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

Strumenti utilizzati per scrivere algoritmi

Tecnologie Adottate

Strumenti utilizzati in questo modulo per realizzare semplici algoritmi:

- **Flowgorithm**: esegue progetti scritti in Flow-Chart, e consente anche di tradurre in altri linguaggi: pseudocodice, C++, PHP, Python, ecc. Per Windows: http://www.flowgorithm.org/download/
- **Python**: Linguaggio di programmazione semplice e potente, gratuito e open source
- **Visual Studio Code**: *Integrated Development Environment* per sviluppare velocemente software in diversi linguaggi.

Installare Python

Verifica se Python è già installato (s.o. Windows)

- Apri il **Prompt dei comandi** (CMD):
  Cerca "cmd" nel menu Start.
- Digita: python - - version
- Oppure: py - - version

Se compare un numero di versione (es. Python 3.12.3), Python è già installato. Se ricevi un errore, prosegui con l'installazione.

Download Python Installer

- **Scarica l'installer per il tuo S.O.**
- Vai sul sito ufficiale: https://www.python.org/downloads/
- Clicca su **"Download Python X.Y.Z"** (la versione più recente)

Avvia l'installer (es. Python X.Y.Z.exe), in una delle prime finestre compare: *Add Python to PATH* → cioè vi si chiede se volete aggiungere la possibilità di eseguire l'interprete Python da qualsiasi cartella vi troviate, quando usate un terminale CMD.

Poi procedete con l'installazione: Clicca su 'Install Now'...

Il programma installerà **Python** e **pip** (il gestore dei pacchetti).

Verificare l'Installazione

- Apri il Prompt dei comandi
- Digita: python --version
- Se appare la versione, l'installazione è avvenuta correttamente

Installazione Visual Studio Code

- Scarica da https://code.visualstudio.com/
- Esegui l'installer
  - Durante l'installazione, seleziona '**Add to PATH**' e '**Open with Code**'
- Completa l'installazione e avvia **VS Code**

Installare l'estensione Python

- Apri VS Code e vai su Estensioni (Ctrl+Shift+X)
  - Cerca 'Python' e installa quella di Microsoft
  - Offre sintassi colorata, debug e integrazione Jupyter

Configura l'Interprete

- Crea un nuovo file script.py
- Dal menu in basso seleziona l'interprete Python (es. Python 3.12)
- Scrivi: print('Ciao, Python da VS Code!')
- Esegui con **Ctrl + F5** per testare

Configura l'Interprete: Estensioni utili

Pylint: *linter* che serve a controllare automaticamente la qualità e la correttezza del codice.

Autopep8: Serve per sistemare automaticamente il codice in modo che rispetti le regole di stile definite nello standard PEP 8 (*Python Enhancement Proposal 8*), che è la "guida ufficiale" su come scrivere codice Python leggibile e ordinato.

Jupyter: Jupyter è un ambiente interattivo che permette di scrivere ed eseguire codice Python (e altri linguaggi) in celle, insieme a testo, immagini e grafici.Il suo nome deriva da JUlia, PYthon e R — i primi linguaggi supportati.

Code Runner: Code Runner è un'estensione per Visual Studio Code che permette di eseguire rapidamente frammenti di codice (snippet) o interi file in vari linguaggi di programmazione — tra cui Python, C, C++, Java, JavaScript, PHP, e molti altri. È pensato per chi vuole testare velocemente il proprio codice senza dover configurare un ambiente di debug completo.

Formalizzare la Soluzione del Problema: «linguaggi di progetto»

Programmazione Strutturata

Si fa risalire alla fine degli anni ‘60, quando si decretò (con Edsger Dijkstra) la fine del «salto incondizionato» (*goto*) come strumento fondamentale per la definizione degli algoritmi.

In particolare, con Jacopini e Bohm, si dimostrò che qualsiasi algoritmo può essere espresso con tre tipi di strutture di controllo: sequenza, selezione e ciclo.

