Corso Pratico Python

[Modulo 1- 2026]

2.2. Tipizzazione

2.2.1. Strutture dati del linguaggio

2.2.2. Array

2.2.3. Oggetti di sistema e convenzioni delle librerie

Richiedente: Dott. Roberto Felici [CNR-ISM] • Insegnante: Dott. Scigè J. Liu [INAF-IAPS]

2.3 Approfondimenti

Simboli Speciali e Costanti Built-in

In Python, alcuni termini sono riservati e rappresentano concetti logici o stati dell'oggetto.

1. I Booleani: True e False

Rappresentano i due valori della logica di verità.

Importante: In Python la prima lettera è sempre **maiuscola**.

Derivano dalla classe int: True vale 1, False vale 0.

2. L'assenza di valore: None

None è l'oggetto utilizzato per rappresentare la mancanza di un valore o uno stato "nullo".

Non è uguale a 0, non è una stringa vuota "", e non è False.

Viene restituito di default dalle funzioni che non hanno un return esplicito.

Best Practice: Per verificare se una variabile è None, si usa l'operatore di identità is.

Oltre a quelli citati, esistono altri "oggetti speciali" che incontrerai spesso nelle librerie:

- **`Ellipsis (...)`**: Usato spesso in NumPy per indicare "tutte le altre dimensioni" o come segnaposto per codice non ancora scritto (simile a `pass`).
- **`NotImplemented`**: Un valore speciale restituito dai metodi binari (es. `__eq__`) per indicare che l'operazione non è supportata per quel tipo.
- **`__debug__`**: Una costante booleana che è `True` a meno che Python non venga avviato con l'ottimizzazione (`-O`).

Categoria	Valutato come False	Valutato come True
Costanti	None, False	Qualsiasi altra costante
Numeri	0, 0.0, 0j	Qualsiasi numero $\neq 0$
Sequenze	[], (), "", {}, set()	Se contengono almeno un elemento

2.3 Approfondimenti : strutture di controllo del flusso

2.3.1 **Il Flusso Condizionale: if, elif, else**

È la struttura decisionale di base. valuta le espressioni nell'ordine in cui sono scritte e esegue **solo il primo** blocco la cui condizione risulta True.

if: La condizione primaria.

elif (else if): Condizioni alternative se la prima fallisce. Puoi usarne infiniti.

else: Il "paracadute" finale eseguito se nessuna delle precedenti è vera.

2.3.2 **Iterazione: while e for**

while <condizione>: Ciclo basato sullo **stato**. Continua finché la condizione è vera. Utile quando non sai a priori quanti passaggi serviranno (es. attendere un input utente valido).

for <var> in <enumerazione>: Ciclo basato sulla **collezione**. Itera su elementi di un oggetto "iterable" (liste, stringhe, range). È il modo più pythonico per scorrere dati.

2.3 Approfondimenti : strutture di controllo del flusso

```python
utenti = ["Alice", "Bob", "Admin", "Charlie"]

for utente in utenti:
    if utente == "Admin":
        print(f"Benvenuto Capo! Accesso totale per {utente}.")
    elif utente == "Alice":
        print(f"Bentornata {utente}, hai 3 messaggi non letti.")
    else:
        print(f"Ciao {utente}, profilo standard caricato.")
```

2.3 Approfondimenti : strutture di controllo del flusso

```python
utenti = ["Alice", "Bob", "Admin", "Charlie"]

for utente in utenti:
    if utente == "Admin":
        print(f"Benvenuto Capo! Accesso totale per {utente}.")
    elif utente == "Alice":
        print(f"Bentornata {utente}, hai 3 messaggi non letti.")
    else:
        print(f"Ciao {utente}, profilo standard caricato.")
```

```python
tentativi = 3
password_corretta = "1234"

while tentativi > 0:
    scelta = input(f"Inserisci password ({tentativi} rimasti): ")
    if scelta == password_corretta:
        print("Accesso garantito!")
        break # Esci dal ciclo immediatamente
    else:
        tentativi -= 1
        print("Password errata.")
else:
    # L'else nel while viene eseguito se il ciclo finisce SENZA break
    print("Account bloccato per troppi tentativi.")
```

