Fundamentals of Python

PYTHON TUPLE, SET & DICTIONARY

Shruti Bharat

@shrutibharat0105

Table of Contents

Tuple

- **Ordered**: The elements in a tuple have a defined order.
- **Immutable**: The elements of a tuple cannot be changed after the tuple is created.
- **Heterogeneous:** Tuples can contain elements of different types.
- **Indexable:** Elements can be accessed using an index.
- **Duplicates:** Tuples can contain duplicates

Tuple Vs List

1. Immutable **Vs** Mutable
2. Parentheses ( ) **Vs** Square brackets [ ]
3. Fast **Vs** Slow
4. Less memory **Vs** More memory

Creating a tuple

```python
# Single item tuple
t1 = (42,)
print("Single item tuple:", t1)
# Output: Single item tuple: (42,)

# Two-item tuple
t2 = (1, "hi")
print("Two-item tuple:", t2)
# Output: Two-item tuple: (1, 'hi')

# Nested tuple
t_nested = (1, (2, 3), (4, 5, (6, 7)))
print("Nested tuple:", t_nested)
# Output: Nested tuple: (1, (2, 3), (4, 5, (6, 7)))

# 2D tuple (matrix)
t2d = (
    (1, 2, 3),
    (4, 5, 6),
    (7, 8, 9)
)
print("2D tuple (matrix):", t2d)
# Output: 2D tuple (matrix): ((1, 2, 3), (4, 5, 6), (7, 8, 9))


my_tuple = (1, "Hello", 3.14, [4, 5, 6], ("a", "b"))

# Accessing elements by index
print("First element:", my_tuple[0])  # Output: 1
print("Second element:", my_tuple[1])  # Output: Hello

# Nested elements can be accessed using multiple indices
print("Fourth element, first sub-element:", my_tuple[3][0])
# Output: 4
```

Operations on tuple

```python
# Define two tuples for arithmetic operations
t1 = (1, 2, 3)
t2 = (4, 5, 6)

# Arithmetic operations (concatenation and repetition)
t_concat = t1 + t2  # Concatenation
t_repeat = t1 * 2   # Repetition

print("Concatenated tuple:", t_concat)
# Output: Concatenated tuple: (1, 2, 3, 4, 5, 6)

print("Repeated tuple:", t_repeat)
# Output: Repeated tuple: (1, 2, 3, 1, 2, 3)

# Membership testing
item = 2
is_member = item in t1
print(f"Is {item} in tuple t1?:", is_member)
# Output: Is 2 in tuple t1?: True

# Iteration over a tuple
print("Iterating over tuple t1:")
for elem in t1:
    print(elem)
# Output:
# Iterating over tuple t1:
# 1
# 2
# 3
```

Tuple functions

```python
# Creating a tuple
t = (1, 2, 3, 4, 2, 5, 2)

# len() function to get the length of the tuple
length = len(t)
print("Length of tuple:", length)
# Output: Length of tuple: 7

# max() function to get the maximum value in the tuple
max_value = max(t)
print("Maximum value in tuple:", max_value)
# Output: Maximum value in tuple: 5

# min() function to get the minimum value in the tuple
min_value = min(t)
print("Minimum value in tuple:", min_value)
# Output: Minimum value in tuple: 1

# sum() function to get the sum of all elements in the tuple
sum_value = sum(t)
print("Sum of all elements in tuple:", sum_value)
# Output: Sum of all elements in tuple: 19

# count() method to count the occurrences of a specific element
count_2 = t.count(2)
print("Count of 2 in tuple:", count_2)
# Output: Count of 2 in tuple: 3

# index() method to find the index of the first occurrence of a specific element
index_2 = t.index(2)
print("Index of first occurrence of 2:", index_2)
# Output: Index of first occurrence of 2: 1

# sorted() function to return a sorted list of the tuple's elements
sorted_t = sorted(t)
print("Sorted tuple elements:", sorted_t)
# Output: Sorted tuple elements: [1, 2, 2, 2, 3, 4, 5]
```

Tuple packing & unpacking

```
# Tuple Packing
packed_tuple = 1, "apple", 3.14
print("Packed tuple:", packed_tuple)
# Output: Packed tuple: (1, 'apple', 3.14)

# Tuple Unpacking
a, b, c = packed_tuple
print("Unpacked values:", a, b, c)
# Output: Unpacked values: 1 apple 3.14

# Unpacking with variable-length argument (*)
t = (1, 2, 3, 4, 5)
x, y, *rest = t
print("x:", x)
print("y:", y)
print("rest:", rest)
# Output: x: 1
# Output: y: 2
# Output: rest: [3, 4, 5]

# Zipping tuples
t1 = (1, 2, 3)
t2 = ('a', 'b', 'c')
zipped = tuple(zip(t1, t2))
print("Zipped tuple:", zipped)
# Output: Zipped tuple: ((1, 'a'), (2, 'b'), (3, 'c'))
```

