PYTHON PROGRAMMING (CS0452)

Unit 1

Introduction

Guido Van Rossum, researcher from Netherlands, developed Python during 1989 - 1991

Why the name Python??

Rossum was reading the scripts of "Monty Python's Flying Circus" - a BBC comedy series of the 70s.
Name had to be small, unique and mysterious!!

Why do people use Python?

Primary factors cited in Python users:

1. **Software Quality:**

Python code is designed to be readable, reusable and maintainable.
Uniformity in Python code makes it easy to understand (even other's code!)
Python has a deep support for more advanced software reuse mechanisms, such as Object-oriented and functional programming.

2. **Developer productivity:**

Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++ and Java.Python is typically one-third to one-fifth the size of equivalent C++ or Java code. So, there is less to type, less to debug and less to maintain.

3. **Program portability :**

Porting Python codes between all major computer platforms is usually just a matter of copying a script's code between machines. Python offers multiple operations for coding portable graphical user interfaces, database access programs, web-based systems and more.

4. **Support libraries:**

Python comes with a large collection of prebuilt and portable functionality in its standard library. It ranges from text pattern matching to network scripting at the application-level tasks to an extended vast collection of third-party application support software. Python's third-party domain offers tools for website construction, numeric programming, serial port access, game development and much more!

5. **Component integration:**

Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms. Python code can also invoke C and C++ libraries, can be called from C and C++ programs, can integrate with Java and .NET components, can communicate over frameworks such as COM and Silverlight, can interface with devices over serial ports and can interact over networks with interfaces like SOAP, XML-RPC, and CORBA.

6. **Enjoyment:**

Due to the ease of use and built-in tool set, programming in Python will be more a 'pleasure' than a 'chore'.

Who Uses Python today?

- Google makes extensive use of Python in its web search systems
- The Popular YouTube video sharing service is largely written in Python
- The Dropbox storage service codes both its server and desktop client software primarily in Python
- The Raspberry Pi single-board computer promotes Python as its educational language
- EVE Online, a massively multiplayer online game (MMOG) by CCP Games, uses Python broadly
- The widespread BitTorrent peer-to-peer file sharing system began its life as a Python program
- Industrial light & Magic, Pixar, and others use Python in the production of animated movies
- ESRI uses Python as an end-user customization tool for its popular GIS mapping products
- Google's App Engine web development framework uses Python as an application language
- Maya, a powerful integrated 3D modeling and animation system, provides a Python scripting API
- The NSA uses Python for cryptography and intelligence analysis
- iRobot uses Python to develop commercial and military robotic devices
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm and IBM use Python for hardware testing
- JPMorgan Chase, UBS, Getco and Citadel apply Python for financial market forecasting
- NASA, Los Alamos, FErmilab, JPL, and others use Python for scientific programming tasks

What can we do with Python?

1. **Systems Programming:**

Python's built-in interfaces to OS services make it ideal for writing portable, maintainable system-administration tools and utilities.
Python programs can search files and directory trees, launch other programs, do parallel processing with processes and threads, and so on.

2. **GUIs:**

Python comes with a standard Object-oriented interface to the Tk GUI API called 'tkinter' (Tkinter in 2.x) that allows Python programs to implement protable GUIs with a native look and feel.
Python/tkinter GUIs are platform independent!
With proper library, you can also use GUI support in other toolkits in Python, such as Qt with PyQt, GTK with PyGTK, MFC with PyWin32, .NET with IronPython and Swing with Jython or JPype

3. **Internet Scripting:**

With standard Internet modules, Python programs can perform a wide variety of networking tasks in both client and server modes.
Scripts can communicate over sockets; extract information sent to server-side CGI scripts; transfer files by FTP; parse and generate XML and JSON s; documents; send, receive, compose and parse email; fetch web pages by URLs; parse the HTML of fetched web pages; communicate over XML-RPC, SOAP and Telnet and more!!

