Data Science Workshop-1

ITER, SOA University

Meenakshi Das

Centre for Data Science
SOA University
meenakshidas@soa.ac.in

Course details

What you will learn in next 2 semesters

SEMESTER 3 (SUBJECTS with BLACK Highlighting ARE REQUIRED FOR PROMOTION TO 3RD YEAR)			
CODE	**SUBJECT**	**CREDITS**	**GRADING PATTERN**
CSE 2195	Data Science Workshop 1	4	5
Preliminaries, Python Language Basics, IPython, and Jupyter Notebooks, Built-In Data Structures, Functions, and Files, NumPy Basics: Arrays and Vectorized Computation, Getting Started with pandas, Data Loading, Storage, and File Formats, Data Cleaning and Preparation, Data Wrangling: Join, Combine, and Reshape, Plotting and Visualization, Data Aggregation and Group Operations.		Textbook – Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter by McKinney, Shroff/O'Reilly Course Format: 8 Lab Contact Hours / week	

SEMESTER 4 (SUBJECTS with BLACK Highlighting ARE REQUIRED FOR PROMOTION TO 3RD YEAR)			
CODE	**SUBJECT**	**CREDITS**	**GRADING PATTERN**
CSE 2196	Data Science Workshop 2	4	5
Preliminaries, Introduction, A Crash Course in Python, Visualizing Data, Linear Algebra, Statistics, Probability, Hypothesis and Inference, Gradient Descent, Getting Data, Working with Data, Machine Learning, k-Nearest Neighbors, Naive Bayes, Simple Linear Regression, Multiple Regression, Logistic Regression, Decision Trees, Neural Networks, Deep Learning, Clustering, Natural Language Processing, Network Analysis, Databases		Textbook – Introducing Data Science: Big Data, Machine Learning, and More, Using Python Tools by Cielen, Dreamtech/Manning Course Format: 8 Lab Contact Hours / week	

Future Specialization Courses

- Introduction to Data Analytics with **SQL**-5
- **Text** analysis with python-5
- Design machine learning system(**ML Model** development)-5
- **Big-data** Analytics with Apache spark-6
- Data architecture and **ware house**-6
- Data Science **projects**-7

Course Outcomes

- CO1- Understand and learn to use the basics of Python language, like built-in data structures, and functions.
- CO2- Use the NumPy library for arithmetic and statistical operations on vectors(NumPy arrays).
- CO3- Utilize the Pandas data frame for Data Loading, Storage, Cleaning, and preparation.
- CO4- Learn the hierarchical indexing in pandas, to combine, join, and rearrange data across number of databases.
- CO5- Visualize the data insights using matplotlib, pandas, and seaborn.
- CO6- Apply the techniques of data aggregation and group operations.

Course Details

Grading Pattern

5. The **fifth grading Pattern** will be for those Subjects which are entirely Lab based. The breakdown required for the calculation of the Numeric Score (out of 100) for Grading Pattern 5 is given below.

ATTENDANCE	5
WEEKLY ASSIGNMENTS/QUIZZES	20
MID TERM	15
TOTAL INTERNAL	40

FINAL EXAM	60
TOTAL EXTERNAL	60

FOR ADMISSION BATCH 2022-23 PAGE 16 OF 38

Reference Book: Python for Data Analysis: : Data Wrangling with pandas, NumPy and Jupyter by McKinney **The Data Wrangling Workshop by Lipp, Et al., Packt Publishing. Hands-On Data Analysis with Pandas by Molin, Packt Publishing**

Contents

Introduction

What is data science?

Amazon

Data science is the study of data to extract meaningful insights. Data scientists ask and answer questions like what happened, why it happened, what will happen, and what can be done with the results.

Introduction

What is data science?

Amazon

Data science is the study of data to extract meaningful insights. Data scientists ask and answer questions like what happened, why it happened, what will happen, and what can be done with the results.

IBM

Data science combines math, statistics, and specialized programming to uncover hidden insights that can be used to guide decision making and strategic planning.

Introduction

What is data science?

Amazon

Data science is the study of data to extract meaningful insights. Data scientists ask and answer questions like what happened, why it happened, what will happen, and what can be done with the results.

IBM

Data science combines math, statistics, and specialized programming to uncover hidden insights that can be used to guide decision making and strategic planning.

Wikipedia

Data science is an interdisciplinary academic field that uses statistics, scientific computing, scientific methods, processes, algorithms and systems to extract or extrapolate knowledge and insights from noisy, structured, and unstructured data.

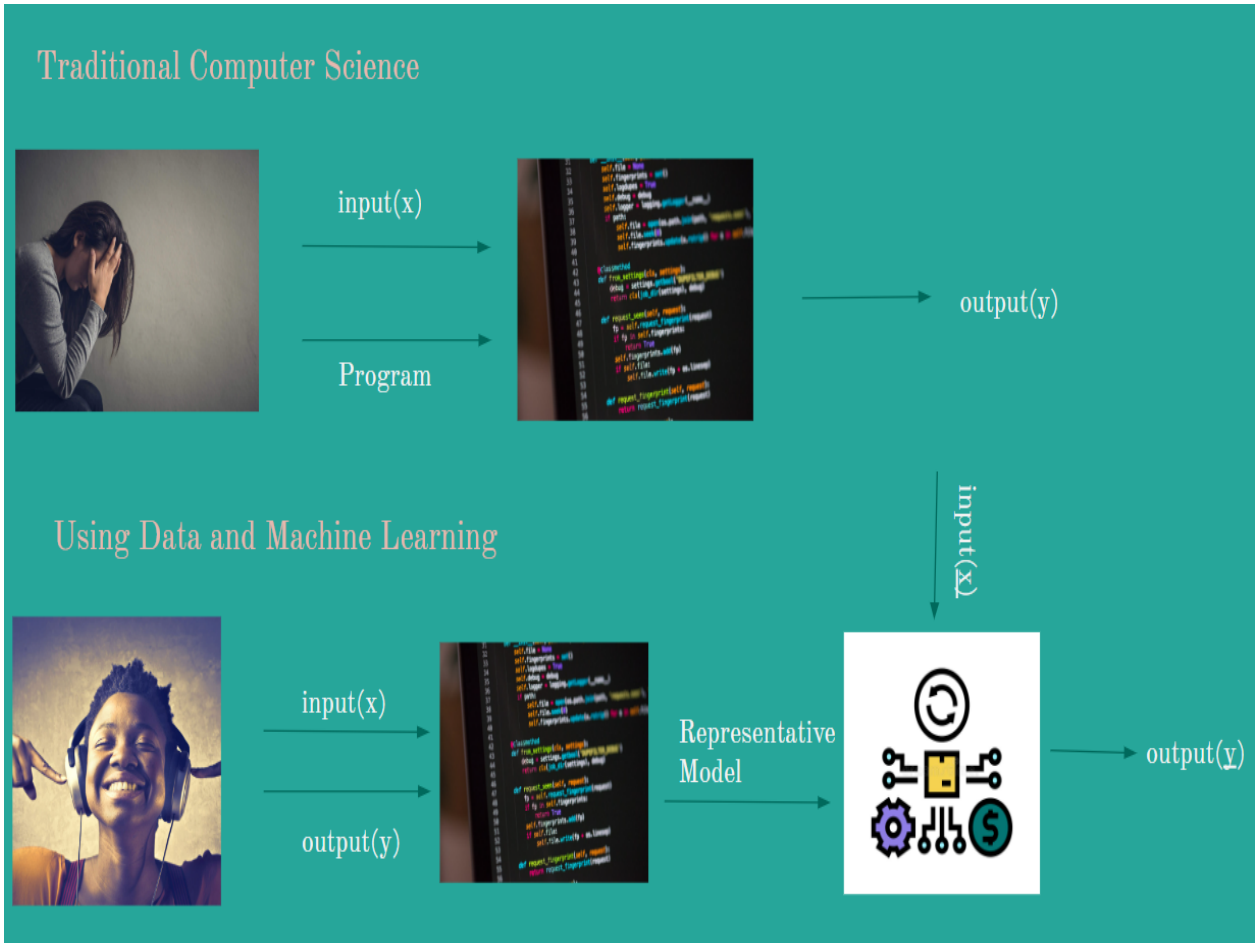
Introduction

Data science is all about getting relevant information about the data.

Introduction

Data science is all about getting relevant information about the data.

