Week 2: Python Data Types and Data Structures

Introduction

In Week 2 of the Python programming course, we covered the essential concepts of Python data types and data structures. These concepts are the building blocks of Python programming, as they allow developers to store, manage, and manipulate data efficiently. This report delves into these core ideas, illustrating their practical uses and providing examples to help learners fully grasp how they are applied in real-world scenarios.

1 Python Primitive Data Types

Python has several built-in data types that allow users to work with various kinds of data. These types form the foundation of any Python program, making it essential to understand their characteristics.

1.1 Integers (`int`)

Integers represent whole numbers (positive or negative). They are frequently used in counting and mathematical calculations.

```
a = 10
b = -5
c = a + b  # Output: 5
```

1.2 Floating-point Numbers (`float`)

Floats are numbers with a decimal point. They are commonly used for calculations that require precision, such as scientific measurements.

```
pi = 3.14159
radius = 5.0
area = pi * (radius ** 2)  # Output: 78.53975
```

1.3 Strings (`str`)

Strings are sequences of characters. They are useful for storing and manipulating text data, such as names, addresses, or any textual information.

```
name = "Alice"
greeting = f"Hello, {name}"  # Output: "Hello, Alice"
```

1.4 Booleans (`bool`)

Booleans represent truth values: `True` or `False`. They are widely used in conditional statements and logic-based decisions.

```
is_valid = True
if is_valid:
    print("The data is valid.")  # Output: "The data is valid."
```

1.5 None (NoneType)

None represents the absence of a value. It is often used as a placeholder when no other value is available.

```
result = None
if result is None:
    print("No result available.")
```

2 Strings and Their Operations

Strings are one of the most versatile data types in Python. They can be indexed, sliced, concatenated, and modified using various built-in methods.

2.1 String Indexing and Slicing

Python allows you to access individual characters in a string using indices. Slicing allows for extracting a portion of the string.

```
text = "Python Programming"
first_letter = text[0]  # Output: 'P'
substring = text[7:18]  # Output: 'Programming'
```

2.2 String Methods

Python provides a rich set of methods to manipulate strings. These include changing the case, replacing parts of a string, and splitting it into a list.

```
text = "Hello World"
upper_text = text.upper()  # Output: "HELLO WORLD"
replaced_text = text.replace("World", "Python")  # Output: "Hello Python"
```

3 Lists – Python's Dynamic Arrays

Lists are ordered, mutable collections that can store elements of different data types. They provide flexibility in handling large sets of data.

3.1 Creating and Accessing Lists

Lists can be created with square brackets, and elements are accessed using zero-based indices.

```
fruits = ["apple", "banana", "cherry"]
first_fruit = fruits[0]  # Output: 'apple'
```

3.2 List Operations

Lists can be modified in various ways:

- **Adding Elements**: Use `.append()` to add elements at the end, or `.insert()` to add at a specific position.
- **Removing Elements**: Use `.remove()` to delete by value, or `.pop()` to remove by index.

```
fruits.append("orange")  # Adds 'orange' to the list
fruits.remove("banana")  # Removes 'banana' from the list
```

3.3 List Comprehension

List comprehension offers a concise way to create lists based on existing lists or ranges.

```
squares = [x**2 for x in range(5)]  # Output: [0, 1, 4, 9, 16]
```

4 Tuples – Immutable Sequences

Tuples are similar to lists but are immutable. This means their values cannot be changed once they are created. Tuples are faster and used when immutability is preferred.

4.1 Creating and Accessing Tuples

Tuples are defined by parentheses, and elements can be accessed like lists.

```
coordinates = (10, 20)
x = coordinates[0]  # Output: 10
```

4.2 Tuple Immutability

Once created, tuples cannot be changed. Any operation that tries to modify the tuple will raise an error.

```
coordinates = (10, 20)
# coordinates[0] = 15  # This will raise an error
```

5 Dictionaries – Key-Value Pairs

Dictionaries are unordered collections of key-value pairs. They provide efficient ways to store and retrieve data based on a unique key.

