BCB Python Workshop
03 23 2018

Urminder Singh

Chapter 0

Prologue

What is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Ref: https://www.python.org/doc/essays/blurb/

What is Python?

Python is a interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Brief history of Python

- Guido van Rossum started developing his own language in 1989 Christmas holidays.
- Named it Python after "Monty Python's Flying Circus"

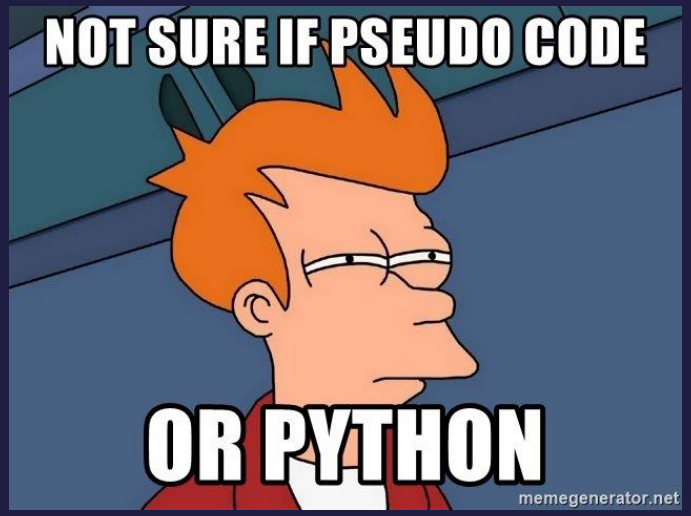
  Q. To learn Python, do I have to like "Monty Python's Flying Circus"

  A. No, but it helps. :) [Ref: https://docs.python.org]
- Python was first used in the Amoeba project.

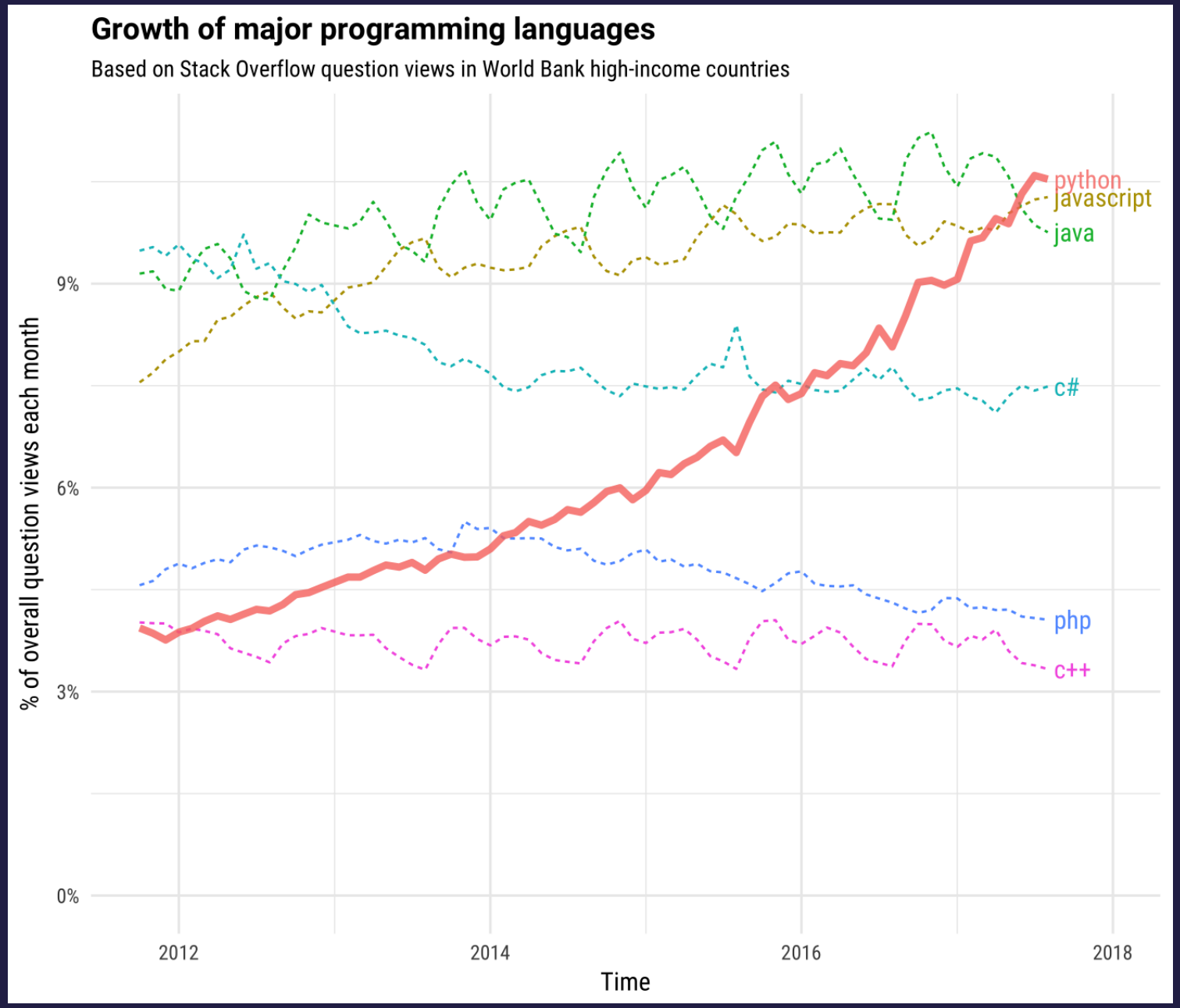
  "In February 1991, after just over a year of development, I decided to post to USENET. The rest is in the Misc/HISTORY file."

