(iv) Hexadecimal (base 16)

The range is from ~~0-15~~ 0-9, A-F

Representation : 0x12, 0XAD

Float : This value is represented by 'float' class. It is a real number with floating point represented by a decimal point (.).
float can also be scientific numbers with e or E to indicate the power of 10

Eg: 2E7, -7.952797771

```
a = 10E2
print(a)
```

Note - There is no double datatype.

Complex : complex number is represented by the 'complex' class. Complex datatypes can be used to solve mathematical functions like a+bj - where a is real part, b is imaginary part
It is specified as (real part) + (imaginary part) j

Eg:

```
imgNum = 7+22j
print(imgNum)
imgNum2 = -2+j
print(imgNum2)
c = 10.2+1.7j
print(c)
Print(imgNum - imgNum2)
```

type function:

type() function in python is used to determine the type of data-type.

Prog:

```
x = 10
y = 3.14
z = "Elon"
c = 2.9j
b = True
print(type(x))  # <class 'int'>
print(type(y))  # <class 'float'>
print(type(z))  # <class 'str'>
print(type(c))  # <class 'complex'>
print(type(b))  # <class 'bool'>
```

---

type conversion: (or) type casting:

type casting is a process to convert a variable or value from one datatype to another datatype. Python have multiple inbuilt functions for type casting.

Inbuilt func	Description
1) int()	- used to convert other types to integer datatype
2) float()	- used to convert other types to float datatype
3) complex()	- used to convert other type to complex datatype

4) bool() - used to convert other type to boolean datatype

5) str() - used to convert other types to string datatype.

integers

int() - It is used to convert any datatype integer except complex datatype and string datatypes

```
float_num = 3.14
string_str = "Elon"
complex_c = 3.14j
str_num = "111"
print(int(float_num)) # 3
print(int(string_str)) # error.
print(int(complex_c)) # error it is not possible
print(int(str_num)) # 111
bool_num = True
print(int(bool_num)) # 1
```

---

float() - It is used to convert any datatype to float data type except complex and string literals.

```
int_num = 19
str_literal = "Musk"
complex_num = 9.9j
print(float(int_num)) # 19.0
# str_liter
print(float(str_literal)) # error
```

4) bool() - used to convert other type to boolean datatype

5) str() - used to convert other types to string datatype.

Integer

int() - It is used to convert any datatype integer except complex datatype and string datatypes

```
float_num = 3.14
string_str = "Elon"
complex_c = 3.14j
str_num = "111"
print(int(float_num)) # 3
print(int(string_str)) # error.
print(int(complex_c)) # error it is not possible
print(int(str_num)) # 111
bool_num = True
print(int(bool_num)) # 1
```

---

float() - It is used to convert any datatype to float data type except complex and string literals.

```
int_num = 19
str_literal = "Musk"
complex_num = 9.9j
print(float(int_num)) # 19.0
# str_liter
print(float(str_literal)) # error
```

```
print(float(complex_num)) # error
print(float("999")) # 999.0
```

---

complex() - complex(). we have two forms in complex() type casting.

(i) complex(x)

there, x is real part that is complex number is like x+0j

(ii) complex(x, y)

here, x is real part and y is imaginary part i.e complex number is like x+yj

Program:

```
print(complex(72)) # 72+0j
print(complex(9.1)) # 9.1+0j
print(complex(True)) # 1+0j
print(complex("true")) # error
print(complex("11")) # 11+0j

print(complex(1, 2)) # 1+2j
print(complex(1.1, 2.2)) # 1.1+2.2j
print(complex(True, False)) # 1+0j
print(complex("Hi", "complex")) # error
print(complex("11", "12")) # 11+12j
```

bool() - It is used to convert any datatype to boolean data type

Prog:

```
print(bool(0))    # False - valid int to boolean is possible
print(bool(10))   # True - valid other than 0 returns True

print(bool(7.2))  # True
print(bool(2+j))  # True
print(bool(0+0j)) # False
print(bool())     # False
print(bool("true")) # True
```

note: If both real and imaginary parts are zero (0) then returns False

If both real and imaginary parts are other than zero, returns True

---

String data type

A string is a collection of one or more characters enclosed in (single quotes; double quotes or triple quotes on both sides. In python, there is no character data type.

