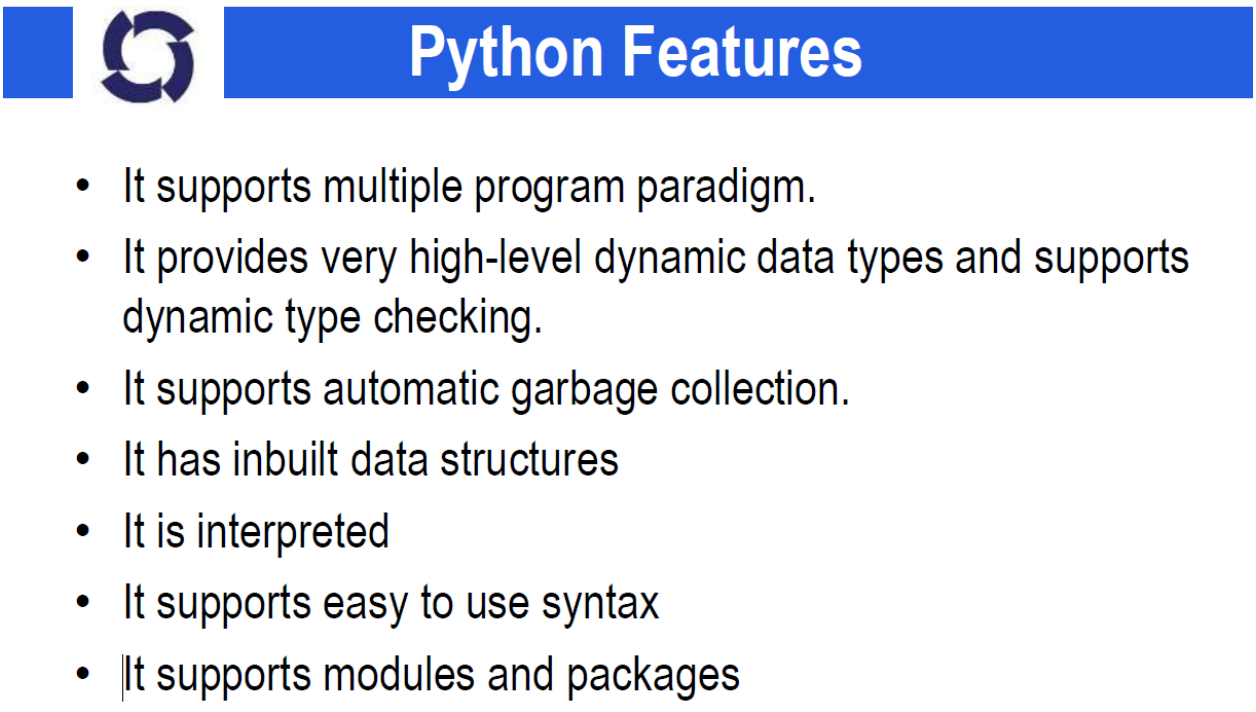
One/Two Marks Questions:

1. What is the difference between a comment and the docstring?

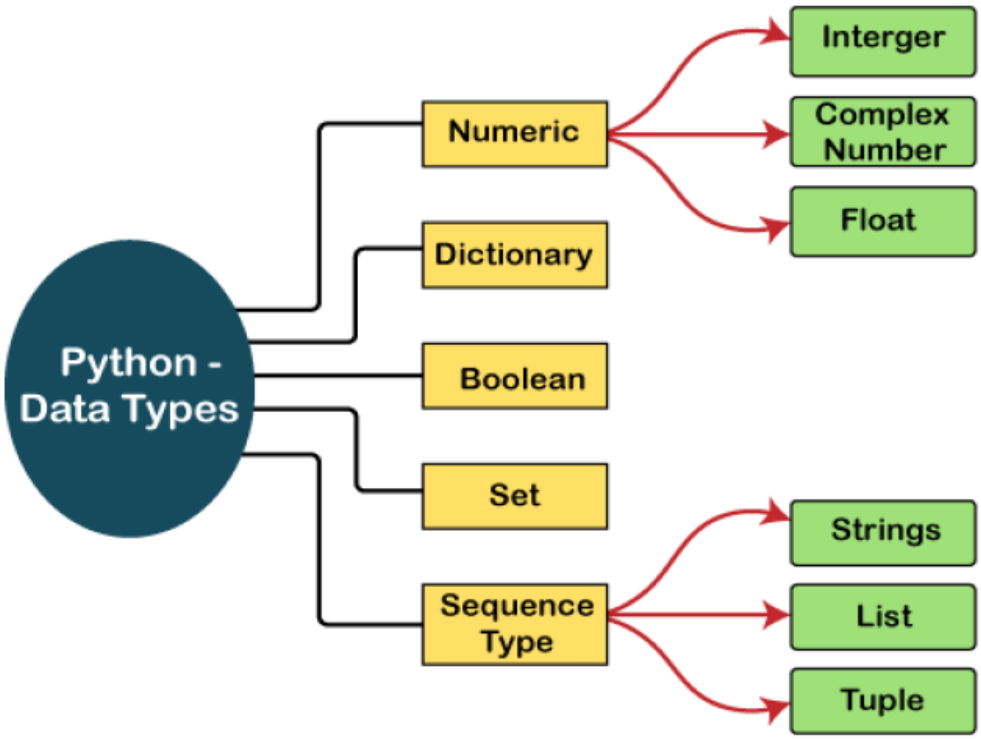
Comments	Docstrings
Comments explain code in simple language.	Docstrings describes what the code does.
In Python, we can write comments using the hash (#) character.	We can write docstrings as string literals using the triple quotes (‘’’).
Example: `#This is a comment`	Example: `'''This is a docstring'''`
There is no restriction on where to write a comment in Python.	We can write docstrings only for modules, classes, functions, and methods in Python.
Comments cannot be accessed from the Python shell.	We can access docstring using either the `__doc__` attribute or `help()` function.