Python logo, 1990s–2005

Why Python



- Python is great for beginners
  - Python is high level language and easy to learn!
- Python is versatile
  - Applications include data analysis, web development, game development
- Python has great modules and libraries
  - PyPI "Python Package Index is a repository of software for the Python programming language. There are currently **132368** packages here."
- Python is supported by many platforms Linux/Windows/Mac
- Python is growing rapidly. It is one of the 3 official languages at Google.



Ref: http://news.codecademy.com/why-learn-python/

Few awesome examples of Python work

Dropbox

YouTube

BitTorrent™

Quora

scikits learn

machine learning in Python

matplotlib

SQLAlchemy

pandas

$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$

What is Python, seriously?

- Python is free and open source high level, interpreted object oriented programming language
  - Free and open source: costs $0 and you can see the python source code
  - High level: closer to humans than to machine, easy to write and read
  - Interpreted: executes directly and freely without compiling
  - Object oriented: programming paradigm that models data into classes and objects
  - Programming language: language used to interact with computers

Getting Started

- Downloading Python is free
  - Python comes preinstalled in most linux distros
  - https://www.python.org/downloads/

- Python 2 vs Python 3
  - Python 2.x is legacy, Python 3.x is the present and future of the language
  - Python 3.x is not backwards compatible with Python 2.x
  - As of now, Python 2.x has more library support and is used more
  - For a beginner, Python 2.x vs Python 3.x shouldn't matter

