# Fundamentals of Computer Programming



Submitted by Submitted to

Aarson Subba Krishna Sir

Section E 9714 BSc(Computing)

# Introduction to Programming

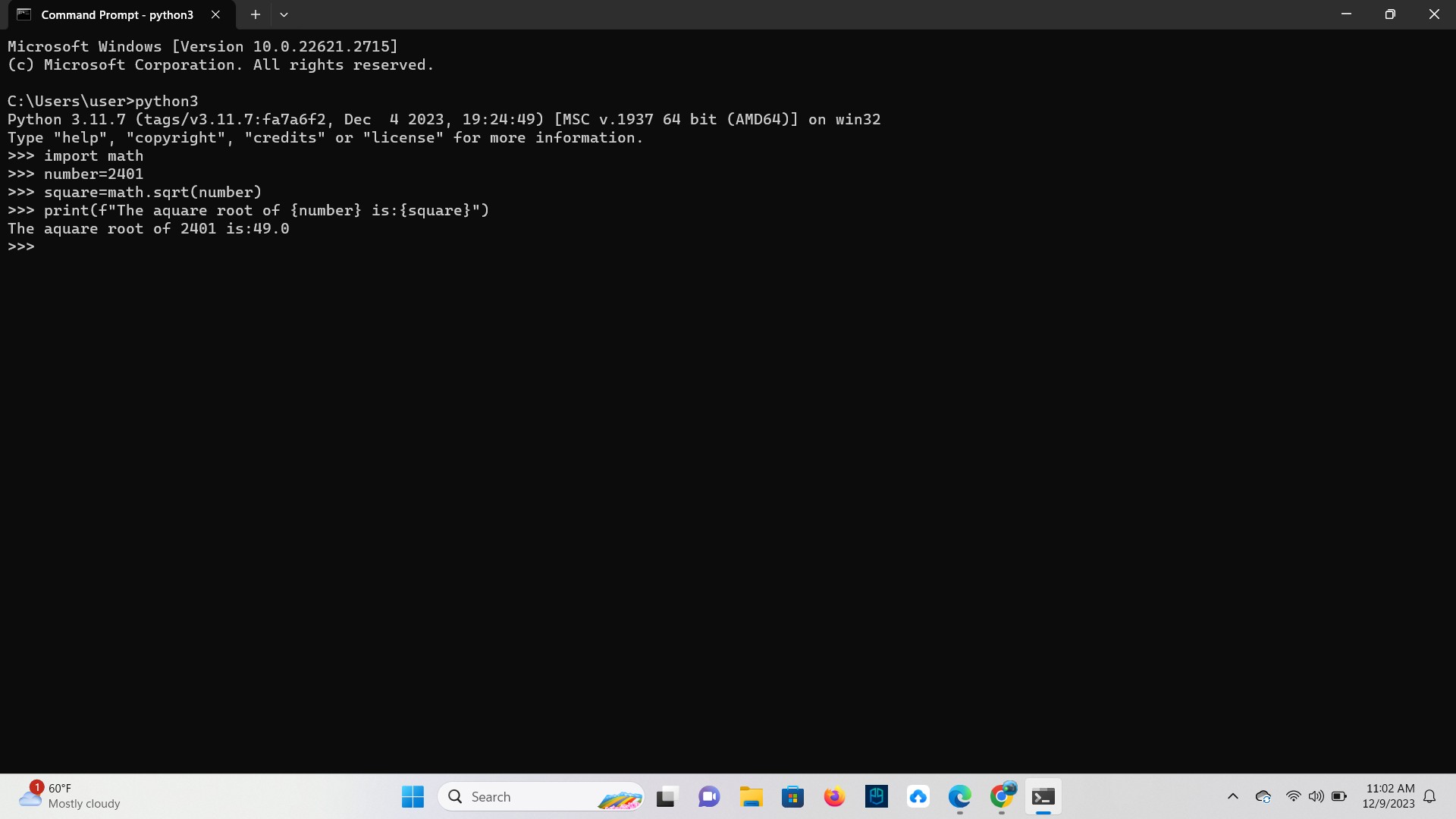
Lab Worksheet

Week 4

## Importing and Using Functions

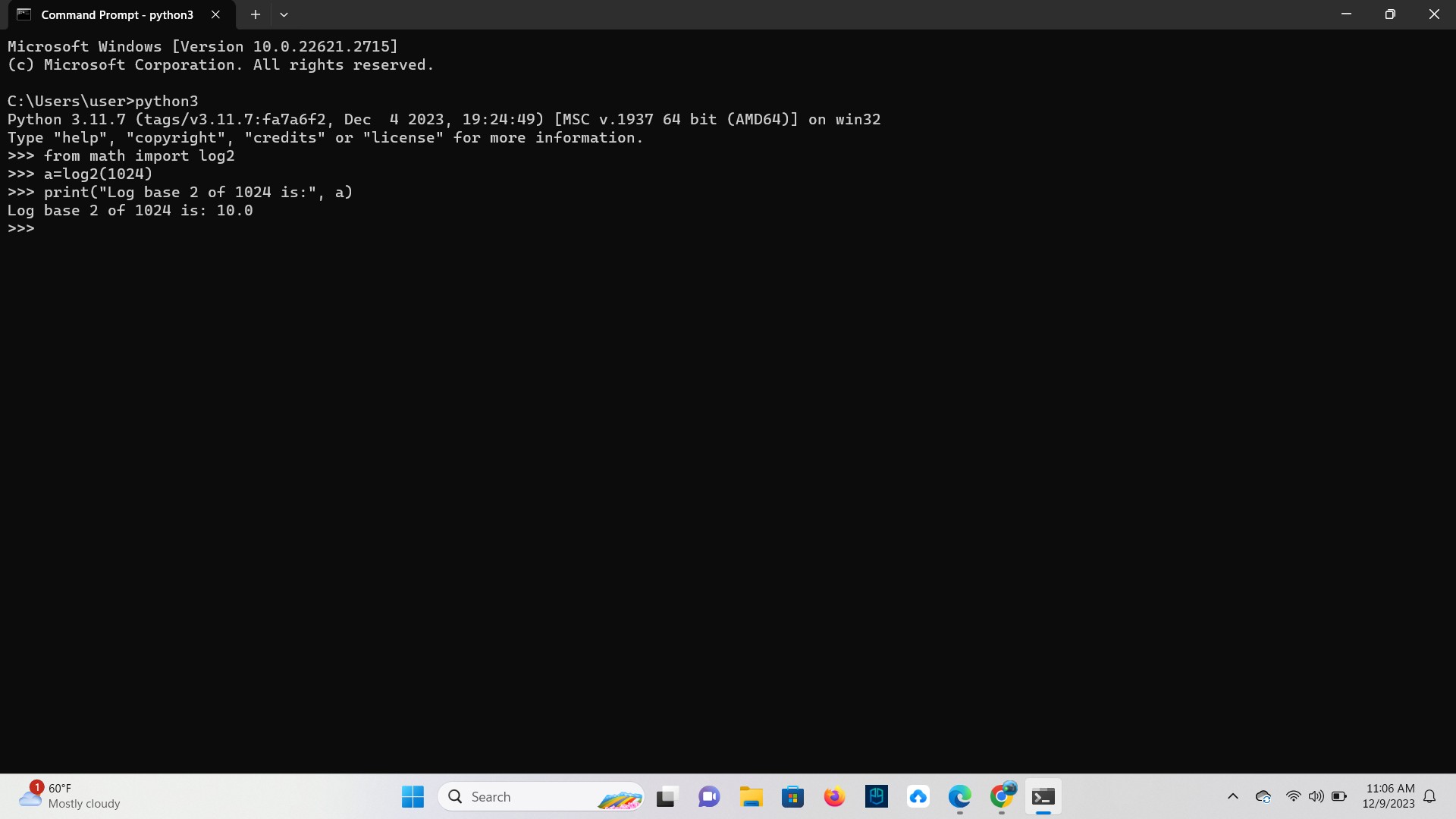
**TASK**: Write some code that imports the math module, then calculates and prints the square root of the number 2401. Use the sqrt () function provided by the math module.

