PROGRAMMING WITH PYTHON

Prof. Rachana V. Modi

What is Python…?

- **Python** is a general purpose, dynamic, high level, and interpreted programming language.
- It supports Object Oriented programming approach to develop applications.
- It is simple and easy to learn and provides lots of high-level data structures.
- It makes the development and debugging *fast* because there is no compilation step included in Python development.

Differences between program and scripting language

Program

A **program is executed** ***(i.e.*** *the source is first compiled, and the result of that compilation is expected)*

A "program" is **a sequence of instructions written so that a computer can perform certain task.**

Script

A **script is interpreted**

A "script" is code written in a scripting language.

A scripting language is **a type of programming language in which we can write code to control another software application.**

Who uses python today…

Python is being applied in real revenue-generating products by real companies. For instance:

- Google makes extensive use of Python in its **web search system**, and employs Python's creator.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm and IBM use Python **for hardware testing**.
- The YouTube **video sharing service** is largely written in Python

What can I do with Python…?

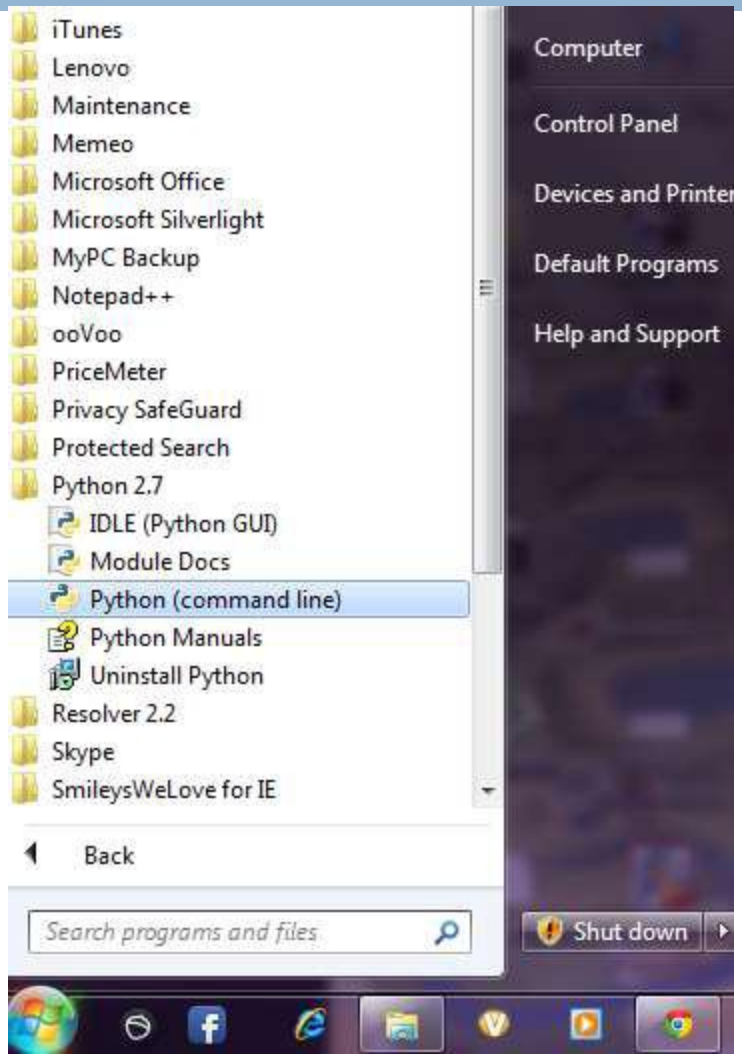
- System programming
- Graphical User Interface Programming
- Internet Scripting
- Component Integration
- Database Programming
- Gaming, Images, XML , Robot and more

Installing Python

- Python is pre-installed on most Unix systems, including Linux and MAC OS X
- But for in Windows Operating Systems , user can download from the https://www.python.org/downloads/

  - from the above link download latest version of python IDE and install, recent version is 3.8.1 but most of them uses version 2.7.7 only

After installing the Python Ver#2.7.7, go to start menu then click on python 2.7 in that one you can select python (command line) it is prompt with >>>



First Python Program

- Python provides us the two ways to run a program:

  1) Using Interactive interpreter prompt

     - Execute the python statement one by one at the interactive prompt.
     - It is preferable in the case where we are concerned about the output of each line of our python program.
     - To open the interactive mode, open the terminal (or command prompt) and type python.