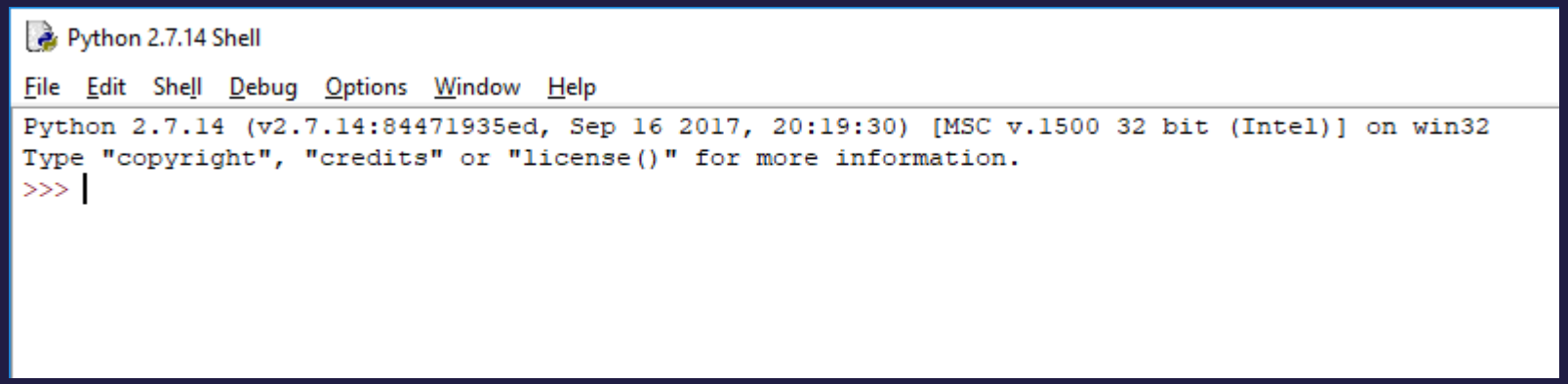
Setting up Python 2.7.x

- Open "Software Center"
  - Search for "Python" and install Python 2.7.x
  - Go to Start → IDLE(Python GUI)

- If working on personal laptop/computer (Windows)
  - Download and install python for windows: https://www.python.org/downloads/
  - Go to Start → IDLE(Python GUI)

- For linux and mac users
  - Open Terminal and type "python"

Congratulations! you're ready to code in Python



Chapter I

First Python Program

Printing text

- Windows users
  - In Python Shell go to “File” → “New File”
- Linux and Mac users
  - Use any text editor (gedit, TextEdit, Vim, nano etc.) and create a new file
- In the new file write

```
print “OK Computer”
```

- Save file as “myfirst.py”
  - Windows users: Go to “Run” → “Run Module” or press F5 key.
  - Linux and Mac: On terminal type python myfirst.py and enter

Comments in Python

- A single line comment in python starts with “#” character
- Multiline comments start and ends with “” or “”””.
- Now add comments to “myfirst.py” before the print statement

```
# This is a single line comment
''' This
    is a
    multiline
    comment
'''
```

- Did anything change after executing the code with comments?

Few more “print” examples python

- “print” always prints in newline, use “print something” followed by a comma to print next print statement on same line

```
print “OK Computer”,
print “OKNOTOK”
```

OK Computer OKNOTOK

```
#Second Option
print “OK Computer”, “OKNOTOK”
```

Strings

- "str" class in python defines string objects
  - Strings are enclosed by either double or single quotes
    - Multiline string can be defined using triple quotes

```
stringA="A string in double quotes"
stringB='A string in single quotes'
stringC='''A multiline
           String in triple quotes'''
```

  - Characters in a string can be accessed using [ ] operator

```
print stringA[0]     A
```

  - Strings are immutable i.e. can't be changed once created

```
stringA[0]='C'
```

  - Note: "stringA", "stringB" and "stringC" are known as variables

String slices

- Slice syntax is easy way to refer to substrings in a string

B	l	a	c	k	s	t	a	r
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

stringD="Blackstar"

print stringD[2:5] "ack"

print stringD[4:] "kstar"

print stringD[:-1] "Blacksta"

print stringD[-4:-1] "sta"

String methods

- "str" class defines many handy methods for string operations
  - For complete list of methods https://docs.python.org/2/library/stdtypes.html#string-methods

Method	Description
str.capitalize()	Return a copy of the string with its first character capitalized and the rest lowercased.
str.count(*sub*[, *start*[, *end*]])	Return the number of non-overlapping occurrences of substring *sub* in the range [*start*, *end*]. Optional arguments *start* and *end* are interpreted as in slice notation.
str.find(*sub*[, *start*[, *end*]])	Return the lowest index in the string where substring *sub* is found within the slice s[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation. Return -1 if *sub* is not found.
str.isdigit()	Return true if all characters in the string are digits and there is at least one character, false otherwise.
str.replace(*old*, *new*[, *count*])	Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