La programmazione strutturata è costituita da **strutture di controllo** e **da funzioni (o metodi)** che vengono richiamate nell’ordine corretto a partire dalla funzione principale chiamata «**main**».

I linguaggi come il Pascal, C, Fortran, ecc. che permettono una programmazione strutturata, consentono di costruire programmi ordinati, basati sulle strutture di controllo e sull’organizzazione modulare del codice.

Selezione

```
if (condizione)
{
    //istruzioni da eseguire quando la «condizione» è «vera»
}
else
{
    //istruzioni da eseguire quando la «condizione» è «falsa»
}
```

Si possono avere anche blocchi di if annidati utilizzando:

« **if(cond1) … else if(cond2) …** »

Selezione in Python

if *condizione*:

    //istruzioni da eseguire quando la «condizione» è «vera»

else:

    //istruzioni da eseguire quando la «condizione» è «falsa»

}

Si possono avere anche blocchi di **if** annidati utilizzando:

```
if cond1:
    istruzioni_cond1
elif cond2:
    istruzioni_cond2
...
else:
    istruzioni_default      #nel caso nessuna delle precedenti elif si verifichi
```

Selezione Multipla

```
Switch (espressione)
{
    case valore1:
        //istruzioni da eseguire quando «espressione» = «valore1»
        break;
    case valore2:
        //istruzioni da eseguire quando «espressione» = «valore2»
        break;
    ….
    default: //istruzioni
        break;
}
```

Ripetizione: «While» e «do…While»

```
While (condizione)
{
    //istruzioni da eseguire quando «condizione» è «vera»
}
```

```
do
{
    //istruzioni da eseguire quando «condizione» è «vera», la prima
    //volta in ogni caso
} While (condizione)
```

Ripetizione: «for»

```
for (inizializzazione; condizione; aggiornamento)
{
    //istruzioni da eseguire quando «condizione» è «vera»
}
```

«inizializzazione» viene eseguita una sola volta prima di entrare nel ciclo for, mentre la «condizione» viene valutata tutte le volte prima di eseguire il blocco di istruzioni. Se è «falsa» le istruzioni non vengono eseguite. Al termini di ogni ciclo, viene eseguito «aggiornamento» e successivamente viene rivalutata la condizione.

Linguaggi Formali di Progetto

I linguaggi più usati nella programmazione strutturatai per definire un progetto per la soluzione algoritmica del problema sono:

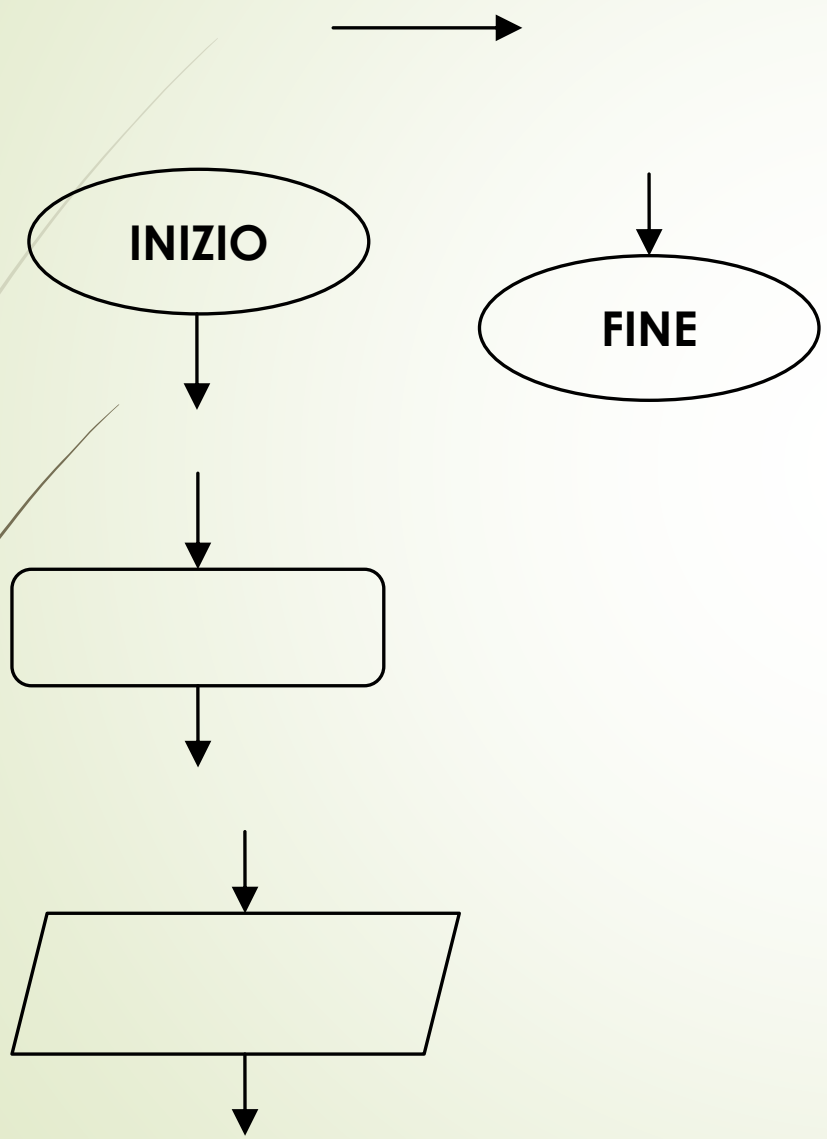
Pseudo-codice

- Vantaggi
  - Più diretto
- Svantaggi
  - Meno astratto
  - Interpretazione più complessa

Flow-Chart

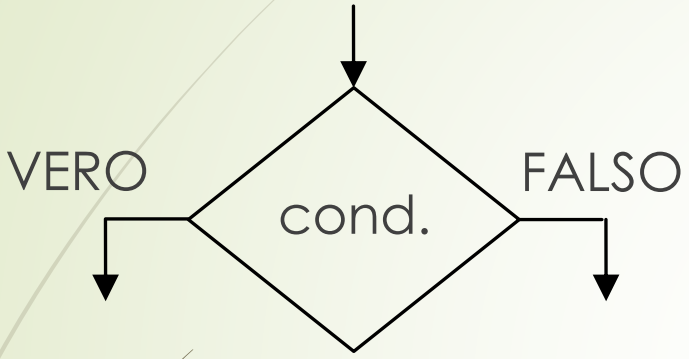
- Vantaggi
  - Più intuitivo perché grafico
  - Più astratto
- Svantaggi
  - Acquisire la semantica dei simboli grafici

Notazione Grafica – Flow Chart

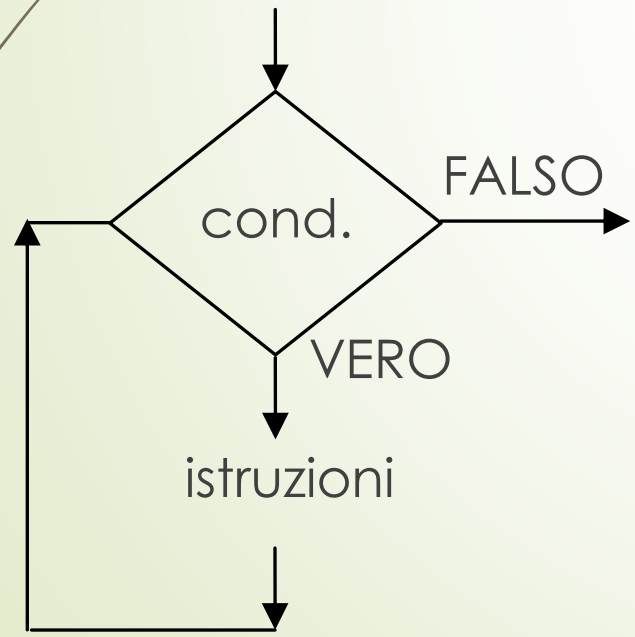


- Rappresenta la direzione del flusso del programma
- Indicano i punti di inizio e di fine dell'Algoritmo
- Indica un'elaborazione, per esempio un assegnazione
- Indica sia un **input** e sia un **output** , si assume per default che l'input dei dati avviene per tastiera e l'output è a video.