Traditional Programming Language Vs Data Science

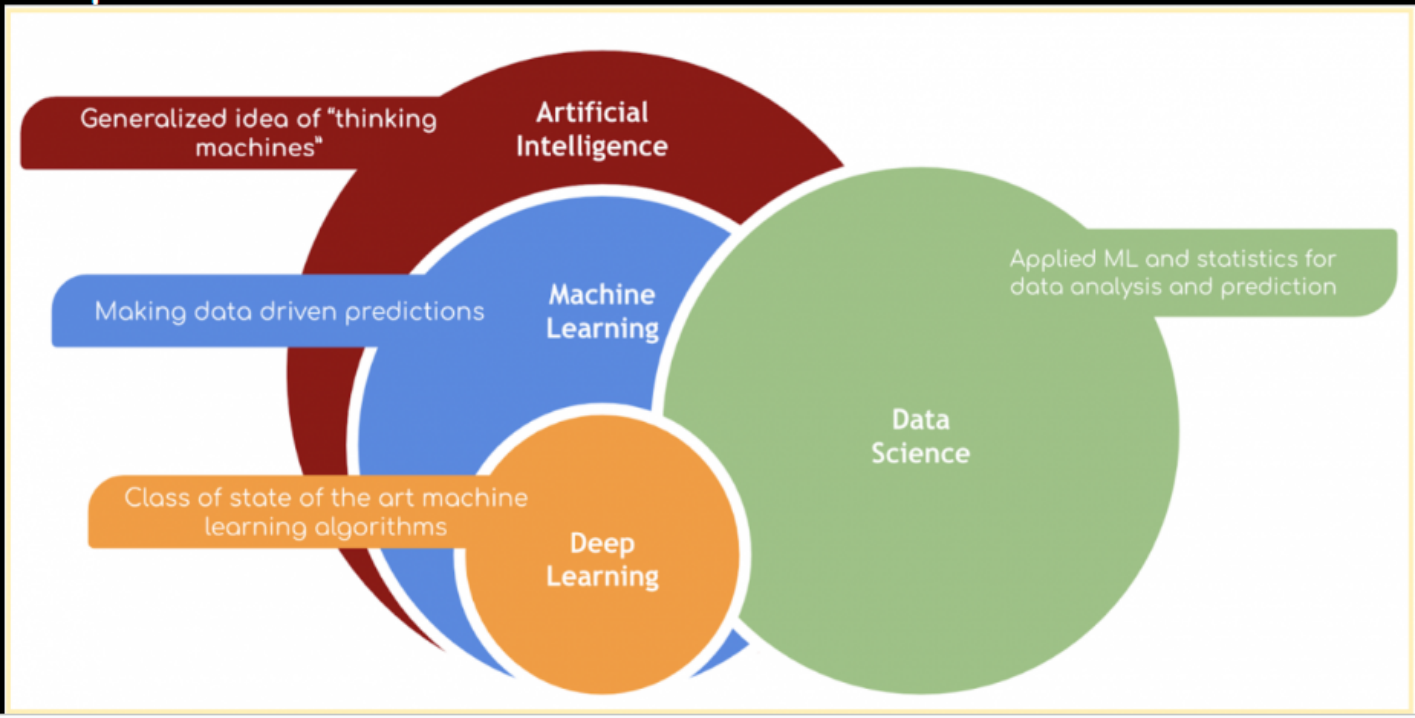


Introduction

- In traditional programming write code, which contains rules and statements needed to transform the provided input into the expected output.
- Rules are not easy to write, mainly when we deal with images and videos.
- In data science we need access to large volumes of data that contain the necessary inputs and their mappings to the expected outputs.
- On those data, we apply algorithms that are generated using mathematical computations to generate rules to map given input to output. This is training.
- Generated rules are a kind of black box, and we cannot understand how the inputs are being transformed into outputs.
- After training, the testing phase comes, where we check the accuracy and other parameters.

AI vs ML vs DL vs DS

- AI is making machines intelligent.
- ML allows computers enabled by algorithms to automatically improve through experience.
- DL uses the concept of neural networks.

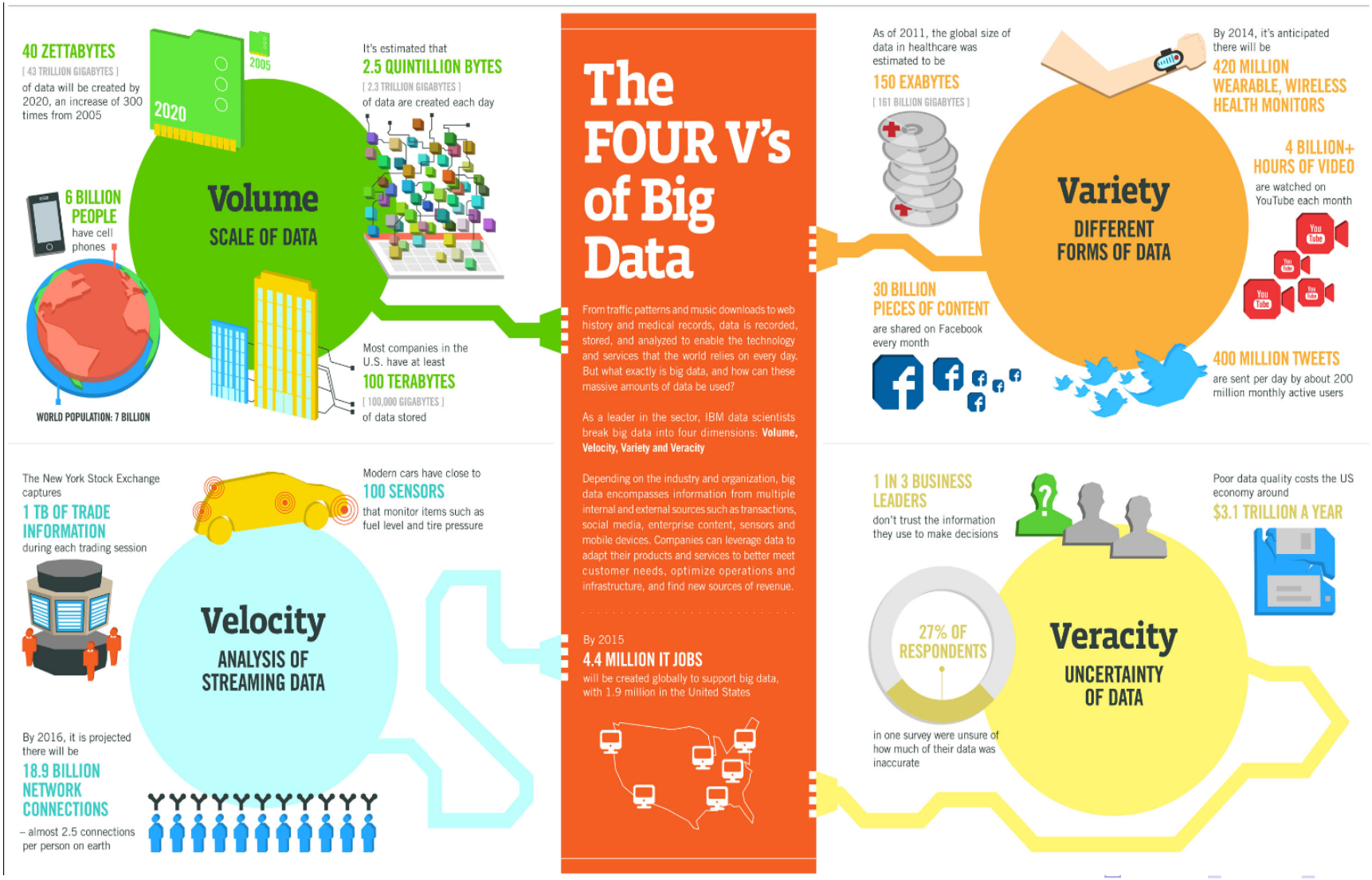


Data Scientist or Data Analyst

Data Scientist - is a combination of various tools, machine learning principles, and algorithm with the aim to find the patterns from the raw data. **Predict future trends** and develop new ways to ask and answer questions about data(SQL,Machine Learning,Visualisation,Storytelling).
Data Analyst - Here data sets are Cleaned(dirty data), Getting basic information, examined to draw conclusions about the information they contain. **Summarize what happened in the past**. (Data Wrangling, Data Mining,SQL,Visualization)
Click here for other roles Data Engineer, ML Engineer,

