Welcome to GKTCS

IT Training .Consultancy . Software Development. Staffing

Surendra Panp

Director, GKTCS Innovations Pvt. Ltd, Pune.

16 + Years of Experience (MCA, PGDCS, BSc. [Electronics] , CCNA)

- Founder,GKTCS Innovations Pvt. Ltd. Pune [Nov 2009 – Till date]
- 500 + Corporate Training for HP, IBM, Cisco,Wipro, Samsung etc.
- **Skills**
  - **Python, Perl,Jython, Django, Android ,**
  - **Ruby, Rail, Cake PHP, LAMP**
  - Data Communication & Networking, CCNA
  - UNIX /Linux Shell Scripting, System Programming
  - CA Siteminder, Autosys, SSO, Service Desk, Service Delivery
- Author of 4 Books
- National Paper Presentation Awards at BARC Mumbai

ICE Breaking

- Form a group of 2 or 3.
- Listen to your colleague.
- Appreciate everyone.
- Know everyone skills and appreciate the same.
- Learn in team.
- Cooperate to each other
- Please switch off / Silent your Mobile phone.

Python Programming

Surendra R. Panpaliya
M:9975072320
surendrarp@gktcs.com
www.gktcs.com

Agenda

Day	**Module**	**Topics**
Day 1	Module 1	Introduction to Python
	Module 2	Simple Program
	Module 3	Basic Language Construct
	Module 4	Control Structure
	Module 5	Functions
	Module 6	Data Structures
Day 2	Module 7	Modules
	Module 8	Files Handling
	Module 9	Exception Handling
	Module 10	Regular Expression

Module 1

Introduction to Python

Introduction to Python

- What is Python?
- Who Uses Python?
- Features of Python
- Why Python

What is Python?

- Python is an interpreted, interactive, object oriented language.
- It incorporates modules, exceptions, high level data types, dynamic typing and classes.
- It has interfaces to system calls and libraries.
- It is portable.

What is Python?

- Python is an easy to learn, powerful programming language.
  - Efficient high-level data structures
  - Simple approach to object-oriented programming.
  - Elegant syntax and dynamic typing
  - Up-and-coming language in the open source world

Brief History of Python

- Invented in the Netherlands, early 90s.
- Invented by Guido van Rossum.
- More than a scripting language.
- Scalable and object oriented from the beginning.
- Evolved from one man’s hobby to a project coded by a team of hundreds of developers around the world.

Who Uses Python?

- Little guys…
  - Mom and pop ISPs
  - Consultants trying to deliver projects on-budget
  - Domain specialists trying to beat their competitions using C or java

  - But also…

Google

- “Python has been an important part of **Google** since the beginning”
  - Peter Norvig, director of search quality at Google, Inc.

Nasa

- “NASA is using Python to implement a … system which will be the core infrastructure for its next generation collaborative engineering environment. Python has met or exceeded every requirement we've had.”

  – Steve Waterbury, Software Group Leader, NASA STEP Testbed .

Industrial Light & Magic

- “Python plays a key role in our production pipeline. Without it a project the size of Star Wars: Episode II would have been very difficult to pull off.”
  - Tommy Burnette, Senior Technical Director, Industrial Light & Magic .

Thawte Consulting

- “Python makes us extremely productive, and makes maintaining a large and rapidly evolving codebase relatively simple.”

  – Mark Shuttleworth, founder and amateur astronaut

Python's Business Case

- Python can displace many other languages in the organization.
- The python interpreter is free.
- Python is legally unencumbered.
- Professional programmers find python more flexible than most languages.
- Amateur programmers are (often) more comfortable than with Perl or java.

- Python programmers are more productive!

Usability Features

- Very clear syntax.
- Obvious way to do most things.
- Huge amount of free code and libraries.
- Interactive.
- Only innovative where innovation is really necessary.
  - Better to steal a good idea than invent a bad one!

Why Python

- High Level Language (HLL)
  - Rapid development
  - Very Readable Code
    - Up to 10x shorter code than Java/C/C++
  - Object Oriented
  - Dynamically Typed
- **Numerous** standard packages and modules
- **Numerous** third-party packages
  - Twisted, NumPy, SciPy, PIL, M2Crypto

Python: head to head

Compared to C/C++

- Much higher level.
- Get more done in less time.
- Richer built-in data structures, type and libraries
- Much less error prone (buffer overruns, pointer dereferences, etc)
- Easier to write secure code!

- But Python code can execute much slower than C/C++ code!

