente, concatenarle e poi stampare la lunghezza della stringa risultante.

3. Creare un modello da riempire successivamente con delle informazioni, chiedere le informazioni all'utente, riempire il modello e stamparlo. Per esempio, vogliamo un output di questo tipo:

Ragione sociale: <nome>Parita IVA: <numero>

4. Data la sequenza 'AATCGGCTTAGCTATATTCGGCGCGCTATATATATCGGCTAGCGATTCATGCTACTAGCTGACTG', contare le occorrenze di ciascun nucleotide e calcolare il contenuto di GC, espresso come (Numero di G e C / Numero totale di basi)

1.3.1 Soluzioni

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

..

1.4 Esercizio 1:

[ ]:
```
stringa = 'Ciao,mi,chiamo,Lorenzo'
stringa_ok = stringa.replace(',', ' ')
print(stringa_ok)
```

1.5 Esercizio 2

[ ]:
```
s1 = input('Dammi una stringa: ')
s2 = input('Dammi un\'altra stringa: ')
s3 = s1 + s2
print(s3, len(s3))
```

1.6 Esercizio 3 v1

[ ]:
```
modello = 'Ragione sociale: {}\ncon Partita IVA No: {}'
rs = input('Inserire la Ragione sociale dell\'azienda: ')
pi = input('Inserire la partita IVA')
print(modello.format(rs, pi))
```

1.7 Esercizio 3 v2

[ ]:
```
rs = input('Inserire la Ragione sociale dell\'azienda: ')
pi = input('Inserire la partita IVA')
print(f'Ragione sociale: {rs}\ncon Partita IVA No: {pi}')
```

1.8 Esercizio 4

[ ]:
```
seq = 'AATCGGCTTAGCTATATTCGGCGCGCTATATATATCGGCTAGCGATTCATGCTACTAGCTGACTG'
n_a = seq.count('A')
n_t = seq.count('T')
n_c = seq.count('C')
n_g = seq.count('G')
gc_content = ((n_c + n_g) / len(seq))

print(f'Numero di A: {n_a}')
print(f'Numero di T: {n_t}')
print(f'Numero di C: {n_c}')
print(f'Numero di G: {n_g}')
print(f'Contenuto in GC: {gc_content}')
```