Python
Parte 4: le funzioni avanzate

Mattia Cozzi
cozzimattia@gmail.com

a.s. 2024/2025

Contenuti

Cosa sono le funzioni

Quando una serie di istruzioni viene eseguita spesso è comodo trasformarla in una funzione per evitare di dover scrivere ogni volta tutto il codice. Le funzioni sono blocchi di codice che vengono essere eseguiti ogni volta che la funzione viene "chiamata".

Ad esempio potremmo creare una funzione che, quando viene chiamata, saluta tutti gli utenti attualmente connessi, oppure una funzione che stampa tutti i prodotti presenti in un database.

Le funzioni hanno dei parametri (argomenti) in ingresso e restituiscono qualche valore come output.

Creazione e invocazione di una funzione

Un esempio di creazione e invocazione con un solo parametro:

```
1  #definizione della funzione:
2  def spelling(nome):
3  #la funzione riceve in ingresso un solo parametro, nome
4    print("Il tuo nome è " + nome)
5    print("Ecco lo spelling del tuo nome")
6    for x in nome:
7      print(x)
8
9  #ricezione di dati
10 utente = input("Come ti chiami?")
11
12 #invocazione della funzione, che "processa"
13 #l'argomento che le viene passato
14 spelling(utente)
```

Esempio (1)

```
1  def scomponi(a, b, c): #la funzione ha tre parametri
2    scomp1 = "a(x - ({0:.2f}))^2"
3    scomp2 = "a(x - ({0:.2f}))(x - ({1:.2f}))"
4    delta = b*b - 4*a*c
5    if (delta < 0):
6      print("Il polinomio non si può scomporre.")
7    elif (delta == 0):
8      radice = (-b)/(2*a)
9      print("Il polinomio si può scomporre come:")
10     print(scomp1.format(radice))
11   else:
12     radice1 = (-b - delta) / (2*a)
13     radice2 = (-b + delta) / (2*a)
14     print("Il polinomio si può scomporre come:")
15     print(scomp2.format(radice1, radice2))
```

Esempio (2)

```
16  print("Inserisci i coefficienti di un polinomio di II grado:")
17  #input dei coefficienti
18  #nota il casting da stringa a float
19  primo = float(input("Primo coefficiente: "))
20  secondo = float(input("Secondo coefficiente: "))
21  terzo = float(input("Terzo coefficiente: "))
22
23  #invocazione della funzione con tre argomenti
24  scomponi(primo, secondo, terzo)
```

Attenzione! Nella definizione della funzione (righe 1-15) abbiamo creato delle variabili (come `scomp1`, `scomp2` e `delta`). Queste sono variabili locali e non sono accessibili al di fuori della funzione (cioè in altre parti del codice).

Parametri di default

Per indicare dei parametri di default, usiamo la seguente sintassi:

```
1  #definizione della funzione:
2  def spelling(nome = "Mario"):
3  #nella creazione, specifico il parametro di default
4    print("Il tuo nome è " + nome)
5    print("Ecco lo spelling del tuo nome")
6    for x in nome:
7      print(x)
8
9  #invocazione con argomento di default
10 spelling()
11
12 #invocazione con parametro personalizzato:
13 utente = input("Ora dimmi come ti chiami: ")
14 spelling(utente)
```

Ritorno di valori

Se $f(x) = x^2 - 1$, allora $f(3) = 8$. Diciamo che usando come argomento 3, alla la funzione ritorna (restituisce) il valore 8.

Anche una funzione in Python può ritornare un certo valore:

```
1  #definizione della funzione:
2  def calcolo_discriminante(a, b, c):
3    d = b*b - 4*a*c
4    return d  #la funzione "restituisce" il valore calcolato
5
6  primo = float(input("Primo coefficiente: "))
7  secondo = float(input("Secondo coefficiente: "))
8  terzo = float(input("Terzo coefficiente: "))
9  #delta assume il valore ritornato dalla funzione
10 delta = calcolo_discriminante(primo, secondo, terzo)
11 print("Il discriminante vale:")
12 print(delta)
```

Attività della funzione fuori da sé

Possiamo aver bisogno che una funzione, nello svolgere il suo compito, vada ad eseguire delle attività su variabili che sono definite al di fuori di essa (ad esempio cambiare il valore di una variabile già esistente).