First Python Program

- Python provides us the two ways to run a program:

  2) Using a script file

  - To write our code into a file which can be executed later.
  - For this purpose, open an editor like notepad, create a file named first.py (python used .py extension) and write the following code in it.

    print "Hello world" or print ("hello world")

  - To run this file named as first.py, we need to run the following command on the terminal.

    python first.py or python3 first.py

Python IDE


IDLE


SPYDER


PC
PyCharm


eric


Vim


Atom


jupyter


Anaconda


PyDev


Thonny
Python IDE for beginners

Anaconda (Python distribution)

- Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.)
- Aims to simplify package management and deployment.
- Package versions are managed by the package management system conda.
- The Anaconda distribution includes data-science packages suitable for Windows, Linux and MacOS.
- It has no IDE of its own. The default IDE bundled with Anaconda is Spyder.

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a **case sensitive** programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.

Variables

- Variables are reserved memory locations to store values. This means when you create a variable you reserve some space in memory.
- No need to declare in advance
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Not typed
  - Eg. A=10 # An integer assignment
    A="Hello" # A string

Standard Data Types

- The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Python has five standard data types:
  - Numbers
  - String
  - List
  - Tuple
  - Dictionary

A Sample Code

Examples:

Sample - 1:

```
>>> print('Hello world')
Hello world
```

Sample - 2:

```
>>> x = 5
>>> y = 5.47
>>> print(x,y)
5  5.47
```

Sample - 3:

```
>>> x = 34 – 23
>>> print(x)
11
```

Sample - 4:

```
>>> x = 12**2
>>> x/2
72
```

Operators

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisions, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Membership Operators

- To check the membership of value inside a data structure.
- If the value is present in the data structure, then the resulting value is true otherwise it returns false.

1) **'in' operator** - It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
2) **'not in' operator** - It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Example:

a = 1

list = [1, 2, 3, 4, 5]

a in list → True

Identity Operators

- To determine whether a value is of a certain class or type.
- It means variables point at the same object.

1) 'is' operator – Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.

2) 'is not' operator – Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

Example:

1) x = 5

   Print(type(x) is int) → True

2) x = 5.3

   Print(type(x) is int) → False

3) X=20

   Y=20

   Print(x is y) → True

Strings

Sample Code	Output	Explanation
"hello"+"world"	"helloworld"	# concatenation
"hello"*3	"hellohellohello“	# repetition
"hello"[0]	"h"	# indexing
"hello"[-1]	"o"	# (from end)
"hello"[1:4]	"ell"	# slicing
len("hello")	5	# size
"hello" < "jello"	1	# comparison
"e" in "hello“	1	# search

- "escapes: \n etc, \033 etc, \if etc“
- “””U.V.Patel college of Engineering Kherva””” # multiline strings

Comments

- Comment is not a part of the program, but it enhances the interactivity of the program and makes the program readable.
- It used to explain any program code.
- It can also be used to hide the code.
- # symbol is known as a single line comment.
- Multi lined comment can be given inside triple quotes.
- The interpreter does not interpret the comment.
- **Example:**

```
#single line comment
print "Hello Python"
'''This is
multiline comment'''
```

Output function

- **print()** function used to display data to the standard output device (screen).
- Example:
- print('This sentence is output to the screen')# Output: This sentence is output to the screen
- a = 5

  print('The value of a is', a)# Output: The value of a is 5

Output function

- print(1,2,3,4)
- Space was added between values.
- This is by default, but we can change it.
- The actual syntax of the print() function is
- print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

objects is the value(s) to be printed.

sep separator is used between the values. It defaults into a space character.

end value is printed after all values are printed. It defaults into a new line.

The file is the object where the values are printed and its default value is sys.stdout (screen).

Here are an example to illustrate this.

print(1,2,3,4,sep='*') # Output: 1*2*3*4

print(1,2,3,4,sep='#',end='&') # Output: 1#2#3#4&

Output function

format() :

To format output to make it look attractive.

This can be done by using the str.format() method. This method is visible to any string object.

curly braces {} are used as placeholders. It can specify the order in which it is printed by using numbers (tuple index).

