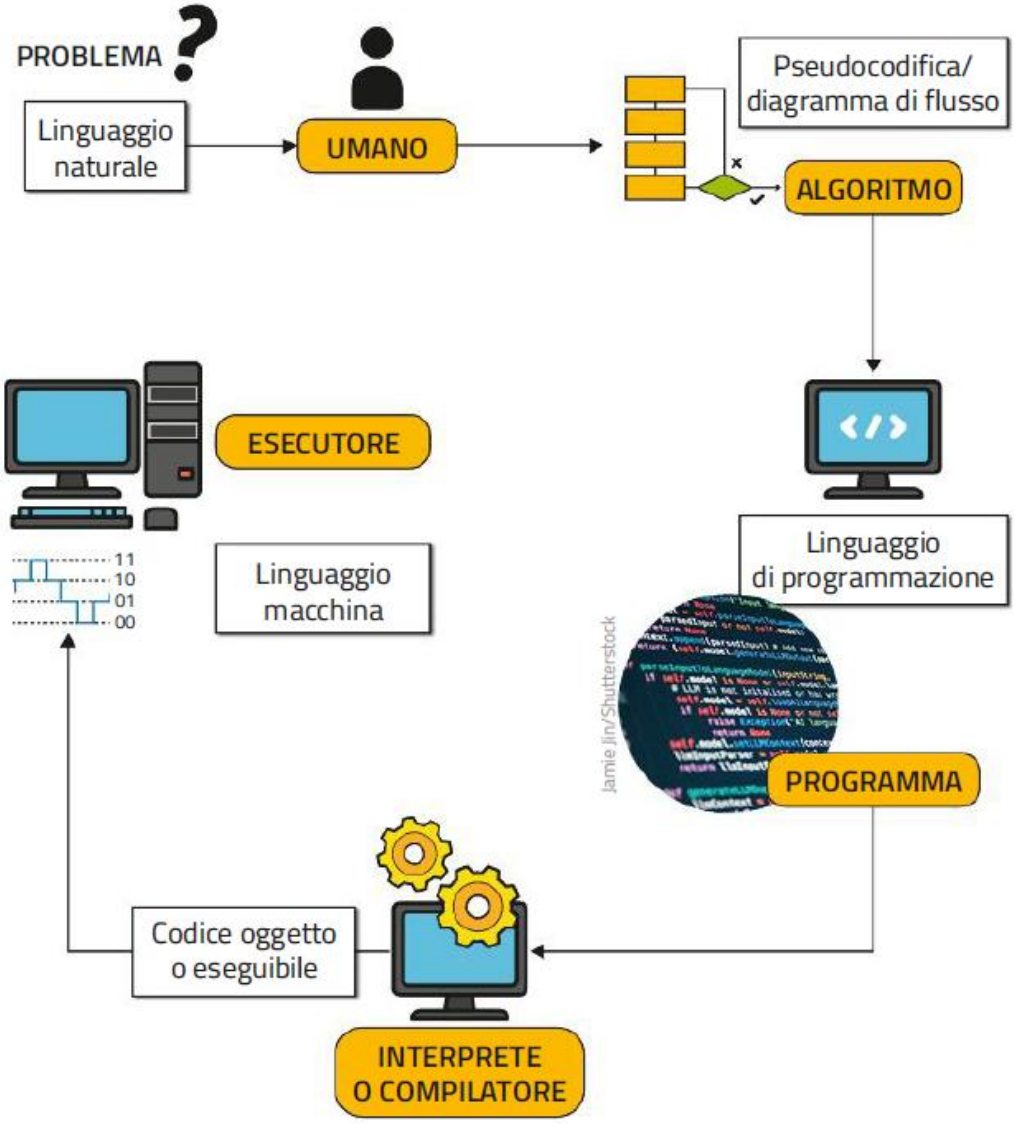
Cominciamo a programmare

Classe II

I crediti fotografici delle immagini presenti in questo PowerPoint sono riportati all'interno del volume *informatica per progetti – Programmare in Python*, di M. Fiore, A. Zanga, ISBN 9788808899477

Dal problema all'esecuzione



Linguaggi di programmazione

Nel corso degli anni i linguaggi di programmazione si sono molto evoluti, ma tutti sono costituiti da un insieme di comandi e strutture funzionali che permettono di tradurre in programmi eseguibili dal computer le istruzioni in pseudocodice degli algoritmi.

