

# **Python For Data Science**

---

# Session - 01

---

# Course Infrastructure

- Python 3
  - Develop Scripts
  - Interactive Shell
- Jupyter - would be preferred way
  - Effective to learn Pandas, Numpy, Matplotlib and other scientific libraries from Python Programming

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types & Interaction**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**

# Hello to Python

```
#!/usr/bin/python3
print ("Hello \"World\"")
print ('Hello World')
print ("Hello, 'My Dear Friend'")
print ('Hello, "My Dear Friend"')
print ("One", "Two")
```

print is built-in function

output

```
Hello "World"
Hello World
Hello, 'My Dear Friend'
Hello, "My Dear Friend"
One Two
```

hello.py

# Hello to Python (Cont...)

```
#!/usr/bin/python3

print("""
One - Hello
two - Python
three - Data Science
""")

print ("One", "Two", 'One-1', 'Two-2')

print ("I am first line\nI am second line")
```

hello\_multiple.py

output

```
One - Hello
two - Python
three - Data Science

One Two One-1 Two-2
I am first line
I am second line
```

# Assignment - 1

- Write “hello\_datascience.py” Script
- It should print following output

```
Hello to Data Science
This is:
- Python Course
- Actually This is Data Science Course
```

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types & Interaction**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**

# Comments

```
#!/usr/bin/python3

#this is a comment in Python

print ("Hello World") #This is also a comment in Python

""" This is an example of a multiline
comment that spans multiple lines
"""

print ("Let me try triple quotes")
"""

I am also comment
in muliple lines

"""


```

output

```
Hello World
Let me try triple quotes
```

comments.py

# Mathematics

Python is perfectly suited to do Mathematics

- addition +
- subtraction -
- multiplication \*
- division /

There is also support for more advanced operations such as

- Exponentiation \*\*
- Modulo: %

# Mathematics (Cont...)

```
#!/usr/bin/python3
# Addition and subtraction
print(5 + 5)
print(5 - 5)

# Multiplication and division
print(3 * 5)
print(10 / 2)

# Exponentiation
print(4 ** 2)

# Modulo
print(18 % 7)
```

```
>>> 5+5
10
>>> 5-5
0
>>> 3*5
15
>>> 10/3
3.333333333333335
>>> int(10/3)
3
>>> 10/2
5.0
>>> int(10)/int(2)
5.0
>>> int(10/2)
5
>>> 5**2
25
>>> 10%3
1
>>>
```

# Assignment 2

Suppose you have Rs 1000. You have invested that with a 10% return per year scheme

After one year, you earnings =  $1000 \times 1.1 = 1100$

After two years, earnings =  $1000 \times 1.1 \times 1.1 = 1210$

- How much money you end up earning after 5 years?
- How much money you end up earning after 10 years?

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types & Interaction**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**
- **Lists Revisits**

# variables

```
#!/usr/bin/python3

counter = 1000          # An integer assignment
miles   = 1050.0         # A floating point
name    = "Hari Sadu"   # A string

print (counter)
print (miles)
print (name)

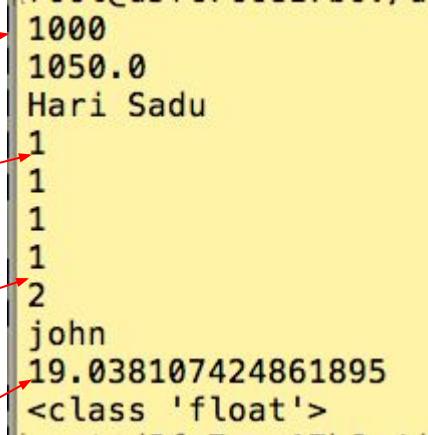
a = b = c = 1

print (a)
print (b)
print (c)

a, b, c = 1, 2, "john"

print (a)
print (b)
print (c)

#BMI - Body Mass Index = weight/(height)^2
weight = 61.0
height = 1.79
bmi = weight / height ** 2
```



The screenshot shows the output of the Python script. It displays the values assigned to variables counter, miles, name, a, b, c, and the BMI calculation. The values are: 1000, 1050.0, Hari Sadu, 1, 1, 1, and 19.038107424861895 respectively. The class of the float value is also printed as <class 'float'>.

1000  
1050.0  
Hari Sadu  
1  
1  
1  
1  
1  
2  
john  
19.038107424861895  
<class 'float'>

variables.py

# Assignment - 3

- Create a variable *savings* equal to 1000
- Create a variable *interest* equal to 1.10.
- Use *savings* and *interest* to calculate the amount of money you end up earning after 8 years.
- Store the result in a new variable, *earnings*
- Print out the value of *earnings*

# Word on Data Types

- Numbers
  - Int
  - float
- Character Strings
- Boolean - True, False
- List
- Tuple
- Dictionary
- Numbers:
  - 3, 6, -7, 1.25
- Character strings:
  - 'hello', 'The answer is: '
- Lists of objects of any type:
  - [1, 2, 3, 4], ['yes', 'no', 'maybe']
- A special datum meaning nothing:
  - None

# Interact with Script

Python has many built-in functions - Example: print, input, type

```
>>> type(7)
<class 'int'>
>>> type('abcd')
<class 'str'>
>>> len('abcd')
4
>>> len([1, 2, 4, 5, 6, 10])
6
>>> type([1, 2, 4, 5, 6, 10])
<class 'list'>
>>> input('Enter your name: ')
Enter your name: Hari
'Hari'
>>> █
```

# Interact with Script (Cont ...)

```
#!/usr/bin/python3
'''Illustrate input and print.'''

applicant = input("Enter the applicant's name: ")
interviewer = input("Enter the interviewer's name: ")
time = input("Enter the appointment time: ")

print(interviewer, "will interview", applicant, "at", time)
```

```
[Enter the applicant's name: John
[Enter the interviewer's name: Hari
[Enter the appointment time: 12:00
Hari will interview John at 12:00
-----
```

interview.py

```
#!/usr/bin/python3

'''Conversion of strings to int before addition'''

xString = input("Enter a number: ")
x = int(xString)
yString = input("Enter a second number: ")
y = int(yString)
print('The sum of ', x, ' and ', y, ' is ', x+y, '.', sep='')
```

```
Enter a number: 23
Enter a second number: 78
The sum of 23 and 78 is 101.
```

addition.py

# Making Decisions

```
#!/usr/bin/python3
# If the number is positive, we print an appropriate message

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")

num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

3 is a positive number.  
This is always printed.  
This is also always printed.

[if\\_demo.py](#)

# Making Decisions (Cont...)

```
#!/usr/bin/python3
# Program checks if the number is positive or negative
# And displays an appropriate message

num = 3

# Try these two variations as well.
# num = -5
# num = 0

if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

Positive or Zero

if\_else\_demo.py

# Making Decisions (Cont...)

```
#!/usr/bin/python3
# In this program,
# we check if the number is positive or
# negative or zero and
# display an appropriate message

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

```
Positive number
```

[if\\_elif\\_demo.py](#)

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types & Interaction**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**

# Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:] )
  - with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

## OPERATIONS

- Concatenation +
- Repetition with the help of asterisk \*

# Strings (Cont...)

```
#!/usr/bin/python3

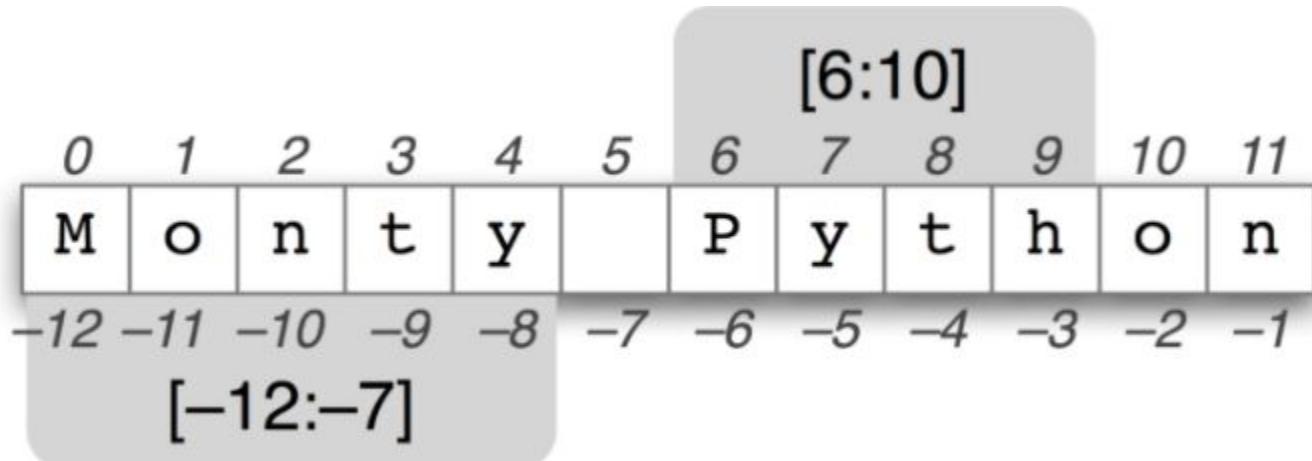
str = 'Hello World!'

print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])       # Prints string starting from 3rd character
print (str * 2)       # Prints string two times
print (str + "TEST") # Prints concatenated string
```

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

strings\_demo.py

# Strings (Cont...)



```
#!/usr/bin/python3

s = 'Hello World'
print (len(s))
print (type(s))

s = 'Monty Python'

print(s[6:10])
print(s[-12:-7])
```

```
11
<class 'str'>
Pyth
Monty
```

strings\_len.py

# Strings - Quiz

- ❖ What is the output when following statements is executed ?
  - `>>>"a"+"bc"`
  - `>>>"abcd"[2:]`
- ❖ What arithmetic operators cannot be used with strings ?
  - `+, -, *, **`
- ❖ What is the output when following code is executed ?
  - `>>>print r"\nhello"`
  - `print '\x97\x98'`
- ❖ What is the output when following code is executed ?
  - `>>>str1="helloworld"`
  - `>>>str1[::-1]`
- ❖ What is the output of the following?
  - `print("xyzxyzxyz".count('yy'))`
  - `print("xyzxyzxyz".count('yy', 1))`
  - `print("xyzxyzxyz".count('xy', 0, 100))`

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types & Interaction**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**
- **Lists Revisits**
- **Handling Errors**
- **Functions**
- **Modules**
- **Packages**
- **Dictionaries**
- **Classes**
- **Iterators**
- **String Formatting**
- **Regular Expressions**

A list contains items separated by commas  
and enclosed within square brackets ([])

# Python Lists

```
#!/usr/bin/python3
```

```
list      = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list )                  # Prints complete list
print (list[0] )                # Prints first element of the list
print (list[1:3] )              # Prints elements starting from 2nd till 3rd
print (list[2:] )               # Prints elements starting from 3rd element
print (tinylist * 2 )           # Prints list two times
print (list + tinylist )        # Prints concatenated lists
```

```
print (type(list))            # type
print(len(list))               # length
```

**List is sequence**

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
<class 'list'>
5
```

lists\_demo.py

# for - Loop

```
#!/usr/bin/python3

languages = ["C", "Java", "Perl", "Python", "Node.js", "JavaScript"]

for x in languages:
    print (x)
```

```
C
Java
Perl
Python
Node.js
JavaScript
```

for\_demo.py

# for - loop (Cont...)

```
#!/usr/bin/python3

edibles = ["ham", "spam","eggs","nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        break
    print("Great, delicious " + food)
else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")
```

```
Great, delicious ham
No more spam please!
Finally, I finished stuffing myself
```

for\_demo\_adv.py

for\_demo\_adv\_nospam.py

# Assignment - 4

Write a script to print pattern like below.



# while - loop

```
#!/usr/bin/python3

count = 0
while (count < 9):
    print ('The count is:', count)
    count = count + 1

print ("Good bye!")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

while\_demo.py

# **Assignment - 5**

Write a program - to count “even” and “odd” numbers which are less than 50.

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types & Interaction**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**

# **Session - 02**

---

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# Handling Errors

```
>>> while True:  
...     age = int(input("Your age please:"))  
...     print(age)  
  File "<stdin>", line 3  
    print(age)  
           ^  
SyntaxError: invalid syntax  
>>>  
>>> while True;  
  File "<stdin>", line 1  
    while True;  
           ^  
SyntaxError: invalid syntax  
>>>  
>>> while True:  
...     age = int (input("Your age please: "))  
...     print (age)  
...  
Your age please: 22  
22  
Your age please: 45  
45  
Your age please: df  
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
ValueError: invalid literal for int() with base 10: 'df'  
>>>
```

```
|>>> a  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'a' is not defined  
>>> 1/0  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: division by zero  
>>> '10'+10  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: Can't convert 'int' object to str implicitly  
>>> █
```

# Handling Errors (cont...)

Two Types of Errors - One is syntax error & other one is Exceptions

Syntax Errors - correct the syntax

Exceptions - you need to handle those exceptions depending on the logic of the application

**IOError** : If the file cannot be opened.

**ImportError** : If python cannot find the module

**ValueError** Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value

**KeyboardInterrupt** Raised when the user hits the interrupt key (normally Control-C or Delete)

**EOFError** Raised when one of the built-in functions (`input()` or `raw_input()`) hits an end-of-file condition (EOF) without reading any data

# Handling Errors(Cont..)

```
>>> try:  
...     d = 1/0  
... except:  
...     print ("I could not divide 1 by 0")  
...  
I could not divide 1 by 0  
>>>  
>>> 1/0  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: division by zero  
>>> try:  
...     d = 1/0  
... except ZeroDivisionError:  
...     print("Don't try to divide by 0")  
...  
Don't try to divide by 0  
>>> try:  
...     d = 10/2  
... except ZeroDivisionError:  
...     print("Don't try to divide by 0")  
... else:  
...     print("I am in else")  
...  
I am in else  
>>> print(d)  
5.0  
>>>
```

```
#!/usr/bin/python3

my_lines = []
flag = True
while(flag):
    try:
        line = input()
        line.rstrip()
        my_lines.append(line)
    except EOFError:
        flag = False

print ("Let us print list of lines")

for l in my_lines:
    print (l)
```

## handle\_eof.py

# Assignment - 1

Develop a script to take two numbers as input. Divide first number by second number. It should handle division by zero. The interaction of the script should look something similar to following

```
$divide.py
```

```
Enter First Number:10
```

```
Enter Second Number:0
```

```
You tried to divide 10 by zero
```

```
$divide.py
```

```
Enter First Number:10
```

```
Enter Second Number:2
```

```
The division is: 5.0
```

# Revisit Lists

## Creation of Lists:

- `L = []`
- `L = [expression, ...]`
- `L = list() # empty list`  
`L = list(sequence)`
- `A = B = [] # both names will point to the same list`
- `A = []`
- `B = A # both names will point to the same list`
- `A = []; B = [] # independent lists`

## Accessing Lists

- `n = len(L)`
- `item = L[index]`
- `seq = L[start:stop]`
- `seq = L[start:stop:step]`
- `seq = L[::-2] # get every other item, starting with the first`
- `seq = L[1::2] # get every other item, starting with the second`

# Revisit Lists (Cont...)

