A logo with blue and red text

Description automatically generated A logo with text on it

Description automatically generated

**Introduction to Programming**

**Lab Worksheet**

**Week 4**

**Python**

Name: Nikita Sah

Level 4 Section: A

British Id: 10011

LBU Id: c7576150

Level 4 BSc. Hons Computing

Subject: Fundamental of Computer Programming

(FOCP)

The British College (TBC)

**Task:**

**#Week 4 Practical**

1. Write some code that imports the math *module*, then calculates and prints the square root of the number 2401. Use the sqrt() function provided by the math module.

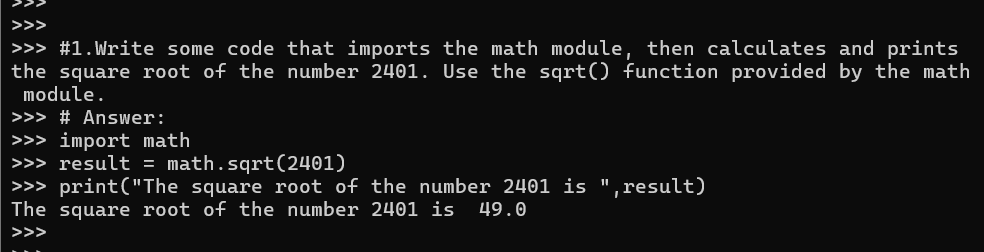
**Answer:**

import math

result = math.sqrt(2401)

print("The square root of the number 2401 is ",result)

**Output of Question No. 1:**

****

1. Write some code that imports only the log2() function from the math *module*, then call this function to calculate the log base 2 value of 1024. Print the result to the screen.

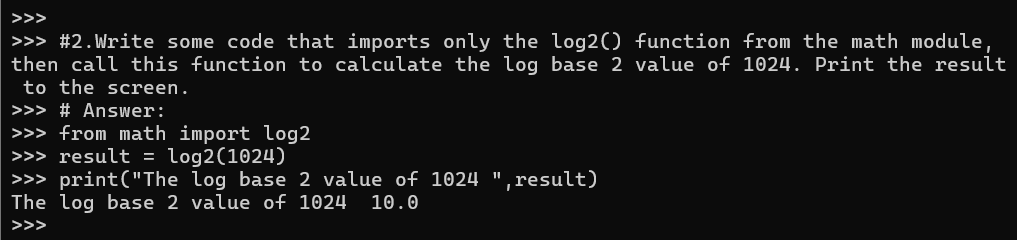
**Answer:**

from math import log2

result = log2(1024)

print("The log base 2 value of 1024 ",result)

**Output of Question No. 2:**



1. Input the below function definition. Once that is done make several calls to the function passing different argument values for testing.

def displayTwice(msg):

print(msg)

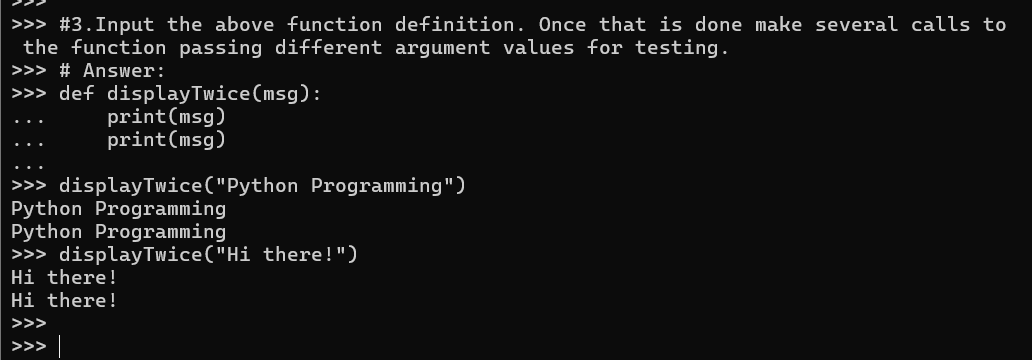
print(msg)

**Answer:**

displayTwice("Hi there!")

displayTwice("Python Programming")

**Output of Question No. 3:**

****

1. Re-Input the above function definition, but this time add a docstring that includes a description of the function’s purpose. Once that is done enter a command such as help(displayTwice) and see what it displays.

