

Python Programming Course

Hand Notes 1

Hand Notes for:

Installation & Getting Started

Basic Programming (Print, Datatypes, Variables, Type)

Basic Programming (Formatting, Indentation)

Session 1: Getting Started

FEATURES OF PYTHON

Simple: Python is a simple and minimalistic language. This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

Easy to Learn: Python has an extraordinarily simple syntax.

Free and Open Source: Python is an example of a FLOSS (Free/Libre and Open Source Software).

High-level Language: You don't have to worry about machine understands the syntaxes.

Portable: Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features. You can use Python on GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC!

Interpreted: A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it. Python, on the other hand, does not need compilation to binary. You just run the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it. All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer and it just works!

Object Oriented: Python supports procedure-oriented programming as well as object-oriented programming. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

Embeddable: You can embed Python within your C/C++ programs to give scripting capabilities for your program's users.

Extensive Libraries: The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), and other system-dependent stuff. We will discuss more about this in our Packages chapter.

PYTHON HISTORY

It's always good to understand the history of the subject you are about to learn because then we will be able to appreciate the effort better. Python is linked to ABC Programming language. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde & Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. Guido van Rossum faced challenges while working with ABC, that's when he decided to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So he started typing. He created a simple virtual machine, a simple parser, and a simple runtime. He made his own version of the various ABC parts that he liked. He created a basic syntax, used indentation for statement grouping instead of

curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers. And a new programming language Python was born.

Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, Benevolent Dictator for Life (BDFL). Python was named for the BBC TV show Monty Python's Flying Circus.

Python 2.0 was released on October 16, 2000, with many major new features, including a cycle-detecting garbage collector (in addition to reference counting) for memory management and support for Unicode. However, the most important change was to the development process itself, with a shift to a more transparent and community-backed process.

Python 3.0, a major, backwards-incompatible release, was released on December 3, 2008 after a long period of testing. Many of its major features have also been backported to the backwards-compatible Python 2.6 and 2.7.



Fig: Python logo 1990s – 2005

Python 2.6 was released to coincide with Python 3.0, and included some features from that release, as well as a "warnings" mode that highlighted the use of features that were removed in Python 3.0. Similarly, Python 2.7 coincided with and included features from Python 3.1, which was released on June 26, 2009. Parallel 2.x and 3.x releases then ceased, and Python 2.7 was the last release in the 2.x series. In November 2014, it was announced that Python 2.7 would be supported until 2020, but users were encouraged to move to Python 3 as soon as possible.

Note: In this course, we will use Python 3 as our programming language.

INSTALLATION OF PYTHON

Installation of Python Environment requires installation of Python Software and an IDE (Integrated Development Environment). IDE installation is not mandatory and we can do all our programs using Python Shell or IDLE (Integrated Development and Learning Environment) which comes in-built with Python installation. But we prefer other IDEs like PyCharm, Jupyter, etc which gives us more flexibility and power to do write code faster. Lets first look at the Python installation.

Install Python on Windows Steps

Installing and running Python on your personal computer is no difficult task. It involves just a few simple steps:

- 1) Download binaries from python.org
- 2) Install the binaries

1. Download binaries from python.org

Firstly, to install Python Windows you need to download required binaries from the following link:

<https://www.python.org/downloads/>

Download the latest version for Windows

Download Python 3.8.2

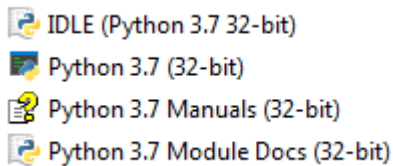
Looking for Python with a different OS? Python for [Windows](#),
[Linux/UNIX](#), [Mac OS X](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#),

2. Download binaries from python.org

Double click on the python.exe installer and follow the instructions. Python Interpreter is now installed on your computer. This will help us do our programs.

You have now installed Python. Now, you can find a list of programs in the Start Menu:



Yes, I am still using Python 3.7 and you will see screenshots from 3.7 but you should see the latest version 😊

Install Python 3 on Windows

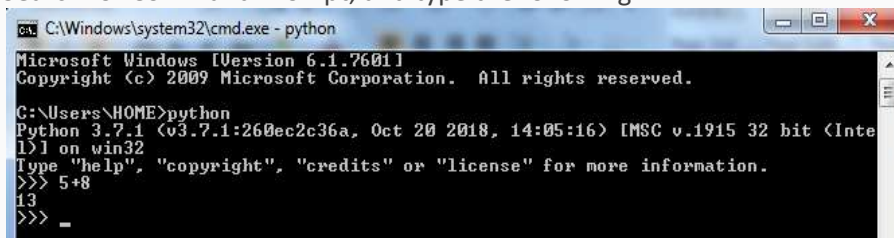
So now, you can reach Python in the following ways:

a. Command Prompt

You can run Python on the command prompt in two ways:

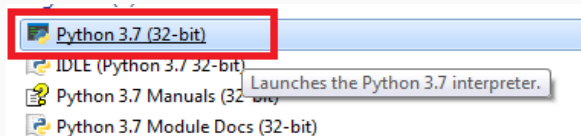
1) The Conventional way

Search for Command Prompt, and type the following:

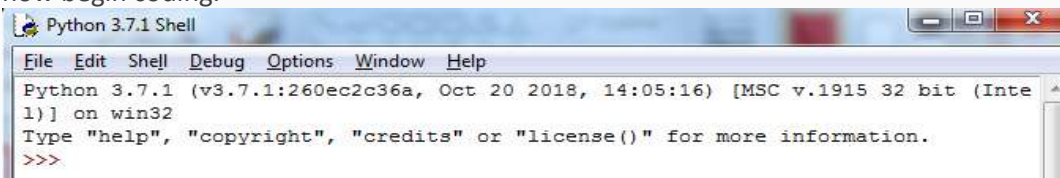


Now, you can use it as an interpreter. As an example, we have calculated 5+8.

2) Running the Program using Python in-built IDLE



Open the Start menu. Click on Python 3.7 (32-bit). This will take you to the command prompt for Python. You can now begin coding.



Python Documentation

Python is very well documented. There might even be copies of documents on your computer, which came with your Python installation.

The home page of Python documentation is at:

<https://www.python.org/doc/>

You can visit this site for complete and updated documentation.

The official Python Tutorial by Guido van Rossum is often a good starting point for general questions.

Getting Help

If you need quick information about any function or statement in Python, then you can use the built-in help functionality. This is very useful especially when using the interpreter prompt. For example, run `help('len')` - this displays the help for the `len` function which is used to count number of items.

```
>>> help('len')
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.

>>> 
```

PYTHON EDITORS AND IDES

We have seen how to run Python command prompt but it is a bad choice because it does not do syntax highlighting and also importantly it does not support indentation of the text which is very important in our case as we will see later. Good editors will automatically do this for us.

Writing Python using IDLE or the Python Shell is great for simple things, but those tools quickly turn larger programming projects into frustrating pits of despair. Using an IDE, or even just a good dedicated code editor, makes coding fun—but which one?

What is an IDE and a Code Editor?