A character is a string of length 1. It is represented by 'str' class. In python, string is an immutable sequence datatype

```
name = "elon musk"
city = 'LA'
country = "USA"
```

```
print("Name:", name)
print("city:", city)
print("country:", country)
```

List :

List is an advanced data type. A List is an ordered collection of data items. The items in a list does not need to be of same data type. Different types of data lists are used to store multiple items in a single variable. Lists are represented using [ ] - a pair of square brackets.

* List is mutable, means we can make changes in the List

Eg: `l = [10, 20, 30, 40]`

`student = ["23608-CM-008", "NU", "Male", True, 30.2, 17]`

Tuple :

Tuple is an advanced data-type. It is provided tuple() function. Tuple works like a list but it is ordered and immutable. Tuples are defined by enclosing element in parentheses - ( ) separated by coma (,)

Eg: `t = (10, 20, 30, 40)`

`student = ("23608-CM-008", True, 17, 30.2)`

Set :

In python, set is unordered collection of advanced datatype which is mutable and has no duplicate elements. elements are enclosed in curly braces - { }

Eg1 S = { 10, 20, 30, 40, 50 }
print(S)
o/p { 40, 50, 20, 10, 30 }

>>> S2 = { 10, 20, 30, 10, 20, 30 }
>>> S2
{ 10, 20, 30 } # duplicate values are eliminated internally and automatically

---

Dictionary :

Python dictionary is advanced datatype which is unordered collection of data items unlike other datatypes have only one value unlike dictionary. Dictionary items are represented in key:value pairs can also referred by using key name

Example :

```
Person = {
    "name" : "elon",
    "age" : 67,
    "companies" : ["spacex", "tesla", "x"]
}
print(person)
```

input( ) function :

Input and output statements in python :

Some of the functions like input() and print() are widely used for standard input and standard output operations.

input statement in python :

We have two ways to define input statements. They are -

1) raw_input(prompt)
2) input(prompt)

raw_input("Enter a string")

By default it takes anything in string format only, if required, do typecasting

input(prompt) - where prompt is the string we wish to display on the screen. This function first takes input from the user and converts it into string. type of returned object always is a string <class 'str'>

Prog :

```
name = input("Enter your name : ")
age = int(input("Enter your age : "))
address = input("Enter your address : ")
print("Name : ", name)
print("Age: ", age)
print("Address: ", address)
```

input() function works in python - when input() function executed, program flow will be stopped until the user has given the input.

* the text or message display on the output screen to ask the user to enter input value is optional i.e prompt will be printed on the screen is optional

* whatever you enter as input, input() function converts it into a string. If you enter an integer value till input() function converts into string. you need to explicitly convert into integer in your code using type casting.

Output statement in python

In python, to print the output to the console print() function is used.

Syntax of print()

print(*args, sep=' ', end='\n', file=None, flush=False)

* prints the values to a stream, or to sys.stdout by default

sep - string inserted between values, default a space
end - string appended after the last value, default a newline
file - a file-like object (stream); defaults to the current sys.stdout
flush - whether to forcibly flush the stream

dictionary

```
car = {
  "brand": "Lamborghini",
  "model": "Aventador",
  "year": 2014
}
print(car)
```

Format-1

print() with no args :-

print() function without args provides an empty newline.

Eg:

```
print()
```

Format-2

print() with String args -

Eg:

```
print("Hi all")  # String argument quoted with single quote or double quote

print('python' + 'programming')  # String arguments only concatenate with the '+' operator

print('Python' * 5)  # Repetition operator (*) apply Here one should be String and other should be int
```

Format-3

print() with n number of args -

```
a,b,c = 10, 20, 30
# print(a,b,c)
# print("are the values")
  print(a,b,c, "are the values")
```

~~The values are: 10 20 30~~

10 20 30 are the values. # By default space in between args.

sep attribute

If you want to print any separator in the args, we can include this "sep" attribute at the end of all args. the default separator is space.