Built-in functions on strings

- Calculate length of a string using "len()"
  - len() is a built-in function in python

```
print len(stringA)
print len("OK Computer")
```

- Concatenate strings using "+" operator

```
print "OK" + str.capitalize("computer")
#same as below
print "OK" + "computer".capitalize()
```

- Can't concatenate string and int type

```
print "Length of stringA"+str(len(stringA))
```

- Note str(len(stringA)) converts len(stringA) and int to string so that it can be concatenated with another string. This is known as typecasting.

Few examples on string operations

- Open the file “strings.py” under the examples directory
- Run the “strings.py” file and try to understand how if-else statements work.
- Try to replace all the ‘t’s with ‘x’s and print your new string

Mutable vs immutable

- Immutable: an object once defined can't be changed e.g. stings

```
s='ABCD'
print s[1]          #prints B
s[1]='E'            #gives error
s='GHI'
print s             #prints GHI, string is mutated? NO.
```

- Mutable: an object once defined can be mutated e.g. lists.

Variables in Python

- Variables store information and are accessed/manipulated by the program. Variable names point to the memory locations where information is stored.
- Different "types" of variables store different types of information e.g. str stores strings, int store integers, float stores real numbers etc.
  - In Python variable types are determined by the value referenced by that variable

```
var="A string"
print type(var)  <type 'str'>
var=2
print type(var)  <type 'int'>
```

Python variable names

- Can contain letters, numbers and underscores. Can't start with a number.
- Can't be a reserved Python keyword
  - Reserved keywords have special meaning in python e.g. len is a function to get an object's length
  - More reserved keywords:

  and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield

Basic arithmetic operations

- Let a=10 b=15 c=3.0

Operator	Description	Example	Output
+ Addition	Adds values on either side of the operator.	a+ b b + c	25 18.0
- Subtraction	Subtracts right hand operand from left hand operand.	a - b b – c	-5 12.0
* Multiplication	Multiplies values on either side of the operator	a * b b * c	150 45.0
/ Division	Divides left hand operand by right hand operand	b / a b / c b/float(a) or float(b)/a	1 5.0 1.5
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a a % c	5 1.0
** Exponent	Performs exponential (power) calculation on operators	a**b a**c	1000000000000000 1000.0
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	b//a b//c b//float(a) or float(b)//a	1 5.0 1.0

You are ready for exercise 1 !!!

- Go to folder exercises and open ex1.py
- Read the questions and write your code in the space provided
- Run your code when done
- Estimated time to complete 15 mins

Chapter II

Data Structures in Python

Data structures

- Data structure defines a particular way of organizing, storing and accessing data
- Lists: An ordered collection of objects
- Sets: An unordered collection of “immutable objects”
- Dictionaries: A mutable data structure that stores key, value pairs

Lists

- Lists are defined with [ ] and work similarly to strings

```
flowers = [‘Orchid’, ’Carnation’, ’Sunflower’, ‘Marigold’]
print flowers[1]            ‘Carnation’
print flowers[2:]   [‘Sunflower’, ‘Marigold’]
print len(flowers)          4
print len(flowers[2])       9
flowers.remove(‘Orchid’ )
flowers.append(564)
flowers.extend(['Roses','Tulips’]) #Adds content of list to flowers
flowers.extend(3.1415)              #Error 3.1415 is not a list
print flowers      ['Carnation', 'Sunflower', 'Marigold', 564, 'Roses’,
                   'Tulips']
```

Sets

- Sets are defined using the built-in set function, set()

```
s=set() #creates empty set
s.add(1) #add to set
print s                set([1])
s=set([1,2,3,4,5,1,2,6,7,2]) #creates set from give list
print s                set([1, 2, 3, 4, 5, 6, 7])
s=set('Mississippi')
print s                 set(['i', 'p', 's', 'M'])
```

Dictionaries

- Dictionary (dict) can be defined by using { }

```
d = {} #empty dict
d[1]='value 1'
d[2]='value 2'
print d            {1: 'value 1', 2: 'value 2'}
d = {1:'value1',2:'value2',3:'value3'}
print d[1]         'value1'
print d[5]         KeyError: 5
d['x']='valueNA'
print d            {1: 'value1', 2: 'value2', 3: 'value3', 'x': 'valueNA'}
```

You are ready for exercise 2 !!!