An IDE (or Integrated Development Environment) is a program dedicated to software development. As the name implies, IDEs integrate several tools specifically designed for software development. These tools usually include an editor tailored to handling code (for example, syntax highlighting and auto-completion); build, execution, and debugging tools; and some form of source control.

Python Specific Editors and IDEs

- | | |
|--|--|
| <ul style="list-style-type: none"> • PyCharm • Spyder • Thonny • Eclipse + PyDev • Sublime Text | <ul style="list-style-type: none"> • Atom • GNU Emacs • Vi / Vim • Visual Studio |
|--|--|

PyCharm from JetBrains

Download and Install PyCharm from JetBrains.com website:

<https://www.jetbrains.com/products.html#lang=python>

Select PyCharm Edu version for us to practice and learn. It's free to use:



Professional tool to learn and teach
programming with Python

[Learn](#) | [Teach](#)

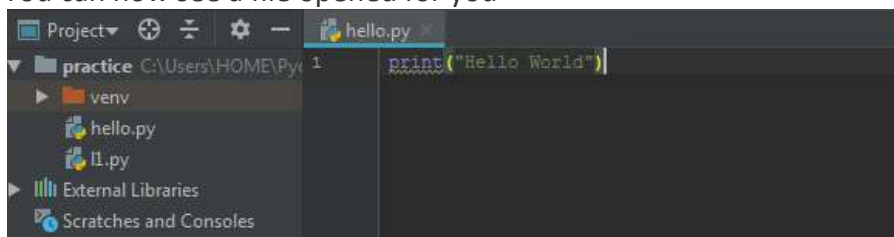
When you open PyCharm, click on Create New Project and then select Pure Python. Change Untitled to a filename. Please refer PyCharm tutorial for detailed:

<https://www.jetbrains.com/help/pycharm/quick-start-guide.html>

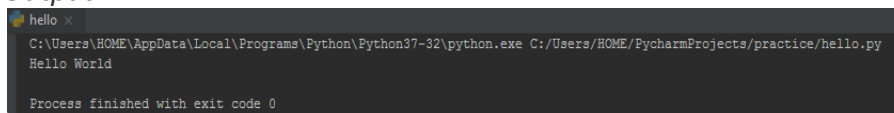
Let's create our first Python Program

1. Using PyCharm editor

- Click the Create button.
- Right-click on the project name (practice is the project name in my example below) in the sidebar and select New → Python File:
- You will be asked to type the name, type hello.
- You can now see a file opened for you



Output:



- Type your program and hit Run
- Select Filename from the list
- Output is shown at the bottom of your screen

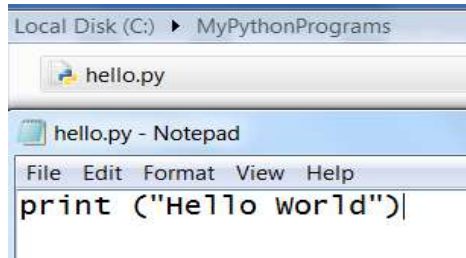
OTHER WAYS TO RUN A PYTHON PROGRAM

Note: We will not do in the class. You can try this yourself

1. Using a source file (Windows Command Line)

- Create a folder where you want to save all your Python programs (I have created a folder at: D:\MyPythonPrograms)

- b. Create a file by name hello.py at this folder. Open with Notepad and type your program as show below:



- c. Navigate to where your python file is, using the commands 'cd' (change directory) and 'dir' (to show files in the directory, to verify your head). For our example something like,

```
> cd D:\MyPythonPrograms
```

try:

```
> python hello.py
```

If you get this message:

```
'python' is not recognized as an internal or external command, operable program or batch file.
```

then python (the interpreter program that can translate Python into 'computer instructions') isn't on your path.

```
> C:\Python37\python.exe hello.py
```

(Advanced users: instead of hello.py, you could write out first.py's full path of C:\MyPythonPrograms\hello.py)

Hopefully I was clear here to show you how to execute Python programs. I use Pycharm to run my programs. There are other popular IDEs which are used by many people. You are encouraged to find out more about them on yourself.

EXECUTING YOUR PROGRAMS ONLINE:

In case you don't want to or cant install all these software on your computer, there exists many websites where you can do the program directly. One such website is:

www.ekapress.org/python

--- End of Session 1 ---

Session 2: Basic Concepts

Let's practice few simple programs to get started.

1. Printing to the screen

```
print("Hello World")  
print(7)  
print("7")
```

ADDING COMMENTS TO YOUR PROGRAMS

Comments are any text to the right of the # symbol and is mainly useful as notes for the reader of the program.

```
## Our first program - using print statement  
print ("Hello World")
```

Use as many useful comments as you can in your program to:

- explain assumptions
- explain important decisions
- explain important details
- explain problems you're trying to solve
- explain problems you're trying to overcome in your program, etc.

CONSTANTS IN PYTHON

Literal Constants

An example of a literal constant is a number like 5, 1.23, or a string like 'This is a string' or "It's a string!" .

Numbers

Numbers are mainly of two types - integers and floats.

An examples of an integer is 2 which is just a whole number.

Examples of floating point numbers (or floats for short) are 3.23 and 52.3E-4 . The E notation indicates powers of 10.

Note: There is no separate long type. The int type can be an integer of any size.

Strings

A string is a sequence of characters. String can be declared in single/double/triple quotes.

Single Quote: You can specify strings using single quotes such as 'Hello World'. All white space i.e. spaces and tabs, within the quotes, are preserved as-is.

Double Quotes: Strings in double quotes work exactly the same way as strings in single quotes. An example is "What's your name?".

Triple Quotes: You can specify multi-line strings using triple quotes - (`"""` or `'''`). You can use single quotes and double quotes freely within the triple quotes. An example is:

```
"""This is a multi-line string. This is the first line.  
This is the second line.  
"What's your name?," I asked.  
He said "Bond, James Bond."  
"""
```

Strings Are Immutable. This means that once you have created a string, you cannot change it

Note: There is no separate char data type in Python.

The format() method

Sometimes we may want to construct strings from other information. This is where the format() method is useful.

```
# Learning Formating Methods
discount = 10
offer = "newsletter"
print("Sign up for our {0} and get {1}% off.".format(offer, discount))
print("Enter your email here to get our {0}".format(offer))
```

Output

```
Sign up for our newsletter and get 10% off.
Enter your email here to get our newsletter
```

A string can use certain specifications and subsequently, the format method can be called to substitute those specifications with corresponding arguments to the format method. The numbers in the format example are optional, so you could have also written as:

```
# Learning Formating Methods
discount = 10
offer = "newsletter"
print("Sign up for our {} and get {}% off.".format(offer, discount))
print("Enter your email here to get our {}".format(offer))
```

What Python does in the format method is that it substitutes each argument value into the place of the specification. There can be more detailed specifications:

```
# Learning Formatting Methods with Numbers
print('{0:.3f}'.format(1.0/3))
# Fill with underscore(_) with text centered
# (^) to 11 width '___hello___'
print('{0:_^11}'.format('hello'))
#Left Align
print('{0:_<11}'.format('hello'))
#Right Align
print('{0:_>11}'.format('hello'))

#keyword based formatting
print('{name} wins {prize}'.format(name='Juan Manuel Santos',
prize='Nobel Peace prize 2016'))
name2 = "Abiy Ahmed Ali"
prize2 = "Nobel Peace prize 2019"
print('{} wins {}'.format(name2, prize2))
```

