Note lezione 1

Primi approcci al Python

Mauro Oi

1 Basi di Python

Python è un linguaggio di programmazione *interpretato* (non compilato) di alto livello con una sintassi molto semplice ed efficace. Qui un link alla documentazione, in cui si possono trovare numerosi esempi e spiegazioni per chi è alle prime armi.

Di seguito, riporto una lista di *built-in features* (costruzioni astratte già implementate nativamente in Python) necessarie per la maggior parte dei programmi.

- **Variabili**: una variabile in python è un "contenitore" in cui il computer può memorizzare un valore o un insieme di valori (ad esempio, in `x = 2`, la variabile è `x`). Il nome della variabile deve iniziare con una lettera maiuscola o minuscola o un *underscore* e può contenere numeri e *underscore* (ad esempio, `x`, `Pippo`, `_Mario3` sono buoni nomi di variabili mentre `1Matteo` non lo è). Inoltre, per far sì che il codice risulti leggibile e comprensibile per voi e chi legge il vostro codice, è buona norma usare nomi *rappresentativi*: se stiamo creando una variabile che deve contenere il valore del raggio della Terra in chilometri, un nome adatto potrebbe essere `R_terra` o `R_earth`, mentre `x` potrebbe non essere di facile interpretazione. Ciascuna variabile assume il *tipo* del valore che contiene (`int` per numeri interi, `float` per numeri in virgola mobile, `str` per stringhe di testo, ...). In Python, a differenza di altri linguaggi di programmazione come C e C++, il tipo di una variabile non è fissato e può essere modificato in ogni momento nel codice.

- **Liste**: una lista è un contenitore di *oggetti* (intesi come variabili, numeri, stringhe o qualunque altro oggetto astratto che si possa creare in Python).

- **Operazioni aritmetiche**: in Python sono implementate tutte le operazioni aritmetiche di base tra variabili, quali *somma* (`+`), *differenza* (`-`), *moltiplicazione* (`*`), *divisione* (`/`), *elevamento a potenza* (`**`) e alcune altre (*modulo*, *divisione intera* ...). Alcune di queste sono implementate anche per variabili di tipo non numerico come le stringhe.

- **Operazioni logiche**: in Python sono implementate alcune operazioni logiche tra variabili di tipo `bool` (*variabili booleane*, possono contenere solo i valori logici `1/True` o `0/False`). Queste operazioni sono: `and`, `or`, `not`, `xor` e tante altre.

- **Loop**: i loop sono degli insiemi di operazioni che vengono ripetuti numerose volte. In Python esistono due tipi di loop, i `for` e i `while`.

  - `for`: sono loop in cui le operazioni vengono eseguite un numero *predefinito di volte*. La sintassi è
    `for <elemento> in <lista>:`
    Con la sintassi di cui sopra, il programma eseguirà le operazioni dopo il simbolo `:` un numero di volte pari al numero di elementi nella lista, selezionando ogni volta un elemento diverso della lista.

  - `while`: sono loop in cui le operazioni vengono eseguite finchè una condizione inizialmente `True` diventa `False`.
    `while <condizione>:`
    Con la sintassi di cui sopra, il programma eseguirà le operazioni dopo il simbolo `:` fintanto che la `<condizione>` non risulterà `False`.

- **Costrutto** `if`: il costrutto `if` è utile per eseguire una parte di codice solo quando una certa condizione è `True`. La sintassi è
  `if <condizione>:` In questo modo, tutto il codice eseguito dopo il simbolo `:` verrà eseguito solo se la `<condizione>` ha il valore logico `True`. Se vogliamo eseguire delle operazioni nel caso in cui la `<condizione>` è `False`, possiamo usare `else:` seguito dalle operazioni da effettuare nel secondo caso. Se, infine, volessimo controllare un'altra condizione nel caso in cui quella del costrutto `if` risulti falsa, possiamo usare `else:`.

2 Alcune librerie esterne utili

Con i comandi *built-in* di Python, in linea di principio, si possono creare programmi di praticamente qualunque tipo. Tuttavia, per alcune applicazioni, scriversi autonomamente un programma potrebbe essere estremamente complicato e richiederebbe una quantità enorme di tempo. Ad esempio, se volessi graficare dei dati e dovessi creare un programma per farlo, sarebbe necessario un immane lavoro per stampare il grafico su schermo. Per fortuna, altre persone hanno già scritto dei codici che facciano alcune di queste operazioni che sarebbero altrimenti particolarmente complicate.

Una libreria può essere importata in un programma Python con la sintassi:
`import <nome libreria>`
Se vogliamo utilizzare la libreria con un nome specifico:
`import <nome libreria> as <nome da attribuire>`
Se volessimo importare tutta la libreria nel nostro codice:
`from <nome libreria> import *`
Attenzione: in quest'ultimo caso se ci dovessero essere eventuali conflitti tra il nostro codice e quello della libreria il programma potrebbe non funzionare. É fortemente consigliato l'utilizzo dei primi due modi per importare librerie in modo da evitare tali conflitti.

Di seguito riporto alcune librerie comunemente utilizzate.

