Overwiew

Akhilesh Joshi	1020100**16**
Rajesh Mohite	1020100**23**
Chetan Parmar	1020100**27**
Dipesh Shah	1020100**59**

Brief History of Python

- Invented in the Netherlands, early 90s by Guido van Rossum
- Named after Monty Python
- Open sourced from the beginning
- Considered a scripting language, but is much more
- Scalable, object oriented and functional from the beginning
- Used by Google from the beginning
- Increasingly popular

Python’s Benevolent Dictator For Life

“Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressive-ness is endangered.”

- Guido van Rossum

What is python?

Python is an **interpreted** *programming language.*

Its a language which is never actually compiled in full.

Seems a waste of time? Maybe, but the fact is that when you come back another day, you might be using a Windows instead of a Mac. You might send the program to a friend, who uses another type of computer. Or you might post your program on the Internet, where everyone using all different types of systems might download it.

That is the wonder of an interpreted programming language - it is like a language that EVERYONE can understand.

What is python?

- A programming language
  - Features from Perl and Java, with influences from C, C++, Scheme, Haskell, Smalltalk, Ada, Simula,…
- Open Source
  - Free; source code available
  - Download on your own system
- First release Feb 1991: 0.9.0
- Current version: 3.2.3
- Evolutionary policy for changes between versions
- Backward incompatibility issues are rare
  - But they do exist…
  - Largest change: from 1.5 to 2.0

How to install python?

- First download Python-2.4.1.exe (Windows)
- Run the file you just downloaded, and follow the prompts.
- To test if that just worked, type this in your DOS window:
  - “python –V”

- Python is default in Linux.
  - $python --version

Opening IDLE (Windows)

- Go to the start menu, find Python, and run the program labeled 'IDLE' (Integrated Development Environment)

- Now you are in the IDLE environment. This is the place you will be spending most time in. Here you can open a new window to write a program, or you can simply mess around with single lines of code, which is what we are going to do. Type the following and press enter:
  - “>>> print "Hello, World!"”

- You just created a program, that prints the words 'Hello, World'.

- This is useful for testing things.

Opening IDLE (Linux)

- As it is default in Linux; open terminal and
  - $python

```
Python 2.5 (r25:51908, Nov 6 2007, 16:54:01)
[GCC 4.1.2 20070925 (Red Hat 4.1.2-27)] on linux2
Type "help", "copyright", "credits" or "license" for
more info.
>>>
```

- Or install IDLE from package manager for latest version of PYTHON
- After installation open IDLE

Comments, Please

The final thing you'll need to know to move on to multi-line programs is the comment. Type the following

```
# This is a single line comment

‘’’
 This is a multi line comment
’’’
```

Arithmetic Operators

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Comparison Operators

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!= <>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	c = a + b assigns a + b into c
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	c += a is c = c + a
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	c -= a is c = c – a
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	c *= a is c = c * a
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	c /= a is c = c / a
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	c %= a is c = c % a

Bitwise Operators

a=60; b=13

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(a | b) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(a ^ b) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	(~a) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	a << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 will give 15 which is 0000 1111

Logical Operators

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

Membership & Identity Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here **in** results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here **not in** results in a 1 if x is a member of sequence y.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Programs in a file

Editing in Notepad

Writing programs in python to a file is VERY easy. Python programs are simply text documents - you can open them up in notepad, and have a look at them, just like that. So, go and open notepad. Type the following:

```
#A simple program.
print "Mary had a little lamb," print "it's fleece was white as snow;"
print "and everywhere that Mary went“,
print "her lamb was sure to go.“            [Save the file as 'mary.py' ]
```

- Now, open up the Python IDLE program and run it!
- You can also run the program from your command line program (e.g. MSDOS) - Open the prompt up, type 'cd path\to\your\file' then type 'python mary.py'. Your program will now execute in the command line.
- You can also use IDLE to create Python programs.

Variables & Strings

- **Variables**
  - Variables are the placeholder in memory.

```
counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string
```

Loosely Type Programming Language.