Eg: print("elon", "musk", sep="_")

o/p : elon_musk

```
print("hi", "all", "welcome", sep=":")
# hi:all:welcome
```

end attribute (args) -

If you want to print two or more lines into a single line with a separator we can use end argument in print statement, to concatenate the next line, we can include 'end' arg at the end.

Eg: print("hi", end=":")
print("all", end=":")
print("welcome")
# hi:all:welcome

10:20:30 $ hi * all

```
print(10, 20, 30, sep=":", end="$")
print("hi", "all", sep="*")
```

Output Formatting

f-string

```
name = "John"
age = 18
print(f"my name is {name}. I am {age} years old")

city = "Guntur"
temperature = 38.2
print(f"today {city} is {temperature} degrees hot")
```

---

print() with formatting quotes :

%d - formatting code for int type.
%i - formatting code for int type
%f - formatting code for float type
%s - formatting code for string type

```
print("formatting code" % (variable))
print("my name is %s. I am %d years old" % ("John", 25))
print("today %s is %f degrees hot" % ("Guntur", 38.2))
```

a, b, c = 10, 20, 30

print("A: %d, B: %d, C: %d" %(a, b, c))

print() with replacement operator

print() function uses replacement operator.

'{}' is replacement operator.

Syntax:

print('indexed/non-indexed replacement operators'.format(variables))

Indexed

name = "JB"

age = 25

print("my name is {0}. I'm {1} years old".format(name, age))

non-Indexed

print("my name is {}. I'm {} years old".format(name, age))

---

name = "Nagavamsi"

branch = "CME"

ROLL_NO = "23G01-CM-008"

Print("I'm {0}. I'm studying {1}. my roll_no is {2}".format(name, branch, ROLL_NO))

print("I'm {}. I'm studying {}. my roll_no is {}".format(name, branch, Roll_NO))

```
x = 5
y = 10
print("the value of x is {} and y is {}".
      format(x,y))
# the value of x is 5 and y is 10
```

there, {} are used as placeholders. we can specify the order in which they are printed by using numbers.

```
a = "python"
b = "programming"
```

Indexed:

```
print("I love {0} and {1}".format(a,b))
print("I love {1} and {0}".format(a,b))
```

Non-Indexed

```
Print("I love {} and {}".format(a,b))
print("I love {} and {}".format(b,a))
```

we can also format strings like old printf style used in c programming language we can use the % operator.

```
a = 12.3456789
print("the value of a is %.2f "%a) #_12.35
Print("the value of a is %.4f "%a) #_12.3457
```

Input formatting

split()

```
a,b,c = input("Enter three values : ").split()
print(f"Entered first value : {a}")
print(f"second value : {b}")
print(f"third value : {c}")

lang1, lang2, lang3 = input("Enter three prog langs").split(",")
print(f"lang1 : {lang1}")
print(f"lang2 : {lang2}")
print(f"lang3 : {lang3}")
```

Enter three prog langs : c,go,rust

lang1 : c

lang2 : go

lang3 : rust

Formatted Input

user to enter multiple values or inputs in one line.

In c/c++, user can enter multiple values in one line, using scanf() but in python, user can enter multiple values or inputs in one line by using split() method.

Using split() method :

This function helps us in getting multiple inputs from the user. It breaks given input by the specified separator. If the separator is not provided, then any whitespace is a separator.

generally, users use a split() method to split a Python string but not one can use it in taking multiple inputs.

Python Operators :

Python operators in general are used to perform operations on values and variables. They are standard symbols used for the purpose of logical and arithmetic operators. In python, there are six types of operators -

(1) Arithmetic operators
(2) Relational or comparison operator
(3) Bitwise operator
(4) Assignment operator
(5) special operator < membership operator, identity operator
(6) Logical operators

Arithmetic operators :-

Arithmetic operators are used to perform various mathematical operations like addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and float division (//), exponentiation (**)

operator	name	Example
+	addition	x+y
-	subtraction	x-y
*	multiplication	x*y
/	division	x/y
%	modulus	x%y
//	float division	x//y
**	exponentiation	x**y

