Packaging and Testing

Activity 1: Run the following code using Pylint and identify the errors.

```
def factorial (x)
   if x == 1:
      return 1

   else:
      return (x * factorial(x-1))

num = 3
print("The factorial of", num, "is", factorial(num))
```

Source: Progamiz. (n.d.) **Python Recursion**.

When I tried to run the code, it came up with the following error:

```
(base) C:\Users\hcham\Desktop\Essex\2. Object Oriented Programming May 2023\Unit 10>pylint
activity_1.py
************* Module activity_1
activity_1.py:1:18: E0001: Parsing failed: 'expected ':' (<unknown>, line 1)' (syntax-error)
```

This means a syntax error in the file activity_1.py on line 1, at character 18. The error message indicates that the parser was expecting a colon (:) at that location.

The code has a missing docstring, meaning the function factorial() does not have a docstring, briefly describing what the function does. A docstring is a good practice, as it helps other developers understand what the function does (pandas.pydata.org, N.D.).

The code has an incorrect indentation, which makes the code difficult to read and understand.

The code has a new variable, x, which is not used after it is defined. This is a good practice to avoid, as it can lead to errors (uk.mathworks.com, N.D.).

The variable factorial is redefined in the function factorial(). This is not allowed, as it results in errors (Programiz.com, 2020).

Activity 2: In 'Packaging & Testing' (unit 9), we examined the use of documentation to support code developments. Add appropriate commenting and documentation for the code below.

```
def add(x, y):
return x + y
def subtract(x, y):
return x - y
def multiply(x, y):
return x * y
def divide(x, y):
return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
while True:
choice = input("Enter choice(1/2/3/4): ")
if choice in ('1', '2', '3', '4'):
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1': print(num1, "+", num2, "=", add(num1, num2))
elif choice == '2': print(num1, "-", num2, "=", subtract(num1, num2))
elif choice == '3': print(num1, "*", num2, "=", multiply(num1, num2))
elif choice == '4': print(num1, "/", num2, "=", divide(num1, num2))
 break the while loop if answer is no next_calculation = input("Let's do next calculation? (yes/no): ")
if next_calculation == "no":
break
else: print("Invalid Input")
```

Source: Progamiz. (n.d.) **Python Program to Make a Simple Calculator**.

Here is my updated code with the necessary documentation and comments based on the example in (Programiz, N.D.):

```
# Define basic arithmetic functions
def add(x, y):
    """Return the sum of two numbers."""
    return x + y

def subtract(x, y):
    """Return the difference of two numbers."""
    return x - y

def multiply(x, y):
    """Return the product of two numbers."""
```

```
    return x * y

def divide(x, y):
    """Return the quotient of two numbers."""
    return x / y

# Print the available operations
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

# Loop until the user exits
while True:
    # Ask the user to choose an operation
    choice = input("Enter choice(1/2/3/4): ")

    # Check if the choice is valid
    if choice in ('1', '2', '3', '4'):
        # Ask the user to enter two numbers
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))

        # Perform the chosen operation and print the result
        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))
        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))
        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))
        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))

        # Ask the user if they want to continue or exit
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
            break # Exit the loop
    else:
        print("Invalid Input") # Handle invalid input
```

Activity 3: Read the article by Rani et al. (2021). What impact does this article have on the way in which you have commented the code in the task above?

The article discusses the significance of code comments and consistency in projects and emphasises the importance of automating software documentation. The article suggests that using tools to evaluate the quality of documentation during each stage

of development can aid developers in maintaining high-quality documentation. Additionally, the article provides a dataset of labelled conventions for comments and a methodology to extract guidelines, which can assist in expanding the scope of work and developing tools to enhance comment quality. This article highlights the importance of following commenting conventions and improving general documentation quality in code development.

Activity 4: Integrate unit tests into the code in Activity 2 to test the operation of the methods.

