

By ADITYA PRABHAKARA





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!

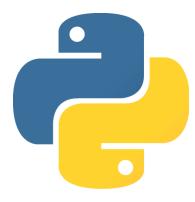
# Getting to know you

#### 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

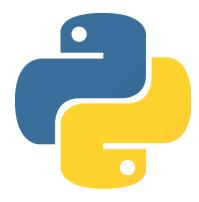
## Agenda

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



# **Course Objectives**

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





## 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



#### 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).







I pronounce tuple too-pull on Mon/Wed/Fri and tub-pull on Tue/Thu/Sat. On Sunday I don't talk about them. :) @avivby

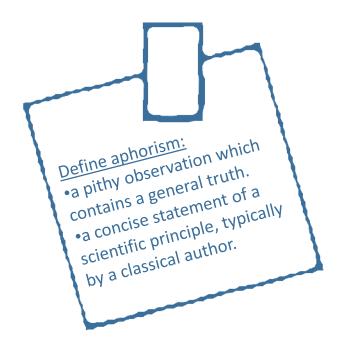
A funny Guy: Check his answer on asking how to pronounce "tuple" – a data structure in Python



A set of 20 aphorisms. My favs below

- ➤ 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





#### Python is very concise

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

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

```
print "Hello world!"
```



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
```

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!"
```



#### 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 equals 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
```

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 2.7.12
- The latest version is 3.7
- Python 3 and Python 2 are radically 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
- Incremental learning from 2.x to 3.x is not very difficult







#### Python setup or installation

• 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)





## 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
```

## 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
>>> a = 10.0
>>> b = True
>>> a + b
11.0
>>> a = True + True
>>> a
>>> a = 3
>>> b = 4
>>> a/b
```

```
>>> 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 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'
```



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
13s \
\# now try a = a * 5
>>> a = a * 5
>>> a
'3s3s3s3s3s'
```

## 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]
171
>>> 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]
1781
>>> a[-1:3]
7 7
```

```
>>> a[-1:3:1]
1 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]
1781
>>> a[-1:3]
7 7
```

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

## Strings – summarizing slice movement

String	а	b	С	d	е	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]
'jihge'
```



## 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

## 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()
```

## Strings – now check this

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



## **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']
```

#### 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)
59350280T
>>> a[3]='Russia'
>>> id(a)
59350280T
```

```
>>> b='hello'
>>> b.replace('h','d')
'dello'
>>> b
'hello'
>>> id(b)
59545200L
>>> b = b.replace('h','d')
>>> b
'dello'
>>> id(b)
56578616T
```

#### 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']]
```

#### 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'l
>>> 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





- 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"





# **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
```

#### 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
```

### **Lab work**

Lab 2

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



#### Functions – defining and calling

## **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)

### 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}
```



#### 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
```



#### Understand Namespace or scope

```
>>> def finda():
       global a
       a = 100
       print a
>>> a = 200
>>> finda()
100
>>> a
100
```



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



#### **Functions (Contd)**

- Functions are first class citizens
- Which basically means
  - They are treated like objects (as in Python everything is an object)
  - They can be passed around like other variables
  - > Assigned to other variables



### Function as variables

```
>>> def sayhi():
       print "Hi"
>>> sayhi()
Ηi
>>> type(sayhi)
<type 'function'>
>>> a = sayhi
>>> type(a)
<type 'function'>
>>> id(a)
59704264T
>>> id(sayhi)
59704264L
```

```
>>> a()
Ηi
>>> def someotherhi(func):
        func()
>>> someotherhi(a)
Ηi
>>>
```

# Inner functions

```
>>> def add(*args):
       def inneradd(*args):
               return sum(*args)
       print inneradd(*args)
>>> add([1,2])
>>> add([1,2,3,4])
10
>>> sum. doc
"sum(sequence[, start]) -> value\n\nReturn the sum of a sequence of
numbers (NOT strings) plus the value\nof parameter 'start' (which
defaults to 0). When the sequence is \nempty, return start."
```

### Closure – it gets crazy here!

```
>>> def bill(*args):
       def withtax(x):
               return sum(args)+x*sum(args)
       return withtax
>>> newbill = bill(10,15,20,50)
>>> newbill(.145)
108.775
>>> newbill(.20)
114.0
>>> newbill2 = bill(2)
>>> newbill2(.145)
2.29
>>> newbill2(.2)
2.4
```



