Università degli Studi di Cagliari
Corsi di Laurea in Ingegneria Chimica ed Ingegneria Meccanica

# FONDAMENTI DI INFORMATICA

http://people.unica.it/gianlucamarcialis

A.A. 2018/2019

Docente: **Gian Luca Marcialis**

## LINGUAGGIO Python
## Funzioni

# Sommario

- Introduzione
- Funzioni in Python
  - Intestazione
  - Variabili interne ed esterne
  - Regole di visibilità
  - Esercizi

# Introduzione

- Finora abbiamo visto le strutture di controllo di base che il Python mette a disposizione per alterare il flusso di esecuzione delle istruzioni

- Da questo punto di vista, le funzioni costituiscono un metodo particolarmente raffinato per il controllo del flusso di esecuzione

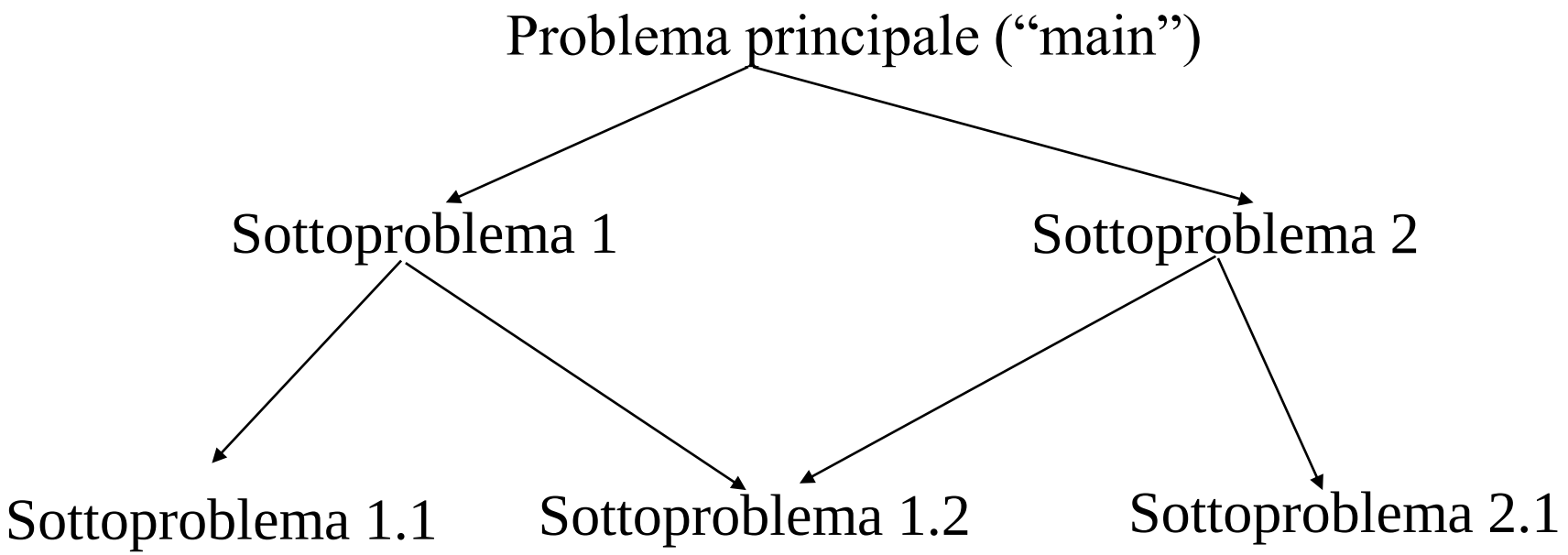
- Possiamo definire una **funzione** come un **raggruppamento di istruzioni volte a risolvere un determinato sottoproblema** entro il problema principale
  - Si pensi per esempio alle “funzioni di libreria” citate ed usate in precedenza

# Approcci di programmazione

- In generale, in problemi complessi possono essere "individuati" sottoproblemi, la cui soluzione ad esempio è necessario calcolare molte volte all'interno dell'algoritmo, ma con "ingressi" differenti
- Questo conduce a due sostanziali approcci (strategie) per la risoluzione dei problemi:
  - Top-Down
  - Bottom-Up

