ಕರ್ನಾಟಕ ಸರ್ಕಾರ
ತಾಂತ್ರಿಕ ಶಿಕ್ಷಣ ಇಲಾಖೆ
ಬೆಂಗಳೂರು

GOVERNMENT POLYTECHNIC, KARWAR

INPLANT TRAINING REPORT
18CS66P

2019 - 2020

CERTIFICATE

This is to certify that Mr. / Miss **SAHANA R DURGEKAR** of **VI Semester** Diploma in **Computer Science and Engineering** has conducted the practical in **Inplant Training (18CS66P)** satisfactorily during the year 2019 - 2020.

Staff Member In charge with date | **Head of the Section with date**

Vinay Bhat | **Vinay Bhat**

REGISTER NUMBER

1	1	4	C	S	1	7	0	3	5

Examiner's Signature: 1.
2.

PRINCIPAL
Govt. Polytechnic, Karwar

Date: - -2020
Place: Karwar

GOVERNMENT OF KARNATAKA
DEPARTMENT OF TECHNICAL EDUCATION

GOVERNMENT POLYTECHNIC KARWAR

2019-20

INPLANT TRAINING REPORT

SUBMITTED BY

SAHANA R DURGEKAR

TOPIC:

PYTHON PROGRAMMING LANGUAGE

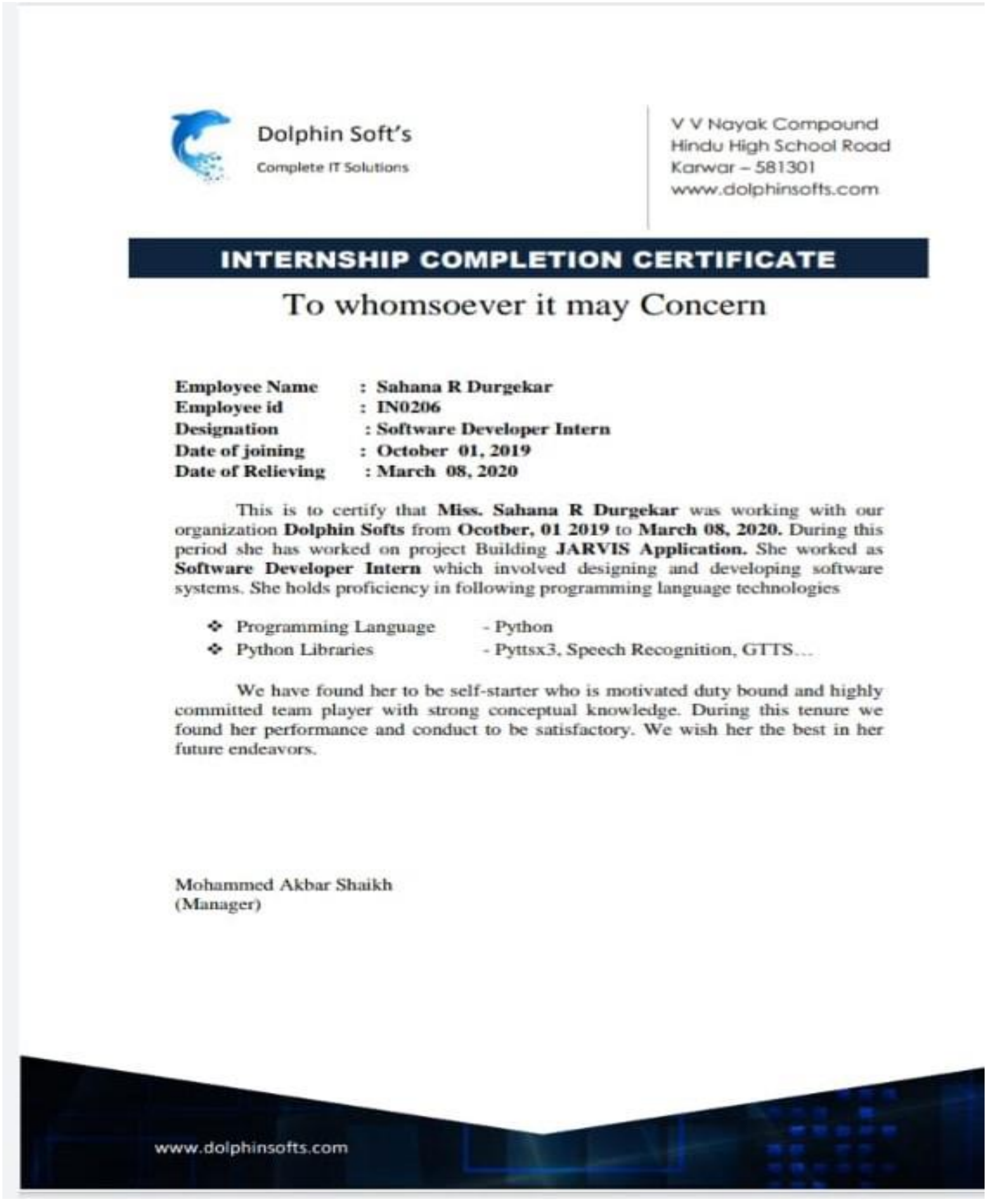
INSTITUTE NAME

DOLPHIN SOFTS, KARWAR

UNDER THE GUIDANCE OF
VINAY BHAT

Lecturer
Department of
Computer Science and Engineering

DURATION OF INPLANT TRAINING: October 01, 2019- March 08, 2020



Dolphin Soft's

Complete IT Solutions

V V Nayak Compound
Hindu High School Road
Karwar – 581301
www.dolphinsofts.com

INTERNSHIP COMPLETION CERTIFICATE

To whomsoever it may Concern

Employee Name	**: Sahana R Durgekar**
Employee id	**: IN0206**
Designation	**: Software Developer Intern**
Date of joining	**: October 01, 2019**
Date of Relieving	**: March 08, 2020**

This is to certify that **Miss. Sahana R Durgekar** was working with our organization **Dolphin Softs** from **Ocotber, 01 2019** to **March 08, 2020.** During this period she has worked on project Building **JARVIS Application.** She worked as **Software Developer Intern** which involved designing and developing software systems. She holds proficiency in following programming language technologies

❖ Programming Language	- Python
❖ Python Libraries	- Pyttsx3, Speech Recognition, GTTS…

We have found her to be self-starter who is motivated duty bound and highly committed team player with strong conceptual knowledge. During this tenure we found her performance and conduct to be satisfactory. We wish her the best in her future endeavors.

Mohammed Akbar Shaikh
(Manager)

www.dolphinsofts.com

ACKNOWLEDGMENT

The successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance crowed our efforts with success.