2. What are the features of Python?



3. List Standard data types in Python

ı provides various standard data types that define the storage method



Standard Data Types

- Numbers (int, float, complex)
- String
- Boolean
- List
- Tuple
- Dictionary

4. Give example of Multiline Comment in Python

Comments

- Multiline comments

- Example:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

- Example:

```
'''
This is a comment
written in
more than just one line
'''
print("Hello, World!")
```

5. List various Bitwise operators in Python

- The following Bitwise operators are supported by Python language.

Operator	Description	Example
& Binary AND	Operator copies a bit to the result, if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit, if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit, if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operand's value is moved left by the number of bits specified by the right operand.	a << = 240 (means 1111 0000)
>> Binary Right Shift	The left operand's value is moved right by the number of bits specified by the right operand.	a >> = 15 (means 0000 1111)

6. What is Unicode String?

Normal strings in Python are stored internally as 8-bit ASCII, while Unicode strings are stored as 16-bit Unicode. This allows for a more varied set of characters, including special characters from most languages in the world. I'll restrict my treatment of Unicode strings to the following −

Example

Live Demo

```
#!/usr/bin/python
print u'Hello, world!'
```

Output

When the above code is executed, it produces the following result −

```
Hello, world!
```

As you can see, Unicode strings use the prefix u, just as raw strings use the prefix r.

7. What is string and Raw String in Python?

- Text is represented in programs by the *string* data type.
- A string is a sequence of characters enclosed within quotation marks (") or apostrophes (').
- Strings are amongst the most popular types in Python.
- We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable.
- For example
- var1= 'Hello World!'
- var2 = "Python Programming“

```
>>> str1="Hello"
>>> str2='spam'
>>> print(str1, str2)
Hello spam
>>> type(str1)
<class 'str'>
```

r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r'\n' prints \n and print R'\n'prints \n

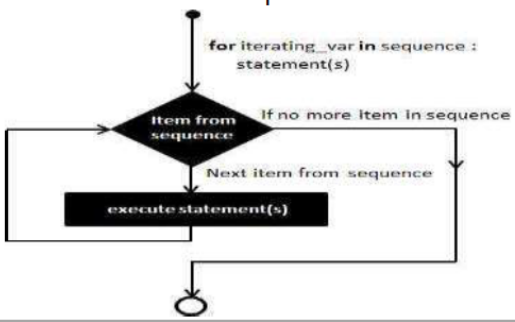
8. Write the syntax of for loop in Python?

for Loop Statements

- The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- **Syntax**

```
for iterating_var in sequence:
    statements(s)
```

- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating_var. Next, the statements block is* executed. Each item in the list is assigned to *iterating_var, and the statement(s) block is* executed until the entire sequence is exhausted.



9. What is Pass Statement?

Pass Statement

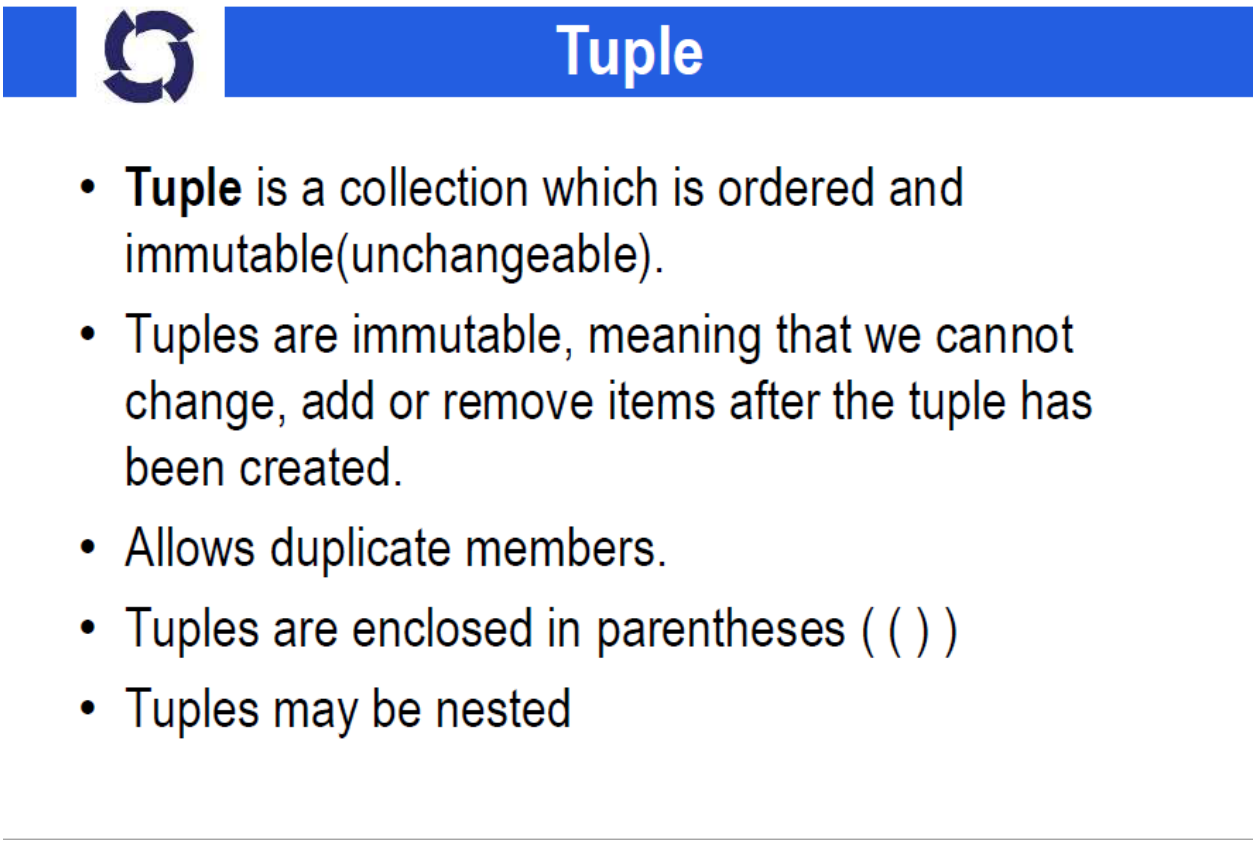
- As the name suggests pass statement simply **does nothing**.
- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- It is like null operation, as nothing will happen is it is executed.
- Pass statement can also be used for writing empty loops. Pass is also used for empty control statement, function and classes.
- **Example:**