Python vs Perl

- Different philosophy.
- Perl emphasizes support for common application-oriented tasks, e.g. by having built-in regular expressions, file scanning and report generating features.
- Python emphasizes support for common programming methodologies such as data structure design and object-oriented programming, and encourages programmers to write readable (and thus maintainable) code by providing an elegant but not overly cryptic notation.

Python vs Perl

1. Perl is a lot older than Python and has a much wider selection modules available.
2. Perl uses the traditional braces to mark statement blocks while Python uses the indentation for the same purpose.
3. Python code is intuitive and easier to learn compared to Perl.
4. Perl is harder to handle and debug compared to Python when the code starts to grow.

Compared to Perl

- Syntactically simpler.
- More object oriented.
- Easier to extend.
- Perl development has stalled waiting for v6.
- On the other hand:
  - Python has slower regular expressions...
  - Python does not have a distributed module repository like CPAN
  - Python's user base is still smaller

Compared to PHP/Javascript

- Excellent for Web apps (PHP on server, Javascript on client) but not much else.

- Python can be used for your Web apps, your complicated algorithms, your GUIs, your COM components, an extension language for Java programs, …
- Even in Web apps, Python handles complexity better.

Ruby/Lua/IO/Rebol/…

- Less mature languages (e.g. threading, Unicode etc.)
- Poorer documentation.
- Fewer third-party libraries/books/conferences/...

- But they all have some very nice ideas from a cool-new-language point of view.

Compared to Java

- Java is more difficult for amateur programmers.
- Static type checking can be inconvenient and inflexible.
- Puritanical OO can be inconvenient.
- Bottom line: Java can make projects harder than they need to be.

Why Not Java: Political

- "100% pure Java" gets in the way.
- The Java environment punishes platform-specific interoperability. (e.g. getenv is deprecated)
- Java is designed to have interoperability limitations.
- Embedding Python in an app is much easier!

Jython (nee JPython)

- A reimplementation of Python in Java.
- Seamless, magical interface to Java libraries
- Good scripting solution for Java.
- Compiles Python classes to Java classes
- Embedded interpreter allows interactive coding.
- Access to all Java classes.
- For better or worse: maintains Java's security/platform-independence bubble.

Python Limitations

- Not the fastest executing programming language:
  - C/C++ is naturally fast
  - Perl's regular expressions and IO are a little faster
  - Some Java implementations have good JITs
  - But Python also has some speed advantages:
    - Psyco is a third-party open source JIT for Python
    - Fast implementations of built-in data structures
    - Pyrex compiles Python code to C
- Dynamic type checking requires more care in testing.
- Language changes (relatively) quickly: this is a strength and a weakness.
- Python is not as resume friendly as Java/C#. Net.

Module 2: Simple Program

- Python shell
- Python command
- Prepare source File and execute the code
- Print “Hello World”
- Comments
- Help command

Starting up of Python…

- Double click on the python icon on the desktop

OR

- Type in python on command prompt

Without much ado…

```
ActivePython 2.7.1.4 (ActiveState Software
  Inc.) based on
Python 2.7.1 (r271:86832, Feb  7 2011,
  11:30:38) [MSC v.1500 32 bit (Intel)] on
  win32
Type "copyright", "credits" or "license()"
  for more information.
>>> print "Hello World"
Hello World
```

How to quit

- Windows users - Press Ctrl-Z followed by enter key.
- Linux/BSD/Mac OS X users - Press Ctrl-D.

Using Source Files

- Open your favorite editor
- print ‘Hello World’

```
>python helloworld.py
Hello World
```

- How It Works
- The print *statement* is supplied with the text Hello World, and it promptly prints it to the screen.

Help command

Getting Help

- Python has two closely-related help modes. One is the general “help” utility, the other is a help function that provides the documentation on a specific object, module, function or class.
- **The help() Utility**

  Help is available through the help() function.

  If you enter just ‘help()’ you will enter the online help utility. This help utility allows you to explore the Python documentation.
- The interaction looks like this:

```
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help("EXPRESSIONS")
```

Module 3: Basic Language Consructs

- Data types and Variables
- String type
- Format method
- Operators and Expressions
- Indentation

Numeric Types

- int: integral number, e.g. "x=5"
- float: accuracy depends on platform, e.g. "x=3.14"
- complex: real+imag., "x=5.3+3.2j”
- bool: True, False (mathematically equivalent to 1 and 0)

Integers

- Simple number literals generate integers.

```
>>> x = 5
```

- We can also convert other types to integers:

```
>>> x=  int(“5”)
>>> print int(5.9)
5
```