Addition operator :- uses plus (+) symbol for adding values blw int, float and in combinations it also acts as concatenation operator in blw strings literals only

```
>>>a=10
>>> b=20
>>> a+b #30
```

```
>>> a="python"
>>>b="programming"
>>> a+b #python programming
```

```
>>>a="python"
>>>b=20
>>> a+b # error
```

+ acts as concatenation of in blw two strings only

```
>>> a=10.5
>>>b= 2
>> a+b # 12.5
```

Subtraction operator :- uses minus(-) symbol for subtracting values between int, float and in combinations.

```
>>> a= 10
>>> b= 20
>>> a-b # -10
```

```
>>> a=105.2
>>>b= 2
>>>a-b # 103.2
```

Multiplication operator :- uses star (*) symbol for multiplying values between int, float and in combination. It works as repetition operation between string, list, tuples etc,

```
>>>a = 10
>>>b= 20
>>>a*b #200
```

```
>>>a = (10, 20)
>>>a * 2
(10, 20, 10, 20)
# twice values will be repeated.
* acts a repetion operator
```

```
>>>a = "Py"
>>> a * 3
Py
Py
Py
```

Division operator :- uses slash(/) symbol for dividing values between int, float and in combinations. Division operator '/' always returns float value only

```
print(5/2) # 5.0
```

---

modulus operator :- uses percentage(%) symbol to know the remainder of the division operation blw two integer values only.

```
>>>a = 11
>>> b = 3
>>> a%b #2
```

Exponent power Operator :→ Exponent/ power operator uses (**) doublestar on integer, float and complex values

```
>>>a=2
>>>a**4
16
```

```
>>> b = 11.1
>>> b ** 1
11.1
```

```
>>> c = 1.2
>>> c ** 2
1.44
```

Float division operator :- uses double slash (//) and it works on both integer and float data types. If two operands are integers, then it returns integer value or if two operands are float then it returns float value. It always prints nearest integer value in float sign when the numerator is integer.

```
>>> a = 5          >>> a = 5.0          >>> a = 5.0
>>> b = 2          >>> b = 2.0          >>> b = 2
>>> a//b          >>> a//b            >>> a//b
# 2               # 2.5               # 2.0
```

```
a = int(input("Enter a number : "))   # 5
b = int(input("Enter another number : "))  # 2
print("a+b :", (a+b))   # 7
print("a-b :", (a-b))   # 3
print("a*b :", (a*b))   # 10
print("a/b :", (a/b))   # 2.0 (float)
print("a%b :", (a%b))   # 1
print("a**b :", (a**b))  # 25
print("a//b :", (a//b))  # 2 (int)
```

Relational or Comparison operators :

It compares the values on either side of the operand and determine the relation blw them.
It is also refered to as relational operator.
various comparison operators in python are
<, >, ==, !=, <=, >=

operator	Name	Example
==	equal	x == y
!=	not equal	x != y
>	greater than	x > y
<	less than	x < y
>=	greater than / equal to	x >= y
<=	less than / equal to	x <= y

---

less than (<) : It checks the content blw two sides of operand and returns boolean values accordingly

```
>>> a = 10
>>> b = 2
>>> a < b
False
```

```
>>> a = "Python"
>>> b = "python"
>>> a < b
False
```

Greater than (>) : It checks the content blw two sides of operands and returns boolean values accordingly

```
>>> a = 10
>>> b = 2
>>> a > b
True
```

```
>>> a = "Python"
>>> b = "python"
>>> a > b
False
```

less than/equal to (<=): It checks the content blw two sides of operands and return boolean value accordingly.

```
>>> a=10
>>> b=2
>>> a <= b
    False
```

```
>>> a= "python"
>>> b= "python"
>>> a<=b
    true
```

greater than/equal to (>=): It checks the content blw two sides of operands and return boolean value accordingly.

```
>>> a=10
>>> b=2
>>> a >= b
    true
```

```
>>> a = "python"
>>> b = "python"
>>> a >= b
    true
```

```
>>> a = "python"
>>> b = 5
a >= b #error
```

equality (==): It checks the content blw two sides of operands and return boolean value accordingly. equality check will be done in blw incompatibility data types only.

```
>>> a=10
>>> b=2
>>> a==b
    False
```

```
>>> a= true
>>> b: true
>>> a==b
    true
```

not-equality (!=): It checks the content blw two sides of operands and return boolean value accordingly. not-equality check will be done blw in compatibility datatypes

```
>>> a = True
>>> b = True
>>> a != b
False
```