2.3 Approfondimenti : strutture di controllo del flusso

```python
utenti = ["Alice", "Bob", "Admin", "Charlie"]

for utente in utenti:
    if utente == "Admin":
        print(f"Benvenuto Capo! Accesso totale per {utente}.")
    elif utente == "Alice":
        print(f"Bentornata {utente}, hai 3 messaggi non letti.")
    else:
        print(f"Ciao {utente}, profilo standard caricato.")
```

```python
tentativi = 3
password_corretta = "1234"

while tentativi > 0:
    scelta = input(f"Inserisci password ({tentativi} rimasti): ")
    if scelta == password_corretta:
        print("Accesso garantito!")
        break # Esci dal ciclo immediatamente
    else:
        tentativi -= 1
        print("Password errata.")
else:
    # L'else nel while viene eseguito se il ciclo finisce SENZA break
    print("Account bloccato per troppi tentativi.")
```

```python
# 'with' garantisce la chiusura del file anche se il codice esplode a metà
with open("diario.txt", "w") as file:
    file.write("Caro diario, oggi sto imparando Python.")
    # Alla fine di questo blocco, il file è chiuso automaticamente.
```

2.3 Approfondimenti : strutture di controllo del flusso

```python
utenti = ["Alice", "Bob", "Admin", "Charlie"]

for utente in utenti:
    if utente == "Admin":
        print(f"Benvenuto Capo! Accesso totale per {utente}.")
    elif utente == "Alice":
        print(f"Bentornata {utente}, hai 3 messaggi non letti.")
    else:
        print(f"Ciao {utente}, profilo standard caricato.")
```

```python
tentativi = 3
password_corretta = "1234"

while tentativi > 0:
    scelta = input(f"Inserisci password ({tentativi} rimasti): ")
    if scelta == password_corretta:
        print("Accesso garantito!")
        break # Esci dal ciclo immediatamente
    else:
        tentativi -= 1
        print("Password errata.")
else:
    # L'else nel while viene eseguito se il ciclo finisce SENZA break
    print("Account bloccato per troppi tentativi.")
```

2.3 Approfondimenti : strutture di controllo del flusso

2.3.3 Gestione Errori: try, except, finally

Questa struttura serve a gestire l'imprevisto senza far crashare l'intero programma.

Definizione di Exception e raise

Exception (Eccezione): È un oggetto che rappresenta un errore verificatosi durante l'esecuzione (Runtime Error). A differenza degli errori di sintassi, le eccezioni possono essere "catturate".

> *Esempi:* ZeroDivisionError, FileNotFoundError, ValueError.

raise: È il comando usato per **generare intenzionalmente** un'eccezione.

> *Perché usarlo?* Se scrivi una funzione che riceve un'età negativa, puoi fare raise ValueError("L'età non può essere negativa") per segnalare a chi usa la tua funzione che ha commesso un errore logico.

Il blocco finally

Viene eseguito **sempre**, indipendentemente dal fatto che l'eccezione sia stata sollevata o catturata. È l'ultimo baluardo per le operazioni di pulizia.

2.3.4 Gestione Risorse: with <call()> as <var>

Il costruttore with implementa il cosiddetto **Context Manager**. Serve a garantire che le risorse (file, connessioni a database, socket) vengano chiuse correttamente, anche se si verifica un errore.

Quali errori "tampona"?

Tecnicamente, with non "nasconde" gli errori (non è un try-except), ma **previene i danni collaterali** causati da essi. Nello specifico, evita:

Resource Leaks (Perdita di risorse): Se apri un file e il programma crasha prima di file.close(), quel file potrebbe rimanere bloccato dal sistema operativo. with assicura la chiusura immediata.

Corruzione dei dati: Garantisce che i buffer vengano svuotati (*flushed*) correttamente su disco.

Deadlocks: In contesti multi-threading, assicura che un *lock* venga rilasciato anche se il codice intermedio fallisce.

2.3 Approfondimenti

```python
def divisione_sicura(a, b):
    try:
        if b < 0:
            # Generiamo noi un errore se il divisore è negativo (regola arbitrar
            raise ValueError("Non accettiamo numeri negativi qui!")

        risultato = a / b
        print(f"Il risultato è: {risultato}")

    except ZeroDivisionError as e:
        print(f"Errore matematico: Impossibile dividere per zero. [{e}]")

    except ValueError as e:
        print(f"Errore di validazione: {e}")

    except Exception as e:
        print(f"Accidenti, un errore imprevisto: {e}")

    finally:
        print("Operazione terminata (questo messaggio appare SEMPRE).")

# Testiamo i casi
divisione_sicura(10, 2)  # Caso OK
divisione_sicura(10, 0)  # Caso ZeroDivisionError
divisione_sicura(10, -5) # Caso raise ValueError
```

