I tipi di dati e le strutture in Python

April 28, 2024

1 TIPI DI DATI E STRUTTURE IN PYTHON

I tipi di dati sono utilizzati per determinare i valori che una variabile può contenere e le operazioni che possono essere eseguite su di essa. Essi possono essere classificati in due categorie principali: semplici e composti (o sequenze).

2 1) I DATI SEMPLICI

I tipi di dati semplici costituiscono le fondamenta dello sviluppo in Python, essenziali come in qualsiasi altro linguaggio di programmazione.

I tipi di dati semplici includono:

- Gli interi (int)
- I numeri in virgola mobile (float)
- I valori booleani (bool)
- I numeri complessi (complex)
- I numeri interi estesi (long)

In altre parole, i tipi di dati semplici comprendono gli interi, i numeri decimali, i valori booleani, i numeri complessi e i numeri interi estesi.

2.1 - GLI INTEGER

Gli integer (int) sono utilizzati per rappresentare valori interi, ovvero numeri senza parte decimale. Con gli integer, è possibile eseguire operazioni aritmetiche di base come l'addizione, la sottrazione, la moltiplicazione, la divisione e la potenza.

[ ]:
```
base = 8
esponente = 2
potenza = base ** esponente

print(f"La potenza di {base} elevato a {esponente} è:")
print(potenza)
print("Il tipo di dato della potenza è:")
print(type(potenza))
```

```
La potenza di 8 elevato a 2 è:
64
```

```
Il tipo di dato della potenza è:
<class 'int'>
```

[ ]:
```python
valore_a = 15
valore_b = 7
differenza = valore_a - valore_b

print(f"La differenza tra {valore_a} e {valore_b} è:")
print(differenza)
print("Il tipo di dato è:")
print(type(differenza))
```

```
La differenza tra 15 e 7 è:
8
Il tipo di dato è:
<class 'int'>
```

[ ]:
```python
valore_a = 12
valore_b = 5
somma = valore_a + valore_b

print(f"La somma di {valore_a} e {valore_b} è:")
print(somma)
print("Il tipo di dato è:")
print(type(somma))
```

```
La somma di 12 e 5 è:
17
Il tipo di dato è:
<class 'int'>
```

[ ]:
```python
valore_a = 20
valore_b = 4
divisione_normale = valore_a / valore_b
divisione_intera = valore_a // valore_b

print(f"La divisione normale tra {valore_a} e {valore_b} è:")
print(divisione_normale)
print(f"La divisione con solo la parte intera tra {valore_a} e {valore_b} è:")
print(divisione_intera)
print("Il tipo di dato della divisione normale è:")
print(type(divisione_normale))
print("Il tipo di dato della divisione con solo la parte intera è:")
print(type(divisione_intera))
```

```
La divisione normale tra 20 e 4 è:
5.0
La divisione con solo la parte intera tra 20 e 4 è:
5
```

```
Il tipo di dato della divisione normale è:
<class 'float'>
Il tipo di dato della divisione con solo la parte intera è:
<class 'int'>
```

[ ]:
```python
valore_primo_int = 6
valore_secondo_int = 4
risultato_moltiplicazione = valore_primo_int * valore_secondo_int

print(f"Il prodotto tra {valore_primo_int} e {valore_secondo_int} è:")
print(risultato_moltiplicazione)
print("Il tipo di dato del prodotto è:")
print(type(risultato_moltiplicazione))
```

```
Il prodotto tra 6 e 4 è:
24
Il tipo di dato del prodotto è:
<class 'int'>
```

2.2 - I FLOAT

I float (float) sono utilizzati per rappresentare valori numerici con la presenza della virgola decimale. Con i float, è possibile effettuare le operazioni aritmetiche fondamentali come l'addizione, la sottrazione, la moltiplicazione, la divisione e la potenza.