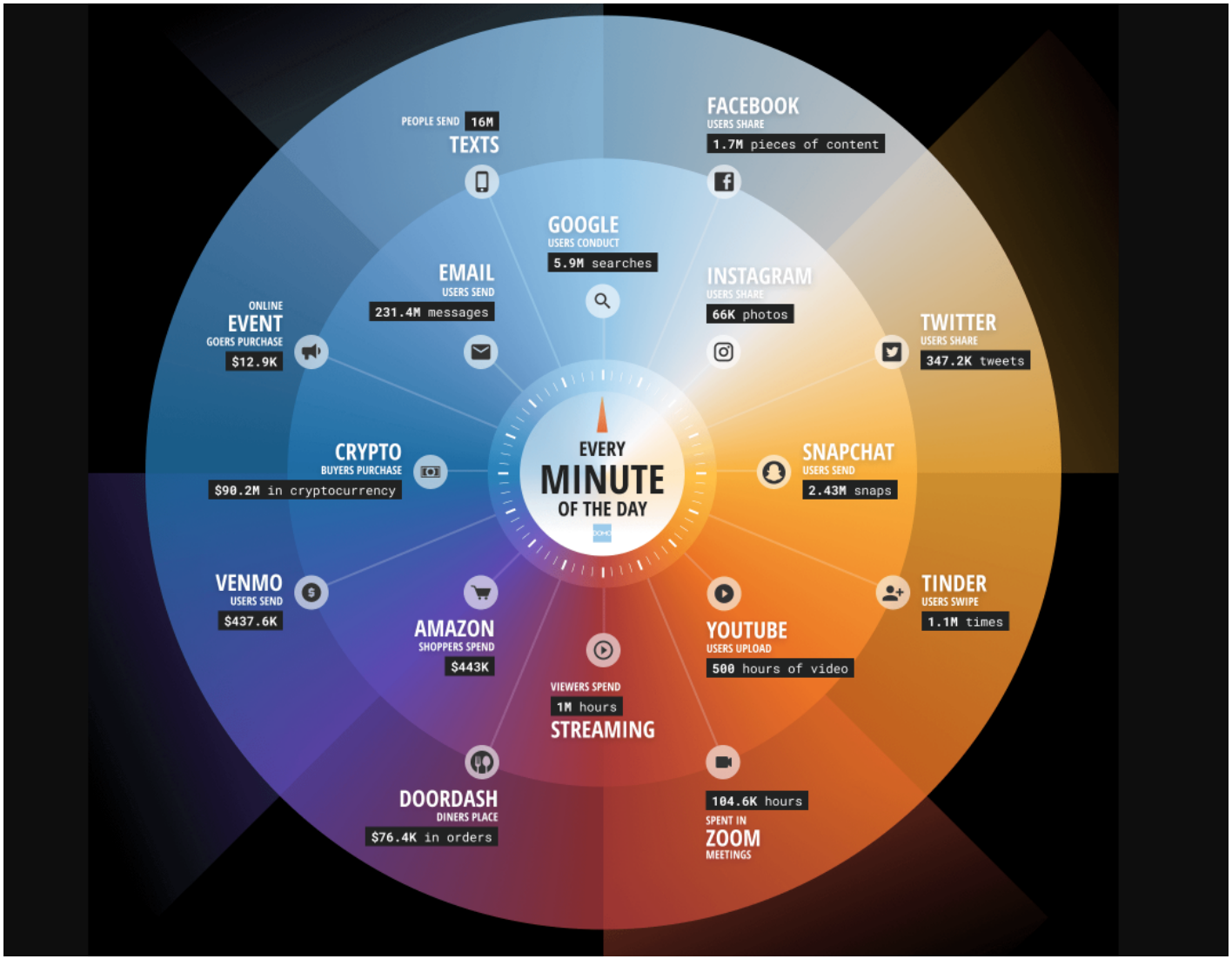
Introduction

Data vs Big data: Value(Usefulness), Variability(Dynamic), Venue(Source)



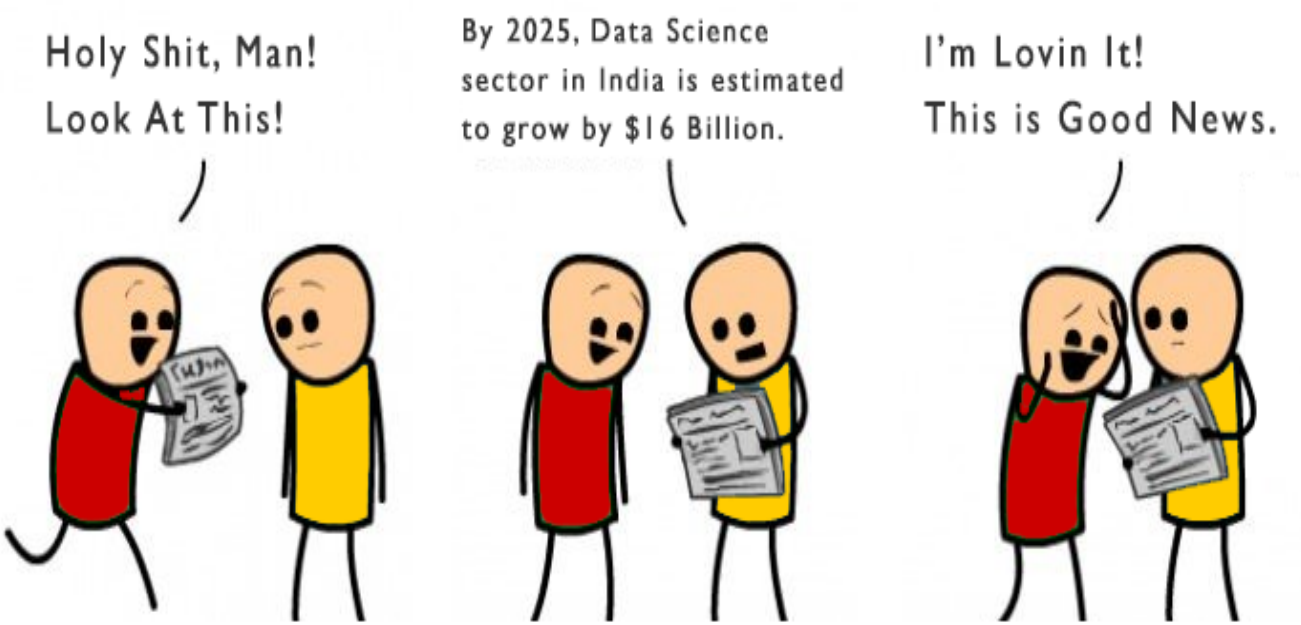
Introduction

How much Data?



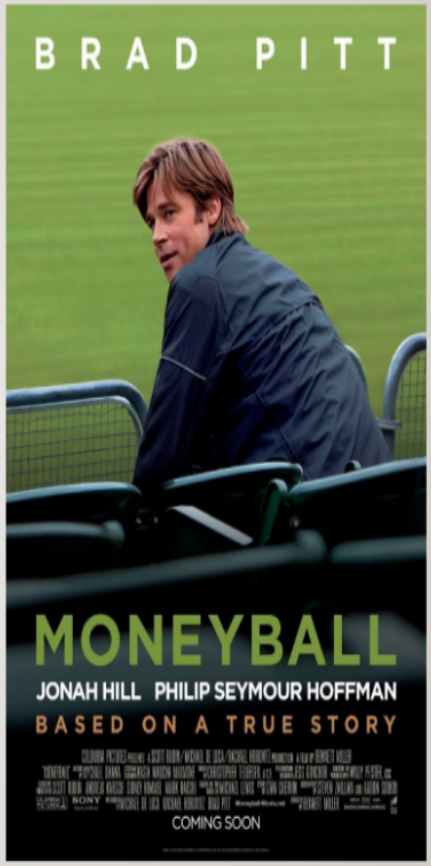
Introduction

Why Data science?



Applications of Data Science

- Moneyball
  - How to build a baseball team on a very low budget by relying on data
  - *Sabermetrics*: the statistical analysis of baseball data to objectively evaluate performance
  - 2002 record of 103-59 was joint best in MLB
    - Team salary budget: $40 million
  - Other team: Yankees
    - Team salary budget: $120 million



Introduction

Applications of Data Science

- Recommender systems
  - The ability to offer unique personalized service
  - Increase sales, click-through rates, conversions, …
    - Netflix recommender system valued at $1B per year
    - Amazon recommender system drives a 20-35% lift in sales annually
  - Collaborative filtering at scale

Introduction

Applications of Data Science

- Fraud detection
  - Investigate fraud patterns in past data
  - Early detection is important
    - Before damage propagates
    - Harder than late detection
  - Precision is important
    - False positive and false negative are both bad
  - Real-time analytics

Introduction

What does a data scientist do?

- Problem Understanding: Understanding the domain and the problem at hand is vital for framing the right questions and selecting appropriate methodologies.
- Data Gathering: The initial phase involves collecting relevant data from various sources, such as databases, APIs, or web scraping.
- Data Cleaning: Often, data comes in messy, incomplete, or inconsistent formats. Data scientists spend a significant portion of their time cleaning and preprocessing the data to ensure its quality.
- Feature Engineering: Crafting meaningful features from raw data is a critical step that significantly impacts the performance of machine learning models.
- Maintenance: Data science is an iterative process, and models need continuous monitoring, updating, and improvement.

Introduction

Python skills required

- Different Data types(numeric, Boolean) and operators(Arithmetic, logical, relational)
- Functions
- Control structures(if-else, for, while)
- Built-in data types.(Strings, lists, dictionaries, tuples and sets).