4. **Component Integration:**

Tools such as the SWIG and SIP code generators can automate much of the work needed to link compiled components into Python for use in scripts and the Cython system allows coders to mix Python and C-like code.
Larger frameworks, such as Python's COM support on Windows, the Jython Java-based implementation, and IronPython .NET-based implementation provide alternate ways to script components.

5. **Database Programming :**

There are Python interfaces to all commonly used relational database systems - Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite and more. The Python world has also defined a portable database API for accessing SQL database systems from Python Scripts, which looks the same on a variety of underlying database systems.
In the non-SQL department, Python's standard 'pickle' module provides a simple object persistence system - it allows programs to easily save and restore entire Python objects to files and file-like objects.

6. **Rapid Prototyping:**

To Python programs, components written in Python and C look the same. Due to this, it is possible to prototype systems in Python initially and then move selected components to a compiled language such as C or C++ for delivery.

7. **Numeric and Scientific Programming:**

Python is also heavily used in numeric programming. The NumPy is a high-performance numeric programming extension for Python which includes advanced tools as an array object, interfaces to standard mathematical libraries and much more.
The popular SciPy and ScientificPython extensions provide additional libraries of scientic programming tools and use NumPy as a core component.
The PyPy implementation of Python has also gained traction in the numeric domain, in part because heavily algorithmic code of the sort that's common in this domain can run dramatically faster in PyPy - often 10x to 100x quicker!

8. **Gaming, Images, Data mining, Robots, Excel, ... :**

You'll find a tool that will allow you to use Python to do:
- Game programming and multimedia : pygame, cgkit, pyglet, PySoy, Panda3D, ...
- Serial port communication onWindows, Linux and more : PySerial extension
- Image processing : PIL and its newer Pillow fork, PyOpenGL, Blender, Maya, ...
- Robot control programming : PyRo toolkit
- Natural language analysis : NLTK package
- Instrumentation on Raspberry Pi and Ardino boards
- Mobile computing with ports of Python to Google Android and Apple iOS platforms
- Excel spreadsheet function and macro programming : PyXLL or DataNitro add-ins
- Media file content and meta data tag processing : PyMedia, ID3, PIL/Pillow, ...
- Artificial Intelligence : PyBrain neural net library and Milk machine learning toolkit
- Expert system programming : PyCLIPS, Pyke, Pyrolog, pyDatalog
- Network monitoring : zenoss
- Python-scripted design and modeling : PythonCAD, PythonOCC, FreeCAD, ...
- Document processing and generation : ReportLab, Sphinx, Cheetah, PyPDF, ...
- Data visualization : Mayavi, matplotlib, VTK, VPython, ...
- XML parsing : xml library package, xmlrpclib module and third-party extensions
- JSON and CSV file processing : json and csv modules
- Data mining : Orange framework, the Pattern bundle, Scrapy and custom code

[Note: You can even play solitaire with PySolFC program :)]

What are Python's technical Strengths?

1. **It's both Object-oriented and Functional**

Its class model supports advanced notions such as Polymorphism, Operator Overloading, and Multiple Inheritance. With its simple syntax, OOP is remarkably easy to apply.
Python's OOP nature makes it ideal as a scripting tool for other object-oriented system languages - eg. Python programs can subclass (specialize) classes implemented in C++, Java and C#.
Python supports both procedural and object-oriented programming modes. Python has built-in support for functional programming which serves as both complement and alternate to its OOP tools.

2. **It's free**

Python is completely free to use and distribute. The entire Python system's source code can be fetched from the Internet for free. Python online user community responds to user queries faster than the most commercial software help desks would.

3. **It's Portable**

Standard implementation of Python is written in portable ANSI C and it compiles and runs on virtually every major platform currently in use. Hence, Python programs using the core language and standard libraries run the same on Linux, Windows and most other systems with a Python interpreter.