Example 1:

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10
```

Example 2:

```
>>> print('I love {0} and {1}'.format('bread','butter')) # Output: I love bread
    and butter
>>> print('I love {1} and {0}'.format('bread','butter'))# Output: I love butter
    and bread
```

Input function

The value of variables were defined or hard coded into the source code.

- **input()** function allow flexibility to take the input from the user.
- The syntax for input() is input([prompt])

where prompt is the string we wish to display on the screen. It is optional.

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num
'10'
```

Import statement

- When our program grows bigger, it is a good idea to break it into different modules.
- A module is a file containing python definitions and statements.
- Python modules have a filename and end with the extension .py.
- Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the import keyword to do this.
- for example, we can import the math module by typing in import math.

```
>>> import math
>>> print(math.pi)
```

- Now all the definitions inside math module are available in our scope. We can also import some specific attributes and functions only, using the from keyword. For example:

```
>>> from math import pi
>>> pi
3.141592653589793
```

Type Casting

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.
- Python has two types of type conversion.
  - Implicit Type Conversion
  - Explicit Type Conversion

Type Casting: Implicit Type Casting

- In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.
- Let's see an example where Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.
- **Example 1: Converting integer to float**

```
num_int = 123
num_float = 1.23
num_new = num_int + num_float
print("datatype of num_int:",type(num_int))
print("datatype of num_float:",type(num_float))
print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

Type Casting: Implicit Type Casting

- **Example 2: Addition of string(higher) data type and integer(lower) datatype**

```
num_int = 123
num_str = "456“
print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))
print(num_int+num_str)
```

Output:

```
Data type of num_int: <class 'int'>
Data type of num_str: <class 'str'>
Traceback (most recent call last):
File "python", line 5, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Type Casting: Explicit Type Casting

- In Explicit Type Casting, users convert the data type of an object to required data type.
- We use the predefined functions like int(), float(), str() etc to perform explicit type conversion.
- Syntax :

  (required_datatype)(expression)
- **Example 3: Addition of string and integer using explicit conversion**

```
num_int = 123
num_str = "456“
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

- **Output:**

```
Data type of num_str after Type Casting: <class 'int'>
Sum of num_int and num_str: 579
 Data type of the sum: <class 'int‘>
```

Python - Numbers

- Python supports four different numerical types:
- **int (signed integers)** − They are often called just integers or ints, are positive or negative whole numbers with no decimal point.
- **long (long integers)** − Also called longs, they are integers of unlimited size, written like integers and followed by an uppercase or lowercase L.
- **float (floating point real values)** − Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = $2.5 \times 10^2 = 250$).
- **complex (complex numbers)** − are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a, and the imaginary part is b. Complex numbers are not used much in Python programming.

Indentation in Python

- For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code.
- In Python, indentation is used to declare a block.
- If two statements are at the same indentation level, then they are the part of the same block.
- Generally, four spaces are given to indent the statements in python.
- Indentation is the most used part of the python language since it declares the block of code.
- All the statements of one block are intended at the same level indentation.

Decision Making

- Decision making statement used to control the flow of execution of program depending upon condition.
- if the condition is true then the block will execute and if the condition is false then block will not execute.
- Decision making statements available in python are:
  - If statement
  - If..else statements
  - Nested if statements
  - If-elif ladder

Decision Making

if statement: The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.

- **Syntax:**

```
if expression:
        statement(s)
```

- **Example:**

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
```

Decision Making

If else statement: If else statements will execute one block of statements if condition is true else another block of statement will be executed.

- **Syntax:**

```
if expression:
        statement(s)
else:
        statement(s)
```

- **Example:**

```
age = int(input("enter age"))
if(age < 18):
    print('you are minior')
else:
    print('you are adult')
```

Decision Making

If-elif ladder statement: The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

- **Syntax:**

```
if expression1:
        statement(s)
elif expression2:
        statement(s)
elif expression3:
        statement(s)
else:
        statement(s)
```

- **Example:**

```
a = 15
b = 15
if a > b:
        print("a is greater")
elif a == b:
        print("both are equal")
else:
        print("b is greater")
```

Decision Making

Nested if-else statement: A situation to check second condition after first condition resolves to true.

- **Syntax:**

```
if expression1:
        statement(s)
        if expression2:
                statement(s)
        elif expression3:
                statement(s)
        elif expression4:
                statement(s)
        else:
                statement(s)
  else:
        statement(s)
```

Example:

1) Find out Number is positive or Negative, and check number has how many digits.

2) Find out largest number from three numbers with all constraints.

What is the output of the following code?

```
if None:
    print("Hello")
```

Iterative Statements (Loops)

- A loop statement allows us to execute a statement or group of statements multiple times.

Loop Types:

- while loop
- for loop

while loop

Syntax of while loop:

```
while expression:
        statement(s)
```

Example:

```
#!/usr/bin/python
count = 0
while (count < 9):
        print 'The count is:', count
        count = count + 1
print "Good bye!"
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

For loop

- For loops are used for sequential traversal.

Syntax of for loop:

```
for iterating_var in sequence:
    statements(s)
```

Example - 1:

```
#!/usr/bin/python
for letter in 'Python':
    print 'Current Letter :', letter
```

Example – 2:

```
fruits = ['banana', 'apple', 'mango']
for fruit in fruits:
    print 'Current fruit :', fruit
print "Good bye!"
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

For loop

- range(): 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.

