Python Basic Syntax

Indice

1 Commenti

- Cominciano col cancelletto `#` e terminano col terminatore di riga `\n`.

2 Parentesi Tonde

- Le parentesi tonde servono per chiarire le precedenze.

3 Termini Speciali

- L'UNDERSCORE _ significa che il valore assegnato a tale simbolo non ci interessa.

- Il termine `None` e' il valore nullo.

4 Esecuzione

- L'esecuzione e' sequenziale, avviene riga per riga. Il linguaggio e' imperativo.

- Molte librerie implementano in modo imperativo dei comportamenti funzionali, quindi nella pratica quando si usano librerie avanzate il tipo di programmazione diventa sia imperativa che funzionale.

5 Valori Semplici e Composti

- Con **valore semplice** intendiamo un valore del tipo: `4`, `True`.

- Con **valore composto** intendiamo un valore del tipo: `[(1, 'g'), 4]`, `3 == 2`.

- Non esistono differenze sintattiche tra valori semplici e valori composti.

- Chiamiamo con VALORE tutti i valori, sia quelli semplici sia quelli composti.

6 Variabili Mutabili e Immutabili

- Esistono due tipi di variabili: quelle **mutabili** e quelle **immutabili**.

- Le variabili immutabili sono le variabili di tipo NUMERO, VERITA', STRINGA e N-UPLA.

- Le variabili mutabili sono le variabili di tutti gli altri tipi.

- Le variabili immutabili sono passate **per valore**, le variabili mutabili sono passate **per riferimento**.

Esempio.

```
A = 1
B = A
```

```
B = 2
# ora A ha valore 1
```

Esempio.
```
A = [1]
B = A
B = [2]
# ora A ha valore [1]
```

Esempio.
```
A = [1]
B = A
B[0] = 2
# ora A ha valore [2]
```

- Per poter passare per valore una variabile mutabile bisogna farne una copia e passare la copia.

- L'espressione `variable_type(variable)` restituisce una copia della variabile.

- L'espressione `copy.copy(variable)` restituisce una copia della variabile, ma eventuali sottovariabili mutabili vengono passate per riferimento – ad esempio quando la variabile e' un oggetto di una classe e contiene attributi mutabili.

- L'espressione `copy.deepcopy(variable)` restituisce una copia della variabile, e anche tutte le sottovariabili vengono copiate. In altre parole, la copia avviene ricorsivamente e tutte le variabili vengono copiate per valore invece che per riferimento.

7 Assegnazione

- L'assegnazione ha la forma: `term` = `term`.

8 Domini

- Non esiste la dichiarazione esplicita dei domini.

- I simboli delle operazioni assumono un significato diverso a seconda del dominio dei termini su cui si applicano.

- Il valore nullo `None` fa parte di tutti i domini.

9 Operatori Booleani

- Gli operatori booleani sono: `==` `<=` `>=` `<` `>` `!=` `and` `or`.

10 Numeri

- Le operazioni usuali sono: `+ - * /`.

- L'elevamento a potenza e': `**`.

- Gli intervalli interi sono definiti dall'espressione:
`range([integer_1, ]integer_2[, integer_3])`
che restituisce la lista degli interi da `integer_1` (o da `0` se non specificato) a `integer_2 - 1` con passo `integer_3` (o con passo `1` se non specificato).

11 Stringhe

- Sono racchiuse dalle doppie virgolette ``...'' o dalle singole virgolette `...'.

- L'incollamento di due stringhe avviene con l'espressione:
``...''S``...''
dove `S` puo' essere: niente, degli spazi, delle andate a capo.

- Il carattere speciale e': \.

- La nuova riga e': \`n`.

- Per trasformare l'andare a capo in un \`n` bisogna usare le triple virgolette doppie ``````...'''''' o le triple virgolette singole ```...'''.

- Per cancellare un andare a capo dentro le triple virgolette si mette il carattere \ esattamente prima dell'andata a capo.

- Non esiste il tipo CARATTERE.

- Le stringhe sono liste di caratteri non modificabili.

12 Liste

- La notazione esplicita e': `[term_1,...,term_n]`.

- Gli elementi di una lista possono essere eterogenei.

- Gli indici positivi o nulli partono dal primo elemento. Gli indici negativi partono dall'ultimo elemento.

- L'espressione `list[index]` restituisce l'elemento di indice specificato.

- L'espressione `list[index_1:index_2]` restituisce la sottolista da `index_1` a `index_2`.

- L'espressione `list[:index]` estrae la sottolista dall'inizio a `index`.

- L'espressione `list[index:]` estrae la sottolista da `index` alla fine.

■ L'espressione `list[:]` estrae la sottolista dall'inizio alla fine.

■ L'espressione `list[index_1:index_2:integer]` estrae la sottolista da `index_1` a `index_2` con passo `integer`.

■ L'estrazione delle sottoliste scarta automaticamente gli indici che sono fuori dal dominio, senza dare errore.

■ L'espressione `len(list)` restituisce la lunghezza della lista.

■ La concatenazione di liste e': `list_1 + list_2`.

■ La moltiplicazione di liste e': `integer * list`.

■ Le espressioni `term in list`, `term not in list` restituiscono rispettivamente la presenza o l'assenza dell'elemento nella lista.

■ L'elemento minimo e' `min(list)`.

■ L'elemento minimo e' `max(list)`.

■ Il primo indice di un elemento e' `list.index(term)`. Se l'elemento non c'e' viene restituito un errore.