Output

```
0.333
___hello___
hello_____
_____hello
```

Juan Manuel Santos wins Nobel Peace prize 2016
 Abiy Ahmed Ali wins Nobel Peace prize 2019

PRINT STATEMENT

In the above example, we have seen that print always ends with an invisible "new line" character (`\n`) so that repeated calls to print will all print on a separate line each. To prevent this newline character from being printed, you can end the statement with a comma:

```
print("Hello")
# is for comment  efpogjoqefpgpoefigq nefkjgpoejqjgpofeiug
print(7)          # 7 is a number - no quote
print("7")        # 7 is a text - given in quotes
print(7+6)
print("7" + "6")  #Text in Python called as String
```

ESCAPE SEQUENCES

You cannot specify 'What's your name?' because Python will be confused as to where the string starts and ends. To specify that this single quote does not indicate the end of the string, we use escape sequence. In Python `\`: backslash character is used as escape sequence. Also, you have to indicate the backslash itself using the escape sequence `\\`. What if you wanted to specify a two-line string? One way is to use a triple-quoted string as shown previously or you can use an escape sequence for the newline character - `\n` to indicate the start of a new line. Another useful escape sequence to know is the tab: `\t`. There are many more escape sequences which we will discuss later. One thing to note is that in a string, a single backslash at the end of the line indicates that the string is continued in the next line, but no newline is added.

Examples

```
print ("Example 1:")
print ('what\'s your name?')
print ("Example 2:")
print ('This is the first sentence. \n This is the second sentence.')
print ("Example 3:")
print 'This is the first sentence. \
This is the second sentence'
```

Output:

```
Example 1:
what's your name?
Example 2:
This is the first sentence.
    This is the second sentence.
Example 3:
This is the first sentence. This is the second sentence
```

VARIABLES IN PYTHON

Identifier naming: Identifiers are names given to identify variables. There are some rules you have to follow for naming identifiers:

- The first character of the identifier must be a letter of the alphabet (uppercase ASCII or lowercase ASCII or Unicode character) or an underscore (_).
- The rest of the identifier name can consist of letters (uppercase ASCII or lowercase ASCII or Unicode character), underscores (_) or digits (0-9).
- Identifier names are case-sensitive. For example, myname and myName are not the same. Note the lowercase n in the former and the uppercase N in the latter.
- Examples of valid identifier names are i , name_2_3 . Examples of invalid identifier names are 2things, this is spaced out , my-name and >a1b2_c3.

Variables can hold values of different types called data types. The basic types are numbers and strings, which we have already discussed. In later chapters, we will see how to create our own types using classes.

Logical and Physical line

A physical line is what you see when you write the program. A logical line is what Python sees as a single statement. Python implicitly assumes that each physical line corresponds to a logical line. If you want to specify more than one logical line on a single physical line, then you have to explicitly specify this using a semicolon (;) which indicates the end of a logical line/statement. For example, all the 4 examples shown below have 2 logical lines:

<code>i = 5 print i</code>	<code>i = 5; print i;</code>	<code>i = 5; print i;</code>	<code>i = 5; print i</code>
--------------------------------	----------------------------------	------------------------------	-----------------------------

However, I strongly recommend that you stick to writing a maximum of a single logical line on each single physical line. The idea is that you should never use the semicolon.

Indentation

Whitespace is important in Python. Actually, whitespace at the beginning of the line is important. This is called indentation. Leading whitespace (spaces and tabs) at the beginning of the logical line is used to determine the indentation level of the logical line, which in turn is used to determine the grouping of statements. This means that statements which go together must have the same indentation. Each such set of statements is called a block. We will see examples of how blocks are important in later chapters.

One thing you should remember is that wrong indentation can give rise to errors. For example:

<code>print ("Example 1:") print ('what\'s your name?')</code>	<code>print ("Example 2:") print ('what\'s your name?')</code>
--	--

Output:

<code>Example 1: what's your name?</code>	<code>print ('what\'s your name?') ^ IndentationError: unexpected indent</code>
---	---

TYPE FUNCTION

type() method returns class type of the argument(object) passed as parameter. type() function is mostly used for debugging purposes.

```
# TYPE Function - it gives the type of the value
print(type(6))      #Datatypes will tell the type of data
print(type(6.0))    # as of today we learn- int, float, str
print(type("6"))
```

Programs done in the class

```
print("Hello")
# is for comment  efpogjoqefpgpoefigq nefkjgpoejqjgpofeiug
print(7)      # 7 is a number - no quote
print("7")    # 7 is a text - given in quotes
print(7+6)
print("7" + "6") #Text in Python called as String
# + in numbers will add but + on String will Concatenate (join)
# numbers are of 2 type: integer (whole numbers) and float (decimal)
#Python is case sensitive - capital and small letters have different
meaning
#All the Python commands are in small letters only
# TYPE Function - it gives the type of the value
print(type(6))      #Datatypes will tell the type of data
print(type(6.0))    # as of today we learn- int, float, str
print(type("6"))
```

```
#Variables X Constants
i=11      #Variable name
#Rules for choosing variable names
n1=131    #1. Starting letter has to be a alphabet or an _
#2. Only 1 special characters are allowed in the name: _
print(i)
print(i*i)
print(i*i*i)
#Suggestion 1: Meaningful names
string1 = "Hello friends"
print(string1) #variables do not need quotation
print("string1")
age_swapnil=10
```

```
#Write a program to find area of a circle: pi * r r
pi = 3.14
radius = 60
area = pi*radius*radius
print("Area of the circle ",area, "sq. cm") #parameters, they are separated by ,
```

```
#### Escape Sequence
#Python string accepts both double and single quotes
# Whatever quote you start with same quote you need to end
print("Hello friends")
print('How are you today?')
print("What's your name?")
```

```
# "What's your name", he said
# Escape sequence (\) do not treat it special
print("\nWhat's your name\n", he said.)
#Best practice all such special character should have escape sequence
print("\nWhat's your name\n", he said.)
#\n - newline
print("Hello \n friends")
#\n is for newline
print("\n\n is for newline")
#\n will remove specialproperty of newline character
print("\n\n\n will remove specialproperty of newline character")
#print() has default \n at the end
# end will remove the unseen newline
print("Hello friends,", end=" ")
print("How are you today?")
```

Assignments

1. # WAP to calculate Area and Perimeter of a rectangle

- - - End of Session 2 - - -

Python Programming Course

Hand Notes 2

Hand Notes for:

Basic Programming (Input, Formatting)

Basic Programming (Formatting, Escape sequence, Bool Datatypes)

Session 3: Basic Concepts - 2

THE INPUT() METHOD

The input() function allows user input.

```
print('Enter your name:')
x = input()
print('Hello, ' + x)

x = input('Enter your name:')
print('Hello, ' + x)
```