Notazione Grafica: semantica decisionale

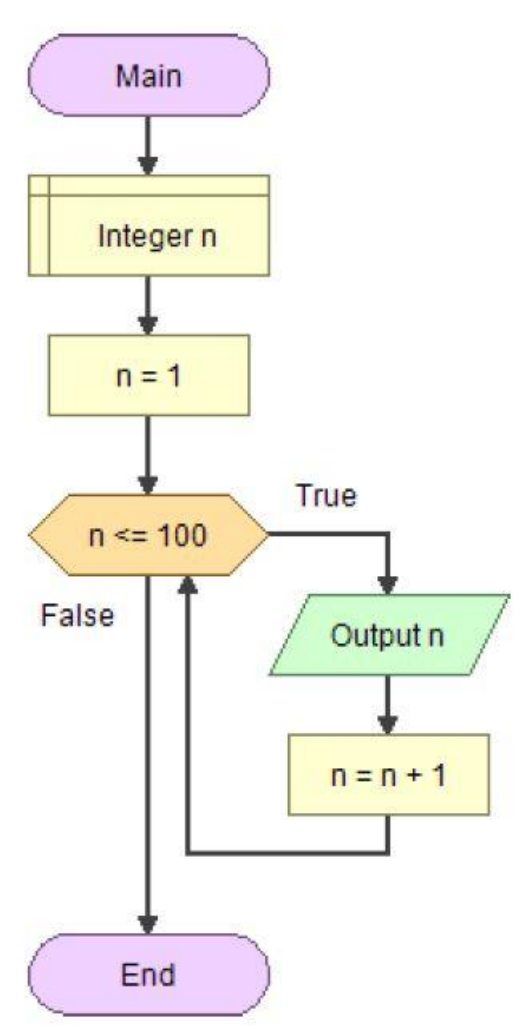


- **Simbolo di decisione** (**IF**): se la condizione «cond.» assume valore booleano «true» allora il flusso del programma percorre il ramo del VERO, altrimenti si passa sul ramo del FALSO.



- **Indica un ciclo/ripeti** (**while**, **for**, …), cioè si ripete il blocco di «istruzioni» fino a quando la «cond.» è VERA. Quando la «cond.» risulta falsa allora si esce dal ciclo, cioè si percorre il ramo del FALSO.

While: Flowgorithm e Python

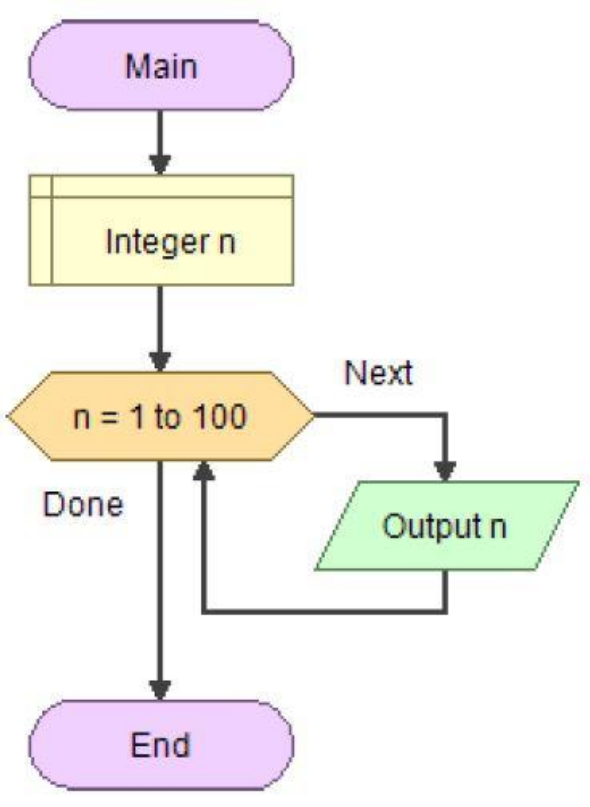


- **Ciclo While**: valuta una espressione booleana e, se vera, esegue le istruzioni contenute al suo interno. Cicla fino a quando la condizione diventa falsa.

