BASICS OF PYTHON

WHAT IS PYTHON?????

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
 It was created by Guido van Rossum, and released in 1991.

ADVANTAGES OF PYTHON:-

- Python can be used on a server to create web applications.

- Python can connect to database systems. It can also read and modify files.

- Python can be used to handle big data and perform complex mathematics.

- Python can be used for rapid prototyping, or for production-ready software development.

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

- Python has a simple syntax similar to the English language.

- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

- Python runs on **an interpreter system**, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

- Python can be treated in **a procedural way, an object-oriented way or a functional way.**

COMPILER VS INTERPRETER

S.No.	Compiler	Interpreter
1.	Compiler scans the whole program in one go.	Translates program one statement at a time.
2.	As it scans the code in one go, the errors (if any) are shown at the end together.	Considering it scans code one line at a time, errors are shown line by line.
3.	Main advantage of compilers is it's execution time.	Due to interpreters being slow in executing the object code, it is preferred less.
4.	It converts the source code into object code.	It does not convert source code into object code instead it scans it line by line
5	It does not require source code for later execution.	It requires source code for later execution.
Eg.	C, C++, C# etc.	Python, Ruby, Perl, SNOBOL, MATLAB, etc.

Source Code → Preprocessing → Object Code → Processing → Machine

Figure: Compiler

Source Code → Preprocessing → Intermediate Code → Processing → Interpreter

Figure: Interpreter

INSTALL PYTHON:

MY FIRST PROGRAM:

print("Hello World")

IDENTIFIER IN PYTHON ?

◦ Python Identifiers are defined as name as assigned to class, function, variables etc.

◦ It helps to differentiate between entities.

What are the rules for writing identifiers properly?

1. They can be group of letters in lowercase character (a to z) or uppercase character (A to Z) or digits (0 to 9) or an underscore "_" . For example: MyFirstProgram, Program_1, etc.

2. Identifiers start with a digit is invalid. Example: 1Program is not valid but Program1 is valid.

3. Don't use keywords to make identifiers.

4. Doesn't use special symbols in identifiers. Such as ! @ $ # : etc.

5. It can be of any length.

6. As Python is case sensitive language. Program1 and program1 will have not be same.

Way of good Naming Conventional for Identifiers:-

1. Use camelCase to write a identifiers, i.e. camelCase is naming convention in which name is form of many different word a with internal capital letters. Example: myProgram, myAddress, etc.

2. Use general words that make more readable and easier to figure out in future.

3. Separate multiple words with underscore. Example program_one, my_Data, etc.

COMMENTS IN PYTHON: Comments makes the code more readable.

◦ Comments use to explain the code what is written in it.

◦ It is to notate your code to remember what type of code is it, how many types of functions, classes we used there for others to understand easily.

◦ It simply begin with **#(hash)** character. Compiler ignores everything after hash. Make comments to the point.

◦ Also another way to **make multi-line comment** is by wrapping your comment inside a set of triple quotes """ .. """.

◦ Example: #this is Python language comment.

VARIABLES: A variable in Python called as a container which store data in it and gives data to the computer to further processing. Every variable have their own data types.

Declaration of a variable:

◦ The declaration happens automatically when we initialize a value to a variable.

◦ By the help of equal sign (=) we are able to assign value to variable.

◦ Also remember, "Equal to" is a operator which is use to assign values in variable.

◦ **Example: a=10   print(a)**

Rules for variables

◦ To make appropriate variable names, there are some rules and guidelines that have to follow. If rules and guideline breaks, it will generate error.

◦ This just helps to make code more readable. Try to remember this rules in your mind.

1. A variable name must start with a alphabets (lowercase or uppercase) or the underscore character. Ex – abc, Abc, ABC, _abc.

2. A variable name cannot start with a number. Ex – abc123 is valid but 123abc is not valid that will generate error.

3. A variable name can only have alpha-numeric characters and underscores (A-z, 0-9, and _). Ex- abc_199, A12, __h420.

4. They are case-sensitive (my and My are different variables). Here case sensitive means "A" and "a" both are different in letter cases, i.e. uppercase and lowercase are not going to consider equal.

5. Spacing is not allowed between the words of variable names, but we can use underscore to separate words as per our understanding or distinguish variables. Ex- abc 123 is wrong naming of variable but abc_123 is correct.

If we miss-spell the variable name. Here Python Interpreter shows error.

1) Trackback (most recent call last):

2) 2) File "to_print.py", line 2, in

3) 3) NameError: name 'tprint' is not defined

- First line is **a trackback.** A trackback is provided by the interpreter when program is unable to run. A trackback is a **record of troubles or error which interrupts the interpreter** to execute a program successfully.

- Second line reports error in line 2 of file name "---". And the interpreter spots the line's error and tells which type of error is found in line x.

- The error which comes out is name error which reports the variable name is not defined in the range of program.

DATA TYPES: Data types are find the types and size of data stored in variables. It holds different types of data value in it. Since it is a dynamically typed language, hence we need not define the type of the variable while declaring it.

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

Python - Data Types

Numeric — Dictionary — Boolean — Set — Sequence Type

Interger — Float — Strings — Tuple

Complex Number

List
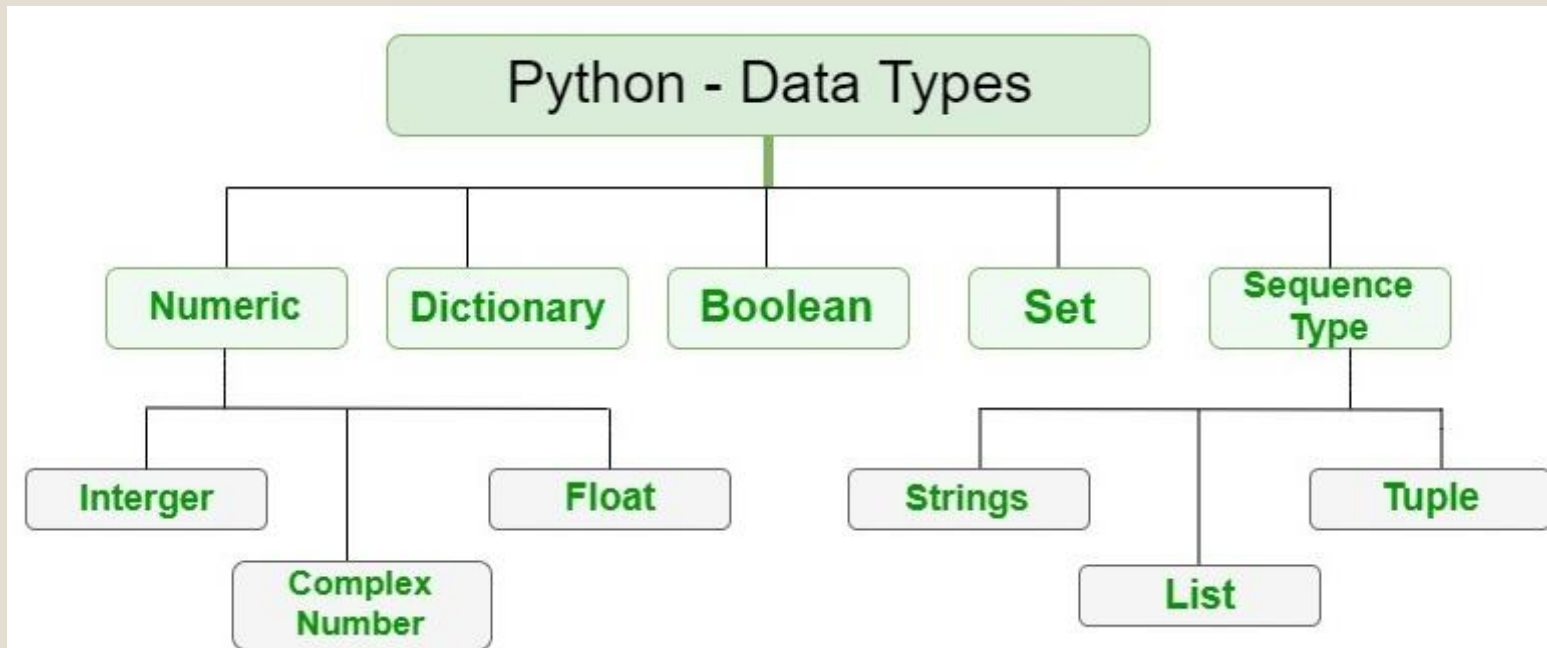
PYTHON INDENTATION:

Most of the programming languages like C, C++, and Java use braces { } to define a block of code. Python, however, uses indentation.

In simple terms indentation refers to adding white space before a statement.

Without indentation, Python does not know which statement to execute next or which statement belongs to which block. This will lead to IndentationError.

Python indentation is a way of telling a Python interpreter that the group of statements belongs to a particular block of code.

Python program showing

indentation

site = 'Hello Trainees'

if site == ' Hello Trainees ':

       print('Logging on to training!!!!')

else:

       print('retype the URL.')

print('All set !')

```
Statement
if condition:
        if condition:
                Statement
        else:
                Statement
Statement
```

How the interpreter
visulaises

→

```
Code block 1 begins
Code block 1 continues
        Code block 2 begins
                Code block 3 begins
        Code block 2 continues
                Code block 3 continues
Code block 1 continues
```

Indentation can be ignored in line continuation, but it's always a good idea to indent. It makes the code more readable. For example:

```
if True:
    print('Hello')
    a = 5
```

both are valid and do the same thing, but the former style is clearer.

```
if True: print('Hello'); a = 5
```

RESERVED WORDS:

◦ The reserved words or special words in compiler called keywords.

◦ They define structure and pattern of Python Programming.

◦ They are case sensitive.

◦ Keywords may change in future as Python is dynamic language.

◦ If we want to check how many keywords are there we use some lines of code, write this following lines in interpreter and see the magic—

◦ >>> import keyword

◦ >>> print (keyword.kwlist)

◦ >>> print(keyword.iskeyword("False"))

PYTHON AS CALCULATOR

There are various types of Math Functions available that are commonly use to perform basic operations on numbers. In Python, we have 7 standard mathematics operations:-

1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Int. Division (//)
6. Modulo (%)
7. Power or Exponential (**)

```
>>> 2+3
5
>>> 2-3
-1
>>> 3-2
1
>>> 2*45
90
>>> 40/5
8.0
>>> 40//5
8
>>> 3**4
81
>>>
```

Note: Also there is optional method in Math Library i.e. mat.pow() is use to calculate the power/exponential but ** is easier to calculate power. So, we prefer ** more than mat.pow() here.

Import math

P=math.pow(2,4)

Print(p)

ORDER OF PRECEDENCE:-

When 2 or more than 2 operators comes in an expression, to evaluate the correct answer we have to follow set of rules. Python have precedence rules for its mathematics.

"B E M D A S"

1. B means BRACKET or PARENTHESES. It has the highest order of precedence and with this we can force the expression to give answer as we want. Example: 2* (3-1) is 4 but (2*3)-1 is 5 both evaluate different values because of parentheses arrangement.

2. E means EXPONENTIATION. It has second order of precedence.

3. Then M means MULTIPLICATION and D means DIVISION, both have the same order of precedence.

4. And A means ADDITION and S means SUBTRACTION, also both have the same precedence order.

Note   One problem arises here if same precedence of operator occurs in a given expression then what to do? Operators who have same precedence order are **evaluated from Left-to-right**.

Also we can say that Left-associative. But in **Exponential operator have Right-to-left** in Python.

>>3**2**2

>>> 81

It will solve right-to-left so first 2**2 will evaluate i.e. 4 then 3**4 is going to evaluate answer 81.

OPERATORS:-

In Python, Operators are standard symbols or character which are used to perform/manipulates the values of operand or variable and computes values.

Operand may be constant values or a variable in which we assigned values to it.

Python have 7 different types of operators in the following groups:-

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Bitwise operators
6. Membership operators
7. Identity operators

Also in other programming languages we have studied the above 5 operators but **the last two operators are the exclusive to python i.e. Membership and Identity operators.**

1. ARITHMETIC OPERATORS: Arithmetic operators are use to perform mathematical operations with numerical values. Example: +, -, **, /, //

Operator	Meaning	Example
+	Add two operands or unary plus	x + y+ 2
-	Subtract right operand from the left or unary minus	x - y- 2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

2. ASSIGNMENT OPERATORS Assignment operators are use to assign values to declared variables.
Example: =, +=, **=, etc.

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

3. COMPARISON OPERATORS Comparison operators are use to compare or differentiate two values.
Example : ==, !=, , etc.

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

4. LOGICAL OPERATORS

Logical operators are use to combine conditional statements together.

Example: and, or, not, etc. The logical operators are as follow:-

AND (When both operands are true , it gives true.)

OR (When any operand is true, it gives true.)

NOT (It is used to reverse the operand state.)

5. BITWISE OPERATORS Bitwise operators are used to

compare numbers by converting it into binary form.

Example : &, |, ^, etc.

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

6. MEMBERSHIP OPERATORS Membership operators are use to predict if a sequence is presented in an object or not. Example : in, not in.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

7. IDENTITY OPERATORS: **is** and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

INPUT AND OUTPUT IN PYTHON

Python provide with large number of built-in functions available to us.

print() function: This print() function is use to give output of the data which is inside the bracket to the standard output screen.

Example:- print("hello everyone")

hello everyone

 x=5

print("value of x is",x)

value of x is 5

Input() function: this input function use to take input from the user.

SYNTAX var_name=input([prompt])

X=input()

Y=int(input())

Z=float(input())

L=list(input())

CONDITIONAL STATEMENTS:

◦ Conditional Statement in Python handles different computations or condition depending on whether a specific given constraint evaluates to true/false or 0/1.