- `numpy`: libreria dedicata al calcolo numerico. Dentro questa libreria sono raccolti una gran quantità di strumenti matematici e numerici estremamente utili. Documentazione qui.

- `matplotlib`: libreria per plot di grafici. Documentazione qui.

- `pandas`: libreria per la manipolazione di dati. Ampiamente utilizzata in computer science e big data. Documentazione qui.

- `scipy`: libreria dedicato all'utilizzo in ambito scientifico. Documentazione qui.

3 Esempi

In questa sezione mostro alcuni esempi di utilizzo del linguaggio di programmazione Python per risolvere problemi reali.

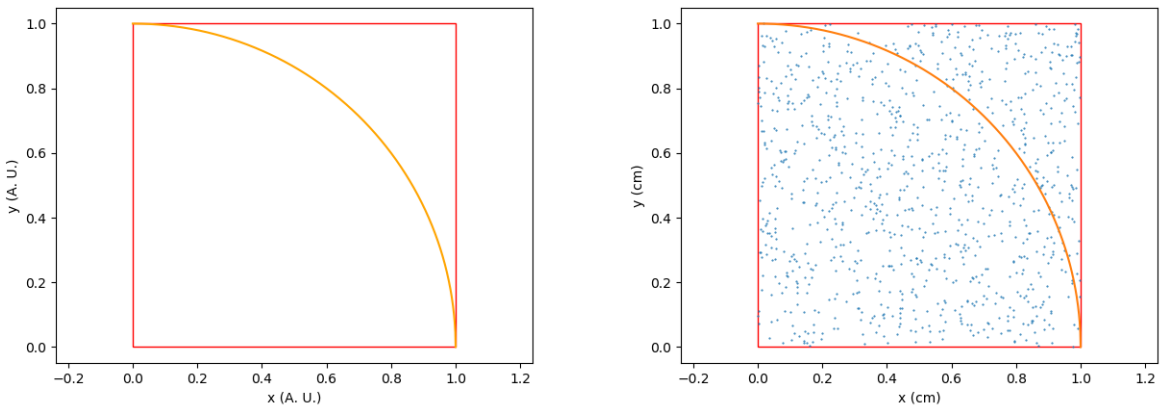

Figure 1: A sinistra: Settore circolare $\mathcal{C}$ inscritto in un quadrato $\mathcal{Q}$ di lato unitario. A destra: $\mathcal{C}$ e $\mathcal{Q}$ dentro cui sono stati generati punti random.

3.1 Approssimazione della serie geometrica

La serie geometrica è una serie ben nota che può essere scritta come

$$S_\infty = \sum_{n=0}^{\infty} ax^n. \tag{1}$$

La quantità x è detta *ragione* della serie. Questa serie è convergente e la somma s_∞ può essere calcolata considerando le somme finite e prendendo il limite per $N \to \infty$, dove N è il numero di termini successivi della successione geometrica considerata.:

$$S_N = \sum_{n=0}^{N} ax^n = \frac{1 - x^{n+1}}{1 - x}. \tag{2}$$

Il limite per $N \to \infty$ è dato da

$$\lim_{N\to\infty} S_N = \frac{1}{1 - x}, \tag{3}$$

se $0 < |X| < 1$, altrimenti per $x = 1$ la somma è infinita. La differenza tra la somma parziale e la serie è data da

$$E(N) \equiv S_\infty - S_N. \tag{4}$$

Numericamente, possiamo approssimare S_∞ con il valore di S_N tale per cui $E(N) < h$, dove h è una soglia minima di errore decisa dall'utente.

3.2 Zeri di una funzione

3.3 Approssimazione di π

In questo esempio vediamo come ottenere un'approssimazione di π generando numeri casuali. Consideriamo un settore circolare $\mathcal{C}$ iscritto in un quadrato $\mathcal{Q}$ di lato 1 come in figura 1. Supponiamo di generare coordinate random compresi nell'intervallo $[0, 1)$: la probabilità che un punto $P(x, y)$, con x e y scelti casualmente, si trovi all'interno del quarto di circonferenza sarà data dal rapporto delle aree del settore circolare e del quadrato

$$p((x, y) \in \mathcal{C}) = \frac{A_\mathcal{C}}{A_\mathcal{Q}}. \tag{5}$$

Poichè il quadrato ha area unitaria, $p((x,y) \in \mathcal{C}) = A_{\mathcal{C}} = \pi/4$. Se generiamo $N \gg 1$ punti e n di questi cadono dentro al cerchio, allora avremo che

$$p((x,y) \in \mathcal{C}) \simeq \frac{n}{N}, \tag{6}$$

quindi

$$p((x,y) \in \mathcal{C}) = \frac{\pi}{4} \simeq \frac{n}{N} \quad \rightarrow \quad \pi \simeq 4\frac{n}{N}. \tag{7}$$

Con $N = 1000$ ottengo $\pi \simeq 3.1$, che è una (scarsa) approssimazione per π. Aumentando N il rapporto n/N approssimerà sempre meglio la probabilità $p((x,y) \in \mathcal{C})$ e la stima di π sarà sempre più accurata.