Topic 1

Introduction to Python

Topics we will cover

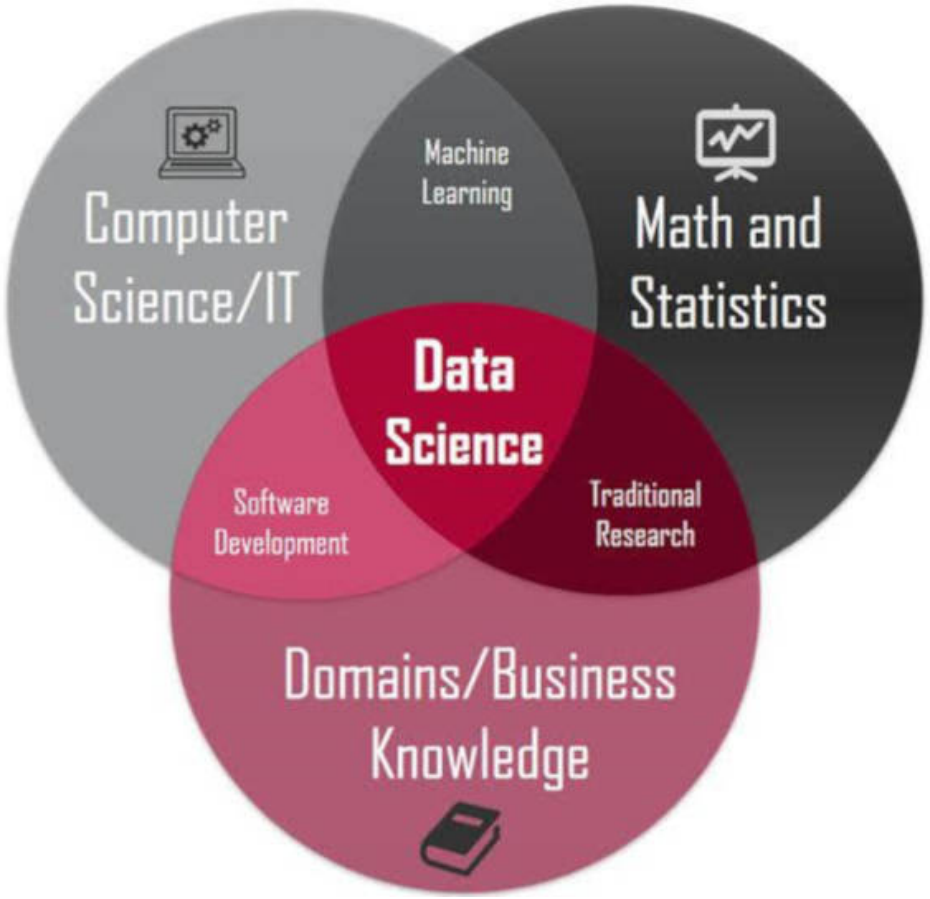
1. What is Data Science and why we need it
2. Intro to Python
3. Install Anaconda Python
4. Running Python programs with Jupyter Notebook

Topics we will cover

1. Perform simple console input and output in Python
2. Inserting comments into your Python code
3. Using import and calling imported functions
4. Work with simple data types such as numbers, strings
5. Use assignment, arithmetic, comparison and logical operators
6. Implement flow control using if-else statements
7. Implement loop control using for and while statements
8. Use Python list, tuple, dictionary object
9. Define Python functions

What Is Data Science

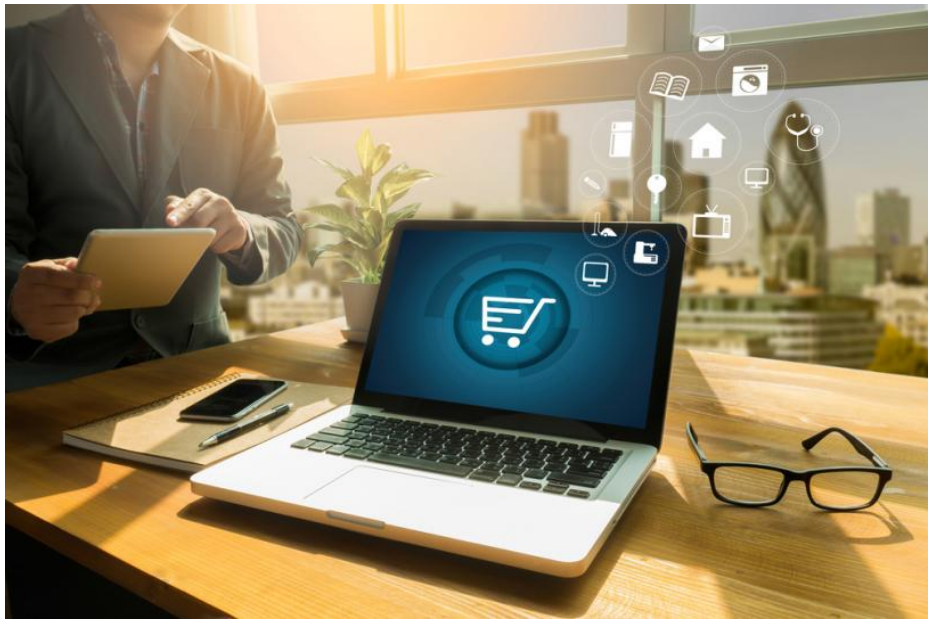
- Data Science is a field of Big data that evaluates massive complex data and gives significant insights about the data.
- This field has been dominating most of the industries today and has become the fuel for industries.



