Assessment

A collection of programme extracts and exercises to assess the knowledge and skills acquired during the OOP module.

Notes from week 1 are illustrative of the type of exercises that will be included in this assessment. Student submissions will feature exerceses and tasks from weeks 2 - 8 of the module.

Week 1

Section 1. Setting Up a Code Editor and Running Python Scripts

The documentation for python can be found at the python documentation website. The documentation for Visual Studio Code can be found at the Visual Studio Code documentation website.

Exercise 0: Setting Up Visual Studio Code Editor

The lab machines havebeen set up with python 3 which can be run on visual studio code. Instructions for installing python 3 on personal machines can be found on the python website, and instructions for installing visual studio code can be found on the visual studio code website.

Exercise 1: Hello World

Starter file lab_weeK1.py contains a simple program that prints "Hello World" to the console. Run the program and ensure it works correctly.

```python
# lab_week1.py
# Your first Python program
print("Hello. Welcome to the OOP Module.")
print("This program has been run successfully.")


class_name = "OOP"
number_of_students = 20
my_string = "Welcome to the " + class_name + " module. There are " +
str(number_of_students) + " students in this class."

my_string = f"Welcome to the {class_name} module. There are {number_of_students}
students in this class."
```

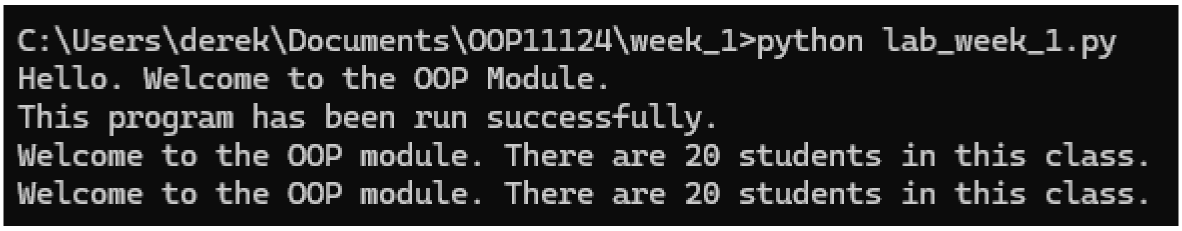
Output when run:

```
Hello. Welcome to the OOP Module.
This program has been run successfully.
```

Exercise 2:Running a Script Using the Command Line

Exercise 2: Running a Script Using the Command Line

The script has been run successfully in the command prompt.



```
C:\Users\derek\Documents\OOP11124\week_1>python lab_week_1.py
Hello. Welcome to the OOP Module.
This program has been run successfully.
Welcome to the OOP module. There are 20 students in this class.
Welcome to the OOP module. There are 20 students in this class.
```

Section 2: Python Introduction

Exercise 1: Variables and Types

Create a file called "lab_week1_part2.py" with starting code from the labsheet, mark the types of the variables as comments and add print statements to confirm the types of the variables.

```
# lab_week1_part2.py
var_1 = True #Type = bool
var_2 = 1 #Type =  int
var_3 = 3.14159 #Type = float
var_4 = "Hello World" #Type = string

print(type(var_1))
print(type(var_2))
print(type(var_3))
print(type(var_4))
```

Output when run:

```
<class 'bool'>
<class 'int'>
<class 'float'>
<class 'str'>
```

Task:

1. Create a variable called my_int and assign it the value 5.
2. Create a variable called my_float and assign it the value 5.5.
3. Create a variable called my_bool and assign it the value True.
4. Print the value of each variable.
5. Cast my_int to a float and store the result in a variable called my_int_float.
6. Cast my_float to an integer and store the result in a variable called my_float_int.
7. Cast my_bool to an integer and store the result in a variable called my_bool_int.
8. Print the value of each variable.

Code produced:

```
# lab_week1_part2.py
my_int = 5
my_float = 5.5
my_bool = True

print(my_int)
print(my_float)
print(my_bool)

my_int_float = float(my_int)
my_float_int = int(my_float)
mybool_int = int(my_bool)

print("after casting")
print(my_int_float)
print(my_float_int)
print(mybool_int)
```

Output when run:

```
5
5.5
True
after casting
5.0
5
1
```

This demonstrates the casting of variables from one type to another, and the output confirms the successful conversion.

