Python: Tipi, variabili e operazioni principali

Rights & Credits

Il notebook è stato variamente trasformato, anche con l'aggiunta di contenuti, da Agostino Migliore, a partire dall'originale di:

- Simone Campagna (CINECA)
- Mirko Cestari (CINECA)
- Nicola Spallanzani (CNR-NANO)

Tipi intrinseci e operazioni di base

interi (*integers*)

In python gli interi hanno precisione arbitraria. Il loro valore è limitato solo dalla RAM del computer in uso. Per esempio, possiamo calcolare quantità così grandi come 2**1024 (vedi sotto).

In [1]:
```
2**1024
```

Out[1]:
```
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
```

Il tipo di un oggetto numero intero è **int**, come si può vedere applicando la funzione intrinseca di Python `type()` vista nella prima lezione a un qualsiasi numero intero:

In [2]:
```
type(7)
```

Out[2]:
```
int
```

numeri a virgola mobile (*floating point numbers*)

Un numero a virgola mobile (floating point) è un numero positivo o negativo con un punto decimale. Il loro tipo è **float**.

In [3]:
```
2.0**-1024
```

Out[3]: 5.562684646268003e-309

In [5]:
```
type(2.0)
```

Out[5]: float

Si noti che:

- l'operazione di potenza è rappresentata dal simbolo ** e non ^;
- se avessimo scritto le due istruzioni di sopra in una stessa cella e le avessimo eseguite, avremmo ottenuto in output solo il risultato della seconda istruzione.

In [7]:
```
2.0**-1024
type(2.0)
```

Out[7]: float

La prima riga di comando viene eseguita, ma il risultato non viene mostrato. Visualizziamo, invece, i risultati di entrambe le righe di comando eseguendo la seguente `tupla` di comandi (una tupla è una sequenza di oggetti separati da virgole; approfondiremo dopo la descrizione di tale **contenitore** di oggetti).

In [9]:
```
2.0**-1024, type(2.0)
```

Out[9]: (5.562684646268003e-309, float)

Mentre i numeri reali hanno una precisione infinita e sono pertanto continui, i numeri in virgola mobile (che ne costituiscono un'approssimazione finita) hanno un numero finito di cidre decimali e quindi hanno una precisione limitata. Così,

In [12]:
```
2.0**1023
```

Out[12]: 8.98846567431158e+307

mentre

In [14]:
```
2.0**1024
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Cell In[14], line 1
----> 1 2.0**1024

OverflowError: (34, 'Result too large')
```

Divisione di interi

La divisione di interi produce un float anche quando il risultato potrebbe essere scritto come un intero perchè non contiene cifre decimali significative:

In [15]:
```
10 / 2
```

Out[15]:
```
5.0
```

Se si vuole una divisione (tra interi o float) troncata che produca un intero, si può usare la divisione con troncamento, denotata da //:

In [16]:
```
10 // 2
```

Out[16]:
```
5
```

In [19]:
```
10 // 4
```

Out[19]:
```
2
```

In [20]:
```
12.0 // 5.0
```

Out[20]:
```
2.0
```

divmod

Per avere sia il quoziente che il resto (cioè, il risultato dell'operatore `modulo di una divisione` o `mod`) di una divisione tra interi, si può usare la funzione divmod(), i cui parametri sono il dividendo e il divisore: `divmod(dividendo, divisore)`.

In [21]:
```
divmod(13,4)
```

Out[21]:
```
(3, 1)
```

Tale funzione può essere usata anche con float.

In [22]:
```
divmod(15.1,4.0)
```

Out[22]:
```
(3.0, 3.0999999999999996)
```

numeri complessi

I numeri complessi sono del tipo **complex**. L'unità immaginaria è indicata con `j` (come si fa nella teoria dei circuiti per non confonderla con la corrente). Sotto mostriamo alcuni esempi con due numeri complessi.

In [23]:
```
w = 3.0 - 4.0j
z = 3.0 + 4.0j
z+w, z*w, z/w
```

Out[23]:
```
((6+0j), (25+0j), (-0.28+0.96j))
```

Come detto sopra:

In [24]:
```
type(z)
```

Out[24]:
```
complex
```

Le parti reale e immmaginaria di un numero complesso sono suoi attributi e possono essere richiesti (per uso in un codice o, come sotto, semplicemente per essere stampati) usando la già vista notazione punto (*dot-notation*):

In [26]:
```
print(z.real, z.imag)
```

```
3.0 4.0
```

La funzione intrinseca `abs()` di Python ci restituisce il modulo di un numero complesso (e, chiaramente, anche quello di un float come caso particolare):

In [28]:
```
abs(z)
```

Out[28]:
```
5.0
```

In [29]:
```
abs(-3.4)
```

Out[29]:
```
3.4
```

Per ottenere il complesso coniugato di un numero complesso si può usare il metodo `conjugate()`:

In [75]:
```
z.conjugate()
```

Out[75]:
```
(3-4j)
```

variabili (*variables*)

Abbiamo già discusso ampiamente il significato di variabile.

