Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

GETTING STARTED WITH PYTHON ROGRAMMING

INTRODUCTION

1→ Python was created by Guido Van Rossum at CWI (Centrum & Informatica) which is a National Research Institute for Mathematics and Computer Science in Netherlands.

2→ Father of Python Language is Guido Van Rossum. (Watch his pics)


3→ The language was released in I991.

4→ Python got its name from a BBC comedy series from seventies- "Monty Python"s Flying Circus".

5→ It is free to use. You can download it from the website www.python.org

SALIENT FEATURES OF PYTHON

1→ General purpose programming language which can be used for both scientific and non scientific programming.

2→ Platform independent hence highly portable language.

3→ simple high level language with vast library of add-on modules (library).

4→ Interpreted as well as compiled language.

5→ Object oriented and functional programming language.

6→ Clean and elegent coding style.

7→ Suitable for bignners (of course having strong maths and general IQ) who have no knowledge of computer language.

8→ The latest updated version of python is Python 3.6.x relesed in March 2017 and continued.

9→ Python code is significantly smaller than the equivalent C++/Java code.

10→ Python is an OSS (Open Source Software). We are free to use it, make amends in the source code and redistribute, even for commercial interests. It is due to such openness that Python has gathered a vast community which is continually growing and adding value.

USES AND APPLICATION OF PYTHON IN VARIOUS FIELDS

1→ Python is used in Google search engine, YouTube, Bit Torrent peer to peer file sharing etc.

2→ Companies like Intel, Cisco, HP, IBM use Python for hardware testing.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

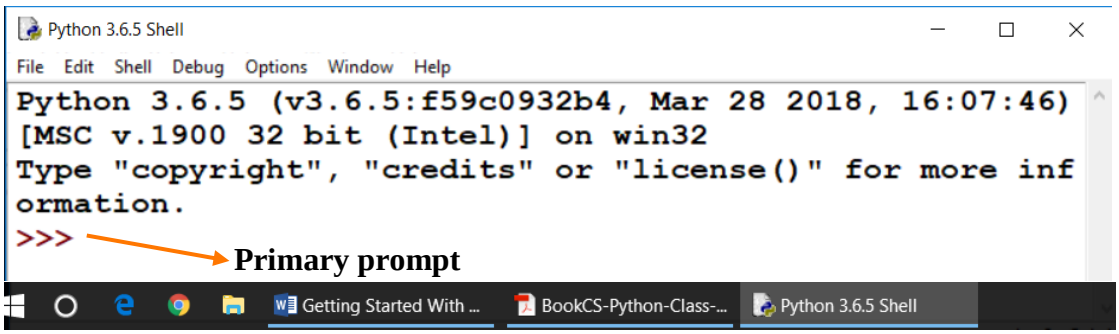
3→ Animation Company Maya uses Python Script to provide API (Application Programming Interface). Similarly, 3ds Max, Blender, Cinema 4D, Houdini apply Python 3D animation productions.
4→ A robot production company i-Robots uses Python to develop commercial robots.
5→ Python has got application in web development.
6→ Python has become the obvious choice for working in Scientific and Numeric Applications.
7→ NASA and others use Python for their scientific programming task.
8→ Python is used for developing complex GUI and image processing applications.
9→ Programmers deliver graphics software like Inkscape, Scribus, Paint Shop Pro etc.

GUI TO WRITE AND RUN PYTHON SOURCE CODE

To write and run (execute) python program we must install in our system a GUI (Graphical User Interface) called IDLE (Integrated Development and Learning Environment)
We will be using version 3.6.5 of Python IDLE to develop and run Python code, in this course. It can be downloaded from www.python.org
Python shell can be used in two ways, viz., interactive mode and script mode.
In interactive mode, we write the python statement (code) and the **interpreter** displays the result(s) immediately.
In **script mode**, we type Python program in a **new file** and then use the interpreter to execute the content from the file. For coding more than few lines, we should always save our code so that we may modify and reuse the code.

WORKING IN INTERACTIVE MODE

When we start python IDLE (Integrated Development and Learning Environment), we observe following window:



>>> is a primary prompt indicating that the interpreter is ready to take a python command. There is secondary prompt also which is "…" indicating that interpreter is waiting for additional input to complete the current statement.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 1→ Concept of script programming

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informat
ion.
>>> print('Welcome to the Python Language')
```

Now as we hit the Enter Key the result is immediately shown, as is given below:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46)
[MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more inf
ormation.
>>> print('welcome to the Python Language')
welcome to the Python Language
```

Example: 2 → What will be the output of following code in interactive mode?

A = -5
B = 7
A + 4, B – 8

Output: It's given below:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> A = -5
>>> B = 7
>>> A + 4, B - 8
(-1, -1)
```

WORKING IN SCRIPT MODE

To create and run a Python script, we will use following steps in IDLE
1. File>Open OR File>New Window (for creating a new script file)
2. Write the Python code as function i.e. script
3. Save it (Ctrl + S)
4. Execute it in interactive mode- by using RUN option (^F5)

Otherwise (if script mode on) start from Step 2

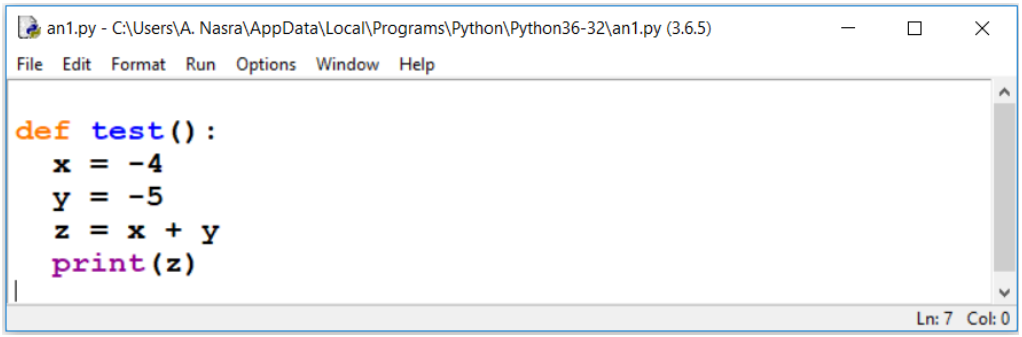
Note: For every updating of script file, we need to repeat step 3 & step 4.

Example: 3 → Find the output of the following python code

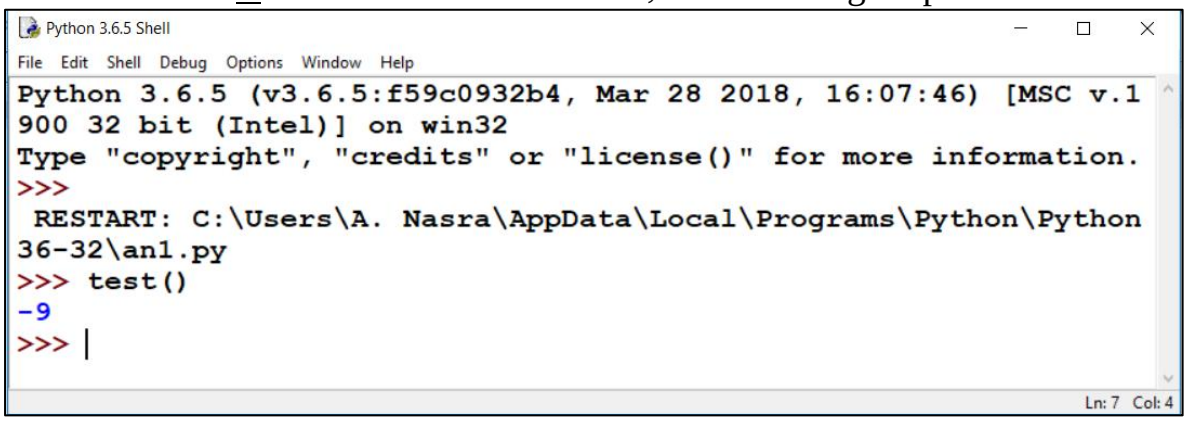
```
def test():
  x = – 4
  y = – 5
  z = x + y
  print z
```

Output: Code has been written in a file an1.py

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]



When we click Run from Menu or click F4, the following output is shown:



INDENTATION

Leading whitespace (spaces and tabs) at the beginning logical line is important. This is called indentation. Indentation is used to determine the level of the logical line, which in turn is used to determine the grouping of statements.
It means that statements which go together must have same indentation. Each such statements are called blocks. Let us understand it by the following code,

Example: 4 →

```
# File Name: indentation.py | Code by: A. Nasra
def test(): #Block.1 upto last
    a = 12
    b = 6
    if a > b:
        x = print('a is greater than b')#Block.2
        y = print('Python is smarter.')
    else:
        x = print('a is less than b')#Block.3
        y = print('Python is smarter.')
# For execution in Shell we have to call function
test()
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

MULTILINE SPANNING

String statements can be multi-line if we use triple quotes, as is given in the following example,

Example: 5 → Multiline spanning

```
>>> big = """This is a multiline
... block of text.
... Python interpreter
... puts end_of_line mark
... at the end of each line."""
>>> big
'This is a multiline\n... block of text.\n... Python
interpreter\n... puts end_of_line mark\n... at the end
of each line.'
```

COMMENTS

Comments are used to explain notes or some specific information about whatever is written in the Python code. Comments are not executed. In Python coding # is used for comments, as shown below.

Example: 6 → Illustration of comments in Python

```
>>> print("I care my students.")#this is comment
I care my students.
>>> #note that comment is not executed.
```

Many IDE's like IDLE provide the ability to comment out selected blocks of code, usually with "#".

OBJECT

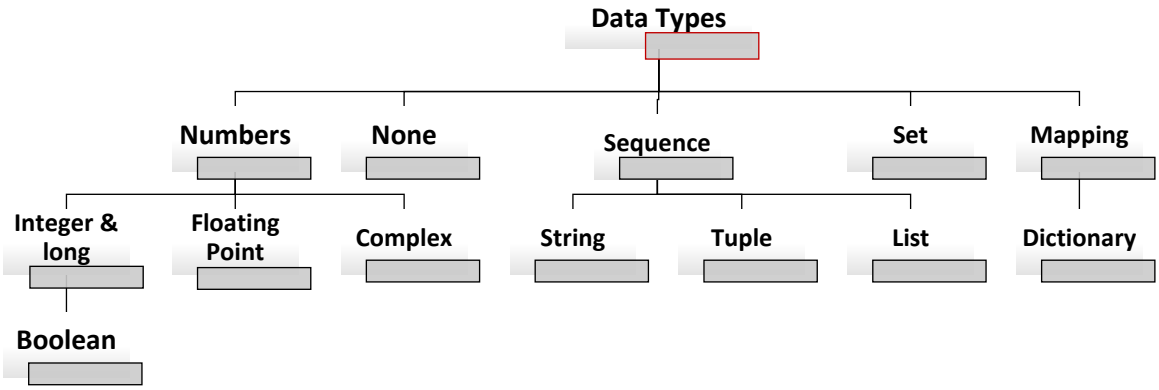
Object is an identifiable entity which combine data and functions. Python refers to anything used in a program as an object. This means that in generic sense instead of saying "Something", we say "object".
Every object has,

1. an identity - object's address in memory and does not change once it has been created.
2. data type - a set of values, and the allowable operations on those values.
3. a value

DATA TYPES

Data type is a set of values, and the allowable operations on those values. Data type is of following kind,

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]



Let us study all these data types one by one,

NUMBERS

The data types which store numbers are called numbers. Numbers are of three kinds,

1. Integer and Long (Integer also contains a data type known as Boolean)
2. Float or Floating point
3. Complex

1. INTEGERS AND LONG

Integers are the whole numbers consisting of + or – sign 100000, -99, 0, 17. While writing a large integer value, we have not to use comma or leading zeros. For writing long integer, we append **L** to the value. Such values are treated as long integers by python.

Example: 7 →

```
>>> a = 4567345
>>> type(a)
<class 'int'>
>>> # Interpreter tells that a is of integer type
>>> type(a*13)
<class 'int'>
```

Integers contain **Boolean Type** which is a unique data type, consisting of two constants, **`True`** and **`False`**. A Boolean **`True`** value is Non-Zero, Non-Null and Non-empty.

