ignou
THE PEOPLE'S
UNIVERSITY

Indira Gandhi National Open University
School of Computer and Information Sciences (SOCIS)

BCS-094 Programming using Python

AN INTRODUCTION TO PYTHON 1

Indira Gandhi National Open University
School of Computer & Information Sciences

MCS-094
PROGRAMMING USING PHYTHON

IGNOU is one of the participants for the development of courses for this Programme.

The Open University of Sri Lanka
P. O. Box 10250, Nawala, Nugegoda
Sri Lanka
Phone: +94 112881481
Fax: +94112821285
Email: hdelect@ou.ac.lk
Website: www.ou.ac.lk

Commonwealth of Learning
4710 Kingsway, Suite 2500, Burnaby
V5H 4M2, British Columbia, Canada
Phone: +1 604 775 8200
Fax: +1 604 775 8210
Email: info@col.org
Website: www.col.org

ACKNOWLEDGEMENT BY AUTHORS

The Open University of Sri Lanka (OUSL), Department of Electrical and Computer Engineering (ECE) wishes to thank those below for their contribution to this course material and accompanying videos and screencasts:

AUTHORS

Units 1 and 2	WASN Perera (Lecturer, Dept. of ECE, OUSL)
Unit 3	AP Madurapperuma (Senior Lecturer, Dept. of ECE, OUSL)
Unit 4	PTR Dabare (Lecturer, Dept. of Mechanical Eng, OUSL)
Unit 5	HUW Ratnayake (Senior Lecturer, Dept. of ECE, OUSL)

Content Editor
DN Koggalahewa (Senior Lecturer, Sri Lanka Institue of IT)

Language Editor:
KARD Gunaratne (Lecturer, Dept. of ECE, OUSL)

Reviewer:
KARD Gunaratne (Lecturer, Dept. of ECE, OUSL)

Video presenters:
S Rajasingham (Lecturer, Dept. of ECE, OUSL) M Abeykoon (Software Engineer, Duo Software) SN Muhandiram (Associate Software Engineer, VizuaMatix)

Screencasters:
MHMND Herath (Lecturer, Dept. of ECE, OUSL) M Abeykoon (Software Engineer, Duo Software)

COURSE COORDINATOR

Dr. Sudhansh Sharma,
Assistant Professor
School of Computers and Information Sciences(SOCIS)
Indira Gandhi National Open University(IGNOU),
New Delhi-110068

In this Block the Introduction to the Block along with the Check Your Progress in all the units, and Answers to Check Your Progress Questions are written and added in 2021, By

Dr. Sudhansh Sharma, Assistant Professor
SOCIS, IGNOU, New Delhi-110068

PRINT PRODUCTION

Mr. Tilak Raj Assistant Registrar (Publication) MPDD, IGNOU, New Delhi	Mr. Yashpal Assistant Registrar (Publication) MPDD, IGNOU, New Delhi

April, 2021

© Indira Gandhi National Open University, 2021

ISBN-

Printed and published on behalf of the Indira Gandhi National Open University by Registrar, MPDD, IGNOU, New Delhi

Laser Composed by Tessa Media & Computers, C-206, Shaheen Bagh, Jamia Nagar, New Delhi

BLOCK 1

AN INTRODUCTION TO PYTHON

UNIT 1

Introduction to Basic Programming Concepts with Python 7

UNIT 2

Variables, Expressions and Statements 23

UNIT 3

Control Structures, Data Structures and Linked Lists, Queues 34

UNIT 4

Functions 46

UNIT 5

Strings 57

COURSE INTRODUCTION

Course Content

The course is broken down into units and presented in this course material. Each unit comprises:

- An introduction to the unit content.
- Unit Objectives
- Unit outcomes.
- New terminology.
- Core content of the unit with a variety of learning activities.

Resources

For those interested in learning more on this subject, we provide you with a list of additional resources at the end of each unit; these may be books, articles or web sites. Links to video lectures for each unit are provided.

Course Overview

This course will enable students with basic ICT skills to develop mobile applications with Python. Fundamental programming concepts of Python, theoretical and practical knowledge to develop programs in Python are discussed here. At the end of this course, you should be able to design and develop a mobile solution to solve a real world problem using Python.

Course Overview Video

You may watch this video to gain an overview of the course. URL : https://youtu.be/bNCGLBewCz0

The objectives of this course are:

- to introduce learners to fundamental concepts in Python programming language
- to enable learners to acquire necessary skills to become competent programmers in Python
- to enable learners to use Python when developing mobile applications

Course Outcomes

Upon completion of Programming in Python course you will be able to:

- *write* a simple algorithm with pseudocode and draw a flowchart
- *use* identifiers, variables, constants, operators and expressions in Python language
- *apply* the three basic control structures in programming: sequence, condition, iteration
- *describe* multi paradigm nature of Python programming
- *use* data structures in Python programming
- *apply* design philosophies in mobile application development
- *use* libraries for building graphical user interfaces for mobile applications
- *apply* persistence techniques in mobile application development
- *identify and debug* coding errors in a program
- *write* programs to communicate with other devices

BLOCK INTRODUCTION

Writing programs could be a very creative and rewarding. You can write programs for many reasons like; as your day job, as a hobby, as a help to someone else to solve a problem etc.

Computers are so efficient and contain a large amount memory to perform different activities. If we have a mechanism or a language to instruct the computers (since computers can only understand the machine language) we can use them to perform/support our day to day activities. Interestingly, the kinds of things computers can do best are often the repetitive tasks that humans find boring.

It is essential to master the languages that you can communicate with the computers (programming languages) so that you can delegate some of your work to the computer and save your time. Python is one of the language that caters almost all requirements of the software industry.

Python programming is widely used in Artificial Intelligence, Machine Learning, Neural Networks and many other advanced fields of Computer Science. Ideally, It is designed for rapid prototyping of complex applications. Python has interfaces with various Operating system calls and libraries, which are extensible to C, C++ or Java. Many large companies like NASA, Google, YouTube, Bit Torrent, etc. uses the Python programming language for the execution of their valuable projects.

This block prepares you about the fundamentals of Python Programming, after going through this block you will be able to understand the basic constructs of Python programming language viz. the data types, functions, packages, scripts and modules. You will learn to apply concepts of Python Programming for solving various problems assigned in Activity section of the respective units.

Problem-solving skills are recognized as an integral component of computer programming and in this block the primary focus of this course is to teach the basic programming constructs of Python language. Emphasis is placed on developing the student's ability to apply problem-solving strategies and to implement these strategies in Python as a programming language.

Basically one must explore possible avenues to a solution one by one until s/he comes across a right path to an optimized and efficient solution. In general, as one gains experience in solving problems, one develops his/her own techniques and strategies, though they are often intangible.

This block consists of 5 units and is organized as follows:

Unit - 1 Introduces to basic programming concepts with python

Unit - 2 Introduces the Data structures and Control Statements in Python i.e. Variables, Expressions and Statements

Unit - 3 Provides understanding of control structures, data structures-and linked lists, queues

Unit – 4 Provides understanding of functions in Python.

Unit - 5 Introduces you the concept of Strings in Python.

Happy Programming!!

UNIT 1 INTRODUCTION TO BASIC PROGRAMMING CONCEPTS WITH PYTHON

Structure

1.1 Introduction

1.2 Obejctives

1.3 Terminologies

1.4 Need for programming languages

1.5 Programming languages

1.6 Examples of Programming languages

1.7 What is programming?

1.7.1 Words and sentences

1.7.2 Understanding Interpreters and Compilers

1.7.3 Programs

1.7.4 Correcting errors

1.7.5 The Programming Process

1.8 Getting to know Python

1.9 Features of Python Programming

1.10 Download, installation and run the first program with Python
Video 1: How to download and install Python

1.11 Python for Mobile App Development

1.12 Summary

1.13 References and Further Reading

1.14 Attribution

1.15 Answer to Check your progress

1.1 INTRODUCTION

The purpose of this study unit is to give you the first step towards the programming language while motivating you to create useful, elegant and clever programs. Hence, you will be able to get the first steps towards turning yourself into a person who is skilled in programming.

The next part of this unit will help you to identify features of Python, which is a high-level programming language that is widely used in web development, mobile application development, analysis of computing, scientific and numeric data, creation of desktop GUIs, and software development. Furthermore, at the end of this unit we discuss about mobile application development using Python since it is regarded as one of the easiest programming languages being developed.

1.2 OBJECTIVES

After going through this unit you will be able to:

- • *identify* the importance of learning to write programs.
- *explain* the concepts of programming and the roles of a programmer.
- *identify* the features and the use of Python programming language.
- *write* down the basic steps of solving a given problem

1.3 TERMINOLOGIES

high level language:	Any programming language like C, Java, Python, which is designed to be easy for programmers to remember, read and write.
interpret:	To execute a program written in a high level language by translating it line by line.
compile:	To convert a program written in a high level language into machine understandable form to be executed later
source code:	A program written in a high-level language before compilation.
program:	A set of instructions that specifies how to carry out a task.
algorithm:	A general process, a set of step by step instructions for solving a problem.
debugging::	The process of finding and correcting errors in a program.
semantics:	study of meaning in words in any language including programming languages

1.4 NEED FOR PROGRAMMING LANGUAGES

We live surrounded by computers, mobile phones, tablet PCs and other digital devices. We can think of these digital devices as our "personal assistants" who can make our lives easy. It is said that the computers are built to continuously ask us the question, "What would you like me to do next?"

Writing programs could be a very creative and rewarding. You can write programs for many reasons like; as your day job, as a hobby, as a help to someone else to solve a problem etc.

Computers are so efficient and contain a large amount memory to perform different activities. If we have a mechanism or a language to instruct the computers (since computers can only understand the machine language) we can use them to perform/support our day to day activities. Interestingly, the kinds of things computers can do best are often the repititive tasks that humans find boring.

It is essential to master the languages that you can communicate with the computers (programming languages) so that you can delegate some of your work to the computer and save your time.

1.5 PROGRAMMING LANGUAGES

Only the machine language can be understood by the microprocessors, i.e. binary values (10101 series). There is a set of machine language instructions available for each microprocessor. These machine languages are also called low level languages. It is difficult to write lengthy instructions using binary language due to poor readability, understandability and the maintainability of binary instructions. Therefore, higher level languages have been created using machine language.

Assembly language is a low level language that is similar to the machine language and it uses symbolic operation code to represent the machine operation code. A high-level language is a programming language that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages

These languages are easy to read and understand. The generation of programing languages are shown in the Figure 1.1.

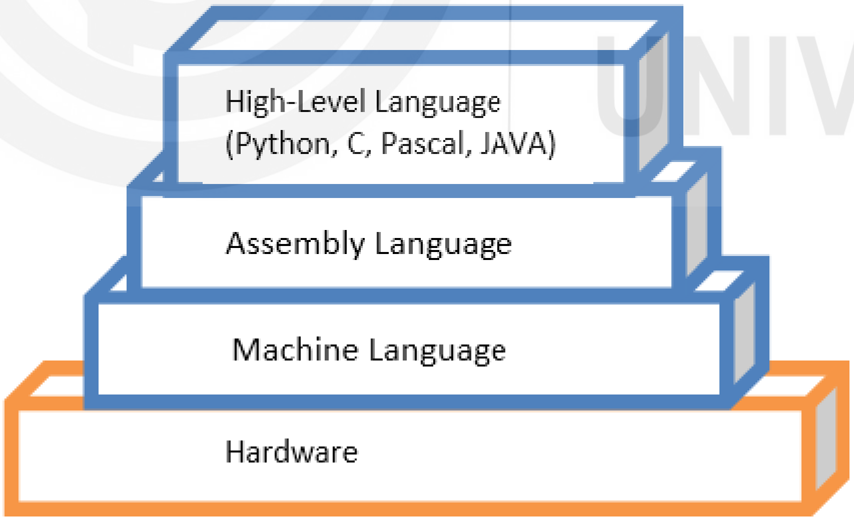


Figure 1.1: Generations of programming languages

A programming language like C is suitable for implementing programs in hardware level (on a micro-controller etc), while some high level languages like Java, C# are really good for large scale software development.

