Scientific Programming

Lezione A05 - Moduli e programmi

Alberto Montresor

Università di Trento

2021/02/02

Acknowledgments: Stefano Teso

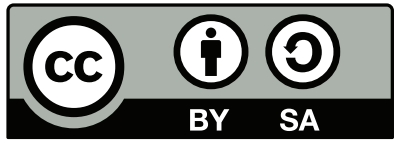

Table of contents

Definizioni

Modulo

Un modulo è un file contenente definizioni e comandi Python. Il nome del file è il nome del modulo più l'estensione `.py`.

Package

Un package è una collezione di moduli, potenzialmente contenente altri subpackage.

Installare un package

```
pip install <packagename>
```

Python Standard Library

Python Standard Library

La Python Standard Library è installata di default insieme a Python 2 e 3. Un sacco di package contenenti funzioni di utilità generale.

`https://docs.python.org/2/library/`

9. Numeric and Mathematical Modules
   - 9.1. `numbers` — Numeric abstract base classes
   - 9.2. `math` — Mathematical functions
   - 9.3. `cmath` — Mathematical functions for complex numbers
   - 9.4. `decimal` — Decimal fixed point and floating point arithmetic
   - 9.5. `fractions` — Rational numbers
   - 9.6. `random` — Generate pseudo-random numbers
   - 9.7. `itertools` — Functions creating iterators for efficient looping
   - 9.8. `functools` — Higher-order functions and operations on callable objects
   - 9.9. `operator` — Standard operators as functions

Python Package Index

Python Package Index

Il Python Package Index è un repository di software per Python, contenente più di 100k package. I package e i moduli che non sono presenti nella standard library devono essere installati.

`https://pypi.python.org/pypi`

Topic

Adaptive Technologies (35) Artistic Software (148)
Communications (1710) Database (1791)
Desktop Environment (225) Documentation (685)
Education (574) Games/Entertainment (585)
Home Automation (254) Internet (11706) Multimedia (1788)
Office/Business (1252) Other/Nonlisted Topic (150) Printing

Importare un modulo/package

Per utilizzare un modulo/package, bisogna importarlo:

```
import numpy
```

Una volta importato, puoi utilizzare le sue definizioni (funzioni, variabili, etc.) prefissandole con il nome del modulo e un punto .

```
print(numpy.arccos(0))
```

Se si prova a importare un package e si ottiene un errore, il modulo non è stato installato.

```
import iamnotinstalled
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError:  No module named iamnotinstalled
```

Importare un modulo/package

Sotto-moduli

- È possibile importare sotto-moduli specifici usando la notazione `import module.submodule`
- Si può poi chiamare le funzioni contenuti in esso con il prefisso `module.submodule.`

```
import numpy
import numpy.linalg
A = numpy.matrix([[1,2], [3,4]])
print(numpy.linalg.eig(A))
```

```
(array([-0.37228132,  5.37228132]),
 matrix([[-0.82456484, -0.41597356],
         [ 0.56576746, -0.90937671]]))
```

Importare un modulo/package

Abbreviazioni

È anche possibile abbreviare il nome di un package, come segue:

```
import numpy as np
import numpy.linalg as la
A = np.matrix([[1,2], [3,4]])
print(la.eig(A))
```

Importare funzioni individuali

Abbreviazioni

È possibile importare funzioni individuali, come segue:

```
from numpy import arccos, arcsin
print(arccos(0))
print(arcsin(0))
```

```
1.57079632679
0.0
```

Importare tutte le funzioni di un modulo

Abbreviazioni

È possibile importare tutte le funzioni individuali, come segue:

```
from math import *
print(factorial(5))
print(floor(3.45))
print(ceil(3.45))
print(sqrt(16))
print(pi)
120
3
4
4.0
3.141592653589793
```

Alcuni commenti

- `import package [as alias]`: legge il file `package.py`, inserisce tutti gli attributi nel namespace `package` (o `alias`, se presente)
- `from package import attribute`: importa alcuni attributi dal file `package.py` e li inserisce nel namespace corrente
- Quando si utilizza `from`, si possono avere sovrapposizioni fra nomi di attribuiti. L'ultimo importato vince!

```
import numpy as np
from numpy import sin
from math import sin

print(sin(np.array((0., 30., 45., 60., 90.)) * np.pi / 180. ))
```

```
TypeError: only size-1 arrays can be converted to Python scalars
```

Definire moduli

Scrivere moduli Python è semplice. Si crea un file `<modulename>.py`, e lo si importa con `import <modulename>`.

Note

- Ogni modulo ha il suo global scope
- Il global scope di un modulo si estende al solo file

Table of contents

Input e output

Leggere e scrivere file di testo

Per accedere al contenuto di un file, dobbiamo prima creare un handle per esso. Questo può essere fatto con la funzione `open()`.

Handle

Un handle è semplicemente un oggetto che mantiene informazioni su un file aperto. Tramite l'handle è possibile leggere e scrivere il file a cui fa riferimento.

