INGEGNERIA PER LA SOSTENIBILITA' INDUSTRIALE - UnivPM

Fondamenti di Informatica

Prof. Ing. Loris Penserini, PhD

elpense@gmail.com

https://orcid.org/0009-0008-6157-0396

Materiale:

https://github.com/penserini/Lezioni_UnivPM.git

Algoritmi con «Funzioni» (overview)

Concetto di «Funzione»

Nell'ingegneria del software, concetti come: usabilità, modularità, manutenibilità, affidabilità, testabilità, portabilità, ecc. sono chiamati «requisiti non funzionali» del sistema e ne determinano gli «obiettivi di qualità» (es. «soft-goal» in Tropos, [Penserini et al., 2007a]).

Nella programmazione, che deve considerare/implementare i requisiti del sistema, si adottano «**convenzioni**» per meglio raggiungere anche questi obiettivi di qualità.

Una di queste convenzioni, molto usata nella programmazione «**top-down**», è quella di procedere per astrazioni successive:

- Prima si realizzano i moduli fondamentali del sistema, trascurando i dettagli
- Poi si procede a realizzare i moduli con i dettagli

Ogni modulo svolge una specifica funzione all'interno del problema, cioè il programma deve essere scomposto in moduli funzionalmente indipendenti che vengono chiamati «**Funzioni**», «**metodi**», o «**procedure**» a seconda del linguaggio usato.

Passaggio di parametri tra «Funzioni»

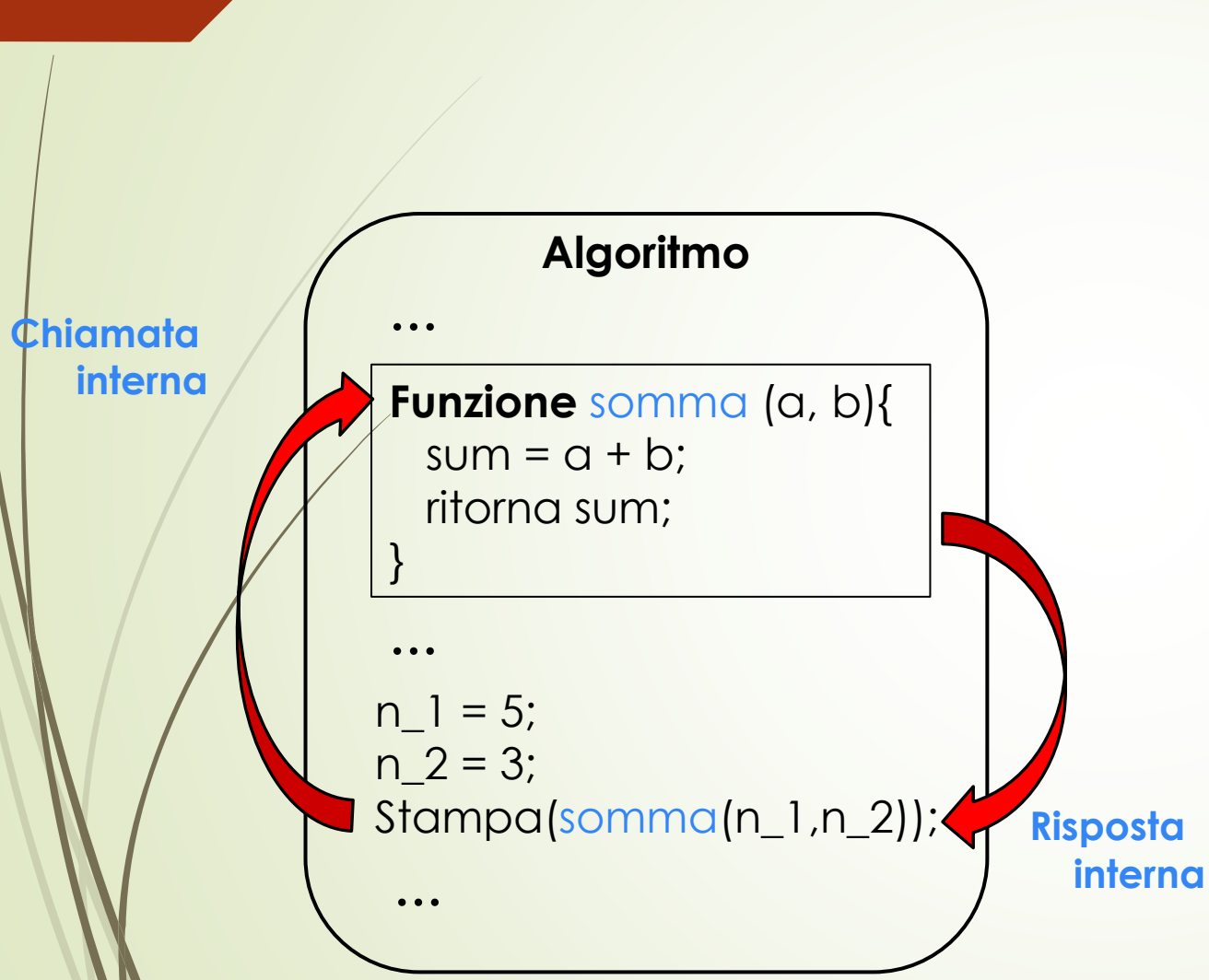
Lo stile di usare un codice modulare, premia sicuramente l'usabilità e la manutenzione del software, per cui è importante poter riutilizzare sottoprogrammi provenienti da altri progetti. Per questo motivo, **le funzioni/sottoprogrammi consentono il passaggio di parametri «da» e «verso» il modulo chiamante.**

