Imparare Python da zero: guida semplice per iniziare

Python è uno dei linguaggi di programmazione più utilizzati al mondo. È apprezzato perché ha una sintassi chiara, è leggibile anche da chi sta iniziando e può essere usato in tantissimi ambiti: sviluppo web, automazione, analisi dati, intelligenza artificiale, cybersecurity, scripting e molto altro.

In questa guida vediamo le basi di Python in modo semplice, senza dare nulla per scontato.

Cos'è Python?

Python è un linguaggio di programmazione, cioè uno strumento che permette di dare istruzioni al computer.

Con Python possiamo scrivere programmi che eseguono operazioni come:

- fare calcoli;
- leggere e modificare file;
- automatizzare attività ripetitive;
- creare siti web e applicazioni;
- analizzare dati;
- interagire con API e servizi online;
- sviluppare strumenti legati all'intelligenza artificiale.

Uno dei motivi per cui Python è consigliato ai principianti è che il codice è molto vicino al linguaggio naturale rispetto ad altri linguaggi più complessi.

Per esempio, per stampare un messaggio a schermo basta scrivere:

print("Ciao, mondo!")

Questa istruzione dice al computer di mostrare il testo `Ciao, mondo!`.

Installare Python

Per iniziare bisogna installare Python dal sito ufficiale:

python.org

Durante l'installazione su Windows è importante selezionare l'opzione:

Add Python to PATH

Questa opzione permette di usare Python dal terminale o dal prompt dei comandi.

Dopo l'installazione, possiamo verificare che Python sia installato correttamente aprendo il terminale e scrivendo:

python --version

oppure:

python3 --version

Se tutto è corretto, comparirà la versione installata.

Il primo programma in Python

Creiamo un file chiamato:

primo_programma.py

Dentro il file scriviamo:

print("Ciao! Sto imparando Python.")

Per eseguirlo, apriamo il terminale nella cartella del file e lanciamo:

python primo_programma.py

Il risultato sarà:

Ciao! Sto imparando Python.

Questo è il primo passo: scrivere un file, inserire istruzioni ed eseguirle.

Variabili: salvare informazioni

Le variabili servono a conservare dati che possiamo usare nel programma.

Esempio:

nome = "Gabriele"

```
eta = 30

print(nome)
print(eta)
```

In questo caso abbiamo creato due variabili:

nome
eta

La variabile `nome` contiene un testo, mentre la variabile `eta` contiene un numero.

Possiamo anche stampare un messaggio usando più elementi insieme. Il modo più semplice per iniziare è usare la virgola dentro `print()`:

```
nome = "Luca"
eta = 25

print("Ciao, mi chiamo", nome, "e ho", eta, "anni.")
```

Risultato:

Ciao, mi chiamo Luca e ho 25 anni.

Questo metodo è molto comodo per i principianti, perché Python si occupa automaticamente di separare le parti e di convertire i numeri in testo quando li stampa.

Esiste anche un altro metodo, usando il simbolo `+`:

```
nome = "Luca"

print("Ciao " + nome)
```

Risultato:

Ciao Luca

Però bisogna fare attenzione: il `+` funziona bene con testi, ma se vogliamo unire un testo e un numero dobbiamo convertire il numero in stringa.

Esempio corretto:

```
nome = "Luca"
eta = 25
```

print("Ciao, mi chiamo " + nome + " e ho " + str(eta) + " anni.")

Senza `str(eta)`, Python darebbe errore, perché non può unire direttamente testo e numeri con il simbolo `+`.

Per questo, all'inizio, usare la virgola è spesso più semplice:

print("Ciao, mi chiamo", nome, "e ho", eta, "anni.")

Le f-string: un modo più moderno

Quando si prende un po' di confidenza con Python, si può usare un metodo più moderno e leggibile: le **f-string**.

Esempio:

nome = "Luca"
eta = 25

print(f"Ciao, mi chiamo {nome} e ho {eta} anni.")

La lettera `f` prima delle virgolette indica a Python che dentro il testo possono esserci delle variabili tra parentesi graffe `{}`.

Quindi:

{nome}

viene sostituito con il valore della variabile `nome`, mentre:

{eta}

viene sostituito con il valore della variabile `eta`.

Il risultato sarà:

Ciao, mi chiamo Luca e ho 25 anni.