I would like to extend heartfelt thanks to my inplant training report guide **Mr. Mohammed Akbar Shaikh**, **DOLPHIN SOFT KARWAR**, for all the resources, suggestions and motivation given to me in the regard of leaning python programming language and extending support at all stages of the inplant training.

I would like to thank our beloved principal **Mr. V.M HEGDE,** for extending his continuous support and providing academic environment.

I would like to thank all the teaching and non-teaching staff, **DOLPHIN SOFT** for their generous help in various ways for the completion of this project.

I thank to my beloved **Parents and Dear Friends** for their continuous encouragement, motivation and blessings without which I would not have reached this stage.

ABOUT THE ORGANIZATION

Since 2010, **Dolphin Softs** have focused on the developing a custom software application and website designing and developing. Focusing towards the potential growth of the company and come up with the reselling software application and 3rd party product. Our select portfolio comprises solutions for communications, IT security, and IT management, for all types of operating systems. All of the software products that we sell offer the highest level of user friendliness, simple installation maintenance, optimum reliability-and all at an excellent price. Though we are new in a software reseller we are very positive towards our business and experience software reseller team who have a very good experience in software product reselling.

We provide the commitment, the experience, and the expertise required to develop and maintain successful long-term business partnerships with both vendors and retailers. since 2010, we at **Dolphin Softs** have focused on the developing a custom software's application and website designing and developing.

Focusing towards the potential growth of the company we have come up with the reselling software application and 3rd party product. Our select portfolio comprises solutions for communications, IT security, and IT management, for all types of operating systems Though we are new in a software reseller we are very positive towards our business and experience software reseller team who have a very good experience in software product reselling.

We provide the commitment, the experience, and the expertise require to develop and maintain successful long-term business partnerships with both vendors and retailers.

