Unit : 1 Review of Class - 11

Introduction to Python Programming Language

Python features:

- **Interpreter based programming language**: Line by line execution of Source code.
- **Free and Open source**: Source code is available free of cost. Free to use for commercial purposes.
- **Portable**: Same code can be used for different machines.
- **Object Oriented Support**: Supports both procedural and OOPs.
- **Extensible**: Python code can be written in other languages.
- **Dynamically typed**: Variable datatype can be decided at runtime.
- **Robust Standard Library**: Extensive standard library available for anyone to use.
- **Easy to code and read**: Simple syntax, indented blocks make it easy to read and code.

Coding modes in python:

- **Interactive mode**: Interactive mode is used when a user wants to run one single line or one block of code. In interactive mode, commands typed at the IDL prompt are executed when the Enter key is pressed.
- **Script mode**: Script mode is where you put a bunch of commands into a file (a script), and then tell Python to run the file. Script mode runs your commands sequentially.

Indentation:

- Indentation refers to the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.

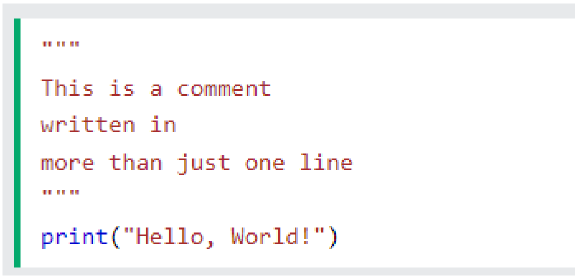
Python Comments:

- Comments are statements in python code that are ignored by the interpreter.
- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Single line comments: These are the statements that start with #

```python
#This is a comment
print("Hello, World!")
```

```python
print("Hello, World!") #This is a comment
```

- Multiline comments: Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:



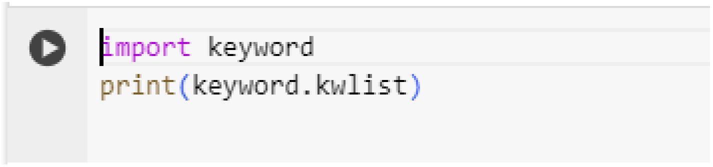
```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Python character set:

- A character set is a set of valid characters acceptable by a programming language in scripting.
- Python supports all ASCII / Unicode characters that include:
  - Alphabets: All capital (A-Z) and small (a-z) alphabets.
  - Digits: All digits from 0-9.
  - Alphabets: All capital (A-Z) and small (a-z) alphabets.
  - Special Symbols: Python supports all kinds of special symbols - " ' | ; : ! ~ @ # $ % ^ ` & * ( ) _ + – = { } [ ] \ .
  - White Spaces: White spaces like tab space, blank space, newline, and carriage return.
  - Other: All ASCII and UNICODE characters are supported by Python that constitutes the Python character set.

Python Tokens:

- A token is the smallest individual unit in a python program.
- All statements and instructions in a program are built with tokens.
- Token Types:
  - **Keywords**: Keywords are reserved by python environment and cannot be used as identifier. There are 35 keywords in python. You may try to use the following code to get a list of keywords supported in your python version.

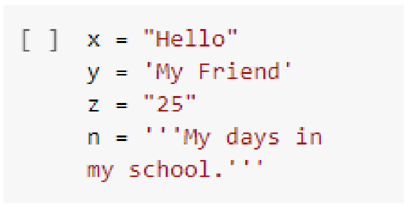


```
import keyword
print(keyword.kwlist)
```

    ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

  - **Identifier**: Identifiers are the names given to any variable, function, class, list, methods, etc. for their identification. Python is a case-sensitive language, and it has some rules and regulations to name an identifier. Here are the rules.
    - An Identifier starts with a capital letter (A-Z) , a small letter (a-z) or an underscore(_).
    - It can have digits but cannot start with a digit.
    - An identifier can't be a keyword.
    - My_name, __init__, Seven10 are valid examples.
    - 20dollers, my.var, True are invalid examples.
  - **Literals**: Literals are the values stored in program memory and are often referred to by an identifier.

- **String Literals**: The text written in single, double, or triple quotes represents the string literals in Python.



```
[ ] x = "Hello"
    y = 'My Friend'
    z = "25"
    n = '''My days in
    my school.'''
```

  - **Escape characters**: To insert characters that are illegal in a string, use an escape character. An escape character is a backslash \ followed by the character you want to insert. Some of the escape characters are as under:

Escape Character	**Result**
\'	Single Quote
\"	Double Quote
\\	Backslash
\n	New Line
\t	Tab
\b	Back space

- **Numeric Literals**: A number represented in various forms is a Numeric Literal.
  - **Integer Literal**: It includes both positive and negative numbers along with 0. It doesn't include fractional parts. It can also include binary, decimal, octal, hexadecimal literal.
  - **Float Literal**: It includes both positive and negative real numbers. It also includes fractional parts. 99.62, 0.35E-7 are valid float literals.
  - **Complex Literal**: It includes a+bi numeral, here a represents the real part and b represents the complex part.
- **Boolean Literal**: Boolean literals have only two values in Python. These are True and False.
- **Special (None) Literal**: Python has a special literal 'None'. It is used to denote nothing, no values, or the absence of value.
- **Collection Literal**: Literals collections in python includes list, tuple, dictionary, and sets.

- **Operators**: Operators are responsible for performing various operations in Python. The operators are of two types Unary (Operates on single operand) and Operators that operates on two operands (binary).
  - **Arithmetic Operators**: Arithmetic operators are used with numeric values to perform common mathematical operations:

Operators	**Name**	**Example**
+	Addition	10+20 gives 30
-	Subtraction	20-10 gives 10
*	Multiplication	30*2 gives 60
/	Division	12/3 gives 4.0
//	Floor Division	10//3 gives 3 10.0//3 gives 3.0
%	Modulus	10%4 gives 2
**	Exponentiation	3**2 gives 9

- **Assignment Operators**: Assignment operators are used to assign values to variables:

Operator	Example	Equivalent
=	n = 10	n = 10
+=	n+=10	n=n+10
-=	n-=10	n=n-10
*=	n*=10	n=n*10
/=	n/=10	n=n/10
//=	n//=10	n=n//10
**=	n**=10	n=n**10
%=	n%=10	n=n%10

- **Relational Operators**: These are used to compare two values and returns a True or False answer.

Operator	Name	Example
==	Equal to	10 == 10 is True
!=	Not Equal to	10 != 10 is False
>	Greater Than	10 > 5 is True
<	Less Than	5 < 10 is False
>=	Greater than or Equal to	10>=5 is True
<=	Less than or Equal to	5 <=10 is True

- **Logical Operators**: They are generally used along with Relational Operators to extend their scope. However, python allows them to be used independently.

Operator	Description	Example
And	Returns True if both statements are true	10 > 20 and 30 <40 will return False
Or	Return True if one or both the statements are True	10 > 20 or 30<40 will return True
Not	Reverses the result	not True is False

- **Membership Operator**: Membership operators are used to test if a sequence is presented in an object/collection:

Operator	Description	Example
In	Returns True if a sequence with the specified value is present in the object	10 in [5,10,20] will return True
not in	Returns True if a sequence with the specified value is not present in the object	20 not in [5,10,15] will return True

- **Identity Operator**: The Identity operator returns true only if two objects occupy the same memory location.

Operator	Description	Example
Is	Returns True if both variables are the same object	[10,20,30] is [10,20,30] will return False since both occupy different memory locations even if they are equal
is not	Returns True if both variables are not the same object	10 is not 10 will return False since both are same objects

- There are other operators like the **Bitwise** operators and **lambda** operator (function) - These are not in syllabus.
- **Operator precedence**: In a mathematical or logical expression the operator precedence plays an important role to decide which operator will be executed first. The following table elaborates their precedence.

Operator	**Remarks**
()	Even though () is not an operator but it plays an important roe in deciding which part of the expression should be evaluated first.
**	The unique feature of ** is that it is the only operator that is evaluated from right to left
*, /, //, %	All the four have same precedence
+, -	They are next
== != > >= < <= is is not in not in	All the relational, identity and membership operators
Not	Not being a unary operator has precedence over and/or
And	Have higher precedence over or
Or	Lowest precedence

Some Interesting operations using operators that are often asked in Examinations:

Expression	**Output**	**Explanation**
2**3**2	512	Since ** is evaluated from right to left, first 3**2 is evaluated to 9 and 2**9 evaluated 512
10 or 20	10	If the first operand of an "or" expression is true, return the first operand. Otherwise, evaluate and return the second operand.
0 or 10	10	0 is False and hence second operand is returned.
10 and 20	20	If the first operand of an "and" expression is false, return the first operand. Otherwise, evaluate and return the second operand.
Note: Any value is interpreted as "false" for the above purposes if it is 0, 0.0, None, False, or an empty collection. Otherwise, it is interpreted as "true" for the above purposes. So, 10 and 20, being nonzero numbers, are "true."		
25 % -4	-3	Python evaluates the modulus with negative numbers using the formula: **(a//b) * b + a%b == a** 25//-4 gives -7 with floor division.

		-7 * -4 gives 28. Hence a%b must be -3 to make the expression correctly equate to 25. **Note**: The sign of the result is always that of the divisor.

Questions:

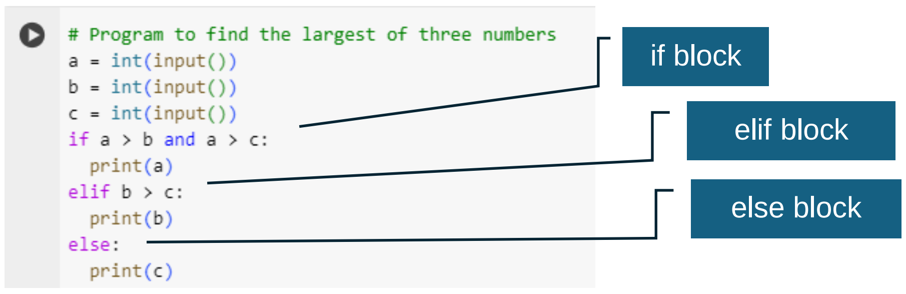
Q.1	Which one of the following is not a valid identifier? a) true b) __init__ c) 20Decades d) My_var
Q.2	Which of the following keywords is a python operator? a) for b) break c) is d) else
Q.3	What will be the output of the operation print("\\\\\\") ? a) \\\\\\ b) \\\ c) \\ d) Error
Q.4	What will be the output of the expression print(10+20*10//2**3-5) a) 30 b) 40 c) 1005 d) 130
Q.5	Evaluate the expression print(20%-3)? a) -1 b) -2 c) 2 d) Error
Q.6	What will be the result of the expression True of False and not True or True a) True b) False c) None d) Error
Q.7	What will be the output of the following program? a = {'A':10,'B':20} b = {'B':20, 'A':10} print(a==b and a is b) a) True b) False c) None d) Error
Q.8	Which of the following statements is false for python programming language? a) Python is free and Open source. b) Python is statically typed. c) Python is portable. d) Python is interpreted.

Flow of Control in Python

- Python supports sequential flow of control.
- Python supports branching flow of control using if, elif and else blocks.
- Python supports iteration control using for loop and while loop.

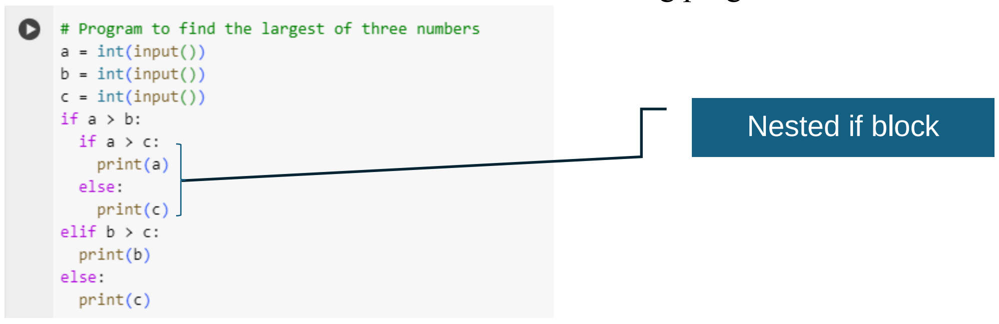
Python if, elif and else blocks:

- Python uses the relational and logical operators along with if and elif to create conditional blocks that executes a set of python statements depending on the truth value of the condition.
- The beginning of a block starts from the next line just after the : symbol and the block is indented.



```python
# Program to find the largest of three numbers
a = int(input())
b = int(input())
c = int(input())
if a > b and a > c:
  print(a)
elif b > c:
  print(b)
else:
  print(c)
```

- There could be a nested if construct as the following program shows:



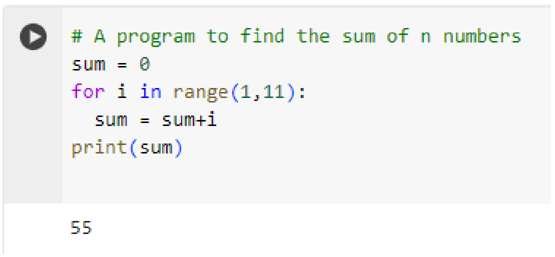
```python
# Program to find the largest of three numbers
a = int(input())
b = int(input())
c = int(input())
if a > b:
  if a > c:
    print(a)
  else:
    print(c)
elif b > c:
  print(b)
else:
  print(c)
```

With respect to the CBSE examination the students should thoroughly understand the construct of if, elif, else and often a question comes where you need to identify the errors in each program.

Q.	Re-write the following program after removing errors, if any, and underline all the corrections made.
	a = input("Enter a number:")
	b = int(input("Enter a number:"))
	if a = b:
	  a + b = a
	else
	  b = b + a
	print(a,b)
	Hint: There are four errors in the program

Python for loop:

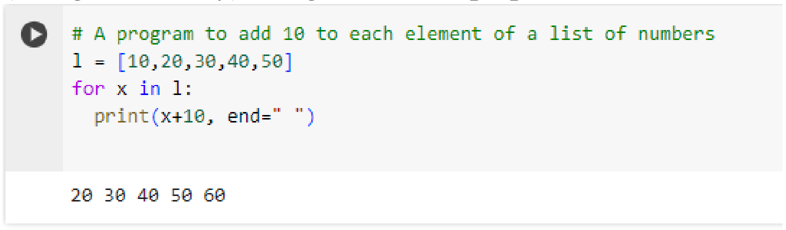
- Python for loop is used to iterate a set of python statements till a counter reaches its limit.



```
# A program to find the sum of n numbers
sum = 0
for i in range(1,11):
  sum = sum+i
print(sum)
```

```
55
```

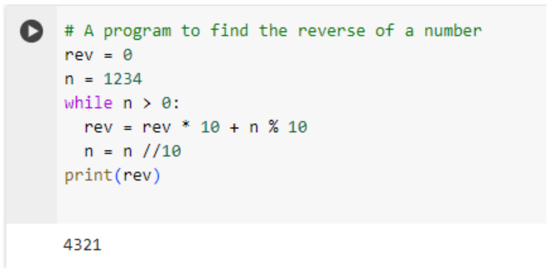
- Python for loop can also be used to iterate over a collection object (List, tuple)/ iterable (string, dictionary) using membership operators.



```
# A program to add 10 to each element of a list of numbers
l = [10,20,30,40,50]
for x in l:
  print(x+10, end=" ")
```

```
20 30 40 50 60
```

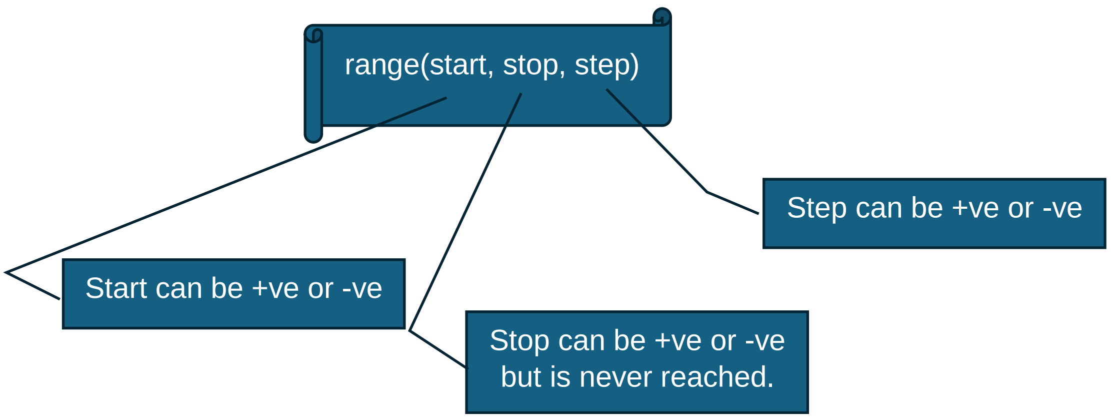
- Python while loop is used in situations where we have no idea as when the loop is going to end since there are no counters.



```
# A program to find the reverse of a number
rev = 0
n = 1234
while n > 0:
  rev = rev * 10 + n % 10
  n = n //10
print(rev)
```

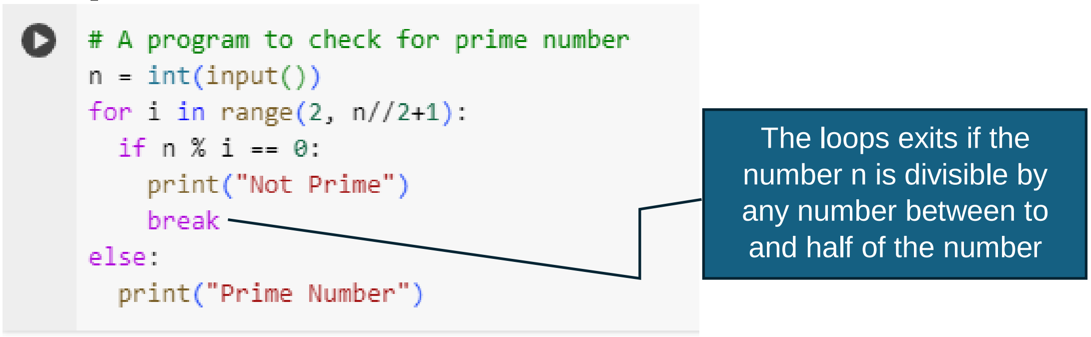
```
4321
```

- **range() function in python**: The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.



range() example	Output sequence
range(10)	0 1 2 3 4 5 6 7 8 9
range(1,11)	1 2 3 4 5 6 7 8 9 10
range(1,11,2)	1 3 5 7 9
range(10,0,-1)	10 9 8 7 6 5 4 3 2 1

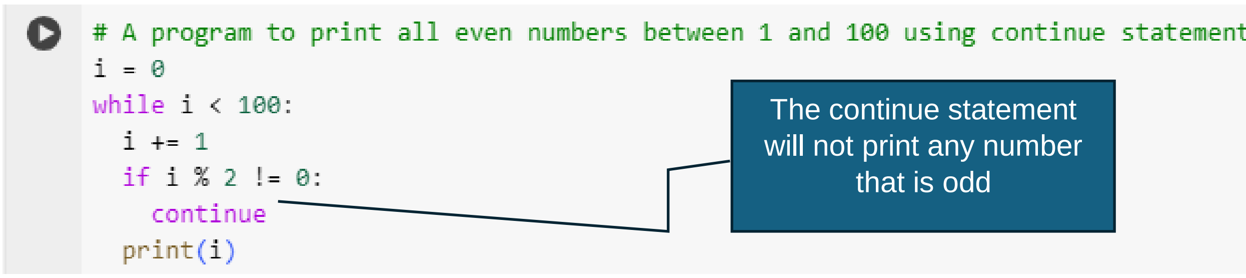
- **break statement in a loop**: The break statement stops the loop iteration and exits from the loop.



```
# A program to check for prime number
n = int(input())
for i in range(2, n//2+1):
  if n % i == 0:
    print("Not Prime")
    break
else:
  print("Prime Number")
```

The loops exits if the number n is divisible by any number between to and half of the number

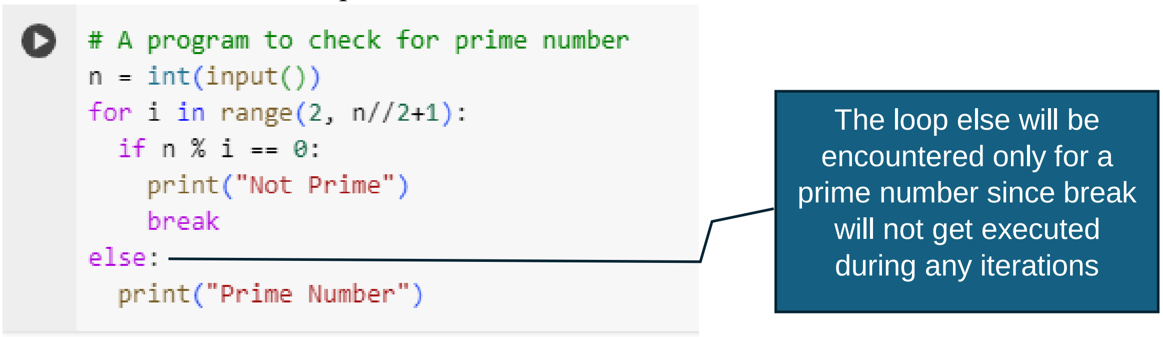
- **continue statement**: Whenever a continue statement is encountered in a loop the remaining statements after the continue statement are not executed and the loop enters next iteration.



```
# A program to print all even numbers between 1 and 100 using continue statement
i = 0
while i < 100:
  i += 1
  if i % 2 != 0:
    continue
  print(i)
```

The continue statement will not print any number that is odd

- **else block in loop**: The else block in a loop is executed when the break statement is not encountered inside the loop.



```
# A program to check for prime number
n = int(input())
for i in range(2, n//2+1):
  if n % i == 0:
    print("Not Prime")
    break
else:
  print("Prime Number")
```

The loop else will be encountered only for a prime number since break will not get executed during any iterations

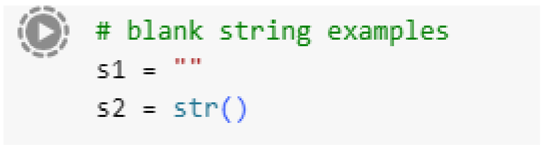
Students are advised to go through the above content since various operations involving the python data structures, user defined functions, data file handling and database interaction will require a thorough understanding of iteration. In CBSE examination you may not get a direct question from this topic except for a few MCQ or Assertion-Reasoning based question.

	The following question is an ASSERTION AND REASONING based Questions. Mark the correct choice as: i) Both A and R are true, and R is the correct explanation for A ii) Both A and R are true, and R is not the correct explanation for A iii) A is True but R is False iv) A is false but R is True
Q.	ASSERTION: In python loop else block will be executed if the loop successfully terminates after complete iteration. REASON: A python loop else block will not execute if a break statement is encountered in a loop.
Q.	ASSERTION: A continue statement in a loop is mandatory. REASON: A continue statement will skip the remaining statements in the loop after it is encountered.

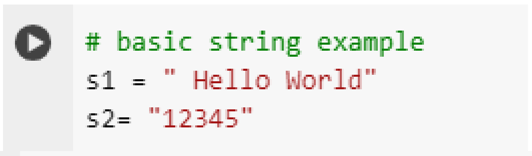
Python Strings

- Python strings are a set of characters enclosed in single quotes, double quotes, or triple quotes.
- Python strings are immutable - Once a string is created, it cannot be changed. You can create a new string with the desired modifications, but the original string remains unchanged.
- Python Strings are ordered: Strings maintain the order of characters in the sequence. This means that the characters in a string have a definite order, and this order will not change.
- Python Strings are iterable: You can iterate over the characters in a string using loops like for loops or comprehensions.
- Characters in a String are indexable: Each character in a string can be accessed using an index. Indexing starts from 0, so the first character of a string has an index of 0, the second character has an index of 1, and so on.

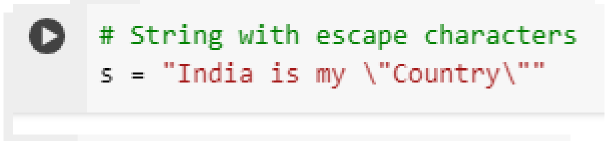
String examples:



```
# blank string examples
s1 = ""
s2 = str()
```

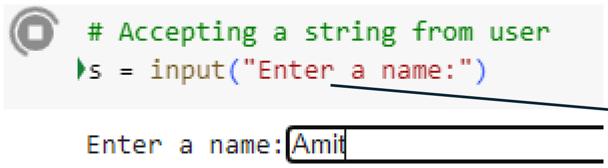


```
# basic string example
s1 = " Hello World"
s2= "12345"
```



```
# String with escape characters
s = "India is my \"Country\""
```

Accepting a string from user: We can use **input()** method to acquire a string from user.

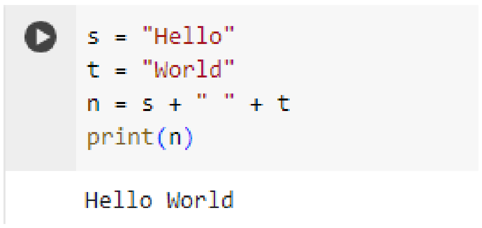


```
# Accepting a string from user
s = input("Enter a name:")

Enter a name:Amit
```

The string inside an input() method is a prompt

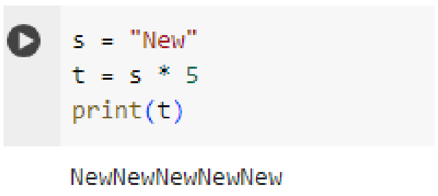
String operations:

- **Concatenation**: More than one string can be joined using the (+) operator to create a new string.



```
s = "Hello"
t = "World"
n = s + " " + t
print(n)

Hello World
```

- **Replication**: A string can be multiplied by a number to create a replicated string.



```
s = "New"
t = s * 5
print(t)

NewNewNewNewNew
```

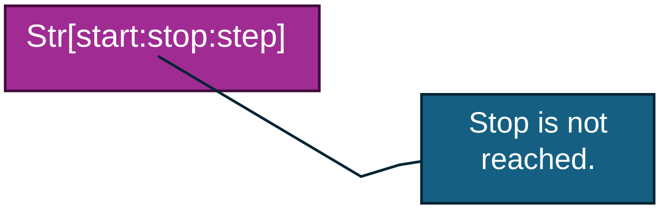
- **Indexing**: Each character of a string can be accessed using two types of indexing.
    - **Forward indexing**: First character of a string has an index 0 and next has 1 and so on.
    - **Reverse indexing**: Last character of the string is having an index of -1 and last but one has -2 and so on.

Forward index	0	1	2	3	4	5
	P	Y	T	H	O	N
Reverse index	-6	-5	-4	-3	-2	-1

We can access any element of the string using indexing.

```
s = "PYTHON"
print(s[0]) # will print P
print(s[4]) # will print O
print(s[-3])# will print H
```

- **Slicing**: A substring can be acquired from an existing string using the slicing operation.



```
[1]  s = "METAMORPHOSIS"
     print(s[0:5]) # METAM
     print(s[2:8:2]) # TMR
     print(s[::-1]) # SISOHPROMATEM
     print(s[-6:-2]) # PHOS
     print(s[-1:-7:-2]) # SSH
```

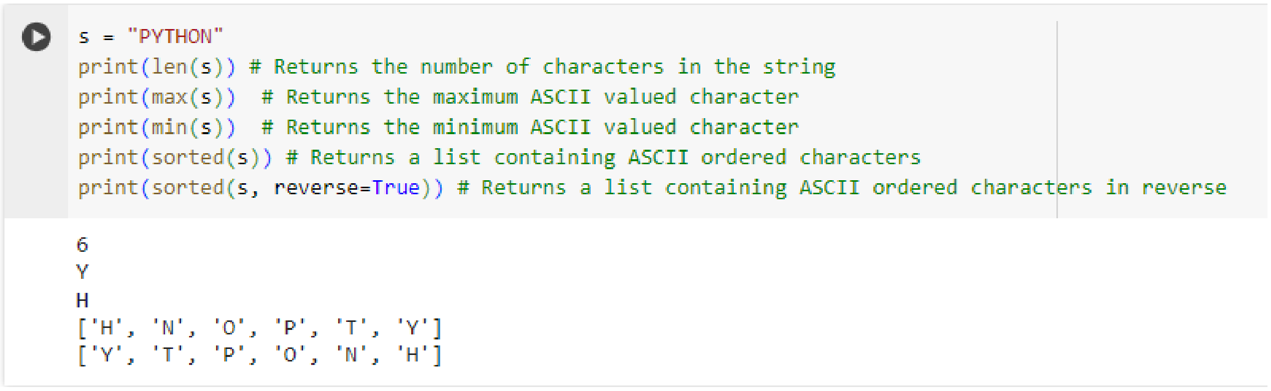
- **Traversal**: We can traverse a string using iteration and specifically using for loop.
    - **Iterate using membership**:

```
[2]  s = "PYTHON"
     for x in s:
       print(x, end=" ")

     P Y T H O N
```

    - **Iterate using indexing**:

```
[3]  s = "PYTHON"
     for i in range(len(s)):
       print(s[i], end=" ")

     P Y T H O N
```

- **String Methods**: Python has a few built-in and string library methods (also built-in) to manipulate strings. Some of them as elaborated below with examples.
    - **Global Methods**: These methods accept string as a parameter – **methodName(string)**

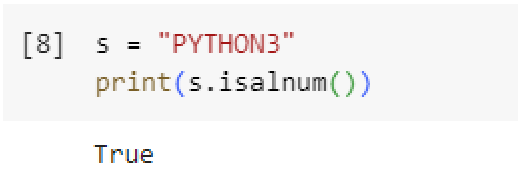


```
s = "PYTHON"
print(len(s)) # Returns the number of characters in the string
print(max(s))  # Returns the maximum ASCII valued character
print(min(s))  # Returns the minimum ASCII valued character
print(sorted(s)) # Returns a list containing ASCII ordered characters
print(sorted(s, reverse=True)) # Returns a list containing ASCII ordered characters in reverse
```

```
6
Y
H
['H', 'N', 'O', 'P', 'T', 'Y']
['Y', 'T', 'P', 'O', 'N', 'H']
```

   - **String Library Methods**: These methods have the syntax **string.methodName()**

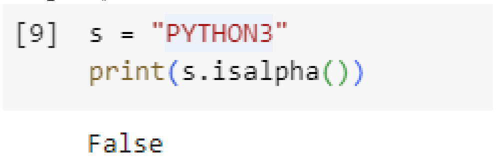
        - **Methods that return True or False**:
isalnum() – Returns True is the string comprises of only alphabets and digits



```
[8]  s = "PYTHON3"
     print(s.isalnum())
```

```
True
```

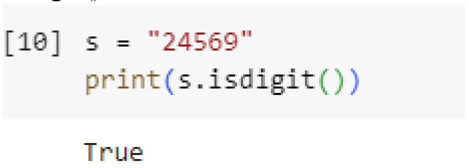
isalpha() – Returns True if all the characters are Alphabets



```
[9]  s = "PYTHON3"
     print(s.isalpha())
```

```
False
```

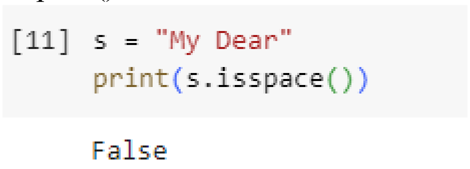
isdigit() – Returns True if all the characters are digits.



```
[10] s = "24569"
     print(s.isdigit())
```

```
True
```

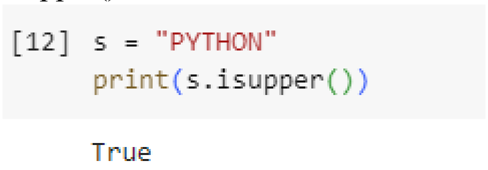
isspace() – Returns True if all the characters are spaces



```
[11] s = "My Dear"
     print(s.isspace())
```

```
False
```

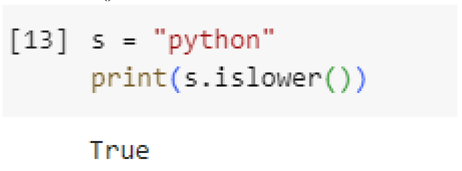
isupper() – Returns True if all the characters are upper-case alphabets



```
[12] s = "PYTHON"
     print(s.isupper())
```

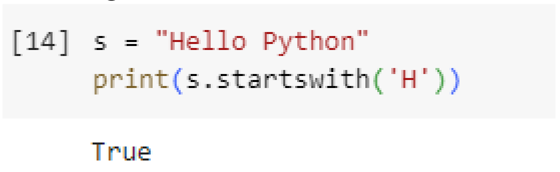
```
True
```

islower() – Returns True if all the characters are lowercase alphabets



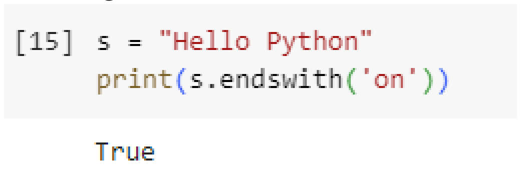
```
[13] s = "python"
     print(s.islower())

     True
```

startswith(substr) – Returns True if a string starts with the given substring.



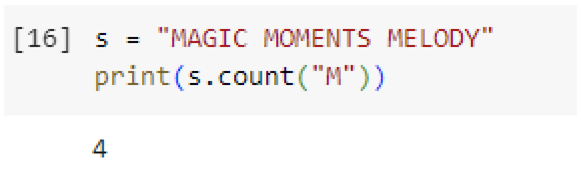
```
[14] s = "Hello Python"
     print(s.startswith('H'))

     True
```

endswith(substr) – Returns True if a string ends with the given substring.

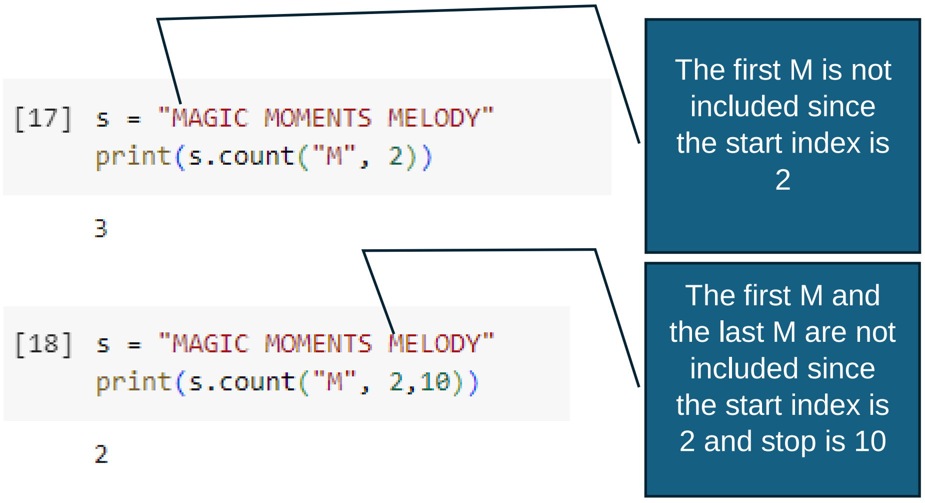


```
[15] s = "Hello Python"
     print(s.endswith('on'))

     True
```

- **Methods that return a number based on the requirement**: count(substr) – counts the occurrence of a substring inside a string. The general format of this function is count(substr, start, stop) where stop index is not included. Both start and stop are optional.



```
[16] s = "MAGIC MOMENTS MELODY"
     print(s.count("M"))

     4
```



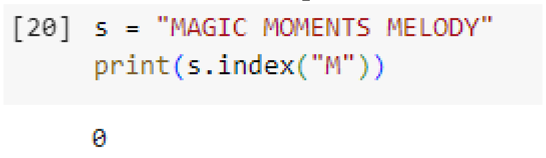
```
[17] s = "MAGIC MOMENTS MELODY"
     print(s.count("M", 2))

     3
```

The first M is not included since the start index is 2

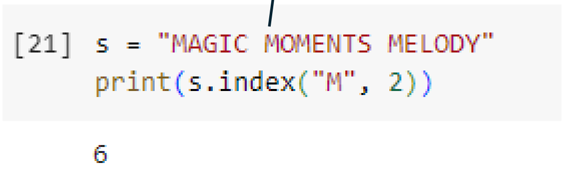
```
[18] s = "MAGIC MOMENTS MELODY"
     print(s.count("M", 2,10))

     2
```

The first M and the last M are not included since the start index is 2 and stop is 10

index(substr) – Returns the index of the first occurrence of a substring inside a given String. The general format of this method is index(substr, start, stop) where stop index is not included. Both start and stop are optional.

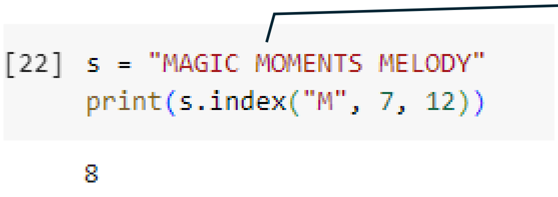


```
[20] s = "MAGIC MOMENTS MELODY"
     print(s.index("M"))

     0
```



```
[21] s = "MAGIC MOMENTS MELODY"
     print(s.index("M", 2))

     6
```

The first M is not included since the start index is 2 and hence the index value of the next M is 6



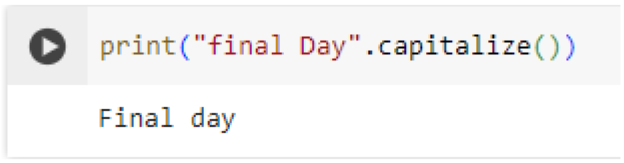
```
[22] s = "MAGIC MOMENTS MELODY"
     print(s.index("M", 7, 12))

     8
```

The start is 7 hence first two M's are not included.

find(substr) – Returns the index of the first occurrence of a substring inside a given String. The general format of this method is index(substr, start, stop) where stop index is not included. Both start and stop are optional. This is same as index()

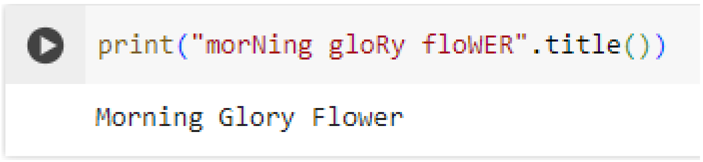
index(substr)	find(substr)
This function throws a ValueError if the substring is missing from the string.	This function returns -1 if the substring is missing from the string.

- **Methods that modify an existing string and returns a new string:**
capitalize(): Converts the first character of a string to upper case and all other alphabets to lower case. Incase the first character is not an Alphabet, only the remaining alphabets will be converted to lower case.



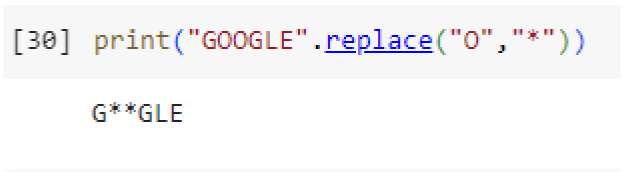
```
print("final Day".capitalize())

Final day
```

title(): Converts all the first characters of each word of a string to upper case, in case they are alphabets. The remaining alphabets are converted to lower case.



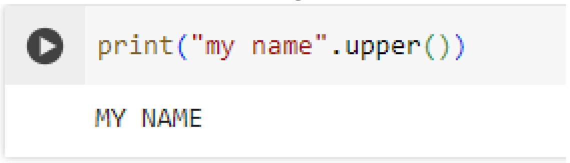
```
print("morNing gloRy floWER".title())

Morning Glory Flower
```

replace(oldsubstr, newsubstr): Replaces all the first parameter with the second parameter and returns a new string.



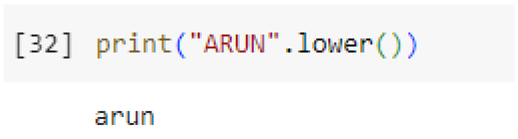
```
[30] print("GOOGLE".replace("O","*"))

     G**GLE
```

upper(): Converts all the lowercase alphabets in a string to upper case and returns a new string.



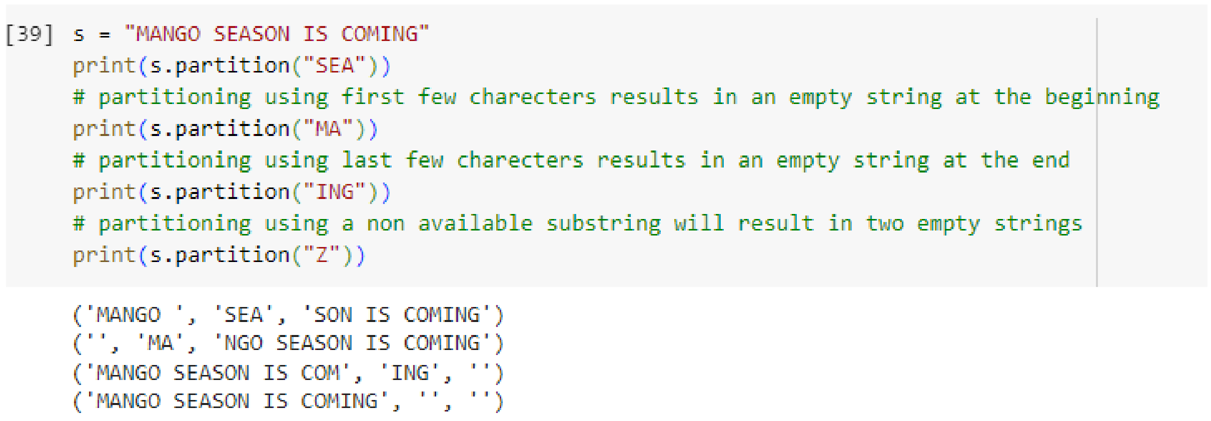
```
print("my name".upper())

MY NAME
```

lower(): Converts all the uppercase alphabets in a string to lowercase and returns a new string.



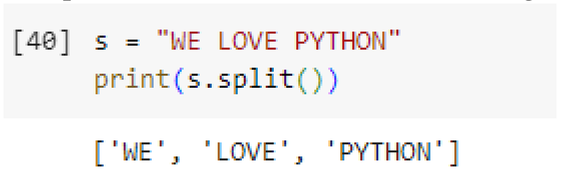
```
[32] print("ARUN".lower())

     arun
```

- **Methods that create a new object from an existing string:**
  partition(substr): Returns a tuple with three elements from a string where the middle element is the substring.



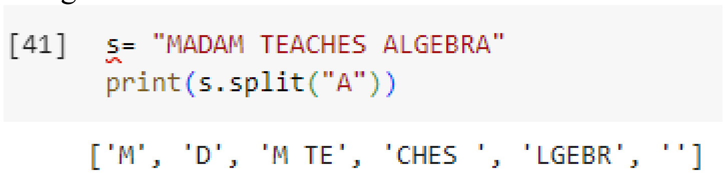
```
[39] s = "MANGO SEASON IS COMING"
     print(s.partition("SEA"))
     # partitioning using first few charecters results in an empty string at the beginning
     print(s.partition("MA"))
     # partitioning using last few charecters results in an empty string at the end
     print(s.partition("ING"))
     # partitioning using a non available substring will result in two empty strings
     print(s.partition("Z"))

     ('MANGO ', 'SEA', 'SON IS COMING')
     ('', 'MA', 'NGO SEASON IS COMING')
     ('MANGO SEASON IS COM', 'ING', '')
     ('MANGO SEASON IS COMING', '', '')
```

split() – Returns a list with a sequence of substrings by eliminating all the spaces and newlines from the existing string.



```
[40] s = "WE LOVE PYTHON"
     print(s.split())

     ['WE', 'LOVE', 'PYTHON']
```

split(substr): Returns a list with a sequence of substrings by eliminating all the occurrences of the substring from the existing string.



```
[41]  s= "MADAM TEACHES ALGEBRA"
      print(s.split("A"))

     ['M', 'D', 'M TE', 'CHES ', 'LGEBR', '']
```

Observe that all the A are removed because of the above method call and since there is no character after the last A, an empty string is introduced.

Questions:

Q.1	What will be the output of the following python statement? s = "HOME ALONE" p = s.split("O") print(p[1][:2]+p[-1]) a) ALNE b) MENE c) MEONE d) MEAL
Q.2	What will be the output of the following python code? s="finaL eXam" print(s.title()) a) FinaL Exam b) Final Exam c) FinaL exam d) Error
Q.3	What is the output of print("hello".find('E'))? a) 1 b) 2 c) -1 d) Error
Q.4	Which of the following statements is False for a python String? a) Python Strings are immutable objects. b) Python Strings can be accessed using indexing. c) Python Strings cannot be empty. d) We can get a substring from an existing string using slicing.
Q.5	What will be the correct output of the following string operation? "MALAYALAM".partition("MA") a) ("MA","LAYAL", "AM") b) ("","MA","LAYALAM") c) ("MA","LAYALA","AM") d) ("MALAYAL","AM","")

Q.6	Which of the following statements will generate an error?
	st = "PYTHON"
	t = st*5 Statement(1)
	u = st[0] + "M" Statement(2)
	st[0] = "K" Statement(3)
	st = st + st Statement(4)
	a) Statement(1)
	b) Statement(2)
	c) Statement(3)
	d) Statement(4)

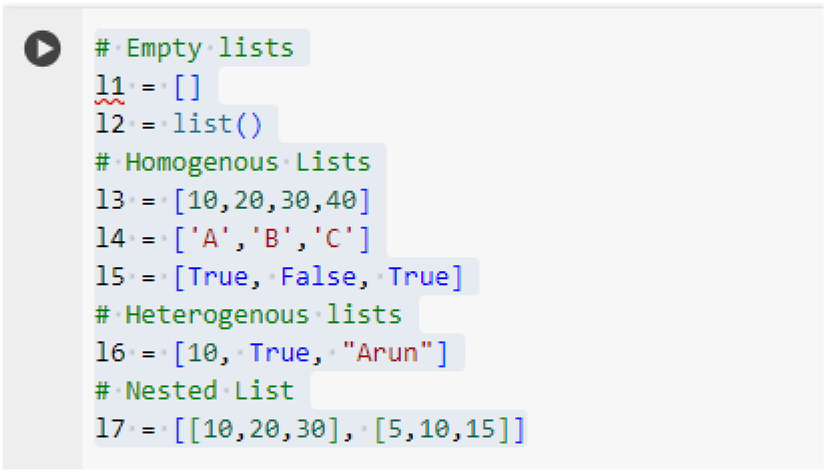
Q.7	What will be the output of the following python statement? s = "MONGO" print(sorted(s)) a) "GMNOO" b) ["GMNOO"] c) ["G","M","N","O","O"] d) Error
Q.8	What will be the output of the following string operations:

	s="Python is osome good" i) print(s.index('o',13,20)) ii) print(s[2:4]+s[14])

Python List

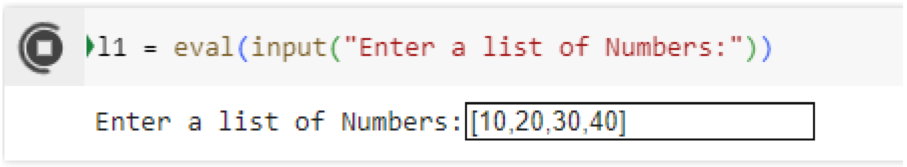
- **Ordered collection of objects** - Lists maintain the order of elements as they are inserted.
- **Lists are mutable** - Lists can be modified after creation. You can add, remove, or modify elements freely.
- **Heterogenous -** Lists can contain elements of different data types. For example, a list can contain integers, strings, floats, and even other lists.
- **Dynamic -** Lists in Python can grow or shrink in size dynamically. You can append new elements, insert elements at specific positions, or remove elements as needed.
- **Indexed -** Elements in a list are indexed with integers starting from 0. This allows for easy access to individual elements using their index.
- **Nesting -** Lists can contain other lists as elements, allowing for the creation of nested data structures.
- **Built-in Methods -** Python lists come with built-in methods for various operations like sorting, reversing, searching, etc., making them versatile for a wide range of tasks.
- **Iterable -** Lists can be used in iterations using loops (e.g., for loop)
- **Slicing -** Lists support slicing operations, allowing you to extract sublists by specifying a range of indices.

List Examples:



```
# Empty lists
l1 = []
l2 = list()
# Homogenous Lists
l3 = [10,20,30,40]
l4 = ['A','B','C']
l5 = [True, False, True]
# Heterogenous lists
l6 = [10, True, "Arun"]
# Nested List
l7 = [[10,20,30], [5,10,15]]
```

Accepting a list from User: eval() method can be used along with input() method to acquire a list from the console.



```
l1 = eval(input("Enter a list of Numbers:"))

Enter a list of Numbers:[10,20,30,40]
```

List Operations: Like a String, python lists also support operations like Concatenation, Replication, indexing, slicing and iteration. The only difference is that we can modify the elements of a list using indexing, slicing and index-based iteration. In this study material we shall see this component of python list that makes it unique mutable data structure.

- **Indexing of nested lists**:

Changing list elements using indexing: We can change the elements of a list using indexing and assignment operation.

```
[43] data = [10,20,30,40]
     data[3]=100
     print(data)

     [10, 20, 30, 100]
```

- **Changing the list elements using slicing**: We can change the replace the contents of a list using slicing too. Given below are some interesting examples.

```
data = [1,2,5,6]
data[2:2]=[3,4]
print(data)

[1, 2, 3, 4, 5, 6]
```

Observe here that the slice 2:2 is not removing any element from the list, but inserting the elements of the new list in the given index 2

```
[45] data = [1,2,5,6]
     data[2:3]=[3,4]
     print(data)

     [1, 2, 3, 4, 6]
```

Observe here that the slice 2:3 is removing element 5 from the list and inserting the elements of the new list in the given index 2

- **Changing list elements using index-based iteration**: We can modify the elements of a list using index-based iteration.

```
[46] data = [10,20,30,40,50]
     for i in range(len(data)):
       data[i]+=10
     print(data)

     [20, 30, 40, 50, 60]
```

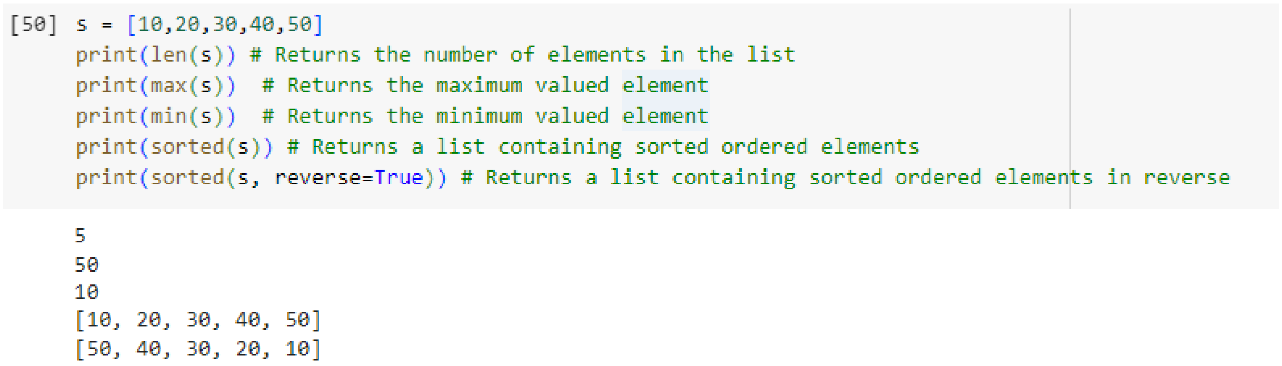
- **Deleting elements of a list using del command**: del command may be used to delete one or more than one element of a list either using indexing or using slicing.

```
[48] data = [10,20,30,40,50]
     del data[3]
     print(data)

     [10, 20, 30, 50]
```

```
[49] data = [10,20,30,40,50]
     del data[2:4]
     print(data)

     [10, 20, 50]
```

Note: Deleted elements using del command cannot be retrieved back.

- **List Methods:** Python has a few built-in and list library methods (also built-in) to manipulate lists. Some of them as elaborated below with examples:
  - **Global Methods**: These methods accept string as a parameter –

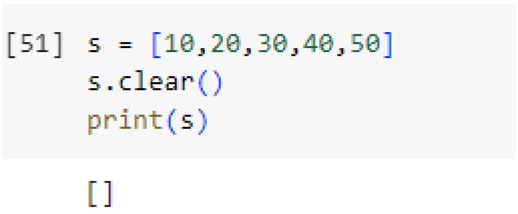


```
[50] s = [10,20,30,40,50]
     print(len(s)) # Returns the number of elements in the list
     print(max(s))  # Returns the maximum valued element
     print(min(s))  # Returns the minimum valued element
     print(sorted(s)) # Returns a list containing sorted ordered elements
     print(sorted(s, reverse=True)) # Returns a list containing sorted ordered elements in reverse

     5
     50
     10
     [10, 20, 30, 40, 50]
     [50, 40, 30, 20, 10]
```

    methodName(list)

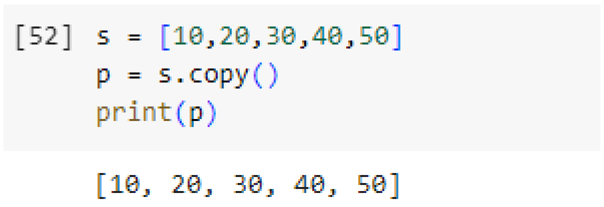
  - **List member methods:** These methods have the format **listName.methodName()**

    clear() – Removes all the elements from a list and makes the list empty.



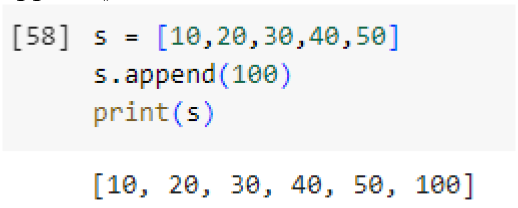
```
[51] s = [10,20,30,40,50]
     s.clear()
     print(s)

     []
```

    copy() – Creates a copy of the existing list and both list occupy different memory locations.



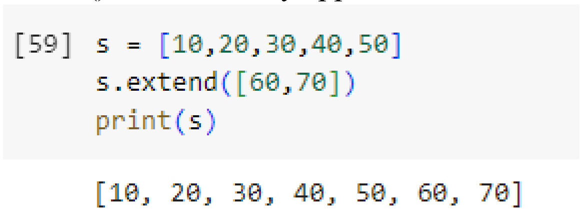
```
[52] s = [10,20,30,40,50]
     p = s.copy()
     print(p)

     [10, 20, 30, 40, 50]
```

    append() – Adds an element to the end of an existing list



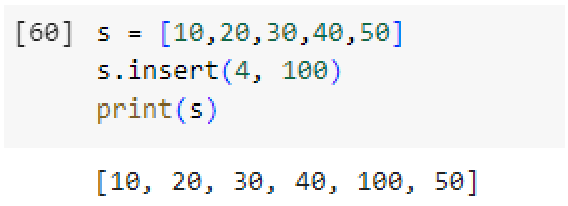
```
[58] s = [10,20,30,40,50]
     s.append(100)
     print(s)

     [10, 20, 30, 40, 50, 100]
```

    extend() – Individually appends the contents of one list to another list



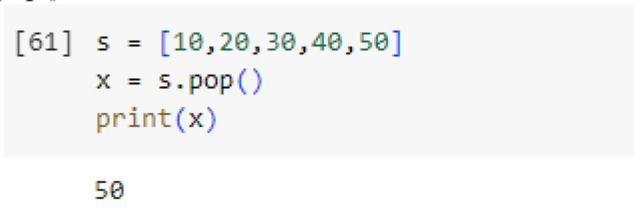
```
[59] s = [10,20,30,40,50]
     s.extend([60,70])
     print(s)

     [10, 20, 30, 40, 50, 60, 70]
```

    insert() – Inserts an element to a given index. The remaining elements are automatically shifted to the right.



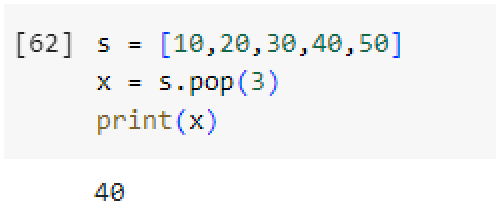
```
[60] s = [10,20,30,40,50]
     s.insert(4, 100)
     print(s)

     [10, 20, 30, 40, 100, 50]
```

pop() – Removes and returns the last element from the existing list.



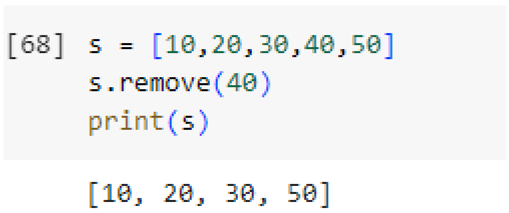
```
[61] s = [10,20,30,40,50]
     x = s.pop()
     print(x)

     50
```

pop(index) – Removes and returns the element from the given index.



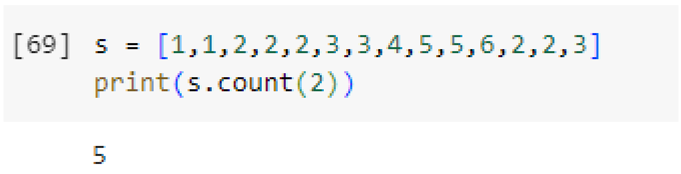
```
[62] s = [10,20,30,40,50]
     x = s.pop(3)
     print(x)

     40
```

remove(element): Removes the element from the given list without returning the element. Return a ValueError is the element is not in the list.



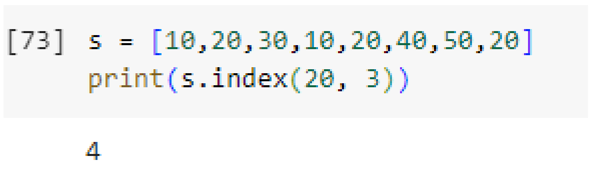
```
[68] s = [10,20,30,40,50]
     s.remove(40)
     print(s)

     [10, 20, 30, 50]
```

count(element) – Counts and returns the number of occurrences of the given element.



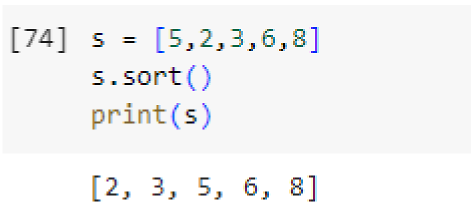
```
[69] s = [1,1,2,2,2,3,3,4,5,5,6,2,2,3]
     print(s.count(2))

     5
```

Note: Unlike count() in String there is only one parameter to count() in list.
index(element, start) – Returns the index of the first occurrence of the given element from the list.

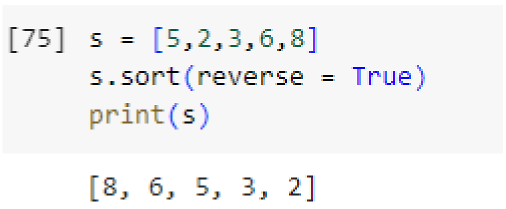


```
[73] s = [10,20,30,10,20,40,50,20]
     print(s.index(20, 3))

     4
```

If the start index is not given, the index() returns the index of first occurrence only.
sort() – Sorts the list in ascending order. Unlike sorted() this method sorts the same list and does not return a new list.

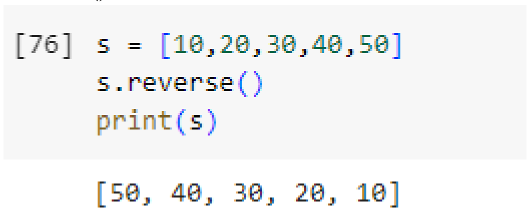


```
[74] s = [5,2,3,6,8]
     s.sort()
     print(s)

     [2, 3, 5, 6, 8]
```



```
[75] s = [5,2,3,6,8]
     s.sort(reverse = True)
     print(s)

     [8, 6, 5, 3, 2]
```

reverse() – Reverses the list based on value (ASCII value)



```
[76] s = [10,20,30,40,50]
     s.reverse()
     print(s)

     [50, 40, 30, 20, 10]
```

Questions:

Q.1	What will be the output of the following list operations?
	data = [10,20,30,[40,50,60],[70,80]]
	a) print(data[3]+data[-1])
	print(data[-2][-2])

Q.2	What will be the output of the following python program:
	data = [10,20,30, 60,70]
	data[3:3]=[40,50]
	print(data)
	data.pop(3)
	print(data)
	data.extend([10,20])
	print(len(data))

Q.3	Ganga is learning to use python Lists. Help her to get the answers of the following operations based on the given list:
	data = [10,20,30]
	data[1:3]=[5,10]
	print(data)
	data.extend([3,4])
	x =data.count(10)
	print(data[x:])
	data.sort()
	print(data)
	print(data.pop(2))

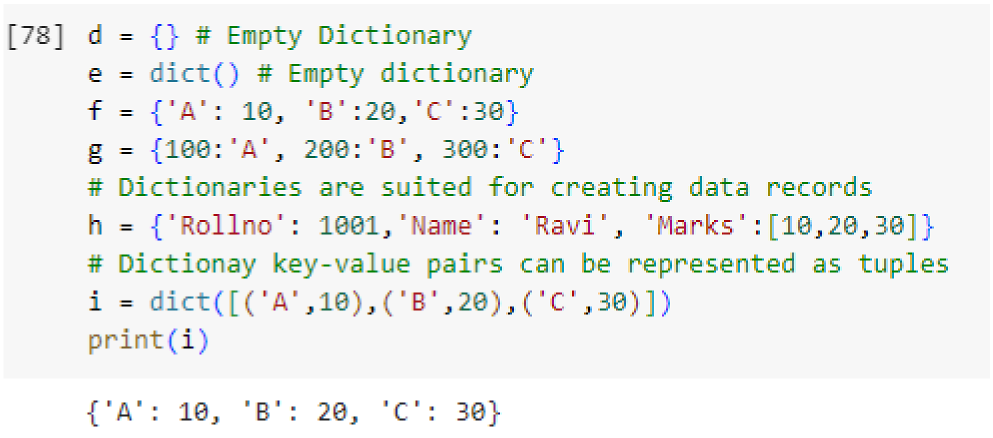
Q.4	Write a python program that accepts a list of integers from user and creates a new list from the existing list containing all the numbers that have three or more digits. Eg: for existing list [10,100, 99,200,1000] the new list should be [100,200,1000]
Q.5	Write a python program that accepts a list of countries from user and prints all the countries whose number of alphabets is more than 5.
Q.6	Write a python program that accepts a list of integers from user and prints all the integers that have 8 as the last digit. Eg: for the list [10, 28, 8, 86, 98] the program should print 28 8 98

Q.7	For the given list d=[10,30,20,15,45,50,80,90] what will be the output of the following slicing operation: d[2:7:2] a) [20,15,45] b) [20, 45, 80] c) [30, 15, 50] d) [20, 45]

Python Dictionary

- Python dictionaries are collection of key value pairs enclosed in {}
- Python dictionaries are un-ordered.
- Python dictionary keys are immutable (numbers, string, tuple)
- Python dictionary values are mutable.

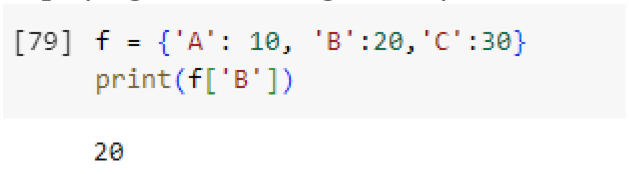
Dictionary Examples:



```
[78] d = {} # Empty Dictionary
     e = dict() # Empty dictionary
     f = {'A': 10, 'B':20,'C':30}
     g = {100:'A', 200:'B', 300:'C'}
     # Dictionaries are suited for creating data records
     h = {'Rollno': 1001,'Name': 'Ravi', 'Marks':[10,20,30]}
     # Dictionay key-value pairs can be represented as tuples
     i = dict([('A',10),('B',20),('C',30)])
     print(i)

     {'A': 10, 'B': 20, 'C': 30}
```

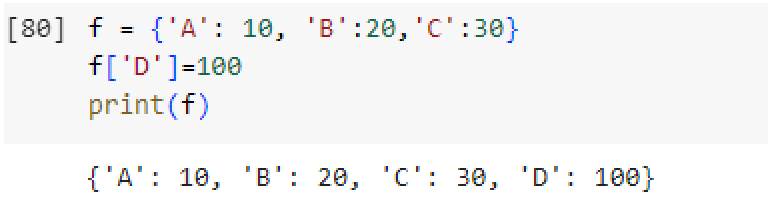
Dictionary Operations:

- **Displaying Values for a given Key**: We can use dictName[key] to get the value.

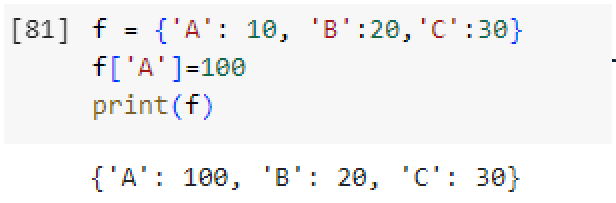


```
[79] f = {'A': 10, 'B':20,'C':30}
     print(f['B'])

     20
```

- **Adding a Key-Value pair to a dictionary**: We can add a key-value pair to a dictionary using the syntax dictName[key]=value. In case we are trying to add an existing key, then the latest value will replace the old value of the existing key without adding a new key-value pair.



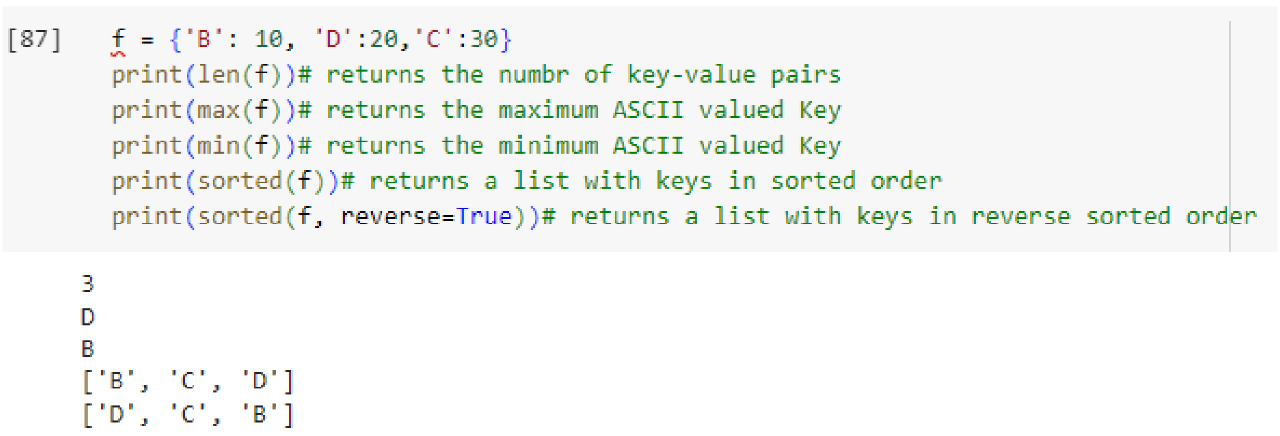
```
[80] f = {'A': 10, 'B':20,'C':30}
     f['D']=100
     print(f)

     {'A': 10, 'B': 20, 'C': 30, 'D': 100}
```



```
[81] f = {'A': 10, 'B':20,'C':30}
     f['A']=100
     print(f)

     {'A': 100, 'B': 20, 'C': 30}
```

See here that the latest value is updated for the existing key A

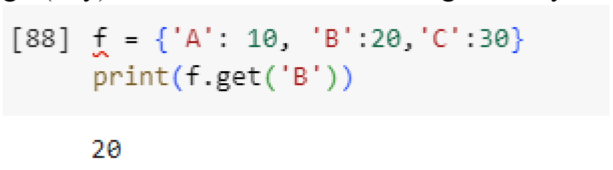
- **Dictionary Methods**: Like Strings and lists, dictionaries too have global and member functions.
    - **Global functions**: The global functions include len(), max(), min(), sum() and



```
[87]  f = {'B': 10, 'D':20,'C':30}
      print(len(f))# returns the numbr of key-value pairs
      print(max(f))# returns the maximum ASCII valued Key
      print(min(f))# returns the minimum ASCII valued Key
      print(sorted(f))# returns a list with keys in sorted order
      print(sorted(f, reverse=True))# returns a list with keys in reverse sorted order

3
D
B
['B', 'C', 'D']
['D', 'C', 'B']
```

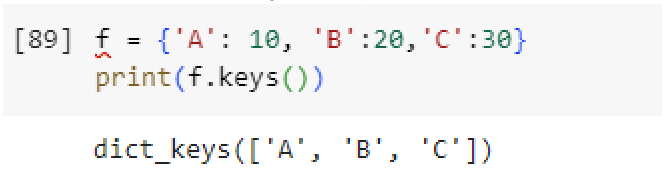
sorted()

    - **Dictionary Member Methods**: These methods are called using the syntax dictName.methodName()
      clear() – Removes all the elements from the dictionary and makes it empty.
      copy() – Creates a copy of the existing dictionary.
      get(key) – Returns the value for a given key.



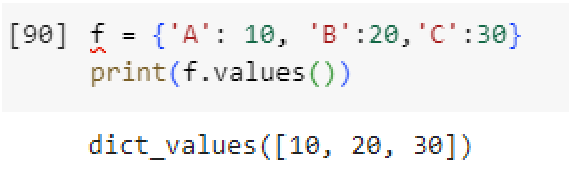
```
[88] f = {'A': 10, 'B':20,'C':30}
     print(f.get('B'))

     20
```

keys() – Returns a view object containing the keys of the dictionary, that can be converted to list using a list() method.



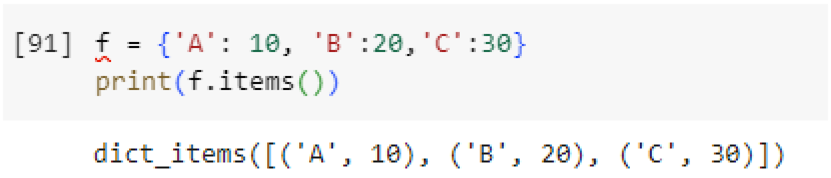
```
[89] f = {'A': 10, 'B':20,'C':30}
     print(f.keys())

     dict_keys(['A', 'B', 'C'])
```

values() - Returns a view object containing the values of the dictionary, that can be converted to list using a list() method.



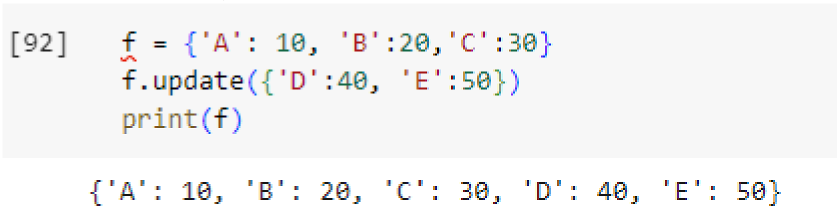
```
[90] f = {'A': 10, 'B':20,'C':30}
     print(f.values())

     dict_values([10, 20, 30])
```

items() - Returns a view object containing the key-value pairs as tuples of the dictionary, that can be converted to list of tuples using a list() method.



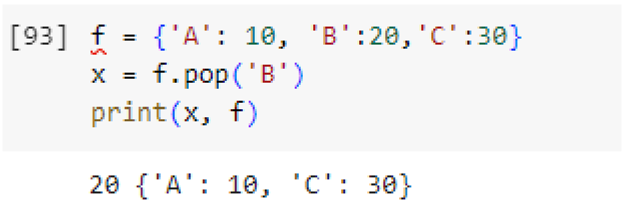
```
[91] f = {'A': 10, 'B':20,'C':30}
     print(f.items())

     dict_items([('A', 10), ('B', 20), ('C', 30)])
```

update() – Used to add the contents of one dictionary as key-value pairs in another dictionary.



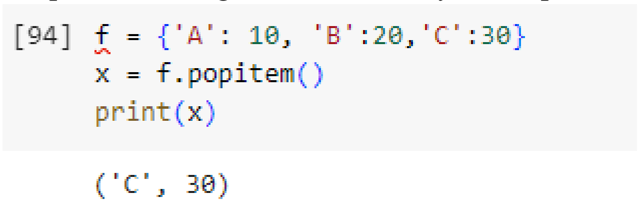
```
[92]  f = {'A': 10, 'B':20,'C':30}
      f.update({'D':40, 'E':50})
      print(f)

    {'A': 10, 'B': 20, 'C': 30, 'D': 40, 'E': 50}
```

pop(key) – Removes a key-value pair from a dictionary and returns only the value.



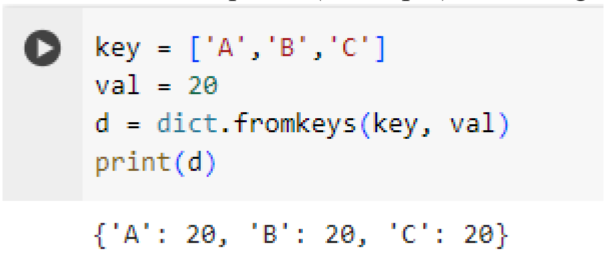
```
[93] f = {'A': 10, 'B':20,'C':30}
     x = f.pop('B')
     print(x, f)

     20 {'A': 10, 'C': 30}
```

popitem() – Reoves the last added key-value pair from the dictionary and returns a tuple containing the removed key-value pair.



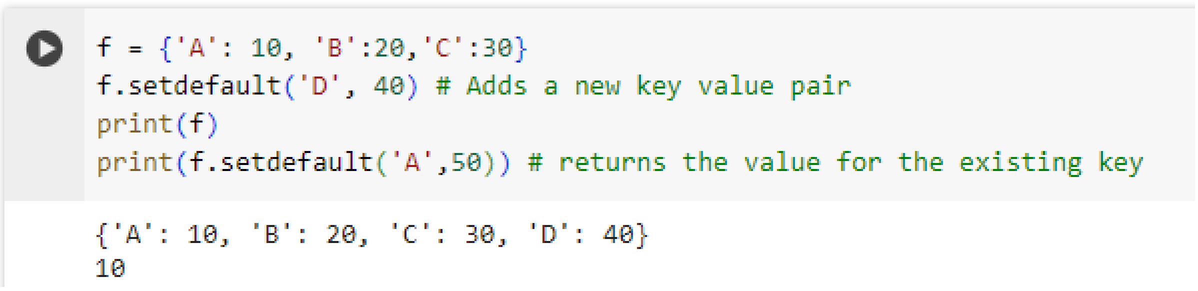
```
[94] f = {'A': 10, 'B':20,'C':30}
     x = f.popitem()
     print(x)

     ('C', 30)
```

fromkeys(key-seq, value) – Returns a dictionary containing the keys as the element of the sequence(list, tuple) and a single optional value.



```
key = ['A','B','C']
val = 20
d = dict.fromkeys(key, val)
print(d)

{'A': 20, 'B': 20, 'C': 20}
```

setdefault(key, value) – Returns the value for the key if the key is in the dictionary, else adds the key-value pair to the dictionary.



```
f = {'A': 10, 'B':20,'C':30}
f.setdefault('D', 40) # Adds a new key value pair
print(f)
print(f.setdefault('A',50)) # returns the value for the existing key

{'A': 10, 'B': 20, 'C': 30, 'D': 40}
10
```

Questions:

Q.1	Which of the following statements is False for a python dictionary? a) Dictionary Keys can be created using another dictionary. b) Dictionary values can be a dictionary. c) Dictionary Values are mutable. d) dict() function can be used to create a dictionary.

Q.2	What will be the output of the following program?
	`d={'A':10,'B':20,'C':30,'D':40}`
	`del d['C']`
	`print(d)`
	`x = d.popitem()`
	`print(x)`

	Questions 3 is an ASSERTION AND REASONING based Questions. Mark the correct choice as: i) Both A and R are true, and R is the correct explanation for A ii) Both A and R are true, and R is not the correct explanation for A iii) A is True but R is False iv) A is false but R is True
Q.3	ASSERTION: A python dictionary remains the same even if we interchange the position of key-value pairs. REASONING: Dictionaries are un-ordered collection of key-value pairs.

Q.4	What will be the output of the following?
	`d = {"A":10, "B":20, "C":30, "A":40}`
	`print(d)`
	a) {"A":10, "B":20, "C":30, "A":40}
	b) {"A":40, "B":20, "C":30}
	c) {"A":50, "B":20, "C":30}
	d) KeyError

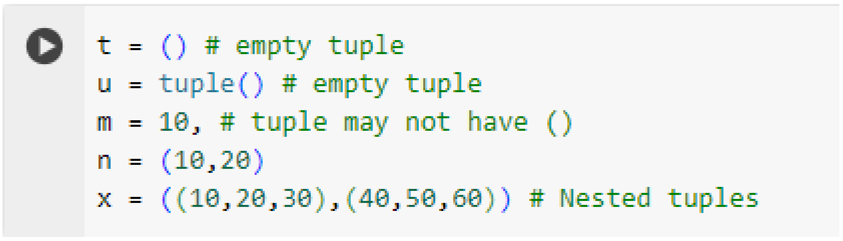
Q.5	Sapna wants to understand the concept of methods in a dictionary. Help her to find the answers of the following operations on a python dictionary:
	`d = {'M':10, 'N':20, 'O':30, 'P':40}`
	`r = d.popitem()`
	`print(r)`
	`x = d.pop('N')`
	`print(x)`
	`print(list(d.keys()))`
	`d.setdefault('X',60)`
	`print(d)`

Q.6	Write a python program that increases the values by 10 for the dictionary alphabets as keys and numbers as values where ever the key is a vowel.

Python Tuples

- Python tuples are a collection of objects enclosed in ().
- Python tuples are immutable.
- Python tuples are ordered.
- Python tuples are indexed like lists and strings.
- Python tuples may contain heterogenous elements.
- Python tuples can be nested.

Tuple examples:

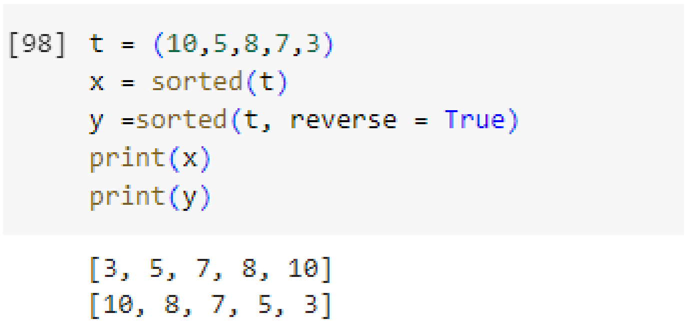


```
t = () # empty tuple
u = tuple() # empty tuple
m = 10, # tuple may not have ()
n = (10,20)
x = ((10,20,30),(40,50,60)) # Nested tuples
```

Tuple operations: Like string and lists tuples too have concatenations, replication, indexing, slicing and iteration operation. We are not going to discuss them here since you can follow the list and strings to learn and practice them.

Tuple methods: Tuples have a few global methods and only two member methods.

- Global Methods – tuple(), min(), max(), len(), sum() and sorted(). We shall discuss here only the sorted() method.



```
[98] t = (10,5,8,7,3)
     x = sorted(t)
     y =sorted(t, reverse = True)
     print(x)
     print(y)

     [3, 5, 7, 8, 10]
     [10, 8, 7, 5, 3]
```

- Tuple member methods:
  index(element) – Like lists tuple too returns the index of the first occurrence of the element.
  count(element) – Counts the occurrences of an element from a tuple as we have learned in lists.

Python Functions

Python Function:- Functions is a block of code that is identified by its name. A function can be executed by calling it. Writing the name of the function will call a function. Functions are internally declared in a separate memory area. So a function can declare variables with the same as declared in the outer part of the program.

Type of function :- Build in function (all functions defined by python min() max() , lent() etc, User-defined functions (defined by the user)

Advantage of function :- (i) Reduces the size of the program (ii) improves reusability of code

def keyword:- def keyword declares a user defined function followed by parameters and terminated with a colon.

return keyword :- whenever the return keyword is executed inside a function it returns the control back to its caller along with some value if passed explicitly. Writing return is not compulsory and we can write as many return keywords as needed but only one return keyword is executed.

Actual parameters :- When we call a function and pass some values to the function. These passed values are called actual parameters.

Formal parameters :- The parameters declared in the header part of the function is called formal parameters or the values received by the functions from its caller is called formal parameters.

Default parameters:- It is formal parameters with the assignment of values. These values are used if the caller does not provide value to that parameter. **Remember default parameters are written after not default parameters.**

def << name of the >> (formal parameters) : function body is always writer in tab indentation

code hare

code here

out of scope of function. The function call can be placed after this part.

Example :-

```
def  myfunction(a,b,c=10) :     a,b and c is formal parameter and c is with default values
        print(a,b,c)
        return (a+b+c)
total = myfunction(10,20,30)  # 10 12 and 30 are actual parameter.
```

Q. Write a function findbig that take 2 integers as parameters and returns the largest value.

```
        def  findbig(a,b):
                if a>b:
                        return a
                else:
                        return b
       x,y=5,10
       bigvalue=findbig(x,y)
```

Practice questions:

(i)
```
def  fun2(name,age):
    print(age,name)
func2(25,"Ramesh")
```
Ans :- Ramesh, 25

(ii)
```
def fun3(a,b,c):
    return a+1,b+2,c+3
```
#if more than 1 values are returned than it will be as tuple
```
t=fun3(10,20,30)
print(t)
```
Ans:- (11,12,33)

(iii)
```
def fun2(list1):
    for x in list1:
        print(x.upper(),end=”#”)
fun2([‘Rajesh’,’Kumar’])
```
Ans:- RAJESH # KUMAR

(iv)
```
def fun2(num1,num2):
    for x in range(num1,num2):if
        x%4==0:
            print(x,end=’ ‘)
fun2(10,20)
```
Ans:- 10 12 16 18

(v)
```
def prog(email):
    for x in email.split(“.”):
        if x.isalpha():
            print(“alphabet”)
       elif x.isdigit():
            print(“digit”)
       elif  x.isupper():
            print(“upper”)
        else:
            print(“all the best”)
prog(“rajesh.123.yahoo”)
```

Ans :- AlphabetDigit
Alphabet

(vi)
```
def check(x,y):
    if x != y:
        return x+5
    else:
        return y+10
    print(check(10,5))
```
Ans :- 15