- Or round floats to the closest integer

```
>>> print round(5.9)
5
```

Standard Math Operations

```
>>>  5+3
8
>>>  5*2+3
13
>>>  3+5*2
13
```

Integer Division

```
>>> 5/2 # Python wart - don't do this
2
>>> 5//2 # Explicit truncating div
2
>>> 5.0/2 # Explicit float division
2.5
>>> float(5)/2 # Another way
2.5
```

Exponents

- The “**” operation means “to the exponent of”.

```
>>>  2**30
1073741824

>>>  2**90
1237940039285380274899124224
```

Integers Are Unbounded

```
>>> 2**150
1427247692705959881058285969449495136382746624
>>> 2**1000
107150860718626732094842504906000181056140481170
  055360744375038837035105112493612249319837881
  569585812759467291755314682518714528569231404
  359845775746985748039345677748242309854210746
  050623711418779541821530464749835819412673987
  675591655439460770629145711964776865421676604
  298316526243868372056680693760
>>> 2**100000
999002093014384507944032764330033590980429139054
  181691771529273863145832464257348327487331332
  4…
```

Float Examples

```
>>> print 4 * 2.5 / 3.3
3.0303030303
>>> print 7.0 / 2
3.5
>>> print float("7.0")
7.0
```

- Floats have all of the complexities of floats in any other language! Beware!

Mathematical Functions

- abs(*x*) – absolute value
- divmod(a, b) - divide a by b and return both the quotient and the remainder
- hex(x) - Convert an integer number (of any size) to a hexadecimal string.
- oct(x) - Convert an integer number (of any size) to an octal string.
- round(x[, n]) - Return the floating point value x rounded to n digits after the decimal point.

Booleans

- True and False are the only values of type “bool”.
- True can be treated as “1” and False as “0” in mathematical expressions:

```
>>> print True+True+True-False
3
>>> print False==0
True
```

Complex Examples

```
>>> print complex(3,1)*3
(9+3j)
```

- Complex numbers can also be created with engineer’s syntax:

```
>>> print (1+2j)/(1+1j)
(1.5+0.5j)
```

Objects All the Way Down

- Everything in Python is an object
- Integers are objects.
- Characters are objects.
- Complex numbers are objects.
- Booleans are objects.
- Functions are objects.
- Methods are objects.
- Modules are objects

Object Type and Identity

- You can find out the type of any object:

```
>>> print type(1)
<type 'int'>
>>>  print type(1.0)
<type 'float'>
```

- Every object also has a unique identifier (usually only for debugging purposes)

```
>>> print id(1)
7629640
>>> print id("1")
7910560
```

Introduction to Attributes

- Attributes are bits of information attached to objects.

```
>>> x = complex(3, 5)
>>> x.real
3.0
>>> x.imag
5.0
```

Introduction to Methods

- Methods are behaviours associated with objects.

```
>>> x = complex(3,5)
>>> x
(3+5j)
>>> x.conjugate()
(3-5j)
```

None

- “None” represents the lack of a value.
- Like “NULL” in some languages or in databases.
- For instance:

```
>>> if y!=0:
...     fraction = x/y
... else:
...     fraction = None
```

Strings

- Can be enclosed in single or double quotes:

```
>>> print ’spam eggs’
spam eggs
>>> print "spam eggs"
spam eggs
>>> print "doesn't"
doesn't
>>> print '"He said", she said.'
"He said”, she said.
>>> #backslash escapes quotes
>>> print ' "Isn\'t," she said.'
"Isn't," she said.
```

Triple Quoted Strings

- Triple quoted strings span lines
- They can also contain quote chars

```
print """
  Usage: thingy [OPTIONS]
  -h Display this "usage message"
  -H hostname Hostname to connect to
  """
```

Strings

```
mystr = "abc"                # assignment
mystr = mystr + "def"# = "abcdef"

for char in myStr: # iterate
    print char

myStr = str(5)      # create from number
```

- Pretty much any type can be converted into a string.

```
>>> str(file("catalog.txt"))
    "<open file 'catalog.txt', mode 'r' at 0x007E6A60>"
```

String methods

```
>>> s="hello there"
>>> print s.replace("h", "j")
jello tjere
>>> print s.capitalize()
Hello there
>>> print s.title()
Hello There
>>> print s.upper()
HELLO THERE
```

String methods

```
>>> string = "positively python pow
ered"
>>> print string.count("po")
2
>>> print string.startswith("abc")
False
>>> print string.endswith("ed")
True
>>> print string.find("pow")
18
```

Splitting and joining

```
>>> joiner = " and "
>>> names = ["Peter", "Paul","Mary"]
>>> joined = joiner.join(names)
>>> joined
'Peter and Paul and Mary'
>>> joined.split(" and ")
['Peter', 'Paul', 'Mary']
```