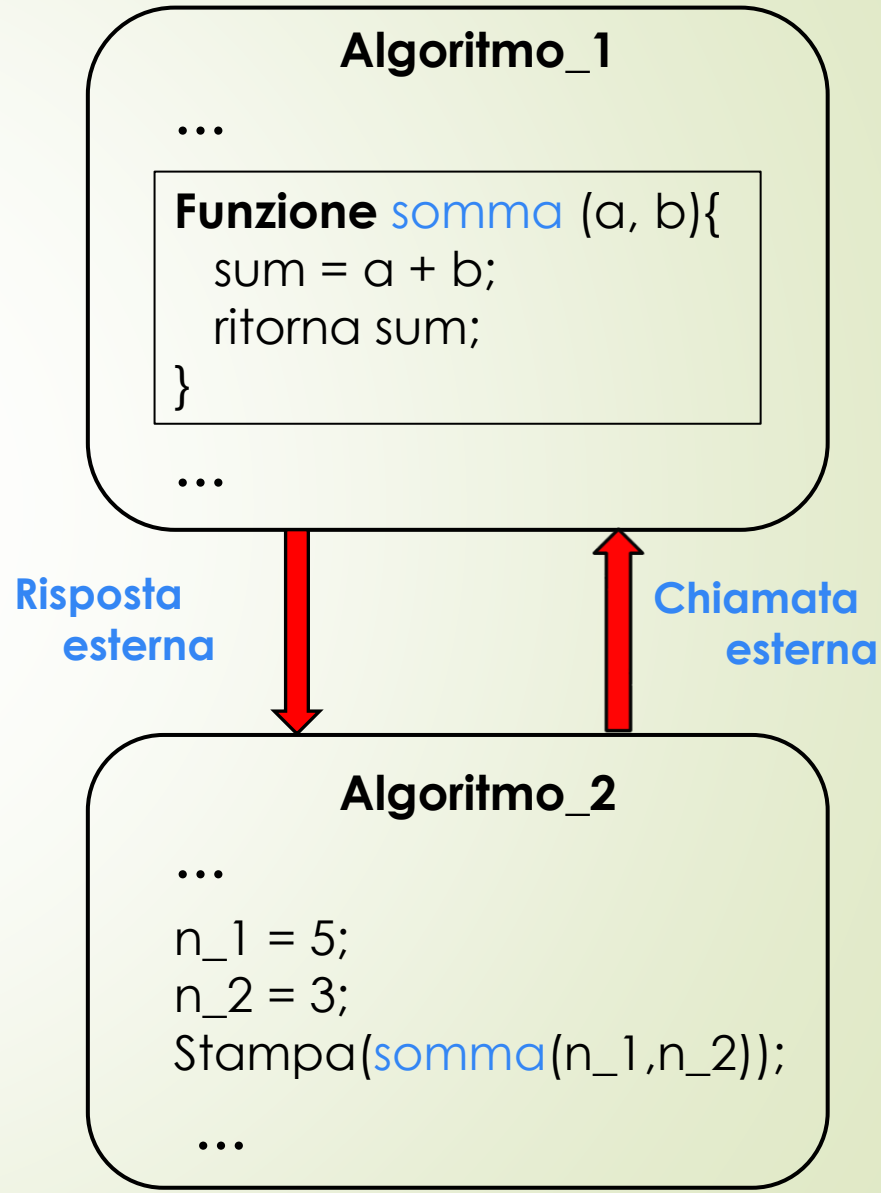
Il chiamante può essere lo stesso modulo che richiama una sua funzione interna, ma si possono avere chiamate a funzioni di librerie/moduli esterni.