5.1 Creating and Accessing Dictionaries

Dictionaries are created using curly braces {}. Values are accessed by their keys.

```
student = {"name": "Alice", "age": 21, "grade": "A"}
name = student["name"]  # Output: "Alice"
```

5.2 Dictionary Operations

- **Adding/Modifying Entries**: Add or modify values using keys.
- **Removing Entries**: Use `.pop()` or `del` to remove key-value pairs.

```
student["age"] = 22  # Modifies the age value
del student["grade"]  # Removes the grade entry
```

6 Sets – Unordered Collections of Unique Elements

Sets are collections that store unique elements, making them useful for tasks that involve eliminating duplicates or performing set operations.

6.1 Creating and Accessing Sets

Sets are defined with curly braces {} and cannot contain duplicate values.

```
unique_numbers = {1, 2, 3, 4, 4, 5}  # Output: {1, 2, 3, 4, 5}
```

6.2 Set Operations

Python sets support operations like union, intersection, and difference, which can be used to compare and combine sets.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union = set1 | set2  # Output: {1, 2, 3, 4, 5}
intersection = set1 & set2  # Output: {3}
```

7 Conclusion

Understanding Python's data types and data structures is vital for writing efficient and effective programs. Each structure serves different purposes:

- **Lists** provide dynamic, mutable sequences for general-purpose storage.
- **Tuples** ensure data integrity by preventing changes.
- **Dictionaries** allow quick lookups using unique keys.
- **Sets** handle unique data and provide efficient mathematical set operations.

By mastering these types and structures, students will be able to handle data in a wide range of applications, from simple scripts to more complex algorithms. This knowledge will form the basis for future topics such as object-oriented programming, file handling, and advanced algorithms.

References

- Python Documentation: `https://docs.python.org/3/library/stdtypes.html`
- *Automate the Boring Stuff with Python* by Al Sweigart

Exercises: Python Data Types and Data Structures

1. Integer Operations
Problem: Write a Python program to perform the following tasks:

- Assign two integers to variables.
- Perform addition, subtraction, multiplication, and division between them.
- Print the results.

Solution:

```python
# Integer operations
a = 10
b = 5

# Perform operations
addition = a + b
subtraction = a - b
multiplication = a * b
division = a / b

# Print the results
print("Addition:", addition)
print("Subtraction:", subtraction)
print("Multiplication:", multiplication)
print("Division:", division)
```

2. Floating-point Precision
Problem: Given two floating-point numbers, calculate the area of a circle with one as the radius and print the result rounded to two decimal places.
Solution:

```python
import math

# Floating-point numbers
radius = 7.5
area = math.pi * radius ** 2

# Print area rounded to two decimal places
print("Area of circle:", round(area, 2))
```

3. String Concatenation
Problem: Write a program that asks the user to input their first name and last name, then prints a full sentence greeting them using string concatenation.
Solution:

```python
# Input first and last name
first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")

# String concatenation
greeting = "Hello " + first_name + " " + last_name + ", welcome!"

# Print greeting
print(greeting)
```

4. String Slicing
Problem: Given the string ‘sentence = ”Learning Python is fun!”‘, write a program that:

- Extracts and prints the word “Python”.
- Extracts and prints the word “fun”.

Solution:

```
sentence = "Learning Python is fun!"

# String slicing
word_python = sentence[9:15]
word_fun = sentence[-4:-1]

# Print results
print("Extracted word:", word_python)
print("Extracted word:", word_fun)
```

5. String Methods
Problem: Given the string ‘text = ”Data Science is amazing!”‘, write a program that:

- Converts all the characters to uppercase.
- Replaces the word ”amazing” with ”challenging”.

Solution:

```
text = "Data Science is amazing!"