Chaining comparision operator in python :

chaining comparision is allowed to check if all the comparisions in expression returns true, then only the complete expression returns true otherwise returns false

Eg: 10 < 20 < 30 < 40 # True, each expression is true from left to right and return true

Prog :

```
a = int(input("Enter a value"))   # 5
b = int(input("Enter b value"))   # 2
print("a == b:", (a == b))        # false
print("a != b:", (a != b))        # True
print("a > b:", (a > b))          # true
print("a >= b:", (a >= b))        # true
print("a < b:", (a < b))          # false
print("a <= b:", (a <= b))        # false
```

Assignment operator -

Assignment operator is used to assign values from rightside to left ~~right~~ side. Single equal to (=) is used as assignment operator.

operator	Same as	Example
=	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 (pipe)

a=10 # 10 value will be assigned to a

a=b=c=d=10 # chaining in assignment operator will be possible in python.

a,b,c,d = 10,20,30,40 # special assignment feature in python but no of left hand side variable and no of right hand side args should be equal.

a,b=b,a # This is also possible in python without using a temporary variable, we can swap two values.

---

a>b ? printf("a is greater") : printf("b is greater");

Ternary operator:

In python, a simplified single lined ternary operator is used for decision making.
Ternary operator reduces the code length.

x = first_value if condition else second_value

---

x = 30 if 40<30 else 40 # x = 40

greater_than = a if a>b else b
print("Greater : ", greater_than)

less_than = a if a<b else b
print("Smaller : ", less_than)

Logical operators :

Logical operators in python are used for conditional statements are true or false. There are three types of logical operators in python -

Logical and
" or
" not

For "and" operator, it returns true if both operands are true (Left and Right)

For "or" operator it returns true either of the operands (Right or left side) is true.

For 'not' operator, returns true if the operand is false and false otherwise.

Operator	Description	Example
and	- returns True if both stmts are True	- a<5 and b<10
or	- returns true if one of the stmts is True	- a<5 or b<10
not	- retrue true if false and vice versa	- not true # false not False # true

not (a<5 and b<10)

```
>>> a = True
>>> b = False
>>> a and b
False
```

Prog:

```
a = bool(input("Enter a value : "))
b = bool(input("Press enter : "))

print("a and b :", (a and b))
print("a or b :", (a or b))
print("not a :", (not a))
```

---

write a program to find minimum from the given two numbers at run time.

Prog:

```
a = int(input("Enter number 1 : "))
b = int(input("Enter number 2 : "))

min = a if a<b else b

print("minimum :", min)
```

Module :

Module is a group of functions, variables, methods, classes. Group of modules are known as libraries

Modules are simply files with the ".py" extension containing python code that can be imported inside another python program.

In python, many libraries and modules are available If any module is needed in your program, then you must import module to your code.

import module ← Syntax

calc.py

```
def sum(a,b):
    return a+b

def diff(a,b):
    return a-b

def power(a,b):
    return a**b

def avg(a,b):
    return (a+b)/2

def product(a,b):
    return a*b

def divide(a,b):
    return (a/b)
```

test_calc.py

```
import calc

print(f"Sum : {calc.sum(10, 20)}")  # 30
print(f"Diff : {calc.diff(10, 20)}")  # -10
print(f"Product : {calc.product(10, 20)}")  # 200
print(f"Division: {calc.divide(10, 20)}")  # 0.5
print(f"Avg: {cal.avg(10, 20)}")  # 15.0
print(f"Power : {calc.power(2, 3)}")  # 8
```

```
import math as m
print(m.pi)
print(f"Factorial of 5 {m.factorial(5)}")
print(f"Sqrt of 2 {m.sqrt(2)}")
```

```
a = int(input("Enter num1 : "))
b = int(input("Enter num 2 : "))
c = int(input("Enter num 3 : "))

min = ~~a if a<b~~
```

