



Pattern Recognition  
and Applications Lab

# Le basi della Programmazione con il linguaggio Python

**Docente:** Ambra Demontis

**Anno Accademico:** 2024 - 2025

Corso di Laurea in Ingegneria Elettronica, Informatica e delle Telecomunicazioni



University of Cagliari,  
Italy

Department of Electrical and  
Electronic Engineering



# Il Linguaggio Python

Inventato nel 1989 da Guido Van Rossum (<https://gvanrossum.github.io//>).

- Supporta il paradigma di programmazione ad oggetti.
- Facile da imparare.
- Creare programmi complessi richiede scrivere poco codice.
- Tante librerie a disposizione (machine learning, servizi web etc.).
- Comunità di sviluppatori molto attiva.

# Il Linguaggio Python

Viene generalmente definito un “linguaggio interpretato”.

I linguaggi interpretati sono quelli nei quali non vi è un compilatore che lo converte in linguaggio assembly ma un interprete che interpreta ed esegue il codice scritto nel linguaggio di programmazione originale.

# Il Linguaggio Python

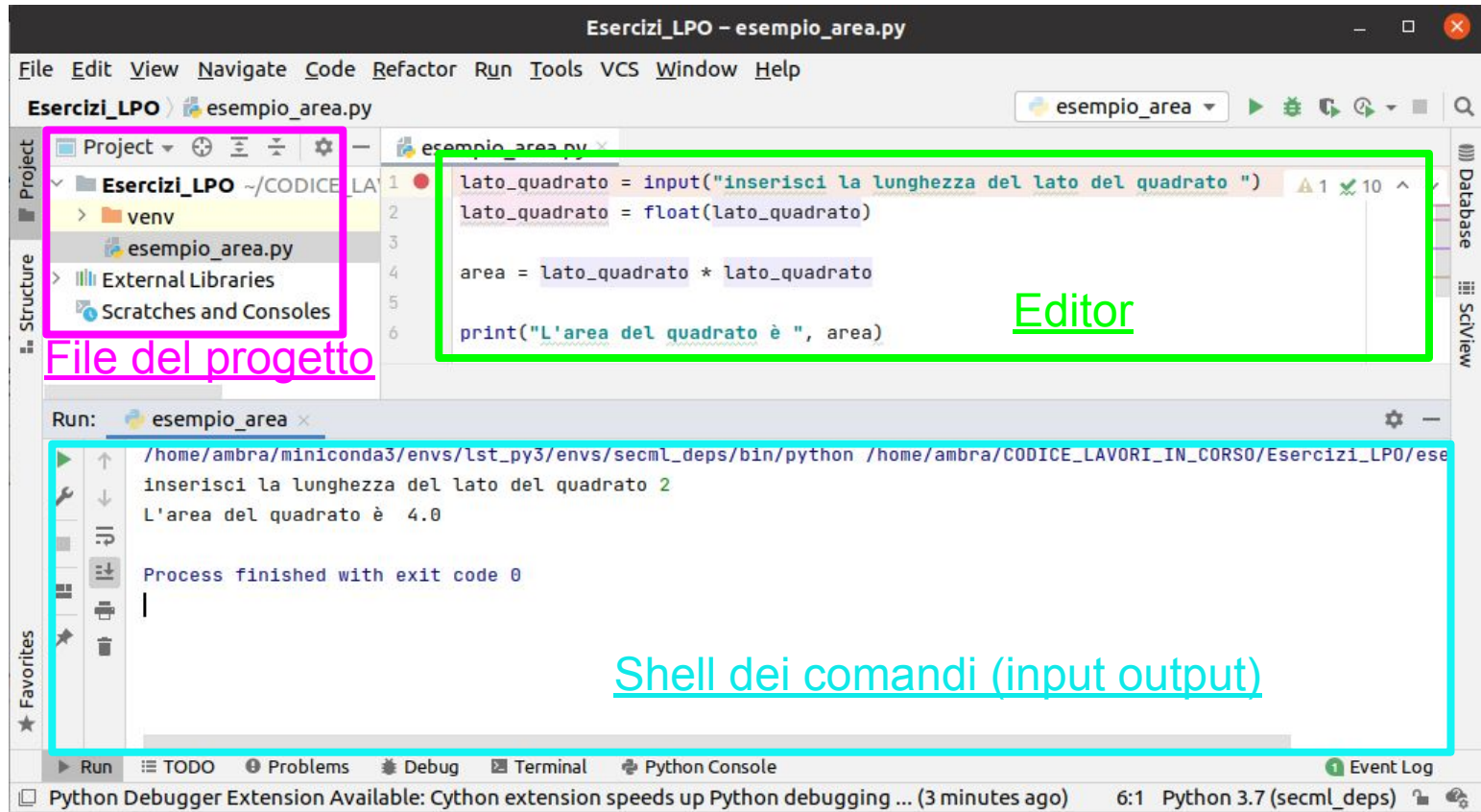
Viene generalmente definito un “linguaggio interpretato”.

