Tipi numerici

Python gestisce diversi formati numerici.

- int - è il formato standard per la gestione dei numeri interi.

```
x = 42
type(x) # -> int
```

- float - è il formato standard per la gestione dei numeri con la virgola

```
x = 3.141
type(x) # -> float
```

Altri formati: long, complex

```
Esempi
n = 3.14 # assegna a n il valore 3.14
m = n # assegna a m il valore riferito da n (3.14)
n = 2**3 # assegna 8 alla variabile n (m vale ancora 3.14)
n = n + 1 # incrementa n di 1
n = float(n) # converte n in formato float
```

Tipo str

Python memorizza sequenze di caratteri in oggetti di tipo str (string).

Operazione +. La somma di due oggetti str s1, s2 è un nuovo oggetto str dato dalla concatenazione dei caratteri di s1 e s2

Operazione *. Il prodotto tra un intero *n* e un oggetto str *s* è un nuovo oggetto str dato dalla sequenza contenuta in *s* ripetuta *n* volte

```
Esempi
s1 = "ciao"
s2 = "mondo"
s3 = s1 + " " + s2 # s3 -> "ciao mondo"
s4 = "!?"*4 # s4 -> "!?!?!?!?"
s3 + s4 # -> "ciao mondo!?!?!?!?"
```

Esercizi di riepilogo (Tipi Atomici)

Esercizio. Dati x = 1, y = 3, z = 0

- calcolare la somma tra x e y e salvare il risultato in z
- porre x uguale a y
- incrementare y di 2

Quanto vale x?

- calcolare il prodotto tra y, z e x e salvare il risultato in z

Quanto vale z?

- decrementare y di 1

E ora?

- convertire y in formato *float*
- salvare in x il risultato di $z^{1/y}$

Verificare che le prime tre cifre decimali di x valgono 783

Bonus. Cosa accade se si omette la conversione di y a float?

In [6]:
```
x = 1
y = 3
z = 0
z = x + y
x = y
y += 2
x # -> 3
z *= x*y
z # -> 60
y -= 1
z # -> 60
y = float(y)
x = z**(1/y)
x
```

Out[6]: 2.7831576837137404

Esercizio. (Precedenze) Calcolare le seguenti espressioni per x = 1, x = 5

$2x + 8\,\frac{4^2}{2}$

$2x + 4^{1/2}$

In [10]:
```
x = 1
2*x + 8 * 4**2 / 2 # -> 66
x = 5
2*x + 8 * 4**2 / 2
```

Out[10]: 74

In [13]:
```
x = 1
2*x + 4**(1.0/2) # -> 4.0
x = 5
2*x + 4**(1.0/2)
```

Out[13]: 12.0

Esercizio. (Precedenze) Calcolare le seguenti espressioni per x = 1, y = 2

$2x + 4^{x/y}$

In [15]:
```
x = 1
y = 2
2*x + 4**(float(x)/y)
```

Out[15]: 4.0

Esercizio (Conversioni).

Dato un oggetto di tipo numerico contenente una temperatura in gradi Celsius, determinare la temperatura equivalente in gradi Farenheit per i valori:

- -273.15,

- 0,
- 36,
- 100

Nota: $t_F = \frac{9}{5} t_C + 32$

In [22]:
```
temp_c = -273.15
temp_f = 9.0/5 * temp_c + 32
temp_f # -459.6699...
temp_c = 0
temp_f = 9.0/5 * temp_c + 32
temp_f
# ...
```

Out[22]: 32.0

Esercizio. Date le assegnazioni (operazioni su str)

- a = 'fa',
- b = 'la ',
- c = 'vo'

scrivere un'espressione che generi la stringa

```
"la favola vola vola vola "
```

usando solo a,b,c

In [23]:
```
a = 'fa'; b = 'la '; c = 'vo'
b + a + 4*(c + b)
```

Out[23]: 'la favola vola vola vola '

Esercizio. (assegnazioni e operazioni su str) Date le assegnazioni

- a = "six",
- b = a,
- c = " > ",
- d = "ty",
- e = "1",

modificare le variabili usando SOLO i valori di a,b,c,d,e in modo che valga

```
a + c + e + b # -> "sixty > 11 > six"
```

In [24]:
```
# soluzione
a = "six"; b = a; c = " > "; d = "ty"; e = "1"
a += d
e *= 2 # equivalente a e = 2*e
e += c
a + c + e + b
```

Out[24]: 'sixty > 11 > six'

Sequenze e indicizzazione

Liste

In Python, una lista rappresenta una sequenza ordinata di oggetti (possibilmente eterogenei).