Characters

- Characters are just strings of length 1.

```
x = "a"
```

- The ord function returns the ASCII (or Unicode if appropriate) character number:

```
>>> ord("a")
97
>>> chr(97) # opposite of ord
"a"
```

Unicode

- Unicode data type:

normal string:	“hello”
unicode string:	u“hello”

- More unicode strings:

  u“\U0265” → µ

  u“\N{MICRO SIGN}” → µ
- unichr()
- The regular expression engine is Unicode aware

Literal Constants

- Numbers
  - Integers, long, floating point and complex numbers
- Strings
  - A sequence of characters in single or double quotes
- For C/C++ programmers
  - There is no separate char data type in Python

Variables

- Exactly what they mean – values can vary
- Used to store information
- Can start with alphabets or underscore
- Rest can be any alphanumeric

Using Python as a Calculator

```
>>> 2+2
4
>>> # This is a comment
... 2+2
4
>>> 2+2 # and a comment on the same line
  as code
4
>>> (50-5*6)/4
5
```

Using assignment operator

```
>>> width = 20
>>> height = 5*9
>>> width * height
900
```

Assignment to several variables

```
>>> x = y = z = 0 # Zero x, y and z
>>> x
0
>>> y
0
>>> z
0
```

Complex numbers

```
>>> 1j * 1j
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
```

Complex Numbers (cont.)

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Abs…

```
>>> a=3.0+4.0j
>>> float(a)
Traceback (most recent call last): File
  "<stdin>", line 1, in ? TypeError: can't
  convert complex to float; use abs(z)
>>> abs(a) # sqrt(a.real**2 + a.imag**2)
5.0
```

The _

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Strings

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't“
>>> '"Yes," he said.'
'"Yes," he said.'
```

String (cont…)

```
hello = "This is a rather long\n\
string containing several lines\n\
of text just as you would do in\n\
C.\n\
    Note that whitespace at the\n\
beginning of the line is\n\
significant."

print hello
```

Raw strings

```
hello = r"This is a rather\n\
long string containing \n\
several lines of text much\n\
as you would do in C."

print hello
```

Print function

- print evaluates each expression in turn and writes the resulting object to standard output
- A "\n" character is written at the end, unless the print statement ends with a comma.
- Standard output is defined as the file object named stdout in the built-in module sys. If no such object exists, or if it does not have a write() method, a RuntimeError exception is raised.

Concatenation

```
>>> word = 'Help' + 'A'
>>> word
'HelpA'
>>> '<' + word*5 + '>'
'<HelpAHelpAHelpAHelpAHelpA>'
```

Subscripting

```
>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[2:4]
'lp'
```

Subscripting

```
>>> word[:2] # The first two
  characters 'He`
 >>> word[2:] # Everything
  except the first two characters
  'lpA'
```

Unlike C ...

```
>>> word[0] = 'x'
Traceback (most recent call
  last): File "<stdin>", line 1,
  in ? TypeError: object doesn't
  support item assignment
```

Out of bound…

```
>>> word[1:100]
'elpA'
>>> word[10:]
''
>>> word[2:1]
''
```

What happens in case of negative indices?

Negative indices

```
>>> word[-1] # The last character 'A`
>>> word[-2] # The last-but-one character
  'p'
>>> word[-2:] # The last two characters
  'pA'
>>> word[:-2] # Everything except the last
  two characters
'Hel'
```

Easy way to remember

```
+---+---+---+---+---+
| H  | e  | l  | p  | A |
+---+---+---+---+---+
 0   1   2   3   4   5
 -5    -4  -3  -2  -1
```

Logical and Physical Lines

- 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.

Examples

- For example,

```
i = 5
print i
```

- is effectively the same as

```
i = 5;
print i;
```

Examples

- and the same can be written as
- i = 5; print i;
- or even
- i = 5; print i

Indentation

- Whitespace is important in Python.
- Actually, whitespace at the beginning of the line is important.
- This means that statements which go together **must** have the same level of indentation.
- Each such set of statements is called a **block**.

Error example

```
>>> i=5
>>> print i
5
>>>  print i
  File "<stdin>", line 1
    print i
    ^
IndentationError: unexpected indent
>>>
```

How to indent

- Do **not** use a mixture of tabs and spaces for the indentation as it does not work reliably across different platforms.
- The official recommended standard is *four spaces* for each level of indentation.
- You can also use tabs instead.

Operators and Expressions

- Operators
- Precedence
- Associativity
- Expressions

Operators

+ -	*	**	/	
%	<<	>>	&	|
^ ~	<	>	<=	
>=	==	!=		

Module 4: Control Structure

- If
- While loop
- For loop
- Break & Continue Statements

The if statement