**Tuttavia**, in Python, in realtà, **il codice non viene convertito direttamente in linguaggio macchina...**

Alla prima esecuzione del codice viene trasformato in *linguaggio bytecode* (con livello di astrazione intermedio tra linguaggio di programmazione e linguaggio macchina) che viene ri-utilizzato per le esecuzioni successive (troverete un file \*.pyc).

**Il bytecode viene automaticamente aggiornato ogni volta che vengono apportate modifiche al programma.**

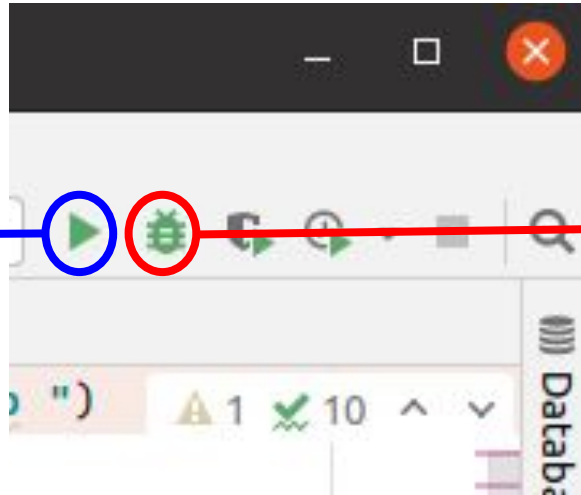
# L'ambiente di Sviluppo Pycharm



# L'ambiente di Sviluppo Pycharm

Tasto per  
l'esecuzione del  
programma.

Nb: Il file del  
programma deve  
avere estensione  
.py.



Tasto per lanciare il  
debugger.

(Permette  
l'esecuzione del  
programma passo  
passo).

# Le Basi di Python

In queste slide vedremo come funzionano in Python:

- Tipi di dato
- Operatori aritmetici
- Operatori di confronto (relazionali)
- Operatori booleani (logici)
- Variabili
- Istruzione di assegnamento
- Utilizzo di funzioni predefinite (conversioni di tipo, input e output di valori)
- Istruzione condizionale
- Istruzione iterativa

# Tipi di Dato

In Python esistono diversi tipi di dato. Quelli che utilizzeremo per ora sono:

- numeri interi, es: `5 7`
- numeri frazionari, es: `5.6 1.5e3 5.0 0.3`  
(`1.5e3` è uguale a  $1.5 \times 10^3$ ).  
Nb: Tutti gli esponenziali sono considerati frazionari (anche `1e3`)  
(gli ultimi numeri due si possono anche scrivere come `5. e .3`).
- stringhe (sequenze di caratteri), es: `"casa"`  
(Le stringhe vanno racchiuse tra apici)



# Tipi di Dato

- Valori booleani:

Nello specifico: `True` o `False`

NB: Python è case sensitive e la lettera maiuscola è necessaria affinché Python riconosca che si tratta del valore booleano `True` (`False`)

- Nessun valore:

`None`

# Operatori Aritmetici

<b>simbolo</b>	<b>operatore</b>
+	somma
-	sottrazione
*	moltiplicazione
/	divisione
//	divisione (quoziente intero)
%	modulo (resto della divisione)
**	elevamento a potenza

# Tipi dei Risultati

Per l'operatore divisione / il risultato è sempre un numero frazionario.

Es:

8./4 risultato: 2.0

8/4 risultato: 2.0

Per tutti gli altri il risultato è:

- intero se gli operandi sono due numeri interi
- frazionario se almeno uno dei due operandi è frazionario

Es:

3 \* 2. risultato: 6.0

3\*2 risultato: 6

# Operatori Aritmetici

L'operatore **elevamento a potenza** `**` segue questa regola.

`5 ** 2`. risultato: 25.

`5 ** 2` risultato: 25

Fa eccezione il caso in cui l'esponente sia un numero negativo. In quel caso, anche se gli operandi sono entrambi numeri interi, il risultato sarà di tipo frazionario.

`5 ** -2` risultato: 0.04

# Operatori Aritmetici

L'operatore `//` restituisce la parte intera della divisione del primo numero per il secondo.

