Python Programming Basics: Series 2

Lists, Dictionaries, Sets, Loops, and Functions

Table of contents

1 Introduction

Welcome to Series 2 of our Python programming journey! In this series, we'll delve into fundamental concepts that are essential for any budding programmer:

- **Lists**
- **Dictionaries**
- **Sets**
- **For Loops**
- **While Loops**
- **Functions** (including type annotations and default values)

Each section will include explanations and examples to help you understand the concepts, followed by exercises to practice what you've learned.

Let's get started!

---

2 Lists

A **list** is a collection of items that are ordered and changeable. Lists allow duplicate elements and are one of the most commonly used data structures in Python.

2.1 List: Methods

- `list.append(elem)`: Adds an element to the end of the list.
- `list.pop(index)`: Removes the element at the specified index.
  - `list.pop()` removes the last element if no `index` is provided.
- `list.remove(elem)`: Removes the first occurrence of the specified element from the list.
- `list.clear()`: Removes all elements from the list.
- `list[index]`: Accesses an element at the specified index.
- `len(list)`: Returns the number of elements in the list.
- `elem in list`: Checks if an element is present in the list.
- `list1 + list2`: Concatenates two lists.

2.2 Example

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

# Accessing elements
print(fruits[0])  # Output: apple

# Modifying elements
fruits[1] = "blueberry"

# Adding elements
fruits.append("orange")

# Removing elements
fruits.remove("cherry")
```

```
# Printing the list
print(fruits)  # Output: ['apple', 'blueberry', 'orange']
```

2.3 Exercise 1: Creating and Modifying Lists

Question

Create a list named `numbers` containing the integers 1 through 5. Then, perform the following operations:

- Append the number 6 to the list.
- Change the second element to 10.
- Remove the number 3 from the list.
- Print the final list.

> *Hints:*
>
> - Use `append()` to add an element.
> - Access elements using their index (remember that indexing starts at 0).
> - Use `remove()` to delete an element.

```
# Create the initial list
numbers = ?

# Append 6
?

# Change the second element to 10
?

# Remove the number 3
?

# Print the final list
?
```

---

3 Dictionaries

A **dictionary** is a collection of key-value pairs that is unordered, changeable, and does not allow duplicates. Dictionaries are used to store data values like a map.

3.1 Dictionary: Methods

- `dictionary[key]`: Accesses the value associated with the key.
  - `dictionary[key] = value`: Modifies the value associated with the key or adds a new key-value pair.
- `dictionary.get(key)`: Returns the value associated with the key. Returns `None` if the key is not found.
- `key in dictionary`: Returns `True` if the `key` is present in the dictionary.
- `len(dictionary)`: Returns the number of key-value pairs in the dictionary.
- `dictionary.items()`: Returns a list of key-value pairs in the dictionary.
  - `dictionary.keys()`: Returns a list of all keys in the dictionary.
  - `dictionary.values()`: Returns a list of all values in the dictionary.
- `dictionary.pop(key)`: Removes the key-value pair with the specified key.
- `del dictionary[key]`: Removes the key-value pair with the specified key.
- `dictionary.clear()`: Removes all key-value pairs from the dictionary.
- `dictionary.copy()`: Returns a copy of the dictionary.

3.2 Example

```
# Creating a dictionary
student = {
    "name": "Alice",
    "age": 14,
    "grade": "9th"
}

# Accessing values
print(student["name"])  # Output: Alice

# Modifying values
student["age"] = 15

# Adding a new key-value pair
student["school"] = "High School"

# Removing a key-value pair
```

```
del student["grade"]

# Getting all keys
print(student.keys())  # Output: dict_keys(['name', 'age', 'school'])

# Printing the dictionary
print(student)
# Output: {'name': 'Alice', 'age': 15, 'school': 'High School'}
```

3.3 Exercise 2: Working with Dictionaries

Question

Create a dictionary named `car` with the following key-value pairs:

- `"brand": "Toyota"`
- `"model": "Corolla"`
- `"year": 2020`

Then, perform the following operations:

- Change the `"year"` to `2021`.
- Add a new key `"color"` with the value `"red"`.
- Remove the `"model"` key from the dictionary.
- Print the final dictionary.

> *Hints:*
>
> - Access and modify values using `dictionary[key]`.
> - Use `del` to remove a key-value pair.

```
# Create the dictionary
car = ?

# Change the "year" to 2021
?

# Add a new key "color" with value "red"
?

# Remove the "model" key
?
```

```
# Print the final dictionary
?
```

---

4 Sets

A **set** is a collection which is unordered, unchangeable*, and unindexed. Sets do not allow duplicate elements.

*Note: You cannot change the items after the set has been created, but you can add new items.

4.1 Sets: Methods

- `set.add(elem)`: Adds an element to the set.
- `set.remove(elem)`: Removes the specified element from the set. Raises `KeyError` if `elem` is not contained in the set.
- `set.discard(elem)`: Removes the specified element from the set (if present).
- `set.pop()`: Removes an element from the set.
- `set.clear()`: Removes all elements from the set.
- `len(set)`: Returns the number of elements in the set.
- `elem in set`: Checks if an element is present in the set.
- `set.union(other_set)`: Returns a new set with all elements from both sets.
- `set.intersection(other_set)`: Returns a new set with elements common to both sets.

4.2 Example

```
# Creating a set
colors = {"red", "green", "blue"}

# Adding an element
colors.add("yellow")

# Removing an element
colors.remove("green")

# Printing the set
print(colors)  # Output: {'red', 'blue', 'yellow'}
```

4.3 Exercise 3: Using Sets

Question

Create a set named `animals` containing `"cat"`, `"dog"`, and `"rabbit"`. Then, perform the following operations:

- Add `"bird"` to the set.
- Remove `"dog"` from the set.
- Try adding `"cat"` to the set again.
- Print the final set.

> *Hints:*
>
> - Use `add()` to add an element.
> - Use `remove()` to remove an element.
> - Remember that sets do not allow duplicates.