**Answer:**

def displayTwice(msg):

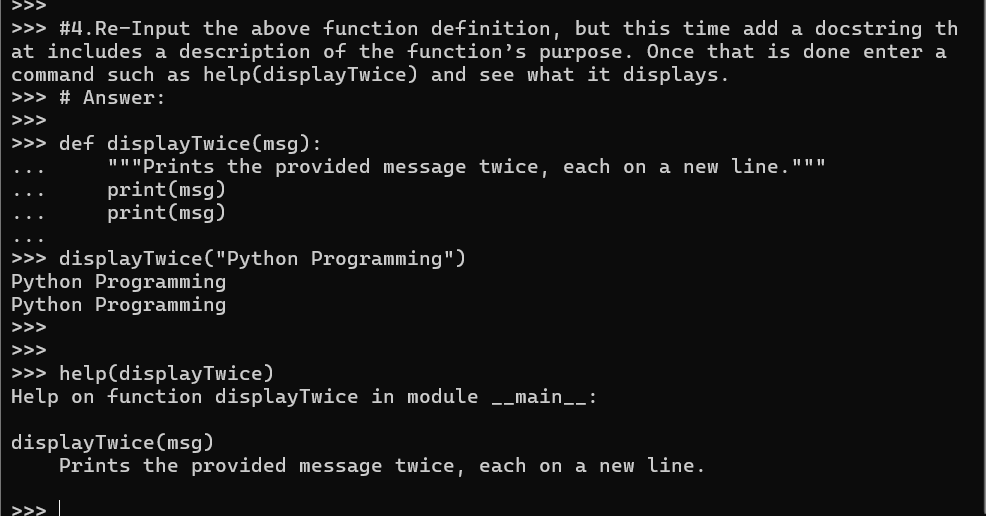
    """Prints the provided message twice, each on a new line."""

    print(msg)

    print(msg)

displayTwice("Hi there!")

**Output of Question No. 4:**



1. Input the below function definition. Once that is done make several calls to the function passing different argument values and displaying the returned value.

**Answer:**

def findMax(a,b):

    """Finds the maximum of two values."""

    if ( a > b ):

        max = a

    else:

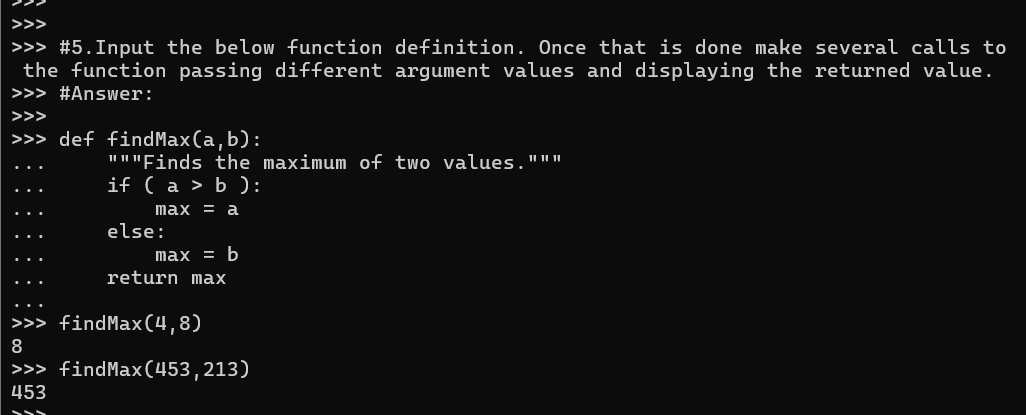
        max = b

    return max

findMax(4,8)

findMax(453,213)

**Output of Question No. 5:**

****

1. Define a function that takes two numeric values, multiplies them together then returns the result. If the function is called with only a single argument however, then the value should be multiplied by itself. Once the function is defined, call it several times and display the returned values for testing purposes.