What Data Science Can Do

1. Financial Fraud Detection

- Tax evasion costs the U.S. government $458 billion a year, so IRS has modernized its fraud-detection protocols in the digital age.
- The agency generates taxpayer profiles by analysing big data, such as social media data, emailing analysis, electronic patterns and more.
- Based on these profiles, it can forecast individual tax returns. Anyone with big difference between forecast and actual will be audited.



What Data Science Can Do

2. Health Care

- Google developed a tool LYNA for identifying breast cancer tumors that transfer to nearby lymph nodes.
- Data science and big data has been applied to new drug discovery and development.
- Data science helps to analyze the reaction of genes to various medications. Big data technologies can also reduce the processing time for genome sequencing significantly.

What Data Science Can Do

3. Road Transportation

- UPS uses data science to optimize package transport from drop-off to delivery. Its latest platform, Network Planning Tools (NPT), incorporates AI to crack challenging logistics puzzles, such as how packages should be rerouted around bad weather or service bottlenecks.
- According to a company forecast, the platform could save UPS $100 to $200 million in 2020

What Data Science Can Do

4. Marketing and E-commerce

- Instagram uses data science to target its sponsored posts, which hawk everything from trendy sneakers to movie websites.
- The company's data scientists pull data from Instagram as well as its owner, Facebook, which has exhaustive web-tracking infrastructure and detailed information on many users, including age and education.
- From there, the team creates AI that convert users' likes and comments, their usage of other apps and their web history into predictions about the products they might buy.

What Data Science Can Do

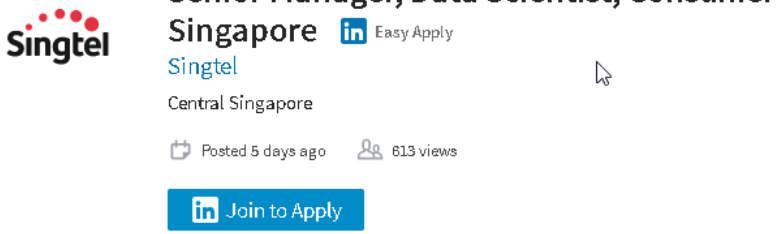
5. Banking: auto credit card approval.

6. Manufacturing: Optimize energy consumption and improve production efficiency.

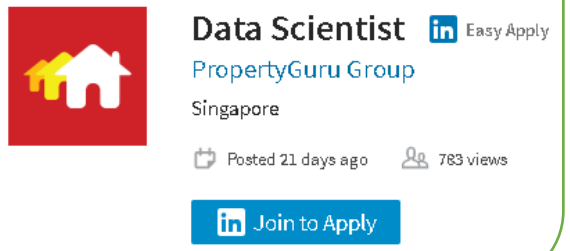
7. Education: Monitor student performance in class.

Data Science jobs - what's needed?