Example: 8 →

```
>>> a, b = 23, 45
>>> a > b
False
>>> b > a + b
False
>>> type(a < c)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> c = a < b
>>> type(c)
<class 'bool'>
```

2. FLOAT (FLOATING POINT)

Numbers with fractions or decimal point are known as floating point numbers. It contain – or + sign (+ is understood if not provided) with decimal point.

Example: 9 →

```
>>> f1 = 2.3 + 4 + 6
>>> print(f1)
12.3
>>> type(f1)
<class 'float'>
```

3. COMPLEX

Complex numbers are pairs of real (float) and imaginary (float attached with the sign 'j' or 'J') numbers. It is of the form A + Bj, where A and B represent float j is defined as $\sqrt{-1}$.

Example: 10 →

```
>>> A = 9 - 7j
>>> print('Real-Part = ',A.real,'Imag-part =',A.imag)
Real Part =  9.0 Imaginary part = -7.0
>>> type(A)
<class 'complex'>
```

None (NoneType)

None is special data type that is used to signify the absence of any value in a particular situation.

One such particular situation occurs with the print function. Print function displays text in the console-window (often monitor); it does not compute and return a value to the caller. It is clear from the following Example.16, where the The inner print function displayed the value 4 on in the console (it doesn't return any value. Now there is no value for outer print function, that's why the outer print function displays None. We can assign the value None to any variable. It represents "nothing" or "no object."

Example: 11 →

```
>>> x = print(print('4'))
4
None
>>> type(x)
<class 'NoneType'>
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

SEQUENCE

An ordered collection of items is known as Sequence. This ordered collection is indexed by positive integers. There are three kinds of Sequence data type – String, List and Tuple.

1. STRING (type str)

A string is a sequence of Unicode characters that may be a combination of letters, numbers, and special symbols. To define a string in Python, we enclose the string in matching single (‘ ’) or double (“ ”) quotes.

Example: 12 →

```
>>> a = 'Do you love Python?'
>>> type(a)
<class 'str'>
>>> type('p')
<class 'str'> #A string of length of 1 is a character.
```

Example: 13 →

```
>>> a = 'Do you love Python?\n'
>>> b = 'Yes, I love Python.'
>>> c = a + b
>>> print(c)
Do you love Python?
Yes, I love Python.
```

2. LIST

List is a sequence of values of any type. These values are called elements or items separated by comma. Elements of list are mutable (changeable) and indexed by integers. List is enclosed in square brackets [ ].

Example: 14 →

```
>>> L1 = [1, 564, 56.88, 'express', 'study', '*@(c)']
>>> L1
[1, 564, 56.88, 'express', 'study', '*@(c)']
>>> print(L1)
[1, 564, 56.88, 'express', 'study', '*@(c)']
>>> print(L1[3])
express
>>> type(L1)
<class 'list'>
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

3. TUPLE

Tuple is a sequence of values of any type. These values are called elements or items. Elements of tuple are immutable (non-changeable) and indexed by integers. List is enclosed in ( ) brackets.

Example: 15 → Concept of tuple.

```
>>> T1 = (7, 102, 'not-out', 50.75)
>>> print(T1)
(7, 102, 'not-out', 50.75)
>>> len(T1)
4
>>> T1[3]
50.75
```

SETS

Set is an unordered collection of values, of any type, with no duplicate entry. Sets are immutable.

Example: 16 →

```
>>> A = set([1,2,3,1,4,5,7,6,5])
>>> print(A)
{1, 2, 3, 4, 5, 6, 7}
>>> len(A)
7
```

MAPPINGS

A mapping is a collection of objects identified by keys instead of order. Mapping is an unordered and mutable. Dictionaries fall under Mappings.

NOTE – In mathematics mapping is defined as: If X and Y are two non-empty sets then a subset f of X×Y is called a function(or mapping) from X to Y if and only if for each x∈X, there exists a unique y∈Y such that (x, y) ∈f. It is written as f: X→Y

1. DICTIONARY (dict)

A dictionary is a collection of objects (values) which are indexed by other objects (keys) where keys are unique. It is like a sequence of 'key : value' pairs, where keys can be found efficiently. We declare dictionaries using curly braces {}. Keys must be immutable objects: ints, strings, tuples etc.

Example: 17 →

```
>>> wheel={0:'green',1:'red',2:'black',3:'red',
4:'yellow',5:'green', 6:'orange',7:'violet'}
>>> wheel
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
{0: 'green', 1: 'red', 2: 'black', 3: 'red', 4: 'yellow', 5: 'green', 6: 'orange', 7: 'violet'}
>>> print(wheel[3])
red
>>> len(wheel)
8
```

VARIABLE

Variable is like a container that stores values that can be accessed or changed as and when we need. If we need a variable, we will just think of a name and declare it by assigning a value with the help of assignment operator = .
In algebra, variables represent numbers. The same is true in Python, except Python variables also can represent values other than numbers.

Example: 18 → Illustration of variable in Python.

```
>>> q = 10 # this is assignment statement
>>> print(q)
10
```

The key to an assignment statement is the symbol = which is known as the assignment operator. The statement assigns the integer value 10 to the variable q. Said another way, this statement binds the variable named q to the value 10. At this point the type of q is int because it is bound to an integer value.

RULES OF DECLARING VARIABLES

1→ Variable names must have only letters, numbers, and underscores. They can start with a letter or an underscore, but not with a number. For example, a variable name may be *message_1* but not *1_message*.
2→ Spaces are not allowed in variable names. Underscores can be used to separate words in variable names. For example, *greeting_message* is valid, but *greeting message* is invalid.
3→ Python keywords and function names cannot be used as variable names.

MUTABLE AND IMMUTABLE VARIABLES

A variable (object) whose value can be changed in place, is called mutable variable. A variable whose value cannot be changed in place, is called immutable variable. Modifying an immutable variable will rebuild the same variable. Here rebuilt menas new object by the same name.
As told before, everything in Python that can be named is an object. Every object has unique identity that refers to memory location. If an object is mutable or not, can be known by the built in function **`id()`**.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 19 → Illustration of mutable and immutable variables (object).

```
>>> x = 7     # x is number object (immutable)
>>> id(x)
1604073824
>>> x = 3*x   # rebuilding x, not change inplace
>>> print(x)
21            # it seams that x in changed, no it is
              # rebuilt because id is different.
>>> id(x)
1604074048
-------------------------------------------------
>>> L = [6, 7, 8]# List is mutable object
>>> id(L)
50857544
>>> L += [1]      ''' L is changed in place, L = L + [1]
                       is rebuilding
>>> print(L)
[6, 7, 8, 1]
>>> id(L)
50857544           # id is same
```

LIST OF IMMUTABLE VARIABLES

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

KEYWORDS

Keywords are the reserved words which are used by the interpreter to recognize the structure of the program, they cannot be used as variable name.

and	as	assert	break	class	continue	def	del
elif	else	except	exec①	finally	for	from	global
if	import	in	is	lambda	not	or	pass
print	raise	return	try	while	with	yield	nonlocal②

① `exec` is no longer a keyword in Python 3.x
② `nonlocal` is added in Python 3.x, it is not in Python 2.x

Note: ① Variables are created when they are assigned a value.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

② Variables refer to object and it must be assigned a value before using it.

OPERATORS AND EXPRESSION

Operator is a symbol that does some operation on one or more than one values or variables. The values or variables on which operation is done is called operand. The operator(s) together with value(s) or/and variables, is called expression.

Example: 20 → Illustration of operators and operands.

```
>>> 3 + 4 # + is operator.3, 4 are operands
7
>>> # 3 + 4 is expression. The result/output of
      the expression is called evaluation.
```

Example: 21 → Illustration of Expression.

```
>>> a, b, c = 9, 12, 3
>>>  x = a - b/3 + c*2 - 1 #-, *, +, / are
operators.variable a, b, c, are variable with values
9,12,3.a-b/3+c*2-1 is expression.
The output(evaluation of  expression) will be stored in
variable x.
>>> print('x = ',x)
x =  10.0
```

SOME IMPORTANT OPERATORS

1. Arithmetic or Mathematical Operators
2. Relational or Comparison Operators
3. Logical Operators
4. Assignment or Shorthand Operators

Arithmetic Operators		
Operator Symbol	*Operator Name*	*Examples*

+	Addition	`>>> 3 + 5`
		`8`
		`>>> 'three' + 'five'`
		`'threefive'`

-	Subtraction	`>>> 3 - 5` `-2`

*	Multiplication	`>>> 3 * 5`
		`15`
		`>>> 'Python' * 3`
		`'PythonPythonPython'`

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

/	Division (Returns float in Python 3.5.6)	>>> 5 / 3 1.6666666666666667
%	Remainder / Modulo	
**	Exponentiation	>>> 5 ** 3 125
//	Integer division in Python 2.7; Floor division in Python 3.5.6	>>> 5 // 3 1

Relational Operators		
Operator Symbol	*Operator Name*	*Examples*

<	Less than	>>> 5 < 3
		False
		>>> 3 < 5
		True
		>>> 'C++' < 'Python'
		True

>	Greater than	>>> 5 > 3
		True
		>>> 'Python' > 'CPP'
		True

<=	Less than or equal to	>>> x = 3; y = 6; x <= y
		True
		>>> x = 4; y = 3; x >= 3
		True

>=	Greater than or equal	>>> x = 4; y = 3; x >= 3 True

= =	Equal to	>>> x = 2; y = 2; x == y
		True
		>>> x = 'str'; y = 'stR'; x == y
		False
		>>> x = 'str'; y = 'str'; x == y
		True

! =	Not equal to	>>> x = 'Low'; y = 'High' ; x != y
		True
		>>> x = 2; y = 3; x != y
		True

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Logical Operators

Operator Symbol	*Evaluation*	*Examples*
or	**x** / **Y** / **x or y** false / false / False false / true / True true / false / True true / true / True	`>>> (2==2) or (9<20)` `True`
not	**x** / **not y** false / True true / False	`>>> not (8 > 2)` `False`

and	**x** / **Y** / **x and y**
	false / false / False
	false / true / False
	true / False / False
	true / true / True
		`>>> (8>9) and (2<9)`
		`False`
		Note: and behaves
		Like multiplication.

PRESIDENCE OF OPERATORS

	Operator	**Description**
	High → Low	
Low ← High	**	Exponentiation (raise to the power)
	+ , –	unary plus and minus
	*, / , % , //	Multiply, divide, modulo, floor division
	+ , –	Addition and subtraction
	< , <= , > , >=	Comparison or Relation operators
	= = , !=	Equality operators
	% = , / = , //= , – = , + = , * =	Assignment operators
	Not, and, or	Logical operators

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 22 → Calculation of area of a rectangle.

```
>>> length = 15
>>> breadth = 9
>>> area = length*breadth
>>> print('Area of \u25AD = ',area)
Area of □ = 135
>>> print('Perimeter of \u25AD =',2*(length+breadth))
Perimeter of □ = 48
```

In the above example, we observe that the value of length and breadth are already given. It means the user has no interaction with the program. For this type of interaction we need input-output capability.

INPUT AND OUTPUT

The print function enables a Python program to display textual information to the user. Programs may use the input function to obtain information from the user. The simplest use of the input function assigns a string to a variable:

```
x = input() # in Python 3.6.5 and Python 3.7.x
x = raw_input() #in Python 2.7.x
```

We must note that the input function produces only strings by default.

```
int(input('Please enter an integer value: '))
float(input('Enter a float value:'))
```

uses a technique known as *functional composition*. The result of the input function is passed directly to the **int** function instead of using the intermediate variables shown in Interactive/Script Mode: **add_2integers.py.**

THE EVAL( ) FUNCTION

The **eval** function in Python is so powerful that it accepts **int**, **float** or **str** type data/value. The **eval** function dynamically translates the text provided by the user into an executable form that the program can process. This allows users to provide input in a variety of flexible ways; for example, users can enter multiple entries separated by commas, and the **eval** function evaluates it as a Python tuple. It is illustrated in the **add_int.py** file.