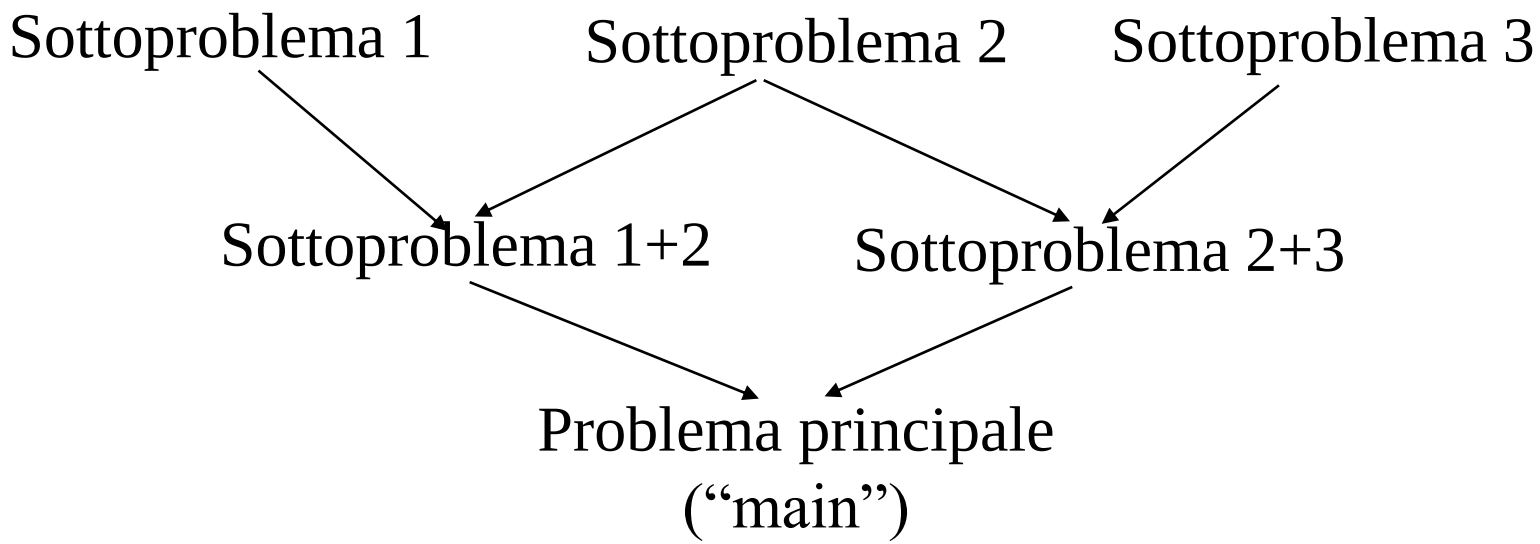
# L’approccio Top-Down

- Si parte dal problema principale e via via si individuano i sottoproblemi che lo compongono



# L’approccio Bottom-Up

- Si individuano problemi essenziali, il più semplici possibile, che, integrati gradualmente, permettono la soluzione del problema principale



# L’approccio “sandwich”

- In generale è difficile seguire il Top-Down od il Bottom-Up in modo rigoroso, in quanto
  - certe funzionalità possono risultare utili in un secondo momento
  - non si ha un’idea chiara di come il problema principale vada risolto nella sua integrità ma si è riusciti ad individuare alcune funzionalità di base
- Si preferisce un approccio “ibrido”, chiamato anche “sandwich”, in cui le due strategie vengono condotte in parallelo
  - Ad es. su una parte del problema si segue la Top-Down, su altre la Bottom-Up
- La convergenza delle due strategie porta alla soluzione del problema
- **Si noti comunque come, in tutti i casi, la capacità di scomporre il problema in problemi più semplici sia essenziale ➔ MODULARITA’ DEL SOFTWARE**
  - si migliora la chiarezza del programma
  - ne si attenua la “rigidità”