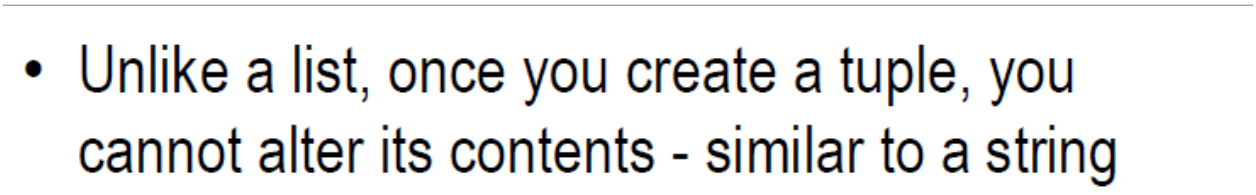
```
a = 20
if(a%2!=0):
   pass
else:
   a=a+10;
   print(a)
```

10. Write the syntax of find () method?

4	**find(str, beg=0 end=len(string))** Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
5	**index(str, beg=0, end=len(string))** Same as find(), but raises an exception if str not found.

11. What is tuple? What is the difference between list and tuple?





12. Which method is used to update lists in Python?

append()	Adds an element at the end of the list
extend()	Add the elements of a list (or any iterable), to the end of the current list
insert()	Adds an element at the specified position

13. List some few common Exception type.

14. What is docstring?

Python Docstrings

- Python documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods.
- It's specified in source code that is used, like a comment, to document a specific segment of code.
- Unlike conventional source code comments, the docstring should describe what the function does, not how.

Python Docstrings

- **What should a docstring look like?**
- The doc string line should begin with a capital letter and end with a period.
- The first line should be a short description.
- If there are more lines in the documentation string, the second line should be blank, visually separating the summary from the rest of the description.
- The following lines should be one or more paragraphs describing the object's calling conventions, its side effects, etc.
- **Declaring Docstrings:** The docstrings are declared using '''triple single quotes''' or """triple double quotes""" just below the class, method or function declaration. All functions should have a docstring.
- **Accessing Docstrings:** The docstrings can be accessed using the __doc__ method of the object or using the help function.

15. How will you create sets in Python?

- A set is a collection which is unordered and unindexed.
- Set items are immutable (unchangeable)
- Set do not allow duplicate values.
- Once a set is created, you cannot change its items, but you can add new items.
- Set items can be of any data type
- In Python sets are written with curly brackets. {}

- Example:

```
colors = {"Red", "Green", "Blue"}
print(colors)
#Output:
#{"Red", "Green", "Blue"}
```


16. Write the definition of class method.

What is a class method?

A class method is a method that is bound to a class rather than its object. It doesn't require creation of a class instance, much like staticmethod.

The difference between a static method and a class method is:

- Static method knows nothing about the class and just deals with the parameters
- Class method works with the class since its parameter is always the class itself.

The class method can be called both by the class and its object.

```
Class.classmethod()
Or even
Class().classmethod()
```

But no matter what, the class method is always attached to a class with the first argument as the class itself `cls`.

```
def classMethod(cls, args...)
```

17. What is __init__method?

What is __init__ in Python?

The Default __init__ Constructor in C++ and Java. Constructors are used to initializing the object's state. The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

Example:

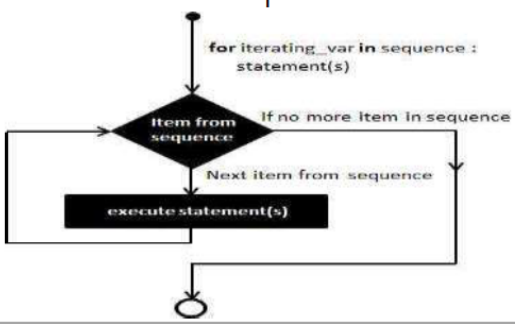
18. Write a syntax of ''for loop'' in Python.

for Loop Statements

- The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- **Syntax**

  for iterating_var in sequence:

  statements(s)
- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating_var. Next, the statements block is* executed. Each item in the list is assigned to *iterating_var, and the statement(s) block is* executed until the entire sequence is exhausted.



19. Give the characteristics of membership operator?

Operator	Description	Example
in	Evaluates to true, if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true, if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