**Ans**



**TASK**: 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.

**Ans**

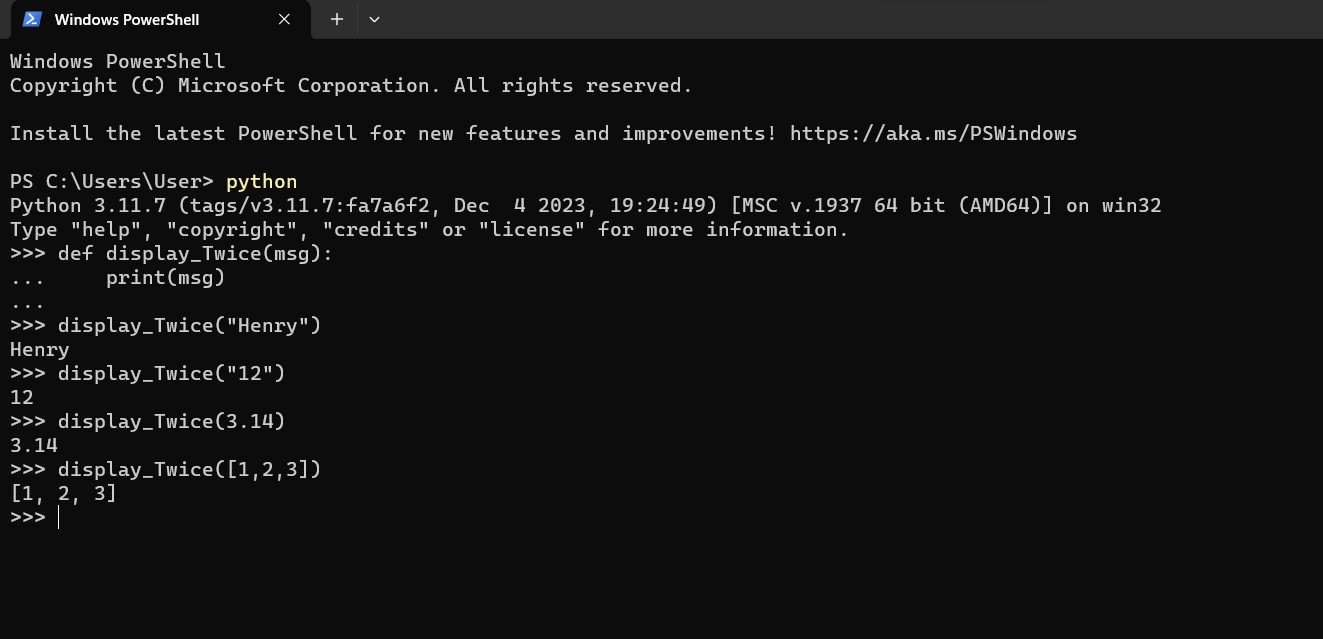


## Defining Functions

def display\_Twice(msg): print(msg) print(msg)

**TASK**: Input the above function definition. Once that is done make several calls to the function passing different argument values for testing.