1.6 EXAMPLES OF PROGRAMMING LANGUAGES

C language is a very popular programming language in the electronic and communication field. C is also used to program most microcontrollers. It is a general-purpose programming language initially developed by Dennis Ritchie. C has facilities for structured programming and its design provides constructs that map efficiently to typical machine instructions. C is one of the most widely used programming languages of all time. C compilers are available for the majority of computer architectures and operating systems.

C++ is a general purpose programming language. It has both structured and object-oriented programming features, while it also facilitates some hardware level programming such as low level memory manipulation. It is designed with a bias for systems programming (e.g. embedded systems, operating system kernels), with performance, efficiency and flexibility of use as its design requirements.

C++ has also been found useful in many other contexts, including desktop applications, servers (e.g. e-commerce, web search, SQL), performance critical applications (e.g. telephone switches, space probes) and entertainment software, such as video games.

Java - is a programming language that is concurrent, object- oriented, and specifically designed with less implementation dependencies. It enables the application developers “write once, run anywhere” (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. So we call Java a platform independent language.

Java applications are typically compiled to byte code that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is one of the most popular programming languages in use, particularly for the client-server web applications. Java was originally developed by James Gosling at Sun Microsystem’s (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems’ Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

Python is a dynamic object-oriented programming language that can be compared with Java and Microsoft’s .NET-based languages as a general-purpose substrate for many kinds of software development. It offers strong support for integrating with other technologies, higher programmer productivity throughout the development life cycle, and is particularly well suited for large or complex projects with changing requirements.

Python is also being used in mission critical applications in the world’s largest stock exchange, forms the basis for high end newspaper websites, runs on millions of cell phones, and is used in industries as diverse as ship building, feature length movie animation, and air traffic control. It is a rapidly growing open source programming language. It is available for most operating systems, including Windows, UNIX, Linux, and Mac OS.

Hardware Description Language (HDL) is a specialised computer language used to program the structure, design and operation of electronic circuits, and most commonly, digital logic circuits.

A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis, simulation, and simulated testing of an electronic circuit. It also allows for the compilation of a HDL program into a lower level specification of physical electronic components, such as set of masks used to create an integrated circuit.

A hardware description language looks much like a programming language such as C. It is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time. Two types of popular HDLs are **VHDL** and **Verilog HDL**.

Prolog is declarative programming language where the programmer specifies a goal to be achieved and the Prolog system works out how to achieve it. This language consists of a series of rules and facts. This is a frequently used language in Artificial Intelligence as this is a logic language that is particularly suited to programs that involve symbolic or non-numeric computation.

VisiRule is a graphical tool for designing, developing and delivering business rule and decision support applications, simply by drawing a flowchart that represents the decision logic.

In computer programming' a language or set of commands/instructions that describe the actions are required. Using these instructions machines that can perform a number of different tasks. In order to write instructions to perform something, each action must have a precise, unambiguous meaning. Therefore, all programming languages intended to manage the process of converting the human requirements to a computer solution. A programmer has to play a vital role in establishing the above goals in a successful manner.

Activity 1.1

List four reasons to choose Python as first language. You may need the help of an Internet search engine to attempt this Check Your Progress.

Feedback: This is a peer-reviewed Activity where the learner should share their findings with other learners in class.

1.7 WHAT IS PROGRAMMING?

There are two main skill you need to acquaire to be a programmer.

- You need to know the syntax of programming language with its concepts and keywords. Only then you would be able to write instructions in that language.
- You must also learn to solve a problem logically. If you know how to come to a solution logically, then you can write an algorithm for it, which can be coded in any programming language.

Every programming language has different vocabulary and grammar (syntax) but the concepts and problem-solving skills are universal across all programming languages.

Here, you will learn the vocabulary and grammar of Python language which will enable you to write a program to solve any problem.

1.7.1 Words and Sentences

Just like any other programming language, Python vocabulary is actually very small. This is in contrast to the huge vocabulary in any natural language. This small "vocabulary" is called the set of "reserved words" (key words) in python. These words contain very special meaning to the Python. When Python sees these words in a program, they have a unique meaning to Python. Later as you write programs you will make up your own words that have meaning to your program called **variables**. You have many options in choosing names for your variables, but you cannot use any of Python's reserved words as a name for a variable.

When we train a dog, we use special words like "sit", "stay", and "fetch". When you talk to a dog and do not use any of the reserved words, they just look at you with a quizzical look on their face until you say a reserved word.

For example, if you say, "I wish more people would walk to improve their overall health", what most dogs likely hear is, "blah blah blah **walk** blah blah blah blah." That is because "walk" is a reserved word in dog language. Many suggest that the language between humans and cats has no such reserved words.

The reserved words in the language where humans talk to Python include the following:

And	**from**	**not**	**while**
Del	**elif**	**global**	**with**
As	**or**	**else**	if
assert	**pass**	**yield**	**break**
except	**import**	**print**	**class**
Exec	**in**	**raise**	**continue**
finally	**is**	**return**	**def**
For	**lambda**	**try**	

Unlike a dog, Python is already completely trained. When you say "try", Python will try every time you say it without fail.

We will learn these reserved words and how they are used, gradually on our way through the course. Now, let us focus on the Python equivalent of "speak". A very simple example of how Python speak is given below:

```
print 'Welcome to the world of programmers!'
```

Here we have even written our first Python instruction, syntactically correct. Our instruction starts with the reserved word **print** followed by a string of text enclosed with single quotes.

1.7.2 Understanding Interpreters and Compilers

Python is a **high-level** language intended to be easy for humans to read, write and understand when solving problems with computers. The actual hardware inside a computer does not understand anything written in high-level languages. (Not only Python but all the other high-level languages such as Java, C++, PHP, Ruby, Pascal, JavaScript, etc)

The Central Processing Unit (CPU) inside a computer, understands only **machine language** or **binary code**. Machine language seems quite simple given that there are only zeros and ones in its vocabulary, but its syntax (set of rules for writing a program) is very complex than any high level language. Therefore, instead of writing programs in machine language, we use various translators so that programs can be written in high-level languages like Python, C or Java and then translators can convert these to machine language for actual execution by the CPU.

Since machine language depends on its computer hardware, it is not **portable.** Therefore, a program written in a high-level language need to be translated to a specific machine language using an interpreter or a compiler, for it to be excuted in any machine.

There are two types of Programming language translators:

- Interpreters
- Compilers

An **interpreter** reads the source code, parses (decompose and analyse) the source code, and interprets the instructions line by line at run-time. Python is an interpreted programming language where we can run the souce code interactively. When a line in Python (a sentence) is typed, Python interpreter processes it immediately and we can type another line of Python code.

A **compiler** takes the entire source program as the input, and runs a process to translate the souce code of a high-level language into the machine language. As a result, compiler creates the resulting machine language instructions which can be used for later execution. The compiled output is called an executable file.

Since it is not easy to read or write a program written in machine language, it is very important to have **interpreters** and **compilers** that allow us to translate the programs written in high-level languages to machine languages.

1.7.3 Programs

A program is a collection of instructions that perform a specific task when executed by a computer. Then the computer behaves in a predefined manner. The tasks performed might be something mathematical, but it can also be a symbolic computation or compiling a program. A computer program is written by a computer programmer in a programming language. A program is like a recipe which consists of list of ingreddients (variables) and a list of directions (statements) that tell the computer what to do with the variables.

A few basic instructuons can be found in many programming languages which are not just for Python programs, they are part of every programming language from machine language up to the high-level languages.

input: It collects whatever inputs the program needs to accomplish a task and get it via the keyboard, from a file, or some other device.

output: This gives the answer to the user and display relevant data on the screen, send data to a file etc.

math: This uses mathematical operations such as addition, substraction etc.

conditional execution: There are some condtions and check them and execute the appropriate code

repetition: Write some instructions once and can use it again and again usually with some variation

program logic/content: content of the program which enables the final solution and it may contain the following parts

Any program (complicated or simple) is made up of instructions that look like these. In programming, you can break a large problem in to small sub problems. Those small sub problems could be able to perform using the identified steps above. How this decomposition is done, we will revisit when discussing algorithms.

1.7.4 Correcting errors

Any program can have errors. It is very difficult to write even a small program without errors. It is the responsibility of the programmer to make sure that the code he writes is error free. Programming errors are called bugs and the process of finding them and correcting them is called debugging.

From early days of programming, it was identified that there are three types of defects possible in programs. These are described in detail below.

1) **Syntax errors**

Just like any other programming language, Python can execute a program only if it is written in correct syntax i.e. in the correct grammar of the language. If the syntax is wrong, the interpreter displays an error message indicating where the error is.

e.g. *print (x)* if written as *print(x* without the closing bracket, a syntax error will be displayed Until you get used to Python syntax, you will have to spend a considerable amount of time correcting syntax errors.

2) **Runtime errors**

Runtime errors will appear once the program start running. These are also called **Exceptions.** In simple programs runtime errors are not very common.

3) **Semantic errors**

Semantic errors are the logical problems in a program. If there is a semantic error, most probably the program will not give an error message but will give a wrong output.

e.g. a never ending running program means it is repeating without a terminating condition.

You will learn more about errors, exception handling and testing in later Units as well.

Indentation and comments in Python

Many languages arrange program instructions into blocks using curly brackets **{ }** or **BEGIN** and **END** keywords. These languages also encourage programmers to indent blocks to make the programs readable though indentation is not compulsory. However, Python uses only indentation to delimit blocks, which makes it mandatory to indent program instructions.

Hash (#) symbol is used to write a comment in Python which is used to increase the readability of the program. Comments are not executed/interpreted and used for the purpose of explaining the instructions for the developers and other users.

```
# new block starts with function definition
def add_numbers(a, b):
        # statements are inside this block
        c = a + b
        return c
# an if statement which starts a new block
if (X > 10):
        # one statement inside this block
        print("X is larger than 10")
# this is outside the block
print("Out of 'if' statement")
```

- In many languages we use special characters like semicolon **(;)** to mark the end of each instruction. Python examines only ends of lines to determine whether instruction has ended. Sometime, if we want to put more than one instruction on a line, we can use a semicolon.

```
# individual instructions
print("Hello World!")
print("Here's a another new
instruction")
a = 2

# This instruction spans more than one
line
b = [1, 2, 3,
        4, 5, 6]
# Following is allowed but
recommended to avoid
c = 1; d = 5
```

1.7.5 The Programming Process

The primary concern of programming is to solve a problem which can range from great scientific or national importance, to something as trivial as relieving personal boredom! Here we discussed a basic approach to solve such problems which consists of the following steps:

1) Identify the Problem
2) Design a Solution
3) Write the Program
4) Check the Solution

Identify the Problem:

In the process of identifying the problem first we need to collect the requirements of the given scenario and then analyze the gathered requirements to identify the problem. In the requirement stage, you are trying to work out exactly what your program will be required to do. The next step, which is analysis, looks at the list of requirements and decide exactly what your solution should do to fulfil them. As there are various solutions to a single problem, here your aim is to focus on only the solution that you have selected.

Design a Solution:

This stage focuses on how you're going to turn the previously identified specification into a working program. A design is simply a higher-level description of a list of steps instructing the computer what it should do. This stage does not depend on any special programming language. Usually, special notations like pseudocode or flowcharts are used in the design stage to illustrate problem solution. This step helps a programmer to take the design as an initial step to build a computer program.