Introduction

NumPy

- NumPy(Numerical Python): Used for Numerical Computing.
- Contains multidimensional array object called *ndarray*
- Used for Linear algebra operations, Fourier transform etc
- More efficient for storing and manipulating data

Introduction

pandas

- stands for panel data
- *dataframe*: with rows and columns
  series: one dimensional data
- Computing ideas of NumPy+Data manipulation techniques found in data bases
- Helps in reshape, slice and dice, reshape, aggregation

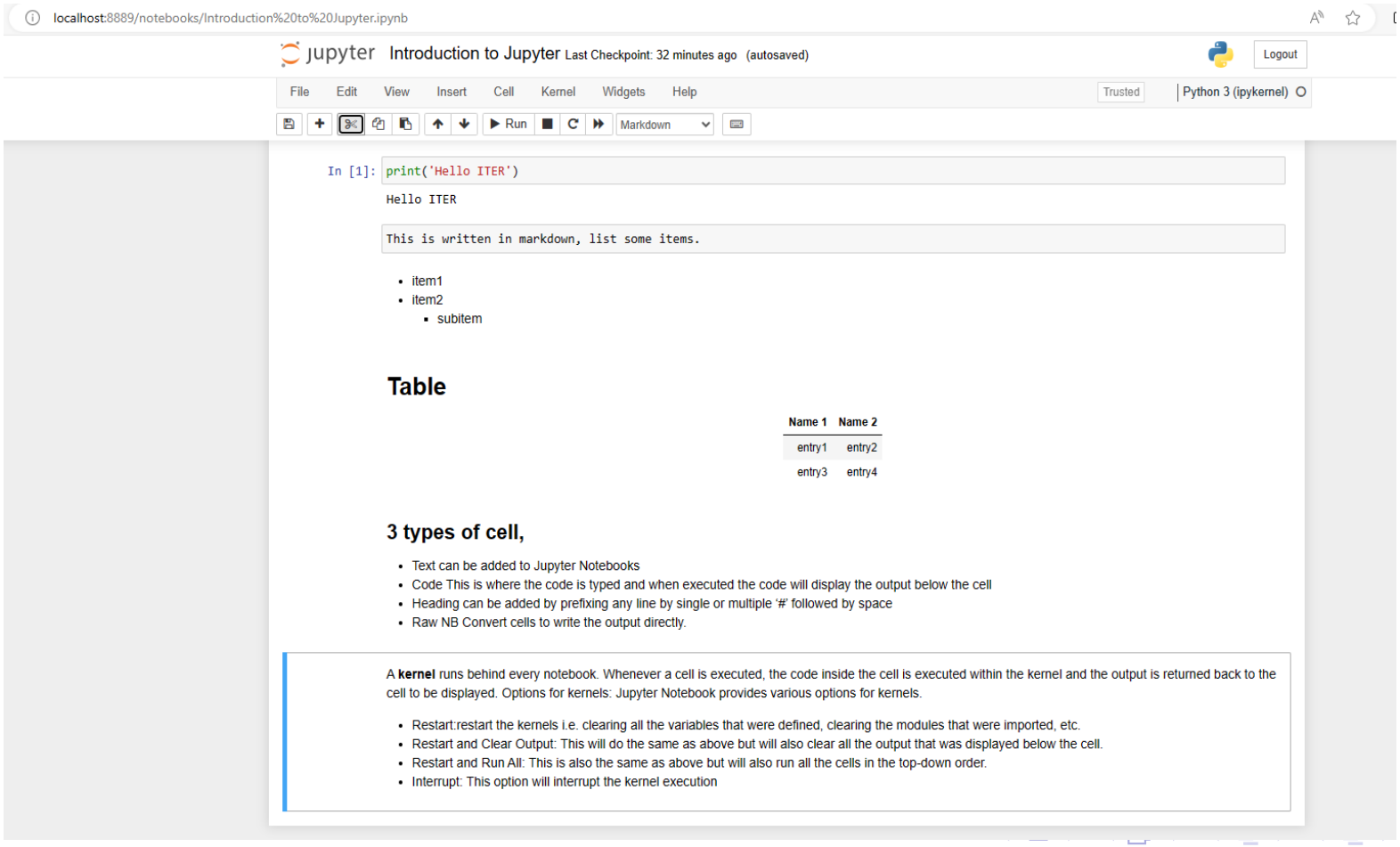
Introduction

Visualization

- For producing plots and other two-dimensional data visualization

Jupyter Notebook

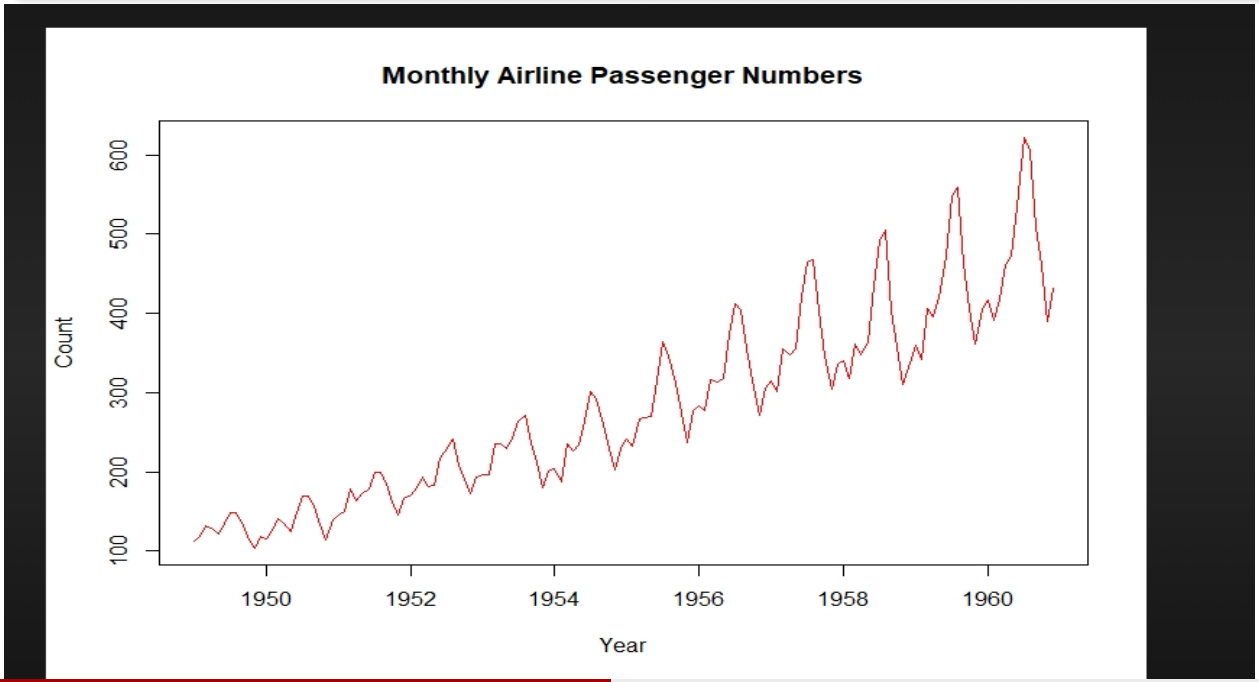
- For interactive Python writing, testing, debugging in one platform



Introduction

Time Series

A time series is a series of data points indexed (or listed or graphed) in time order. Simply put, a time series is a sequence taken at successive equally spaced points in time. In plain language, time-series data is a dataset that tracks a sample over time and is collected regularly. Examples are commodity price, stock price, house price over time, weather records, company sales data, and patient health metrics like ECG.



Python

What is Python?

- Started by Guido Van Rossum, 1991, Labs of National Institute of research in Math and Computer science
- The name was inspired by 'Monty Pythons flying circus'

Python

What is Python?

- Started by Guido Van Rossum, 1991, Labs of National Institute of research in Math and Computer science
- The name was inspired by 'Monty Pythons flying circus'

Why Python?

Why Python?

- Simple Easy to read, write and understand and Easy to code.

Python

Why Python?

- Simple Easy to read, write and understand and Easy to code.
- General Purpose: Different range of applications

Python

Why Python?

- Simple Easy to read, write and understand and Easy to code.
- General Purpose: Different range of applications
- Dynamically Typed: Variable types are determined at run time.

Python

Why Python?

- Simple Easy to read, write and understand and Easy to code.
- General Purpose: Different range of applications
- Dynamically Typed: Variable types are determined at run time.
- Extensive Libraries: Around 1,37,000 libraries and supports external libraries like NumPy and pandas.

Python

Why Python?

- Simple Easy to read, write and understand and Easy to code.
- General Purpose: Different range of applications
- Dynamically Typed: Variable types are determined at run time.
- Extensive Libraries: Around 1,37,000 libraries and supports external libraries like NumPy and pandas.
- Object Oriented Language: Every thing including variables, functions are objects

Python

Why Python?

