Fundamentals of Computer Programming

Submitted by

Rishav Rauniyar

Section E ID No 8711

Bsc(Computing)

Level 4 1st Semester

Submitted to

Krishna Sir

Introduction to Programming

Lab Worksheet

Week 5

Writing code in files

The Python interpreter allows code to be entered interactively in Read-Evaluate-Print and Loop (REPL mode) or executed directly from a text file. REPL mode is mainly used to write very small pieces of code for learning and experimentation purposes. Larger more complex programs need to be stored in one or more text files, then executed later. Such Python programs are often called scripts. Placing code in text files allows the code to grow and evolve over time. It also provides an easy mechanism of re-running programs over and over again. The files are simple plain text files and there is nothing special about them other than the name which must include the .py suffix. The program code within the files is exactly the same as the code that can be typed in REPL mode. However, there is one important difference between executing statements in interactive-mode compared to from a file. When in REPL mode the result of expressions is automatically printed to the screen (this is the 'P' within REPL). Such output is not done automatically when executing scripts, and an explicit call to the print() function is required to generate output. Text files will often contain many thousands of lines of code. We can also define functions and classes within our files. These may be reused across several different programs. i.e. we can import the same content (to share the same functionality) between different programs.

Choosing an Editor

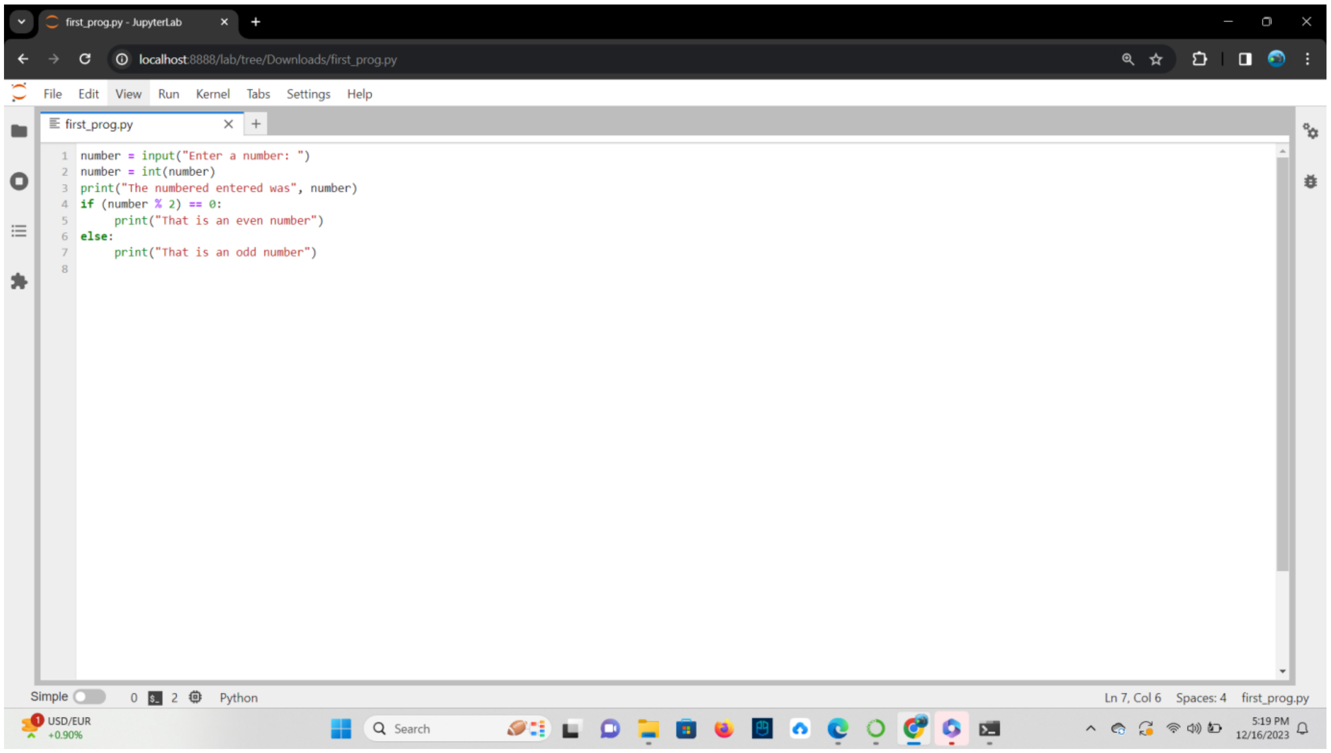
The contents of the scripts can be produced in many ways, given that they are simply plain text files. Any generic editor such as 'notepad' on Windows, or 'vi' on Linux can be used. Many professional programmers tend to use Integrated Development Environments (IDEs). These include text editors that provide additional features such as syntax colouring, auto-completion, and automatic fixes to syntax errors. They also provide a way to directly execute and debug a program from within the same environment. Popular IDEs that support Python development include PyCharm, IDLE, Eclipse and IntelliJ. These lessons focus on the Python language rather than the chosen development environment, so everything discussed can be done using either a basic text editor, or via a fully blown professional IDE. That being said, it is useful to know how to develop and execute code manually without the use of an IDE, so a better understanding of how the whole process actually works can be gained.

Executing a Script

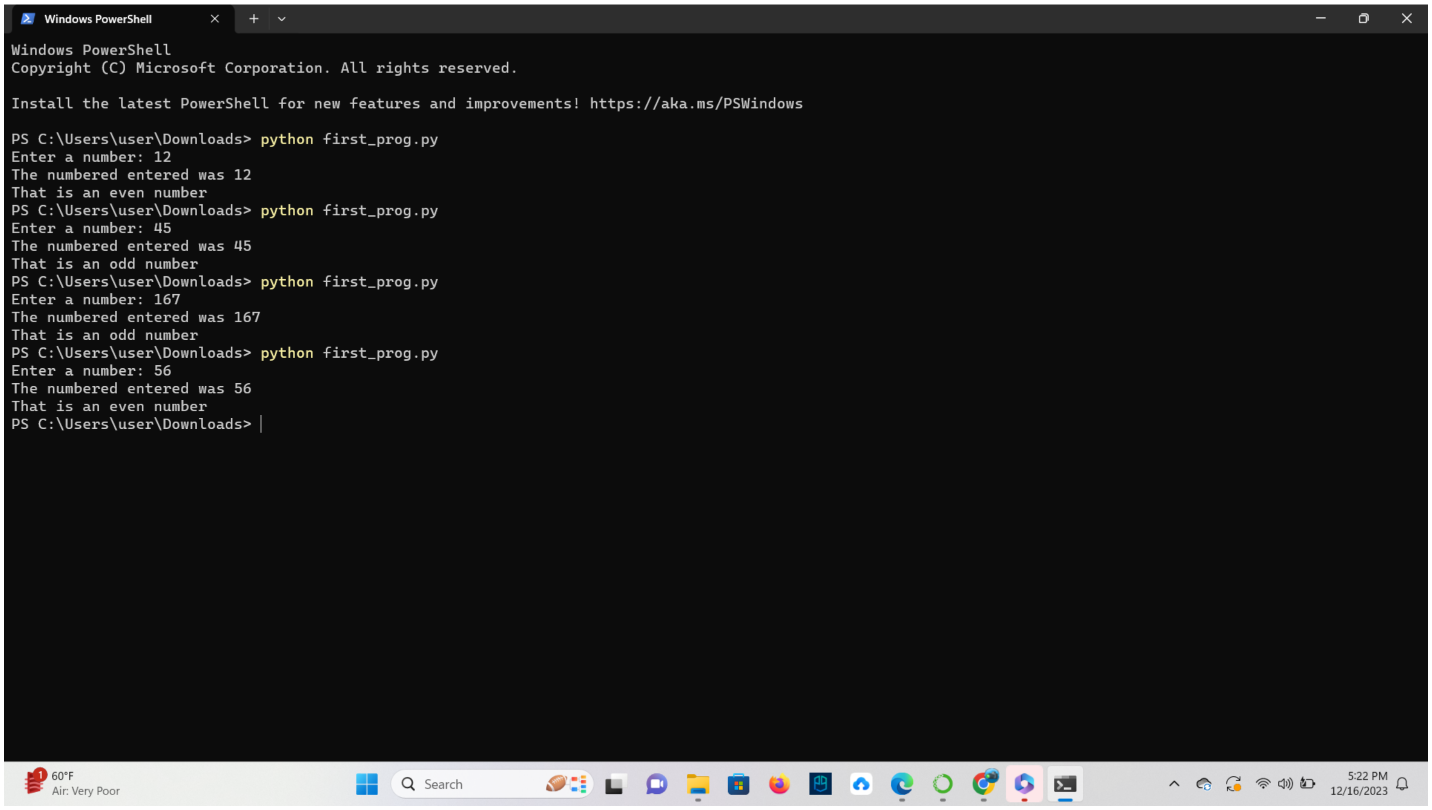
TASK: Use an editor to input the Python program shown below then save it to a file called first_prog.py. Once that has been done, execute the program from the command line.

```
number = input("Enter a number: ")
number = int(number)
 print("The numbered entered was", number)
 if (number % 2) == 0:
        print("That is an even number")
 else:
        print("That is an odd number")
```

Ans



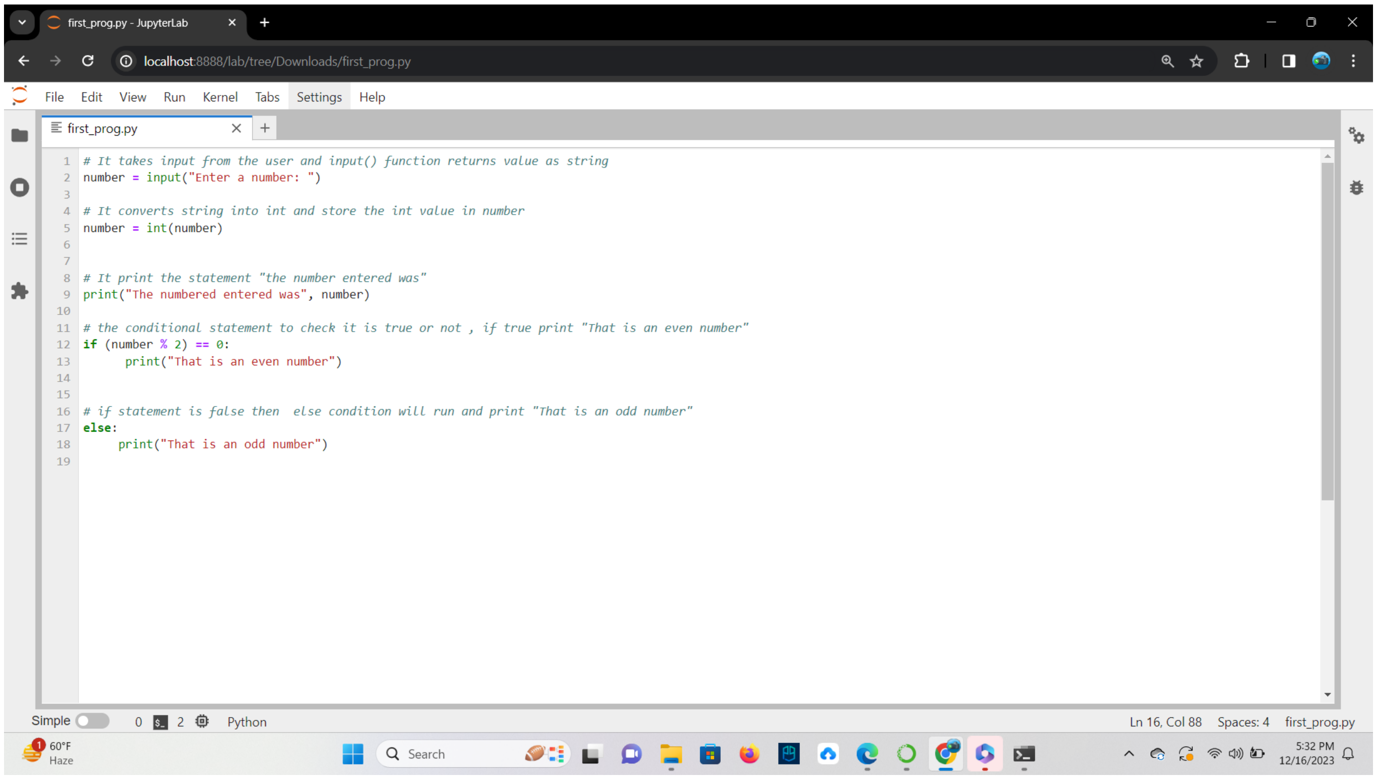
```
number = input("Enter a number: ")
number = int(number)
print("The numbered entered was", number)
if (number % 2) == 0:
    print("That is an even number")
else:
    print("That is an odd number")
```