Exercise 2: Arithmetic Operators

Practice using the python operators +, -, *, /, //, %, ** with the following code:

```
# Exercise 2 Arithmetic operators
#Addition
result_addition = 10 + 5
print("Addition:", result_addition)

#Subtraction
result_subtraction = 20 - 8
print("Subtraction:", result_subtraction)

#Multiplication
result_multiplication = 6 * 4
```

```
print("Multiplication:", result_multiplication)

#Division
result_division = 15 / 3
print("Division:", result_division)

# Floor Division
result_floor_division = 17 // 4
print("Floor Division:", result_floor_division)

#Modulus
result_modulus = 17 % 4
print("Modulus:", result_modulus)

#Exponentiation
result_exponentiation = 2 ** 3
print("Exponentiation:", result_exponentiation)
```

Output when run:

```
Addition: 15
Subtraction: 12
Multiplication: 24
Division: 5.0
Floor Division: 4
Modulus: 1
Exponentiation: 8
```

This confirms the operations have been performed correctly.

Task: Calculate the average of three numbers

```
num1 = 10
num2 = 20
num3 = 20
average = (num1 + num2 + num3) / 3
print("Average of 3 numbers:", average)
```

Output when run:

```
Average of 3 numbers: 16.666666666666668
```

A refinement of this could display the average to two decimal places:

```
print("Average of 3 numbers:", round(average, 2))
```

Output when run:

```
Average of 3 numbers: 16.67
```

Task: Calculate the area of a rectangle

1. Create two variables, length and width, representing the dimensions of a rectangle.
2. Calculate the area of the rectangle using the arithmetic operators and assign it to a variable called area.
3. Print a message that displays the values of length, width, and the calculated area

```
# Task : Calculate the area of a rectangle
length = 5
width = 3
area = length * width
print("Area of the rectangle:", area)
```

Output when run:

```
Area of the rectangle: 15
```

Exercise 3: Strings and f-Strings

Strings are a fundamental part of Python. They can be created using single or double quotes. f-Strings allow for easy string formatting. Strings are objects in Python, and you can use methods on them.

The following table of string methods is found at the W3 schools.

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string

Method	Description
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list

Method	Description
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Task: Modify Strings

This short example demonstrates the use of some of these methods:

1. Create a new variable called my_string and store the following in it: “This class covers OOP.”
2. Print the my_string variable
3. Using the documentation in the link above, look through the examples and do the following:
    1. Create a new variable my_uppercase_string that stores the my_string converted to uppercase
    2. Create a new variable my_lowercase_string that stores the my_string converted to lowercase
    3. Create a new variable my_new_string that replaces the word “OOP” in the string with “Object Oriented Programming”.
    4. Create a new variable my_string_length that stores the length of the string

```
my_string = "This class covers OOP"
print(my_string)
my_uppercase_string = my_string.upper()
print(my_uppercase_string)
my_lowercase_string = my_string.lower()
print(my_lowercase_string)
my_new_string = my_string.replace("OOP", "Object Oriented Programming")
print(my_new_string)
my_string_length = len(my_string)
print(my_string_length)
```

Output when run:

```
This class covers OOP
THIS CLASS COVERS OOP
this class covers oop
This class covers Object Oriented Programming
21
```

Task: String Formatting with f-Strings

f-Strings allow for easy string formatting. The following example demonstrates how to use f-Strings to format a string with variables.

```
my_name = "Peter"
number_of_classes = 4
campus = "Paisley"
my_text = f"My name is {my_name} and I am studying {number_of_classes} classes in {campus}."
print(my_text)
```

Output when run:

```
My name is Peter and I am studying 4 classes in Paisley.
```

Section 2: First Program in Python

The following code is a simple program that converts a number stored as celsius_input to fahrenheit and kelvin temperature scales and prints the result.

The forulas used can be found at the cuemath.

The code is as follows:

```
# Temperatuire Converter
# This program converts temperature from Celsius to Fahrenheit and Kelvin.

celcius_input = 20  # Example input edit line to test the program without user input
# Convert the input to a float
celcius_input = float(celcius_input)
degree_fahrenheit = (celcius_input * 9/5) + 32
degree_kelvin = celcius_input + 273.15
print(f"{celcius_input} degree Celsius is equal to {degree_fahrenheit:.2f} Fahrenheit.")
print(f"{celcius_input} degree Celsius is equal to {degree_kelvin:.2f} Kelvin.")
print("Thank you for using the Temperature Converter!")
# End of program

# note that this programm needs to be run in a terminal that supports input and output, such as a command prompt or terminal window.
# It will not work in an IDE that does not support input.
```

Output when run:

```
20.0 degree Celsius is equal to 68.00 Fahrenheit.
20.0 degree Celsius is equal to 293.15 Kelvin.
Thank you for using the Temperature Converter!
```

A refinement of this program could be to allow the user to input a temperature in Celsius, and then convert it to Fahrenheit and Kelvin. This can be done using the input() function.