- Go to the folder exercises and open ex2.py
- Read the questions and write your code in the space provided
- Take a look at “data-structures.py” file under examples to see syntax
- Run your code when done
- Estimated time to complete 10 mins

Chapter III

Conditionals and Loops

Indentation

- Python uses indentation to define code blocks
- A code block is a lexical structure of the source code
  - E.g. use *for* loop to repeat statement 10 times

```
i=0
for i in range(10):
        print i          #prints 0,1,2…9
print i                  #prints 9
```

- Note the colon (:) and indentation after the *for* statement. All the statements having larger and equal margin from left are interpreted as a part of the *for* block.

If statements

- *If* statements or the *if/else* statements are used to execute/skip a code block based on a condition
- Basic syntax looks like

```
if(condition1):
        planA()
else:
        planB()
```

- Can have multiple conditions

```
if(condition1):
        planA()
elif(condition2):
        planB()
else:
        planC()
```

- Can be without else

```
if(condition1):
        planA()
```

Logical expressions

- The conditions checked by if statements are called logical expressions
- A logical expression can have a value “True” or “False” only
- If value for a condition is “True”, the corresponding code block will get executed otherwise it will be skipped

Comparison operators

Operator	Description	Example expression	Output
==	Equals to	‘str1’ == ‘str1’	True
!=	Not equal to	‘str1’ != ‘str1’	False
>	Greater than	3 > 3	False
>=	Greater than or equal to	3 >= 3	True
<	Less than	5 < 8	True
<=	Less than or equal to	5 <= 8	True
is	Is the same object	x=[‘1’,’2’] y=[‘1’,’2’] x is y y=x x is y	 False True
or	Boolean OR	5<8 or 5>8	True
and	Boolean AND	5<8 and 5>8	False
in	Membership test	‘s’ in ‘books’	True
not	Boolean NOT	not ‘s’ in ‘books’	False

Few examples on if-else

- Open the file “if-else.py” under the examples directory
- Run the “if-else.py” file and try to understand how if-else statements work.
- Make changes to the if-else conditions to allow the user to enter a power till 7 but limit the range of number from 0 till 5
- Add an additional 4th option to let user find the reciprocal of the number e.g. reciprocal of 5 is 1/5 = 0.2. What will be the reciprocal if user enters 0 ?? Can you handle this exception using an if-else statement

Loops

- A loop is a structure which allows execution of a code block repeatedly until a terminating criteria is reached.
- For loop: repeat a block of code fixed number of times
  - Requires starting and ending criteria

```
for i in range(0,10):
    print i
```

  - range() is an in-built function that will generate a list of numbers. E.g. range(0,3) will generate [0,1,2] and so will range(3). range(1,10,3) will generate [1,4,7] last argument is the step.
- While loops run until a certain condition is satisfied
  - Requires a stopping criteria

```
x=0
while(x<10):
    print x
```

Break and continue

- break and continue are special statements. *break* is used to break out of the loop and *continue* is used to skip code block and return to for or while statement and start over.

```
mylist = [1,2,3,4,5,6,7,8]
for i in l:
    if (i % 2==0):              #skips any even number
        continue
    if (i == 7):                #exits when i is equal to 7
        break
    print i                     prints 1,3,5
```

Few examples on loops

- Open the file “loops.py” under the examples directory
- Run the “loop.py” file and try to understand how for and while loop statements work.
- Write a for or while loop that allows user to see the last n primes in the list. E.g. if input is 3 output should be 61, 67, 71.