```
x = int(raw_input("Please enter an
  integer: "))
if x < 0:
  x = 0
  print 'Negative changed to zero'
```

elif and else

```
elif x == 0:
    print 'Zero'
elif x == 1:
    print 'Single'
else:
    print 'More'
```

for Statements

```
>>>a = ['cat', 'window', 'defenestrate']
>>> for x in a:
          print x, len(x)
O/p
cat 3
window 6
defenestrate 12
```

The range function

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

Iterate over indices

```
>>> a = ['Mary', 'had', 'a',
  'little', 'lamb']
>>> for i in range(len(a)):
... print i, a[i]
...
```

Left overs…

- break
- continue
- pass (to be covered later)

Functions

- **Module 5: Functions**
- Simple Function declaration
- Function with parameter
- Variable Scope
- The “global” statement

Functions

```
def fib(n): # write Fibonacci
                #series up to n
  a, b = 0, 1
  while b < n:
     print b,
     a, b = b, a+b # Now call
     #the function we just defined:
fib(200)
```

Functions

- Note
  - A function definition introduces the function name in the current symbol table. The value of the function name has a type that is recognized by the interpreter as a user-defined function.
  - This value can be assigned to another name which can then also be used as a function.

Example

```
>>> fib
<function fib at  10042ed0>
>>> f = fib
>>> f(100)
1 1 2 3 5 8 13 21 34 55 89
```

Multiple arguments

```
def print_max(x, y):
  if x > y:
     print x, 'is maximum'
  else:
     print y, 'is maximum'
print_max(3, 4)
```

Local variables

```
def func(x):
  print 'x is', x
  x = 2
  print 'Changed local x to', x
>>>x = 50
>>>func(x)
>>>print 'x is still', x
```

global statement

```
def func():
  global x
  print 'x is', x
  x = 2
  print 'Changed global x to', x
>>>x = 50
>>>func()
>>>print 'Value of x is', x
```

Default argument values

```
def say(message, times = 2):
  print message * times

>>>say('Hello')
>>>say('World', 5)
```

Keyword arguments

```
def func(a, b=5, c=10):
    print 'a is', a, 'and b is',
  b, 'and c is', c
func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Return statement

```
def maximum(x, y):
  if x > y:
     return x
  else:
     return y
print maximum(2, 3)
```

DocStrings

```
def print_max(x, y):
    '''Prints the maximum of two numbers. The two values
    must be integers or strings containing integers.'''
    x = int(x) # convert to integers, if possible
    y = int(y)
    if x > y:
        print x, 'is maximum'
    else:
        print y, 'is maximum'
print_max(3, 5)
print print_max.__doc__
```

Variable argument in python

```
• def all_of(*args):
      result = 1
      for i in args:
          result *= i
      return result

print all_of(3,4,5)
```

Multiple Arguments

```
def  f(*args, **kwargs):
        print args
        print kwargs
>>> f(1, 2, "cow", "kitty")
(1, 2, "cow", "kitty") {}
>>> f(arg1=1, sample=2, name="cow", hero="kitty")
() {"arg1": 1, "sample": 2, "name": "cow", "hero":
   "kitty"}
>>> f(1, 2, name="cow", hero="kitty")
(1, 2) {"name": "cow", "hero": "kitty"}
```

Lambda calculus

- Alonzo Church, 1936
- An alternative view of the ’meaning of computation’
- Is the core foundation for:
  - Theoretical computer science
  - Functional programming languages
  - Constructive logics
- Think of it this way: if you didn’t have any programming language and had to build one, where would you start?

Syntax of the λ calculus

- **variable** Any variable "v" is an expression
- **application** Given any two expressions e1 and e2, then "(e1 e2)" is a valid expression, and denotes the application of e1 to e2.
- **abstraction** Given any variable v, and any expression e, then "(v · e)" is an expression representing a function with v as the formal parameter, and e as the body of the function
- Note that the application of f to x is written "functional style" as (f x) and not the more common f(x)

In python

```
>>>def make_incrementor(n):
...         return lambda x: x + n
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Summary

What did we learn today?

Recap and Memory Test.

Question and Answers

Contact Us on:

G K T C S Innovations Pvt. Ltd.

IT Training, Consultancy, Software Development, Staffing
#11,4th Floor,Sneh Deep, Near Warje Flyover Bridge,
Warje-Malwadi, Pune -411058, Maharashtra, India.
Mobile: +91- 9975072320, 8308761477
Email : surendra@gktcs.com
Web: www.gktcs.com