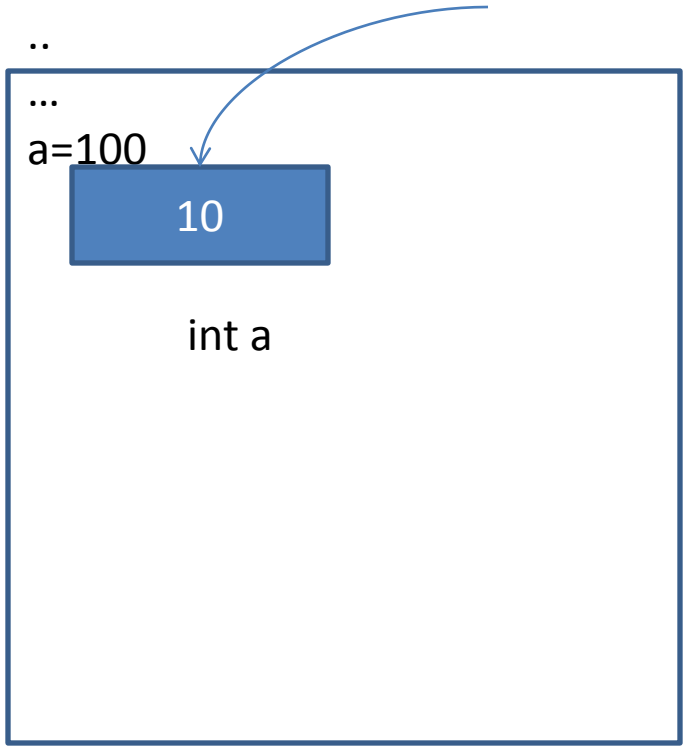
PYTHON

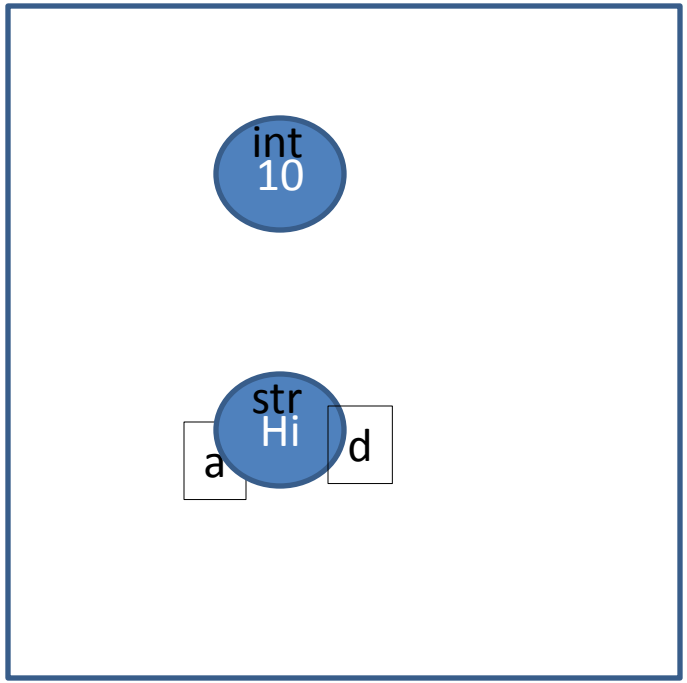
PYTHON PROGRAMMING

By
ADITYA PRABHAKARA

int a = 10
int a
..
...
a=100



a=10



Introducing Myself

Aditya S P (sp.aditya@gmail.com)
Freelance trainer and technologist

Boring Stuff about me:

- 14+ years of experience in development and training
- Started with Java, moved to Android and now working on Big Data Technologies

Interesting Things about me:

- Actually Nothing !

Getting to know you

Show of hands please!

