UNIVERSITÀ di VERONA

Linguaggio di Programmazione Python

29/03/2022

Giulio Mazzi

giulio.mazzi@univr.it

Python con stile

- Un istruzione per riga, non servono caratteri speciali di fine istruzione (e.g., ; in c e Java).
- Tuttavia è possibile usare il carattere **** per spezzare un istruzione su due o più righe, e il carattere ; per concatenare due istruzioni (sconsigliato)
- Le istruzioni contenute nelle parentesi [], {} o () non necessitano del carattere \ per continuare sulle righe successive (per sempio, liste di numeri {1, 2, ….., 100}).
- Per raggruppare un blocco di istruzioni si usa l'indentazione (e non ad esempio le parentesi come in C, Java).
- Le virgolette, o apici, servono per definire stringhe di caratteri. Python accetta indistintamente virgolette singole (') o doppie ("). Inoltre le virgolette triple (''' o """) vengono utilizzate per definire una stringa su più righe.
- I commenti su una riga iniziano col simbolo #.
- Linee guida per la scrittura del Codice python definite nel PEP 8 - Style Guide for Python Code. (https://pep8.org)

Python con stile - esempio

```
def concatena(parola, n):
    """
    Questa funzione concatena la stringa parola n volte.
    La funzione è ricorsiva.
    """
    if n <= 1:
        return parola
    else:
        # chiamata ricorsiva
        return parola + "," + concatena(parola, n-1)

s = concatena('ciao', 3)
print(s)
```

Tipi di dato in Python

Python è un linguaggio dinamicamente tipizzato.

Text Type	str
Numeric Types	int, float, complex
Sequence Types	list, tuple, range
Mapping Type	dict
Set Types	set, frozenset
Boolean Type	bool
Binary Types	bytes, bytearray, memoryview

Esercizio di riscaldamento - Tipi

La funzione **type()** restituisce il tipo dell'oggetto.

```
>>> type(5)
<type 'int'>
```

Ispezionate il tipo di:

- 10
- 10/2
- 10//2
- 1000000000000000000000000000000000000000000000000000000000000000
- "ciao" (e anche 'ciao', """ciao""")
- type(5)
- print(5)
- [1, 2, 3] (una lista di 3 numeri)
- 2**(1/2) (radice di 2)
- (-2)**(1/2) (radice di –2)

Variabili

"Una delle caratteristiche più potenti di un linguaggio di programmazione è la capacità di elaborare delle variabili. "

- Una variabile è un nome che fa riferimento ad un oggetto che contiene valore
- **Attenzione:** Una variabile non è il nome simbolico di un valore, come avviene ad esempio in C.
- Un nome valido di variabile:
  - può contenere solo lettere, numeri e il carattere di sottolineatura (_)
  - non può contenere spazi
  - non può iniziare con un numero
  - non deve essere una parola chiave del linguaggio
  - I nomi delle variabili sono case-sensitive
  - Può contenere caratteri non ASCII (utf-8)

Variabili: convenzioni

Ci sono delle convenzioni relative alla denominazione degli identificatori di Python:

- I nomi delle classi iniziano con una lettera maiuscola, tutti gli altri identificatori iniziano con una lettera minuscola
- Un identificatore che inizi con un trattino basso indica che si tratta di un identificatore privato
- Un identificatore che inizi con due trattini bassi indica che si tratta di un identificatore fortemente privato
- Se l'identificatore inizia e finisce con due trattini bassi allora è un nome speciale definito nel linguaggio.

Parole chiave del linguaggio

Keywords in Python				
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Variabili e Istruzioni di Assegnamento

Un'istruzione di assegnamento serve a creare una nuova variabile, specificandone il nome, e ad assegnarle un valore (non indichiamo esplicitamente il tipo).

Il carattere = associa ad un nome un riferimento ad un oggetto che diventa raggiungibile.

```
>>> a = 10
>>> b = 10
```

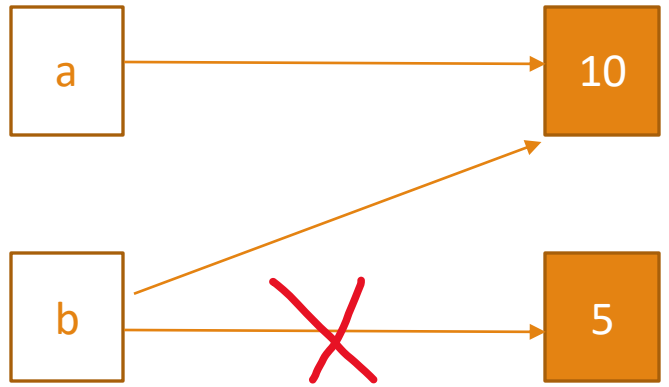
L'operatore 'is' confronta l'identità di 2 oggetti; la funzione id() restituisce un intero che rappresenta l'identità dell'oggetto

hex() converte un valore numerico in esadecimale, utile per le stampe gli indirizzi

Vediamo in jupiter...

