Laboratory Activity No. 5	
Functions, Modules, and Packages	
Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 02/15/2025
Section:BSCPE 1A	**Date Submitted:**02/15/2025
Name: PALMES, LEWIS CLARK	**Instructor:** ENGR. MARIA RIZETTE SAYO

1. Objective(s):

This activity aims to introduce students to the concept of modules and packages in Python.

2. Intended Learning Outcomes (ILOs):

The students should be able to:

2.1 Use the different Python built-in functions, modules, and packages.

2.2 Create a program with user-defined functions

2.3 Create a program that will import or load a user-created module and import its functions.

2.4 Create a program that will import or load a user-created package and import its modules.

3. Discussion:

As the program gets larger and more complex, it will be unavoidable for a programmer to split related and repeated codes into separate files, this in Python is referred to as Modules and Packages.

A **module** contains variables and functions that can be imported or used again in another program (Python file) without the programmer having to code the same variables and functions again. A python program (.py file) is considered to be a module. A **package** is simply a group of related modules or .py files combined together. A package commonly is considered to be a library which contains an enormous amount of functions and modules.

Python comes with built-in modules and packages such as the **math** module, **statistics** module, **random** module. For scientific computing, the following packages is mentioned here. The **NumPy** package is a general-purpose array-processing package. It provides a high-performance multidimensional array object. The **NumPy** package is used for scientific computing. Another well-known Python package is **Scikit-learn** which is an open source machine learning library that supports supervised and unsupervised learning for implementing Artificial Intelligence related tasks. For Software Development, the **Tkinter** and the **PyQt5** are the most commonly used. For Web Development, **Flask** and **Django** are the most widely used framework(composed of packages) for building websites with Python.

Functions

To implement a simple module or package, we need to refamiliarize ourselves with the concept of functions. Recall that a function is composed of a block of codes stored together that when called it executes those codes stored inside it. Functions help programmers organize related code into modular pieces that can be called upon to perform repetitive tasks. The Python interpreter has a number of functions and types built into it that are always available but programmers can add their own custom functions to implement more customized logic. The image below shows an example of a function written in Python.

```python
def function_name(parameter1,parameter2):
    # code 1
    # code 2
    # ...
    # code n
    return value # optional return
```

The def is a special keyword used to indicate a function definition or creation. The function_name is the name of the function
to be created. Recall that a function is also declared using parenthesis. Inside the parenthesis are inputs or arguments that can be accepted in the function through the declaration of parameters. Parameters are temporary variables or placeholder that can

be used inside the function. In Python, a return value is optional since the interpreter automatically makes the decision if it will be a void datatype function, an int datatype function, string datatype function, boolean datatype function and adjusts memory allocation accordingly.

Functions can be built-in or user-defined, for instance the Python print() is an example of a built-in function. The len() is also a built-in function. Other examples of built-in functions are: int(), str(), float(), bool(), list(), tuple(), dict(), open(). Functions will be used in the activity to create modules and packages. More built-in functions can be found in the official Python documentation.

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Source: https://docs.python.org/3/library/functions.html

In this activity, you will be exploring the various built-in Python functions, modules, and packages, and how to create functions in Python, put those functions in a module, call functions from a module to another program, and create your own package of modules.

For more information you may also visit the official python documentation:
https://docs.python.org/3/tutorial/modules.html
https://docs.python.org/3/tutorial/modules.html#packages

4. Materials and Equipment:

Desktop Computer with Anaconda Python or
Google Colab Windows Operating System

5. Procedure:

For this activity we will be creating Python programs using the Spyder IDE.

Functions

Exploring built-in functions

1. Create a folder named **builtinfunctions**
2. Create a Python file inside the functions1 folder named **evaluator.py** and copy the code shown below:

```
# Propositional logic evaluator for discrete math for 2-3 variables
print("Propositional logic evaluator for discrete math")
variables = int(input("How many variables? "))
total_combinations = 2**variables

combinations_list = [] # store all the possible combinations

# generate the combinations
for i in range(total_combinations):
    bin_equivalent = bin(i)[2:]
    while len(bin_equivalent)<variables:
        bin_equivalent="0"+bin_equivalent
    combinations_list.append(tuple(int(val) for val in bin_equivalent))
    # this will generate a list with values [(0,0),(0,1),(1,0),(1,1)]
    # for two variables

# main program
expression = input("Enter the propositional logic expression: ")
# note: Only the letters A,B,and C are allowed to be used
# example: not(A and B) or (A and C)
if variables == 2:
    print("A B f")
    for A,B in combinations_list:
        evaluated_expression = eval(expression)
        print(A,B, evaluated_expression)
elif variables == 3:
    print("A B C f")
    for A,B,C in combinations_list:
        evaluated_expression = eval(expression)
        print(A,B,C, evaluated_expression)
```

   Note: and, or, not must be small cases.

3. Run the program and observe the output. Try to analyze the purpose of the built-in functions used in the program (the keywords in color violet).
4. You may modify the code in order to study it as it will be used later in the next sections of the activity.