- Any freshers in this group?
- What is the general development experience of this group
  - 0-2 years, 0-5 years, 5 and above
- What programming area are you currently working on?
  - Java, Web Stack, Analytics, Big data, any other
- Why are you learning python programming?
  - Sys admin, Web development, Data Analytics, IoT, any other

Agenda

- Python programming
- Advanced Python
- Object Oriented Programming in Python

Course Objectives

- At ease with python programming
- Pythonic way of coding
- Learn OOP in python

Chapter: Introduction

Python

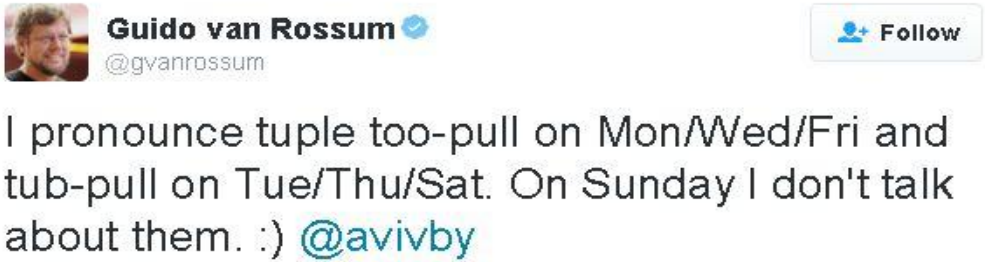
High Level
Interpreted
Dynamic Programming language
Multi-paradigm language
- OO
- Functional
- Procedural
- Imperative

The idea of Python started in 1980 and the implementation began by 1990
- Author: Guido Von Rossum

Guido van Rossum

In Guido van Rossum's words

Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).



A funny Guy : Check his answer on asking how to pronounce "tuple" – a data structure in Python

PYTHON

Zen of Python

A set of 20 aphorisms. My favs below

Define aphorism:
- a pithy observation which contains a general truth.
- a concise statement of a scientific principle, typically by a classical author.

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- Errors should never pass silently.
- In the face of ambiguity, refuse the temptation to guess.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

--Tim Peters

Why should I love python?

Python is very concise

```
public class HelloWorld
{
        public static void main
(String[] args)        {

        System.out.println("Hello
world!");
        }
}
```

```
print("Hello world!")
```

Why should I love python?

Python is as readable as a pseudo code.
Readability is very important because very often code is read more than its written

```
fileHandle = open(“somefile.txt”, “r”)

for line in fileHandle:
    print(line)
```

Why should I love python?

Eliminates a lot of syntax overheads.

- No dreaded semi-colon at the end “;”
- No flower brackets “}” for blocks

```
var1 = 100

if var1>100:
    print(“Not a century” )
    print(var1)
else:
    print(“It’s a century” )
    print(var1)

print “Bye!”
```

Why should I love python?

Follows duck typing

- Lot less coding.
- Forces a clean understanding of code before using it.
- Can be dangerous too, depending on what kind of a programmer you are!

```
def f(someobj):
    someobj.quack()
```

```
if a == 10:
   f(b)

#suppose b cannot quack and a is
#very rarely 10. you will not see
#an error as long as a is never 10
```

Why should I love python?

Python comes with batteries included.

Python Standard Library

Simply put, Python has a large standard library. Some examples

- HTTP Protocols
- Database access
- IPC
- File system access

Python Package Index

- Python Package Index is a repository for python libraries
- Currently has over 89000 packages
- These can be installed through pip

Version of Python for this course

- We will be using version 3.x
- Python 3 and Python 2 are different at a many places.
  - Implies all the libraries have to be ported to Python 3
  - Due the compatibility issues Python 2 versions continue to find favour
  - In a couple of years time, we would see everything in Python 3

Chapter: Setup

Python setup or installation

Step 1
- Download from https://www.python.org/downloads/

Step 2
- Run the installer and click through

Step 3
- Run IDLE and the python shell should come up

Check the installation guide provided (You will not need this)

Chapter: Basic data types

PYTHON

The proverbial “hello world” program

```
>>> print(“Hello World!“)
Hello World!

>>> print('Hello World!‘)
Hello World!

>>> print('''Hello World!''‘)
Hello World!

>>> print("""Hello World!""“)
Hello World!
```

```
>>> print(“Hello \
World!“)
Hello World!

>>> print (‘’’Hello
World!''‘)
Hello
World
```