~~if a<b else b<c~~

a < b

a<c b<c

a c b c

```
min = a if (a<b and a<c) else b if (b<a and b<c)
                         else c

print("minimum : ", min)
```

Second form (ternary operator)

x = first_value if condition_1 else second_value
    if condition_2 else third_value.

write a program to print value by two conditions

x = 10 if 20 < 30, else 40 if 50 < 60 else 70
print(x)

In the above, condition 1 calculated first after that finalized value 1 calculated to the remaining condition is evaluated

---

Maximum of three numbers:

a = int(input("Enter num1: "))
b = int(input("Enter num2: "))
c = int(input("Enter num3: "))

max = a if a > b and a > c else b if b > c else c
print(max)

Bitwise operator:

It indicates that the operators perform wise in the memory. Bitwise operators are used to compare binary numbers. These operators act on bit and perform bit-by-bit operation.

The operator can be applied on int and boolean datatypes only, otherwise if you try to apply other than it simply returns error

operator	Name	Description
&	and	– sets each bit to one if both bits are one.
|	or	– Sets each bit to one if one of the two bits is one
^	XOR	– Sets each bit to one if only one of two bits is one.
~	complement	– Inverts all the bits
>>	Signed Right Shift	– shifts right by pushing copies of the leftmost bit in from the left and let right most bits fall off
<<	zero fill left shift	– Shift left by pushing zeroes in the form the right and let leftmost bits fall off.

Bitwise 'and (&)': If both bits are one, then only it returns 1 otherwise returns 0 (Zero)

4 & 5 — 100 & 101 = 100 — 4

5 & 3 — 101 & 011 = 001 — 1

101 & 010 = 000

Bitwise 'or' (|): If atleast one bit is one, then it returns 1 otherwise returns 0

4 | 5 — 100 | 101 = 101 — 5

5 | 3 — 111 — 7

Bitwise XOR (^): If both bits are different then only it returns 1 otherwise returns 0

4 ^ 5 — 001 — 1

5 ^ 3 — 110 — 6

complement (~) — this is one's complement.

The formula is $\boxed{\sim x = -x-1}$

$\sim 5 = -5-1 = -6$

left shift (<<) — shifting the bits towards left and vacent places of the right side will be filled with zeroes only.

7 << 2 — 0000 0111 — 0001 1100 = 28

10 << 2 = 40

0000 1010

00101000 = 8 + 32 = 40

7 6 5 4 3 2 1 0

0101

1 << 2 = 4

0000 0001

0000 0100 = 4

Right shift (>>) - shifting the bits towards right cell and vacant (places) of the left most end side are filled by sign bit only

7 >> 2 = 1

0000 0111

0000 0001 = 1

10 >> 2 = 2

0000 1010

0000 0010 = 2

5 >> 3 = 0

0000 0101

0000 0000 = 0

Prog:

```
a = int(input("Enter a number:"))  # 5
b = int(input("Enter b number:"))  # 2
print("a & b:", (a & b))  # 0
print("a | b:", (a | b))  # 7
print("a ^ b:", (a ^ b))  # 0
print("~a:", (~a))  # -6
print("a << b:", (a << b))  # *
print("a >> b:", (a >> b))  #
```

Special operators:

There are two types of special operators available in python. they are -

(i) Identity operator
(ii) membership operator

Identity operator: are used to deal with addresses of two objects and to compare addresses. the two Identity operators used in python are - is, is not

is - returns true if two operands have same address

is not - returns true if two operands have different location address

Operator	description	Example
is	- returns true if both variables are the same object	x is y
is not	- returns true if both variables are not the same object	x is not y

```
>>> a = 10
>>> b = 10
>>> id(a)  # id() function is used for address of
           # id(a) returns address of a
>>> a is b  # true
```

```
a = 30
b = 10
c = a
print(a is not b)  # true
print(a is c)  # true
```

Membership operator : to check if the object is a member in the given sequence or not. the sequence may be string, list, tuple, etc;
there are two types of membership operator -