**Answer:**

def multiply\_or\_square(x, y=None):

    """Multiplies two numbers or squares the number if only one is provided."""

    if y is None:

        return x \* x

    return x \* y

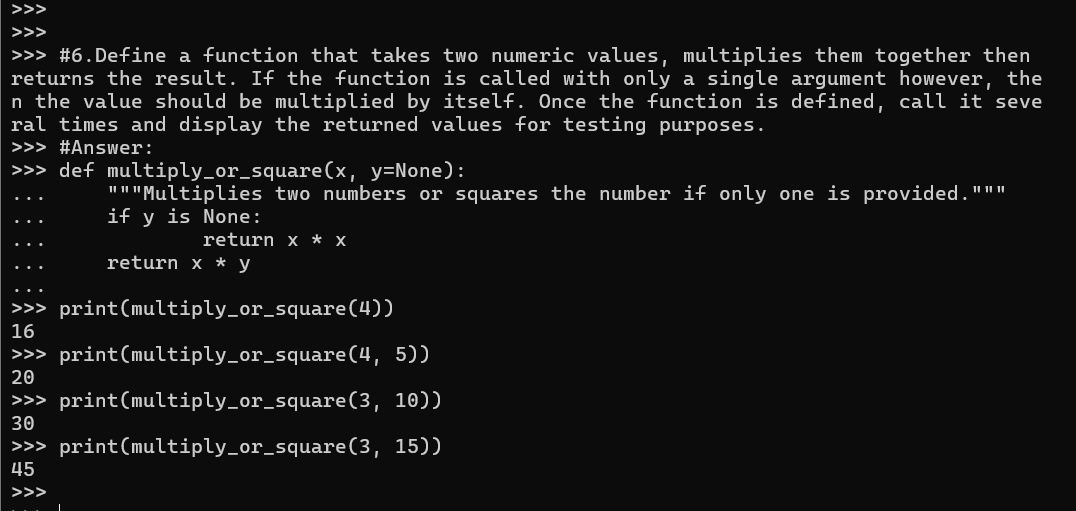
print(multiply\_or\_square(4))

print(multiply\_or\_square(4, 5))

print(multiply\_or\_square(7))

print(multiply\_or\_square(3, 15))

**Output of Question No. 6:**



1. Enter the example function shown above, then try calling it using the *keyword arguments* in several different orders.

**Answer:**

def multiply\_or\_square(x, y=None):

    """Multiplies two numbers or squares the number if only one is provided."""

    if y is None:

        return x \* x

    return x \* y

# Call using the required argument

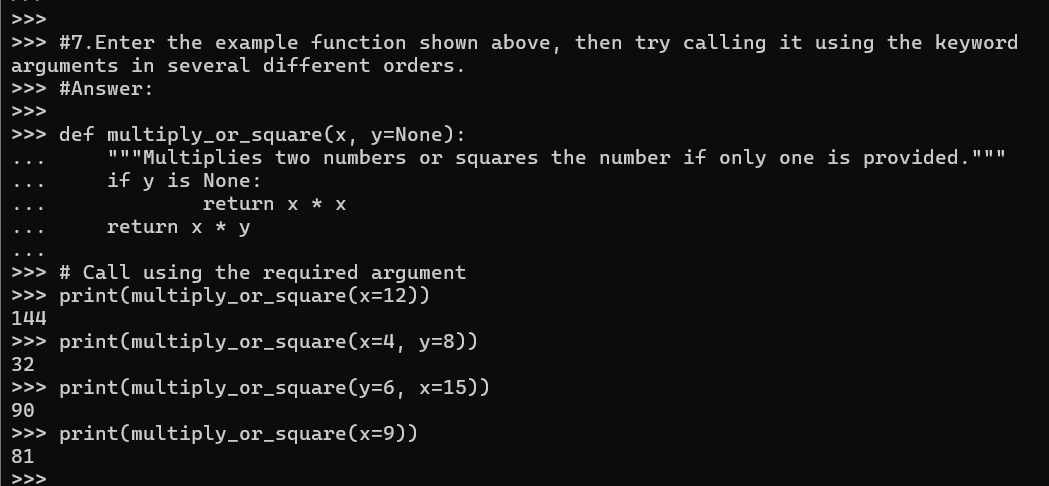
print(multiply\_or\_square(x=12))

print(multiply\_or\_square(x=4, y=8))

print(multiply\_or\_square(y=6, x=15))

print(multiply\_or\_square(x=9))