Es:

`5 // 2` risultato: `2`

`10. // 2` risultato: `5.`

# Operatori Aritmetici

L'operatore **modulo** % restituisce il resto della divisione del primo numero per il secondo.

Es:

5 % 2 risultato: 1 (il risultato della divisione è 2 con resto 1)

10. % 2 risultato: 0. (il risultato della divisione è 5 con resto 0)

# Operatori Aritmetici

**Gli operatori aritmetici si possono applicare a dati di tipo booleano.**

In quel caso i dati di tipo booleano vengono interpretati come segue:

True viene considerato 1

False viene considerato 0

# Operatori Aritmetici

**Gli operatori aritmetici si possono applicare a dati di tipo:**

- Intero;
- Frazionario;
- Booleano.

2 + 2. + True risultato 5.

Non alle stringhe!



# Operatori di Confronto

simbolo	operatore
==	uguaglianza
!=	diversità
<	minore di
>	maggiore di
<=	minore o uguale di
>=	maggiore o uguale di

# Operatori di Confronto

**Il risultato di un'operazione che contiene un operatore di confronto è un valore booleano (True o False).**

Un numero intero e uno frazionario vengono considerati uguali se il loro valore è uguale.

Es: `3 == 3.0` il risultato è `True`

Mentre se proviamo a confrontare un valore intero o frazionario con una stringa il valore sarà False

Es: `3 == "3"` il risultato è `False`

# Operatori di Confronto

**Gli operatori == e != vengono spesso applicati anche alle stringhe.**

Due stringhe vengono considerate uguali se sono composte esattamente dagli stessi caratteri (compreso il carattere spazio).

Es:

"casa" == "casa " il risultato è False

"casa" == "casa" il risultato è True

# Operatori Booleani

In Python la loro sintassi è `and`, `or`, `not`.

Es: `True` and `False`, il risultato è `False`

La tavola di verità:

<b>x</b>	<b>y</b>	<b>x and y</b>	<b>x or y</b>	<b>not x</b>
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

# Variabili

In un programma è spesso utile **memorizzare dei dati in celle di memoria**.

Nei linguaggi ad alto livello ci si riferisce a queste celle di memoria con nomi simbolici scelti dal programmatore.

# Come Scegliere il Nome delle Variabili?

Può essere composto da uno o più dei seguenti caratteri:

- lettere minuscole e maiuscole (anche accentate)

[Python è case-sensitive *A* e *a* sono due variabili differenti]

- cifre

- il carattere \_ (underscore)

**Non deve iniziare con una cifra!**

esempio: 2lato non è un nome ammesso

**Non deve coincidere con i nomi predefiniti delle istruzioni e di altri elementi del linguaggio**

esempio: `if`, `while`, `print`

# Come Scegliere il Nome delle Variabili?

Non deve coincidere con i nomi predefiniti delle istruzioni e di altri elementi del linguaggio

esempio: `if`, `while`, `print`

Quindi devo conoscere a memoria tutte le istruzioni di Python per essere sicura/o di non scegliere uno di quelli?

Fortunatamente no..

L'ambiente di sviluppo vi segnala gli elementi del linguaggio colorandoli di un colore differente appena vengono scritti.

# Come Scegliere il Nome delle Variabili?

Posso scegliere qualsiasi nome rispetti le caratteristiche viste prima?

No..

Il principale scopo delle variabili è permettere ad altri programmatori/utilizzatori di capire cosa abbiamo memorizzato in quella cella di memoria. I nomi devono essere **esplicativi**.

A non è un nome di variabile accettabile.

lunghezza\_lato\_maggiore è un nome di variabile accettabile.



# Convenzioni Sulle Variabili

Spesso le variabili sono formate da più parole ma non possono contenere spazi.

Esistono due convenzioni:

- **CamelCase:** le parole sono attaccate e iniziano con la lettera maiuscola  
Es: LunghezzaLatoMaggiore
- **lowercase\_separated\_by\_underscores**  
Es: lunghezza\_lato\_maggiore

**In Python, per le variabili si utilizza la seconda, la prima viene utilizzata per i nomi delle classi.**

# Creazione di Variabili e Istruzione di Assegnamento

**In Python una variabile viene creata assegnandogli un valore.**

Il valore viene assegnato alle variabili utilizzando l'istruzione di assegnamento. Viene prima valutata l'espressione, poi il risultato viene assegnato alla variabile.

## **Sintassi:**

**<variabile> = <espressione>**

Es: `area_triangolo = 2 + 2`

Crea la variabile *area\_triangolo* e gli assegna il valore 4.

