🔷 Modulo 4: Cicli e Flussi

13. While loops ∞

Obiettivi di apprendimento:

- Sintassi e logica dei cicli while
- Condizioni di terminazione
- Cicli infiniti e come evitarli
- Break e continue

Contenuto teorico:

```
# While loop base
print("=== WHILE LOOP BASE ===")
contatore = 1
while contatore <= 5:
    print(f"Iterazione {contatore}")
    contatore += 1

print("Fine del ciclo!")

# Countdown
print(f"\n=== COUNTDOWN ===")
numero = 10
while numero > 0:
    print(f"Countdown: {numero}")
    numero -= 1
print("🚀 Decollo!")

# Input validation con while
print(f"\n=== VALIDAZIONE INPUT ===")
while True:
    eta = input("Inserisci la tua età (0-120): ")
    if eta.isdigit():
        eta_num = int(eta)
        if 0 <= eta_num <= 120:
            print(f"Età valida: {eta_num} anni")
            break
        else:
            print("❌ L'età deve essere tra 0 e 120!")
    else:
        print("❌ Inserisci solo numeri!")

# Break e continue
print(f"\n=== BREAK E CONTINUE ===")
numero = 0
while numero < 10:
    numero += 1

    if numero == 3:
```

```
        print(f"Salto il numero {numero}")
        continue

    if numero == 8:
        print(f"Mi fermo al numero {numero}")
        break

    print(f"Numero: {numero}")
```

Menu interattivo:

```
def menu_interattivo():
    print("🍕 SISTEMA PIZZERIA 🍕")

    menu = {
        '1': ('Margherita', 8.50),
        '2': ('Marinara', 7.00),
        '3': ('Quattro Stagioni', 12.00),
        '4': ('Capricciosa', 11.50),
        '5': ('Diavola', 10.00)
    }

    ordine = []
    totale = 0

    while True:
        print("\n" + "="*40)
        print("MENU PIZZE:")
        for codice, (nome, prezzo) in menu.items():
            print(f"{codice}. {nome:<20} €{prezzo:.2f}")
        print("0. Finalizza ordine")
        print("9. Cancella ordine")
        print("="*40)

        scelta = input("Seleziona un'opzione: ").strip()

        if scelta == '0':
            if ordine:
                print(f"\n📝 RIEPILOGO ORDINE:")
                for item in ordine:
                    print(f"- {item['nome']} x{item['quantita']} = €
{item['subtotale']:.2f}")
                print(f"TOTALE: €{totale:.2f}")
                break
            else:
                print("❌ Nessun elemento nell'ordine!")

        elif scelta == '9':
            ordine.clear()
            totale = 0
            print("🗑️ Ordine cancellato!")
```

```
        elif scelta in menu:
            nome_pizza, prezzo_pizza = menu[scelta]

            while True:
                quantita_str = input(f"Quantità di {nome_pizza}: ")
                if quantita_str.isdigit() and int(quantita_str) > 0:
                    quantita = int(quantita_str)
                    break
                print("❌ Inserisci un numero valido!")

            subtotale = prezzo_pizza * quantita
            ordine.append({
                'nome': nome_pizza,
                'quantita': quantita,
                'prezzo_unitario': prezzo_pizza,
                'subtotale': subtotale
            })
            totale += subtotale
            print(f"✅ Aggiunto: {quantita}x {nome_pizza}")

        else:
            print("❌ Scelta non valida!")

menu_interattivo()
```