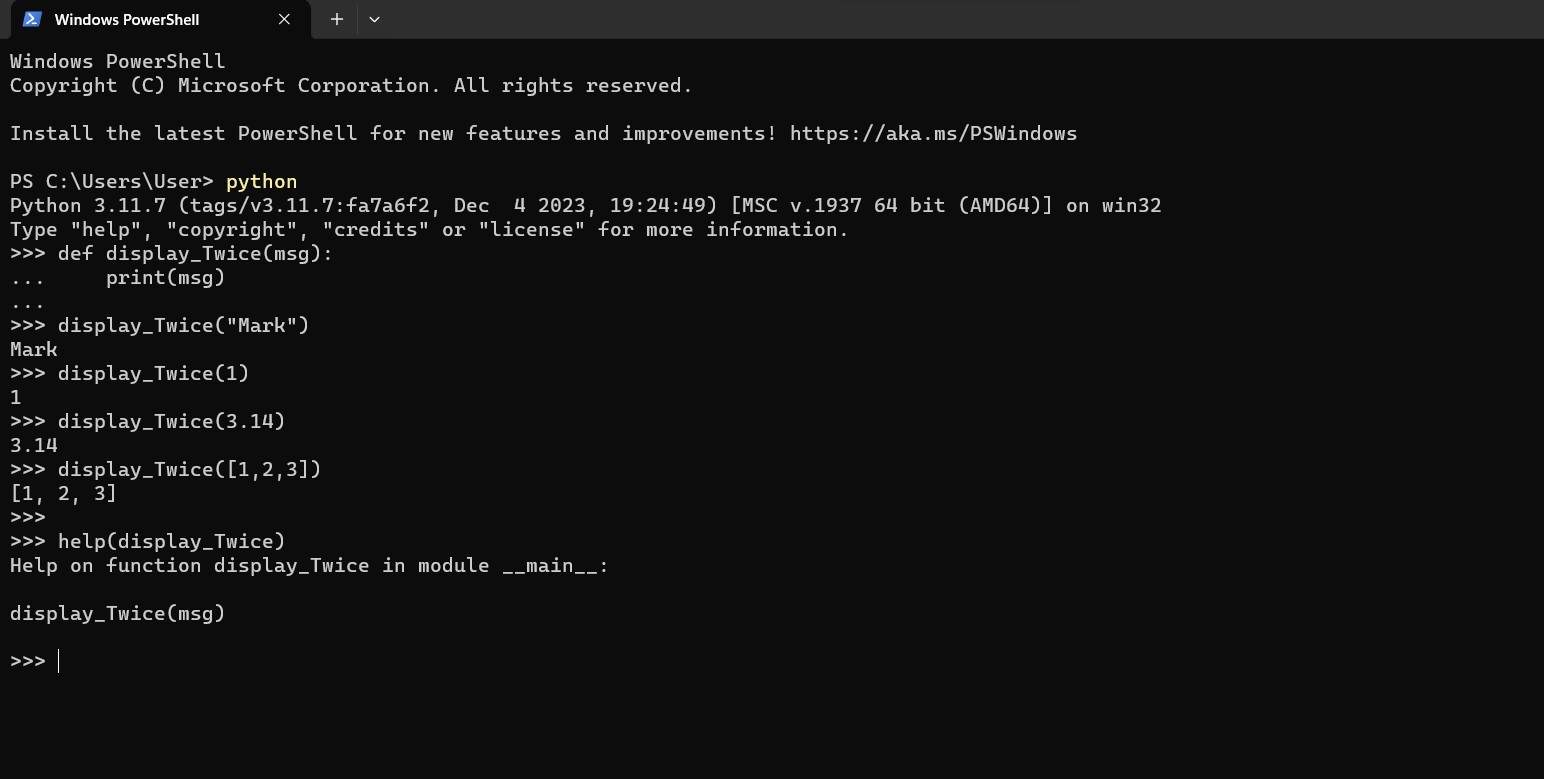
**Ans**



## Docstrings

**TASK**: 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 (display Twice) and see what it displays.

**Ans**



## Formal and Actual Parameters

When a function is being defined, the arguments we specify are usually referred to as the formal parameters of the function. In the above example the formal parameter is called msg. When a function is being called, the argument values provided are called the actual parameters. The formal parameter names act like local variables within the function and can only be accessed within that specific function block. Any variables declared within the function block can also only be accessed from within that block and cease to exist once the function ends. This idea of local-scope means functions do not have to worry about using variable names that may already exist elsewhere in the program or within other function

## Returning a value

def findMax(a,b):

"""Finds the maximum of two values.""" if ( a > b ):

max = a

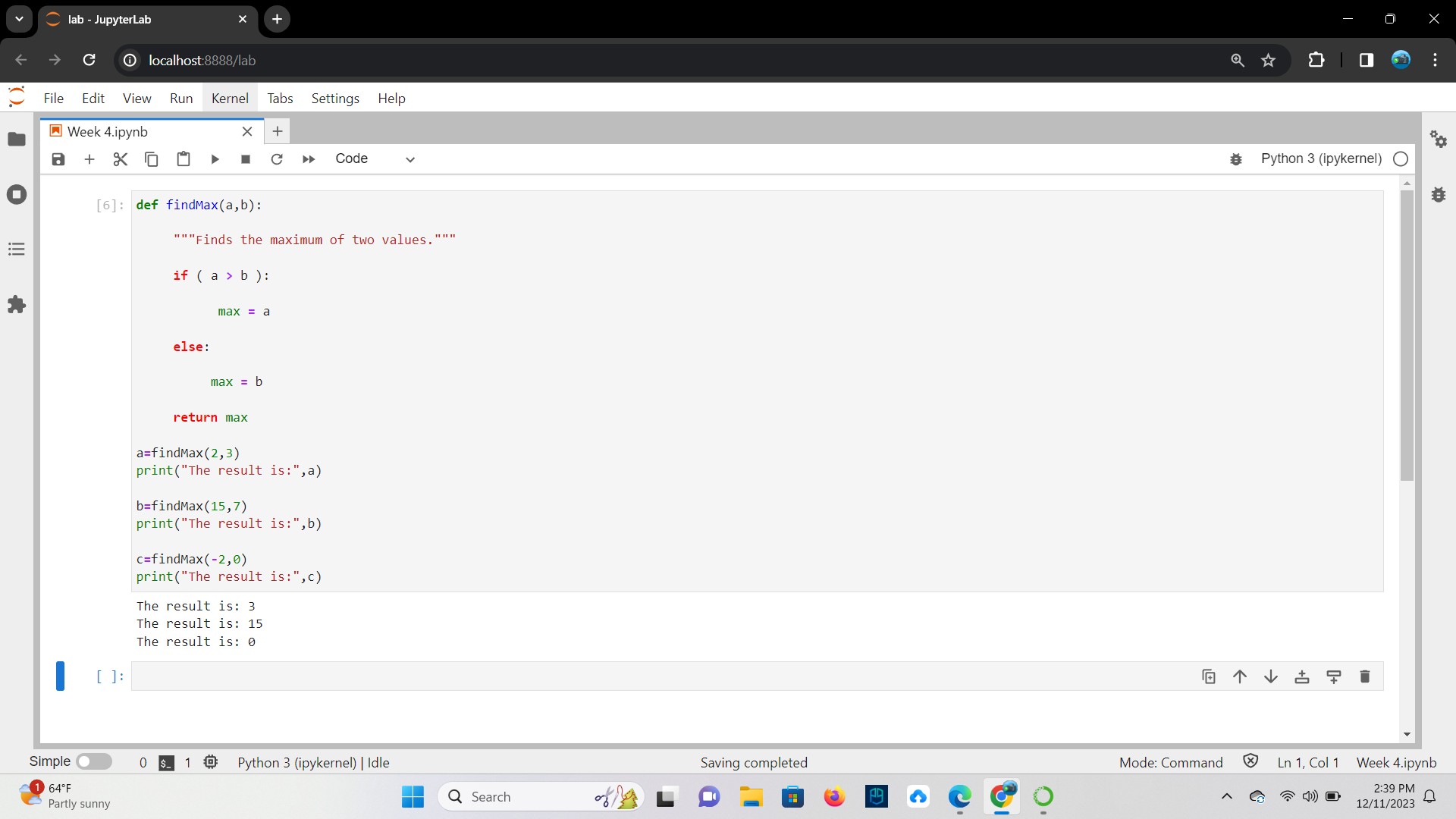
else:

max = b

return max

**TASK**: Input the above function definition. Once that is done make several calls to the function passing different argument values and displaying the returned value.

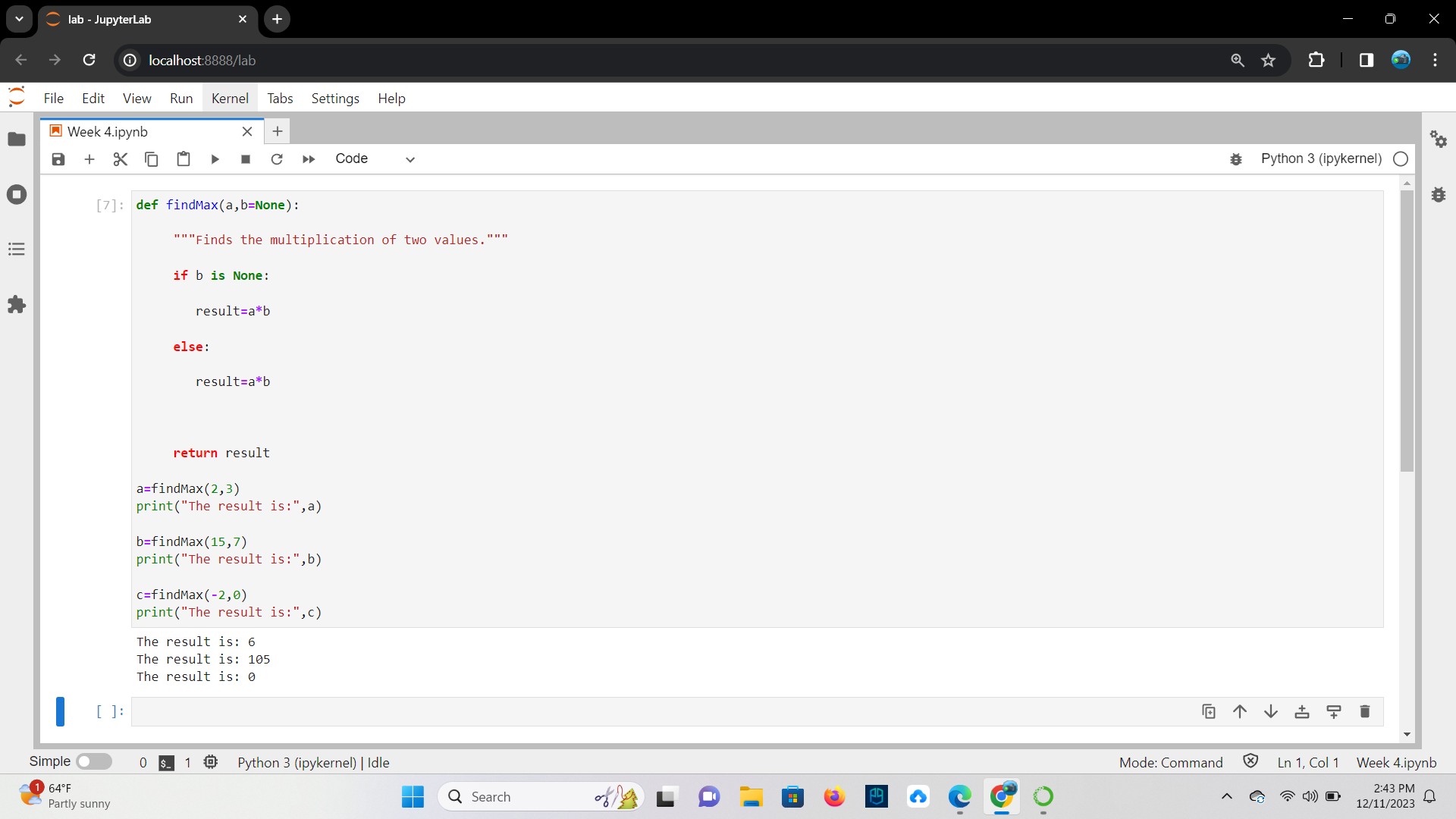
**Ans**



## Default Arguments

**TASK**: 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.