```
for item in L:  
    print (item)
```

```
for index, item in enumerate(L):  
    print index, item
```

# Revisit Lists (Cont...)

## Other important Operations

- `L.append(item)`
- `L.extend(sequence)`
- `L.insert(index, item)`
- `del L[i]`
- `del L[i:j]`
- `item = L.pop() # last item`
- `item = L.pop(0) # first item`
- `item = L.pop(index)`
- `L.remove(item)`
- `L.reverse()`
- `L.sort()`
- `out = sorted(L)`

# Revisit Lists (Cont ...)

```
[>>> numbers = [1, 2, 1, 2, 4]
[>>> numbers
[1, 2, 1, 2, 4]
[>>> numbers.sort()
[>>> numbers
[1, 1, 2, 2, 4]
[>>>
[>>>
[>>> new_numbers = [1, 2, 4, 10, 1, 3]
[>>> sorted_new_numbers = sorted(new_numbers)
[>>> sorted_new_numbers
[1, 1, 2, 3, 4, 10]
[>>>
[>>> new_numbers
[1, 2, 4, 10, 1, 3]
[>>> another_way_sorted_new_numbers = new_numbers.sort()
[>>> another_way_sorted_new_numbers
[>>> print (another_way_sorted_new_numbers)
None
[>>> ]
```

# Quiz

- Q.1. Select the right options

```
names = ['Amir', 'Sahrukh', 'Chales', 'Dao']
print names[-1][-1]
```

So what's the output?

- i. A
- ii. r
- iii. Amir
- iv. Dao

- Q. 2. What gets printed?

```
names1 = ['Amir', 'Sahrukh', 'Chales', 'Dao']
names2 = names1
names3 = names1[:]

names2[0] = 'Alice'
names3[1] = 'Bob'

sum = 0
for ls in (names1, names2, names3):
    if ls[0] == 'Alice':
        sum += 1
    if ls[1] == 'Bob':
        sum += 10

print (sum)
```

# Quiz (Cont...)

- Q. 3. What gets printed?

```
names1 = ['Amir', 'Sahrukh', 'Chales', 'Dao']
loc = names1.index("Edward")

print (loc)
```

- Q. 4. What's printed in following code?

```
names1 = ['Amir', 'Sahrukh', 'Chales', 'Dao']

if 'amir' in names1:
    print (1)
else:
    print (2)
```

- Q. 5. What gets printed?

```
numbers = [1, 2, 3, 4]
numbers.append([5,6,7,8])
print (len(numbers))
```

# Assignment - 2

Q 1 Develop a script to create list of lists. The element list should be factors of index of the list.

(Note - We are yet to talk about “functions”)

Q 2. Develop a script to find the Highest Common Factor (HCF) of given two numbers.

\$indexes_are_lists.py	\$indexes_are_lists.py
0	3
[]	[[0], [1], [1, 2]]
\$indexes_are_lists.py	\$indexes_are_lists.py
1	4
[[0]]	[[[0], [1], [1, 2], [1, 2, 3]]]
\$indexes_are_lists.py	
2	
[[0], [1]]	

# Functions

Let us look at the problem of converting decimal number into binary number

```
11 = 5 * 2 + 1
5 = 2 * 2 + 1
2 = 2 * 1 + 0
1 = 2 * 0 + 1

0 's binary expression is - 1
1 's binary expression is - 1

so 11 's binary expression is - 1011
```

```
def functionname( parameters ):
    "function_docstring"
    python statements - logic/code of function
    return [expression]
```

# Functions

```
#!/usr/bin/python3

name = "Aegis"

def hellofunction(name=None):
    '''hello function'''
    if name:
        print ("Hello " + name )
    else:
        print ("Hello World!")

#hellofunction(name)
hellofunction()
```

```
#!/usr/bin/python3
def decToBin(n):
    if n==0:
        return '0'
    else:
        return decToBin(int(n/2)) + str(n%2)

d = decToBin(11)
print (d)
```

function\_dec2binary.py

function\_hello.py

# Assignment - 3

Q 1 Develop a script to create list of lists. The element list should be factors of index of list.

(Note - We have talked about “functions”)

Q 2. Develop a script to find the Highest Common Factor (HCF) of given two numbers.

```
$indexes_are_lists.py      $indexes_are_lists.py  
0                          3  
[]                         [[0], [1], [1, 2]]  
$indexes_are_lists.py      $indexes_are_lists.py  
1                          4  
[[0]]                      [[0], [1], [1, 2], [1, 2, 3]]  
$indexes_are_lists.py  
2  
[[0], [1]]
```

# Error Handling - Revisit

Observe how “try”, “except”, “else”, “finally” is used here

```
#!/usr/bin/python3

def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")

divide(2, 1)
divide(2, 0)
divide("2", "1")
```

error\_with\_finally.py

```
result is 2.0
executing finally clause
division by zero!
executing finally clause
executing finally clause
Traceback (most recent call last):
  File "./error_with_finally.py", line 15, in <module>
    divide("2", "1")
  File "./error_with_finally.py", line 5, in divide
    result = x / y
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

# Error Handling - Revisit (Cont ... )

A ***finally* clause** is always executed before leaving the **try** statement, whether an **exception** has occurred or not. When an **exception** has occurred in the **try** clause and has not been handled by an **except** clause (or it has occurred in an **except** or **else** clause), it is re-raised after the **finally** clause has been executed.

foo\_finally.py

```
#!/usr/bin/python3
def foo():
    try:
        return 1
    finally:
        return 2
k = foo()
print(k)
```

- a) 1
- b) 2
- c) 3
- d) error, there is more than one return statement in a single try-finally block

How about replacing “return 1” by “print (1)” and “return 2” by “print (2)” ?

# Revisit Lists

- Comprehensions
- Lambda Functions
- Map
- Filters

```
>>> for x in [1, 3, 4, 2, 5, 6]:  
...     for y in [2, 3, 5, 4, 10]:  
...         if x != y:  
...             print ("(", x, y, ")")  
  
(...  
( 1 2 )  
( 1 3 )  
( 1 5 )  
( 1 4 )
```

```
|>>> squares = []  
|>>> for x in range(10):  
|...     sq = x ** 2  
|...     squares.append(sq)  
|...  
|>>>  
>>> squares  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
>>>
```

Old Way

```
|>>> new_style = [x ** 2 for x in range(10)]  
>>> new_style  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
>>>
```

New Way

Old Way

```
>>> [(x, y) for x in [1, 3, 4, 2, 5, 6] for y in [2, 3, 5, 4, 10] if x != y]  
[(1, 2), (1, 3), (1, 5), (1, 4), (1, 10), (3, 2), (3, 5), (3, 4), (3, 10), (4, 2), (4, 3), (4, 10), (5, 2), (5, 3), (5, 4), (5, 10), (6, 2), (6, 3), (6, 5), (6, 4), (6, 10)]  
>>>
```

New Way

# Revisit Lists (Cont ...)

lambda arguments: expression

```
[>>> sqr = lambda x: x ** 2
[>>> sqr(10)
100
[>>> sqr(15)
225
[>>> multiplication = lambda x, y: x * y
[>>> multiplication(7, 5)
35
[>>> multiplication(10, 5000)
50000
...]
```

```
[>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[>>> matrix
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[>>> flat = [x for row in matrix for x in row]
[>>> flat
[1, 2, 3, 4, 5, 6, 7, 8, 9]
...]
```

The filter() function in Python takes in a function and a list as arguments. Function is evaluated for True value.

The map() function in Python takes in a function and a list. The new list has elements returned by function.

# Revisited Lists (Cont ...)

```
|>>> my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 16, 34, 35, 33]
|>>> new_list = list(filter(lambda x: (x %2 == 0), my_list))
|>>> new_list
[2, 4, 6, 8, 10, 12, 16, 34]
|>>>
```

```
|>>> another_list = list(map(lambda x: x ** 2, my_list))
|>>> another_list
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 144, 256, 1156, 1225, 1089]
|>>>
```

- List Comprehensions are cleaner than map, filter (because they don't require any extra "lambda" function)
- In comprehensions - you can skip items from input lists easily. Same behaviour in Maps can not be avoided. It depends on "filter"

# **Assignment -4**

Q 1 Develop a script to create list of lists. The element list should be factors of index of list.

Q 2. Develop a script to find the Highest Common Factor (HCF) of given two numbers.

Use Lambda, Map, Filter to solve above

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# Modules

- Modules are Python files with the .py extension.
- These files implement a set of functions and can have python statements.
- The modules can be imported from other modules using the import command.

The modules are best way to share your work, your tools with others.

Have a look at:

- <https://docs.python.org/3/library/>
- <https://docs.python.org/2/library/>

# Modules (cont...)

```
#!/usr/bin/python3

def helloworld():
    print ("Hello World!")

def goodbye():
    print ("Good Bye Dear!")
```

```
#!/usr/bin/python3
from hello_module import goodbye

print ("-----I")
print (goodbye)

print ("-----II")
goodbye()
```

```
-----I
<function goodbye at 0x7f538a412ea0>
-----II
Good Bye Dear!
```

hello\_module.py

test\_hello\_module.py

# Modules (Cont...)

```
>>> import hello_module
>>> help(hello_module)
Help on module hello_module:

NAME
    hello_module

FUNCTIONS
    goodbye()

    helloworld()

FILE
    /datascience/hello_module.py

>>> dir(hello_module)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
 'world']
>>>
>>> from hello_module import goodbye
>>> help(goodbye)
Help on function goodbye in module hello_module:

goodbye()

>>> goodbye()
Good Bye Dear!
>>> █
```

# Modules (Cont ...)

```
>>> import sys
>>> sys.version
'3.5.2 (default, Nov 17 2016, 17:05:23) \n[GCC 5.4.0 20160609]'
>>> sys.version_info
sys.version_info(major=3, minor=5, micro=2, releaselevel='final', serial=0)
>>>
```

```
#!/usr/bin/python3
# import module sys to get the type of exception
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
        print()
print("The reciprocal of",entry,"is",r)
```

The entry is a  
Oops! <class 'ValueError'> occurred.  
Next entry.

The entry is 0  
Oops! <class 'ZeroDivisionError'> occurred.  
Next entry.

The entry is 2  
The reciprocal of 2 is 0.5

[one\\_more\\_exception.py](#)

# Module (Cont ...)

```
#!/usr/bin/python3

import subprocess
def disk(partition="/"):
    info = subprocess.call(["df", partition])

if __name__ == '__main__':
    import sys
    disk(sys.argv[1])
```

new\_monitor.py

```
import subprocess
def disk(partition="/"):
    info = subprocess.call(["df", partition])
```

monitor.py

```
[root@d5fc7cce17b6:/datascience# ./new_monitor.py /home
Filesystem      1K-blocks      Used      Available      Use%      Mounted on
none            61890340  24866440  33856976  43% /
[root@d5fc7cce17b6:/datascience#
[root@d5fc7cce17b6:/datascience#
[root@d5fc7cce17b6:/datascience#
[root@d5fc7cce17b6:/datascience# python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from new_monitor import disk
>>> disk("/home")
Filesystem      1K-blocks      Used      Available      Use%      Mounted on
none            61890340  24866440  33856976  43% /
```

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# Packages

- Help structure Python's module namespace by using “dotted module names”.
  - The module name *package\_a.mod\_b* designates a submodule named *mod\_b* in a package named *package\_a*.
- Use of dotted module names saves the developers of multi-module packages from having to worry about each other's global variable names.

Let us assume we have the following directory structure. Here, `hello.py` & `monotor.py` are same modules described in *Module* section, and `init.py` is an empty file:

# Packages (Cont ... )

```
mypackage  
|-- __init__.py  
|-- hello_module.py  
`-- monitor.py  
  
0 directories, 3 files
```

*init.py* helps Python to treat this directory (*/mypackage*) as package directory.

```
>>> from mypackage import hello  
>>> from mypackage import monitor  
>>> dir(monitor)  
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__  
s']  
>>> dir(hello)  
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__  
orld']  
>>> hello.helloworld()  
Hello World!  
>>> monitor.disk("/home")  
Filesystem      1K-blocks      Used Available Use% Mounted on  
none          61890340  24866440  33856976  43% /  
>>>
```

# Assignment - 5

Q1. Create Package - which has modules to

- List factors of a given number
- Calculate HCF of given numbers
- Calculate the Factorial

# Strings

[1] What is str ? [2] How they are created? [3] How to escape chars like “\n”? [4] how to create multi-line strings? [5] Python strings are “immutable” - what does this mean?

- `s.lower()`, `s.upper()`
- `s.strip()` -- returns a string with whitespace removed from the start and end
- `s.isalpha()`/`s.isdigit()`/`s.isspace()`... -- tests if all the string chars are in the various character classes
- `s.startswith('other')`, `s.endswith('other')` -- tests if the string starts or ends with the given other string
- `s.find('other')` -- searches for the given other string within s, and returns the first index where it begins or -1 if not found
- `s.replace('old', 'new')` -- returns a string where all occurrences of 'old' have been replaced by 'new'
- `s.split('delim')` -- returns a list of substrings separated by the given delimiter. it's just text.  
`'aaa,bbb,ccc'.split(',') -> ['aaa', 'bbb', 'ccc'].`
- `s.join(list)` -- opposite of `split()`, joins the elements in the given list together using the string as the delimiter. e.g. `'---'.join(['aaa', 'bbb', 'ccc']) -> aaa---bbb---ccc`

# String (Cont ... )

- %s - String (or any object with a string representation, like numbers)
- %d - Integers
- %f - Floating point numbers
- %.f - Floating point numbers with a fixed amount of digits to the right of the dot.
- %x/%X - Integers in hex representation (lowercase/uppercase)
- New style {} & format function on strings

```
>>> name = "Donalt"
>>> age = 72
>>> print ("%s is %d years old " % (name, age))
Donalt is 72 years old
>>> mylist = [1,2,3]
>>> print("A list: %s" % mylist)
A list: [1, 2, 3]
>>>
```

```
>>> print ("The President {} is {} old".format(name, age))
The President Donalt is 72 old
>>> █
```

# String (Cont ... )

```
[>>> print ("{:.2f}".format(1.123456))
1.12
```

Number	Format	Output	Description
3.1415926	{:.2f}	3.14	2 decimal places
3.1415926	{:+.2f}	+3.14	2 decimal places with sign
-1	{:+.2f}	-1.00	2 decimal places with sign
2.71828	{:.0f}	3	No decimal places
5	{:>2d}	05	Pad number with zeros (left padding, width 2)
5	{:x<4d}	5xxx	Pad number with x's (right padding, width 4)
10	{:x<4d}	10xx	Pad number with x's (right padding, width 4)
1000000	{:,}	1,000,000	Number format with comma separator
0.25	{:.2%}	25.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
13	{:10d}	13	Right aligned (default, width 10)
13	{:<10d}	13	Left aligned (width 10)
13	{:^10d}	13	Center aligned (width 10)

# Assignment - 6

Find the Mean, Median, Mode for given list of integers. Make sure following criteria is achieved:

- [1] The script takes input as integers, space separated
- [2] Mean, Mode, Median are printed in separate lines. The numbers should be printed upto one decimal point (e.g. 3.1 and not 3.1414)

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# Dictionaries

Also known as:

- *Associate Array*
- *Map*
- *Hash Map*
- *Unordered Map*

# Dictionaries (Cont ... )

```
#!/usr/bin/python3

d1 = {}
print ("-----I")
print (type(d1))

d2 = {'one': 1, 'two':2}
print ("-----II")
print (d2)
print ("-----III")
print (type(d2))

d3 = dict(one=2, three=4)
print ("-----IV")
print (d3)
print (type(d3))

d4 = dict([(1, 2), (3, 4)])
print ("-----V")
print (d4)

d5 = dict({1:2, 3:4})
print ("-----V")
print (d5)
```

```
-----I
<class 'dict'>
-----II
{'one': 1, 'two': 2}
-----III
<class 'dict'>
-----IV
{'one': 2, 'three': 4}
<class 'dict'>
-----V
{1: 2, 3: 4}
-----V
{1: 2, 3: 4}
```

dictionaries.py

# Dictionaries (Cont ... )

- d.clear()
- d.copy()
- del k[d]
- dict.fromkeys(seq[, value])
- iteration/accessing elements of dictionaries
- for key in my\_dictionary:
- for key, value in my\_dictionary.items():
- for value in my\_dictionary.values():
- iter(d)
- len(d)
- d.keys()
- d.values()
- d.items()

# Assignment - 7

- Create a List - "A", elements are integers. List may have repeated elements
- Create new "B", which has same element from "A" but elements of "B" are not repeated elements
- Develop a function - to take generic list and return sorted, unique element list (we are still talking about integers)

# Assignment 8

Develop A Phone-Book using Dictionaries. Functions similar to following:

\$dictionaries\_maps.py

2

Hari 9850111111

Sadu 9850222222

Enter Names for which you would like to query numbers

John

Hari

Sadu

Printing Phone Book for Queried Names

For John Number not found

Hari=9850111111

Sadu=9850222222

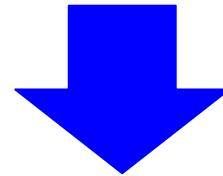
# Assignment 9

Create Dictionary - Take input as Integer, say it's "N". The keys are list of integers upto "N". The values for these keys are factors of the keys.