Write the program

The three stages of writing a program are Coding, Compiling and Debugging. Coding is the act of translating the design into an actual program using some form of programming language. Compilation is the translation of source code which is written in some programming language into the machine code which can be understood by the processor. Debugging is the process of finding and resolving of defects that prevent correct operation of computer software or a system.

Solution

This step is very important where it tests your creation to check that it does what you wanted it to do. This step is necessary because although the compiler has checked that the program is correctly written, it cannot check whether what you've written actually solves your original problem.

1.8 GETTING TO KNOW PYTHON

Python is a popular high-level programming language used for general-purpose programming, created by Guido van Rossum and first released in 1991. It has wide range of applications such as web development, scientific and mathematical computing, network programming, desktop graphical user Interfaces etc.

About the origin of Python, Van Rossum wrote in 1996:

“Over six years ago, in December 1989, I was looking for a ‘hobby’ programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).”

Activity 1.2

Find applications of Python in different areas. You may need the help of an Internet search engine to attempt this Check Your Progress.

Feedback: This is a peer-reviewed Activity where the learner should share their findings with other learners in class.

1.9 FEATURES OF PYTHON PROGRAMMING

Python's features include:

- Python is intended to be a simple and highly readable language. It is designed to have an uncluttered visual layout, often using English keywords where other languages use punctuation.
- It is a free and open-source software where you can freely use and distribute even for commercial use.
- It ensures portability by providing you to move a Python program from one platform to another, and run it without any changes.
- It is a high-level and interpreted language where you don't have to worry about memory management, garbage collection and so on.
- A large number of standard libraries are available to solve common tasks, which makes life of a programmer much easier since you don't have to write all the code yourself.

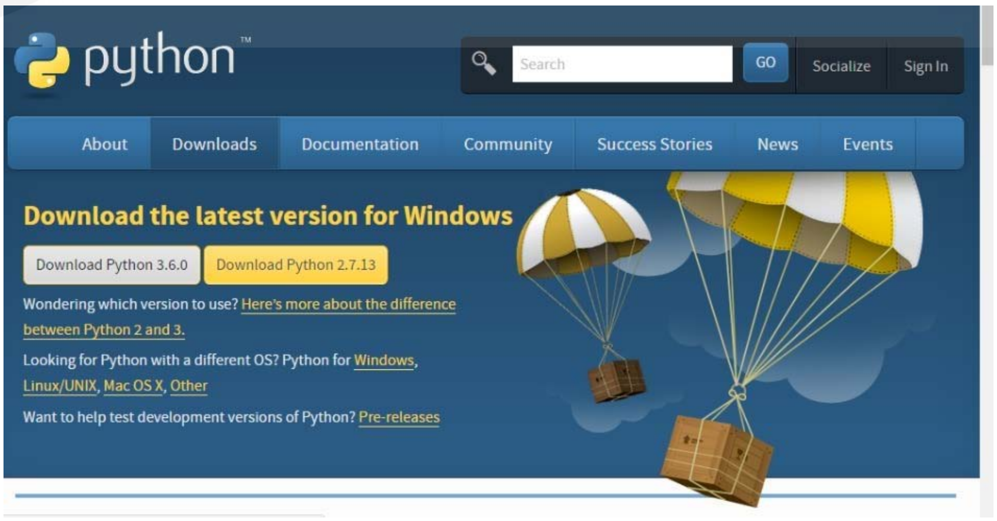
1.10 DOWNLOAD, INSTALLATION AND RUN THE FIRST PROGRAM WITH PYTHON

1) First, go to the https://www.python.org and download Python on your computer. Click on Python Download.

VIDEO 1: HOW TO DOWNLOAD AND INSTALL PYTHON

You may watch this video with screen cast to see how to install Python in your computer. Then install Python in your computer and get ready for coding.

URL: https://youtu.be/T1JBsYZqbOY



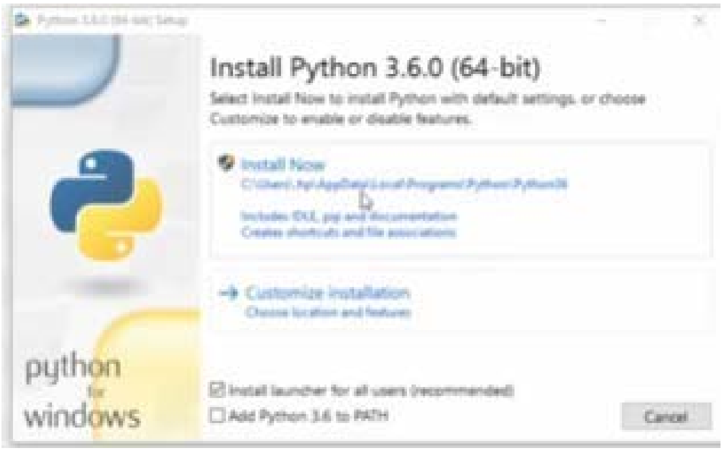
Download the latest version for Windows

Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

2) After downloading, click on the downloaded file. Select "Install now" and select the check box to add Python to the path.


Python 3.6.0 (64-bit) Setup

Install Python 3.6.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

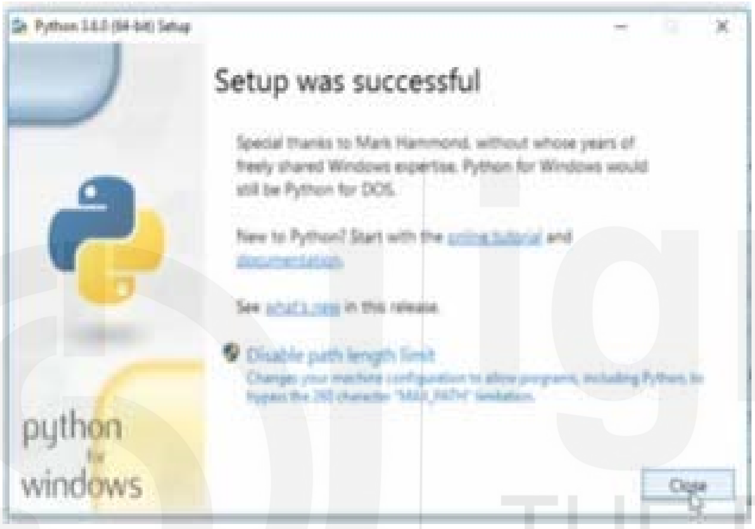
Install Now
C:\Users\...\AppData\Local\Programs\Python\Python36

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.6 to PATH

3) Wait till the installation process complete.


Python 3.6.0 (64-bit) Setup

Setup was successful

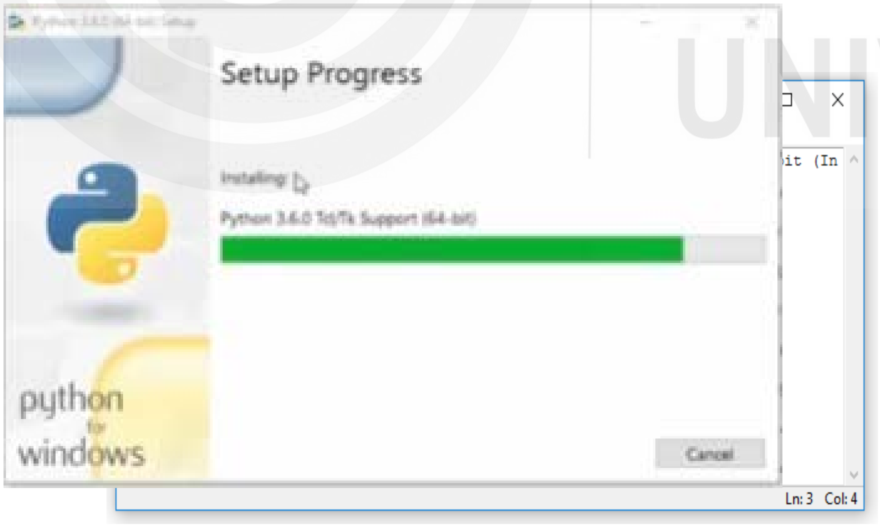
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

4) The setup is complete now. Click on close.


Python 3.6.0 (64-bit) Setup

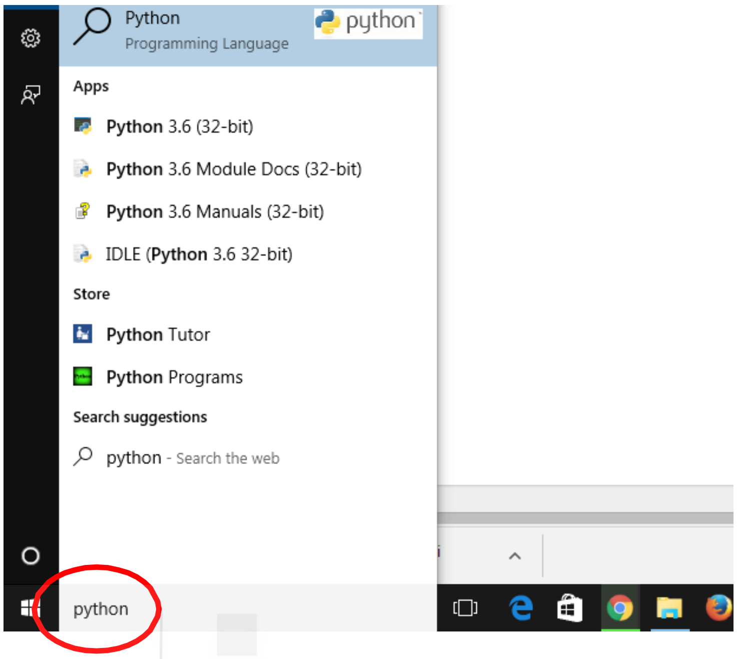
Setup Progress

Installing:

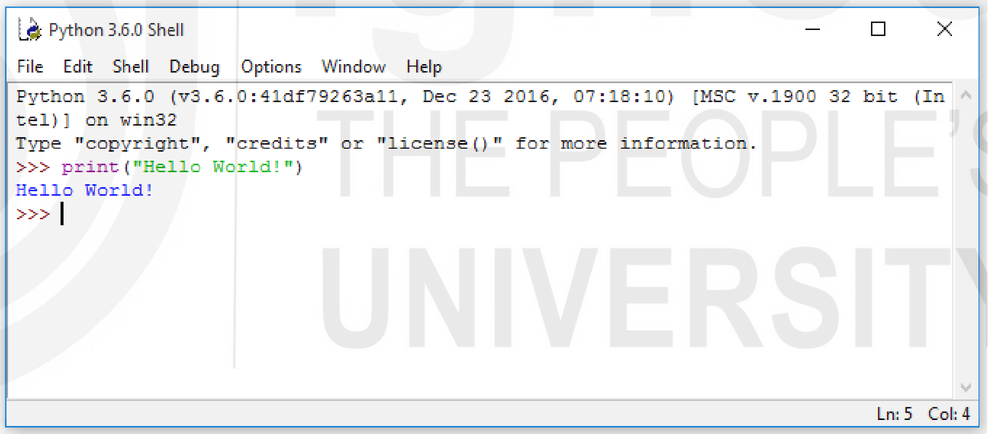
Python 3.6.0 Tcl/Tk Support (64-bit)

5) Search for Python IDLE console for your first program from the start menu.

6) Now you can write your own Python scrip



7) Write your first program with Python.



```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World!")
Hello World!
>>> 
```

To run a program with more than one line you have to write the program in the editor in Python IDLE. A program with more than one statement is called a script. Then save it in the dirctory that you would use to save all programs. It will be saved as a *.py for example myFirstProg.py file.