I linguaggi di programmazione «di basso livello», sono quelli il cui funzionamento è molto vicino all'hardware del computer (e molto lontano dalla logica del programmatore!): ad esempio Assembly.
I linguaggi di programmazione «di alto livello», sono quelli che consentono di scrivere codice indipendente dall'hardware del computer e che hanno caratteristiche che lo rendono sempre più semplice da utilizzare dal punto di vista del programmatore: ad esempio Pascal, C, Fortran e i più recenti C++, Java, Python.

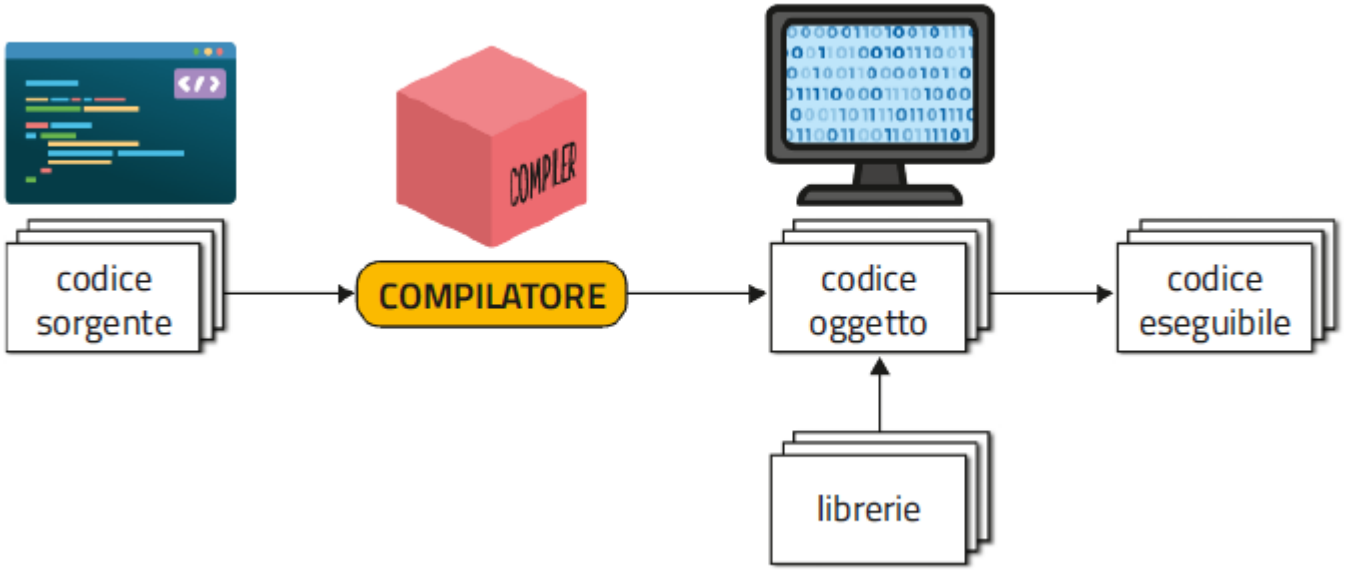
Compilatori e Interpreti

Il programma scritto in un linguaggio di programmazione, per poter essere eseguito dal computer (l'esecutore), ha necessità di essere tradotto in linguaggio macchina, cioè quel linguaggio che può essere compreso dal computer.
Esistono diversi programmi software che consentono questa traduzione da programma scritto in linguaggio di alto livello a programma scritto in linguaggio di basso livello, e si identificano principalmente in due categorie: compilatori e interpreti.