How about using dictionaries to improve the speed of program calculations?

# Caching or Memoization

```
def some_function (arg1, arg2, arg3,..., argN):
    result = some computation involving arg1, arg2, arg3, ... argN
    return result
```



```
cache = {}
def some_function_with_catching(arg1, arg2, arg3, ..., argN):
    key = str(arg1) + str(arg2) + str(arg3) + .... + str(argN)
    if key in cache:
        return cache[key]
    else:
        result = same computation involving arg1, arg2, arg3, ..., argN
        cache[key]=result
    return result
```

# Fibonacci Numbers

The Fibonacci Sequence is the series of numbers:

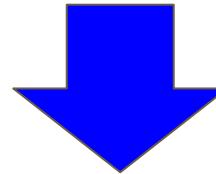
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

- The 2 is found by adding the two numbers before it (1+1)
- The 3 is found by adding the two numbers before it (1+2),
- And the 5 is (2+3),
- and so on!

# Caching or Memoization - Fibonacci Numbers

```
def fib(n):
    return n if n < 2 else fib(n-2) + fib(n-1)
```



```
__fib_cache = {}
def fib(n):
    if n in __fib_cache:
        return __fib_cache[n]
    else:
        __fib_cache[n] = n if n < 2 else fib(n-2) + fib(n-1)
    return __fib_cache[n]
```

# Class

```
#!/usr/bin/python3
class MyClass(object):
    variable = "myvalue"

    def function(self):
        print("This is a message inside the class.")

myobj = MyClass()

print (type(myobj))
print (myobj.variable)

yourobj = MyClass()
print (yourobj.variable)

yourobj.variable = "yourvalue"

print (yourobj.variable)
yourobj.function()
```

```
<class '__main__.MyClass'>
myvalue
myvalue
yourvalue
This is a message inside the class.
```

my\_class.py

# Class (Cont ... )

- Classes are essentially a template to create your objects.
- `init` is the constructor for a class.
  - The `self` parameter refers to the instance of the object

```
#!/usr/bin/python3
class MyClass:
    variable = 'myvalue'
    def __init__(self, value = None):
        if value:
            self.variable = value

    def function(self):
        print("This is a message inside the class.")

    def __repr__(self):
        return "I am representation"

myobj = MyClass("Hello")
print (type(myobj))
print (myobj.variable)

print (myobj)
```

```
<class '__main__.MyClass'>
Hello
I am representation
```

my\_advance\_class.py

# Class (Cont ... )

```
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self):
        return "Name: %s , Age: %d \n" % (self.name, self.age)
```

Person.py

sort\_person.py

```
#!/usr/bin/python3
from Person import Person

def byAge(Person):
    return Person.age

p1 = Person("Doland Trump", 70)
p2 = Person("Barack Obama", 55)
p3 = Person("G Bush", 62)
p4 = Person("Bill Clinton", 54)
p5 = Person("Ronald Reagan", 77)

presidents = [p1, p2, p3, p4, p5]
print(presidents)
sorted_presidents = sorted(presidents, key=byAge)
print(sorted_presidents)
```

```
[Name: Doland Trump , Age: 70
, Name: Barack Obama , Age: 55
, Name: G Bush , Age: 62
, Name: Bill Clinton , Age: 54
, Name: Ronald Reagan , Age: 77
]
[Name: Bill Clinton , Age: 54
, Name: Barack Obama , Age: 55
, Name: G Bush , Age: 62
, Name: Doland Trump , Age: 70
, Name: Ronald Reagan , Age: 77
]
```

# Assignment - 10

- Create Class - "City"
- Attributes
  - a. Population
  - b. Country
- Develop Script - to sort Cities (City instances) by Population
- Add GDP to City Class
  - a. sort by GDP

# **Explore - Python Modules (optparser)**

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# Interacting with Files

Open - function in Python

```
file_object = open ("filename", "mode")
```

Mode is one of the following:

- r : Read mode which is used when the file is only being read
- w : Write mode which is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)
- a : Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end
- r+ : Special read and write mode, which is used to handle both actions when working with a file

# Interacting with Files (Cont...)

```
#!/usr/bin/python3

f = open ('simple_data.txt', 'w')

f.write("Hello to writing data into file")
f.write("Line -----2")
f.write("Line -----3")
f.close()
```

simple\_file\_creation.py

simple\_data.txt

```
Hello to writing data into fileLine -----2Line -----3
```

```
#!/usr/bin/python3

f = open('simple_data.txt', 'r')

print (f.read())
f.close()
```

simple\_file\_read.py

# Interacting with Files (Cont...)

```
#!/usr/bin/python3

def do_stuff_that_fails():
    print("I actually failed to work")

def do_stuff_when_it_doesnt_work():
    print("Tried but it did not work")

try:
    with open('my_file_which_do_not_exist') as f:
        read_data = f.read()
        print (read_data)

        do_stuff_that_fails()
except (IOError, OSError) as e:
    do_stuff_when_it_doesnt_work()
```

good\_deal.py

# Getting Data from Internet

```
#!/usr/bin/python3
import urllib.request
import time

stocks_to_pull = ['AMD', 'BAC', 'MSFT', 'TXN', 'GOOG']

def pull_data_for_stock(stock):
    stock_file = stock + '.txt'
    url = 'http://chartapi.finance.yahoo.com/instrument/1.0/' + stock + '/chartdata;type=quote;range=1y/csv'
    print ("Data from URL: " + url)
    with urllib.request.urlopen(url) as f:
        source = f.read().decode('utf-8')

    split_source = source.split('\n')
    for line in split_source:
        saveFile = open(stock_file, 'a')
        linetoWrite = line + '\n'
        saveFile.write(linetoWrite)

    print('Pulled', stock)
    print('...')
    time.sleep(.5)

if __name__=="__main__":
    for stock in stocks_to_pull:
        print ("working on " + stock)
        pull_data_for_stock(stock)
```

get\_stock.py

# Assignment - 11

Get data from -

[https://web.williams.edu/Mathematics/sjmiller/public\\_html/317/DietProblemData.csv](https://web.williams.edu/Mathematics/sjmiller/public_html/317/DietProblemData.csv)

assignment\_url.txt

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# Regular Expressions

Let us define some rules to form some strings:

- Write a letter "a" at least once
- Append to this the letter "b" exactly five times
- Append to this the letter "c" any even number of times
- Optionally, write the letter "d" at the end

Examples of such strings are:

aaaabbbbccccd

aabbcc

...

# Regular Expression ( Cont ... )

There are infinitely many such strings which satisfy above rules.

Regular Expressions are merely a shorthand way of expressing these sets of rules

- Regex are text matching patterns described with a formal syntax
- The patterns which are executed on text as input to produce either matching subset or modified version of original text
- Regular Expression is kind of programming language itself
- "re" module provides this functionality in Python Programming

**Your friendship with re will always add advantage to your skills if you ever need to deal with Text Processing in your project.**

# Regular Expressions (Cont ...)

```
#!/usr/bin/python3
import re

pattern = 'Hello'
text = 'Hello Data Science Folks, How are you?'

match = re.search(pattern, text)

s = match.start()
e = match.end()

print('Found "{}" in "{}" from {} to {} ("{}")'.format(match.re.pattern, match.string, s, e, text[s:e]))
```

```
Found "Hello" in "Hello Data Science Folks, How are you?" from 0 to 5 ("Hello")
```

simple\_match.py

# Regular Expression (Cont ... )

Python supports compilation of pattern - it's more efficient to compile the pattern and use it. The `compile()` function converts an expression string into a `RegexObject`.

```
#!/usr/bin/python3
import re

# Precompile the patterns
regexes = [ re.compile(p) for p in ['Hello', 'Donald'] ]
text = 'Hello DataScience folks, How are you doing today?'

print('Text: {!r}\n'.format(text))

for regex in regexes:
    print(type(regex))
    print('Seeking "{}" ->'.format(regex.pattern), end=' ')
    if regex.search(text):
        print('Matchig!')
    else:
        print('No, I am not matching')

Text: 'Hello DataScience folks, How are you doing today?'
<class '_sre.SRE_Pattern'>
Seeking "Hello" -> Matchig!
<class '_sre.SRE_Pattern'>
Seeking "Donald" -> No, I am not matching
```

simple\_compiled.py

# Regular Expression (Cont ... )

```
#!/usr/bin/python3
import re

text = 'abbaaabbbbaaaaa'
pattern = 'ab'

for match in re.findall(pattern, text):
    print (type(match) )
    print ('Found "%s"' % match)
```

```
<class '_sre.SRE_Match'>
Found "ab" at 0:2
<class '_sre.SRE_Match'>
Found "ab" at 5:7
```

find\_all.py

```
<class 'str'>
Found "ab"
<class 'str'>
Found "ab"
```

find\_iter.py

```
#!/usr/bin/python3
import re

text = 'abbaaabbbbaaaaa'
pattern = 'ab'

for match in re.finditer(pattern, text):
    print (type(match))
    s = match.start()
    e = match.end()
    print ('Found "%s" at %d:%d' % (text[s:e], s, e))
```

# Regular Expression (Cont ... )

## A Few Rules: Commonly Used RegEx symbols

symbol	Meaning	Example Pattern	Example Matches
*	Matches Preceding Char, Subexpression, or bracketed char 0 or more times	ab	aaaaaa, aaabbbb, bbbb
+	Matches Preceding Char, Subexpression, or bracketed char 1 or more times	a+b+	aaaaab, aaabbbb, abbbb
[]	Matches any char within bracket	[A-Z]*	APPLE, CAPITAL,
()	A group subexpression	(ab)	aaabaab, abaaab
{m, n}	Matches the preceding character, subexpression, or bracketed chars between m and n times	a{2,3}b{2,3}	
[^]	Matches any single character that is not in the brackets	[^A-Z]*	aapple
		Matches any char, or subexpression, separated by	
.	Matches any single character	b.d	bed, bzd, b\$d
^	Beging of line	^a	apple, an,
An Escape Char			
\$	Used for end of line char	[A-Z][a-z]\$	ABCabc, zzzyz, Bob

## Matching Codes

Code	Meaning
\d	a digit
\D	a non-digit
\s	whitespace (tab, space, newline, etc.)
\S	non-whitespace
\w	alphanumeric
\W	non-alphanumeric

# Regular Expression (Cont ... )

Mostly used functions from re module,

- `compile(pattern, flags=0)` – it compiles a regular expression pattern into a regular expression object, which can be used for matching using the `match` and `search` methods.
- `match(pattern, string, flags=0)` – if zero or more characters from the beginning of the string match, it returns a `Match` object, otherwise, it returns `None`.
- `search(pattern, string, flags=0)` – similar to `match()`, but it scans all the string, not only it's beginning.
- `sub(pattern, repl, string, count=0, flags=0)` - Return the string obtained by replacing the leftmost non-overlapping occurrences of the pattern in string by the replacement repl. repl can be either a string or a callable; if a string, backslash escapes in it are processed. If it is a callable, it's passed the match object and must return a replacement string to be used.

# Regular Expression (Cont ...)

```
#!/usr/bin/python3
import re

line = 'The    fox jumped    over      the          log'
pattern = re.compile('\s+')

line = re.sub(pattern, '_', line)
print (line)
```

The\_fox\_jumped\_over\_the\_log

How about following expression?

```
re.sub('\s{2,}', ' ', line)
```

remove\_multiple\_spaces.py

# Practice - Read Files & Regular Expression

- Declare “finish\_time”, “year” Lists - empty lists
- Read File 100\_meter.csv using Python (use **with** )
- Remove white-spaces with the help of regex
- Read Line by Line, Split Each lines at “,” (strings have split method)
- Append first element in year list & second element in finish\_time list
- Handle empty elements, elements with strings
- Create “finish\_time\_np” & “year\_np” as numpy arrays from “finish\_time” and “year” lists

# Assignment - 12

- Remove starting spaces in line
- Remove ending spaces in line
- How about removing all digits from line?

Develop Regular Expression to extract numbers (float, integers) from given text string.