20. Demonstrate self in Python?

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

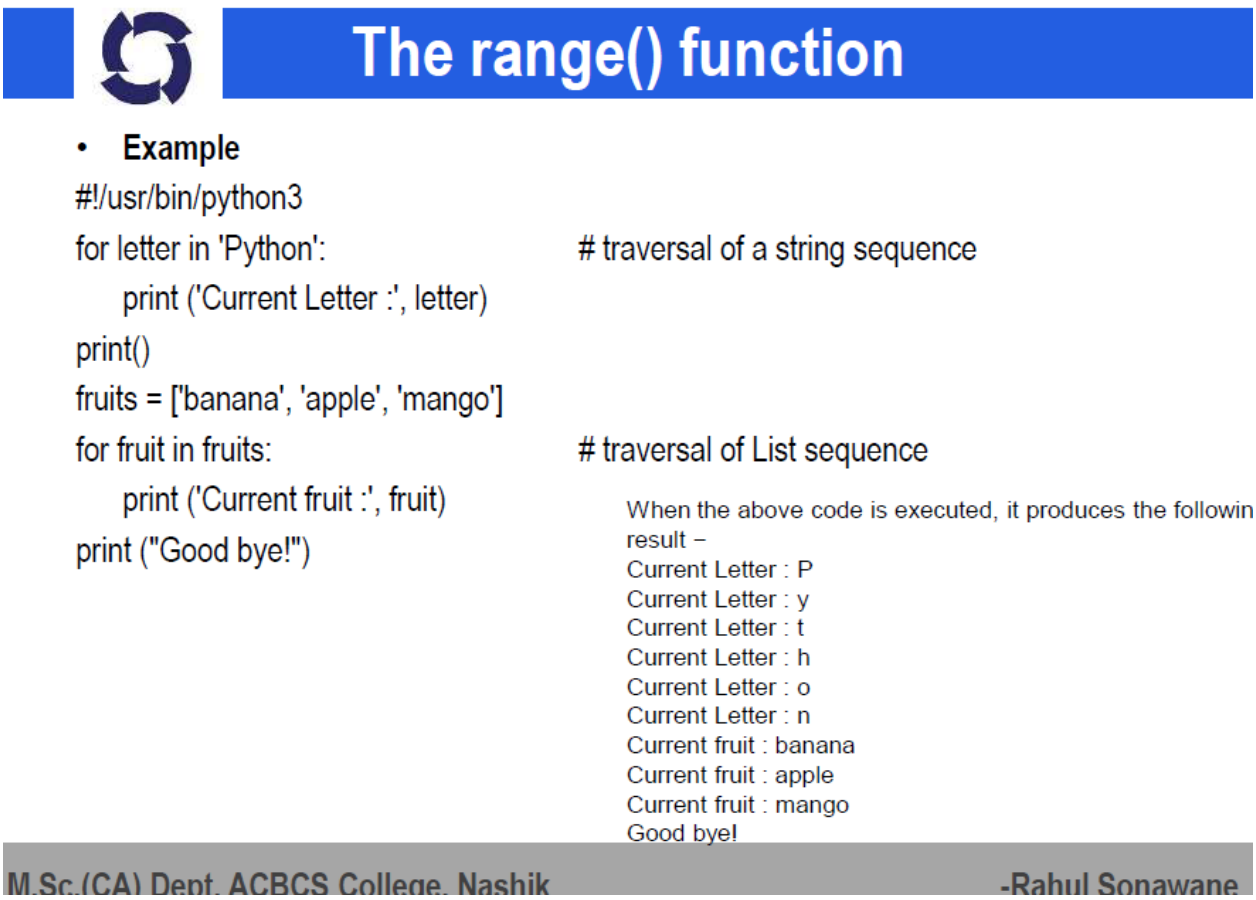
```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Try it Yourself »

21. Explain what is range() function and how it is used in lists?



22. What is the output of print tuple[1:3] if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

O/P: (786, 2.23)

23. What are the built-in functions that are used in Tuple?

Tuple functions

- 1. len - Gives the total length of the tuple.
- 2. max - Returns item from the tuple with max value.
- 3. min - Returns item from the tuple with min value.
- 4. tuple - Converts a list into tuple.

24. What is tuple? What is the difference between list and tuple?.....*repeated question*

25. How does del operation work on dictionaries? Give an example.

Dictionary-Mutable

The del keyword removes the item with the specified key name

```
myDict = {
  "Rno": "1",
  "Name": "Meera",
  "marks": 87,
  "Class": "TY"
}
>>> del myDict['Class']
>>> print(myDict)
{'Rno': 1, 'Name': 'Meera', 'Marks': 87}
```

The del keyword can also delete the dictionary completely

```
myDict = {
  "Rno": "1",
  "Name": "Meera",
  "marks": 87,
  "Class": "TY"
}
>>> del myDict
>>> print(myDict)
NameError: name 'myDict' is not defined
```

26. What is the output of the following statements?

```
l= [1, 2, 3,4]
print (l [0: 2])
```

O/P: [1, 2]

Five Marks Questions:

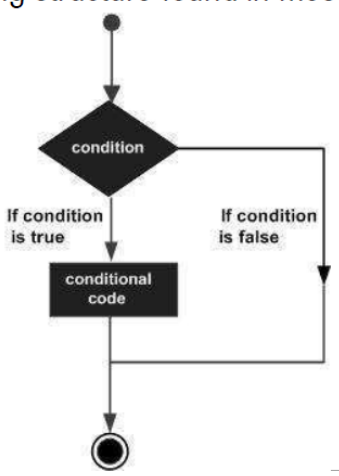
1. Explain Different Conditional Statements In Python?

- There comes situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arises in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.
- Decision making statements in programming languages decides the direction of flow of program execution. Decision making statements available in python are:

1. if
2. if…else
3. nested if…else
4. if else ladder
5. Short hand if statement
6. short hand if…else statement

Conditional Statements in Python

- Decision-making is the anticipation of conditions occurring during the execution of aprogram and specified actions taken according to the conditions.
- Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.
- Following is the general form of a typical decision making structure found in most of the programming languages-



- Python programming language assumes any

non-zero and non-null values as TRUE, and

any **zero or null values as FALSE value.**

If statement

- if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.
- **Syntax :**
- **if *condition*:**

  # Statements to execute if condition is true

- Here, condition after evaluation will be either true or false.
- if statement accepts Boolean values – if the value is true then it will execute the block of statements below it otherwise not.
- We can use *condition* with bracket '(' ')' also.

If statement

- **Example:**

```
if condition:
        statement1
 statement2
# Here if the condition is true, if block will consider only
    statement1 to be inside its block.
```

- **# python program to illustrate If statement**

```
i = 10
if (i > 15):
   print ("10 is less than 15")
print (“Outside If”)
```

If statement

- **Example:**

```
#!/usr/bin/python3
var1 = 100
if var1:
    print ("1 - Got a true expression value")
    print (var1)
var2 = 0
if var2:
    print ("2 - Got a true expression value")
    print (var2)
print ("Good bye!")
```

When the above code is executed, it produces the following result −

```
1 - Got a true expression value
100
Good bye!
```

IF...ELIF...ELSE Statements

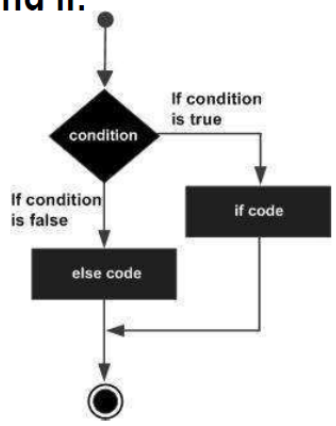
- An **else statement can be combined with an if statement.**
- **An else statement contains a** block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.
- The else statement is an optional statement and there could be at the most only one **else statement followinq if.**
- **Syntax**

If expression:

    statement(s)

else:

    statement(s)



IF...ELIF...ELSE Statements

Example

