Python Collections

In Python, collections are used to store multiple values in a single variable. There are several types of collections available in Python, including lists, tuples, sets, and dictionaries. Each type of collection has its own characteristics and use cases.

Lists

A list is a collection of items that are ordered and changeable. Lists are defined by enclosing the items in square brackets `[]` and separating them with commas. Lists can contain items of different data types, including integers, strings, and other lists.

Characteristics of Lists

- Lists are mutable, which means that you can change, add, or remove items from a list after it has been created.
- Lists are ordered, which means that the items in a list have a specific order and can be accessed by their index.
- Lists can contain duplicate items.
- Lists can contain items of different data types. (e.g., integers, strings, lists, etc. but it is not recommended to mix data types in a list)
- List are accessed by index starting from 0.

Creating Lists

Lists can be created by enclosing the items in square brackets `[]` and separating them with commas.

Example:

```
# Creating a list
fruits = ["apple", "banana", "cherry", "orange"]

# Accessing items in a list
print(fruits[0])  # Output: apple

# Changing an item in a list
fruits[1] = "kiwi"
print(fruits)  # Output: ["apple", "kiwi", "cherry", "orange"]
```

In the above example, we create a list of fruits and access the first item in the list using its index. We then change the second item in the list to `"kiwi"`.

List Methods

Python provides several built-in methods for working with lists. Some common methods include:

- `append()`: Adds an item to the end of the list.
- `insert()`: Adds an item at a specified position in the list.
- `remove()`: Removes the first occurrence of a specified item from the list.
- `pop()`: Removes an item at a specified index from the list and returns it.
- `clear()`: Removes all items from the list.
- `sort()`: Sorts the items in the list.
- `reverse()`: Reverses the order of the items in the list.
- `len()`: Returns the number of items in the list.

Example:

```
# get the length of the list
fruits = ["apple", "banana", "cherry"]
print(len(fruits))  # Output: 4


# List methods
fruits = ["apple", "banana", "cherry"]

# Append an item to the list
fruits.append("orange")
print(fruits)  # Output: ["apple", "banana", "cherry", "orange"]

# Insert an item at a specific position
fruits.insert(1, "kiwi")
print(fruits)  # Output: ["apple", "kiwi", "banana", "cherry",
"orange"]

# Remove an item from the list
fruits.remove("banana")
print(fruits)  # Output: ["apple", "kiwi", "cherry", "orange"]

# Pop an item from the list
popped_fruit = fruits.pop(2)
print(popped_fruit)  # Output: cherry

# Clear the list
fruits.clear()
print(fruits)  # Output: []

# Sort the list
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort()
print(numbers)  # Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

# Reverse the list
numbers.reverse()
print(numbers)  # Output: [9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]
```

In the above example, we demonstrate some common list methods, such as `append()`, `insert()`, `remove()`, `pop()`, `clear()`, `sort()`, and `reverse()`.

To learn more about lists, Check out W3Schools' Python Lists tutorial.

Dictionaries

A dictionary is a collection of key-value pairs that are unordered, changeable, and indexed. Dictionaries are defined by enclosing the key-value pairs in curly braces `{}` and separating them with commas. Each key-value pair is separated by a colon `:`.

Characteristics of Dictionaries

- Dictionaries are mutable, which means that you can change, add, or remove key-value pairs from a dictionary after it has been created.
- Dictionaries are unordered (for older version of python 3.6 and earlier), which means that the items in a dictionary do not have a specific order, but for python 3.7 and later, dictionaries maintain the order of insertion.
- Dictionaries are accessed by key, not by index.
- Dictionaries cannot contain duplicate keys, but they can contain duplicate values.
- Dictionaries can contain items of different data types.
- Dictionaries are indexed by keys.

Creating Dictionaries

Dictionaries can be created by enclosing the key-value pairs in curly braces `{}` and separating them with commas. Each key-value pair is separated by a colon `:`.

NOTE: The keys in a dictionary must be unique, and they must be immutable data types (e.g., strings, numbers, tuples). Strings are the most common data type used as keys in dictionaries.

Example:

```
# Creating a dictionary
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

# Accessing items in a dictionary

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

The above example creates a dictionary with key-value pairs representing information about a person. We then access the value associated with the key `"name"` using the key. `name`, `age`, and `city` are the keys in the dictionary, and `"Alice"`, `30`, and `"New York"` are the corresponding values.

Dictionary Methods

Python provides several built-in methods for working with dictionaries. Some common methods include:

- `get()`: Returns the value associated with a specified key.
- `keys()`: Returns a list of all the keys in the dictionary.
- `values()`: Returns a list of all the values in the dictionary.
- `items()`: Returns a list of key-value pairs in the dictionary.
- `update()`: Updates the dictionary with key-value pairs from another dictionary.
- `pop()`: Removes the key-value pair with the specified key from the dictionary and returns the value.

More read here