You can run the script from Windows command prompt typing ***python myFirstProg.py*** if the path to Python.exe file is already set. To set the path you have to go to Window ***settings*** → ***environment*** → ***path*** and add the new path for example something like c:\user\prog\python36-32\python\ Similary for Unix and AppleOS also the path to python need to be set before executing a program.

As a script is a sequence of instructions or statements, as and when a statement is excuted, the result appears one after the other. Write following set of instructions in an editor and run it to see the result.

Note that the assignment statement produces no output. We will discuss scripting again in Unit 2.

```
print('Hello world!') x = 2020
print(x)


output
Hello World! 2020
```

Activity 1.3

Download and install Python as described above. Then set the path and type the given example script and run it.

Check Your Progress

Q-1 What are language translators? Briefly discuss the type of language translators

Q-2 Compare syntax error, semantic error and run time errors. Give suitable example of each

Q-3 Prepare a table to show the point wise comparison between Java and Python

1.11 PYTHON FOR MOBILE APP DEVELOPMENT

Python is a high-level programming language that is widely used in mobile application (commonly called 'app') development. Kivy is a new library that Python programmers can use to program apps. It runs on iOS, Android, MacOS, Windows and Linux. As Python is Object-oriented, it helps to solve a complex problem intuitively. This provides you the ability to divide these complex problems into smaller sets by creating objects.

1.12 SUMMARY

In the first part of this unit, we discussed basics in programming and the necessary skills to be a programmer. The second part of this unit is about the types of programming languages which includes C, C++, Java, Prolog etc. Further, it explains the basic concepts of programming related to Python programming language. Then we discuss about how to download, install and run the first program with Python. At the end of this unit you were introduced to using Python for Mobile App Development.

1.13 FURTHER READING

1) Allen B. Downey (2012). Think Python. Retrieved 20 Dec2016, http://greenteapress.com/wp/think-python/

Download this book for free at

http://greenteapress.com/thinkpython/thinkpython.pdf

2) Python Basics. © Copyright 2013, 2014, University of Cape Town and individual contributors. This work is released under the CC BY-SA 4.0 licence. Revision 8e685e710775.

https://python-textbok.readthedocs.io/en/1.0/Python_Basics.html

1.14 ATTRIBUTION

Some content of this unit – Unit 1, are taken from 'Python for Everyone (PY4E)' site https://www.py4e.com/materials

Copyright Creative Commons Attribution 3.0 - Charles R. Severance

1.15 ANSWERS TO CHECK YOUR PROGRESS

Ans-1 Refer section 1.7.2

Ans-2 Refer section 1.7.4

Ans-3 Refer section 1.6

UNIT 2 VARIABLES, EXPRESSIONS AND STATEMENTS

2.1 Introduction

2.2 Objectives

2.3 Terminologies

2.4 Data types and their values

2.5 Variables in Python

2.6 Differentiating variable names and keywords

2.7 Operators, Operands and Expressions

2.8 Interactive mode and script mode

Video 2: Data Types in Python

2.9 Order of operations

2.10 Comments in Programs

2.11 Unit summary

2.12 References and Further Reading

2.13 Answer to Check your progress

2.1 INTRODUCTION

In this unit we will discuss about one of the most powerful features of any programming language, which is the ability to manipulate variables. Further it focuses your attention towards statements and expressions. A statements is an instruction that the Python interpreter can execute and an expression is a combination of values, variables, operators, and calls to functions.

2.2 OVERVIEW

Upon completion of this unit you will be able to:

- *explain* identifiers, variable, constants, assignment and expressions used in Python.
- *identify* basic concepts of input and output .
- *apply* string manipulation techniques in python.

2.3 TERMINOLOGIES

value: a unit of data such as a number, set of characters etc

variable: A memory location that holds a data unit or a value.

2.4 DATA TYPES AND THEIR VALUES

In any programming language we find values of different data types. Value is an important concept in programming. It could be a letter, a string of characters or a number. Since we relate values to a certain data types, let us look at few examples.

There are different types of values such as:

Integers (int):

```
>>> x = 1
```

Floating point values (float):

```
>>> x = 4.3
```

Complex values (complex):

```
>>> x = complex(4., -1.)
```

String:

```
>>> x = "Hello World"
```

Python supports all primary data types such as integers, floating point numbers and strings. Other than those, it has built-in support for data types such as lists, tuples, dictionaries and complex numbers.

Python interpreter can find the type of a given value using 'type' keyword.

```
>>> type('Hello World !')
<type 'str'>
>>> type(20)
<type 'int'>
```

Python supports a number of built-in types and operations. Though different languages may handle variable types uniquely, most are quite similar. Python variable types, like the rest of the language, are quite simple. Variable types are determined by the value stored within the variable. Unlike most other languages, keywords like "int", "String", or "bool" are not required in Python, as Python supports type inferencing. The most general compatible variable type is chosen.

2.5 VARIABLES IN PYTHON

A variable is something that holds a value which may change later. In simplest terms, a variable is just a box that you can put stuff in. You can use variables to store all kinds of stuff, but for now, we are just going to look at storing numbers in variables. Ability to manipulate variables is a versatile feature in any programming language.

Use of assignment statement (‘= ‘ operator) in Pthon, creates new variables and gives them values as well:

```
>>> student_name = 'Senuri Gamage'
>>> student_no = 5
>>> student_average = 47.561
```

In the above example:

The first assigns a string to a new variable named student_name. The second gives the integer 5 to a new variable named student_no. The third assigns an approximate value for student_average.

A state diagram as shown below can be used to represent variables in a paper. Variable’s current value shows what state each of the variables is in. Below shows the result of the previous example.

```
 student_name -> 'Senuri Gamage'
 student_no -> 5
 student_average -> 47.561
```

The assignment (‘=’) statement links a name, on the left hand side of the operator, with a value, on the right hand side. This is why you will get an error if you enter:

17 = n

When reading or writing code, say to yourself “n is assigned 17” or “n gets the value 17” or “n is a reference to the object 17” or “n refers to the object 17”. Don’t say “n equals 17”.

Unlike many other high level languages, variable types are not declared before assigning values. The type of a variable is decided by the type of the value it is initialized or later refers to. For example;

```
>>> type(student_name)
output
<type 'str'>
>>> type(student_no)
output
<type 'int'>
>>> type(student_average)
output
<type float >
```

2.6 DIFFERENCIATING VARIABLE NAMES AND KEYWORDS

Generally, any pProgrammer is taught to choose meaningful names for variables which helps human readers to understand what they are used for. You may construct variable names to represent what the variable is used for.

Naming Conventions of the variables in Python

Variable names can be quite long and can have both English letters and numbers. However, the variable name must start with a letter. It is adviced to start with a lowercase letter. The underscore character “_” can appear in a name where it can connect multiple names such as student_name and student_number. Variable names are case sensitive. Case sensitivity example: myVariable, myvariable, and Myvariable represent three separate variable names.

```
>>> 3subjects_average = 57
SyntaxError: invalid syntax

>>> class = 'Botany' SyntaxError:
invalid syntax
```

If your variable has an illegal name, it will cause a syntax error:

3subjects_average is illegal because it does not begin with a letter. It seems that class is wrong too. Can you guess why?

class is keyword in Python. Keywords define the language’s syntax rules and structure, and they cannot be used as variable names.

There are 31 keywords in Python version 2 as shown below.

and	del	from	not
	while		
as	elif	global	or
	with		
assert	else	if	pass
	yield		
break	except	import	print
class	exec	in	raise
continue	finally	is	return
def	for	lambda	try

Keywords may change with the new versions and in Python version 3, *exec* is not considered as a keyword. Instead, a new keyword *nonlocal* is added.

When you come across compilation errors with variable names, and it is not clear why, see whether it is on this list of keywords.

2.7 OPERATORS, OPERANDS AND EXPRESSIONS

Symbols that represent some actions are called Operators. The action could be addition of two numbers. The operator performs an action on **operands** such as two numbers.

The operators use +, -, *, / and % to perform computations Addition, Subtraction, Multiplication, Division and Modulus. See the blow examples.

```
>>> x = 2
>>> y = 3
>>> z = 5
>>> x * y
6
>>> x * y + z
11
>>> (x + y) * z
25
```

Division operation results in different results for Python 2.x (like floor division) and Python 3.x. Floor division means after performing the division, result is given as the lower integer to the value. Therefore, in Python 2, when both of the operands are integers, the result is truncated to be an integer.

In Python 2:

```
>>> 10 / 3
3
```

In Python 3:

```
>>> 10 / 3
3.3333333333333335
```

In Python 3, the result is given as type *float*.

If any one of the operands is a floating-point number, Python performs floating-point division, and the result is given as type *float*:

```
>>> 5.0 / 2
2.5
```

In programming, an expression is any legal combination of symbols that represents a value. The expression is consists with values, variables, and operators. In the domain of computing, expressions are written by developers, interpreted by computers and evaluated.

In the expression,

x + 3

x and 3 are operands

+ is an operator

An expression does not necessarily do anything, it evaluates to, that is, reflects, a value. For example, 3 is an expression which evaluates to 3. Following are examples for expressions assuming that the variables has been assigned a particular value.

8888

x

x + 88

One physical line of code is considered as one statement which does something. Most often, one physical line of code will correspond to one statement. Within a statement you can find expressions. A statement is an instruction that the Python interpreter can execute.

2.8 INTERACTIVE MODE AND SCRIPT MODE

There are certain advantages that can be achieved with an interpreted language compared to compiled one. Here, you can test line by line in interactive mode before you write few lines together in a script. However, there are some differences with Interactive and Script mode.

For example,

```
>>> minutes = 25
>>> minutes * 60 1500
```

The first line, the variable minutes represent a value. However, it does not have a visible effect. The second line is an expression where the python compiler is able to interpret. Therefore, it displays the result in seconds for the input value of 25 minutes.

If same lines are typed into a script and executed, you cannot expect the same output. That is because these expressions have no visible effect. However, Python does evaluate all statements but do not display result unless asked with a print statement.

```
minutes = 25
print(minutes * 60)
```

VIDEO 2: DATA TYPES IN PYTHON

You may watch this video first and then attempt Activity 2.1.

URL: https://youtu.be/F_TWZi4rg-0

Activity 2.1

Write following statements one by one to see what they do:

55

x = 55

x + 1

Then write all 3 statements in notepad as a script, run it and observe the output.

Finally, change all expressions into a print statement and then run the script.

2.9 ORDER OF OPERATIONS

In mathematics and computer programming, the order of operations (or operator precedence) is a collection of rules that reflect conventions about which procedures to perform first in order to evaluate a given mathematical expression. Python uses the standard order of operations as taught in Algebra and Geometry classes at high school or secondary school. That is, mathematical expressions are evaluated in the following order (memorized by many as PEMDAS), which is also applied to parentheticals.

Note that operations which share a table row are performed from left to right. That is, a division to the left of a multiplication, with no parentheses between them, is performed before the multiplication simply because it is to the left.

The following Table 2.1 shows the operator precedence in Python from **lowest to highest.** Operators in the same box have the same precedence. Unless the syntax is explicitly given, operators are binary. Operators in the same box group left to right (except for exponentiation, which groups from right to left).