Nella OOP questa tecnica avviene in modo chiaro a diversi livelli:

- di classe: cioè metodi/funzioni dello stesso modulo/classe
- di oggetto: cioè metodi/funzioni dello stesso gruppo di moduli/classi che costituiscono il programma/oggetto
- di package: cioè metodi/funzioni esterne, cioè scritte da terze parti

Passaggio di parametri tra «Funzioni»





Utilizzare Funzioni del Linguaggio

Le funzioni in Python, oltre a quanto già detto sul passaggio di parametri e la possibilità di restituzione di un valore, possono avere argomenti facoltativi e/o argomenti obbligatori.

Per esempio, se usiamo la funzione del linguaggio per arrotondare numeri decimali con un solo argomento (parametro):

```
>>> round(4.37676)
4
>>> round(8.578)
9
```

Per cui, **obbligatoriamente** «round» vuole un parametro di cui calcola l'arrotondamento e restituisce solo la parte intera, cioè senza la parte decimale del numero arrotondato.

Utilizzare Funzioni del Linguaggio

Mentre, se usiamo la funzione «round» del linguaggio fornendo due parametri come argomenti:

>>> round(4.37676, 3)

4.377

>>> round(8.578, 1)

8.6

Per cui, in modo **facoltativo**, se inseriamo nella funzione «round» il secondo parametro questo viene utilizzato per determinare il numero di cifre decimali da visualizzare per il numero arrotondato.

Modularità in Python

Parlare di **funzioni** e **moduli** in Python è fondamentale per capire come organizzare e riutilizzare il codice. La parola **modulo** deriva dal concetto di **modularità**, un principio fondamentale dell'ingegneria del software (e non solo).

Significa **suddividere un sistema complesso in parti più piccole, indipendenti e riutilizzabili**. Queste "parti", in Python, sono appunto i **moduli**.

Vediamo, in modo chiaro e completo, **le diverse tecniche per richiamare funzioni esterne**.

Caratteristiche del Modulo

In Python un **modulo è un blocco di codice autonomo**, racchiuso in un file *.py con le seguenti caratteristiche:

- Serve per organizzare, riusare e mantenere meglio il codice
- Ogni modulo può definire funzioni, classi o variabili che altri moduli possono importare
- In altre parole, ogni file Python può essere un "modulo" di un programma più grande

Differenza tra modulo e libreria

- Un **modulo** = un singolo file *.py
- Una **libreria** (o **package**) = un insieme di moduli organizzati in cartelle.

Importare Funzioni da moduli esterni

Bisogna sempre scrivere il nome del modulo prima della funzione:

matematica.somma(...)

