Python Fundamentals

Collections

Collections in Python refer to data structures or containers used to store and manage multiple items or elements. They are essential for organizing and working with data efficiently. Python provides several built-in collection types, each with its own characteristics and use cases. Here are some of the most commonly used collections in Python:

Lists:

Definition: Lists are ordered collections of items, and they can contain elements of different data types.

Characteristics: Lists are mutable, meaning you can change their content by adding, removing, or modifying elements.

Example

```python
numbers = [1, 2, 3, 4, 5]
fruits = ['apple', 'banana', 'cherry']
print(numbers)
print(fruits)
```

Key Operations

Accessing Elements:

You can access elements in a list by their index (position):

```python
first_fruit = fruits[0]  # Access the first element ('apple')
first_fruit
```

Adding Elements:

You can append elements to the end of a list using append():

```python
fruits.append('orange')  # Add 'orange' to the end
fruits
```

Removing Elements:

You can remove elements by their value using remove():

```
fruits.remove('banana')  # Remove 'banana' from the list
fruits
```

Slicing:

You can extract a portion of a list using slicing:

```
sliced_fruits = fruits[1:3]  # Get elements from index 1 to 2
sliced_fruits
```

Tuples:

Definition: Tuples are similar to lists but are immutable, meaning their elements cannot be changed once defined.

Characteristics: Tuples are often used when you need a collection of items that should not be modified.

Example

```
coordinates = (3, 4)
rgb_color = (255, 0, 0)
print(coordinates)
print(rgb_color)
```

Accessing Elements:

Accessing elements in a tuple is done by index, just like lists:

```
x = coordinates[0]  # Access the first element (3)
y = coordinates[1]  # Access the first element (4)

print(x, y)
```

Unpacking Tuples:

You can unpack the elements of a tuple into variables:

```
x, y = coordinates  # x = 3, y = 4

print(x, y)
```

Sets:

Definition: Sets are unordered collections of unique elements. They are defined using curly braces {} or the set() constructor.

Characteristics: Sets are useful for storing unique values and performing set operations like union, intersection, and difference.

Example

```
colors = {'red', 'green', 'blue'}
prime_numbers = {2, 3, 5, 7, 11}
print(colors)
print(prime_numbers)
```

Adding Elements:

You can add elements to a set using the add() method:

```
colors.add('orange')  # Add 'orange' to the set
colors
```

Removing Elements:

You can remove elements from a set using the remove() method:

```
colors.remove("red")
colors
```

Checking Membership:

You can check if an element is in a set using the in operator:

```
is_apple_in_fruits = 'orange' in colors  # True
is_apple_in_fruits
```

Dictionaries:

Definition: Dictionaries are collections of key-value pairs. Each key is unique within a dictionary.

Characteristics: Dictionaries provide efficient key-based access to values and are often used for storing and retrieving data by a specific identifier.

Example

```
person = {'name': 'Alice', 'age': 30, 'city': 'New York'}
print(person)
```

Accessing Values:

You can access values in a dictionary using keys:

```
name = person['name']  # Access the value associated with 'name' ('John')
name
```

Adding Key-Value Pairs:

You can add new key-value pairs to a dictionary:

```
person['job'] = 'Engineer'  # Add 'job': 'Engineer' to the dictionary
person
```

Iterating Over Keys and Values:

You can iterate through a dictionary's keys, values, or key-value pairs:

```
for key in person.keys():
    print("key = ", key)  # Print keys ('name', 'age', 'city', 'job')

for value in person.values():
    print("value = ", value)  # Print values ('John', 30, 'New York', 'Engineer')

for key, value in person.items():
    print(f"{key} = {value}")  # Print key-value pairs
```