Gioco indovina il numero:

```
import random

def indovina_numero():
    print("🎯 GIOCO: INDOVINA IL NUMERO 🎯")

    numero_segreto = random.randint(1, 100)
    tentativi = 0
    max_tentativi = 7

    print(f"Ho pensato un numero tra 1 e 100!")
    print(f"Hai {max_tentativi} tentativi per indovinarlo.")

    while tentativi < max_tentativi:
        try:
            guess = int(input(f"\nTentativo {tentativi +
1}/{max_tentativi}: "))
            tentativi += 1

            if guess == numero_segreto:
                print(f"🎉 BRAVO! Hai indovinato in {tentativi}
tentativi!")
                break
            elif guess < numero_segreto:
                print("📈 Troppo basso!")
            else:
```

```
                print("📉 Troppo alto!")

            # Hint aggiuntivi
            if tentativi == max_tentativi // 2:
                if numero_segreto % 2 == 0:
                    print("💡 Hint: Il numero è pari")
                else:
                    print("💡 Hint: Il numero è dispari")

        except ValueError:
            print("❌ Inserisci un numero valido!")
            tentativi -= 1  # Non conta come tentativo

    if tentativi == max_tentativi and guess != numero_segreto:
        print(f"😔 Hai esaurito i tentativi! Il numero era
{numero_segreto}")

    # Chiedi se vuole giocare ancora
    while True:
        again = input("\nVuoi giocare ancora? (s/n): ").lower()
        if again in ['s', 'si', 'sì']:
            indovina_numero()
            break
        elif again in ['n', 'no']:
            print("Grazie per aver giocato! 👋")
            break
        else:
            print("Rispondi s/n")

indovina_numero()
```

Simulatore ATM:

```
def simulatore_atm():
    print("🏧 SIMULATORE ATM 🏧")

    saldo = 1000.00
    pin_corretto = "1234"
    tentativi_pin = 0
    max_tentativi = 3

    # Autenticazione
    while tentativi_pin < max_tentativi:
        pin = input("Inserisci il PIN: ")

        if pin == pin_corretto:
            print("✅ PIN corretto!")
            break
        else:
            tentativi_pin += 1
            rimanenti = max_tentativi - tentativi_pin
            if rimanenti > 0:
```

```
                print(f"❌ PIN errato! Tentativi rimanenti: {rimanenti}")
            else:
                print("🚫 Carta bloccata! Troppi tentativi errati.")
                return

    # Menu ATM
    while True:
        print(f"\n{'='*30}")
        print("MENU ATM:")
        print("1. Visualizza saldo")
        print("2. Preleva denaro")
        print("3. Deposita denaro")
        print("4. Esci")
        print("="*30)

        scelta = input("Seleziona un'opzione: ")

        if scelta == '1':
            print(f"💰 Saldo attuale: €{saldo:.2f}")

        elif scelta == '2':
            try:
                importo = float(input("Importo da prelevare: €"))
                if importo <= 0:
                    print("❌ Importo non valido!")
                elif importo > saldo:
                    print("❌ Saldo insufficiente!")
                elif importo > 500:
                    print("❌ Limite giornaliero: €500")
                else:
                    saldo -= importo
                    print(f"✅ Prelevati €{importo:.2f}")
                    print(f"💰 Saldo rimanente: €{saldo:.2f}")
            except ValueError:
                print("❌ Importo non valido!")

        elif scelta == '3':
            try:
                importo = float(input("Importo da depositare: €"))
                if importo <= 0:
                    print("❌ Importo non valido!")
                else:
                    saldo += importo
                    print(f"✅ Depositati €{importo:.2f}")
                    print(f"💰 Nuovo saldo: €{saldo:.2f}")
            except ValueError:
                print("❌ Importo non valido!")

        elif scelta == '4':
            print("👋 Grazie per aver utilizzato l'ATM!")
            break

        else:
            print("❌ Scelta non valida!")
```

```
simulatore_atm()
```

Esercizi pratici:

1. Sistema di login con limite tentativi
2. Calcolatrice interattiva
3. Gioco della morra cinese

---

14. **For loops** 🔁

Obiettivi di apprendimento:

- Sintassi dei cicli for
- Iterazione su sequenze (range, liste, stringhe)
- Enumerate e zip
- List comprehensions base