**Ans**



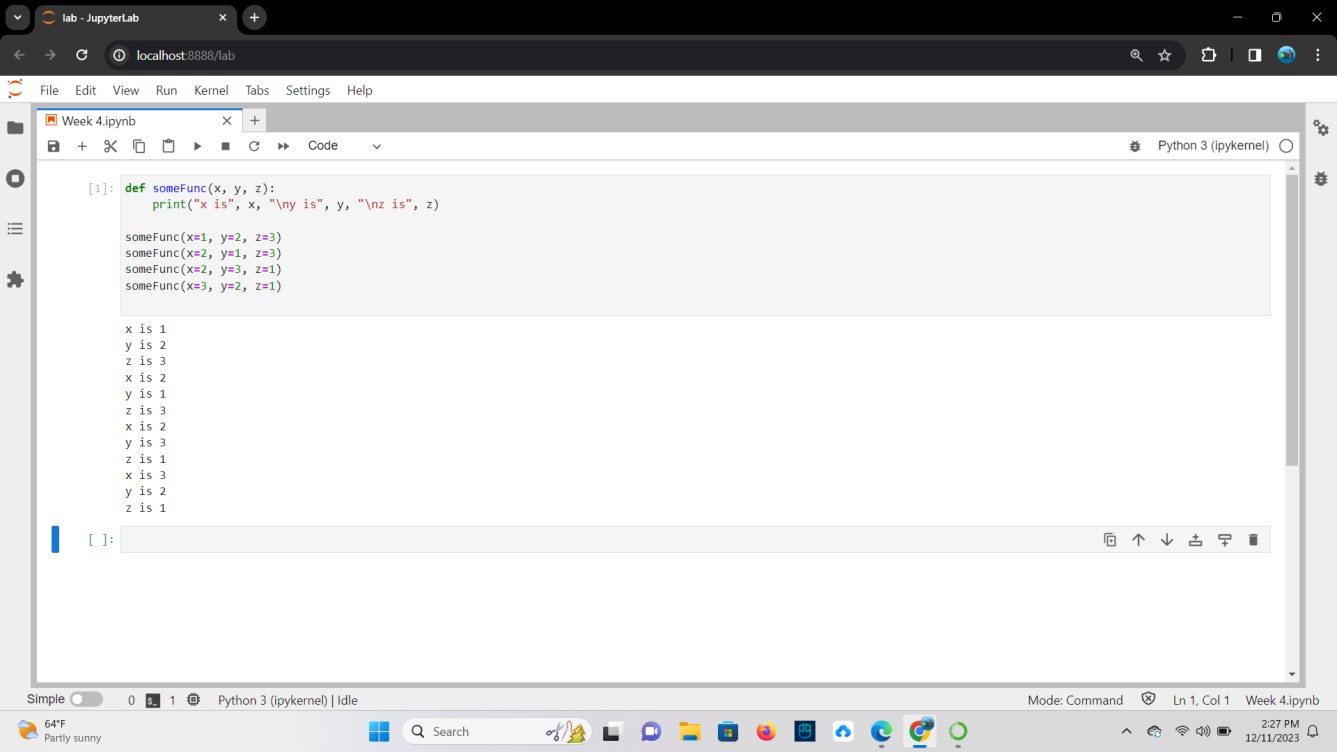
## Keyword Arguments

def someFunc(x, y, z):

print("x is", x, "\ny is", y, "\nz is", z)

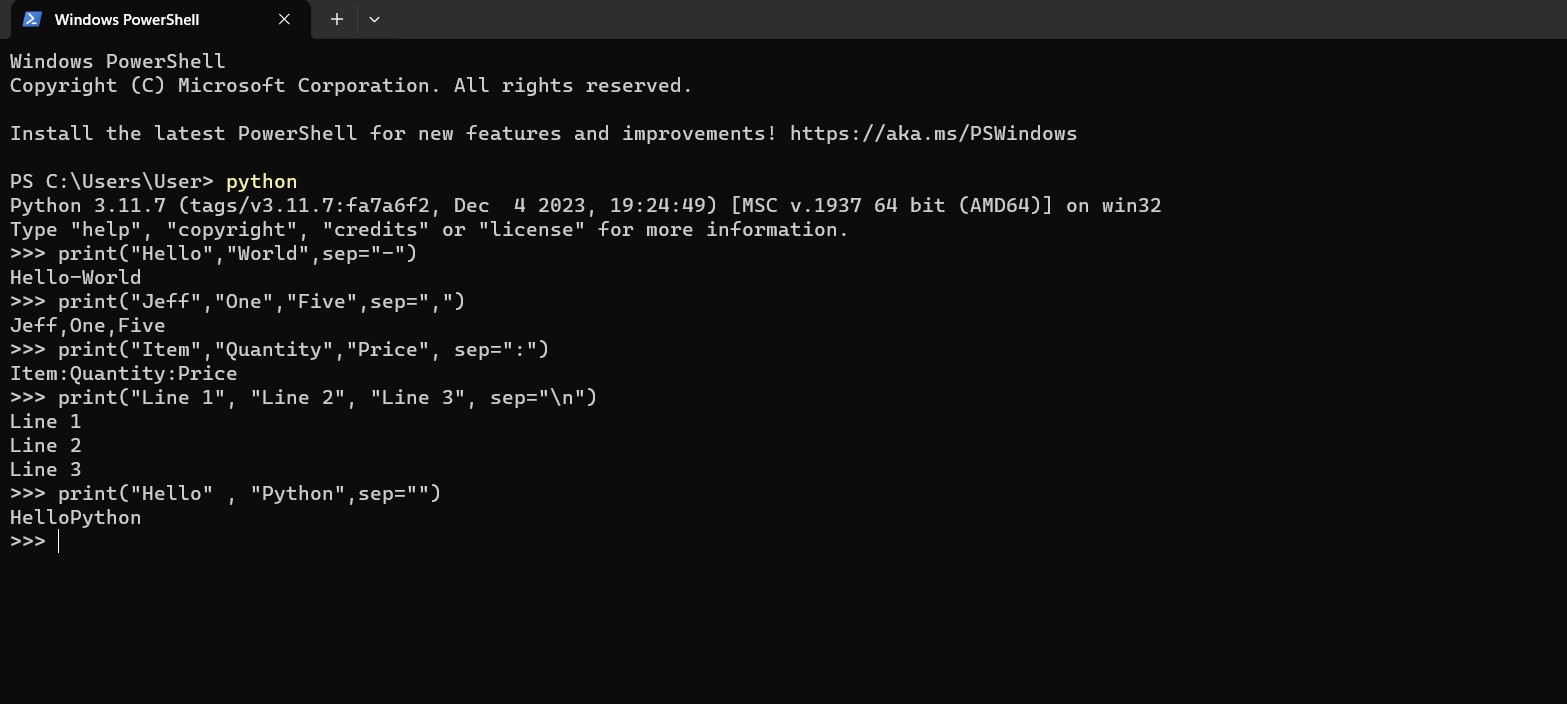
**TASK**: Enter the example function shown above, then try calling it using the keyword arguments in several different orders.