### Decorators – it gets crazier

```
def deco(func):
       def new func(*args, **kwargs):
               print 'in mydeco'
               result=func(*args, **kwargs)
               return result
       return new func
def addthem(*args):
       return sum(args)
>>> decoratedadd = deco(addthem(3,4))
>>> decoratedadd(3,4)
>>> @deco
def addthem(*args):
       return sum(args)
```





```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for i in range(10):
       print i
```

#### Comprehension

- Comprehension is a compact way of creating a Python data structure
- Leads to more readable code
- List comprehension is a compact way of creating lists
- > A generic syntax of comprehension is as follows

```
[expression for item in iterable]
```

- What qualifies as iterable?
  - > Lists
  - Range
  - > Generators
  - ... and a lot more

### List comprehension

```
>>> a = [x for x in range(5)]
>>> a
[0, 1, 2, 3, 4]
>>> a = [x*x for x in range(5)]
>>> a
[0, 1, 4, 9, 16]
>>> b = 'abcdefghijklmnopqrstuvwxyz'
>>> a = [x*2 \text{ for } x \text{ in } b]
>>> def sqit(x):
        return x*x
>>> a = [sqit(x) for x in range(5)]
>>> a
[0, 1, 4, 9, 16]
```



# List comprehension – advanced

```
[expression for item in iterable if expression]
```

```
>>> a = [x for x in range(5) if x%2 == 0]
>>> a
[0, 2, 4]
>>> a = [x for x in range(5) if is_even(x)]
>>> a
[0, 2, 4]
```

#### Dict comprehension

{keyexpr:valueexpr for item in iterable if expression}

```
>>> a = {x:x*x for x in range(5)}
>>> a
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# comprehend and deliberate on what is happening below
>>> a = 'the quick brown fox jumps over the lazy dog'
>>> a={x:a.count(x) for x in a}
```

### Generator

- >A python sequence creation object
- > Are source of data for iterators
- > Range is generator which creates a sequence of integers
- > A generic syntax of comprehension is as follows
- ➤ Special key work yield

### Generator

```
>>> def my range(first=0, num=10, step=1):
       current range = first
       count = 0
       while count < 10:
               yield current range
               current range += step
               count +=1
>>> for i in my range():
       print i
```



Lab 5

- Attempt Lab work question #23, #24
- Duration : Approx 20 minutes





- >Anonymous functions
- > Expressed as a single statement
- > Use it instead of a normal tiny functions
- Can be used as closures

# **Lambda**

- >Import only what you like
- ➤ Some hate it. Some love it. But you can't ignore it ③

```
>>> def f(x):
        return x%2==0
>>> f(1)
False
>>> f(2)
True
>>> d = lambda x:x%2==0
>>> d(5)
False
>>> d(4)
True
```



> Takes a function and applies it to every item in the list

```
>>> raceinkm =[5,10,21,42]
>>> def kmtomi(x):
       return x*0.621
>>> map(kmtomi, raceinkm)
[3.105, 6.21, 13.041, 26.082]
>>> map(lambda x:x*0.621, raceinkm)
[3.105, 6.21, 13.041, 26.082]
>>>  salary =[1.4,0.8,2.6,5.8]
>>> bonus=[0.2,.10,.40,.01]
>>> map(lambda x,y:x*y, salary, bonus)
[0.279999999999997, 0.080000000000002, 1.04, 0.05799999999999999
>>> map(lambda x,y:x*y + x, salary, bonus)
[1.68, 0.88000000000001, 3.64, 5.858]
```



> Takes a boolean function and applies it to every item in the list

```
>>> agelist=[12,23,78,95,22,36,71,22,20]
>>> filter(lambda x:18<x<50, agelist)
[23, 22, 36, 22, 20]
>>> allnum =list(range(10))
>>> filter(lambda x:x%2==0, allnum)
[0, 2, 4, 6, 8]
```



- > Takes the first two numbers
- >Applies a reduction
- > Takes the next number

```
>>> a = list(range(10))
>>> reduce(lambda x,y:x+y, a)
45
```





- Similar to dictionary but with no values
- >=> unique, order does not matter

```
>>> a = set()
>>> a
set([])
>>> a = set('letters')
>>> a
set(['s', 'r', 'e', 'l', 't'])
```

```
>>> a = set()
>>> a
set([])
```

#### Sets – combination operations

```
>>> a = set('letters')
>>> a
set(['s', 'r', 'e', 'l', 't'])
>>> b = set('postman')
>>> h
set(['a', 'm', 'o', 'n', 'p', 's',
't'1)
# intersection
>>> a & b
set(['s', 't'])
# union
>>> a | b
set(['a', 'e', 'm', 'l', 'o', 'n',
'p', 's', 'r', 't'])
```

```
# in a but not in b
>>> a -b
set(['r', 'e', 'l'])
# in b but not in a
>>> b-a
set(['a', 'p', 'm', 'o', 'n'])
# exclusive - either in a or in b
but not in both
>>> a ^ b
set(['a', 'p', 'r', 'e', 'm', 'l',
'o', 'n'])
```