Un **compilatore** è un programma software che traduce una serie di istruzioni scritte in un linguaggio di alto livello (detto codice sorgente) in istruzioni di un altro linguaggio (detto codice oggetto).

Con una operazione successiva del compilatore, il codice oggetto viene tradotto in linguaggio macchina (codice eseguibile) ed è pronto per l'esecuzione.

Fasi della compilazione



Interprete

Un **interprete** è un programma software che è in grado di eseguire un programma scritto in un linguaggio di programmazione di alto livello, elaborando le istruzioni una alla volta e traducendole direttamente in linguaggio macchina così che siano eseguibili dal computer.

Un interprete, a differenza di un compilatore, non prevede le fasi di creazione del codice oggetto e poi del codice eseguibile, ma traduce direttamente le singole istruzioni del codice sorgente in istruzioni eseguibili dal computer, una dopo l'altra.

Python, usa un interprete ed è attualmente tra i più utilizzati al mondo proprio per le sue caratteristiche di versatilità e flessibilità.

Il linguaggio Python

Python è un linguaggio di alto livello creato nel 1991 da Guido Van Rossum.
Il suo nome nasce come omaggio a una serie televisiva britannica: *Monty Python's Flying Circus*.
Python è stato sviluppato a partire dal linguaggio di programmazione ABC, con l'intenzione di semplificarne notevolmente le strutture e ottenere un linguaggio che fosse molto vicino allo pseudocodice e avesse tutte le caratteristiche necessarie per sviluppare applicazioni di ogni genere.

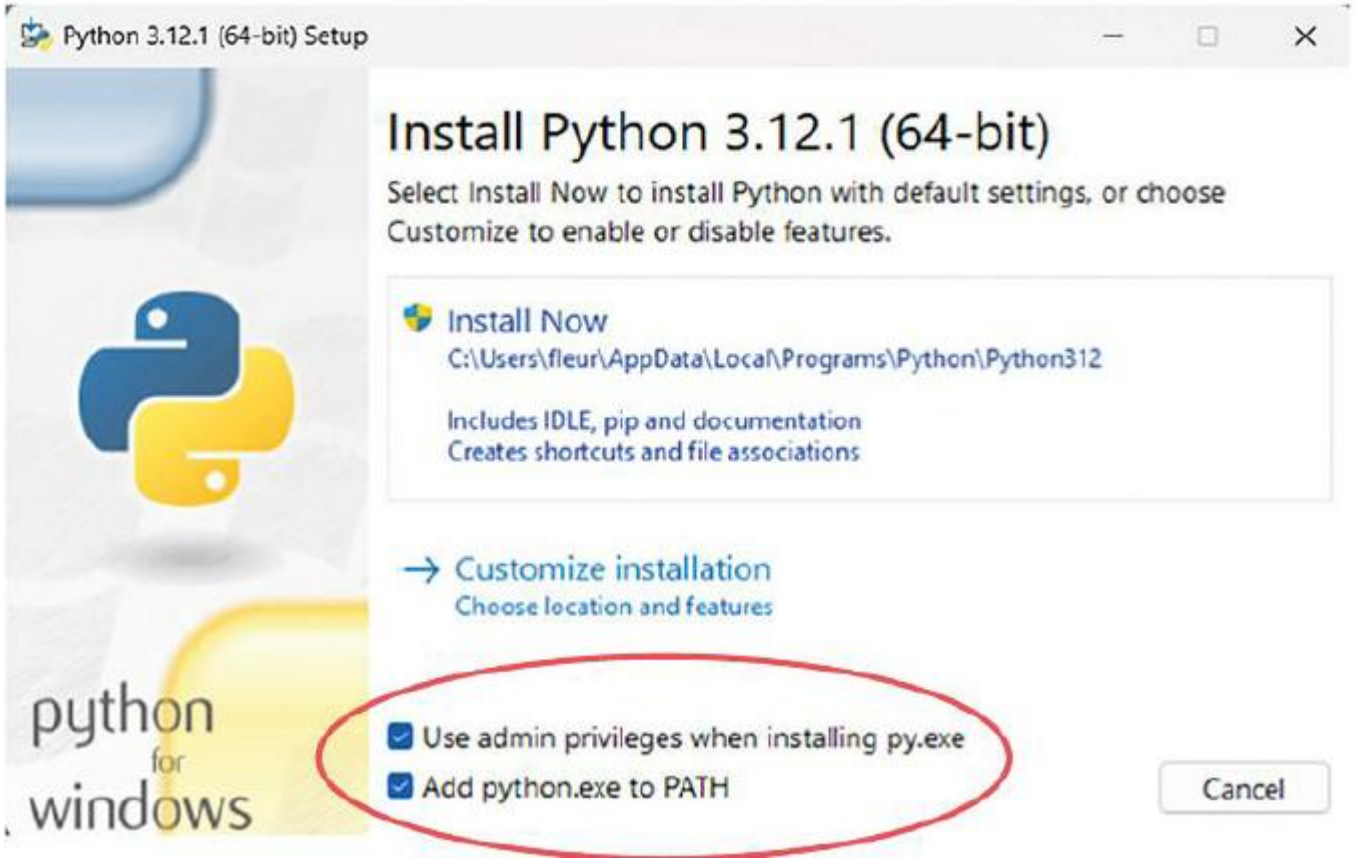
Python supporta infatti la programmazione strutturata, quella a oggetti, la programmazione web e la programmazione numerica, ed è adatto per lavorare su data science, applicazioni di intelligenza artificiale e molte altre ancora.