The numbers can be in any of the following format

```
# '10.5', '-10.5', '- 10.5', '+ .2e10', ' 1.01e-2', '    1.01', '-.2', ' 456', '.123'
```

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# datetime Library

one\_datetime.py

```
[root@6879840ae648:/datascience/sessions/eleven# more 1.py
#!/usr/bin/python3
from datetime import datetime

now = datetime.now()
print('The dateformat: {}'.format(now))
print ('The year is = {}'.format(now.year))
print ('The month is = {}'.format(now.month))
print ('The day is = {} '.format(now.day))

#           datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
#           Y   M   d   h   m   s       ms
that_day = datetime(2017, 3, 1, 17, 9, 21, 832092)
print('The dateformat: {}'.format(that_day))
print ('The year is = {}'.format(that_day.year))
print ('The month is = {}'.format( that_day.month))
print ('The day is = {} '.format(that_day.day))
```

```
[root@6879840ae648:/datascience/sessions/eleven# ./1.py
The dateformat: 2017-03-31 14:52:04.730385
The year is = 2017
The month is = 3
The day is = 31
The dateformat: 2017-03-01 17:09:21.832092
The year is = 2017
The month is = 3
The day is = 1
root@6879840ae648:/datascience/sessions/eleven#
```

# Difference - datetime

delta.py

```
root@6879840ae648:/datascience/sessions/eleven# more delta.py
#!/usr/bin/python3
from datetime import datetime
#           datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
#           Y   M   d   h   m   s       ms           Y   M   d   h   m   s       ms
delta = datetime(2017, 3, 25, 18, 21, 1, 123000) - datetime (2016, 1, 1, 18, 51, 51, 123000)
print(delta)
print (delta.days)
print (delta.seconds)
```

```
root@6879840ae648:/datascience/sessions/eleven# ./delta.py
448 days, 23:29:10
448
84550
```

# timedelta

timedelta.py

```
[root@6879840ae648:/datascience/sessions/eleven# more timedelta.py
#!/usr/bin/python3
from datetime import datetime, timedelta

start = datetime(2017, 4, 1)
print (start)

new_s = start + timedelta (12)
print (new_s)

prev_s = start - 2 * timedelta(12)
print (prev_s)
```

```
[root@6879840ae648:/datascience/sessions/eleven# ./timedelta.py
2017-04-01 00:00:00
2017-04-13 00:00:00
2017-03-08 00:00:00
```

# strings and datetime

```
[root@6879840ae648:/datascience/sessions/eleven# more str_1.py
#!/usr/bin/python3
from datetime import datetime

d_stamp = datetime(2017, 3, 31)
print(d_stamp)
print (str(d_stamp))
print ("-----\n")

# Use of strftime - datetime to string
print (d_stamp.strftime('%Y/%m/%d'))
print (d_stamp.strftime('%Y-%m-%d'))
print (d_stamp.strftime('%Y, %d %m'))
print ("-----\n")

# Use of strptime - strings to datetime
d = '2017-03-31'

print (datetime.strptime(d, '%Y-%m-%d'))
print ("-----\n")

datestrs = ['01/Jan/2017', '01/Feb/2017', '01/Mar/2017', '01/Apr/2017']
print ([datetime.strptime(x, '%d/%b/%Y') for x in datestrs])
print ("-----\n")
```

str\_1.py

# strings and datetime (cont...)

```
[root@6879840ae648:/data-science/sessions/eleven# ./str_1.py
2017-03-31 00:00:00
2017-03-31 00:00:00
-----
2017/03/31
2017-03-31
2017, 31 03
-----
2017-03-31 00:00:00
-----
[datetime.datetime(2017, 1, 1, 0, 0), datetime.datetime(2017, 2, 1, 0, 0), datetime.datetime(2017, 3, 1, 0, 0), datetime.datetime(2017, 4, 1, 0, 0)]
```

# dateutil

## [pars\\_util.py](#)

```
[root@6879840ae648:/datascience/sessions/eleven# more pars_util.py
#!/usr/bin/python3
from dateutil.parser import parse

d1 = parse('2017-01-25')
print (type(d1))
print (d1)
print ("-----\n")

d2 = parse('Jan 01, 2017 10:30 AM')
print (type(d2))
print (d2)
print ("-----\n")

d3 = parse('01/01/2017', dayfirst=True)
print (type(d3))
print (d3)
print ("-----\n")

[root@6879840ae648:/datascience/sessions/eleven# ./pars_util.py
<class 'datetime.datetime'>
2017-01-25 00:00:00
-----
<class 'datetime.datetime'>
2017-01-01 10:30:00
-----
<class 'datetime.datetime'>
2017-01-01 00:00:00
```

# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

# Data Visualization - Plotting

`matplotlib.pyplot` is a collection of command style functions that helps draw graphs

## Types of Graph

- Bar Graphs
- Box and Whiskers (Boxplots)
- Frequency Distribution
- Histogram
- Line Graphs
- Pie Graphs
- Scatter Graphs
- Stemplots

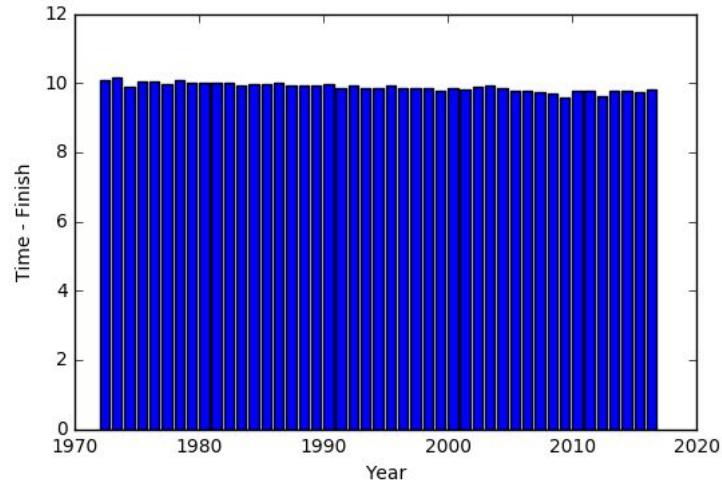
# Simple Bar Graph

```
import re
year = []
finish_time = []

with open('100_meter.csv', 'r') as f:
    for line in f:
        line = re.sub('\s+', '', line)
        line_elements = line.split(',')
        if line_elements[0] != '' and line_elements[0] != 'Year':
            year.append(int(line_elements[0]))
        if line_elements[1] != '' and line_elements[1] != 'Time':
            finish_time.append(float(line_elements[1]))

#print (year)
#print (finish_time)

import matplotlib.pyplot as plt
plt.bar(year, finish_time)
plt.xlabel('Year')
plt.ylabel('Time - Finish')
plt.show()
```



running\_simple\_bar.ipynb

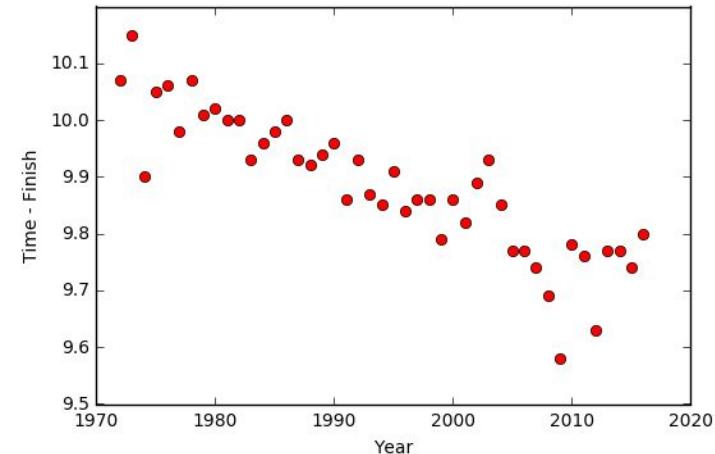
# Simple Plot

```
import re
year = []
finish_time = []

with open('100_meter.csv', 'r') as f:
    for line in f:
        line = re.sub('\s+', '', line)
        line_elements = line.split(',')
        if line_elements[0] != '' and line_elements[0] != 'Year':
            year.append(int(line_elements[0]))
        if line_elements[1] != '' and line_elements[1] != 'Time':
            finish_time.append(float(line_elements[1]))

#print (year)
#print (finish_time)

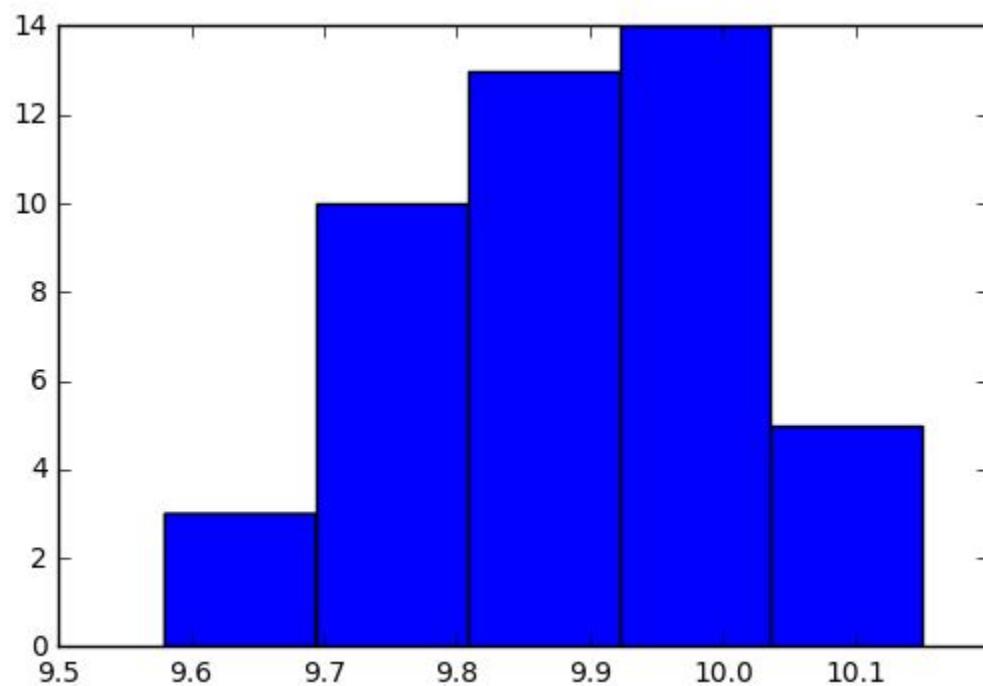
import matplotlib.pyplot as plt
plt.plot(year, finish_time, 'ro')
plt.xlabel('Year')
plt.ylabel('Time - Finish')
plt.show()
```



running\_simple\_plot.ipynb

# Histogram - Finish Time

```
plt.hist(finish_time, bins=5)  
plt.show()
```

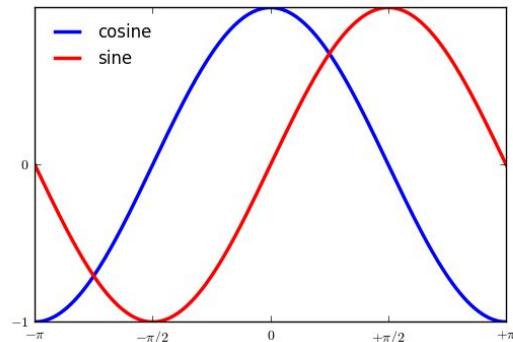


# Customizing Graphs

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C,S = np.cos(X), np.sin(X)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],[r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.yticks([-1, 0, +1],[r'$-1$', r'$0$', r'$+1$'])
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="--", label="cosine")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="--", label="sine")
plt.legend(loc='upper left', frameon=False)

plt.show()
```



# Google Stock Price

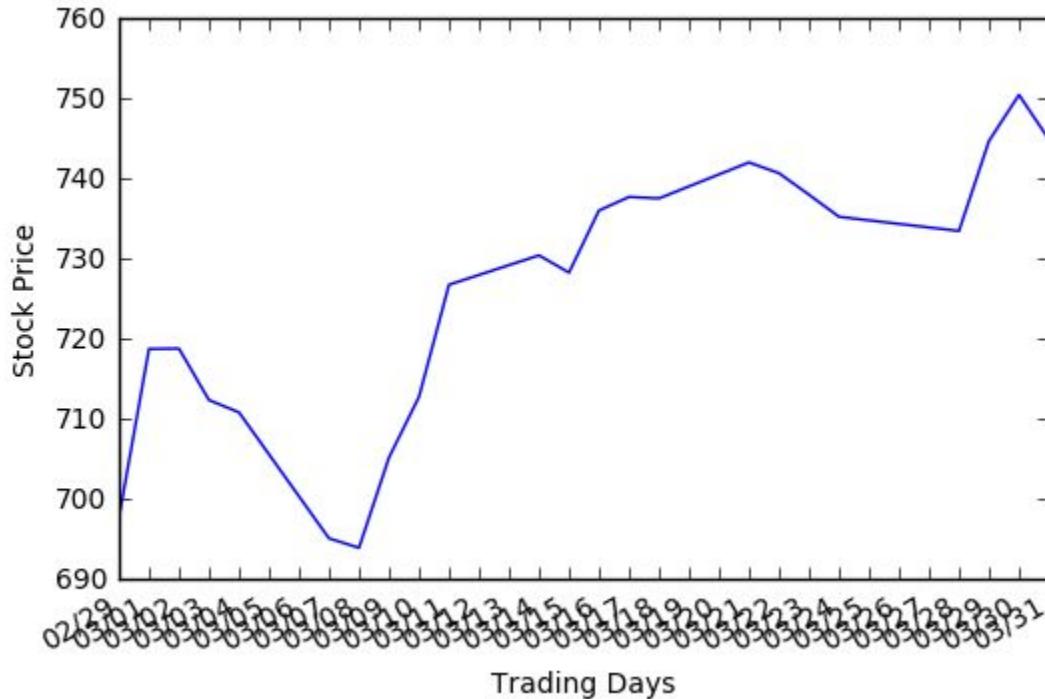
```
import datetime as dt
prices = []
dates = []
with open('GOOG.txt', "r") as f:
    for line in f:
        data = line.split(",")
        prices.append(data[1])
        day_y = data[0][0:4]
        day_m = data[0][4:6]
        day_d = data[0][6:8]
        dates.append(str(day_m) + '/' + str(day_d) + '/' + str(day_y))

days = [dt.datetime.strptime(d, '%m/%d/%Y').date() for d in dates]

import matplotlib.pyplot as plt
import matplotlib.dates as mdates

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator())
plt.xlabel("Trading Days")
plt.ylabel("Stock Price")
plt.plot(days,prices)
plt.gcf().autofmt_xdate()
plt.show()
```

# Google Stock Price (cont...)



# Topics

- **Handling Errors**
- **Lists**
- **Functions**
- **Modules**
- **Packages**
- **Strings**
- **Dictionaries**
- **Class**
- **Files - Read, Write**
- **Getting Data from Internet**
- **Regular Expression**
- **Datetime - Module**
- **Data Visualization**

**End - Two**

# Revisit Iterators

Python iterator objects are required to support two methods while following the iterator protocol.

- *iter* returns the iterator object itself. This is used in **for** and **in** statements.
- *next* method returns the next value from the iterator. If there is no more items to return then it should raise `StopIteration` exception.

# Iterator Revisits (Cont ... )

```
class EvenNumber(object):
    def __init__(self, low):
        if low % 2 != 0:
            self.current = low + 1
        else:
            self.current = low

    def __iter__(self):
        'Returns itself as an iterator object'
        return self

    def __next__(self):
        'Returns the next value till current'
        self.current += 2
```

```
5   6   7   8   9   10
2
4
6
```

test\_counter\_evennumber.py

EvenNumber.py

Counter.py