Before we can read or write to a file we need to obtain a handle to the file

```
filehandler = open(filename, mode)
```

- filehandler is the reference that open() call returned
- > filename is the filepath mentioned as a string
- > mode is a string which indicates the file type and what we want to do on it
  - First character of mode: r for read, w for write, x write iff file does not exist, a means append

Lastly we need to close the file handle after we are done with some read or write

```
filehandler.close()
```

### Writing to a file

```
>>> stringtowrite = '''I changed my password to \"incorrect\" so when
ever I
enter a wrong password it reminds me saying your password is
incorrect'''
>>> fh= open('trialfile.txt', 'wt')
>>> fh.write(stringtowrite)
>>> fh.close()
>>> fh= open('C:/users/aditya/desktop/trialfile.txt', 'wt')
>>> fh.write(stringtowrite)
>>> fh.close()
```

### Reading from a file



#### Reading from a file – chunked reading

```
>>> fh = open('C:/users/aditya/desktop/trialfile.txt','r')
>>> chunk=3
>>> while True:
       mytext=fh.read(chunk)
       if not mytext:
               break
       print mytext
```





Continuing from previous example

```
>>> for line in filehandle:
          print line
>>> filehandle.seek(0)
>>> for line in filehandle:
          print line
```



Lab 3

- Attempt Lab work question #22
- Duration : Approx 5 minutes



#### **PYTHON**



- > We have already encountered a lot of errors or exceptions
  - > ZeroDivisionError
  - ➤ NameError
  - > IndexError
  - ➤ ValueError
  - ➤ TypeError



### Handle Exception

```
>>> a=list('abcdefghij')
>>> try:
       a[11]
except:
       print "Position out of bound"
Position out of bound
>>> try:
       a[9]
except:
       print "Position out of bound"
```



#### Handle multiple exception

```
>>> try:
       a[11]
       a=a/0
except IndexError as err:
       print err
except Exception as err:
       print err
list index out of range
```

```
>>> try:
       a[7]
       a=10/0
except IndexError as err:
       print err
except Exception as err:
       print err
'h'
integer division or modulo by zero
```



#### **PYTHON**



➤ Modules are files containing Python definitions and statements (ex. mymodule.py)

```
import mymodule
```

> If my mymodule.py has a function by name myfunc. Then you can call that using

```
mymodule.myfunc()
```

Modules can call their own modules



Lab 5

- Attempt Lab work question #21
- Duration : Approx 15 minutes

#### **PYTHON**

## Class work

- Create two .py files
- ➤ Import one in the other
- ≻Run it

#### **PYTHON**



Import only what you like

```
from greeting import sayhi
```

> If my mymodule.py has a function by name myfunc. Then you can call that using

```
mymodule.myfunc()
```

Modules can call their own modules





### What is an object

- Which occupies space
- Which may have data
- Which may have behavior
- For e.g. In real world "Car" is an object which has a "name" (data) and which can drive (behavior)

#### **PYTHON**

#### What is an Class

- Objects of similar type will have similar attributes and similar behaviour
- > All objects of car have a name, can drive
- ➤Or in other words if we somehow create a type then we can use that as a template/blueprint to create objects from that
- >Such a template or blueprint is called as class
- ➤ For eg.
- >a=10;b=20;c=30 are all "objects" of the "class" int
- For eg.
- We are all "objects" of "class" called "human beings"

### Create a class in python

>A pretty useless car. No data No behaviour

```
>>> class Car():
    pass
>>> b = Car()
```

### Create an object in python

Oh a little better at least it has a name

```
>>> class Account():
    def __init__(self,holdername, acctype="Savings"):
        self.balance = 1000.00
        self.acctype=acctype
        self.holdername = holdername
>>> ac = Account('Aditya')
```

## Pretty print an object

```
>>> class Account():
       def init (self,holdername, acctype="Savings"):
              self.minbalance = 1000.00
              self.acctype=acctype
              self.holdername = holdername
       def repr (self):
              return '{}, {}'.format(self.holdername, self.acctype)
       def str (self):
              return 'Account Holder : {}, Acc Type :
{}'.format(self.holdername, self.acctype)
```



### Object of an account

- So now an Object of account has data (balance, holdername, acctype)
- So now an Object of account has behaviour ( print )
- >In other words we can now say that the object has encapsulated data and behaviour