(i) in
(ii) not in

membership opr gives result based on variable present in the specified sequence or string

Operator	Description	Example
in	- returns true if a sequence the with specified value is present in the object	x in y
not in	- returns true if a sequence with the specified value is not present in the object	x not in y.

```
l = [10, 20, 30, 40]
print (10 in l) # true

l = (10, 'hi', 40, 10.5)
print ('h' in l)  # true
print (90 not in l) # true
s = "hello world"
print ("hello" in s) # true
print (" " in s)  # true
print ('o' in s)  # true
```

Operator Precedence :- (PEMDAS) for arithmet

PEMDAS - For arithmetic (operators) expressions

10 + 12 * 3 / . 34 / 8

10 + 36 / . 34 / 8

10 + 2 / . 34 / 8

10 + 2 / 8 80 + 2 =

10 + 0.25

10.25

9) 200.25
16
40
40

9) 82 [illegible]
8
20
18
40
40

9) 34 4
32
2

Operator precedence :-

This is used in an expression with more than one operator with different precedencies to determine which operator performs first.

precedence of + and *

```
a = 2 + (0*3)
print(a)
```

Operator associativity :

If an expression contains two or more operators with same precedence. then, operator associativity is used to determine. It can be either be left-to-right or from right-to-left. all operators except exponent follows left-to-right * and / are equal evaluation, left-to-right associativity is followed

```
2**3**2
2** 9 = 512
```

Operators	Meaning
decreasing order of precedence - ** -	exponent
*, //, *, / , -	multiplication, division, float division, modulus
+, - -	addition, subtraction
comparision Operational >=, <=, <, >, ==, !=	greater than equal, less than equal, greater than, less than,

Assignment operator	meaning
%=, /=, *=, -=, +=, //=,	assignment operator
Identity operators is, is not	—
Membership operator in, not in	—
not, or, and	— logical operator

---

45 + 13 - 9/3 * 7

~~45 + 13 - 9/21~~

45 + 13 - 3 * 7

45 + 13 - 21

56 - 21 = 35

~~xxxxxxxxxx~~

/ > * if appear in same expression
% > / "

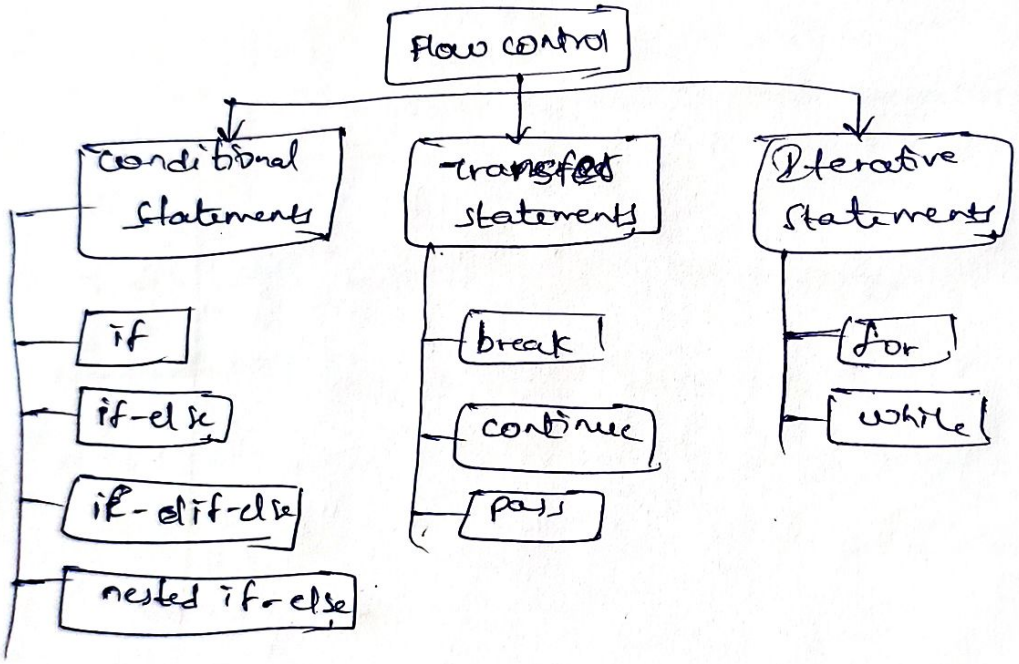
Program to check whether eligible to vote or not

```
age = int(input("Enter your age : "))

if age >= 18 :
    print ("Eligible to vote")

else:
    print ("not eligible to vote!")
```