Set

1. **Unordered:** The elements in a set do not have a specific order.
2. **Unique Elements:** A set automatically removes duplicate elements.
3. **Mutable**: You can add and remove elements from a set.
4. **No Indexing:** Since sets are unordered, they do not support indexing or slicing.
5. **Dynamic Size:** Sets can grow and shrink as elements are added and removed.

```
# Creating an empty set
empty_set = set()
print("Empty set:", empty_set)

# Using curly braces
set1 = {1, 2, 3, 4}
print("Set created using curly braces:", set1)

# Hetrogeneous set
Heterogeneous_set: {1, 'hello', 3.14, (4, 5)}
print("Hetrogeneous set:", Heterogeneous_set)
```

Adding & deleting items

```
# Create a set
my_set = {1, 2, 3}

# Add an item to the set using add()
my_set.add(4)
print("Set after adding 4:", my_set)

# Add multiple items to the set using update()
my_set.update({5, 6})
print("Set after updating with {5, 6}:", my_set)

# Add multiple items to the set using update() with another set
my_set.update({7, 8}, {9, 10})
print("Set after updating with {7, 8} and {9, 10}:", my_set)

# Remove an item from the set using remove()
my_set.remove(10)
print("Set after removing 10:", my_set)

# Remove an item from the set using discard()
# (handles non-existent items gracefully)
my_set.discard(11)
print("Set after discarding 11:", my_set)

# Remove and return an arbitrary item from the set using pop()
popped_item = my_set.pop()
print("Popped item:", popped_item)
print("Set after popping:", my_set)

# Remove all items from the set using clear()
my_set.clear()
print("Set after clearing:", my_set)
```

Set operations

```python
# Define two sets
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

# Union of sets
union_set = set1 | set2  # or set1.union(set2)
print("Union of sets:", union_set)

# Intersection of sets
intersection_set = set1 & set2  # or set1.intersection(set2)
print("Intersection of sets:", intersection_set)

# Difference of sets
difference_set = set1 - set2  # or set1.difference(set2)
print("Difference of set1 - set2:", difference_set)

# Symmetric difference of sets
symmetric_difference_set = set1 ^ set2
# or set1.symmetric_difference(set2)
print("Symmetric difference of sets:", symmetric_difference_set)

# Subset and Superset
subset = {3, 4}
print("Is subset:", subset.issubset(set1))
print("Is superset:", set1.issuperset(subset))

# Disjoint sets
set3 = {5, 6}
print("Are set1 and set3 disjoint?:", set1.isdisjoint(set3))
```

Set functions

```python
# Define two sets
set1 = {1, 2, 3}
set2 = {3, 4, 5}

# Union of sets using union() method
union_set = set1.union(set2)
print("Union of sets:", union_set) # {1,2,3,5,6}

# Intersection of sets using intersection() method
intersection_set = set1.intersection(set2)
print("Intersection of sets:", intersection_set) # {3}

# Difference of sets using difference() method
difference_set = set1.difference(set2)
print("Difference of set1 - set2:", difference_set) # {1,2}

# Symmetric difference of sets using symmetric_difference() method
symmetric_difference_set = set1.symmetric_difference(set2)
print("Symmetric difference of sets:", symmetric_difference_set)
# {1,2,4,5}

# Check if a set is a subset of another using issubset() method
subset = {3, 4}
print("Is subset:", subset.issubset(set1)) # False

# Check if a set is a superset of another using issuperset() method
print("Is superset:", set1.issuperset(subset)) # False

# Check if two sets are disjoint using isdisjoint() method
set3 = {5, 6}
print("Are set1 and set3 disjoint?:", set1.isdisjoint(set3)) # False

# Length of a set using len() function
print("Length of set1:", len(set1)) # 3

# Maximum and minimum elements of a set using max() and min() fun
print("Maximum element of set1:", max(set1)) # 3
print("Minimum element of set1:", min(set1)) # 1
```

Frozenset

- Frozensets are like immutable sets in Python.
- Once created, their elements cannot be changed, added, or removed.
- They're useful for storing unique, immutable collections and can be used as keys in dictionaries or elements in other sets.
- All read functions of a set will work here as well, but write functions of a set will not work.

```
# Creating a frozenset
frozen_set = frozenset([1, 2, 3, 4, 5])
print("Frozenset:", frozen_set) # {1,2,3,4,5}

# set comprehension
squares_set = {x**2 for x in range(10)}

# Printing the resulting set
print("Set of squares from 0 to 9:", squares_set)
# Set of squares from 0 to 9: {0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
# remember that set is unordered
```

Dictionary

- **Unordered:** The order of elements in a dictionary is not guaranteed.
- **Key-Value Pairs:** Each element in a dictionary consists of a key-value pair. The key is used to access the corresponding value.
- **Mutable:** Dictionaries are mutable, meaning you can add, remove, or modify key-value pairs.
- **Key Uniqueness:** Keys within a dictionary must be unique. If you try to add a duplicate key, the existing value associated with that key will be overwritten.