[ ]:
```python
valore_primo_float = 10.5
valore_secondo_float = 2.5
valore_terzo_int = 4

risultato_divisione = valore_primo_float / valore_secondo_float
risultato_divisione_intera = valore_primo_float // valore_terzo_int

print(f"Il risultato della divisione tra {valore_primo_float} e␣
 ↪{valore_secondo_float} è:")
print(risultato_divisione)
print("Il tipo di dato è:")
print(type(risultato_divisione))
print(f"Il risultato della divisione intera tra {valore_primo_float} e␣
 ↪{valore_terzo_int} è:")
print(risultato_divisione_intera)
print("Il tipo di dato è:")
print(type(risultato_divisione_intera))
```

```
Il risultato della divisione tra 10.5 e 2.5 è:
4.2
Il tipo di dato è:
<class 'float'>
Il risultato della divisione intera tra 10.5 e 4 è:
2.0
```

```
Il tipo di dato è:
<class 'float'>
```

[ ]:
```python
valore_primo_float = 5.5
valore_secondo_float = 3.5
valore_terzo_int = 2

somma_float = valore_primo_float + valore_secondo_float
somma_float_int = valore_primo_float + valore_terzo_int

print(f"La somma tra {valore_primo_float} e {valore_secondo_float} è:")
print(somma_float)
print("Il tipo di dato della somma è:")
print(type(somma_float))
print(f"La somma tra {valore_primo_float} e {valore_terzo_int} è:")
print(somma_float_int)
print("Il tipo di dato della somma è:")
print(type(somma_float_int))
```

```
La somma tra 5.5 e 3.5 è:
9.0
Il tipo di dato della somma è:
<class 'float'>
La somma tra 5.5 e 2 è:
7.5
Il tipo di dato della somma è:
<class 'float'>
```

[ ]:
```python
valore_primo_float = 12.5
valore_secondo_float = 7.5
valore_terzo_int = 4

differenza_float = valore_primo_float - valore_secondo_float
differenza_float_int = valore_primo_float - valore_terzo_int

print(f"La differenza tra {valore_primo_float} e {valore_secondo_float} è:")
print(differenza_float)
print("Il tipo di dato della differenza è:")
print(type(differenza_float))
print(f"La differenza tra {valore_primo_float} e {valore_terzo_int} è:")
print(differenza_float_int)
print("Il tipo di dato della differenza è:")
print(type(differenza_float_int))
```

```
La differenza tra 12.5 e 7.5 è:
5.0
Il tipo di dato della differenza è:
<class 'float'>
La differenza tra 12.5 e 4 è:
```

```
8.5
Il tipo di dato della differenza è:
<class 'float'>
```

[ ]:
```
valore_primo_float = 7.5
valore_secondo_float = 3.2
valore_terzo_int = 2

prodotto_float = valore_primo_float * valore_secondo_float
prodotto_float_int = valore_primo_float * valore_terzo_int

print(f"Il prodotto tra {valore_primo_float} e {valore_secondo_float} è:")
print(prodotto_float)
print("Il tipo di dato del prodotto è:")
print(type(prodotto_float))
print(f"Il prodotto tra {valore_primo_float} e {valore_terzo_int} è:")
print(prodotto_float_int)
print("Il tipo di dato del prodotto è:")
print(type(prodotto_float_int))
```

```
Il prodotto tra 7.5 e 3.2 è:
24.0
Il tipo di dato del prodotto è:
<class 'float'>
Il prodotto tra 7.5 e 2 è:
15.0
Il tipo di dato del prodotto è:
<class 'float'>
```

[ ]:
```
valore_primo_float = 5.0
valore_secondo_float = 3.0
valore_terzo_int = 2

potenza = valore_primo_float ** valore_secondo_float
divisione_intera = valore_primo_float // valore_terzo_int

print(f"La potenza di {valore_primo_float} e {valore_secondo_float} è:")
print(potenza)
print("Il tipo di dato della potenza è:")
print(type(potenza))
print(f"La divisione intera tra {valore_primo_float} e {valore_terzo_int} è:")
print(divisione_intera)
print("Il tipo di dato della divisione intera è:")
print(type(divisione_intera))
```

```
La potenza di 5.0 e 3.0 è:
125.0
Il tipo di dato della potenza è:
<class 'float'>
```

```
La divisione intera tra 5.0 e 2 è:
2.0
Il tipo di dato della divisione intera è:
<class 'float'>
```

2.3 - I BOOLEANI

I booleani (bool) sono utilizzati per rappresentare valori di verità, ovvero True o False. Con i booleani, è possibile eseguire operazioni logiche fondamentali come l'and, l'or e il not.