Attualmente è tra i linguaggi più utilizzati, è open source e può contare su una community di sviluppatori tra le più estese al mondo.

Il sito di riferimento per il linguaggio Python si trova alla URL
https://www.python.org

Installazione di Python

Dal sito ufficiale scaricare la versione più recente di Python e idonea al sistema operativo del proprio personal computer.

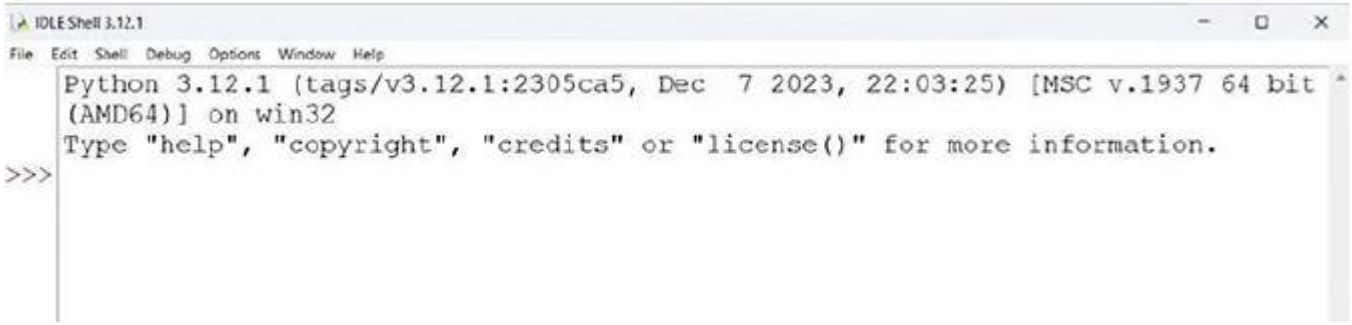


Usare l'interprete Python

L'interprete Python è utilizzabile in due modalità:

1. linea di comandi (modalità interattiva) IDLE Shell;
2. script.