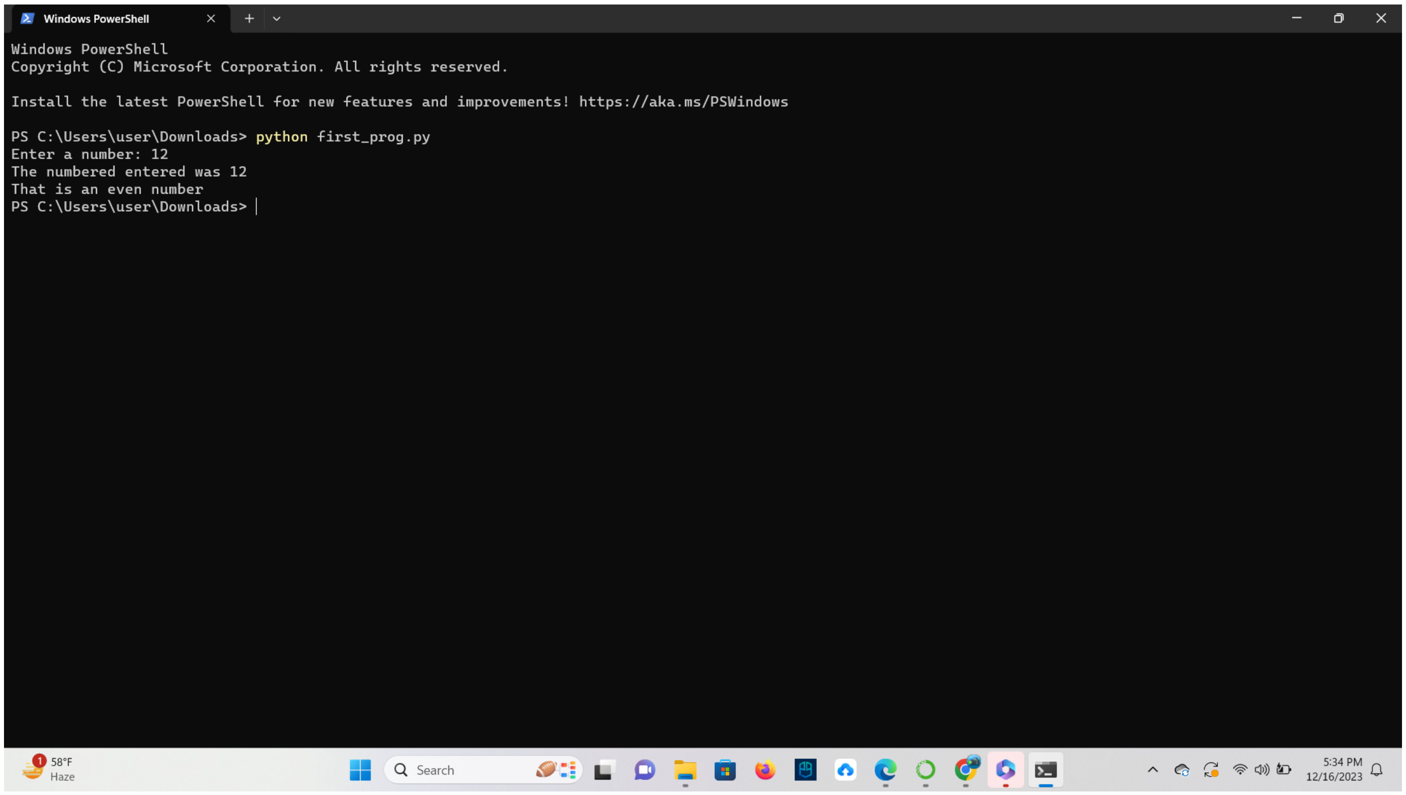
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python first_prog.py
Enter a number: 12
The numbered entered was 12
That is an even number
PS C:\Users\user\Downloads> python first_prog.py
Enter a number: 45
The numbered entered was 45
That is an odd number
PS C:\Users\user\Downloads> python first_prog.py
Enter a number: 167
The numbered entered was 167
That is an odd number
PS C:\Users\user\Downloads> python first_prog.py
Enter a number: 56
The numbered entered was 56
That is an even number
PS C:\Users\user\Downloads>
```

TASK: Open the first_prog.py and add comments above each statement within the file describing what that statement does (using a '#' at the beginning of the line). Save the file and execute it again for testing purposes.

Ans



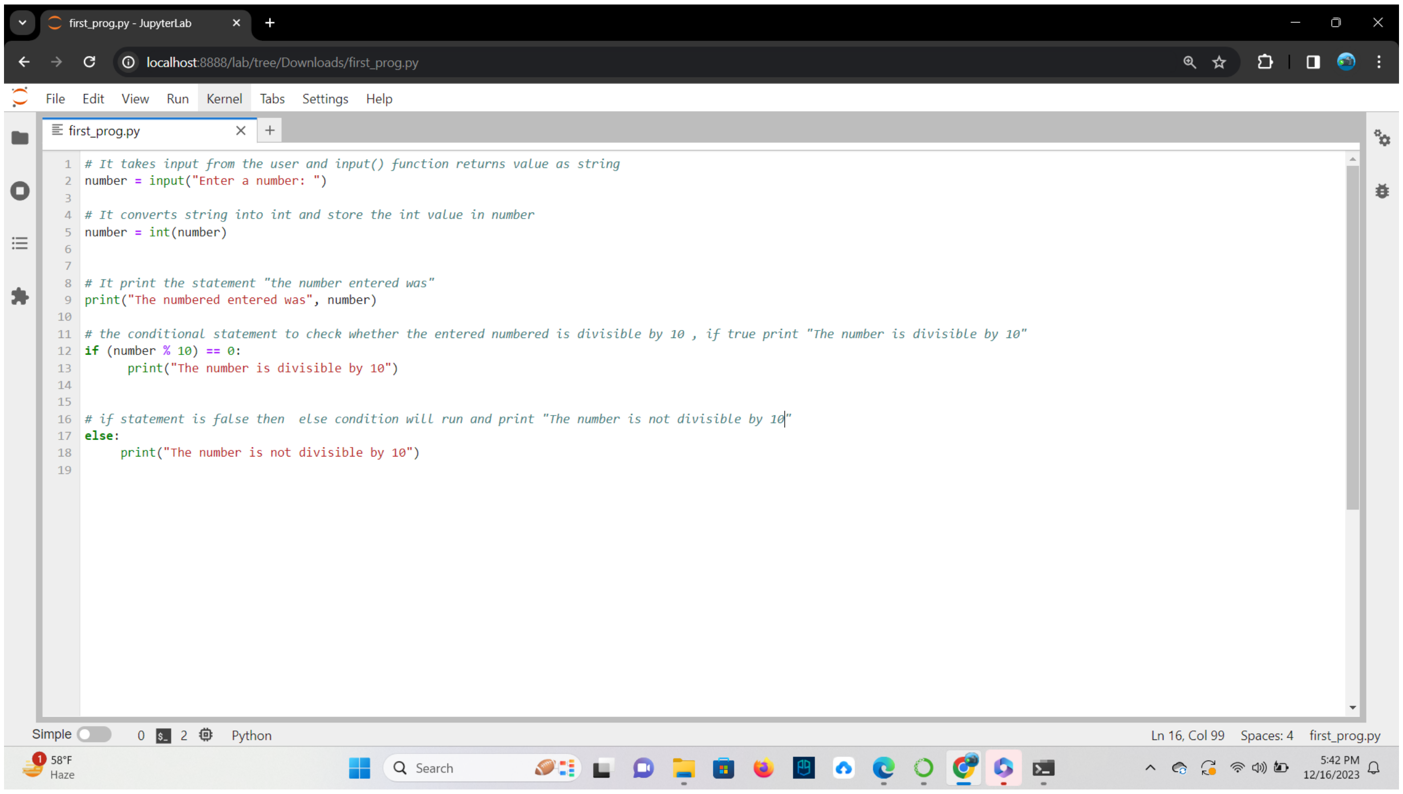
```python
# It takes input from the user and input() function returns value as string
number = input("Enter a number: ")

# It converts string into int and store the int value in number
number = int(number)


# It print the statement "the number entered was"
print("The numbered entered was", number)

# the conditional statement to check it is true or not , if true print "That is an even number"
if (number % 2) == 0:
      print("That is an even number")