O LOGICO

[ ]:
```python
valore_true = True
valore_false = False

## COSTRUIAMO UNA TABELLA DELLA VERITA' PER L'OR

print(f"L'OR di {valore_true} e {valore_false} è:")
risultato_or = valore_true or valore_false
print(risultato_or)
print(f"L'OR di {valore_true} e {valore_true} è:")
risultato_or = valore_true or valore_true
print(risultato_or)
print(f"L'OR di {valore_false} e {valore_true} è:")
risultato_or = valore_false or valore_true
print(risultato_or)
print(f"L'OR di {valore_false} e {valore_false} è:")
risultato_or = valore_false or valore_false
print(risultato_or)
```

```
L'OR di True e False è:
True
L'OR di True e True è:
True
L'OR di False e True è:
True
L'OR di False e False è:
False
```

E LOGICA

[ ]:
```python
valore_true = True
valore_false = False

## COSTRUIAMO UNA TABELLA DELLA VERITA' PER L'AND

print(f"L'AND di {valore_true} e {valore_false} è:")
risultato_and = valore_true and valore_false
print(risultato_and)
print(f"L'AND di {valore_true} e {valore_true} è:")
```

```
risultato_and = valore_true and valore_true
print(risultato_and)
print(f"L'AND di {valore_false} e {valore_true} è:")
risultato_and = valore_false and valore_true
print(risultato_and)
print(f"L'AND di {valore_false} e {valore_false} è:")
risultato_and = valore_false and valore_false
print(risultato_and)
```

```
L'AND di True e False è:
False
L'AND di True e True è:
True
L'AND di False e True è:
False
L'AND di False e False è:
False
```

NOT

[ ]:
```
valore_true = True
valore_false = False

print(f"Il not di {valore_true} è:")
risultato_not_true = not valore_true
print(risultato_not_true)
print(f"Il not di {valore_false} è:")
risultato_not_false = not valore_false
print(risultato_not_false)
```

```
Il not di True è:
False
Il not di False è:
True
```

2.4 - I NUMERI COMPLESSI

I complessi (complex) sono utilizzati per rappresentare valori di variabili corrispondenti a numeri complessi. Con i numeri complessi, è possibile eseguire operazioni aritmetiche di base come l'addizione, la sottrazione, la moltiplicazione e la divisione.

[ ]:
```
complesso_1 = 3 + 1j*3
complesso_2 = 2 + 1j*2
float_1 = 3.0

risultato_complessi = complesso_1 / complesso_2
risultato_complesso_float = complesso_1 / float_1

print(f"La divisione tra {complesso_1} e {complesso_2} è:")
```

```
print(risultato_complessi)
print("Il tipo di dato è:")
print(type(risultato_complessi))
print(f"La divisione tra {complesso_1} e {float_1} è:")
print(risultato_complesso_float)
print("Il tipo di dato è:")
print(type(risultato_complesso_float))
```

```
La divisione tra (3+3j) e (2+2j) è:
(1.5+0j)
Il tipo di dato è:
<class 'complex'>
La divisione tra (3+3j) e 3.0 è:
(1+1j)
Il tipo di dato è:
<class 'complex'>
```