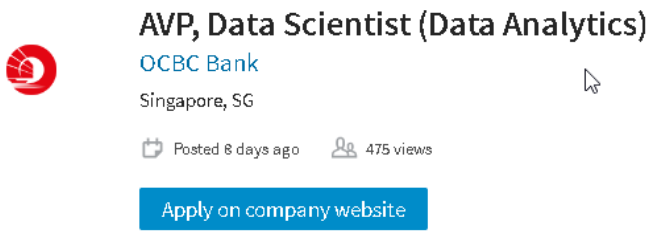
- Excellent experience with data management and statistical languages / packages such as R, Python, Scala, Spark, Matlab



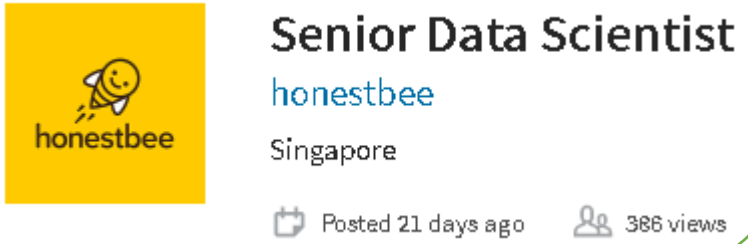
- Proficient in a data science language like Python, R, or Scala
- Familiar with machine learning packages like scikit-learn, TensorFlow



- Experienced with common data science toolkits, such as R, SAS, Python or Spark-ML. Excellence in at least one of these is highly desirable



- Proficient in SQL, Python, or any other programming languages
- Experience working with distributing systems such as Redshift, BigQuery, Hadoop, Spark, Beam, etc



https://sg.linkedin.com/jobs/data-science-jobs

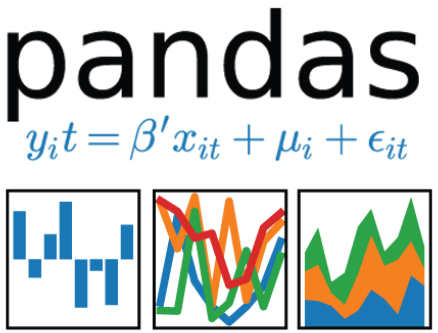
Brief History of Python

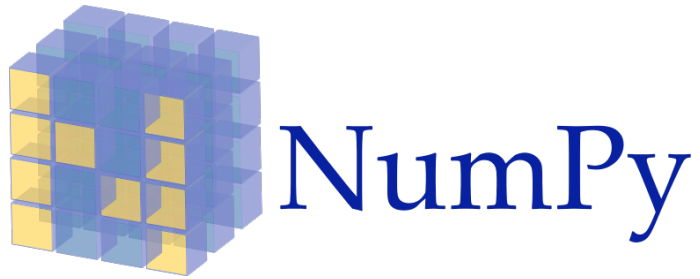
- Python is a widely used high-level programming language created by Guido van Rossum and first released in 1991.
- Python 2.0 released in Oct 2000, followed by Python 3.0 in 2008
- Latest version of Python, version 3.9 released in Oct 2020

Creator of Python

Why Python for Data Science?

- Python is a great tool for data manipulation, visualization and analysis!
- Flexible general programming language
- Easy to learn and work with
- Has many math / science libraries that makes working with data much easier





Anaconda

- Python can be run on many operating systems : Windows, Linux, Mac etc
- For this module, we will install the Anaconda Individual Edition
- Anaconda is a free Python distribution that comes with most of the data science Python packages we need to write data analysis programs
- Includes modules such as NumPy, Pandas, Matplotlib
- https://www.anaconda.com/products/individual

Jupyter Notebook

- In this module, you will be largely using the Anaconda Jupyter Notebook to write and run your Python code
- The Jupyter Notebook App is a web-based application that allows you to edit and run Python programs via your favourite browser.
- Refer to Practical 00 to setup the Anaconda Jupyter Notebook environment for the rest of the Labs.

Practical 00

Run Python code via Jupyter Notebook

Python Inputs and Outputs (1)

- To output to the console, use the print() function
- To get input from the console, use the input() function

- The input() function automatically interprets all entries as strings
- To convert them to numbers, use the int() or float() function
- To convert numbers to string, use the str() function

Ask user for name and print out his name

```
name = input("What's your name?")
print('Hello' + name)
```

Ask user for length of square and print out area and perimeter

```
length = int(input("Enter length: "))
breadth = int(input("Enter breadth:"))

print('Area:' + str(length*breadth))

#Another way to print using f-string
print(f'Area: {length*breadth}')
```

https://docs.python.org/3/library/string.html#formatspec
https://docs.python.org/3/library/string.html#formatstrings

Python Inputs and Outputs (2)

- Use f-string to specify decimal precision

```
weight = float(input("Enter your weight:"))
height = float(input("Enter your height:"))

bmi = weight/(height*height)

print(f'Your bmi is {bmi}')

print(f'Your bmi is {bmi:.2f}')

print(f'Your Height is {height}, weight is {weight}
and BMI is {bmi:.4f}')
```

Ask user for his weight and height then calculate and print out his BMI

```
Enter your weight:50
Enter your height:1.55
Your bmi is  20.811654526534856
Your bmi is 20.81
```