# Individuazione delle funzionalità: un esempio semplice

- Scrivere un programma Python che, leggendo da tastiera due valori interi senza segno N e K, con K<=N, calcoli il numero M di combinazioni di N oggetti a gruppi di K e lo stampi a video
- La formula per il calcolo di M è la seguente:

$$M = \binom{N}{K} = \frac{N!}{K!(N-K)!}$$

- Dove l'espressione **j!** significa "fattoriale di j" ed è data a sua volta dalla seguente. Dato un intero j>=0, posto che j!=1 se j=0:

$$j! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot j = \prod_{i=1}^{j} i$$

# Algoritmo: soluzione di “alto livello”

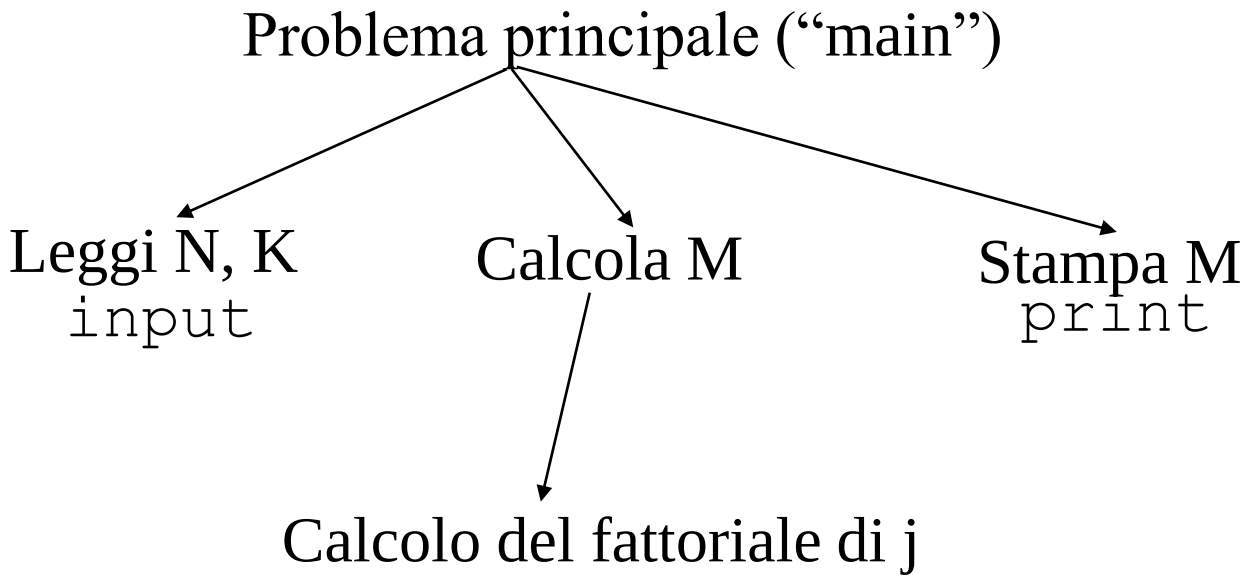
- Leggi due valori da tastiera N e K
- Calcola M
- Stampa a video M

# Algoritmo: individuazione delle funzionalità

- Leggi due valori da tastiera N e K
  - Uso `input`
- Calcola M
  - Calcola N!
  - Calcola (N-K)!
  - Calcola K!
- Stampa a video M
  - Uso `print`
- Si noti che il calcolo del fattoriale è un'operazione ripetitiva
  - per ognuno dei tre valori dovremmo ripetere lo stesso codice ma con diverso valore di partenza

# Quindi...

- Siamo riusciti a scomporre il problema in problemi più elementari
- Resta da capire come scrivere “Calcolo del fattoriale di j”



# Funzioni in Python

- Una funzione in Python è caratterizzata dalla seguente sintassi:

```
def NOME_FUNZIONE(Lista_parametri):
    #Seguono le istruzioni indentate rappresentanti
    #il corpo della funzione, ovvero
    #la soluzione del problema assolto
    #da NOME_FUNZIONE
    …
    [return Variabili_Risultato] #opzionale
```