PYTHON

Variables

```
>>> a = 10
>>> b = 10.2
>>> c = True
>>> d = 'Hello‘
>>> print(a , b, c, d)
10 10.2 True Hello
```

```
>>> type(a)
<type 'int'>

>>> type(b)
<type 'float'>

>>> type(c)
<type 'bool'>

>>> type(d)
<type 'str'>
```

Python Errors

```
>>> a = 10/0

Traceback (most recent call last):
  File "<pyshell#67>", line 1, in
<module>
    a = 10/0
ZeroDivisionError: integer
division or modulo by zero
```

```
>>> hello

Traceback (most recent call last):
  File "<pyshell#65>", line 1, in
<module>
    hello
NameError: name 'hello' is not
defined
```

Python Integers and Floats

```
>>> a = 10 + 4
>>> a
14

>>> a = a*a
>>> a
169

>>> a = 2**10
>>> a
1024
```

```
>>> a= 1.0
>>> type(a)
<type 'float'>
```

Python type conversions

```
>>> a= 10
>>> b = True
>>> a + b
11
>>> a = 10.0
>>> b = True
>>> a + b
11.0
>>> a = True + True
>>> a
2
>>> a = 3
>>> b = 4
>>> a/b
0
```

```
>>> a = 3
>>> b=4.0
>>> a/b
0.75
>>> b/a
1.3333333333333333
>>> a = 5 + '5‘
Traceback (most recent call last):
  File "<pyshell#144>", line 1, in
<module>
    a = 5 + '5'
TypeError: unsupported operand
type(s) for +: 'int' and 'str'
```

PYTHON

Python type conversions

```
>>> a = 5 + int('5')
>>> a
10

>>> a = 5 + float(1)
>>> a
6.0

>>> a = 3
>>> b = 4
>>> a / float(b)
0.75
```

```
>>> a = 3
>>> b=4.0
>>> a/b
0.75
>>> b/a
1.3333333333333333
>>> a = 5 + '5‘
Traceback (most recent call last):
  File "<pyshell#144>", line 1, in
<module>
    a = 5 + '5'
TypeError: unsupported operand
type(s) for +: 'int' and 'str'
```

Strings

Our Hello World program actually dealt with a lot of strings
Consider strings to be a sequence of ‘char’ s

```
>>> a = "hello"
>>> a
'hello'
>>> a = "Hello" + " World"
>>> a
'Hello World'
>>> a += " Again"
>>> a
'Hello World Again'
```

```
# convert with str
>>> a = str(3) + 's'
>>> a
'3s'

# now try a = a * 5
>>> a = a * 5
>>> a
'3s3s3s3s3s'
```

PYTHON

Strings – access through index

Consider strings to be a sequence of ‘char’ s
Can we extract characters?

```
>>> a = "Hello Bangalore"
>>> a[1]
'e'
>>> a[20]
Traceback (most recent call last):
  File "<pyshell#202>", line 1, in
<module>
    a[20]
IndexError: string index out of
range
```

```
>>> a[-2]
'r'
>>> a[-1]
'e'
>>> a[0]
'H'
>>> len(a)
15

# try changing the char at index 0
a[0]="h"
```

Strings - Slicing

- Slicing a String with a start, end and step
  - To extract a substring
  - [start:end:step]
  - If a is a string then a[0:3] gives a substring
    - which contains a[0], a[1], a[2] characters
    - It is a lot forgiving in terms of index checks. Try with “out of bound” indices

Strings

```
>>> a='0123456789'
>>> a[0:9]
'012345678'
>>> a[0:10]
'0123456789'
>>> a[0:100]
'0123456789'
>>> a[-3:-1]
'78'
>>> a[-1:3]
''
```

```
>>> a[-1:3:1]
''
>>> a[-1:3:-1]
'987654'
>>> a[-1:-8:-1]
'9876543'
```