Note that comparisons, membership tests, and identity tests, all have the same precedence and have a left-to-right chaining feature

Table 2.1: Operator precedence

Operator	Description
lambda	Lambda Expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in	Membership tests
is, is not	Identity tests
<, <=, >, >=, !=, ==	Comparisons
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, /, %	Multiplication, Division and Remainder
+x, -x	Positive, Negative
~x	Bitwise NOT
**	Exponentiation
x.attribute	Attribute reference
x[index]	Subscription
x[index:index]	Slicing
f(arguments ...)	Function call
(expressions, ...)	Binding or tuple display
[expressions, ...]	List display
{key:datum, ...}	Dictionary display
\`expressions, ...\`	String conversion

Operator precedence affects how an expression is evaluated. For example, if a = 20, b = 10, c = 15 and d = 5 the value of the following expressions are:

x = (a + b) * c / d; The value of x is 90

x = a + (b * c) / d; The value of x is 50

2.10 COMMENTS IN PROGRAMS

There will always be a time in which you have to return to your code. Perhaps it is to fix a bug, or to add a new feature. Regardless, looking at your own code after six months is almost as bad as looking at someone else's code. What one needs is a means to leave reminders to yourself as to what you were doing.

For this purpose, you leave comments. Comments are little snippets of text embedded inside your code that are ignored by the Python interpreter. The natural language can be used for commenting the code. It can appear anywhere in the source code where whitespaces are allowed. It is useful for explaining what the source code does by:

- explaining the adopted technical choice: why this given algorithm and not another, why calling this given method...
- explaining what should be done in the next steps (the TODO list): improvement, issue to fix...
- giving the required explanation to understand the code and be able to update it yourself later or by other developers.

It can also be used to make the compiler ignore a portion of code: temporary code for debugging and code under development.

The following guidelines are from PEP 8 (Index of Python Enhancement Proposal), written by Guido van Rossum.

- Comments that contradict the code are worse than no comments. Always make a priority of keeping thecomments up-to-date when the code changes!
- Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lower case letter (never alter the case of identifiers!).
- Block comments generally consist of one or more paragraphs built out of complete sentences, with each sentence ending in a period.
- You should use two spaces after a sentence-ending period in multi-sentence comments, except after the final sentence.
- When writing English, follow Strunk and White.
- Python coders from non-English speaking countries: please write your comments in English, unless you are 120% sure that the code will never be read by people who don't speak your language.

Block comments generally apply to some (or all) code that follows them, and are indented to the same level as that code. Each line of a block comment starts with a # and a single space (unless it is indented text inside the comment). Paragraphs inside a block comment are separated by a line containing a single #.

Use inline comments sparingly. An inline comment is a comment on the same line as a statement. Inline comments should be separated by at least two spaces from the statement. They should start with a# and a single space.

Inline comments are unnecessary and in fact distracting if they state the

```
x = x + 1                # Increment x
```

But sometimes, this is useful:

```
x = x + 1                # Compensate for border
```

obvious. Don't do this:

Activity 2.2

There are three assignment statements as follows:

width = 42

length = 14.0

Write the value of the expression given below.

- *width/3*
- *width/3.0*
- *length/3*
- *13 + 25 * 10*

Check your answers using Python interpreter.

Let us try solving a different kind of a problem now. How do you solve an equation in Python?

e.g. if volume of a sphere with radius r is given as $\frac{4}{3}\pi r^3$. What is the

volume of a sphere with radius 10? (Assume π = 3.14)

```
>> volume_sphere = 4/3*3.14 *10***3
```

Activity 2.3

Volume of a cylinder with height h and radius r is given as $\pi r^2 h$. What is the volume of a cylinder with height 6 and radius 2?

Check Your Progress

Q-1 What are primary data types in Python? how do they differ from the built in support for data types?

Ans-1 Refer section 2.4

Q-2 List the different types of Operators in Python, also mention their precedence.

Ans-2 Refer section 2.9

2.11 SUMMARY

The first part of this study unit focused on a very important feature of Python language, which is about variables and their types. Moreover, it discussed about variable naming conventions and Python's keywords. We also discussed about statements which is a 'line of code' that the Python interpreter can execute. A statement could be one or many expressions. An expression is a combination of operands and operators. Furthermore, we were able to identify the types of operators which are supported for Python language.

2.12 FURTHER READING

1) https://www.py4e.com/materials

© Copyright 2013, 2014, University of Cape Town and individual contributors. CC BY-SA 4.0 licence. Revision 8e685e710775.

Copyright CC-BY 3.0 - Charles R. Severance

2) https://python-textbok.readthedocs.io/en/1.0/index.html

3) http://opensask.ca/Python/Overview/VariablesAndDataTypes.html

2.13 ANSWER TO CHECK YOUR PROGRESS

Ans-1 Refer section 2.4

Ans-2 Refer section 2.9

UNIT 3 CONTROL STRUCTURES, DATA STRUCTURES-AND LINKED LISTS, QUEUES

Structure

3.1 Introduction

3.2 Objectives

3.3 Terminologies

3.4 Selection Structure

3.5 Iteration structures

3.5.1 While statement

3.5.2 for statement

3.5.3 Video 3: Control Structures in Python

3.5.4 break, continue and pass Statements

3.6 Data Structures

3.6.1 Video 4: Working with Data Types and Data Structures

3.6.2 List

3.6.3 Video 5: Operators and Control Structures in Python

3.6.4 Range Type

3.6.5 Tuples

3.6.6 Dictionaries

3.7 Summary

3.8 References and Further Reading

3.9 Answer to Check your progress

3.1 INTRODUCTION

In a program, a control structure is a block of code that determines the order in which the statements are executed. It also directs the flow of the program based on the given logic. There are two key control structures, namely, selection and iteration to structure the segments of code. Selection statements allow a programmer to write statements that will be executed based on the satisfaction of some condition while iteration statements allow the repetition of the same set of statements multiple times.

A data structure is a particular way of organising data in a computer that enables efficient use. In this unit we will also discuss common data structures in Python.

3.2 OBJECTIVES

Upon completion of this unit you will be able to:

- *explain* the use of control structures and data structures in a program.
- *identify* appropriate control structures and data structures for a given scenario.
- *apply* suitable data structures to model a solution for a simple problem.

3.3 TERMINOLOGIES

iterative:	repeating
slice:	part of a data structure
string:	a special data type

3.4 SELECTION STRUCTURE

As in many other languages, Python provides an *if ....else* statement. the general syntax of the *if ... else* statement is;

```
if <condition>:
<Statements to be run if the condition evaluates to true>
else:
<Statements to be run if the condition evaluates to false>
```

For example, the following program prints a warning message if the weight of a customer's luggage exceeds the allowed limit of 25 kgs.

Example 3.1:

```
allowedlimit = 25
extraweight=0
weight = float(input("How many kilograms does your” “baggage
weigh? "))
if weight > 25:
   extraweight = weight - allowedlimit
   print(‘Your luggage has ‘,extraweight,’kgs extra’)
   print(‘There will be an extra chrage of’,extraweight*10)
else:
   print("Your luggage weight is within the limit")
print("Thank you for your business.")
```

The above program checks if the weight is above the allowed limit. If it above the limit, it informs the customer the extra payment involved. Otherwise, it will inform that the weight is within the limit. Else clause is used to execute involved in the ‘otherwise’ part of the condition.

If statements can be embedded within one another and are called nested if condition. The level of nesting depends on the logic of the program and you can have any number of nested statements. For example, the following modifies the example 3.1 to allow customers with a small extra weightage to go through without a payment.

Example 3.2

```
allowedlimit = 25
extraweight=0
weight = float(input("How many kilograms does your baggage weigh?
"))
if weight > allowedlimit:
   extraweight = weight - allowedlimit
   if extraweight <5:
      print("Your luggage has ", extraweight, "kgs extra")
      print("Since it is a small value, we will not charge you this
time")
   else:
      print("Your luggage has ", extraweight, "kgs extra")
      print("There will be an extra chrage of ",extraweight*10,"\n")
else:
   print("Your luggage weight is within the limit")
   print("Thank you for your business.")
```

In some cases, we need to do things based on multiple tests. For example, let us say a teacher decided to give grades to her students as follows:

Mark greater than or equal to 80 Excellent

Mark greater than or equal to 65 but less than 80 Good

Mark greater than or equal to 50 but less than 65 Pass

Mark less than 50 Fail

We can write a program to print a grade according to a student's mark with multiple if statements as given in example 3.3a.

Example 3.3 a

```
#Program: Printing grades        mark =
float(input("Enter the
students mark"))
if mark >= 80:
    grade = "Excellent" else:
# check grades for the condition Good
    if mark >= 65:
        grade = "Good"
    else:
# check grades for the condition Pass

    if mark >= 50:
        grade = "Pass"
    else:
# check grades for the condition Fail
        grade = "Fail"
print("Your grade for this”
“subject is:", grade)
```

Example 3.3 b

```
#Program: Printing grades
mark = float(input("Enter the
   students mark"))
if mark >= 80:
      grade = "Excellent"
elif mark >= 65:
      grade = "Good"
elif mark >= 50:
      grade = "Pass"
else:
# grade has to be fail
      grade = "Fail"
print("Your grade for this”
   “subject is:", grade)
```

Python provides another statement called elif, which allows us to write more compact code with less indentation as given in example 3.3b. elif is a short hand for else if. In all of the above else part is optional.

3.5 ITERATION STRUCTURES

Python provides multiple statements to implement iteration, i.e. repeating the same set of instructions with different value sets. **while** statement and **for** statement are two such key statements.

3.5.1 While statement

In a while statement executes a set of instructions if a given condition is true. It takes the form of

```
while <condition>
    <set of statements>
```

Only if the condition is true, then the set of statements is executed. This is how the while loop operates:

Step 1: Evaluate the condition
Step 2: If the *condition is false* continuing the program with the next statement that comes after the set of statements within while condition
Step 3: If the *condition is true* execute the set of statements within while condition
Step 4: Go back to step 1 and evaluate the condition again

The following is an example of the use of the while loop: It prints numbers from 1 to 9 (last number not less than 10). The condition n < 10 will be true for all numbers from 0 to 9. the condition will become false when n becomes 10, because 10 is not less than 10).

```
#while loop n = 0
while n < 10:
     n = n + 1
     print(n)
     print('All numbers printed')
```

As you can see, it is important that when the condition is true, it changes sometime during the execution of the set of statements. Otherwise, it will run for ever (known as an infinite loop). In the above example, if we forget to increase n by 1 during each iteration, it will become an infinite loop.

3.5.2 for statement

Unlike other languages, Python for statement is not based on a counter variable. Instead, for loop iterates through a set of values given as a list or a string. The syntax of the for loop is

```
for <item> in <list>:
    <do something with the item>
```

The loop will iterate through the list or the string assigning one item or character to the control variable, then executing the block of statements using that value.

```
import math
numberlist = [1,2,3,4,5,6]
for targetvalue  in numberlist:
   print(targetvalue, "squared is",int(math.pow(targetvalue,2)))
```

In the example, control variable is named *targetvalue*. Starting with the first value in the list, it will take values in the list in the given sequence. In this example, it will take values 1,2,3,4,5 and 6 in respectively. During each iteration it will print the square value of the control variable.

3.5.3 VIDEO 3: CONTROL STRUCTURES IN PYTHON

You may watch this video on Control Structures in Python

Before attempti ng the Activity 3.1.

URL: https://youtu.be/5ogPFfF8wGw

Activity 3.1

Write a program using a while loop to execute even numbers starting from 20 to 60.

3.5.4 break, continue and pass Statements

The break statement ends execution of the nearest enclosing loop or conditional statement in which it appears. It can be used to terminate or change the ongoing iteration. break statement is commonly used with the looping statements. In while and for loops, it terminates the nearest enclosing loop, skipping the optional else clause if the loop has one. If a for loop is terminated by break, the loop control target keeps its current value. When break passes control out of a try statement with a finally clause, that finally clause is executed before really leaving the loop.

Another useful statement is the continue statement. When the continue statement is found, It continues with the next cycle of the nearest enclosing loop. When continue passes control out of a try statement with a finally clause, that finally clause is executed before really starting the next loop cycle.