# Funzioni in Python

- **Variabili_Risultato** corrisponde ad uno o più valori, separati da virgole, calcolati dalla funzione che ci si aspettano essere il risultato di interesse, come parametri in uscita === rappresentazione della soluzione del sottoproblema
- Espresso in forma numerica, di stringa, di lista, di dizionario o mista, a seconda del problema
- Nel caso in cui la soluzione sia «incorporata» nella funzione e non necessiti essere rappresentata in uscita, **return** può essere omessa

```
def NOME_FUNZIONE(Lista_parametri):
    #Sequenza di istruzioni indentate rappresentante
    #il corpo della funzione
    …
    return Variabili_Risultato
```

# Funzioni in Python

- **NOME_FUNZIONE** identifica il sottoproblema che essa risolve (es. `Calcola_Fattoriale`):

```
def NOME_FUNZIONE(Lista_parametri):
    #Sequenza di istruzioni indentate rappresentante
    #il corpo della funzione
    …
    return Variabili_Risultato
```

# Funzioni in Python

- **Lista_parametri** è una lista di variabili che vengono fornite alla funzione (parametri di ingresso) ed eventualmente altre che la funzione fornisce (parametri di uscita)

```
def NOME_FUNZIONE(Lista_parametri):
    #Sequenza di istruzioni indentate rappresentante
    #il corpo della funzione
    …
    return Variabili_Risultato
```

# Parametri di ingresso e uscita

- `Lista_parametri` presenta questa forma
  - `Lista_parametri = NomeVariabile1, NomeVariabile2, …, NomeVariabileN`
- Alcune di queste variabili vengono “prestate” alla funzione dall’esterno (es. dalla parte “principale” del programma) e vengono usate per calcolare l’uscita senza essere modificate ➔ parametri di ingresso
- Altre possono fare “parte” della soluzione del sottoproblema e quindi essere modificate dopo l’esecuzione dell’ultima istruzione della funzione (**`return`**) ➔ parametri di uscita
- La `Lista_parametri` può anche essere vuota, ovvero la funzione può non aspettarsi in ingresso alcun parametro

# Funzioni in Python

- Il **Corpo della funzione** è il codice che serve per risolvere il sottoproblema, che si servirà delle variabili nella `Lista_parametri`

```
def NOME_FUNZIONE(Lista_parametri):
    #Sequenza di istruzioni indentate rappresentante
    #il corpo della funzione
    … Qui le istruzioni …
    return Variabili_Risultato
```

# Chiamata a funzione

- Per chiamare una funzione che, è sufficiente seguire la sintassi:

```
[Var1[, Var2, …, VarN] =] NOME_FUNZIONE(Par1, Par2, …, ParM)
```

- Esempi
  - La funzione `f()` prevede in ingresso tre valori `x, y, z` e ne restituisce due. Possiamo scegliere di assegnarli con l'espressione: `u, v = f(x, y, z)`. Le variabili `u` e `v` conterranno i valori restituiti dalla funzione.
  - Se scrivessimo solo `u = f(x, y, z)`, la variabile `u` conterrebbe una lista di dimensione prefissata ed immutabile nei contenuti, come le stringhe, chiamata **tupla**, con i due valori restituiti dalla funzione stessa.
  - Se scrivessimo infine solo `f(x, y, z)` la funzione verrebbe avviata ma perderemmo tutti i risultati dell'elaborazione, ovvero i due valori restituiti in uscita.

