Python:

- Python is a high-level, interpreted programming language.
- Created by Guido Van Rossum and released in 1991.
- It is a versatile and beginner-friendly language.
- Python is known for its readability and efficiency.
- Easily adaptable for Web applications and Database projects.
- Majorly supports Software Development.
- Python has simple syntax, which makes it easier to grasp.
- Code will be executed line by line, easy to debug.
- Rich library sets make it easier to perform various tasks from web development to manipulate the data.
- Supports object oriented programming, allows for the creation of the objects.
- Can run on various platforms.

WHY PYTHON?

- Python works on different platforms like Windows, Mac, Linux etc.
- Python has simple syntax.
- Allows the developer to write programs in fewer lines and some other languages.
- Interpreter language, which means the language can be executed line by line.
- Procedural execution process and it is an Object Oriented Language.

APPLICATIONS OF PYTHON:

- Web Development
- Data Science and Analytics
- Machine language and Artificial Intelligence
- Automation and Scripting

COMMENTS:

- Comments used to explain the python code but it won't be executed.
- Comments will be started with ' # '.
- Ex:
    - #To print Hello World.
    - print("Hello World")

Variables:

- Containers used to store the data values.
- Variables must be case-sensitive.
- Should not contain special characters.
- Must be alphanumeric characters.
- Can store multiple values.
- Variables can not be keywords.
- Variables should not start with a digit.
- Ex:
    - a=20
    - b=55
    - Name="Lekha"

Data Types:

- Data types define the kind of the value that the variable holds.
- The list of Data Types;
- Numeric :
    - Integer - can be directly defined
    - Ex: a=10
    - Float - can be directly defined
    - Ex: a=90
    - Complex - a+ib, here a is real number and b is an imaginary number.
- Sequence:
    - String - The names will be defined in this. Uses double Quotations(" ").

    - Ex: Name = “Lekha”
- Boolean:
    - Boolean - has 2 built-in functions. True / False.
    - According to the situation it displays the values.

OPERATORS:

- Arithmetic Operations: used to do the mathematical calculations.
- SUM: Addition of two numbers.
    - Ex:

```
a=10
b=5
c=a+b
print("Sum is: ", c)
```

    - Sum is: 15

- Substracton: Difference of two numbers.
    - Ex:

```
a=10
b=5
c=a-b
print("Difference is: ", c)
```

- Difference is: 5

- Multiplication: Product of two numbers.
    - Ex:

```
a=10
b=5
c=a*b
print("Product is: ", c)
```

    - Product is: 50

- Division: division of two numbers.
  - Ex:

```
a=10
b=4
c=a/b
print("Div is: ", c)
```

  - Div is: 2.5

- Modulus Division: Displays the remind of the two numbers.
  - Ex:

```
a=10
b=5
c=a%b
print("Moddiv is: ", c)
```

  - Moddiv is: 0

- Floor Division:
  - Ex:

```
a=10
b=5
c=a//b
print("Fdiv is: ", c)
```

  - Fdiv is: 2

ASSIGNMENT OPERATORS:

- Equals to:

```
a=10
print(a)
```

  - 10

- +=:

```
a=10
a+=25
```

```
print(a)
```

- 35

- -=:

```
a=10
a-=25
print(a)
```

- -15

- *=:

```
a=10
a*=25
print(a)
```

- 250

- /=:

```
a=10
a/=5
print(a)
```

- 2.0

- &=:

```
a=10
a%=5
print(a)
```

- 0

- //=:

```
a=10
a//=5
print(a)
```

- 2

- |=:
  - ```
    a=10
    a|=5
    print(a)
    ```
  - 15
- ^=:
  - ```
    a=10
    a^=5
    print(a)
    ```
  - 15
- >>=:
  - ```
    a=10
    a>>=5
    print(a)
    ```
  - 0
- <<=:
  - ```
    a=10
    a<<=5
    print(a)
    ```
  - 320

COMPARISON OPERATORS:

- DOUBLE EQUALS TO :
  - ```
    a=10
    b=60
    print(a==b)
    ```
  - False
    ```

- NOT EQUALS TO:
  ```
  a=10
  b=60
  print(a!=b)
  ```
  - True

- Greater than:
  ```
  a=10
  b=60
  print(a>b)
  ```
  - False

- Less than:
  ```
  a=10
  b=60
  print(a<b)
  ```
  - True

- Greater than equal to:
  ```
  a=10
  b=10
  print(a>=b)
  ```
  - True

- Less than equal to:
  ```
  a=10
  b=5
  print(a<=b)
  ```
  - False

LOGICAL OPERATORS :

- AND: Return TRUE if two statements are true.

```
a=10
print(a > 3 and a < 10)
```

  - False

- OR: Return TRUE if any of the two statements are true.

```
a=10
print(a > 3 or a < 10)
```

  - True

- NOT: Reveses the result.