◦ In this Python deals with IF statements.

◦ It decides the direction of execution of a program.

There are 3 different types of conditional statements in Python:-

      1. If statements

      2. If-else statements

      3. If-elif-else statements

SYNTAX
if condition.
     statements

1. **IF STATEMENTS**

◦ Here the condition is evaluated first and will execute statement(s) only if the text expression is True and also interprets non-zero values as True.

◦ None and 0 are interpreted as False. If the text expression is False, the statement(s) is not executed.

2. IF... ELSE:

◦ The if..else statement evaluates condition and will execute the body of if only when the condition is True or have any non zero value in it.

◦ If the condition is False, the body of else is executed.

◦ Indentation is used to separate the blocks.

◦ An else statement can be combined with an if statement and it is a optional statement.

3. IF....ELIF......ELSE STATEMENT:

```
SYNTAX
if condition:
    statement
elif condition:
    statement
else:
    statement
```

```
SYNTAX
if condition:
    statements
else:
    statements
```

◦ Here elif allows to check for multiple expressions in a program.

◦ If the condition for if is False/0, it will check the condition of the next elif block and so on.

◦ If both if and elif conditions are False/0, the else statement is going to execute.

◦ Only one block among the this if...elif...else will execute according to the condition given in program.

LOOPING Loop is sequence of instructions that repeats itself until the condition is satisfies.

Loops are use to cycle through values, add numbers, repeat functions, etc. A loop allow to execute a statement or group of statements multiple times in a program .

Python provide 3 types of following loops to handle looping in programs.

1. While loops

2. For loops

3. Nested loops

1.   **While Loop** Python while loop is used to execute a statement(s) repeatedly until a condition is satisfied. And when the condition becomes false, immediately that line terminates the loop.

Note : Here, statement(s) may be a single line statement or a multiple lines of statements. The condition may be any expression or any non-zero value. The loop iterates while the condition is true or have any non zero vale in it.

```
SYNTAX
while condition expression:
    statement(s)
```

2. FOR LOOP

- For loop has the ability to iterate over the items of any sequence.
- For loops are used for traverse in sequential manner.
- For example: traversing a list or string or array etc.
- It can be used to iterate over a range.

Here, val is the variable that takes the value of items which is inside the sequence on each iteration i.e. val is called as iterative variable. Loop continues until it reaches the last item in the iteration of sequence.

SYNTAX

```
for val in sequence:
    statement(s) of for
```

3. Nested loops

Nesting of loops are allow us to use one loop inside 2 or more loops. Any of the loop inside of any other type of loop. For example a for/while loop can be inside a while/for loop or vice versa.

NESTED LOOP OF FOR

```
for va in sequence:
    for val in sequence:
        statements(s)
    statements(s)
```

NESTED LOOP OF WHILE

```
while expression:
    while expression:
        statement(s)
    statement(s)
```

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




list1=[ 1, 2, 3, 4, 5, 6, 7, 8]
for i   in list1:
        print(i)




tuple1=('a', 'b', 'c', 'd', 'e')
For   i   in tuple1:
        print(i)
```

RANGE() FUNCTION:

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

- Syntax: range(*start, stop, step*)

- It is used **to generate a sequence of numbers over time**. At its simplest, it accepts an integer and returns a range object (a type of iterable).

BREAK / CONTINUE / PASS:

◦ In Python, to alter the flow of a normal loop in a program we use break and continue statements.

◦ Loops repeat itself in a block of code until the condition is false, but sometimes to terminate the current iteration or even the whole loop without checking the condition of loops.

SYNTAX break

SYNTAX continue

◦ The break statement terminates the control of the program. It allows to terminate the statement immediately after the body of the loop.

◦ The continue statement allow to skip the rest of the code inside a loop for the current iteration only but continues on with the next iteration without terminating the loop.

◦ It is mainly used for a particular condition inside the loop so that it can skip some specific line of code for a particular condition.

PASS In Python, the pass statement is a null statement because nothing happens when the pass is executed. So we get no result of any operation. It is use as a placeholder.

SYNTAX pass

STRING IN PYTHON

◦ A sequence/series of characters is known as string.

◦ internally computer turns the entered data in the format of 0's and 1's. encoding is the process of conversion of characters to numbers, decoding is the opposite of encoding. Here popular encoding used like Unicode and ASCII values.

◦ Creating a string: String is created by simply using quotes.

>>> x = "Hello Coder" >>> y = 'Bye Coder'

◦ Generally triple quotes can be used to create a string even though it is use to generate multiline string and docstring.

>>> a = """hello how are you all let us start Python"""

>>> b = '''hello everyone'''

>>> x = "hello"

>>>type(x)

>>> dir(x)

◦ Also there is a flexibility allows to use double/single quotes and apostrophes in your created string.

>>> 'Python is one of the best language, "tell this to everyone in the class". '

Output:- Python is one of the best language, "tell this to everyone in the class"

INDEXING OF STRING:

◦ we use factor called indexing to access the characters of string.

◦ Also Python follows the **negative indexing** to represent the character series, but use that negative values also in int type. Otherwise it will generate an error.

◦ There are some errors comes when we make mistakes like,

◦ **IndexError** comes when we try to achieve a character which is out of the index range. Then

◦ **TypeError** comes when we use float or other types to represent the index values of string.

◦ **Len(): to count the length of the string**

	LENGTH = 5				
	"H"	"e"	"l"	"l"	"o"
INDEX	0	1	2	3	4
NEGATIVE INDEX	-5	-4	-3	-2	-1

STRING SLICING The string slicing is refers to accessing the sub-parts of the string.

The slicing represent as**:**   **varName[starting index : ending index : step]**

varName[starting index : ending index]

varName[ending index]

Return Type: *Returns a sliced object containing elements in the given range only.*

.

DIFFERENT METHODS ON STRINGS:

1. capitalize(): Converts the first character to upper case

SYNTAX: *string*.capitalize()

2. count(): returns the number of times a specified value appears in the string.

SYNTAX: *string*.count(*value, starting index, ending index*)

3. find(): finds the first occurrence of the specified value. returns -1 if the value is not found.

SYNTAX: *string*.find(*value, start, end*)

4. index(): finds the first occurrence of the specified value. raises an exception if the value is not found.

Syntax: *string*.index(*value, start, end*)