Funzioni e metodi built-in

Output	**Funzione built-in**	**Significato**
`file`	`open(str, [str])`	Ottiene un handle per il file

Output	**Metodo**	**Significato**
`str`	`file.read()`	Legge tutto il file come una singola stringa
`list of str`	`file.readlines()`	Legge tutte le linee del file come una lista di stringhe
`str`	`file.readline()`	Legge una riga del file come stringa
`None`	`file.write(str)`	Scrive una stringa sul file
`None`	`file.close()`	`Chiude il file` (flush sul disco)

Aprire file

Argomenti della funzione `open()`

- Il primo argomento della `open()` è il path del file da aprire
- Il secondo argomento (opzionale) specifica la modalità di apertura

Modalità di apertura

- `"r"`: lettura (default)
- `"w"`: scrittura, sovrascrivendo il contenuto (se esiste)
- `"a"`: scrittura, in modalità append
- `"b"`: file letto in modalità binaria

Aprire file

Aprire un file restituisce un tipo handle molto particolare

```
f = open("data/table.csv", "r")
print(type(f))
print(f)
```

```
<class '\_io.TextIOWrapper'>
<\_io.TextIOWrapper name='data/table.csv' mode='r'
  encoding='US-ASCII'>
```

Chiudere file

Una volta completato il lavoro su un file (in lettura o scrittura), assicuratevi di chiamare il metodo `close()` per finalizzare la scrittura.

```
file.close()
```

Una volta che il file è chiuso, non è più possibile operare su di esso.

```
print(file.readlines())
```

```
Traceback (most recent call last):
  File "data/table.csv", line 3, in <module>
    print(file.readlines())
ValueError:  I/O operation on closed file.
```

Aprire e chiudere

```
with open("data") as f:
    print(f.readlines())
```

È buona norma usare la parola chiave `with` quando si ha a che fare con oggetti file. Il vantaggio è che il file viene chiuso correttamente al termine dell'esecuzione, anche se ad un certo punto viene sollevata un'eccezione.

Leggere file - Metodo 1

Come una singola stringa

```
contents = file.read()
print(contents)
surname,name,email address
montresor,alberto, alberto.montresor@unitn.it
bianco,luca,luca.bianco@fmach.it
```

`read()` ha senso se il file è abbastanza piccolo (sta in RAM) e non è strutturato come una sequenza di righe separate.

Leggere file - Metodo 2

Come una lista di righe (rappresentate da stringhe)

```
lines = f.readlines()
print(lines)
['surname,name,email address\n',
 'montresor,alberto, alberto.montresor@unitn.it\n',
 'bianco,luca,luca.bianco@fmach.it\n']
```

`readlines()` ha senso se il file è abbastanza piccolo (sta in RAM) ed è strutturato come una sequenza di righe.

Leggere file - Metodo 3

Una linea alla volta, sequenzialmente, a partire dalla prima, utilizzando il metodo `readline()`

```
f = open("table.csv")
line = f.readline()          # skip first line
line = f.readline()
while (line != ""):
    print(line, end="")
    line = f.readline()
montresor,alberto, alberto.montresor@unitn.it
bianco,luca,luca.bianco@fmach.it
```

`readline()` ha senso per file grandi, perché viene letta una riga alla volta, senza saturare la memoria.

Leggere file - Metodo 4

Utilizzando l'iteratore associato all'oggetto file

```
f = open("table.csv")
line = f.readline()          # skip first line
for line in f:
    print(line, end="")
```

```
montresor,alberto, alberto.montresor@unitn.it
bianco,luca,luca.bianco@fmach.it
```

Questo approccio ha senso per file grandi, perché viene letta una riga alla volta, senza saturare la memoria.

Reading through the file as a stream

Warning

- Internamente, Python tiene traccia di quali righe di un file sono già state lette.
- Una volta che una riga è stata letta, non può essere letta dallo stesso handle di file.
- Questa limitazione interessa tutti e quattro i metodi.

```
f = open("table.csv")
lines = f.readlines()          # read entire file
for line in f:
    print(line, end="")
```

Questo codice non stampa nulla.

Scrivere file

```
# Open a file for writing
f = open("result.txt", "w")

# TODO: write a complex calculation whose result is 42
result = 42

# Convert the result into a string, write a newline
f.write(str(result))
f.write("\n")

# Make sure that our writes are written to disk.
f.close()
```

Scrivere file

Attenzione

I dati sui file non vengono scritti immediatamente su disco, per motivi di efficienza. Invece, sono archiviati in memoria finché Python non decide di fare flush. `close()` è un modo per dire a Python di memorizzare le modifiche.

- Dimenticare di chiudere un file aperto in modalità di sola lettura non è dannoso (forse si supera il numero massimo di file aperti)
- Dimenticare di chiudere i file aperti in modalità di scrittura può avere gravi conseguenze
- Se il programma termina in assenza di `close()` (ad esempio a causa di un errore), le modifiche non vengono scritte su file.

Esercizio

```
f1 = open("result.txt", "w")
f1.write("Text 1\n")
f2 = open("result.txt", "w")
f2.write("Text 2\n")
f2.close()
f1.close()
```

Esercizio

```
f1 = open("result.txt", "w")
f1.write("Text 1\n")
f2 = open("result.txt", "w")
f2.write("Text 2\n")
f2.close()
f1.close()
```

```
Text 1
```

Esercizio

```
f1 = open("result.txt", "w")
f1.write("Text 1\n")
f2 = open("result.txt", "a")
f2.write("Text 2\n")
f2.close()
f1.close()
```

Esercizio

```
f1 = open("result.txt", "w")
f1.write("Text 1\n")
f2 = open("result.txt", "a")
f2.write("Text 2\n")
f2.close()
f1.close()
```

```
Text 1
```

Esercizio

```
f1 = open("result.txt", "w")
f1.write("Text 1\n")
f1.close()
f2 = open("result.txt", "a")
f2.write("Text 2\n")
f2.close()
```

Esercizio

```
f1 = open("result.txt", "w")
f1.write("Text 1\n")
f1.close()
f2 = open("result.txt", "a")
f2.write("Text 2\n")
f2.close()
```

```
Text 1
Text 2
```