```
#!/usr/bin/python3
amount=int(input("Enter amount: "))
if amount<1000:
      discount=amount*0.05
      print ("Discount",discount)
else:
      discount=amount*0.10
      print ("Discount",discount)
print ("Net payable:",amount-discount)
```

OUTPUT

Enter amount: 600
Discount 30.0
Net payable: 570.0
Enter amount: 1200
Discount 120.0
Net payable: 1080.0

The elif Statement

- The **elif statement allows you to check multiple expressions for TRUE and execute a block** of code as soon as one of the conditions evaluates to TRUE.
- Similar to the **else, the elif statement is optional. However, unlike else, for which there** can be at the most one statement, there can be an arbitrary number of **elif statements** following an **if.**

- **Syntax**

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

Core Python does not provide switch or case statements as in other languages, but we can use if..elif...statements to simulate switch case

The elif Statement

- **Example**

```
#!/usr/bin/python3
amount=int(input("Enter amount: "))
if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
elif amount<5000:
    discount=amount*0.10
    print ("Discount",discount)
else:
    discount=amount*0.15
    print ("Discount",discount)
print ("Net payable:",amount-discount)
```

OUTPUT

```
Enter amount: 600
Discount 30.0
Net payable: 570.0
Enter amount: 3000
Discount 300.0
Net payable: 2700.0
Enter amount: 6000
Discount 900.0
Net payable: 5100.0
```

Single Statement Suites

- If the suite of an **if clause consists only of a single line, it may go on the same line as the** header statement.
- Here is an example of a **one-line if clause-**

```
#!/usr/bin/python3
var = 100
if ( var == 100 ) : print ("Value of expression is 100")
print ("Good bye!")
```

When the above code is executed, it produces the following result-

```
Value of expression is 100
Good bye!
```

2. Write a short note on string formatting operator.

String Formatting Operator

- One of Python's coolest features is the string format operator %.
- This operator is unique to strings and makes up for the pack of having functions from C's printf() family.
- Following is a simple example −

```
print ("My name is %s and weight is %d kg!" % ('Zara', 21))
```

- When the above code is executed, it produces the following result −

```
My name is Zara and weight is 21 kg!
```

String Formatting Operator

- Here is the list of complete set of symbols which can be used along with %-

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')

3. Explain Following methods: a. join() b. len() c. index()

13	**join(seq)** Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
14	**len(string)** Returns the length of the string

4	**find(str, beg=0 end=len(string))** Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
5	**index(str, beg=0, end=len(string))** Same as find(), but raises an exception if str not found.

4. Write a python program to check whether the string is or Palindrome

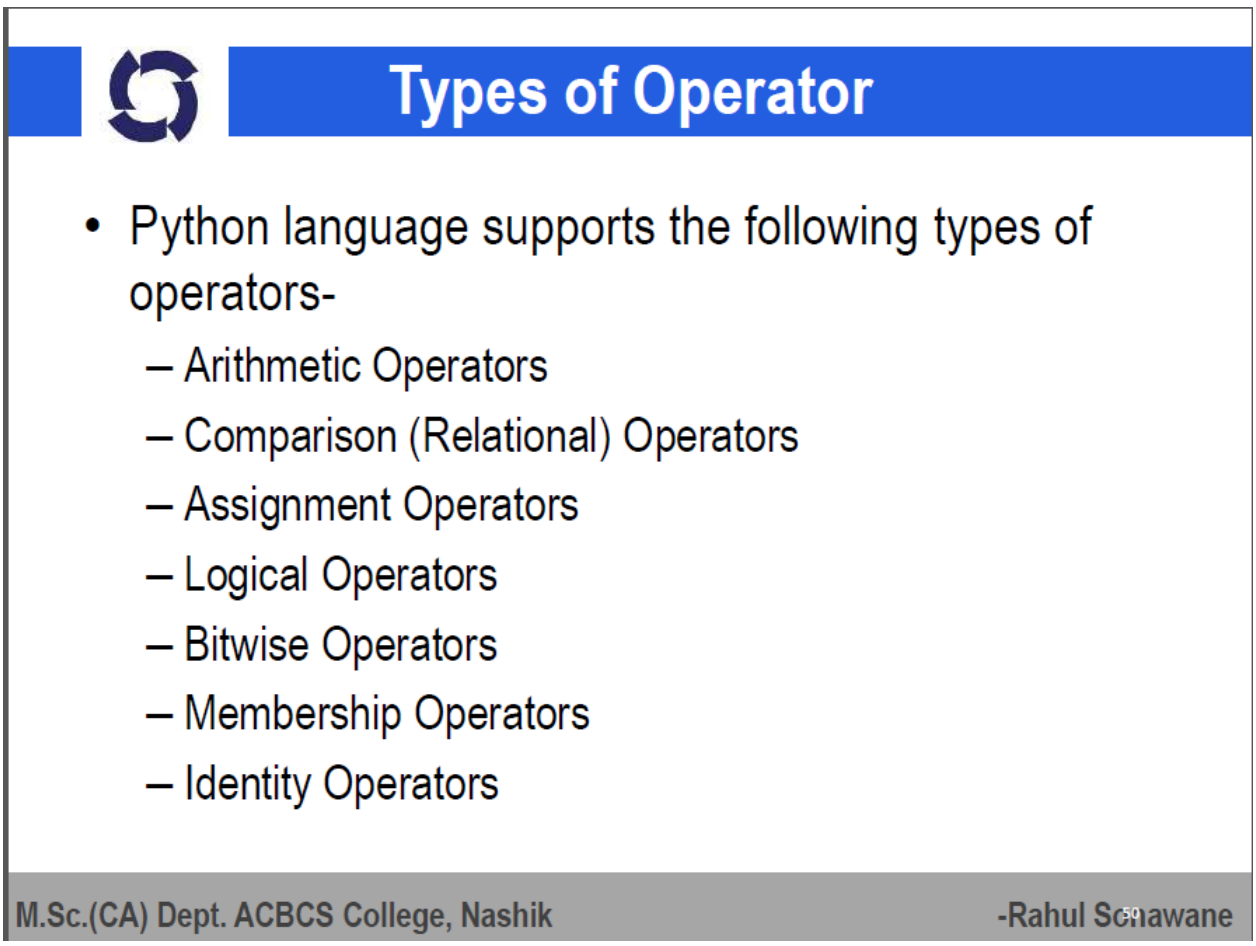
```
str = input("Enter a string: ")