# Encapsulation

### Add more behaviour

```
def credit(self, amount=0):
    self.balance += num
    return self.balance
def debit(self, amount=0):
    if(self.balance - amount < 0):
        # raise an exception
    else
    return self.balance -= amount</pre>
```

## Raising an exception

```
def debit(self, amount=0):
    self.balance += num
    return self.balance

def credit(self, amount=0):
    if(self.balance - amount < 0):
        # raise an exception
    else
        return self.balance -= amount</pre>
```



#### Living Being – Human Being inherits

```
>>> class HumanBeing(LivingBeing):
    pass
```



#### Living Being – Human Being inherits



### Getters and Setters

```
class Shape():
    def __init__(self,shapename):
        self.name = shapename
    def get_shapename(self):
        print "In getter"
        return self.name
    def set_shapename(self,shapename):
        print "In setter"
        self.name=shapename
```



## Getters and Setters - private names

```
class Shape(object):
    def __init__(self,shapename):
        self.__name = shapename
    def get_shapename(self):
        print "In getter"
        return self.__name
    def set_shapename(self,shapename):
        print "In setter"
        self.__name=shapename
    name = property(get_shapename, set_shapename)
```

## Class Methods

### Static Methods

```
>>> class Shape():
       counter = 0
       def init (self):
               print "In init"
               Shape.counter += 1
       @classmethod
       def objscreated(cls):
               print "Shape has ", Shape.counter, " Objects created!"
       @staticmethod
       def somestaticnoise():
               print "I am a shape!"
>>> Shape.somestaticnoise()
I am a shape!
```

### **e** Equality?

### **e** Equality?

```
class Student():
       def init (self, stud id, stud name):
               self.stud id = stud id
               self.stud name = stud name
       def eq (self, stud2):
               return self.stud id == stud2.stud id
>>> b = Student(1, 'John Doe')
>>> c = Student(1, 'John Doe')
>>> b == c
True
```



#### Methods for comparision

```
lt (self, other)
le (self, other)
eq (self, other)
ne (self, other)
gt (self, other)
ge (self, other)
```



### Methods for Math

```
__add__(self, other)
__sub__(self, other)
__mul__(self, other)
__mod__(self, other)
__pow__(self, other)
```

#### **PYTHON**



```
str__(self)
_repr__(self)
_len__(self)
```



### List Sorting

```
>>> sorted(student_objects, key=lambda student: student.age)
```

#### Classes - Challenge

```
Create a class to represent complex numbers (for eg. 5 + i6)
c = Complex(5,7)
d = Complex(3,7)
c + d should give 8 + i14
```

#### Object Composition

- One Object containing multiple objects
- > For eg. Car containing tyres, body, engine, seats
- > For eg. Pizaa containing cheese, capsicum, onion

### Super

> "B" inherits from "A". But why doesn't it contain 'a'?

```
>>> class A(object):
    def __init__(self):
        self.a = 'a'

>>> class B(A):
    def __init__(self):
        self.b = 'b'
```

```
>>> b = B()
>>> b.a
Traceback (most recent call last):
  File "<pyshell#758>", line 1, in
  <module>
   b.a
AttributeError: 'B' object has no
   attribute 'a'
```

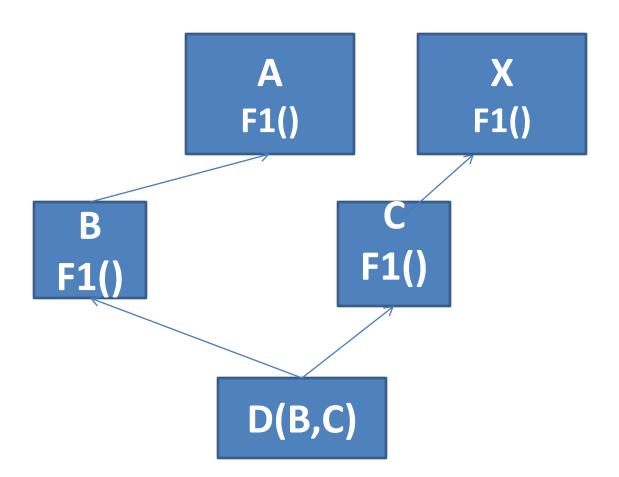
## Super - contd...

Use super to execute the init of A

```
>>> class A(object):
    def __init__(self):
        self.a = 'a'

>>> class B(A):
    def __init__(self):
        self.b = 'b'
        super(B,self).__init__()
```

```
>>> b = B()
>>> b.a
'a'
```



## Multiple Inheritance

> "B" inherits from "A". But why doesn't it contain 'a'?

```
>>> class A(object):
    def __init__(self):
        self.a = 'a'

>>> class B(A):
    def __init__(self):
        self.b = 'b'
        super(B,self).__init__()
```

```
>>> class C(A):
  def init (self):
       self.c='c'
       super(C, self). init ()
>>> class D(B, C):
  def init (self):
       self.d='d'
```



# Method Resolution Order (MRO)

- >To resolve a method , Python searches
  - > First in the current class
  - > This order is called as Linearization of Multi Derived Classes
  - The rules to find this order is called MRO



## Method Resolution Order (MRO)

```
>>>D. mro
(<class ' main .D'>, <class ' main .B'>, <class ' main .C'>,
  <class ' main .A'>, <type 'object'>)
#OR you can also execute
>>> D.mro()
[<class ' main .D'>, <class ' main .B'>, <class ' main .C'>,
  <class ' main .A'>, <type 'object'>]
```



## custom classes or objects Vs Module?

- > When you require multiple instances of the same behaviour but different attributes
- ➤ Inheritance leads to code reuse. Not possible with modules





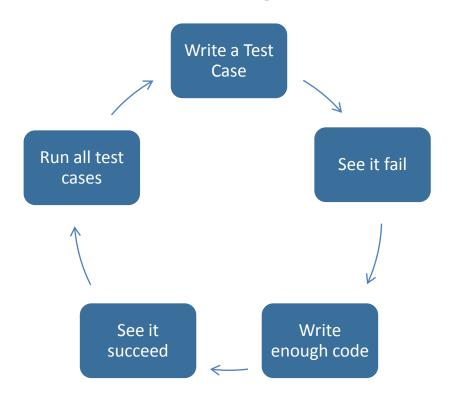
## **Test Driven Development**

**Test-driven development (TDD)** is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements.

Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.

-- Wikipedia

## **Test Driven Development - Process**





## Requires a good Testing Framework

- PyUnit -> Pythons Unit Testing framework
- Pythons version of Java's JUnit
- > Which in turn is a version of Small Talks testing framework
- PyUnit is consider an automatic choice for Python's unit testing

### **PYTHON**



- Object oriented in nature
- PyUnit is a part of Pythons standard library
- The name of the module is "unittest"
- >unittest is commonly called as PyUnit loosely based on its java counterpart JUnit

import unittest

### Creating our unit tests

- >Terminology:
  - > Test Case Single scenario that must be setup and checked for correctness
    - For eg. Create an object of Account and it should have a default acctype of "Savings"
  - ➤ Test Suite Collection of Test Cases
  - Assert : An expression to raise an AsssertError

```
>>> a = 1
>>> assert a==0, 'A was expected to be zero. How did it change?'
Traceback (most recent call last):
  File "<pyshell#796>", line 1, in <module>
    assert a==0, 'A was expected to be zero. How did it change?'
AssertionError: A was expected to be zero. How did it change?
```



## Creating our unit tests

> Asserts Available

assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	not isinstance(a, b)

### **PYTHON**



"Shape" Unit Test cases

```
C:\Users\aditya\Desktop\testing>python TestShape.py
...
Ran 2 tests in 0.000s
OK
```

### **PYTHON**



#### >Command line interface

```
python -m unittest <module1> <module2>
python -m unittest <modulename.testclassname>
python -m unittest <modulename.testclassname.testmethod>
```

## Other Testing Frameworks

- ▶PyMock
  - Testing framework built on Mock Objects
  - Created by Jeff Younker
  - Creates mock objects
  - Access to a simulated object / system in test env
  - ➤ For eg;
    - Mock Database
    - Mock Payment gateway
    - Mock server etc.



## Other Testing Frameworks

- ➤ Nose
  - > Discovers and runs unit test
  - Very much similar to unit test
  - > Is more of a runner
  - Can run unittest test case in Nose too
  - Good support for test case organization



# **Chapter: Database Access**

#### **PYTHON**



- ➤ C library
- Light weight disk-based database
- Support SQL like query languagle (A variant of standard SQL spec)
- ➤ Module sqlite3 is Compliant with PEP 249
- It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.



## Database access steps

Step 1

Create Connection object

Step 2

Create a cursor

Step 3

Execute db statements

Step 4

• Commit

Step 5

Close Connection



## Creating a Connection

```
>>> import sqlite3
>>> conn = sqlite3.connect('users.db')
>>> conn
<sqlite3.Connection object at 0x000000003943A28>
```



## Creating a Cursor

```
>>> cur = conn.cursor()
>>> cur
<sqlite3.Cursor object at 0x000000003A7A2D0>
```





### **Data Definition Language**

```
CREATE TABLE `User` (
`name` TEXT,
`salary` REAL
);
```

### Data Modification Language

INSERT INTO `User`(`name`,`salary`) VALUES ('Aditya',10000000.00);

### Query

Select \* from User



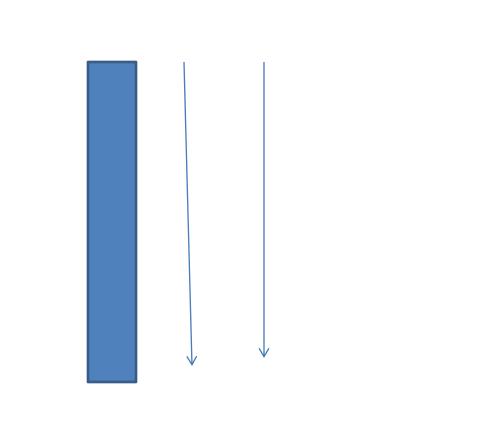
```
cur.execute ('''CREATE TABLE User (name TEXT, salary REAL)''')
cur.execute ('''INSERT INTO User values('Aditya',10000000.00)''')
conn.commit()
conn.close()
```

```
>>> cur.execute('select * from User')
<sqlite3.Cursor object at 0x000000003A7A2D0>
>>> cur.fetchall()
[(u'Aditya', 10000000.0)]
>>> cur.execute('select * from User where name=?', ('Aditya',))
<sqlite3.Cursor object at 0x000000003A7A500>
>>> cur.fetchall()
[(u'Aditya', 10000000.0)]
```

```
>>> users = [('user1',10000),('user2',100023)]
>>> cur.executemany('insert into users values(?,?)', users)
>>> cur.execute('select * from User')
<sqlite3.Cursor object at 0x000000003A7A500>
>>> cur.fetchall()
[(u'Aditya', 10000000.0), (u'user1', 10000.0), (u'user2', 100023.0)]
```

```
>>> for row in cur.execute('select * from user'):
       print row
(u'Aditya', 10000000.0)
(u'user1', 10000.0)
(u'user2', 100023.0)
```





#### **PYTHON**

## What are threads?

- ➤ Smallest schedulable unit in a computer
- ➤ Threads are contained in processes
- Threads share memory and state of the process
- >Two kinds of threads
- > a. Threads created by the kernel / operating system Kernel Threads
- b. User threads
- ➤ Implemented through Threading Module



## Advantages of threads

- ➤ Faster Execution
- Let us deliberate on the difference in "return"
- ➤Threads can share memory => threads from a same process have access to the same variables Both a cause for joy and a cause for worrying



## Starting threads

```
import threading
thread()
(refer program thread1.py)
```

## Passing arguments to threads

```
t = threading.Thread(target=somemethod, args=(i,))
(refer program thread1.py)
```

## Naming of threads

```
threading.currentThread().getName()

threading.Thread(name = 'myname' + str(i), target=somemethod,
args=(i,))

(refer program thread3.py)
```



## **Talk to each other**

```
threading.Event()
(refer to program thread4.py)
```



## Locking a resource

Execute thread2.py
Note that the output is garbled
Execute thread6.py
Note how lock solved the problem
This has made the code "Thread Safe"



### **PYTHON**



Loading ctypes
Interface with .dll files on windows and .so files on linux or unix machines

from ctypes import \*



## Ctypes - loading

Loading ctypes
Interface with .dll files on windows and .so files on linux or unix machines

```
from ctypes import *
library=cdll.LoadLibrary('libc.so.6')
library=CDD('msvcrt.dll')
library.printf('Hello World')
```



## Ctypes - loading

Loading ctypes
Interface with .dll files on windows and .so files on linux or unix machines

```
from ctypes import *
library=cdll.LoadLibrary('libc.so.6')
library=CDD('msvcrt.dll')
```



## Ctypes – calling a function

May not work from interpreter. Run it as a module

```
libc.printf("Hello World")
print libc.time(None)
```



### Ctypes – datatypes to interface with C

ctypes type	C type	Python type
c_char	char	1-character string
c_wchar	wchar_t	1-character unicode string
c_byte	char	int/long
c_ubyte	unsigned char	int/long
c_short	short	int/long
c_ushort	unsigned short	int/long
c_int	int	int/long
c_uint	unsigned int	int/long
c_long	long	int/long
c_ulong	unsigned long	int/long
c_longlong	int64 or long long	int/long
c_ulonglong	unsignedint64 or unsigned long long	int/long
c_float	float	float
c_double	double	float
c_char_p	char * (NUL terminated)	string or None
c_wchar_p	wchar_t * (NUL terminated)	unicode or None
c_void_p	void *	int/long or None



## Ctypes – convert to datatype of C

Only Strings and int automatically convert For all others we have to explicitly convert

```
libc.printf("Hello World")
libc.printf("%d", 42)
libc.printf("%f", 20.42)
```



### Ctypes – calling our own functions

### Calling our own functions

- 1. Create a linked library
- 2. Get a handle to the function
- 3. Set argtypes optional
- 4. Set returntypes = optional

```
qcc -o libtest.so -shared -fPIC libtest.c
```



### **PYTHON**



Python > 2.7.9 comes with pip but we need to update pip pip Python Package Manager
PyPi – Python package index – Most of the packages are found here

```
python -m pip install -U pip setuptools

Install a package using

pip install "projectname"

For eg.
pip install "openpyxl"
pip install "poster"
```



### **PYTHON**



```
# from within the scripts directory of Python
pip install openpyxl
```

# now we can import openpyxl
import openpyxl



## **Excel files - Reading**

```
#Step 1 Open the workbook
wb = openpyxl.load workbook(<path of xls>)
#Step 2 Get a handle to the Sheet
wb.get sheet names
sheet = wb.get sheet by name('Sheet1')
#use generators
sheet.rows
sheet.columns
for i in sheet.rows:
       for cell in i:
               print cell.value
```



## **Excel files - Writing**

```
#Step 1 Open the workbook
wb = openpyxl.Workbook()
#Step 2 Get a handle to the Sheet
wb.get sheet names
sheet = wb.get sheet by name('Sheet')
# Can create sheets
wb.create sheet('Sheet2')
#write values
wb.get sheet by name('Sheet')['A1'] = "First Created Value"
#Save
wb.save(<filename>)
```

#### **PYTHON**



Introduced in 2.6 and a part of the standard lib => no third party lib required loads() dumps()

```
#read json data
>>> fh = open('jsondata.txt','r')
>>> data = json.loads(fh.read())
>>> fh.close()

#write json data
>>> fh = open('jsondata.txt','w')
>>> fh.write(json.dumps(data))
>>> fh.close()
```



## requests - package

```
pip install requests
>>> import requests
>>> r = requests.get('http://www.omdbapi.com/?s=batman')
>>> r.text
>>> r = requests.post('http://httpbin.org/post', data={'key':'value'})
>>> files dict = { 'file': open('trial.xlsx', 'rb') }
>>> r = requests.post('http://localhost/upload.php', files=files dict)
>>> r.text
```





## Regular Expression

Text matching patterns

Analogous to wild cards like \*

Has a formal syntax with almost global acceptance

Almost all programming languages have regular expression support

Short hand – regex or regexp

## re module of Python

regexp support provided through the "re" module

## Return value of re.search

In the previous example what object did re.search return? — match object match object is a storehouse of information

```
>>> print match.re.pattern
this
>>> print match.string
Is this found here
>>> print match.start()
3
>>> print match.end()
7
```



## Pre-compiling patterns (regexp object)

Compiled patterns lead to faster searches
Repeated patterns can be cached. In the previous case caching is limited

```
>>> reo = re.compile(pattern)
>>> m = reo.search(sometext)
>>> m.start(), m.end()
(3, 7)
```

## Multiple matches

findall instead of search

```
>>> pattern = 'xy'
>>> string = 'xxyyxyxyxyxy'
>>> re.findall(pattern, string)
['xy', 'xy', 'xy', 'xy', 'xy']
>>> for m in re.finditer(pattern, string):
       print "found at", m.start()
found at 1
found at 4
found at 6
found at 8
found at 10
```

## More complex patterns

```
Pattern syntax = metacharacters + literal text
Repetition
Complete list of metacharacters
. ^ $ * + ? { } [ ] \ | ( )
Some special sequences
\d \D \s \S \w \W
```



## Literal repetition

```
a followed by zero or more b
'ab*'
a followed by one or more b
'ab+'
a followed by zero or one b
'ab?'
a followed by three b
'ab{3}'
a followed by two to three b
'ab{2,3}'
```

## Literal repetition

```
>>> re.findall('xy*', string)
['x', 'xyy', 'xy', 'xy', 'xy', 'xy']
>>> re.findall('xy+', string)
['xyy', 'xy', 'xy', 'xy', 'xy']
>>> re.findall('xy?', string)
['x', 'xy', 'xy', 'xy', 'xy', 'xy']
>>> re.findall('xy{3}', string)
>>> re.findall('xy{1,2}', string)
['xyy', 'xy', 'xy', 'xy', 'xy']
```

## Character set

```
either a or b
'[ab]'

followed by one or more a or b
'a[ab]+'

a followed by one or more a or b, not greedy
a 'a[ab]+?'
```

## Character Set

```
>>> re.findall('[xy]', string)
['x', 'x', 'y', 'y', 'x', 'y', 'x', 'y', 'x', 'y', 'x', 'y']
>>> re.findall('x[xy]+', string)
['xxyyxyxyxyxy']
>>> re.findall('x[xy]+?', string)
['xx', 'xy', 'xy', 'xy', 'xy']
```

# Positioning

```
start of string, or line
end of string, or line
$
start of string
\A
end of string
\backslash Z
```

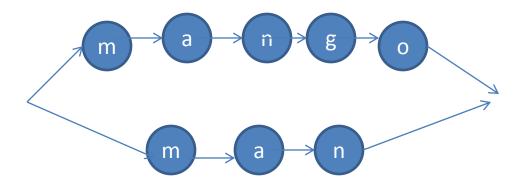
## **Examples 1**

```
>>> re.findall('^[0-9]', '0abs')
['0']
>>> re.findall('^[0-9]+', '0abs')
['0']
>>> re.findall('^[0-9]+', 'what')
[]
>>> re.findall('[0-9]$', 'abs0')
['0']
>>> re.findall('[0-9]$', 'abs')
```

## Grouping

```
'a' followed by literal 'ab'
'a(ab)'
'a' followed by 0-n 'a' and 0-n 'b'
'a(a*b*)'
'a' followed by 0-n 'ab'
'a(ab)*'
'a' followed by 1-n 'ab'
'a(ab)+'
```

### man eats mango





## **Common regex**

```
# username
^[a-z0-9_-]{3,16}$

#color hex code
^#?([a-f0-9]{6}|[a-f0-9]{3})$

#email
^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$
```

## **Examples 1**

```
>>> re.findall('^[0-9]', '0abs')
['0']
>>> re.findall('^[0-9]+', '0abs')
['0']
>>> re.findall('^[0-9]+', 'what')
[]
>>> re.findall('[0-9]$', 'abs0')
['0']
>>> re.findall('[0-9]$', 'abs')
```





- > PEP -8 https://www.python.org/dev/peps/pep-0008/
- ➤ Written by Rossum

#Naming Style – there is much more to pep-0008

joined\_lower for functions, methods, attributes
joined\_lower or ALL\_CAPS for constants
StudlyCaps for classes
camelCase only to conform to pre-existing conventions



Compound statements – decreases readablity

GOOD BAD

a=10 b=10 c=fun(1)

a=10;b=10;c=fun(1)

- > Strings
- The algorithm of join does only a single pass through the list to arrive at allcountries
- Bad memory usage with loops as at every step an object is discarded

GOOD BAD

```
>>>countries =['IN','KA','RU','BN','PK]
>>>allcountries = " ".join(countries)
```

```
>>>allcountries = "
>>> for i in countries:
allcountries +=I
```



>Testing for truth values

### **GOOD**

>>> if a:

...do something here

#### **BAD**

>>>if a == 'True':

... do something here

➤ Use list comprehension and sum

### **GOOD**

>>> total = sum([num \* num for num in range(1,
101)])

#### **BAD**

```
>>> total = 0
>>>for num in range(1, 101):
total += num * num
```





➤ Wild card import

GOOD

reference names through their module (fully qualified identifiers

import a long module using a shorter name (alias; recommended),

or explicitly import just the names you need.

**BAD** 

>>>from module import \*



- Many other languages
- int a = 10 means it creates space for int called "a" and then stores value 10 in it
- ►In Python
- It creates an object with value of 10 and then assigns a name called "a" to it



➤ Use dictionaries get

GOOD BAD

>>>somedict.get('what','Not Available')

>>> somedict['what']



## Pythonic coding – Many more at

http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html

## **Summary**

- Basics Python programming
  - int, str, bool, list, set, tuple, dict, functions, conditionals, loops, file handling, error handling, modules and packages
- Advanced Python
  - Comprehensions, filter, map, reduce, closures, lambda, higher order functions, generators
- Object Oriented Programming in Python
  - > Data encapsulation, inheritance, polymorphism, private members, MRO, super,
- Unit Testing in Python
  - PyUnit demo , TDD using PyUnit, Mock, Nose