4. **It's Powerful**

Python is in between traditional scripting languages and system development languages. Python provides simplicity and ease of use of a scripting language along with more advanced software-engineering tools found in compiled languages. Features of Python's toolbox include: Dynamic typing, Automatic memory management, Programming-in-the-large support, Built-in object types, Built-in tools, Library utilities, Third-party utilities.

5. **It's mixable**

Python programs can easily be "glued" to components written in other languages in different ways:

- Using Python's C API lets C programs call and be called by Python programs.
- Mixing Python with libraries coded in C or C++ makes it easy-to-use frontend language and customization tool.

6. **It's relatively Easy to Use**

Programming in Python is simple compared to C++, Java and C#. A Python program can simply typed and run, no intermediate compile and link steps required. Its sometimes called as 'executable pseudocode'!

7. **It's relatively easy to Learn**

Core Python programming is relatively easy to learn (matter of days!). But true investment required to master Python is worthwhile - the programming skills can be applied to nearly every computer application domain.

Python is an interpreter language (uses 'byte-code' compilation!)

(Byte-code: is a lower-level, platform independent representation of source code)

Two ways to use the Python interpreter:

1) Interactive mode
2) Scripting mode

1. **Interactive mode:** A way of using the Python interpreter by typing commands and expressions at the prompt.

In the command prompt, type 'python'

```
>>>
# This is the chevron prompt or the python interpreter
```

2. **Scripting mode:** A python program stored in a file (which will be interpreted)

eg:

Our first Python program: Hello, world! program

In interactive mode:

```
# always check the version of Python interpreter you are working on
# in Python 2.x,
>>>print 'Hello, world!'
Hello, world!
>>>
```

```
# in Python 3.x,
>>>print('Hello, world!')
Hello, world!
>>>
```

In scripting mode:
Type the print statement using an editor and save the file with extension, '.py'

```
# in the command prompt,
$python filename.py
Hello, world!
$
```

Note:
It's preferable to use the scripting mode for large programs, while interactive mode is preferred for simple statements.

Using the Python interpreter as a calculator

```
>>>2+3
5
>>>(5-3)*6
12
>>>x=5
>>>y=10
>>>x+y
15
```

Variable names in camel case naming convention:
    eg: NumberOfGirls, ClassCount, CourseAverage

Data types: integer, floating point, string, Boolean

Python keywords

In Python 2.x:

and	*del*	*for*	*is*	*raise*	*assert*
elif	*from*	*lambda*	*return*	*break*	*else*

global	*not*	*try*	*class*	*except*	*if*
or	*while*	*continue*	*exec*	*import*	*pass*
yield	*def*	*finally*	*in*	*print*	*as*
with					

Note:
In Python 3.x, '*exec*' is not a keyword, '*nonlocal*' is included.

Built-in functions
Usage:

```
import module
module.function_name(args)

from module import *
function_name(args)
```

eg:

```
import math
x=5
print('Square root = ', math.sqrt(x))
```

User defined functions (UDFs)
with/without arguments, with/without return value

Syntax

Function definition :

```
def function_name(argument_list):          #function definition
    statement_block
    return_statement
```

Function call:

```
function_name(argument_list)               #function call
```

Eg:

```
def print_lines():
    print('Hello there!!')
    print('Bye for now! :)')
    return

print_lines()
```

Types of functions (both built-in and user-defined)

fruitful functions

void functions

Exercise:

Write a Python program that draws a grid (using UDF):

```
+ - - - - + - - - - +
|         |         |
|         |         |
|         |         |
+ - - - - + - - - - +
```

Conditionals, Recursion and Iteration

Arithmetic operators: +, - , *, / , //, %, **

Boolean Expressions

returns True or False

relational operators:

!=, ==, <=, <, >, >=

Logical Operators

and, or, not

Conditional Execution

```
if expn:
   statement
```

Alternative execution

```
if expn:
   statement1
else:
   statement2
```

Chained Conditionals

```
if expn1:
   statement1
elif expn2:
   statement2
else:
   statement3
```

Nested Conditionals (avoid usage)

```
if expn1:
    statement1
else:
    if expn2:
        statement2
    else:
        statement3
```

Use of pass statement:

eg:
```
if x < 0:
    pass    #a place keeper for code you want to write in future
```

Reading input from user

```
# ver. 2.x
x = raw_input('Enter value of x: ')

#ver. 3.x
x = input('Enter value of x: ')
```

Note:
The input read from keyboard is always in string data type.