Some examples of the use of break and continue statements in while and for loops are shown below.

```
n = 0
while n < 10:
    n = n + 1
    print(n)
        if n == 5:
        break
print('All numbers printed')
```
	```
	#while loop
	n = 0
	while n < 10:
	    n = n + 1
	    if n % 2 == 0:
	        continue
	    else:
	        print(n)
	```

Print all numbers from 1 to 4. Though the condition to check is n < 10, when n is 5 the break statement will go to the next statement after the while statement.	This program will only print odd numbers. When an even number is found, it will go to the next iteration of the loop, starting at the whilestatement.

This program will print only numbers from 1 to 4. When n becomes 5 it will finish the current while statement and run the next statement which is print('All numbers printed').

This program will print numbers from 1 to 10 only if the number is odd. When the number is even (i.e. the remainder when the number is divided by 2 is 0), it goes to the next iteration. When n becomes 5 it will finish the current whilestatement and run the next statement which is print('All numbers printed').

There is one other statement known as the pass statement which is used when the body segment in a program is empty. That is, when a statement is required to be written so that the program syntax is correct, but no action is

```
if x < 0:
pass # need to handle negative values!
```

required. In such cases a passstatement is used as follows.

or

```
>>> while True:
... pass # Busy-wait for keyboard interrupt (Ctrl+C)
...
```

3.6 DATA STRUCTURES

Python has a rich repertoire of data structures that allow the manipulation of sets of data easily and efficiently. Here we discuss some of the basic data structures such as List, Tuple, Dict.

At this point, it is useful to understand two types of data: **immutable** and **mutable**. Immutable data are values that cannot be changed once created. - For example, int, float, long, str, tupe etc are immutable data types whereas list, set and dict are mutable data types. It is important to understand that only mutable objects support methods that change the object in place, such as reassignment of a sequence slice, which will work for lists, but raise an error for tuples and strings.

VIDEO 4: WORKING WITH DATA TYPES AND DATA STRUCTURES

You may watch this video with a screen cast to see how to work with Python Data Types.

URL: https://youtu.be/1lwY4eaI-0o

A slice is a part of a data item such as a list and the range of the slice is marked using the notation [start index: stop index]. Slice operator (:) also has an extended version that includes a stride parameter as follows: [start index: stop index: stride]. The stride value is similar to the step value in range statement.

Example 3.4

```
# Slicing lists
characters =
['a','b','c','d','e','f','g','h','i','j','k','l','m','n']
simpleslice = characters[3:8]
print('Some characters are: ', simpleslice)
slicewithstride = characters[3:8:2]
print('Slicing with stride parameter:',slicewithstride)

output
Simple slice:  ['d', 'e', 'f', 'g', 'h']
Slicing with stride paramter:  ['d', 'f', 'h']
```

In the above, the Simple slicing extracts the characters from elements at index 3 to element at 7 (upper index - 1). Remember that indexing starts with 0.

Slicing with strid parameters extracts every 2nd character from elements at index 3 to element at 7 (upper index - 1).

Since these parameters are optional, omitting some characters will work as follows:

characters[start:end] # items start through end-1

Table 3.1 slicing operator examples with a list

characters[start:]	items start through the rest of the array
characters[:end]	items from the beginning through end-1
characters[:]	a copy of the whole array
characters[-1]	last item in the array. - sign indicates extracting from the end of the list.
characters[-2:]	last two items in the array
characters[:-2]	everything except the last two items

The following list of operations with mutable objects is given in official Python documentation.

Table 3.2 list of operations with mutable objects

s[i] = x	item i of s is replaced by x
s[i:j] = t	slice of s from i to j is replaced by the contents of the iterable t of the same size
del s[i:j]	same as s[i:j] = []
s[i:j:k] = t	the elements of s[i:j:k] are replaced by those of t
del s[i:j:k]	removes the elements of s[i:j:k] from the list
s.append(x)	appends x to the end of the sequence (same as s[len(s):len(s)] = [x])
s.clear()	removes all items from s (same as del s[:])
s.copy()	creates a shallow copy of s (same as s[:])
s.extend(t) or s += t	extends s with the contents of t (for the most part the same as s[len(s):len(s)] = t)
s *= n	updates s with its contents repeated n times
s.insert(i, x)	inserts x into s at the index given by i (same as s[i:i] = [x])
s.pop([i])	retrieves the item at i and also removes it from s
s.remove(x)	remove the first item from s where s[i] == x. Raises an error when the value is not found.
s.reverse()	reverse the items of s in place. Note that it does not return the reversed list.

These operations are applicable to mutable data types.

3.6.1 List

A list is a data structure that can represent an ordered set of items. Lists are said to mutable in that the size of the list and the items within it can be changed after creating a list.

The format for a list is a set of items enclosed by square brackets [ ].

Example 3.5

```
List of strings:['Sri Lanka', ' India', 'Pakistan',
'bangaladesh', 'Nepal']

List of numbers:[1,3,5,7,11,13]

A list of mixed items:['Saman', 45, '-34.67, 'kumari']

The empty list:[]
```

We have already seen how we can iterate through values in a list using a for statement. Other operations that can be done on lists were described in the previous section under Data Structures.

VIDEO 5: OPERATORS AND CONTROL STRUCTURES IN PYTHON

You may watch this video on Operators and Control Structures in Python before attempti ng the Activity 3.2.

URL: https://youtu.be/qxw690359YY

Activity 3.2

Write a program using a **for** loop to print the cube value (power of 3) of five numbers given in a list.

3.6.2 Range Type

Range is a built in type that returns an arithmetic sequence. In versions prior to version 3, range was a function that returned a list. Range generates a list of numbers upto, but not including, the stop value in the range statement. The general formats of range are;

```
range[stop].
range([start], stop [,step]) : where values for
[start] and [,step] are optional.
```

All values for start, stop and step must be positive or negative integers. If value for start is not specified, the default values of 0 is assigned. For step the default value is 1.

For example:

```
range(5)  will generate 0,1,2,3,4
range(5,10)  will generate 5,6,7,8,9
range(0,10,3)  will generate 0,3,6,9
```

range type is commonly used in for loops to iterate through a known sequence of values.

Example 3.6: Repeat a sequence 10 times, the following uses the range type to generate numbers from 0 to 9 and then print each one.

```
for item in range(10):
# range will generate values from 0 to 9 print(item)
```

Example 3.7 : To iterate over a list using the index value of items in the list

```
cities = ['Colombo', 'Kandy', 'Jaffna', 'Matara',
'Anuradhapura']
for item in range(len(cities)):
# range will use the length of the list

print('City in position ',item,' of the list is ',
cities[item])
```

If you want to convert the arithmetic progression generated by range to a list, a conversion can be used.

Example 3.8 : Convert values generated by range into a list of value

```
valuelist = list(range(5)
Now, valuelist = [0,1,2,3,4]
```

3.6.3 Tuples

Tuples are similar to lists in that they collect related data together. A tuple is of fixed size and is immutable, i.e. cannot be changed once created. Tuples are useful when you want to create a record structure of related data items. Tuples are enclosed within brackets. For example,

```
aruna = ('Aruna', 'Silva', 1978, '747872567').
```

You can use indexes to select items from a tuple as in a list or a string. for example aruna[2] will return 1978.

Tuples are immutable objects, so you cannot change the values of a tuple, for example, the following statement is illegal.

aruna[2] = 1979

However, we can create a new tuple using the values from the existing tuple, and adding some extra values given as a tuple

for example,

aruna = aruna[0:1] + (1979,) + aruna[3:]

will assign the tuple ('Aruna', 'Silva', 1979, '747872567') to the variable aruna. Note that a tuple with a single value has to be ended with a comma after the value.

One of the very powerful operation with tuples is the tuple assignment. It allows values in a tuple given in the right side of the assignment operator to be assigned to a set of variables given in the left.

for example

```
(firstname, lastname, birthyear, phone) = aruna
<will do the following assignments>
firstname = 'Aruna', lastname = 'Silva', birthyear =
1979, phone = '747872567'
```

Remember that the number of variables on the left side should be the same as the number of values in the tuple given in right.

Activity 3.3

Write a program to create a list with a car details and display them.

3.6.4 Dictionaries

Dictionaries are another useful data structure. It consists of a set of

key: value pairs enclosed within curly brackets {}. For example, the following is a list of telephone numbers:

```
phonebook = {'Saman':112847365,
'Iresha':772726286,'Aziz':1124273254,
'Bhavani':717976475}
```

We use an index to access value from a list, string or a tuple, values of

Access a value from a dict	phonebook['Aziz']
New values can be assigned to a dict	phonebook['Harry'] = 779274946
Values from a dict can be deleted with del	del phonebook['Saman']

Keys and values can be assigned to lists	names = list(phonebook.keys())
	phonenos = list(phonebook.values())

Check if a key is in the dict	'Iresha' in phonebook:
Use an expression to create a dict	(number:number** for number in (1,2,3,4,5)

Both key and value from a dict can be retrieved with a loop using the items() method of dict	for key,value in phonebook.items():
	print(key,value)

Table 3.3 Way to access values from a list, string or tuple

Check Your Progress

Q-1 Write a program in Python to place a sports person in the right category Viz. Beginner Level, Middle Level, Expert Level; as per the performance score of the person entered by the Coach.

Q-2 Compare the break , continue and pass statements, give suitable example of each

Q-3 What are mutable data structures in Python? How do they differ from the immutable data structures? List all mutable and immutable data structures in Python.

3.7 SUMMARY

In this unit you learned the control structures that determine the execution of a program. We discussed the two main control structures, selection and iteration providing appropriate examples on how to program in Python. We also discussed the manupulation of simple data structures that is available in Python

3.8 FURTHER READING

1) 1. Allen B. Downey (2012). Think Python. http://greenteapress.com/wp/think-python/

2) https://www.ibiblio.org/g2swap/byteofpython/read/index.html

3.9 ANSWER TO CHECK YOUR PROGRESS

Ans-1 Refer section 3.4

Ans-2 Refer section 3.5.4

Ans-3 Refer section 3.6

UNIT 4 FUNCTIONS

4.1 Introduction

4.2 Objectives

4.3 Terminologies

4.4 Why Functions?

4.5 How to write a function definition in Python

4.6 Key things to remember when defining functions

4.6.1 How to call a function in Python

4.7 Flow of execution

Video 6: Introduction to Functions

4.8 Functions with arguments

4.8.1 Different Argument types used in Python

4.9 Void Functions

4.10 Functions with return statements

4.11 Built-in functions

Video 7: Working with Functions

4.12 Type Conversion functions

4.13 Math functions

4.14 Unit summary

4.15 References and Further Reading

4.16 Answer to Check your progress

4.1 INTRODUCTION

Functions in Python help you to reuse the code and make your coding more reliable.

4.2 OBJECTIVES

Upon completion of this unit you will be able to:

- *describe* the importance of functions in Python
- *explain* the function definition in Python
- *use* the void functions and return statements
- *explain* the difference between different function argument types and use them when defining Python functions

- *select* built-in functions in Python to write programs in Python

4.3 TERMINOLOGIES

Header:	The first line of a function
Body:	The set of Python statements written inside a function definition
Void function:	A function that does not return any value
Flow of execution:	The order in which the Python statements are executed during a program run

4.4 WHY FUNCTIONS?

In the previous unit you may have learned some Python programming. Sometimes you may have noticed that certain tasks must be repeatedly carried out and so, some Python statements must have been repeated without you noticing. A way to correct this problem is to introduce functions to your coding. By introducing functions you can reuse your coding which were already written.

4.5 HOW TO WRITE A FUNCTION DEFINITION IN PYTHON

To introduce a function, you need to extract commonly used sequences of steps into one body of coding and label it as a function.

The header is the first line of the function definition and the body is the rest of the function definition.