Come con tutte le sequenze (list,str,tuple) è possibile

- accedere agli elementi di una lista tramite l'operatore di indicizzazione *[]*
- ottenere una sottolista di una lista attraverso l'operatore di slicing *[begin:end]*
- concatenare due liste ottenendo una nuova lista (operatore +)
- generare un oggetto lista in cui si ripetono *n* volte gli elementi di una lista *l* (operatore *)

Inoltre:

- gli oggetti di una lista possono essere sostituiti
- la lunghezza di una lista può essere modificata aggiungendo o rimuovendo elementi.

```
Esempi
l = [6, "due", 8.6, "quattro"]
l[-1]*2 + l[1] # -> "quattroquattrodue"
h = l[0:3]
h[0] = "uno" # h -> ["uno","due",8.6], l -> [6, "due", 8.6, "quattro"]
del h[2] # h -> ["uno","due"], l -> [6, "due", 8.6, "quattro"]
h.extend([5,6,"sette"]) # h -> ["uno","due",5,6,"sette"], l -
> [6, "due", 8.6, "quattro"]
```

Esercizio. Data a = [1,2,3] si modifichi *a* in modo da ottenere

```
a = [4,2,3]
```

```
In [26]: a = [1,2,3]
         a[0] = 4
         a

Out[26]: [4, 2, 3]
```

Esercizio. Data a = [1,2,3,4,5], utilizzare l'operatore di slicing e l'assegnazione in modo da ottenere

```
a = [3,4,5]
```

```
In [25]: a = [1,2,3,4,5]
         a = a[2:] # sottolista di a dall'elemento in posizione 2, fino all'ultimo elemento
         a

Out[25]: [3, 4, 5]
```

Mutabilità e Immutabilità

- Data una variabile *obj* La funzione *id(obj)* restituisce un identificativo univoco per l'oggetto a cui *obj* si riferisce.
- Date due variabili *obj1* e *obj2*, il costrutto *is* restituisce *True* se *obj1* e *obj2* si riferiscono allo stesso oggetto. Restituisce *False* altrimenti.

Quando assegnamo (un riferimento ad) una lista ad una variabile *l1*, id(l1) identifica la lista riferita da *l1*. L'istruzione

```
l2 = l1
```

assegna a l2 lo stesso riferimento di l1 (l1 e l2 puntano alla stessa lista). Quindi modificando (la lista riferita da) l1, modifichiamo anche (la lista riferita da) l2, e viceversa. Se questo non è il comportamento desiderato, possiamo clonare l1 tramite l'operatore di slicing:

```
l2 = l1[:]
```

Ora l2 e l1 si riferiscono a due oggetti list diversi. Possiamo verificarlo confrontando id(l1) e id(l2)

Esercizio. Data la lista a = [1,2,3] creare *b* a partire da *a* in modo che, rimuovendo il secondo elemento di *b* si abbia

```
a = [1,2,3] e b = [1,3]
```

In [27]:
```
a = [1,2,3]
b = a[:]
del b[1]
a,b
```

Out[27]: ([1, 2, 3], [1, 3])

Esercizio. Data la lista a = [1,2,3] creare *b* a partire da *a* in modo che, rimuovendo il secondo elemento di *b* si abbia

```
a = [1,3] e b = [1,3]
```

In [28]:
```
a = [1,2,3]
b = a
del b[1]
a,b
```

Out[28]: ([1, 3], [1, 3])

Esercizio. Dati a = [1,2,3], b = a, si modifichi *a* in modo da ottenere

```
a = [1,2,3,4,5,6] e b = [1,2,3]
```