- Simple Easy to read, write and understand and Easy to code.
- General Purpose: Different range of applications
- Dynamically Typed: Variable types are determined at run time.
- Extensive Libraries: Around 1,37,000 libraries and supports external libraries like NumPy and pandas.
- Object Oriented Language: Every thing including variables, functions are objects
- Interpretd ~~vs Compiled~~: Runs line by line

Python

Why Python?

- Simple Easy to read, write and understand and Easy to code.
- General Purpose: Different range of applications
- Dynamically Typed: Variable types are determined at run time.
- Extensive Libraries: Around 1,37,000 libraries and supports external libraries like NumPy and pandas.
- Object Oriented Language: Every thing including variables, functions are objects
- Interpretd ~~vs Compiled~~: Runs line by line
- Interactive and script mode Write a line execute it before writing next line.(Immediate feedback for each statement)

Interpreted vs Compiled. Can you give a real-world example in this context?

Interpreted vs Compiled

- Let You want to make a chicken biriyani, and the recipe is written in odia, and you don't understand odia. So there are two ways to proceed. The first is if someone had already translated it into English for you. You could read the English version of the recipe and make Biriyani. Think of this translated recipe as the compiled version.
- The second way is if you have a friend who knows odia. When you're ready to make biriyani, your odia friend sits next to you and translates the recipe into English as you go, line by line, and you proceed after each step. In this case, your friend is the interpreter for the interpreted version of the recipe.

Python

Comments

for single line comments
Triple quotes '''multiline''' for multi line comments.

Indentation

The spaces at the beginning of a code line. The way of telling a Python interpreter that the group of statements belongs to a particular block of code.

variable assignment

Name given to a memory location(the objects, so that can be used later. As Python is dynamically typed, so we don't need to specify the type of the variable.
>>>x=10

Python data types

- Numeric Types: int, float, complex
- Sequence Data type: (The ordered collection of similar or different data types) String, list, tuple.
- Boolean True and False
- Set type Set
- Mapping Type Dictionary

Python

Type casting: It is the method to convert the Python variable of one datatype into another data type.

- type() to check the type of an object
- Python Implicit Type Conversion(Python converts the datatype into another datatype automatically. Users don't have to involve in this process)
- Python Explicit Type Conversion
  - int() function take float or string as an argument and returns int type object.
  - float() function take int or string as an argument and return float type object.
  - str() function takes float or int as an argument and returns string type object.

Example of Python type casting

Implicit Type Conversion

```
In [1]: a = 7  #int
        b = 3.0  #float
        c = a + b
        print(c)
        print(type(c))
```

```
10.0
<class 'float'>
```

Explicit Type Conversion

```
In [2]: a = 5
        b = float(a)
        print(b)
        print(type(b))
```

```
5.0
<class 'float'>
```

```
In [3]: a = 5.9
        b = int(a)
        print(b)
        print(type(b))
```

```
5
<class 'int'>
```

```
In [4]: a = 5.9
        b = str(a)
        print(b)
        print(type(b))
```

```
5.9
<class 'str'>
```

Python operators

- A symbol that performs a specific operation between two operands(python variables).
- Arithmetic Operators
  - + Addition >>>x + y
  - - Subtraction >>>x - y
  - * Multiplication>>>x * y
  - / Division >>>x / y
  - // Floor division >>>x // y
  - % Modulus (remainder)>>>x % y
  - ** >>>Exponentiation x ** y
- Comparison operators return a true or false Boolean value

Python operators

- Arithmetic operator
- Comparison operator
  - == If the value of two operands is equal, then the condition becomes true.
  - != If the value of two operands is not equal, then the condition becomes true.
  - <= The condition is met if the first operand is smaller than or equal to the second operand.
  - >= The condition is met if the first operand is greater than or equal to the second operand.
  - <If the first operand is less than the second operand, then the condition becomes true.
  - >If the first operand is greater than the second operand, then the condition becomes true.

Python operators

- Arithmetic operator
- Comparison operator
- Logical Operators
  - and: True if both the operands are true x and y
  - or: True if either of the operands is true x or y
  - not True if operand is false
- Assignment Operator
  - = Assign value of right side of expression to left side operand
  - += Add and Assign: Add right side operand with left side operand and then assign to left operand
  - -= Subtract AND: Subtract right operand from left operand and then assign to left operand

Python Operators

- Arithmetic Operators
- Comparison Operators
- Logical operators
- Assignment operators
- Membership operators
  - in: If the first operand can be found in the second operand(list, tuple, or dictionary), it is evaluated to be true.
  - not in: If the first operand is not present in the second operand, the evaluation is true.

Python Operators

- Arithmetic, Comparison, logical, Assignment, Membership Operator
- Bitwise operators
  - & Bitwise AND: 1 if both are 1 otherwise 0
  - | Bitwise OR: 0 if both are 0 otherwise 1
  - $\sim$ Bitwise NOT: inverts individual bits
  - $\wedge$ Bitwise XOR: 1 if any bit is 1 but not both, otherwise 0
  - $>>$ Bitwise right shift: The left operand's value is moved toward right by the number of bits specified by the right operand.
  - $<<$ Bitwise left shift: The left operand's value is moved toward left by the number of bits specified by the right operand

Example of Python Operators

```
[5]: #exponentiation operator
print(3**2)
#addition
print(4+5)
#subtraction
print(9-8)
#division
print(9/3)
#integer division
print(9//3)
#modulus
print(9%5)
```

```
9
9
1
3.0
3
4
```

```
[7]: print(5>3)
print(3<5)
print(8==9)
print(5<=4)
print(8>=1)
print(7!=9)
```

```
True
True
False
False
True
True
```

```
[13]: print((10<5) and (5<9))
print((10<5) or (9>8))
print(not(5<9))
```

```
False
True
False
```

```
[21]: print(10 & 6)
print(10 | 6)
print(~10)
print(10^6)
print(5<<2)
print(5>>2)
```

```
2
14
-11
12
20
1
```

Precedence of Operators

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Escape sequence

- Escape sequence is a sequence of characters that, when used inside a character or string,
  does not represent itself but is converted into another character or series of characters
  that may be difficult or impossible to express directly
- Frequently used escape characters

\'	Single quote
\\'	Double quote
\\	Backslash
\n	New line
\t	Tab character

- To ignore all the escape sequences in the string, we have to make a string as a raw string using 'r' or 'R'.

Python Strings

- Using single quotes or double quotes or even triple quotes.
- indexing:The indexing of the Python strings starts from 0

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

Trying to change the string, will throw an error

```
str='HELLO PYTHON'
str[2]='I'
```

```
---------------------------------------------------------
TypeError                                 Traceback (mos
~\AppData\Local\Temp\ipykernel_22380\844003175.py in <mo
      1 str='HELLO PYTHON'
----> 2 str[2]='I'

TypeError: 'str' object does not support item assignment
```

- String is an immutable data type, meaning that once you have created a string, you cannot change it

Strings slicing

- Syntax for slicing is [start : stop : step]
- start is the starting index from where to start slicing.
- stop is the ending index or where to sop. End index is stop-1.
- step is the number of steps to jump.
- Default value for start is 0 or -length(String), stop is length(String), step is 1.
- Slicing can be done on strings, arrays, lists, and tuples.

Strings slicing example

```
str="HELLO PYTHON"
print('str[2:5]=',str[2:5])
print('str[:5]=',str[:5])
print('str[2:]=',str[2:])
print('str[:]=',str[:])
```

```
str[2:5]= LLO
str[:5]= HELLO
str[2:]= LLO PYTHON
str[:]= HELLO PYTHON
```

```
str="HELLO PYTHON"
print('str[-5:-2]=',str[-5:-2])
print('str[:-2]=',str[:-2])
print('str[-5:]=',str[-5:])
print('str[:]=',str[:])
```

```
str[-5:-2]= YTH
str[:-2]= HELLO PYTH
str[-5:]= YTHON
str[:]= HELLO PYTHON
```

- Join(concatenate) Two or More Strings using the '+' operator.
  >>>str1='HELLO'
  >>>str2='PYTHON'
  >>>str1+str2(Concatenate operation)
  >>>'HELLO PYTHON'(This is the output)
- Membership Test Check if a substring exists within a string or not, using the keyword in.
  >>>'He' in 'Hello"
  >>>True (Output)
- String Repetition: Using * operator
  >>>"Hi"*5
  >>>'HiHiHiHiHi'

In built functions for Python

- len() method to find the length of a string.
- upper() converts the string to uppercase[Similarly lower(), capitalize(), title(), swapcase()]
- partition() returns a tuple
- replace() replaces substring inside
- find() returns the index of first occurrence of substring[Similarly rfind()]
- strip() removes trailing characters[rstrip(),lstrip()]
- split() splits string in to a list based on delimiter.