Strings

```
>>> a='0123456789'
>>> a[0:9]
'012345678'
>>> a[0:10]
'0123456789'
>>> a[0:100]
'0123456789'
>>> a[-3:-1]
'78'
>>> a[-1:3]
''
```

```
>>> a[-1:3:1]
''
>>> a[-1:3:-1]
'987654'
>>> a[-1:-8:-1]
'9876543'
```

Strings – summarizing slice movement

String	a	b	c	d	e	f	g	h	i	j
+ve index	0	1	2	3	4	5	6	7	8	9
-ve index	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> a[-1:3]
''
>>> a[-1:3:-1]
'jihgfe'
```

Strings - challenge

```
>>> a='0123456789’
# Challenge 1 : Print the reverse of the string using slice
# Challenge 2 : Print only the even number indices
# Duration : 3 minutes
```

Lab work

Lab 1

- Attempt Lab work questions #1 – #8 (can skip #2)
- Duration : Approx 12 minutes

PYTHON

Strings – lots of string functions

```
a.startwith('0')
a.endswith('9')
a.find('a')
# funny thing with "find". Returns -1 to say it did not find
a.count('0')
a.isalnum()
a = "the discovery of india"
a.title()
a.capitalize()
a.lower()
a.upper()
```

PYTHON

Strings – now check this

```
>>> a ="the discovery of india"
>>> a.split()
['the', 'discovery', 'of', 'india']
>>> a.split(" ")
['the', 'discovery', 'of', 'india']

# any guesses which movie has this star cast?

>>> b = 'Samuel Jackson, John Trovolta, Bruce Wills, Uma Thurman'
>>> b.split(',')
['Samuel Jackson', ' John Trovolta', ' Bruce Wills', ' Uma Thurman']
```

Chapter: Lists

Lists

- Data structure to hold a list of items
- These items can be of different data types too
- Can access items of the list using index
- The same slicing operations work well on lists too but at a list item level
  - [start:end:step]
  - [:end:step]
  - [::]
  - [::step]
  - [:end]
  - [:]
- List can contain lists too
- Can be as deeply nested as you want

Lists - creation

```
>>> a = []
>>> a = list()
>>> a
[]
>>> a = []
>>> a
[]
>>> a = list('cat')
>>> a
['c', 'a', 't']
>>> a= 'India,Japan,China,UK,USA'.split(',')
>>> a
['India', 'Japan', 'China', 'UK', 'USA']
```

PYTHON

Lists – accessing list element using index

```
>>> a = ['India', 'Japan', 'China', 'UK', 'USA']
>>> a[1]
'Japan'
>>> a[-1]
'USA'
>>> a[100]

Traceback (most recent call last):
  File "<pyshell#317>", line 1, in <module>
    a[100]
IndexError: list index out of range
```

Lists – slice

```
>>> a = ['India', 'Japan', 'China', 'UK', 'USA']
>>> a[1:3]
['Japan', 'China']
>>> a[-1:]
['USA']
>>> a[-1::-1]
['USA', 'UK', 'China', 'Japan', 'India']
```

Lists – Modifying Lists

```
>>> a
['India', 'Japan', 'China', 'UK',
'USA']
>>> a[3]='Burma'
>>> a
['India', 'Japan', 'China',
'Burma', 'USA']
>>> id(a)
59350280L
>>> a[3]='Russia'
>>> id(a)
59350280L
```

```
>>> b='hello'
>>> b.replace('h','d')
'dello'
>>> b
'hello'
>>> id(b)
59545200L
>>> b = b.replace('h','d')
>>> b
'dello'
>>> id(b)
56578616L
```

Lists – Modifying Lists

```
>>> a.append('UK')
>>> a
['India', 'Japan', 'China',
'Russia', 'USA', 'UK'] >>> b =
['Sri Lanka','Thailand','Nigeria']
>>> a + b
['India', 'Japan', 'China',
'Russia', 'USA', 'UK', 'Sri
Lanka', 'Thailand', 'Nigeria']
>>> a+=b
>>> a
['India', 'Japan', 'China',
'Russia', 'USA', 'UK', 'Sri
Lanka', 'Thailand', 'Nigeria']
```