A function definition consists of the keyword *def* in the **header** followed by the function name which needs to be followed by a sequence of parameters enclosed in parentheses. The header has to end with a colon after these parentheses. If the parentheses are left empty it means that this function does not take any arguments.

The inputs to the functions are known as **arguments** which can be either a value or a variable. The function body may consist of many Python statements which are indented.

Remember that you need to always define the function before it is first called.

Example 4.1

```
def print_twice():
        print "Hello"
        print "Hello"
```

Line 1 is the header of the function. It state that the name of the function is *print_twice* and there is no argument accepted by this function.

Line 2 and Line 3 constitute the body of the function which will perform the task of printing.

Output of the program written for Example 4.1

Hello

Hello

4.6 KEY THINGS TO REMEMBER WHEN DEFINING FUNCTIONS

- first character of a function name cannot be a number or a special character
- Python keywords are reserved and cannot be used as the name of a function
- Avoid giving the same name to a variable and a function
- To end a function definition simply enter an empty line
- Statements inside a function will not execute until the function is called
- A function definition will not generate any output left uncalled

4.6.1 How to call a function in Python

Simply call the function using the function name. If any arguments are there then you need to write the arguments as well.

To call function written in Example 1 just type the function name as follows

print_twice()

4.7 FLOW OF EXECUTION

The order in which the Python statements are executed by the interpreter is known as flow of execution. Execution of Python statements in a function starts with the first one, and carried out in a sequence one after the other, from start to end.

In the middle of a function if a function call appears then the interpreter will jump to this function execute it and come back to original function.

Video 6: Introduction to Functions

This video will further explain the basic concepts of function in Python.

URL https://youtu.be/XHAA_tCkfXM

4.8 FUNCTIONS WITH ARGUMENTS

Consider the following function definition. This function is written so that it will accept any value as an argument and print the value twice.

Example 4.2

```
1.  def  print_twice( myVariable ):
2.         print(myVariable)
3.         print(myVariable)
```

Line 1 is the header of the function. It states that the name of the function as print_twiceand the argument is myVariable

Line 2 and Line 3 are the body of the function which performs the task of printing.

The above function can be called with any argument value as shown below.

```
4.   >>> print_twice('Hello')
5.   >>>Hello
6.   >>>Hello
7.   >>> print_twice('Now I know how to define a function
     in Python')
8.   >>>Now I know how to define a function in Python
9.   >>>Now I know how to define a function in Python
10.  >>>print_twice(21)

11.  >>>21

12.  >>>21

13. >>>print_twice( '#' * 5)

14.  >>>#####

15.  >>>#####

16.  >>>x = ' Python programming is easy'
```

In Line 10 this function uses an expression as an argument.

Lines 13, 14, 15 and 16 depict how the same function can be used with a variable as an argument. In this case the variable is *x* and its value is 'Python programming is easy'.

4.8.1 Different Argument types used in Python

There are 4 argument types which are commonly used when writing Python codes.

- Default arguments

- Required arguments
- Keyword arguments
- Variable-length arguments

Default arguments

Consider the following example which demonstrate the use of default arguments.

Example 4.3:

```
1.  def find_average(x, y= 12):
2.     average = (x + y)/2
3.     print('The average of', x , 'and',y, 'is', average)
4.  def main():
5.     find_average(4, 6)
6.     find_average(4)
7.  main()
```

The output of the above program is:

```
The average of 4 and 6 is 5.0

The average of 4 and 12 is 8.0
```

Line 5 and 6 both call the function *find_average* with arguments. Line 6 uses the default argument, as the function is called with only one argument.

In the above Example the result 8 is given as the function uses the default argument 12 for the variable y.

Required arguments

As it name implies required arguments expect to be called matching with exactly the function definition. Consider the following example which demonstrate the use of required arguments.

Example 4.4

```
1.  def print_twice( myVariable ):
2.      print myVariable
3.      print myVariable
4.  def main():
5.      print_twice( )
6.  main()
```

Line 5 of the above program calls the function *print_twice()* without any arguments.

The above program will give the following result:

```
TypeError: print_twice() takes exactly 1 argument (0 given)
```

Keyword arguments

With the use of the Keyword arguments, the programmer is able to call the function with arguments in any order and still the interpreter will match the values for the arguments and execute the program accordingly.

Consider the following example which depicts the use of Keyword arguments:

Example 4.5

```
def printNumbers( x , y ):
     print'Value of x variable is' , x)
     print('Value of y variable is', y)
def main():
printNumbers( y = 3, x = 6)
main()
```

The above program gives the following output:

```
Value of x variable is 6 Value of y variable is 3
```

Variable-length arguments

When programming you may come across situations where you will not be aware of the number of arguments involved with a certain function when defining the function. Still you can use such functions with the variable-length arguments as shown in the following example.

Example 4.6

```
def add(*args):
    i = 0
    for j in args:
        i = i + j
    print(i)
add(3,6)
add(2.3,5)
add(3,6,8)
add(3,6,8,12)
```

The output of the above program is

```
9

7.3

17

29
```

The important thing to remember here is the use of the single-asterisk in front of the argument, *args.

Activity 4.1

Write a Python program which will accept a number within the range of 1 and 20, and find whether it is a prime number or not. If it is a prime number, function should return Trueand if not False.

Note that prime numbers are numbers which are divisible only by 1 and by the number it-self. What is the argument type that you use here?

4.9 VOID FUNCTIONS

In Example 1 and 2 we were working with **void**functions. voidfunctions simply carry out the tasks inside the function but will not return any value to where it is called.

4.10 FUNCTIONS WITH RETURN STATEMENTS

To return any result from a function we need to use the returnstatement inside the function body.

Consider the following example written to find the square root of any number.

Example 4.7

```
1.  def   squareN ( y );
2.                squareValue = y × y
3.                return squareValue
4.
5.  answer = squareN(5)
6.  print(answer)
```

The output of the above program is 25. You can see that the print statement is written outside the function *squareN.*

Line 1 is the header of the function and it takes *y* variable as the argument.

Line 2 executes the mathematical operation to find the square root of the given number.

Line 3 return the value of the *squareValue*

Line 4 depicts that it is end of the function

Line 5 calls the function *squareN* and assigns the result returned by the function

Line 6 prints the *answer*of the result

4.11 BUILT-IN FUNCTIONS

The functions that we used in earlier examples are all **user defined functions**. Python has **built-in functions** which we can use by simply calling them by their names and relevant arguments. You do not have to define these **built-in functions.**

Consider the built-in function *len* which gives the length of a string as its output.

Example 4.8

```
len('Now I know how to use functions in Python')
```

This function will give the output *33*

Built-in functions can be dealing with strings, numbers and/or mathematical operations.

Some built-in-functions and their functionality are listed below. For the complete list of built- in functions available in Python, visit the web page https://docs.python.org/3/library/functions.html

- abs(x) :- Return the absolute value of a number. Here the argument x may be an integer or a floating point number. When the argument is a complex number, the function will return its magnitude.
- bool([x]):- Return a Boolean value, either True or False. Here if x value is false or omitted, this returns False; otherwise the function bool will return True.
- chr(i):- Return the string representing a character whose Unicode code point is the integer i. For example, if i value is 112 then the chr(112) returns the string 'p' and if the i value is 169 then the chr(169) returns the string '©'.
- divmod(a, b):- Return a pair of numbers consisting of their quotient and remainder of an integer division. This function accepts two (non complex) numbers as arguments. If the operand types are mixed, then the rules for binary arithmetic operators apply.
- max(iterable, *[, key, default]):- Return the largest item in an iterable argument.
- max(arg1, arg2, *args[, key]):- Return the largest of two or more arguments.
- min(iterable, *[, key, default]):- Return the smallest item in an iterable argument
- min(arg1, arg2, *args[, key]):-Return the smallest of two or more arguments.
- round(number[, ndigits]):- Return number rounded to ndigits precision after the decimal point. If ndigits argument is omitted or is None, then the function returns the nearest integer to its input number.

VIDEO 7: WORKING WITH FUNCTIONS

You may watch this video with a screen cast to see how to write functions in Python before attempting Activity 4.2.

URL https://youtu.be/P8ZDJ2876NA

Activity 4.2

Using built-in functions available in Python write a program to round the number 397.234567 to 3 decimal places.

4.12 TYPE CONVERSION FUNCTIONS

One type of built-in functions that are available in Python are type conversion functions. These functions convert values from one type of values to another type.

- The *int* function converts any value to an integer but it does not round the number.
  - *Int('9')*gives the output as *9*
  - *int(9.7)*gives the output as *9*
  - *int(-102)*gives the output as -*102*
  - *int('Introduction to Python')*gives the output as an *error message*
- The *float*function converts integers and strings to floating point numbers
  - *float(88)* gives the output as *88.0*
  - *float('88')* gives the output as *errormessage*
- The *str*function converts its arguments to a string
  - *str(9.7)*will give the output as *'9.7'*

4.13 MATH FUNCTIONS

To execute mathematical operations in Python you can use the math module available in its library. Before we use any math function we need to import this math module to our program by typing *import math*

To perform different tasks, different types of math functions are available such as

- Number representation functions
- Power and logarithmic functions
- Trigonometric functions
- Angular Conversion functions
- Hyperbolic functions
- Constants

After importing the math module to access the functions in it, it is necessary to specify the name of the module and the name of the function. Module name and function name are separated by a **dot**.

Example 4.9

```
ratio = signalPower / noisePower
decibels = 10 * math.log10(ratio)
```

Lambda Function

Lambda functions in Python are unnamed functions which are also known as anonymous functions. These functions can be very helpful when writing short functions which do not need a function definition with the keyword **def**.

Consider the following two Python programs written to get the average of two numbers using a normal function and using a lambda function.

Example 4.10

Without the lambda function	**lambda function**

def find_average(x, y):
    average = (x + y)/2
    print average
def main():
    find_average(4, 6)
main()
 	
5
	print(lambda x, y: (x+ y)/2) (4,6)
	or
	g = lambda x, y: (x+ y)/2
	print (g((4,6))
	
	5

Both gives same output (5).

__Activity 4.3__

Write a lambda function to print the square of a number.

Check Your Progress

Q-1 Write a function to print cube of a number

Q-2 What are lambda functions? How do they differ from the built-in functions?

4.14 SUMMARY

In this unit you learnet the importance of using functions and how to use functions. Further you learnt how to define functions, to write functions with different argument types, describe the difference between the in- built functions and user defined functions, to write user defined functions, to use in- built functions, import math module in to a Python program and to use lambda functions.

4.15 FURTHER READING

1) Allen B. Downey (2012). Think Python.
http://greenteapress.com/wp/think-python/

Download this book for free at

http://greenteapress.com/thinkpython/thinkpython.pdf

2) Swaroop C H , A Byte of Python, https://python.swaroopch.com/l

Download this book for free at

https://python.swaroopch.com/

4.16 ANSWER TO CHECK YOUR PROGRESS

Ans-1 Refer section 4.10

Ans-2 Refer section 4.11 and 4.13

UNIT 5 STRINGS

5.1 Introduction

5.2 Objectives

5.3 Terminologies

5.4 Strings in Python

  5.4.1 String formatting

  5.4.2 Escape Sequences

5.5 String Operations

  5.5.1 Finding the length of a string using len

  5.5.2 Using a Loop to Traverse a String

  5.5.3 String slices

  5.5.4 Strings are immutable

  5.5.5 Substring

  5.5.6 String comparison

5.6 String methods

5.7 Parsing strings

5.8 Unit summary

5.9 References and Further Reading

5.10 Answer to check your progress

5.1 INTRODUCTION

Strings are an important data type commonly used in Python. In this unit we learn how to create strings and manipulate them.