In built functions for python

- startswith() checks if string starts with the specified string
- isnumeric() checks numeric characters[islower(),isupper(),istitle()]
- index() returns index of substring
- count() Number of occurrences of a character

Python Strings

Examples

```
S='Hello Bhai, Kaise Ho'
print('len(S):=', len(S))
```

```
len(S):= 20
```

```
S='Hello Bhai, Kaise Ho'
print('S.upper():=',x)
```

```
S.upper():= ('Hello Bhai', ',', ' Kaise Ho')
```

```
S='Hello Bhai, Kaise Ho'
print("S.partition(\",\")=",S.partition(","))
```

```
S.partition(",")= ('Hello Bhai', ',', ' Kaise Ho')
```

```
S='Hello Bhai, Kaise Ho'
print('S.replace(\"Bhai\", \"Behen\")=',S.replace("Bhai", "Behen"))
```

```
S.replace("Bhai", "Behen")= Hello Behen, Kaise Ho
```

```
S='Hello Bhai, Kaise Ho'
print('S.find(\"Kaise\")=',S.find("Kaise"))
```

```
S.find("Kaise")= 12
```

Python Strings

Examples

```
S='   Hello   '
print('S.strip()=',S.strip())
```

```
S.strip()= Hello
```

```
S='Hello! Hello!, Please! Listen to me'
print('S.split(\'!\')=',S.split('!'))
```

```
S.split('!')= ['Hello', ' Hello', ', Please', ' Listen to me']
```

```
S='Hello! Hello!, Please! Listen to me'
print('S.startswith(\'Hello!\'):=',S.startswith('Hello!'))
```

```
S.startswith('Hello!'):= True
```

```
S='Hello! Hello!, Please! Listen to me'
print('S.count(\'Hello!\')=',S.count('Hello!'))
```

```
S.count('Hello!')= 2
```

```
S='Hello! Hello!, Please! Listen to me'
print('S.index(\'Hello!\')=',S.index('Hello!'))
```

```
S.index('Hello!')= 0
```

Python Strings

Examples

```
[21]: s='python is a programming language'
      print(s.capitalize())
```

```
Python is a programming language
```

```
[23]: s='python is a programming language'
      print(s.title())
```

```
Python Is A Programming Language
```

```
[25]: s='python is a programming language'
      print(s.upper())
```

```
PYTHON IS A PROGRAMMING LANGUAGE
```

```
[27]: s='python is a programming language'
      print(s.lower())
```

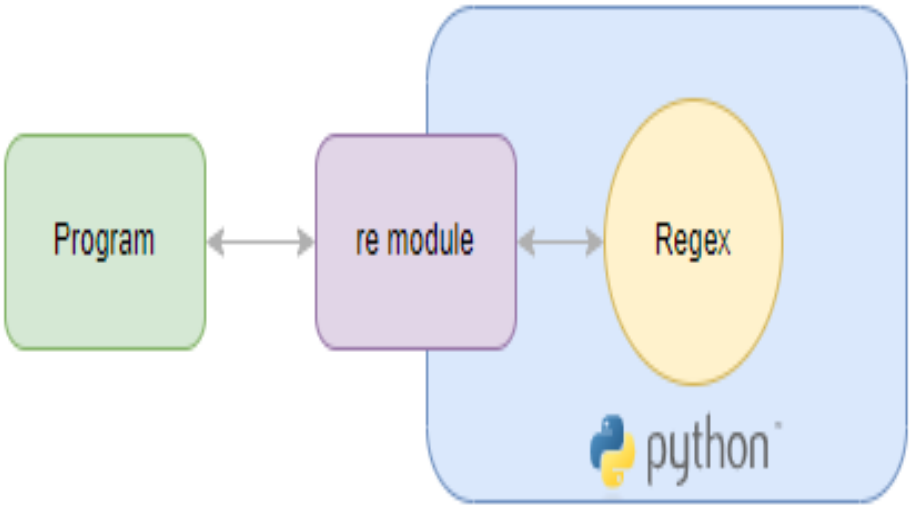
```
python is a programming language
```

Now,
can we check
if the string is
in upper case,
lower case,
title format??

Python Strings

Regular Expressions

- Regular expressions (called regex or regexp) specify search patterns. Typical examples of regular expressions are the patterns for matching email addresses, phone numbers, and credit card numbers.
- Regular expressions are essentially a specialized programming language embedded in Python. And you can interact with regular expressions via the built-in re module in Python.



RegEx MetaCharacters

\	Used to drop the special meaning of character following it.
[ ]	Represent a character class.
^	Matches the beginning of the string.
$	Matches the end of the string or just before the newline character at the end of the string
.	Exactly one arbitrary character excluding the newline.
*	Zero or more occurrences of the preceding pattern.
?	Zero or one occurrence of the preceding pattern.
+	One or more occurrences of the preceding pattern.
{m}	Exactly m occurrences of the preceding pattern.
{m,n}	At least m occurrences and at most n occurrences of the preceding pattern.
[a-z]	A single alphabet in range a to z.Similarly for A-Z, [A-Z]
\d	Any digit.
\s	Whitespace character.
\w	Any alphanumeric character including underscore.

Python Strings

Regular Expression functions

- match()- provides the information about the matched string.
- group()- it belongs to the match object. Returns the substring that matches the regular expression.
- findall()- method returns a list of strings containing all matches.
- split()- method splits the string where there is a match and returns a list of strings where the splits have occurred.
- search()- method takes two arguments: a pattern and a string. The method looks for the first location where the RegEx pattern produces a match with the string.
- sub()- return a new string with occurrences of the pattern replaced by a new string.

Regular Expression MetaCharacters Examples

[12]:
```
import re
# String to check
string = "cat caa"
# Searches if the string matches the pattern
match = re.search('ca*t',string)
# Print result
print(match.group())
```

```
cat
```

[20]:
```
import re
# String to check
string = "cat"
# Searches if the string matches the pattern
match = re.search('ca?t',string)
# Print result
print(match)
```

```
<re.Match object; span=(0, 3), match='cat'>
```

Python Strings

Regular Expression functions

[1]:
```
import re
text= "foo bar\t baz \tqux"
re.split(r"\s+",text)
```

[1]:
```
['foo', 'bar', 'baz', 'qux']
```

[3]:
```
text="""Mansi mansi@google.com
Apurva apurva@gmail.com
Sowmya sowmya@yahoo.com
Vidushi vidushi@gmail.com"""
pattern=r"[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}"
regex= re.compile(pattern,flags=re.IGNORECASE)
regex.findall(text)
```

[3]:
```
['mansi@google.com',
 'apurva@gmail.com',
 'sowmya@yahoo.com',
 'vidushi@gmail.com']
```

[7]:
```
m=regex.search(text)
m
```

[7]:
```
<re.Match object; span=(6, 22), match='mansi@google.com'>
```

[9]:
```
print(regex.sub("Pattern removed",text))
```

```
Mansi Pattern removed
Apurva Pattern removed
Sowmya Pattern removed
Vidushi Pattern removed
```

[21]:
```
pattern= r"([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})"
regex= re.compile(pattern,flags=re.IGNORECASE)
regex.findall(text)
```

[21]:
```
[('mansi', 'google', 'com'),
 ('apurva', 'gmail', 'com'),
 ('sowmya', 'yahoo', 'com'),
 ('vidushi', 'gmail', 'com')]
```

[23]:
```
m=regex.match("aman@bright.net")
m.groups()
```

[23]:
```
('aman', 'bright', 'net')
```

Python Strings

Format string

- String formatting in Python allows you to create dynamic strings by combining variables and values
- There are five different ways to perform string formatting in Python
  - Formatting with % Operator.
  - Formatting with format() string method.
  - Other methods: Formatting with string literals(f-strings), string Template Class, center() string method
- It tells Python how to format a value using a format specifier. The specifier is a sequence of characters that describe a variety of formatting details
- %d or %i – integer %f – float %s – string
- The minimum number of characters to use is placed before the d, i, f or s
- 4.7f indicates that a value should be formatted as a floating-point number using a minimum of 4 characters, including the decimal point and the seven digits to its right.

Python Strings

Format string

```
x=12
print("%d" % x)
```

```
12
```

```
x = 123.454646
print("int is %d float is %2.3f string is %s"%(x,x,x))
#%2.3f at lease 2 before decimal and at most 3 after decimal
```

```
int is 123 float is 123.455 string is 123.454646
```

```
print("%8s" %'Hey')#at least 8 length string
print("%8s" %'Hello World')#at least 8 length string
```

```
     Hey
Hello World
```

```
# Using Curly braces
print("{} and {}".format("Ashis","Namish"))

#Positional Argument
print("{1} and {0}".format("Ashis","Namish"))

#Keyword Argument
print("{a} and {b}".format(a = "Ashis", b = "Namish"))
```

```
Ashis and Namish
Namish and Ashis
Ashis and Namish
```

Python Functions

- function is a piece of code written to carry out a specific task
- Sometimes we bundle a set of instructions that we want to use repeatedly or sometimes because of their complexity, we make a sub-program and call when needed.Readability, and Reusability
- Three types of functions in Python:
  - Built-in functions, written by python developers
  - User-Defined Functions (UDFs), which are functions that users create to help them out
  - Anonymous functions, which are also called lambda functions.

