1. Basic Data Types:

In Python, data types are fundamental classifications of data that determine the nature of operations that can be performed on them. Here are the basic data types in Python:

- **Integer (int):** Represents whole numbers without any decimal points. For example, 5, -10, 1000.
- **Float (float):** Represents real numbers with a decimal point. For example, 3.14, -0.5, 2.0.
- **String (str):** Represents a sequence of characters enclosed within single quotes (' ') or double quotes (" "). For example, 'hello', "Python", "123".
- **Boolean (bool):** Represents the truth values True or False. Used for logical operations and comparisons.

Example:

```
# Assigning values to variables of different data types
integer_var = 10
float_var = 3.14
string_var = 'hello'
boolean_var = True

# Printing the variables
print("Integer Variable:", integer_var)
print("Float Variable:", float_var)
print("String Variable:", string_var)
print("Boolean Variable:", boolean_var)
```

2. Type Casting:

Type casting refers to the conversion of one data type to another in Python. This can be useful when you want to perform operations or comparisons involving different data types.

Example:

```
# Type casting from int to float
integer_var = 10
float_var = float(integer_var)
print("Float Variable after Type Casting:", float_var)

# Type casting from float to int
float_var = 3.14
integer_var = int(float_var)
print("Integer Variable after Type Casting:", integer_var)

# Type casting from int to string
integer_var = 10
string_var = str(integer_var)
print("String Variable after Type Casting:", string_var)
```

3. Control Flow and Conditionals:

Control flow refers to the order in which statements are executed in a program. Conditionals allow us to execute certain code blocks based on specified conditions.

3.1. Math and Logical Operators:

Math operators are used for mathematical operations, while logical operators are used to combine conditional statements.

Math Operators:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Modulus (Remainder): %
- Exponentiation: **

Logical Operators:

- AND: and
- OR: or
- NOT: not

Example:

```python
# Math operators example
x = 10
y = 3
print("Addition:", x + y)
print("Subtraction:", x - y)
print("Multiplication:", x * y)
print("Division:", x / y)
print("Modulus:", x % y)
print("Exponentiation:", x ** y)

# Logical operators example
a = True
b = False
print("AND:", a and b)
print("OR:", a or b)
print("NOT:", not a)
```

Practice Exercise: Write a Python program that calculates the area of a rectangle. Ask the user to input the length and width of the rectangle, then calculate and print the area.

3.2. Control Flow: If/else, else if:

In Python, the if/else statement is used to execute a block of code if a condition is true. The else if statement is represented by elif in Python.

Syntax:

```python
if condition1:
    # block of code to execute if condition1 is True
elif condition2:
    # block of code to execute if condition2 is True
else:
    # block of code to execute if all conditions are False
```

Example:

```python
# If/else statement example
x = 10
if x > 0:
    print("x is positive")
else:
    print("x is non-positive")

# If/elif/else statement example
y = -5
if y > 0:
    print("y is positive")
elif y == 0:
    print("y is zero")
else:
    print("y is negative")
```

Practice Exercise: Write a Python program that takes an integer as input from the user and prints whether it is positive, negative, or zero.

3.3. Switch Statement:

Python doesn't have a built-in switch statement like some other programming languages. Instead, if/elif/else statements are used for similar functionality.

Practice Exercise: Write a Python program that takes a grade (A, B, C, D, or F) as input from the user and prints a corresponding message (e.g., "Excellent", "Good", "Average", "Pass", "Fail").

3.4. Looping Constructs:

Python supports various looping constructs such as for loops and while loops.

For Loop Syntax:

```python
for variable in sequence:
    # block of code to execute
```

While Loop Syntax:

```
while condition:
    # block of code to execute
```

Example:

```
# For loop example
for i in range(5):
    print(i)

# While loop example
x = 0
while x < 5:
    print(x)
    x += 1
```

Practice Exercise: Write a Python program that prints the first 10 natural numbers using a for loop and then using a while loop.

3.5. Nested Loops:

Python allows nesting of loops, meaning you can have one loop inside another.

Example:

```
# Nested loops example
for i in range(3):
    for j in range(2):
        print(i, j)
```

Practice Exercise: Write a Python program that prints the multiplication table (up to 10) using nested loops.

4. Functions:

Functions in Python are blocks of reusable code that perform a specific task. They help in organizing code into manageable pieces and promoting code reusability.

Syntax of Function Definition:

```
def function_name(parameters):
    """
    Docstring: Description of the function
    """
    # Block of code to execute
    return [expression]  # Optional return statement
```

- **def** keyword is used to define a function.
- **function_name** is the name of the function.
- **parameters** are inputs to the function (optional).
- Docstring provides documentation for the function (optional).
- The block of code to execute is indented.
- **return** statement (optional) exits the function and optionally returns a value.

Example:

```
def greet(name):
    """
    Greets the user with the provided name
    """
    return "Hello, " + name + "!"

print(greet("Alice"))  # Output: Hello, Alice!
```

Practice Exercise: Write a Python function called **calculate_area** that takes the length and width of a rectangle as parameters and returns its area.

5. Variable Scope:

Variable scope refers to the visibility and accessibility of variables within different parts of a Python program.

- **Global Scope:** Variables defined outside of any function have global scope and can be accessed from anywhere in the program.
- **Local Scope:** Variables defined inside a function have local scope and can only be accessed within that function.

Example:

```
x = 10  # Global variable

def my_function():
    y = 20  # Local variable
    print("Inside function:", x, y)

my_function()
print("Outside function:", x)
# print(y)  # This line will result in an error because y is not defined outside the
```

Practice Exercise: Write a Python program that demonstrates variable scope. Define a global variable **global_var** and a function called **test_scope** that defines a local variable **local_var**. Inside the function, print both **global_var** and **local_var**. Outside the function, print only **global_var**.