str = str.upper()

rev = str[::-1]

if rev==str: print("It is palindrome")
else: print("it is not palindrome")
```

5. List and explain types of operators in Python.



Python Arithmetic Operators

- Assume variable **a holds the value 10 and variable b holds the value 21, then-**

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 31
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -11
* Multiplication	Multiplies values on either side of the operator	a * b = 210
/ Division	Divides left hand operand by right hand operand	b / a = 2.1
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 1
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20

Python Comparison Operators

- Assume variable **a holds the value 10 and variable b holds the value 20, then-**

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!= b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

- =
- += Add AND
- - = Subtract AND
- *= Multiply AND
- /= Divide AND
- %= Modulus AND
- **= Exponent AND
- //= Floor Division

Python Bitwise Operators

- The following Bitwise operators are supported by Python language.

Operator	Description	Example
& Binary AND	Operator copies a bit to the result, if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit, if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit, if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operand's value is moved left by the number of bits specified by the right operand.	a << = 240 (means 1111 0000)
>> Binary Right Shift	The left operand's value is moved right by the number of bits specified by the right operand.	a >> = 15 (means 0000 1111)

Python Logical Operators

- Assume variable a holds True and variable b holds False then-

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is False.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is True.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is True.

Python Membership Operators

Operator	Description	Example
in	Evaluates to true, if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true, if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Identity Operators

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

6. Explain the term List, Set, Tuple and Dictionary

List

- **List** is a collection which is ordered and mutable.
- Allows duplicate members.
- A List can have elements of different datatypes
- A list contains items separated by commas and enclosed within square brackets ([]).
- Example:

```
myList=[10,34,45,12,67]
Colors= ["Red", "Green", "Blue", "Yellow"]
list1= [123, "Asha"]
```

List

- Example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
print (list) # Prints complete list
#Output:
#['abcd', 786, 2.23, 'john', 70.2]
```


List

- The values stored in a list can be accessed using the slice operator ([ ] and [:])
- Example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
print (list[0]) # Prints first element of the list
#Output:
#abcd
```

List

- Example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
print (list[1:3]) # Prints elements starting from 2nd till 3rd
#Output:
#[786, 2.23]
```


List

- The asterisk (*) is the repetition operator.

- Example

```
mylist = [123, 'Reeta']
print (mylist * 2) # Prints list two times
#Output:
#[123, 'Reeta', 123, 'Reeta']
```

List

- The plus (+) sign is the list concatenation operator

- Example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
mylist = [123, 'Reeta']
print (list + mylist) # Prints concatenated lists
#Output:
#[ 'abcd', 786 , 2.23, 'john', 70.2 , 123, 'Reeta']
```


List

- Lists can be nested

Example:

```
l1=[1,2,[4,4,5]]
 print l1
#Output: [1, 2, [4, 4, 5]]
```

List is mutable

- Can change content without changing the identity.

Example:

```
my_list = [100, 200, 300]
my_list[0] = 40
print(my_list)
```

Output:

```
[40, 200, 300]
```


List Comprehension

- List comprehension is an elegant way to define and create lists
- Syntax:
  new_list = [*expression* for *member* in *iterable*]
  – Example:

```
list= [letter for letter in "human"]
print (list)
#Output:
# ['h', 'u', 'm', 'a', 'n']
```

List Comprehension

- Example: List of all numbers from 0 to 9

```
list= [x for x in range(10)]
print (list)
#Output:
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Dictionary

- Dictionaries are used to store data values in key: value pairs.
- Dictionary is a collection which is **ordered***

*(As of Python version 3.7, dictionaries are ***ordered***. In Python 3.6 and earlier, dictionaries are ***unordered***.)

- Dictionary is **mutable(Changable)** and indexed.
- No duplicate members.
- Dictionaries are written with curly brackets { }


```
>>> myDict={
        "Rno":1,
        "Name":"Meera",
        "Marks":87
        }
>>> print(myDict)
{'Rno': '1', 'Name': 'Meera', 'Marks': 87}
```

Dictionary Accessing Item

You can access the items of a dictionary by referring to its key name, inside square brackets

```
>>> myDict={
        "Rno":"1",
        "Name":"Meera",
        "Marks":87
        }
>>> print(myDict['Rno'])
1
```


Dictionary Accessing Item

There is also a method called get() that will give you the same result:

```
>>> myDict={
        "Rno":"1",
        "Name":"Meera",
        "Marks":87
        }
>>> a=myDict.get('Rno')
>>> print(a)
1
```

The keys() method will return a list of all the keys in the dictionary.

```
>>> myDict={
        "Rno":"1",
        "Name":"Meera",
        "Marks":87
        }
>>> a=myDict.keys()
>>> print(a)
dict_keys(['Rno', 'Name', 'Marks'])
```

The values() method will return a list of all the values in the dictionary.

```
>>> myDict={
        "Rno":"1",
        "Name":"Meera",
        "Marks":87
        }
>>> a=myDict.values()
>>> print(a)
dict_values([‘1', 'Meera', 87])
```

Dictionary-Mutable

- You can change the value of a specific item by referring to its key name
- myDict={
  "Rno":"1",
  "Name":"Meera",
  "Marks":87
  }

```
>>> myDict['Name']='Kabir'
>>> print(myDict)
{'Rno': '1', 'Name': 'Kabir', 'Marks': 87}
```