Se vogliamo usare una variabile globale all'interno di una funzione, usiamo la keyword `global`:

```
1  def cambiaNome():
2    global nome #usa la variabile globale nome
3    nome = input("Inserisci il NUOVO nome: ")
4
5  nome = input("Inserisci il tuo nome: ")
6  print("Ti chiami " + nome)
7  cambiaNome()
8  print("Il tuo nuovo nome è " + nome)
```

Esercizi

1. Definisci una funzione che prende una lista di numeri e restituisce la somma di tutti gli elementi.
2. Definisci e implementa una funzione che richieda una stringa all'utente e la mostri poi al contrario.
3. Definisci e implementa una funzione che richieda una stringa all'utente e verifichi se tale stringa è palindroma.
4. Scrivi un algoritmo in cui viene chiesto all'utente di inserire dei dati numerici fino a che non digita "X". I dati devono essere salvati in una lista. Definisci e implementa una funzione che stampi solo i valori della lista maggiori della media dei valori contenuti nella lista stessa.

Programmazione orientata agli oggetti

La OOP (*Object-Oriented Programming*) nasce nel 1965 e introduce l'idea che un programma non contenga solo una serie di istruzioni, ma anche degli oggetti con determinate caratteristiche.

Gli oggetti sono delle entità informatiche che rendono oggetti del mondo reale, con le loro caratteristiche e le loro azioni (dette "metodi") specifiche.

Sono orientati agli oggetti i linguaggi C++, JavaScript, PHP, Python e molti altri.

Oggetti

Un oggetto (cioè un'istanza di una certa classe) ha delle caratteristiche e può compiere delle azioni.

- Caratteristiche:
  - prezzo
  - modello
  - peso
  - numero_porte
- Azioni:
  - accelera
  - frena
  - cambia_marcia

Modificando i valori delle caratteristiche, possiamo descrivere qualsiasi automobile.

Attributi e metodi di un oggetto

In informatica utilizziamo nomi leggermente diversi:

- le caratteristiche sono chiamate attributi;
- le azioni sono chiamate metodi.

Per l'oggetto `car`:

Attributi

```
car.name = "Fiat"
car.model = 500
car.weight = 850
car.color = "yellow"
```

Metodi

```
car.start()
car.drive()
car.brake()
car.stop()
```

Cosa sono le classi

Le classi sono un'astrazione di un tipo di entità del mondo reale.

È il "tipo di oggetto" e ne descrive le caratteristiche:

```
1  class Persona:
2    nome = "Mario"        #indico
3    cognome = "Rossi"     #i valori
4    annoNascita = 1980    #di default
5    meseNascita = 1
6    giornoNascita = 1
7    altezza = 170
8    codFiscale = "MRORSS80A01A123B"
```

Cosa sono gli oggetti

Gli oggetti sono le istanze delle classi, ovvero gli specifici oggetti che appartengono ad una certa classe.

Per la classe `Persona` creata prima, le istanze saranno i singoli individui con le loro caratteristiche.

```
1  class Persona:
2    nome = "Mario"        #indico i
3    cognome = "Rossi"     #valori di default
4
5  #creazione istanza con valori di default
6  persona1 = Persona()
7
8  #stampa delle caratteristiche di persona1
9  print(persona1.nome)
10 print(persona1.cognome)
```

Creazione ed eliminazione di istanze

Per creare un'istanza con valori personalizzati, dobbiamo usare il costruttore degli oggetti.

La sintassi è:

```
1  class Persona: #il seguente metodo è di fatto un costruttore
2    def __init__(self,nome,cognome): #richiede i parametri
3      self.nome = nome                #"self" deve sempre
4      self.cognome = cognome          #essere riportato!
5  #creazione istanza con valori personalizzati
6  persona1 = Persona("Andrea","Bianchi")
7  persona2 = Persona("Napoleone","Bonaparte")
8  print(persona1.nome)
9  print(persona1.cognome)
10 print(persona2.nome)
11 print(persona2.cognome)
12 #eliminare un'istanza
13 del persona2
```

Esercizi

5. Definisci una classe relativa ad un prodotto cosmetico od un profumo e crea almeno due istanze di quella classe. Stampa poi a schermo tutte le caratteristiche degli oggetti che hai creato.