In [29]:
```
a = [1,2,3]
b = a
a = a + [4,5,6]
a,b
```

Out[29]: ([1, 2, 3, 4, 5, 6], [1, 2, 3])

Esercizio. Dati a = [1,2,3], b = a, si modifichi *a* in modo da ottenere

```
a = [1,2,3,4,5,6] e b = [1,2,3,4,5,6]
```

In [31]:
```
a = [1,2,3]
b = a
a.extend([4,5,6])
a,b
```

Out[31]: ([1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6])

tuple

In Python una tupla rappresenta una *sequenza* ordinata di oggetti **NON SOSTITUIBILI**

```
Esempi
x = 2.7
```

```
t = ("parole", x, ["una","lista"]) # t -> ("parole",2.7,["una","lista"])
t[1]*2 # -> 5.4
t[0][1] # -> 'a'
t[1] = 1.41 # ERRORE, le tuple sono immutabili
t[-1][0] = "minima" # t -> ("parole",2.7,["minima","lista"])
t[0][0] = 'P' # ERRORE, anche le stringhe sono immutabili
```

Esercizio. Dati

- x = 1
- l = [x,2,3]
- t1 = (1,2,l) # -> ...
- t2 = t1[:]
- t3 = (1,2,l[:])

Dopo aver applicato la funzione *id* all'ultimo elemento delle tuple t1,t2,t3, determinare il minimo numero di istruzioni tale che

```
t1[-1][0], t2[-1][0], t3[-1][0] abbiano valore 0
```

In [33]:
```
x = 1
l = [x,2,3]
t1 = (1,2,l)
t2 = t1[:]
t3 = (1,2,l[:])
id(t1[-1]), id(t2[-1]), id(t3[-1])
t1[-1][0] = 0 # t1[-1] e t2[-1] sono riferimenti alla stessa lista
t3[-1][0] = 0 # t1[-1] è un riferimento ad un'altra lista!
t1[-1][0], t3[-1][0], t3[-1][0]
```

Out[33]:
```
(0, 0, 0)
```

Esercizio. Dati

- x = 1, y = 0
- t = (x,x+1,x+3,[y,3])

si eseguano le seguenti istruzioni

```
x *= 4
t[1] += 2
y = 1
t[-1].extend([x,y])
```

Nel caso in cui l'esecuzione vada a buon fine, quanto vale l'espressione

```
t[0]*t[1] + t[-1][0]*t[-1][-1]
```

In [37]:
```
x = 1
y = 0
t = (x,x+1,x+3,[y,3]) # attenzione: il primo elemento della tupla è un riferimento
                      #  all'intero riferito da x, non alla variabile x
x *= 4
t[1] += 2 #  errore, gli oggetti di una tupla non possono essere
          # sostituiti: ricordate che t[1] += 2 è equivalente a t[1] = t[1] + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
/home/ice9/Desktop/programmazione/<ipython-input-37-96b084c2fa65> in <module>()
      3 t = (x,x+1,x+3,[y,3]) # attenzione: il primo elemento della tupla è un
riferimento all'intero riferito da x, non alla variabile x
      4 x *= 4
----> 5 t[1] += 2 #  errore, gli oggetti di una tupla non possono essere sostituiti:
ricordate che t[1] += 2 è equivalente a t[1] = t[1] + 2

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

In [38]:
```
y = 1
t[-1].extend([x,y])
t # -> (1,2,4,[0,3,4,1])
t[0] * t[1] + t[-1][0] * t[-1][-2]
```

Out[38]: 2

Esercizio extra. Dati

- x = 1.41
- l = [7,11,13]
- t = ("tag", x, l)

Determinare quali sequenze di istruzioni verificano

```
a # -> ('tag', 1.41, [7, 11, 15])
t # -> ('tag', 1.41, [7, 11, 13])
```

a.

```
a = t[:2] + l[:]
a[-1][2] = 15
```

b.

```
a = t[:]
a[-1][2] = 15
```

c.

```
a = t[:2] + (l[:],)
a[-1][-1] += 2
```

d.

```
a = t[:]
a[-1] = l[:]
a[-1][2] = 15
```

e.

```
nessuna delle precedenti
```