2. Control flow and Loop stmts in Python

Control flow statements :

A program's control flow is the order in which program's code execute. the control flow of Python program is regulated by conditional statements, loops and function calls



if-statement :

In python, conditional statements act depending on whether a given condition is true or false. you can execute different blocks of depending on outcome of a condition.
If the condition is true, then the true block of code will be executed. if the condition is false then the block of code is skipped. controller moves to next line

Syntax - simple if

```
if condition:
    indentation -> statements
```

(or)

```
if condition:
    statement 1
    statement 2
    :
    statement n
```

even or odd -

```
num = int(input("Enter a number: "))
if num%2 == 0:
    print(num, "is even")
if num%2 != 0:
    print(num, "is odd")
```

---

if - else :

if-else statement checks the condition and executes if-block code when the condition is true and if the condition is false, it will execute else - block code

Syntax :

```
if condition:
    Indentation -> statements
else:
    Indentation -> statements
```

Eg:

```
num = int(input("Enter a number:"))
if num%.2 == 0:
    print(num, " is even")
else:
    print(num, " is odd")
```

Write a program to find whether given number is divisible by 5 or not

Prog:

```
num = int(input("Enter a number to check
        divisibility by 5:"))
if num%.5 == 0:
    print(num, "is divisible by 5")
else:
    print(num, " is not divisible by 5")
```

---

write a program to find maximum of two nums.

Prog:

```
num1 = int(input("Enter num 1:"))
num2 = int(input("Enter num 2:"))
if num1 > num2:
    print(num1, " is greater")
else:
    print(num2, " is greater")
```

if-elif-else

the if-elif-else statement is used to conditionally execute a statement or block of code.

Syntax:

```
if condition1:
    statement1
elif condition2:
    statement 2
else:
    statement
```

Ex:

```
a = 15
b = 12

if a == b:
    print("Both are equal")
elif a > b:
    print(a, " is greater")
else:
    print(b, " is greater")
```

If the first condition is true, then if-block will be executed and remaining blocks will be eliminated or if elif-condition is true, then elif-block will be executed other blocks will be eliminated or if the two conditions are false, then the else-block will be executed.

Biggest among three numbers

```
a = int(input("Enter a-number :"))
b = int(input("Enter b-number :"))
c = int(input("Enter c-number :"))

if a>b and a>c:
    print(a, " is the biggest")
elif b>c:
    print(b, " is the biggest")
else:
    print(c, " is the biggest").
```

Nested if :

In python, the nested if else statement by placing if statement inside another if-else statement. It is allowed in python to put any no. of if statements in another if statement.

Syntax :

```
if condition_outer:
    if condition_inner:
        statement of inner if
    else:
        statement of inner else
    statement of outer if
else:
    statement of outer else
```

```
a = int(input("Enter a number:"))
b = int(input("Enter b number:"))
c = int(input("Enter c number:"))

if a>b:
    if a>c:
        print(a,"is the biggest")
    else:
        print(c,"is the biggest")
elif b>c:
    print(b,"is the biggest")
else:
    print(c,"is the biggest")
```

Iterative or Repetition or Looping statements.

A Repetition statement is used to repeat a group of programming instructions. In python, we generally have two loops.

(i) for loop (ii) while loop

for loop:

for loop is used to iterate over a sequence that is list, string, tuple, dictionary, set. we can execute a set of statements once for each item in a list, tuple, or dictionary.

Syntax:

```
for variable in sequence:
    # code to be executed
```

a temporary variable that takes the value of the item inside a sequence on each iteration, loop continues until we reach the last item in the sequence.

Eg:

```
fruits = ["apple", "banana", "coconut", "orange"]
for fruit in fruits:
    print(fruit)
```