CONTENTS

1. Introduction
    - 1.1 Characteristics of Python

2. Getting started with Python
    - 2.1 Python Installation
    - 2.2 Writing our first program
    - 2.3 Run Hello World

3. Python Comments

4. Python concepts
    - 4.1 Variables in python
        - 4.1.1 Multiple Assignment
    - 4.2 Python Data Type
        - 4.2.1 Built-in Data Types
        - 4.2.2 Getting the Data Type
        - 4.2.3 Setting the Data Type
        - 4.2.4 String
        - 4.2.5 List
        - 4.2.6 Tuple
        - 4.2.7 Set
        - 4.2.8 Dictionary
    - 4.3 Functions in Python
    - 4.4 Basic Operators in Python
        - 4.4.1 Arithmetic operators
        - 4.4.2 Relational Operators
        - 4.4.3 Logical operators
        - 4.4.4 Bitwise operators
        - 4.4.5 Assignment operators
        - 4.4.6; Special operators
    - 4.5. Control flow
        - 4.5.1 Python if else
            - 4.5.1.1 If statement
        - 4.5.2 While Loop
            - 4.5.2.1While-else loop
        - 4.5.3 for Loop
            - 4.5.3.1The range ( ) Function
            - 4.5.3.2 Else in For Loop
            - 4.5.3.3 Nested for loop
        - 4.5.4 The break Statement
        - 4.5.5 The continue Statement
        - 4.5.6 The pass Statement
    - 4.6 Python Exception Handling
        - 4.6.1Else Clause
        - 4.6.2 User-defined Exceptions
    - 4.7 Python Modules
    - 4.8 The import statement
        - 4.8.1 The from import Statement

PYTHON PROGRAMMING LANGUAGE

1. Introduction

Python is a widely used general-purpose, high level programming language. It was created by **Guido van Rossum** in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

There are two major Python versions: **Python 2 and Python 3**. Both are quite different.

If you are learning to code in Python, we recommend you download both the latest version of **Python 2 and 3**

1.1 Characteristics of Python:

Following are important characteristics of **Python Programming** −

- ❖ It supports functional and structured programming methods as well as OOP.
- ❖ It can be used as a scripting language or can be compiled to byte-code for building large applications.
- ❖ It provides very high-level dynamic data types and supports dynamic type checking.
- ❖ It supports automatic garbage collection.
- ❖ It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

2. Getting started with Python

To begin with, writing Python Codes and performing various intriguing and useful operations, one must have Python installed on their System. This can be done by following the step by step instructions provided below:

2.1 Python Installation:

Download Python Latest Version from python.org

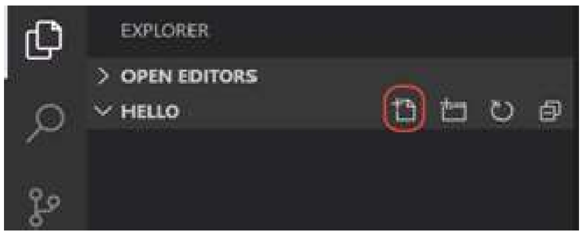
1. Go to the link: https://www.python.org/downloads/ and install the latest version on your machines.
2. Run the Python Installer from downloads folder
3. After installation is complete click on **Close**

Python programs can be written on any plain text editor like notepad, notepad++, or anything of that sort. One can also use an online IDE for writing Python codes or can even install one on their system to make it more feasible to write these codes because IDEs provide a lot of features like intuitive code editor, debugger, compiler, etc.+. Here we use the ~~Visu l Studio Code~~ installer ide

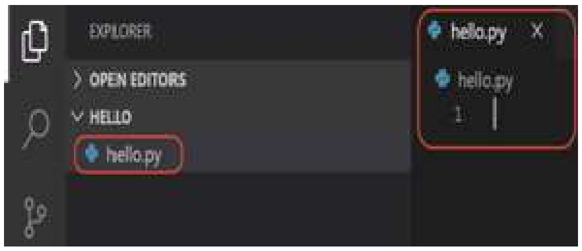
Download the ~~Visual Studio Code installer~~ for Windows.

2.2 Writing our first program:

From the File Explorer toolbar, create HELO folder. Now select the **New File** button on the HELLO folder.