```
Traceback (most recent call last):
  File "FileName.py", line 4 in <module>
    print(varName.index("x"))
ValueError: substring not found
```

5. isalpha():

returns True if all the characters

are alphabet letters (a-z).

○ Syntax: *string*.isalpha()

isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
isupper()	Returns True if all characters in the string are upper case

6. split(): Splits the string at the specified separator, and returns a list.

Syntax: *string*.split(*separator, maxsplit*)

7. swapcase(): returns a string where all the upper case letters are lower case and vice versa.

Syntax: *string*.swapcase()

8. title(): returns a string where the first character in every word is upper case.

Syntax: *string*.title()

9. upper(): returns a string where all characters are in upper case.

Syntax: *string*.upper()

10. lower(): returns a string where all characters are lower case.

Syntax: *string*.lower()

STRING IS MUTABLE OR IMMUTABLE?

String is immutable object in Python.

Since we know variable in Python is a instance of object.

Immutable means we can't change or update the existing string or we can say that not at all matters how many times the method is called and operated with the same instance, the output will be the same.

Example:-

>>> x = "string is immutable"

>>> x[0] = "o"

>>> print(x)

Instead of showing the output 'string is immutable'. This code will generate an error called TypeError.

Traceback (most recent call last):

File "", line 2, in TypeError: 'str' object does not support item assignment

Python Collections (Arrays):

◦ There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.

- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.

- **Set** is a collection which is unordered and unindexed. No duplicate members.

- **Dictionary** is a collection which is ordered* and changeable. No duplicate members.

LIST:

◦ A list is a data type in python that allows you to store various types data in it.

◦ Also called compound data type which means different data types under a list can be made. for example we can have integer, float and string items in a same list.

◦ **To create a list** Firstly we have to use **[] (square bracket)** and inside it we place the items that are separated by comma(,).For example. **list1=[1,2,3, 1.23, "hello"].**

◦ **To access any item In the given list** use: **List_Name[index_number]**

◦ Note: Do not use float vales in index number. And The index must be in range to avoid IndexError.

◦ Negative index can be use that allow us to access the list elements starting from the end. For example an index of -1 would access the last element of the list, -2 second last, -3 third last and so on.

◦ **FOR SLICING OF LIST** For example: n_list=[1,2,3,4,5]   SYNTAX: L_name(start : end : step)

print(n_list(1:3))

Ans. [2,3] By colon operator ' : ' we slice the list in given range.

◦ **<u>Several ways to insert item in a list</u>**

◦ **<u>1.using insert():</u>** Add elements in desired location.

Example: n_list = [1, 2, 3, 4]

        n_list.insert(3, 60)

        print(n_list)

Ans. [1, 2, 3, 60, 4]

 <u>2.using append():</u> Add items in the end of the list.

Example : n_list = [1, 2, 3, 4]

        n_list.append(60)

        print(n_list)

Ans. [1, 2, 3, 4, 60]

<u>3. using extend():</u> Add 2 or more element in the list.

Example: n_list = [1, 2, 3, 4]

        n_list.extend([30, 40])

        print(n_list)

Ans. [1, 2, 3, 4, 30, 40]

<u>To delete an item from the list.</u>

1.**using "del" keyword.**

Example : n_list = [1, 2, 3, 4, 5, 6]

        del n_list[1] print(n_list) del n_list[2:4] print(n_list) Ans [1, 3, 4, 5, 6] [1, 3, 6]

2**. DELETE USING REMOVE():**

remove(item): Removes specified location item from list. Example : ch_list = ['A', 'F', 'B', 'Z', 'O','L'] ch_list.remove('B') print(ch_list) Ans list: ['A', 'F', 'Z', 'O', 'L']

3.DELETE USING POP

<u>1. pop(index):</u> Removes the element from the given index. Example : ch_list=['A','F','Z','O','L'] ch_list.pop(1) print(ch_list) Ans ['A', 'Z', 'O', 'L']

<u>2. pop():</u> Removes the last element. Example : ch_list=['A','Z','O','L'] ch_list.pop() print(ch_list) Ans. ['A', 'Z', 'O']

TO CLEAR A GIVEN LIST. clear(): Removes all the elements from the list.

Example : ch_list=['A','Z','O'] ch_list.clear() print(ch_list) Ans []

TO SEE THE METHODS IN THE CLASS OF LIST:- >>> x=list() >>> print(type(x)) >>> print(dir(x))