```
# Create the set
animals = ?

# Add "bird" to the set
?

# Remove "dog" from the set
?

# Try adding "cat" again
?

# Print the final set
?
```

---

5 For Loops

A **for loop** is used for iterating over a sequence (such as a list, tuple, dictionary, set, or string).

5.1 Example

```
# Looping through a list
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num)

# Output:
# 1
# 2
# 3
# 4
# 5
```

5.2 Exercise 4: Using For Loops

Question

Given the list `numbers = [2, 4, 6, 8, 10]`, write a for loop that:

- Iterates through each number in the list.
- Prints the square of each number.

> *Hint:*
>
> - Use the exponentiation operator `**` to calculate the square.

```
numbers = [2, 4, 6, 8, 10]

# Write your for loop here
```

---

6 While Loops

A **while loop** repeatedly executes a block of code as long as a given condition is true.

6.1 Example

```
# Using a while loop
count = 1
while count <= 5:
    print(count)
    count += 1

# Output:
# 1
# 2
# 3
# 4
# 5
```

6.2 Exercise 5: Using While Loops

Question

Write a while loop that:

- Starts with a variable `n = 10`.
- Continues to loop while `n` is greater than 0.
- Prints the value of `n`.
- Decreases the value of `n` by 2 each time.

> *Hint:*
>
> - Use `n -= 2` to decrease `n` by 2.

```
# Initialize n
n = 10

# Write your while loop here
```

---

7 Functions

A **function** is a block of code that runs when it is called. Functions can accept parameters and return results.

7.1 Example

```
# Defining a function
def greet(name):
    return f"Hello, {name}!"

# Calling the function
message = greet("Alice")
print(message)  # Output: Hello, Alice!
```

7.1.1 Type Annotations and Default Values

You can add **type annotations** to function parameters and return values for clarity. **Default values** allow parameters to have a default if no argument is provided.

```
def add(a: int, b: int = 5) -> int:
    return a + b

result = add(10)
print(result)  # Output: 15
```

Note: Type annotations are optional in Python but can help improve code readability and maintainability.
Syntax conventions for type annotations:

- `def function_name(parameter):`
- `def function_name(parameter=default_value):`
- `def function_name(parameter: type) -> return_type:`
- `def function_name(parameter: type = default_value) -> return_type:`

Setting default values

- **without type annotations** should not contain spaces around the `=` sign.
- **with type annotations** should have spaces around the `:` and `=` signs.

7.1.2 Supported Type Annotations

- `int`: Integer
  - `1`, `-2`, `1000`, etc.
- `float`: Floating-point number
  - `1.0`, `-2.5`, `3.14`, etc.

- `str`: String
  - `"Hello"`, `"Python"`, etc.
- `bool`: Boolean
  - `True`, `False`
- `list`: List
  - `[1, 2, 3]`, `["apple", "banana"]`, etc.
- `dict`: Dictionary
  - `{"name": "Alice", "age": 25}`, etc.
- `set`: Set
  - `{"apple", "banana"}`, etc.
- `tuple`: Tuple
  - `(1, 2, 3)`, etc.
- `None`: NoneType
  - `None`
- Custom classes and more
  - `class MyClass: ...`

7.2 Exercise 6: Creating Functions with Type Annotations and Default Values

Question

Create a function named `calculate_area` that:

- Accepts two parameters: `length` and `width`, both of type `float`.
- Has a default value of `width = 1.0`.
- Returns the area of a rectangle (length × width).
- Includes type annotations for the parameters and the return type.

Then:

- Call the function with `length = 5.0` and the default `width`.
- Call the function with `length = 5.0` and `width = 2.0`.
- Print the results.

> *Hints:*
>
> - Use `def function_name(parameters) -> return_type:`
> - Default values are set using `parameter=default_value`.

```
# Define your function here

# Call the function with length = 5.0 (default width)
# area1 = ?

# Call the function with length = 5.0 and width = 2.0
# area2 = ?

# Print the results
?
```

---

8 Conclusion

In this series, we've explored fundamental Python concepts:

- **Lists**: Ordered collections that can be modified.
- **Dictionaries**: Collections of key-value pairs.
- **Sets**: Unordered collections with no duplicate elements.
- **For Loops**: Iterate over sequences.
- **While Loops**: Repeat code while a condition is true.
- **Functions**: Reusable blocks of code with parameters, type annotations, and default values.

Understanding these basics is essential as you continue your programming journey. Keep practicing, and don't hesitate to experiment with your own examples!

---

9 Additional Resources

- **Python Official Documentation**: https://docs.python.org/3/
- **W3Schools Python Tutorial**: https://www.w3schools.com/python/
- **Python for Beginners**: https://www.python.org/about/gettingstarted/

---

10 Acknowledgments

We hope this series has helped you understand the fundamental concepts of Python programming. Keep coding, and have fun exploring the world of programming!