#90DaysOfDevOps Challenge - Day 14 - Python Data Types and Data Structures for DevOps

Welcome to Day 14 of the #90DaysOfDevOps challenge. Today, we will explore Python Data Types and Data Structures, which are essential concepts for DevOps engineers. Python offers a rich set of data types and structures that enable efficient data manipulation and organization. Understanding these concepts will enable you to write cleaner and more efficient code, making you a more effective DevOps practitioner. Let's dive in.

Data Types

In summary, Python offers a variety of data types that allow you to work with different kinds of information. The main data types covered are:

- **Numeric Data Types:** Integers and floating-point numbers for numerical computations.
- **String Data Type:** Used to handle and manipulate textual data.
- **List Data Type:** An ordered collection that can hold elements of different types and supports various operations like adding, removing, and accessing items.
- **Tuple Data Type:** Similar to lists, but immutable, meaning they cannot be modified once created.
- **Dictionary Data Type:** Key-value pairs that allow you to store and retrieve values based on unique keys.
- **Boolean Data Type:** Represents either True or False, used in logical operations and conditional statements.
- **Other Data Types:** Python also provides additional data types like sets and byte arrays, each with their own specific use cases and functionalities.

Review the "#90DaysOfDevOps Challenge - Day 13 - The Basics of Python" article for further details.

Understanding the Difference Between Data Types and Data Structures in Python

In Python programming, it's essential to grasp the concepts of **data types** and **data structures**. While they both deal with organizing and manipulating data, they serve different purposes. In this article, we will explore the difference between data types and data structures in Python, providing examples to help you understand their distinctions and applications.

Data Types: Building Blocks of Data

Data types in Python represent the different kinds of values that can be stored and manipulated in a program. They define the nature of the data and the operations that can be performed on it. Python offers several built-in data types, including **integers**, **floating-point numbers**, **strings**, **booleans**, and **more**.

For instance, the integer data type represents whole numbers, such as 5 or -10. We can perform arithmetic operations like addition, subtraction, and multiplication on integers, providing the foundation for numerical calculations and logic within our programs.

```
x = 5
y = 3
z = x + y
print(z)  # Output: 8
```

In this example, `x` and `y` are variables of the **integer** data type. We use the `+` operator to perform addition on the **integer** values stored in `x` and `y`, and the result is assigned to the variable `z`.

Data Structures: Organizing and Managing Data

While data types deal with individual values, **data structures** focus on organizing and managing **collections of data**. They provide mechanisms to store and manipulate multiple elements, allowing for more complex data representations. Python offers various built-in data structures, such as **lists**, **tuples**, **dictionaries**, **sets**, and **more**.

Let's consider the **list** data structure, which is an **ordered collection of item**s. Each item in a list can be of any data type, including other lists. Lists are **mutable**, meaning their elements can be modified after creation. They offer operations and methods to add, remove, and access elements.

```
fruits = ['apple', 'banana', 'orange']
print(fruits)  # Output: ['apple', 'banana', 'orange']

fruits.append('grape')
print(fruits)  # Output: ['apple', 'banana', 'orange', 'grape']

print(fruits[0])  # Output: 'apple'
```

In this example, `fruits` is a **list** that stores multiple **string values**. We use the `append()` method to add a new element, 'grape', to the end of the list. We can access individual elements in the list using square brackets and the index of the element.

Task 1 - Give the Difference between List, Tuple and Set. Do Hands-on and provide screenshots as per your understanding

Lists

- Ordered collection of elements enclosed in square brackets ([]).
- Mutable, meaning elements can be added, removed, or modified after creation.
- Allows duplicate values.
- Elements can be accessed using indexing.

- Lists are commonly used when you need to store and manipulate a collection of items in a specific order.

```python
# Creating a list
my_list = [1, 2, 3, 3, 4, 5]

# Modifying elements
my_list[0] = 10

# Adding elements
my_list.append(6)

# Removing elements
my_list.remove(3)

print("List:", my_list)
```

```
esteban@Estebans-MacBook-Pro ~/Library/CloudStorage/GoogleDrive-morenoramirezesteban@gmail.com/My Drive/GitHub/90DaysOfDevOps master
/GitHub/90DaysOfDevOps/2023/day14/python/task 1/lists.py"
List: [10, 2, 3, 4, 5, 6]
```

Tuples

- Ordered collection of elements enclosed in parentheses ().
- Immutable, meaning elements cannot be modified after creation.
- Allows duplicate values.
- Elements can be accessed using indexing.
- Tuples are suitable when you want to store a collection of values that should not be changed.