Inserting comments into Python code

- Sometimes, you may want to insert comments into your Python code
- Start comments with a # symbol

Example to show how you insert comments in Python code

```
# This example shows you how to write a comment
# The code below prompts the user to enter his current age
# and automatically computes the year he was born

age = int(input('Enter your age this year: '))
year_of_birth = 2020 - age
print(f"You were born in the year {year_of_birth} ")
```

Practical 01 Section 2

Using import (1)

- Sometimes, we need to use a function that is not in the default Python library
- To use such functions, we use the import keyword to ask the Python interpreter to let us use it
- In the example below, we want to generate a random number using the randint() function from the random library.

Example to show how you can generate random number using the random.randint() function by first importing the random library

```
import random
secret_number  =
random.randint(1,100)
print(secret_number)
```

Using import (2)

Another example to show how you can get the current time using datetime.now() function which is available only if you import datetime library

```
# This example shows you how to do an import
# as well as how to format a datetime

from datetime import datetime

now = datetime.now()

print(f'Today is {now:%d-%b-%Y %H:%M}')
```

Using import (3)

- In the example below, we want to print two messages "Hello" and "Hello again"
- However, we want to wait for 5 seconds after the first message, before printing the second
- To do this, we can import the sleep function from the time library as shown below

This example shows how delays can be introduced in your program by using sleep() function from time library

```
from time import sleep

print("Hello")
sleep(5)
print("Hello again")
```

Using import (4)

- Here's another example that uses the time() function from the time library to calculate the time elapsed between two timings

Calculate duration between two times

```
from time import time

print('Enter your name in the quickest amount of time:')
start = time()    # Store the current time

name = input()
reaction_time = time() - start    # Calculate how much time has passed

print(f'You took {reaction_time:.2f} seconds')
```

Using import (5)

This example shows how you calculate the number of years, months and days between two dates by using two libraries: datetime and dateutil

```
from datetime import datetime
from dateutil import relativedelta as rdelta

date_of_birth = "17-08-1973"  # in string format
date_of_birth = datetime.strptime(date_of_birth, '%d-%m-%Y')  # datetime format
today = datetime.now()  # this is in datetime format

# Calculate diff
rd = rdelta.relativedelta(today,date_of_birth)

print(f"You were born on {date_of_birth:%d-%b-%Y}“)
print(f"{rd.years} years, {rd.months} months and {rd.days} days since you were born")
```

Working with numeric data types (1)

- Python has 3 numeric types: integers (int), floating point numbers (float) and complex numbers
- Integers are whole numbers while floating point numbers have decimal points

Performing simple arithmetic operations on numeric data types

```
x = 2
y = 8
z = 3.459
print(x*y)    # product of x and y →  output: 16
print(x/y)    # quotient of x and y →  output: 0.25
print(x % y)  # remainder of  x divided by y  →  output: 2
print(x**y)  # x to the power of y  →  output: 256
print(round(z,1))   # z rounded to 1 decimal place →  output: 3.5
```

https://docs.python.org/3.6/library/functions.html#int

Working with numeric data types (2)

- Besides the standard functions, you can apply several additional mathematical functions from the **math** library to numeric data types as shown below

Example to show how you use functions from the math library

```
import math

x = 349.4378

print(math.isnan(x)) # returns True if x is NOT a number
print(math.ceil(x)) # round x upwards →   output: 350
print(math.floor(x)) # round x downwards →   output: 349
print(math.sqrt(x)) # square root of x →   output: 18.693255468216336
```

Practical 01 Section 4

Working with strings

- Python has a built-in string class with many handy features
- String literals can be enclosed by either double or single quotes

Creating string objects

```
s1 = 'hi how are you'

s2 = "I am fine, thank you"
```

String indexing

- Individual characters in a string can be accessed via their INDEX
- Indexing starts with zero in Python strings

0	1	2	3	4	5	6	7	8	9	10	11	12	13
h	i		h	o	w		a	r	e		y	o	u

```
s = 'hi how are you'

print(s[0]) # this extracts the character with index 0 →   h
print(s[1])   # this extracts the character with index 1 →   i
print(s[3:6])   # this extracts 3 characters from index 3 to index 5
→   how
```

Getting length of a string

- You can retrieve the length of a string with the **len** function

0	1	2	3	4	5	6	7	8	9	10	11	12	13
h	i		h	o	w		a	r	e		y	o	u

```
s = 'hi how are you'

print(len(s))  # this prints the length of the string →  14
```

