Python Parte 1: Caratteristiche di base

Parte del ciclo di seminari su

Programmazione Orientata agli Oggetti
e
Scripting in Python

a cura di:
Giancarlo Cherchi

Introduzione

- Un programma Python è composto da una sequenza di linee *logiche*
- Ogni linea logica è costituita da una o più linee *fisiche*
- La fine di una linea fisica costituisce il termine di un'*istruzione*
- A differenza di altri linguaggi, le istruzioni non hanno necessariamente un carattere terminatore (come ad esempio il classico ';')

Introduzione

- Ciascuna linea fisica può terminare con un *commento*, che inizia per ‘#’: tutti i caratteri dopo il simbolo sulla stessa linea sono ignorati
- E’ possibile unire due linee fisiche in un’unica linea logica attraverso il simbolo ‘\’
- L’interprete Python unisce due linee fisiche anche nei casi in cui le parentesi ‘(‘, ‘[‘, ‘{‘ non sono state chiuse

Introduzione

- Per definire dei *blocchi* all'interno di un programma non si usano dei simboli delimitatori ma l'*indentazione*
- Un *blocco* è una sequenza contigua di linee logiche, che siano indentate della stessa quantità
- Di solito si utilizzano 4 spazi per ogni livello di indentazione
- Il primo simbolo del programma NON deve avere spazi alla sua sinistra

Introduzione

- Le istruzioni digitate al prompt '>>>' dell'interprete interattivo non devono avere spazi iniziali
- La tabulazione è generalmente sostituita da 8 spazi
- E' opportuno non mescolare tab e spazi!
- E' dunque consigliabile impostare l'editor per trasformare le tabulazioni in un numero fissato di spazi, onde evitare inconsistenze

Introduzione

- Ogni linea logica è decomposta in una serie di componenti lessicali elementari, detti *token*
- I principali sono:
  - identificatori
  - parole chiave
  - operatori
  - delimitatori
  - letterali.
- E’ possibile utilizzare liberamente degli spazi per separare i token tra di loro

Identificatori

- Un *identificatore* è un nome utilizzato per identificare una variabile, una funzione, una classe, un modulo o altri oggetti
- Incomincia con una lettera (A..Z o a..z) oppure con ‘_’ (*underscore*) e contiene eventuali altre lettere e/o numeri e/o caratteri ‘_’
- Python fa distinzione tra lettere maiuscole e minuscole
- Non sono ammessi altri simboli negli identificatori (come @,$,#,%)

Identificatori

- I nomi delle classi iniziano convenzionalmente con una lettera maiuscola mentre gli altri simboli con una lettera minuscola
- Iniziare con il simbolo ‘_’ vuole indicare che l’identificatore è “privato”
- Iniziare con il doppio ‘_’ vuole indicare un identificatore fortemente privato
- Iniziare e terminare con doppio ‘_’ indica un nome speciale definito nel linguaggio
- Il singolo carattere ‘_’ è speciale e rappresenta il risultato dell’ultima operazione

Parole Chiave

- Le *parole chiave* del linguaggio sono 28 (in Python 2.2) e sono scritte in caratteri minuscoli
- Alcune di esse sono parte di istruzioni composte, mentre altre sono operatori
- Non è possibile usare come simbolo una parola chiave!
- Esempi: and, assert, def, finally, for…

Operatori

- Python utilizza come *operatori* caratteri non alfanumerici e combinazioni particolari di caratteri.
- Gli operatori principali sono:
- + - * / % ** // << >> &
  | ^ ~ < <= > >= <> != ==

Delimitatori

- Python utilizza alcuni simboli speciali come *delimitatori* in espressioni, liste, dizionari, stringhe, etc.:
  ( ) [ ] { }
  , : . = ;
  += -= *= /= //= %=
  &= |= ^= >>= <<= **=
- Il '.' è utilizzato anche nei *floating point* e nei *complessi*.

Delimitatori

- I seguenti caratteri hanno significato speciale come parte di altri token:
  ‘ “ #\
- I caratteri @, $ e ?, tutti i caratteri di controllo (tranne gli spazi bianchi) e i caratteri con codice ISO superiore a 126, NON sono utilizzati se non all’interno di stringhe e/o commenti

Letterali

- Un *letterale* è un dato che appare direttamente in un programma. Ad esempio:

  42
  3.14
  1.0J
  ‘ciao’
  “messaggio”
  “””” Buona notte!””””

Letterali

- Tramite composizione di *letterali* e *delimitatori* è possibile creare dati di altro tipo:

  [42, 3.14, ‘ciao’] #lista
  (100, 200, 300) #tupla
  { ‘x’:42, ‘y’:3.14 } #dizionario

Istruzioni

- Un sorgente Python può essere considerato come una sequenza di *istruzioni semplici* e *composte*
- A differenza di altri linguaggi, Python non ha dichiarazioni o altri elementi sintattici di alto livello

Istruzioni semplici

- Sono istruzioni che non contengono altre istruzioni
- Stanno interamente in una linea logica
- E’ possibile inserire una o più istruzioni semplici all’interno di una singola linea logica, ma ciò rende i programmi meno leggibili (va contro il ‘Python style’!)
- L’assegnamento è un’istruzione semplice e NON può essere parte di un’espressione!