Example: 23 → CIllustration of **eval()** function.

```
# File Name: name.py | Script by: A. Nasra
x = input('Enter your name: ')
print('Hello dear',x,'! Do you want to ask something?')
y = input('...')
print('Dear',x,',you think too much! Your time is up!')
# Sample run of name.py
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Enter your name: Pillai
Hello dear Pillai ! Do you want to ask something?
...What is science
Dear Pillai ,you think too much! Your time is up!
```

Example: 24 → A program to find area.

```
# File Name: area1.py | Script by: A. Nasra
l = float(input('Length of \u25AD = '))
b = float(input('Breadth of \u25AD = '))
area = l*b
print('Area of \u25AD = ',area)
# Sample run of area1.py
Length of □ = 13.9
Breadth of □ = 11
Area of □ =  152.9
```

Example: 25 → A program to tell future.

```
# File Name: num_fun.py | Script by: A. Nasra
x = int(input('Enter a number to know the future: '))
print('You\'ve entered',x,'!\nYou silly student!.\nCan
a numbers tell future?')
```

```
# Sample run of num_fun.py
Enter a number to know the future: 45
You've entered 45 !
You silly student!.
Can a numbers tell future?
```

Example: 26 → Program to add two integers.

```
# File Name: add_2int.py | Script by: A. Nasra
x = int(input('Enter an integer: '))
y = int(input('Enter another integer: '))
print(x,'+',y,'=',x+y)
```

```
# Sample run of add_2int.py
Enter an integer: 23
Enter another integer: 98
23 + 98 = 121
```

Example: 27 → Program showing flexibility of eval() function.

```
>>> n = eval(input('Enter a number: '))
Enter a number: 76
>>> print('Number is',n,'of',type(n))
Number is 76 of <class 'int'>
>>> n = 78.6
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> print('Number is',n,'of',type(n))
Number is 78.6 of <class 'float'>
>>> n = 'om'
>>> print('Number is',n,'of',type(n))
Number is om of <class 'str'>
>>> n = 78 + 6j
>>> print('Number is',n,'of',type(n))
Number is (78+6j) of <class 'complex'>
```

Example: 28 → Efficient use os eval() function.

```
# File Name: eval_sum.py | Script by: A. Nasra
num1, num2 = eval(input('Enter number1, number2: '))
print(num1,'+',num2,'=',num1+num2)
# While entering numbers, comma must be used.
```

```
# Sample run(s) of eval_sum.py
# Sample run - 1
Enter number1, number2: 78, 86
78 + 86 = 164
# Sample run - 2
Enter number1, number2: 7.8, 8.6
7.8 + 8.6 = 16.4
# Sample run - 3
Enter number1, number2: 7 + 8j, 8 + 6j
(7+8j) + (8+6j) = (15+14j)
# Sample run - 4
Enter number1, number2: 'School', '-Captain'
School + -Captain = School-Captain
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

UNDERSTANDING FUNCTIONS

INTRODUCTION TO FUNCTIONS

In Python, a function is a named block of code that performs a specific task i.e. in the context of programming, a *function* is a named sequence of statements that performs a computation.
Functions can be categorized as belonging to
① Modules ② Built in Functions ③ User Defined Functions

MODULES

Modules are Python.py files that consist of Python code. Any Python file can be referenced as a module. Modules contains define functions, classes, and variables, that we can refer in our Python.py files or via the Python command line interpreter. There are a number of modules that are built into the Python Standard Library. Let us understand how to use modules by some examples.

Example: 29 → Example of **`import module`** format.

```
# File Name:sq_root.py | Source Code by: A.Nasra
import math     # math is module, sqrt() is function.
x = eval(input('Enter a number: '))
print('\u221A(',x,')=',math.sqrt(x))
```

```
# Sample run of sq_root.py
Enter a number: 49.49
√( 49.49 )= 7.034912934784623
```

In the above the coding of file sq_root.py imports the whole module named math and then with the help of dot notation executes/invokes/accesses/calls the sqrt() function out of many functions within the math module.
Now, let us witness the new format for using sqrt() function in our file.

Example: 30 → Example of: **`from module import function`** format.

```
# File Name:sq_root.py | Source Code by: A.Nasra
from math import sqrt
x = eval(input('Enter a number: '))
print('\u221A(',x,')=',sqrt(x))
```

```
# Sample run of sq_root.py
Enter a number: 49.49
√( 49.49 )= 7.034912934784623
```

The both format of calling **`sqrt()`** function can also be used in Python shell, as is exemplified below.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 31 → (Python shell) **`from module import function`** format

```
>>> from math import sqrt
>>> sqrt(381)
19.519221295943137
>>> sqrt(sqrt(625))
5.0
>>> import math
>>> math.sqrt(1016)
31.874754901018456
```

Example: 32 → CCalculation of Complex Numbers

```
>>> import cmath
>>> cmath.sqrt(3 + 7j)
(2.3038850997677716+1.5191729832155236j)
>>> cmath.polar(25 + 36j)
(43.829214001622255, 0.9638086627484886)
>>> cmath.exp(3 + 4j)
(-13.128783081462158-15.200784463067954j)
```

SOME IMPORTANT FUNCITONS OF MATH MODULE

Function Name	Meaning of Function	Examples

ceil(x)	It returns the smallest integer not less than x, where *x is a numeric expression.*	`>>> import math`
		`>>> math.ceil(-7.08)`
		`-7`
		`>>> math.ceil(7.08)`
		`8`

floor(x)	It returns the largest integer not greater than x, where *x is a numeric expression.*	`>>> import math`
		`>>> math.floor(22.77)`
		`22`
		`>>> math.floor(-22.77)`
		`-23`

fabs(x)	It returns the absolute value of x, *where x is a numeric value.*	`>>> import math`
		`>>> math.fabs(-13)`
		`13.0`
		`>>> math.fabs(13.07)`
		`13.07`

exp(x)	It returns exponential of x: e^x , where x is a numeric expression.	`>>> import math`
		`>>> math.exp(3)`
		`20.085536923187668`
		`>>> math.exp(math.log(2))`
		`2.0`

log(x)	It returns natural logarithm of x, for x > 0, where *x is a numeric expression.*	`>>> import math`
		`>>> math.log(10)`
		`2.302585092994046`
		`>>> math.log(math.exp(3))`
		`3.0`

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

log10(x)	It returns base-10 logarithm of x for x > 0, where *x is a numeric expression.*	`>>> import math`
		`>>> math.log10(2)`
		`0.3010299956639812`
		`>>> math.log10(1000)`
		`3.0`

pow(x, y)	It returns the value of x^y, *where x and y are numeric expressions.*	`>>> import math`
		`>>> math.pow(2,8)`
		`256.0`
		`>>> math.pow(2, 10)`
		`1024.0`

sqrt(x)	It returns the square root of x for x > 0, where x is a numeric expression.	`>>> import math`
		`>>> math.sqrt(324)`
		`18.0`
		`>>> a = 25; b = 36;`
		`>>> math.sqrt(a + b)`
		`7.810249675906654`

cos(x)	It returns the cosine of x in radians, *where x is a numeric expression*	`>>> import math`
		`>>> pi = 22/7`
		`>>> math.cos(pi/4)`
		`0.706883213624587`

sin(x)	It returns the sine of x, in radians, *where x must be a numeric value.*	`>>> import math`
		`>>> pi = 3.1415`
		`>>> math.sin(pi/4)`
		`0.7070904020014415`

tan(x)	It returns the tangent of x in radians, *where x must be a numeric value.*	`>>> import math`
		`>>> math.tan(22/16)`
		`5.041915256481364`

degrees(x)	It converts angle x from radians to degrees, *where x must be a numeric value.*	`>>> import math`
		`>>> math.degrees(3.1415)`
		`179.99469134034814`

radians(x)	It converts angle x from degrees to radians, *where x must be a numeric value.*	`>>> import math`
		`>>> math.radians(180)`
		`3.141592653589793`

SOME IMPORTANT FUNCITONS OF RANDOM MODULE

Function Name	Meaning of Function	Examples

random( )	It returns a random float x, such that $0 \leq x < 1$	`>>> import random`
		`>>> random.random()`
		`0.1221307683291536`

randint (a, b)	It returns a int x between a & b such that $a \leq x \leq b$	`>>> import random`
		`>>> random.randint(9, 99)`
		`49`

uniform (a,b)	It returns a floating point number x, such that
	a <= x < b
		`>>> import random`
		`>>> random.uniform(9, 99)`
		`28.121517332325915`

randrange (start, stop, step)	It returns a random item from the given range	`>>> import random`
		`>>> random.randrange(10, 50, 11)`
		`43`
		`>>> random.randrange(10, 50, 11)`
		`21`

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

SOME IMPORTANT FUNCITONS RELATED TO TIME

The time package (Package is folder that contains modules) has a number of functions that relate to time. We will consider two functions: **`clock()`** and **`sleep()`**. The **`clock()`** function measures the time of parts of a program's execution and returns a floating-point value, in seconds. The **`sleep()`** function suspends the program's execution for a specified number of seconds. Exact times will vary depending on the speed of the computer.

Note:- In Python 3.3 **`clock()`** has been deprecated and will be removed in Python 3.8. Instead, we can use **`process_time()`** function.

Let us understand the use of these functions by following examples.

Example: 33 → CExample of **`time`** module.

```
# File Name: time_passed.py | Source Code by: A.Nasra
from time import process_time
# Notice the sequence of following 2 statements.
start = process_time()
name = input('Enter your name: ')
time_passed = process_time()-start
print(name,',you took',time_passed,'seconds to write
your name.')
```

```
# Sample run of time_passed.py
Enter your name: Mr. India
Mr. India ,you took 0.03125 seconds to write your name.
```

Example: 34 → CFinding current time.

```
# File Name: time1.py | Source Code by: A.Nasra
import time;
localtime = time.asctime(time.localtime(time.time()))
print("Local current time :", localtime)
```

```
# Sample run of time1.py
Local current time : Fri Aug 17 10:57:52 2018
```

Example: 35 → Finding the calender of current year and month.

```
import calendar
cal = calendar.month(2018,8)
print("Here is the calendar:")
print(cal)
```

Run the above program and see the result/output.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

BUILT IN FUNCITONS

The Python interpreter has a number of functions that are always available for use. These functions are called built-in functions. We don't have to import any module (file). Built in functions are used with appropriate object. In Python 3.6.7 there are 68 built in functions. Some important functions are listed below.

SOME IMPORTANT BUILT IN FUNCITONS

Function Name	Meaning of Function	Examples

`abs (x)`	It returns absolute value, where *x is a numeric expression.*	`>>> abs(-45)`
		`45`
		`>>> abs(3 + 4j)`
		`5.0`

`max(x, y, z, ...)`	It returns the largest of its arguments: where x, y and z are numeric variable/expression.	`>>> max(34,89,67,98,34)` `98`
`min(x, y, z, ...)`	It returns the smallest of its arguments; where x, y, and z are numeric variable / expression.	`>>> min(34,89,67,98,34)` `34`
`cmp(x, y)`	It returns the sign of the difference of two numbers: -1 if x < y, 0 if x == y, or 1 if x > y, *where x and y are numeric variable/expression.*	[N/A for Python 3.6.5]

`divmod(x,y)`	Returns both quotient and remainder by division through a tuple, when x is divided by y; where x & y are variable / expression	`>>> divmod(67, 9)`
		`(7, 4)`
		`>>> divmod(5.67, 1.2)`
		`(4.0, 0.8700000000000001)`

`len(s)`	Return the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).	`>>> len((2,3,4,5,7))`
		`5`
		`>>> len([3,5,7,8])`
		`4`
		`>>> len((3,'tab',4,'less'))`
		`4`

`range(start, stop, step)`	This is a versatile function to create lists containing arithmetic progressions. It is most often used in for loops. The arguments must be plain integers. If the *step* argument	`>>> n = 0`
		`>>> for n in range(21, 0, -3):`
		`print(n, '', end='')`
		
		`21 18 15 12 9 6 3`

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

	is omitted, it defaults to 1. If the *start* argument is omitted, it defaults to 0. The full form returns a list of plain integers [start, start + step, start + 2 * step, ...]. If *step* is positive, the last element is the largest start + i * step less than *stop*; if *step* is negative, the last element is the smallest start + i * step greater than *stop*. *Step* must not be zero (or else Value Error is raised).	

`round(x, n)`	It returns float x rounded to n digits from the decimal point, *where x and n are numeric expressions*. If n is not provided then x is rounded to 0 decimal digits.	`>>> round(3.14153,3)`
		`3.142`
		`>>> round(3.14153,2)`
		`3.14`

`bin(d)`	Convert an integer number to a binary string prefixed with "0b". The result is a valid Python expression.	`>>> bin(3) ; bin(-3)`
		`'0b11'`
		`'-0b11'`
		`>>> # to convert in dec`
		`>>> int(0b11) ; int(-0b11)`
		`3`
		`-3`

`oct(d)`	Convert an integer number to an octal string prefixed with "0o".	`>>> oct(8) ; oct(-56)`
		`'0o10'`
		`'-0o70'`
		`>>> int (-0o70)`
		`-56`

`hex(d)`	Convert an integer number to a lowercase hexadecimal string prefixed with "0x".	`>>> hex(255) ; hex(-42)`
		`'0xff'`
		`'-0x2a'`
		`>>> int(0xff)`
		`255`

USER DEFINED FUNCITONS

A function is a block of code which only runs when it is called. We can pass data, known as parameters, into a function. A function can return data as a result.

CREATING A FUNCTION

In Python a function is defined using the **`def`** keyword.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 36 → Understanding **def** keyword.