Repeating a string with *

- You can repeat a string by multiplying it using the * operator

0	1	2	3	4	5	6	7	8	9	10	11	12	13
h	i		h	o	w		a	r	e		y	o	u

```
s = 'hi how are you'

print(s + ' today')  # this concatenates the two strings
print(s*2)  # prints s twice
```

Concatenate two strings

- You can combine two strings by using the + operator

0	1	2	3	4	5	6	7	8	9	10	11	12	13
h	i		h	o	w		a	r	e		y	o	u

```
s1 = 'hi how are you'
s2 = ' today'
print(s1 + s2)  # this concatenates the two strings
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
h	i		h	o	w		a	r	e		y	o	u		t	o	d	a	y

Concatenate string and number

This example emphasizes the need to use a conversion function when concatenating a string and a non-string

```
pi = 3.14
text = 'The value of pi is ' + pi      ## This does NOT work because pi is number
text = 'The value of pi is '  + str(pi)  ## This is ok
print(text)
```


Useful methods of the str class

- Turn all characters uppercase/ lowercase and check if a string is numeric

```
s = 'hi how are you'

print(s.upper())    # this prints out the string in CAPS
print(s.lower())    # this prints out the string in lowercase

print(s.isnumeric())  # prints true if s is numeric   →   returns
False
```

Useful methods of the str class

- Find if a substring exists

0	1	2	3	4	5	6	7	8	9	10	11	12	13
h	i		h	o	w		a	r	e		y	o	u

```
s = 'hi how are you'

print(s.find('ar'))  # find 'ar' inside the string,
                     # if found it returns the first index position of 'ar' → returns 7
```

Useful methods of the str class

- Splitting a string

0	1	2	3	4	5	6	7	8	9	10	11	12	13
h	i		h	o	w		a	r	e		y	o	u

```
s = 'hi how are you'

words = s.split(sep=' ')    # splits s into an array, using space as separator
print(words[2]) → returns are
```

words[0]	words[1]	words[2]	words[3]
hi	how	are	you

Practical 01 Section 5

Types of Operators in Python

Python supports the following types of operators:

- Arithmetic Operators
- Comparison Operators
- Assignment Operators
- Logical Operators
- Membership Operators
- Identity Operators

Arithmetic operators

Assume variable *num1* holds 10, variable *num2* is 5 and variable *num3* is 3

+	Add num1 and num2	num1 + num2 = 15
–	Subtract num2 from num1	num1 - num2 = 5
*	Multiply num1 by num2	num1 * num2 =50
/	Divide num1 by num2	num1 / num2 = 2
%	Get the remainder after dividing num1 by num2	num1 % num2 = 0
**	num1 to the power of num2	num1 ** num2 = 10^5
//	Divide num1 by num3 and discard any decimal points from the answer	num1 // num3 = 3

Assignment operators (1)

- Python allows you to assign single or multiple variables at one go using the = operator

example_assignment_1.py

```
counter = 100
name    = "John"

a = b = c = 1

num1, num2, name = 1, 2, "john"
```

Comparison operators

Assume variable *num1* holds 10, variable *num2* is 20

```
num1=10
num2=20

print(num1==num2)
print(num1!=num2)
```

==	num1 == num2	False
!=	num1 != num2	True
>	num1 > num2	False
<	num1 < num2	True
>=	num1 >= num2	False
<=	num1 <= num2	True

Logical operators

Assume variable num1 =10, variable num2 = 20

and	num1>=10 and num2>=10 num1>=20 and num2>=20	True False
or	num1>=20 or num2>=20 num1>20 or num2<10	True False
not	not num1==num2 not num1<num2	True False

example_logicaloperators_1.py

Membership operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples.

example_membershipoperators_1.py

```
a = 5
b = 10
list = [1, 2, 3, 4, 5 ]

print(a in list) # True
print(b in list) # False
print(b not in list) # True
```

Identity operators

Identity operators compare the memory locations of two objects.

example_identityoperators_1.py

```
a = 20
b = 20
c = a
d= 30

print( a is b)  # True
print(a is not b) # False
print(a is c)  # True
print(b is c) # True
print(a is d) # False
```

If-else statements (1)

Python provides if-else statements to enable conditional programming

Check if two numbers are equa

```
input_1 = int(input("Enter number 1:"))
input_2 = int(input("Enter number 2:"))

if input_1==input_2:
   print("The two numbers are the same")
else:
   print("The two numbers are not the same")
```

If-else statements (2)

Python provides if-else statements to enable conditional programming

example_ifelse_1.py