Name the file hello.py, and it automatically opens in the editor:



Enter the following source code in hello.py

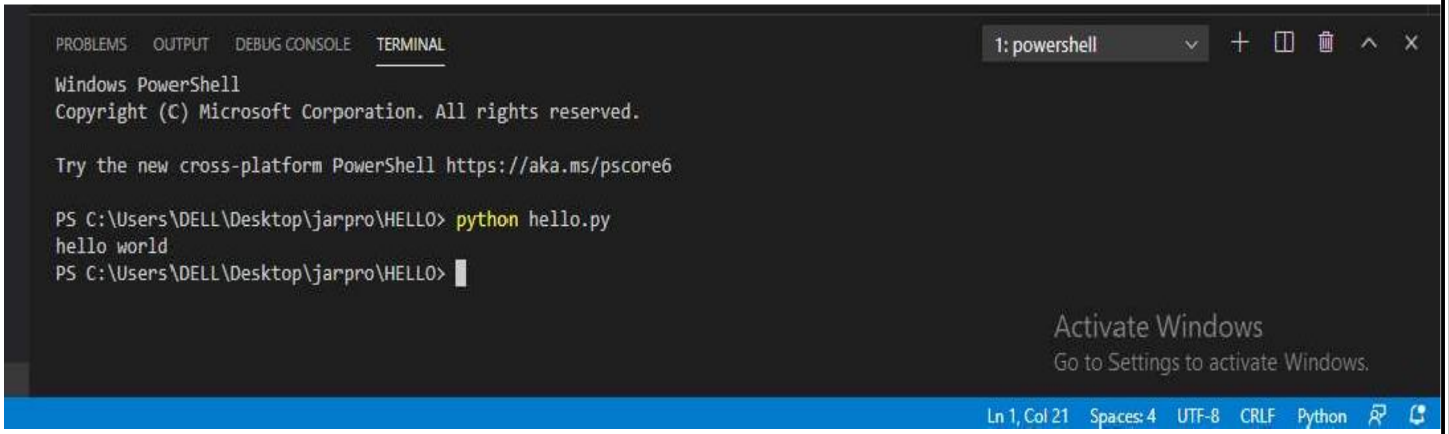
```
# Script Begins

    print("Hello World ")

 # Scripts Ends
```

2.3 Run Hello World:

It's simple to run hello.py with Python. Just click the **Run Python File in Terminal** play button in the top-right side of the editor.

The button opens a terminal panel in which your Python interpreter is automatically activated, then runs python3 hello.py (macOS/Linux) or python hello.py (Windows):

Output:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\DELL\Desktop\jarpro\HELLO> python hello.py
hello world
PS C:\Users\DELL\Desktop\jarpro\HELLO>
```

3. Python Comments

Creating a Comment
Comments starts with a #, and Python will ignore them:
Example-

```
#This is a comment
print("Hello, World!")
```

4. PYTHON CONCEPTS

4.1 Variables in python:

Variables need not be declared first in python. They can be used directly. Variables in python are case sensitive as most of the other programming languages.

For example –

```
Counter=100     # An integer assignment
Miles=1000.0    # A floating point
Name=”John”     # A string
Print(counter)
Print(Miles)
Print(Name)
```

4.1.1 Multiple Assignment

Python allows you to assign a single value to several variables simultaneously.

For example −

```
a = b = c = 1
```

4.2 Python Data Type:

4.2.1 Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text type:	str
Numeric type:	int, float, complex
Sequence type:	list, tuple, range
Mapping type:	dict
Set type:	set, fronzeset

4.2.2 Getting the Data Type

You can get the data type of any object by using the type() function:

Example

Print the data typeof the variable x:

```
x = 5
print(type(x))
```

4.2.3 Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	**Data Type**
x=”Hello World”	str
x=20	int
x=20.4	float

4.2.4 String

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example:

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example:

```
a = "Hello"
print(a)
```

4.2.5 List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

4.2.6 Tuple

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round bracket

Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

4.2.7 Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

4.2.8 Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

4.3 Functions in Python:

A function in python is declared by the keyword 'def' before the name of the function. The return type of the function need not be specified explicitly in python. The function can be invoked by writing the function name followed by the parameter list in the brackets.

