Indentation:

- refers to the spaces at the beginning of a code line.
- indicates a block of code.
- EX:

```
if 5 > 2:
  print("Five is greater than two!")
```

- Error:

```
if 5 > 2:
print("Five is greater than two!")
```

- We can give any number of spaces

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

- But you have to use same number of spaces in the same block of code:

```
#Error
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

```
#Error:
if 5 > 2:
 print("Five is greater than two!")
 print("Five is greater than two!")
```

Global Variables:

- created outside of a function.
- can be used by both inside and outside of a function

```
#Question
Create a function named greet that:
1) Displays: Hello myName
2) Where myName should be a global variable
```

```
myName = "Ravi" #Global Variable
def greet():
    print("Hello " +myName)

greet()
```

Local Variable:

- created inside a fun
- can only be accessible inside the fun, not outside,
- Ex:

```
def greet():
    myName = "Ravi"  #Local Variable
    print("Hello "+myName)

greet()
```

Both Global and Local variable having same name

- Can both global and local variable have same name> yes

```
myName = "Ravi" #Global

def greet():
  myName = "Darshan" #Local
  print("Hello " + myName)

greet()

print("Hello " + myName)
```

```
Python is Darshan
Python is Ravi
```

Problem:

- I want to create a global variable inside the fun. Is it possible??

```
# Error
def greet():
    myName = "Ravi" #Local Vraibale
greet()

print("Hello "+myName)
```

- Here comes the global keyword. It is used to create a global variable inside a function.

```
#Error
def greet():
    global myName = "Ravi";

greet()

print("Hello "+myName)
```

- The above is error because In Python, you need to declare it as global within the function before you can assign a new value to it

```
def greet():
    global myName
    myName = "Ravi"

greet()

print("Hello "+myName)
```

- Also, use the global keyword if you want to change a global variable inside a function.

```
x = "awesome"

def myfunc():
  x = "fantastic"

myfunc()

print("Python is " + x)
```

```
Python is awesome
```

- But i want output: "Python is fantastic"

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

BOOLEANS

- Two values: True or False

```
print(10>20)  # False
print(20==20) # True
print(10<=10) # True
print(5=5) #Syntax error
```

> bool() function:

- evaluates any values and returns True or False
- Almost all values are True if it has content
- Any String is true except empty string
- Any number is True, except 0
- Any list, tuple, set, and dictionary are True, except empty ones.
- Empty values are false: [],(),{},"", 0, None

```
print(bool("Ravi")) #True
print(bool(15)) # True
print(bool("")) # False (Empty String)
print(bool(0))  # False
print(bool(["red", "green", "blue"])) #False
print(bool([])) # False
print(bool({})) # False
print(bool(())) # False
print(bool(None)) # False
print(bool(False)) # False
```

- Can functions return boolean value: Yes

```
Create a function:
1) takes two paramters
2) if param1 > param2, fun should return True, Otherwise false
```

```
def checkParam(x,y):
    return x>y

print(checkParam(10,6)) #True
```

> isinstance() function:

- it is built-in function that returns a boolean value
- used to determine if an object has a certain data type

```
x = 7
print(isinstance(x, int)) # True
```

```
x = 5.6
print(isinstance(x, float)) # True
```

```
x = "Ravi"
print(isinstance(x, string)) # Error
```

```
x = "Ravi"
print(isinstance(x, str)) # True
```

Operators:

- Used to perform operations on variables and values.

Types:

> 1 . Arithmetic:

```
+ , - , * , / , % , **(expoentiation), //(Floor Divison)
```

```
x = 10
y = 3

print(x % y) # 1
print(x ** y) # 1000
```

Differnce between / and //

- /(Regular divison):
- returns the result as floating point number
- Means includes the decimal even if it is whole numbers

```
print(5/2) # 2.5
print(2/2) # 1.0
print(38/10) #  3.8
```

- //(Floor Divison):
- returns the result as integer number(no decimal part)
- returns largest integer which is <= result

```
print(5//2) #  2
print(2//2) #  1
print(38//10) # 3.8
```

2 . Assignment:

```
x = 5
x += 5 # x = x + 5
print(10)
```

```
x = 10
x /= 3 # x = 10 /3
print(x) # 3.3333333333333335
```

```
x = 5
x **= 3
print(x)
```

```

```

```
# Bitwise AND
x = 5
x &= 3
print(x)

"""
5 (x)   =  101
3       =  011
--------------
x &= 3  =  001 (1 in decimal)
"""
```

- Bitwise or

```
x = 5
x |= 3
print(x) # 7

"""
5 (x)   =  101
3       =  011
--------------
x |= 3  =  111 (7 in decimal)
"""
```

- Bitwise XOR(^) : 1 if bits are different

```
x = 5
x ^= 3
print(x) # 6
```

```
"""
5 (x)   =  101
3       =  011
--------------
x ^= 3  =  110 (6 in decimal)
"""
```

3 . Comparison Operators:

```
==, != , > , < , >= , <=
```

4 . Logical Operators:

```
and(&)
or(|)
not()
```

> 5 . Idenitity operators(is, is not):

- used to compare the objects
- doesnot compare whether they have equal value.
- it compares whether they are actually the same object i.e having same memory location.

```
x = ["red", "blue"]
y = ["red", "blue"]
z = x

print(x is z)

# returns True because z is the same object as x

print(x is y)

# returns False because x is not the same object as y, even if they have the same
content

print(x == y) # True
```

```
x = ["red", "blue"]
y = ["red", "blue"]
z = x

print(x is not z)

# returns False because z is the same object as x

print(x is not y)

# returns True because x is not the same object as y, even if they have the same
content

print(x != y) # False
```

LOOPS

- Python has two loops: for loop and while loop.

Diff bet While and For loop:

- While loop iterates over a condition
- For loop iterates over a sequence(list,tuple,dictionary,set, string)

WHILE LOOP:

- Executes the code as long as the condition is true.

```
i = 0
while i<5:
  print(i)
  i=i+1
```

- If we don't add the updation statement, it will be an infinite loop.

Control Statements:

- Manages the flow of program by determining which code should be executed next.

> Break :

- can stop the loop even if the condition is true

```
i=0
while i<5:
  print(i)
  if i==4:
    break
  i = i+1
```

> Continue

- can stop the current iteration and continue with the next.
- Question: Write a while loop that prints from 1 to 5 except 3:

```
# Wrong Code
i = 0
while i < 5:
  if i == 3:
    continue
  print(i)
  i = i + 1

# it will print 0, 1, 2 and after that it will go to infinite loop
```

- Right code

```
i=0
while i<5:
  i = i+1
  if i==3:
    continue
  print(i)
```

FOR LOOP

- iterates over a sequence(list, tuple, dictionary, set, string)
- Ex: Create a list and print its items using for loop:

```
colors= ["red", "blue", "green"]
for x in colors:
  print(x)
```

- Ex: Create a string and print its characters using for loop in one line.

```
collegeName = "Silicon Institute"
for x in collegeName:
  print(x, end="")
```

- Ex: Create a list and print its items using for loop except the any one item

```
colors = ["red","blue","green"]
for x in colors:
  if x=="red":
    continue
  print(x)
```

- Ex: Create a list and demonstrate the working of for loop using break

```
colors = ["red","blue","green"]
for x in colors:
  if x == "blue":
    break
  print(x)
```

> range() function:

- helps to loop for a speceified number of times

- it returns a sequence of numbers starting from 0(by default), increments by 1(by default) till reaches at a specified number
- Ex: range(6) means values from 0 to 5.
- Question: print first 10 whole numbers using for loop and range() function

```
for x in range(10):
  print(x)
```

- Starting avlue can be specified by adding parameter:

```
for x in range(3, 10):
  print(x)
```

- Range() fun increments bydefault by 1
- Increment value can be specified by adding third parameter

```
for x in range(2,10,3):
  print(x)
```

```
2
5
8
```

> Else in For Loop:

- executes when the loop is finished

```
for x in range(10):
  print(x)
else:
  print("Loop Over")
```

- If break is used else will not work:

```
for x in range(10):
  if x==5:
    break
  print(x)
else:
  print("Loop over")
```

> pass in For loop:

- for loops can't be empty:

```
# error:
for x in range(6):
```

- So we can use pass here

```
for x in range(6):
  pass
```

Do while loop

- In c

```
do {
  //Code
} while(condition);
```

- In Python?
- Python doesnot has do while to reduce complexity
- The same functionality of do while can be achieved through 'while' and 'break'
- In C:

```
int num = 0;
do {
  printf("%d",num);
  num = num+1;
} while(num<=0);
```

- In Python:

```
num = 0
while True:
  print(num)
  num=num-1
  if num<=0:
    break;
```

FUNCTIONS:

- Block of code
- Runs when called
- Data passed into functions are parameters:
- Created with **def** keyword.

```
def greet():
    print("Hello World!")

greet()  #function call
```

> Arguments(or args):

- Data can be passed into functions as arguments
- Specified after fun name inside ().
- multiple arguments are separated by comma
- EX:

```
def greet(name):
    print("Hello "+name);

greet("Ravi")
```

> Parameters(param) or Arguments:

- Both are same actually
- paramter is the variable inside () in fun defination
- Argument is the value sent during fun call
- Ex:

```
def greet(firstName, lastName):
    print("Hello! My Name is "+firstName+ " "+lastName);

greet("Ravi", "Nayak")

# firstName,lastName : parameters
# "Ravi", "Nayak" : arguments
```

- Error:

```
def greet(firstName. lastName):
    print(firstName+ " " +lastName);
greet("ravi");

# TypeError: greet() missing 1 required positional argument: 'lastName'
```

Arbitary Arguments(*args):

- If no. of argumenrts is unknown, add (*) before parameter name in function def
- In this way, the fun will receive a tuple of arguments and can be accessed using index

```
def progLanguages(*names):
    print("Now we are learning: "+names[1])
progLangueages("Java", "Python", "CPP")

# Now we are learning: CPP
```

KeyWord Arguments:

- Arguments can be sent in key-value pairs
- In this way, order of arguments does not matter.

```
def progLanguages(lang3, lang1, lang2):
    print("Now we are learning: "+lang2)
progLanguages(lang1="Java", lang2="Python", lang3="CPP")
```

Key Arguments along with Arbitary arguments

```
def progLanguages(**names):
    print("Now we are learning: "+names["lang2"])

progLanguages(lang1="Java", lang2="Python", lang3="CPP")
```

Default Parameter Value

- If we call the function without argument, it uses the default value:

```
def cities(name = "Delhi"):
    print("My City name is : "+name)

cities("Mumbai")
cities("Pune")
cities()
```

Collection data Types

- There are 4 built-in data types in Python to store collection of data:
- 1 . Lists: Ordered, Mutable, Allows Duplicate
- 2 . Tuple: Ordered, Unmutable, Allows Duplicate
- 3 . Set: Unordered, Unchangeable(but items can be removed or add), Unindexed, No Duplicate

- 4 . Dictionary: Ordered(in Python version 3.7 but unordered in version 3.6 and earlier), changeable, No duplicate

Python Lists:

- used to store multiple items in a variable
- created using square brackets.
- Ex:

```
iplTeams = ["CSK", "MI", "RCB"];
print(iplTeams);
```

- list items ordered, mutable, can have duplicate values.
- First item of list has index 0.

Ordered:

- List items have defined order
- Order will not change
- New elements are placed st the end.
- Note: Some list methods changes the order:

Mutable:

- we can change, add and remove items in a list

Allow Duplicates:

- List can have duplicate values
- EX:

```
iplTeams = ["CSK", "MI", "RCB", "CSK"]
print(iplTeams) #['CSK', 'MI', 'RCB', 'CSK']
```

List Length:

- len() fun is used to find the number of items in a list
- Ex:

```
colors = ["Yellow", "Blue", "Red"]
print(len(colors)) # 3
```

Data Types of List Items

- List Items can be of any data type

- Ex:

```
fruits = ["apple", "orange","Guava"] #string type
numbers = [1,2,4,5] # Integer Type
decision = [True, False, True] # Boolean Type

print(fruits) # ['apple', 'orange', 'Guava']
print(numbers) # [1, 2, 4, 5]
print(decision) # [True, False, True]
```

- List items can be of different data types:
- Ex:

```
myList = ["Ravi", 22, True, "BBSR"]
print(myList);
```

type():

- List are defined as objects with the data type list

```
myList = ["Mercury","Venus", "Earth","Mars"];
print(type(myList));  # <class 'list'>
```

list() constructor:

- used to create new list
- Ex:

```
myList = list(("apple", "orange", "Banana"));
print(myList) # ['apple', 'orange', 'Banana']
```

ACCESS ITEMS:

- Since list are indexed, its items can be accessed using index
- EX:

```
colors = ["blue","red","Green"]
print(colors[1]) #red
```

- Negative indexing means start from end.
- -1 refers to last item.

```
colors = ["blue","red","Green"]
print(colors[-2]); #red
```

Range of Indexes:

- Specify start index(included) and where to end(not included).
- Return value is new list with specified items.
- EX:

```
colors = ["red", "blue", "green" , "yellow" ,"orange", "violet"];
print(colors[1:4]) # ['blue', 'green', 'yellow']
```

- If start index is not mentioned, range will start from first item

```
colors = ["red", "blue", "green" , "yellow" ,"orange", "violet"];
print(colors[:4]); # ['red', 'blue', 'green', 'yellow']
```

- If the end index is not mentioned, range will start go on to the end of the list:

```
colors = ["red", "blue", "green" , "yellow" ,"orange", "violet"];
print(colors[2:]); #['green', 'yellow', 'orange', 'violet']
```

Range of Negative Indexes

- EX:

```
colors = ["red", "blue", "green" , "yellow" ,"orange", "violet"];
print(colors[-4:-1]); #['green', 'yellow', 'orange']
```

CHheck if Item Exists:

- Use **in** keyword

```
color = ["red", "blue", "green"]
if "green" in color:
    print("Yes")
```

Change Item Value:

- Refer Index Value:

```
# create a list, modify any item value and print the new list

progLang = ["Java","C","CPP"]
print(progLang)

# changinng 2nd item:
progLang[1] = "Python"

print(progLang)
```

```
['Java', 'C', 'CPP']
['Java', 'Python', 'CPP']
```

> Changing Item Values within a Range:

- Question: create a of list 5 items and then replace a range of list items between index 2 and 4(not included).
- Ans:

```
# creating a list of 5 items
progLang = ["C","Java","Python","CPP","Javascript","Kotlin"]
print(progLang)

# inserting items between index 2 and 4
progLang[2:4] = ["HTML","CSS"]

print(progLang)
```

```
['C', 'Java', 'Python', 'CPP', 'Javascript', 'Kotlin']
['C', 'Java', 'HTML', 'CSS', 'Javascript', 'Kotlin']
```

- If you insert more than you replace, new items will added and remaining items will be moved.

```
progLang = ["C","Java","Python","CPP","Javascript","Kotlin"]
print(progLang)

#replacing only 2nd index(3 is not counted) with two items
progLang[2:3] = ["HTML","CSS"]

print(progLang)
```

```
['C', 'Java', 'Python', 'CPP', 'Javascript', 'Kotlin']
['C', 'Java', 'HTML', 'CSS', 'CPP', 'Javascript', 'Kotlin']
```

- In this case the length of list changes when no. of items inserted does not match the no. of items replaced.
- Now, inserting less than replace:

```
progLang = ["C","Java","Python","CPP","Javascript","Kotlin"]
print(progLang)

progLang[3:6] = ["HTML"]

print(progLang)
```

```
['C', 'Java', 'Python', 'CPP', 'Javascript', 'Kotlin']
['C', 'Java', 'Python', 'HTML']
```

Insert List Items:

- How to insert list items without replacing
- using insert() fun:
- It inserts items at the mentioned index.

```
Question: Create a list of 4 items and insert a new item at index 3
```

```
colors = ["Red", "Blue" ,"Green","Black"]
print(colors)
colors.insert(3, "Orange")
print(colors)
```

```
['Red', 'Blue', 'Green', 'Black']
['Red', 'Blue', 'Green', 'Orange', 'Black']
```

Append List Items:

- what is append() methods
- to add an item at the end of the list

```
color = ["red","blue","green"]
print(colors)
colors.append("black")
print(colors)
```

```
['red', 'blue', 'green']
['red', 'blue', 'green', 'black']
```

Extend List Items:

- what is extend() method used for?
- used to append or add items of another list to the end of current list

```
colors = ["yellow","purple","green"]
rgb = ["red","blue","green"]
colors.extend(rgb)
print(colors)
```

```
['yellow', 'purple', 'green', 'red', 'blue', 'green']
```

- Extend can append any iterable object(list, tuples, set, dictionaries, etc).

```
#extending tuple with list
colors = ["yellow","purple","green"]
rgb = ("red","blue","green")
colors.extend(rgb)
print(colors)
```

Looping Through a List:

> For Loop:

```
 Print all list items using for loop:
```

```
myList = [1,2,3,4]
for x in myList:
  print(x)
```

```
1
2
3
4
```

```
print  all list items using for loop by refering index number:
```

```
cities = ["Delhi", "Mumbai", "Pune"]
length = len(cities)
for i in range(length):
  print(cities[i])
```

```
Delhi
Mumbai
Pune
```

> Using While Loop:

```
print all list items using while loop
```

```
cities = ["Delhi","Mumbai","Pune"]
i = 0
length = len(cities)
while i < length:
  print(cities[i])
  i = i+1
```

Sorting the Lists:

- sort(): built-in fun to sort list in ascending by default:

```
#sort alphabetically
colors = ["red", "blue", "green","yellow","orange"]
colors.sort()
print(colors)
```

```
#sort numerically
thislist = [100, 50, 65, 82, 23]

thislist.sort()

print(thislist)
```

```
#Sort in descending
colors = ["red", "blue", "green","yellow","orange"]
colors.sort(reverse=False)
print(colors)
```

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```

Copy Lists:

List Comprehension:

- Process of reducing the syntax during looping through lists
- It helps when we create a new list based on the values of existing list
-
- Ex

```
cities = ["Delhi","Mumbai","Pune"]
for x in cities:
  print(x)
```

- The above code can be reduced using list comphrension:

```
cities = ["Delhi","Mumbai","Pune"]
[print(x) for x in cities]
```

Tuples:

- used to store multiple items in a single variable.
- created using round brackets
- EX:

```
myTuple = ("red", "blue", "green");
print(myTuple) #('red', 'blue', 'green')
```

- Tuple items are ordered, unchangeable , can have duplicates

Dictionary

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

IF ELSE PROGRAM:

Check Whether a number is poistive, Negative or Zero

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

Check Even or Odd:

Using Modulus Operator(%)

```
num = int(input("Enter the number: "))

if num%2 == 0:
    print(num, "is Even")
else:
    print(num, "is Odd")
```

Using Bitwise AND (&)

```
num = int(input("Enter the number: "))

if num & 1:
    print(num, "is Odd")
else:
    print(num, "is Even")
```

```
"""
2. The program then uses the bitwise AND operator (&) with 1 to check the least
significant bit of the binary representation of the number.
3. If the result is non-zero, it means the number is odd.
"""
```

Program to Find the Grade of a student:

```
Total 4 subjects (Eng, Math, history, Science)
If Avg >= 90 : Grade A
If Avg >= 80 and < 90 : Grade B
If Avg >= 70 and <80 : Grade C
If Avg <= 70 : Grade d
```

```
eng = int(input("Enter marks of English: "))
math = int(input("Enter marks of math: "))
his = int(input("Enter marks of History: "))
sci =int(input("Enter marks of Science: "))

Avg = (eng + math + his + sci) / 4
if Avg >= 90:
    result = "A"
elif Avg >= 80 and Avg < 90:
    result = "B"
elif Avg >= 70 and Avg < 80:
    result = "C"
else:
    result = "D"

print("Average: ", Avg)
print("Grade: ", result)
```

LOOPS PROGRAM

Print All Even Numbers in a Range

```
lowRange = int(input("Enter the lower limit :"))
highRange =int(input("Enter the higher limit :"))

for i in range(lowRange,highRange):
    if(i%2==0):
        print(i)
```

Program to print the table of a given number:

```
num = int(input("Enter the number :"))
for i in range(1,11):
    print(num , "x" ,i,"=",num * i)
```

Find the reverse of a Number

```
num = int(input("Enter the number : "))
revNum=0
while(num>0):
    digit=num % 10
    revNum = revNum*10 + digit
    num =int(num/10) #or num = num//10 (Floor divison)

print("Reverse : ", revNum)
```

Check If a Number is Palindrome

- A number is a palindrome if its reverse is equal to the number itself.
- EX: 121, 454, 888, etc.

```
num = int(input("Enter the number : "))
temp=num
revNum=0
while(num>0):
    digit=num % 10
    revNum = revNum*10 + digit
    num =int(num/10) #or num = num//10 (Floor divison)
if(temp==revNum):
    print("Palindrome")
```

```
else:
    print("Not a Palindrome")
```

Print all Numbers between 0 and 100 which is not divisible by 2 and but divisible by 3

```
for i in range(0, 31):
    if(i % 2 !=0 and i % 3 == 0):  # u can use & also.
        print(i)
```

Program to Count Number of digits in a Number:

```
num = int(input("Enter number:"))
count=0
while num>0:
    count=count+1
    num = num//10
print("Total Number of digits : ",count)
```

Program to print sum of digits of a number

```
num = int(input("Enter a number: "))
sum=0
while num>0:
    digit = num % 10
    sum = sum + digit
    num = num//10;

print(sum)
```

Programs to print sum of digits of a number using list

```
1. create an empty list
2. insert all digits into the list
3. Add all list items
```

```
digitsList = []
num = int(input("Enter a number: "))
while num>0:
    digit = num % 10
    digitsList.append(digit)
    num=num//10
print("Sum: ", sum(digitsList))
```

Program to print all divisors of a number and also its sum

- Ex: if num = 10, divisrors are 2,5,10. So sum = 18

```
num = int(input("Enter an integer : "))
print("Divisors are : ")
sum = 0
for i in range(2, num+1):
    if(num % i==0):
        print(i)
        sum = sum+i
print("Sum :", sum)
```

- Using while loop:

```
num = int(input("Enter an integer : "))
print("Divisors are : ")
sum = 0
i = 2
while i <= num:
    if num % i == 0:
        print(i)
        sum = sum+i
    i += 1
print("Sum :", sum)
```

Program to print the smallest divisor of a Number

```
num = int(input("Enter an integer:"))

for i in range(2,num+1):
    if num % i==0:
        print("Smallest divisor is:",i)
        break;
```

Program to find the largest divisor of a Number except the number

- Largest divisor = number/smallest divisor

```
num = int(input("Enter an integer:"))

for i in range(2,num+1):
    if num % i==0:
        break;
print("Largest Divisor: ", num//i);
```