# String methods
uppercase_text = text.upper()
replaced_text = text.replace("amazing", "challenging")

# Print results
print("Uppercase text:", uppercase_text)
print("Replaced text:", replaced_text)
```

6. Boolean Logic
Problem: Write a program that checks if a number entered by the user is both greater than 10 and less than 100. Print ”Valid” if it meets the condition, otherwise print ”Invalid”.
Solution:

```
# Input number
number = int(input("Enter a number: "))

# Check condition
if number > 10 and number < 100:
    print("Valid")
else:
    print("Invalid")
```

7. List Creation and Access
Problem: Create a list of 5 fruits. Write a program that:

- Prints the first and last fruit.
- Adds a new fruit to the list.
- Removes the second fruit from the list.

Solution:

```
# List of fruits
fruits = ["apple", "banana", "cherry", "date", "elderberry"]

# Access first and last fruit
print("First fruit:", fruits[0])
print("Last fruit:", fruits[-1])

# Add a new fruit
fruits.append("fig")
print("Updated list:", fruits)

```

```python
# Remove the second fruit
fruits.pop(1)
print("List after removal:", fruits)
```

8. List Operations
Problem: Create a list of 5 numbers. Write a program to:

- Find and print the sum of all numbers.
- Multiply each number by 2 and print the new list.

Solution:

```python
# List of numbers
numbers = [2, 4, 6, 8, 10]

# Find sum of all numbers
total_sum = sum(numbers)
print("Sum of numbers:", total_sum)

# Multiply each number by 2
new_numbers = [num * 2 for num in numbers]
print("Numbers multiplied by 2:", new_numbers)
```

9. Tuple Creation and Access
Problem: Create a tuple with 4 elements: a string, an integer, a float, and a boolean. Write a program to:

- Print the third element of the tuple.
- Print whether the tuple contains the boolean value `True`.

Solution:

```python
# Tuple with different data types
my_tuple = ("Python", 42, 3.14, True)

# Access the third element
print("Third element:", my_tuple[2])

# Check if the tuple contains True
contains_true = True in my_tuple
print("Contains True:", contains_true)
```

10. Set Operations
Problem: Create two sets: ‘set1 = 1, 2, 3, 4, 5‘ and ‘set2 = 4, 5, 6, 7, 8‘. Write a program to:

- Find and print the intersection of the two sets.
- Find and print the union of the two sets.

Solution:

```python
# Sets
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Intersection
intersection = set1.intersection(set2)
print("Intersection:", intersection)

# Union
union = set1.union(set2)
print("Union:", union)
```

11. Dictionary Creation and Access
Problem: Create a dictionary that stores a person’s name, age, and city. Write a program to:

- Print the person's name.
- Add a new key-value pair for their profession.

Solution:

```python
# Dictionary of person information
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Print name
print("Name:", person["name"])

# Add profession
person["profession"] = "Engineer"
print("Updated dictionary:", person)
```

12. Dictionary Operations
Problem: Write a program that creates a dictionary of 3 students' names as keys and their grades as values. Then, do the following:

- Print all the students' names.
- Remove one student from the dictionary.

Solution:

```python
# Dictionary of students and their grades
students = {
    "Alice": 85,
    "Bob": 90,
    "Charlie": 78
}

# Print student names
print("Student names:", students.keys())

# Remove one student
students.pop("Charlie")
print("Updated dictionary:", students)
```

13. List Slicing
Problem: Write a program that creates a list of 10 numbers and prints:

- The first 3 elements.
- The last 3 elements.
- The elements from index 2 to index 6 (inclusive).

Solution:

```python
# List of 10 numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Print first 3 elements
print("First 3 elements:", numbers[:3])

# Print last 3 elements
print("Last 3 elements:", numbers[-3:])

# Print elements from index 2 to 6
print("Elements from index 2 to 6:", numbers[2:7])
```

14. List Sorting
Problem: Write a program that creates a list of 5 random numbers. Then:

- Sort the list in ascending order.
- Sort the list in descending order.

Solution:

```python
# List of random numbers
numbers = [4, 1, 7, 3, 9]

# Sort in ascending order
numbers.sort()
print("Ascending:", numbers)

# Sort in descending order
numbers.sort(reverse=True)
print("Descending:", numbers)
```

15. Set Uniqueness
Problem: Write a program that takes a list of numbers with duplicates and prints only the unique values using a set.
Solution:

```python
# List with duplicates
numbers = [1, 2, 2, 3, 4, 4, 5]

# Convert list to set to get unique values
unique_numbers = set(numbers)
print("Unique numbers:", unique_numbers)
```

16. Set Difference
Problem: Given two sets 'A = 1, 2, 3, 4, 5' and 'B = 4, 5, 6, 7, 8', write a program to:

- Print the difference of set A from set B.
- Print the difference of set B from set A.

Solution:

```python
# Sets
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# Difference A - B
print("A - B:", A.difference(B))

# Difference B - A
print("B - A:", B.difference(A))
```

17. String Formatting
Problem: Write a program that uses string formatting to print a sentence like "John is 30 years old and lives in New York." based on variables for the name, age, and city.
Solution:

```python
# Variables
name = "John"
age = 30
city = "New York"

# String formatting
sentence = "{} is {} years old and lives in {}.".format(name, age, city)
print(sentence)
```

18. List Comprehension
Problem: Create a list of the first 10 square numbers using a list comprehension.
Solution:

```python
# List comprehension for square numbers
squares = [x**2 for x in range(1, 11)]
print("Squares:", squares)
```

19. Dictionary from Two Lists
Problem: Given two lists, one of names and one of scores, create a dictionary where each name corresponds to its score.
Solution:

```python
# Two lists
names = ["Alice", "Bob", "Charlie"]
scores = [85, 90, 78]

# Create dictionary from two lists
student_scores = dict(zip(names, scores))
print("Student scores:", student_scores)
```

20. Nested List Access
Problem: Given a nested list ‘matrix = [[1, 2], [3, 4], [5, 6]]‘, write a program to:

- Print the second row of the matrix.
- Print the element in the third row and second column.

Solution:

```python
# Nested list (matrix)
matrix = [[1, 2], [3, 4], [5, 6]]

# Print second row
print("Second row:", matrix[1])

# Print element at third row, second column
print("Element at [2][1]:", matrix[2][1])
```

Problems to be Solved

1. **List Manipulation**: Create a list of your favorite five movies. Write a program to:
   - Print the second movie in the list.
   - Add a new movie to the end of the list.
   - Remove the first movie from the list and print the updated list.
2. **Dictionary Operations**: Write a program to create a dictionary that stores information about a book (title, author, year, and genre). Then:
   - Print the title of the book.
   - Update the year to the current year and print the updated dictionary.
3. **Tuple Unpacking**: Create a tuple with three different data types (e.g., string, integer, float). Write a program that:
   - Unpacks the tuple into three variables and prints each variable separately.
4. **Set Operations**: Create two sets: `set_A = {1, 2, 3, 4, 5}` and `set_B = {4, 5, 6, 7, 8}`. Write a program to:
   - Find and print the symmetric difference of the two sets.
5. **String Formatting**: Write a program that takes the user's first name and last name as input and prints a message in the format: "Hello, [First Name] [Last Name]! Welcome to Python programming."
6. **List Comprehension**: Write a program that generates a list of squares of even numbers from 1 to 20 using list comprehension. Print the resulting list.
7. **Nested Lists**: Create a nested list (matrix) representing a 2x3 grid of numbers. Write a program to:
   - Print the element at the second row, first column.
   - Print the entire second row of the matrix.
8. **Boolean Expressions**: Write a program that prompts the user to enter an age. Print "Eligible" if the age is 18 or older, otherwise print "Not Eligible."
9. **Data Structure: Stacks**: Implement a stack using a list in Python. Write a program that:
   - Allows the user to perform the following operations: push an element onto the stack, pop an element from the stack, and display the current stack.
   - Include error handling for cases where a user tries to pop from an empty stack.
10. **Random Number Generation**: Write a program that generates 10 random integers between 1 and 100 and stores them in a list. Then:
    - Print the list, find the maximum and minimum values, and calculate the average of the numbers.