# if statement is false then  else condition will run and print "That is an odd number"
else:
     print("That is an odd number")
```



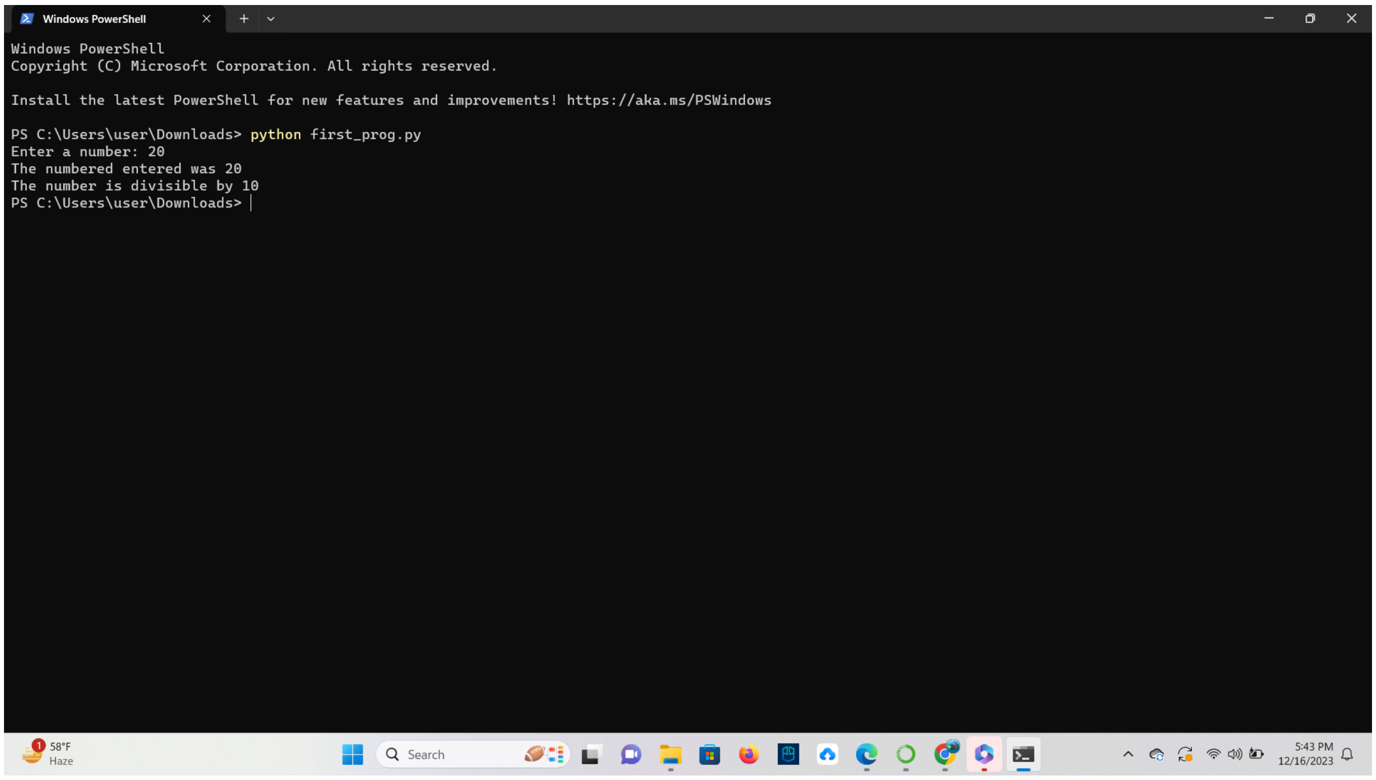
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python first_prog.py
Enter a number: 12
The numbered entered was 12
That is an even number
PS C:\Users\user\Downloads>
```

TASK: Open the first_prog.py and add some extra code that identifies and prints a message stating whether the entered number is divisible by 10. You should be able to base the new code on the if statement already provided. Once completed, save the file and execute it again for testing.

Ans



```python
# It takes input from the user and input() function returns value as string
number = input("Enter a number: ")

# It converts string into int and store the int value in number
number = int(number)


# It print the statement "the number entered was"
print("The numbered entered was", number)

# the conditional statement to check whether the entered numbered is divisible by 10 , if true print "The number is divisible by 10"
if (number % 10) == 0:
      print("The number is divisible by 10")


# if statement is false then  else condition will run and print "The number is not divisible by 10"
else:
     print("The number is not divisible by 10")

```



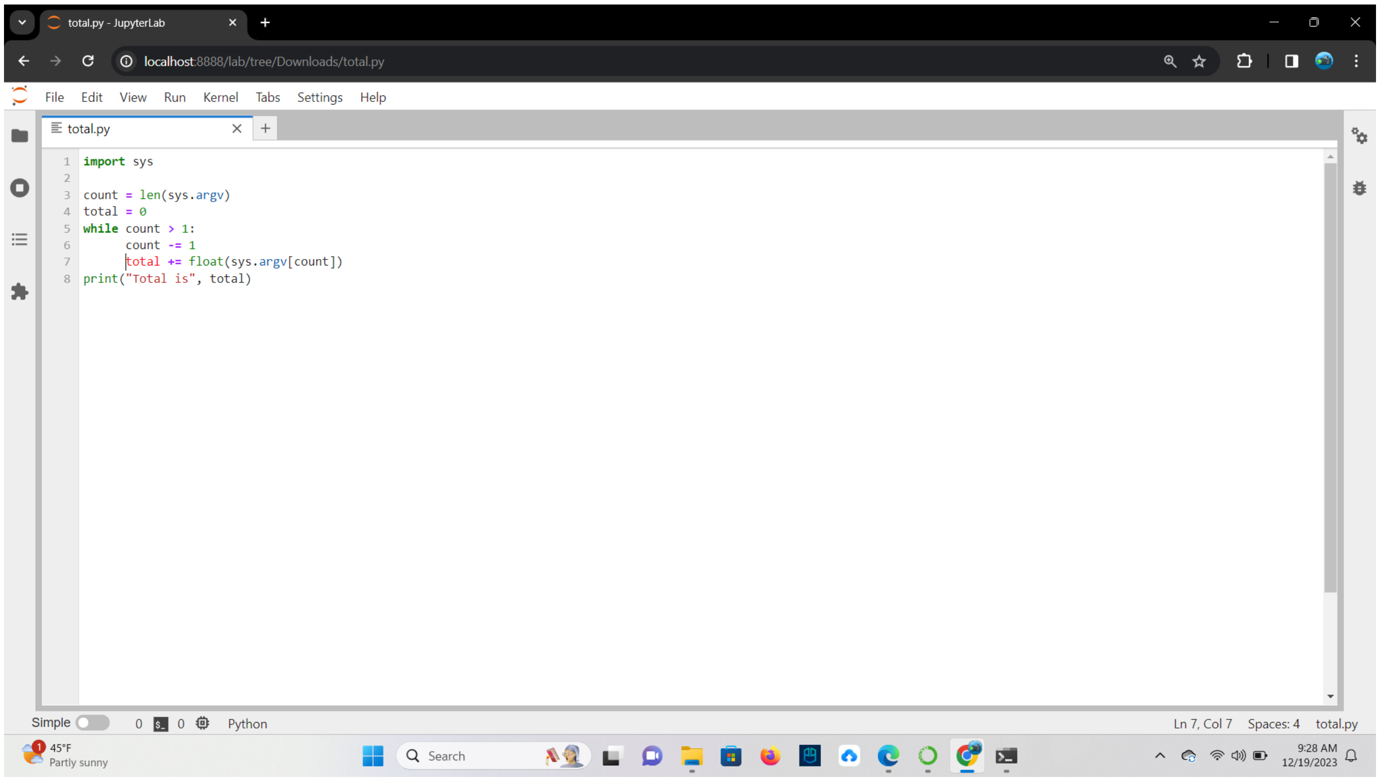
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python first_prog.py
Enter a number: 20
The numbered entered was 20
The number is divisible by 10
PS C:\Users\user\Downloads>
```

Command Line Arguments

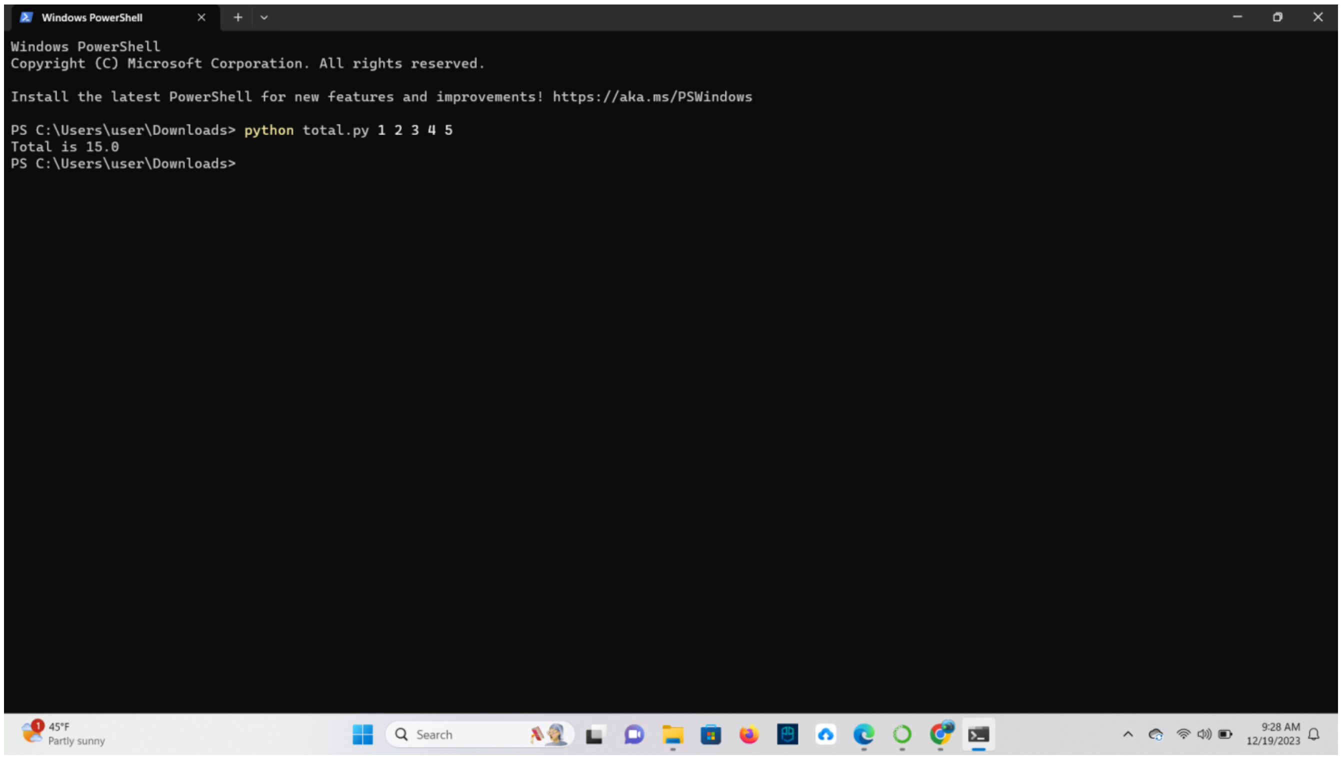
TASK: Use an editor to input the Python program shown below then save it to a file called total.py. Once that has been done, execute the program from the command line, passing several numeric values for testing.

