COMP2152 LAB MANUAL

OPEN SOURCE DEVELOPMENT

This booklet will help the reader understand the concepts, principles, and implementation of the Python programming language. By the end of the booklet, the reader will be able to code comfortably in Python.

Prepared by Ben Blanc

TABLE OF CONTENTS

CHAPTER 7

FUNCTIONS & MODULES

FUNCTIONS

A function is a series of code that executes a specific task. You organize code into a function to avoid re-writing the same block of code to perform the same series of tasks.

Creating A Function

The structure for creating a function is:

def NameOFFunction([parameters]) :

statements

NO ARGUMENT FUNCTION

The following are examples of functions that take no arguments

```
def say_hello():
    pass
```

The function above has no body content

```
def say_hello():
    print("hello")
```

Run Function

To run the function above, type the following code at the after the function declaration:

```
say_hello()
```

CHAPTER 7 AT A GLANCE

In this chapter you will learn how about create functions and modules. You will learn how to import your modules and how to call and run your functions.

FUNCTION WITH ARGUMENTS (PARAMETERS)

The following is an example of a function that takes arguments

```
def say_hello(first_name, last_name):
    print(first_name, last_name)
```

The function above has body content and outputs the two arguments passed to it.

Run Function

To run the function above, type the following code:

```
say_hello("John", "Doe")
```

FUNCTION WITH ARGUMENTS AND RETURN VALUE

The following is an example of a function that takes arguments and returns a value

```
def say_hello(first_name, last_name):
    return "Hello, "+ first_name+" "+last_name
```

The function above has body content and returns the two arguments passed to it, prepended with the greeting of “Hello, “

Run Function

To run the function above, type the following code:

```
print(say_hello("John", "Doe"))

value = say_hello("John", "Doe")

print(value)
```

The first line outputs the return statement directly, while the second line stores the value into a variable, to be used later.

FUNCTION WITH ARGUMENTS LIST AND RETURN VALUE

The following is an example of a function that seeks for arguments and returns a value

```
def say_hello(*args):
    greeting = "Hello "

    for person in args :
        greeting+=person + ","

    return greeting[:-1]
```

The function above uses the *args magic variable to lool for any arguments passed to the function. It then loops over those arguments to create a greeting. Then the function returns to greeting, chopping off the trailing comma and space characters.

Run Function

To run the function above, type the following code:

```
print(say_hello("John", "Jane", "Mary", "Peter", "Jack"))

value = say_hello("John", "Jane", "Mary", "Peter", "Jack")

print(value)
```

FUNCTION DESCRIPTION

To describe a function, use triple quotes below the function declaration. It is good practice to declare the functions parameters and return value in the documentation

```
def say_hello(*args):
    """
    The function greets names of all arguments
    :params any string values
    :returns string of names with greeting
    """

    greeting = "Hello "

    for person in args :
        greeting+=person + ","

    return greeting[:-1]
```

To view a functions documentation, use the **help()** method

```
help(say_hello)

help(input)
```

MODULE

A module is a file of python code. Variables declared outside of functions are global variable. Variables declared inside of a function are local variable.

Creating a Module

Let us create a new python file called **mymod.** In this file, insert the following code:

```
CourseCode = 2152
def say_hello(*args):
    """
    The function greets names of all arguments
    :params any string values
    :returns string of names with greeting
    """

    greeting = "Hello "

    for person in args :
        greeting+=person + ","

    return greeting[:-1]
```

Now let us create a file called **modrunner**. In this file, we will import the **mymod** file and run our say_hello function

```
import mymod

print(mymod.say_hello())
```

By default, the namespace of the module is the filename. To import our say_hello function into the global namespace, we code:

```
from mymod import *

print(say_hello())
```

We can access our CourseCode variable, but not our greeting variable

```
import mymod

from mymod import *

print(mymod.say_hello())

print(say_hello())

print(mymod.CourseCode)

print(CourseCode)

#print(mymod.greeting) #error

#print(greeting) #error
```

FUNCTION DESCRIPTION

To describe a function, use triple quotes at the top of the file. Navigate to **mymod** and add a description

```
"""The is our first module
:variable CourseCode = course code for this course
"""
CourseCode = 2152
def say_hello(*args):
    """
    The function greets names of all arguments
    :params any string values
    :returns string of names with greeting
    """

    greeting = "Hello "

    for person in args :
        greeting+=person + ","

    return greeting[:-1]
```

We can see the documentation for the module by typing the following code in our **modrunner**

```
help(mymod)
```

VARIABLE SCOPE

If we create another function in our **mymod**, we can access our global variable by the following code

```
def say_goodbye():
    print(CourseCode)
```

In our **modrunner** file, we can call the say_goodbye function

```
mymod.say_goodbye()

say_goodbye()
```

In order to change the value of CourseCode, we declare to python that we are accessing and possible altering the global variable to typing the following code:

```
def say_goodbye():
    global CourseCode
    CourseCode = 5221
    print(CourseCode)
```

To access a global variable in a function, use the keyword global then the variable name

global Variable_Name

Then you can alter the value of the global variable.

```
mymod.say_goodbye()

say_goodbye()

print(mymod.CourseCode)
```