[ ]:
```
complesso_1 = 5 + 1j*4
complesso_2 = 1 + 1j*1
float_1 = 2.5

differenza_complessi = complesso_1 - complesso_2
differenza_complesso_float = complesso_1 - float_1

print(f"La differenza tra {complesso_1} e {complesso_2} è:")
print(differenza_complessi)
print("Il tipo di dato è:")
print(type(differenza_complessi))
print(f"La differenza tra {complesso_1} e {float_1} è:")
print(differenza_complesso_float)
print("Il tipo di dato è:")
print(type(differenza_complesso_float))
```

```
La differenza tra (5+4j) e (1+1j) è:
(4+3j)
Il tipo di dato è:
<class 'complex'>
La differenza tra (5+4j) e 2.5 è:
(2.5+4j)
Il tipo di dato è:
<class 'complex'>
```

[ ]:
```
# Definizione dei numeri complessi
complesso_1 = 3 + 2j
complesso_2 = 1 - 1j

# Operazioni aritmetiche sui numeri complessi
somma = complesso_1 + complesso_2
```

```
differenza = complesso_1 - complesso_2
prodotto = complesso_1 * complesso_2
divisione = complesso_1 / complesso_2

# Stampare i risultati
print("Somma:", somma)
print("Differenza:", differenza)
print("Prodotto:", prodotto)
print("Divisione:", divisione)
```

```
Somma: (4+1j)
Differenza: (2+3j)
Prodotto: (5-1j)
Divisione: (0.5+2.5j)
```

[ ]:
```
# Definizione dei numeri complessi
complesso_a = 2 + 3j
complesso_b = 1 - 2j

# Operazioni aritmetiche sui numeri complessi
somma = complesso_a + complesso_b
differenza = complesso_a - complesso_b
prodotto = complesso_a * complesso_b
divisione = complesso_a / complesso_b

# Stampare i risultati
print("Somma:", somma)
print("Differenza:", differenza)
print("Prodotto:", prodotto)
print("Divisione:", divisione)
```

```
Somma: (3+1j)
Differenza: (1+5j)
Prodotto: (8-1j)
Divisione: (-0.8+1.4j)
```

3 2) SEQUENZE

Le sequenze sono tipi di dati più elaborati rispetto a quelli semplici e mantengono un ordine specifico di elementi, che possono includere sia tipi di dati semplici che complessi. Tuttavia, va notato che le stringhe sono un'eccezione a questa regola, in quanto rappresentano sequenze di caratteri che formano parole o frasi.

Le sequenze possono essere classificate in due categorie principali: mutabili e immutabili. La distinzione fondamentale tra le due risiede nel fatto che le sequenze mutabili consentono modifiche ai loro elementi dopo la creazione, mentre le sequenze immutabili mantengono i loro elementi invariati una volta creati.

Le sequenze mutabili includono:

- Le liste (list)
- I dizionari (dict)
- Gli insiemi (set)

In altre parole, le sequenze mutabili sono rappresentate dalle liste, dai dizionari, dagli insiemi e dai file, che consentono modifiche ai loro contenuti dopo la creazione.

Le sequenze immutabili includono:

- Le stringhe (str)
- Le tuple (tuple)
- Gli insiemi immutabili (frozenset)

In altre parole, le sequenze immutabili comprendono le stringhe, le tuple e gli insiemi immutabili, i quali non possono essere modificati una volta creati.

3.1 - LISTE MUTABILI

3.1.1 - LE LISTE

Una lista è una struttura dati che può contenere una collezione ordinata di elementi. Gli elementi all'interno di una lista possono essere di qualsiasi tipo di dati, inclusi numeri, stringhe, liste annidate, tuple, dizionari, oggetti e così via. Le liste sono definite tra parentesi quadre [] e gli elementi sono separati da virgole.

Le liste sono dinamiche e possono essere modificate dopo la loro creazione. È possibile aggiungere, rimuovere e modificare gli elementi all'interno di una lista in base alle esigenze del programma.