```
import sys
count = len(sys.argv)
total = 0
while count > 1:
    count -= 1
    total += float(sys.argv[count])
print("Total is", total)
```

Ans



```python
import sys

count = len(sys.argv)
total = 0
while count > 1:
      count -= 1
      total += float(sys.argv[count])
print("Total is", total)
```

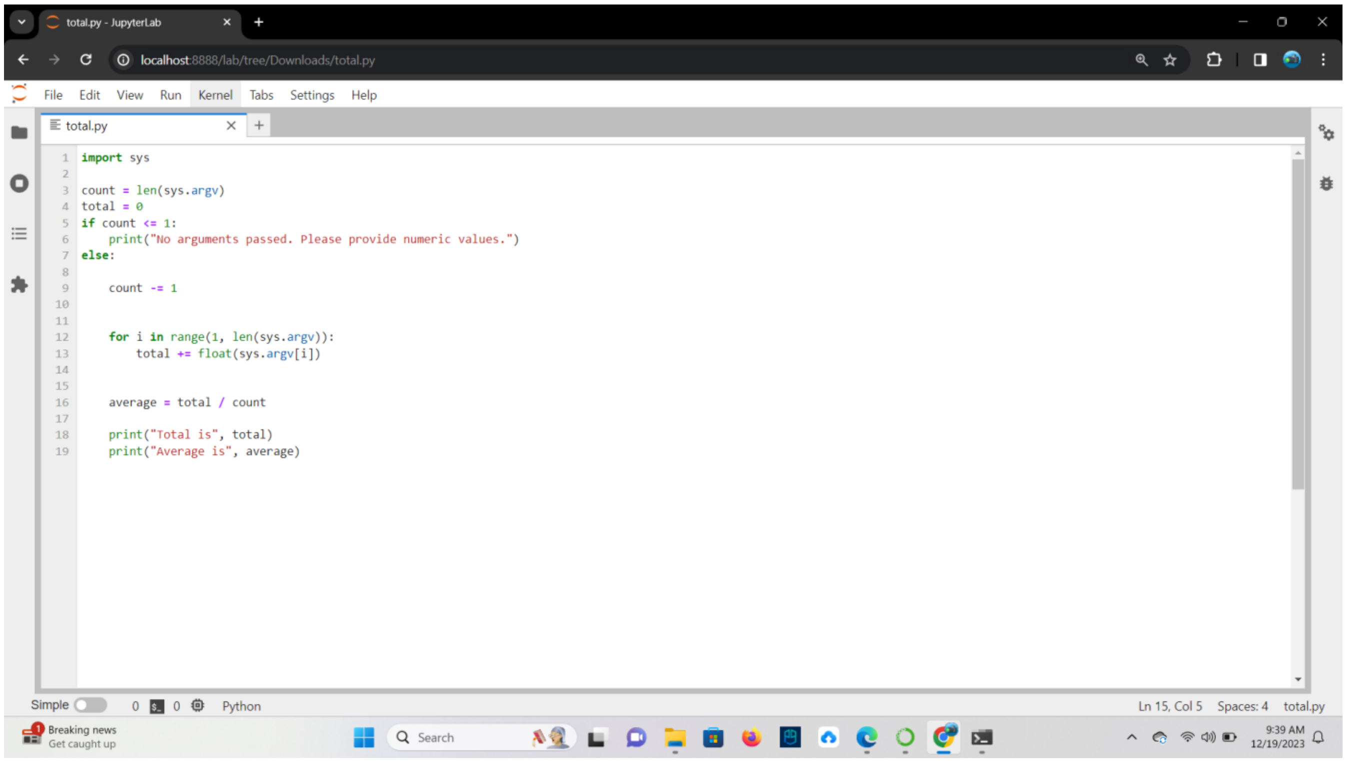


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python total.py 1 2 3 4 5
Total is 15.0
PS C:\Users\user\Downloads>
```

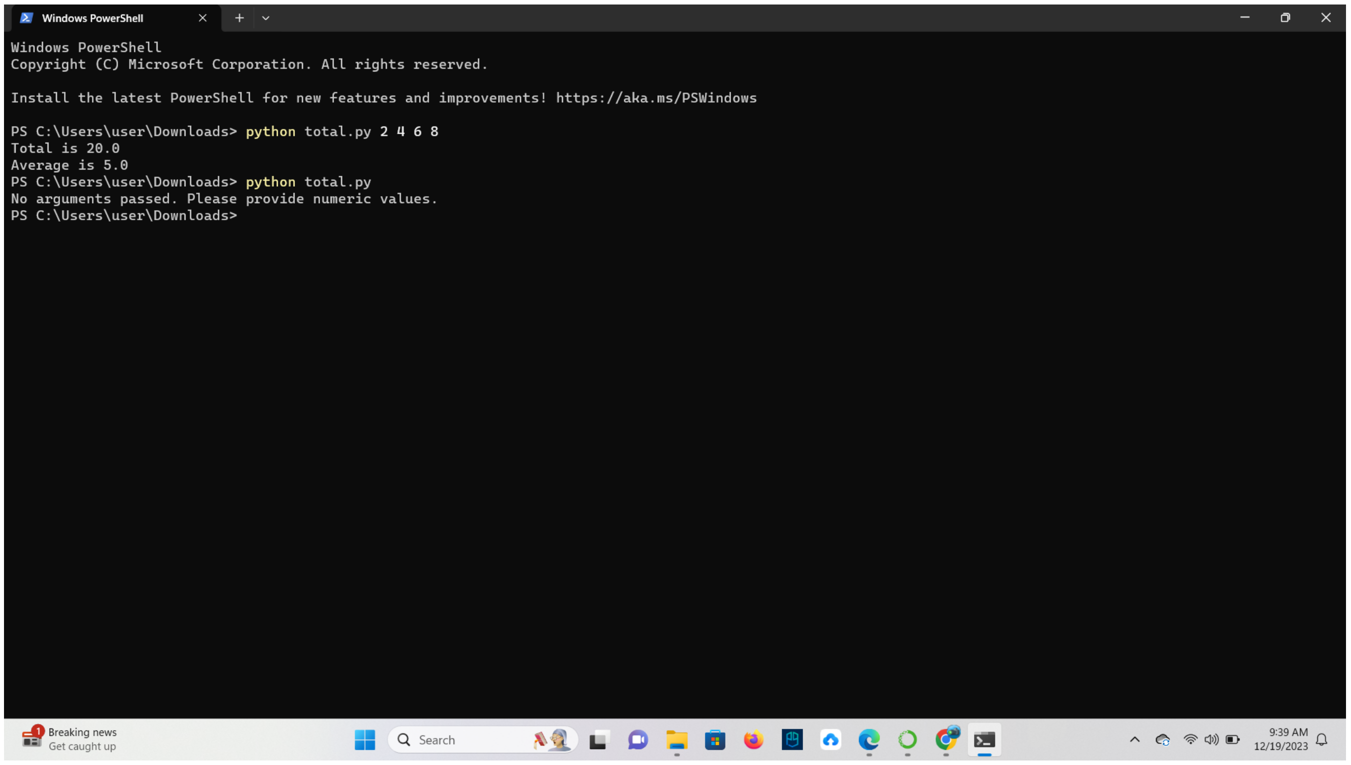
TASK: Improve the previous program by adding additional code that not only prints the total of any passed arguments, but also calculates and prints the average. Execute the program several times for testing. What happens if no arguments are passed?

Remember, the first element of the sys.argv list is the program name itself, so it is not uncommon to ignore or skip processing of that first element.

Ans If no arguments are passed, then in the code if condition will be true and the output will be “No arguments passed. Please provide numeric values.”


total.py - JupyterLab
localhost:8888/lab/tree/Downloads/total.py
File Edit View Run Kernel Tabs Settings Help
total.py
Simple
Python
Ln 15, Col 5 Spaces: 4 total.py

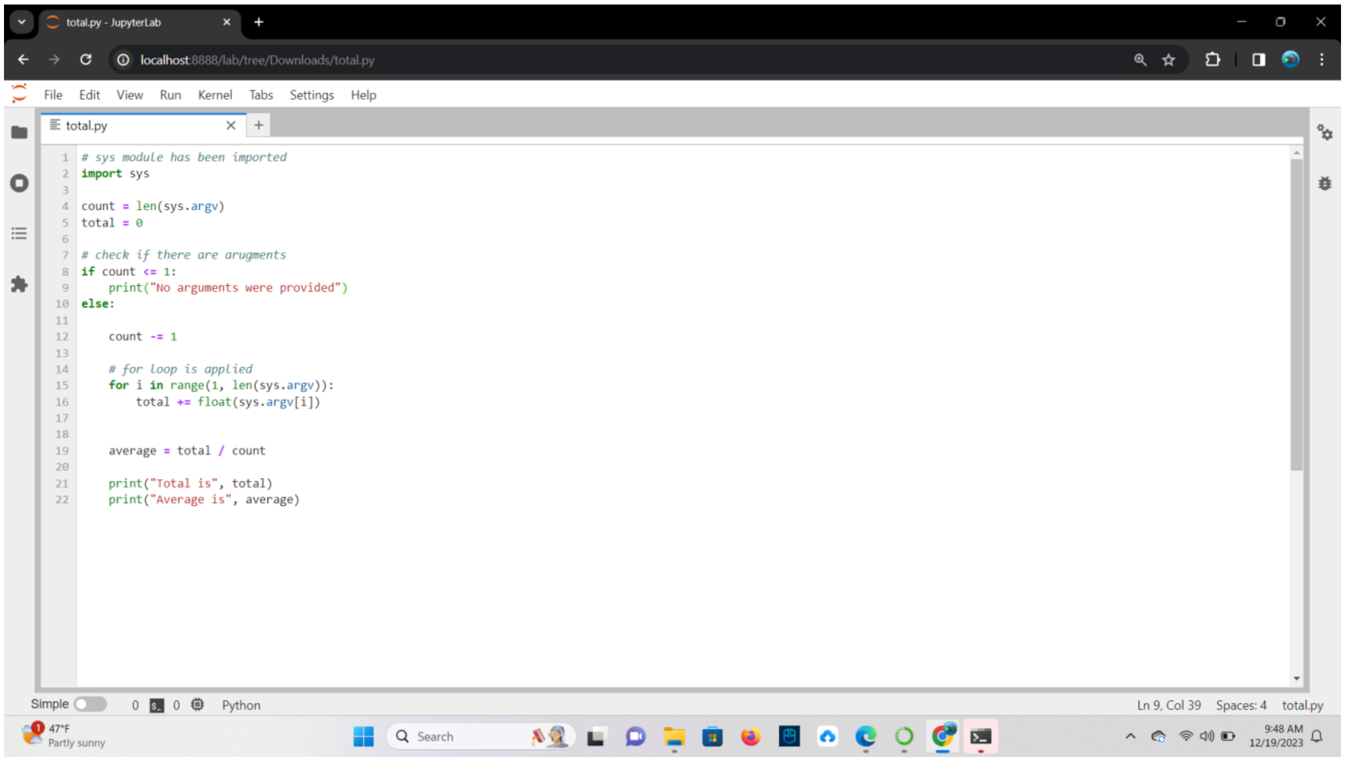
```
import sys

count = len(sys.argv)
total = 0
if count <= 1:
    print("No arguments passed. Please provide numeric values.")
else:

    count -= 1


    for i in range(1, len(sys.argv)):
        total += float(sys.argv[i])


    average = total / count

    print("Total is", total)
    print("Average is", average)
```