Assegnamento

```
In [2]: a = 10
        b = 5
        print(hex(id(a)), hex(id(b)))

        0x955f60 0x955ec0

In [3]: b = 10
        print(hex(id(a)), hex(id(b)))

        0x955f60 0x955f60
```

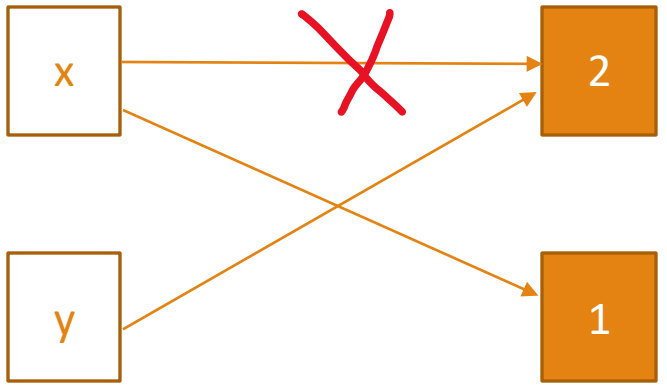


a e b sono oggetti che puntano rispettivamente a 10 e 5. Dopo l'assegnamento b = 10, entrambi puntano allo stesso oggetto.

Assegnamento – cont.

```
In [4]: x = 2
        y = x
        print(hex(id(x)), hex(id(y)))

        0x955e60 0x955e60
```

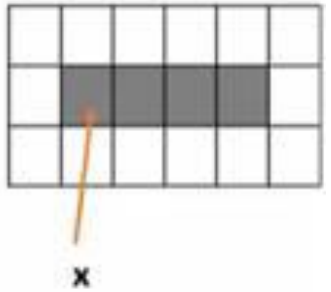
```
In [5]: x = 1
        print(hex(id(x)), hex(id(y)))

        0x955e40 0x955e60
```



L'assegnamento y = x mi dice che y punta allo stesso elemento di x, ma la loro vita separata (non è un = come in matematica!)

Confronto Python – Java/C

	Python	JAVA / C
Statement	x = 10	int x = 10;
Data type declaration	Not needed. Dynamically typed.	Mandatory. Statically typed.
What is 10?	An Object created on heap memory.	A primitive data stored in 4 byte (@JAVA) or 2 bytes (@C).
What does x contain?	Reference to Object 10	Memory location where 10 is stored
x = x + 1	x starts referring to a new object whose value is 11	x continues to point to the same memory, with value equal to 11
x = 10 y = 10	Both x and y will refer to the same object.	x and y are two variables pointing to different memory locations.



Espressioni matematiche e logiche

Espressione è una combinazione di valori e operatori

Operatore	Operazione	Esempio	Valutata in ..
+	addizione	2 + 2	4
-	sottrazione	5 - 2	3
*	moltiplicazione	3 * 5	15
/	divisione	22 / 8	2.75
%	modulo	22 % 8	6
//	divisione intera	22 // 8	2
**	esponente	2 ** 3	8
and	and logico	True and False	True
or	or logico	True and False	False

Esercizio jupyter:

Dichiarate una variabile pi = 3.14 e una variabile r (lunga a piacere)

Calcolte il volume di una sfera:

$$V = \frac{4}{3}\pi r^3$$

Espressioni matematiche e logiche: priorità degli operatori

Quando un'espressione contiene più operatori, la successione con cui viene eseguito il calcolo dipende dall'ordine delle operazioni. Per quelle matematiche, Python segue le stesse regole di precedenza comunemente usate in matematica. L'acronimo PEMDAS è un modo utile per ricordare regole

Parentesi, **E**levamento a Potenza, **M**oltiplicazione e **D**ivisione, **A**ddizione e **S**ottrazione

Usando sempre le parentesi, si evita di commettere errori!

Espressioni con stringhe

Operatore	Operazione	Esempio	Valutata in ..
+	concatenamento	‘ciao’+’ragazzi’	‘ciaoragazzi’
*	ripetizione	‘ciao’ * 2	‘ciaociao’

Vediamo come si possono concatenare le stringhe, attenzione una stringa è un oggetto immutabile!
Che succede se concateniamo una stringa con un numero?