```
# Function for checking the divisibility
# Notice the indentation after function declaration
# and if and else statements
def checkDivisibility(a, b):
    if a % b == 0 :
        print "a is divisible by b"
    else:
        print "a is not divisible by b"
#Driver program to test the above function
checkDivisibility(4, 2)
```

The output is :

```
a is divisible by b
```

4.4 <u>Basic Operators in Python:</u>

4.4.1. Arithmetic operators: Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

Examples of Arithmetic Operator

```
a = 9
b = 4

# Addition of numbers
add = a + b
# Subtraction of numbers
sub = a - b
```

```
# Multiplication of number
mul = a * b
# Division(float) of number
div1 = a / b
# Division(floor) of number
div2 = a // b
# Modulo of both number
mod = a % b
# print results
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
```

Output:

```
13
5
36
2.25
2
1
```

4.4.2. Relational Operators: Relational operators compares the values. It either returns **True** or **False** according to the condition.

Examples of Relational Operators

```
a=13
b = 33

# a > b is False

print(a > b)

# a < b is True

print(a < b)

# a == b is False

print(a == b)

# a != b is True

print(a != b)

# a >= b is False

print(a >= b)

# a <= b is True

print(a <= b)
```

Output:

```
False

True

False

True

False

True
```

4.4.3. Logical operators: Logical operators perform **Logical AND**, **Logical OR** and **Logical NOT** operations.

4.4.4.

Examples of Logical Operator

```
a = True
b = False
```

```
# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

Output:

```
False

True

False
```

4.4.5. Bi**twise operators:** Bitwise operators acts on bits and performs bit by bit operation.

Examples of Bitwise operators

```
a = 10
b = 4

# Print bitwise AND operation
print(a & b)

# Print bitwise OR operation
print(a | b)

# Print bitwise NOT operation
print(~a)

# print bitwise XOR operation
print(a ^ b)

# print bitwise right shift operation
print(a >> 2)

# print bitwise left shift operation
print(a << 2)
```

The Output is:

```
0

14

-11

14

2

40
```

4.4.6. Assignment operators:

OPERATOR	DESCRIPTION	SYNTAX
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b

&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise XOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b a= a << b

4.4.6 **Special operators:** There are some special type of operators like-

- **Identity operators**
- **Membership operators**

4.5 Control flow:

Python programming language provides following types of loops to handle looping requirement Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

4.5.1 Python if else

4.5.1.1 If statement: An "if statement" is written by using the if keyword.

Example:

```
a = 33
b = 200
if b > a:
print("b is greater than a")
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example:

```
a = 33
b = 33
if b > a:
print("b is greater than a")
elif a == b:
print("a and b are equal")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

Else

The else keyword catches anything which isn't caught by the preceding conditions.

Example:

```
a = 200
b = 33
if b > a:
print("b is greater than a")
elif a == b:
print("a and b are equal")
else:
print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

4.5.2 While Loop: In python, while loop is used to execute a block of statements repeatedly until a given a condition is satisfied. And when the condition becomes false, the line immediately after the loop in program is executed.

Syntax :

```
While expression:
      Statement(s)
```

```
# Python program to illustrate while loop

count = 0

while (count < 4):

    count = count + 1

    print("Government Polytechnic Karwar")
```

Output:

```
Government Polytechnic Karwar

Government Polytechnic Karwar

Government Polytechnic Karwar

Government Polytechnic Karwar
```

4.5.2.1 While-else loop

While loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed.

The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won't be executed.

```
# Python program to demonstrate while-else loop
i = 0
while i < 4:
    i += 1
    print(i)
else: # Executed because no break in for
    print("No Break\n")
  i = 0
while i < 4:
    i += 1
    print(i)
    break
else: # Not executed as there is a break
    print("No Break")
```

Output:

```
1

2

3

4

No Break

1
```

4.5.3 for Loop: A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

Example:

Print each fruit in a fruit list

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

4.5.3.1The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example:

```
for x in range(6):
  print(x)
```

4.5.3.2 Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished.

Example:

Print all numbers from 0 to 5 and print a message when the loops has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

4.5.3.3 Nested for Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example:

Print each adjective for every fruit

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

4.5.4 The break Statement

The **break** statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

Example:

```
#!/usr/bin/python

for letter in 'Python':     # First Example
   if letter == 'h':
    break
  print 'Current Letter :', letter

var = 10                    # Second Example
while var > 0:
  print 'Current variable value :', var
  var = var -1
  if var == 5:
    break

print "Good bye!"
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value: 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

4.5.5 The continue Statement

The **continue** statement in Python returns the control to the beginning of the while loop. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The **continue** statement can be used in both *while* and *for* loops.