```
input_1 = input("Enter number 1:")
input_2 = input("Enter number 2:")

if input_1.isnumeric() and input_2.isnumeric() :
    num_1 = int(input_1)
    num_2 = int(input_2)
    sum = num_1 + num_2
    print("The sum of {} and {} is {}".format(num_1,num_2,sum))

else:
   print("Please enter numeric values only")
```

elif

You can also add in an unlimited number of elif statements following an if to check for multiple conditions

example_ifelse_2.py

```
print("Below is our drinks menu:")
print("1.Coke   2.Coffee   3.Juice")
drink = int(input("Enter your choice of drink:"))

if drink==1:
   print("Coke is $1.00")
elif drink==2:
   print("Coffee is $0.50")
elif drink==3:
   print("Juice is $2.00")
else:
   print("Sorry, you have entered an invalid
choice")
```

Nested if-else

You can also nest if-else and elif statements

Practical 01 Section 6

```
number = input("Enter your choice (1-2): ")
staff = input("Are you a staff (Y/N)? ")

if number=="1":
   if staff.upper() == "Y":
        price = 5*0.9
   else:
        price = 5*0.95
elif number=="2":
    if staff.upper() == "Y":
        price = 10*0.8
    else:
        price = 10
else:
   print("You did not enter a valid input")
   #exit()

print("Price is ${}".format(price))
```

iteration with for and while loops

- We often want computers to repeat some process several times
- Programming languages provide structures that enable you to repeat blocks of instructions over and over again
- This type of repetition is known as iteration.
- There are 2 types of loops in Python:
  - **for** loop
  - **while** loop

- These simple for loop and while loop examples would write "hello world" 5 times:

```
for counter in range(0,5):
    print("hello world")
```

```
counter = 0
while counter < 5:
    print("hello world")
    counter+=1
```

Using for loop

- Syntax

```
for stepper_variable in
sequence_variable
   something_you_want_to_do
```

This prints even numbers from 2 to 100

```
for i in range(2,102,2):
   print(i)
```

This prints the individual letters in the word Python

```
for letter in 'Python':     # First
Example
   print('Current Letter :', letter)
```

- Use the for loop when you know how many times you want to repeat a series of statements
- The first line of the for statement is used to state how many times the code should be repeated
- A stepper variable is used to count through each iteration of the loop.

while loop (1)

The **while** loop repeatedly executes as long as a given condition is true.

This prints "The count is 1", "The count is 2" etc until 10

```
count = 1
while count <= 10:
   print('The count is:
{}'.format(count))
   count += 1

print("Good bye!")
```

while loop (2)

```
password = ""
while password != "secret":
    password = input("Please enter the password: ")
    if password == "secret":
        print("Thank you. You have entered the correct password")
    else:
        print("Sorry the value entered in incorrect - try again")
```

Here is an example of a while loop being used to test a password. The password is secret and the code within the loop is executed until the user inputs the correct password.

Python Lists

- The Python list class is used to store collections of similar or dissimilar items
- Creating a list is as simple as putting different comma-separated values between square brackets

```
var1 = 'red`   # this is not a list

list1 = ['red', 'green', 'blue']
list2 = ['NSDDA1', 'NSDDA2', 2017,
2018]
list3 = [1, 2, 3, 4, 5 ]
list4 = ["a", "b", "c", "d"]
```

Accessing Values in Lists (1)

- To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index
- You can get the length of a Python list by using the len() function

```
countries = ['Austria', 'Belgium', 'Canada', 'Denmark', 'Ecuador',
'France']

print(countries[0])    ## Austria
print(countries[1])    ## Belgium
print(countries[2:4])   ## Canada, Denmar
print(countries[-1:])   ## France
print(countries[-2:])   ## Ecuador, France

print(len(countries))  ## 6
```

```
Austria
Belgium
['Canada', 'Denmark']
['France']
['Ecuador', 'France']
6
```

Accessing Values in Lists (2)

- Another example to show how you can "slice" lists

```
list1 = [50,20,30,10,100,40,200]

print(list1[2]) # 3rd element from the front -> 30
print(list1[-2]) # 2nd element from the back -> 40
print(list1[3:5]) # 4th to 5th element -> 10 100
print(list1[-4:-1]) #10 100 40
print(list1[-1:-4:-1]) #200 40 100
print(list1[::-1]) #200 40 100 ...
```

```
30
40
[10, 100]
[10, 100, 40]
[200, 40, 100]
[200, 40, 100, 10, 30, 20, 50]
```

Iterate through a List (1)

- You can use the **for** loop to iterate through a Python list

