

Master of Engineering in Internetworking

INWK 6312 – Programming

for Internetworking Applications

Lab # 1

Python Introduction, Fruitful Functions, Conditionals and Recursion,

Table of Contents

[SECTION 1 4](#_Toc523073312)

[Chapter 1 - Introduction 4](#_Toc523073313)

[Chapter 2 - Objectives of the section 4](#_Toc523073314)

[Chapter 3 - The way of the program 4](#_Toc523073315)

[Task 1 4](#_Toc523073316)

[Chapter 4 - Variables, Expressions, and Statements 5](#_Toc523073317)

[Task 2 5](#_Toc523073318)

[Chapter 5 - Functions 5](#_Toc523073319)

[Task 3 5](#_Toc523073320)

[Task 4 6](#_Toc523073321)

[Task 5 7](#_Toc523073322)

[SECTION 2 7](#_Toc523073323)

[Chapter 6 - Objectives of the Section 7](#_Toc523073324)

[Chapter 7 - Conditionals and Recursion 8](#_Toc523073325)

[Important concepts 8](#_Toc523073326)

[**modulus operator:** 8](#_Toc523073327)

[**boolean expression:** 8](#_Toc523073328)

[**relational operator:** 8](#_Toc523073329)

[**logical operator:** 8](#_Toc523073330)

[**conditional statement:** 8](#_Toc523073331)

[**condition:** 8](#_Toc523073332)

[**compound statement:** 8](#_Toc523073333)

[**branch:** 8](#_Toc523073334)

[**chained conditional:** 8](#_Toc523073335)

[**nested** **conditional**: 8](#_Toc523073336)

[**recursion:** 8](#_Toc523073337)

[**base case:** 8](#_Toc523073338)

[**infinite** **recursion**: 8](#_Toc523073339)

[Task 6 8](#_Toc523073340)

[Task 7 9](#_Toc523073341)

[Task 8 9](#_Toc523073342)

[Task 9 9](#_Toc523073343)

[Task 10 9](#_Toc523073344)

[Chapter 8 - Fruitfull Functions 10](#_Toc523073345)

[Important concepts 10](#_Toc523073346)

[temporary variable: 10](#_Toc523073347)

[dead code: 10](#_Toc523073348)

[None: 10](#_Toc523073349)

[incremental development: 10](#_Toc523073350)

[Task 11 10](#_Toc523073351)

[Task 12 11](#_Toc523073352)

## SECTION 1

# Introduction

This section is to introduce you to the python language and enables you to learn the basic tasks of using the python interpreter. We will be using an Ubuntu VM for performing the lab exercises. The recommended IDE for this lab is vim/nano/gedit or the python interpreter itself.

# Objectives of the section

At the end of this section you will have learned the following:

* The difference between python and the bash shell
* Python as a basic calculator
* The different types of variable in pythons
* The syntax of functions
* Use built-in functions
* Write your own basic functions in python

# The way of the program

## Task 1

*Start the Python interpreter and use it as a calculator. Python’s syntax for math operations is almost the same as standard mathematical notation. For example, the symbols +, - and /denote addition, subtraction, and division, as you would expect. The symbol for multiplication is \*.*

*If you run a 10-kilometer race in 43 minutes 30 seconds, what is your average time per mile? What is your average speed in miles per hour? (Hint: there are 1.61 kilometers in a mile).*

# Variables, Expressions, and Statements

## Task 2

*Practice using the Python interpreter as a calculator:*

1. *The volume of a sphere with radius r is 4/3 π r3. What is the volume of a sphere with radius 5? Hint: 392.7 is wrong!*
2. *Suppose the cover price of a book is $24.95, but bookstores get a 40% discount. Shipping costs $3 for the first copy and 75 cents for each additional copy. What is the total wholesale cost for 60 copies?*
3. *If I leave my house at 6:52 am and run 1 mile at an easy pace (8:15 per mile), then 3 miles at tempo (7:12 per mile) and 1 mile at easy pace again, what time do I get home for breakfast?*

(8:15 per mile), 🡺 8 minutes, 15 seconds

(7:12 per mile) 🡺 7 minutes, 12 seconds

# Functions

## Task 3

*Python provides a built-in function called len that returns the length of a string, so the value of len('allen') is 5.*

*Write a function named right\_justify that takes a string named s as a parameter and prints the string with enough leading spaces so that the last letter of the string is in column 70 of the display.*

*>>> right\_justify('allen') allen*

## Task 4

*A function object is a value you can assign to a variable or pass as an argument. For example, do\_twice is a function that takes a function object as an argument and calls it twice:*

*def do\_twice(f):*

*f()*

*f()*

*Here’s an example that uses do\_twice to call a function named print\_spam twice.*

*def print\_spam():*

*print( 'spam')*

*do\_twice(print\_spam)*

1. *Type this example into a script and test it.*
2. *Modify do\_twice so that it takes two arguments, a function object and a value, and calls the function twice, passing the value as an argument.*
3. *Write a more general version of print\_spam, called print\_twice, that takes a string as a parameter and prints it twice.*
4. *Use the modified version of do\_twice to call print\_twice twice, passing 'spam' as an argument.*
5. *Define a new function called do\_four that takes a function object and a value and calls the function four times, passing the value as a parameter. There should be only two statements in the body of this function, not four.*

## Task 5

*This exercise can be done using only the statements and other features we have learned in task 4.*