Creating a dictionary

```
# Method 1: Using curly braces
dict1 = {'a': 1, 'b': 2, 'c': 3}
print("Dictionary1:", dict1)
# {'a': 1, 'b': 2, 'c': 3}

# Method 2: Using dict() constructor with keyword arguments
dict2 = dict(x=10, y=20, z=30)
print("Dictionary2:", dict2)
# {'x': 10, 'y': 20, 'z': 30}

# Method 3: Using dict() constructor with a list of tuples
list_of_tuples = [('x', 100), ('y', 200), ('z', 300)]
dict3 = dict(list_of_tuples)
print("Dictionary3:", dict3)
# {'x': 100, 'y': 200, 'z': 300}

# Method 4: Using dict() constructor with zip() function
keys = ['p', 'q', 'r']
values = [1000, 2000, 3000]
dict4 = dict(zip(keys, values))
print("Dictionary4:", dict4)
# {'p': 1000, 'q': 2000, 'r': 3000}
```

Adding & deleting items

```python
# Creating a dictionary
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

# Accessing value using key
print("Value associated with key 'name':", my_dict['name'])

# Modifying value using key
my_dict['age'] = 35
print("Updated dictionary after modifying 'age':", my_dict)

# Adding a new key-value pair
my_dict['occupation'] = 'Engineer'
print("Dictionary after adding new key-value pair:", my_dict)

# Removing a key-value pair using del keyword
del my_dict['city']
print("Dictionary after deleting 'city' using del:", my_dict)

# Removing a key-value pair using pop() method
removed_value = my_dict.pop('occupation')
print("Dictionary after removing 'occupation' using pop():", my_dict)
print("Removed value:", removed_value)

# Removing the last inserted key-value pair using popitem() method
removed_item = my_dict.popitem()
print("Dictionary after removing using popitem():", my_dict)
print("Removed item:", removed_item)

# Clearing all key-value pairs using clear() method
my_dict.clear()
print("Dictionary after clearing all items using clear():", my_dict)
```

Dictionary functions

```
# Creating a dictionary
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

# Length of a dictionary using len() function
print("Length of dictionary:", len(my_dict))

# Accessing value using get() method
print("Value associated with key 'name':", my_dict.get('name'))

# Checking if a key exists in the dictionary using in keyword
print("Is 'city' in dictionary?:", 'city' in my_dict)

# Getting list of all keys using keys() method
keys_list = my_dict.keys()
print("List of keys:", keys_list)

# Getting list of all values using values() method
values_list = my_dict.values()
print("List of values:", values_list)

# Getting list of all key-value pairs using items() method
items_list = my_dict.items()
print("List of key-value pairs:", items_list)

# Copying a dictionary using copy() method
copied_dict = my_dict.copy()
print("Copied dictionary:", copied_dict)
```

Dictionary comprehension

```python
# Basic dictionary comprehension
basic_dict = {x: x**2 for x in range(1, 6)}
print("Basic Dictionary Comprehension:", basic_dict)
# Basic Dictionary Comprehension: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# Conditional dictionary comprehension
conditional_dict = {x: x**2 for x in range(1, 11) if x % 2 == 0}
print("Conditional Dictionary:", conditional_dict)
# Conditional Dictionary: {2: 4, 4: 16, 6: 36, 8: 64, 10: 100}
```

Coding problems

Tuple:

1. Python – Adding Tuple to List and vice – versa
2. Python – Closest Pair to Kth index element in Tuple
3. Python – Join Tuples if similar initial element

Set:

1. Python – Find missing and additional values in two lists
2. Python program to find the difference between two lists
3. Python Set difference to find lost element from a duplicated array

Dictionary:

1. Ways to sort list of dictionaries by values in Python – Using lambda function
2. Python | Merging two Dictionaries
3. Python – Convert key-values list to flat dictionary

🚀 Ready to Level Up?

CONNECT FOR MORE INSIGHTS AND FUTURE UPDATES!

Shruti Bharat

@shrutibharat0105