**Worksheet 1: Variables, Expressions, and Assignments**

#### **Activity 1: Simple Calculations**

# Defining variables x and y

x = 5

y = 10

# Performing operations

addition = x + y # 15

subtraction = y - x # 5

multiplication = x \* y # 50

division = y / x # 2.0

modulus = y % x # 0

# Output the results

print("Addition:", addition)

print("Subtraction:", subtraction)

print("Multiplication:", multiplication)

print("Division:", division)

print("Modulus:", modulus)

#### **Activity 2: Working with Variables**

# Assigning a string to a variable

name = "Alice"

print(name) # Output: Alice

# Concatenating strings

greeting = "Hello, " + name + "!"

print(greeting) # Output: Hello, Alice!

# Changing variable values

name = "Bob"

greeting = "Hello, " + name + "!"

print(greeting) # Output: Hello, Bob!

#### **Reflection Questions**

* **What happens when you try to add an integer and a string?**
  + You get a TypeError. Python does not automatically convert data types like this.
* **Can a variable change types after being assigned?**
  + Yes, Python is dynamically typed. A variable can be reassigned to a value of a different data type at any time.

### ****Worksheet 2: Strings, Lists, and Tuples****

#### **Activity 1: String Manipulation**

# Assigning a sentence to a string variable

sentence = "The quick brown fox jumps over the lazy dog"

# Using string methods

print(sentence.upper()) # Output: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

print(sentence.lower()) # Output: the quick brown fox jumps over the lazy dog

print(sentence.replace("fox", "cat")) # Output: The quick brown cat jumps over the lazy dog

words = sentence.split() # Splits the sentence into a list of words

print(words) # Output: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

# Concatenating two strings

new\_sentence = sentence + " and runs away"

print(new\_sentence) # Output: The quick brown fox jumps over the lazy dog and runs away

#### **Activity 2: List Operations**

# Creating a list and a tuple

my\_list = [1, 2, 3, 4, 5]

my\_tuple = (6, 7, 8, 9, 10)

# List operations

my\_list.append(6) # Adding to the list

print(my\_list) # Output: [1, 2, 3, 4, 5, 6]

my\_list.remove(3) # Removing an element

print(my\_list) # Output: [1, 2, 4, 5, 6]

my\_list.sort() # Sorting the list

print(my\_list) # Output: [1, 2, 4, 5, 6]

# Tuple operations (tuples are immutable, so they can't be modified directly)

try:

my\_tuple[0] = 11

except TypeError as e:

print(e) # Output: 'tuple' object does not support item assignment

# Converting tuple to list for modification

my\_list\_from\_tuple = list(my\_tuple)

my\_list\_from\_tuple.append(11)

print(my\_list\_from\_tuple) # Output: [6, 7, 8, 9, 10, 11]

#### **Reflection Questions**

* **Why can’t we modify tuples?**
  + Tuples are immutable, meaning their contents cannot be changed after they are created.
* **How would you convert a tuple to a list if you need to modify it?**
  + You can use the list() function to convert a tuple into a list, which can then be modified.

### ****Worksheet 3: Objects, Classes, and Python Standard Library****

#### **Activity 1: Concepts of Classes, Objects, and Modules**

# Defining a Dog class

class Dog:

def \_\_init\_\_(self, name, breed):

self.name = name

self.breed = breed

def bark(self):

return f"{self.name} is barking!"

# Creating an object of the Dog class

my\_dog = Dog("Buddy", "Golden Retriever")

print(my\_dog.bark()) # Output: Buddy is barking!

# Example of a module (math)

import math

print(math.sqrt(16)) # Output: 4.0

#### **Activity 2: Using the Python Standard Library**

# Importing and using the random module

import random

print(random.randint(1, 10)) # Random number between 1 and 10

# Importing and using the datetime module

import datetime

print(datetime.datetime.now()) # Output: Current date and time

# Importing and using the math module

import math

print(math.sqrt(25)) # Output: 5.0

#### **Reflection Questions**

* **What are the benefits of organizing code into classes and modules?**
  + Classes encapsulate related data and functions, promoting code reuse and readability. Modules allow for separation of concerns, making code easier to maintain and reuse across different projects.
* **How do modules make Python code easier to manage and reuse?**
  + Modules package code into reusable components. Instead of rewriting functionality, you can import pre-existing modules (like math, random, etc.), making development more efficient.