```
# Creation of a simple function.
def urn():       # urn means your name
    name = input('Enter your name: ')
    print('Hello',name,'You are a God-feating
person.')
```

CALLING A FUNCTION

A sample run of **urn.py** is given below: when we write urn() in Python shell, it means, we are calling the function that we had created in Python script.

```
# Sample run of urn.py
>>> urn()
Enter your name: Alina
Hello Alina .You are a God-fearing person.
```

PARAMETERS AND ARGUMENTS

Parameters are specified within the pair of parentheses in the function definition, separated by commas. When we call the function, we supply the values in the same way. Note the terminology used - the names given in the function definition are called *parameters* whereas the values you supply in the function call are called *arguments*. Let us inspect the following example.

Example: 37 → Understanding role of parameters and arguments in definition.

```
# File Name: max_min.py | Source Code: A. Nasra
def tell_max(a,b):  # a and b are parameters
    if a > b:
        print(a,'is maximum.')
    elif a == b:
        print(a,'is equal to', b)
    else:
        print(b,'is maximum.')
```

```
# Sample run of max_min.py
>>> tell_max(43,56)   # Here 56(value of a) and
56 is maximum.        # 56(value of b) are arguments
```

DEFAULT PARAMETERS

While calling, if the user forgets or does not want to provide values for the parameter, then the *default argument* values are provided by the program while coding the parameter that is called *default parameter*. Note that the default argument value should be a constant (immutable). How to do it, let us understand by the following example.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 38 → C (File Name: **volume2.py**)

```
# File Name: volume.py | Source code by: A. Nasra
print('Call the function vol(length,breadth,height)')
def vol(l, b, h = 10): # Here h is default parameter
    print('Volume =',l*b*h)
```

See, the sample run, third argument is not provided, hence it is taken as 10.

```
# Sample run of volume.py
Call the function vol(length, breadth, height)
>>> vol(23,40)
Volume = 9200
#When height is not given it is taken as 10, the value
of default parameter
```

RETURN STATEMENT

Return statement returns a value from the function. Return statement may contain a constant/literal, variable, expression or function, if return is used without anything, it will return **None**. Let us understand the following coding.

Example: 39 → Program to find the Fibonacci Series.

```
#Fibonacci series
#File Name: fib1.py | Code by: A. Nasra
print('Call the function fib(n) for the Fibonacci
series up to n terms.')
n=0
def fib(n):
    a, b = 0, 1
    for n in range(1,n,1):
        print(a, end=' ')
        a, b = b, a+b
    return a
fib(n) #calling the function
```

```
# Sample run of fib1.py
Call the function fib(n) for the Fibonacci series for
n terms.
>>> fib(9)
0 1 1 2 3 5 8 13 21
```

Example: 40 → Default parameter.

```
# File Name: volume1.py | Source Code by: A. Nasra
print('call the function vol(length,breadth,height)')
def vol(l, b, h = 2):
    v = l*b*h
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
    print('Volume = ',end='')
    return v
```

```
# Sample run of volume1.py
call the function vol(length,breadth,height)
>>> vol(15,7)
Volume = 210
>>> vol(1.1,2.2,3.3)
Volume = 7.986000000000001
>>> vol(1 + 2j, 3 + 4j)
Volume = (-10+20j)
```

SCOPE OF VARIABLES

Scope of variable refers to the part of the program, up to which the variable is accessible. We will study two types of scopes – Global and Local scopes.
When a variable is created outside all functions/blocks, it is called global ***variable.***

Example: 23 → Example of global variable.

```
# File Name: global.py | Code by: A. Nasra
x = eval(input('Enter a global value: '))
print('Now call the function gal()')
def gal():  # gal stands for global
    print('Global value is',x)
```

```
# Sample run to global.py
Enter a global value: 78
Now call the function gal()
>>> gal()
Global value is 78
```

Example: 42 → Example of local variable.

```
# File Name: global.py | Code by: A. Nasra
print('Call the function lal()')
def lal():  # lal stands for local
    y = eval(input('Enter a local value: '))
    print('Local value is',y)
```

```
# File Name: local.py
Call the function lal()
>>> lal()
Enter a local value: 86
Local value is 86
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 43 → Recognising global and local values.

```
# File Name: global_local.py | Coded by: A. Nasra
x = eval(input('Enter a global value: '))
print('Now, call gl()')
def gl():    # gl stand for global local
    y = eval(input('Enter a local value: '))
    print('Global value is',x)
    print('Local value is',y)
```

```
# Sample run of global_local.py
Enter a global value: 47
Now, call gl()
>>> gl()
Enter a local value: 45
Global value is 47
Local value is 45
```

DOCSTRING

The multiline string in the block of a function definition (or the first line in a module) is known as a *documentation string*, or *docstring* for short.
Docstring provides information about
• The purpose of the function.
• The role of each parameter.
• The nature of the return value.

Example: 44 → Calculation of distance between two points.

```
# File Name: distance.py | Coded by: A. Nasra
from math import sqrt
print('Call the function dist(x1,y1,x2,y2)')
def dist(x1,y1,x2,y2):
    '''The function dist(x1,y1,x2,y2) returns the
    distance between two coordinates (x1,y1) and
    (x2,y2)'''
    d = sqrt((x2-x1)**2 + (y2-y1)**2)
    print('Distance =',end=' ')
    return d
```

```
# Sample run of file distance.py
Call the the function dist(x1,y1,x2,y2)
>>> dist(1,2,3,4)
Distance = 2.8284271247461903
```

KEYWORD ARGUMENT

We know that a print function accepts an additional argument that allows the cursor to remain on the same line as the printed text:

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
print('Please enter an integer value:', end='')
```

The expression **end=''** is known as a keyword argument. The term keyword here means something different from the term keyword used to mean a reserved word.
Another keyword argument named **sep** specifies the string to use insert between items. The default value of **sep** is the string **''**, a string containing a single space. The name **sep** stands for separator.

Example: 45 → Use of separator.

```
# File Name: separator.py | Coded by: A. Nasra
d, m, y = 19, 8, 2018
print('TODAY\'S DATE is given below:')
print(d, m, y)
print(d, m, y, sep= ':')
print(d, m, y, sep= '/')
print(d, m, y, sep= '|')
```

```
# Sample run of printsep.py
TODAY'S DATE is given below:
19 8 2018
19:8:2018
19/8/2018
19|8|2018
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

CONDITIONAL STATEMNTS AND IRETATION

STATEMENTS AND IT TYPE IN PYTHON

Statements are the instruction to the computer to do some work and produce the result. There are three types of statements in Python:

1. Empty Statement
2. Simple Statement (Single statement)
3. Compound Statement

EMPTY STATEMENT

The statement that does nothing is called **empty statement**. In Python empty statement is **pass** statement. In Python programming, pass is a null statement.

Example: 46 → Illustration of pass statement.

```
#Ex of pass statement | File Name: ex_pass.py
#Code by: A. Nasra
for letter in 'abba':
    if letter == 'b':
        pass
        print('Letter b is pass.')
    else:
        print('Current Letter :', letter)
```

```
# Sample run ot ex_pass.py
Current Letter : a
Letter b is pass.
Letter b is pass.
Current Letter : a
```

SIMPLE STATEMENT

A single line logical statement that is executable is called simple statement in Python. Several simple statements may occur on a single line separated by comma. One of the example of simple statement is assignment statement.
Simple statements may be of following types:

1. expression_stmt
2. assert_stmt
3. assignment_stmt
4. augmented_assignment_stmt
5. annotated_assignment stmt
6. pass_stmt
7. del_stmt
8. return_stmt
9. yield_stmt
10. raise_stmt
11. break_stmt
12. continue_stmt
13. import_stmt
14. global_stmt
15. nonlocal_stmt

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Many of the statements we are already apprised of or will be apprised of in further course of study.

COMPUND STATEMENT

A group of statements, headed by a line is called compound statement. All of the following comes under the compound statement,

1. if_stmt
2. while_stmt
3. for_stmt
4. try_stmt
5. with_stmt
6. funcdef
7. classdef
8. async_with_stmt
9. async_for_stmt
10. async_funcdef

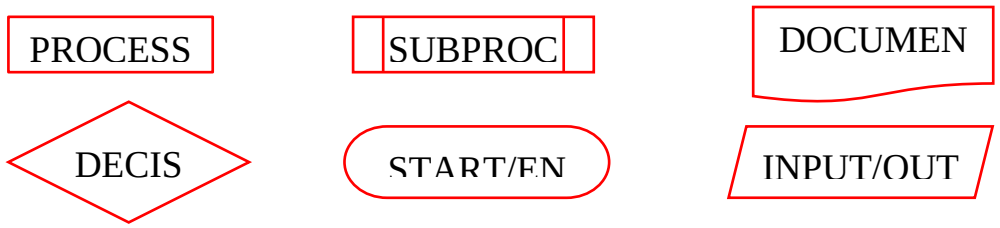
PROGRAM LOGIC DEVELOPMENT TOOLS

These are the tools that facilitate the writing the actual program one of such tools is algorithm. The well-defined step by step instructions (or procedures) is celled algorithm. Algorithm is made with the help of following tools

1. Flowchart/Flow-diagram
2. Pseudocode
3. Decision Trees

FLOWCHART

A flowchart is a graphical representation of an algoritham to solve a given problem. It consist of following symbols for different steps of the algorithm.



PSEUDOCODE

Pseudocode is an easy going natural language that helps programmers develop algorithm without using any programming language. Pseudocode is a 'text based' detail design tool

DECISION TREES (DT)

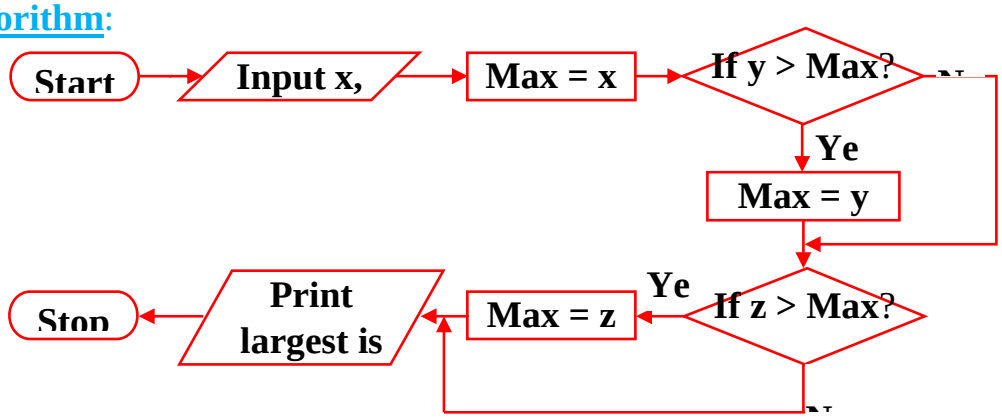
Decision tree is a type of hierarchical and sequential algorithm that is mostly used in classification problem. The basic idea of DT is to split the data continually according to one (or multiple) attributes (rules), so that we end up with sub-sets that have single outcomes.