Istruzioni composte

- Contengono altre istruzioni e controllano la loro esecuzione
- Un'istruzione composta ha una o più *clausole*, allineate con la stessa indentazione
- Ciascuna clausola ha un *header* che comincia con una parola chiave e termina con i due punti ':', ed è seguita da un *corpo* (*body*)
- Un corpo è una sequenza di una o più istruzioni

Istruzioni composte

- Se il corpo contiene più di un'istruzione (quindi un blocco) le istruzioni dovrebbero essere disposte su linee logiche separate, aventi la stessa indentazione
- Il blocco termina quando l'indentazione è uguale a quella dell'header della clausola
- Alternativamente, il corpo può contenere diverse istruzioni semplici separate da ';' (anche se non è un buon "Python style"!)

Tipi di dati

- I *data value* in Python sono rappresentati da oggetti e ciascun oggetto o valore, ha un *tipo*
- Il tipo di un oggetto determina quale operazione è supportata dall'oggetto
- Il tipo determina anche gli *attributi* di un oggetto e il fatto che possa essere alterato o meno
- Un oggetto che può essere modificato si definisce *mutabile*
- Un oggetto non modificabile è definito *immutabile*

Tipi di dati

- La funzione interna *type*(*obj*) restituisce il tipo dell'oggetto *obj* passato come parametro
- La funzione interna *isinstance*(*obj*, *type*) rende True se l'oggetto *obj* è di tipo *type*, rende False in caso contrario
- Python ha oggetti per gestire i tipi di dati fondamentali come *numeri*, *stringhe*, *tuple*, *liste* e *dizionari*

Numeri

- Gli oggetti di tipo numerico supportano interi (semplici e “long”), floating point e complessi.
- Tutti i *numeri* sono immutabili, quindi ogni operazione tra numeri produce sempre un nuovo oggetto
- I letterali di tipo intero possono essere *decimali*, *ottali* o *esadecimali*

Numeri

- I *decimali* sono rappresentati da una sequenza di cifre non inizianti con lo 0
- Gli *ottali* sono rappresentati da una sequenza di cifre tra 0 e 7, cominciante per 0
- Un *esadecimale* è indicato come sequenza di cifre e lettere tra A ed F, avente prefisso 0x
- Esempi:

```
1, 23, 3493          #interi decimali
01, 027, 06645       #interi ottali
0x1, 0x17, 0xDA5 #interi esadecimali
```

Numeri

- Ciascuno dei letterali interi può essere seguito da ‘l’ o ‘L’ per indicare che si tratta di un *long* int:
  1L, 23L, 999943439234L # interi decimali long
  01L, 027L, 0432432L # interi ottali long
  0x1L, 0x18L, 0x17FE54L # interi esadec. long

- Un intero long NON ha limite predefinito: può essere tanto grande quanto consente la memoria
- Un intero semplice ha dimensioni massime date da *sys.maxint* e minime date da -*sys.maxint*-1

Numeri

- Un letterale *floating-point* è rappresentato da una sequenza di cifre che contengono il punto ‘.’, una parte esponenziale o entrambi:
  0., 0.0, .0, 1., 1.0, 1e0, 1.e0, 1.0e0
- Il primo carattere non può essere ‘e’ o ‘E’, ma deve essere una cifra o il punto decimale
- Il tipo floating point corrisponde al double del C e ha come limite tipico 53 bit di precisione

Numeri

- Un numero *complesso* è costituito da due valori floating-point, uno per la parte reale e uno per la parte immaginaria
- E' possibile accedere in lettura alle parti di un complesso mediante gli attributi *real* e *imag*.
- Un floating point seguito da 'j' indica un immaginario puro
- I letterali numerici non hanno segno! Un + o un – precedenti il numero sono infatti gestiti come un *operatore* unario

Sequenze

- Una *sequenza* è un contenitore ordinato di elementi, indicizzati da interi
- Python gestisce come sequenze le *stringhe*, le *tuple* e le *liste*
- Moduli esterni e librerie forniscono altri tipi di sequenze ed è possibile personalizzarli
- E’ possibile elaborare le sequenze in modi diversi

Stringhe

- L’oggetto *stringa* è una collezione ordinata di caratteri usata per rappresentare del testo
- Le stringhe sono immutabili: ogni operazione su una stringa produce un nuovo oggetto stringa, anziché modificare l’originale
- Una stringa può essere racchiusa tra apici (‘), doppi apici (“), oppure tra tripli doppi apici (“””)

Stringhe

- I due tipi di apici hanno medesima funzionalità ma permettono di includere in modo più leggibile apici dell'altro tipo:

  "Il linguaggio \"Python\" è interessante"
  'Il linguaggio "Python" è interessante'

  'E\' possibile scrivere in modo diverso'
  "E' possibile scrivere in modo diverso"

Stringhe

- Per spezzare una stringa su più linee si utilizza il carattere ‘\’:

  “Stringa su due\
  linee” #non contiene a capo!