- **Strings**
  - A variable that holds text is called a string.

Conditional statement

- Conditional statement is a section of code which only run if condition are met.
- Conditional statement run only once.
- The most common conditional in any program language is ‘IF’ statement.

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

While Loop

You tell computer to repeat bit of code between Point A to Point B , until the time comes that you need it to stop, such a thing called loop

```
<syntax>
initialization;
while expression:
      statement(s)
```

Functions

Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.

The code block within every function starts with a colon (:) and is indented.

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
def functionname( parameters ):
      "function_docstring"
       function_suite
       return [expression]
```

Lists

- **Lists** are what they seem - a list of values. Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc. You can remove values from the list, and add new values to the end.

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

Tuples

- **Tuples** are just like lists, but you can't change their values. The values that you give it first up, are the values that you are stuck with for the rest of the program. Again, each value is numbered starting from zero, for easy reference.

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

Dictionary

- **Dictionaries** are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'.

```
dict = {'Alice': '2341', 'Beth': '9102'}
dict1 = { 'abc': 456 };
dict2 = { 'abc': 123, 98.6: 37 };
```

The For Loop

- The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- Basically, the for loop does something *for* every value in a list. The way it is set out is a little confusing, but otherwise is very basic.

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

Class

- What is a class? Think of a class as a blueprint. It isn't something in itself, it simply describes how to make something. You can create lots of objects from that blueprint - known technically as an *instance*.
- So how do you make these so-called 'classes'? very easily, with the class operator:

```
class class_name:
      [statement 1]
      [statement n]
```

Imports

- You need to create a module or you can import system defined module.

Eg.

```
def print_func( par ):
      print "Hello : ", par
      return



import hello
hello.print_func("Zara")
```

File I/O

- Taking input from the user

```
file object = open(file_name [, access_mode][, buffering])
```

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode:** The access_mode determines the mode in which the file has to be opened ie. read, write append etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r)
- **buffering:** If the buffering value is set to 0, no buffering will take place. If the buffering value is 1, line buffering will be performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action will be performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

File I/O

```
>>> f = open(“names.txt")
>>> f.readline()
```

Or a shorter way

```
lst= [ x for x in open("text.txt","r").readlines() ]
```

File Output

```
input_file = open(“in.txt")
output_file = open(“out.txt", "w")
for line in input_file:
      output_file.write(line)
```

Exception Handling

- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it can't cope with, it raises an exception. An exception is a Python object that represents an error.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it would terminate and come out.

Exception Handling

- If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

```
try:
      You do your operations here;
except ExceptionI:
      If there is ExceptionI, then execute this block.
except ExceptionII:
      If there is ExceptionII, then execute this block.
else:
      If there is no exception then execute this block.
```

Python Features

no compiling or linking	rapid development cycle
no type declarations	simpler, shorter, more flexible
automatic memory management	garbage collection
high-level data types and operations	fast development
object-oriented programming	code structuring and reuse, C++
embedding and extending in C	mixed language systems
classes, modules, exceptions	"programming-in-the-large" support
dynamic loading of C modules	simplified extensions, smaller binaries
dynamic reloading of C modules	programs can be modified without stopping

Python Features

universal "first-class" object model	fewer restrictions and rules
run-time program construction	handles unforeseen needs, end-user coding
interactive, dynamic nature	incremental development and testing
access to interpreter information	metaprogramming, introspective objects
wide portability	cross-platform programming without ports
compilation to portable byte-code	execution speed, protecting source code
built-in interfaces to external services	system tools, GUIs, persistence, databases, etc.

References:

http://www.python.org

http://sthurlow.com/python/

http://www.tutorialspoint.com/python/

http://www.avatar.se/python/

http://www.cs.columbia.edu/~hgs/teaching/ap/

http://homes.cs.washington.edu/~stepp/bridge/2007/

http://www.csc.villanova.edu/~nlp/python

http://www.cs.unb.ca/~boley/FLP/python-intro

http://www.wag.caltech.edu/home/rpm/python

Any Questions……………………………?

»Open For Discussion