EXAMPLES OF FLOWCHART, PSEDUCODE, DT WITH PYTHON CODE

Example: 47 → Write a program to accept three integers and print the largest of the three. Make use of only `if` statement.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Algorithm:



Pseudocode:

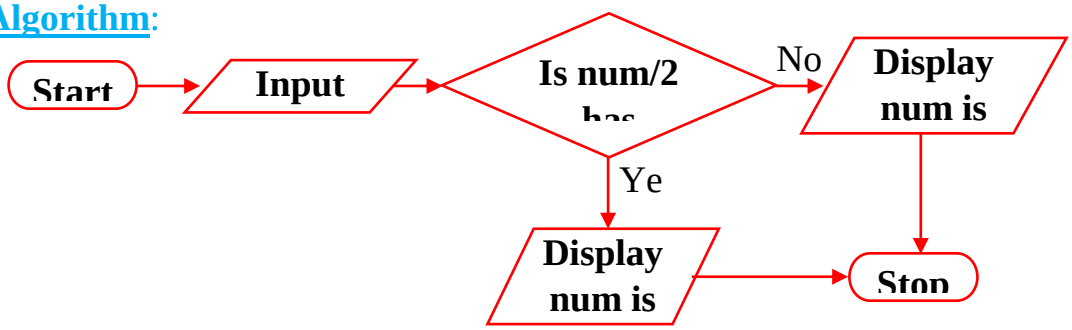
```
Input three numbers x, y, z
Max = x      (Note x is the first number)
If second number y is more than max then
Max = y
If third number z is more than max then
Max = z
Display max number as largest number
```

Code In Python:

```
x = y = z = 0
x = eval(input("Enter first number: "))
y = eval(input("Enter second number: "))
z = eval(input(Enter third number: "))
max = x
if y > max:
    max = y
if z > max:
    max = z
print("Largest number is ", max)
```

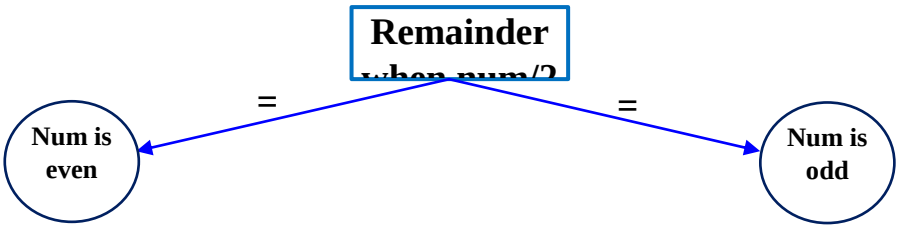
Example: 48 → Write a program that takes a number and checks whether the given number is off or even.

Algorithm:



Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Decision Tree:



Pseudocode:

```
Input num
    If num divide by 2 is equal to 0, then
        Display num is even
    Else display num is odd.
```

Code In Python

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

CONDITIONAL STATEMENTS

Conditional statements are such constructs that allow program statements to be optionally executed, depending on the condition/situation of the program's execution. Important conational statements are: simple **`if`**, nested **`if`**, **`if`**...**`else`**, **`if`**...**`elif`**...**`else`**.

THE SIMPLE IF

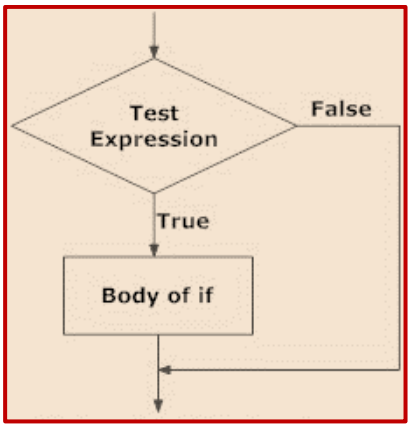
The simple **`if`** statement tests a particular condition/Boolean expression; if the condition evaluates to be true the block of statement(s) i.e., body of if is executed otherwise (condition evaluates to be false) execution is ignored.
The general form of if statement along with flow control diagram is worth seeing.

```
if condition:
    block
```

① **Reserved word if begins the if statement.**

② **Condition is Boolean expression followed by colon(:)**

③ **The block is set of statements to be**



Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 49 → Illustration of **if** statement.

```
# File Name: passgrade.py | Coded by: A. Nasra
marks = eval(input('Enter your marks:'))
if marks >= 65:
    print('You obtained passing grade.')
```

```
# Sample run of the file passgrade.py
Enter your marks:74.8
You obtained passing grade.
```

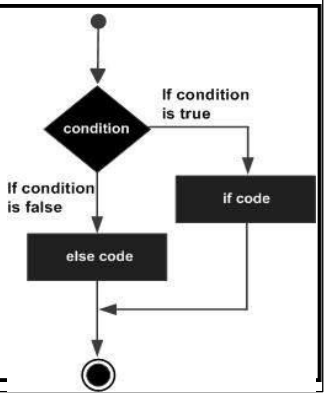
IF … ELSE STATEMENT

When **if** statement has an optional **else** clause that is executed only if the Boolean condition is false, the statement is known as **if … else** statement. The else block, like the if block, consists of one or more statements indented to the same level.

```
if condition :
    if_block
else:
    else_block
```

① **The reserved word if begins if … else statement.**

② **The condition is a Boolean expression that determines whether or not if block or else block will be executed. A colon (:) must follow the condition.**



Example: 50 → Program to find the division of two numbers.

```
# File Name: division.py | Coded by: A. Nasra
a , b = eval(input('Enter two numbers: '))
if b != 0:
    print(a,'/',b,'=',a/b)
else:
    print('Division by zero is not allowed')
```

```
# Sample run of the file division.py
# Sample run - 1
Enter two numbers: 456, 17
456 / 17 = 26.823529411764707
# Sample run - 2
Enter two numbers: 17, 0
Division by zero is not allowed
```

NESTED IF

When under if or else or both some if statement is used, it is known as nested if. This concept be understood by the following examples.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 51 → Illustration of **nested if**.

```
# File Name: in_range.py | Coded by: A. Nasra
value = int(input('Enter an integer value in the range
of 0 to 10: '))
if value >= 0:
    if value <= 10:
        print("Value entered is in range.")
    else:
        print("Value entered isn\'t in range.")
print("Done!")
```

```
# Sample run of in_range.py..
# Sample run - 1
Enter an integer value in the range of 0 to 10: 7
Value entered is in range.
Done!
# Sample run - 2
Enter an integer value in the range of 0 to 10: 11
Value entered isn't in range.
Done!
Enter an integer value in the range of 0 to 10: -3
Done!
```

Now, see the improved version of the same program in following program.

Example: 52 → Illustration of **nested if**.

```
# File Name: in_range1.py | Coded by: A. Nasra
value = int(input('Enter an integer value in the range
of 0 to 10: '))
if value >= 0:
    if value <= 10:
        print(value,"is in range.")
    else:
        print(value,"is too large (greater than 10)")
else:
    print(value,"is too small (less than 0)")
print('Done!')
```

```
# Sample run of file in_range1.py
# Sample run - 1
Enter an integer value in the range of 0 to 10: 7
7 is in range.
Done!
# Sample run - 2
Enter an integer value in the range of 0 to 10: 17
17 is too large (greater than 10)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Done!
# Sample run - 3
Enter an integer value in the range of 0 to 10: -7
-7 is too small (less than 0)
Done!
```

Example: 53 → Write a program to print roots of a quadratic equation:
$ax^2 + bx + c = 0$ (a != 0)

```
#File Name: root.py | Script by: A. Nasra
#Prog. to calculate and print roots of quadratic equation:
  ax^2 + bx + c = 0 (a not equal to 0)
from math import sqrt
import cmath   #for complex roots
print("Enter the co-efficient of Quadratic Eq ax**2+bx+c ")
a = eval(input(" a = "))
b = eval(input(" b = "))
c = eval(input(" c = "))
if a = 0:
    print("Value of a must not be 0")
else:
    delta = b*b-4*a*c
    if delta > 0:
        x1 = (-b+sqrt(delta))/2*a
        x2 = (-b-sqrt(delta))/2*a
        print("Roots are REAL & UNEQUAL")
        print("Root1 =",x1," Root2 =",x2)
    elif delta == 0:
        x1 = x2 = -b/2*a
        print("Roots are REAL & EQUAL")
        print("Root1 =",x1," ; ","Root2 =",x2)
    else:
        x1 = -b/2*a + cmath.sqrt(delta)
        x2 = -b/2*a - cmath.sqrt(delta)
        print("Roots are IMAGINARY & COMPLEX")
        print("Root1 =",x1," Root2 =",x2)
```

```
#Sample run-1
Enter the co-efficient of Quadratic Eq ax**2+bx+c
 a = 6
 b = 10
 c = 4
Roots are REAL & UNEQUAL
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Root1 = -24.0 ;  Root2 = -36.0
#Sample run-2
Enter the co-efficient of Quadratic Eq ax**2+bx+c
 a = 4
 b = 8
 c = 4
Roots are REAL & EQUAL
Root1 = -16.0 ;  Root2 = -16.0
# Sample run-3
Enter the co-efficient of Quadratic Eq ax**2+bx+c
 a = 1
 b = 2
 c = 3
Roots are IMAGINARY & COMPLEX
Root1   =   (-1+1.4142135623730951j)     Root2   =   (-1-
1.4142135623730951j)
```

IF … ELIF … ELSE

The **if**…**elif**…**else** statement is valuable for selecting accurately one block of code to execute from several different options. The **if** part of an **if**…**elif**…**else** statement is mandatory. The **else** part is optional. After the **if** part and before **else** part (if present) we may use as many **elif** blocks as necessary.

Example: 54 → Illustration of **if** … **elif** … **else** statement.

```
# File name: number2day.py
day = int(input('Enter the day as number (0 to 6): '))
if day == 0:
    print('Sunday')
elif day == 1:
    print('Monday')
elif day == 2:
    print('Tuesday')
elif day == 3:
    print('Wednesday')
elif day == 4:
    print('Thrusday')
elif day == 5:
    print('Friday')
elif day == 6:
    print('Saturday’)
else:
    print('Enter the correct day as number.')
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run of the file number2day.py\
# Samplr run - 1
Enter the day as number (0 to 6): 5
Friday
# Sampler run - 2
Enter the day as number (0 to 6): 10
Enter the correct day as number.
```

Example: 55 → Write a program that reads two numbers and a arithmetic operator and calculates the computed result.

```
# File: calculator.py | Script by: A. Nasra
n1 = eval(input("Enter first number: "))
op = input("Enter an [+ - * / %]: ")
n2 = eval(input("Enter second number: "))
result = 0
if op == '+':
    result = n1 + n2
elif op == '-':
    result = n1 - n2
elif op == '*':
    result = n1 * n2
elif op == '/':
    result = n1 / n2
elif op == '%':
    result = n1 % n2
else:
    print("Invalid operator!")
print(n1, op, n2, ' = ',result)
```

```
# Sample run of calculator.py
Enter first number: 15
Enter an operator [+ - * / %]: /
Enter second number: 6
15 / 6  =  2.5
```

Example: 56 → Write a program to print whether a given character is an uppercase or a lovercase character or a dight or a special character.

```
# File Name: char1.py | Script by: A. Nasra
ch = input("Enter a single character ")
if ch >= 'A'and ch<= 'Z':
    print("You have entered an Upper character.")
elif ch >= 'a'and ch<= 'z':
    print("You have entered a Lower character.")
elif ch >= '0'and ch<= '9':
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
    print("You have entered a Digit.")
else:
    print("You have entered a Special character.")
```

```
# Sample run of char1.py
Enter a single character %
You have entered a Special character.
```

Example: 57 →Write a program that reads three numbers inputted by the user and print them in ascending order.

```
#File Name: 3num_asc.py  |  Script by: A. Nasra
#Three numbers in ascending order
x = eval(input("Enter first number: "))
y = eval(input("Enter second number: "))
z = eval(input("Enter third number: "))
if x<y and x<z:
    if y<z:
        big, bigger, biggest = x, y, z
    else:
        big, bigger, biggest = x, z, y
elif y<x and y<z:
    if y<z:
        big, bigger, biggest = y, x, z
    else:
        big, bigger, biggest = y, z, x
else:
    big, bigger, biggest = z, y, x
print("Numbers   in   ascending   order:",big,bigger,
biggest)
```

```
#Sample run of 3num_asc.py
Enter first number: 9
Enter second number: 6
Enter third number: 11
Numbers in ascending order: 6 9 11
```

ITERATION

Iteration repeats/loops the execution of a sequence of code. Iteration is useful for solving many programming problems. Important iterations are: while statement, for statements,

