Python 2

Jacopo, Luca

GOLEM

15 maggio 2019

Nelle puntate precedenti...

- introduzione al Python
- la console
- operazioni
- print
- variabili e tipi
- `if..else`
- funzioni

Nelle puntate precedenti...

- introduzione al Python
- la console
- operazioni
- print
- variabili e tipi
- `if..else`
- funzioni

Nelle puntate precedenti...

- introduzione al Python
- la console
- operazioni
- print
- variabili e tipi
- `if..else`
- funzioni

Nelle puntate precedenti...

- introduzione al Python
- la console
- operazioni
- print
- variabili e tipi
- `if..else`
- funzioni

Nelle puntate precedenti...

- introduzione al Python
- la console
- operazioni
- print
- variabili e tipi
- `if..else`
- funzioni

Nelle puntate precedenti...

- introduzione al Python
- la console
- operazioni
- print
- variabili e tipi
- `if..else`
- funzioni

Nelle puntate precedenti...

- introduzione al Python
- la console
- operazioni
- print
- variabili e tipi
- `if..else`
- funzioni

Programma di oggi

- Sequenze: liste, tuple, dizionari, ...
- Costrutti iterativi
- Moduli esterni

Programma di oggi

- Sequenze: liste, tuple, dizionari, ...
- Costrutti iterativi
- Moduli esterni

Programma di oggi

- Sequenze: liste, tuple, dizionari, ...
- Costrutti iterativi
- Moduli esterni

Liste

Collezione **eterogenea** di oggetti. Elementi separati da `,` e racchiusi tra `[ ]`

```
lista1 = [1, 2, 3]
lista2 = [0, 'a', 'ciao', True, -3.6]
lista_vuota = []
print( lista1 )
print( lista2 )
```

Cosa posso fare con una lista?

```
print(dir( lista1 ))
```

Operazioni sulle liste (1)

Accesso ai singoli elementi tramite indice con l'operatore `[ ]`. Si conta da **zero!!**

```
lista1[0]
lista1[0] = 5
```

Lunghezza di una lista

```
len( lista2 )
```

Aggiunta di elementi

```
lista3 = lista1 + lista2 + [100, 200]    # concatenazione di liste
lista1.append(4)                         # aggiungo 4 in coda
lista2.insert(1, 'z')                    # aggiungo 'z' in posizione 1
```

Rimozione di elementi

```
lista1.pop(2)          # ritorna l'elemento di indice 2
lista2.remove('a')     # rimuove l'elemento 'a'
```

Operazioni sulle liste (2)

Ripetizione

```
lista1*3      # ripete 3 volte la lista
```

Se una lista contiene solo numeri:

```
lista4 = [3, -5, 9, 0]
sum(lista4)     # somma i numeri
```

Se una lista contiene solo numeri o solo stringhe:

```
min(lista4)     # trova il minimo
max(lista4)     # trova il massimo
sorted(lista4)  # ordina la lista
reverse(lista4) # inverte l'ordine
```

Cos'altro posso fare con una lista?

```
dir( lista4 )
```

Liste e stringhe (1)

Il tipo stringa e il tipo lista hanno molte operazioni in comune.

```
stringa = "ciao"
stringa[0]
len( stringa )
stringa + stringa
stringa * 3
```

Estrarre le parole di una stringa in una lista

```
frase = "oggi è mercoledì"
parole = frase.split()        # di default il separatore è lo spazio
```

Conversione lista/stringa

```
lettere = list("ciao")
stringa = "-".join(lettere)   # c-i-a-o
```

Liste e stringhe (2)

Operatore di appartenenza

```
stringa = "enciclopedia"
lista = [1, 2, 'a', 'b', 'c']

if "ciclope" in stringa:
  print("presente")
if 2 in lista:
  print("presente")
```

Operazione di **slicing**: estrarre una "fetta" di una lista/stringa

```
lista[primo_elemento : ultimo_elemento : passo]
```

```
stringa[2:6]        # "cicl"
lista[2:]           # dall'elemento 2 alla fine
lista[:-1]          # dall'inizio al penultimo elemento
stringa[::-1]       # dalla fine all'inizio (passo -1)
```