```
colors = ['red', 'green', 'blue', 'yellow', 'magenta', 'cyan

for color in colors:
    print(color)
```

```
red
green
blue
yellow
magenta
cyan
```

```
numbers = [1,2,3,4,5,6,7,8,9,10]

for n in numbers
   print(n)   # 1,2,3,4,5,6,7,8,9,10
```

Iterate through a List (2)

```
colors = ["red car", "green car", "blue", "purple"]
cars  = ["Toyota", "Mercedes"]

for i in range(4):
    print(colors[i])
```

```
fruits = ['banana', 'apple',  'mango', 'pear', 'grape']

for index in range(2,5):
   print('Current fruit :', fruits[index])   #mango, pear, grape
```

Updating Lists

```
subjects =
['English','Maths','Geography']

## add new item at the back
subjects.append("Physics")

## insert new item at the  front
subjects.insert(0, "Chem")

## update value at index 0
subjects[0] = "Chemistry"
```

```
## remove at index 1
del(subjects[1])

## removes the last object
subjects.pop()
```

- You can update a list element by giving the slice on the left-hand side of the assignment operator
- To add elements in a list, use **append()** or **insert()** methods
- To remove elements in a list, use **del()** or **pop()** methods

+ and * operations on Lists

- Lists respond to the + and * operators much like strings, equating to concatenation (+) and repetition (*) as well
- The result is a new list

```
list_1 =
[50,20,30,10,100,40,200]
list_2 = [300,65,80]
list_3 = ["apple",
"orange","pear"]

print(list_1 + list_2)
print()

print(list_3*2)
```

```
[50, 20, 30, 10, 100, 40, 200, 300, 65, 80]

['apple', 'orange', 'pear', 'apple', 'orange', 'pear']
```

Built-in List Functions

- Python includes the following list functions

Function	Description
len(list)	Gives total length of the list
max(list)	returns item with the max value from the list
min(list)	returns item with minimum value from the list
list(seq)	Converts tuple from the list

```
mylist = [50,20,30,10,100,40,200]
mytuple = (50,20,30,10,100,40,100)

print(len(mylist))

print(max(mylist
print(min(mylist)

print(list(mytuple))
```

```
7
200
10
[50, 20, 30, 10, 100, 40, 100]
```

Built-in List Methods

- Python includes the following list met

Function	Description
count(obj)	Returns count of how many times obj occurs in listGives total length of the list
extend(seq)	Appends the contents of seq to list
index(obj)	Returns the lowest index in list that obj appears
reverse()	Reverses objects of list in place
sort([func])	Sorts objects of list, use compare func if given

```
list1 = [50,20,30,20]

print(list1.count(20))
print(list1.index(30))

list1.extend([10,11])
print(list1)

list1.reverse()
print(list1)

list1.sort()
print(list1)
```

```
2
2
[50, 20, 30, 20, 10, 11]
[11, 10, 20, 30, 20, 50]
[10, 11, 20, 20, 30, 50]
```

Copying lists

- Make a copy of a list to another list
- Changes to the copy do not affect the original

```
list1 = [50,20,30,10,100,40,200]

list1copy = list(list1)

list1copy.sort()

print(list1) # list remains the same
print(list1copy) # listcopy is now sorted
```

```
[50, 20, 30, 10, 100, 40, 200]
[10, 20, 30, 40, 50, 100, 200]
```

Practical 01 Section 9

List Comprehensions (ADVANCED)

- Python supports a concept called "list comprehensions"
- It eliminates need for loop loops and makes code very concise
- It is an advanced concept, so don't worry too much about it if its a little hard to grasp

```
S = [x**2 for x in range(10)]
print(S) #0 1 4 9 16 25 36 49 64 81

col = ['Red','Green','Blue','Yellow']
info = [[c.upper(), c.lower(), len(c)] for c in
col]
print(info)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[['RED', 'red', 3], ['GREEN', 'green', 5], ['BLUE', 'blue', 4], ['YELLOW', 'yellow', 6]]
```

Working with tuples (1)

- A tuple is like a list. The difference is that, it is immutable (cannot be changed)
- Tuples use round brackers instead of square brackets

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
tup4 = (50,) # for tuples with a single value, it is a must to put a comma after first value

print(tup1)
print(tup2[2:4])
print(tup3[-1])
print(len(tup4))
```

example_tuples_1.py

Dictionaries

- A dictionary is a data type similar to lists, but works with keys and values instead of indexes
- Each value stored in a dictionary can be accessed using a **unique** key, which is any type of object instead of using its index to address it.
- For example, a database of phone numbers could be stored using a dictionary like this

```
phonebook = {}

phonebook["John"] =
93847756
phonebook["Jack"] =
93837726
phonebook["Jill"] =
947662781

prin
```

```
phonebook = {
   "John" :
938477566,
   "Jack" :
938377264,
    "Jill" : 947662781
}

print(phonebook)
```

Iterating through a dictionary

- It is best to think of a dictionary as an unordered set of *key: value* pairs, with the requirement that the keys are unique (within one dictionary).