---

6. Data Structures:

6.1. Lists:

Lists are ordered collections of items in Python. They are mutable, meaning their elements can be changed after they are created.

Syntax:

```
# Creating a list
my_list = [item1, item2, item3, ...]

# Accessing elements
element = my_list[index]

# Modifying elements
my_list[index] = new_value

# List methods
my_list.append(item)  # Adds an item to the end of the list
my_list.insert(index, item)  # Inserts an item at the specified index
my_list.remove(item)  # Removes the first occurrence of the specified item
```

Example:

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

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

# Modifying elements
fruits[1] = 'grape'
print(fruits)  # Output: ['apple', 'grape', 'orange']

# List methods
fruits.append('mango')
print(fruits)  # Output: ['apple', 'grape', 'orange', 'mango']
```

Practice Exercise: Write a Python program that creates a list of your favorite movies, then adds a new movie to the list and prints the updated list.

6.2. Tuples:

Tuples are ordered collections similar to lists, but they are immutable, meaning their elements cannot be changed after creation.

Syntax:

```
# Creating a tuple
days_of_week = ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',

# Accessing elements
print(days_of_week[0])  # Output: Monday
```

Practice Exercise: Write a Python program that creates a tuple containing the names of the months, then prints the months in reverse order.

6.3. Dictionaries:

Dictionaries are unordered collections of key-value pairs in Python. They are mutable and indexed by keys, which can be of any immutable type.

Syntax:

```
# Creating a dictionary
my_dict = {key1: value1, key2: value2, ...}

# Accessing elements
value = my_dict[key]

# Modifying elements
my_dict[key] = new_value

# Dictionary methods
my_dict[key] = value  # Adds or updates an item with the specified key
my_dict.pop(key)  # Removes the item with the specified key
```

Example:

```python
# Creating a dictionary
student = {'name': 'Alice', 'age': 20, 'grade': 'A'}

# Accessing elements
print(student['name'])  # Output: Alice

# Modifying elements
student['age'] = 21
print(student)  # Output: {'name': 'Alice', 'age': 21, 'grade': 'A'}

# Dictionary methods
student['city'] = 'New York'
print(student)  # Output: {'name': 'Alice', 'age': 21, 'grade': 'A', 'city': 'New Yo
```

Practice Exercise: Write a Python program that creates a dictionary containing the names of your friends as keys and their corresponding ages as values. Then, print the age of a specific friend.

6.4. Sets:

Sets are unordered collections of unique elements in Python. They do not allow duplicate elements.

Syntax:

```python
# Creating a set
my_set = {item1, item2, item3, ...}

# Set methods
my_set.add(item)  # Adds an item to the set
my_set.remove(item)  # Removes the specified item from the set
```

Example:

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

# Set methods
colors.add('yellow')
print(colors)  # Output: {'red', 'green', 'blue', 'yellow'}

colors.add('red')  # Adding a duplicate element
print(colors)  # Output: {'red', 'green', 'blue', 'yellow'}
```

Practice Exercise: Write a Python program that takes a list of numbers as input from the user and creates a set containing only the unique numbers from the list.

6.5. kwargs:

kwargs is not a standalone data structure but rather a special syntax in Python used to pass a variable number of keyword arguments to a function.

Syntax:

```python
def my_function(**kwargs):
    # kwargs is a dictionary containing the keyword arguments
    for key, value in kwargs.items():
        print(key, value)

# Calling the function
my_function(arg1=value1, arg2=value2, ...)
```

Example:

```python
def greet(**kwargs):
    for key, value in kwargs.items():
        print(key, ":", value)

greet(name="Alice", age=30)
# Output:
# name : Alice
# age : 30
```

Practice Exercise: Write a Python function called **calculate_sum** that takes an arbitrary number of keyword arguments representing numbers and returns their sum.

Task 1: Type Casting

Lab Instructions: Type Casting Input

In this lab you will be presented with three exercises to demonstrate how explicit type casting can be used to solve data being inputted from an end user. Each exercise will ask you to solve a particular problem relating to types.

****Tips: Before you Begin****

To view your code and instructions side-by-side, select the following in your VSCode toolbar:

> - View -> Editor Layout -> Two Columns

> - To view this file in Preview mode, right click on this README.md file and `Open Preview`

> - Select your code file in the code tree, which will open it up in a new VSCode tab.

> - Drag your assessment code files over to the second column.

****To run your Python code****

> - Select your Python file in the Visual Studio Code file tree

> - You can right click the file and select "Run Python File in Terminal"

> or run the file using the smaller play button in the upper right-hand corner of VSCode.

(Select "Run Python File in Terminal" in the provided button dropdown)

> - Alternatively, you can follow lab instructions which use python3 commands to run your code in terminal.

There are two exercises and objectives of this activity:

- **Exercise 1:** Use explicit casting to apply the correct cast type

- **Exercise 2:** Fix the script so it correctly outputs the bill total

Exercise 1:

In this exercise, you'll use explicit casting to apply the correct cast type.

Instructions

1. Open the script exercise1.py present inside the project folder

2. To run the script, open terminal and execute the following command:

```
python3 exercise1.py
```

3. Step 3: Fix the script so the variables have the correct type.

Exercise 2:

Your goal of this exercise is to fix the script so that each variable has the correct type.

Instructions

1. Open the script exercise2.py present inside the project folder

2. To run the script, open terminal and execute the following command. You will be prompted to enter some values.

```
python3 exercise2.py
```

3. Fix the script so it outputs the correct bill total based on the data being entered.