BUILDIN – FUNCTIONS IN LIST:

1. TO SORT A LIST: list_name.sort() : method to sort the items (i.e. change the order) accordingly the ascii values of first character. This function **sorts** the list in **increasing order**.

Example: x=['hello', 'coders']

 >>> x.sort()

>>> print(x) ['coders', 'hello']

>>> n=[3, 14, 12, 9, 74, 15]

2. >>> print(len(n)) ----- use to count the items in a list ----- 6

3. >>> print(max(n)) ----- use to find the maximum value of the item ---- 74

4. >>> print(min(n)) ----- use to find the minimum value of the item ---- 3

5. >>> print(sum(n)) ----- use to return the sum of all items present in list ----154

6. >>> print(sum(n)/len(n)) ----- 25.6 ----- to find average

7. >>> print(n.count('14')) ---- Calculates total occurrence of given element of List. ------ 1

8. >>> print(n.index('12')) ---- Returns the index of first occurrence. --- 2.

9. >>> print(n.sort()) ----- This function **sorts** the list in **increasing order**.

10. >>> print(n.reverse()) ----- use to **reverses** the elements of list.

Note: to concate list use "+".

STRINGS AND LISTS ARE BEST FRIENDS(USE OF SPLIT() FUNCTION) :-

>>> a = "hello coders how are you"

>>> z = a.split()

>>> print(z)

['hello', 'coders', 'how', 'are', 'you']

>>>print(len(z))   ------   5

>>> print(z[0])

Hello Split breaks a string into parts and produces a list of strings.

TUPLES IN PYTHON:

◦ Tuples are collection of objects.

◦ It is quite similar to lists in terms if indexing, nested instances but The only difference between tuple and list is that **we cannot change** the items of a tuple once it is assigned to a variable whereas we can change the items of a list i.e. **tuple is immutable objects** unlike **lists are mutable objects.**

◦ **To create a tuple:** by () placing all the items inside it and separated by comma (,).

◦ >>> tuple1 = ()

◦ **TUPLE PACKING** Creating tuples without using of parenthesis () is called as tuple packing. Also unpacking of tuples is possible.

◦ Example:-

>>> tuple5 = 3, 968.637, "hello"

>>> print(tuple5)   -- (3, 968.637, 'hello')

tuple unpacking

>>> a, b, c = tuple5

>>> print(a)   ----- 3

>>> print(b) ------- 968.637

>>> print(c) ------- hello

◦ Creating a single element tuple without a comma confuse the interpreter and takes as a string.

◦ If we have to create a single element tuple we have to use a comma.

◦ Example:

>>> tuple6 = ("hello")   >>> print(type(tuple6))

◦ # Creating a tuple having one element

>>> tuple7 = ("hello",)

>>> print(type(tuple7))

Parentheses is optional

>>> tuple8= "hello",

>>> print(type(tuple8))

ACCESSING ELEMENT IN TUPLES:

1. INDEXING: To access the elements, we can use the default values of index used. Remember index always starts from 0.

>>>tuple8 = (True,'420','ok','1.00','100')

>>> print(tuple8[0])

>>> print(tuple8[4])

So let's understand with the help of example:-

tuple8 = (True,'420','ok','1.00','100')

print(tuple8[5]) -- index should be in the range.

That **generate IndexError. Traceback (most recent call last):** File "", line 3, in IndexError: tuple index out of range.

print(tuple8[2.00]) - index should be int type

Traceback (most recent call last): File "", line 2, in **TypeError: tuple indices must be integers or slices, not float**

nested tuple tuple9 = ("hello", [568, "bye", 6.98], (1, 2, 3))

nested index print(tuple9[0][3.0]) Traceback (most recent call last): File "", line 2, in TypeError: tuple indices must be integers or slices

2. SLICING: In second method we use slicing, so that we can access the elements by using ':' (colon) operator.

Example:- tuple10 = (False,'0011','90.00','hello','10')

print(tuple10[1:4])

print(tuple10[:-4])

print(tuple10[3:])

print(tuple10[:])

Output:-

('0011', '90.00', 'hello')

(False,)

('hello', '10')

(False, '0011', '90.00', 'hello', '10')

Concatenation of tuples

To concatenate tuples, we use '+' operator. Example:- >>> print((1, 2, 3) + (4, 5, 6))

Repeat the elements of a tuple

'*' operator use to repeat the elements and with number how time we want to print it. Example:- print(("hello",) * 3)

DELETING TUPLE :

Since tuple is immutable we can't delete a element from the tuple but we can delete whole tuple by using "del" keyword.

Example:- tuple11 = ('1.000', '556', 'hello')

del tuple11

