PYTHON TUPLES

A Comprehensive Guide for College Students

Introduction to Python Tuples

● Tuples are ordered, immutable collections in Python.

● Defined using parentheses: my_tuple = (1, 2, 3).

● Can store elements of different data types.

● Example: my_tuple = ('apple', 3, True).

Accessing Values in Tuples

- Access tuple elements using indexing.
- Example:

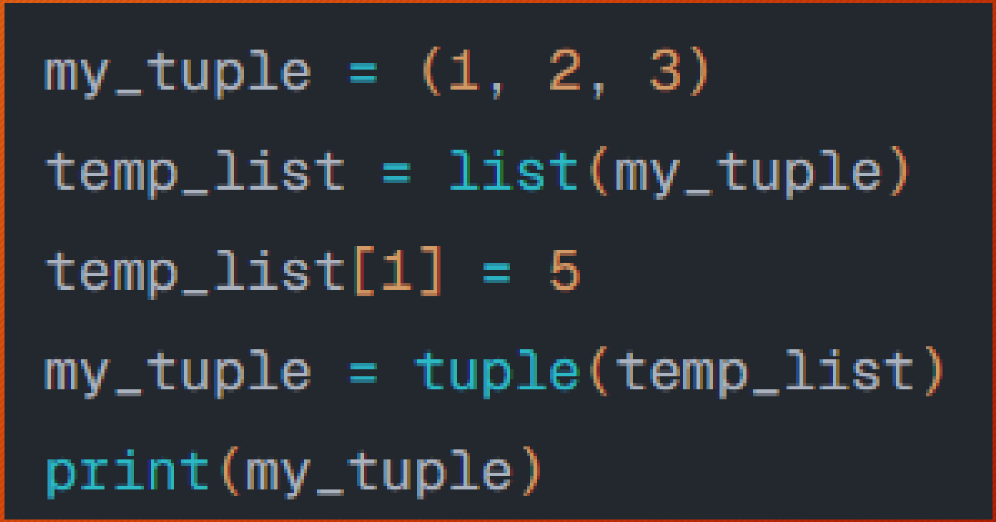
```
my_tuple = (10, 20, 30)
print(my_tuple[1])  # Output: 20
```

- Use negative indexing to access from the end.

```
print(my_tuple[-1])  # Output: 30
```

Updating Tuples

- Tuples are immutable; you cannot directly update them.
- Workaround: Convert to a list, update, and convert back.
- Example:

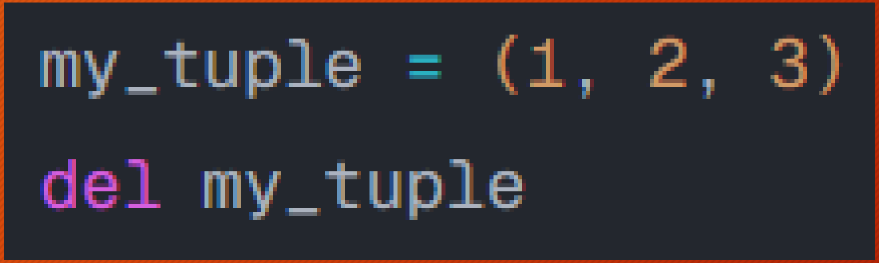


```python
my_tuple = (1, 2, 3)
temp_list = list(my_tuple)
temp_list[1] = 5
my_tuple = tuple(temp_list)
print(my_tuple)
```

- OUPUT: (1, 5, 3)

Delete Tuple Elements

- Tuples are immutable, so you cannot delete specific elements.
- You can delete the entire tuple using the del statement.
- Example:



```python
my_tuple = (1, 2, 3)
del my_tuple
```

# my_tuple no longer exists.

Basic Tuple Operations

- ● Concatenation (+): Combines two tuples.
  Example: (1, 2) + (3, 4) → (1, 2, 3, 4)

● Repetition (*): Repeats a tuple.
  Example: (1, 2) * 2 → (1, 2, 1, 2)

● Membership (in): Checks if an element exists.
  Example: 3 in (1, 2, 3) → True

Indexing, Slicing, and Matrixes

● Indexing: Access elements by their position.

● Slicing: Extract sub-tuples.
Example: my_tuple[1:3] → (2, 3)

● Nested tuples represent matrices.
Example: matrix = ((1, 2), (3, 4))
Access matrix[1][0] → 3

No Enclosing Delimiters

- Tuples can be created without parentheses in assignments.
- Example:

```
my_tuple = 1, 2, 3
print(my_tuple)  # Output: (1, 2, 3)
```

Built-in Tuple Functions

- len(tuple): Returns the number of elements.
- max(tuple): Returns the largest element.
- min(tuple): Returns the smallest element.

When to Use Tuples vs Lists

- Use **tuples** for:
  - Data that shouldn’t change (e.g., coordinates, configuration settings).
  - As keys in dictionaries or elements in sets.
  - Performance-critical operations.
- Use **lists** for:
  - Dynamic data that requires frequent modification.
  - Operations requiring built-in methods like sorting or extending.