Operations:

It is an in-built task performed on operands(values) based on the operator used.

Operand:

They are the values (or variable holding value) involved in an operation

Operator:

They are special symbols or keywords which has got in-built functionalities

There are totally **7 types** of operators:

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

1. Arithmetic op:

- returns a numeric value

- +, -, *, //(Floor division), /(Float division), %(modulus),**(exponentiation)

+:

-When used with 2 or more numeric values, performs addition operation

-When used with a single numeric operand, represents "positive integer"

eg:

```
print(6 + 12) # addition
print(3 + 4 + 8 + 9)#addition
print(+3)#positive int
```

-:

-When used with 2 or more numeric values, performs subtraction operation

-When used with a single numeric operand, represents "negative integer"

eg:

```
print(6 - 12) # subtraction
print(3 - 4 - 8 - 9)#subtraction
print(-3)# negative int
```

*****:

- When used with atleast 2 numeric operands performs multiplication

eg:

```
print(10*33)#multiplication
print(2*3*8)#multiplication
print(*3)#TypeError
```

** [Exponentiation]:

- it helps to return the exponent value of the given base and power

- **Syn:**

```
base ** power
```

- **eg:**

```
print(3 ** 2) #9
```

// [Floor Division]:

- In a division operation, when a "non decimal quotient" is expected as the output

- eg:

```
print(11 // 2)   # 5
print(22 // 7)      # 3
```

/ [Float Division]:

- In a division operation, when a "decimal quotient" is expected
as the output

- **eg**:

```
print(11 / 2)   # 5.5
print(22 / 7)       # 3.142857142857143
```

% [Modulus]:

- In a division operation, when a "remainder" is expected
as the output

- eg:

```
print(11 % 2)   # 1
print(22 % 7)      # 1
```

2. Comparison / Relational Op:

- It returns only a Boolean value as the O/P

- **>, <, >= , <=, ==, !=**

Ex:

```
print(10 < 12)#True
print(5 > 6)#False
print(22.67 >= 10.67778888)# True
print((10*3) <= (15**2))#True
print(True == True)#True
print(False != (2**0))#False
print("ap" == "aa")#False
print(97 == "97")#False
```

[Note:

boolean conditions:

a. boolean values

b. expressions of comparison operators

c. variable holding boolean values]

3. Logical Op:

- It returns only a boolean value as the O/P

- It helps to combine and checks multiple boolean conditions together

- **and, or, not**

logical and:

Condition 1	Condition 2	Output
True	True	True
True	False	False
False	-	False

logical or:

Condition 1	Condition 2	Output
False	True	True
True	-	True
False	False	False

logical not:

- It requires only one operand to perform the operation

- It returns the inverted bit of the actual O/P bit

Input	Output
True	False
False	True

4. Assignment OP: (=)

- It helps to store a value present in RHS to the named memory location present in LHS

Variables:

- It is a container which holds a value
- It is a named memory location which can hold "single valued data"
- At a time only one value can be stored in a variable
- If we try to store multiple values one after the other then variable will only hold the latest updated value
- It is compulsory to assign a value to a variable before utilization

2 varieties in assignment op:

1. Arithmetic Op + Assignment Op:

+=, -=, *=, **=, //=, /=, %=

2. Bitwise Op + Assignment Op:

&=, |=, ^=, <<= , >>=

[Note:

- The combined form of (Arithmetic Op + Assignment Op) or (Bitwise Op + Assignment Op) are called as **compound statements**

- **Rules to remember while using compound statements:**

1. They are used only when the operand used in RHS(in operation) and the operand used in LHS(for assignment) is same

2. At least 2 operands are required to use them

3. It is compulsory to initialize the operand which will be used in compound statements

- **"~="** is not supported as the complement operator(~) requires only

one operand to perform operation**]**

<u>4. Bitwise Op:</u>

& --> Bitwise AND

| --> Bitwise OR

^ --> Bitwise XOR

~ --> Bitwise Complement (requires only one operand)

1 --> ~1 ==> 0

0 --> ~0 ==> 1

<< --> Bitwise Left Shift

>> --> Bitwise Right Shift

<u>6. Identity OP: ["is", is not]:</u>

- returns boolean value

- It helps to compare and return a boolean
  value if the variable id's are same, when used "is"

- "is not" --> vice versa

<u>7. Membership Op:["in", not in]</u>

- It is allowed to be used only on iterable obj(The object which stores multiple individual values into a shared memory.

eg: string, list, dictionary, tuples, set)

- It helps to check whether the defined value is available among the iterable obj or not by returning a boolean value

- If used with variables, then throws errors

===

Control Flow Statements:

- It controls the execution flow of a program
- 2 types:
    1. Conditional Statements:
        - simple if, if-else, elif
    2. Looping Statements:
        - **a. for loop**
            - **i.with range()**
                - 2 varieties:
                    - incrementing loop
                    - decrementing loop
            - **ii.with iterable objects**
        - **b. while loop**
            - 2 varieties:
                - incrementing loop
                - decrementing loop

Solve the below tasks:

Scenario 1: Texting msg to your BF/GF

user I/P: ask user the what's app "online"/"offline"

"online:" && and the tick status:

double tick grey--> Ignoring your msg

double tick blue--> seen your msg

"offline"

single tick --> Msg sent

you will call

Scenario 2: Banking Transaction

Write a program to simulate a withdrawal from a bank account:

Input: Current balance and withdrawal amount

Conditions:

If the withdrawal amount is less than or equal to the balance, allow the transaction and display the remaining balance.