In [30]:
```
a = 5
b = 3.2
c = a
C = b # it's not "c", it's another variable
```

Si noti che quanto scritto dopo il simbolo # viene interpretato come un commento e quindi non inficia il codice.

Utilizzando l'operatore di eguaglianza (==) possiamo verificare che `c` e `C` non sono la stessa cosa:

In [31]:
```
C == c
```

Out[31]:
```
False
```

Abbiamo detto nella prima lezione che una variabile è un riferimento, un nome che punta alla specifica posizione in memoria di un valore, che è l'oggetto. Vi è, tuttavia, una sottile differenza tra i concetti di variabile e riferimento.

Per spiegare tale punto, facciamo uso della funzione intrinseca `id()`, che fornisce l'"identità" di un oggetto, cioè un numero facente riferimento ad una locazione di memoria che rimane immutato per tutta la vita dell'oggetto.

In [32]:
```
id(a)
```

Out[32]:
```
140703768560552
```

Possiamo cambiare il valore associato ad `a` nel corso di un programma, per esempio come segue.

In [33]:
```
a = a + 4
a
```

Out[33]:
```
9
```

Si noti che possiamo ottenere in output il valore di `a` sia usando `print` che invocando direttamente `a`. Possiamo ora vedere che

In [34]:
```
id(a)
```

Out[34]:
```
140703768560680
```

è cambiato rispetto a prima. Adesso abbiamo davvero usato `a` come una variabile, mentre prima `a` era un nome che fungeva semplicemente da riferimento a un valore numerico dato.

operatori (*operators*)

Ecco gli operatori più comuni disponibili:

- + (somma di numeri o concatenazione di stringhe)
- - (sottrazione)
- * (prodotto)
- / (divisione)
- // (divisione con troncamento)

- % (resto o modulo)
- ** (elevamento a potenza)

operatori doppi

Tali operatori consentono di eseguire simultaneamente una operazione e un'assegnazione. Essi sono:

- += (aumenta)
- -= (dominuisce)
- ...

Per esempio,

In [35]:
```
a = 10
a += 3
a
```

Out[35]:
```
13
```

Nella seconda istruzione di sopra, `+=` aggiunge `4` al valore iniziale di `a` e assegna il risultato dell'operazione ad `a` stessa.

stringhe (*strings*)

Una stringa è un insieme ordinato (chiaramente, cambiando l'ordine dei caratteri si ottiene una stringa diversa) di caratteri racchiuso, secondo la sintassi di Python, da virgolette singole ('...') o doppie ("...") Il loro tipo è denominato **str**. Si noti che una specifica stringa di per sé, per esempio `"Hello"`, è un *literal* (come lo è il numero `3` o un altro numero) e quindi una costante. Loperazione + diventa una concatenazione quando agisce fra stringhe:

In [36]:
```
"alpha" + 'bet'
```

Out[36]:
```
'alphabet'
```

In [40]:
```
"Hi " + "John!"
```

Out[40]:
```
'Hi John!'
```

In [42]:
```
"Hi " "John!"
```

Out[42]:
```
'Hi John!'
```

`print()` mostra il contenuto di una stringa, quindi senza gli apici.

In [43]:
```
print("Hi " "John!")
```

```
Hi John!
```

stringhe su più linee

Le virgolette triple sono usate per creare stringhe che si estendono su più righe e anche nel caso in cui una stringa contenga delle virgolette singole o doppie che fanno parte della stringa e devono quindi essere interpretate come caratteri (a differenza delle virgolette esterne che identificano il tipo di oggetto come una stringa). Per esempio,

In [45]:
```
a = """This sentence spans two lines and
contains 'single quotes' and "double quotes"."""
print(a)
```

```
This sentence spans two lines and
contains 'single quotes' and "double quotes".
```

o, equivalentemente,

In [46]:
```
a = '''This sentence spans two lines and
contains 'single quotes' and "double quotes".'''
print(a)
```

```
This sentence spans two lines and
contains 'single quotes' and "double quotes".
```

Caratteri (o sequenze) speciali in stringhe

Le stringhe possono anche contenere caratteri speciali (o meglio, *sequenze speciali* contenenti uno slash e almeno un carattere, dette in inglese *escape sequences*) che non sono stampati, ma controllano il modo in cui gli altri caratteri veri e propri della stringa vengono stampati:

- \n a capo
- \t tab
- \ include uno slash \ come carattere vero e proprio e non come parte di una *escape sequence*.