# Un esempio (...prima di tornare al problema)

- Data le seguente definizione di funzione:

```
def operazioni(p, q):
      x=p+q
      y=p-q
      z=p*q
      u=p ** q
      return x, y, z, u
```

- Ora utilizziamola:

```
>>>a=2
>>>b=5
>>>e,f,g,h=operazioni(a,b)
>>>k=operazioni(a,b)
>>>e
7
>>>k
(7, -3, 10, 32)
>>>k[0]
7
```

# Torniamo al problema

- Procediamo in modo top-down, e scriviamo una prima versione del programma

```
#Programma per il calcolo combinatorio
#E' buon stile di programmazione inserire il corpo del
  programma entro una funzione «principale» (main)

def main():
   #ATTENZIONE all'indentazione!!!
   #Leggi N, K da tastiera
   #Calcola M
   #Stampa M a video
```

# Leggi N, K da tastiera: funzione `leggi`

- Scriviamo una funzione che legga da tastiera due interi e li memorizzi in una lista:

```
def leggi():

  N=input("Inserisci un intero non negativo N.\n")
  K=input("Inserisci un intero non negativo K.\n")

  return N, K   #restituisco i due valori
```

# Stampa M a video: procedura `stampa`

- Soluzione banalissima:

```
def stampa(M):
  print("Il valore richiesto è pari a: " +
  str(M) + "\n",M)
  return #posso ometterlo
```

# Calcola M: funzione `combinazioni`

- Scriviamo una funzione che restituisca appunto il valore richiesto:

```
def combinazioni(N, K):

    #Calcola il fattoriale di N
    #Calcola il fattoriale di K
    #Calcola il fattoriale di D=N-K


    M=fattN/(fattK*fattD) #formula del calcolo di (N K)
    return M
```

# Calcolo del fattoriale: funzione `fattoriale`

- A questo punto scriviamo la funzione del fattoriale parametrizzata su un generico intero senza segno `j`:

```
def fattoriale(j):

    fattj=1
    if j>1:
        i=2
        while i<=j:
            fattj *= i
            i=i+1

    return fattj
```

# Ritorniamo a `combinazioni`

- Sostituiamo ai commenti le **chiamate** alla funzione con i relativi parametri

```
def combinazioni(N, K):


    fattN=fattoriale(N)     #Calcola il fattoriale di N
    fattK=fattoriale(K)     #Calcola il fattoriale di K
    fattD=fattoriale(N-K)   #Calcola il fattoriale di D=N-K

    M=fattN/(fattK*fattD)   #formula del calcolo di (N K)
    return M
```

# Completamento del programma

- Completiamo il programma sostituendo ai commenti le opportune chiamate

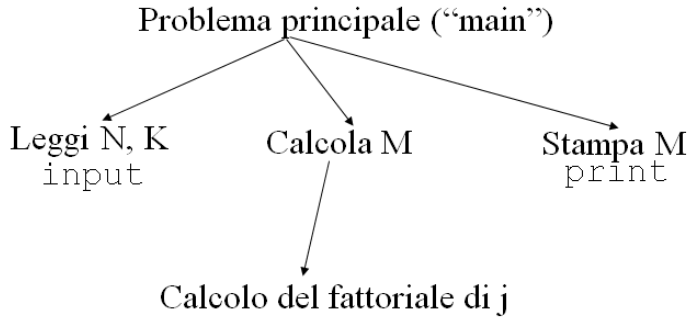
```
#Programma per il calcolo combinatorio
#E' buon stile di programmazione inserire il
  corpo del programma entro una funzione
  «principale» (main)

def main():
  N,K=leggi()          #Leggi N, K da tastiera
  M=combinazioni(N,K)     #Calcola M
  stampa(M)               #Stampa M a video
```

# Completamento del programma

- Le funzioni come sono definite, non sono viste dal sistema come «codice da eseguire» finché non vengono **invocate**

- Quindi facciamo così:
  - Scriviamo il programma sull'editor (possibilmente, ma non obbligatoriamente) rispettando l'ordine dei livelli top-down, e poi salviamolo
  - Per lanciarlo, è sufficiente:
    - Invocare la funzione `main()` appena scritta
    - Importare il file come una qualunque libreria