Tuple

Una tupla è una lista **non mutabile**. Gli elementi sono racchiusi tra `( )`, tuttavia non sono obbligatorie!

```
tupla1 = (1, 2, 3, 'a', 'b')
tupla2 = False, 'z', 20, -2.5

mia_tupla = (3.14, 'pi', True)
mia_tupla[0] = 3            # ERRORE! la tupla è un oggetto NON mutabile

mia_lista = list(mia_tupla)
mia_lista[0] = 3            # OK, la lista è mutabile

mia_stringa = "hello"
mia_stringa[0] = 'b'     # ERRORE! la stringa è un oggetto NON mutabile
```

Sequence **unpacking**: scompattare una sequenza (tipicamente tupla) nei suoi valori:

```
n1, n2, n3 = 1, 2, 3      # Unpacking esplicito della tupla (1, 2, 3)
```

Costrutti iterativi: `while`

I costrutti iterativi **ripetono** un blocco di istruzioni

Sintassi generale:

```
while condizione_vera:
  istruzione1
  istruzione2
istruzione 3      # fuori dal while! l'indentazione è cruciale
```

Esempio:

```
a = 1
while a != 0:
  a = int( input("inserisci un numero:") )
  print("Hai inserito", a)
```

Ciclo infinito:

```
while True:
  istruzioni
```

while e menu

Usare un ciclo infinito per costruire una CLI (Command Line Interface) con menù

```
a = 1; b = 2
while True:
  print()
  print("A - Aggiungi")
  print("S - Sottrai")
  print("M - Moltiplica")
  print("D - Dividi")
  print("X - Esci")
  scelta = input("Scelta?")
  if scelta == 'A':
    print(a + b)
  elif scelta == 'B':
    print(a - b)
  #...
  elif scelta == 'X':
    break
```

Il tipo speciale `range`

Oggetto iterabile contenente numeri

```
rg1 = range(5)          # contiene gli interi da 0 a 4
rg2 = range(3,8)        # contiene gli interi da 3 a 7
rg3 = range(1, 9, 2)    # contiene gli interi 1, 3, 5, 7 (passo 2)

type(rg1)               # il tipo è range
print(rg1)              # non viene stampato direttamente l'elenco!
```

Come lo utilizzo?
Posso **convertirlo** in lista con `list(rg1)` o usarlo nel ciclo `for` per ripetere alcune istruzioni un numero preciso di volte.

Costrutti iterativi: `for`

Il `for` è utilizzato come costrutto iterativo nel 99% dei casi.

Identifichiamo 2 modi di utilizzo:

- il `for` vecchio stile: si utilizza un **indice** per scorrere una sequenza (analogo a BASIC, C/C++, etc...)
- il `for` nuovo stile: l'indice scorrere direttamente gli **elementi** della sequenza, assumendone tutti i valori iterazione dopo iterazione

`for` vecchio stile

Esempio 1: ripetere un blocco di istruzioni un numero preciso di volte. Sfrutto la funzione `range`

```
for i in range(5):
  print(i, "eco")          # stampa 5 volte "eco" preceduto dall'indice
```

Esempio 2: usare l'indice per operare su ogni elemento della sequenza

```
stringa = "ciao"
for j in range(0, len(stringa)):
  print(stringa[j])

lista = [9, 'h', False, 'sole']
for k in range(len(lista)):
  print(lista[k])
```

Ogni sequenza (oggetto iterabile) può essere scorsa dal `for`: lista, stringa, tupla, range, etc...

`for` nuovo stile

```
lista = [9, 'h', False, 'sole']
for e in lista:
  print(e)
```

Lista di coppie

```
elementi = [(1, 'a'), (2, 'b'), (3, 'c')]
for i, j in elementi:      # unpacking implicito
  print(i)
  print(j)
```

Se ho comunque bisogno degli indici nel ciclo, uso la funzione `enumerate`, che restituisce una tupla (indice, valore):

```
lista = ["True", "giallo", 3]
for i, v in enumerate(lista):
  print(i, v)
```

`for` annidati

```
lista = ['mio', 'tuo', 'suo']

for parola in lista:
  for lettera in parola:
    print(lettera)
    print()
```

break e continue

break interrompe il ciclo. **continue** passa all'iterazione successiva.