- Per includere nella stringa un carattere di “a capo” si utilizza la sequenza ‘\n’:

  “Stringa su due\nlinee”

Stringhe

- Per generare automaticamente tutti i caratteri di controllo, è possibile usare i tripli doppi apici “””

  “””In questa stringa vengono aggiunti

  tutti i caratteri di controllo necessari!!”””

- L’unica sequenza che non può far parte della stringa con i tripli doppi apici è la singola ‘\’

Sequenze di escape

Sequenza	Significato	Codice ASCII/ISO
\<newline>	ignora la fine della linea	Nessuno
\\	backslash	0x5c
\’	single quote	0x27
\”	double quote	0x22
\a	bell	0x07
\b	backspace	0x08
\f	form feed	0x0c
\n	newline	0x0a
\r	carriage return	0x0d
\t	tab	0x09
\v	vertical tab	0x0b
\DDD	valore ottale DDD	
\xXX	valore esadec. XX	
\other	altro carattere	0x5c + carattere

Stringhe Raw

- Una variante della stringa è la *raw string* che comincia per ‘r’ o ‘R’
- Non interpretano le sequenze di escape ma i caratteri corrispondenti vengono copiati interamente
- Sono utili per stringhe che contengono molti ‘\’
- Non possono terminare con un numero dispari di ‘\’ perché l’ultima sarebbe interpretata come una sequenza di escape con i doppi apici

Stringhe Unicode

- Il Python supporta le stringhe con caratteri *unicode*
- Sono precedute da ‘u’ o U’
- E’ possibile utilizzare al loro interno la sequenza \u seguita da 4 caratteri esadecimali per indicare il carattere unicode corrispondente
- Gestiscono anche le sequenze \N{nome} dove nome è un nome dello standard unicode.
  Ad esempio \N{Copyright Sign}

Stringhe Raw Unicode

- Le stringhe *raw unicode* vengono indicate dal prefisso ‘ur’ e non ‘ru’
- E’ possibile scrivere, affiancandole, stringhe di vario tipo: è a cura dell’interprete la loro concatenazione in un’unica stringa
- Se tra le stringhe ne esiste almeno una in formato unicode, la risultante sarà anch’essa unicode:

  s = u’CIAO’ “\n a tutti” ‘!’

Tuple

- Una *tupla* è una sequenza ordinata e immutabile di elementi
- Gli elementi di una tupla possono essere di tipo diverso
- Per definire una tupla è possibile utilizzare una serie di espressioni separate da virgole
- E’ possibile aggiungere un’ulteriore virgola dopo l’ultimo elemento della sequenza
- E’ possibile raggruppare gli elementi tra parentesi tonde

Tuple

- Alcuni esempi:

```
100, 200      # parentesi opzionali!
(100, 200)
(100, 200, ) # virgola ridondante
(3.14,)       # singleton
3.14,
()            # tupla vuota

tuple (‘ciao’)  # equivale a (‘c’,’i’,’a’,’o’)
tuple ()        #tupla vuota
```

Liste

- Una *lista* è una sequenza ordinata e mutabile di elementi
- Gli elementi della lista sono oggetti arbitrari e possono essere di tipi differenti
- Per definire una lista si usa una serie di espressioni separate da virgole, racchiusa tra parentesi quadre ‘[]’
- E’ possibile utilizzare una virgola extra dopo l’ultimo elemento

Liste

- Alcuni esempi:

```
[42, 3.14, ‘ciao’]      #lista di 3 elementi
[100]                   #lista di un elemento
[]                      #lista vuota

list(‘wow’)             #equivale a [‘w’,’o’,’w’]
list()                  #equivale a []
```

Dizionari

- I *mapping*, sono collezioni di oggetti indicizzati tramite dei valori definiti *chiavi*
- Sono mutabili e non ordinati
- Il Python ha come tipo interno di mapping il *Dizionario*
- Esistono librerie e moduli esterni in grado di gestire tipi di mapping differenti
- Le chiavi di un dizionario possono essere di vario tipo purché gestibili da un algoritmo di *hashing*

Dizionari

- I *valori* di un dizionario sono oggetti qualunque, anche di tipo differente tra loro
- Un elemento di un dizionario è una coppia *chiave/valore*: può essere dunque pensato come un array associativo
- Le chiavi non ammettono duplicati
- I dizionari sono definiti tramite una sequenza di coppie key:value, separate da virgole, racchiuse tra parentesi graffe

Dizionari

- Alcuni esempi:

```
{ ‘x’: 42, ‘y’:3.14, ‘z’:7 }
{ 1:2, 3:4 }
{}                                  #dizionario vuoto

dict( [ [1,2],[3,4] ] )        #equivale a {1:2, 3:4}

dict(x) # se x è una sequenza, deve contenere
        # delle coppie (liste) chiave valore
```

None

- Il tipo predefinito *None* denota un oggetto nullo
- None non ha né metodi né attributi
- Si utilizza quando è necessaria una referenza ma non importa a quale tipo ci si vuole riferire
- Le funzioni restituiscono None se non hanno un’istruzione esplicita di return per restituire un valore d’altro tipo