Dictionary-Mutable

- The update() method will update the dictionary with the items from the given argument. The argument must be a dictionary, or an iterable object with key : value pairs.

```
myDict = {
 "R No": "1",
 "Name": "Meera",
 "Marks": 87
}
>>> myDict.update({'Marks': 97})
>>> print(myDict)
{'Rno': '1', 'Name': 'Kabir', 'Marks': 97}
```

Dictionary-Mutable

- Adding an item to the dictionary is done by using a new index key and assigning a value to it

```
• myDict =        {
 “Rno”: “1",
 "Name": "Meera",
 "marks": 87
}
>>> myDict[‘Class’]=“TY”
>>> print(myDict)
{'Rno': '1', 'Name': 'Kabir', 'Marks': 97, ‘Class’:’TY’}
```

Set

- A set is a collection which is unordered and unindexed.
- Set items are immutable (unchangeable)
- Set do not allow duplicate values.
- Once a set is created, you cannot change its items, but you can add new items.
- Set items can be of any data type
- In Python sets are written with curly brackets. {}

```
• Example:
colors = {"Red", "Green", "Blue"}
print(colors)
#Output:
#{"Red", "Green", "Blue"}
```

Set items

- Set items can be of any data types

```
set1 = {"Pune", "Nashik", "Mumbai"}
set2 = {11, 45, 77,69, 83}
set3 = {False, True, True}
set4 = {"Ram", 80, False, 70, "Male"}
```

The set() Constructor :

```
set5= set(("Pune", "Nashik", "Mumbai"))
type(set5)
<class 'set'>
```


Access Set Items

- Set items cannot accessed by referring to an index or a key.
- But it can be possible through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example:

```
set1 = {"Pune", "Nashik", "Mumbai"}
for  var  in set1:
        print(var)
```

Output –

Pune
Mumbai
Nashik

```
>>>print("Nashik" in set1)
>>>True
```

Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)

Set Methods

Method	Description
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and

FOR MORE INFO REFER TO SIR'S PPT…..

7. Explain the following statements:

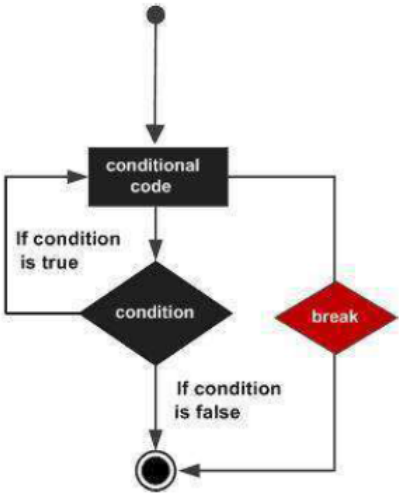
a. If ii. If else iii. Break iv. Continue

Break Statement

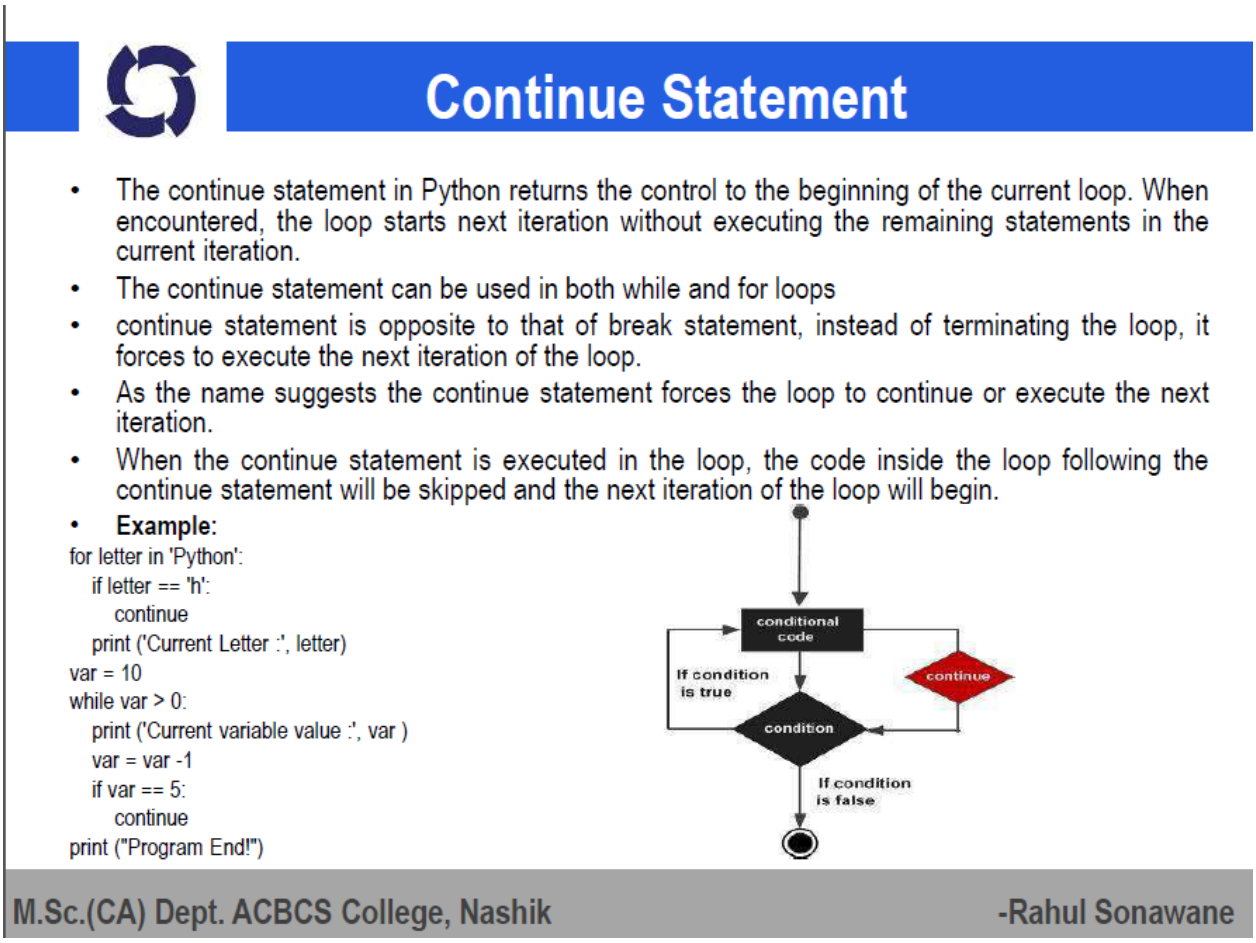
- The break statement is used to terminate the loop or statement in which it is present.
- The most common use of break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.
- If the break statement is present in the nested loop, then it terminates only those loops which contains break statement.