Syntax: range(start, stop, increment)

Example 1:

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

Output: values 0 to 5

Example 2:

```
for i in range(3, 16, 3):
    print(i)
```

Output: 3 6 9 12 15

For loop

Iterating by index of sequences.

Example 3:	**Output:**
languages = ['Spanish', 'English', 'French', 'German', 'Irish', 'Chinese'] for index in range(len(languages)): print('Language:', languages[index])	Language: Spanish Language: English Language: French Language: German Language: Irish Language: Chinese

Loop Control Statements

- Loop control statements change execution from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Python supports the following control statements:
  - **Continue Statement**
  - **Break Statement**
  - **Pass Statement**

Loop Control Statements

- **Continue Statement:**

  It returns the control to the beginning of the loop.

  Example:

```
# Prints all letters except 'e' and 's'
for letter in ‘welcome to uvpce':
if letter == 'e' or letter == ‘c':
      continue
print 'Current Letter :', letter
```

  Output:

  Current Letter : w l o m t o u v p

Loop Control Statements

- **Break Statement:**

  It brings control out of the loop.

  Example:

```
for letter in ‘welcome to uvpce':
    # break the loop as soon it sees 'e'  or 's'
    if letter == 'e' or letter == 's':
        break
print 'Current Letter :', letter
```

  Output:

```
Current Letter: e
```

Loop Control Statements

- **Pass Statement:**
  - The **pass** statement is used when a statement is required syntactically but do not want any command or code to execute.
  - pass is a null statement; nothing happens when it executes.
  - It is useful in places where your code will eventually go, but has not been written yet (empty control statement, function and classes).

Loop Control Statements

- **Pass Statement:**

Example:

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

Output:

```
Current Letter: p
Current Letter: y
Current Letter: t
This is pass block
Current Letter: h
Current Letter: o
Current Letter: n
```

Loop Control Statements

Difference Between pass And continue Statement in Python

- **pass** statement simply does nothing. You use pass statement when you create a method that you don't want to implement, yet.
- Where **continue** statement skip all the remaining statements in the loop and move controls back to the top of the loop.

Mathematical Functions

- The **math** module is used to access mathematical functions in the Python. All methods of this functions are used for integer or real type objects, not for complex numbers.
- To use mathematical functions import math module into code.

e.g. import math

```
print('The value of 5^8: ' + str(math.pow(5, 8)))
print('Square root of 400: ' +
str(math.sqrt(400)))
```

Mathematical Functions

Function & Returns (description)

1 abs(x)The absolute value of x: the (positive) distance between x and zero.

2 ceil(x)The ceiling of x: the smallest integer not less than x

3 cmp(x, y)-1 if x < y, 0 if x == y, or 1 if x > y

4 exp(x)The exponential of x: e^x

5 floor(x)The floor of x: the largest integer not greater than x

6 log(x)The natural logarithm of x, for x> 0

7 log10(x)The base-10 logarithm of x for x> 0.

8 max(x1, x2,...)The largest of its arguments: the value closest to positive infinity

9 min(x1, x2,...)The smallest of its arguments: the value closest to negative infinity

10 modf(x)The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.

11 pow(x, y)The value of x**y.

12 round(x [,n])**x** rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.

13 sqrt(x)The square root of x for x > 0

Mathematical Functions

Function & Returns (description)

14 cos(x)Return the cosine of x radians.

15 sin(x)Return the sine of x radians.

16 tan(x)Return the tangent of x radians.

17 degrees(x)Converts angle x from radians to degrees.

18 radians(x)Converts angle x from degrees to radians.

19 gcd(x,y)Returns the Greatest Common Divisor of x and y.

20 isnan(x) Checks whether x is not a number.

21 isinf(x) Checks whether x is infinity. Returns True if x is a positive or negative infinity.

Random Functions

Function & Returns (description)

1 choice(seq)A random item from a list, tuple, or string.

2 randrange ([start,] stop [,step])A randomly selected element from range(start, stop, step)

3 random()A random float r, such that 0 is less than or equal to r and r is less than 1

4 seed([x])Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.

5 shuffle(lst)Randomizes the items of a list in place. Returns None.

6 uniform(x, y)A random float r, such that x is less than or equal to r and r is less than y

7 randint(start,end) Generates random integers within range.

Random Functions

Example: Generate 10 random integer numbers.

```
import random
for _ in range(10):
    value = random.randint(0, 10)
    print(value)
```