1. *Write a function that draws a grid like the following:*

+ - - - + - - - +

| | |

| | |

| | |

| | |

+ - - - -+ - - -+

| | |

| | |

| | |

| | |

+ - - - -+ - - - +

*Hint: to print more than one value on a line, you can print a comma-separated sequence of values:*

*print('+', '-')*

*By default, print advances to the next line, but you can override that behavior and put a space at the end, like this:*

*print('+', end=' ')*

*print('-')*

*The output of these statements is '+ -' on the same line. The output from the next print statement would begin on the next line.*

1. *Write a function that draws a similar grid with four rows and four columns.*

## SECTION 2

# Objectives of the Section

At the end of this lab you will have learnt the following:

In Conditional and Recursion:

* Understand and use the python operators correctly
* Boolean Expression and logical operators
* Conditional and Alternative execution
* Chained and Nested conditionals
* Recursion

In Fruitful Functions:

* Function - Return type and value
* Boolean Function

# Conditionals and Recursion

## Important concepts

**modulus operator:** An operator, denoted with a percent sign (%), that works on integers and yields the remainder when one number is divided by another.

**boolean expression:** An expression whose value is either True or False.

**relational operator:**  One of the operators that compares its operands: ==, !=, >, <, >=, and <=.

**logical operator:** One of the operators that combines boolean expressions: and, or, and not.

**conditional statement:** A statement that controls the flow of execution depending on some condition.

**condition:** The boolean expression in a conditional statement that determines which branch is executed.

**compound statement:** A statement that consists of a header and a body. The header ends with a colon (:). The body is indented relative to the header.

**branch:** One of the alternative sequences of statements in a conditional statement.

**chained conditional:** A conditional statement with a series of alternative branches.

**nested** **conditional**:A conditional statement that appears in one of the branches of another conditional statement.

**recursion:** The process of calling the function that is currently executing.

**base case:** A conditional branch in a recursive function that does not make a recursive call.

**infinite** **recursion**:A recursion that doesn’t have a base case, or never reaches it. Eventually, an infinite recursion causes a runtime error.

## Task 6

*Write a function that checks if a number is even or not.*

## Task 7

*Write a function called do\_n that takes a function object and a number, n, as arguments, and that calls the given function n times.*

## Task 8

*Fermat’s Last Theorem says that there are no positive integers a, b, and c such that*

|  |
| --- |
| *an + bn = cn* |

*for any values of n greater than 2.*

1. *Write a function named check\_fermat that takes four parameters—a, b, c and n—and that checks to see if Fermat’s theorem holds. If n is greater than 2 and it turns out to be true that*

|  |
| --- |
| *an + bn = cn* |

1. *the program should print, “Holy smokes, Fermat was wrong!” Otherwise the program should print, “No, that doesn’t work.”*
2. *Write a function that prompts the user to input values for a, b, c and n, converts them to integers, and uses check\_fermat to check whether they violate Fermat’s theorem.*

## Task 9

*If you are given three sticks, you may or may not be able to arrange them in a triangle. For example, if one of the sticks is 12 inches long and the other two are one inch long, it is clear that you will not be able to get the short sticks to meet in the middle. For any three lengths, there is a simple test to see if it is possible to form a triangle:*

*If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a “degenerate” triangle.)*

1. *Write a function named is\_triangle that takes three integers as arguments, and that prints either “Yes” or “No,” depending on whether you can or cannot form a triangle from sticks with the given lengths.*
2. *Write a function that prompts the user to input three stick lengths, converts them to integers, and uses is\_triangle to check whether sticks with the given lengths can form a triangle.*

## Task 10

*Write a function is\_between(x, y, z) that return True  if x ≤ y ≤ z  or False otherwise.*

# Fruitfull Functions

## Important concepts

### temporary variable:

*A variable used to store an intermediate value in a complex calculation*

### dead code:

*Part of a program that can never be executed, often because it appears after a return statement.*

### None:

*A special value returned by functions that have no return statement or a return statement without an argument.*

### incremental development:

*A program development plan intended to avoid debugging by adding and testing only a small amount of code at a time.*

## Task 11

*The Ackermann function, A(m, n), is defined:*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | *A(m, n) =* | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | *⎧ ⎪ ⎨ ⎪ ⎩* | |  |  | | --- | --- | | *n+1* | *if  m = 0* | | *A(m−1, 1)* | *if  m > 0  and  n = 0* | | *A(m−1, A(m, n−1))* | *if  m > 0  and  n > 0.* | | | | |

*Write a function named ack that evaluates Ackermann’s function.  Use your function to evaluate ack(3, 4), which should be 125. What happens for larger values of m and n?*

## Task 12

*A palindrome is a word that is spelled the same backward and forward, like “noon” and “redivider”. Recursively, a word is a palindrome if the first and last letters are the same and the middle is a palindrome.*

*The following are functions that take a string argument and return the first, last, and middle letters:*

def first(word):

return word[0]

def last(word):

return word[-1]

def middle(word):

return word[1:-1]

1. *Type these functions into a file named palindrome.py and test them out. What happens if you call middle with a string with two letters? One letter? What about the empty string, which is written '' and contains no letters?*
2. *Write a function called is\_palindrome that takes a string argument and returns True if it is a palindrome  and False otherwise. Remember that you can use the built-in function len to check the length of a string.*