Convert Datatypes

`int()`, `str()`, `float()`

THE FORMAT() METHOD

Sometimes we may want to construct strings from other information. This is where the format() method is useful.

```
# Learning Formatting Methods
discount = 10
offer = "newsletter"
print("Sign up for our {0} and get {1}% off.".format(offer, discount))
print("Enter your email here to get our {0}".format(offer))
```

Output

```
Sign up for our newsletter and get 10% off.
Enter your email here to get our newsletter
```

A string can use certain specifications and subsequently, the format method can be called to substitute those specifications with corresponding arguments to the format method. The numbers in the format example are optional, so you could have also written as:

```
# Learning Formatting Methods
discount = 10
offer = "newsletter"
print("Sign up for our {} and get {}% off.".format(offer, discount))
print("Enter your email here to get our {}".format(offer))
```

What Python does in the format method is that it substitutes each argument value into the place of the specification. There can be more detailed specifications:

```
# Learning Formatting Methods with Numbers
print('{0:.3f}'.format(1.0/3))
# Fill with underscore(_) with text centered
# (^) to 11 width '___hello___'
print('{0:_^11}'.format('hello'))
#Left Align
print('{0:_<11}'.format('hello'))
#Right Align
print('{0:_>11}'.format('hello'))
```

```
#keyword based formatting
print('{name} wins {prize}'.format(name='Juan Manuel Santos', prize='Nobel
Peace prize 2016'))
name2 = "Abiy Ahmed Ali"
prize2 = "Nobel Peace prize 2019"
print('{} wins {}'.format(name2, prize2))
```

Output

```
0.333
__hello__
hello__
__hello
Juan Manuel Santos wins Nobel Peace prize 2016
Abiy Ahmed Ali wins Nobel Peace prize 2019
```

ESCAPE SEQUENCES

You cannot specify 'What's your name?' because Python will be confused as to where the string starts and ends. To specify that this single quote does not indicate the end of the string, we use escape sequence. In Python \: backslash character is used as escape sequence. Also, you have to indicate the backslash itself using the escape sequence \\. What if you wanted to specify a two-line string? One way is to use a triple-quoted string as shown previously or you can use an escape sequence for the newline character - \n to indicate the start of a new line. Another useful escape sequence to know is the tab: \t. There are many more escape sequences which we will discuss later. One thing to note is that in a string, a single backslash at the end of the line indicates that the string is continued in the next line, but no newline is added.

Examples

```
print ("Example 1:")
print ('what\'s your name?')
print ("Example 2:")
print ('This is the first sentence. \n This is the second sentence.')
print ("Example 3:")
print 'This is the first sentence. \
This is the second sentence'
```

Output:

```
Example 1:
what's your name?
Example 2:
This is the first sentence.
    This is the second sentence.
Example 3:
This is the first sentence. This is the second sentence
```


LOGICAL AND PHYSICAL LINE

A physical line is what you see when you write the program. A logical line is what Python sees as a single statement. Python implicitly assumes that each physical line corresponds to a logical line. If you want to specify more than one logical line on a single physical line, then you have to explicitly specify this using a semicolon (;) which indicates the end of a logical line/statement. For example, all the 4 examples shown below have 2 logical lines:

<code>i = 5</code>	<code>i = 5;</code>	<code>i = 5; print i;</code>	<code>i = 5; print i</code>
<code>print i</code>	<code>print i;</code>		

However, I strongly recommend that you stick to writing a maximum of a single logical line on each single physical line. The idea is that you should never use the semicolon.

INDENTATION

Whitespace is important in Python. Actually, whitespace at the beginning of the line is important. This is called indentation. Leading whitespace (spaces and tabs) at the beginning of the logical line is used to determine the indentation level of the logical line, which in turn is used to determine the grouping of statements. This means that statements which go together must have the same indentation. Each such set of statements is called a block. We will see examples of how blocks are important in later chapters.

One thing you should remember is that wrong indentation can give rise to errors. For example:

<code>print ("Example 1:")</code> <code>print ('what\'s your name?')</code>	<code>print ("Example 2:")</code> <code>print ('what\'s your name?')</code>
--	--

Output:

Example 1: what's your name?	<code>print ('what\'s your name?')</code> ^ IndentationError: unexpected indent
---------------------------------	---

Programs in the Class

```
#Type Conversion
var1 = "59" # implicit (by itself or automatically) conversion to String
print(type(var1))
var2 = float(var1) #explicit (outside/forced) conversion
print(type(var2))
print(var2)
### anything to integer: int()
### anything to float: float()
### anything to string: str()
```

```
#Input from the user
usrInput = input("Enter your name: ")
print(usrInput)
m1 = input("Enter your marks in subject 1: ")
m1 = float(m1)
m2 = float(input("Enter your marks in subject 2: "))
m3 = float(input("Enter your marks in subject 3: "))
```

```
sum = m1+m2+m3
print("Data type of sum is ",type(sum))
print("You have got total marks of ",sum)
```

```
sum=149
avg=sum/3
print("The total cost is {} and also total price is {} and the average cost is {}").format(sum,sum,avg))
print("The total cost is {0} and also total price is {0} and the average cost is {1}").format(sum,avg))
print("The total cost is {0} and also total price is {0} and the average cost is Rs. {1:.2f}").format(sum,avg))
```

```
name="Sachin"
typ="Batsman"
pos="Opener"
print("{0} is a {1}").format(name,typ))
print("{0:_^14} is a {1:_<20} and comes at {2:_>15} position".format(name,typ,pos))
```

```
name="Saurav"
txt = "{}"
nl="\n"
print(" Hello, I am {} and I am going to talk about {} today!".format(name,txt))
print("Hello \nMy Name is Saurav \nHow are you today?")
print("\\n is used to print in nextline") # \ is Escape Sequence
print("\\\\n is used to print in nextline")
```

Assignments

1. # WAP to find average of marks in 5 subjects and print as below:

You have obtained marks are as 45,85,95,78 and 54 which gives total of 357 and average of 71.4

The input marks needs to be taken from the user input (Use INPUT())

--- End of Session 3 ---

Python Programming Course

Hand Notes 3

Hand Notes for:

Basic Programming (Bool Datatypes, Operators and Expressions)

Basic Programming (Assignment and Logical Operators)

Session 5 & 6: Basic Concepts

BOOL DATATYPES

The `bool()` method is used to return or convert a value to a Boolean value i.e., True or False, using the standard truth testing procedure. The `bool()` method in general takes only one parameter (here `x`), on which the standard truth testing procedure can be applied. If no parameter is passed, then by default it returns False. So, passing a parameter is optional. It can return one of the two values.

- It returns True if the parameter or value passed is True. True is represented as 1.
- It returns False if the parameter or value passed is False. False is represented as 0.

