

# ☆ Course Contents ☆

---

## Introduzione alla Programmazione — Scaletta del Corso

---

### ■ Introduction

- Presentazione del corso e degli obiettivi.
  - Cosa significa “programmare” e perché è una competenza fondamentale oggi.
  - Strumenti che useremo (editor, linguaggio, esempi).
  - Breve panoramica su cosa impareremo nelle prossime lezioni.
- 

### ■ What is Programming?

- Definizione di programmazione come processo per “dare istruzioni a un computer”.
  - Differenza tra linguaggio naturale e linguaggio di programmazione.
  - Cenni storici (da Ada Lovelace al software moderno).
  - Scopo del codice: automatizzare, risolvere problemi, creare applicazioni.
- 

### ■ How do we write Code?

- Come si scrive il codice: editor di testo, IDE, salvataggio dei file.
  - Struttura base di un programma: istruzioni sequenziali.
  - Esempi di codice in un linguaggio semplice (es. Python o JavaScript).
  - Importanza della sintassi e dell’indentazione.
- 

### ■ How do we get Information from Computers?

- Input e output: come comunicare con il computer.
  - Funzioni `input()` e `print()` (o equivalenti).
  - Esempi pratici: leggere il nome dell’utente, mostrare un messaggio.
  - Concetto di interattività.
- 

### ■ What can Computers Do?

- Cosa può e cosa **non può** fare un computer.
  - Capacità di calcolo, velocità, memoria.
  - Limiti: dipende sempre dalle istruzioni ricevute.
  - Introduzione al concetto di algoritmo.
- 

### ■ What are Variables?

- Cosa sono le variabili e a cosa servono.
- Come “memorizzano” i dati in memoria.

- Tipi di variabili: numeri, testo, booleani.
  - Esempi pratici con assegnazioni.
- 

## ■ How do we Manipulate Variables?

- Operazioni aritmetiche e concatenate su variabili.
  - Conversioni di tipo (es. da testo a numero).
  - Concetto di aggiornamento del valore.
  - Esercizi con incremento e concatenazione.
- 

## ■ What are Conditional Statements?

- Istruzioni condizionali `if`, `else if`, `else`.
  - Struttura logica “se ... allora ... altrimenti”.
  - Operatori di confronto e logici.
  - Esempi: controllo età, scelta di percorsi diversi nel programma.
- 

## ■ What are Arrays?

- Definizione di array o lista: collezione di elementi.
  - Come si creano e come si accede ai valori.
  - Indici e iterazione base.
  - Esempi: elenco di nomi, numeri o voti.
- 

## ■ What are Loops?

- Cicli `for` e `while`.
  - Eseguire più volte le stesse istruzioni.
  - Differenza tra ciclo controllato da contatore e da condizione.
  - Esempi: contare, sommare, ripetere azioni.
- 

## ■ What are Errors?

- Tipi di errori: sintattici, logici, di runtime.
  - Come si leggono i messaggi di errore.
  - Strategie per prevenirli (test, ordine, chiarezza).
  - Esempi di errori comuni e loro correzione.
- 

## ■ How do we Debug Code?

- Cosa significa “debuggare”.
  - Tecniche: stampa di controllo, breakpoints, strumenti IDE.
  - Importanza della pazienza e della logica.
  - Esempio passo-passo di correzione di un bug.
-

## ■ What are Functions?

- Definizione di funzione: blocco di codice riutilizzabile.
  - Struttura: nome, parametri, corpo, valore di ritorno.
  - Esempi semplici: somma di due numeri, saluto personalizzato.
- 

## ■ How can we Import Functions?

- Librerie e moduli: codice già pronto che possiamo riutilizzare.
  - Come si importano e si usano.
  - Esempi: `math`, `datetime`, `random`.
  - Vantaggi dell'uso di codice esterno.
- 

## ■ How do we make our own Functions?

- Creazione di funzioni personalizzate.
  - Parametri e valori di ritorno.
  - Buone pratiche: nomi chiari, scopo preciso.
  - Esempi: funzione per calcolare la media o il massimo di tre numeri.
- 