```
>>> a = ['India', 'Japan',
'China', 'Russia', 'USA']
>>> a.extend(b)
>>> a
['India', 'Japan', 'China',
'Russia', 'USA', 'Sri Lanka',
'Thailand', 'Nigeria']
>>> a = ['India', 'Japan',
'China', 'Russia', 'USA']
>>> a.append(b)
>>> a
['India', 'Japan', 'China',
'Russia', 'USA', ['Sri Lanka',
'Thailand', 'Nigeria']]
```

PYTHON

Lists – Modifying Lists

```
>>> a = ['India', 'Japan',
'China', 'Russia', 'USA']
>>> a.insert(3,'Ukraine')
>>> a
['India', 'Japan', 'China',
'Ukraine', 'Russia', 'USA']

>>> a.insert(200, 'Bangkok')
>>> a
['India', 'Japan', 'China',
'Ukraine', 'Russia', 'USA',
'Bangkok`]
```

```
>>> a.insert(-1, 'Indonesia')
>>> a
['India', 'Japan', 'China',
'Ukraine', 'Russia', 'USA',
'Indonesia', 'Bangkok']

>>> a.insert(-200, 'New York')
>>> a
['New York', 'India', 'Japan',
'China', 'Ukraine', 'Russia',
'USA', 'Indonesia', 'Bangkok`]
```

Lists – Deleting

```
>>> a.remove('Bangkok')
>>> a
['New York', 'India', 'Japan', 'China', 'Ukraine', 'Russia', 'USA',
'Indonesia']

>>> del a[0]
>>> a
['India', 'Japan', 'China', 'Ukraine', 'Russia', 'USA', 'Indonesia']

>>> a.pop()
'Indonesia'
>>> a
['India', 'Japan', 'China', 'Ukraine', 'Russia', 'USA']
```

Lists – Test for a value

Pythonic way of coding

```
>>> a = ['India', 'Japan', 'China', 'UK', 'USA']
>>> 'India' in a
True
>>> country = 'India'
>>> country in a
True
```

A little digression

Pythonic way of coding

When a veteran Python developer (a “Pythonista”) calls portions of code not “Pythonic”, they usually mean that these lines of code do not follow the common guidelines and fail to express its intent in what is considered the best (hear: most readable) way.

```
#Pythonic way
>>> a = ['India', 'Japan',
'China', 'UK', 'USA']
>>> ‘Japan' in a
True
```

```
# non Pythonic way
>>> a.index(‘Japan')
1
# then compare the index if its
positive or if it gave an error
and then confirm whether ‘Japan’
exists or not
```

Lists – Copying

```
>>> a = ['India', 'Japan',
'China', 'UK', 'USA']
>>> b = a
>>> b
['India', 'Japan', 'China', 'UK',
'USA']
>>> a[1] = 'Nigeria'
>>> a
['India', 'Nigeria', 'China',
'UK', 'USA']
>>> b
['India', 'Nigeria', 'China',
'UK', 'USA']
```

```
>>> b = list(a)
>>> c = a[:]
```

Tuples

- Similar to lists
- Uses “(“ and “)” for being and end
- Tuples are Immutable. So they lack
  - append(), insert(), pop() etc

Tuples

```
>>> a_tuple= ()
>>> a_tuple = ('batman','spiderman','superman','ironman')
>>> a_tuple = 'batman','spiderman','superman','ironman'
>>> a_tuple
('batman', 'spiderman', 'superman', 'ironman')
>>> type(a_tuple)
<type 'tuple'>
```

Tuples - unpacking

```
>>> sh1, sh2, sh3, sh4 = a_tuple
>>> print sh1, sh2, sh3, sh4
batman spiderman superman ironman

>>> sh1, sh2 = a_tuple

Traceback (most recent call last):
  File "<pyshell#497>", line 1, in <module>
    sh1, sh2 = a_tuple
ValueError: too many values to unpack
```

Tuples – Slicing is same as in lists