Python

```
n = 1
while n <= 100:
    print(n)
    n = n + 1
```

For: Flowgorithm e Python

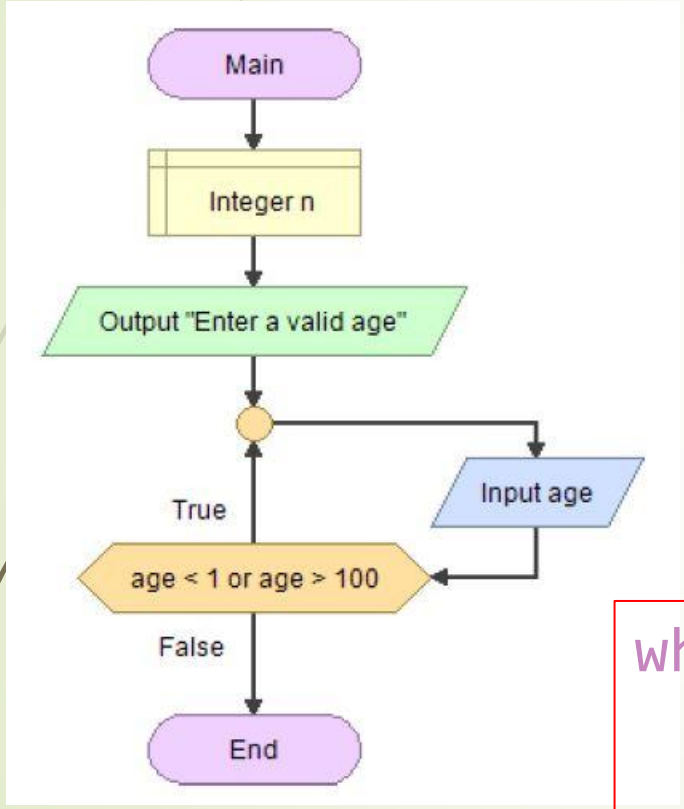


- **Ciclo for**: incrementa una variabile all'interno di un intervallo di valori assegnati.

Python

```
for n in range(101):
    print(n)
```

Do-While: Flowgorithm e Python



- Indica un ciclo DO. L'esempio mostra che si accetta solo un input valido, cioè se l'età è compresa tra 1 e 100, altrimenti si ripete il ciclo. Rispetto ad un ciclo «While» le istruzioni al suo interno vengono eseguite almeno una volta, prima di verificare la condizione.

Python

```
while True:
    age = int(input("Enter a valid age: "))
    if age >= 1 and age <= 100:
        break
```

Esempi con «strutture di controllo»

Progettare la soluzione

Problema (confronto)

Realizzare un algoritmo che riceva in input due numeri interi e determini quale dei due è più grande.

Soluzione informale

Si utilizza un'operazione di confronto tra i due numeri e in base al risultato si stampa in uscita il messaggio per l'utente.