Contenuto teorico:

```
# For loop con range
print("=== FOR LOOP CON RANGE ===")
for i in range(5):
    print(f"Iterazione {i}")

print("\nDa 1 a 5:")
for i in range(1, 6):
    print(f"Numero {i}")

print("\nNumeri pari da 0 a 10:")
for i in range(0, 11, 2):
    print(f"Pari: {i}")

# Iterazione su stringhe
print(f"\n=== ITERAZIONE SU STRINGHE ===")
parola = "Python"
for carattere in parola:
    print(f"Carattere: {carattere}")

# Iterazione su liste
print(f"\n=== ITERAZIONE SU LISTE ===")
frutti = ["mela", "banana", "arancia", "kiwi"]
for frutto in frutti:
    print(f"Frutto: {frutto}")

# Enumerate - ottenere indice e valore
print(f"\n=== ENUMERATE ===")
# **enumerate()** restituisce coppie di (indice, valore)
for indice, frutto in enumerate(frutti):
    print(f"Posizione {indice}: {frutto}")
```

```
# Enumerate con start personalizzato
for numero, frutto in enumerate(frutti, 1):
    print(f"#{numero}: {frutto}")

# Zip - iterare su multiple sequenze
print(f"\n=== ZIP ===")
# **zip()** aggrega gli elementi di più iterabili in tuple
nomi = ["Alice", "Bob", "Charlie"]
età = [25, 30, 35]
città = ["Roma", "Milano", "Napoli"]

for nome, eta, citta in zip(nomi, età, città):
    print(f"{nome} ha {eta} anni e vive a {citta}")

# Zip con lunghezze diverse (si ferma al più corto)
colori = ["Rosso", "Verde"]
dimensioni = [10, 20, 30, 40]
print("\nZip con lunghezze diverse:")
for colore, dimensione in zip(colori, dimensioni):
    print(f"Colore: {colore}, Dimensione: {dimensione}")
```

Analizzatore di dati:

```
def analizza_vendite():
    print("📊 ANALIZZATORE VENDITE 📊")

    # Dati di vendita mensili
    mesi = ["Gen", "Feb", "Mar", "Apr", "Mag", "Giu",
            "Lug", "Ago", "Set", "Ott", "Nov", "Dic"]
    vendite = [15000, 18000, 22000, 19000, 25000, 28000,
               32000, 30000, 26000, 23000, 20000, 35000]

    # Statistiche base
    totale_vendite = sum(vendite)
    media_mensile = totale_vendite / len(vendite)

    print(f"Totale vendite annuali: €{totale_vendite:,.2f}")
    print(f"Media mensile: €{media_mensile:,.2f}")

    # Trova migliore e peggiore mese
    vendita_max = max(vendite)
    vendita_min = min(vendite)

    # Metodo alternativo e più efficiente per trovare l'indice
    indice_migliore = vendite.index(vendita_max)
    indice_peggiore = vendite.index(vendita_min)

    mese_migliore = mesi[indice_migliore]
    mese_peggiore = mesi[indice_peggiore]

    print(f"Miglior mese: {mese_migliore} (€{vendita_max:,.2f})")
```

```
    print(f"Peggior mese: {mese_peggiore} (€{vendita_min:,.2f})")

    # Report dettagliato
    print(f"\n{'MESE':<5} {'VENDITE':<12} {'vs MEDIA':<10} {'TREND':<6}")
    print("-" * 40)

    for mese, vendita in zip(mesi, vendite):
        differenza = vendita - media_mensile
        percentuale = (differenza / media_mensile) * 100
        trend = "📈" if vendita > media_mensile else "📉"

        print(f"{mese:<5} €{vendita:<11,.0f} {percentuale:>+6.1f}%
{trend}")

    # Trimestri
    print(f"\n=== ANALISI TRIMESTRALE ===")
    for trimestre in range(4):
        inizio = trimestre * 3
        fine = inizio + 3
        vendite_trimestre = vendite[inizio:fine]
        mesi_trimestre = mesi[inizio:fine]
        totale_trimestre = sum(vendite_trimestre)

        print(f"Q{trimestre + 1} ({', '.join(mesi_trimestre)}): €
{totale_trimestre:,.2f}")

analizza_vendite()
```

Generatore di pattern:

```
def genera_pattern():
    print("🎨 GENERATORE DI PATTERN 🎨")

    # Pattern 1: Triangolo di stelle
    print("1. Triangolo di stelle:")
    for i in range(1, 6):
        print("*" * i)

    print("\n2. Triangolo centrato:")
    altezza = 5
    for i in range(1, altezza + 1):
        spazi = " " * (altezza - i)
        stelle = "*" * (2 * i - 1)
        print(spazi + stelle)

    print("\n3. Tabellina pitagorica:")
    print("    ", end="")
    for i in range(1, 11):
        print(f"{i:4}", end="")
    print()
    print("    " + "-" * 40)
```

```
    for i in range(1, 11):
        print(f"{i:2} |", end="")
        for j in range(1, 11):
            prodotto = i * j
            print(f"{prodotto:4}", end="")
        print()

    print("\n4. Pattern numerico:")
    for i in range(1, 6):
        for j in range(1, i + 1):
            print(j, end="")
        print()

    print("\n5. Pattern alfabetico:")
    # **chr()** converte un codice numerico (ASCII/Unicode) nel carattere
corrispondente
    # **ord()** converte un carattere nel suo codice numerico
    for i in range(5):
        for j in range(i + 1):
            lettera = chr(ord('A') + j)
            print(lettera, end="")
        print()

genera_pattern()
```