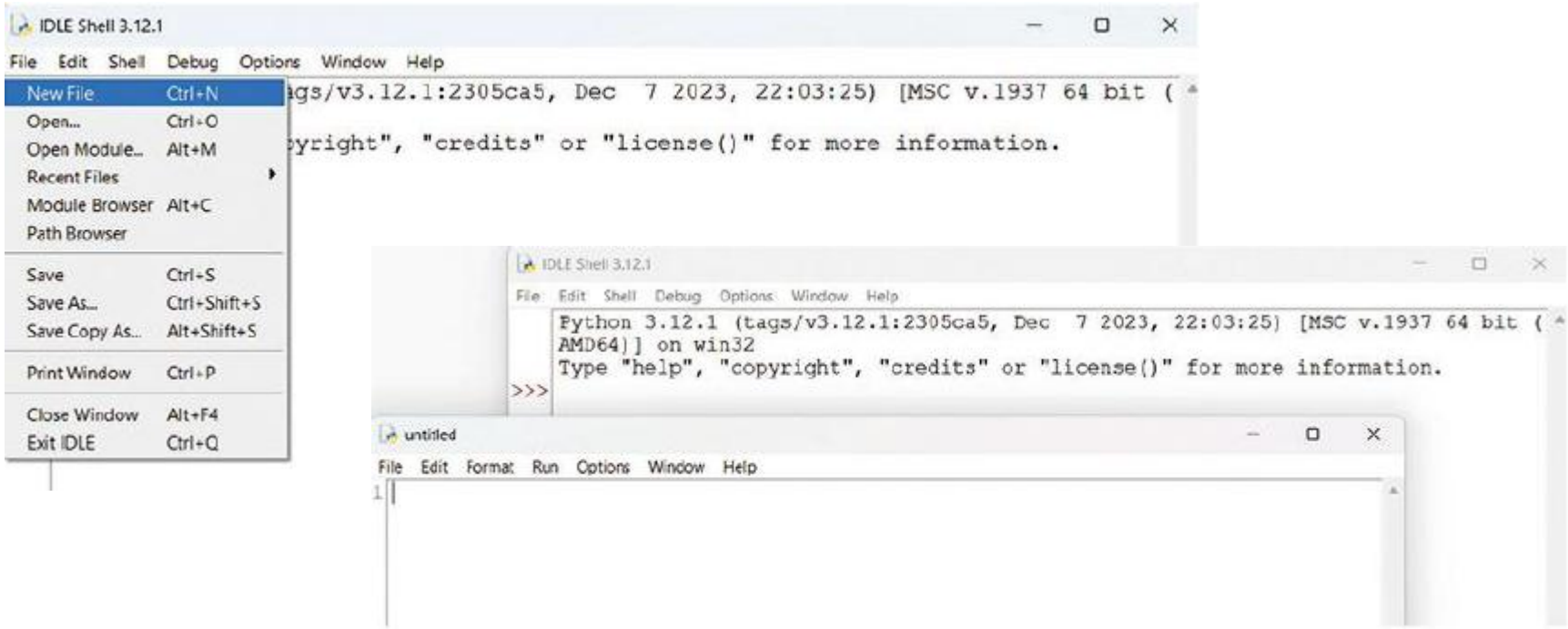
Per avviare Python in modalità «linea di comandi», bisogna avviare l'applicazione denominata IDLE (*Integrated Development and Learning Environment*).



Il prompt dei comandi è rappresentato dal simbolo >>> che compare a sinistra e che mostra la riga disponibile per inserire i comandi del linguaggio Python. Ogni comando scritto in una riga verrà eseguito da Python immediatamente.

Script

La modalità «linea di comandi» non è molto comoda quando vogliamo scrivere più righe di codice e memorizzarle per poterle utilizzare anche in futuro.
Python consente quindi la scrittura di script che contengono più istruzioni e che vengono salvati in file con estensione .py



Commenti

Quando si inizia a scrivere uno script Python è buona norma utilizzare i commenti per documentare il codice. I commenti sono molto utili e possono per esempio servire a:
1. illustrare lo scopo dello script che si sta sviluppando;
2. spiegare il significato di un blocco di istruzioni;
3. documentare il funzionamento di specifiche parti di codice.