# Istruzione di Assegnamento

Ad una variabile alla quale è stato assegnato un valore, **può essere successivamente assegnato un valore di un tipo di dato differente.**

```
area_triangolo = 2
```

```
..
```

```
area_triangolo = 2.3
```

# Creazione di Variabili e Istruzione di Assegnamento

Ovviamente **tutti gli operatori che abbiamo visto possono essere applicati anche a delle variabili**. In quel caso prima verrà sostituito il nome della variabili con il loro valore e poi viene calcolato il valore dell'espressione.

Es:

base = 2

altezza = 4

area\_triangolo = base \* altezza / 2

L'espressione diventa:

area\_triangolo = 2 \* 4 / 2 e il risultato è 4

# Utilizzo di Funzioni Predefinite - Conversione di Tipo

La **sintassi** per **richiamare una** qualsiasi **funzione** è la seguente:

**<funzione>(<argomento1>, <argomento2>..)**

Un esempio di funzioni predefinite sono quelle per **convertire i tipi di dato**.

La funzione:

**str**(<argomento>) converte l'argomento in una stringa e lo restituisce

**int**(<argomento>) converte l'argomento in intero

**float**(<argomento>) converte l'argomento in numero frazionario

Ad esempio il risultato di: **str(3.0)** è la stringa **"3"**

# Utilizzo di Funzioni Predefinite - Output (Print)

Una delle funzioni predefinite di Python è la ***print***, che stampa a schermo tutti gli argomenti ricevuti. Ad esempio:

```
lato = 3  
print("La lunghezza del lato è ", lato)
```

Stampa a schermo: La lunghezza del lato è 4

# Utilizzo di Funzioni Predefinite - Input

Un'altra funzione predefinita è la funzione ***input***.

Questa funzione permette di far inserire all'utente un valore in input.

Es:

```
nome = input("inserisci il nome dello studente ")
```

Stamperà a schermo il messaggio: inserisci il nome dello studente e memorizzerà nella variabile *nome* il valore inserito dall'utente.

# Utilizzo di Funzioni Predefinite - Input

NB: qualsiasi valore verrà memorizzato come stringa!

Quando vogliamo che l'utente inserisca un valore numerico, riceveremo comunque una stringa e dovremo ricordarci di convertirla.

Es:

```
lato = input("inserisci la lunghezza del lato del quadrato ")
```

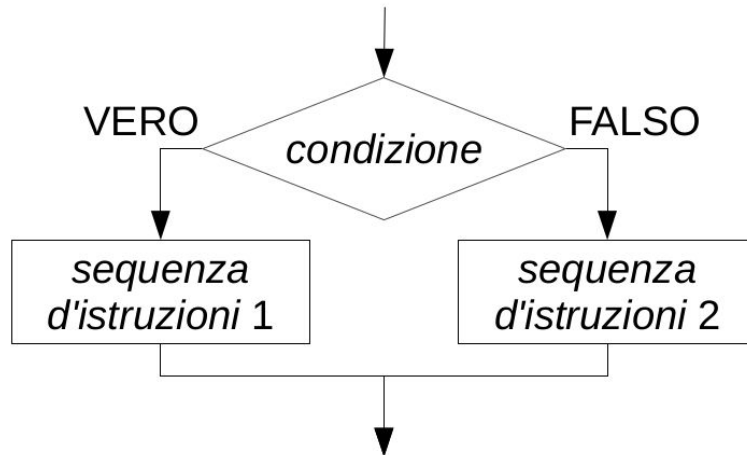
```
lato = float(lato)
```



# Istruzione Condizionale

Permette di eseguire istruzioni differenti a seconda del valore di un'espressione.

Diagramma a blocchi:



Sintassi:

```
If <espressione>:  
    <sequenza_istr_1>  
else:  
    <sequenza_istr_2>
```

# Istruzione Condizionale

Esempio:

```
numero = 4
```

```
if numero % 2 == 0:  
    numero_pari = True  
else:  
    numero_pari = False
```

Se l'espressione è vera, la variabile `numero_pari` assumerà il valore booleano `True` altrimenti `False`.

# Istruzione Condizionale

Quando in un ramo dell'if c'è più di una istruzione vengono semplicemente scritte una sotto l'altra. Ad esempio:

```
numero_1 = 1
```

```
numero_2 = 2
```

```
if numero_1 == numero_2:
```

```
    numero_1 = numero_1 + 1
```

```
    numero_2 = numero_2 + 1
```

# Istruzione Condizionale

Quando in un ramo dell'if c'è più di una istruzione vengono semplicemente scritte una sotto l'altra. Ad esempio:

```
numero_1 = 1
```

```
numero_2 = 2
```

```
if numero_1 == numero_2:
```

```
    numero_1 = numero_1 + 1
```

```
    numero_2 = numero_2 + 1
```

Come fa l'interprete a capire quali istruzioni fanno parte del ramo "vero" dell'if e quali no?

