SVILUPPO APPLICAZIONI E ANALISI DATI CON PYTHON

Python Avanzato

Dizionari, Stringhe, File e Funzioni

Docente: Marino Speciale

Indice della Lezione

Dizionari: Introduzione

Un dizionario è una struttura dati **mutabile** che memorizza coppie **chiave-valore**.

```
# Creazione di dizionari
persona = {"nome": "Marco", "età": 25, "città": "Milano"}
vuoto = {}

# Accesso ai valori
print(persona["nome"])  # Marco
print(persona.get("età"))  # 25

# Le chiavi devono essere immutabili
d = {1: "uno", "due": 2, (1,2): "tupla"}
```

Operazioni su Dizionari

```
d = {"nome": "Alice", "età": 30}

# Aggiungere/Modificare
d["città"] = "Roma"       # Aggiunge nuova coppia
d["età"] = 31             # Modifica il valore

# Rimuovere
del d["città"]            # Rimuove la chiave
valore = d.pop("età")     # Rimuove e ritorna il valore

# Iterare
for chiave, valore in d.items():
    print(f"{chiave}: {valore}")

for chiave in d.keys():
    print(chiave)
```

Metodi Principali dei Dizionari

```
d = {"Python": 8, "Java": 7, "C++": 6}

# keys(), values(), items()
print(d.keys())           # dict_keys(['Python', 'Java', 'C++'])
print(d.values())         # dict_values([8, 7, 6])

# get() - accesso sicuro
valore = d.get("Go", "Non trovato")  # Ritorna default

# update() - merge di dizionari
d.update({"JavaScript": 8, "Rust": 7})

# setdefault() - aggiunge se assente
d.setdefault("Kotlin", 5)
```

Stringhe: Creazione e Accesso

```
# Stringhe sono immutabili
s = "Python"
s2 = 'Single quotes'
s3 = """Multi
line"""

# Accesso carattere (come liste)
print(s[0])              # P
print(s[-1])             # n
print(s[1:4])            # yth

# Lunghezza
print(len(s))            # 6

# Concatenazione
frase = "Ciao" + " " + "Mondo"
```

Metodi Stringhe: Trasformazione

```
testo = "Python è Fantastico"

# Trasformazione maiuscole/minuscole
print(testo.upper())          # PYTHON È FANTASTICO
print(testo.lower())          # python è fantastico
print(testo.capitalize())     # Python è fantastico
print(testo.title())          # Python È Fantastico

# Ricerca e sostituzione
print(testo.find("è"))        # 7 (indice)
print(testo.count("a"))       # 2 (occorrenze)
print(testo.replace("Python", "Java"))  # Java è Fantastico
```

Metodi Stringhe: Parsing e Formattazione

```
# split() - divide in liste
frase = "Python,Java,C++,Go"
linguaggi = frase.split(",")  # ['Python', 'Java', 'C++', 'Go']

# join() - unisce elementi
lista = ["Mela", "Banana", "Ciliegia"]
risultato = ", ".join(lista)   # "Mela, Banana, Ciliegia"

# strip() - rimuove spazi
spazio = "  Testo  "
print(spazio.strip())  # "Testo"

# Formattazione
nome = "Marco"
età = 25
messaggio = f"Mi chiamo {nome} e ho {età} anni"
```

Funzioni: Definizione e Uso

```
# Definizione semplice
def saluta():
    print("Ciao!")

saluta()  # Esecuzione

# Con parametri
def somma(a, b):
    return a + b

risultato = somma(5, 3)  # risultato = 8

# Parametri di default
def quadrato(x=0):
    return x ** 2

print(quadrato())    # 0
print(quadrato(5))   # 25
```

Funzioni: Argomenti *args e **kwargs

```
# *args - numero variabile di argomenti
def somma_multipla(*numeri):
    return sum(numeri)

print(somma_multipla(1, 2, 3, 4))  # 10

# **kwargs - argomenti nominati
def info_persona(**dati):
    for chiave, valore in dati.items():
        print(f"{chiave}: {valore}")

info_persona(nome="Alice", età=30, città="Roma")

# Combinare tutti
def funzione_completa(a, b, *args, **kwargs):
    print(f"a={a}, b={b}")
    print(f"Extra args: {args}")
    print(f"Extra kwargs: {kwargs}")
```

Scope: Variabili Locali e Globali

```
x = 10  # Variabile globale

def funzione1():
    x = 20  # Variabile locale (non modifica la globale)
    print(x)  # 20

funzione1()
print(x)  # 10 (rimane inalterata)

# Usare global per modificare la globale
def funzione2():
    global x
    x = 30

funzione2()
print(x)  # 30 (ora è modificata)

# BEST PRACTICE: passare valori come parametri
def funzione3(valore):
    return valore * 2
```

Gestione File: Lettura

```
# Apertura e lettura semplice
file = open("dati.txt", "r")
contenuto = file.read()  # Legge tutto il file
file.close()

# Lettura linea per linea
with open("dati.txt", "r") as file:
    for linea in file:
        print(linea.strip())  # strip() rimuove newline

# Lettura in lista di linee
with open("dati.txt", "r") as file:
    linee = file.readlines()  # ['linea1\n', 'linea2\n', ...]

# BEST PRACTICE: usare 'with' per chiusura automatica
```