- Mantiene chiaro da dove provengono le funzioni
- Evita conflitti di nomi.

```python
# matematica.py
def somma(a, b):
    return a + b

def differenza(a, b):
    return a - b
```

Importare l'intero modulo:

```python
import matematica

ris = matematica.somma(3, 5)
print(ris)  # 8
```

Importare solo alcune Funzioni

Importare solo alcune funzioni:

- (pros) Più conciso, non serve scrivere «matematica. » ogni volta
- (cons) Se due moduli hanno funzioni con lo stesso nome, possono entrare in conflitto

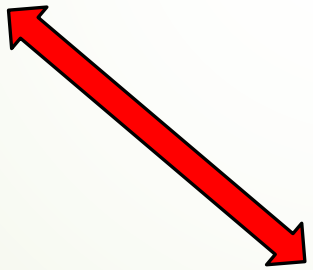
```python
# matematica.py
def somma(a, b):
    return a + b

def differenza(a, b):
    return a - b
```

```python
from matematica import somma

print(somma(3, 5))  # 8
```

Importare più funzioni:



```python
from matematica import somma, differenza

print(somma(10, 2))
print(differenza(10, 2))
```

Assegnare un alias al modulo

Utile per abbreviare nomi lunghi (es. **import numpy as np**)

```python
# matematica.py
def somma(a, b):
    return a + b

def differenza(a, b):
    return a - b
```

```python
import matematica as m

print(m.somma(4, 6))
```

Import dinamico con «importlib»

A volte si vuole importare un modulo in modo dinamico (ad esempio, il nome è deciso a **runtime**):

```python
# matematica.py
def somma(a, b):
    return a + b

def differenza(a, b):
    return a - b
```

```python
import importlib

nome_modulo = "matematica"
m = importlib.import_module(nome_modulo)
print(m.somma(2, 3))
```

Differenti usi dei Moduli

Moduli standard e moduli personalizzati

- Python ha molti moduli standard (come math, random, os, datetime, ecc.)
- Si possono anche creare moduli personalizzati (progetti con file *.py)
- Oppure installare moduli esterni tramite pip (es. pip install requests)

```
import math
print(math.sqrt(16))  # 4.0
```

Per creare un modulo eseguibile sia da solo che importabile da altri, si usa:

```
if __name__ == "__main__":
    # Codice eseguito solo se il file è lanciato direttamente
    print(somma(2, 3))
```

Differenza importante

Queste due istruzioni:

1. **import** matematica
2. **from** matematica **import ***

sembrano fare la stessa cosa, cioè "rendere disponibili" le funzioni di «matematica». Ma in realtà non sono affatto equivalenti.

- Nel primo caso, importa il modulo intero come oggetto. Le sue funzioni rimangono "dentro" lo spazio dei nomi (namespace) del modulo. Il nome del modulo (matematica) diventa una sorta di etichetta che contiene tutto ciò che c'è dentro quel file. È quindi più sicuro e più chiaro.
- Nel secondo caso, importa tutte le funzioni e variabili pubbliche del modulo (matematica) direttamente nello spazio dei nomi corrente del programma che si sta scrivendo.

Creiamo Funzioni

Problema (potenza)

Realizzare un algoritmo che riceva in input un numero reale, la «base», e un numero intero positivo, l' «esponente», e calcoli e stampi l'elevamento a potenza. Ovviamente senza utilizzare eventuali operatori già preconfezionati del linguaggio adottato. Per esempio:

base = 3

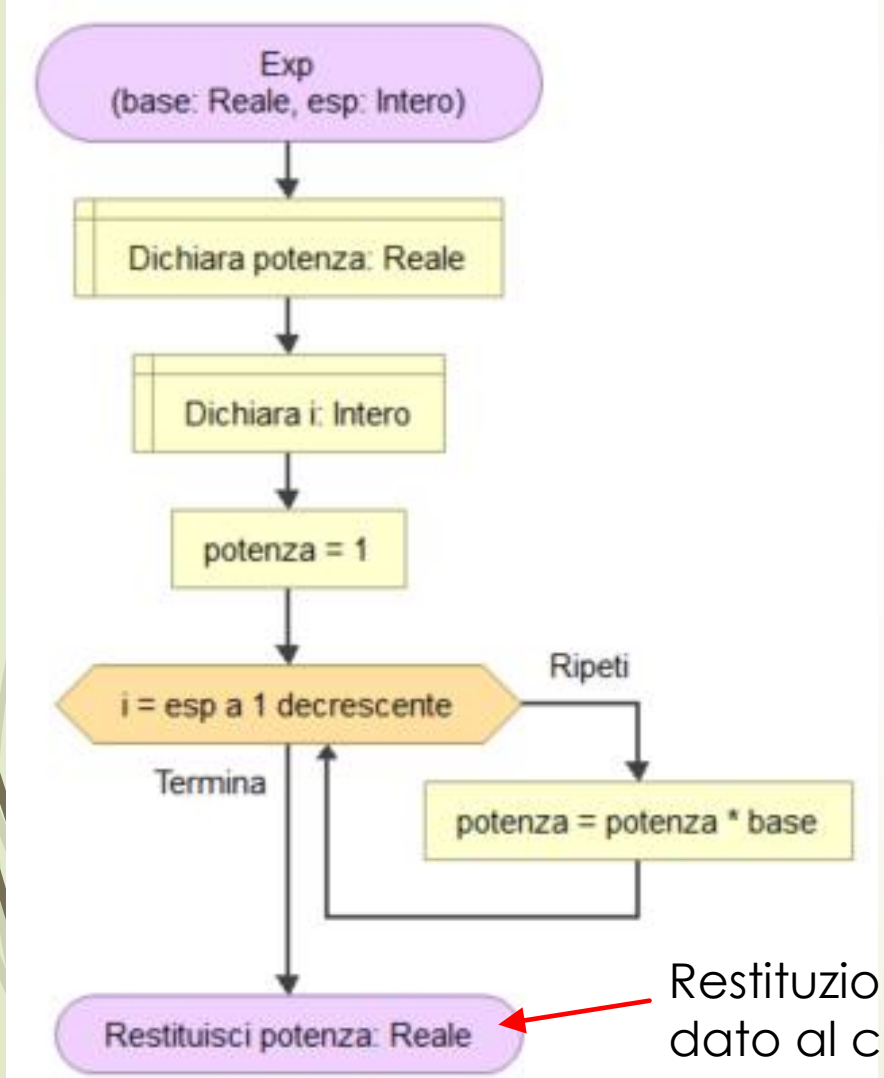
esponente = 2 **=>** **$3^2 = 9$**

Idea (informale)

L' «esponente» diventa la variabile che conta le volte che bisogna moltiplicare la «base» per ottenere il risultato. Raggruppiamo le istruzioni che svolgono l'operazione del calcolo dell'esponenziale dentro una **funzione**.