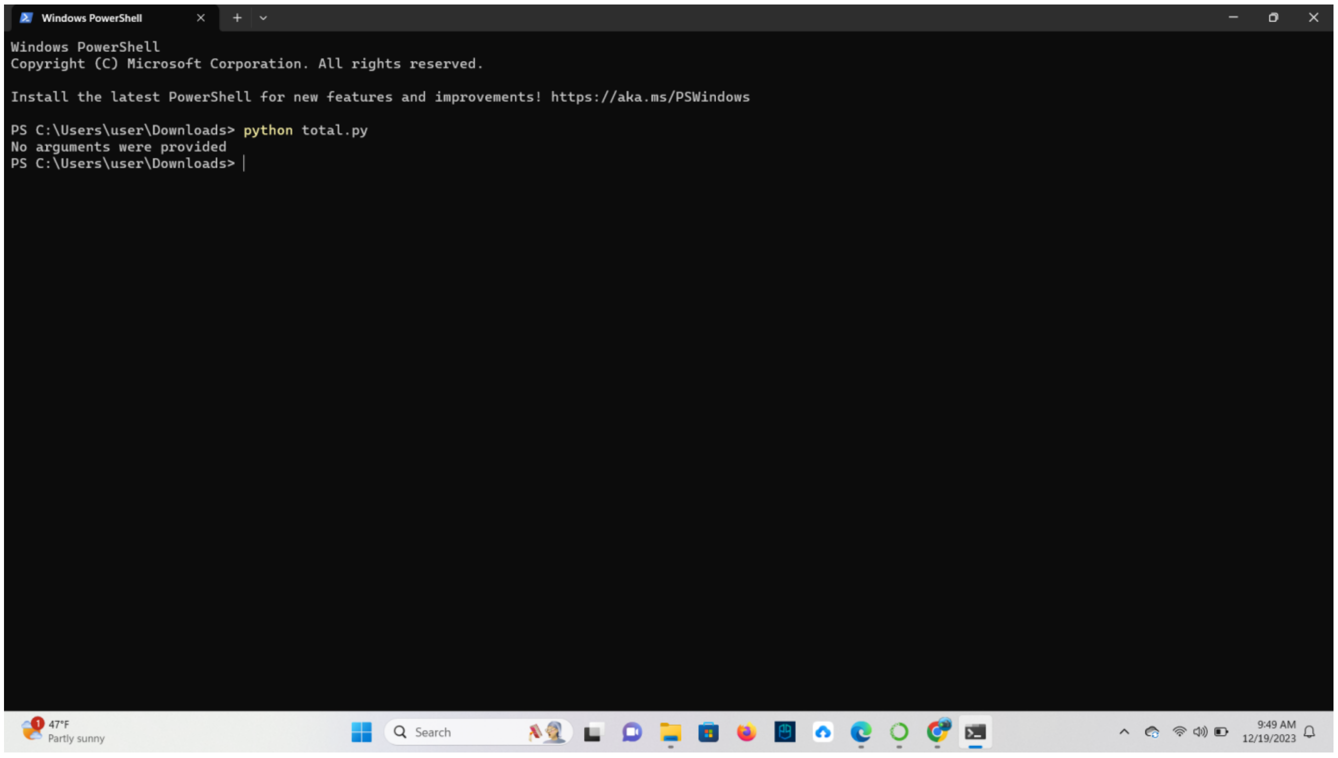
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python total.py 2 4 6 8
Total is 20.0
Average is 5.0
PS C:\Users\user\Downloads> python total.py
No arguments passed. Please provide numeric values.
PS C:\Users\user\Downloads>
```

TASK: Improve the program once more by adding a check to see whether no arguments have been passed, if so, print a message saying, “no arguments were provided”. Also add comments to the program. Execute the program several times for testing.

Ans



```python
# sys module has been imported
import sys

count = len(sys.argv)
total = 0

# check if there are arugments
if count <= 1:
    print("No arguments were provided")
else:

    count -= 1

    # for loop is applied
    for i in range(1, len(sys.argv)):
        total += float(sys.argv[i])


    average = total / count

    print("Total is", total)
    print("Average is", average)
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python total.py
No arguments were provided
PS C:\Users\user\Downloads>
```

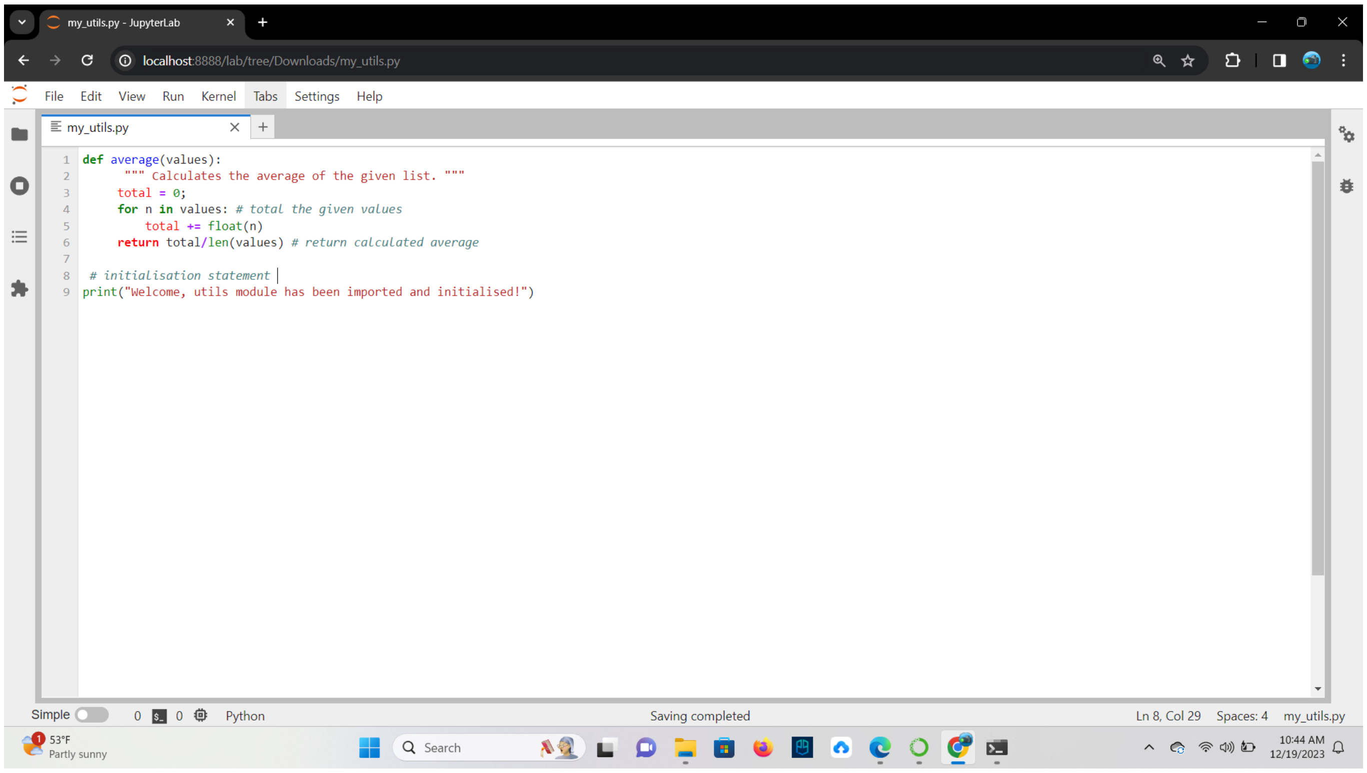
Writing Modules

TASK: Use an editor to input the Python program shown below then save it to a file called my_utils.py.

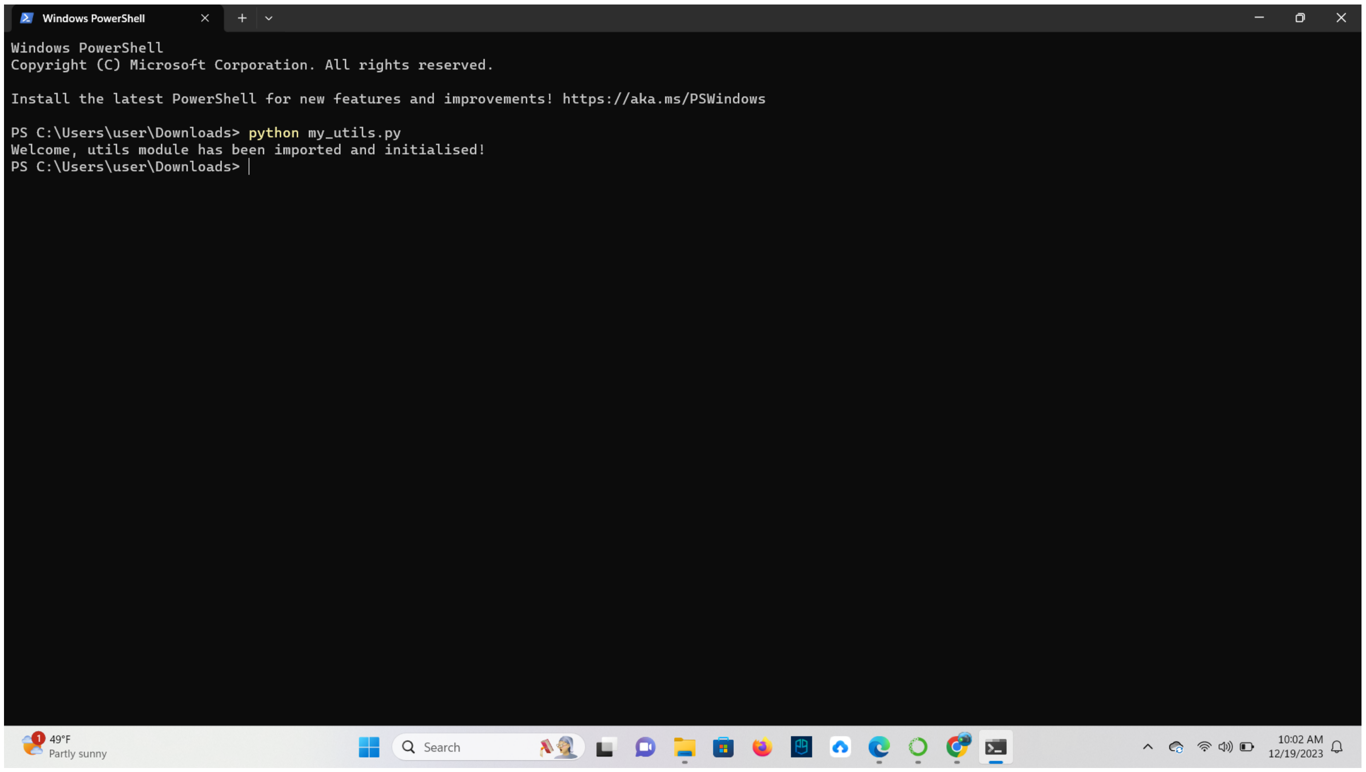
Once that file has been saved, execute it from the command line using the Python interpreter. If this is working correctly the ‘welcome’ message should be displayed.