Funzioni

una funzione è una serie di istruzioni che esegue un calcolo, alla quale viene assegnato un nome. Per definire una funzione, dovete specificarne il nome e scrivere la serie di istruzioni. In un secondo tempo, potete “chiamare” la funzione mediante il nome che le avete assegnato

```
def stampa_memoria(x):
    print(hex(id(x)))

def stampa_due_volte(x):
    print(x*2)

stampa_due_volte('ciao')
stampa_memoria('ciao')
```

```
ciaociao
0x7fe3539d0ef0
```

Funzioni – scopo variabili

Una variabile dichiarata dentro a una funzione si può usare ('vedere') solo li dentro. Si dice che questa variabile è **locale**

```
def concatena(x, y):
    cat = x + ' ' + y
    print(cat)

x = 'ciao'
y = 'mondo'
concatena(x, y)
# ERRORE! non posso vedere cat
print(cat)
```

```
ciao mondo
Traceback (most recent call last):
  File "pep8.py", line 33, in <module>
    print(cat)
NameError: name 'cat' is not defined
```

La variabile **cat** è locale, non la posso stampare!

Notare che **x** e **y** nella funzione NON sono la stessa cosa di **x** e **y** fuori (parametri formali vs parametri attuali)

Funzioni – scope variabili

Se si vuole usare una variabile esterna dentro una funzione, senza passarla come parametro, questa deve essere **globale**. Tutte le variabili create fuori dalle funzioni, sono dette variabili globali. Le posso leggere, ma NON modificare (serve parola chiave **global**)

```
x = 'ciao'

def modifica_locale():
    x = 10

def modifica_globale():
    global x
    x = 10

modifica_locale()
print(x)
modifica_globale()
print(x)
```

```
ciao
10
```

Liste

Le liste sono **sequenze di valori**. Possono contenere elementi eterogenei (e.g., possono combinare stringhe e interi)

```
terna = [23, 72, 8]
primari = ['giallo', 'magenta', 'ciano']
voti = [27, 30, 24, 'idoneo']
```

Le liste sono un primo esempio di **variabili mutabili.**

Diversi tipi di variabili sono mutabili: liste, dizionari, set, oggetti.

Tra i tipi immutabili abbiamo: numeri, stringhe e tuple, frozen_set

Liste

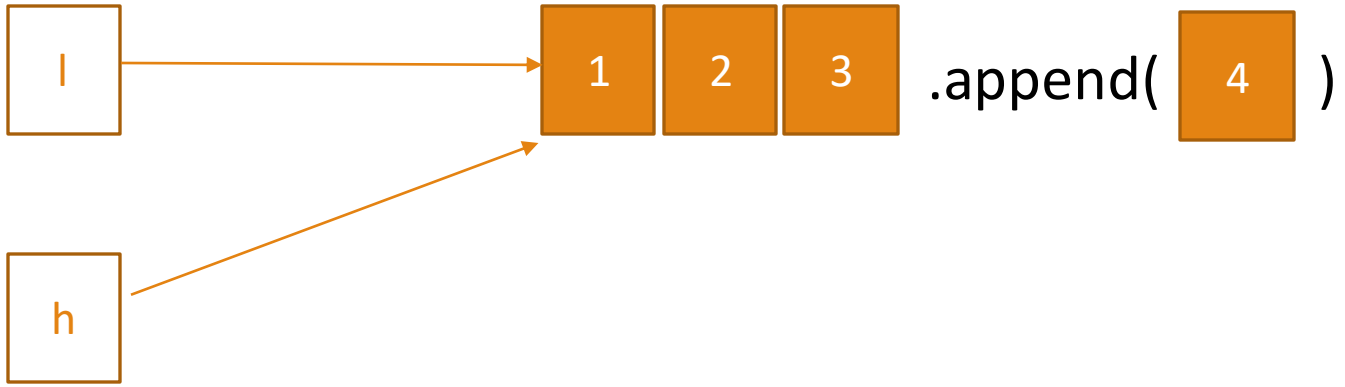
Una lista punta ad un elemento che posso mutare. Se più cose puntano allo stesso elemento, le modifiche vengono viste da tutte le variabili che puntano allo stesso elemento (mutabile).

```
l = [1, 2, 3]
h = l
print(l, h)

l.append(4)
print(l, h)
```

```
[1, 2, 3] [1, 2, 3]
[1, 2, 3, 4] [1, 2, 3, 4]
```

Liste

```
l = [1, 2, 3]
h = l
print(l, h)

l.append(4)
print(l, h)
```



Liste – creare copie

Se voglio modificare una lista mantendo anche loriginale, devo creare una copia.

Oggi vediamo un modo semplice di creare copie, l'operatore +

```
l = [1, 2, 3]
h = l
print(l, h)

l = l + [4]
print(l, h)
```

```
[1, 2, 3] [1, 2, 3]
[1, 2, 3, 4] [1, 2, 3]
```

Liste come argomenti di funzioni

Se passo una lista come argomento di funzione, questo crea un nuovo puntatore allo stesso elemento

Un append modificherà la lista originale! Se voglio gestire la lista senza cambiarla, serve una copia.