Faculty of Engineering South Eastern University of Sri Lanka	Course	BSc Eng (Hons)
	Semester	1
	Subject	CS 13001 Introduction to Computing

SEU/IS/...../EG/.....

Lab Session 4: Functions

Lab Objectives:

1) Understanding functions in the context of programming.
2) Utilising certain built-in Python functions
3) Building or defining custom functions
4) Understanding parameters and arguments
5) Distinguishing fruitful functions and void functions

1. Function calls

In the context of programming, a function is a named sequence of statements that performs a computation. A function "takes" an argument and "returns" a result. The result is called the return value.

For example in,

```
>>> type(32)
<class 'int'>
```

The name of the function is *type,* the argument to the function is *32*, and the result or return value is *<class 'int'>.*

2. Built-in Python functions

Python provides a number of important built-in functions that we can directly use, simply by calling their name. The function *type* that we used above is such a built-in function. It is generally a good practice to treat built-in functions as reserved words. Therefore, avoid using them as variable names in your programmes.

Exercise 1: Try the following functions along with the arguments in the Python interpreter and write down the answers. Also observe and comment on its nature – i.e. what the function is doing.

Function	Answer	What is the function doing?
>>> max ('Hello world')		
>>> min ('Hello world')		
>>> len ('Hello world')		
>>> round (2.54978, 2)		

3. Type conversion functions

Python also provides built-in functions that convert values from one type to another. We have used some of these type coversion functions in our previous lab sessions.

Exercise 2: Try the following functions along with the arguments in the Python interpreter and write down the answers. Also observe and comment on each function's nature – i.e. what the function is doing.

Function	Answer	What is the function doing?
>>> int ('32')		
>>> int ('Hello')		
>>> int (3.99999)		
>>> int (-2.3)		
>>> float (32)		
>>> float ('3.14159')		
>>> str (32)		
>>> str (3.14159)		
>>> str (bingo)		

4. Random numbers

Random numbers are needed in many computer applications. A very familiar example is games. Python provides several ways of generating random numbers through a special module called *random*.

When you use special modules, you have to import them into your programme first. The module object contains the functions and variables defined in the module. To access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period). This format is called dot notation.

Exercise 3: Write the following code as a Python script, save it as ex3.py and run it through the terminal.

```
import random #importing special module called random

for i in range(10):
    x = random.random()  #accessing the function called random() in the random module via dot notation
    print(x)
```

After you run the programme, check with the neighbouring student whether he or she got the same answer. Then rerun the programme again to see if the answer you got is the same as on the previous occasion.

Exercise 4: Execute the following functions along with the arguments in the Python interpreter and write down the answers. Also observe and comment on each function's nature – i.e. what the function is doing.

Function	Answer	What is the function doing?
>>> random.randint(5, 10)		
>>> random.randint(5, 10)		
>>> t = [1, 2, 3] >>> random.choice(t)		
>>> random.choice(t)		

The random module also provides functions to generate random values from continuous distributions including Gaussian, exponential, gamma, and a few more.

5. Mathematical functions

Python has a math module that provides most of the familiar mathematical functions.

Exercise 5: Write the following code as a Python script, save it as ex5.py and run it through the terminal. The program prints out the sin value for the angle (in degrees).

```
import math

degrees = 45
radians = degrees / 360.0 * 2 * math.pi
sin_value = math.sin(radians)
print (sin_value)
```

Exercise 6: Convert the above code such that the programme asks for the user for the angle in degrees and returns the sin value as the answer. Round the answer to four decimal points. Make sure that an error message is printed if the user inputs text.

```
Enter angle in degrees : 45
Sin value for  45.0  degrees is  0.7071
```

Exercise 7: Check the answer for $\frac{1}{\sqrt{2}}$ in the Python interpreter. Does the answer match with the answer you got for *sin 45*?

Hint: use the sqrt() function inside the math module. You have to import the math module in the interpreter before using the functions in the module.

Exercise 8: Write a programme that takes in the radius of a circle from the user and computes the area and the circumference of the circle and prints them out.

6. Defining custom functions

So far, we have only used built-in Python functions. However, we can define our own functions and use them again and again inside our programme.

Exercise 9: Write the following code as a Python script, save it as ex9.py and run it through the terminal.

```
def print_lyrics():
    print('Rambari kiyapan numbe nombare')
    print(' Vaadi pulla vaadi . . .  vaadi pulla vaadi')

print_lyrics()
```

Naming conventions for function names are the same as for variable name. Also, as indicated earlier, you should avoid using the same name for a variable and a function inside a programme.