## ■ What are ArrayLists and Dictionaries?

- Strutture dati più avanzate: liste dinamiche e dizionari (mappe chiave/valore).
  - Differenze e casi d'uso.
  - Esempi: elenco studenti, dizionario con nomi e voti.
- 

## ■ How can we use Data Structures?

- Applicazioni pratiche delle strutture dati.
  - Combinazione di array, liste e dizionari in programmi reali.
  - Esempio: registro studenti o rubrica telefonica.
  - Introduzione alla complessità e all'efficienza.
- 

## ■ What is Recursion?

- Concetto di funzione che richiama sé stessa.
  - Quando e perché usarla.
  - Esempi semplici: conto alla rovescia, fattoriale.
  - Differenza tra ricorsione e ciclo.
- 

## ■ What is Pseudocode?

- Cos'è il pseudocodice e a cosa serve.
- Scrivere algoritmi prima di programmarli.
- Regole e convenzioni.

- Esempio: pseudocodice per sommare numeri o trovare il massimo.
- 

## 💻 Choosing the Right Language?

- Panoramica sui linguaggi di programmazione più usati (Python, Java, C, JavaScript, PHP...).
  - Come scegliere il linguaggio giusto in base all'obiettivo.
  - Differenze tra linguaggi compilati e interpretati.
- 

## 💻 Applications of Programming

- Applicazioni reali della programmazione:
    - Web, app, giochi, intelligenza artificiale, automazione.
  - Come proseguire nello studio.
  - Invito alla curiosità e alla sperimentazione continua.
- 

# ★ COURSE CONTENTS — SCHEDE ESPLICATIVE ★

---

## 💻 Introduction

⌚ **Cosa impariamo:** Programmare significa insegnare a un computer a svolgere dei compiti. In questo corso impareremo come scrivere istruzioni, capire i dati, controllare il flusso del programma e risolvere problemi.

⌨️ **Esempio:** Come diciamo a un computer "salutami per nome"? Con un come:

```
print("Ciao, mondo!")
```

💡 **Attività:** Scrivi un messaggio personalizzato che il computer deve mostrare (es. "Ciao, sono [nome] e sto imparando a programmare!"). 🎧 (01:37) **What is Programming?**

⌚ **Spiegazione:** La **programmazione** è il modo in cui comunichiamo con il computer usando un linguaggio che capisce. Noi scriviamo *codice*, il computer lo interpreta e lo esegue passo dopo passo.

⌨️ **Esempio:** Nel linguaggio umano: "Accendi la luce." Nel linguaggio del computer (Python):

```
accendi_luce = True
```

💡 **Attività:** Descrivi un'azione quotidiana (es. preparare il tè) come una sequenza di istruzioni chiare e ordinate. ↗ È il tuo "algoritmo" in linguaggio naturale.

---

## 💻 How do we write Code?

 **Spiegazione:** Per scrivere codice serve un **editor di testo** o un **ambiente di sviluppo (IDE)**. Ogni linguaggio ha le sue regole (sintassi). Il codice va scritto in modo ordinato, leggibile e salvato con l'estensione giusta (es. `.py`, `.java`, `.js`).

 **Esempio:**

```
print("Benvenuto nel corso di programmazione!")
```

 **Attività:** Scrivi un piccolo programma che mostri due frasi su due righe diverse.

Suggerimento: usa più `print()`.

---

## How do we get Information from Computers?

 **Spiegazione:** Il computer comunica con noi tramite **input** (dati che inseriamo) e **output** (risposte che mostra).

 **Esempio:**

```
nome = input("Come ti chiami? ")  
print("Ciao", nome)
```

 **Attività:** Scrivi un programma che chieda all'utente la sua età e mostri un messaggio personalizzato.

---

## What can Computers Do?

 **Spiegazione:** Un computer può fare milioni di operazioni al secondo, ma **solo ciò che gli diciamo**. Può contare, calcolare, disegnare, controllare dispositivi, ma **non può "capire" o "decidere" da solo**.