Using the open() function for file handling

1. Create a folder named **filehandling** outside of **builtinfunctions** folder
2. Create a Python file inside the **filehandling** folder named **filewriter.py**
3. Open the **filehandler.py** using Spyder IDE and type the code as shown below:

```python
name = "Royce Chua"
file = open("newfile1.txt",'w')
file.write(f"Hello, {name}!\n")
file.write("Isn't this amazing!\n")
file.write("that we can create and write on text files\n")
file.write("using Python.")
file.close()
```

Note: You may use help(open) either in a Python program or in the shell for more information about the built-in function.

4. Run the program and observe the output.
5. Modify the program to create a file called **newfile2.txt** and print the message (excluding the " ")
   "This message was created using Python!"
6. Create another Python file inside the **filehandling** folder named **filereader.py** and type the code as shown below:

```
file = open("new.txt",'r')
data = file.read()
print(data)
file.close()
```

7. It should display an error. Identify and resolve the cause of the error based on the message given by Python. The file that should be read is newfile1.
8. After fixing the error, run the program again and observe the output.
9. Modify the filereader.py program so that the message of **newfile2.txt** is displayed.
10. Modify again the program with the following code below:

```
file = open("newfile2.txt",'r')
data = file.read(12)
print(data)
file.close()
```

11. Create a new Python file still inside the oop1 folder called **fileappender.py** and copy the code shown below:

```
file = open("newfile2.txt",'a')
file.write("and also by the programmer of course. ")
file.close()
```

12. Run the program and observe the output.

    OBSERVATION: Creating and running the filewriter.py and filereader.py programs showed how to handle file operations in Python. We observed the successful creation and reading of files using the open() function. Identifying and resolving errors helped us understand file handling better.

User-defined Functions

1. Create a new folder called **userfunctions** outside of **filehandling** folder
2. In the **userfunctions** folder create a program called **truthtablegenerator.py** and copy the code below:

```python
def generate_truthtable(number_of_variables):
    total_combinations = 2**number_of_variables
    combinations_list = []
    for i in range(total_combinations):
        bin_equivalent = bin(i)[2:]
        while len(bin_equivalent)<number_of_variables:
            bin_equivalent="0"+bin_equivalent
        combinations_list.append(tuple(int(val) for val in bin_equivalent))
    return combinations_list

print(generate_truthtable(3))
```

3. Run the program and observe the output.
4. Modify the program by changing *print(generate_truthtable(3))* to *print(generate_truthtable())* then run the program
5. An error should occur, modify the program according to the code shown below:

```python
def generate_truthtable(number_of_variables=0):
    if number_of_variables == 0:
        return "You need to enter an integer"
    else:
        total_combinations = 2**number_of_variables
        combinations_list = []
        for i in range(total_combinations):
            bin_equivalent = bin(i)[2:]
            while len(bin_equivalent)<number_of_variables:
                bin_equivalent="0"+bin_equivalent
            combinations_list.append(tuple(int(val) for val in bin_equivalent))
        return combinations_list
```

Note: The code modifications are underlined in red.

6. In the same file/program, create a new function with the name **evaluate_propositional_logic()**. The parameter is c_list(combinations list) and the code can be found under the **# main program** comment in the first program made earlier.
7. After successfully placing the code in the function, call the function using this code

   evaluate_propositional_logic(generate_truthtable(3))
8. Analyze why in the generate_truthtable function we needed to print the function whereas in the evaluate_propositional_logic function, it prints the values on its own.
9. Compare the program **truthtablegenerator.py** with **evaluator.py**. Identify the advantages of placing code within functions against the sequential code done in the first.

OBSERVATION: The truthtablegenerator.py program demonstrated how to create and use functions. Modifying the program highlighted the importance of parameters and return values. The evaluate_propositional_logic() function showed better organization for readability and maintainability.

Modules
Built-in Modules
math module

1. Create a new folder called **modules1** outside of the **userfunctions**
2. Create a new Python file called **mathmodule.py** and copy the following codes as shown below:

```python
import math

def quadratic_formula(a,b,c):
    if b**2-(4*a*c)<0: # involving imaginary numbers
        x1 = (complex(-b,math.floor(math.sqrt(abs(b**2-(4*a*c))))))/2*a
        x2 = (complex(-b,-1*math.floor(math.sqrt(abs(b**2-(4*a*c))))))/2*a
        return x1, x2
    else:
        x1 = (-b+math.sqrt(b**2-(4*a*c)))/(2*a)
        x2 = (-b-math.sqrt(b**2-(4*a*c)))/(2*a)
        return x1, x2

#print(quadratic_formula(1,12,32))
print(quadratic_formula(1,2,3))
```

3. Run the program and observe the output. You may switch between the two sets of values.
4. Create a new Python file called **mathmodule2.py** and copy the following codes as shown below:

```python
import math

def angle_demo():
    angle = math.sin(math.pi/2) # the default input is in radians
    # angle sin(90)=1 in degrees == sin(pi/2)=1 in radians
    print(angle)
    # to make it convenient, convert to radians
    angle = math.sin(math.radians(90))
    print(angle)
    # this is also similar for cosine and other trigonometric and
    # hyperbolic functions

angle_demo()
```