In [54]:
```
print("alpha\tbeta\ngamma\t\\nome")
```

```
alpha	beta
gamma	\nome
```

Come facciamo allora se, per esempio, vogliamo scrivere letteralmente `\n` in una stringa, senza che venga interpretata come una *escape sequence*? Lo si può fare premettendo il prefisso `r` o `R` alla stringa. La *r* deriva dalla parola inglese *raw*, che significa *crudo*, *greggio*, *nudo e crudo*, con riferimento al voler rendere la stringa per come è, senza alcuna interpretazione di caratteri speciali.

In [55]:
```
print(r"alfa\nbeta\tgamma")
```

```
alfa\nbeta\tgamma
```

Operazioni su stringhe

Ci sono varie operazioni (metodi) che si possono applicare alle stringhe. Esse non modificano la stringa assegnata originariamente, ma restituiscono una nuova stringa con modifiche rispetto a quella originaria. Esaminiamo un primo esempio.

In [65]:
```
s = "Hello world!"
s
```

Out[65]:
```
'Hello world!'
```

Con l'istruzione di sopra abbiamo assegnato il valore "Hello, world!" ad `s`. Adesso agiamo su `s` come segue:

In [61]:
```
s.lower()
```

Out[61]:
```
'hello world!'
```

Abbiamo visto nella prima lezione che una certa classe viene definita con un insieme di attributi che descrivono le proprietà degli oggetti che ne fanno parte e funzioni, chiamate *metodi*, per operare su tali oggetti e così manipolarli in tutti i modi consentiti dalla definizione della classe. Qui `s` è l'oggetto e `lower()` è un metodo che si applica ad `s` (più precisamente, all'oggetto chiamato `s`) usando la dot-notation. Inoltre, tale metodo non prende argomenti dentro le parentesi. Come vedete, tale metodo rende tutte le lettere minuscole. Come anticipato, il contenuto di `s` non è stato variato. Infatti:

In [66]:
```
s
```

Out[66]:
```
'Hello world!'
```

Se vogliamo riutilizzare l'espressione can caratteri tutti minuscoli altrove in un programma, dobbiamo assegnare un nome al risultato dell'operazione di sopra; per esempio,

In [68]:
```
t = s.lower()
t
```

Out[68]:
```
'hello world!'
```

Adesso possiamo far riferimento a `t` tutte le volte che ci serve l'espressione con caratteri minuscoli in un codice:

In [69]:
```
u = "John said: " + t + " I am happy today!"
print(u)
```

```
John said: hello world! I am happy today!
```

In [70]:
```
s.upper()
```

Out[70]:
```
'HELLO WORLD!'
```

In [71]:
```
s.title()
```

Out[71]:
```
'Hello World!'
```

In [73]:
```
v = s.replace('o', 'x')
s, v
```

Out[73]:
```
('Hello world!', 'Hellx wxrld!')
```

Per trovare la posizione alla quale si trova un carattere o inizia una sequenza di caratteri (una *sottostringa*) in una stringa, si usa il metodo `find()`. La posizione viene contata a partire da 0, cioè considerando come posizione 0 quella del primo carattere della stringa. Gli spazi vuoti sono anch'essi caratteri e quindi inclusi nel conteggio.

In [75]:
```
s.find('orl')
```

Out[75]:
```
7
```

Se ci sono più caratteri o sequenze di caratteri uguali all'argomento passato a `find()`, viene considerata la posizione del primo:

In [76]:
```
s.find('o')
```

Out[76]:
```
4
```

I metodi si possono anche applicare all'oggetto stesso, senza assegnarlo ad una variabile:

In [77]:
```
"Hello world!".upper()
```

Out[77]:
```
'HELLO WORLD!'
```

La funzione intrinseca `len()` fornisce la lunghezza della stringa, cioè il numero di caratteri che essa contiene:

In [79]:
```
len(s)
```

Out[79]:
```
12
```

Si noti che tale funzione si può applicare ad altri oggetti, come la tupla `(1, 3, 4)`, di cui conta il numero di elementi:

In [80]:
```
len((1, 3, 4))
```

Out[80]:
```
3
```

Come "affettare" stringhe

Si può accedere a singoli caratteri di una stringa e, tramite operazioni di affettamento (*slicing*), anche a sottostringhe.