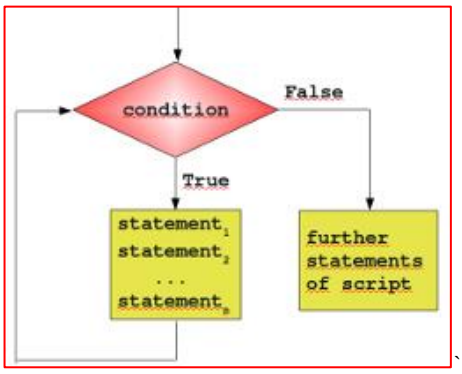
THE WHILE LOOP

The while loop is the most general looping statement in Python. it consists of the word while, followed by an expression that is interpreted as a true or false result, followed by a nested block of code that is repeated while the test at the top is true.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

The general form of while loop is as given below.

```
while condition :
    block
```



Example: 58 → Illustration of **while** statement.

```
# File Name: table.py | Code by: A. Nasra
x = eval(input('Table of (enter number): '))
n = 1
while n <= 10:
    print(x,'\u00D7',n,'=',x*n)
    n = n + 1
```

```
# Sample run of the file table.py
Table of (enter number): 77
77 × 1 = 77
77 × 2 = 154
77 × 3 = 231
77 × 4 = 308
77 × 5 = 385
77 × 6 = 462
77 × 7 = 539
77 × 8 = 616
77 × 9 = 693
77 × 10 = 770
```

Example: 59 → Illustration of Fibonacci series.

```
# File Name: fibonacci.py | Code written by: A. Nasra
n = eval(input('Enter no. of terms:'))
a, b = 0, 1
i = 1
print(a, end=' ')
while i < n :
    print(b, end=' ')
    a, b = b, a+b
    i = i+1
# IMPORTANT NOTES:
'''The last line is very remarkable.
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
    Writing in one line means a = b and b = a+b
    simultaneously. Whereas,
    a = b
    b = a+b
    means, firstly a = b, then b = a+b.'''
```

```
# Sample run of the file fibonacci.py
Enter no. of terms:10
0 1 1 2 3 5 8 13 21 34
```

Example: 60 → Write a program to find factorial of a number using while loop.

```
# program to find factorial of anumber using while loop
# File Name: fact_while.py  |  Script by: A. Nasra
num = eval(input('Enter a number: '))
fact = 1
i = 1
while i <= num:
    fact *= i
    i += 1
print("Factorial of",num,"=",fact)
```

```
# Sample run of fact_while.py
Enter a number: 6
Factorial of 6 = 720
```

Example: 61 → Write a program to calculate and print the sums of even and odd integers of the first n natural numbers.

```
#Program to calculate the sum of even and odd integers
of the first n natural numbers.
#File Name: even_odd_sum.py  |  Script by: A. Nasra
n = int(input('Upto which natural number? '))
even_sum = 0
odd_sum = 0
i = 1
while i<= n:
    if i%2 == 0:
        even_sum += i
    else:
    odd_sum += i
    i += 1
print("Sum of even natural numbers =",even_sum)
print("Sum of odd natural numbers =",odd_sum)
```

```
# Sample run of even_odd_sum
Upto which natural number? 11
Sum of even natural numbers = 30
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Sum of odd natural numbers = 36
```

Example: 62 → Write a program to input some numbers repeatedly and print their sum. The program ends when the user opts no to enter further numbers.

```
#Program to input some numbers repeatedly and print
their sum
#File Name: sum_num.py  |  Code by: Akhlaque Nasra
Sum = 0
ans = 'y'
while ans == 'y' or ans =='Y':
    num = eval(input('Enter number: '))
    Sum = Sum + num
    ans = input('Want ot enter more number? (y/n)')
print('Sum of numbers entered by you =',Sum)
```

```
# Sample run of sum_num.py
Enter number: 25
Want ot enter more number? (y/n)y
Enter number: -10
Want ot enter more number? (y/n)Y
Enter number: 5
Want ot enter more number? (y/n)n
Sum of numbers entered by you = 20
```

NESTED WHILE LOOP

Nested while loop means, while loop within while loop. Let us understand by an example of scripting multiplication table (from 1 to 10 =).

Example: 63 → Ca (File Name: **mult_table.py**)

```
# File Name: mult_table.py | Code by: A. Nasra
print('\t\t\tMULTIPLICATION TABLE')
i = 1
while i <= 10:
    print('\nTable of',i)
    n = 1
    while n <= 10:
        print(i*n, end=' | ')
        n += 1
    i += 1
```

```
# Sample run of the file mult_table.py
                MULTIPLICATION TABLE
Table of 1
1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
Table of 2
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
Table of 3
3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
Table of 4
4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
Table of 5
5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
Table of 6
6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
Table of 7
7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
Table of 8
8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
Table of 9
9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
Table of 10
10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
```

Important Notes: When the number of iteration may be determined by the inspection of the code of a loop (for or while or any other loop), the loop is called ***definite loop***, and when the number of iteration may not be determined, the loop is called ***indefinite loop***.

RANGE FUNCTION

The general form of the range function is

```
range(begin, end, step)
```

Where,

- **`begin`** is the first value in the range; if omitted, the default value is 0
- **`end`** is one past the last value in the range; the end value may not be omitted
- **`step`** is increment or decrement; if omitted the defaults value is 1
- **`begin`**, **`end`**, and **`step`** must all be integer values; floating-point values and other types are not allowed.

The following examples show how range can be used to produce a variety of sequences:

- range(10) → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- range(1, 10) → 1, 2, 3, 4, 5, 6, 7, 8, 9
- range(1, 10, 2) → 1, 3, 5, 7, 9
- range(10, 0, -1) → 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
- range(10, 0, -2) → 10, 8, 6, 4, 2
- range(2, 11, 2) → 2, 4, 6, 8, 10
- range(-5, 5) → −5, −4, −3, −2, −1, 0, 1, 2, 3, 4
- range(1, 2) → 1

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

- range(1, 1) → (empty)
- range(1, -1) → (empty)
- range(1, -1, -1) → 1, 0
- range(0) → (empty)

MEMBERSHIP OPERATORS

Membership operators are operators used to validate the membership of a value. It test for membership in a sequence, such as strings, lists, or tuples. The operators **in** and **not in** are important membership operators.

The **in** operator is used to check if a value exists in a sequence or not. It evaluates to **true** if it finds a variable in the specified sequence and **false** otherwise.

The **not in** operator evaluates to **true** if it does not finds a variable in the specified sequence and **false** otherwise.

Example: 64 → Ca

```
#File Name: in_op.py  |  Code by: A. Nasra
a = eval(input("Enter a number: "))
list = [1, 2, 3, 4, 5, 6, 7]
if(a in list):
    print(a,"is available in the given list.")
else:
    print(a,"is not available in the given list.")
```

Example: 65 →

```
#File Name: not_in_op.py  |  Code by: A. Nasra
a = eval(input("Enter a number: "))
list = [1, 2, 3, 4, 5, 6, 7]
if (a not in list):
    print(a,"is not available in the given list.")
else:
    print(a,"is available in the given list.")
```

IDENTITY OPERATORS Identity operators are used to determine whether a value is of a certain class or type. They are usually used to determine the type of data certain variable contains. The operators **is** and **is not** are important identity operators.

Example: 66 →

```
#File Name: is_op.py  |  Code by: A. Nasra
a = eval(input("Enter a number: "))
b = eval(input("Enter another number: "))
if(a is b):
    print(a,"and",b, "have same identity.")
else:
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
    print(a,"and",b, "have not same identity.")
```

Example: 67 →

```
#File Name: is_not_op.py  |  Code by: A. Nasra
a = eval(input("Enter a number: "))
b = eval(input("Enter another number: "))
if(a is not b):
    print(a,"and",b, "have not same identity.")
else:
    print(a,"and",b, "have same identity.")
```

FOR STATEMENTS

The **for** statement iterates over a range of values. These values can be a numeric range, or, as elements of a data structure like a string, list, or tuple.

Example: 68 → Write a program to find the sum of n natural numbers, where the value of n is given by the user.

```
# File name: sum1.py  |  Code by: Akhlaque Nasra
# sum of n nutural numbers i.e. 1 + 2 + 3 + ... + n
n = eval(input('Enter a number upto which you want sum:
'))
sum = 0
for i in range(1,n+1):
    sum += i
print('sum = 1+2+3+...+',n,'=',sum)
```

Example: 69 → (File Name: **for_loop+1.py**)

```
# File: for_loop_1.py | Script by: A. Nasra
for i in range(3):
    print(i, 'Pythons')
```

```
# Sample run of for_loop_1.py
0 Pythons
1 Pythons
2 Pythons
```

Example: 70 → (File Name: **for_loop.py**)

```
# File: for_loop_2 | Script by: A. Nasra
for n in range(21, 0, -3):
    print(n, '', end='')
```

```
# Sample run of the file for_loop.py
21 18 15 12 9 6 3
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 71 → (File Name: `for_loop.py`)

```
#Fibonacci series
#File Name: fib2.py | Code by: A. Nasra
i = int(input("Enter number of terms: "))
a, b = 0, 1
print('Fibonnaci series for',i,'terms is:')
for i in range(0,i,1):
    print(a, end=' ')
    a, b = b, a+b
```

```
# Sample run of fib2.py
Enter number of terms: 9
Fibonacci series for 9 terms is:
0 1 1 2 3 5 8 13 21
```

Example: 72 →Write a program to input a number and test if it is a prime number.

```
#File Name: prime.py  |  Coded by: A. Nasra
#Program to check if a number is prime number.
#File Name: prime.py  |  Coded by: A. Nasra
#Prime numbers are the numbers are the numbers that
can't be devided by any number except 1 and itself.
num = int(input("Enter a number: "))
k = 0
for i in range(2,num//2+1):
    if num%i == 0:
        k = k + 1
if(k <= 0):
    print(num,"is prime")
else:
    print(num,"isn't prime")
```

Example: 73 → Write a program to input a number as many times as user needs and test if it is a prime number.

```
#Program to check if a number is prime number.
#File Name: prime_improved.py  |  Coded by: A. Nasra
ans = 'y'
while ans=='y' or ans== 'Y':
    num = int(input("Enter a number: "))
    k = 0
    for i in range(2,num//2+1):
        if num%i == 0:
            k = k + 1
    if(k <= 0):
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
        print(num,"is prime")
    else:
        print(num,"isn't prime")
    ans = input('Do you wnat to test more? (y/n)...')
```

```
#Sample run of prime_improved.py
Enter a number: 45
45 isn't prime
Do you want to test more? (y/n)...y
Enter a number: 69
69 isn't prime
Do you want to test more? (y/n)...y
Enter a number: 67
67 is prime
Do you want to test more? (y/n)...n
```

NESTED FOR LOOPS

Nested **for** loop means, **for** loop within a **for** loop.

Example: 74 → Write a program to output following pattern,

```
*
* *
* * *
* * * *
```

```
# File name: pattern1.py   |  Code by: A. Nasra
for i in range(1,5):
    for j in range(1,i+1):
        print("* ",end="")
    print()
```

Example: 75 → Write a program to output following pattern,

```
1
1 3
1 3 5
1 3 5 7
```

```
# File name: pattern2.py   |  Code by: A. Nasra
for i in range(3,10,2):
    for j in range(1,i,2):
        print(j,end= ' ')
    print()
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 76 → Write a program to output following pattern,

```
4 3 2 1
4 3 2
4 3
4
```

```
# File name: pattern3.py  |  Code by: A. Nasra
for i in range(4):
    for j in range(4,i,-1):
        print(j,end= ' ')
    print()
```

Example: 77 → Write a program to output following pattern,

```
      *
    * *
  * * *
* * * *
```

```
# File name: pattern4.py  |  Code by: A. Nasra
k = 8
for i in range(0, 5):
    for j in range(0, k):
        print(end=" ")
    k = k - 2
    for j in range(0, i+1):
        print("* ", end="")
    print()
```

Example: 78 → Write a program to output following pattern,

```
    *
  * * *
* * * * *
```

```
# File name: pattern5.py  |  Code by: A. Nasra
k = 0
rows = 4
for i in range(1, rows+1):
    for space in range(1, (rows-i)+1):
        print(end="  ")
    while k != (2*i-1):
        print("* ", end="")
        k = k + 1
    k = 0
    print()
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 79 → Write a program to output following pattern,