Esistono due diverse modalità di implementazione per i commenti:
1. commento su una singola riga;
2. commento su più righe.

Per inserire un commento su una singola riga si usa il simbolo # (hashtag) all'inizio del commento.
Per inserire un commento su più righe vanno inseriti i caratteri ''' (tre apici singoli o doppi) all'inizio del commento e ''' (tre apici singoli o doppi) alla fine del commento.

Tipi di dati semplici

Per **tipo di dato** intendiamo l'insieme dei valori che il dato può assumere.
Python è un linguaggio di programmazione debolmente tipizzato, ovvero unlinguaggio per il quale non è necessario indicare esplicitamente il tipo di dato.
È l'interprete a decidere il tipo di dato, a seconda del valore che gli assegneremo.

I seguenti sono i quattro tipi di dato semplice che è possibile rappresentare in Python:

Tipo	Codice	Valore	Codice di esempio
Valore booleano	bool	{False, True}	`>>> soleggiato = True`
Numero intero	int	Insieme Z	`>>> prodotti = 42`
Numero in virgola mobile	float	Insieme R	`>>> sconto = 0.31`
Numero complesso	complex	Insieme C	`>>> fase = 1.23 + 3.50j`

Operatori aritmetici e logici

Per i tipi numerici:

Operatore	Operazione
`>>> 12 + 13` `25`	Somma
`>>> 12 - 13` `-1`	Sottrazione
`>>> 12 * 13` `156`	Moltiplicazione
`>>> 12 / 13` `0.9230769230769231`	Divisione
`>>> 12 // 13` `0`	Divisione intera
`>>> 12 % 13` `12`	Modulo, resto della divisione intera
`>>> 12 ** 13` `106993205379072`	Elevamento a potenza

Per i tipi booleani:

- not
- and
- or

```
>>> piove = True
>>> soleggiato = False
>>> not piove and soleggiato
False
```

Conversione di tipo o casting

Il risultato delle operazioni aritmetiche può essere un tipo di dato diverso da quelli di partenza: ad esempio la divisione di due int può restituire un float.
Il passaggio da un tipo di dato a un altro viene detto conversione o casting.
La conversione può essere implicita o esplicita.

- implicita: quando sommiamo due numeri di tipo diverso stiamo convertendo il tipo intero a virgola mobile prima di effettuare l'operazione
- esplicita: siamo noi a decidere il tipo del dato che vogliamo utilizzare.

Per decidere il tipo di numero che vogliamo usare è sufficiente scrivere il valore del dato tra parentesi, preceduto dal tipo di dato.

```
>>> float(78)
78.0
```

Per controllare il tipo di un dato o una variabile possiamo utilizzare type.

```
>>> type(42)
<class 'int'>
```

Input/Output

In Python i dati in input sono trattati come testo e di solito vengono convertiti subito dopo a un altro tipo, per l'output, invece, è necessario convertire gli altri tipi in testo.

Flusso delle operazioni:

1. Inserimento dei dati in input.
2. Conversione da stringa a un altro tipo di dato.
3. Esecuzione del programma.
4. Conversione da un altro tipo di dato a stringa.
5. Output per visualizzare il risultato.

```
>>> sconto = input("Inserire lo sconto da applicare: ")  #Assegnazione dell'input
    Inserire lo sconto da applicare al totale: 0.35
>>> type(sconto)                                        #Controllo del tipo in input
    <class 'str'>
>>> sconto = float(sconto)                              #Conversione a float
>>> type(sconto)                                        #Controllo del tipo convertito
    <class 'float'>
```

Formattazione output

Per utilizzare l'istruzione print con stringhe combinate con le variabili in modo da ottenere un output più complesso, dobbiamo effettuare la formattazione delle stringhe.
Possiamo formattare le stringhe in due modi:

1. usando l'istruzione format;
2. tramite l'interpolazione delle stringhe, dette anche f-strings.