Updating values inside loops

- Often we need to change values inside a loop depending on the computational problem
  - Be careful to initialize variables before the loop

```
l=['abc', 'def']
string=''                    #important to initialize string to empty string
for x in l:
    string= string+x
print string                 #prints abcdef. Note: ''.join(l) does same thing
```

  - Example finding sum on first n natural numbers

```
n=5
totalsum=0                   #important to initialize totalsum =0
for x in range(n+1):
    totalsum=totalsum+x      #adds numbers 0 till 5
print totalsum               #prints 15
```

You are ready for exercise 3 !!!

- Go to the folder exercises/excercise3 and open ex3.py
- Read the questions and write your code in the space provided
- Run your code when done
- Estimated time to complete 15-20 mins

Chapter IV

Functions

Functions

- Functions are modules of code that perform a specific task
- Functions promote reusability of code
  - E.g. Imagine if the built-in function "len()" was not defined. You would have to write your code every time you needed to get length of an object.
- Functions make development easier by splitting a large complex program into smaller modules
- Functions make it easier to detect bugs in the program

Functions in Python

- Function in Python begins with “def” keyword followed by function name and parentheses.
- The arguments the function takes are placed in the parentheses.
- The function block starts after a “:”
- “return” statement returns a value from the function. If “return” is absent the function returns “None”

```
def funcSum(a,b):          #function name is funcSum, arguments are a and b
    return a+b     #returns a+b
print funcSum(5,2)         #calls the function funcSum with a=5 and b=2 and
                              prints 7
```

Few examples on functions

- Open the file “functions.py” under the examples directory
- Run the “functions.py” file and try to understand how functions work.
- How many arguments each functions take ?
- Write a function “getAge” which will ask for user’s age and print it.

Global and local scope

- A variable with global scope can be accessed any where in the program
- A variable with local scope is valid only in the code-block it is defined

```
a=10            #a has global scope
def func():
    b=10        #b is local to func
    a=5         #a is local to func
    print b     #prints b=10
    print a     #prints a=5, local scope
func()
print a         #prints a=10
print b         #error b is not defined, its scope ended with the function
```

Function overloading

- Python lets you define a multiple functions with same name but different arguments
  - Helpful when a given function needed to be computed differently depending

```
def areaSquare(a,b):
    return a*b
def areaCircle(r):
    return 3.14*r*r

print areaSquare(2,2)
print areaCircle(1)
```

```
def area(a,b):                  #calculate area for square
    return a*b
def area(r):                    #calculate area for circle
    return 3.14*r*r

print area(2,2)                 #prints 4
print area(1)                   #prints 3.14
```

You are ready for exercise 4 !!!

- Go to the folder exercises and open ex4.py
- Read the questions and write your code in the space provided
- Run your code when done
- Estimated time to complete 10-15 mins

Chapter V

Input/Output

Reading files in Python

- Data is usually stored in plain text files and to process/analyze data we need to first read it in our program
- Python provides a very good support via built-in functions to do file operations
- "open('filename', 'mode')" function opens the file
  - Filename is the name of the file to open, mode is one of the following mode
    1. 'r': Read only mode
    2. 'w': Write only mode
    3. 'a': Append mode
    4. 'r+': Read and write both.
- Use close() functions to close the file when done.

Simple example (easy way)

```
with open('datafile.txt') as f:          #f is a File object
        data=f.read().splitlines()       #read file line-by-line
print data                               #now file is in the list data
print len(data)                          #print total number of lines
```

- "with" allows for simpler syntax and make sure file is closed after reading is done. No need to use close()
- f.write() writes to file when opened in 'w', 'a' or 'r+' mode

Few examples on reading files

- Open the file “readfile.py” under the examples directory
- Run the “readfile.py” file and try to understand how it works.
- What happens if you use readlines() instead of read().splitlines() ?

You are ready for exercise 5 !!!

- Go to the folder exercises and open ex5.py
- Read the questions and write your code in the space provided
- Run your code when done
- Estimated time to complete 15-20 mins

Epilogue