OPERATORS& EXPRESSIONS

Operators	Description
+ (plus)	Adds two objects: <code>3 + 5</code> gives 8 . <code>'a' + 'b'</code> gives 'ab' .
- (minus)	Gives the subtraction of one number from the other; if the first operand is absent it is assumed to be zero.
* (multiply)	Gives the multiplication of the two numbers or returns the string repeated that many times : <code>2 * 3</code> gives 6 . <code>'la' * 3</code> gives 'lalala'.
** (power)	Returns x to the power of y: <code>3 ** 4</code> gives 81 (i.e. <code>3 * 3 * 3 * 3</code>)
/ (divide)	Divide x by y: <code>13 / 3</code> gives 4 . <code>13.0 / 3</code> gives 4.333333333333333
% (modulo)	Returns the remainder of the division: <code>13 % 3</code> gives 1 . <code>-25.5 % 2.25</code> gives 1.5
<< (left shift)	Shifts the bits of the number to the left by the number of bits specified. (Each number is represented in memory by bits or binary digits i.e. 0 and 1) <code>2 << 2</code> gives 8 . 2 is represented by 10 in bits. Left shifting by 2 bits gives 1000 which represents the decimal 8 .
>> (right shift)	Shifts the bits of the number to the right by the number of bits specified. <code>11 >> 1</code> gives 5. 11 is represented in bits by 1011 which when right shifted by 1 bit gives 101`which is the decimal `5 .
& (bit-wise AND)	Bit-wise AND of the numbers: <code>5 & 3</code> gives 1 .
(bit-wise OR)	Bitwise OR of the numbers: <code>5 3</code> gives 7
^ (bit-wise XOR)	Bitwise XOR of the numbers: <code>5 ^ 3</code> gives 6
~ (bit-wise invert)	The bit-wise inversion of x is <code>-(x+1)</code> : <code>~5</code> gives -6 . More details at http://stackoverflow.com/a/11810203
< (less than)	Returns whether x is less than y. All comparison operators return True or False. Note the capitalization of these names. <code>5 < 3</code> gives False and <code>3 < 5</code> gives True . Comparisons can be chained arbitrarily: <code>3 < 5 < 7</code> gives True.
> (greater than)	Returns whether x is greater than y <code>5 > 3</code> returns True . If both operands are numbers, they are first converted to a common type. Otherwise, it always returns False
# (less than or equal to)	Returns whether x is less than or equal to y <code>x = 3; y = 6; x # y</code> returns True
>= (greater than or equal to)	Returns whether x is greater than or equal to y <code>x = 4; y = 3; x >= 3</code> returns True

Operators	Description
== (equal to)	Compares if the objects are equal $x = 2; y = 2; x == y$ returns True . $x = 'str'; y = 'str'; x == y$ returns True . $x = 'str'; y = 'str'; x == y$ returns True .
!= (not equal to)	Compares if the objects are not equal $x = 2; y = 3; x != y$ returns True .
not (boolean)	If x is True , it returns False . If x is False , it returns True . $x = True; not x$ returns False .
and (boolean)	x and y returns False if x is False , else it returns evaluation of y $x = False; y = True; x and y$ returns False since x is False. In this case, Python will not evaluate y since it knows that the left hand side of the 'and' expression is False which implies that the whole expression will be False irrespective of the other values. This is called short-circuit evaluation.
or (boolean)	If x is True , it returns True, else it returns evaluation of y $x = True; y = False; x or y$ returns True . Short-circuit evaluation applies here as well.

Shortcut for Math operation: $a = a * 3$ can also be written as $a *= 3$

$5 = 5 \times 10^0 = 5$
 $125 = 5 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 125$
 $7643 = 7 \times 10^3 + 6 \times 10^2 + 4 \times 10^1 + 3 \times 10^0 = 7643$

$10/431 = 431$

$10/0/1 = 1 + 2 + 8 + 32 = 43$

$2/43 = 43$

$2/21 = 21$

$2/10 = 10$

$2/5 = 5$

$2/2 = 2$

$1/0 = 0$

INPUTS		OUTPUTS					
A	B	AND	NAND	OR	NOR	EXOR	EXNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Programs in the Class

```

bool1 = True    #bool type
bool2 = False
bool3 = "TRUE"
bool4 = bool(bool3)
print(type(bool3))
print(type(bool4))
# ===== Operations #####
a=5    # 101    (binary)
b=3    # 11    (binary)
print(a+b)
print(a-b)
print(a*b)
print(a/b)    # / simple divide
print(a//b)    # // will integer - Quotient
print(a%b)    # % (modulus) will Remainder
print(a**b)    # 5 * 5 * 5
print(a<<2)    # Left Shift works on Binary number
print(a>>2)    # Right Shift works on Binary number

#AND  X NAND
#OR   X NOR
#XOR   Even or Odd number of True conditions : Even T -> F / Odd T -> T
#NOT -> Reverse the output
#logical statements
a=5 # Assignment
#Mathematical: +,-,*,**,/,//, %
x=a>7
y=a>=5    #We read it as Is a greater than or equal to 5
z=a<=5    #We read it as Is a less than or equal to 5
print("Datatype of Z: ",type(z))
# >, <, <=, >=, ==, !=
print(x)    # We read it as Is a greater than 7
print(y)
print(z)
print(x or y)    # or, and, not
print(x or not (y and z))    # OR is like PLUS AND is like MULTIPLY
print(x and not y)

a==10 # Is a equal to 10
print(a==10)    # = : assignment == Condition
print(a==5)    # Is a equal to 5
print(a!=10)    # Is a not equal to 10
print(not a==10)

#Swapping values
a=5
b=10
a,b,c,d,e,f=b,a,a,a,a,a
print("Value of a = ",a)
print("Value of b = ",b)

#Short hands
a = 5

```

```
a=a+10
a+=10 # a=a+10
a/=10 #a=a/10

print(a)
```

Assignments

1. WAP to input values of 3 sides of a rectangle - area and perimeter - > input, format and make it user friendly
2. WAP to input values of 3 sides of a triangle - area and perimeter - > input, format and make it user friendly
3. WAP Marks, total, avg -> input, format
4. WAP to input a number and display its Double and Half values using SHIFT operator
5. WAP to take input Dividend and Divisor and print its Quotient and Remainder
6. WAP to input 2 numbers and print 2 output: 1) in Decimal format and 2) Mixed Integer format (e.g. Q . R/D, $10/3 = 3.1/3$)
7. WAP to take a person's nationality and age and display the same in the following format:

Person is 23 year old and is American Citizen

8. Once I had been to the post-office to buy stamps of five rupees, two rupees and one rupee. I paid the clerk Rs. 20, and since he did not have change, he gave me three more stamps of one rupee. If the number of stamps of each type that I had ordered initially was more than one, what was the total number of stamps that I bought.
9. When an object is thrown into the air with a starting velocity of r feet per second, its distance d in feet, above its starting point t seconds after it is thrown is about $d=rt-16t^2$. Evaluate to show the distance of an object from its starting point that has an initial velocity of 80 feet per second and 5 seconds after it left the ground.