**Ans**



**TASK**: 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)

**Ans**



## Arbitrary Length Argument Lists

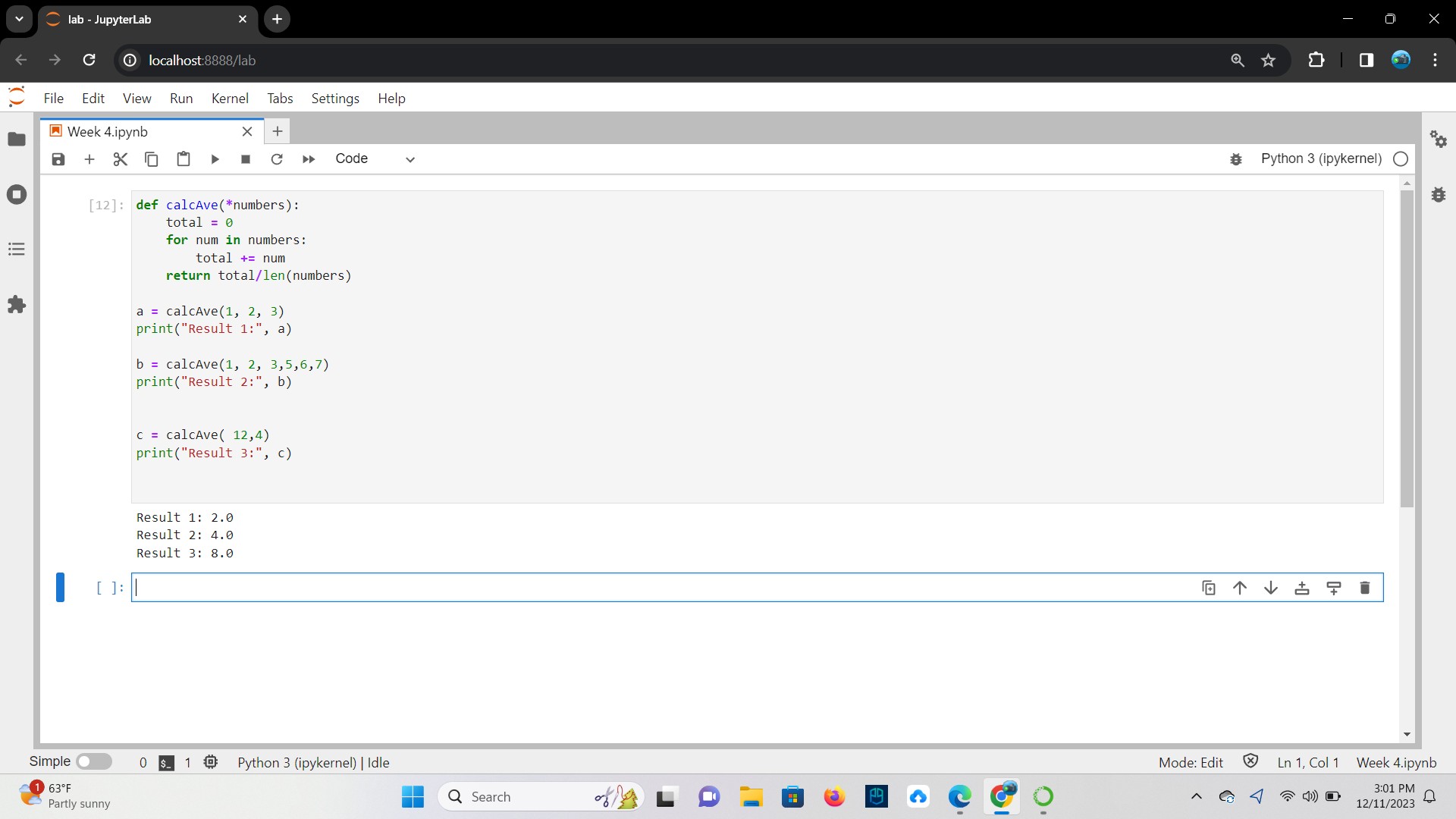
def calcAve(\*numbers): total = 0

for num in numbers: total += num

return total/len(numbers)

**TASK**: Enter the example function shown above, then try calling it several times, passing a different number of numeric arguments each time. Note: variadic arguments are normally defined last in the formal parameter list (and can only be followed by keyword type parameters).