- The example here shows how you can iterate through a dictionary using its keys

Create a dictionary and iterate through it

```
tel = {
    'Ms Dora': 68706085,
    'Ms Eileen': 68704739,
    'Mr Calvin':67721917,
    'Mr Hu-Shien': 67721922
}

print(tel.keys())

for t in tel: # t is the key
   print(t) # print the keys
   print(tel[t]) # access via the keys
```

Iterating through dictionary (items)

- You can also use the dict constructor to create the key-value pairs as shown
- This example also shows another way to iterate through dictionary by using .items() method

```
phonebook = {
   "John" : 938477566,
   "Jack" : 938377264,
   "Jill" : 947662781
}

for name, number in phonebook.items():
    print(f"{name}: {number}")
```

Introduction to Functions

- A **function** is a block of organized, reusable code that is used to perform a single, related action
- Functions provide better modularity for your application and a high degree of code reusing

```
# The familiar print() function
print("Hello")

# The familiar input() function
num1 = input("Enter number 1: ")
num2 = input("Enter number 2: ")

# The familiar len() function
name = input("Enter your name: ")
print(len(name))
```

Defining a Python function

- Function blocks begin with the keyword def followed by the function name and round brackets
- The round brackets may enclose input parameters
- The code block within every function starts with a colon (:) and is indented

```
def printme(string, num_times):
    for i in range(num_times):
        print(string)
```

- **This function accepts two input parameters**
- **Does not return any value**

```
printme('Hello',3)
printme('*****',2)
```

Defining a Python function

```
from time import sleep

def printme(string, num_times,del
    for i in range(num_times):
        print(string)
        sleep(delay)
```

```
printme('Hello',10,5)
printme('*****',20,3)
```

- This function accepts thtree input parameters
- Does not return any value

Defining a Python function

```
def traffic_light(color):
    if color == 'red':
        return "Cannot cross"
    elif color == 'green':
        return "Can cross"
    else:
        return "Invalid color"
```

- This function accepts one input parameter
- Returns a single string value

```
color = "red"
print(traffic_light(color
))

color = "green"
print(traffic_light(color
))
```

Defining a Python function

```
def isDivisibleBy(num1,num2)
    if num1 % num2 == 0:
        return True
    else:
        return False
```

```
n1 = 180
n2 = 4

print(isDivisiblyBy(n1,
n2))
```

- This function accepts two input parameters
- Returns a single Boolean value

Defining a Python function

```
def divideList(list, number):
    list1 = []
    list2 = []

    for n in list:
        if n<number:
            list1.append(n)
        elif n> number:
            list2.append(n)
    return list1, list2
```

- This function accepts two input parameters, first is a list, second is a number
- The function iterates through the list elements, and divides up the list
- First list -> numbers smaller than number
- Second list -> numbers greater than number
- Returns both the First and Second List

```
d = [100,5,11,30,35,16]
d1, d2 = divideList(d,20)
print(d1)
```

Practical 01 Section 10

try-except

- It is good practice to write Python programs that have exception handling
- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
- When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash
- You can handle exceptions using try, except as shown

```
try:
    number = int(input('Please enter a
number: '))
    print(number)
except ValueError:
    print('You should enter a number')
```

Handling known error types in Python
ValueError, ZeroDivisionError, TypeError etc are known error type

The End

Some useful Python resources

- https://www.tutorialspoint.com/python
- http://www.practicepython.org/
- https://www.learnpython.org
- https://www.w3resource.com/python/python-tutorial.php
- https://pythonschool.net/
- https://developers.google.com/edu/python/
- https://chrisalbon.com/python
- http://pbpython.com/
- www.pythonforbeginners.com
- https://www.programiz.com/python-programming
- pythoncentral.io
- https://learnpythonthehardway.org
- http://codingbat.com/python

Some useful Python resources

- https://stackoverflow.com/questions/tagged/python
- https://pyformat.info/