5. To view additional functions in the module. Type and run help(math) while it is imported.

OBSERVATION: Using the math and time modules in the mathmodule.py and dateandtime.py programs demonstrated how to import and use built-in modules. Observing the outputs helped us understand their functionalities and how to access more functions using help().

time and datetime module

1. Create a new file in the folder named **dateandtime.py**
2. Copy and run the code as show below:

```python
import time

def pause():
    for i in range(10,0,-1):
        print(f"The program will end in {i}..")
        time.sleep(1)

pause()
```

3. Observe the output.
4. In the same file, copy and add the code to the file as shown below:

```python
def current_time():
    t = time.strftime("%I:%M %p")
    return t

print(current_time())
```

5. Run the program, and observe the output.
6. In the same file, copy and add the code to the file as shown below:

```python
def current_date():
    d = time.strftime("%b %d %Y")
    return d

print(current_date())
```

7. Run the program, and observe the output.

OBSERVATION: The dateandtime.py file was used as a module in the main.py program. Modifying the code to display the current date and time showed the benefits of organizing and reusing code, improving readability and reducing redundancy by importing functions directly from the module.

User-defined Modules

1. The previously created dateandtime.py is considered to be a module that you can import.
2. In the same **modules1** folder, create a new file called main.py and copy the following code:

```python
import dateandtime

print("The current time is",dateandtime.current_time())
```

3. The program will not run as expected, and you will need to remove the following codes in the dateandtime.py which are underlined in red.

```python
import time

def pause():
    for i in range(10,0,-1):
        print(f"The program will end in {i}..")
        time.sleep(1)

#pause()

def current_time():
     t = time.strftime("%I:%M %p")
     return t

#print(current_time())

def current_date():
     d = time.strftime("%b %d %Y")
     return d

#print(current_date())
```

4. Run the main.py program again and you will now see the correct output which is the current time.
5. Modify the main.py to also display the current date using the current_date() in dateandtime.py
6. To remove the need to constantly indicate the module name dateandtime. in each function, modify the code as shown below:

```python
from dateandtime import current_time, current_date

print("The current time is",current_time())
print("The current time is",current_date())
```

6. Supplementary Activity:

Tasks

Simple Word Filter

1. Create a function that would accept two inputs: a sentence(string), and a list containing bad words that the user would like to censor but not remove. The function should return the newly filtered sentence wherein the bad words are replaced with asterisks equal to the length of the censored word.

2. Given a certain Physics problem create a function(projectilemotion_solver) that would take in the following inputs below and return the needed information when the function is called. Name the program containing the function projectilemotion.py then create another program main_program.py and import projectilemotion.py

   “A long jumper leaves the ground at an angle of 20.0° above the horizontal and at a speed of 11.0 m/s. “

   (a) How far does he jump in the horizontal direction?
   (b) What is the maximum height reached?

   Given a projectile motion problem like this where the angle and speed are given, the range or distance travelled in the horizontal direction can be determined by using the formula:

   $$R = \frac{v_i^2 sin2\theta_i}{g}$$

   The maximum height can be determined using the formula:

   $$h = \frac{v_i^2 sin2\theta_i}{2g}$$

   Reference: Serway, Jewet (2019), Physics for Scientists and Engineers 9e

3. Create a quadratic equation solver module that would write the inputs of the user and the corresponding output into text files.

FOR THE PROGRAM PLEASE REFER TO THIS LINK:
GITHUB LAB ACT 5

Questions

1. Why do built-in functions exist?

   Built-in functions provide essential, commonly used functionalities that save time and effort by offering pre-defined, optimized solutions for various tasks, allowing developers to focus on solving specific problems.

2. What is the advantages/disadvantages of placing code inside functions vs sequential codes.

   Functions offer reusability, modularity, and improved readability, making code easier to maintain. However, they introduce slight performance overhead and can increase complexity if overused.

3. What is the difference between a function and a module?

   A function is a block of code designed to perform a specific task, while a module is a file containing Python code (including functions, variables, and classes) that helps organize related code into separate files.

4. What is the difference between a module and a package?
   A module is a single file containing Python code, while a package is a collection of modules organized in directories, with a coherent structure for easier management and distribution.

7. Conclusion: In this laboratory activity, we explored the concepts of functions, modules, and packages in Python. Understanding and utilizing built-in functions can save time and effort in our programming tasks. Creating user-defined functions allows us to modularize and organize our code, making it more readable and maintainable.

Modules help us group related code into separate files, promoting reusability and reducing redundancy. Packages enable us to organize modules into a coherent structure, facilitating the management and distribution of larger codebases.

Through hands-on exercises, we demonstrated how to create and use functions, import and use modules, and organize modules into packages. By applying these concepts, we can develop more efficient, organized, and maintainable Python programs

8. Assessment Rubric: