

# Introduction to Python



# PYTHON...

- ▶ Python is Beginner's Language
- ▶ Easy-to-learn
- ▶ Interactive
- ▶ Handles File Operations

# PYTHON...

- ▶ Python provides two modes:
  - Interactive Mode
  - Script Mode
- ▶ Python Identifiers:
  - A to Z or a to z
  - (\_) followed by 0 or more letters
  - Digits (0–9)

# PYTHON...

## ▶ Lines and Indentation:

- No braces to indicate blocks of code
- Blocks of code are denoted by line indentation (rigidly enforced)
- Number of spaces in the indentation is variable if True:

```
    print "True"  
else:  
    print "False"  
# No error
```

# PYTHON...

- Indentation within a block

```
if True:
```

```
    print "Answer"
```

```
    print "True"
```

```
else:
```

```
    print "Answer"
```

```
    print "False"
```

```
# Indentation Error
```

# Variables

- ▶ Integer (int)

- `a = 5`
  - `b = 10`
  - `print a * b # returns 50`

- ▶ Float (float)

- `c = 1.5`
  - `d = 2.0`
  - `print c * d # returns 3.0`
  - `print a * c # returns 7.5`

# Variables

- ▶ String (str)
  - fName = ‘John’
  - lName = “Payne”
  - print fName + ‘ ‘ + lName
  - print fName\*2

# Variables

## ▶ String (str)

- fName = ‘John’
- lName = “Payne”
- print fName + ‘ ‘ + lName # returns ‘John Payne’
- print fName\*2 # returns ‘JohnJohn’

# Variables

## ▶ String (str)

- fName = ‘John’
- lName = “Payne”
- print fName + ‘ ‘ + lName # returns ‘John Payne’
- print fName\*2 # returns ‘JohnJohn’
- a = ‘234’
- b = 234
- print a\*2
- print b\*2
- print a+b

# Variables

- ▶ String (str)
  - fName = ‘John’
  - lName = “Payne”
  - print fName + ‘ ‘ + lName # returns ‘John Payne’
  - print fName\*2 # returns ‘JohnJohn’
  - a = ‘234’
  - b = 234
  - print a\*2 # returns ‘234234’
  - print b\*2 # returns 468
  - print a+b # Gives error

# Operators

## ▶ Assignment Operators

- $a = 8$
- $b = 5$
- $c = 1$
- $c = a \quad \# c = 8$
- $c += a \quad (c = c + a) \quad \# c = 9$
- $c -= a \quad (c = c - a) \quad \# c = -7$

# Operators

## ► Comparison Operators :

- $a = 10$
- $b = 5$
- $a == b$  ?
- $a > b$  ?

# Operators

## ► Comparison Operators :

- $a = 10$
- $b = 5$
- $a == b$  # returns False
- $a > b$  # returns True

# Operators

## ▶ Comparison Operators :

- $a = 10$
- $b = 5$
- $a == b$  # returns False
- $a > b$  # returns True

## ▶ Logical Operators :

- $(a == b)$  and  $(a > b)$  ?
- $(a == b)$  or  $(a > b)$  ?

# Operators

## ► Comparison Operators :

- $a = 10$
- $b = 5$
- $a == b$  # returns False
- $a > b$  # returns True

## ► Logical Operators :

- $(a == b)$  and  $(a > b)$  # returns False
- $(a == b)$  or  $(a > b)$  # returns True

# Lists

- ▶ `a = [2 , 3 , 'John' , 4]`
- ▶ `print a[0]` # returns 2
- ▶ `print a[-1]` # returns 4

## SLICING :

- ▶ `print a[1:3]` # returns [3 , 'John']
- ▶ `print a[:3]` # returns [2 , 3 , 'John']
- ▶ `print a[:]` # returns [2 , 3 , 'John' ,4]
- ▶ `b = a[1:3]` # b = [3 , 'John']

# Lists

- ▶ `a = [ 2 , 4 , 'John' , 3 ]`
- ▶ `print len(a)` ?
- ▶ `a.append(2)` ?
- ▶ `b = [ 1, 2 ]`
- ▶ `a.append(b)` ?
- ▶ `a.reverse()` ?
- ▶ `a.count(2)` ?

# Lists

- ▶ `a = [ 2 , 4 , 'John' , 3 ]`
- ▶ `print len(a)` # returns 4
- ▶ `a.append(2)` # a = [2 , 4 , 'John' , 3 , 2]
- ▶ `b = [ 1, 2 ]`
- ▶ `a.append(b)` # a = [2,4, 'John',3,2,[1,2]]
- ▶ `a.reverse()` # a = [[1,2],2,3, 'John',4,2]
- ▶ `a.count(2)` # returns 2

# Lists

- ▶ `a = [7 , 2 , 2 , 1 , 4]`      `b = [ 'a' , 4 ]`
- ▶ `a.sort()`      ?
- ▶ `a.remove(2)`      ?
- ▶ `print 2 in a`      ?
- ▶ `print ['a']*4`      ?
- ▶ `print a + b`      ?