**Output of Question No. 7:**



1. The built-in print() function supports a keyword argument called sep. This is used to decide what character to display between each of the provided positional parameters. Write some code that makes several calls to the print() function while setting the sep argument to values other than a space (which is the default).

**Answer:**

# Using a comma as the separator

print("apple", "banana", "cherry", sep=", ")

# Using a dash as the separator

print("2024", "12", "29", sep="-")

# Using an asterisk as the separator

print("Python", "is", "fun", sep="\*")

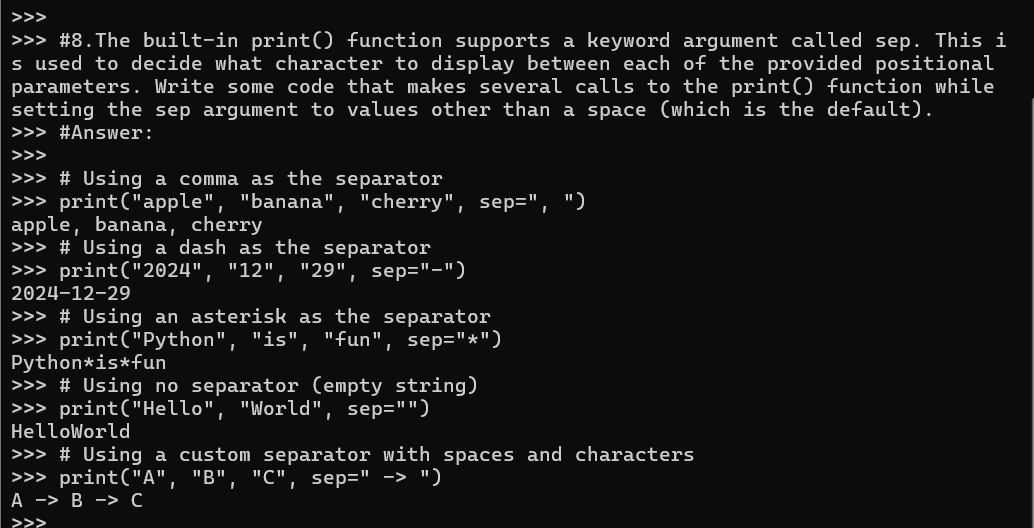
# Using no separator (empty string)

print("Hello", "World", sep="")

# Using a custom separator with spaces and characters

print("A", "B", "C", sep=" -> ")

**Output of Question No. 8:**



1. Enter the example function shown below, then try calling it several times, passing a different number of numeric arguments each time.

**Answer:**

def calcAve(\*numbers):

    """Calculates the average of the given numbers."""

    total = 0

    for num in numbers:

        total += num

    return total/len(numbers)

# Call with 2 arguments

print(calcAve(10, 20))

# Call with 3 arguments

print(calcAve(5, 15, 25))