Output:- ('1.000', '556', 'hello') Traceback (most recent call last): File "", line 11, in NameError: name 'tuple11' is not defined. As you can see after using del method, print(tuple11) shows error because del delete the tuple11. Since the tuple is deleted then how interpreter print the tuple i.e. why interpreter show the NameError.

PYTHON TUPLES BUILD-IN FUNCTIONS

1. cmp(tuple1, tuple2) -It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.

2. len(tuple) - It calculates the length of the tuple.

3. max (tuple) - It returns the maximum element of the tuple.

4. min (tuple) - It returns the minimum element of the tuple.

5. tuple(seq) - It converts the specified sequence to the tuple.

TO SEE THE METHODS OF TUPLES

x=tuple()

print(type(x))

print(dir(x)) -------- ['count','index']

There are only 2 methods in tuple.

◦ count() It return the number of occurrences of a element.

◦ index() It returns the index value of element.

DICTIONARY IN PYTHON In Python, dictionary is an unordered collection of elements associated with key-value pairs. Each key is connected to a value, and can be use a key to access the value associated with that key.

- Value of a key can be a number, a string, a list, or even another dictionary.
- It is a compound data type with key and values pairs.

CREATE DICTIONARY

- We create dictionary by using {} curly bracket and placing element into it which is separated by comma(,).
- An element itself consist of two parts 1. Key 2. Value Expressed as key : value or k:v.

Key must be of immutable type (as string, numbers or tuple with immutable elements) and must be unique but **value** can be of any data type and need not to be unique.

# dictionary with integer keys
d = {1: 'hello', 2: 'coders'}

◦ Since dictionary is mutable object, we can add update or change the values of items using assignment operator.

◦ If the key is already present, then the existing value gets updated in the existing dictionary. In case the key is not in the dictionary , a new (key: value) pair is added to it.

DELETE THE ELEMENTS

We can delete the elements of dictionary by using following methods

1. Using "del" keyword. 2. Using pop() method. 3. Using popitem() method. 4. Using clear() method.

a. Using del keyword:- Here we use del key word before the element individually we want to delete or entire dictionary we have to delete.

b. Using pop() method:- To remove a particular element in a dictionary by using the pop() method. This method removes an element with the key and returns the value only.

c. Using popitem() method:- This method use to remove the item and return an arbitrary (key, value) item pair from the dictionary.

d. Using clear() method:- By using clear() method, we can remove all the item at once.

◦ TO SEE THE METHODS IN DICTIONARIES:-

◦ The below given command are used to see the methods in dictionaries

◦ >>> d5=dict() >>> print(type(d5)) >>> print(dir(d5))

Different Methods:

1. copy() It returns a shallow copy of the dictionary. SYNYAX dict_name.copy().

2. update() This method updates the dictionary with the elements from the another dictionary objects.

SYNTAX dict_name.update(parameter)

3. keys() This method returns a object that shows a list of all the keys in the dictionary. SYNTAX dict_name.keys()

4. cmp(dict1, dict2)- Compares elements of both dictionary.

5. len(dict)- Gives the total length of the dictionary.

6. str(dict)- Produces a printable string representation of a dictionary

7. type(variable)- Returns the type of the variable.

FUNCTIONS: Function is a block of statements which only execute when it is called to preform the specified task.

Functions help break our program into smaller and modules that also helps us to manage the large lines of code in proper manner. Also it help to reuse the code of functions.

SYNTAX

def fun_name(parameter1, parameter2,.......):

       """ docstring"""

       Statements

Function have 2 parts

1. Definition: Definition is done by using def keyword.

2. Calling : Then come on the calling part, it is important. Because after definition, we have to call that function. We can call the function from the other function, or program or even the Prompt. To call a function, we type the function name with parameters.

WHAT IS DOCSTRING? Docstring is a line of codes which is written after the header of function. This short documentation of string briefly defined the work what function is going to do.