Builtin Functions

built-in functions

- print() Prints to the standard output device
- input() Allowing user input
- abs() Returns the absolute value of a number
- eval() Evaluates and executes an expression
- type() gives type of an object.
- float(), int(), str() Returns a floating point number,integer, string respectively[similarly set(), list(), tuple()]
- id() Returns the id of an object
- range() Returns a sequence of numbers, starting from 0 and increments by 1 *by default*.
- min() and max() Return minimum, maximum among all elements
- open() open file and read corresponding file object.
- ord() return an integer representing the Unicode code point of that character, inverse of the chr()
- map(), filter(), reduce() takes a function and iterable.

All Builtin-functions

Built-in Functions

A
abs()
aiter()
all()
anext()
any()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delattr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

V
vars()

Z
zip()

_
__import__()

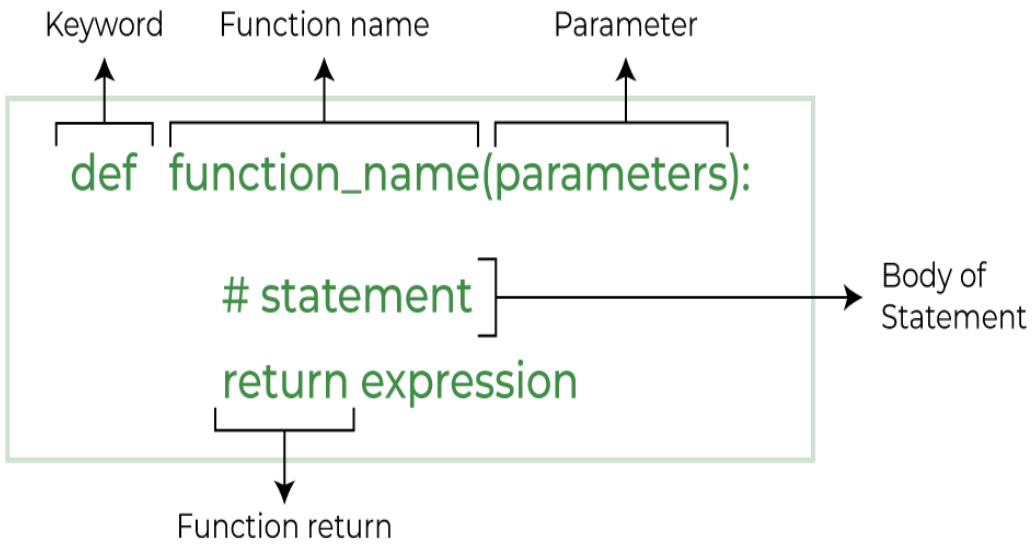
User defined functions

The four steps of defining functions

- Use the keyword def to declare the function and follow this up with the function name.
- Add parameters(at most 255) to the function: they should be within the parentheses of the function. End your line with a colon.
- Add statements that the functions should execute. The bodies of functions are indented.
- End your function with a return statement if the function should output something. Without the return statement, your function will return an object None.

User defined functions

```
def name_of_the_function(parameter1,par2):
    works the function
    should do for you
    return statement
    anything after return
    has nothing to do
```



User Defined functions

return statement

- If you want to continue to work with the result of your function and try out some operations on it, you will need to use the return statement
- Using return statement we can return multiple values, we have to use tuples.

User defined functions

Return vs none-type

```
def f(a,b):
    print(a+b)
def g(a,b):
    return a+b
print('5+g(1,2)=',5+g(1,2))
print('5+f(1,2)=',5+f(1,2))
```

```
5+g(1,2)= 8
3

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17276\4025539089.py in <module>
      4     return a+b
      5 print('5+g(1,2)=',5+g(1,2))
----> 6 print('5+f(1,2)=',5+f(1,2))

TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

```
def f(a,b):
    return (a+b,a-b)
print(f(1,2))
```

```
(3, -1)
```

User defined functions

Function arguments are of four types

- Default arguments are values that are provided while defining functions. So they become optional during the function calls. If we provide a value to the default arguments during function calls, it overrides the default value.

User defined functions

Function arguments are of four types

- Default arguments are values that are provided while defining functions. So they become optional during the function calls. If we provide a value to the default arguments during function calls, it overrides the default value.
- Required arguments that have to be in there. These arguments need to be passed during the function call and in precisely the right order

User defined functions

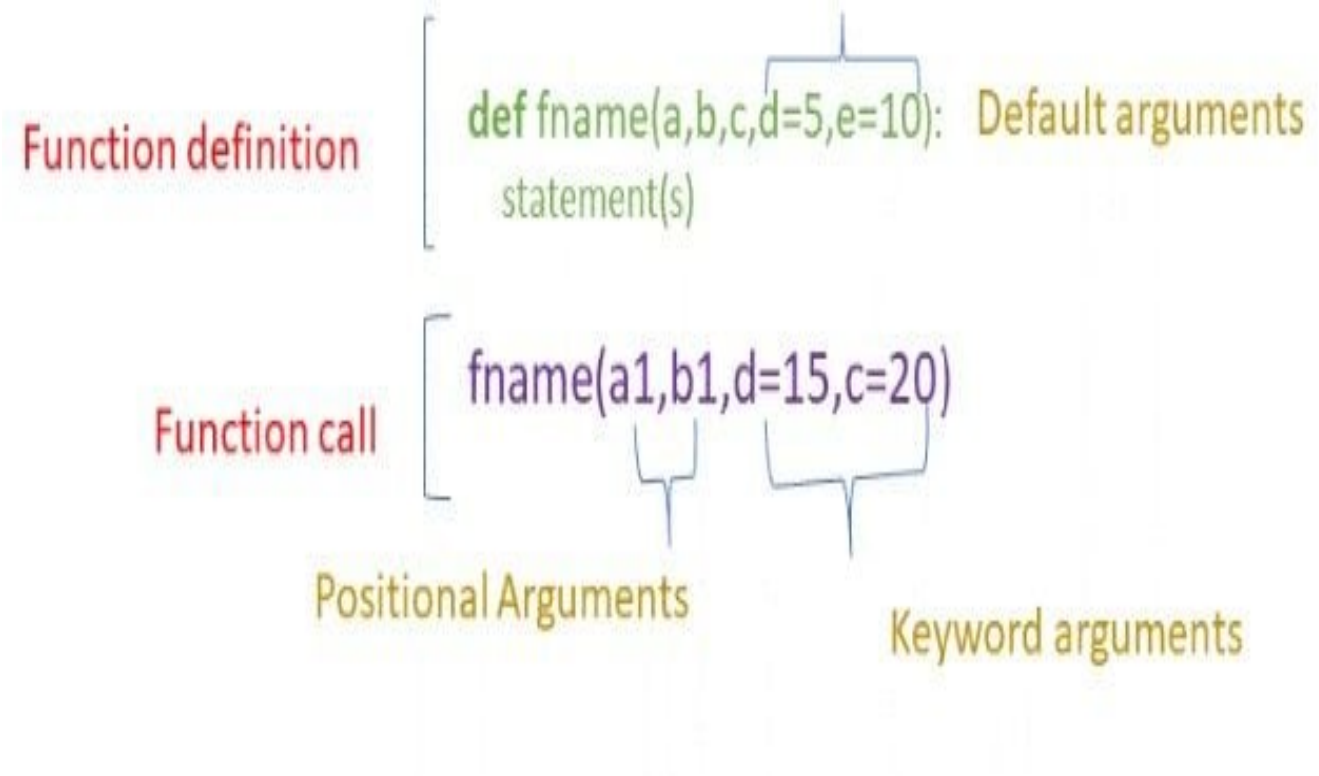
Function arguments are of four types

- Default arguments are values that are provided while defining functions. So they become optional during the function calls. If we provide a value to the default arguments during function calls, it overrides the default value.
- Required arguments that have to be in there. These arguments need to be passed during the function call and in precisely the right order
- Keyword arguments: During a function call, we use these to identify the arguments by their parameter name. Values passed through arguments don't need to be in the order of parameters in the function definition.
- Variable number of arguments In cases where you don't know the exact number of arguments that you want to pass to a function, you can use *args(Arbitrary positional arguments), **kwargs(arbitrary keyword arguments).

Things to remember about functions

arguments

In function definition Default arguments should come after non-default(positional or required) arguments.
While calling the function keyword arguments should come after positional arguments.



Things to remember about functions

Summary:

- default arguments should follow non-default arguments

Things to remember about functions

Summary:

- default arguments should follow non-default arguments
- keyword arguments should follow positional arguments

Things to remember about functions

Summary:

- default arguments should follow non-default arguments
- keyword arguments should follow positional arguments
- All the keyword arguments passed must match one of the arguments accepted by the function and their order is not important.

Things to remember about functions

Summary:

- default arguments should follow non-default arguments
- keyword arguments should follow positional arguments
- All the keyword arguments passed must match one of the arguments accepted by the function and their order is not important.
- No arguments should receive a value more than once.

Things to remember about functions

Summary:

- default arguments should follow non-default arguments
- keyword arguments should follow positional arguments
- All the keyword arguments passed must match one of the arguments accepted by the function and their order is not important.
- No arguments should receive a value more than once.
- Default arguments are optional arguments.

Things to remember about functions

Summary:

- default arguments should follow non-default arguments
- keyword arguments should follow positional arguments
- All the keyword arguments passed must match one of the arguments accepted by the function and their order is not important.
- No arguments should receive a value more than once.
- Default arguments are optional arguments.
- Orders of Positional arguments is important.

Why do we use *args?