Nel primo caso, l'istruzione format permette di inserire una variabile all'interno di una stringa tramite un segnaposto indicato da una coppia di parentesi graffe: {}.

```
>>> imc = 65 / 1.70 ** 2
>>> print("L'IMC è: {}".format(imc))
L'IMC è: 22.49134948096886
```

Possiamo anche utilizzare più di un segnaposto alla volta, i segnaposto saranno sostituiti nello stesso ordine delle variabili in format.

```
>>> massa  = 65
>>> altezza = 1.70
>>> imc = massa / altezza ** 2
>>> print("L'IMC per massa {} kg e altezza {} m è: {}".format(massa, altezza, imc))
L'IMC per massa 65 kg e altezza 1.7 m è: 22.49134948096886
```

f-strings

L'istruzione format viene sostituita con una lettera f anteposta agli apici della stringa in cui sostituiamo i nomi dei segnaposto direttamente con le variabili da visualizzare.

```
>>> massa  = 65
>>> altezza = 1.70
>>> imc = massa / altezza ** 2
>>> print(f"L'IMC per altezza {altezza} m e massa {massa} kg è: {imc}")
L'IMC per altezza 1.7 m e massa 65 kg è: 22.49134948096886
```

Sia con l'istruzione format sia con le *f-strings* possiamo ottenere un output ordinato, ma per renderlo più leggibile abbiamo bisogno di un modo per poter controllare come visualizzare il valore di ogni variabile: lo specificatore di formato.

Specificatore di formato	Tipo della variabile
:d	Rappresentazione di int.
:f	Rappresentazione di float con sei cifre dopo la virgola.
:a.bf	Rappresentazione di float con: a - Lunghezza minima da visualizzare; b - Numero di cifre da visualizzare dopo la virgola. Possiamo anche omettere a e utilizzare solo b.

Selezione: if else elif

Selezione ...	Pseudocodice	Codice in Python
... semplice senza sequenza finale	`SE condizione ALLORA:` `    Sequenza.`	`if condizione:` `    Sequenza.`

... semplice con sequenza finale	`SE condizione ALLORA:`
	`    Sequenza.`
	`ALTRIMENTI:`
	`    Sequenza finale.`
		`if condizione:`
		`    Sequenza.`
		`else:`
		`    Sequenza finale.`

... composta senza sequenza finale	`SE condizione 1 ALLORA:`
	`    Sequenza 1.`
	`ALTRIMENTI SE cond. 2 ALLORA:`
	`    Sequenza 2.`
	`...`
	`ALTRIMENTI SE cond. N ALLORA:`
	`    Sequenza N.`
		`if condizione 1:`
		`    Sequenza 1.`
		`elif condizione 2:`
		`    Sequenza 2.`
		`...`
		`elif condizione N:`
		`    Sequenza N.`

... composta con sequenza finale	`SE condizione 1 ALLORA:`
	`    Sequenza 1.`
	`ALTRIMENTI SE cond. 2 ALLORA:`
	`    Sequenza 2.`
	`...`
	`ALTRIMENTI SE cond. N ALLORA:`
	`    Sequenza N.`
	`ALTRIMENTI:`
	`    Sequenza finale.`
		`if condizione 1:`
		`    Sequenza 1.`
		`elif condizione 2:`
		`    Sequenza 2.`
		`...`
		`elif condizione N:`
		`    Sequenza N.`
		`else:`
		`    Sequenza finale.`