```
def average(values):
    """ Calculates the average of the given list. """
    total = 0;
   for n in values: # total the given values
       total += float(n)
   return total/len(values) # return calculated average


 # initialisation statement
 print("Welcome, utils module has been imported and initialised!")
```

Ans



```
def average(values):
      """ Calculates the average of the given list. """
     total = 0;
     for n in values: # total the given values
         total += float(n)
     return total/len(values) # return calculated average

 # initialisation statement 
print("Welcome, utils module has been imported and initialised!")
```



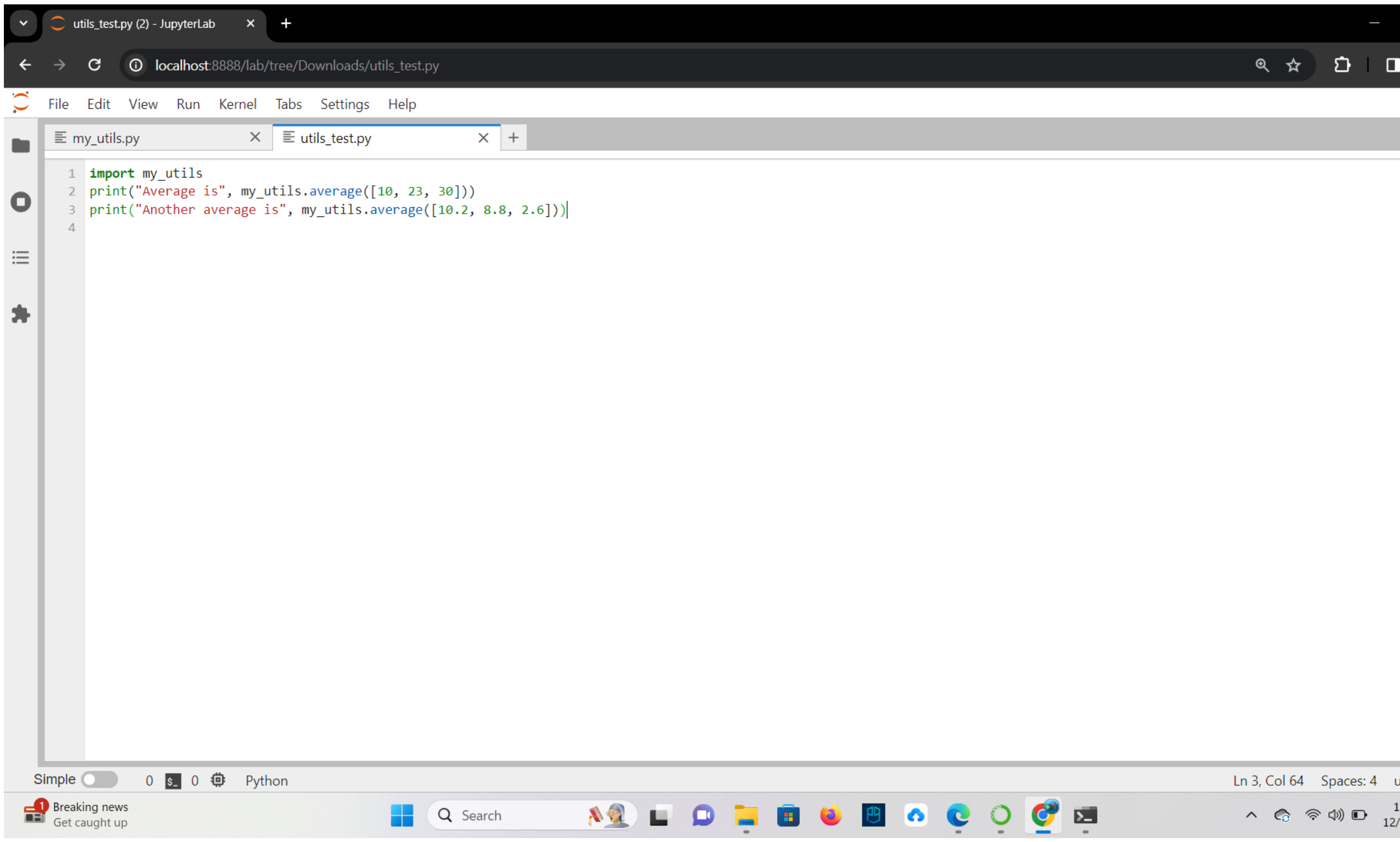
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python my_utils.py
Welcome, utils module has been imported and initialised!
PS C:\Users\user\Downloads>
```

TASK: Use an editor to input another Python program utils_test.py. This program should import my_utils then call the average() function several times, passing a list of values as a parameter, e.g.

print("Average is", my_utils.average([10, 23, 30]))

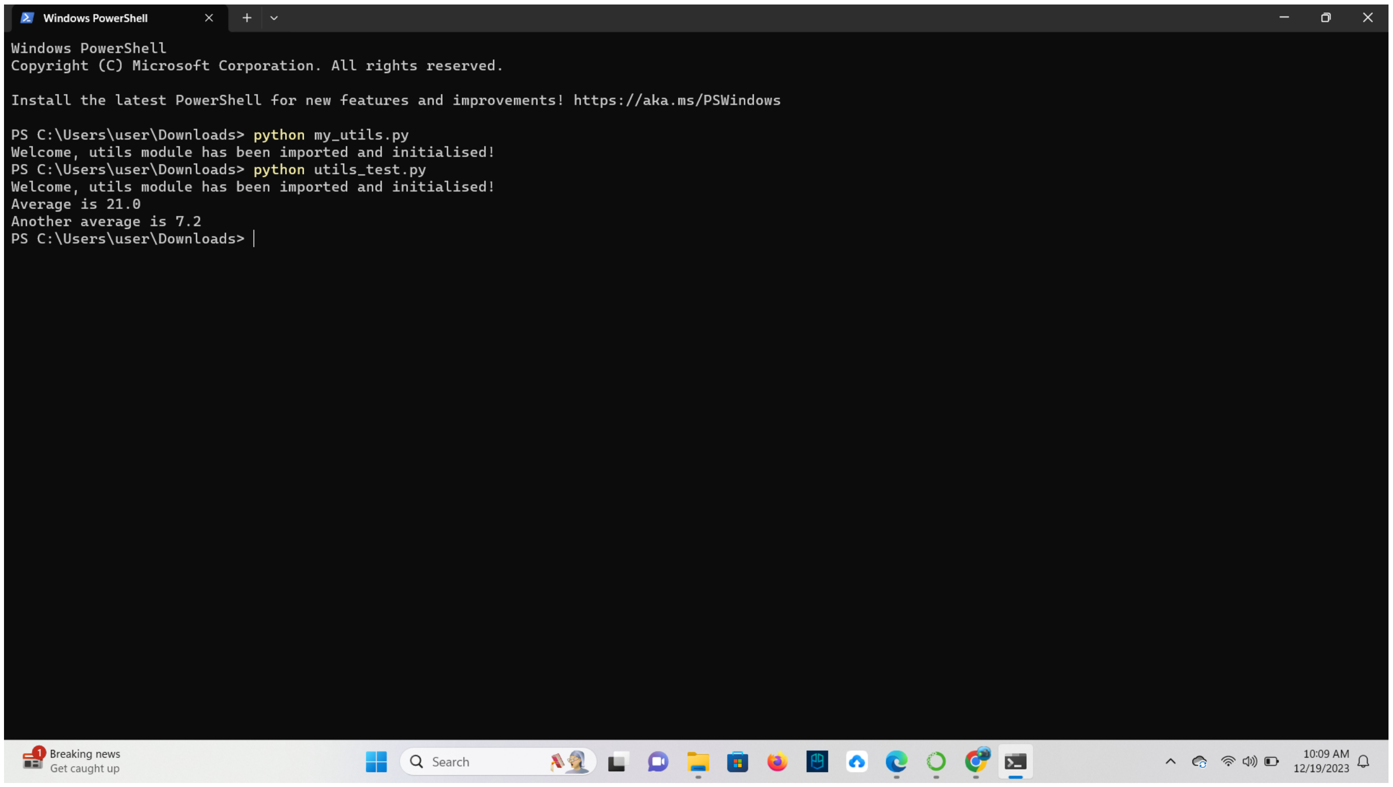
print("Another average is", my_utils.average([10.2, 8.8, 2.6])

Ans



utils_test.py (2) - JupyterLab
localhost:8888/lab/tree/Downloads/utils_test.py
File Edit View Run Kernel Tabs Settings Help
my_utils.py
utils_test.py
Simple 0 0 Python
Ln 3, Col 64 Spaces: 4

```python
import my_utils
print("Average is", my_utils.average([10, 23, 30]))
print("Another average is", my_utils.average([10.2, 8.8, 2.6]))
```

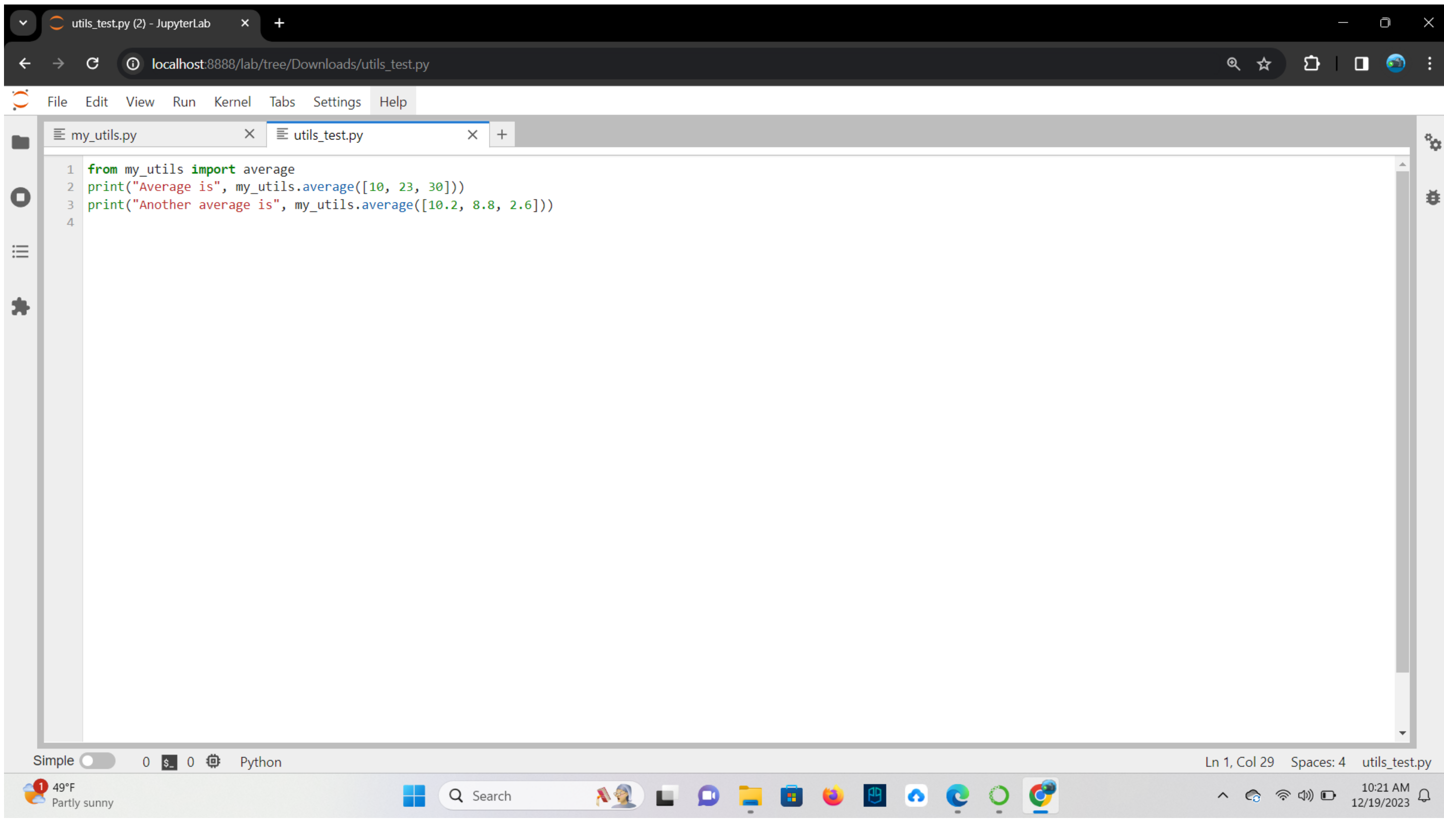


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python my_utils.py
Welcome, utils module has been imported and initialised!
PS C:\Users\user\Downloads> python utils_test.py
Welcome, utils module has been imported and initialised!
Average is 21.0
Another average is 7.2
PS C:\Users\user\Downloads>
```

Methods of Importing

TASK: Update the previous program utils_test.py, so that the import statement explicitly imports the average () function directly into the program’s symbol table, allowing the prefix to be removed from the later function calls.

Ans


utils_test.py (2) - JupyterLab
localhost:8888/lab/tree/Downloads/utils_test.py
File Edit View Run Kernel Tabs Settings Help
my_utils.py
utils_test.py
Simple 0 0 Python
Ln 1, Col 29 Spaces: 4 utils_test.py

```
from my_utils import average
print("Average is", my_utils.average([10, 23, 30]))
print("Another average is", my_utils.average([10.2, 8.8, 2.6]))
```

Using Import Aliases

The above three approaches to importing have been covered within earlier lessons. However there is another mechanism available during importing which helps avoid name clashes, or can be used to save time while typing.

The import statement can be extended to allow an alternative name, or alias, to be applied to the imported element. This means that the names of either modules or specific elements, such as functions, can be explicitly changed within the current program. This makes avoiding clashes easier, e.g. the following example explicitly changes the name of the imported sqrt() function to be root() instead.