```
A
B C
D E F
G H I J
K L M N O
```

```
# File name: pattern6.py  |  Code by: A. Nasra
val = 65
for i in range(0, 5):
    for j in range(0, i+1):
        ch = chr(val)
        print(ch, end=" ")
        val = val + 1
    print()
```

example → 56 Write a program to output following pattern,

Example: 80 → CaWrite a program to output following pattern,

```
A
B B
C C C
D D D D
E E E E E
```

```
# File name: pattern7.py  |  Code by: A. Nasra
val = 65
for i in range(0, 5):
    for j in range(0, i+1):
        ch = chr(val)
        print(ch, end=" ")
    val = val + 1
    print()
```

Example: 81 → Write a program to output following pattern,

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# File Name: pattern8.py  |  Coded by: A. Nasra
n=5;
for i in range(n):
    for j in range(i):
        print ('* ', end="")
    print('')
for i in range(n,0,-1):
    for j in range(i):
        print('* ', end="")
    print('')
```

Example: 82 → Write a program to output diamond pattern of the diamond length given by the user.

```
# Program to print diamond
# File Name: pattern9.py  |  Code by: A. Nasra
side = int(input("Please input side length of diamond: "))
for x in list(range(side)) + list(reversed(range(side-1))):
    print('{: <{w1}}{:*<{w2}}'.format('', '', w1=side-x-1, w2=x*2+1))
```

```
# Sample run of the file pattern9.py
Please input side length of diamond: 4
   *
  ***
 *****
*******
 *****
  ***
   *
```

Example: 83 → (a) Write a Python script to read a number and reverse that number. (b) Write a Python script to read a string and reverse that string.

```
# Reversing a number or string
# File Name: reverse1.py  |  Script by: A. Nasra
a=input('Enter a string or a number : ')
print('Reverse of ',a,'is\u2B95 ',end='')
n = len(a)
for i in range(-1,(-n-1),-1):
    print(a[i],end='')
```

```
# Sample run of the file reverse1.py
# Sample run-1
Enter a string or a number: interesting
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Reverse of interesting is➞ gnitseretni
# Sample run-2
Enter a string or a number: 98786
Reverse of  98786 is➞ 68789
```

Example: 84 → Write a program to produce the table from 1 to 10.

```
# Print a multiplication table from 1 to 10.
# File Name: for_loop3.py | Script by: A. Nasra
# Print column heading
print("       1   2   3   4   5   6   7   8   9  10")
print("    +-----------------------------------------
")
for row in range(1, 11): # 1<=row<=10,table has 10 rows
    if row < 10:         # Need to add space?
        print(" ", end="")
    print(row, "| ", end="") #Print heading for row.
    for column in range(1, 11): # Table has 10 columns.
        product = row*column; # Compute product
        if product < 100: # Need to add space?
            print(end=" ")
        if product < 10: # Need to add another space?
            print(end=" ")
        print(product, end=" ") # Display product
    print()  # Move cursor to next row
```

```
# Sample run of for_loop3.py
      1   2   3   4   5   6   7   8   9  10
   +-----------------------------------------
 1 |   1   2   3   4   5   6   7   8   9  10
 2 |   2   4   6   8  10  12  14  16  18  20
 3 |   3   6   9  12  15  18  21  24  27  30
 4 |   4   8  12  16  20  24  28  32  36  40
 5 |   5  10  15  20  25  30  35  40  45  50
 6 |   6  12  18  24  30  36  42  48  54  60
 7 |   7  14  21  28  35  42  49  56  63  70
 8 |   8  16  24  32  40  48  56  64  72  80
 9 |   9  18  27  36  45  54  63  72  81  90
10 |  10  20  30  40  50  60  70  80  90 100
```

ABNORMAL LOOP TERMINATION

Normally, a while statement executes until its condition becomes false. Sometimes, however, it is desirable to immediately exit the body of the loop or recheck the condition from some specific stage of the loop. For this purpose,

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Python provides the **break** and **continue** statements to give programmers more flexibility designing the control logic of loops.

BREAK STATEMENTS

Python provides the **break** statement is used to jump out of closest enclosing loop (**for** or **while** loop). The break statement causes the immediate exit from the body of the loop.

Example: 85 → (File Name: **break1.py**)

```
# File Name: break1.py | Script by: A. Nasra
a = eval(input("Enter a number upto which Prime Numbers are to be searched: "))
for n in range(2, a):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'isn\'t prime, because', n, ' = ', x, '*', n//x)
            break
    else:
    # loop without finding a factor
        print(n, 'is a prime number')
```

```
Enter a number upto which Prime Numbers are to be searched: 7
2 is a prime number
3 is a prime number
4 isn't prime, because 4 = 2 * 2
5 is a prime number
6 isn't prime, because 6 = 2 * 3
```

CONTINUE STATEMENTS

When a continue statement is encountered within a loop, the remaining statements within the body are skipped, but the loop condition is checked to see if the loop should continue or be exited. If the loop's condition is still true, the loop is not exited, but the loop's execution continues at the top of the loop.

Example: 86 → (File Name: **cont.py**)

```
# Ex.of continue: Finding even/odd numbers.
# File Name: cont.py  |  Script by: Akhlaque Nasra
x = eval(input("Enter a number upto where\neven and odd numbers is to be searched: "))
print("Even numbers are:")
for num in range(1, x):
    if num % 2 == 0:
        print(num,end="  ")
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
        continue
print("\nOdd numbers are:")
for num in range(1,x):
    if num % 2 != 0:
        print(num,end="  ")
        continue
```

Example: 87 → (File Name: **cont2.py**)

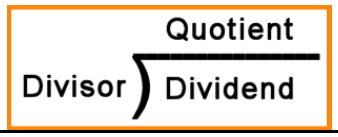
```
# Ex. of continue. | Script by: Akhlaque Nasra
for letter in 'Python for smart students':
    if letter == 's' or letter == 't':
        continue
    print(letter,end=' ')
```

Example: 88 → Numbers in the form of $2^n - 1$ are called Mersenne Numbers. In other words, Mersenne No = 2^1-1, 2^2-1, 2^3-1, 2^4-1, 2^5-1, 2^6-1 … … 2^n-1
= 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, … … &co.
Write a Python Script to display Mersenne Numbers up to the term user desires.

```
# File Name: merse.py  |  Script By: A. Nasra
n = eval(input('How many Mersenne No's you want?..'))
for i in range(1,n+1):
    print(2**i - 1, end=' ')
print()
```

```
# Sample run of merse.py
How many Mersenne Numbers you want?...10
1 3 7 15 31 63 127 255 511 1023
```

Example: 89 → Write a program to input two numbers and print their LCM (Least Common Multiple) and GCD or HCF (Greatest Common Divisor).




```
# Program to find LCM and GCD (HCF)
# File Name: lcm_gcd.py  |  Script By: A. Nasra
x = eval(input('First number  = '))
y = eval(input('Second number = '))
if x>y:
    divisor = y
else:
    divisor = x
for i in range(1,divisor+1):
    if x%i==0 and y%i==0:
        gcd = i
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
lcm = x*y/gcd
print('LCM of',x,'and',y,'=',lcm)
print('GCD of',x,'and',y,'=',gcd)
```

```
# Sample run of lcm_gcd.py
First number  = 123
Second number = 72
LCM of 123 and 72 = 2952.0
GCD of 123 and 72 = 3
```

Example: 90 → Write a Python script to find the sum of following series,
S = (1) + (1 + 2) + (1 + 2 + 3) + (1 + 2 + 3 + 4) + … … + (1 + 2 + 3 + … + n)

```
'''Python Script to find the Sum of n terms of the
   series,
   (1)+(1+2)+(1+2+3)+ ...+(1+2+3+...+n))'''
# File Name: sum_sum.py  |  Script by: A. Nasra
n = eval(input('Enter number of terms:'))
def in_sum(n):
    Sum = 0
    for i in range(1,n+1):
        Sum = Sum + i
    return Sum

s = 0
for j in range(1,n+1):
    s = s + in_sum(j)
print('Sum = ',s)
```

```
# Sample run of sum_sum.py
Enter number of terms:5
Sum =  35
```

Example: 91 → Write a Python program to find the sum of the following series up to n terms.

$$\frac{2}{9}-\frac{5}{13}+\frac{8}{17}-\frac{11}{21}+\cdots\cdots \; upto\; n\; terms$$

Hint: $n^{th}\ term\ of\ series = \frac{(-1)^{n+1}\times(3n-1)}{4n+1}$

```
# File Name: sum_series1.py  |  Script by: A. Nasra
n = eval(input("Input number of terms: "))
Sum = 0
for i in range(1,n+1):
    Sum = Sum + pow(-1,i+1)*(3*i-1)/(4*i+5)

print('Sum of Series for',n,'terms = ',Sum)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run of sum_series1.py
Input number of terms: 4
Sum of Series for 4 terms =  -0.21561445090856862
```

INFINITE LOOP

An infinite loop is a loop that executes its block of statements repeatedly until the user forces the program to Quit. Infinite loops are sometimes designed.
For example, a long-running server application like a Web server may need to continuously check for incoming connections.

Example: 92 → (File Name: **infin_1.py**)

```
#Example of infinite loop.
#File Name: infin_1.py  |  Script by: A. Nasra
x = 1
while True:
    print(x,'Your system is infected with VIRUS!!\n')
    x += 1
```

To deal with such infinite loop, we can modify the above program as given below:

```
#Example of infinite loop.
#File Name: infin_1.py  |  Script by: A. Nasra
x = 1
while True:
    print(x,'Your system is infected with VIRUS!!\n')
    x += 1
    if a > 100:
        break
```

Example: 93 → (File Name: **infin_2.py**)

```
#File Name: infin_2.py  |  Coded by: A. Nasra
print('\u00A9A.Nasra')
var = 1
while var == 1 :  # This constructs an infinite loop
   num = input("Enter a number:")
   print("You entered: ", num)
print("Good bye!")
```

Example: 94 → **(File Name: infin_2.py)**

```
# File Name: infin_3.py  |  Code by: A. Nasra
from itertools import cycle
for t in cycle(range(0, 4)):
    print(t)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

STRINGS AND ITS MANIPULATION

STRING

According to the Python language, string can be defined as the sequence of characters enclosed between single, double or tipple quotation marks. Python strings are immutable. Strings are used to store text type data.

CREATING STRINGS

Single or double quotes are used for single line strings. The triple quotes are used for multiline strings.

Example: 95 →

```
>>> S = 'Hi Students'
>>> T = "Hi Students"
>>> U = '''Hi Students of Computer Science,
you all must be smart and your capacity to
use your mind must be more than normal studnets.'''
>>> print(S)
Hi Students
>>> print(T)
Hi Students
>>> print(U)
Hi Students of Computer Science,
you all must be smart and your capacity to
use your mind must be more than normal studnets.
```

EMPTY STRING

Empty string is a string that has no characters. It means empty string has only pairs of quotation marks.

Example: 96 →

```
>>> empty_string = ''
>>> print(empty_string)
                          # No output
>>>
```

INDEXING THE STRING

Each character of a string has a unique position-id (index).the indexes of a string starts for 0 to (string_length – 1) in forward direction. Strings can also be indexed from -1 to string_length in backward direction.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

String G	P	y	t	h	o	n		g	a	m	e
Positive Index	0	1	2	3	4	5	6	7	8	9	10
Forward Direction											
Negative Index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
Backward Direction											

Example: 97 →

```
#Example of Indexing the String
#File Name: str_index.py  |  Script by: A. Nasra
G = 'Python game'
print('G[0]  =',G[0],'\tG[5]   =',G[5])
print('G[9]  =',G[9],'\tG[-1]  =',G[-1])
print('G[-7] =',G[-7],'\tG[-11] =',G[-11])
```

```
# Sample run of str_index.py
G[0]  = P    G[5]   = n
G[9]  = m    G[-1]  = e
G[-7] = o    G[-11] = P
```

STRING SLICING

A slice is used to pick out part of a string. In other words, slice is a sequence of characters within an original string. The basic structure of string slice is:

```
String_name[start location : end location+1]
```

Example: 98 → Let us consider the following string

String G	P	y	t	h	o	n		g	a	m	e	!
Positive Index	0	1	2	3	4	5	6	7	8	9	10	11
Negative Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
# Examples of string slice
# File Name: str_slice.py  |  Script by: A. Nasra

G = 'Python game!'
print(' 1. G[6:11]\t',G[6:11])
print(' 2. G[0:11]\t', G[0:11])
print(' 3. G[:0]\t', G[:0])
print(' 4. G[:5]\t', G[:5])
print(' 5. G[7:]\t', G[7:])
print(' 6. G[-4:-1]\t', G[-4:-1])
print(' 7. G[5:-4]\t', G[5:-4])
print(' 8. G[-4:5]\t', G[-4:5])
print(' 9. G[-4:]\t', G[-4:])
print('10. G[:-4]\t', G[:-4])
print('11. G[:]\t', G[:])
```

```
 # Sample run of str_slice.py
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
 1. G[6:11]     game
 2. G[0:11]    Python game
 3. G[:0]
 4. G[:5]      Pytho
 5. G[7:]      game!
 6. G[-4:-1]   ame
 7. G[5:-4]    n g
 8. G[-4:5]
 9. G[-4:]     ame!
10. G[:-4]     Python g
11. G[:]       Python game!
```

STRIDE WHILE SLICING STRING

There is an optional third argument, just like in the **range** statement, that can specify the step. This parameter specifies the stride (stepping forward), which refers to how many characters to move forward after the first character is retrieved from the string. By default the third parameter in slicing, i.e. stride is 1.