```
>>> a_tuple[1:2]
('spiderman',)
>>> a_tuple[1:]
('spiderman', 'superman', 'ironman')
>>> a_tuple[1:100]
('spiderman', 'superman', 'ironman')
>>> a_tuple[1::2]
('spiderman', 'ironman')
```

Tuple Vs Lists

- Uses lesser space
- Immutable and hence cannot change by mistake
- Function arguments are passed as tuples

Dictionaries

- Uses key value pairs instead of index
- Similar to associative array (PHP), hash maps (Java) of other languages
- Mutable data structure => can change its values
- Uses “{“ and “}” to define its being and end

Dictionary

```
>>> a_d = {1:'January', 2:'February', 3:'March'}
>>> a_d
{1: 'January', 2: 'February', 3: 'March'}
>>> type(a_d)
<type 'dict'>
>>> a = [1,2,3,4,5,6,7]
>>> dict(a)

Traceback (most recent call last):
  File "<pyshell#525>", line 1, in <module>
    dict(a)
TypeError: cannot convert dictionary update sequence element #0 to a
sequence
>>> a = [[1,2],[3,4],[5,6]]
>>> dict(a)
{1: 2, 3: 4, 5: 6}
```

Dictionary – accessing elements

```
>>> a_d = {'name': 'Aditya', 'email' : 'sp.aditya@gmail.com'}
>>> len(a_d)
2
>>> a_d['name']
'Aditya'
>>> keystr = 'name'
>>> a_d[keystr]
'Aditya'
>>> a_d[keystr]="Aditya Prabhakara"
>>> a_d
{'name': 'Aditya Prabhakara', 'email': 'sp.aditya@gmail.com'}
>>> a_d['city'] = "Bangalore"
>>> a_d
{'city': 'Bangalore', 'name': 'Aditya Prabhakara', 'email':
'sp.aditya@gmail.com'}
```

Dictionary – combine dictionaries

```
>>> a_d = {'name': 'Aditya Prabhakara', 'email': 'sp.aditya@gmail.com'}
>>> update_d = {'name' : 'Aditya S P', 'city' : 'Bangalore'}
>>> a_d.update(update_d)
>>> a_d
{'city': 'Bangalore', 'name': 'Aditya S P', 'email':
'sp.aditya@gmail.com'}
```

Dictionary – Working with keys

```
>>> a_d
{'city': 'Bangalore', 'name': 'Aditya S P', 'email':
'sp.aditya@gmail.com'}
>>> 'city' in a_d
True
>>> a_d.keys()
['city', 'name', 'email']
>>> a_d.values()
['Bangalore', 'Aditya S P', 'sp.aditya@gmail.com']
```

Lab work

Lab 2

- Attempt Lab work questions #9 – #16
- Duration : Approx 10 minutes

Chapter: Conditionals

Conditional – if elif else

- The syntax might feel a bit strange
- Whitespaces matter – forced indentation
- Leads to indented formatted code
- I personally was not a huge fan of this in the beginning and then it grew on me and now it feels “so obvious”

Conditional – if elif else

```
>>> a = True
>>> type(a)
<type 'bool'>
>>> if a:
        print("Truthful")

Truthful
```

Conditional – if elif else

```
>>> a = 100
>>> if a<10 :
        print("Single Digit")
else:
        print("May be double digit")
```

```
May be double digit
```

Conditional – if elif else

```
>>> a = 100
>>> if a<10 :
        print("Single Digit")
elif a < 100:
        print("Double Digit")
else :
        print("2+ digits")

2+ digits
```

Conditionals

Description	Operator
Equality	==
Inequality	!=
less than	<
Less than or equal	<=
Greater than	>
Greater than or equal	>=
Membership	in

Conditional operators

```
>>> x = 5
>>> y = 10
>>> x < y
True
>>> x < 5 and y < 20
False
>>> x < 6 and y < 20
True
>>> x < 5 or y < 20
True
>>> x < 5 and not y < 6
False
>>> x < 6 and not y < 6
True
```

PYTHON

Conditional operators – Cool & readable

```
>>> x = 5
>>> y = 10
>>> 3 < x < 10
True
>>> 3 < x < y < 20
True
>>> if x > 3 and x < y and y < 20 :
        print("Truthful")