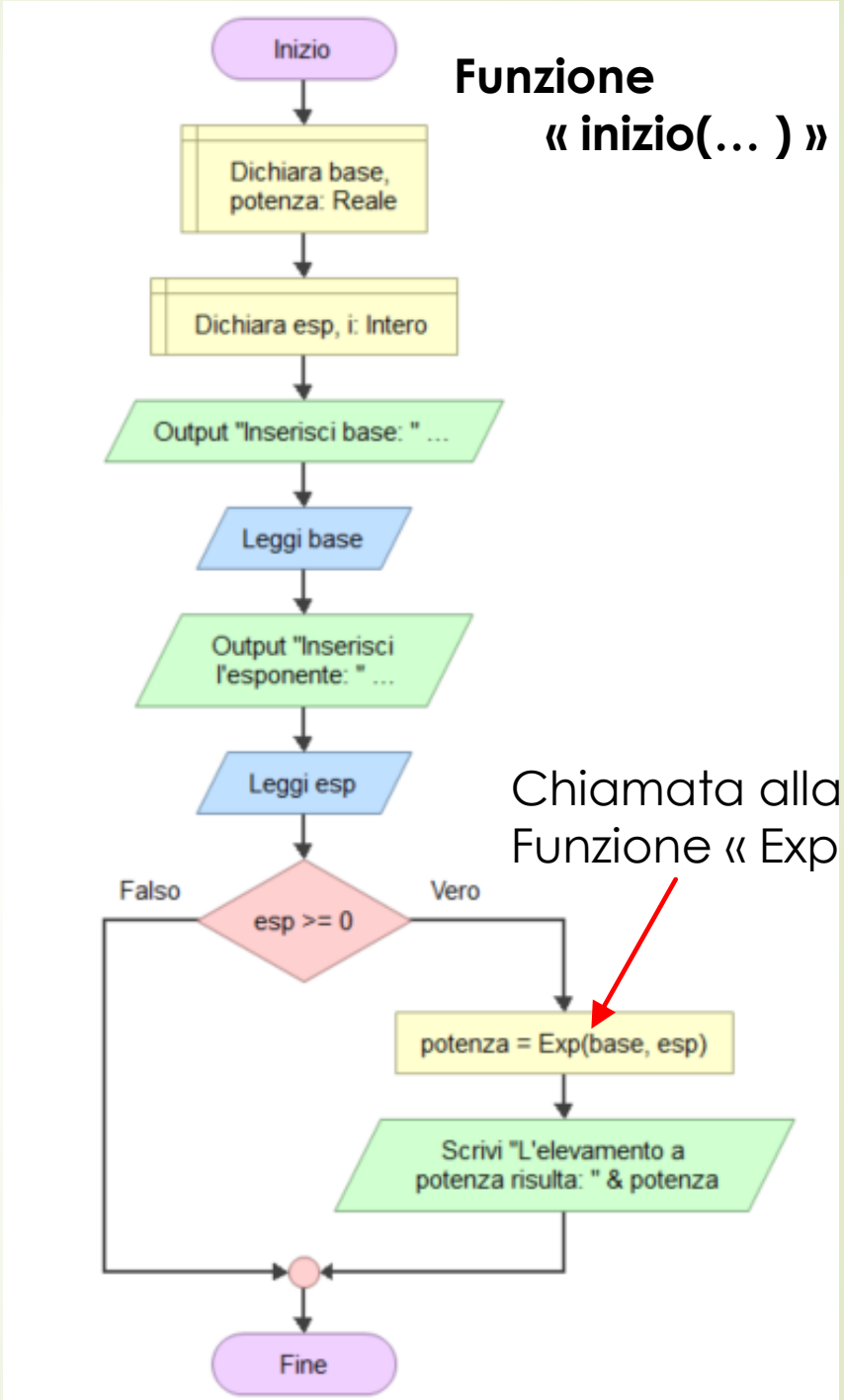
Ex. Potenza- Soluzione

Sottoprogramma: « Exp(…) »



Restituzione del dato al chiamante

Funzione « inizio(…) »



Chiamata alla Funzione « Exp(…) »

Ex. Potenza- Soluzione in Python

esponente_new.py > ...

```python
def exp(base, esp): # definizione di una 'funzione'
    potenza = 1
    for i in range(esp, 0, -1):
        potenza = potenza * base

    return potenza

# Main
print("Inserisci base: ", end='', flush=True)
base = float(input())
print("Inserisci l'esponente: ", end='', flush=True)
esp = int(input())
if esp >= 0:
    potenza = exp(base, esp) # richiama/esegue la funzione 'exp(..,..)'
    print("L'elevamento a potenza risulta: " + str(potenza))
```

Ex. Potenza

Frammento di codice della soluzione in Java.

```java
public class Power {

    public Power() throws IOException{
        double base = 0;
        int esp;
        String str;

        InputStreamReader input = new InputStreamReader(System.in);
        BufferedReader tastiera = new BufferedReader(input);

        System.out.print("Inserisci la 'base': ");
        str = tastiera.readLine();
        base = Double.parseDouble(str);

        System.out.print("Inserisci l' 'esponente': ");
        str = tastiera.readLine();
        esp = Integer.parseInt(str);
        System.out.print("");

        if(esp >= 0) {
            System.out.print("L'elevamento a potenza risulta: " + Exp(base, esp));
        }
    }

    public double Exp(double base, int esp) {
        double potenza = 1;
        int i;

        for(i = esp; i > 0; i--) {
            potenza = potenza * base;
        }
        return potenza;
    }

```

GRAZIE!