Le f-string sono molto usate perché rendono il codice più ordinato, soprattutto quando dobbiamo inserire molte variabili dentro una frase.

Per iniziare, però, va benissimo usare questo metodo:

```
print("Ciao, mi chiamo", nome, "e ho", eta, "anni.")
```

Poi, quando il concetto è chiaro, si può passare alle f-string:

```
print(f"Ciao, mi chiamo {nome} e ho {eta} anni.")
```

Tipi di dati principali

In Python esistono diversi tipi di dati. I più comuni sono:

```
testo = "Ciao"          # stringa
numero = 10             # intero
prezzo = 19.99          # numero decimale
attivo = True           # booleano
```

Una stringa è un testo.

```
nome = "Mario"
```

Un intero è un numero senza virgola.

```
anni = 20
```

Un numero decimale usa il punto, non la virgola.

```
prezzo = 12.50
```

Un booleano può essere vero o falso.

```
utente_attivo = True
```

Operazioni matematiche

Python può essere usato anche come una calcolatrice.

```
a = 10
b = 5

print(a + b)
```

```
print(a - b)
print(a * b)
print(a / b)
```

Risultato:

```
15
5
50
2.0
```

Gli operatori principali sono:

```
+   addizione
-   sottrazione
*   moltiplicazione
/   divisione
```

Esistono anche altri operatori utili:

```
print(10 // 3)  # divisione intera
print(10 % 3)   # resto della divisione
print(2 ** 3)   # potenza
```

Chiedere dati all’utente

Con `input()` possiamo chiedere informazioni all’utente.

```
nome = input("Come ti chiami? ")

print(f"Ciao {nome}!")
```

Quando il programma viene eseguito, chiede all’utente di inserire un nome.

Attenzione: `input()` restituisce sempre un testo. Se vogliamo usare un numero, dobbiamo convertirlo.

```
eta = input("Quanti anni hai? ")
eta = int(eta)

print(f"L'anno prossimo avrai {eta + 1} anni.")
```

Condizioni: if, else, elif

Le condizioni permettono al programma di prendere decisioni.

Esempio:

```
eta = 18

if eta >= 18:
    print("Sei maggiorenne.")
else:
    print("Sei minorenne.")
```

Python controlla se la condizione è vera. Se è vera, esegue il primo blocco. Altrimenti esegue il blocco `else`.

Possiamo usare anche `elif`, cioè “altrimenti se”.

```
voto = 7

if voto >= 9:
    print("Ottimo")
elif voto >= 6:
    print("Sufficiente")
else:
    print("Insufficiente")
```

Attenzione all’indentazione

In Python gli spazi all’inizio delle righe sono importanti.

Questo codice è corretto:

```
if True:
    print("Funziona")
```

Questo invece genera errore:

```
if True:
print("Errore")
```

Python usa l’indentazione per capire quali istruzioni appartengono a un blocco.

Liste: conservare più valori

Una lista permette di conservare più elementi in una sola variabile.

```
frutti = ["mela", "banana", "arancia"]

print(frutti)
```

Possiamo accedere a un elemento usando la posizione, chiamata indice.

```
print(frutti[0])
```

Risultato:

```
mela
```

In Python il conteggio parte da zero.

Possiamo aggiungere un elemento:

```
frutti.append("kiwi")
```

Possiamo rimuoverne uno:

```
frutti.remove("banana")
```

Cicli: ripetere istruzioni

I cicli permettono di ripetere una o più istruzioni.

Il ciclo `for` è utile quando vogliamo scorrere una lista.

```
frutti = ["mela", "banana", "arancia"]

for frutto in frutti:
    print(frutto)
```

Risultato:

```
mela
banana
arancia
```

Il ciclo `while` invece continua finché una condizione è vera.

```
contatore = 1

while contatore <= 5:
    print(contatore)
    contatore = contatore + 1
```

Risultato:

```
1
2
3
4
5
```

Funzioni: riutilizzare codice

Le funzioni servono a raggruppare istruzioni che possiamo riutilizzare.

```
def saluta():
    print("Ciao!")
```

Per usare la funzione:

```
saluta()
```

Possiamo anche passare dei valori alla funzione.

```
def saluta(nome):
    print(f"Ciao {nome}!")

saluta("Marco")
saluta("Anna")
```

Risultato:

```
Ciao Marco!
Ciao Anna!
```

Le funzioni aiutano a scrivere codice più ordinato e meno ripetitivo.