Example:

```
#!/usr/bin/python

for letter in 'Python':     # First Example
  if letter == 'h':
    continue
  print 'Current Letter :', letter

var = 10                    # Second Example
while var > 0:
  var = var -1
```

```
   if var == 5:
      continue
   print 'Current variable value :', var
print "Good bye!"
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Good bye!
```

4.5.6 The pass Statement

The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet

(e.g., in stubs for example):

Example:

```
#!/usr/bin/python

for letter in 'Python':
   if letter == 'h':
      pass
      print 'This is pass block'
   print 'Current Letter :', letter
print "Good bye!"
```

Output:

```
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

4.6 Python Exception Handling:

Error in Python can be of two types i.e. **Syntax errors and Exceptions.** Errors are the problems in a program due to which the program will stop the execution.

Exceptions are raised when the program is syntactically correct but the code resulted in an error. This error does not stop the execution of the program, however, it changes the normal flow of the program.

Example:

```
# initialize the amount variable
marks = 10000

# perform division with 0
a = marks / 0
print(a)
```

Output:

```
Traceback (most recent call last):

   File "/home/f3ad05420ab851d4bd106ffb0422990.py" , line 4, in  <module>

       A=marks/0

ZeroDivisionError : division by zero
```

In the above example raised the ZeroDivisionError as we are trying to divide a number by 0.

Try and Except in Exception Handling
Let us try to access the array element whose index is out of bound and handle the corresponding exception.

Python program to handle simple runtime error

```
 a = [1, 2, 3]

try:

    print "Second element = %d" %(a[1])



    # Throws error since there are only 3 elements in array

    print "Fourth element = %d" %(a[3])



except IndexError:

    print "An error occurred"
```

Output:

```
Second Element = 2

An error occurred
```

4.6.1 Else Clause :

In python, you can also use else clause on try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.

# Program to depict else clause with try-except

```
 # Function which returns a/b

def AbyB(a , b):

    try:

        c = ((a+b) / (a-b))

    except ZeroDivisionError:

        print "a/b result in 0"

    else:
        print c
```

Driver program to test above function

```
AbyB(2.0, 3.0)

AbyB(3.0, 3.0)
```

Output:

```
-5.0

a/b result in 0
```

4.6.2 User-defined Exceptions

Python throws errors and exceptions, when there is a code gone wrong, which may cause program to stop abruptly. Python also provides exception handling method with the help of try-except. Some of the standard exceptions which are most frequent include IndexError, ImportError, IOError, ZeroDivisionError, TypeError and FileNotFoundError. A user can create his own error using exception class.

Creating User-defined Exception

Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in **"Error"** similar to naming of the standard exceptions in python. For example:

# A python program to create user-defined exception

```
# class MyError is derived from super class Exception

class MyError(Exception):

  # Constructor or Initializer

 def__init__(self, value):

     self.value = value


 #__str__is to print() the value
 def___str__(self):

     return(repr(self.value))


try:

    raise(MyError(3*2))


# Value of Exception is stored in error

except MyError as error:

    print('A New Exception occured: ',error. Value)
```

Output:

```
(‘A New Exception Occurred:’, 6)
```

4.7 <u>Python Modules</u>:

A module is a file containing Python definitions and statements. A module can define functions, classes and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.

Example:

```
# A simple module, calc.py

  def add(x, y):

    return (x+y)

def subtract(x, y):

    return (x-y)
```

4.8 <u>The import statement</u>

We can use any Python source file as a module by executing an import statement in some other Python source file.

When interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches for importing a module. For example, to import the module calc.py, we need to put the following command at the top of the script :

```
# importing module calc.py

import calc

print add(10, 2)
```

Output:

```
12
```

4.8.1 The from import Statement

Python’s from statement lets you import specific attributes from a module. The from import has the following syntax :

```
# importing sqrt() and factorial from the

# module math

from math import sqrt, factorial

  # if we simply do "import math", then

# math.sqrt(16) and math.factorial()

# are required.

print sqrt(16)

print factorial(6)
```

Output:

```
4.0

720
```