In [3]:
```
h_f = "Hi folks!"
#      012345678 sono le posizioni dei caratteri di sopra, qui indicate
# per semplificare la lettura delle istruzioni seguenti.
```

In [82]:
```
h_f[0]
```

Out[82]:
```
'H'
```

In [84]:
```
h_f[4]
```

Out[84]:
```
'o'
```

Si può procedere all'indietro usando numeri negativi nelle parentesi quadre. Per esempio,

In [85]:
```
h_f[8]
```

Out[85]:
```
'!'
```

e la stessa cosa si ottiene come segue:

In [87]:
```
h_f[-1]
```

Out[87]:
```
'!'
```

Continuando,

In [88]:
```
h_f[-2], h_f[-3], h_f[-4]
```

Out[88]:
```
('s', 'k', 'l')
```

Se si eccede il numero di posizioni disponibili, si ottiene un errore di indice (*index error*):

In [64]:
```
h_f[13]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[64], line 1
----> 1 h_f[13]

IndexError: string index out of range
```

In [65]:
```
h_f[2:]
```

Out[65]:
```
' folks!'
```

Quella di sopra è una fetta estesa (*extended slice*). L'operazione di slicing indica di mantenere i caratteri a partire dalla posizione 2, quindi escludendo `H` (posizione 0) ed `i` (posizione 1). Al contrario:

In [66]:
```
h_f[:2]
```

Out[66]:
```
'Hi'
```

Notate, però, che lo spazio vuoto alla posizione `2` non è incluso. Infatti, le due fette sono complementari e quindi

In [68]:
```
h_f[:2] + h_f[2:]
```

Out[68]:
```
'Hi folks!'
```

restituisce la stringa intera. Per ottenere una stringa con i caratteri dalla posizione 2 inclusa alla posizione 4 (ossia alla posizione 5 esclusa):

In [4]:
```
h_f[2:5]
```

Out[4]:
```
' fo'
```

Si noti il seguente messaggio di errore:

In [5]:
```
h_f[2] = 'x'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[5], line 1
----> 1 h_f[2] = 'x'

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

Non si può modificare la stringa (assegnata ad `s`) assegnado a una fetta di essa un diverso carattere o insieme di caratteri.

Spezzettare e ricongiungere una stringa

Possiamo spezzettare una stringa (nel senso che ne creiamo una nuova spezzettando il contenuto di quella originaria) come segue.

In [86]:
```
p = "alpha:beta:epsylon:gamma"
p.split(":")
```

Out[86]:
```
['alpha', 'beta', 'epsylon', 'gamma']
```

Il risultato è un *contenitore* (in questo caso un contenitore di stringhe) chiamato lista, che vederemo dopo.

Il metodo `split()` prende come argomento un set di caratteri (cioè una stringa) che funziona da separatore. Ogni volta che incontra tale separatore nella stringa originaria `p`, spezza il contenuto: i caratteri tra due spezzature successive costituiscono un elemento della lista di output.

In [87]:
```
p = "alpha:beta:epsylon:gamma"
p.split("a:")
```

Out[87]:
```
['alph', 'bet', 'epsylon:gamma']
```

Ovviamente,

In [88]:
```
p
```

Out[88]:
```
'alpha:beta:epsylon:gamma'
```

ma possiamo asseggnare il risultato dello splitting (scissione, separazione) ad un riferimento o varabile e poi usarlo altrove:

In [89]:
```
q = p.split("a:")
print(q)
```

```
['alph', 'bet', 'epsylon:gamma']
```

Al contrario dello splitting,

In [95]:
```
':'.join(['alpha', 'beta', 'epsyolon', 'gamma'])
```

Out[95]:
```
'alpha-beta-gamma'
```

cioè i vari pezzi sono riuniti. Potevamo anche inserire un altro carattere o insieme di caratteri nel mezzo, per esempio `" - "`:

In [96]:
```
' - '.join(['alpha', 'beta', 'epsyolon', 'gamma'])
```

Out[96]:
```
'alpha - beta - epsyolon - gamma'
```

bool

Infine, esiste il tipo **bool** (da Boolean*), che ha solo due valori o oggetti: True e False. Tale tipo di dato può essere usato, per esempio, come risultato di operazioni confronto (lo abbiamo visto come risultato di `age >= 18` nel programmino della prima lezione).

None

In Python esiste un oggetto speciale che è l'unico del suo tipo, None. Denota un valore indefinito o la mancanza di valore, la non esecuzione di una istruzione, il fatto che tutto va

bene e non vi è "nulla da dichiarare" in un test, ecc.

Esercizi

- Definite una stringa che contiene il vostro nome e una che contiene il vostro cognome.
- Stampate la lunghezza di entrambe le stringhe.
- Concatenate le due stringhe, con uno spazio bianco in mezzo, e assegnate il risulato alla variabile name_surname.
- Stampate la lunghezza di name_surname.