[ ]:
```
lista_numeri = [1, 2, 3, 4, 5]
lista_stringhe = ["ciao", "mondo", "!"]
lista_mista = [1, "due", 3.0, [4, 5]]

print(lista_numeri)  # Output: [1, 2, 3, 4, 5]
print(lista_stringhe)  # Output: ['ciao', 'mondo', '!']
print(lista_mista)  # Output: [1, 'due', 3.0, [4, 5]]
```

```
[1, 2, 3, 4, 5]
['ciao', 'mondo', '!']
[1, 'due', 3.0, [4, 5]]
```

[ ]:
```
# Creazione di una lista e accesso agli elementi
# Definizione di una lista di numeri
numeri = [1, 2, 3, 4, 5]

# Accesso agli elementi della lista tramite indicizzazione
print(numeri[0])  # Output: 1
print(numeri[2])  # Output: 3
print(numeri[-1])  # Output: 5 (elemento dall'ultimo)
```

```
1
3
5
```

[ ]:
```python
# Modifica degli elementi di una lista
# Definizione di una lista di stringhe
stringhe = ["ciao", "mondo", "!"]

# Modifica di un elemento della lista
stringhe[1] = "Python"
print(stringhe)  # Output: ['ciao', 'Python', '!']
```

```
['ciao', 'Python', '!']
```

[ ]:
```python
# Aggiunta e rimozione di elementi da una lista
# Aggiunta di un elemento alla fine della lista
numeri.append(6)
print(numeri)  # Output: [1, 2, 3, 4, 5, 6]

# Rimozione di un elemento dalla lista
numeri.remove(3)
print(numeri)  # Output: [1, 2, 4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 4, 5, 6]
```

[ ]:
```python
# Iterazione attraverso una lista
# Iterazione attraverso gli elementi della lista
for numero in numeri:
    print(numero)
```

[ ]:
```python
# Unione di liste
# Definizione di due liste
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]

# Unione delle due liste
lista_unione = lista1 + lista2
print(lista_unione)  # Output: [1, 2, 3, 4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

3.1.2 - I DIZIONARI

Un dizionario è una struttura dati che consente di memorizzare una collezione di coppie chiave-valore. Le chiavi all'interno di un dizionario devono essere uniche e immutabili, mentre i valori possono essere di qualsiasi tipo di dato, inclusi numeri, stringhe, liste, tuple, dizionari, oggetti e così via. I dizionari sono definiti tra parentesi graffe {} e le coppie chiave-valore sono separate da virgole, con ogni coppia chiave-valore separata da due punti :.

[ ]:
```python
dizionario = {
    "nome": "Mario",
    "cognome": "Rossi",
```

```
    "età": 30,
    "indirizzo": {
        "via": "Via Roma",
        "città": "Roma",
        "CAP": "00100"
    }
}

print(dizionario)  # Output: {'nome': 'Mario', 'cognome': 'Rossi', 'età': 30,␣
 ↪'indirizzo': {'via': 'Via Roma', 'città': 'Roma', 'CAP': '00100'}}
```

```
{'nome': 'Mario', 'cognome': 'Rossi', 'età': 30, 'indirizzo': {'via': 'Via
Roma', 'città': 'Roma', 'CAP': '00100'}}
```

[ ]:
```
# Creazione di un dizionario e accesso ai valori tramite chiavi
# Definizione di un dizionario di persone con nome e età
persone = {
    "Alice": 30,
    "Bob": 25,
    "Charlie": 35
}

# Accesso ai valori tramite chiavi
print(persone["Alice"])  # Output: 30
print(persone["Bob"])  # Output: 25
```

```
30
25
```

[ ]:
```
# Aggiunta e rimozione di elementi da un dizionario
# Aggiunta di un nuovo elemento al dizionario
persone["David"] = 40
print(persone)  # Output: {'Alice': 30, 'Bob': 25, 'Charlie': 35, 'David': 40}

# Rimozione di un elemento dal dizionario
del persone["Bob"]
print(persone)  # Output: {'Alice': 30, 'Charlie': 35, 'David': 40}
```

```
{'Alice': 30, 'Bob': 25, 'Charlie': 35, 'David': 40}
{'Alice': 30, 'Charlie': 35, 'David': 40}
```

[ ]:
```
# Iterazione attraverso un dizionario
# Iterazione attraverso le chiavi del dizionario
for nome in persone:
    print(nome, persone[nome])
```