- - - End of Session 3 - - -

Python Programming Course

Hand Notes 4

Hand Notes for:

Basic Programming (Flowcharts, Conditional statements IF ELIF ELSE)









Basic Programming (Conditional statements IF ELIF ELSE)

Session 6: Flowcharts

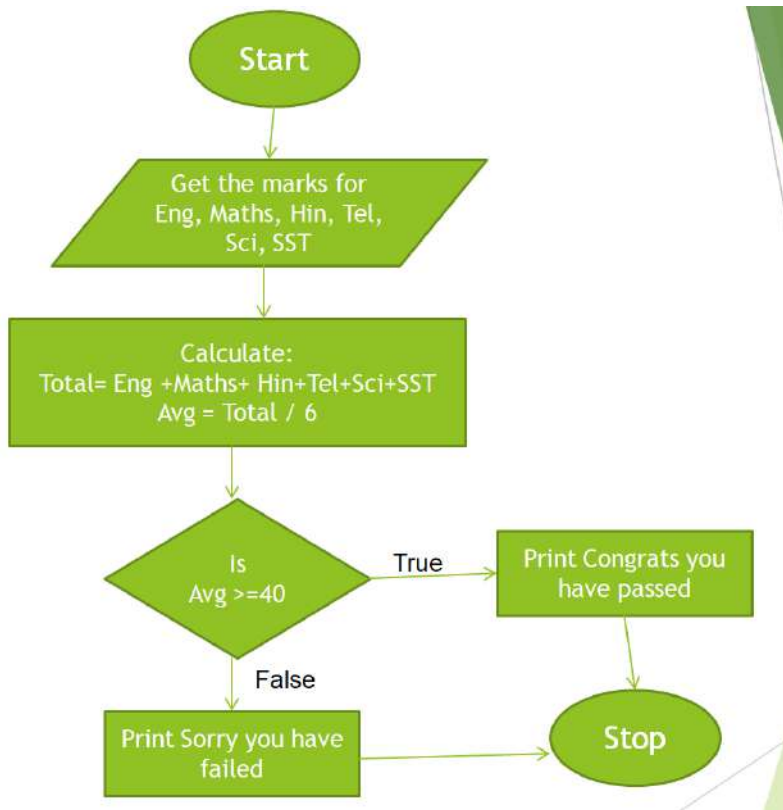
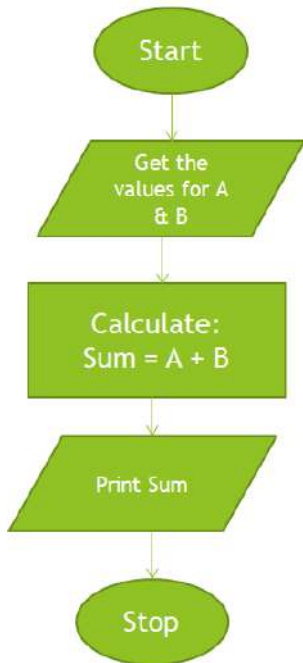
FLOWCHARTS

A flowchart is a diagrammatic representation of an algorithm. A flowchart can be helpful for both writing programs and explaining the program to others.

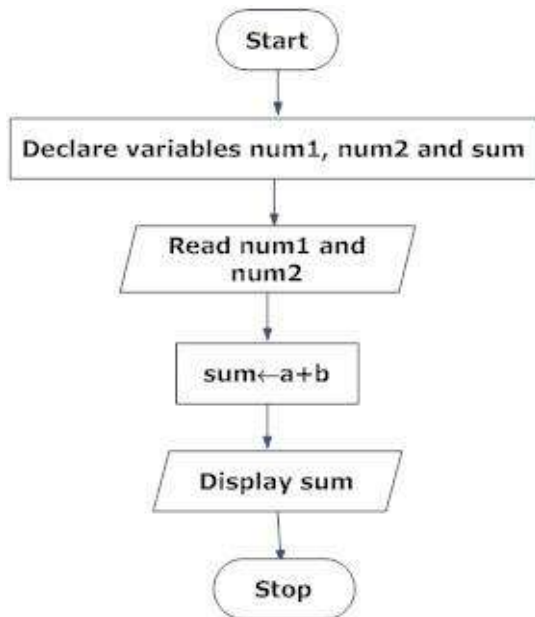
SYMBOLS USED IN FLOWCHART

Symbol	Purpose	Description
	Flow line	Indicates the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Represents the start and the end of a flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for arithmetic operations and data-manipulations.
	Decision	Used for decision making between two or more alternatives.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect the flowchart portion on a different page.
	Predefined Process/Function	Represents a group of statements performing one processing task.