2.3 Approfondimenti

```python
def divisione_sicura(a, b):
    try:
        if b < 0:
            # Generiamo noi un errore se il divisor
            raise ValueError("Non accettiamo numeri negativi qui.")

        risultato = a / b
        print(f"Il risultato è: {risultato}")

    except ZeroDivisionError as e:
        print(f"Errore matematico: Impossibile dividere per zero. [{e}]")

    except ValueError as e:
        print(f"Errore di validazione: {e}")

    except Exception as e:
        print(f"Accidenti, un errore imprevisto: {e}")

    finally:
        print("Operazione terminata (questo messaggio appare SEMPRE).")

# Testiamo i casi
divisione_sicura(10, 2)  # Caso OK
divisione_sicura(10, 0)  # Caso ZeroDivisionError
divisione_sicura(10, -5) # Caso raise ValueError
```

```python
# 'with' garantisce la chiusura del file anche se il codice esplode a metà
with open("diario.txt", "w") as file:
    file.write("Caro diario, oggi sto imparando Python.")
    # Alla fine di questo blocco, il file è chiuso automaticamente.
```

2.3 Approfondimenti

```python
def divisione_sicura(a, b):
    try:
        if b < 0:
            # Generiamo noi un err
            raise ValueError("Non 

        risultato = a / b
        print(f"Il risultato è: {r

    except ZeroDivisionError as e:
        print(f"Errore matematico:

    except ValueError as e:
        print(f"Errore di validazi

    except Exception as e:
        print(f"Accidenti, un erro

    finally:
        print("Operazione terminat

# Testiamo i casi
divisione_sicura(10, 2)  # Caso OK
divisione_sicura(10, 0)  # Caso Ze
divisione_sicura(10, -5) # Caso ra
```

```python
nome_file = "dati_utenti.txt"

try:
    # 'with' apre il file e si assicura che venga CHIUSO anche in caso di errore
    with open(nome_file, "r", encoding="utf-8") as file:
        print(f"--- Inizio lettura di {nome_file} ---")

        # Il file object è un 'iterable': il ciclo for legge una riga alla volta
        # Questo metodo è ottimo per file enormi perché non carica tutto in RAM
        for numero, riga in enumerate(file, start=1):
            # .strip() rimuove il carattere di "a capo" (\n) a fine riga
            contenuto = riga.strip()

            if contenuto:  # Salta le righe vuote
                print(f"Riga {numero}: {contenuto}")

except FileNotFoundError:
    print(f"Errore: Il file '{nome_file}' non esiste.")
except PermissionError:
    print(f"Errore: Non hai i permessi per leggere '{nome_file}'.")
except Exception as e:
    print(f"Si è verificato un errore imprevisto: {e}")
finally:
    print("--- Operazione conclusa ---")
```

```
lode a metà

nte.
```

2.4 Approfondimenti

In Python non esiste un comando nativo break che permetta di uscire da più cicli nidificati contemporaneamente (il break standard interrompe solo il ciclo più interno).

- Per ottenere questo comportamento in modo elegante e "pulito", si definisce una **Custom Exception**. Questo permette di "saltare" fuori da tutta la struttura dei cicli e atterrare direttamente nel blocco except.

```
# Definizione della custom exception
class ExitLoop(Exception):
    """Eccezione usata per uscire da cicli nidificati."""
    pass

# Simuliamo una ricerca in una griglia 3D (es. coordinate X, Y, Z)
try:
    print("Inizio ricerca nel cubo di dati...")

    for x in range(10):
        for y in range(10):
            for z in range(10):
                # Condizione di uscita: abbiamo trovato quello che cercavamo
                if x == 2 and y == 5 and z == 3:
                    print(f"Bersaglio trovato a: ({x}, {y}, {z})!")
                    # Lanciamo l'eccezione per interrompere TUTTI i for
                    raise ExitLoop

                # Questo print mostra che il ciclo sta lavorando
                # (lo limitiamo per non intasare la console)
                if x < 1 and y < 1:
                    print(f"Scansione: {x},{y},{z}")

except ExitLoop:
    # Il "tampone" intercetta il segnale e ferma la propagazione dell'errore
    print("Uscita d'emergenza dai cicli eseguita con successo.")

finally:
    print("Pulizia sistema completata.")
```