Seminar Preparation - 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))
```

The result is as follows:

```
************* Module recursion_pylint
recursion_pylint.py:6:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)
recursion_pylint.py:1:0: C0114: Missing module docstring (missing-module-docstring)
recursion_pylint.py:1:0: C0116: Missing function or method docstring (missing-function-docstring)
recursion_pylint.py:1:15: C0103: Argument name "x" doesn't conform to snake_case naming style
(invalid-name)
recursion_pylint.py:2:4: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the
code inside it (no-else-return)
recursion_pylint.py:8:0: C0103: Constant name "num" doesn't conform to UPPER_CASE naming style
(invalid-name)

------------------------------------------------------------------
Your code has been rated at 0.00/10 (previous run: 0.00/10, +0.00)
```

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

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):
    “””Addition of 2 int

    Parameters
    ----------
    x: int
      any numbers
    y: int
      any numbers

    Returns
    -------
    int
      result of addition
    “””
    return x + y

def subtract(x, y):
   “””Subtraction of 2 int

    Parameters
    ----------
    x: int
      any numbers
    y: int
      any numbers

    Returns
    -------
    int
      result of subtraction
    “””
    return x - y

def multiply(x, y):
   “””Multiplication of 2 int

    Parameters
    ----------
    x: int
      any numbers
    y: int
      any numbers

    Returns
    -------
    int
      result of multiplication
    “””
    return x * y

def divide(x, y):
    “””Division of 2 int

    Parameters
    ----------
    x: int
```

```
      any numbers
    y: int
      any numbers

    Returns
    -------
    int
      result of division
    “””
    return x / y

#Printing selection of operation for user
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
    “””The while loop is executed until the break statement occurs.
    While the while loop is executing, the user can select one of the
    arithmetic operation by typing in the corresponding number”””




    choice = input("Enter choice(1/2/3/4): ")
    if choice in ('1', '2', '3', '4'):
                #convert string input to float
        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**.

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 by Rani et al. (2021) shows how developers use comments and documentation for code. In summary, developers do not often comment clearly on their code because writing comments besides writing code is an extra effort. One solution is to have a template for commenting and documentation to increase efficiency and reduce the time and effort required to write these comments. Compared to the Google, Oracle and PEP257 commenting guidelines, the NumPy style has stricter conventions but is more readable than the others. In addition, linters (like pylint) and other automatic checkers of the code can help maintain high-quality documentation. In Activity 2, I used the NumPy comment guidelines to document the code. In my opinion, it is easier to understand and more transparent than the other guidelines (Rani et al., 2021).

Activity 4

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

```
import unittest
from using_unittest import add, subtract, multiply, divide


class TestUsing_unitest(unittest.TestCase):

  def test_add(self):
      result = add(2,2)
      self.assertEqual(result, 4)

  def test_subtract(self):
      result = subtract(2,2)
      self.assertEqual(result, 0)

  def test_multiply(self):
      result = multiply(2,2)
      self.assertEqual(result, 4)

  def test_divide(self):
      result = divide(2,2)
      self.assertEqual(result, 1)


if __name__ == '__main__':
    unittest.main()
```

References:

Rani, P., Abukar, S., Stulova, N., Bergel, A. & O. Nierstrasz (2021) Do Comments follow Commenting Conventions? A Case Study in Java and Python. 2021 IEEE 21st

International Working Conference on Source Code Analysis and Manipulation (SCAM).

165-169. DOI: 10.1109/SCAM52516.2021.00028.