Se i due numeri sono uguali li incrementa entrambi, altrimenti non fa nulla.

# Istruzione Condizionale

L'interprete capisce quali istruzioni appartengono ai rami dell'if grazie **all'indentazione** (al numero di **rientri**).

```
numero_1 = 1
```


```
numero_2 = 2
```

```
if numero_1 == numero_2:
```

```
    numero_1 = numero_1 + 1
```

```
    numero_2 = numero_2 + 1
```

Queste due istruzioni sono scritte con un rientro rispetto all'if, quindi fanno parte del primo ramo dell'if.



# Istruzione Condizionale

Stessa cosa per gli if con più rami.

if <espressione>:  
    <istruzione1>

else:  
    <istruzione2>

<istruzione3>

Le istruzioni che si trovano rientrate rispetto all'if e prima dell'else vengono eseguite quando l'espressione è vera

Le istruzioni che si trovano rientrate rispetto all' else vengono eseguite quando la condizione è falsa

Le istruzioni verticalmente in linea con l'istruzione if, vengono eseguite sempre dopo che sono state eseguite le istruzioni in uno dei due rami dell'if.

# Istruzioni Condizionali Annidate

Esempio di sintassi nel caso in cui ci sia un if nel ramo True dell'if esterno.

```
if <espressione>:  
    if <espressione2>:  
        <istruzione1>  
    else:  
        <istruzione2>  
else:  
    <istruzione3>  
  
<istruzione4>
```

## Esercizio sull'Uso delle Istruzioni Condizionali

Creare un programma Python che chieda all'utente di inserire un numero e stampi a schermo il valore assoluto del numero inserito.



# Esercizio sull'Uso delle Istruzioni Condizionali

## Soluzione:

```
n = float(input("Inserire un numero: "))
```

```
if n < 0:
```

```
    n = n * -1
```

```
print("Il valore assoluto è", n)
```

# Istruzioni Iterative - Ciclo While

Il ciclo while esegue il codice **fino a che** il valore dell'espressione è vero.

La sintassi è:

```
while <espressione>:  
    <istruzione>
```

Fino a che l'espressione è vera esegue le istruzioni indentate rispetto all'istruzione while.

# Istruzioni Iterative - Ciclo While

Il ciclo while esegue il codice **fino a che** il valore dell'espressione è vero.

Esempio:

```
contatore = 0
```

```
while contatore < 3:
```

```
    print(contatore)
```

```
    contatore = contatore + 1
```

Stamperà:

0

1

2

## Esercizio sull'Uso delle Istruzioni Iterative

Scrivere un programma che chieda all'utente di inserire cinque numeri pari. Per ogni numero inserito il programma dovrà controllare se sia pari o meno. Se un numero inserito non è pari, il programma dovrà ripetere la richiesta di inserire un numero fino a che l'utente non avrà inserito un numero pari.

(Questo programma non è molto utile visto che i numeri non vengono memorizzati da nessuna parte ma più avanti vedremo come memorizzare un insieme di numeri).

# Esercizio sull'Uso delle Istruzioni Iterative

## Soluzione:

```
i = 0
```

```
num_numeri = 5
```

```
while i < num_numeri:
```

```
    numero = float(input("Inserire un numero pari: "))
```

```
    if numero % 2 != 0:
```

```
        print("Il numero inserito non è pari devi inserire un numero pari.")
```

```
    else:
```

```
        i = i + 1
```

## Esercizio sull'Uso delle Istruzioni Iterative

Scrivete un programma che acquisisca in input una sequenza di numeri e stampi il minimo tra quei numeri.

Sarà l'utente ad inserire il numero di valori dal quale vuole sia composta la sequenza.

# Esercizio sull'Uso delle Istruzioni Iterative

```
num_desiderati = int(input("Quanti sono gli elementi della sequenza?"))
primo_valore_acquisito = False
num_val_acquisiti = 0
minimo = None serve solo per definire la variabile, poi gli viene assegnato il primo valore
while num_val_acquisiti < num_desiderati:
    x = float(input("Inserisci un valore: "))
    if primo_valore_acquisito == False:
        minimo = x
        primo_valore_acquisito = True
    else:
        if x < minimo:
            minimo = x
    num_val_acquisiti = num_val_acquisiti + 1
if num_desiderati > 0:
    print("Il valore più piccolo è", minimo)
```