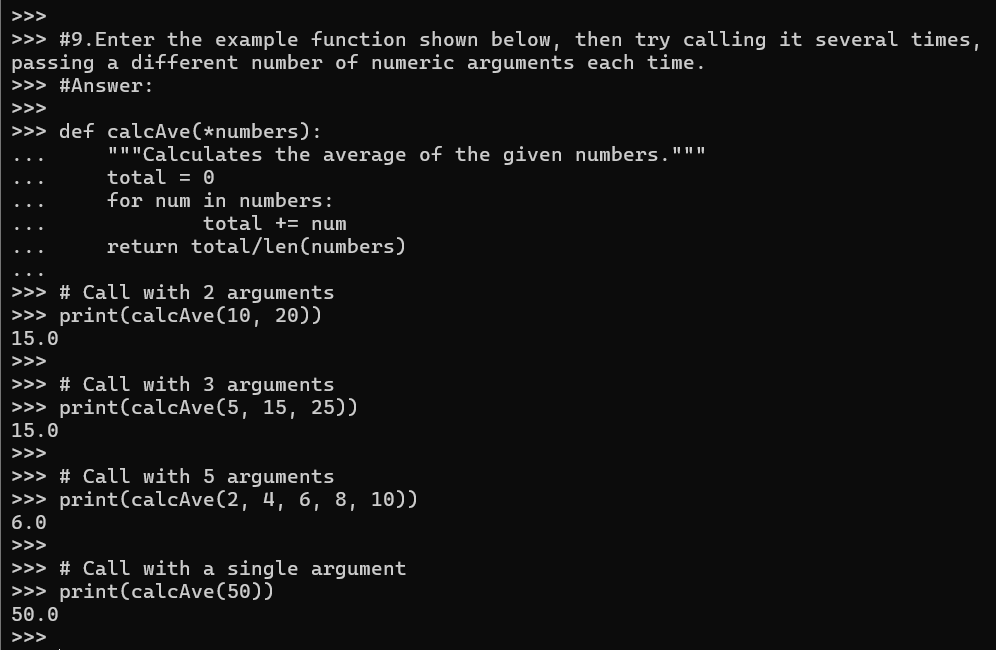
# Call with 5 arguments

print(calcAve(2, 4, 6, 8, 10))

# Call with a single argument

print(calcAve(50))

**Output of Question No. 9:**



1. Enter the example lambda expression shown below, then find out the *data type* of the hypot variable using a call to the type() function.

hypot = lambda a,b : math.sqrt(a \* a + b \* b)

hypot(3,4)

**Answer:**

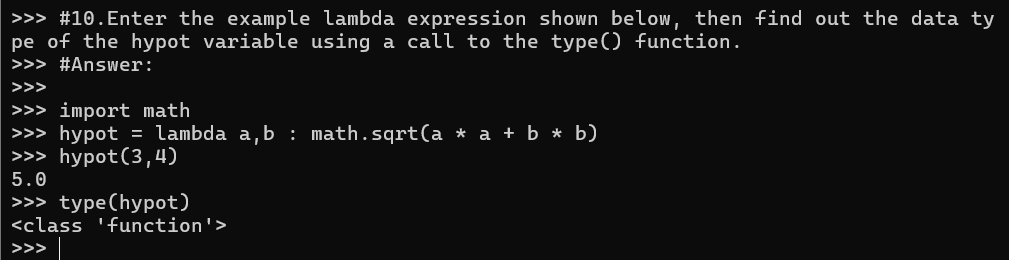
import math

hypot = lambda a,b : math.sqrt(a \* a + b \* b)

hypot(3,4)

type(hypot)

**Output of Question No. 10:**



1. Write a lambda expression that takes two formal parameters, hours and minutes. The expression should calculate and return the total number of equivalent seconds. Assign the expression to a variable called to\_seconds, then call the function several times for testing.

**Answer:**

to\_seconds = lambda hours, minutes: hours \* 3600 + minutes \* 60

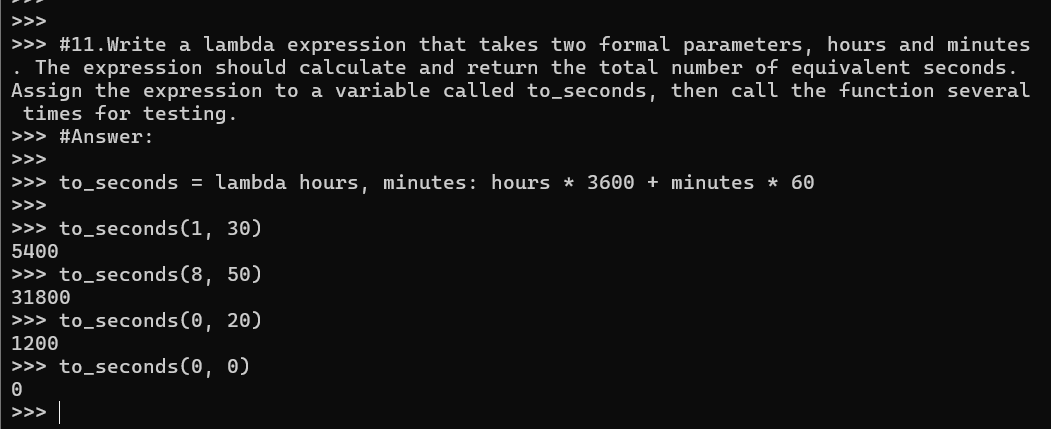
to\_seconds(1, 30)

