Lezione 01: Introduzione a Python

Python

Python è un linguaggio di programmazione:

- *interpretato*: non ha un entrypoint, ogni riga viene eseguita in sequenza (anche le dichiarazioni di funzione, viene creato l'oggetto funzione, ma non viene eseguito il codice all'interno)
- *dinamicamente tipizzato*: duck typing
- *basato sugli oggetti*: qualsiasi cosa è un oggetto (alla Javascript, non alla Java), ovvero qualsiasi cosa è un dizionario (un **prototipo**), a cui si posso aggiungere campi (come in JavaScript) e non ha uno schema fisso (opposto a Java).

Supporta diversi paradigmi di programmazione:

- *imperativo* (funzioni, stato, ...)
- *object oriented/based* (oggetti, metodi, ereditarietà, ...)
- *funzionale* (lambda, generatori, ...)

Storia e utilizzo

È nato come linguaggio di scripting per un sistema operativo (Amoeba?).

La modularità e la pacchettizzazione fanno schifo, per questo motivo non è adatto a scrivere programmi di grandi dimensioni (secondo Cazzola non bisognerebbe superare le 100 righe), nonostante a Google lo si faccia...

Caratteristiche da sottolineare

Molte cose "ovvie" o "note" sono ovviamente omesse.

- Le variabili (o metodi) con doppio underscore all'inizio e alla fine (`__doc__`, `__name__`, ...) sono considerate riservate (ma non causano errori) e si chiamano *dunder*
- Documentazione di funzioni con una *docstring* immediatamente dopo la definizione.

  ```
  def parity(num):
    '''Restituisce se il numero è pari.'''
    return num % 2 == 0

  print(parity.__doc__) # Restituisce se il numero è pari.
  ```

- Tutto è un *first-class object*, ovvero tutto può essere assegnato e passato in giro. Ad esempio l'oggetto funzione (che viene creato quando viene definita, ma senza eseguire il codice all'interno) può essere assegnato a una variabile, passato come argomento o restituito da una funzione.
- L'indentazione è significativa e deve essere coerente (ma non importa di quanto o cosa), fare attenzione tra tab e spazi.
- I parametri opzionali devono essere definiti alla fine della lista dei parametri. Se si specifica il nome del parametro durante la chiamata, si può scambiare l'ordine dei parametri.

  ```
  def func(a, b, c=3, d=4):
    print(a, b, c, d)

  func(1, 2) # 1 2 3 4
  func(1, 2, d=3) # 1 2 3 4
  func(c=3, d=4, a=1, b=2) # 1 2 3 4
  ```

- Gli import possono essere messi ovunque
- Esistono le *eccezioni* (oggetti che possono essere personalizzati): quando si verifica un errore, viene sollevata un'eccezione che può essere gestita tramite i blocchi `try/except`. Se non viene gestita, il programma termina mostrando un traceback.

```python
try:
  from lxml import etree
except ImportError:
  import xml.etree.ElementTree as etree
```

- Ogni file Python è un modulo, ovvero può essere importato da altri file. Quando viene importato viene eseguito tutto il codice presente nel file. Ogni modulo ha un attributo speciale `__name__` che indica il nome del modulo, se viene eseguito direttamente (come script) il suo valore è `__main__`, altrimenti è il nome del file (senza path e senza estensione).

```python
# file test.py
print("pre")
import ciao
print("post")

# file ciao.py
print(__name__)
if __name__ == "__main__":
  print("hello")
```

```
> python test.py
pre
ciao
post
> python ciao.py
__main__
hello
```

- Quando si importa un modulo, allora l'interprete lo cerca nell'ordine stabilito da `sys.path` (che può essere modificata a runtime) e di default contiene:
  1. directory corrente: la cartella da cui è stato lanciato lo script (attenzione quando si chiama un file locale ad esempio `math.py`, nasconde il modulo standard `math`)
  2. variabile d'ambiente `PYTHONPATH`: se impostata, aggiunge le directory specificate al percorso di ricerca
  3. site-packages: i pacchetti installati globalmente (ad esempio tramite `pip`)
  4. librerie di sistema: le directory standard di Python