Truthful
```

Loops – while loop

```
>>> a = ['KA','TN','DL','AP','KL','PY']
>>> while count < len(a):
        print(a[count])
        count+=1

KA
TN
DL
AP
KL
PY
```

Loops – for loop

A better Pythonic way of previous while loop

```
>>> a = ['KA','TN','DL','AP','KL','PY']
>>> for itr in a:
        print(itr)


KA
TN
DL
AP
KL
PY
```

Loops – break and continue

```
>>> a =
['KA','TN','DL','AP','KL','PY']
>>> for itr in a:
        if itr == 'UP':
                break
else:
        print('did not find code‘)
```

```
did not find code
```

```
# not Pythonic. Sake of example
>>> found = False
>>> for itr in a:
        if itr == 'UP':
                found = True
>>> if not found:
        print("Did not find what I
was looking for“)
```

```
Did not find what I was looking
for
```

Lab work

Lab 2

- Attempt Lab work questions #17 – #18
- Duration : Approx 5 minutes

Chapter: Functions

PYTHON

Functions – defining and calling

```
>>> def sayhi():
        print("Hi")

>>> type(sayhi)
<type 'function'>
>>> sayhi()
Hi
```

```
>>> def sayhi():
        ''' this is a function
that says hi '''
        print "Hi"


>>> sayhi()
Hi
>>> sayhi.__doc__
' this is a function that says hi
'
```

Functions

```
>>> def is_even(num):
        if num%2 == 0:
                return True
        else:
                return False
>>> if is_even(2):
        print("This is even")
else:
        print("This is odd")
```

```
This is even
```

```
if is_even(3):
        print("This is even")
else:
        print("This is odd")
```

```
This is odd
```

Functions – Parameters (key word)

```
>>> def full_name(fname, lname):
        print(fname + " " + lname)


>>> full_name('Aditya', 'Prabhakara')
Aditya Prabhakara
>>> full_name (lname='Prabhakara', fname='Aditya')
Aditya Prabhakara
```

PYTHON

Functions – Parameters (default args)

```
>>> def full_name(fname, lname, title="Mr"):
        print(title + " " + fname + " " + lname)


>>> full_name (lname='Prabhakara', fname='Aditya')
Mr Aditya Prabhakara
>>> full_name(lname='Prabhakara')

Traceback (most recent call last):
  File "<pyshell#785>", line 1, in <module>
    full_name(lname='Prabhakara')
TypeError: full_name() takes at least 2 arguments (1 given)
```

Functions – any number of args -1

```
>>> def any_args(*args):
        print(args)


>>> any_args('a',1,2,'c')
('a', 1, 2, 'c')
>>> def any_args(*args):
        print(args)
        print(type(args))


>>> any_args('a',1,2,'c')
('a', 1, 2, 'c')
<type 'tuple'>
```

Functions – any number of args - 2

```
>>> def any_args(**kwargs):
        print(kwargs)

>>> any_args(what=2, where=3)
{'what': 2, 'where': 3}

>>> def any_args(*args, **kwargs):
        print(args, kwargs)


>>> any_args(1, 2, 3,a=4, b = 5, c = 6)
(1, 2, 3) {'a': 4, 'c': 6, 'b': 5}
```

PYTHON

Understand Namespace or scope

```
>>> a=10
>>> def finda():
        print(a)


>>> finda()
10
>>> a=30
>>> finda()
30
```

```
>>> def finda():
        a=100
        print(a)


>>> a=20
>>> finda()
100
```

PYTHON

Understand Namespace or scope

```
>>> def finda():
        global a
        a = 100
        print(a)


>>> a = 200
>>> finda()
100
>>> a
100
```

Lab work

Lab 3

- Attempt Lab work question #19, #20
- Duration : Approx 5 minutes

Functions

- We have only treated functions as just that.

Wait till we start treating them as first class citizens.

We can pass functions around as variables
We can make a function return a function
We can have nested functions
We can have anonymous functions
It gets exciting