to\_seconds(8, 50)

to\_seconds(0, 20)

to\_seconds(0, 0)

**Output of Question No. 11:**



1. Improve your previous lambda expression so that if only one argument is passed within a call, then the number of minutes defaults to 0, as detailed below:

to\_seconds(1)

3600

to\_seconds(2)

7200

**Answer:**

to\_seconds = lambda hours, minutes=0: hours \* 3600 + minutes \* 60

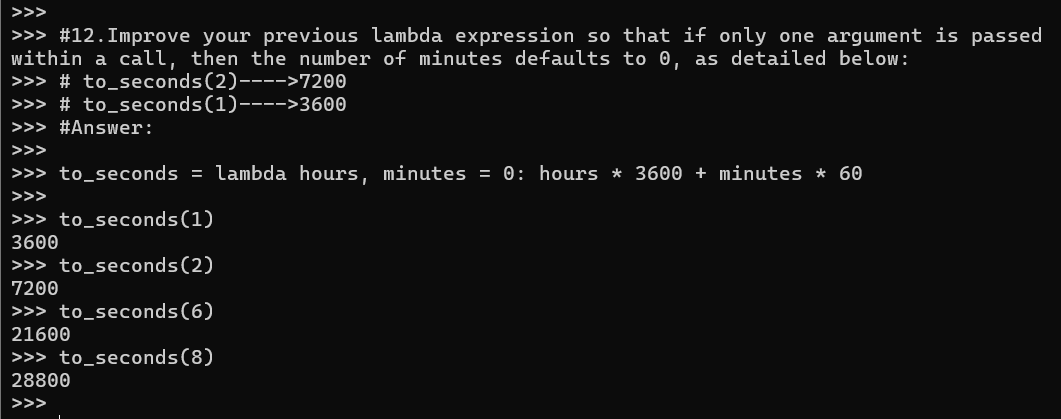
to\_seconds(1)

to\_seconds(2)

to\_seconds(6)

to\_seconds(8)

**Output of Question No. 12:**



1. Look at each of the phrases below and ensure you understand what each of these means. For any that you do not understand, do a little research to find a definition of each term. This research may involve looking back over these notes, or the associated lecture notes. It may also involve searching for these terms on the Internet.

* Module
* The Python Standard Library
* Formal Parameters
* Actual Parameters (argument values)
* Default and Keyword Arguments
* Lambda Expression

**Answer:**

* **Module:** Module is a file containing Python code that can define functions, classes, and variables, and can also include runnable code  that can be imported inside another Python Modules Operations Program.
* **The Python Standard Library:** The Python Standard Library is a collection of exact syntax, token, and semantics of Python which comes bundled with core Python distribution.
* **Formal Parameters:** Formal parameters, also known as formal arguments, are placeholders defined in the function signature or declaration.
* **Actual Parameters (argument values):** Actual parameters, also called actual arguments or arguments, are the values or expressions provided to a function or method when it is called.
* **Default and Keyword Arguments:** 
  + - **Default arguments** is the function arguments that will be used if no arguments are passed to the function call.
    - **Keyword arguments** are values that, when passed into a function, are identifiable by specific parameter names.
* **Lambda Expression:** Lambda functions are anonymous, single-expression functions defined using the lambda keyword, useful for concise operations.