**Ans**

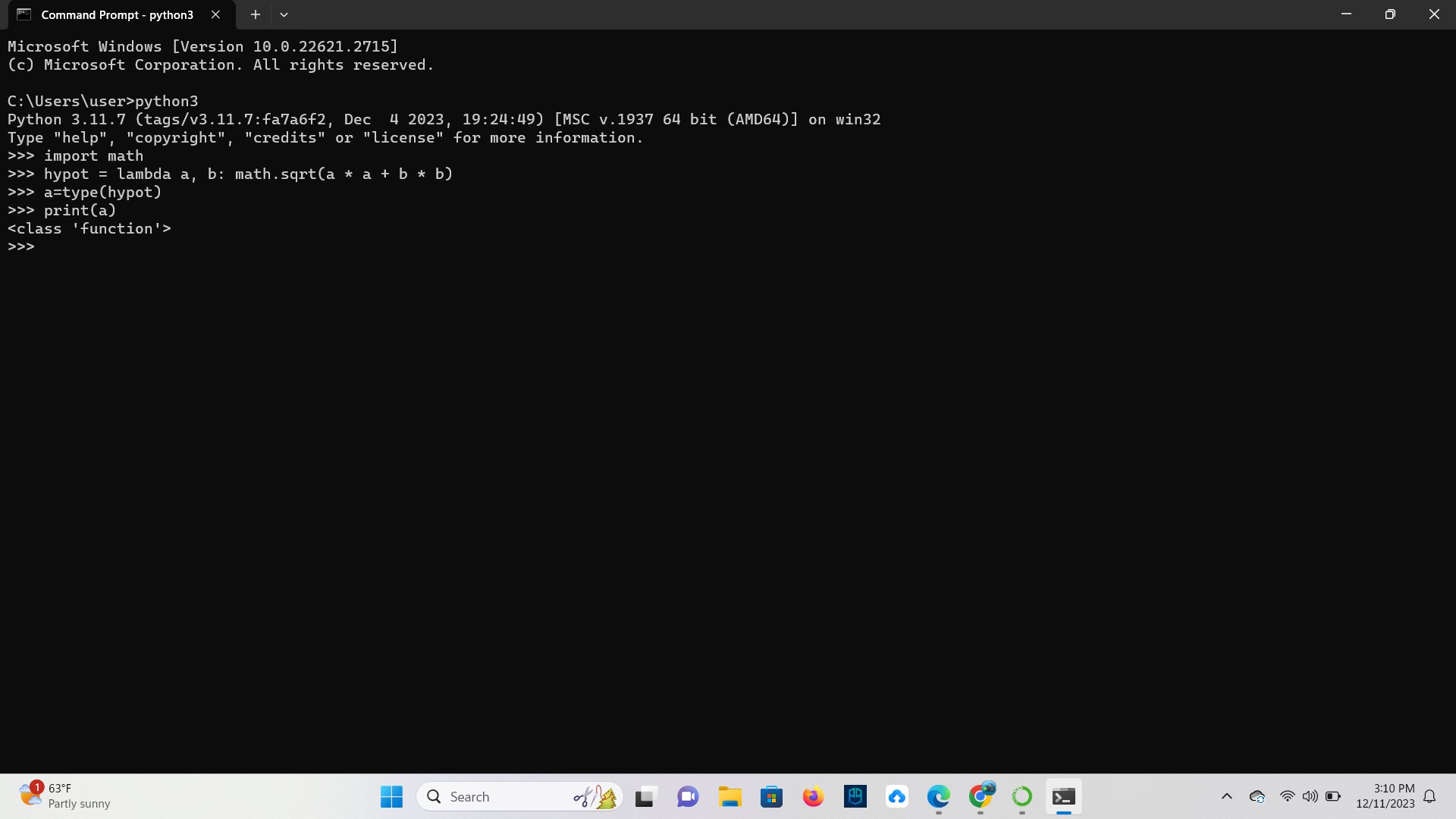


## Lambda Expressions

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

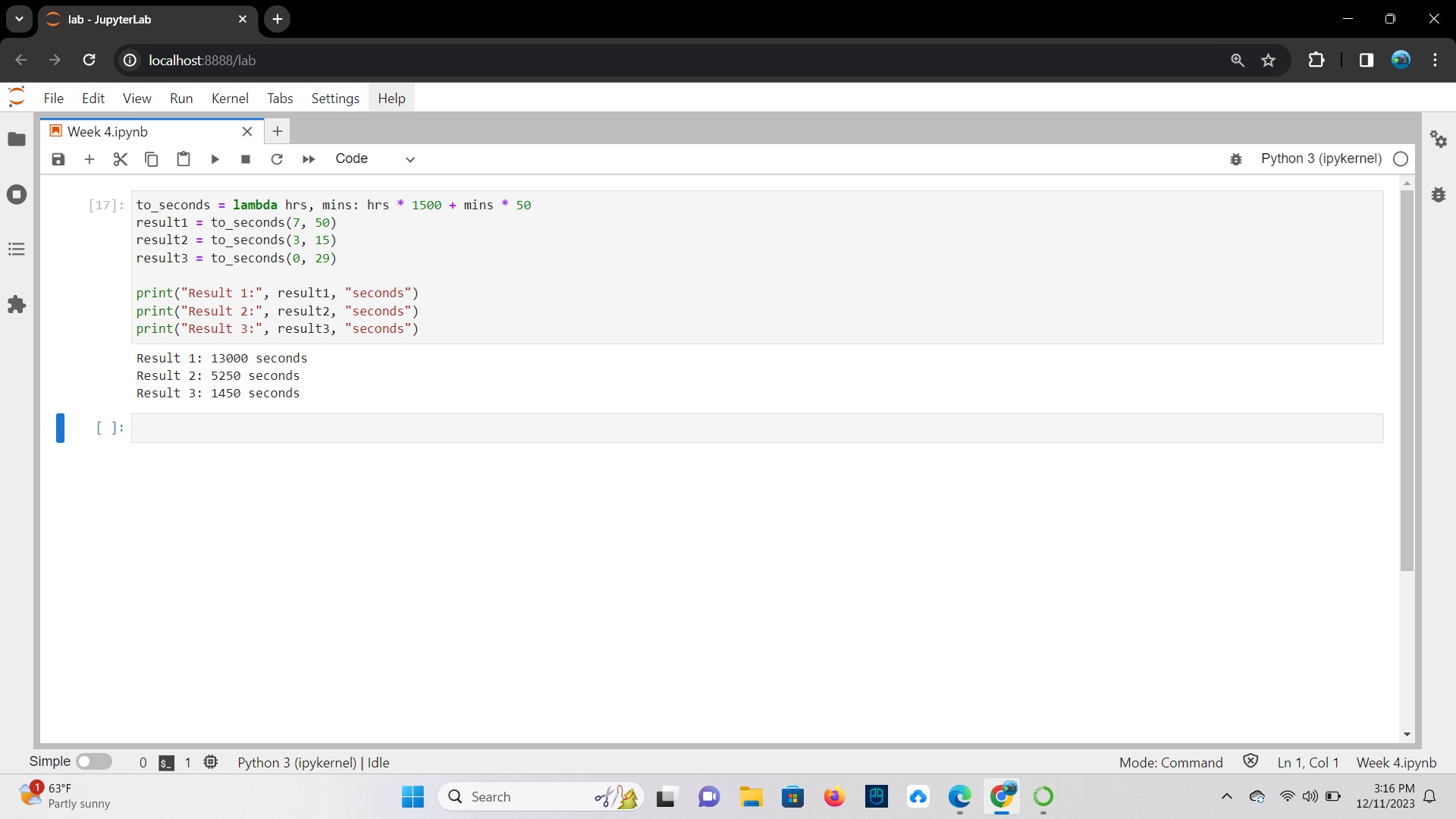
**TASK**: Enter the example lambda expression shown above, then find out the data type of the hypot variable using a call to the type () function. Notice the result.