 **Esempio:** Un computer può calcolare la somma:

```
a = 7  
b = 5  
print(a + b)
```

 **Attività:** Scrivi un programma che calcoli la somma di tre numeri scelti da te.

---

## What are Variables?

 **Spiegazione:** Le **variabili** servono per **memorizzare dati** in modo temporaneo nella memoria del computer. Ogni variabile ha un **nome** e un **valore**.

 **Esempio:**

```
nome = "Luca"
eta = 14
print("Mi chiamo", nome, "e ho", eta, "anni.")
```

 **Attività:** Crea tre variabili: `nome`, `scuola`, `materia_preferita` e stampale in una frase.

---

## 💻 How do we Manipulate Variables?

 **Spiegazione:** Le variabili si possono modificare con **operazioni matematiche** o **testuali**. Puoi sommare numeri, concatenare testi, aggiornare valori.

 **Esempio:**

```
x = 10
x = x + 5
print(x) # 15
```

Oppure:

```
saluto = "Ciao "
saluto = saluto + "Marco"
print(saluto)
```

 **Attività:** Crea due variabili numeriche, sommale e mostra il risultato in una frase del tipo “La somma è...”.

---

## 💻 What are Conditional Statements?

 **Spiegazione:** I **comandi condizionali** permettono al programma di prendere decisioni. Usiamo `if`, `else`, `elif` (oppure `switch` in altri linguaggi).

 **Esempio:**

```
eta = int(input("Quanti anni hai? "))
if eta >= 18:
    print("Sei maggiorenne.")
else:
    print("Sei minorenne.")
```

 **Attività:** Scrivi un programma che controlli se un numero inserito è positivo, negativo o zero.

---

## 💻 What are Arrays?

 **Spiegazione:** Un **array** (o lista) serve per memorizzare **più valori** nello stesso contenitore. Ogni valore ha una **posizione** (indice), che parte da 0.

#### **Esempio (Python):**

```
numeri = [10, 20, 30, 40]
print(numeri[0]) # mostra 10
```

 **Attività:** Crea una lista con i nomi dei tuoi amici e stampa il primo e l'ultimo elemento.

---

## **What are Loops?**

 **Spiegazione:** I **cicli** servono per ripetere più volte un'istruzione senza riscriverla. Ci sono due tipi principali:

- `for` → quando sappiamo quante volte ripetere
- `while` → quando ripetiamo finché una condizione è vera

#### **Esempio:**

```
for i in range(5):
    print("Ciclo numero", i)
```

 **Attività:** Scrivi un ciclo che mostri tutti i numeri da 1 a 10.

---

## **What are Errors?**

 **Spiegazione:** Gli **errori** (bug) sono problemi nel codice che impediscono l'esecuzione corretta. Possono essere:

- **Sintattici:** hai scritto male una parola chiave.
- **Logici:** il programma funziona, ma dà risultati sbagliati.
- **Di runtime:** errori durante l'esecuzione (es. divisione per 0).

#### **Esempio:**

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

✗ Mancano le virgolette finali → errore sintattico.

 **Attività:** Scrivi un piccolo programma con un errore, poi correggilo leggendo il messaggio che il computer ti dà.

---

## **How do we Debug Code?**

 **Spiegazione:** Debuggare significa trovare e correggere errori nel programma. Si può fare stampando i valori delle variabili o usando strumenti dell'IDE.

 **Esempio:**

```
x = 5
y = 0
print("x =", x)
print("y =", y)
print(x / y) # genera errore
```

 **Attività:** Prova a inserire stampe (`print()`) nel tuo codice per capire dove si ferma o cosa non funziona.

---

## What are Functions?

 **Spiegazione:** Una **funzione** è un gruppo di istruzioni che svolge un compito preciso e può essere richiamata più volte.