Examples of flowcharts in programming

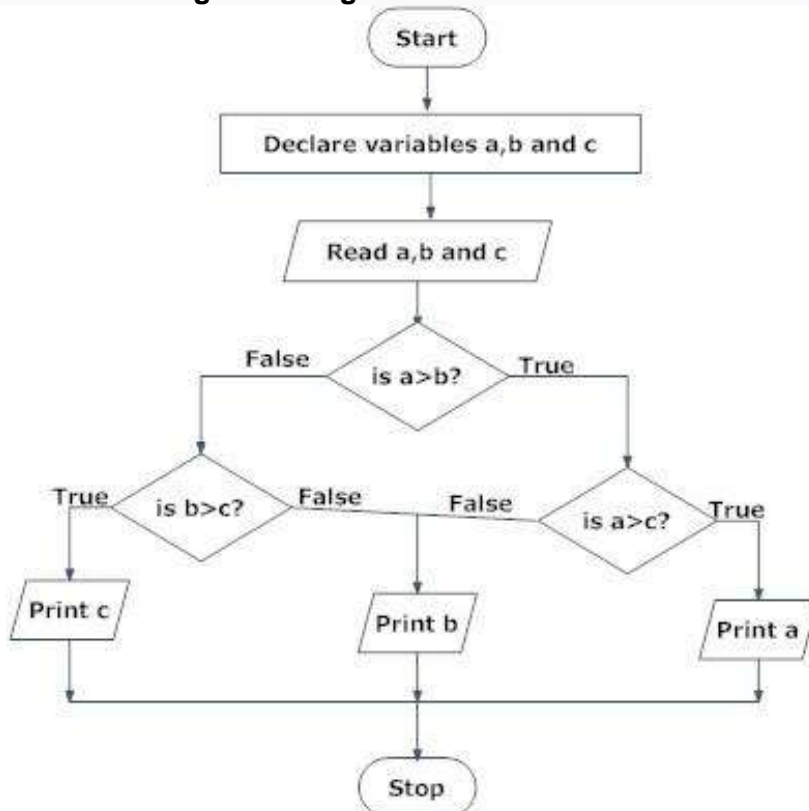


1. Add two numbers entered by the user.



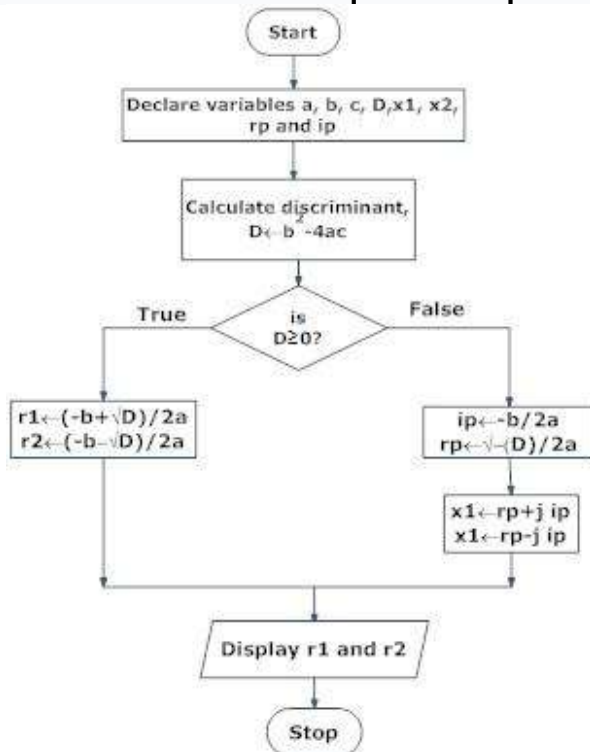
Flowchart to add two numbers

2. Find the largest among three different numbers entered by the user.



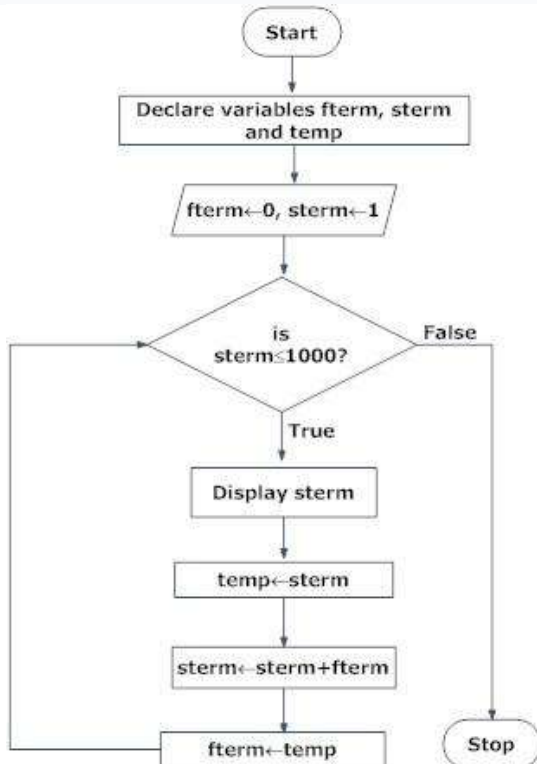
Flowchart to find the largest among three numbers.

3. Find all the roots of a quadratic equation $ax^2+bx+c=0$



Flowchart to find roots of a quadratic equation

4. Find the Fibonacci series till term ≤ 1000.



Flowchart to find the Fibonacci Series

Evaluation Order

The following table gives the precedence table for Python, from the lowest precedence (least binding) to the highest precedence (most binding).

lambda: Lambda Expression	
if - else: Conditional expression	
or: Boolean OR	
and: Boolean AND	
not x: Boolean NOT	
in, not in, is, is not, <, #, >, >=, !=, ==	Comparisons, including membership tests and identity tests
Bitwise OR	
^ Bitwise XOR	
& Bitwise AND	
<<, >> Shifts	
+, - Addition and subtraction	
*, /, //, % Multiplication, Division, Floor Division and Remainder	
+x, -x, ~x Positive, Negative, bitwise NOT	
** Exponentiation	
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or tuple display, list display, dictionary display, set display

Session 7: Conditional Statements

CONDITIONAL STATEMENTS

The if Statement

The if statement is used to check a condition: if the condition is true, we run a block of statements (called the if-block), else we process another block of statements (called the else-block). The else clause is optional.

```

number = 23
guess = int(raw_input('Enter an integer : '))
if guess == number:
    # New block starts here (Note the Indentation)
    print 'Congratulations, you guessed it.'
    print '(but you do not win any prizes!)'
    # New block ends here
elif guess < number:
    # Another block
    print 'No, it is a little higher than that'
    # You can do whatever you want in a block ...
else:
    print 'No, it is a little lower than that'
    # you must have guessed > number to reach here
print 'Done'

```

Output:

```

Enter an integer : 34
No, it is a little lower than that
Done

```

We supply a string to the built-in `raw_input` function which prints it to the screen and waits for input from the user. Once we enter something and press enter key, the `input()` function returns what we entered, as a string. We then convert this string to an integer using `int` and then store it in the variable `guess`. Actually, the `int` is a class but all you need to know right now is that you can use it to convert a string to an integer (assuming the string contains a valid integer in the text).

Next, we compare the guess of the user with the number we have chosen. If they are equal, we print a success message. Notice that we use indentation levels to tell Python which statements belong to which block. This is why indentation is so important in Python. Notice how the `if` statement contains a colon at the end - we are indicating to Python that a block of statements follows.

There is no switch statement in Python. You can use an `if..elif..else` statement to do the same thing.

Programs in the Class

```

Eng = 0
Maths = 60
Hin = 100
Tel = 50
Sci = 56
SST = 50
Total = Eng + Maths + Hin + Tel + Sci + SST
Avg = Total / 6
print("Average = ", Avg)
if Avg >= 70:
    print("Congratulations you have got Grade A!")
    #
elif Avg >= 40:
    print("Congratulations you have got Grade B!")
else:

```

```
print("Sorry you have failed")

print("Thank you for using this program!")
```

```
Eng = 80
Maths = 60
Hin = 100
Tel = 80
Sci = 50
SST = 50
Total = Eng + Maths + Hin + Tel + Sci + SST
Avg = Total / 6
print("Average = ", Avg)

if Avg >=70:
    print("Grade A")
if Avg >=40 and Avg <70:
    print("Grade B")
if Avg <40:
    print("Failed!")
print("Thank you!")
```

```
#Once I had been to the post-office to buy stamps of five rupees, two rupees and one rupee.
# I paid the clerk Rs. 20, and since he did not have change, he gave me three more stamps of one rupee.
# If the number of stamps of each type that I had ordered initially was more than one,
# what was the total number of stamps that I bought.

t=43 #total amount i have paid to clerk
print("The total amount that i paid to clerk is:RS.{}".format(t)) #money that i paid
#ch=3 #change to be given by clerk
#print("The balance that the clerk given as Rs.1 stamp is:RS.{}".format(ch)) # the change

#diff costs of stamps
v5=5 #rs.5
v2=2 #rs.2
v1=1 #rs.1
mq=2 # Min qty odered
av1=3 +mq
av5 = mq
av2 = mq

cmq=av5*v5+av2*v2+av1*v1 #cost of minimum qty
print("cost of minimum qty Rs. : ",cmq)
bam=t-cmq # balnce amount after minimum qty + change given
print("Balance after minimum and change given qty Rs. : ",bam)

#av5 = 2
av5+= bam//v5 #av5=av5 +2
bam %= v5

av2+= bam//v2 #av5=av5 +2
bam %= v2

av1+= bam//v1 #av5=av5 +2
bam %= v1
```

```

newTot = v5*av5 + v2*av2 + v1*av1
print("Total value that we account for is {} and the balance is {}".format(newTot, bam))
print("Total value of stamps: Rs 5 = {}, Rs 2 = {} and Re 1 = {}".format(av5,av2,av1))

# check if eligible to vote in India or not
# Indian citizen and age>=18
age=21
#citizen = input("Please enter your nationality (Indian/Other): ")
citizen = "iNdiAn"
if age >=18 and citizen.upper() == "INDIAN":
    print("You are eligible to vote in India")
elif age >=18 and citizen.upper() != "INDIAN":
    print("Your age is above 18 but you are not an Indian citizen hence you cant")
else:
    print("Sorry you cant vote in India")
#####
if age>=18:
    if citizen.upper() == 'INDIAN':
        print("You are eligible to vote in India")
    else:
        print("You can not vote because you are not an Indian citizen")
else:
    print("Your age is not yet 18, hence you can not vote in India")

# WAP to input 3 numbers and display in ascending order DO NOT USE AND, OR, NOT
#Use Nested IF
# 5,9,3 - Input
# 3 <= 5 <=9 - Output