```
>>> def person(first_name):
...     print(first_name)
>>> person("Ram")
Ram
>>> person()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: person() missing 1 required positional argument:
'first_name'
```

*args

```
def add(*b):
    result=0
    for i in b:
        result=result+i
    return result


print (add(1,2,3,4,5))
#Output:15
print (add(10,20))
#Output:30
```

**kwargs

```
def fn(**a):
    for i in a.items():
        print (i)
fn(numbers=5,colors="blue",fruits="apple")
'''
Output:
('numbers', 5)
('colors', 'blue')
('fruits', 'apple')
'''
```

Lambda(Anonymous) functions

Lambda(Anonymous) functions

- Anonymous functions are also called lambda functions in Python because instead of declaring them with the standard def keyword, you use the lambda keyword.
- A lambda function can take any number of arguments, but can only have one expression.

syntax

lambda arguments: expression

Example

square = lambda x: x * x
sum = lambda x,y: x+y
Max = lambda x, y : x if(x>y) else y

User defined functions

__name__=='__main__'

- Functions from an old Python program can be imported into a new one using the import keyword, followed by the name of the Python file that contains the functions of interest (without the .py extension). This allows the new program to call all of the functions in the old file, but it also causes the program in the old file
- While this may be desirable in some situations, we often want access to the old program's functions without actually running the program. This is nor mally accomplished by creating a function named main

Date time module

- A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

```
#print the current date and time
from datetime import datetime
#calling the now function
now=datetime.now()
print('Current date and time is:',now)
```

```
Current date and time is: 2023-10-06 10:43:21.293122
```

```
print('Current year is:',now.year)
print('Current month is:',now.month)
print('Current date is:',now.day)
print('Current hour is:',now.hour)
print('Current minute is:',now.minute)
print('Current second is:',now.second)
print('Current microsecond is:',now.microsecond)
```

```
Current year is: 2023
Current month is: 10
Current date is: 6
Current hour is: 10
Current minute is: 43
Current second is: 21
Current microsecond is: 293122
```

Date time module

```
# print current date only
from datetime import date
# calling the today function
today = date.today()
print("Today's date is", today)
```

```
Today's date is 2023-10-06
```

```
now2=datetime.now()
now2-now
```

```
datetime.timedelta(microseconds=49539)
```

- strftime(): date objects into readable strings.
- takes one parameter, format to specify the format of the returned string

```
#24-hour format
print(now.strftime('%Y/%m/%d %H:%M:%S'))
```

```
2023/10/06 10:43:21
```

```
#12-hour format
print(now.strftime('%Y/%m/%d %I:%M:%S'))
print(now)
```

```
2023/10/06 10:43:21
2023-10-06 10:43:21.293122
```

```
from datetime import datetime
now = datetime.now()
print ('%02d/%02d/%04d' % (now.month, now.day, now.year))
```

```
10/06/2023
```

random module

- used to generate random numbers in Python
- randint() Returns a random integer within the range
- random() Generate random floating numbers
- randrange((start, stop, step) Returns a random number within the range(step has to be an integer)
- shuffle(seq) Shuffle the sequence
- choice(seq) Returns a random item from a sequence
- sample(sequence, k) Return a k length list of unique elements chosen from the sequence.
- seed(): If the seeding value is same, the sequence will be the same
- normalvariate() Return a random floating point number with normal distribution

Random Module

```
import random
random.seed(1)
x=random.randint(1,5)
y=random.random()
z=random.randrange(1,5,2)
print('x={},y={},z={}'.format(x,y,z))
```

```
x=2,y=0.5692038748222122,z=1
```

```
random.seed(1)
x=random.randint(1,5)
print('x={}'.format(x))
```

```
x=2
```

```
L=[1,2,3,4,5]
random.shuffle(L)
print(L)
```

```
[2, 1, 3, 5, 4]
```

random module

```
a=random.choice(L)
b=random.choice(L)
c=random.choice(L)
print([a,b,c])
```

```
[1, 1, 5]
```

```
A = [20, 40, 80, 100, 120]
B = random.sample(A, 3)
print(B)
```

```
[100, 40, 120]
```

file handling

created a text file sample.txt and saved it in the same location.

sample.txt:

This file has two lines.

This is the second line.

```
f = open("sample.txt", "r")
print(f.read())
my_file.close()
```

```
This file has two lines.
This is the second line.
```

file handling

```
f = open("sample.txt", "a")
f.write("\nAs we  have opened in append mode")
f.write('\nSo new lines will be added to the file')
f.close()
f = open("sample.txt", "r")
print(f.read())
my_file.close()
```

```
This file has two lines.
This is the second line.
As we  have opened in append mode
So new lines will be added to the file
```

```
f = open("sample.txt", "w")
f.write("\nOpening in w mode will overwrite the file")
f.close()
f = open("sample.txt", "r")
print(f.read())
my_file.close()
```

```
Opening in w mode will overwrite the file
```

file handling

[23]:
```
f=open('Python','w')
f.write('Python is a programming language. It is easy and simple. We can use it in AI, Information Security')
f.close()
```

[25]:
```
f=open('Python','r')
f.tell()
```

[25]:
```
0
```

[27]:
```
f.read(4)
```

[27]:
```
'Pyth'
```

[29]:
```
f.tell()
```

[29]:
```
4
```

[31]:
```
f.seek(0)
```

[31]:
```
0
```

file handling

[35]:
```
f=open('Python','w')
f.write('Python is a programming language. \nIt is easy and simple. We can use it in AI, Information Security')
f.close()
```

[47]:
```
f=open('Python','r')
f.readline()
```

[47]:
```
'Python is a programming language. \n'
```

[49]:
```
f.readlines()
```

[49]:
```
['It is easy and simple. We can use it in AI, Information Security']
```

[51]:
```
f=open('python1','w')
des=['python: \n','Python is an interative programming language. \n']
f.writelines(des)
f.close()
```

[53]:
```
f=open('python1','r')
f.read()
```

[53]:
```
'python: \nPython is an interative programming language. \n'
```

[55]:
```
f.close()
```

try..except..

- Error in Python can be of two types i.e. Syntax errors and Exceptions.
- Some of the common Exception Errors are :
  - IOError: if the file can't be opened
  - KeyboardInterrupt: when an unrequired key is pressed by the user
  - ValueError: when the built-in function receives a wrong argument
  - EOFError: if End-Of-File is hit without reading any data
  - ImportError: if it is unable to find the module
  - NameError: If a variable is used, which is not defined before
- Try and Except statement is used to handle these errors within our code in Python
- The try block lets you test a block of code for errors.
- The except block lets you handle the error.

try-except example

```
print(x)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11688\1353120783.py in <module>
----> 1 print(x)

NameError: name 'x' is not defined
```

```
try:
    print(x)
except NameError:
    print('Some variable is not defined')
```

```
Some variable is not defined
```

try-except example

```
[11]: #type error
      print('Sum of 6 and 4 is:'+10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 2
      1 #type error
----> 2 print('Sum of 6 and 4 is:'+10)

TypeError: can only concatenate str (not "int") to str
```

```
[13]: 78/(2+3-5)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[13], line 1
----> 1 78/(2+3-5)

ZeroDivisionError: division by zero
```

```
[15]: nos=[1,2,3,4]
      nos[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[15], line 2
      1 nos=[1,2,3,4]
----> 2 nos[5]

IndexError: list index out of range
```

try-except

- The else block lets you execute code when there is no error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.

```
try:
    k = 5//0 # raises divide by zero exception.
    print(k)

# handles zerodivision exception
except ZeroDivisionError:
    print("Can't divide by zero")

finally:
    # this block is always executed
    # regardless of exception generation.
    print('This is always executed')
```

```
Can't divide by zero
This is always executed
```

```
try:
    k = 5//2 # raises divide by zero exception.
    print(k)
except ZeroDivisionError:
    print("Can't divide by zero")
else:
    print('No exception Exception raised in try block')
finally:
    print('This is always executed')
```

```
2
No exception Exception raised in try block
This is always executed
```