Tuples in Python

A tuple is an ordered, immutable collection of items in Python, defined by enclosing elements in parentheses (). Tuples are similar to lists but cannot be changed after creation (immutable), making them useful for fixed collections of data.

Characteristics of Tuples

- Ordered: Items have a defined order accessed via zero-based indexing.
- Immutable: Once created, tuples cannot be modified, meaning no addition, removal, or change of elements.
- Allow Duplicates: Tuples can contain duplicate values.
- Heterogeneous: Tuples can hold elements of different data types.
- Indexable and Iterable: You can access tuple elements by index and iterate over them in loops.
- Can be Nested: Tuples can contain other tuples or collections as elements.
- Memory Efficient: Tuples use less memory compared to lists due to immutability.
- Hashable: Tuples can be used as keys in dictionaries if all their elements are immutable.

Creating Tuples

python

```
my_tuple = ("apple", "banana", "cherry")
single_element_tuple = ("apple",)  # Note the trailing comma for single element
empty_tuple = ()
```

Accessing Tuple Elements

python

```
print(my_tuple[0])   # 'apple'
print(my_tuple[-1])  # 'cherry'
```

Tuple Operations

- Concatenation: tuple1 + tuple2
- Repetition: tuple1 * 3
- Membership test: 'apple' in my_tuple
- Slicing: my_tuple[1:3]

Why Use Tuples?

- Tuples protect data integrity by preventing modification.
- Suitable for fixed datasets like coordinates, RGB colors, or database records.
- Lightweight and faster than lists for read-only data.
- Can be dictionary keys due to immutability.

Example

python

```
coordinates = (10, 20, 30)
print(f"The coordinates are: {coordinates}")
# coordinates[0] = 15  # This will raise a TypeError
```

Dictionaries in Python

A dictionary is a built-in Python data structure used to store data in key-value pairs. Each key in a dictionary is unique and maps to a corresponding value.

Characteristics of Dictionaries

- Unordered: Dictionaries do not guarantee the order of items before Python 3.7. From Python 3.7 onwards, dictionaries maintain insertion order.
- Mutable: You can modify a dictionary after creation by adding, updating, or removing key-value pairs.
- Indexed by Keys: Unlike lists, which are indexed by numbers, dictionaries are indexed by keys.
- Keys Must be Immutable: Keys can be strings, numbers, or tuples (if they contain only immutable elements), but cannot be mutable types like lists.
- Values Can Be Any Data Type: Values can be strings, numbers, lists, dictionaries, or any other Python object.
- Keys Are Unique: If duplicate keys are added, the latest value overwrites the previous one.

Creating Dictionaries

Using curly braces {} with key-value pairs separated by colons:

python

```python
person = {
   "name": "Alice",
   "age": 30,
   "city": "New York"
}
```

Using the dict() constructor:

python

```python
person = dict(name="Alice", age=30, city="New York")
```

Accessing and Modifying

Access values by keys:

python

```python
print(person["name"])  # Output: Alice
```

Add or update values:

python

```python
person["email"] = "alice@example.com"
```

Remove items using del or pop():

python

```python
del person["age"]
email = person.pop("email")
```

Common Dictionary Methods

- keys(): Get all keys
- values(): Get all values

- items(): Get all key-value pairs as tuples
- get(key, default): Safely access a key with a default if not found
- update(other_dict): Update dictionary with another dictionary's pairs
- clear(): Remove all items

Example

python

```
student = {
   "id": 1,
   "name": "John",
   "marks": [90, 85, 88]
}

print(student.get("name"))  # John
student["grade"] = "A"
print(student)
```