■ Il numero di occorrenze di un elemento e' `list.count(term)`.

■ L'aggiunta di un elemento in coda e' `list.append(term)`.

■ L'espressione `list.insert(index, term)` inserisce il termine nella posizione dell'indice specificato. I termini successivi vengono spostati in avanti di un posto.

13 N-uple

■ La notazione esplicita e': `term_1,...,term_n`.

14 Indentazione

■ L'indentazione ha una semantica: ha il ruolo di quelle che negli altri linguaggi di programmazione sono le parentesi graffe.

■ Le istruzioni allineate fanno parte dello stesso contesto. Le istruzioni che sono spostate a destra fanno parte del contesto successivo. Le istruzioni che stanno a sinistra fanno parte del contesto precedente.

■ Un'istruzione ha termine con un'andata a capo.

15 Operatori di Flusso

■ Il ciclo WHILE e':

```
while boolean_term:
    instructions
[else:
    instructions]
```

- L'`else` viene eseguito solo se il `while` non e' terminato da un `break`.

- Il ciclo IF e':

```
if boolean_term:
    instructions
[elif:
    instructions]
[else:
    instructions]
```

- `elif` e' un'abbreviazione di `else if`.

- Il ciclo FOR e':

```
for term in iterable_object:
    istructions
[else:
    istructions]
```

- L'`else` viene eseguito solo se il `for` non e' terminato da un `break`.

16 Break

- La notazione e': `break`.

- Esce dal ciclo `while` o `for` piu' vicino.

17 Continue

- La notazione e': `continue`.

- Salta l'esecuzione delle istruzioni successive del termine attuale del `for` e passa direttamente all'esecuzione del `for` del termine successivo.

18 Pass

- La notazione e': `pass`.

- E' l'istruzione vuota. Serve solo per inserire almeno un'istruzione dove altrimenti ci sarebbe un errore.

19 Try

■ La notazione e':

```
try:
    instructions
except [(]string_1[,...,string_n)]:
    instructions
```

■ Ogni errore e' associato ad una stringa.

■ Se le istruzioni danno un errore, allora la loro esecuzione viene interrotta e l'esecuzione riparte dall'`except` piu' vicino che nella n-upla di stringhe al proprio fianco presenta stringa associata all'errore.

20 Funzioni

■ La notazione e':

```
def string([string_1[= term_1],...,string_n[= term_n]]):
    instructions
    [return term]
    [yield term]
```

■ `string` e' il nome della funzione. Le `string_i` sono gli argomenti della funzione. `term` e' il valore di ritorno della funzione. Se non c'e' esplicitamente un valore di ritorno, il valore di ritorno e' posto `None`.

■ Se viene eseguito il comando `return` allora le variabili interne della funzione vengono distrutte e alla prossima chiamata della funzione le istruzioni ripartono da capo.

■ Se viene eseguito il comando `yield` allora le variabili interne della funzione mantengono il proprio valore e alla prossima chiamata della funzione le istruzioni ripartono dalla riga successiva allo `yield`.

■ Una funzione puo' essere chiamata specificando esplicitamente il valore degli argomenti – per esempio:
`function(argument_1 = 4, argument_2 = ciao)`
Questo e' utile per non dover necessariamente dare importanza all'ordine degli argomenti.

■ Alcuni argomenti possono avere un valore di inizializzazione – ad esempio:
`def function(argument_1 = 4, argument_2)`
Gli argomenti inizializzati sono facoltativi, e se non vengono specificati esplicitamente nel momento della chiamata della funzione assumono implicitamente il loro valore di inizializzazione.

21 Librerie

■ Una notazione e': `import library[.sublibrary]`. La notazione e' ricorsiva.

■ In questo caso un oggetto della libreria viene chiamato scrivendo `library.object`.

■ Un'altra notazione e':
`from library[.sublibrary] import object_1 [as nickname_1][,...,object_n [as nickname_n]]`

■ In questo caso un oggetto della libreria viene chiamato scrivendo `object` o equivalentemente `nickname`.

22 Oggetti Context

■ Gli oggetti di tipo CONTEXT possono incorrere nell'espressione WITH:

```
with object as nickname:
    instructions
```

■ All'interno delle istruzioni ci si riferisce all'oggetto col termine `nickname`.

■ L'oggetto viene aperto quando viene chiamato il `with` e viene chiuso automaticamente quando finisce il contesto del `with`. Anche tutti gli altri oggetti che vengono aperti dentro al `with` vengono chiusi automaticamente quando finisce il contesto del `with`.

■ Il `with` e' utile perche' gestisce da solo la chiusura degli oggetti aperti nel suo contesto.

23 Classi

■ Gli elementi primitivi sono i seguenti:

`classe` : e' un nuovo dominio, composto sia da variabili sia da funzioni
`oggetto` : e' una variabile di dominio classe – detta anche istanziazione della classe
`attributo` : e' una componente di tipo variabile dell'oggetto
`metodo` : e' componente di tipo funzione dell'oggetto

24 Main

■ Cio' che segue la seguente espressione:

```
if __name__ == ''__main__'':
```

viene eseguito solo se il file che e' stato indicato all'interprete di PYTHON e' il file attuale.

■ La struttura tipica di un file sorgente PYTHON e':

```
definizioni di domini e funzioni
# sono utili in questo file ma possibilmente anche in altri file

if __name__ == ''__main__'':

codice che viene eseguito quando si comunica all'interprete di eseguire questo file
# usa i domini e le funzioni di questo file ma possibilmente anche quelle di altri file
```