```
Alice 30
Charlie 35
David 40
```

[ ]:
```python
# Controllo dell'esistenza di una chiave in un dizionario
# Verifica se una chiave esiste nel dizionario
if "Alice" in persone:
    print("Alice è nel dizionario")
```

```
Alice è nel dizionario
```

[ ]:
```python
# Ottenere una lista di chiavi o valori da un dizionario
# Ottenere una lista di chiavi dal dizionario
chiavi = persone.keys()
print(chiavi)  # Output: dict_keys(['Alice', 'Charlie', 'David'])

# Ottenere una lista di valori dal dizionario
valori = persone.values()
print(valori)  # Output: dict_values([30, 35, 40])
```

3.1.3 - GLI INSIEMI

Un insieme è una struttura dati non ordinata che contiene elementi unici. Gli insiemi sono definiti tra parentesi graffe {} e gli elementi sono separati da virgole. Gli insiemi possono contenere elementi di qualsiasi tipo di dato immutabile, come numeri, stringhe, tuple, ma non possono contenere elementi mutabili come liste o altri insiemi.

Gli insiemi supportano operazioni come unione, intersezione, differenza e verifica di appartenenza. Essi sono utilizzati quando è necessario memorizzare una collezione di elementi unici e l'ordine degli elementi non è importante.

[ ]:
```python
insieme_numeri = {1, 2, 3, 4, 5}
insieme_stringhe = {"ciao", "mondo", "!"}
insieme_misto = {1, "due", 3.0}

print(insieme_numeri)  # Output: {1, 2, 3, 4, 5}
print(insieme_stringhe)  # Output: {'ciao', 'mondo', '!'}
print(insieme_misto)  # Output: {'due', 1, 3.0}
```

```
{1, 2, 3, 4, 5}
{'mondo', 'ciao', '!'}
{1, 3.0, 'due'}
```

[ ]:
```python
# Creazione di un insieme e accesso agli elementi
# Definizione di un insieme di numeri
insieme_numeri = {1, 2, 3, 4, 5}

# Accesso agli elementi dell'insieme
for numero in insieme_numeri:
    print(numero)
```

```
1
2
3
```

```
4
5
```

[ ]:
```python
# Aggiunta e rimozione di elementi da un insieme
# Aggiunta di un elemento all'insieme
insieme_numeri.add(6)
print(insieme_numeri)  # Output: {1, 2, 3, 4, 5, 6}

# Rimozione di un elemento dall'insieme
insieme_numeri.remove(3)
print(insieme_numeri)  # Output: {1, 2, 4, 5, 6}
```

```
{1, 2, 3, 4, 5, 6}
{1, 2, 4, 5, 6}
```

[ ]:
```python
# Unione, intersezione e differenza tra insiemi
# Definizione di due insiemi
insieme1 = {1, 2, 3, 4, 5}
insieme2 = {4, 5, 6, 7, 8}

# Unione degli insiemi
insieme_unione = insieme1.union(insieme2)
print(insieme_unione)  # Output: {1, 2, 3, 4, 5, 6, 7, 8}

# Intersezione degli insiemi
insieme_intersezione = insieme1.intersection(insieme2)
print(insieme_intersezione)  # Output: {4, 5}

# Differenza tra gli insiemi
insieme_differenza = insieme1.difference(insieme2)
print(insieme_differenza)  # Output: {1, 2, 3}
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
{4, 5}
{1, 2, 3}
```

[ ]:
```python
# Verifica di appartenenza e confronto tra insiemi
# Verifica se un elemento appartiene all'insieme
if 3 in insieme_numeri:
    print("3 appartiene all'insieme")

# Verifica se un insieme è un sottoinsieme di un altro
if insieme1.issubset(insieme_unione):
    print("insieme1 è un sottoinsieme di insieme_unione")
```