If the withdrawal amount exceeds the balance, display "Insufficient funds."

Scenario 3:Write a program to check the type of triangle based on the sides provided.

Input: Lengths of three sides of a triangle

Conditions:

If all sides are equal, it's an Equilateral triangle.

If two sides are equal, it's an Isosceles triangle.

If no sides are equal, it's a Scalene triangle.

Scenario 4: Travel Booking with Discounts

A travel agency offers discounts based on destination and mode of transport:

Input: destination type and type of ticket

Conditions:

If the destination is international:

If flying first class, give a 15% discount.

If economy, give a 10% discount.

If the destination is domestic:

No discount for economy.

5% discount for first class.

Scenario 5: Online Shopping Delivery Time Estimation

A program that estimates delivery time based on location:

Input: Location type (1 for Local, 2 for Regional, 3 for National, 4 for International)

Conditions:

for local ➔ 1 day

for regional ➔ 2 days

for national ➔ 4 days

for international ➔ 1 week

Scenario 6: Age-Based Ticket Pricing

A theme park charges ticket prices based on age and group discounts:

Conditions:

If the person is a child (age < 12):

If in a group, ticket is free.

Otherwise, charge $10.

If the person is an adult:

If in a group, apply a 20% discount on $20.

Otherwise, charge $20.

==

range(start_val, end_val, step) :

- It helps to set a sequence of values based on the mentioned arguments.

start_val: defines the starting value of the sequence range

default start_val : 0

end_val: defines the ending value of the sequence range

- It is compulsory to define the end range

step: defines the difference between the future and current sequence value

default step : +1(should be +ve integer)

ex:

```
val = range(1, 5)
print(val) #O/P: range(1, 5)==> bcz val is a
                             #variable which can hold only one
                             #value at a time and range() returns
                             #a sequence of values
```

2. Looping Stmts:

a. for loop:

i. **with range() function:**

Syn: `for var_name in range(start_val, end_val, step)`

Note: The mentioned end_val will not be considered while setting up the sequence values

2 varieties:

i. **incrementing for loop:**

1. The start_val < end_val
2. If the programmer does not mention the step, then it takes the default step
3. The loop will work until the sequence value is less than the end_val

==

ii. **decrementing for loop:**

1. The start_val > end_val
2. If the programmer does not mention the step, then it takes the default step. Therefore it is compulsory to include a negative decrementing step.
3. The loop will work until the sequence value is greater than the end_val

==

Note:

-If the start_val == end_val then the loop will not be executed

-In a incrementing loop if start_val > end_val then the loop will not be executed

-In a decrementing loop if start_val < end_val then the loop will not be executed

[**Note**:

- In a **incrementing loop** if the mentioned end values also has to be executed then **"actual end + 1"**

- In a **decrementing loop** if the mentioned end values also has to be executed then **"actual end - 1"]**

ii**. with iterable objects:**

Syn:

```
for var_name in iterable_obj_name:
        #logic
#remaining lines of code
```

Here, the var_name will store the elements of the mentioned object.

WAP to print "Hello World" until and unless the user do not enter 5.

1 HW

100 HW

55 HW

5 --> Prg exec

```
i = 5
while i >= 10: # 5 >= 10 --> Terminates the loop, bcz it never starts
    print(i)
    i -= 1
```

```
i = 5
while i <= 10: # Gets into an infinite loop
    print(i)
    i -= 1
```

```
i = 5 ; 5 <= 10 --> T
i = 4 ; 4 <= 10 --> T
i = 3 ; 3 <= 10.... "i" will never reach the end condition
```

Predict the O/P's:

1.

```
        n = 4
        for i in range(n+1, 1-1, -1):
                print(i)#5 4 3 2 1
```

2.

```
        n = 3
        for i in range(n, 1+2):
                print(i)#does not execute as start == end
```

3.

```
        n = 10
```

```
for i in range(1, n + 1, -1):
    print(i)
```

- does not execute, As it is a decrementing loop (step = -1)

therefore PVM is expecting start > end here, 1 > 11 --> False

Tasks:

WAL to print all the alternative values of a series of natural numbers
with a customised end value

I/P: n = 7

O/P: 1 3 5 7

WAL to print all the multiples of 5 of a series of natural numbers
with a customised value in a decrementing order

I/P: n = 10

O/P: 10 5

I/P: n = 8

O/P: 5

#WAP to display the natural numbers till the defined "n" value

n = 8

possible ans,

start = 1

1 2 3 4 5 6 7 8

start = 3

```
3 4 5 6 7 8


start = 2
2 3 4 5 6 7 8


start = 8
8
```

Functions:

- It is a block of code which performs a specific task

eg: print() --> to print if any value passed, and to move the control
to next line
input() --> get the input from user during execution time

Syn:

```
def function_name(parameters):
        #logic
        return #or return value
```

- Functions will be executed only when it is called.

Parts of a function:

1. func_name : it is defined based on the task to be performed

2. parameters[optional]: They are the I/P variables required by a function to perform some specific task

3. logic : the task to be performed

4. return [optional]:

- It majorily helps return back the execution flow from called function back to function call

- It helps to return value / values i.e., O/P from a called function back to the function call, which has to be received by a variable at function call.

- It is last the executable line of a function

5. function call: A function won't be executed until and unless

it is not called

6. arguments: The value or a variable holding val provided in a

function call that is required by the parameters of a function