I have integrated unit tests into the code using the Python unittest module, as instructed in Activity 2. I am sharing the updated version of the code, which includes the unittests:

```
import unittest

# Define basic arithmetic functions
def add(x, y):
    """Return the sum of two numbers."""
    return x + y

def subtract(x, y):
    """Return the difference of two numbers."""
    return x - y

def multiply(x, y):
    """Return the product of two numbers."""
    return x * y

def divide(x, y):
    """Return the quotient of two numbers."""
    return x / y

# Unit tests for arithmetic functions
class TestArithmeticFunctions(unittest.TestCase):

    def test_add(self):
        result = add(3, 5)
        self.assertEqual(result, 8)

    def test_subtract(self):
```

```
        result = subtract(10, 4)
        self.assertEqual(result, 6)

    def test_multiply(self):
        result = multiply(2, 6)
        self.assertEqual(result, 12)

    def test_divide(self):
        result = divide(10, 2)
        self.assertEqual(result, 5)

# Run the tests
if __name__ == '__main__':
    unittest.main()

# Print the available operations
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

# Loop until the user exits
while True:
    # Ask the user to choose an operation
    choice = input("Enter choice (1/2/3/4): ")

    # Check if the choice is valid
    if choice in ('1', '2', '3', '4'):
        # Ask the user to enter two numbers
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))

        # Perform the chosen operation and print the result
        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))
        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))
        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))
        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))

        # Ask the user if they want to continue or exit
        next_calculation = input("Let's do the next calculation? (yes/no): ")
        if next_calculation == "no":
            break  # Exit the loop
    else:
        print("Invalid Input")  # Handle invalid input
```

I have included a new class, TestArithmeticFunctions, in this modified code, which derives from unittest.TestCase. I specify unique test procedures for each arithmetic operation within this class. Each test function makes use of assertions that unittest provides.TestCase to see if the function provides the desired outcome.

When I ran the test with the command python -m unittest test_filename.py, as explained by (Shaik, 2021), the following results came up:

```
(base) C:\Users\hcham\Desktop\Essex\2. Object Oriented Programming May 2023\Unit 10>pylint
activity_4.py
************* Module activity_4
activity_4.py:28:0: C0303: Trailing whitespace (trailing-whitespace)
activity_4.py:34:0: C0303: Trailing whitespace (trailing-whitespace)
activity_4.py:44:0: C0303: Trailing whitespace (trailing-whitespace)
activity_4.py:50:0: C0304: Final newline missing (missing-final-newline)
activity_4.py:1:0: C0114: Missing module docstring (missing-module-docstring)
activity_4.py:1:8: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-
name)
activity_4.py:1:11: C0103: Argument name "y" doesn't conform to snake_case naming style (invalid-
name)
activity_4.py:5:13: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-
name)
activity_4.py:5:16: C0103: Argument name "y" doesn't conform to snake_case naming style (invalid-
name)
activity_4.py:9:13: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-
name)
activity_4.py:9:16: C0103: Argument name "y" doesn't conform to snake_case naming style (invalid-
name)
activity_4.py:13:11: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-
name)
activity_4.py:13:14: C0103: Argument name "y" doesn't conform to snake_case naming style (invalid-
name)

-----------------------------------
Your code has been rated at 5.67/10


(base) C:\Users\hcham\Desktop\Essex\2. Object Oriented Programming May 2023\Unit 10>python -m
unittest activity_4.py
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 1
```

References:

pandas.pydata.org. (N.D.). *pandas docstring guide — pandas 1.4.3 documentation*. [online] Available at: https://pandas.pydata.org/docs/development/contributing_docstring.html#:~:text=A%20Python%20docstring%20is%20a.

uk.mathworks.com. (n.d.). *Best Practices for Defining Variables for C/C++ Code Generation - MATLAB & Simulink - MathWorks United Kingdom*. [online] Available at: https://uk.mathworks.com/help/coder/ug/best-practices-for-defining-variables-for-c-c-code-generation.html [Accessed 9 Jul. 2023].

Programiz.com. (2020). *Python Recursion (Recursive Function)*. [online] Available at: https://www.programiz.com/python-programming/recursion.

Programiz (N.D.). *Python Program to Make a Simple Calculator*. [online] Available at: https://www.programiz.com/python-programming/examples/calculator.

Shaik, H.K. (2021). *Unit Testing with Python unittest Module*. [online] Geekflare. Available at: https://geekflare.com/unit-testing-with-python-unittest/.