```
a=10
print(not(a > 3 or a < 10))
```

  - False

IDENTITY OPERATORS:

- Identity Operators are used to compare the objects not by the equality, but by the object, stored in the same memory location.
  - is - Returns TRUE,if both are the same objects.
  - Is not - Returns TRUE if, both are not the same objects.

- is : Returns TRUE,if both are the same objects.

```
a=["apple", "banana", "grape","cherry"]
b=["apple", "banana","grape", "cherry"]
c=a
print(a is b)
```

  - False

- is not : Returns TRUE, if both are not the same objects.

```
a=["apple", "banana", "grape","cherry"]
b=["apple", "banana","grape", "cherry"]
c=a
print(a is not b)
```

  - True

MEMBERSHIP OPERATORS:

- Membership Operators are used to test if sequence is presented in an object.
  - in - Returns True if a sequence with the specified value is present in the object
  - not in - Returns True if a sequence with the specified value is not present in the object

- in : Returns True if a sequence with the specified value is present in the object

```
a=["apple", "banana", "grape","cherry"]
print("apple" in a)
```

  - True

- not in : Returns True if a sequence with the specified value is not present in the object

```
a=["apple", "banana", "grape","cherry"]
print("apple" not in a)
```

  - False

BITWISE OPERATORS:

- AND: Sets 1 if both parameters are one.

```
a=10
b=20
print(3&4)
```

  - 0

```
a=10
b=6
print(a&b)
```

- 2

- OR: Sets 1 if any of the both parameters are one.

```
a=10
b=6
print(a|b)
```

- 14

- NOT: Done with a single number. Reverses the result.

```
a=7
print(~a)
```

- -8

- XOR: Returns 1 if one and only if one of the Operands is 1.

```
a=7
b=5
print(a^b)
```

- 2

- LEFT SHIFT: Shifts the bits to the left from right and fills the empty spaces with 0's.

```
a=10
b=5
print(a<<b)
```

- 320

- RIGHT SHIFT: Shifts the bits to the right from left and fills the empty spaces with 0's.

```
a=10
b=5
print(a>>b)
```

- 0

LISTS:

- List is similar to an array which is dynamic.
- Used to store multiple values in a variable.
- Defined in the square braces([]).
- Contains duplicate values.
- Items are ordered, changeable and allow the duplicate values.
- Accessing can be done easily by using index number.

- Creating a List:

```
l=["apple", "banana", " grapes", "cherry"]
print(l)
```

  - ['apple', 'banana', ' grapes', 'cherry']

- Accessing the Items:

```
l=["apple", "banana", " grapes", "cherry"]
print(l[3])
```

  - cherry

- Changing the List Items: manipulating the data values in the list.

```
l=["apple", "banana", " grapes", "cherry"]
l[1]="kiwi"
print(l)
```

  - ['apple', 'kiwi', ' grapes', 'cherry']

- Append: adding the values at the end of the list

```
l=["apple", "banana", " grapes", "cherry"]
l.append("kiwi")
print(l)
```

  - ['apple', 'banana', ' grapes', 'cherry', 'kiwi']

- Extend: adding the values in the from two lists
  ```
  l=["apple", "banana", " grapes", "cherry"]
  l1=["mango", "kiwi", "pineapple"]
  l.extend(l1)
  print(l)
  ```
  - ['apple', 'banana', ' grapes', 'cherry', 'mango', 'kiwi', 'pineapple']

- Insert: allows the values to add values in the list at the specific index
  ```
  l=["apple", "banana", " grapes", "cherry"]
  l.insert(1, 'mango')
  print(l)
  ```
  - ['apple', 'mango', 'banana', ' grapes', 'cherry']

- Remove: removes the values from the list
  ```
  l=["apple", "banana", " grapes", "cherry"]
  l.remove('banana')
  print(l)
  ```
  - ['apple', ' grapes', 'cherry']

- POP: removes the values at specific index in the list
  ```
  l=["apple", "banana", " grapes", "cherry"]
  l.pop(1)
  print(l)
  ```
  - ['apple', ' grapes', 'cherry']

- Delete: Deletes the list completely/ deletes the values at the specific index
  ```
  l=["apple", "banana", " grapes", "cherry"]
  del l[2]
  print(l)
  ```
  - ['apple', 'banana', 'cherry']
  ```
  l=["apple", "banana", " grapes", "cherry"]
  del l
  ```
  -

- Clear: Clears the list

```
l=["apple", "banana", " grapes", "cherry"]
l.clear()
print(l)
```

  - []

FOR LOOP for printing List:

- For loop:

```
l=["apple", "banana", " grapes", "cherry"]
for i in l:
   print(i)
```

  - apple
  - banana
  - grapes
  - Cherry

