Strings properties:

↳ It is a sequence of characters

↳ Use index to access characters in a string

↳ If we try to access index out of range or use decimal numbers we will get an error.

↳ Supports both +ve & -ve index.

↳ These are Immutable.

↳ Elements can't be changed once it has been assigned.

↳ We can simply reassign different strings to the same name.

↳ Supports indexing as well as slicing.

list Properties:

↳ It is a sequence data structure.

↳ Used to store collection of items.

↳ Order is preserved in a list.

↳ Mutable.

↳ Indexing and slicing are supported.

↳ Dynamic i.e., increase or decrease in siz

Tuples properties:

↳ Similar to list but Immutable.

↳ It is a sequence data type.

↳ Supports Indexing and slicing.

Set Properties:

↳ Unordered collection of items. i.e., they can't be indexed.

↳ No duplicates are allowed.

↳ Mutable.

↳ Used to perform mathematical operation like union, intersection etc.

Dictionary Properties:

↳ Unordered collection of items i.e., can't be indexed.

↳ Other compound data types have only valu

as an element where as a dictionary has a KEY : VALUE pair.

↳ Keys are unique, values may not be unique.

↳ Values can be of any type, but Keys must be immutable.

DECISION - CONTROL - STATEMENTS :

Indentation :

→ Python relies on indentation (whitespaces at the beginning of a line or a tab space) to define scope in the code.

IF :

The 'if' condition is considered, the simplest of the three and makes a decision based on whether the condition is True or Not.

↳ If the condition is true, it prints out the intended expression.

↳ If the condition is false, it skips printing

the intended expression.

```
*I/p - a = 10
       b = 10
       if (a == b):
           print ('yes', a == b)
           Print ('HI')
           Print ('welcome')
       Print ('exit')
```

O/p - 'yes', a == b
'HI'
'Welcome'
'exit'

```
*I/p - num = None
       if (num):
           print ('Number is positive')
       Print ('this line will print always')
```

O/p - This line will print always.

NOTE : '0', NONE → FALSE
1 → TRUE

* I/p - num = -1

```
if(num >= 0):
    Print('Number is positive')
else:
    Print('Number is negative')

Print('This line will print always')
```

O/p - Number is negative
this line will print always.

IF-ELSE:

The 'if-else' statement evaluates test expression and will execute the body of 'if' only when the test condition is 'True'.

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

ELIF:

If the previous conditions were not true, then this condition is used.

* I/p - num = 0

```
if(num > 0):
    Print('positive')
elif(num == 0):
    Print('zero')
```

```
else:
    Print('Negative')
```

o/p - zero.

*i/p - num = 20

```
if(num >0):
    Print('positive')
elif(num <=0):
    Print('zero')
else:
    Print('negative')
```

o/p - positive.

*i/p - num = -10

```
if(num >0):
    Print('positive')
elif(num>=0):
    Print('zero')
else:
    Print('negative')
```

o/p - Negative.

NESTED IF-ELSE:

where there is an if statement (or) if-else (or) if-elif-else) is present inside another if statement (or) if-else or if-elif-else) then it this is called as nesting of control statements.

```
* I/P - num = 0
        if num >= 0:
            if (num == 0):
                Print('Zero')
            else:
                Print('positive')
        else:
            Print('negative')
```

O/P - Zero.

WHILE LOOP:

we can execute a set of statements as long as a condition is true.

* I/p -
```
count = 0
while (count < 3):
    Print('the count is : ', count)
    count = count + 1
Print('Good bye! count is : ', count)
```

O/p -
```
The count is : 0
The count is : 1
The count is : 2
Good bye! count is : 3
```

* I/p -
```
tup = (1, 2, 3, 4, 5)
prod = 1
index = 0
while (index < len(tup)):
    Prod* = tup[index]
    index + = 1
Print('product = {}, Index = {}'.format
                                (prod, index))
```

O/p - product = 120, Index = 5

Sol:

	Prod.	Index.
0 < 5	1	@1
1 < 5	2	2
2 < 5	6	3
3 < 5	24	4
4 < 5	120	120

WHILE-ELSE:

We can run a block of code once when the condition is no longer is true.

* I/p - tup = (1, 2, 3, 4, 5)

```
Prod = 1
index = 0
while (index < len(tup)):
    Prod* = tup[index]
    index + = 1
else:
    Print ('No more numbers in tuple')
Print('product = {}', format(prod))
```

O/p - No more numbers in tuple.

Product = 120.

MEMBERSHIP OPERATORS:

Used to find whether a value or variable is found in a sequence.

"in and not in" are the membership operators.

↳ It works on sets and dictionary.

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 ↳↳ sequence data structure.

↓

list / string / tuple / Range.

* I/P - lst = [1, 2, 3, 4]
  Print (5 in list)
  O/P - False.

* I/P - lst = [1, 2, 3]
  Print (2 in lst)
  O/P - True.

* I/P - dict = {1 : 'a', 2 : 'b'}
  Print ('a' in dict)
  O/P - False.

* I/P - dict = {1 : 'a', 2 : 'b'}
  Print ('1' in dict)
  O/P - True.

*I/p - s = 'Welcome'
Print('o' in dict)

O/p - True.

FOR LOOP:

A 'for' loop is used for iterating over a sequence (that is either a list / tuple / set / dictionary / string)

*I/p - tup(10, 20, 30, 40)
for x in tup:
Print(x)
Print('Finish')

O/p - 10
20
30
40
Finish.

RANGE:

It returns a sequence of numbers, starting from '0' by default and increments by 1 (by default), and stops before a specified number

SYNTAX: (start, stop, step)

Start : Optional, An integer number specifying at which position to start. Default is '0'.

Stop : Required, An integer number specifying at which position to stop (not included).

Step : Optional, An integer number specifying the incrementation. Default is '1'.

* I/p - for count in range(1,10,2):
    Print ('the count is :', count)
Print('Good Bye')

O/p - the count is : 1
the count is : 3
the count is : 5
the count is : 7
the count is : 9

Good Bye

*I/P - for count in range (1, 9, 2):

    Print ('The count is : ', count)

Print ('Good')

O/P - The count is : 1

The count is : 3

The count is : 5

The count is : 7

Good