Soluzione formale

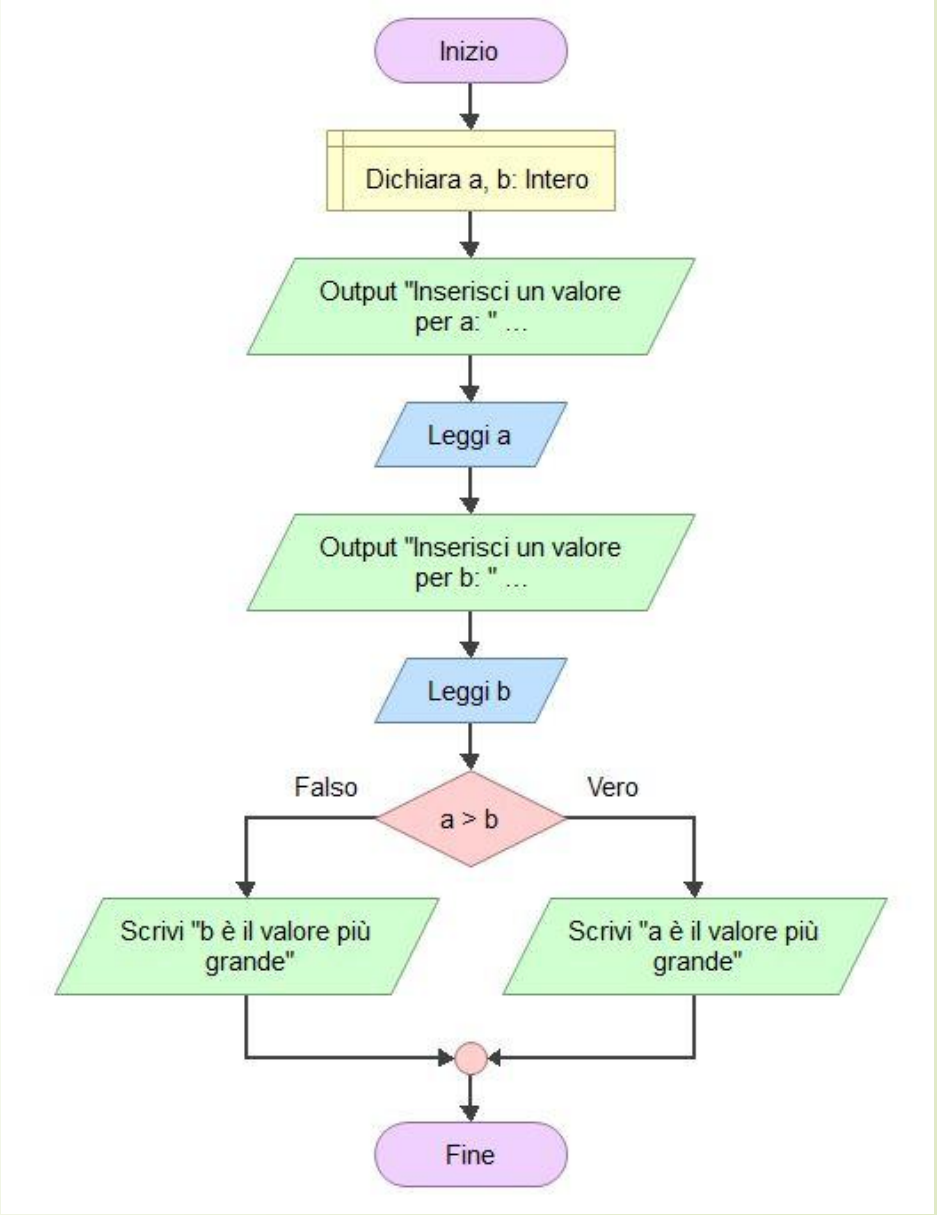
Con linguaggi di progetto: *pseudo-codice, Flow-Chart e Python*