Exercise

I (a) Write a function named is_triangle that takes 3 integers as arguments and prints either 'YES' or 'NO', depending on whether you can or cannot form a triangle from the 3 integers.

(b) Write a function that prompts the user to input the length of the 3 sides, converts them into integers.
[Hint: any one length cannot be greater than sum of the other two]

II. Write a Python program that reads a number from the user and prints the square of the number if and only if: the number is a positive odd integer.

Recursion

Syntax:
```
def function_name(arg):
    if expn: return value
    else: function_name(arg)
```

Infinite Recursion

Syntax:

```
def function_name(arg):
    function_name(arg)
```

Exercise:

I. Find the factorial of a positive integer using recursion
II. Find the sum of Natural numbers using recursion
III. Find the gcd of 2 numbers using recursion

Fruitful Functions (UDFs)

```
def function_name(arguments):
    statements
    return expression

result=function_name(arguments)
```

eg:

```
def area(radius):
    temp = math.pi * radius**2
    return temp
```

Composition

The ability of one function being called from another function

eg:

```
def circle_area(xc,yc,xp,yp)
    radius = distance(xc,yc,xp,yp)
    result = area(radius)
    return result
```

Boolean functions

Functions that return Boolean values

eg:

```
def is_divisible(x,y):
    if x % y == 0:
        return True
    else:
        return False


>>>is_divisible(6,4)
False
```

```
>>>is_divisible(6,3)
True
```

Checking Types (data types of variables)

Using isinstance()

Syntax:
isinstance(var,data_type)

eg:
isinstance(x,int)

Exercise:

I. The Ackermann function, A(m,n) is defined as:

```
            n+1                     if m=0
A(m,n) =    A(m-1,1)                if m >0 and n =0
            A(m-1, A(m,n-1))        if m > 0 and n >0
```

Write a recursive function named ack() that evaluates Ackermann's function.

II. A number, a, is a power of b if it is divisible by b and a/b is a power of b. Write a recursive function called is_power() that takes parameters a and b and returns True if a is a power of b.
[Note: you will have to think about the base case.]

Iteration

1. The *while* statement
Syntax:

```
while expn:
    statements
     . . .
next statement
```

eg:

```
def countdown(n):
    while n > 0:
        print n
        n = n - 1
    print 'Done!!'
```

2. *break* statement
Used to jump out of a loop

eg:(1)

```
while True:
    line = raw_input('> ')
    if line == 'done':
        break
    print line
print 'End'
```

eg:(2)

```
# to estimate the square root of a number, 'a'
# using Netwton's method, y = (x+a/x)/2
# x & y are estimates of the square root
while True:
    print x
    y = (x +a/x)/2
    if y == x: break
    x = y
```

3. The *for* loop

Syntax:

```
for var in iterative_object:
    statements
```

iterative_object??

eg:

(1)
```
for i in range(1, 10):
    print(i)
```

(2)
```
for ch in str1:
    print(ch)
```

Exercise:

I. The built-in function 'eval' takes a string and evaluates it using the Python interpreter. Write a function called 'eval_loop()' that iteratively prompts the user, takes the resulting input and evaluates it using 'eval' and prints the result.

II. Write a function test_sqroot that prints a table like this:

```
num   math.sqrt(num)     Newton's method
1.0   1.0                1.0
2.0   1.414...           1.414...
3.0   ...                ...
```

The first column is the number, second column is the square root of number using math.sqrt() and the third column is the square root of number estimated from Newton's method.