```
insieme1 è un sottoinsieme di insieme_unione
```

3.2 - LISTE IMMUTABILI

3.2.1 - LE STRINGHE

Una stringa è una sequenza di caratteri immutabile, che può essere utilizzata per rappresentare testo. Le stringhe sono definite tra virgolette singole (') o doppie (“), e possono contenere qualsiasi tipo di carattere, come lettere, numeri, simboli e spazi.

Le stringhe supportano molte operazioni, come l'accesso agli elementi attraverso l'indicizzazione, il taglio delle sottostringhe, la concatenazione, la ripetizione, la ricerca di sottostringhe, la sostituzione e molto altro. Sono uno dei tipi di dati più fondamentali e ampiamente utilizzati in Python.

[ ]:
```
stringa_singola = 'Questo è un esempio di stringa.'
stringa_doppia = "Anche questa è una stringa, ma con virgolette doppie."
stringa_mista = 'Le stringhe possono contenere numeri, come 123, e simboli,␣
 ↪come @#$.'

print(stringa_singola)
print(stringa_doppia)
print(stringa_mista)
```

```
Questo è un esempio di stringa.
Anche questa è una stringa, ma con virgolette doppie.
Le stringhe possono contenere numeri, come 123, e simboli, come @#$.
```

[ ]:
```
# Concatenazione di stringhe
# Definizione di due stringhe
stringa1 = "Ciao"
stringa2 = "mondo!"

# Concatenazione delle stringhe
stringa_concatenata = stringa1 + " " + stringa2
print(stringa_concatenata)  # Output: Ciao mondo!
```

```
Ciao mondo!
```

[ ]:
```
# Accesso ai caratteri di una stringa tramite indicizzazione
# Accesso ai singoli caratteri di una stringa
testo = "Python"
print(testo[0])  # Output: P
print(testo[1])  # Output: y
print(testo[-1])  # Output: n (ultimo carattere)
```

```
P
y
n
```

[ ]:
```
# Taglio di una stringa per ottenere una sottostringa
testo = "Python è un linguaggio di programmazione"
sottostringa = testo[7:19]
```

```
print(sottostringa)  # Output: è un linguag
```

```
è un linguag
```

[ ]:
```
# Ricerca di una sottostringa all'interno di una stringa
testo = "Python è un linguaggio di programmazione"
if "linguaggio" in testo:
    print("La parola 'linguaggio' è presente nella stringa.")
```

```
La parola 'linguaggio' è presente nella stringa.
```

3.2.2 - LE TUPLE

Una tuple è una sequenza ordinata e immutabile di elementi. Le tuple sono simili alle liste, ma differiscono per il fatto che le tuple sono immutabili, il che significa che una volta definite, non è possibile modificarle. Le tuple sono definite tra parentesi tonde () e gli elementi sono separati da virgole.

Come le liste, le tuple possono contenere elementi di qualsiasi tipo di dato, inclusi numeri, stringhe, tuple annidate, liste, dizionari e così via. Tuttavia, poiché le tuple sono immutabili, non è possibile aggiungere, rimuovere o modificare gli elementi all'interno di una tuple dopo la sua creazione. Le tuple sono spesso utilizzate per rappresentare collezioni di dati che non devono essere modificate.

[ ]:
```
tuple_numeri = (1, 2, 3, 4, 5)
tuple_stringhe = ("ciao", "mondo", "!")
tuple_mista = (1, "due", 3.0)

print(tuple_numeri)  # Output: (1, 2, 3, 4, 5)
print(tuple_stringhe)  # Output: ('ciao', 'mondo', '!')
print(tuple_mista)  # Output: (1, 'due', 3.0)
```

```
(1, 2, 3, 4, 5)
('ciao', 'mondo', '!')
(1, 'due', 3.0)
```

[ ]:
```
# Creazione di una tuple
tuple1 = (1, 2, 3, 4, 5)
print(tuple1)
```