5.2 OBJECTIVES

Upon completion of this unit you will be able to:

- *explain* the structure of a string.
- *apply* basic operations on strings
- *apply* methods in srings to a program

5.3 TERMINOLOGIES

immutable:	An assigned value to a sequence cannot be changed during execution
sequence:	A set of values ordered according to meaning or value
slice:	A part of a string witin a specific range of indices
traverse:	To go through the items in a sequence

5.4 STRINGS IN PYTHON

A string is a sequence of characters and can be created by enclosing characters in quotes.

flower = 'jasmine’

A character can be accessed one at a time with the bracket operator:

```
>>> flower = 'jasmine’ >>> letterOne = flower [0]
```

The second statement extracts the character at index position 1 from the flower variable and assigns it to letterOne variable. In Python, similar to C language, the index is an offset from the start of the string. Offeset starts from 0 and increase, so the offset of the first letter is zero.

value	J	A	s	m	i	n	e
offset	0	1	2	3	4	5	6

Value of the index must be an integer always.

Python has no character type and the characters are treated as strings of length one which each one is called a ‘substring’.

A string can be specified using single quotes or double quotes which means the same in Python.

‘Can you pick some flowers?’ works exactly the same way as “Can you pick some flowrs?”

Multi-line strings can be written with triple quotes and single or double quotes can be given inside triple quotes.

For example you can write;

```
"` Anjana asked “Hello there, are you picking flowers?”
“Yes, do you want any?” answered Mala
…..”’
```

5.4.1 String formatting

Often we need to print a message including text, numbers etc stored in other variables. In such a case string formatting is needed. The % sign which is used to show where the value of variable should be, is called a placeholder.

```
countryName = “Sri Lanka”
flowerName = “Blue water lily”


print(“ Name of my country is %s, “ %countryName)
print(National flower of my country is %s, %flowerName)
```

Example 5.1

The format sequence '%s' is a place holder for a string as shown above, while %d is for an integer as shown below:

```
>>> mark = 50 >>> '%d' % mark '50'
```

There is a special built-in function in Python for string formatting called

format, which allows a string to be constructed from other information.

Example 5.2

```
marks =  75
name = 'Dulani'
print('{0} has obtained {1} marks for Physics last
year'.format(name, age))
```

5.4.2 Escape Sequences

A character in escape sequence is used to denote a special character or which has been reversed for a special purpose. An escape sequence starts with a backslash '\'.

e.g. inserting a new line in a string;

```
print("Hello I am Neela.\n, This is my friend Anjali")
```

e.g. inserting apostrophes;

```
print("Hello I\'m Neela.\n, This is my friend
Anjali.")
```

Common escape sequences are given in Table 5.1

Table 5.1 A Common set of escape sequence

Sequence	**Meaning**
\\	Literal backslash
\'	Single quote
\"	Double quote
\n	New line
\t	Tab

5.5 STRING OPERATIONS

In Python language, we find many built-in functions which perform operations on strings. Using 'len' function to find length, finding substrings, comparisons and concatenation are few common operations.

5.5.1 Finding the length of a string using len

Using 'len' function we can find out the number of characters in a string:

```
>>> flower = 'jasmine' >>> len (flower) 7
```

However, if you try to get the last letter of a string as given below, you will see an error message:

```
>>> length = len(flower) >>> last = flower[length]
IndexError: string index out of range
```

Though there are seven letters in 'Jasmine', we started counting from zero. Therefore, the seven letters are numbered from 0 to 6. In oreder to find the last character, we should subtract 1 from length:

```
>>> last = flower[length-1] >>> print last
```

Otherwise, there is apossibility to use negative indices, which count backward from the end to the beginning. An expression like flower[-1] would give the last letter of the sring as the output. Similarly, index can be further decreased to [-2], [-3] until [-6].

5.5.2 Using a Loop to Traverse a String

When a string is manipulated, one character is accessed at a time. Usually, we start from the first position, select each character in turn, work with that character, and continue till we reach the last character. Going along a string, inspecting each character is called a 'traversing the string'. String traversal can be easily done with a while loop:

Example 5.3

```
index = 0
while index < len(flower):
      character_in_flower = flower[index]
print character_in_flower
index = index + 1
```

Activity 5.1

Write a 'while' loop that starts at the last character in the string and works its way backwards to the first character in the string, printing each letter on a separate line. You should print the characters in reverse order.

5.5.3 String slices

When talking about strings, a 'slice' means it is part of a string. A string slice may contain one or more characters:

```
>>> mystring = 'MorningGlory' >>> print mystring[0:7]
Morning
```

The operator used to extract a slice, [n:m] returns the part of the string from the character "n" to the character "m". This operator includes the character denoted by "n" but exclude the character denoted by "m".

If the first index (before the colon) is omitted, then extracting the slice starts from the beginning. Similarly,if the second index is omitted, the slice will be extracted from the given position till end of the string:

```
>>> flower = 'jasmine' >>> flower[:3] 'jas' >>> f[3:]
'ine'
```

When specifying a slice, if the first and the second indices are equal or if the first one greater than or equal to the second one, extracted slice would be an empty string:

```
>>> flower = 'jasmine' >>> flower[3:3]
```

A string is called 'empty' if it contains no characters. Length of an empty string is 0.

Activity 5.2

Given that flower is a string, what would be the results if following statements are excuted,

i flower[:],

ii flower[3:7],

iii flower[3:3] ?

5.5.4 Strings are immutable

In Unit 3 we discussed about immutable data types. Strings are also immutable, which means once a value is assigned to a string the value cannot be changed during the execution. Therefore, one item cannot be changed using the [] operator on the left side of an assignment.

Example 5.4

```
>>> greeting = 'hi,there!'

>>> greeting[0] = 'H'


TypeError: ‘str’ object does not support item
assignment
```

Error message above indicates that a new value cannot be assigned to an existing string during the process of execution. If a change is necessary, we

```
>>> greeting = 'hi,there!' >>> new_greeting = 'H' +
greeting[1:] >>> print new_greeting Hi,there!
```

can use string a manipulation function like concatenation to carry out the change we want as shown below.

You may note that in this manipulation the original string is not changed at all.

5.5.5 Substring

The word ‘in’ acts as a boolean operator in Python. It compares two strings and returns True if the first string is a substring of the second:

```
>>> 's' in 'Jasmine'
True
>>> 'e' in 'Mala'
False
```

5.5.6 String comparison

The comparison operators work on two strings to see if the two strings are the same:

```
fruit1 = ‘orange’
if fruit1 == 'orange': print 'correct fruit'
if fruit1 != ‘banana’: print ‘wrong fruit’
```

Comparison operations are also useful for arranging words in alphabetical order.

```
word1 = green
word2 = blue
If word1< word2:
 Print(word1 + “comes before”+ word2)
elif word1 > word 2:
  Print(word1 + “comes after”+ word2)
```

Uppercase and lowercase letters are handled differently in Python and if arranged, all uppercase letters come before lowercase letters. Therefore, where case is not considered, all letters should be converted to lowercase before comparison.

5.6 STRING METHODS

String is an example of a Python object. An object contains both data (the actual string itself) as well as methods (or functions) which are built into the object and are available to any instance of the object.

Use the type function to find out the type of the object as shown below.

```
>>> myFolder = 'findFunctions'
>>> type(myFolder)

Output
<class 'str'>
```

Python has a function called 'dir' which lists the methods available for an object.

```
>>>dir(myFolder)

output

['_add_', '_class_', '_contains__',
'_delattr__', '__dir__', '__doc__', '__eq__',
'_format_', '_ge_', '_getattribute__',
'__getitem_ ', '_ getnewargs_ ', '_ gt  ',
'__hash ', '_ init_ ', '_ init_subclass  ',
' iter ', ' le ', ' len ', ' lt  ',
'_mod__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__',
'__rmod ', '_ rmul_ ', '_ setattr_ ', '_ sizeof  ',
' str ', ' subclasshook_ ', 'capitalize',
'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isascii',
'isdecimal', 'isdigit', 'isidentifier', 'islower',
'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

You can use 'help' to get some information about a function or a method.

```
 >>> help(str.capitalize)
 Output
Help on method_descriptor:
capitalize(self, /)

   Return a capitalized version of the string.
More specifically, make the first character have upper
case and the rest lower case.
```

Refer following website for more documentation for string methods would be https://docs.python.org/library/stdtypes.html#string-methods.

Calling a method is similar to calling a function. But the syntax is different. We call a method by appending the method name to the variable name using the period as a delimiter. A method also takes arguments and return a value.

Let us take another example where the method upper takes a string and returns another string after converting the original string to all uppercase letters:

Method uses the syntax word.upper() in contrast to function syntax which could be upper(word).

```
 >>> string1 = 'angel' >>> string2 = string1.upper()
 >>> print string2
 Output

ANGEL
```

Here the name of the method is upper and the method is applied to string1. Since the method does not take any arguments, empty parentheses are used.

Calling a method is named as invocation. Here we invoke upper method on string1. There is another useful method called find in Python language. In the following example, we invoke find on string1 while passing the letter we search as a parameter.

```
>>> string1 = 'angel' >>> letter = string1.find('g')
>>> print letter
Output
2
```

In fact, findmethod can find substrings, not only characters:

```
>>> string1.find('gel')
Output
2
```

One common task we face usually is to remove white space (spaces, tabs, or newlines) from the beginning and end of a string. For that, we can use use stripmethod:

```
>>> line = ' Hello world ! '>>> line.strip()
Output
>>>'Hello world!'
```

5.7 PARSING STRINGS

Often, we want to look into a string and find a substring. For example if we are presented a series of lines formatted as follows:

```
From nirmal@ ou.ac.lk Fri Jan 2 08:10:110 2016
```

And we want to pull out only the second half of the address (i.e. ou.ac.lk) from each line. We can do this using the find method and string slicing.

First, we find the position of the @ symbol in the string. Then we find the position of the first space after the @ symbol. And then we use string slicing to extract the portion of the string which we are looking for.

```
>>> data = 'nirmal@ou.ac.lkFriJan208:10:102016' >>>
atpos = data.find('@') >>> print atpos 21 >>> sppos =
data.find(' ',atpos) >>> print sp-pos 31 >>> host =
data[atpos+1:sppos]


>>> print host ou.ac.lk >>>
```

We use a version of the find method which allows us to specify a position in the string where we want find method to start searching. When we slice, we extract the characters from "one beyond the @-sign through up to but not including the space character".

The documentation for the find method is available at docs.python.org/ library/string.html

Activity 5.3

Write a program to concatenate two strings *Divya asked,* and *Great! Then can you let me have all the mangos, limes, oranges and apples?* And then remove all punctuations from resulting string.

You may use one '*if then else* condition and a *while* loop

Hint: two strings can be concatenated using '+' operator

Check Your Progress

Q-1 Write Python statements to perform following:

a) Find length of string
b) Slicing of string
c) Finding substring
d) Compare strings
e) Convert uppercase string to lowercase string
f) Convert lowercase string to Uppercase string

5.8 SUMMARY

In this Unit you learned how to define a string, format a string and how to perform string operations on strings. String comparison, finding string length, string concatenation, extracting substrings and string slicing are the main operations that were discussed. We also discussed that strings are immutable and are treated as objects in Python which has its own set of functions.

5.9 FURTHER READING

1. Swaroop C H , A Byte of Python, CC-BY 4.0 International License **Download this book for free at** https://python.swaroopch.com/

2. Allen B. Downey (2012). Think Python. CC-BY-NC

   Download this book for free at
   http://greenteapress.com/thinkpython/thinkpython.pdf

3. Basic String Operations, Retrrieved June 2017 from https://www.learnpython.org/en/Basic_String_Operations

5.10 ANSWER TO CHECK YOUR PROGRESS

Ans-1 Refer section 5.5 and 5.6