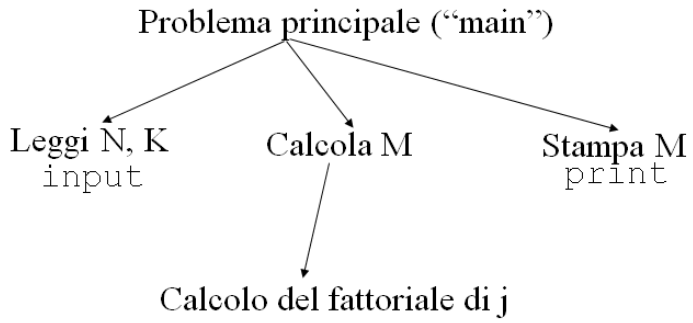
# Soluzione 1: lanciare il programma invocando la `main`




```
#Programma per il calcolo combinatorio

def fattoriale(j):

    fattj=1
    if j>1:
            i=2
            while i<=j:
                        fattj *= i

    return fattj

def leggi():

    N=input("Inserisci il valore non negativo N.\n")
    K=input("Inserisci il valore non negativo K.\n")

    return N, K
```

```
def stampa(M):
    print("Il valore richiesto è pari a: " + str(M) +
    "\n",M)
    return

def combinazioni(N, K):


    fattN=fattoriale(N)  #Calcola il fattoriale di N
    fattK=fattoriale(K)  #Calcola il fattoriale di K
    fattD=fattoriale(N-K)            #Calcola il
    fattoriale di D=N-K


    M=fattN/(fattK*fattD)  #formula del calcolo di (N K)
    return M


def main():
    N,K=leggi()          #Leggi N, K da tastiera
    M=combinazioni(N,K) #Calcola M
    stampa(M)                                    #Stampa
    M a video


main()

    #cliccando l'opzione «Run» dell'ambiente di
    sviluppo, il sistema eseguirà questa istruzione
    perché le funzioni non sono viste come codice
    eseguibile finché non vengono invocate!!!
    #oppure digitando da console
    >>>runfile(NOME_DEL_FILE)
```

# Soluzione 2: salvare il programma .py invocando la `main` nella console




```
#Programma per il calcolo combinatorio

def fattoriale(j):

    fattj=1
    if j>1:
            i=2
            while i<=j:
                        fattj *= i

    return fattj


def leggi():

    N=input("Inserisci il valore non negativo N.\n")
    K=input("Inserisci il valore non negativo K.\n")

    return N, K
```

```
def stampa(M):
    print("Il valore richiesto è pari a: " + str(M) +
    "\n",M)
    return

def calcola(N, K):


    fattN=fattoriale(N)  #Calcola il fattoriale di N
    fattK=fattoriale(K)  #Calcola il fattoriale di K
    fattD=fattoriale(N-K)            #Calcola il
    fattoriale di D=N-K

    M=fattN/(fattK*fattD)  #formula del calcolo di (N K)
    return M


def main():
    N,K=leggi()          #Leggi N, K da tastiera
    M=calcola(N,K)       #Calcola M
    stampa(M)                        #Stampa M a video

#si può eseguire da console digitando:
>>>import NOME_DEL_FILE
>>>NOME_DEL_FILE.main()
```

# Esercizio: forme geometriche

- Scrivere un programma che legge un file di testo `geometri.txt` così formattato:

  `<Forma Geometrica> v1 [v2 v3]`

- Dove
  - `<Forma Geometrica> = Triangolo | Quadrato | Rettangolo`
  - `v1 [v2 v3]` sono al massimo tre valori reali

- Per ogni riga di ingresso, il programma calcola e scrive, nel file di uscita `risultati.txt`, ed in modalità “append”, la forma della figura e il suo perimetro
- Si sviluppi la soluzione in forma **modulare**
  - Attraverso funzioni

# Esempio di file geometri.txt

```
Triangolo 34.2 45.1 90.4
Rettangolo 40.1 20.9
Rettangolo 12.1 55.5
Quadrato 45.0
```

# Vincoli per scrivere la soluzione

- Ogni riga del file dev'essere memorizzata, formattando gli opportuni campi, in una lista
  - Esempio. Se si legge la prima riga del file di esempio, la lista deve contenere `["Triangolo", 34.2, 45.1, 90.4]`
- Si scriva inoltre:
  - una funzione che calcoli il perimetro per ognuna delle tre forme: essa riceve in ingresso il vettore dei lati e restituisce un valore reale pari al perimetro
  - una funzione che legga il tipo di forma da file, e memorizzi i dati in una lista (come visto sopra); restituita in uscita
  - una funzione che, ricevendo in ingresso una lista formattata come nell'esempio, scriva la forma e stampi il perimetro nel file su file in modalità "append"

# Implementazioni delle funzioni richieste: perimetro della forma

```
def perimetro_forma(forma):

    if(forma[0]==‘Quadrato’):
        return 4.*forma[1]
    elif (forma[0]==‘Rettangolo’):
        return 2.*(forma[1]+forma[2])
    else:
        return forma[1]+forma[2]+forma[3]
```

# Aggiornamento della forma

```
def leggi_forma(fp):

    riga=fp.readline()
    if riga==””:
        return 0

    forma=riga.split()

    i=1
    while i<len(forma):
        forma[i]=float(forma[i])
        i=i+1

    return forma
```

# Stampa della forma col perimetro

```
def stampa_forma(forma,perimetro):

   out=open(“risultati.txt”,”a”)

   out.write(forma[0] + ” ” + str(perimetro) + ”\n”)

   out.close()
```

# Soluzione: vista top-down

```
#Programma per la stampa del perimetro di una forma geometrica su file

def main():

   #Apri il file geometri.txt in modalità lettura

   #Leggi una forma
   while ("""forma esiste"""):
        #Calcola il perimetro della forma letta
        #Stampa il perimetro della forma su file risultati.txt
        #Leggi un'altra forma


   #Chiudi il file geometri.txt
```

**/*Apri il file geometri.txt*/**

```
in=open(“geometri.txt”,”r”)
```

**/*Chiudi il file geometri.txt*/**

```
in.close(i)
```

# Ritorniamo al main

```
#Programma per la stampa del perimetro di una forma
  geometrica su file

def main():

  in=open("geometri.txt","r")

  forma=leggi_forma(in)
  while forma:
      perimetro=perimetro_forma(forma)
      stampa_forma(forma)
      forma=leggi_forma(in)

  in.close()
```

# Il programma completo

```
#Programma per la stampa del perimetro di una
    forma geometrica su file

def main():

    in=open("geometri.txt","r")

    forma=leggi_forma(in)
    while forma:
          perimetro=perimetro_forma(forma)
          stampa_forma(forma)
          forma=leggi_forma(in)

    in.close()


def stampa_forma(forma,perimetro):

    out=open("risultati.txt","a")
    out.write(forma[0] + " " + str(perimetro)
    + "\n")
    out.close()
```

```
def leggi_forma(fp):

    riga=fp.readline()
    if riga=="":
          return 0

    forma=riga.split()

    i=1
    while i<len(forma):
          forma[i]=float(forma[i])
          i=i+1

    return forma


def perimetro_forma(forma):

    if(forma[0]=='Quadrato'):
          return 4.*forma[1]
    elif (forma[0]=='Rettangolo'):
          return 2.*(forma[1]+forma[2])
    else:
          return forma[1]+forma[2]+forma[3]
```

# E se il file non esistesse?

- La funzione `open()`, in caso di errore dovuto ad errore di percorso o file inesistente, restituisce un valore speciale chiamato *eccezione*, ma normalmente interromperà l'esecuzione del programma.
- Per gestire questo particolare errore senza causare l'interruzione del progamma utilizziamo sempre, se richiesto, il formalismo seguente:

```
try:
        idFile = open(nomeFile,modalità_di_apertura)
except IOError:
        #istruzioni da eseguire se l'apertura è fallita
        …
```

- Il valore **`IOError`** è quello restituito in caso di mancata apertura ed è il nome dell'eccezione

# Esercizio

- Scrivere una funzione Python `apriFile` che, aprendo un file di nome `nomeFile` in modalità `modalita`, valori forniti in ingresso, restituisca l'identificativo del file se l'apertura va a buon fine, ***False*** altrimenti.

```
def apriFile(nomeFile,modalita):
    try:
        identificativoFile=open(nomeFile,modalita)
    except IOError:
        return False
    return identificativoFile
```

# Visibilità e vita delle variabili

- Le variabili nelle funzioni sono visibili esclusivamente all'interno delle funzioni stesse
- Una volta al di fuori del blocco funzionale, non «esistono» più
- Provare il codice:

```
>>>def f(x):
      x=10
>>>x=5
>>>f()
>>>print x     #secondo voi cosa stampa?
```

# Visibilità e vita delle variabili

- Ora provate questo:

```
>>>def g():
      x=10
>>>x=5
>>>g()
>>>print x    #secondo voi cosa stampa?
```

- **Conclusione:**

Se una funzione modifica ***una variabile*** già assegnata al di fuori, passata o no come parametro, **la modifica non è permanente**

# Visibilità su istanze composte

- Provare:

```
>>>def h(l):
      l[1]=10
>>>lista=[1,2,3]
>>>h(lista)
>>>print lista #secondo voi cosa stampa?
```

# Visibilità su istanze composte

- Ora provare:

```
>>>def k():
      l[1]=-1
>>>lista=[1,2,3]
>>>k(lista)
>>>print lista #secondo voi cosa stampa?
```

- **Conclusione:**

Se una funzione modifica ***una componente di variabile lista o dizionario*** già assegnata al di fuori, passata o no come parametro, **la modifica è permanente**

# Problema n.1 : scambio di valori

- Scrivere una funzione Python che, ricevendo in ingresso una lista di interi `v` e due valori `i` e `j`, permuti il valore in posizione `i` con quello in posizione `j`
- Salvare la funzione in un file scambia.py ed invocarla da console dopo averla importata

```
#Funzione per lo scambio di due valori
  in una lista
def scambia(v, i, j):
  temp=v[i]
  v[i]=v[j]
  v[j]=temp
```

# Problema n. 2: inversione di vettori

- Scrivere una funzione Python che, ricevendo una lista (vettore) `v` di `n` elementi, restituisca un secondo vettore `w` che presenti la seguente corrispondenza: `w[N-1] == v[0], w[N-2]== v[1], …, w[N-i] == v[i-1], …, w[0] == v[N-1]`

```
def inverti(v):

    w=[]

    for x in v:
        w=[x]+w    #mette in coda il primo
                   #elemento…

    return w
```

# Problema n. 2 variato

- Scrivere una funzione Python che, ricevendo una lista `v` di `n` elementi, restituisca lo stesso vettore `v` con gli elementi invertiti: `v[N-1]` ← `v[0]`, `v[N-2]`← `v[1]`, …, `v[N-i]` ← `v[i-1]`, …, `v[0]` ← `v[N-1]`
  - Nota: Usare la funzione `scambia` implementata in precedenza

```
def inverti(v):

    n=len(v)

    i=0
    while i<n/2: #occhio versione Python 3
        scambia(v,i,n-1-i)
        i=i+1
```

# Questo fatelo voi...

- Scrivere una funzione `media_mobile` che, ricevendo in ingresso una lista `v` di `N` valori numerici, restituisca un vettore `w` tale che:
  - `w[i] = (v[i] + v[i+1])/2;`
    - per `i=0,…,N-2`
  - `w[N-1] = (v[N-1] + v[0])/2`

# Soluzione

```
def media_mobile(v):

  N=len(v)
  w=[]          #inizializzazione lista vuota

  i=0
  while i<=N-2:
     w= w + [(v[i]+v[i+1])/2]

  w = w + [(v[N-1]+v[0])/2]

  return w
```

# Homework

- Scrivere una funzione `somma` che, ricevendo in ingresso due vettori di valori numerici `v` e `w` di dimensione `N` restituisca un vettore `z` tale che:
  - `z[0] = v[0] + w[N-1]`
  - `z[1] = v[1] + w[N-2]`
  - …
  - `z[N-1] = v[N-1] + w[0]`

# Esercizi

- Sviluppare gli stessi esercizi precedenti come per la variante del problema 2.

# Per saperne di più

- K.A. Lambert, Programmazione in Python, Cap. 6, Apogeo