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

# Accessing elements
print("Tuple[0]:", my_tuple[0])

# Trying to modify elements (throws an error)
my_tuple[0] = 10
```

```
esteban@Estebans-MacBook-Pro ~/Library/CloudStorage/GoogleDrive-morenoramirezesteban@gmail.com/My Drive/GitHub/90DaysOfDevOps master /usr/local/bin/python3 "/Users/este
/GitHub/90DaysOfDevOps/2023/day14/python/task 1/tuple.py"
Tuple[0]: 1
Traceback (most recent call last):
  File "/Users/esteban/Library/CloudStorage/GoogleDrive-morenoramirezesteban@gmail.com/My Drive/GitHub/90DaysOfDevOps/2023/day14/python/task 1/tuple.py", line 8, in <module>
    my_tuple[0] = 10
    ~~~~~~~~^^^
TypeError: 'tuple' object does not support item assignment
```

Sets

- An unordered collection of unique elements enclosed in curly braces ({}).

- Mutable, meaning elements can be added or removed after creation.
- Does not allow duplicate values.
- Elements cannot be accessed using indexing.
- Sets are useful when you want to store a collection of unique elements and perform operations like union, intersection, and difference.

```
# Creating a set
my_set = {1, 2, 3, 3, 4, 5}

# Adding elements
my_set.add(6)

# Removing elements
my_set.remove(3)

print("Set:", my_set)
```

```
esteban@Estebans-MacBook-Pro ~/Library/CloudStorage/GoogleDrive-mor
/GitHub/90DaysOfDevOps/2023/day14/python/task 1/sets.py"
Set: {1, 2, 4, 5, 6}
```

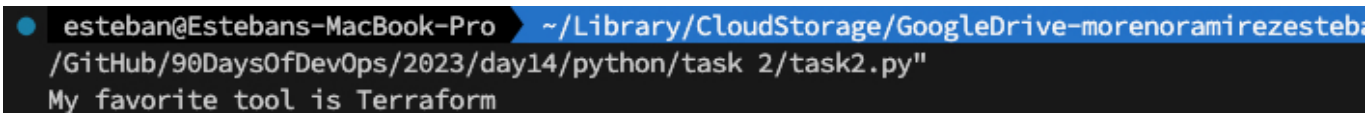
Task 2 - Create the below Dictionary and use Dictionary methods to print your favourite tool just by using the keys of the Dictionary

```
fav_tools = {
  1:"Linux",
  2:"Git",
  3:"Docker",
  4:"Kubernetes",
  5:"Terraform",
  6:"Ansible",
  7:"Chef"
}
```

Python code:

```
fav_tools = {
  1:"Linux",
  2:"Git",
  3:"Docker",
  4:"Kubernetes",
  5:"Terraform",
  6:"Ansible",
  7:"Chef"
}
```

```
# Accessing a specific tool using its key
favorite_tool = fav_tools[5]
print("My favorite tool is", favorite_tool)
```



Task 3 - Create a List of cloud service providers

```
cloud_providers = ["AWS","GCP","Azure"]
```

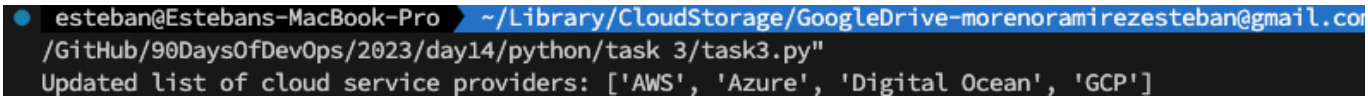
Write a program to add `Digital Ocean` to the list of `cloud_providers` and sort the list in alphabetical order.

```
cloud_providers = ["AWS","GCP","Azure"]

# Adding "Digital Ocean" to the list
cloud_providers.append("Digital Ocean")

# Sorting the list
cloud_providers.sort()

# Printing the list
print("Updated list of cloud service providers:", cloud_providers)
```



By using the `append()` function, we add `"Digital Ocean"` to the `cloud_providers` list. Then, we use the `sort()` function to sort the list in alphabetical order. Finally, we print the updated list, which now includes "Digital Ocean."

In this article, we explored the differences between lists, tuples, and sets in Python. We learned about their characteristics, use cases, and demonstrated hands-on examples to solidify our understanding.

Stay tuned for Day 15 of the #90DaysOfDevOps challenge, where we will explore Python libraries for DevOps and discover how they can enhance your automation and infrastructure management tasks.