- **Example**:

```
for letter in 'Python':
   if letter == 'h':
      break
   print ('Current Letter :', letter)
var = 10
while var > 0:
   print ('Current variable value :', var )
   var = var -1
   if var == 5:
      break
print ("Program End!")
```



M.Sc.(CA) Dept. ACBCS College, Nashik -Rahul Sonawane



8. Write a python program to reverse words in a given String

```
str = input("Enter a string: ")

split = str.split()[::-1]

li = []

for i in split:
    li.append(i)

print(" ".join(li))
```

9. Write a Python program to remove all duplicates from the given string in Python.

```
str = input("Enter a string: ")

new = ""

for c in str:
    if c not in new:
        new = new +  c

print(new)
```

10. Write a short note on function arguments.

Default Arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
def display(name1, name2="Rohit"):
    print(name1 + "and" + name2 + "are
friends")

display("Ram")
display("Ram", "Shyam")
```

Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- So, the function will receive a *tuple* of arguments, and can access the items accordingly:

```
def my_function(*kids):
    print("The youngest child is " + kids[2])

my_function("Ram", "Rohit", "Mona")
```


Keyword Arguments

- You can also send arguments with the *key* = *value* syntax.
- This way the order of the arguments does not matter.

```
def display(rollno, name, cls):
    print("Roll no " + rollno + " is " + name)

    display(name = " Ram ", rollno = 10, cls = "TYBBA(CA)")
```

Arbitrary Keyword Arguments, **kwargs

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.
- So, the function will receive a *dictionary* of arguments, and can access the items accordingly:

```
def display(**kid):
  print("His first name is " + kid["fname"])

 display(fname = "Ram", lname = "Sharma")
```


Pass by reference vs value

- All parameters (arguments) in the Python language are passed by reference.
- It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

```
def printlist( mylist ):
       mylist.append([1,2,3,4])
        print ("Values inside the function: ", mylist )
mylist = [10,20,30];
printlist( mylist );
print ("Values outside the function: ", mylist)
```

OUTPUT

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

Pass by reference Vs value

- There is one more example where argument is being passed by reference and the reference is being overwritten inside the called function.

```
def printlist( mylist ):
      mylist=[1,2,3,4]
       print ("Values inside the function: ", mylist )
mylist = [10,20,30];
printlist( mylist );
print ("Values outside the function: ", mylist)
```

OUTPUT

```
Values inside the function:  [1, 2, 3, 4]
Values outside the function:  [10, 20, 30]
```

- The parameter *mylist* is local to the function printlist.
- Changing mylist within the function does not affect *mylist*.

11. Write a python program to create tuple with different data types.

```
t = (78, 45.23, True, "Hello", complex(2,3), [243, "hello"], {"Name":"Someone"}, {23, "hhid"})
print(t)
```

12. Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as a argument.

```
def fact(no):
    f=1
    for i in range(2, no+1):
        f *= i
    return f;


no = int(input("Enter a number: "))
print("The factorial is: ", fact(no))
```

13. Write a Python Program to Check Prime Number

```
def prime(no):
    f=1
    for i in range(2, no):
        if no%i == 0:
            f=0
            break
    if(f==0): return "It is not prime"
    else: return "Ir is prime"


no = int(input("Enter a number: "))
print(prime(no))
```

14. Python Program to Convert Celsius To Fahrenheit

```
def convert(t):
    return (t*(9/5))+32;
t = int(input("Enter temperature in celsius: "))
print("The temperature in Fahrenheit is: ", convert(t))
```

15. Write a Python Program to Check Leap Year

```
yr = int(input("Enter temperature in celsius: "))

if yr%400==0 or yr%100!=0 and yr%4==0: print("Leap year")
else: print("Not Leap Year")
```

16. Write a Python Program to Print all Prime Numbers in an Interval

```
def prime(no):
    for i in range(2, no):
        if no%i==0: return False
    return True

ll = int(input("Enter Lower limit(inclusive): "))
ul = int(input("Enter upper limit(inclusive): "))
print ("Prime Numbers: ")
for i in range(ll, ul+1):
    if prime(i): print(i)
```

17. Write a Python Program to Print the Fibonacci sequence

```
li = int(input("Enter number of values of fibonacci: "))

a = 0
b = 1
if li == 1:
    print(a)
elif li==2:
    print(a)
    print(b)
else:
    print(a)
    print(b)
    for i in range(3, li+1):
        c = a + b
        print(c)
        a = b
        b = c
```

18. Write a Python Program to Find Armstrong Number in an Interval

```
def arms(n):
    sum = 0
    no = n
    pow = len(str(no))
    while(no!=0):
        r = int(no%10)
        sum += r**pow
        no /= 10
    if(sum == n): return True
    else: return False


ll = int(input("Enter lower limit (inclusive): "))
ul = int(input("Enter upper limit (exclusive): "))

print("Armstrong numbers in given sequence are: ")
for i in range(ll, ul+1):
    if arms(i): print(i)
```

19. Write a Python Program to Find the Sum of Natural Numbers

```
li = int(input("Enter the limit(inclusive): "))

sum = 0
for i in range(1, li+1):
    sum += i
print("SUM: ", sum)
```

20. Write a Python program to copy the contents of a file to another file.

```
source = open("f.txt", "r")
dest = open("f1.txt", "w")

for line in source:
    dest.write(line)

source.close()
dest.close()
print("FILE COPIED!!!!")
```

21. Write a program that accepts sequence of lines as input and prints the lines after making all characters in the sentence capitalized.

```
str = input("Enter a sequence of lines: ")
print(str.upper())
```