

# Corso ITS: *ARTIFICIAL INTELLIGENCE SPECIALIST*

## Modulo: Programmazione Procedurale in Python

Docente: *Andrea Ribuoli*

---

**Lunedì 27 Gennaio 2025**

**08:30 - 14:30**

---

- operatore `in` (e la sua negazione `not in`)
- metodi `startswith` e `endswith`
- metodo `count`
- metodo `find`
- metodi `isalpha`, `isdigit` and `isalnum`
- metodi `islower` and `isupper`
- metodo `isspace`
- considerazioni:
  - - sono operatore e metodi utili nella **input validation**
  - - rischio di **eccezioni** (*run-time exception*)
- terminare un programma con la funzione `exit`

### il ciclo `while`

- è il primo degli **enunciati di ciclo**
- esegue ripetutamente istruzioni fino al raggiungimento di un obiettivo

```
while condizione :  
    enunciato_1  
    enunciato_2
```

- fintanto che la condizione è vera gli enunciati presenti all'interno vengono eseguiti
- gli enunciati interni al `while` ne costituiscono il **corpo** (*body*)
- l'enunciato **`while`** è un esempio di **ciclo** (*loop*)
- cicli controllati da **contatore** (*definito*) o da **evento** (*indefinito*)

- considerazioni:
  - - *Abbiamo finito? .vs. Fino a quando?*
  - - cicli infiniti
  - - errori per scarto di uno
- **named argument** (*argomento con nome*): `print con end=""`
- cicli per acquisire valori in sequenza
- valore **sentinella** per segnalare la fine di una sequenza
- esercizio: sviluppo del programma `media.py`
- utilizzo di `break`
- redirectione **I/O** (*input/output*)

## Il ciclo **for**

```
nomeRegione = "Marche"
for letter in nomeRegione :
    print(letter)
```

```
nomeRegione = "Marche"
i = 0
while i < len(nomeRegione) :
    letter = nomeRegione[i]
    print(letter)
    i = i + 1
```

- il ciclo `for` può essere usato su **qualunque contenitore**
- funzione `range()` (crea una sequenza di numeri utilizzabile come contenitori)
- **cicli annidati** (*nested loop*)
- es: trovare la prima o l'ultima corrispondenza di una cifra in una stringa

- una **funzione** è una sequenza di istruzioni datata di un **nome**
  - una funzione viene **invocata** (*called*)
  - una funzione **restituisce** (*returns*)
  - una funzione viene chiamata con *dati di ingresso* detti **argomenti**
  - una funzione può **restituire** un valore (*returned value*)
- 
- in fase di definizione di una **funzione utente** è necessario:
    - - scegliere un **nome**
    - - definire una variabile per ciascuno dei suoi *argomenti* (detti **variabili parametro**)

```
In [6]: def volumeCubo(lunghezzaLato) :  
        volume = lunghezzaLato ** 3  
        return volume
```

- gli enunciati nella definizione di funzione non vengono eseguiti
  - gli enunciati non interni a definizioni di funzione vengono eseguiti nell'ordine
  - è importante che ciascuna funzione sia **definita prima di essere invocata**
- 
- ambito di visibilità di una variabile (**variable scope**)
  - **variabile locale** (*local variable*)
  - **variabile globale** (*global variable*)
  - suggerimenti:
    - - evitare l'uso di variabili globali

## funzioni ricorsive (cenni)