```
class Counter(object):
    def __init__(self, low, high):
        self.current = low
        self.high = high

    def __iter__(self):
        'Returns itself as an iterator object'
        return self

    def __next__(self):
        'Returns the next value till current is lower than high'
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
```

# Numpy: Scientific Library for Data Science!

Numpy - Numerical Python

- ndarray: multi-dimensional array proving vectorized arithmetic operations
- Mathematical operations on entire array without having to use loops
- Tools to write, read data from disk, tools to work with memory-mapped files
- Linear Algebra, Random Number Generation, and Fourier Transform capabilities.
- Tools to integrate code with lower level languages like C, C++

# Numpy (cont..)

Focus is on :

- Fast Vectorized Array Operations for data munging & Cleaning, subsetting, and filtering, transformation and any kind of computation.
- Common array operations - sorting, unique & set operations
- Statistics operations - aggregation, summarizing data
- Merging, joining data sets together
- Play with conditional logic (if-else)
- Grouping & manipulating

# Data Used

2012 - 100 meter Olympics Result

	Time	Height (Inches)	Weight (Pounds)	Country
Usain Bolt	9.63	76.77	209.439	JAM
Yohan Blake	9.75	70.86	167.551	JAM
Justin Gatlin	9.79	72.83	182.984	USA
Tyson Gay	9.8	70.07	165.347	USA
Ryan Bailey	9.88	75.98	216.053	USA
Churandy Martina	9.94	70.07	163.142	NED
Richard Thompson	9.98	70.01	176.37	TTO
Asafa Powell	11.99	70.8	191.802	JAM

100\_runners.csv

# Revisit Lists - Count BMI List Way

```
#!/usr/bin/python3
# height in inches, weight in pounds
height = [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8]
weight = [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]

m_height = [x * 0.0254 for x in height]
k_weight = [x * 0.453592 for x in weight]

#print (m_height)
#print (k_weight)

# kg/m^2                                         bmi_lists.py

msqr_height = [x * x for x in m_height]
#print (msqr_height)

bmi = [k_weight[i]/msqr_height[i] for i in range(len(msqr_height))]
print (bmi)
```

```
#!/usr/bin/python3
```

```
import numpy as np
```

Importing Numpy Library

```
# height in inches, weight in pounds
```

```
height = [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8]
```

```
weight = [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]
```

```
np_height = np.array(height)
```

```
np_weight = np.array(weight)
```

Creating Numpy Array - from List

```
np_height_meters = np_height * 0.0254
```

```
np_weight_kgs = np_weight * 0.453592
```

Multiply each element of Numpy Array

```
#print (m_height)
```

```
#print (k_weight)
```

```
# kg/m^2
```

```
np_height_meters_sqr = np_height_meters ** 2
```

```
#print (msqr_height)
```

```
bmi = np_weight_kgs / np_height_meters_sqr
```

Square each element

```
print (bmi)
```

Divide each element

bmi\_numpy.py

# Numpy - A few Observations

```
[>>> import numpy as np
[>>> my_numpy = np.array([11.00, 12, 'numpy', True, False])
[>>> type(my_numpy)
<class 'numpy.ndarray'>
[>>> my_numpy
array(['11.0', '12', 'numpy', 'True', 'False'],
      dtype='<U32') ←
[>>> my_list = [11.00, 12, 'numpy', True, False]
[>>> type(my_list)
<class 'list'>
[>>> my_list + my_list ←
[11.0, 12, 'numpy', True, False, 11.0, 12, 'numpy', True, False] ←
[>>> my_numpy + my_numpy
Traceback (most recent call last): ←
  File "<stdin>", line 1, in <module>
TypeError: ufunc 'add' did not contain a loop with signature matching types dtype('<U32') dtype('<U32') dtype('<U32')
[>>> my_numpy_n = np.array([1, 2, 3])
[>>> type(my_numpy_n)
<class 'numpy.ndarray'>
[>>> my_numpy_n
array([1, 2, 3]) ←
[>>> my_numpy_n + my_numpy_n
array([2, 4, 6]) ←
>>> █
```

# Numpy - Subsetting

```
>>> import numpy as np
>>> bmi = np.array([ 24.98460153,  23.46079266,  24.25439523,  23.67718338,  26.31235232, 23.36143414,  25.29895093,  26.9
005])
>>> type(bmi)
<class 'numpy.ndarray'>
>>> bmi[1]
23.460792659999999
>>> bmi[0]
24.984601529999999
>>> bmi[11]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index 11 is out of bounds for axis 0 with size 8
>>> bmi > 24
array([ True, False,  True, False,  True, False,  True,  True], dtype=bool)
>>> bmi > 23
array([ True,  True,  True,  True,  True,  True,  True,  True], dtype=bool)
>>> bmi > 22
array([ True,  True,  True,  True,  True,  True,  True,  True], dtype=bool)
>>> bmi[bmi > 22]
array([ 24.98460153,  23.46079266,  24.25439523,  23.67718338,
       26.31235232,  23.36143414,  25.29895093,  26.90200005])
>>> bmi[bmi > 24]
array([ 24.98460153,  24.25439523,  26.31235232,  25.29895093,  26.90200005])
>>> bmi[bmi > 25]
array([ 26.31235232,  25.29895093,  26.90200005])
>>> bmi[bmi > 26]
array([ 26.31235232,  26.90200005])
>>> bmi[bmi > 26.9]
array([ 26.90200005])
>>>
```

# Practice - One (Create Numpy Array)

- Import the `numpy` package as `np`
- Use `np.array()` to create a Numpy array from `m_runners`. Name this array `np_m_runners` Print out the `type` of `np_m_runners` to check that you got it right.

```
#      one.py

# This is 2012 Olympics Game 100 meter speed result
m_runners = [9.63, 9.75, 9.79, 9.80, 9.88, 9.94, 9.98, 11.99]

# Import the numpy package as np

# Create a Numpy array from m_runners: np_m_runners

# Print out type of np_m_runners

# Print np_m_runners
```

# Practice Two - Count Height

- Create a Numpy array from height. Name this new array np\_height.
- Print np\_height.
- Multiply np\_height with 0.0254 to convert all height measurements from inches to meters. Store the new values in a new array, np\_height\_m.
- Print out np\_height\_m

```
#  #two.py
# Import numpy
import numpy as np

height = [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8]
# Create a Numpy array from height: np_height

# Print out np_height

# Convert np_height to m: np_height_m
# You need to multiply by 0.0254

# Print np_height_m
```

# Practice Three - Count bmi

- Create a Numpy array from the weight list, height list with the correct units. Call them - np\_weight\_kg & np\_height\_m
- Use np\_height\_m and np\_weight\_kg to calculate the BMI of each player. Use the following equation:  $BMI = \text{weight(kg)} / \text{height(m)}^2$
- Save the resulting numpy array as bmi.
- Print out bmi.

# three.py

```
# height in inches & weight in pounds
height = [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8]
weight = [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]
# Import numpy
# Create array from height with correct units: np_height_m - multiplication by 0.0254
# Create array from weight with correct units: np_weight_kg - multiplication by 0.453592
# Calculate the BMI: bmi - use kg/m^2
# Print out bmi
```

# Practice Four - Lightweight

- Create a boolean Numpy array: the element of the array should be True if the corresponding runners BMI is below 23. You can use the < operator for this. Name the array light.
- Print the array light.
- Print out a Numpy array with the BMIs of all runners whose BMI is below 23. Use light inside square brackets to do a selection on the bmi array.

```
# four.py
#height and weight are available as a regular lists
height = [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8]
weight = [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]
# Import numpy

# Calculate the BMI: bmi
np_height_m = np.array(height) * 0.0254
np_weight_kg = np.array(weight) * 0.453592
bmi = np_weight_kg / np_height_m ** 2

# Create the light array
# Print out light
# Print out BMIs of all runners whose BMI is below 23
```

# Practice Five - Match the Following

1

```
>>> demo = np.array([1, 2, 3, 4, 5, 6, 7, 8])  
>>> demo>2
```

2

```
>>> demo = np.array([1, 2, 3, 4, 5, 6, 7, 8])  
>>> demo[demo >5]
```

3

```
>>> a = np.array([1, 2, 3, True])  
>>> b = np.array([False, 2, 3, 4])  
>>> a + b
```

4

```
>>> demo = np.array([1, 2, 3, 4, 5, 6, 7, 8])  
>>> demo[demo < 1]
```

A

```
array([6, 7, 8])
```

B

```
array([1, 4, 6, 5])
```

C

```
array([], dtype=int64)
```

D

```
array([False, False,  True,  True,  True,  True,  True,  True], dtype=bool)
```

# Practice Six - Subsetting

- Subset np\_weight: print out the element at index 5.
- Print out a sub-array of np\_height: It contains the elements at index 3 up to and including index 6

```
#six.py
height = [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8]
weight = [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]
```

# N-Dimensional Arrays Numpy

- ndarray - generic multidimensional container for homogeneous data
- shape - tuple indicating size of each dimension
- dtype - an object indicating data type of the array

```
[>>> import numpy as np
[>>> data = ([0.1000, 0.1200, 0.1300, 0.1400], [0.2100, 0.2200, 0.2300, 0.2400])
[>>> n_data = np.array(data) <----- red arrow
[>>> n_data.shape <----- red arrow      2 rows, 4 columns
(2, 4)
[>>> n_data.dtype
dtype('float64') <----- red arrow]
```

# Additional Functions to Create ndarrays

- zeros
- ones
- empty

```
>>> o = np.zeros(5)
>>> print(o)
[ 0.  0.  0.  0.  0.]
>>> o.dtype
dtype('float64')
>>> o.size
5
>>> one = np.ones(7)
>>> print(one)
[ 1.  1.  1.  1.  1.  1.  1.]
>>> one.dtype
dtype('float64')
>>> one.shape
(7,)
>>> one.size
7
>>>
```

```
>>> e = np.empty(9)
>>> print(e)
[ 6.90268625e-310  6.90268625e-310  2.03650144e-316  6.90264011e-310
 2.37151510e-322  2.37151510e-322  2.03589354e-316  6.90268625e-310
 2.03650381e-316]
>>> e.dtype
dtype('float64')
>>>
```

# Additional Functions to Create ndarrays

```
[>>> np.zeros((2, 3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
[>>> np.zeros((2, 3, 1))
array([[[ 0.],
       [ 0.],
       [ 0.]],

      [[ 0.],
       [ 0.],
       [ 0.]]])
[>>> np.ones((2, 3, 1))
array([[[ 1.],
       [ 1.],
       [ 1.]],

      [[ 1.],
       [ 1.],
       [ 1.]]])
[>>> np.ones((2, 3))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])]
```

```
[>>> np.empty((3, 2, 1))
array([[[ 0.],
       [ 0.]],

      [[ 0.],
       [ 0.]],

      [[ 0.],
       [ 0.]]])
[>>> np.empty((3, 2))
array([[ 0.,  0.],
       [ 0.,  0.]])
[>>> np.empty((2, 5))
array([[ 6.90268625e-310,   6.90268625e-310,   2.87311426e-317,
         1.97626258e-323,   1.85541175e-316],
       [ 6.90268034e-310,   6.90267919e-310,   0.00000000e+000,
         0.00000000e+000,   0.00000000e+000]])
[>>> np.empty((2, 2))
array([[ 6.90268625e-310,   1.88584462e-316],
       [ 6.90268053e-310,   5.30563263e-317]])
```

>>> █

# Operation between Arrays and Scalars

```
[>>> my_array = np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
[>>> my_array.dtype
dtype('float64')
[>>> my_array * my_array      ◀-----  
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
[>>> my_array - my_array      ◀-----  
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
[>>> 1/my_array               ◀-----  
array([[ 1.          ,  0.5         ,  0.33333333],
       [ 0.25        ,  0.2         ,  0.16666667]])
[>>> my_array ** 2            ◀-----  
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
[>>>
```

# Basic Indexing and Slicing

```
>>> exp_array = np.arange(10)
>>> exp_array
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> exp_array[5]
5
>>> exp_array[0]
0
>>> exp_array[2:5]
array([2, 3, 4])
>>> exp_array[2:5]=22
>>> exp_array
array([ 0,  1, 22, 22, 22,  5,  6,  7,  8,  9])
>>>
```

# Basic Indexing and Slicing (cont...)

```
>>> exp_array = np.arange(10)
>>> exp_array[5]
5
>>> exp_array[2:5]
array([2, 3, 4])
>>> exp_array[2:5] = 29
>>> exp_array
array([ 0,  1, 29, 29, 29,  5,  6,  7,  8,  9])
>>> my_slice = exp_array[2:5]    ←----- red dotted line
>>> my_slice
array([29, 29, 29])
>>> my_slice[1] = 44           ←----- red arrow
>>> my_slice
array([29, 44, 29])
>>> exp_array
array([ 0,  1, 29, 44, 29,  5,  6,  7,  8,  9])
>>> my_slice[:] = 400          ←----- red dotted line
>>> exp_array
array([ 0,  1, 400, 400, 400,  5,  6,  7,  8,  9])
>>>
```

# Basic Indexing and Slicing (cont..)

```
[>>> import numpy as np  
[>>> arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
[>>> arr2d[2]  
array([7, 8, 9])  
[>>> arr2d[0][2]  
3  
[>>> ]
```

			axis 1	
			0      1      2	
axis 0	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

# Basic Indexing and Slicing (cont..)

1	2	3
4	5	6
7	8	9

```
>>> arr2d[:2,1:]  
array([[2, 3],  
      [5, 6]])
```

1	2	
4	5	
7	8	

```
[>>> arr2d[:,2]  
array([[1, 2],  
      [4, 5],  
      [7, 8]])
```

7	8	9

```
>>> arr2d[2]  
array([7, 8, 9])  
>>> arr2d[2, :]  
array([7, 8, 9])  
>>> arr2d[2:, :]  
array([[7, 8, 9]])
```

4	5	

```
[>>> arr2d[1,:2]  
array([4, 5])  
>>> arr2d[1:2,:2]  
array([[4, 5]])
```

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**
- **Lists Revisits**
- **Handling Errors**
- **Functions**
- **Modules**
- **Packages**
- **Dictionaries**
- **Classes**
- **Iterators**
- **String Formatting**
- **Regular Expressions**

# Topics

- **Hello Python**
- **Comments**
- **Mathematics**
- **Variables**
- **Data Types**
- **Making Decisions**
- **Strings**
- **Python Lists**
- **For, While - Loops**
- **Lists Revisits**
- **Handling Errors**
- **Functions**
- **Modules**
- **Packages**
- **Dictionaries**
- **Classes**
- **Iterators**
- **String Formatting**
- **Regular Expressions**

# Pandas - Library for Data Analysis

# Pandas Data Structures

- Series
- DataFrame

Both of them are built on the top of Numpy (... so are fast)

# Series

- A Series is a one-dimensional object similar to an array, list, or column in a table.
- It will assign a labeled index to each item in the Series.
- By default, each item will receive an index label from 0 to N, where N is the length of the Series minus one.

```
[>>> import pandas as pd
[>>> s = pd.Series(['Usain Bolt', 9.63, 'Yohan Blake', 9.75, 'Justin Gatlin', 9.79])
[>>> s
0      Usain Bolt
1          9.63
2      Yohan Blake
3          9.75
4      Justin Gatlin
5          9.79
dtype: object
>>> ]
```

# Series (Cont...)