**Ans**



**TASK**: 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.

**Ans**

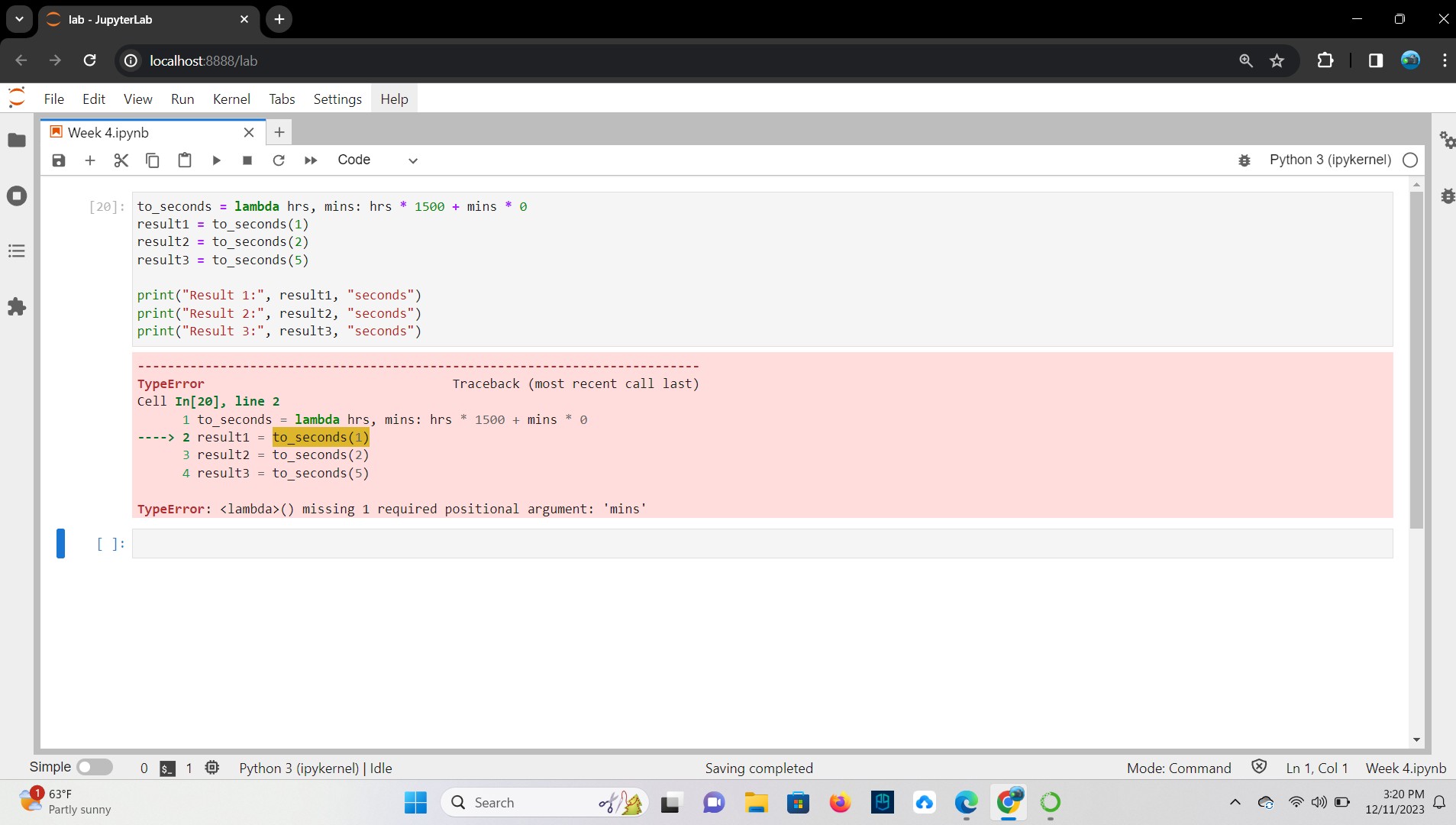


**TASK**: Improve your previous lambda expression so that if only one argument is passed within a call, then the number of minutes defaults to 0, as detailed below:

>>> to\_seconds(1) 3600

>>> to\_seconds(2) 7200

**Ans**



## Key Terminology

**TASK**: 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

**Ans** They are simply file with .py extension containing python code that can be imported inside another python program.

* The Python Standard Library

**Ans** The Python Standard Library is a collection of script modules accessible to a Python program.

* Formal Parameters

**Ans** Formal parameters are the variables defined by the function that receives values when the function is called.

* Actual Parameters (argument values)

**Ans** the actual value that is passed into the method by a caller.

* Default and Keyword Arguments

**Ans** Default Arguments in Python represent 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

Ans A lambda expression is a short block of code which takes in parameters and returns a value.