Cosa sono i metodi

I metodi sono le attività che una certa istanza può eseguire.
Vengono definiti quando viene creata una classe:

```
1  class Persona:
2    def __init__(self,nome,cognome):
3      self.nome = nome
4      self.cognome = cognome
5    #creazione di un metodo che usa i suoi stessi attributi
6    def saluta(self):
7      print("Ciao, io sono " + self.nome + " " + self.cognome)
8    #creazione di un metodo che ne usa anche altri:
9    def occhiolino(self, x):
10     print(self.nome + " fa l'occhiolino a " + x.nome + " " +
       ↪  x.cognome)
11 persona1 = Persona("Andrea","Bianchi")
12 persona2 = Persona("Napoleone","Bonaparte")
13 persona1.presentati()
14 persona1.occhiolino(persona2)
```

Esercizi

6. Crea due istanze della classe “Gatto” con le seguenti caratteristiche: nome, padrone, colorePelo, peso, sesso. Crea poi il metodo “litiga” e scrivi un algoritmo che mostri a schermo che una istanza sta litigando con l’altra.
7. Crea due istanze della classe “Gatto” con le seguenti caratteristiche: nome, padrone, colorePelo, peso, sesso. Crea poi il metodo “litiga” e scrivi un algoritmo che mostri a schermo che “il gatto di X sta litigando con il gatto di Y”.

Modificare le proprietà di un oggetto

La sintassi è molto semplice:

```
1  class Persona:
2    def __init__(self,nome,cognome,annoN,meseN,giornoN):
3      self.nome = nome
4      self.cognome = cognome
5      self.annoN = annoN
6      self.meseN = meseN
7      self.giornoN = giornoN
8  #creazione istanza
9  persona1 = Persona("Andrea","Bianchi",2001, 11, 24)
10 #cambiamento valore di un attributo
11 persona1.nome = "Filippo"
12 print("Il nuovo nome è " + persona1.nome)
13 #eliminazione di un attributo
14 del persona1.giornoN
```

Esercizi

8. Definisci una classe relativa ad un prodotto cosmetico od un profumo e crea una istanza di quella classe. Stampa a schermo le sue caratteristiche. Scrivi poi un algoritmo che chieda all'utente quale dato di questa istanza vuole modificare. Permetti all'utente di modificare il dato scelto e mostra poi le caratteristiche dell'istanza modificata.

Esercizi

9. Definisci una classe relativa ad un prodotto cosmetico od un profumo e crea almeno due istanze di quella classe. Stampa a schermo le loro caratteristiche. Scrivi poi un algoritmo che chieda all'utente quale dato di quale istanza vuole modificare. Permetti all'utente di modificare il dato scelto nell'istanza corretta e mostra poi le caratteristiche dell'istanza modificata.

Sottoclassi (1)

Nella OOP possiamo definire una nuova classe a partire da un'altra: la nuova classe eredita tutte le caratteristiche di quella da cui deriva, e ne aggiunge delle altre.

Ad esempio potremmo avere la classe `Persona` (definita sotto) e, come sottoclassi, `Studenti` e `Docenti`.

```
1  class Persona:
2    def __init__(self,nome,cognome,email):
3      self.nome = nome
4      self.cognome = cognome
5      self.email = email
6    def saluta(self):
7      print("Ciao, io sono " + self.nome + " " + self.cognome)
```

Sottoclassi (2)

Creiamo le entità "figlie":

```
8  class Studente(Persona): #Studente deriva da Persona
9    def __init__(self,nome,cognome,email,matricola,media):
10     #recupera gli attributi dalla classe madre
11     super().__init__(nome,cognome,email)
12     self.matricola = matricola  #nuovi
13     self.media = media          #attributi
14
15 class Docente(Persona):
16   def __init__(self,nome,cognome,email,materia):
17     super().__init__(nome,cognome,email)
18     self.materia = materia
19
20 stud1 = Studente("Andrea","Bianchi","ab@mail.com",123456, 27)
21 doc1 = Docente("Silvia","Neri","sn@email.com","Informatica")
22 stud1.saluta()
23 doc1.saluta()
```

Esercizi

10. Crea la classe generale “Animale” con le seguenti caratteristiche: nomeScientifico, peso, specie. Crea poi la sottoclasse “gattoDomestico”, che aggiunge come caratteristiche: nome, padrone, indirizzo. Crea poi un metodo specifico per questa classe, come “X mangia i croccantini”. Crea infine la sottoclasse “gattoSelvatico”, che aggiunge come caratteristiche: carattere, habitat e un metodo specifico a tua scelta.
11. Usando le classi dell’esercizio precedente, genera delle istanze delle due sottoclassi e fai compiere a queste istanze delle azioni usando i loro metodi specifici.