```
(1, 2, 3, 4, 5)
```

[ ]:
```
# Accesso agli elementi di una tuple
tuple1 = (1, 2, 3, 4, 5)
print(tuple1[0])  # Stampa il primo elemento della tuple
print(tuple1[2])  # Stampa il terzo elemento della tuple
```

```
1
3
```

[ ]:
```
# Iterazione attraverso una tuple
tuple1 = (1, 2, 3, 4, 5)
for item in tuple1:
    print(item)
```

```
1
2
3
4
5
```

[ ]:
```
# Concatenazione di tuple
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
tuple3 = tuple1 + tuple2
print(tuple3)
```

```
(1, 2, 3, 4, 5, 6)
```

[ ]:
```
# Slicing di tuple
tuple1 = (1, 2, 3, 4, 5)
slice_tuple = tuple1[1:4]  # Estrae una porzione della tuple
print(slice_tuple)
```

```
(2, 3, 4)
```

[ ]:
```
# Verifica di un elemento nella tuple
tuple1 = (1, 2, 3, 4, 5)
if 3 in tuple1:
    print("3 è presente nella tuple.")
```

```
3 è presente nella tuple.
```

[ ]:
```
# Creazione di tuple con valori di tipo diverso
mixed_tuple = ("apple", 5, True)
print(mixed_tuple)
```

```
('apple', 5, True)
```

[ ]:
```
# Assegnazione multipla utilizzando tuple
point = (3, 7)
x, y = point
print("x =", x)
print("y =", y)
```

```
x = 3
y = 7
```

3.2.3 - GLI INSIEMI IMMUTABILI

Un insieme immutabile è una collezione non ordinata di elementi unici che non può essere modificata dopo la sua creazione. Gli insiemi immutabili sono rappresentati dall'oggetto frozenset. Come gli insiemi ordinari, gli insiemi immutabili possono contenere elementi di qualsiasi tipo di dato immutabile, come numeri, stringhe, tuple, ma non possono contenere elementi mutabili come liste o altri insiemi.

Una volta creato un insieme immutabile, non è possibile aggiungere, rimuovere o modificare gli elementi al suo interno. Gli insiemi immutabili sono utilizzati quando si desidera avere una collezione di elementi unici che non può essere modificata.

[ ]:
```
insieme_imm = frozenset([1, 2, 3, 4, 5])
print(insieme_imm)  # Output: frozenset({1, 2, 3, 4, 5})
```

```
frozenset({1, 2, 3, 4, 5})
```

[ ]:
```
# Accesso agli elementi di un insieme immutabile
immutable_set = frozenset({1, 2, 3, 4, 5})
for item in immutable_set:
    print(item)
```

```
1
2
3
4
5
```

[ ]:
```
# Unione di insiemi immutabili
set1 = frozenset({1, 2, 3})
set2 = frozenset({3, 4, 5})
union_set = set1 | set2
print(union_set)
```

```
frozenset({1, 2, 3, 4, 5})
```

[ ]:
```
# Intersezione di insiemi immutabili
set1 = frozenset({1, 2, 3, 4})
set2 = frozenset({3, 4, 5, 6})
intersection_set = set1 & set2
print(intersection_set)
```

```
frozenset({3, 4})
```

[ ]:
```
# Differenza di insiemi immutabili
set1 = frozenset({1, 2, 3, 4})
set2 = frozenset({3, 4, 5, 6})
difference_set = set1 - set2
print(difference_set)
```

```
frozenset({1, 2})
```

[ ]:
```python
# Verifica di un sottoinsieme in un insieme immutabile
set1 = frozenset({1, 2, 3})
set2 = frozenset({1, 2})
if set2.issubset(set1):
    print("set2 è un sottoinsieme di set1")
```

```
set2 è un sottoinsieme di set1
```

[ ]:
```python
# Verifica di disgiunzione tra insiemi immutabili
set1 = frozenset({1, 2, 3})
set2 = frozenset({4, 5, 6})
if set1.isdisjoint(set2):
    print("set1 e set2 sono disgiunti")
```

```
set1 e set2 sono disgiunti
```