Sistema di voti:

```
def sistema_voti():
    print("🎓 SISTEMA GESTIONE VOTI 🎓")

    studenti = ["Alice", "Bob", "Charlie", "Diana", "Eve"]
    materie = ["Matematica", "Italiano", "Inglese", "Storia"]

    # Genera voti casuali per dimostrazione
    import random
    voti = {}

    for studente in studenti:
        voti[studente] = {}
        for materia in materie:
            # Genera 3-5 voti per materia
            num_voti = random.randint(3, 5)
            # List comprehension per generare i voti
            voti_materia = [random.randint(4, 10) for _ in
range(num_voti)]
            voti[studente][materia] = voti_materia

    # Calcola medie
    print("📋 REPORT VOTI E MEDIE:")
    print("="*80)

    # Ciclo sui dizionari annidati per stampare i report
```

```
    for studente in studenti:
        print(f"\n👤 STUDENTE: {studente.upper()}")
        print("-"*50)

        somma_medie = 0
        for materia in materie:
            voti_materia = voti[studente][materia]
            media = sum(voti_materia) / len(voti_materia)
            somma_medie += media

            # Formatta i voti
            voti_str = ", ".join(map(str, voti_materia))

            print(f"{materia:<12}: [{voti_str:<15}] Media: {media:.2f}")

        media_generale = somma_medie / len(materie)
        print(f"{'MEDIA GENERALE':<12}: {media_generale:.2f}")

        # Valutazione
        if media_generale >= 8:
            valutazione = "ECCELLENTE 🌟"
        elif media_generale >= 7:
            valutazione = "BUONO 👍"
        elif media_generale >= 6:
            valutazione = "SUFFICIENTE ✅"
        else:
            valutazione = "INSUFFICIENTE ❌"

        print(f"{'VALUTAZIONE':<12}: {valutazione}")

    # Statistiche di classe
    print(f"\n{'='*80}")
    print("📊 STATISTICHE DI CLASSE:")
    print("="*80)

    # Ciclo per materia per calcolare le statistiche di classe
    for materia in materie:
        tutti_voti = []
        # Ciclo annidato per raccogliere tutti i voti per una specifica
materia
        for studente in studenti:
            tutti_voti.extend(voti[studente][materia])

        media_classe = sum(tutti_voti) / len(tutti_voti)
        voto_max = max(tutti_voti)
        voto_min = min(tutti_voti)

        print(f"{materia:<12}: Media {media_classe:.2f} | Max {voto_max} |
Min {voto_min}")

sistema_voti()
```

List comprehensions introduzione:

```python
def intro_list_comprehensions():
    print("📝 INTRODUZIONE LIST COMPREHENSIONS 📝")

    # Sintassi: [espressione for elemento in iterabile]
    numeri = [1, 2, 3, 4, 5]

    # Tradizionale
    quadrati_trad = []
    for num in numeri:
        quadrati_trad.append(num ** 2)

    # List comprehension
    quadrati_comp = [num ** 2 for num in numeri]

    print(f"Numeri originali: {numeri}")
    print(f"Quadrati (tradizionale): {quadrati_trad}")
    print(f"Quadrati (comprehension): {quadrati_comp}")

    # List comprehension con condizione IF
    # Sintassi: [espressione for elemento in iterabile if condizione]
    pari_comp = [num for num in range(1, 11) if num % 2 == 0]
    print(f"Numeri pari 1-10: {pari_comp}")

    # Trasformazioni stringhe
    parole = ["python", "java", "javascript", "c++"]
    parole_maiusc = [parola.upper() for parola in parole]
    # List comprehension con condizione su stringhe
    parole_lunghe = [parola for parola in parole if len(parola) > 4]

    print(f"Parole maiuscole: {parole_maiusc}")
    print(f"Parole lunghe: {parole_lunghe}")

intro_list_comprehensions()
```

Esercizi pratici:

1. Calcolatore statistiche classe (Vedi `sistema_voti()`)
2. Generatore di sequenze matematiche (Fibonacci, numeri primi)
3. Analizzatore di testo (conta parole, lettere, etc.)