```
>>> s = pd.Series(['Usain Bolt', 9.63, 'Yohan Blake', 9.75, 'Justin Gatlin', 9.79], index=['A', 'B', 'C', 'D', 'E', 'F'])
>>> s
A      Usain Bolt
B          9.63
C      Yohan Blake
D          9.75
E    Justin Gatlin
F          9.79
dtype: object
>>>
```

You can also use Dictionary to create Series

```
>>> runners = {'Usain Bolt': 9.63, 'Yohan Blake': 9.75, 'Justin Gatlin': 9.79}
>>> runners
{'Yohan Blake': 9.75, 'Usain Bolt': 9.63, 'Justin Gatlin': 9.79}
>>> r = pd.Series(runners)
>>> r
Justin Gatlin    9.79
Usain Bolt     9.63
Yohan Blake     9.75
dtype: float64
>>>
```

# Series (Cont...)

Accessing Elements are Easy

```
[>>> r['Usain Bolt']
9.6300000000000008
[>>> r['Yohan Blake']
9.75
>>> ]
```

```
[>>> r[['Yohan Blake', 'Yohan Blake']]
Yohan Blake    9.75
Yohan Blake    9.75
dtype: float64
[>>>
[>>> r[['Yohan Blake', 'Usain Bolt']]
Yohan Blake    9.75
Usain Bolt     9.63
dtype: float64
>>> ]
```

```
>>> runners = {'Usain Bolt': 9.63, 'Yohan Blake': 9.75, 'Justin Gatlin': 9.79, 'Tyson Gay': 9.8, 'Ryan Bailey': 9.88, 'Churandy Martina': 9.94, 'Richard T
ompson': 9.98, 'Asafa Powell': 11.99}
>>>
>>> r = pd.Series(runners)
>>> r
Asafa Powell      11.99
Churandy Martina  9.94
Justin Gatlin     9.79
Richard Thompson   9.98
Ryan Bailey       9.88
Tyson Gay         9.80
Usain Bolt        9.63
Yohan Blake       9.75
dtype: float64
>>> r[r>10]
Asafa Powell      11.99
dtype: float64
>>> r[r<9.8]
Justin Gatlin     9.79
Usain Bolt        9.63
Yohan Blake       9.75
dtype: float64
>>> ]
```

# Series (Cont...)

```
>>> r <9.8
Asafa Powell      False
Churandy Martina False
Justin Gatlin    True
Richard Thompson False
Ryan Bailey       False
Tyson Gay         False
Usain Bolt        True
Yohan Blake       True
dtype: bool
>>> my_requirement = r<9.8
>>> r[my_requirement]
Justin Gatlin    9.79
Usain Bolt       9.63
Yohan Blake      9.75
dtype: float64
>>> █
```

Actually  $r < 9.8$  or  $r > 10$   
are series with True & False values.

Those can be passed to r - returning  
corresponding “True” value Series.

# Series(Cont...)

One can change values on the fly

You can also check the presence of the value

```
>>> print ('Ironman' in r)
False
>>> print ('Superman' in r)
True
>>> print ('Milkha Singh' in r)
False
>>>
```

```
>>> r[['Superman']]
Superman      NaN
dtype: float64
>>> r['Superman']=5.0
>>> r['Batman'] = 6.0
>>> r
Asafa Powell      11.99
Churandy Martina   9.94
Justin Gatlin      9.79
Richard Thompson    9.98
Ryan Bailey        9.88
Tyson Gay          9.80
Usain Bolt         9.63
Yohan Blake        9.75
Superman           5.00
Batman              6.00
dtype: float64
>>> r[r<9] = 1.0
>>> r
Asafa Powell      11.99
Churandy Martina   9.94
Justin Gatlin      9.79
Richard Thompson    9.98
Ryan Bailey        9.88
Tyson Gay          9.80
Usain Bolt         9.63
Yohan Blake        9.75
Superman           1.00
Batman              1.00
dtype: float64
>>>
```

# Series (Cont...)

## Mathematics

```
>>> r * r
Asafa Powell      143.7601
Churandy Martina  98.8036
Justin Gatlin     95.8441
Richard Thompson   99.6004
Ryan Bailey        97.6144
Tyson Gay          96.0400
Usain Bolt         92.7369
Yohan Blake        95.0625
Superman           1.0000
Batman              1.0000
dtype: float64
```

```
>>> r / 2
Asafa Powell      5.995
Churandy Martina  4.970
Justin Gatlin     4.895
Richard Thompson   4.990
Ryan Bailey        4.940
Tyson Gay          4.900
Usain Bolt         4.815
Yohan Blake        4.875
Superman           0.500
Batman              0.500
dtype: float64
```

```
>>> r + r
Asafa Powell      23.98
Churandy Martina  19.88
Justin Gatlin     19.58
Richard Thompson   19.96
Ryan Bailey        19.76
Tyson Gay          19.60
Usain Bolt         19.26
Yohan Blake        19.50
Superman           2.00
Batman              2.00
dtype: float64
```

```
>>> np.square(r)
Asafa Powell      143.7601
Churandy Martina  98.8036
Justin Gatlin     95.8441
Richard Thompson   99.6004
Ryan Bailey        97.6144
Tyson Gay          96.0400
```

# Series (Cont...)

## Mathematics

```
[>>> new_r = pd.Series({'Lalita Baber': 15.0, 'O P Jaisha': 16.0, 'Kavita Raut': 20.0
[>>> new_r
Kavita Raut      20.0
Lalita Baber    15.0
O P Jaisha      16.0
dtype: float64
```

```
[>>> r + new_r
Asafa Powell      NaN
Batman            NaN
Churandy Martina NaN
Justin Gatlin     NaN
Kavita Raut       NaN
Lalita Baber      NaN
O P Jaisha        NaN
Richard Thompson   NaN
Ryan Bailey        NaN
Superman          NaN
Tyson Gay          NaN
Usain Bolt         NaN
Yohan Blake        NaN
dtype: float64
```

# Series (Cont...)

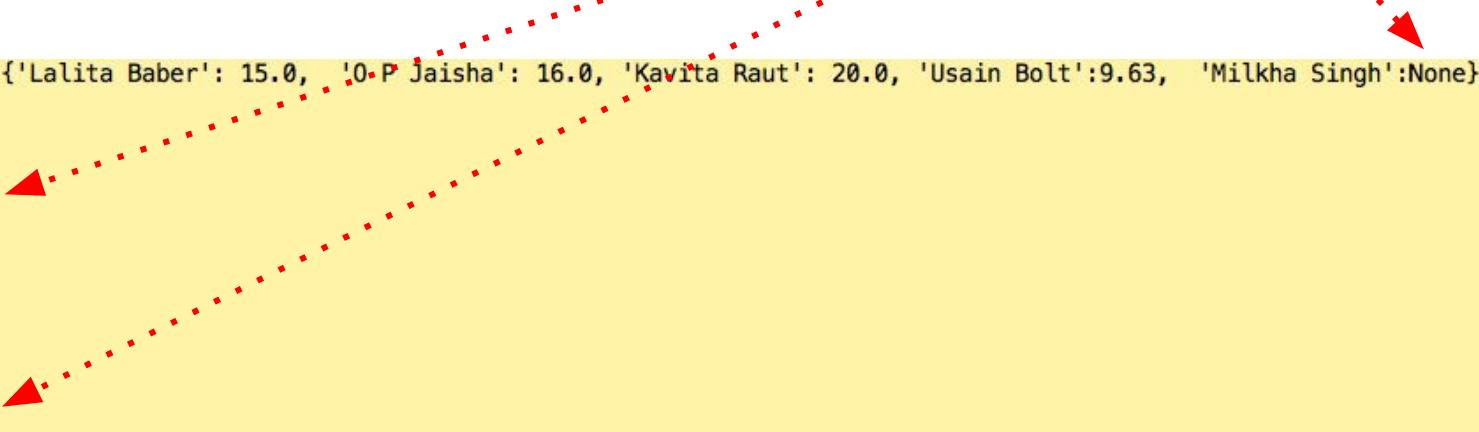
```
|>>> new_r = pd.Series({'Lalita Baber': 15.0, 'O P Jaisha': 16.0, 'Kavita Raut': 20.0, 'Usain Bolt':9.63})  
|>>> r + new_r  
Asafa Powell      NaN  
Batman            NaN  
Churandy Martina NaN  
Justin Gatlin    NaN  
Kavita Raut      NaN  
Lalita Baber     NaN  
O P Jaisha       NaN  
Richard Thompson  NaN  
Ryan Bailey       NaN  
Superman          NaN  
Tyson Gay         NaN  
Usain Bolt        19.26  
Yohan Blake       NaN  
dtype: float64  
>>>
```

```
|>>> r  
Asafa Powell      11.99  
Churandy Martina  9.94  
Justin Gatlin     9.79  
Richard Thompson   9.98  
Ryan Bailey        9.88  
Tyson Gay          9.80  
Usain Bolt         9.63  
Yohan Blake        9.75  
Superman           1.00  
Batman             1.00  
dtype: float64
```

# Series (Cont ...)

isnull & notnull instance methods

```
>>> new_r = pd.Series({'Lalita Baber': 15.0, 'O P Jaisha': 16.0, 'Kavita Raut': 20.0, 'Usain Bolt':9.63, 'Milkha Singh':None})  
>>> new_r.isnull()  
Kavita Raut    False  
Lalita Baber  False  
Milkha Singh   True  
O P Jaisha    False  
Usain Bolt    False  
dtype: bool  
>>> new_r.notnull()  
Kavita Raut    True  
Lalita Baber  True  
Milkha Singh   False  
O P Jaisha    True  
Usain Bolt    True  
dtype: bool  
>>>
```



# Series - Practice

- Use Fruits Data - create dictionary to work on Protein details. Use Fruits Names as Keys and values as Protien. Name dictionary as - “fruits”
- Create Series from “fruits” - call it “s\_fruits”
- Perform
  - >, multiplication, np.square, addition, set values on the fly.
- Add “Onion” as key, with value= None, Recreate “s\_fruits” Series
  - Observe “NaN” value
- Create new dictionary - “new\_fruits” with keys “Raisins”, “Onion” (use same values of Protien as above)
- Perform “s\_fruits” + “new\_fruits”

# DataFrame

"tabular" data: a data structure representing cases (rows), each of which consists of a number of observations or measurements (columns).

Player Name	Time	Height (Inches)	Weight (Pounds)	Country
Usain Bolt	9.63	76.77	209.439	JAM
Yohan Blake	9.75	70.86	167.551	JAM
Justin Gatlin	9.79	72.83	182.984	USA
Tyson Gay	9.8	70.07	165.347	USA
Ryan Bailey	9.88	75.98	216.053	USA
Churandy Martina	9.94	70.07	163.142	NED
Richard Thompson	9.98	70.01	176.37	TTO
Asafa Powell	11.99	70.8	191.802	JAM

# DataFrame

```
[>>> data = {'name': ['Usain Bolt', 'Yohan Blake', 'Justin Gatlin', 'Tyson Gay', 'Ryan Bailey', 'Churandy Martina', 'Richard Thompson', 'Asafa Powell'],
[...     'speed': [9.63, 9.75, 9.79, 9.8, 9.88, 9.94, 9.98, 11.99],
[...     'height': [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8],
[...     'weight': [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]}
>>>
```

```
[>>> data
{'speed': [9.63, 9.75, 9.79, 9.8, 9.88, 9.94, 9.98, 11.99], 'name': ['Usain Bolt', 'Yohan Blake', 'Justin Gatlin', 'Tyson Gay', 'Ryan Bailey', 'Churandy Martina', 'Richard Thompson', 'Asafa Powell'], 'height': [76.77, 70.86, 72.83, 70.07, 75.98, 70.07, 70.01, 70.8], 'weight': [209.439, 167.551, 182.984, 165.347, 216.053, 163.142, 176.37, 191.802]}
>>>
```

```
[>>> runners = pd.DataFrame(data, columns=["name", "speed", "height", "weight"])
>>> runners
      name    speed   height   weight
0  Usain Bolt    9.63    76.77  209.439
1  Yohan Blake    9.75    70.86  167.551
2  Justin Gatlin    9.79    72.83  182.984
3    Tyson Gay    9.80    70.07  165.347
4   Ryan Bailey    9.88    75.98  216.053
5 Churandy Martina    9.94    70.07  163.142
6 Richard Thompson    9.98    70.01  176.370
7   Asafa Powell   11.99    70.80  191.802
>>> ■
```

# DataFrame

```
|>>> food = pd.read_csv("food.csv")
|>>> food
```

	Food	Index	Calories	Cholesterol	Total_Fat	Sodium	\
0	Frozen Broccoli	1	73.8	0.0	0.8	68.2	
1	Carrots,Raw	2	23.7	0.0	0.1	19.2	
2	Celery, Raw	3	6.4	0.0	0.1	34.8	
3	Frozen Corn	4	72.2	0.0	0.6	2.5	
4	Lettuce,Iceberg,Raw	5	2.6	0.0	0.0	1.8	
5	Peppers, Sweet, Raw	6	20.0	0.0	0.1	1.5	
6	Potatoes, Baked	7	171.5	0.0	0.2	15.2	
7	Tofu	8	88.2	0.0	5.5	8.1	
8	Roasted Chicken	9	277.4	129.9	10.8	125.6	
9	Spaghetti W/ Sauce	10	358.2	0.0	12.3	1237.1	
10	Tomato,Red,Ripe,Raw	11	25.8	0.0	0.4	11.1	
11	Apple,Raw,W/Skin	12	81.4	0.0	0.5	0.0	
12	Banana	13	104.9	0.0	0.5	1.1	
13	Grapes	14	15.1	0.0	0.1	0.5	
14	Kiwifruit,Raw,Fresh	15	46.4	0.0	0.3	3.8	
15	Oranges	16	61.6	0.0	0.2	0.0	
16	Bagels	17	78.0	0.0	0.5	151.4	
17	Wheat Bread	18	65.0	0.0	1.0	134.5	
18	White Bread	19	65.0	0.0	1.0	132.5	
19	Oatmeal Cookies	20	81.0	0.0	3.3	68.9	
20	Apple Pie	21	67.2	0.0	2.1	75.4	

# DataFrame

```
|>>> food.head()
      Food Index Calories Cholesterol Total_Fat Sodium \
0   Frozen Broccoli    1     73.8        0.0       0.8   68.2
1   Carrots,Raw       2     23.7        0.0       0.1   19.2
2   Celery, Raw       3      6.4        0.0       0.1   34.8
3   Frozen Corn       4     72.2        0.0       0.6   2.5
4 Lettuce,Iceberg,Raw  5      2.6        0.0       0.0   1.8

   Carbohydrates Dietary_Fiber Protein   Vit_A   Vit_C Calcium Iron \
0            13.6          8.5     8.0  5867.4  160.2  159.0  2.3
1             5.6          1.6     0.6 15471.0    5.1   14.9  0.3
2             1.5          0.7     0.3    53.6    2.8   16.0  0.2
3            17.1          2.0     2.5  106.6    5.2    3.3  0.3
4             0.4          0.3     0.2    66.0    0.8    3.8  0.1

   Price/Serving ($)
0            0.16
1            0.07
2            0.04
3            0.18
4            0.02
|>>> food.tail(1)
      Food Index Calories Cholesterol Total_Fat Sodium \
63 Beanbancn Soup,W/Watr    64     172.0        2.5       5.9  951.3

   Carbohydrates Dietary_Fiber Protein   Vit_A   Vit_C Calcium Iron \
63           22.8          8.6     7.9  888.0    1.5   81.0  2.0

   Price/Serving ($)
63            0.67
|>>> food.tail(3)
      Food Index Calories Cholesterol Total_Fat Sodium \
61 New E Clamchwd,W/Mlk    62    163.7        22.3      6.6  992.0
62 Crm Mshrm Soup,W/Mlk    63    203.4        19.8     13.6 1076.3
```

- head()
- tail()
- info()
- describe()
- dtype
- columns

# DataFrame