```
from math import sqrt as root

print("The square root of 20954 is", root(20954))
```

Using an alias is also convenient when a general import is being used, since the name of the whole module can be aliased into something shorter, therefore avoiding the need to import all content into the program's symbol-table. So:

```
 import math as m

print("The square root of 20954 is", m.sqrt(20954))

print("The sine 0.653 is", m.sin(0.653))

 print("The cosine 0.623 is", m.cos(0.623))
```

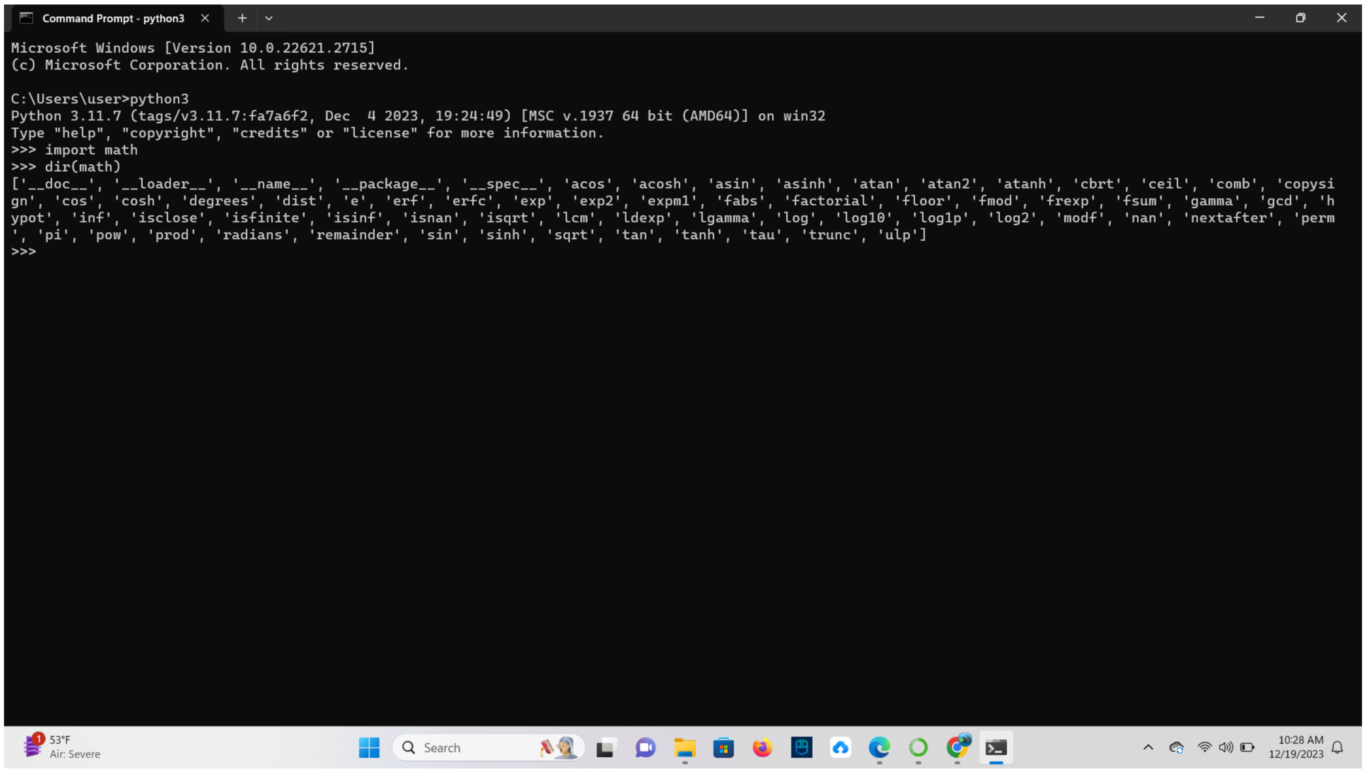
Listing Symbol Table Names

TASK: Start Python in interactive mode and input the following statements.

```
>>> import math

>>> dir(math)
```

Look at the list of names shown, these should generally look familiar by now. These are all of the names defined within the math module.

Ans



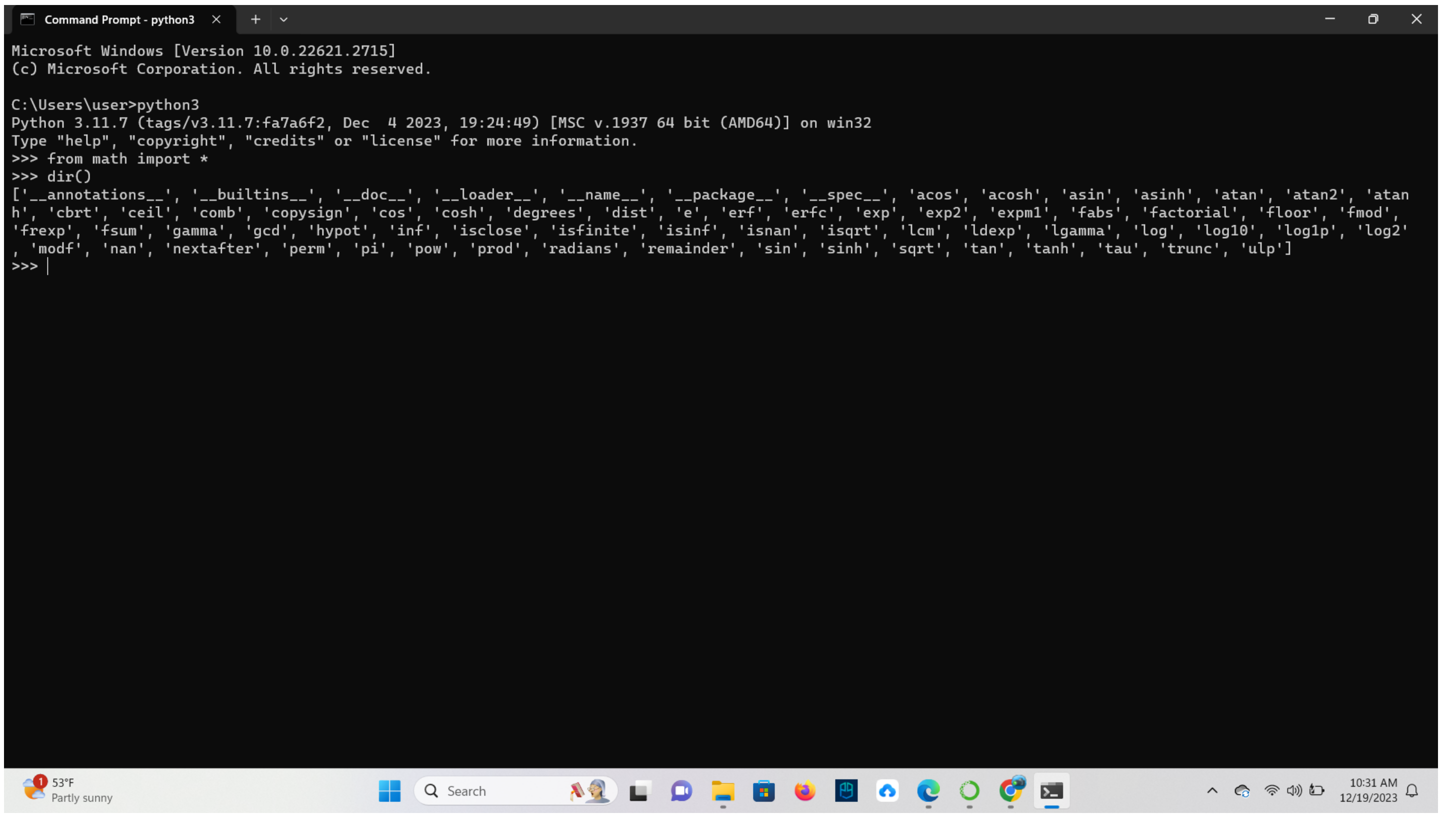
```
Microsoft Windows [Version 10.0.22621.2715]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>python3
Python 3.11.7 (tags/v3.11.7:fa7a6f2, Dec  4 2023, 19:24:49) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysi
gn', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'h
ypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm
', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
>>>
```

TASK: Now enter the following statements:

```
>>> from math import *
>>> dir()
```

Since an asterisk '*' was used to import the entire contents of the math module into the program's own symbol table, it now contains everything that was defined within the math module.

Ans