---

15. **Nested loops** 👓

Obiettivi di apprendimento:

- Cicli annidati (for dentro for)
- Pattern bidimensionali
- Matrici e tabelle
- Ottimizzazione e complessità

Contenuto teorico:

```python
# Nested loops base
print("=== NESTED LOOPS BASE ===")
# Il ciclo interno viene completato ad ogni singola iterazione del ciclo
esterno.
for i in range(3):
    print(f"Ciclo esterno: {i}")
    for j in range(3):
        print(f"  Ciclo interno: {j}")
    print() # Riga vuota per separare le iterazioni esterne

# Esempio: Coordinata (riga, colonna)
print("\n=== STAMPA DI COORDINATE (MATRICE 3x3) ===")
righe = 3
colonne = 3
for r in range(righe):
    for c in range(colonne):
        print(f"({r}, {c})", end=" ") # end=" " evita il newline
    print() # Newline dopo ogni riga (ciclo esterno)

# Esempio: Tabelline (Vedi anche Generatore di Pattern)
print("\n=== TABELLINE (Prodotto) ===")
for i in range(1, 6): # Primo fattore
    for j in range(1, 6): # Secondo fattore
        prodotto = i * j
        print(f"{prodotto:2}", end=" ")
    print() # Nuova riga dopo aver completato una tabellina

# Ottimizzazione: Usare break e continue
print("\n=== NESTED LOOPS CON BREAK ===")
# Uscire dal ciclo interno quando si trova un 1 in posizione (1, y)
matrice = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
trovato = False

for r in range(len(matrice)):
    for c in range(len(matrice[r])):
        if matrice[r][c] == 1:
            print(f"Trovato 1 in ({r}, {c})!")
            trovato = True
            break # Esce solo dal ciclo FOR INTERNO (sulla colonna)

    if trovato:
        break # Esce dal ciclo FOR ESTERNO (sulla riga)

print(f"Stato finale 'trovato': {trovato}")
```

List Comprehensions annidate (Introduzione):
Le List Comprehensions possono anche essere **annidate** per creare liste di liste (matrici) in modo compatto.

```
print("\n=== LIST COMPREHENSIONS ANNIDATE ===")

# Matrice 3x3 di zeri
matrice_zeri = [[0 for col in range(3)] for row in range(3)]
print(f"Matrice di zeri 3x3:\n{matrice_zeri}")

# Generazione di una matrice I x J dove ogni elemento è (i, j)
I, J = 2, 4
coppie_ij = [[(i, j) for j in range(J)] for i in range(I)]
print(f"\nCoppie (i, j) 2x4:\n{coppie_ij}")

# Flattening (appiattimento) di una lista di liste
lista_di_liste = [[1, 2, 3], [4, 5], [6, 7, 8]]

# Tradizionale
lista_appiattita_trad = []
for sublist in lista_di_liste:
    for item in sublist:
        lista_appiattita_trad.append(item)

# List comprehension annidata (nota l'ordine: ciclo esterno prima)
lista_appiattita_comp = [item for sublist in lista_di_liste for item in
sublist]

print(f"\nLista di liste: {lista_di_liste}")
print(f"Appiattita (comprehension): {lista_appiattita_comp}")
```

Complessità (Cenni)

Un ciclo `for` ha una complessità temporale di solito **O(n)**, dove n è il numero di elementi nell'iterabile (lineare).

Un ciclo **annidato** (come un ciclo `for` dentro un altro `for` che itera per n volte) ha una complessità temporale di **O(n^2)** (quadratica). Ad esempio, iterare su una matrice $n \times n$ o stampare una tabellina $n \times n$ richiede $n \times n$ operazioni. È importante esserne consapevoli in caso di grandi quantità di dati.