RETURN STATEMENT IN FUNCTIONS : The return statement is use to exit the function and go back to the place from where it was called in the body of main function.

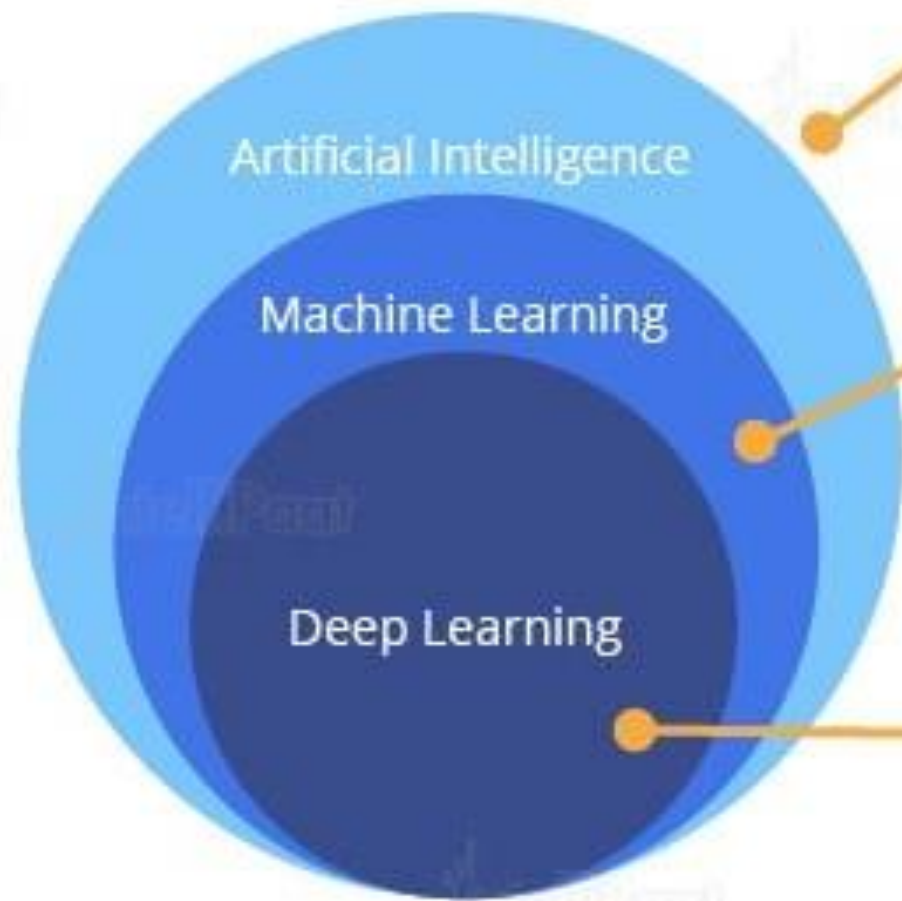SYNTAX   def func_name()

        Statements

        return()

TYPES OF FUNCTIONS

Basically there are 2 types of functions

1. Built-in functions which are already defined into the python libraries.

2. User-defined functions which are made by user themselves using def keyword.

MACHINE LEARNING

Artificial Intelligence

Machine Learning

Deep Learning

A technique for incorporating human intelligence to machine. It is the superset of Machine Learning and Deep Learning.

It is a sub-field of AI that comprises of algorithms and statistical models used by machines to perform a specific task and improve with experience by learning on its own.

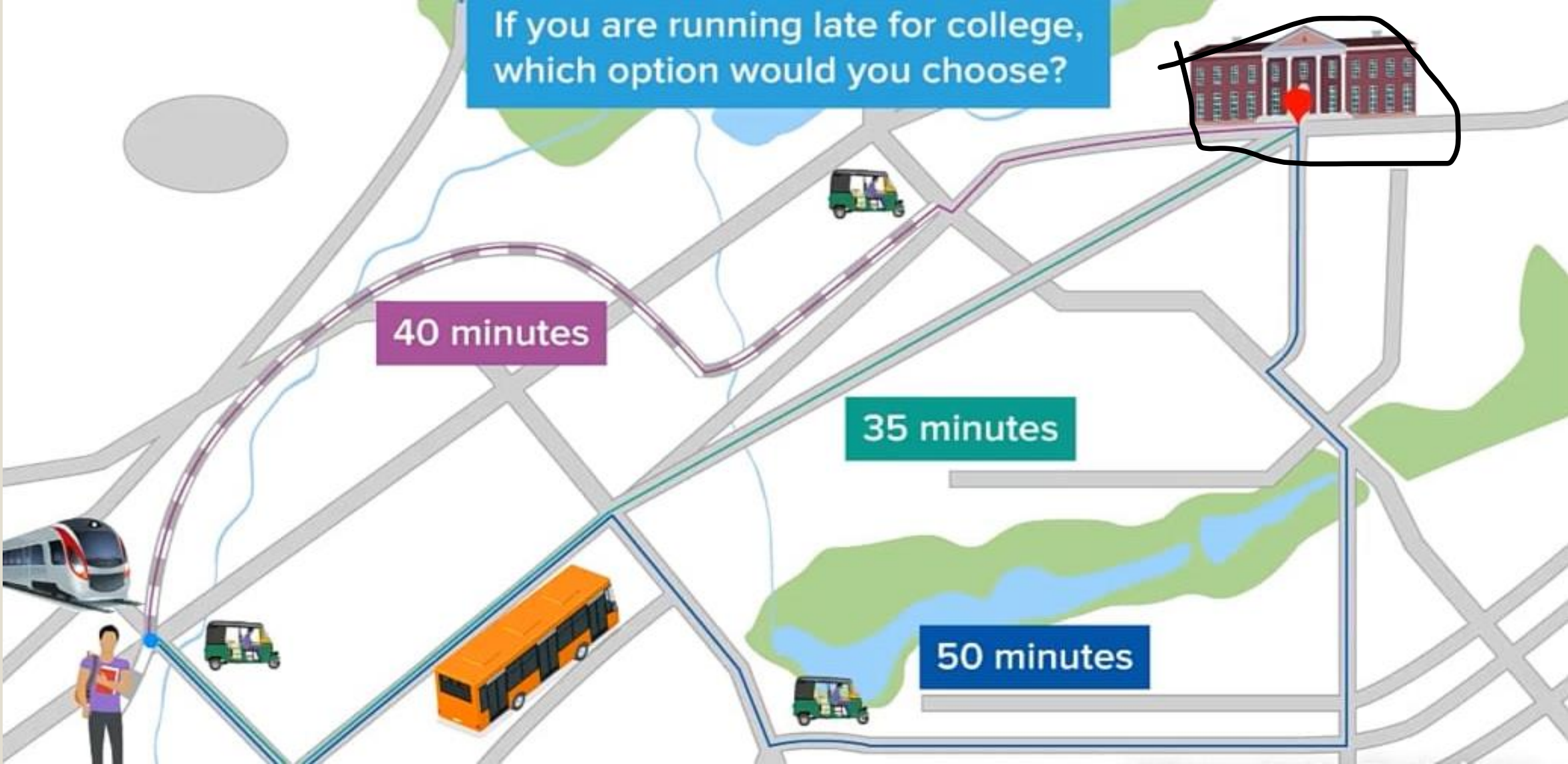It is the sub field of Machine Learning that uses Artificial Neural Network for computation for giving accurate results.

If you are running late for college, which option would you choose?

40 minutes

35 minutes

50 minutes

Machine learning is the ability of machines (i.e., computers or ideally computer programs) to learn from the past behavior or data and to predict the future outcomes without being explicitly programmed to do so.

Pattern Recognition

Past Experience → → **Predictions**

Machine Learning is the ability of Machines, that is, computers to learn and improve from their past experience or data, without being explicitly programmed.

Data

Machine learning algorithms are constantly analyzing and learning from the data to improve their future predictions and outcomes automatically.

News shared with the user → **f** → News similar to the news liked/commented on by the user

Which news user liked/ignored/commented on?

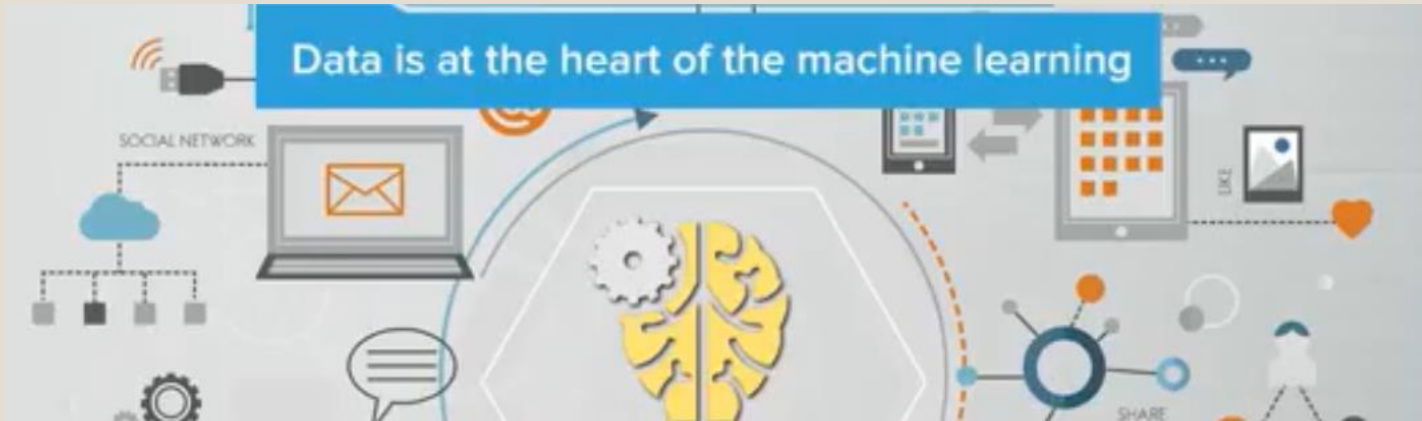Your and other customer's buying and browsing history → **amazon** → Product Recommendation

Which product you may like?

Advanced Applications of Machine Learning

- Identifying Frauds in Banking
- Sentiment Analysis
- Amazon Go
- Chatbots
- Self Driven Cars

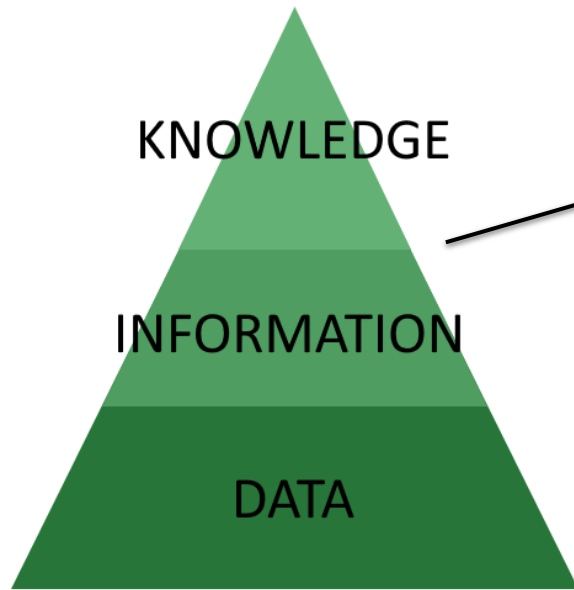Data is at the heart of the machine learning

HOW MACHINE LEARNING WORDS?

Past Data → ML Model → Intelligence → New Data → Predictions/ Recommend action

WHAT IS DATA?

Data is a collection of information about something.

KNOWLEDGE

INFORMATION

UNLABELED DATA | LABELED DATA

KNOWLEDGE

INFORMATION

DATA

What is a variable?
A variable represents one specific characteristic of the data or tells one specific information about the data under consideration.

The data which contains a target variable or an output variable that answers a question of interest is called labelled data.

- ☑ Patient name
- ☑ Age
- ☑ Gender
- ☑ Location
- ☑ Hospital name
- ☑ Time since patient is suffering from the disease
- ☑ Type of disease
- ☑ Complexity level
- ☑ Patient's weight
- ☑ Body temperature
- ☑ Blood pressure
- ☑ Heart rate
- ☑ Amount of dose given

Depends on

Patient was completely cured or not after the treatment?

Yes

No

Target variable/ Dependent variable

Unlabelled data is one which contains information about something but does not have a predefined target variable.

Independent variables

TYPES OF VARIABLE:

EXAMPLE: MOBILE
CATEGORICAL: CHEAP/COSTLY
NUMERIC: 20000/50000

Numeric variables are those which are quantifiable.

Categorical variables are those which are like adjectives and express a feeling or a characteristic.

The categorical variables consist of string or text values.
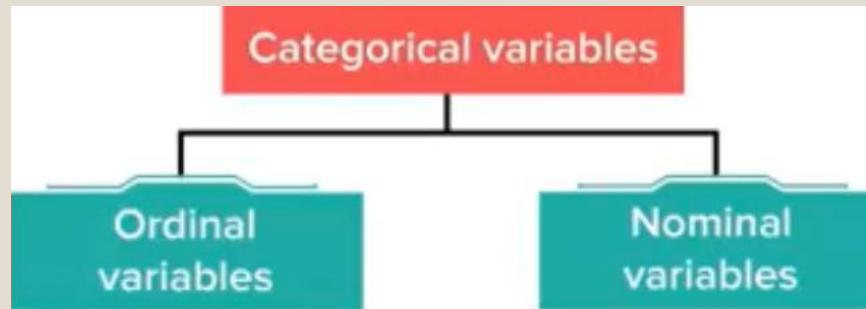
7 3 9
1 5
6 8
2 4

Yes NO GOOD BAD Slow Fast

Numeric variables

Continuous variables

Discrete variables

Continuous variables are those numeric variables which can take any value between a certain set of real numbers.

Discrete variables are those which are countable and can take only whole numbers (i.e. Integers) as value.

Categorical variables

Ordinal variables

Nominal variables

Ordinal variables are those categorical variables which can be arranged in a logical order.

Nominal variables are those categorical variables which cannot be ranked based on their values.

Product

Poor | Average | Good

Samsung | Apple | Nokia

Data Analysis

- Is the process of studying the available data and drawing valuable insights or information from it with the help of any software.