```
l=["apple", "banana", " grapes", "cherry"]
for i in range(len(l)):
   print(l[i])
```

  - apple
  - banana
  - grapes
  - cherry

WHILE LOOP for printing List:

  - While loop:

```
l=["apple", "banana", " grapes",]
i=0
while i < len(l):
   print(l[i])
   i += 1
```

  - apple
  - banana
  - grapes

LIST COMPREHENSION:

- Creating new list

```
l=["apple", "banana", " grapes", "cherry"]
n=[]
for i in  l:
   if i in l:
       n.append(i)
print(n)
```

- ['apple', 'banana', ' grapes', 'cherry']

SORTING:

- Sorting list in ascending order:

```
l=["banana", " grapes", "cherry"]
l.sort()
print(l)
```

  - ['banana', 'cherry', 'grapes']

- Sorting list in descending order:

```
l=["banana", " grapes", "cherry"]
l.sort(reverse= 'true')
print(l)
```

- [' grapes', cherry', 'banana']

TUPLE:

- TUPLE: It is a collection of objects separated by commas.
- Used to store multiple values in a single variable.
- Tuples are ordered and immutable.
- Tuples are written with round braces.
- Tuples allows duplicate values.
- Similar to Lists in Python.
- Tuple cannot be updated once it is created.
- Cannot be appended or extended.
- Items cannot be removed from the tuple.

- CREATING TUPLE:
  - ```
    f=("apple", "banana", "cherry", "grapes", "Orange")
    print(f)
    ```
  - ('apple', 'banana', 'cherry', 'grapes', 'Orange')

- CREATING A SINGLE ITEM IN TUPLE:
  - ```
    f=("apple",)
    print(f)
    ```
  - ('apple',)

- ACCESSING THE ITEMS IN TUPLE:
  - ```
    f=("apple", "banana", "cherry", "grapes", "Orange")
    print(f[1])
    ```
  - banana

- CREATING TUPLES WITHOUT PARENTHESES:
  - ```
    f="apple", "banana", "cherry", "grapes", "Orange"
    print(f)
    ```
  - ('apple', 'banana', 'cherry', 'grapes', 'Orange')

- CHECKING LENGTH OF THE TUPLE:
```
f=("apple", "banana", "cherry", "grapes", "Orange")
print(len(f))
```
  - 5

- CREATING TUPLE USING tuple() CONSTRUCTOR:
```
f=tuple(("apple", "banana", "cherry", "grapes", "Orange"))
print(f)
```
  - ('apple', 'banana', 'cherry', 'grapes', 'Orange')

- CREATING TUPLE EMPTY:
```
f=()
print(f)
```
  - ()

- CHECKING THE EXISTENCE IN THE TUPLE:
```
f=("apple", "banana", "cherry", "grapes", "Orange")
print("cherry" in f)
```
  - True
```
f=("apple", "banana", "cherry", "grapes", "Orange")
print("kiwi" in f)
```
  - False

- SLICING: EXTRACTS THE Part of the TUPLE:
```
f=("apple", "banana", "cherry", "grapes", "Orange")
print(f[0:3])
```
  - ('apple', 'banana', 'cherry')

- REPETITION: Uses *, repeats the values certain no.of times.
```
f=("apple", "banana", "cherry", "grapes", "Orange")
print(f*3)
```
  - ('apple', 'banana', 'cherry', 'grapes', 'Orange', 'apple', 'banana', 'cherry', 'grapes', 'Orange', 'apple', 'banana', 'cherry', 'grapes', 'Orange')

- CONCATENATION: Concatenates the 2 TUPLES
  ```
  f=("apple", "banana", "cherry", "grapes", "Orange")
  f2=("kiwi", "avacado", "strawberry", "blueberry", "blackberry")
  r=(f+f2)
  print(r)
  ```
  - ('apple', 'banana', 'cherry', 'grapes', 'Orange', 'kiwi', 'avacado', 'strawberry', 'blueberry', 'blackberry')

- CREATING A TUPLE WITH MULTIPLE DATA TYPES:
  ```
  f=("apple", 45, 3.14, "TRUE")
  print(f)
  ```
  - ('apple', 45, 3.14, 'TRUE')

- FOR LOOP:
  ```
  f=("apple", "banana", "cherry", "grapes", "Orange")
  for i in f:
     print(i)
  ```
  - apple
  - banana
  - cherry
  - grapes
  - Orange

- SET:
  - Unordered collection of items/elements.
  - No duplicates are allowed.
  - Set items can appear in a different order every time and cannot be referred to by index or key.
  - Sets are immutable.
  - Sets can be immutable by using frozen set.
  - Represented by flower/curly braces”{}”.
  - Sets are Inbuilt Versions of Python.