```

Assignments

1. #WAP enter 3 sides, check if triangle is possible or not
and also to check if the triangle is Isoceses or Equilateral or Scalene or Right angled
2. #Modify this program to make it look like your school's rule
A >=80 B>=70 C>=60, D >=40 <40: Failed
3. Write if statements to do the following:
 - If character variable taxCode is 'T', increase price by adding the taxRate percentage of price to it.
 - If integer variable opCode has the value 1, read in double values for X and Y and calculate and print their sum.
 - If integer variable currentNumber is odd, change its value so that it is now 3 times currentNumber plus 1, otherwise change its value so that it is now half of currentNumber (rounded down when currentNumber is odd).
 - Assign true to the boolean variable leapYear if the integer variable year is a leap year. (A leap year is a multiple of 4, and if it is a multiple of 100, it must also be a multiple of 400.)
 - Assign a value to double variable cost depending on the value of integer variable distance as follows:

Distance	-	Cost

0 through 100		5.00
More than 100 but not more than 500		8.00
More than 500 but less than 1,000		10.00
1,000 or more		12.00

- - - End of Handout Notes 4 - - -

Python Programming Course

Hand Notes 5

Hand Notes for:

Advanced Programming (Loop statements FOR)

Advanced Programming (Loop statements WHILE)

Advanced Programming (Printing Patterns)

Session 6: LOOPS

WHILE STATEMENTS

while

- **while loop:** Executes a group of statements as long as a condition is True.
 - good for *indefinite loops* (repeat an unknown number of times)

- Syntax:

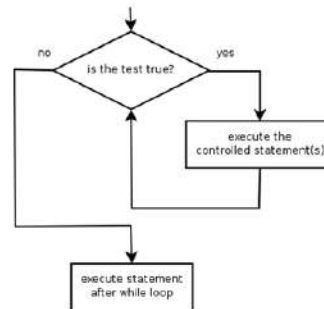
```
while condition:
    statements
```

- Example:

```
number = 1
while number < 200:
    print (number)
    number = number * 2
```

- Output:

```
1 2 4 8 16 32 64 128
```



FOR STATEMENTS

The for loop

- **for loop:** Repeats a set of statements over a group of values.

- Syntax:

```
for variableName in groupOfValues:
    statements
```

- We indent the statements to be repeated with tabs or spaces.
- **variableName** gives a name to each value, so you can refer to it in the **statements**.
- **groupOfValues** can be a range of integers, specified with the **range** function.

- Example:

```
for x in range(1, 6):
    print x, "squared is", x * x
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

range

- The `range` function specifies a range of integers:
 - `range(start, stop)` - the integers between **start** (inclusive) and **stop** (exclusive)
 - It can also accept a third value specifying the change between values.
 - `range(start, stop, step)` - the integers between **start** (inclusive) and **stop** (exclusive) by **step**

- Example:

```
for x in range(5, 0, -1):
    print x
print ("Blastoff!")
```

Output:

```
5
4
3
2
1
Blastoff!
```

Programs in the Class

```
#WAP to print multiples of 3 until you reach 200
value = 3
print("Below are the multiples of 3: ")
while value <=200:
    print(value)
    value+=3

print("Thank You!")

#For Loop
#Range(65): 0,1,2,3,4,... 64
#By default starting no. is 0
#By Default increment by 1
#You can change these values
#Range(start, end, increment) - upto end number, not including end number
#Range(1,9,2): 1, 3, 5, 7 => when Range 3 numbers
#Range(1,9) : (increment is default value which is 1) : 1,2,3,4...8 => Range has 2
input parameters or 2 values
#Range(9): (increment is 1 and start is 0): 0,1,2,3,4,...8 => Range has 1 input
parameter or 1 value

print("First For Loop")
for l in range(6): #Here range is taking 1 input (ending value) - default start is 0
and increment is 1
    print((l+1)*3)
print("Second For Loop")
for l in range(1,6): #Here range is taking 2 input (start and end value) - default
increment is 1
    print(l*3)
```

```

print("Third For Loop")
for l in range(3,201,3): #Here range is taking all 3 input (start, end & increment
    values)
    print(l)
#Break, Continue and pass
#For loop to check Range in descending order
for l in range(10,5,1):
    print(l)

```

```

#Programs to generate different patterns
# As assignment, modify these programs to use WHILE Loop
##+, -- Unary Plus or minus is not in Python
##Pattern 1: Star in Square shape
counter=0
for i in range(5):
    for j in range (5):
        #print("* ",end="Sourav\n")    #print("* ")
        print("* ", end="")
        #by default when you use print => print("",end="\n")
    print("")

##Pattern 2: Star in Triangle shape
print("Pattern 2")
for i in range(5):
    for j in range (i+1):
        print("* ", end="")
    print("")

##Pattern 3: Inverse Triangle shape
print("Pattern 3")
for i in range(5):
    for j in range (i,5):
        print("* ", end="")
    print("")

##Pattern 4: Inverse Triangle shape
print("Pattern 4")
for i in range(5):
    for k in range(5):
        print(" ",end="")
    for j in range (i,5):
        print("* ", end="")
    print("")

#Assignment 1: modify above program
#Assignment 2: Print equilateral triangle pattern

```

```

#Assignment 2: Print equilateral triangle pattern
for i in range(5):
    for k in range(i):
        print(" ",end="")
    for j in range (i,5):
        print("* ", end="")
    print("")

```

```

#Assignment 2: Print inverted equilateral triangle pattern

```

```

numlines=10
for i in range(numlines):
    for k in range(numlines-i-1):
        print(" ",end="")
    for j in range(i+1):
        print("* ", end="")
    print("",end="\n")
#####
print("Pattern 3")
a = 20
for i in range(0, 5):
    print(" " * a, end='') # repeat space for a times
    print("* " * (i)) # repeat stars for i times
    a -= 1

```

Assignments

1. *# Assignment 1: Modify the Total Avg marks calculation program to do it for 5 students*
Assignment 2: Modify your Voting program (eligible to vote or not) to a repeat it for multiple input until
user wants to continue
2. Write a Python program that computes the factorial of an integer.
3. Program to find sum N natural numbers
4. Write code to display and count the factors of a number
5. Program to check if eligible to vote in India
6. Enter marks of 3 subjects for 5 students and grade them. Check for data validity and use BREAK and CONTINUE where necessary
7. Check the type of a Triangle: Isosceles, Equilateral, Scalene, Right Angle
8. Input 3 numbers and re-arrange them in ascending order. Use BOOLEAN

--- End of Session 5 ---