Il progetto

pseudo-codice

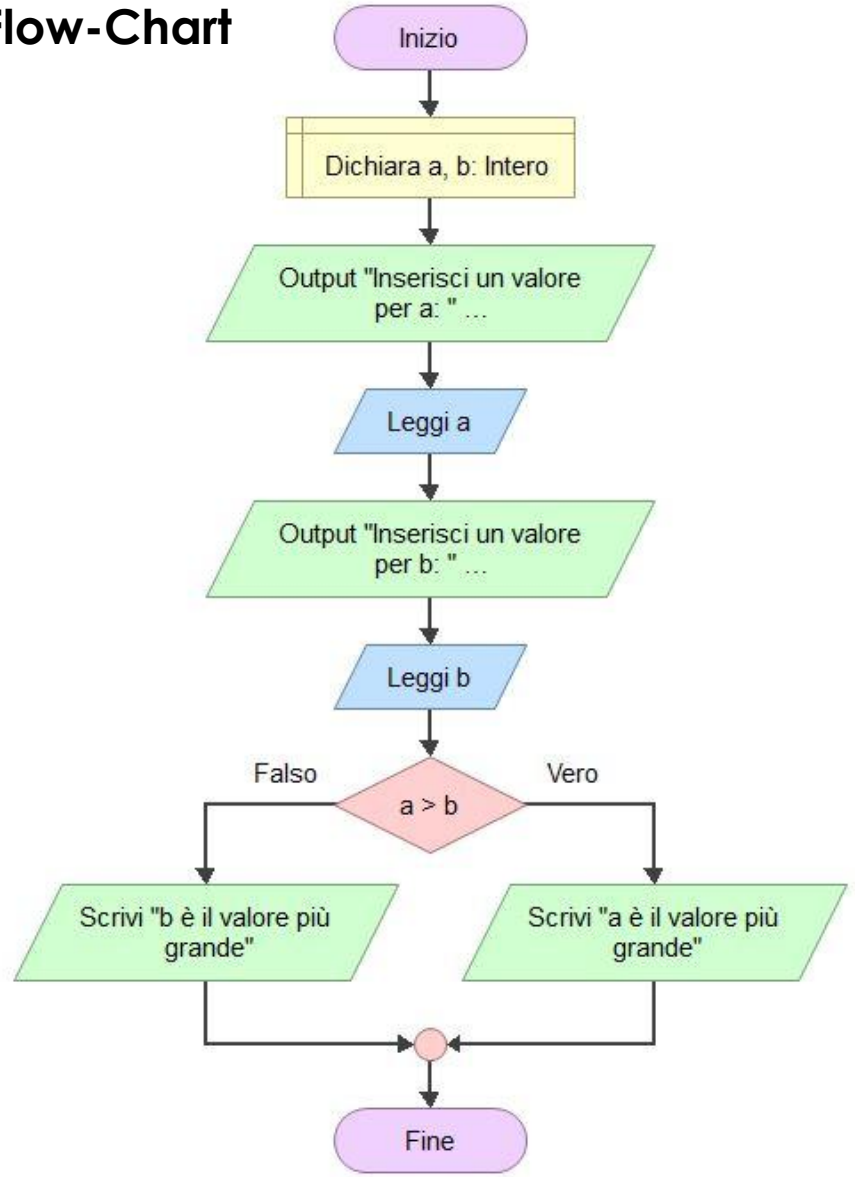
```
0   Funzione Inizio
1       Dichiara a, b: Intero
2
3       Scrivi "Inserisci un valore per a: "
4       Leggi a
5       Scrivi "Inserisci un valore per b: "
6       Leggi b
7       Se a > b
8           Scrivi "a è il valore più grande"
9       Altro
10          Scrivi "b è il valore più grande"
11      Fine
12  Fine
```

Flow-Chart



Dal progetto all'implementazione

Flow-Chart



Codice Python

ConfrontoInteri.py > ...

```python
# Dichiarazione delle variabili
numero1 = int(input("Inserisci il primo numero intero: "))
numero2 = int(input("Inserisci il secondo numero intero: "))

# Determinazione del maggiore
if numero1 > numero2:
    print("Il numero maggiore è:", numero1)
elif numero2 > numero1:
    print("Il numero maggiore è:", numero2)
else:
    print("I due numeri sono uguali.")

```

Progettare la soluzione

Problema

Realizzare un algoritmo che stampi il quadrato di N numeri reali inseriti dall'utente.

Soluzione informale

Si utilizza un'operazione di ripeti, all'interno del blocco di istruzioni da ripetere deve esserci l'input dell'utente e l'operazione di elevamento a quadrato.