Cosa sono i moduli

Un modulo è una libreria (cioè un file esterno) che contiene un insieme di funzioni o altro che vogliamo includere nel nostro programma, per estendere le possibilità del nostro codice.

Possiamo crearne di nuovi o caricarne di già esistenti.

Creiamo un file chiamato `mioModulo1.py` nella cartella dei nostri esercizi e scriviamoci:

```
1  def chiediNome():
2    nome = input("Come ti chiami? ")
3    return nome
4
5  def saluta(nome):
6    print("Ciao " + nome + "!")
```

Importare e utilizzare un modulo

Una volta che abbiamo importato il modulo nel nostro codice, possiamo utilizzare le funzioni in esso definite con la solita dot notation.

```
1  #il modulo deve essere nella stessa cartella
2  #del file .py che contiene il nostro programma
3  import mioModulo1 #non riportare l'estensione!
4
5  who = mioModulo1.chiediNome()
6  mioModulo1.saluta(who)
```

Alias

Quando utilizziamo molto i moduli, può diventare complesso scrivere l'intero nome del modulo.

Possiamo creare un alias più breve per un modulo quando importiamo il modulo stesso.

```
1  import mioModulo1 as my
2
3  who = my.chiediNome()
4  my.saluta(who)
```

Attenzione! Una volta fornito un alias, non possiamo più riferirci al modulo importato con il nome originario, ma solo con il suo alias.

Esercizi

12. Crea un modulo che contenga la funzione “X coccola Y”. Il modulo deve anche contenere la definizione di due classi, “Persona” e “Gatto”, con caratteristiche a tua scelta. Crea una istanza per ogni classe. Importa il modulo creato in un algoritmo e utilizza in esso le funzioni e gli oggetti definiti nel modulo.
13. Crea un modulo che contenga almeno due funzioni a tua scelta. Importa il modulo in un algoritmo e utilizza in esso le due funzioni presenti nel modulo.

Moduli built-in

Alcuni moduli utili già presenti in Python:

```
1  import math          #operazioni matematiche avanzate
2  print(dir(math))     #elenco delle nuove funzioni
3
4  import os            #gestione dei file
5  import datetime      #manipolazione di data e ora
6  import random        #generazione di valori casuali
7  import json          #gestione di dati in formato JSON
```

- Documentazione sul modulo math
- Documentazione sul modulo datetime
- Documentazione sul modulo random

Esercizi

14. Richiedi un numero intero positivo all'utente e calcola il logaritmo naturale di quel numero.
15. Richiedi all'utente due angoli in gradi e convertili in radianti.
16. Richiedi all'utente due angoli in gradi, convertili in radianti e mostra a schermo: il seno del primo angolo, il seno del secondo angolo, la somma dei due seni e il seno della somma.
17. Scrivere un programma che chieda all'utente un numero intero e che costruisca il triangolo di Pascal fino al livello corrispondente al numero inserito.
18. Scrivi un algoritmo che richieda all'utente un numero intero X e restituisca in che giorno della settimana saremo tra X giorni.

Esercizi

19. Richiedi all'utente la sua data di nascita (escludi ore, minuti, ecc.) e calcola l'età dell'utente in giorni.
20. Scrivi un algoritmo che richieda all'utente la data del suo ultimo aggiornamento della password. Se tale data risale a più di 100 giorni fa, richiede all'utente una nuova password e la salva in una variabile. Stampa poi a schermo una stringa che abbia la stessa lunghezza della password ma in cui tutti i caratteri sono stati sostituiti da asterischi.
21. Scrivi un algoritmo che richieda all'utente di inserire un numero intero e che stampi poi a schermo un numero casuale compreso tra 1 e il numero inserito (attenzione a quest'ultima richiesta).

pip

Pip è un gestore di pacchetti per Python e permette di installare e disinstallare moduli creati da altri utenti.