Selezione: match-case

Selezione composta con if-else

```
1  giorno = 3
2  if giorno == 1:
3     giorno = "Lunedì"
4  elif giorno == 2:
5     giorno = "Martedì"
6  elif giorno == 3:
7     giorno = "Mercoledì"
8  elif giorno == 4:
9     giorno = "Giovedì"
10 elif giorno == 5:
11    giorno = "Venerdì"
12 elif giorno == 6:
13    giorno = "Sabato"
14 elif giorno == 7:
15    giorno = "Domenica"
16 else:
17    giorno = "ERRORE"
18 print(f"Oggi è {giorno}!")


   OUTPUT:
   Oggi è Mercoledì!
```

Selezione composta con match-case

```
1  giorno = 3
2  match giorno:
3     case 1:
4        giorno = "Lunedì"
5     case 2:
6        giorno = "Martedì"
7     case 3:
8        giorno = "Mercoledì"
9     case 4:
10       giorno = "Giovedì"
11    case 5:
12       giorno = "Venerdì"
13    case 6:
14       giorno = "Sabato"
15    case 7:
16       giorno = "Domenica"
17    case _:
18       giorno = "ERRORE"
19 print(f"Oggi è {giorno}!")

   OUTPUT:
   Oggi è Mercoledì!
```

Iterazione definita: for

Pseudocodice

```
1  somma = 0
2  RIPETI PER i DA 1 A 3:
3      somma = somma + i
4  OUTPUT somma

   OUTPUT:
   6
```

Codice in Python

```
1  somma = 0
2  for i in range(1, 4):
3      somma += i
4  print(somma)

   OUTPUT:
   6
```

L'intervallo può essere implementato con l'istruzione range in diversi modi:

- range(n + 1) per valori da 0 a n;
- range(m, n + 1) per valori da m a n;
- range(m, n + 1, s) per valori da m a n, dove il valore successivo è ottenuto sommando s al valore precedente.

L'intervallo che costruiamo con l'istruzione range include sempre il valore di partenza ed esclude sempre il valore di arrivo.

Uso di range

Per i da ...	Pseudocodice	Codice in Python

... 0 a *n*	`RIPETI PER i DA 0 A 3:`
	`  OUTPUT i`
	
	`OUTPUT:`
	`0`
	`1`
	`2`
	`3`
		`for i in range(4):`
		`  print(i)`
		
		`OUTPUT:`
		`0`
		`1`
		`2`
		`3`

... *m* a *n*	`RIPETI PER i DA 1 A 3:`
	`  OUTPUT i`
	
	`OUTPUT:`
	`1`
	`2`
	`3`
		`for i in range(1, 4):`
		`  print(i)`
		
		`OUTPUT:`
		`1`
		`2`
		`3`

... *m* a *n*, con incremento 2	`RIPETI PER i DA 1 A 3:`
	`  OUTPUT i`
	`  i = i + 1`
	
	`OUTPUT:`
	`1`
	`3`
		`for i in range(1, 4, 2):`
		`  print(i)`
		
		`OUTPUT:`
		`1`
		`3`

... *m* a *n*, con decremento 1	`RIPETI PER i DA 3 a 0:`
	`  OUTPUT i`
	
	`OUTPUT:`
	`3`
	`2`
	`1`
		`for i in range(3, 0, -1):`
		`  print(i)`
		
		`OUTPUT:`
		`3`
		`2`
		`1`

Iterazione indefinita: while

Pseudocodice

```
1  somma = 0
2  i = 1
3  RIPETI MENTRE i < 4:
4      somma = somma + i
5      i = i + 1
6  OUTPUT somma

   OUTPUT:
   6
```

Codice in Python

```
1  somma = 0
2  i = 1
3  while i < 4:
4      somma += i
5      i += 1
6  print(somma)

   OUTPUT:
   6
```

L'iterazione indefinita con il while funziona esattamente come una selezione con l'if, con l'unica differenza che quando la condizione del while è vera la sequenza all'interno della struttura viene eseguita più di una volta, finché la condizione di iterazione non diventa falsa.