```
>>> food.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64 entries, 0 to 63
Data columns (total 14 columns):
Food                 64 non-null object
Index                64 non-null int64
Calories              64 non-null float64
Cholesterol           64 non-null float64
Total_Fat             64 non-null float64
Sodium                64 non-null float64
Carbohydrates         64 non-null float64
Dietary_Fiber          64 non-null float64
Protein               64 non-null float64
Vit_A                 64 non-null float64
Vit_C                 64 non-null float64
Calcium                64 non-null float64
Iron                  64 non-null float64
Price/Serving ($)      64 non-null float64
dtypes: float64(12), int64(1), object(1)
memory usage: 7.1+ KB
>>>
```

```
>>> runners.describe()
           speed      height      weight
count    8.000000   8.000000   8.000000
mean    10.095000  72.173750 184.086000
std     0.773656  2.756555  20.171263
min     9.630000  70.010000 163.142000
25%    9.780000  70.070000 167.000000
50%    9.840000  70.830000 179.677000
75%    9.950000  73.617500 196.211250
max   11.990000  76.770000 216.053000
>>>
```

# Databases - SQLite

```
[root@6879840ae648:/datascience/sessions/data# sqlite3  
SQLite version 3.11.0 2016-02-15 17:29:24  
Enter ".help" for usage hints.  
Connected to a transient in-memory database.  
Use ".open FILENAME" to reopen on a persistent database.  
sqlite>  
sqlite>  
sqlite>  
sqlite> .help  
.backup ?DB? FILE          Backup DB (default "main") to FILE  
.bail on|off                Stop after hitting an error. Default OFF  
.binary on|off              Turn binary output on or off. Default OFF  
.changes on|off             Show number of rows changed by SQL  
.clone NEWDB                Clone data into NEWDB from the existing database  
.databases                  List names and files of attached databases  
.dbinfo ?DB?                Show status information about the database  
.dump ?TABLE? ...           Dump the database in an SQL text format  
                            If TABLE specified, only dump tables matching  
                            LIKE pattern TABLE.  
.echo on|off                 Turn command echo on or off  
.eqp on|off                  Enable or disable automatic EXPLAIN QUERY PLAN  
.exit                       Exit this program  
.explain ?on|off|auto?      Turn EXPLAIN output mode on or off or to automatic
```

# Databases - SQLite (Cont..)

```
sqlite> .schema
sqlite> CREATE TABLE presidents (
...> id int primary key NOT NULL,
...> name char(100) NOT NULL,
...> age int NOT NULL);
sqlite> .schema
CREATE TABLE presidents (
id int primary key NOT NULL,
name char(100) NOT NULL,
age int NOT NULL);
sqlite> select * from presidents;
sqlite> INSERT INTO presidents (id, name, age) VALUES(1, 'Donalt T', 74);
sqlite> select * from presidents;
1|Donalt T|74
sqlite> select name from presidents;
Donalt T
sqlite> INSERT INTO presidents (id, name, age) VALUES(2, 'Barack O', 54);
sqlite> select * from presidents;
1|Donalt T|74
2|Barack O|54
sqlite> select * from presidents where id=2;
2|Barack O|54
sqlite>
```

# Database - Python Module

```
[>>> conn = sqlite3.connect('example.db')
[>>> type(conn)
<class 'sqlite3.Connection'>
[>>> c = conn.cursor()    ←
[>>> type(c)
<class 'sqlite3.Cursor'>
[>>> c.execute('''CREATE TABLE stocks
[...           (date text, trans text, symbol text, qty real, price real)''')
<sqlite3.Cursor object at 0x7f54c2ed9e30>
[>>> c.execute("INSERT INTO stocks VALUES ('2017-01-05','BUY','GOOG',100,35.14)") ←
<sqlite3.Cursor object at 0x7f54c2ed9e30>
[>>> conn.commit()
[>>> conn.close()    ←
[>>> type(c)
<class 'sqlite3.Cursor'>
[>>> █
```

# Database Python Module

```
root@6879840ae648:/datascience/sessions/data# python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sqlite3
>>> conn = sqlite3.connect('example.db')
>>> c = conn.cursor()
>>> symbol = 'GOOG'
>>> c.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)
<sqlite3.Cursor object at 0x7f9d737f8e30>
>>> print(c.fetchone())
('2017-01-05', 'BUY', 'GOOG', 100.0, 35.14)
>>>
>>>
>>> t = ('GOOG',)
>>> c.execute('SELECT * FROM stocks WHERE symbol=?', t)
<sqlite3.Cursor object at 0x7f9d737f8e30>
>>> print(c.fetchone())
('2017-01-05', 'BUY', 'GOOG', 100.0, 35.14)
>>> purchases = [ ('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
...                 ('2006-04-05', 'BUY', 'MSFT', 1000, 72.00),
...                 ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
...
...             ]
>>>
>>> c.executemany('INSERT INTO stocks VALUES (?,?,?,?,?)', purchases)
<sqlite3.Cursor object at 0x7f9d737f8e30>
>>> for row in c.execute('SELECT * FROM stocks ORDER BY price'):
...     print(row)
...
('2017-01-05', 'BUY', 'GOOG', 100.0, 35.14)
('2006-03-28', 'BUY', 'IBM', 1000.0, 45.0)
('2006-04-06', 'SELL', 'IBM', 500.0, 53.0)
('2006-04-05', 'BUY', 'MSFT', 1000.0, 72.0)
>>>
```

# create\_db.py

```
#!/usr/bin/python3
import os
import sqlite3

db_filename = 'todo.db'
new_db = not os.path.exists(db_filename)

conn = sqlite3.connect(db_filename)

if new_db:
    print('Please create Schema')
else:
    print('Database already created - mostly schema exists')
conn.close()
```

# Database - Project TODO

project		
Column	Type	Description
name	text	Project Name
description	text	Project Description
deadline	date	Due Date

task		
Column	Type	Description
id	number	Uniq Task Identifier
priority	integer	Priority of the task
details	text	Task Description
status	text	Status
deadline	date	Due Date
completed_on	date	Completion Date
project	text	Task Belongs to Project

# Database - create schema & add data

```
#!/usr/bin/python3
import os
import sqlite3

db_filename = 'todo.db'
schema_filename = 'todo_schema.sql'

new_db = not os.path.exists(db_filename)

with sqlite3.connect(db_filename) as conn:
    if new_db:
        print('Let us create schema')
        with open(schema_filename, 'r') as f:
            schema = f.read()
        conn.executescript(schema)

        print('Inserting initial data')

        conn.executescript("""
            insert into project (name, description, deadline)
            values ('assignments', 'Assignments - Python for Data Science', '2017-05-24');

            insert into task (details, status, deadline, project)
            values ('assignment 1', 'done', '2017-01-29', 'assignments');

            insert into task (details, status, deadline, project)
            values ('assignment 2', 'in progress', '2017-02-22', 'assignments');

            insert into task (details, status, deadline, project)
            values ('assignment 3', 'active', '2017-03-31', 'assignments');
        """)

    else:
        print('Database exists, assume schema does, too.')
        #os.remove(db_filename)
```

# Database - Retrieve Data

```
#!/usr/bin/python3
import sqlite3

db_filename = 'todo.db'

with sqlite3.connect(db_filename) as conn:
    cursor = conn.cursor()

    cursor.execute(""" select id, priority, details, status, deadline from task where project = 'assignments' """")

    for row in cursor.fetchall():
        task_id, priority, details, status, deadline = row
        print('{:2d} [{:d}] {:<25} [{:<8}] {}'.format( task_id, priority, details, status, deadline))
```

1 [1]	assignment 1	[done]	(2017-01-29)
2 [1]	assignment 2	[in progress]	(2017-02-22)
3 [1]	assignment 3	[active]	(2017-03-31)

# Database - positional argument

```
#!/usr/bin/python3
import sqlite3
import sys

db_filename = 'todo.db'
project_name = sys.argv[1]

with sqlite3.connect(db_filename) as conn:
    cursor = conn.cursor()

    query = """ select id, priority, details, status, deadline from task where project = ? """
    cursor.execute(query, (project_name,))

    for row in cursor.fetchall():
        task_id, priority, details, status, deadline = row
        print('{:2d} [{:d}] {:<25} [{:<8}] {}'.format(task_id, priority, details, status, deadline))
```

```
[root@6879840ae648:/datascience/sessions/ten# ./argument_positional.py assignments
1 [1] assignment 1          [done      ] (2017-01-29)
2 [1] assignment 2          [in progress] (2017-02-22)
3 [1] assignment 3          [active     ] (2017-03-31)
```

# argument\_named.py

```
#!/usr/bin/python3
import sqlite3
import sys

db_filename = 'todo.db'
project_name = sys.argv[1]

with sqlite3.connect(db_filename) as conn:
    cursor = conn.cursor()

    query = """ select id, priority, details, status, deadline from task where project = :project_name order by deadline, priority """
    cursor.execute(query, {'project_name': project_name})

    for row in cursor.fetchall():
        task_id, priority, details, status, deadline = row
        print('{:2d} [{:d}] {:<25} [{:<8}] ({})'.format(task_id, priority, details, status, deadline))
```

```
[root@6879840ae648:/datascience/sessions/ten# ./argument_named.py assignments
 1 [1] assignment 1          [done      ] (2017-01-29)
 2 [1] assignment 2          [in progress] (2017-02-22)
 3 [1] assignment 3          [active     ] (2017-03-31)
```

# argument\_update.py

```
#!/usr/bin/python3
import sqlite3
import sys

db_filename = 'todo.db'
project_name = sys.argv[1]

with sqlite3.connect(db_filename) as conn:
    cursor = conn.cursor()

    query = """ select id, priority, details, status, deadline from task where project = :project_name order by deadline, priority """
    cursor.execute(query, {'project_name': project_name})

    for row in cursor.fetchall():
        task_id, priority, details, status, deadline = row
        print('{:2d} [{:d}] {:<25} [{:<8}] ({})'.format(task_id, priority, details, status, deadline))
```

```
[root@6879840ae648:/datascience/sessions/ten# ./argument_named.py assignments
 1 [1] assignment 1          [done      ] (2017-01-29)
 2 [1] assignment 2          [in progress] (2017-02-22)
 3 [1] assignment 3          [active     ] (2017-03-31)
[root@6879840ae648:/datascience/sessions/ten# ./argument_update.py 2 done
[root@6879840ae648:/datascience/sessions/ten# ./argument_named.py assignments
 1 [1] assignment 1          [done      ] (2017-01-29)
 2 [1] assignment 2          [done      ] (2017-02-22)
 3 [1] assignment 3          [active     ] (2017-03-31)
```

# load\_csv.py

```
#!/usr/bin/python3
import csv
import sqlite3
import sys

db_filename = 'todo.db'
data_filename = sys.argv[1]

SQL = """ insert into task (details, priority, status, deadline, project) values (:details, :priority, 'active', :deadline, :project) """
with open(data_filename, 'r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    with sqlite3.connect(db_filename) as conn:
        cursor = conn.cursor()
        cursor.executemany(SQL, csv_reader)
```

```
root@6879840ae648:/datascience/sessions/ten# more tasks.csv
deadline,project,priority,details
2017-03-01,assignments,2,"Submission of all assignments"
2017-03-08,assignments,3,"Work on Project"
2017-03-16,assignments,1,"Finish Documentation"
```

```
root@6879840ae648:/datascience/sessions/ten# ./load_csv.py tasks.csv
root@6879840ae648:/datascience/sessions/ten# ./argument_named.py assignments
 1 [1] assignment 1          [done   ] (2017-01-29)
 2 [1] assignment 2          [done   ] (2017-02-22)
 4 [2] Submission of all assignments [active ] (2017-03-01)
 5 [3] Work on Project      [active ] (2017-03-08)
 6 [1] Finish Documentation  [active ] (2017-03-16)
 3 [1] assignment 3          [active ] (2017-03-31)
```

# Back to Pandas

# pandas & datetime

```
[root@6879840ae648:/datascience/sessions/eleven# python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
[>>> import pandas as pd
[>>> datestrs = ['01/01/2017', '02/01/2017']
[>>> pd.to_datetime(datestrs)
DatetimeIndex(['2017-01-01', '2017-02-01'], dtype='datetime64[ns]', freq=None)
[>>> idx = pd.to_datetime(datestrs + [None, ''])
[>>> idx
DatetimeIndex(['2017-01-01', '2017-02-01', 'NaT', 'NaT'], dtype='datetime64[ns]', freq=None)
[>>> idx[2]
NaT
[>>> idx[3]
NaT
[>>> pd.isnull(idx)
array([False, False,  True,  True], dtype=bool)
[>>>
```

# Series

```
[root@6879840ae648:/datascience/sessions/eleven# more time_basics_pandas.py
#!/usr/bin/python3
import pandas as pd
import numpy as np

from pandas import Series, DataFrame
from datetime import datetime

dates = [
    datetime(2017, 1, 1),
    datetime(2017, 1, 2),
    datetime(2017, 1, 3),
    datetime(2017, 1, 4),
    datetime(2017, 1, 5),
]

ts = Series(np.random.randn(5), index=dates)
print(type(ts))
print(ts)
print ("-----\n")

print(type(ts.index))
print(ts.index)
print ("-----\n")

print (ts.index[0])
print (type(ts.index[0]))
print (ts.index.dtype)
print ("-----\n")]
```

# Series (Cont...)