# Lists

- ▶ `a = [7 , 2 , 2 , 1 , 4]`                    `b = [ 'a' , 4 ]`
- ▶ `a.sort()`                                    `# a = [ 1 , 2 , 2 , 4 , 7 ]`
- ▶ `a.remove(2)`                                `# a = [ 1 , 2 , 4 , 7 ]`
- ▶ `print 2 in a`                                `# returns True`
- ▶ `print ['a']*4`                                `# returns ['a' , 'a' , 'a' , 'a']`
- ▶ `print a + b`                                `# returns [1 ,2 ,4 ,7 , 'a' ,4]`

# Dictionary

- ▶ Dictionaries consist of pairs (called items) of keys and their corresponding values.
- ▶ `a = {}` # empty dict
- ▶ `b = {'Name': 'John', 'Age': 7, 'Class': 'First'};`
- ▶ `print b['Age']` # returns 7
- ▶ `print a.keys()` # returns ['Name' , 'Age' , 'Class']
- ▶ `print a.values()` # returns ['John', 7 , 'First']
- ▶ `b['Age'] = 8`
- ▶ `Print b['Age']` # returns 8

# Dictionary

- ▶ `b = {'Name': 'John', 'Age': 8, 'Class': 'First'}`
- ▶ `print b.has_key('Age')` ?
- ▶ `print len(b)` ?
- ▶ `del dict['Class']` ?
- ▶ `b.clear()` ?

# Dictionary

- ▶ `b = {'Name': 'John', 'Age': 8, 'Class': 'First'}`
- ▶ `print b.has_key('Age')` # returns True
- ▶ `print len(b)` # returns 3
- ▶ `del dict['Class']` # removes 'Class'  
key
- ▶ `b.clear()` # b = {}

# If..else..

- ▶ if expression:  
    statement(s)
- else:  
        statement(s)
- ▶ a = 100
- ▶ if a >= 50 :  
        print 'a is greater than equal to 50'  
    else :  
        print 'a is lesser than 50'
- ▶ # returns 'a is greater than equal to 50'

# If..else..

- ▶ `a = 10`
- ▶ `if a > 20 :`
  - `print 'greater than 20'`
- `elif a>40 :`
  - `print 'greater than 40'`
- `else :`
  - `if a> 60 :`
    - `print 'greater than 60'`
  - `else :`
    - `print 'less than 20'`

# Loops

- ▶ while expression:  
    statement(s)
- ▶ count =0  
    while (count < 5) :  
        print count ,  
        count += 1
- ▶ Result : 0 1 2 3 4

# Loops

- ▶ `for iterator in sequence:`  
          statements(s)
- ▶ `for x in range(0,5) :`  
          `print x ,`
- ▶ Result : 0 1 2 3 4
- ▶ `for x in range(0,10,2) :`  
          `print x ,`
- ▶ Result : 0 2 4 6 8

# Iterating..

## ▶ Over Lists :

- `list1 = ['a' , 'c' , 'b' , 'n']`
- `for var1 in list1 :`

Print var1 ,

- Result : a b c d

## ▶ Over Dict :

- `dict1 = {1 :'a' , 2:'b' , 3:'c'}`
- `for var1 in dict1 :` #iterates over keys  
    `print str(var1) + ':' + dict1[var1] ,`
- Result : 1:a 2:b 3:c

# Split

- ▶ 

```
a = 'My name is John'  
aList = a.split()  
print aList
```
- ▶ 

```
# returns ['My' , 'name' , 'is' , 'John']
```
- ▶ 

```
for word in aList :  
    print len(word)
```
- ▶ 

```
Result : 2 4 2 4
```

# Split

- ▶ 

```
a = 'My name_is_John'  
aList = a.split('_')  
print aList
```
- ▶ # returns ['My name' , 'is' , 'John']
- ▶ 

```
for word in aList :  
    print len(word)
```
- ▶ Result : 7 2 4

# Strip

- ▶ a = ' John '  
    stripA = a.strip()  
    print stripA
- ▶ Result : John
- ▶ a = '\_\_John\_Payne\_\_'  
    stripA = a.strip('\_')  
    print stripA
- ▶ Result : John\_Payne

# Modules

- ▶ File : myprint.py

```
def print_func( person ):  
    print "Hello –" , person  
    return
```

- ▶ File : mycode.py

```
import myprint  
myprint.print_func('John')
```

- ▶ Run : python mycode.py

- ▶ Result : Hello – John

# Modules

- ▶ Common modules
- ▶ import os
  - os.system(command) # runs the command
  - Ex : os.system('mkdir test')  
Creates a directory “test”
  - os.path.isdir(name) # returns True if Directory
  - os.path.isfile(name) # returns True if File

# Modules

- ▶ import sys
- ▶ sys.argv : List with sentence arguments
- ▶ File : test.py
  - import sys
  - print sys.argv
- ▶ Run : python test.py John Mary
- ▶ Result : ['test.py' , 'John' , 'Mary']

# File I/O

- ▶ File Descriptor : Item that handles the I/O
- ▶ aFD = open('test.txt' , 'r')
  - for sentence in aFD :
    - print sentence
- ▶ # Reads the file linewise and prints the sentence
- ▶ text = aFD.read()
- ▶ # Reads the whole text at once in variable “text”.
- ▶ aFD.close()
- ▶ # closes the descriptor

# File I/O (hindi)

- ▶ import codecs  
aFD = codecs.open('hindi.txt' ,  
'r',encoding='utf-8')  
text = aFD.read()  
print text
- ▶ Difference
  - Use “import codecs”
  - Use “codecs.open”
  - Specify argument “encoding='utf-8'”

# Short Assignment

- ▶ Take an ssf file as input and extract the sentences from the file.
- ▶ Take a directory as an argument to your python file.
- ▶ Run :`python yourcode.py ssfDirectory`
- ▶ Output : For each ssf file create a file with raw sentences extracted in it.