The first line of the function definition is called the header; the rest is called the body. The header has to end with a colon and the body has to be indented. By convention, the indentation is always four spaces or one tab.

Exercise 10: Define the same function in the Python interpreter and run it.

```
>>> def print_lyrics():
...     print('Rambari kiyapan number nombare')
...     print('Vaadi pulla vaadi . . . vaadi pulla vaadi')
...
>>> print_lyrics()
Rambari kiyapan number nombare
Vaadi pulla vaadi . . . vaadi pulla vaadi
>>> print(type(print_lyrics))
<class 'function'>
>>> print(print_lyrics)
<function print_lyrics at 0x04F75108>
```

Once you have defined a function, you can use it inside new functions. For example, you can define a new function called repeat_lyrics() and call the print_lyrics() function inside of it as many times as you want.

Exercise 11: Define a new function called *repeat_lyrics()* that uses the *print_lyrics()* function twice.

```
>>> def repeat_lyrics():
...     print_lyrics()
...     print_lyrics()
...
>>> repeat_lyrics()
Rambari kiyapan number nombare
Vaadi pulla vaadi . . . vaadi pulla vaadi
Rambari kiyapan number nombare
Vaadi pulla vaadi . . . vaadi pulla vaadi
```

Exercise 12: Amend the code in Exercise 10 to include a new function called repeat_lyrics() which repeats the lyrics three times. Run it from the terminal.

7. Flow of execution

Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom.

Function *definitions* do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.

A function call is like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the body of the function, executes all the statements there, and then comes back to pick up where it left off.

Exercise 13: Move the last line of this program you wrote in Exercise 12 to the top, so the function call appears before the definitions. Run the program and see what error message you get.

Exercise 14: Move the function call back to the bottom and move the definition of *print_lyrics()* after the definition of *repeat_lyrics()*. What happens when you run this program?

8. Parameters and arguments

Take a look at the following example. We are defining a function called *print_twice*. This function takes an argument from the user and prints it out two times. In the first execution, the argument is 'How are you?'. In the second execution, the argument is 255.

```
>>> def print_twice(x):
...     print(x)
...     print(x)
...
>>> print_twice('How are you?')
How are you?
How are you?
>>> print_twice(255)
255
255
```

The formal way of describing this process is the function print_twice takes an argument from the user and assigns it to a parameter x. A function may take more than one argument and have more than one internal parameters, as needed.

Exercise 15: Execute the following functions along with the arguments in the Python interpreter and write down the answers. Also observe and comment on what's happening to the argument.

Function	Answer	What's happening to the argument?
>>> import math >>> print_twice(math.pi)		
>>> print_twice('Spam '*4)		
>>> print_twice(math.cos(math.pi))		
>>> michael = 'Eric, the half a bee.' >>> print_twice(michael)		

9. Fruitful vs. void functions

Some of the functions we are using yield results. The math functions we used, for example, returned results or values. These are called *fruitful functions*. Other functions, like print_twice, perform an action but don't return a value. They are called *void functions*. When try to assign the result of a void function to a variable, the special value None gets stored in that variable.

```
>>> import math
>>> var_x = math.sqrt(5)
>>> print(var_x)
2.23606797749979
>>> var_y = print_twice('how do you do?')
how do you do?
how do you do?
>>> print(var_y)
None
```

To return a result from a function, we use the return statement in our function. For example, we could make a very simple function called addtwo that adds two numbers together and returns a result.

Exercise 16: Write the following code as a Python script, save it as ex16.py and run it through the terminal.

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print(x)
```

10. Why use functions?

There are several reasons why writing functions inside our programmes is a good idea:

- Creating a new function gives you an opportunity to name a group of statements, which makes your program easier to read, understand, and debug.
- Functions can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.
- Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.

Review Questions

1. What is the purpose of the “def” keyword in Python?

   a) It is slang that means “the following code is really cool”

   b) It indicates the start of a function

   c) It indicates that the following indented section of code is to be stored for later

   d) b and c are both true

   e) None of the above

2. What will the following Python program print out?

```
def fred():
    print("Zap")
def jane():
    print("ABC")

jane()
fred()
jane()
```

   a) Zap ABC jane fred jane

   b) Zap ABC Zap

   c) ABC Zap jane

   d) ABC Zap ABC

   e) Zap Zap Zap

3. Write a program that asks the user for the number of hours worked and the payment per hour. You should give the employee 1.5 times the hourly rate for hours worked above 40 hours. Inside the program define a function called compute_pay() that takes two arguments – i.e. hours and rate, and performs all the computations necessary and returns the total payment.

4. Rewrite the grade program from the previous lab session using a function called computegrade that takes a score as its parameter and returns a grade as a string.