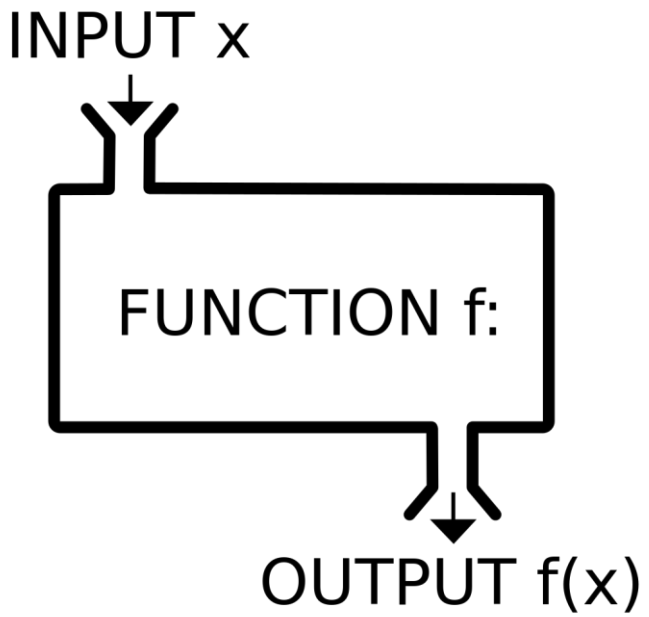

funzioni

funzioni

- funzioni
- parametri
- documentazione
- moduli
  - Python Standard Library

definizione di funzione

- ***operatore***, applicato a ***operandi***, per ottenere un ***risultato***
- **`def`** per ***definire*** una funzione
- **`return`** per terminare e restituire un ***risultato***

```
def ipotenusa(a, b):
    c = (a ** 2 + b ** 2) ** 0.5
    return c
```

chiamata (esecuzione) di funzione

- `def` definisce una funzione, ma non la esegue
- per far ***eseguire*** una funzione è necessario «***chiamarla***»
  - in corrispondenza della chiamata ad una funzione, l'esecuzione del codice ***chiamante*** si ***interrompe*** e ***comincia*** l'esecuzione della funzione ***chiamata***
  - al ***termine*** della funzione ***ricomincia*** l'esecuzione del codice chiamante dal punto in cui si era interrotto
  - la funzione, quando viene eseguita, crea nuovo spazio di nomi
  - i parametri e le variabili hanno ambito locale
  - non sono visibili nel resto del programma
  - nomi uguali, definiti in ambiti diversi, restano distinti

un esempio

- **`a`** e **`b`** sono ***parametri formali*** su cui opera la funzione
- **`c`** è una ***variabile locale*** visibile solo all'interno della funzione
- **`return c`** fornisce il valore ***risultato*** dell'esecuzione della funzione
- **`lato1`** e **`lato2`** sono i ***parametri attuali*** 'passati' alla funzione

```
def ipotenusa(a, b):
    c = (a ** 2 + b ** 2) ** 0.5
    return c

lato1 = float(input('primo lato: '))
lato2 = float(input('secondo lato: '))
lato3 = ipotenusa(lato1, lato2)
print('terzo lato:', lato3)
```

funzione main

- è spesso preferibile creare una ***funzione principale (main)***
- in questo modo si limitano le variabili globali

```
# def ipotenusa ...

def main():
    lato1 = float(input('primo lato: '))
    lato2 = float(input('secondo lato: '))
    lato3 = ipotenusa(lato1, lato2)
    print('terzo lato:', lato3)

main()
```

parametri

- la definizione della funzione opera sui ***parametri formali***
- al momento della chiamata si definiscono i ***parametri attuali***
- le variabili definite nella funzione rimangono locali a questa

```
def dummy(f1, f2):
    loc = f1 ** f2
    f1 = f1 * 2
    return loc

a1 = float(input("primo valore: "))
a2 = float(input("secondo valore: "))
print(dummy(a1,a2))
print(loc)     # NameError: name 'loc' is not defined
print(a1)      # print ???
```

documentazione di funzioni

- ***annotazioni***: utili per documentare il tipo dei parametri e il tipo del valore di ritorno *(ma non c'è verifica!)*
- ***docstring***: descrizione testuale di una funzione
- ***help***: funzione per visualizzare la documentazione

```
def hypotenuse(cathetus1: float, cathetus2: float) -> float:
    '''
    Return the hypotenuse of a right triangle, given both its legs (catheti).
    '''
    return (cathetus1 ** 2 + cathetus2 ** 2) ** 0.5
```

docstring

- la stringa di documentazione, posta all'inizio di una funzione, ne ***illustra l'interfaccia***
- per convenzione, la ***docstring*** è racchiusa tra triple virgolette `('''')` o triple doppie virgolette `("""")`, che le consentono di essere divisibile su più righe
- è breve, ma contiene le informazioni essenziali per usare la funzione
  - spiega in modo conciso ***cosa fa*** la funzione (***non come lo fa***)
  - spiega il significato di ciascun parametro e il suo tipo
- è una parte importante della progettazione dell'interfaccia
  - un'interfaccia deve essere ***semplice*** da spiegare
- i ***commenti*** (preceduti da #) hanno l'obiettivo di chiarire al programmatore il funzionamento di una parte di codice mentre le ***docstring*** sono rivolte a chi utilizzerà una funzione o una classe per chiarirne lo scopo e le modalità di interfacciamento

docstring - esempio

```
def fattoriale(n: int) -> int:
    '''
    calcolo del fattoriale
        Parameters:
            n (int): valore intero positivo

        Returns:
            fattoriale di n (int)
    '''
    fatt = 1
    for i in range(2,n+1):
        fatt *= i
    return fatt
```

procedura

- funzione ***senza return***
  - non restituisce valori
  - solo I/O ed effetti collaterali
- ***astrazione*** per riuso e leggibilità
- esempio:
  riduce i livelli di annidamento

```
def print_row(y: int, size: int):
    for x in range(1, size + 1):
        val = x * y
        print(f"{val:3}", end=" ")
    print()

def print_table(size: int):
    for y in range(1, size + 1):
        print_row(y, size)

def main():
    print_table(10)

main()
```

moduli

Python Standard Library: http://docs.python.org/3/library/

Python Standard Library

- the standard library ***is distributed with Python***
- the library contains built-in modules (written in C) that provide ***access to system functionality*** that would otherwise be inaccessible to Python programmers
- as well as modules written in Python that provide ***standardized solutions*** for many problems that occur in everyday programming

moduli

```
# import module
import math
y = math.sin(math.pi / 4)
print(y)

# import functions and constants from a module
from math import sin, pi
print(sin(pi / 4))

from random import randint
die1 = randint(1, 6)  # like rolling a die
die2 = randint(1, 6)  # like rolling a die
print(die1, die2)
```