Gestione File: Scrittura

```
# Modalità "w" - sovrascrive il file
with open("output.txt", "w") as file:
    file.write("Prima linea\n")
    file.write("Seconda linea\n")

# Modalità "a" - aggiunge al fine del file
with open("output.txt", "a") as file:
    file.write("Linea aggiunta\n")

# Scrivere liste/dizionari (JSON)
import json

dati = {"nome": "Marco", "età": 25}
with open("dati.json", "w") as file:
    json.dump(dati, file, indent=4)

# Leggere JSON
with open("dati.json", "r") as file:
    dati_letti = json.load(file)
```

Gestione File: Errori e Eccezioni

```
# Try-except per gestire errori
try:
    with open("file_inesistente.txt", "r") as file:
        contenuto = file.read()
except FileNotFoundError:
    print("File non trovato!")
except IOError as e:
    print(f"Errore I/O: {e}")
finally:
    print("Esecuzione conclusa")

# Controllare se un file esiste
import os
if os.path.exists("dati.txt"):
    # Leggere il file
    pass
else:
    print("File non trovato")
```

Esempio Pratico: Gestore Contatti (Part 1)

```
import json

def salva_contatti(lista_contatti, file_path):
    """Salva dizionari di contatti in file JSON"""
    with open(file_path, "w") as file:
        json.dump(lista_contatti, file, indent=4)
    print(f"Contatti salvati in {file_path}")

def carica_contatti(file_path):
    """Carica contatti da file JSON"""
    try:
        with open(file_path, "r") as file:
            return json.load(file)
    except FileNotFoundError:
        return []
```

Esempio Pratico: Gestore Contatti (Part 2)

```
def aggiungi_contatto(contatti, nome, email, telefono):
    """Aggiunge un nuovo contatto"""
    nuovo = {"nome": nome, "email": email, "telefono": telefono}
    contatti.append(nuovo)

def cerca_contatto(contatti, nome):
    """Cerca un contatto per nome (case-insensitive)"""
    for contatto in contatti:
        if contatto["nome"].lower() == nome.lower():
            return contatto
    return None

def stampa_contatti(contatti):
    """Stampa tutti i contatti formattati"""
    if not contatti:
        print("Nessun contatto disponibile")
        return
    for contatto in contatti:
        print(f"Nome: {contatto['nome']}")
        print(f"Email: {contatto['email']}")
        print(f"Tel: {contatto['telefono']}\n")
```

Esempio Pratico: Menu Principale

```
def main():
    FILE_PATH = "contatti.json"
    contatti = carica_contatti(FILE_PATH)

    while True:
        print("\n--- GESTIONE CONTATTI ---")
        print("1. Visualizza tutti")
        print("2. Aggiungi contatto")
        print("3. Cerca contatto")
        print("4. Salva ed esci")

        scelta = input("Scelta: ")

        if scelta == "1":
            stampa_contatti(contatti)
        elif scelta == "2":
            nome = input("Nome: ")
            email = input("Email: ")
            tel = input("Telefono: ")
            aggiungi_contatto(contatti, nome, email, tel)
        elif scelta == "3":
            nome = input("Nome da cercare: ")
            result = cerca_contatto(contatti, nome)
            if result:
                print(f"Trovato: {result}")
            else:
                print("Non trovato")
        elif scelta == "4":
            salva_contatti(contatti, FILE_PATH)
            break

if __name__ == "__main__":
    main()
```

Decoratori: Introduzione (Avanzato)

```
# Un decoratore è una funzione che modifica il comportamento di un'altra

def decoratore(funzione):
    def wrapper():
        print("Prima di eseguire")
        funzione()
        print("Dopo l'esecuzione")
    return wrapper

@decoratore
def saluta():
    print("Ciao!")

saluta()
# Output:
# Prima di eseguire
# Ciao!
# Dopo l'esecuzione
```

Lambda e Funzioni di Ordine Superiore

```
# Lambda - funzioni anonime in una linea
quadrato = lambda x: x ** 2
print(quadrato(5))  # 25

# map() - applica funzione a ogni elemento
numeri = [1, 2, 3, 4, 5]
quadrati = list(map(lambda x: x ** 2, numeri))

# filter() - filtra elementi
pari = list(filter(lambda x: x % 2 == 0, numeri))

# sorted() con key personalizzato
persone = [{"nome": "Alice", "età": 30},
           {"nome": "Bob", "età": 25}]
ordinati = sorted(persone, key=lambda x: x["età"])
```

Strumenti per il Debugging

```
# pdb - Python Debugger (debugger integrato)
import pdb

x = 10
pdb.set_trace()  # Pausa l'esecuzione qui
y = x * 2

# print debugging
print(f"Valore di x: {x}")

# Logging (meglio del print)
import logging
logging.basicConfig(level=logging.DEBUG)
logging.debug("Messaggio debug")
logging.error("Errore!")
```

Best Practices Generali

- Usa per gestire i file automaticamente
- Documenta le funzioni con docstring
- Evita variabili globali quando possibile
- Valida l'input dell'utente (try-except)
- Usa nomi significativi per variabili e funzioni
- Scrivi test unitari per il codice critico

Riepilogo Lezione

✓ Dizionari: strutture chiave-valore mutabili

✓ Stringhe: metodi ricchi e f-string

✓ Funzioni: scope, *args, **kwargs

✓ File: lettura, scrittura, JSON, gestione errori

Domande?

Potrebbero interessarti:

→ Programmazione Orientata agli Oggetti (OOP)

→ Moduli e Pacchetti

→ Framework Web (Django, Flask)