Per controllare se è installato (su Windows), apriamo il terminale e digitiamo il comando `pip --version`.

Pip è preinstallato se abbiamo la versione 3.4 o superiore. Per installarlo su Windows, da terminale digitiamo:

```
python get-pip.py
```

Installare nuovi moduli

Per installare o rimuovere un pacchetto, da terminale:

```
#installazione
pip install nomePacchetto

#disinstallazione
pip uninstall nomePacchetto
```

Una volta installato, un pacchetto può essere importato nel nostro file .py come abbiamo visto in precedenza.

▸ Repository dei pacchetti installabili

Esercizi

22. Installa il pacchetto `camelcase` tramite pip. Importalo in un algoritmo che richieda all'utente una stringa e usi poi il metodo `CamelCase()` (presente nel nuovo pacchetto) per mostrare all'utente la stringa che ha inserito convertita in camelcase.

Manipolazione di file di testo

È molto importante imparare a manipolare file esterni al nostro file .py, ad esempio per salvare i risultati dei nostri calcoli o le nostre manipolazioni di dati.

Impareremo a manipolare file di testo. Ci sono quattro modi per farlo:

- `r`, apre il file per leggere, errore se non esiste;
- `a`, apre il file per appendere, crea il file se non esiste;
- `w`, apre il file per scrivere, crea il file se non esiste;
- `x`, crea il file, errore se esiste già.

Mostrare un file

Creiamo, nella nostra cartella degli esercizi, un file chiamato "poesia.txt" e scriviamoci dentro qualche riga.

Ora, in un file .py:

```
1  #il file è memorizzato in una variabile,
2  #di cui posso utilizzare i metodi
3
4  #apro il file poesia.txt in lettura (read)
5  mioFile = open("poesia.txt","r")
6
7  #stampo il contenuto con il metodo read()
8  print(mioFile.read())      #stampa tutto
9  print(mioFile.read(8))     #stampa i primi 8 caratteri
10
11 #chiusura del file
12 mioFile.close()
```

Scrivere in un file

```
1  #apro in modalità append (aggiungi al fondo)
2  mioFile = open("poesia.txt","a")
3
4  #aggiunge una riga in fondo
5  mioFile.write("Una nuova linea!")
6
7  #chiusura del file
8  mioFile.close()
```

Se vogliamo mostrare poi il contenuto del file, dobbiamo chiuderlo e riaprirlo in modalità `r`.

In modalità `w`, se il file già esiste, il suo contenuto viene sovrascritto.

Esercizi

23. Crea un file di testo con un contenuto a piacere (puoi copiarlo da https://it.lipsum.com/feed/html) e scrivi un algoritmo che ne mostri a schermo il contenuto.
24. Crea un file di testo con un contenuto a piacere e scrivi un algoritmo che aggiunga una nuova linea di testo a tale file.
25. Scrivi un algoritmo che richieda all'utente di inserire alcuni suoi dati: nome, cognome, compleanno, ecc. Salva le risposte dell'utente in un file di testo che viene creato all'esecuzione dell'algoritmo.
26. Scrivi un algoritmo che permetta di creare un file di testo con un nome definito dall'utente. Scrivi successivamente la stringa "I <3 Python" nel file appena creato.

Eliminare file e gestire cartelle

Per manipolare il *file system* dobbiamo importare un modulo:

```
1  import os   #importa modulo
2
3  #se esiste (in questa cartella), elimina un file
4  if os.path.exists("poesia.txt"):
5    os.remove("poesia.txt")
6  else:
7    print("Il file non esiste.")
8
9  os.mkdir("prova")   #"make directory" crea una cartella
10 os.rmdir("prova")   #"remove directory" elimina una cartella
11
12 #"change directory" spostati in una cartella
13 os.chdir("/home/mattia/Documenti/python")
```

▸ Documentazione sul modulo os

Esercizi

27. Scrivi un algoritmo che controlla se un certo file esiste.
28. Scrivi un algoritmo che crea una nuova cartella chiamata “test”. Salva un file di testo a tuo piacere nella cartella appena creata.
29. Scrivi un algoritmo che crea una copia di un certo file di testo in una nuova cartella chiamata “backup”.