Dizionari: dati con chiave e valore

Un dizionario permette di salvare informazioni associate a una chiave.

```
utente = {
    "nome": "Luca",
    "eta": 28,
    "citta": "Milano"
}

print(utente["nome"])
print(utente["eta"])
```

Risultato:

```
Luca
28
```

I dizionari sono molto utili quando vogliamo rappresentare dati strutturati, come un utente, un prodotto o una configurazione.

Esempio pratico: piccolo programma completo

Vediamo un esempio semplice che mette insieme input, condizioni e variabili.

```
nome = input("Come ti chiami? ")
eta = int(input("Quanti anni hai? "))

if eta >= 18:
    print(f"Ciao {nome}, sei maggiorenne.")
else:
    print(f"Ciao {nome}, sei minorenne.")
```

Questo programma:

1. chiede il nome;
2. chiede l'età;
3. controlla se l'utente è maggiorenne;
4. stampa un messaggio personalizzato.

Errori comuni di chi inizia

Quando si impara Python è normale fare errori. Alcuni dei più comuni sono:

Dimenticare le virgolette

Sbagliato:

```
print(Ciao)
```

Corretto:

```
print("Ciao")
```

Usare la virgola nei numeri decimali

Sbagliato:

```
prezzo = 10,50
```

Corretto:

```
prezzo = 10.50
```

Sbagliare l’indentazione

Sbagliato:

```
if eta >= 18:
print("Maggiorenne")
```

Corretto:

```
if eta >= 18:
    print("Maggiorenne")
```

Confondere testo e numeri

```
eta = input("Età: ")
print(eta + 1)
```

Questo genera errore, perché `eta` è testo. Bisogna convertirla:

```
eta = int(input("Età: "))
print(eta + 1)
```

Come esercitarsi

Per imparare Python non basta leggere: bisogna scrivere codice.

Alcuni esercizi utili per iniziare sono:

- creare una calcolatrice semplice;
- chiedere nome ed età all'utente;
- controllare se un numero è pari o dispari;
- creare una lista della spesa;
- creare un piccolo quiz;
- leggere e scrivere un file di testo;
- automatizzare una piccola attività ripetitiva.

L'importante è partire da esercizi piccoli e aumentare gradualmente la difficoltà.

Mini esercizio: numero pari o dispari

Prova a scrivere questo programma:

```
numero = int(input("Inserisci un numero: "))

if numero % 2 == 0:
    print("Il numero è pari")
else:
    print("Il numero è dispari")
```

Il simbolo % restituisce il resto della divisione. Se un numero diviso per 2 ha resto 0, allora è pari.

Mini esercizio: lista della spesa

```
lista_spesa = []

prodotto = input("Inserisci un prodotto: ")
lista_spesa.append(prodotto)

prodotto = input("Inserisci un altro prodotto: ")
lista_spesa.append(prodotto)

print("La tua lista della spesa:")
for item in lista_spesa:
    print("- " + item)
```

Questo programma crea una lista vuota, aggiunge prodotti inseriti dall'utente e poi li stampa.

Cosa imparare dopo le basi

Dopo aver capito variabili, condizioni, cicli, liste, dizionari e funzioni, si può continuare con argomenti più avanzati:

- gestione dei file;
- moduli e librerie;
- programmazione orientata agli oggetti;
- richieste HTTP e API;
- sviluppo web con Flask o Django;
- analisi dati con Pandas;
- automazione con script Python;
- basi di intelligenza artificiale e machine learning.

Python è molto versatile: una volta comprese le fondamenta, si può scegliere un percorso in base ai propri obiettivi.

Conclusione

Python è uno dei linguaggi migliori per iniziare a programmare. È semplice da leggere, potente e usato in moltissimi settori.

Per impararlo davvero, il consiglio è partire dalle basi e fare pratica ogni giorno, anche solo per pochi minuti. Scrivere piccoli programmi, modificare gli esempi e provare a risolvere problemi concreti è il modo migliore per migliorare.

Non serve imparare tutto subito. L'importante è capire un concetto alla volta e usarlo in pratica.

Con costanza, Python può diventare uno strumento molto utile per studiare, lavorare, automatizzare attività e sviluppare nuovi progetti.