```
[root@6879840ae648:/datascience/sessions/eleven# ./time_basics_pandas.py
<class 'pandas.core.series.Series'>
2017-01-01    0.949770
2017-01-02   -0.165319
2017-01-03   -1.112256
2017-01-04   -0.393262
2017-01-05   -0.828009
dtype: float64
-----
<class 'pandas.tseries.index.DatetimeIndex'>
DatetimeIndex(['2017-01-01', '2017-01-02', '2017-01-03', '2017-01-04',
               '2017-01-05'],
              dtype='datetime64[ns]', freq=None)
-----
2017-01-01 00:00:00
<class 'pandas.tslib.Timestamp'>
datetime64[ns]
-----
```

---

# Handling Missing Values

---



# Data Sets Missing - Why?

So many Reasons -

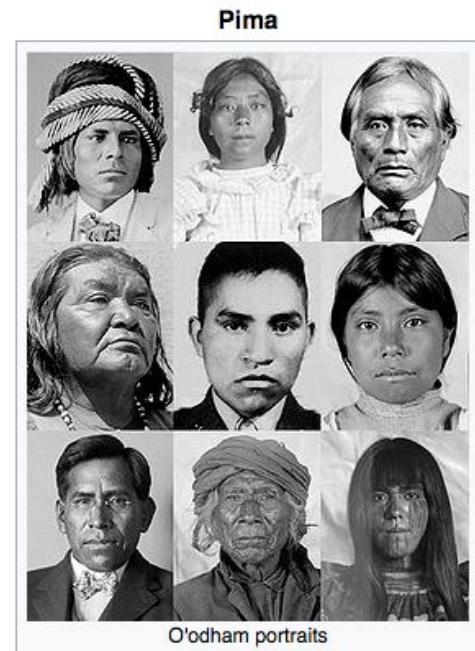
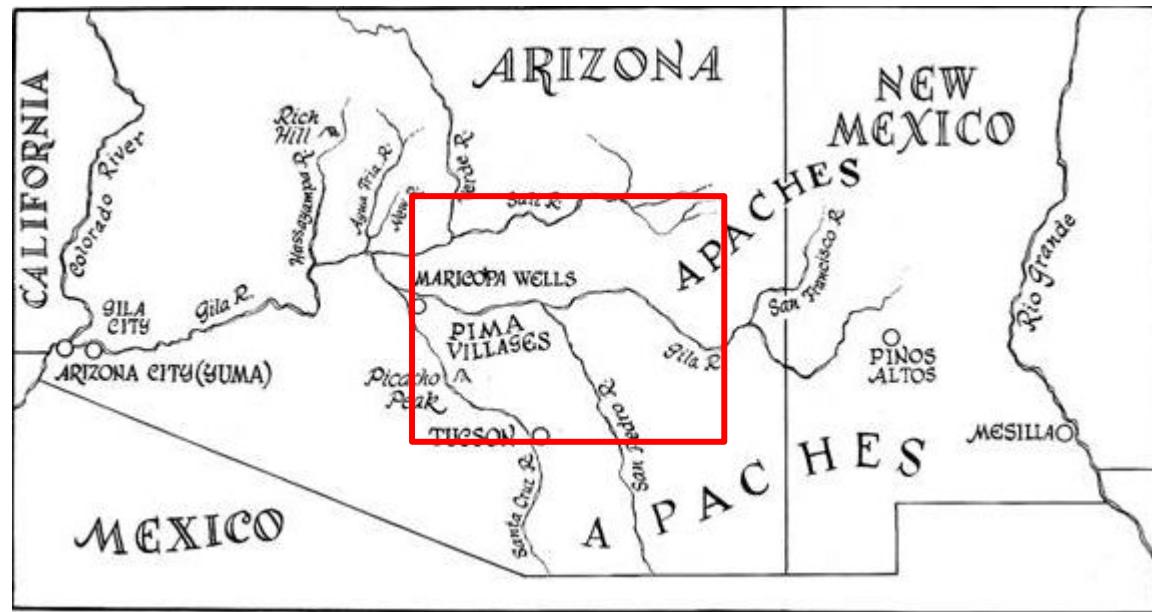
- Data could not get recorded - Manual, Automation (both can fail - its magical)
- Data got corrupted - world is not perfect

What do we do then?

- We need to handle it sensibly - off course, knowledge about data plays important Role

# Example - Pima Indians Diabetes Dataset

- Info on Pima Indians - [https://en.wikipedia.org/wiki/Pima\\_people](https://en.wikipedia.org/wiki/Pima_people)
- Data Source - <https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes>



# Pima Indians Diabetes Dataset (Cont...)

The number of observations for each class is not balanced. There are 768 observations with 8 input variables. The variable names are as follows:

0	Number of times pregnant.
1	Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
2	Diastolic blood pressure (mm Hg).
3	Triceps skinfold thickness (mm).
4	2-Hour serum insulin (mu U/ml).
5	Body mass index (weight in kg/(height in m)^2).
6	Diabetes pedigree function.
7	Age (years).
8	Class variable (0 or 1).

# Pima Indians Diabetes Dataset (Cont...)

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1

zero for body mass index or blood pressure is invalid.

# Pima Indians Diabetes Dataset (Cont...)

You can not have “Zero” to a few columns

1, 2, 3, 4, 5

1	Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
2	Diastolic blood pressure (mm Hg).
3	Triceps skinfold thickness (mm).
4	2-Hour serum insulin (mu U/ml).
5	Body mass index (weight in kg/(height in m)^2).

```
In [15]: import pandas as pd  
dataset = pd.read_csv('pima_diab.csv', header=None)  
dataset.describe()
```

Zero is affecting here

Out[15]:

	0	1	2	3	4	5	6	7	8
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
In [9]: dataset.head(20)
```

```
Out[9]:
```

	0	1	2	3	4	5	6	7	8
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1
11	4	110	92	0	0	37.6	0.191	30	0
12	10	168	74	0	0	38	0.537	34	1
13	10	139	80	0	0	27.1	1.441	57	0
14	1	189	60	23	846	30.1	0.398	59	1
15	5	166	72	19	175	25.8	0.587	51	1
16	7	100	0	0	0	30	0.484	32	1
17	0	118	84	47	230	45.8	0.551	31	1
18	7	107	74	0	0	29.6	0.254	31	1
19	1	103	30	38	83	43.3	0.183	33	0

The columns

2, 3, 4, 5 - clearly has  
“0” as values.

That's confirmation :)

# How many invalid records?

```
In [16]: print((dataset[[1,2,3,4,5]] == 0).sum())  
1      5  
2     35  
3    227  
4    374  
5     11  
dtype: int64
```

# Replace invalid with meaningful values

```
In [18]: import numpy as np  
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, np.NaN)
```

```
In [19]: dataset
```

Out[19]:

	0	1	2	3	4	5	6	7	8
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
5	5	116.0	74.0	NaN	NaN	25.6	0.201	30	0
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26	1
7	10	115.0	NaN	NaN	NaN	35.3	0.134	29	0
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	1
9	8	125.0	96.0	NaN	NaN	NaN	0.232	54	1
10	4	110.0	92.0	NaN	NaN	37.6	0.191	30	0
11	10	168.0	74.0	NaN	NaN	38.0	0.537	34	1
12	10	139.0	80.0	NaN	NaN	27.1	1.441	57	0

*Let us mark them with “NaN” using pandas “replace” method*

```
In [20]: print(dataset.isnull().sum())
```

```
0      0  
1      5  
2     35  
3    227  
4    374  
5     11  
6      0  
7      0  
8      0  
dtype: int64
```

# Missing Values are trouble

Missing Values can be troublesome to some of the Machine Learning Algorithms

Let us evaluate Linear Discriminant Analysis (LDA) here.

# Missing Values are trouble (Cont...)

```
In [13]: from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

dataset = read_csv('pima_diab.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]

# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')

print(result.mean())
|
```

# Missing Values are trouble (Cont...)

We are prevented from evaluating an LDA algorithm (and other algorithms) on the dataset with missing values.

```
522     if multi_output:
523         y = check_array(y, 'csr', force_all_finite=True, ensure_2d=False,
524
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py in check_array(array, accept_sparse, dtype, order,
copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    405             % (array.ndim, estimator_name))
    406         if force_all_finite:
--> 407             _assert_all_finite(array)
    408
    409     shape_repr = _shape_repr(array.shape)
    410
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py in _assert_all_finite(X)
    56         and not np.isfinite(X).all()):
    57         raise ValueError("Input contains NaN, infinity"
---> 58                 " or a value too large for %r." % X.dtype)
    59
    60
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

# Drop the Rows with Missing Values

Pandas provides the “dropna()” function that can be used to drop either columns or rows with missing data

```
In [14]: from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

dataset = read_csv('pima_diab.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

# drop rows with missing values
dataset.dropna(inplace=True)
# summarize the number of rows and columns in the dataset
print(dataset.shape)
```

(392, 9)

# Finally

```
In [15]: from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

dataset = read_csv('pima_diab.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

# drop rows with missing values
dataset.dropna(inplace=True)
# summarize the number of rows and columns in the dataset
print(dataset.shape)

# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]

# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```

(392, 9)  
0.78582892934

# Impute those Values

Removing rows with missing values can be too limiting on some predictive modeling problems, an alternative is to impute missing values.

A few Options to replace a missing value

- A constant value that has meaning within the domain, such as 0, distinct from all other values.
- A value from another randomly selected record.
- A mean, median or mode value for the column.
- A value estimated by another predictive model.

So new Data Derived - and new models applied on the data

Let us go with mean column values.

These mean column values will need to be stored to file for later use.

Pandas provides the “fillna()” function for replacing missing values with a specific value.

# Replace with Mean

```
In [16]: from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

dataset = read_csv('pima_diab.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# count the number of NaN values in each column
print(dataset.isnull().sum())
```

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
dtype: int64
```

# Finally

```
In [17]: from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

dataset = read_csv('pima_diab.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)

# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# count the number of NaN values in each column
#print(dataset.isnull().sum())

values = dataset.values
X = values[:,0:8]
y = values[:,8]
# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```

0.766927083333

```
>>> import pandas as pd
>>> raw_data = {
...     'subject': ['1', '2', '3', '4', '5'],
...     'first_name': ['Michel', 'Mark', 'Matt', 'Ryan', 'Gary'],
...     'last_name': ['Phelps', 'Spitz', 'Biondi', 'Lochte', 'Hall']
... }
>>>
>>> raw_data
{'subject': ['1', '2', '3', '4', '5'], 'first_name': ['Michel', 'Mark', 'Matt', 'Ryan', 'Gary'], 'last_name': ['Phelps', 'Spitz', 'Biondi', 'Lochte', 'Hall']}
>>> df_a = pd.DataFrame(raw_data, columns = ['subject', 'first_name', 'last_name'])
>>> df_a
   subject first_name last_name
0         1      Michel    Phelps
1         2       Mark     Spitz
2         3       Matt    Biondi
3         4       Ryan    Lochte
4         5       Gary     Hall
```

```
[>>> raw_data = {  
[...     'subject': ['4', '5', '6', '7', '8', '9'],  
[...     'first_name': ['Ian', 'Aaron', 'Nathan', 'Tom', 'Don', 'Johny'],  
[...     'last_name': ['Thorpe', 'Peirsol', 'Adrian', 'Jager', 'Schollander', 'Weis']  
[... }  
>>> df_b = pd.DataFrame(raw_data, columns = ['subject', 'first_name', 'last_name'])  
>>> df_b  
    subject first_name   last_name  
0        4        Ian      Thorpe  
1        5       Aaron     Peirsol  
2        6      Nathan     Adrian  
3        7        Tom      Jager  
4        8        Don  Schollander  
5        9      Johny      Weis  
>>> █
```

```
>>> raw_data = {  
...     'subject': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],  
...     'test'   : [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]  
... }  
>>> df_n = pd.DataFrame(raw_data, columns = ['subject','test'])  
>>> df_n  
    subject  test  
0          1    51  
1          2    15  
2          3    15  
3          4    61  
4          5    16  
5          7    14  
6          8    15  
7          9     1  
8         10    61  
9         11    16  
>>>  
>>> █
```

# Join Two DataFrames - along rows

```
>>> df_new = pd.concat([df_a, df_b])
>>> df_new
   subject  first_name    last_name
0         1      Michel     Phelps
1         2       Mark      Spitz
2         3       Matt    Biondi
3         4       Ryan   Lochte
4         5       Gary      Hall
0         4       Ian   Thorpe
1         5     Aaron  Peirsol
2         6    Nathan   Adrian
3         7       Tom    Jager
4         8       Don  Schollander
5         9     Johny     Weis
>>> █
```

# Join Two DataFrames - along rows

```
>>> df_new = pd.concat([df_a, df_b])
>>> df_new
   subject  first_name    last_name
0         1      Michel     Phelps
1         2       Mark      Spitz
2         3       Matt    Biondi
3         4       Ryan   Lochte
4         5       Gary      Hall
0         4       Ian   Thorpe
1         5     Aaron  Peirsol
2         6    Nathan    Adrian
3         7       Tom     Jager
4         8       Don  Schollander
5         9     Johny      Weis
>>> █
```

# Join Two Dataframes - along axis

```
[>>>
[>>> pd.concat([df_a, df_b], axis=1)
   subject first_name last_name subject first_name      last_name
0        1      Michel    Phelps        4       Ian      Thorpe
1        2       Mark     Spitz        5     Aaron      Peirsol
2        3       Matt    Biondi        6    Nathan      Adrian
3        4       Ryan   Lochte        7       Tom      Jager
4        5       Gary     Hall        8       Don  Schollander
5      NaN      NaN      NaN        9     Johny      Weis
>>> ]
```

# Merge Two DataFrames - along one column

```
[>>> pd.merge(df_new, df_n, on='subject')
   subject first_name    last_name  test
0         1      Michel     Phelps    51
1         2       Mark      Spitz     15
2         3       Matt     Biondi     15
3         4       Ryan     Lochte    61
4         4       Ian     Thorpe    61
5         5       Gary      Hall     16
6         5      Aaron    Peirsol    16
7         7       Tom     Jager     14
8         8       Don  Schollander    15
9         9     Johny      Weis      1
>>> ]
```

## Merge dataframes with both the left and right dataframes using the one key

```
>>> pd.merge(df_new, df_n, left_on='subject', right_on='subject')
    subject  first_name   last_name  test
 0         1      Michel     Phelps    51
 1         2       Mark      Spitz     15
 2         3       Matt     Biondi     15
 3         4       Ryan     Lochte    61
 4         4       Ian     Thorpe    61
 5         5       Gary      Hall     16
 6         5      Aaron    Peirsol    16
 7         7       Tom     Jager     14
 8         8       Don  Schollander    15
 9         9     Johny      Weis      1
>>>
```

# Merge with outer join

```
[>>> pd.merge(df_a, df_b, on='subject', how='outer')
   subject first_name_x last_name_x first_name_y last_name_y
0         1      Michel     Phelps        NaN       NaN
1         2       Mark      Spitz        NaN       NaN
2         3       Matt     Biondi        NaN       NaN
3         4       Ryan    Lochte        Ian     Thorpe
4         5       Gary      Hall     Aaron     Peirsol
5         6       NaN       NaN    Nathan     Adrian
6         7       NaN       NaN       Tom     Jager
7         8       NaN       NaN       Don Schollander
8         9       NaN       NaN     Johny      Weis
>>> ]
```

# Merge with Inner Join

```
[>>> pd.merge(df_a, df_b, on='subject', how='inner')
   subject first_name_x last_name_x first_name_y last_name_y
0         4        Ryan      Lochte        Ian     Thorpe
1         5        Gary       Hall      Aaron    Peirsol
>>> ]
```

# Left and Right Join

```
[>>> pd.merge(df_a, df_b, on='subject', how='right')
   subject first_name_x last_name_x first_name_y last_name_y
0        4        Ryan      Lochte        Ian    Thorpe
1        5        Gary       Hall     Aaron  Peirsol
2        6        NaN       NaN    Nathan  Adrian
3        7        NaN       NaN        Tom   Jager
4        8        NaN       NaN        Don Schollander
5        9        NaN       NaN     Johny   Weis
[>>> pd.merge(df_a, df_b, on='subject', how='left')
   subject first_name_x last_name_x first_name_y last_name_y
0        1      Michel     Phelps        NaN      NaN
1        2       Mark      Spitz        NaN      NaN
2        3       Matt     Biondi        NaN      NaN
3        4       Ryan      Lochte        Ian    Thorpe
4        5       Gary       Hall     Aaron  Peirsol
[>>>
[>>> _
```

# Add Suffixes

```
>>> pd.merge(df_a, df_b, on='subject', how='left', suffixes=('_left', '_right'))
      subject first_name_left last_name_left first_name_right last_name_right
0          1           Michel       Phelps            NaN        NaN
1          2            Mark        Spitz            NaN        NaN
2          3            Matt       Biondi            NaN        NaN
3          4            Ryan       Lochte            Ian     Thorpe
4          5            Gary        Hall           Aaron    Peirsol
>>> █
```

**Thank you!**