 **Esempio:**

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

 **Attività:** Crea una funzione chiamata `presentati()` che stampi nome e età.

---

## How can we Import Functions?

 **Spiegazione:** I linguaggi permettono di usare **moduli o librerie**: raccolte di funzioni già pronte. Così non dobbiamo riscrivere tutto da zero.

 **Esempio:**

```
import math
print(math.sqrt(25)) # radice quadrata
```

 **Attività:** Importa il modulo `random` e stampa un numero casuale tra 1 e 10.

---

## How do we make our own Functions?

 **Spiegazione:** Possiamo creare **funzioni personalizzate** con parametri e valori di ritorno.

 **Esempio:**

```
def somma(a, b):  
    return a + b  
  
risultato = somma(3, 4)  
print("La somma è", risultato)
```

 **Attività:** Crea una funzione `quadrato(n)` che restituisca il quadrato di un numero inserito dall'utente.

## ■ What are ArrayLists and Dictionaries?

 **Spiegazione:** Oltre alle liste, esistono **dizionari** (o mappe), dove ogni elemento ha una **chiave** e un **valore**. Sono ottimi per organizzare dati complessi.

 **Esempio:**

```
studente = {"nome": "Anna", "eta": 15, "classe": "3B"}  
print(studente["nome"])
```

 **Attività:** Crea un dizionario con i dati di un libro (`titolo`, `autore`, `anno`) e stampane uno dei campi.

## ■ How can we use Data Structures?

 **Spiegazione:** Le **strutture dati** (liste, dizionari, insiemi, tuple) servono per **organizzare informazioni** in modo efficiente. Permettono di creare programmi più complessi e realistici.

 **Esempio:**

```
studenti = [  
    {"nome": "Luca", "voto": 8},  
    {"nome": "Sara", "voto": 9}  
]  
for s in studenti:  
    print(s["nome"], "ha preso", s["voto"])
```

 **Attività:** Crea una lista di almeno tre dizionari, ognuno con nome e voto di uno studente.

## ■ What is Recursion?

 **Spiegazione:** La **ricorsione** è quando una funzione richiama **sé stessa** per risolvere un problema. Utile per problemi ripetitivi o strutture a livelli (come le cartelle di un computer).

 **Esempio:**

```
def conto(n):
    if n > 0:
        print(n)
        conto(n - 1)

conto(5)
```

 **Attività:** Scrivi una funzione ricorsiva che conti da 1 a un numero scelto dall'utente.

---

## What is Pseudocode?

 **Spiegazione:** Il **pseudocodice** è un modo per scrivere gli algoritmi **in linguaggio naturale**, senza preoccuparsi della sintassi. Serve per progettare il programma prima di scriverlo davvero.

### Esempio:

```
INIZIO
    chiedi numero A
    chiedi numero B
    somma = A + B
    mostra somma
FINE
```

 **Attività:** Scrivi in pseudocodice le istruzioni per preparare un panino.

---

## Choosing the Right Language?

 **Spiegazione:** Esistono molti linguaggi di programmazione. La scelta dipende dallo scopo:

- **Python:** facile e versatile
- **Java:** applicazioni grandi e stabili
- **JavaScript:** pagine web interattive
- **C:** velocità e controllo
- **PHP:** siti dinamici

### Esempio:

Python è ottimo per iniziare perché ha una sintassi semplice e leggibile.

 **Attività:** Scrivi tre esempi di progetti e indica quale linguaggio useresti per ciascuno (es. videogioco, sito web, app mobile).

---

## Applications of Programming

 **Spiegazione:** La programmazione è ovunque:

- nei siti web
- nei videogiochi
- nelle app per smartphone
- nei sistemi di controllo, nei robot, nelle auto

 **Esempio:** Un'app come WhatsApp usa codice per inviare messaggi, salvare dati, mostrare notifiche.

 **Attività:** Fai una lista di almeno 5 cose che usi ogni giorno e che funzionano grazie alla programmazione.

---

 **Conclusione:** Hai ora una panoramica completa e strutturata dei concetti base della programmazione. Ogni scheda può essere usata come lezione breve, esercitazione o supporto per spiegazioni in classe.