```
Microsoft Windows [Version 10.0.22621.2715]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>python3
Python 3.11.7 (tags/v3.11.7:fa7a6f2, Dec  4 2023, 19:24:49) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from math import *
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atan
h', 'cbrt', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2'
, 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
>>> 
```

The Module Search Path

TASK: Use an editor to input a Python program show_path.py. Execute the script and examine the results. Notice the first entry will be the directory in which the program file itself is stored.

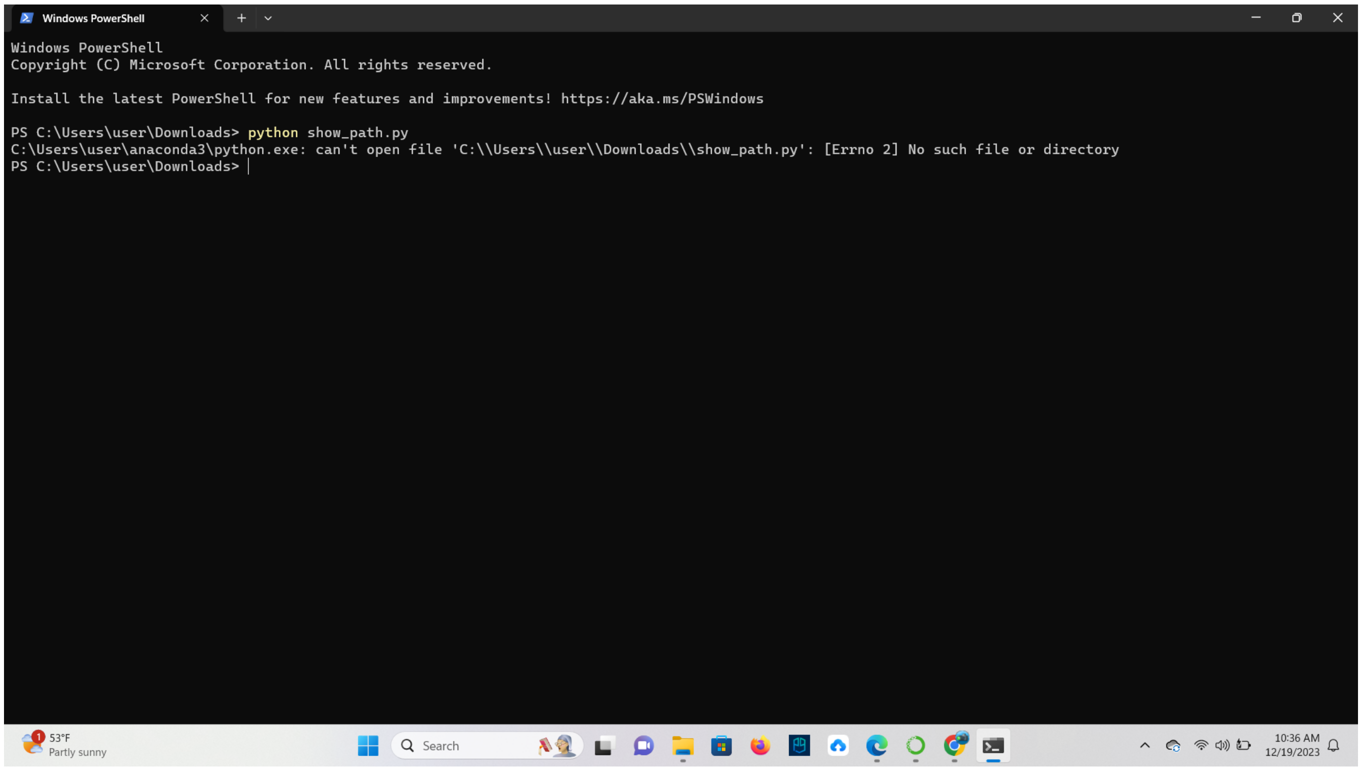
```
import sys
 print("The import search path for this program is", sys.path)
```

The sys.path value can be programmatically changed, allowing the program itself to include specific search locations when an import statement is being performed. However, hard coding system specific directory paths into code is bad practice, as it reduces program portability. A better option is to use the environment variable called PYTHONPATH. This can be setup within the host environment (OS). Any paths included in this variable are also included within the sys.path list. How this environment variable is set is Operating System specific, within a Unix type system it could be set a follows -

```
export PYTHONPATH='/my_modules:/util_modules'
```

This example will add two additional paths to the sys.path list the next time the python interpreter is used. Setting the search path in this way makes the code more portable between different systems, since only the environment variable needs changing rather than the program code itself.

Ans

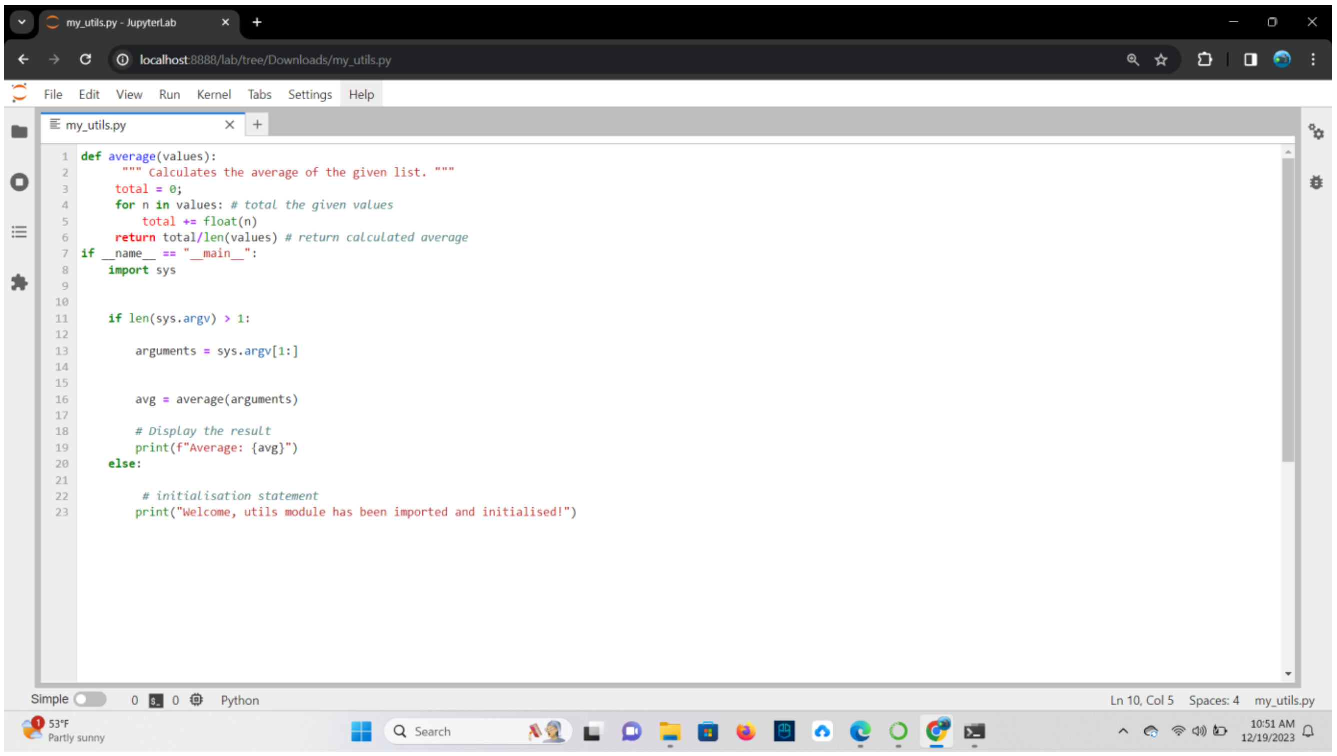


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python show_path.py
C:\Users\user\anaconda3\python.exe: can't open file 'C:\\Users\\user\\Downloads\\show_path.py': [Errno 2] No such file or directory
PS C:\Users\user\Downloads> |
```

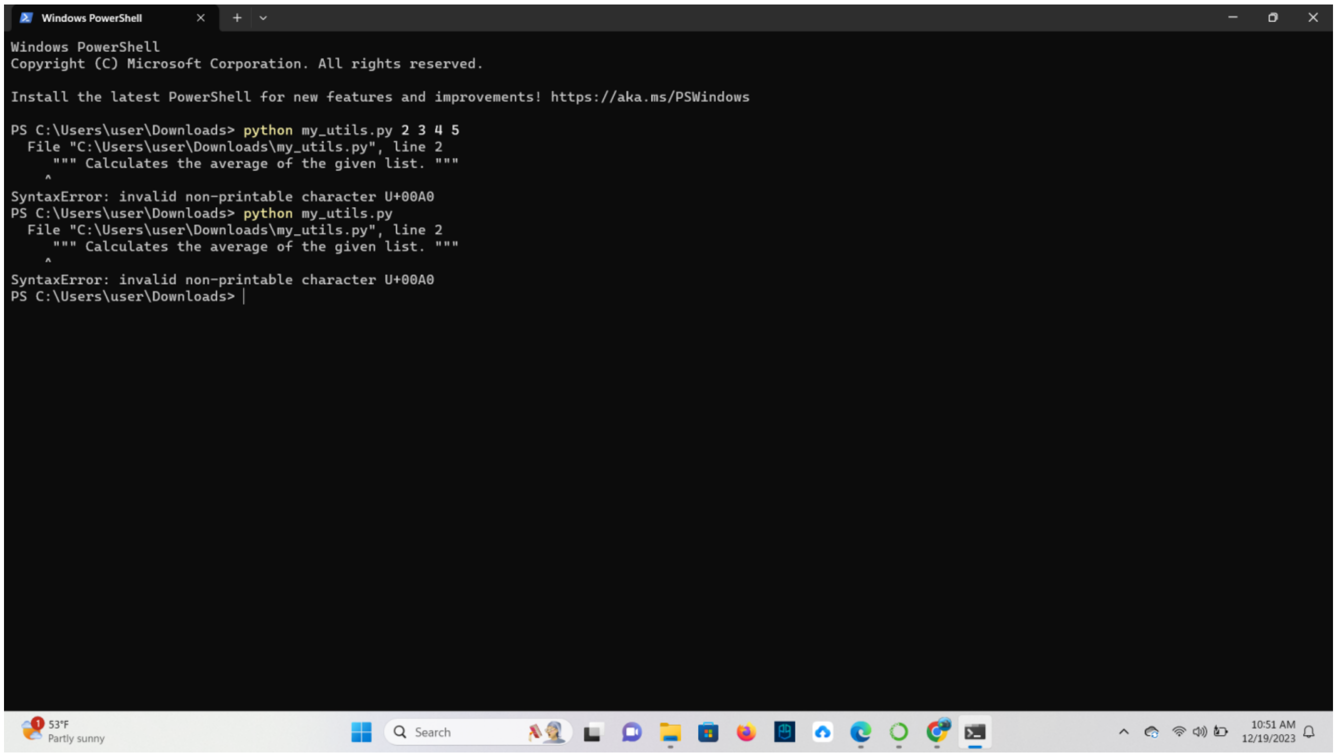
Writing a ‘Script’ and ‘Module’

TASK: Update the earlier program my_utils.py, so that when executed directly from the command line it displays the average of any provided command-line arguments. However, when imported by another program, nothing is displayed until the average () function is explicitly called.

Ans



```python
def average(values):
     """ Calculates the average of the given list. """
    total = 0;
    for n in values: # total the given values
        total += float(n)
    return total/len(values) # return calculated average
if __name__ == "__main__":
    import sys


    if len(sys.argv) > 1:

        arguments = sys.argv[1:]


        avg = average(arguments)

        # Display the result
        print(f"Average: {avg}")
    else:

         # initialisation statement
        print("Welcome, utils module has been imported and initialised!")
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\user\Downloads> python my_utils.py 2 3 4 5
  File "C:\Users\user\Downloads\my_utils.py", line 2
     """ Calculates the average of the given list. """
    ^
SyntaxError: invalid non-printable character U+00A0
PS C:\Users\user\Downloads> python my_utils.py
  File "C:\Users\user\Downloads\my_utils.py", line 2
     """ Calculates the average of the given list. """
    ^
SyntaxError: invalid non-printable character U+00A0
PS C:\Users\user\Downloads>
```

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.

- IDE

Ans An integrated development environment (IDE) is a program dedicated to software development.

- Module

Ans It is a file consisting of python code which can be functions, classes and variables.

- Command Line Arguments

Ans Command line arguments are input parameters passed to the script when executing them.

- Symbol-table

Ans It is a data structure maintained by the python compiler that contains all the essential information about each identifier.

- Search path

Ans It provides guidance to the Python interpreter about where to find various libraries and applications.