```
s={"apple", "banana", "cherry", "grapes", "Orange"}
print(s)
```

- {'banana', 'apple', 'grapes', 'cherry', 'Orange'}

- Doesn't allow Duplicates:

```
s={"apple", "banana", "cherry", "grapes", "Orange", "banana",
"cherry", "grapes"}
print(s)
```

- {'apple', 'cherry', 'grapes', 'Orange', 'banana'}

- Finding the Length of the Set:

```
s={"apple", "banana", "cherry", "grapes", "Orange",}
print(len(s))
```

- 5

- Sets can be defined as the objects with Set datatype:

```
s={"apple", "banana", "cherry", "grapes", "Orange",}
print(type(s))
```

- <class 'set'>

- Sets can be have different data types:

```
s={"apple", "true", 1, 2.0}
print(s)
```

- {'apple', 1, 2.0, 'true'}

- DICTIONARIES:
  - Used to store the data values in the form of key:values pairs.
  - Duplicates cannot be allowed.
  - It is an Ordered Collection of data/elements.
  - Dictionaries: keys are immutable, value are mutable.
  - Case-Sensitive.
  - Written within the curly braces.

- Creating Dictionary:
  ```
  d={"empname": "Lekha", "Org": "Puropale", "Dept": "AWS"}
  print(d)
  ```
  - {'empname': 'Lekha', 'Org': 'Puropale', 'Dept': 'AWS'}

- Length of Dictionary:
  ```
  d={"empname": "Lekha", "Org": "Puropale", "Dept": "AWS"}
  print(len(d))
  ```
  - 3

- Accessing items in the Dictionary:
  ```
  d={"empname": "Lekha", "Org": "Puropale", "Dept": "AWS"}
  print(d["Org"])
  ```
  - Puropale

- Print the data type of Dictionary:
  ```
  d={"empname": "Lekha", "Org": "Puropale", "Dept": "AWS"}
  print(type(d))
  ```
  - <class 'dict'>

CONDITIONAL STATEMENTS:

- If: If condition is used to compare logical operations from mathematics.
- Written by using the if keyword.
- Logical conditions:
  - Equals: ==
  - Not Equals: !=
  - Less than: <
  - Greater than: >
  - Less than / Equals to: <=
  - Greater than / Equals to: >=

- If:
  - ```
    a=10
    b=54
    if b>a:
       print("True")
    ```
  - True

- Indentation: Defines the scope of the code in Python.
  - ```
    a=10
    b=54
    if b>a:
    print("True")
    ```
  - IndentationError: expected an indented block after 'if' statement on line 3

- Elif: The ‘elif’ keyword is similar to the else statement but the condition is defined.
  - ```
    a=100
    b=54
    if b>a:
       print("b is greater")
    ```
    ```

```
elif a>b:
   print('a is greater')
```

- a is greater

- Else:

```
a=10
b=54
if b>a:
   print("b is greater")
else:
   print('a is greater')
```

- b is greater

- If, else and Elif:

```
a=10
if a>0:
   print("a is +ve")
elif a==0:
   print("zero")
else:
   print('a is -ve')
```

- a is +ve

LOOPS:

For loop:

- For loop is used for iterating over a sequence.
- Used by using ‘for’ keyword.
- With the for loop we can execute a set of statements.
- Ex:

```
f=["apple", "banana", "mango", "grapes", "orange"]
for i in f:
   print(i)
```

  - apple
  - banana
  - mango
  - grapes
  - orange

- Break statement: we can stop the iteration of the loop.
- Ex:

```
f=["apple", "banana", "mango", "grapes", "orange"]
for i in f:
   print(i)
   if i=='mango':
       break
```

  - apple
  - banana
  - Mango

```
f=["apple", "banana", "mango", "grapes", "orange"]
for i in f:
   if i=='mango':
       break
   print(i)
```

  - apple
  - banana

Continue:

- Continue statement can stop the current iteration of the loop and continues with the next.
- Used during the filtering process.

```
f=["apple", "banana", "mango", "grapes", "orange"]
for i in f:
   if i=='mango':
       continue
   print(i)
```

  - apple
  - banana
  - grapes
  - orange

Range:

- Specified no.of times looping will be done.
- The ‘range()’ keyword will be used.

```
for i in range(6):
   print(i)
```

  - 0
  - 1
  - 2
  - 3
  - 4
  - 5

```
for i in range(6):
   print(i)
else:
   print("finished")
```

  - 0
  - 1
  - 2
  - 3

- 4
- 5
- finished

While loop:

- Used to execute a set of statements as long as condition is true.

```
i=1
while i<6:
   print(i)
   i+=1
```

- 1
- 2
- 3
- 4
- 5

```
i=1
while i<6:
   print(i)
   if i==3:
       break
   i+=1
```

-
- 1
- 2
- 3

```
i=1
while i<6:
   i+=1
   if i==3:
       continue
   print(i)
```

- 2
- 4
- 5
- 6