Ciclo che somma i numeri in una lista, ignorando quelli pari e interrompendosi se incontra uno 0

```
lista = [1, 4, 6, 3, 8, 7, 0, 3, 2, 6]
somma = 0

for i in lista:
  if i == 0:
    break            # interrompo!
  elif i % 2 == 0:
    continue         # passo al successivo
  else:
    somma += i

print(i)             # 11
```

List comprehensions

Costrutto potente e sintetico che utilizza `for` e `if` (opzionale) per creare una lista nuova a partire da un oggetto iterabile

Sintassi generale:

```
lista = [espressione for variable in lista if condizione]
```

Esempio senza `if`:
Creare una lista2 i cui valori sono quelli della lista1 dimezzati

```
lista1 = list( range(10) )
lista2 = [i/2 for i in lista1]
```

Esempio con `if`:
Creare una lista2 i cui valori sono i quadrati dei soli numeri dispari tra 0 e 10

```
lista = [k**2 for k in range(10) if k%2 != 0]
```

Dizionari

Collezione di coppie chiave:valore racchiuse tra { }. La chiave può essere una stringa o un intero; il valore può essere un oggetto qualunque! (anche un dizionario)

```
diz = {}
diz = {'nome': 'luca', 'altezza': 178, 5: 3}
print(diz['altezza'])
chiavi = diz.keys()       # tipo speciale dict_keys, iterabile!
valori = diz.values()     # tipo speciale dict_values, iterabile!
```

Stampare gli elementi

```
for i in diz:
  print(i)              # chiavi
  print( diz[i] )       # valori
```

Operazioni sui dizionari

Controllare l'esistenza di una chiave

```
if 'altezza' in diz:
  print("c'è")
```

Eliminare una coppia chiave-valore

```
diz.pop(k)      # estrae l'elemento con chiave k
```

List comprehensions per i dizionari

Costruire un dizionario in modo conciso. List comprehension racchiusa dalle { }
Costruire un dizionario in cui le coppie chiave:valore siano rispettivamente una stringa e la sua lunghezza, a partire da una tupla:

```
parole = "python", "GOLEM", "programmazione"
diz = { p: len(p) for p in parole }
```

Iterabili

Oggetti speciali non stampabili ma “scorribili” col `for`

```
r = range(5)
lista = ['a', 'b', 'c', 'd']

i1 = iter(r)
i2 = iter(lista)
type(i1)
print(i2)

next(i1)          # scorre nell'iterabile
for j in i2:
  print(i2)
```

Moduli

File esterni che includo nel file principale.

`file1.py`

```
def somma(a, b):
  return a+b
```

`main.py`

```
import file1
s = file1.somma(2, 3)
```

`main.py` alternativo: richiama solo 1 funzione

```
from file1 import somma
s = somma(2, 3)
```

```
import this
```

Modulo matematico

Alcuni moduli (o librerie) importanti sono già presenti

```
import math as m

dir(m)       # mostra tutti i metodi disponibili

print(m.sqrt(16))
print(m.pow(2, 5))

print(m.pi)
print(m.cos( m.pi/3 ))

print(m.log(m.e))
from math import log as ln
print(ln(m.e))
print(m.log10(10))
```

Modulo `time`

```
import time as t

adesso = t.localtime()
print(adesso.tm_hour, adesso.tm_min, adesso.tm_sec)

timestamp = int( t.mktime(adesso) )

t.sleep(2)     # pausa di 2 secondi
t.sleep(.5)    # pausa di 500 millisecondi
```

Esercizi per casa I

1. Scrivere un programma che, ciclicamente, chiede l'inserimento di un numero e ne calcola la radice quadrata. Il programma termina con l'inserimento di un numero negativo.
2. Data una lista di numeri, ricreare le funzioni `sum`, `min` e `max`, che restituiscono rispettivamente la somma degli elementi, il massimo e il minimo.
3. Data una stringa, trovare il primo carattere ripetuto.
4. Esercizi 9 e 10 a questa pagina `https://www.programmareinpython.it/esercizi-python/`
5. Creare una rubrica, strutturata come lista di dizionari. Ogni elemento della lista è un dizionario contenente i campi "nome", "cognome", e "numero". Presentare all'utente un menù che permetta di inserire nuovi contatti e cercare in quelli esistenti.

Grazie per l'attenzione!

GOLEM - Gruppo Operativo Linux Empoli

GOLEM – Gruppo Operativo Linux Empoli
presso "La Vela – Margherita Hack"
via Magolo, 32 – 50053 Empoli (FI)
tutti i martedì sera dalle 21.30 alle 24.00

Crediti & licenza

Questa serata è offerta da Jacopo e Luca a partire dal materiale di giuliof (GOLEM), utilizzando un template LaTeX a cura di giomba.
Materiale rilasciato sotto GPL3 presso `golem.linux.it`