Soluzione formale

Con linguaggi di progetto: *Flow-Chart e Python*

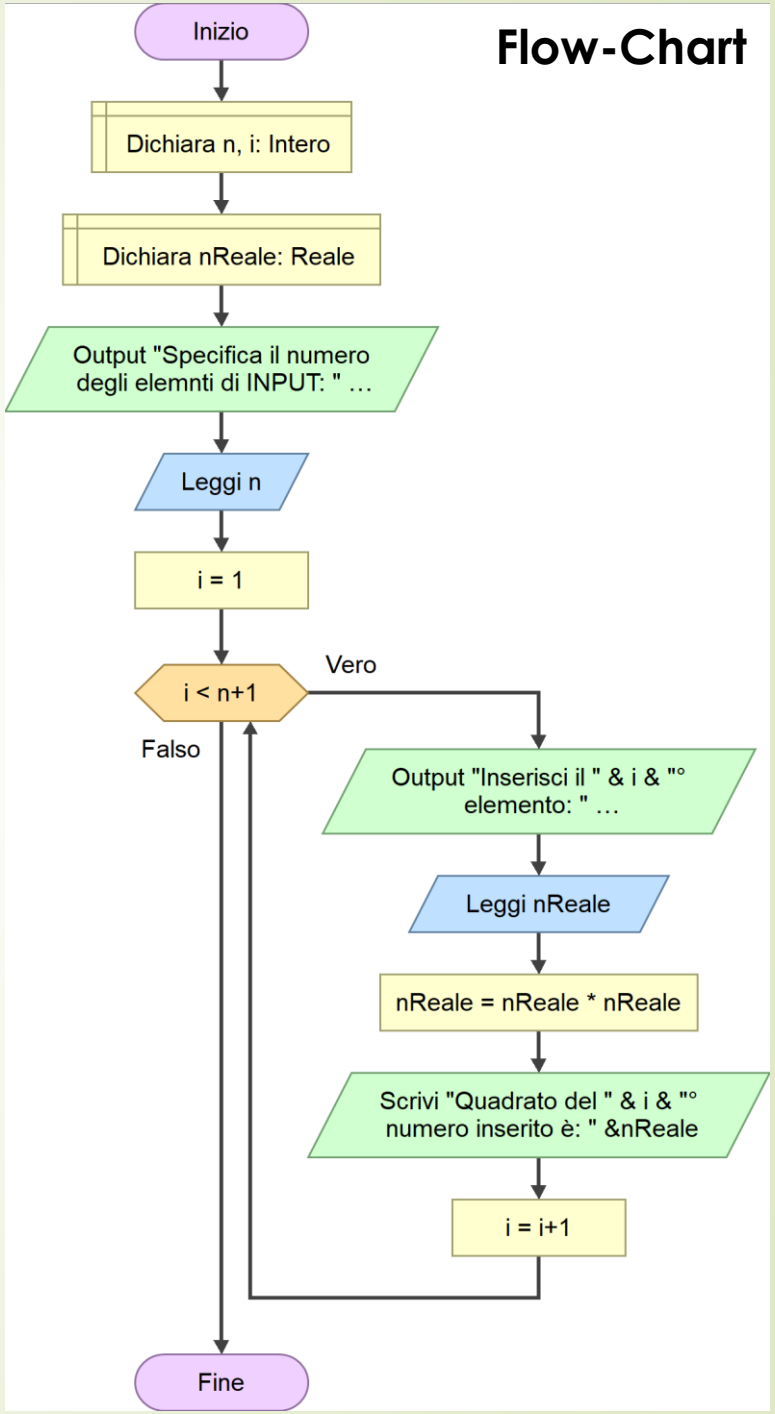
Il progetto

Codice Python

usoRipeti.py > ...

```python
n = int(input("Specifica il numero degli elementi di INPUT: "))
i = 1
while(i < n+1):
    print("Inserisci il ", i, "° elemento: ", end='', flush=True)
    nReale = float(input(""))
    nReale = nReale **2
    print("Quadrato del ", i, "° numero inserito è: ", nReale)
    i = i + 1

```



GRAZIE!