- Is being used everyday and everywhere to enable the businesses to take smart and accurate decisions.
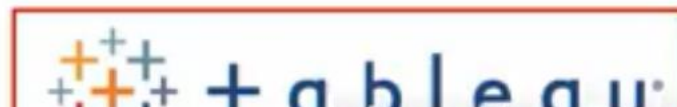
Exploratory Data Analysis (EDA)

To find out the patterns and relationships among the data variables.
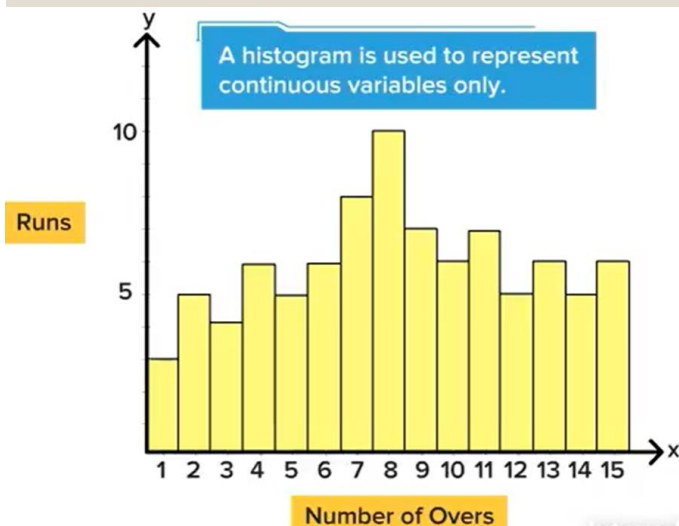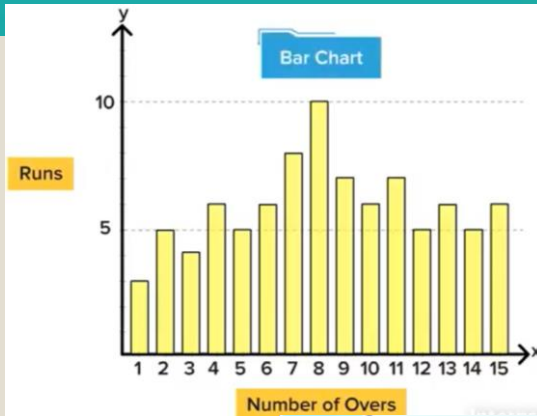
Commonly Used Analytical Tools

python™

R

SAS
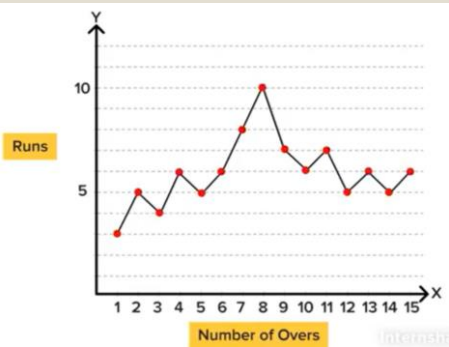
Open for Innovation
KNIME
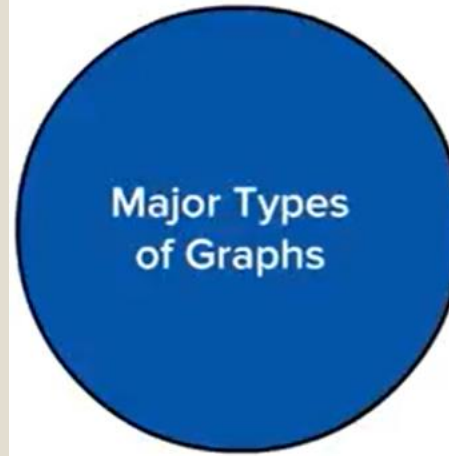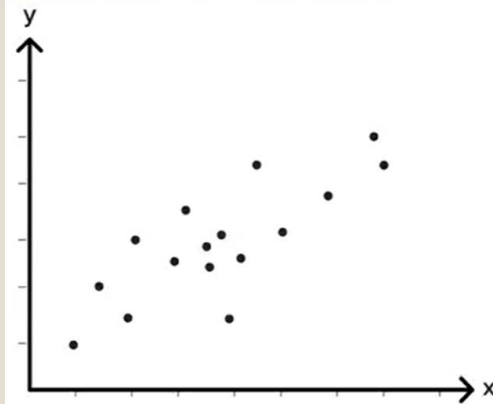
SPSS

MATLAB®

Minitab® 18

tableau

Graphical Representation of Data:

- Is one of the simplest techniques for drawing insights from the data.
- Helps us to study the relationship between the variables.
- Helps to identify the trend and patterns across the variables.

Bar Chart

Runs

Number of Overs

Major Types of Graphs

- Line Graph
- Bar Graph
- Histogram/Frequency Polygon
- Pie Chart
- Scatter Plot
- Box and Whisker Plot
- Time Series

A histogram is used to represent continuous variables only.

Runs

Number of Overs

A scatter plot is very useful in informing us the relationship between two variables.

STEP 01 — Data collection

STEP 02 — Data preparation

STEP 03 — Choosing a model

STEP 04 — Training

STEP 05 — Evaluation

STEP 06 — Parameter tuning

STEP 07 — Prediction

NUMPY

- NumPy is a Python library used for working with arrays.

- It also has functions for working in domain of linear algebra, fourier transform, and matrices.

- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

- NumPy stands for Numerical Python.

- To install in anaconda

- conda install -c anaconda numpy

alias: In Python alias are an alternate name for referring to the same thing.
- Create an alias with the as keyword while importing:
import numpy as np

Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

PANDAS

- Pandas is a Python library used for working with data sets.

- It has functions for analyzing, cleaning, exploring, and manipulating data.

- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

TO INSTALL: conda install pandas   OR   conda install -c anaconda pandas

DataFrames

- Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

- Series is like a column, a DataFrame is the whole table.

Read CSV Files

- A simple way to store big data sets is to use CSV files (comma separated files).

- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

What is a CSV?

- **CSV (Comma Separated Values)** may be a simple file format accustomed to store tabular data, like a spreadsheet or database. Each line of the file could be a data record. Each record consists of 1 or more fields, separated by commas, the utilization of the comma as a field separator is that the source of the name for this file format.

Table Data

Programming language	Designed by	Appeared	Extension
Python	Guido van Rossum	1991	.py
Java	James Gosling	1995	.java
C++	Bjarne Stroustrup	1983	.cpp

If we convert this data into a **CSV Format**, then its look like this:

CSV Data

Programming language, Designed by, Appeared, Extension

Python, Guido van Rossum, 1991, .py

Java, James Gosling, 1995, .java

C++, Bjarne Stroustrup,1983,.cpp

Viewing the Data
One of the most used method for getting a quick overview of the DataFrame, is the head() method.
The head() method returns the headers and a specified number of rows, starting from the top.

Info About the Data

The DataFrames object has a method called info(), that gives you more information about the data set.

TO DEAL WITH EMPTY OR NULL VALUES:

1. Remove Rows

◦ One way to deal with empty cells is to remove rows that contain empty cells.

2. Replace Empty Values

◦ Another way of dealing with empty cells is to insert a *new* value instead.
◦ This way you do not have to delete entire rows just because of some empty cells.
◦ The fillna() method allows us to replace empty cells with a value.

MATPLOTLIB:

◦ Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

◦ Matplotlib was created by John D. Hunter.

◦ Matplotlib is open source and we can use it freely.

◦ Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

TO INSTALL MATPLOTLIB:

```
conda install -c conda-forge matplotlib
```

- LAMBDA FUNCTION:
- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- Syntax: lambda *arguments* : *expression*
- x = lambda a : a + 10
  print(x(5))