BCB Python Workshop
10-19-2018

Urminder Singh

Welcome to intermediate Python workshop

- Organized by Bioinformatics and Computational Biology Graduate Student Organization (BCBGSO)

- Future workshops in Spring 2019. R, Unix, Python and bioinformatics

Acknowledgement

Basil

Ian

Pranav

Priyanka

Valeria

Zachary

BCB GSO

Acknowledgement

Akshay

Gaurav

Paul

Sayane

Robert Shane Therin

Volunteers

Download workshop material

- Go to: https://tinyurl.com/bcb-python

OR

- git clone https://github.com/urmi-21/python3-dataScience18.git

Contents

1. Chapter I: Conditionals and Loops

2. Chapter II: Functions

3. Chapter III: Input/Output

4. Chapter IV: Introduction to Pandas

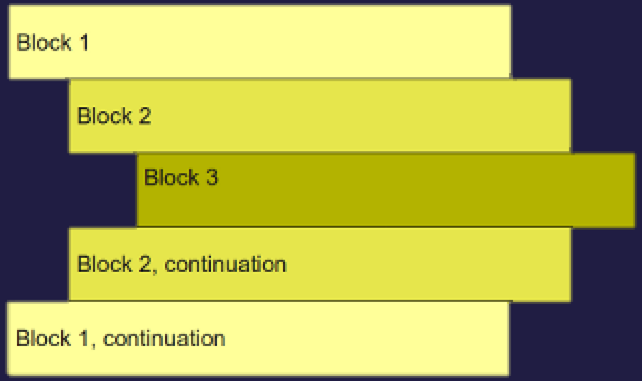
Chapter I

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
```



Block 1

Block 2

Block 3

Block 2, continuation

Block 1, continuation

- 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

For Loop

- 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. In python2, range(0,3) will generate a list [0,1,2] and so will range(3). range(1,10,3) will generate [1,4,7] last argument is the step.
  - In python3, range() returns an immutable sequence of numbers.

While loop

I call that running out of memory.
INFINITE LOOPS
Be sure to mind your repetitive control structures!

- While loops run until a certain condition is satisfied
  - Requires a stopping criteria

```
x=0
while(x<10):
    print x
    x=x+1                    #if x is not updated loop will never finish
```

- Note: Python doesn't support x++ use x=x+1 or x+=1

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 below it and return to start of the *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 1 !!!

- Go to the 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 15-20 mins

Chapter II

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
```

Lambda functions

- Lambda functions are anonymous function
- Defined without a name
- Can take any number of arguments, but can only have one expression
- Example

```
x = lambda a, b : a + b
print(x(5, 6))                #prints 11
```

- Anonymous functions are useful inside other functions.

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
```

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
- Run your code when done
- Estimated time to complete 10-15 mins

Chapter III

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(‘filepath/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
- Note: Make sure to convert data from type “str” to int or float

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 3 !!!

- Go to the folder exercises 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

Introduction to Pandas

Pandas

- Pandas is an open-source python library
- Provides flexible data structures and tools for data analysis
- Key features
  - Easy to keep data with rows and columns as data-frames
  - Easy to mutate data frames
  - Handles missing data (as NaN)
  - Intuitive joining and merging of datasets
  - Pandas is fast
- NumPy is required by pandas

Wes McKinney
Original author of Pandas

Data structures in Pandas: Series

- A *series* is a data structure which can hold a number of objects.
- Equivalent to a one dimensional array.
- Series can hold an object of any type (int, float, string etc.)
- Can be created using the constructor:

  pandas.Series(data, index, dtype, copy)

Series example

```
import pandas as pd                              #important to import the library
ser1 = pd.Series([1, 2, 3])                      #creates series with the list [1,2,3]
print(ser1)

serA = pd.Series(['a', 'b', 'c'])
print(serA)

Ser1A = pd.Series(['a', 'b', 'c’, 22.0])
print(ser1A)
print(ser1A[1])                                  #prints b
print(ser1A[3]*2)

#define your own index
cars = ['NSX', 'R8', 'chiron', '488 GTB']
mpg = [22, 22, 14, 22]
serCars = pd.Series(mpg, index=cars)
print(serCars)
print(serCars[['NSX','chiron']])
```

Data structures in Pandas: Dataframe

- A *dataframe* (DF) can hold tabular data (2-dimensional) with rows and columns.
- Logically same as an excel sheet.
- Each column in a data frame is a series. DF is a dict-like container for Series objects.
- DF is size-mutable, labelled, and capable of arithmetic operations on rows and columns
- Constructor for DF:

  pandas.DataFrame(data, index, columns, dtype, copy)

Dataframe example

```
#using lists
states=['AZ','CA','IA','KS','NY']
statesFull=['Arizona','California','Iowa','Kansas','New York']
dfStates=pd. DataFrame(list(zip(states,statesFull))) #Creates 5 rows and 2 cols
print(dfStates)

#data frame from dict example
d = {'Col1' : pd. Series ([1. , 2., 3.] ,index =[ '1', 'b', 'c']) , 'Col2' : pd. Series ([2. , 9., 4.] ,index
=[ 'a', 'b', 'c'])}
df1 = pd. DataFrame (d)
print(df1)                              #prints some NaN values
```

Importing data with pandas

- Read .csv file into dataframe using pd.read_csv("data/iris.data.csv")
- Read files using a url pd.read_csv("https://raw.githubusercontent.com/urmi-21/python3-dataScience18/master/data/iris.data.csv")
- Get summary of data using df.describe()
- See data dimensions df.shape
- Print first 10 rows df.head(10)

Pandas examples

- Open the file pandas notebook (“pandas.ipynb”)

You are ready for exercise 4 !!!

- Open “ExpresssionAnalysisEx.ipynb”
- Read the questions and write your code
- Estimated time to complete 15-20 mins

Epilogue

What you have learned

- If/else and loops
- Defining functions
- Reading data from files
- Pandas
- You can access all the workshop material at https://github.com/urmi-21/python3-dataScience18

What's next …

- Try out these really great (free) sources of python knowledge
  - The Python Tutorial: https://docs.python.org/3/tutorial/
  - Pandas Tutorial: https://pandas.pydata.org/pandas-docs/stable/tutorials.html
- If you get stuck or need help, **Google it!** Or post your questions to forums such as stackoverflow.com only if you couldn't find an answer online
- Always write comments in you code to make it more readable

That's all Folks