Example: 99 → Let us consider the following string

String G	P	y	t	h	o	n		g	a	m	e	!
Positive Index	0	1	2	3	4	5	6	7	8	9	10	11
Negative Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
# Example of Slicing a string with stride
# File Name: stride.py | Script by: A. Nasra
G = 'Python game'
print('1. G[0:11:1]\t', G[0:11:1])
print('2. G[0:12:2]\t', G[0:12:2])
print('3. G[0:12:4]\t', G[0:12:4])
print('4. G[::4]\t', G[::4])
print('5. G[::-1]\t', G[::-1])
print('6. G[::-2]\t', G[::-2])
```

```
# Sample run of stride.py
1. G[0:11:1]   Python game
2. G[0:12:2]   Pto ae
3. G[0:12:4]   Poa
4. G[::4]      Poa
5. G[::-1]     emag nohtyP
6. G[::-2]     ea otP
```

CHANGING STRINGS

We know that the strings are *immutable*, that means we cannot change a string in place by assigning new value to an index, as is shown in the following example,

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 100 → Changing string.

```
>>> s = "Copyright"
>>> s[3] = 'f'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    s[3] = 'f'
TypeError: 'str' object does not support item assignment
```

To change a string, we need to build and assign a new string with the help of concatenation and slicing, and then, if desired, assign the result back to the string's original name, as is shown in the following example,

Example: 101 → Changing string.

```
>>> s = "Copyright"
>>> s = s + ''
>>> s
'Copyright'
>>> s = s[:4] + 'f' + s[5:]
>>> s
'Copyfight'
```

We can realize similar effects with string method replace(), as is shown below,

Example: 102 → Changing string.

```
>>> s = 'Copyright'
>>> s = s.replace('r','f')
>>> s
'Copyfight'
```

STRING UNICODE VALUES

Unicode is a new universal coding standard adopted by all new platforms. It is promoted by Unicode Consortium which is a non-profit organization. Unicode provides a unique number for every character irrespective of the platform, program and the language. *In Python 3.X*, the normal str string handles Unicode text (including ASCII, which is just a simple kind of Unicode).

Example: 103 →

```
# Unicode examples | File Name: unicode.py | Script: A. Nasra
print(' 1. 00A7 \u21E8 \u00A7\t 2. 00D7 \u21E8 \u00D7\t 3. 00F7 \u21E8 \u00F7\t 4. 00F8 \u21E8 \u00F8')
print(' 5. 00A9 \u21E8 \u00A9\t 6. 00AE \u21E8 \u00AE\t 7. 0283 \u21E8 \u0285\t 8. 0302 \u21E8 A\u0302')
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
print(' 9. 0342 \u21E8 \u0342B\t10. 0342 \u21E8 \u0342\t11.
0303 \u21E8 \u0303C\t12. 0304 \u21E8 \u0304D')
print('13. 030A \u21E8 E\u030A\t14. 03C0 \u21E8 \u03C0\t15.
03A0 \u21E8 \u03a0\t16. 03B1 \u21E8 \u03B1')
print('17. 03B2 \u21E8 \u03B2\t18. 03B3 \u21E8 \u03B3\t19.
03A3 \u21E8 \u03A3\t20. 03B8 \u21E8 \u03B8')
print('21. 03A9 \u21E8 \u03A9\t22. 2020 \u21E8 \u2020\t23.
2022 \u21E8 \u2022\t24. 2023 \u21E8 \u2023')
print('25. 2043 \u21E8 \u2043\t26. 205D \u21E8 \u205D\t27.
205E \u21E8 \u205E\t28. 2103 \u21E8 \u2103')
print('29. 2109 \u21E8 \u2109\t30. 212B \u21E8 \u212B\t31.
2192 \u21E8 \u2192\t32. 21D2 \u21E8 \u21D2')
print('33. 21A3 \u21E8 \u21A3\t34. 21E8 \u21E8 \u21E8\t35.
2605 \u21E8 \u2605\t36. 2606 \u21E8 \u2606')
print('37. 2615 \u21E8 \u2615\t38. 261B \u21E8 \u261B\t39.
261E \u21E8 \u261E\t40. 2620 \u21E8 \u2620')
print('41. 2660 \u21E8 \u2660\t42. 2661 \u21E8 \u2661\t43.
2662 \u21E8 \u2662\t44. 2663 \u21E8 \u2663')
print('45. 2664 \u21E8 \u2664\t46. 2665 \u21E8 \u2665\t47.
2666 \u21E8 \u2666\t58. 2667 \u21E8 \u2667/')
print('49. 2702 \u21E8 \u2702\t50. 2704 \u21E8 \u2704\t51.
270D \u21E8 \u270D\t52. 279E \u21E8 \u279E')
print('53. 2600 \u21E8 \u2600\t54. 2601 \u21E8 \u2601\t55.
2602 \u21E8 \u2602\t56. 2616 \u21E8 \u2616')
print('57. 2460 \u21E8 \u2461\t58. 2462 \u21E8 \u2462\t59.
2463 \u21E8 \u2463\t60. 2464 \u21E8 \u2464')
print('61. 2465 \u21E8 \u2465\t62. 2466 \u21E8 \u2466\t63.
2467 \u21E8 \u2467\t64. 2468 \u21E8 \u2468')
print('65. 2469 \u21E8 \u2469\t66. 246A \u21E8 \u246A\t67.
246B \u21E8 \u246B\t68. 246C \u21E8 \u246C')
print('69. 246D \u21E8 \u246D\t70. 246E \u21E8 \u246E\t71.
246F \u21E8 \u246F\t72. 2470 \u21E8 \u2470')
print('73. 2471 \u21E8 \u2471\t74. 2472 \u21E8 \u2472\t75.
2473 \u21E8 \u2473')
```

```
# Sample run of the file unicode.py
1. 00A7 ⇨ §    2. 00D7 ⇨ ×   3. 00F7 ⇨ ÷   4. 00F8 ⇨ ø
5. 00A9 ⇨ ©    6. 00AE ⇨ ®   7. 0283 ⇨ ʃ   8. 0302 ⇨ Â
9. 0342 ⇨ ͂B 10. 0342 ⇨ ͂  11. 0303 ⇨ C̃ 12. 0304 ⇨ ¯D
13. 030A ⇨ E̊ 14. 03C0 ⇨ π 15. 03A0 ⇨ Π 16. 03B1 ⇨ α
17. 03B2 ⇨ β 18. 03B3 ⇨ γ 19. 03A3 ⇨ Σ 20. 03B8 ⇨ θ
21. 03A9 ⇨ Ω 22. 2020 ⇨ † 23. 2022 ⇨ • 24. 2023 ⇨ ‣
25. 2043 ⇨ ⁃ 26. 205D ⇨ ⁝ 27. 205E ⇨ ⁞ 28. 2103 ⇨ ℃
29. 2109 ⇨ ℉ 30. 212B ⇨ Å 31. 2192 ⇨ → 32. 21D2 ⇨ ⇒
33. 21A3 ⇨ ↣ 34. 21E8 ⇨ ⇨ 35. 2605 ⇨ ★ 36. 2606 ⇨ ☆
37. 2615 ⇨ ☕ 38. 261B ⇨ ☛ 39. 261E ⇨ ☞ 40. 2620 ⇨ ☠
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

41. 2660 ⇨ ♠ 42. 2661 ⇨ ♡ 43. 2662 ⇨ ♢ 44. 2663 ⇨ ♣
45. 2664 ⇨ ♤ 46. 2665 ⇨ ♥ 47. 2666 ⇨ ♦ 58. 2667 ⇨ ♔/
49. 2702 ⇨ ✂ 50. 2704 ⇨ ✄ 51. 270D ⇨ ✍ 52. 279E ⇨ ➞
53. 2600 ⇨ ☀ 54. 2601 ⇨ ☁ 55. 2602 ⇨ ☂ 56. 2616 ⇨ ☖
57. 2460 ⇨ ② 58. 2462 ⇨ ③ 59. 2463 ⇨ ④ 60. 2464 ⇨ ⑤
61. 2465 ⇨ ⑥ 62. 2466 ⇨ ⑦ 63. 2467 ⇨ ⑧ 64. 2468 ⇨ ⑨
65. 2469 ⇨ ⑩ 66. 246A ⇨ ⑪ 67. 246B ⇨ ⑫ 68. 246C ⇨ ⑬
69. 246D ⇨ ⑭ 70. 246E ⇨ ⑮ 71. 246F ⇨ ⑯ 72. 2470 ⇨ ⑰
73. 2471 ⇨ ⑱ 74. 2472 ⇨ ⑲ 75. 2473 ⇨ ⑳

Example: 104 → Devnagari in Unicode.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
090	◌ँ	◌ं	◌ः	ऄ	अ	आ	इ	ई	उ	ऊ	ऋ	ऌ	ऍ	ऎ	ए	
091	ऐ	ऑ	ऒ	ओ	औ	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
092	ठ	ड	ढ	ण	त	थ	द	द	न	ऩ	प	फ	ब	भ	म	य
093	र	ऱ	ल	ळ	ऴ	व	श	ष	स	ह	◌ऺ	◌ऻ	़	ऽ	◌ा	◌ि
094	◌ी	◌ु	◌ू	◌ृ	◌ॄ	◌ॅ	◌ॆ	◌े	◌ै	◌ॉ	◌ॊ	◌ो	◌ौ	◌्	◌॑	◌॒
095	ॐ	◌॓	◌॔	◌ॕ	◌ॖ	◌ॗ	◌क़	◌ख़	◌ग़	◌ज़	◌ड़	◌ढ़	◌फ़	◌य़		
096	ॠ	ॡ	◌ॢ	◌ॣ	।	॥	०	१	२	३	४	५	६	७	८	९
097	॰	ॱ	ॲ	ॳ	ॴ	ॵ	ॶ	ॷ	ॸ	ॹ	ॺ	ॻ	ॼ	ॽ	ॾ	ॿ

STRING CONSTANTS

A string constant (literal) is a sequence of characters surrounded by quotes. We can use both single, double and triple quotes for a string. And, a character constant is a single character surrounded by single or double quotes.

Let's discuss some interesting strings constants defined in string module, as is given in the following example,

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 105 →

```
# File Name: stringconst.py | Script by: A. Nasra
import string
print('\u2460                    string.ascii_uppercase
\u279E',string.ascii_uppercase)
print('\u2461                    string.ascii_lowercase
\u279E',string.ascii_lowercase)
print('\u2462                      string.ascii_letters
\u279E',string.ascii_letters)
print('\u2463 string.digits \u279E',string.digits)
print('\u2464                             string.hexdigits
\u279E',string.hexdigits)
print('\u2465                             string.octdigits
\u279E',string.octdigits)
print('\u2466                             string.printable
\u279E',string.printable)
print('\u2467string.punctuation
\u279E',string.punctuation)
```

```
# Sample run of stringconst.py
① string.ascii_uppercase ➞ ABCDEFGHIJKLMNOPQRSTUVWXYZ
② string.ascii_lowercase ➞ abcdefghijklmnopqrstuvwxyz
③  string.ascii_letters  ➞  abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
④ string.digits ➞ 0123456789
⑤ string.hexdigits ➞ 0123456789abcdefABCDEF
⑥ string.octdigits ➞ 01234567
⑦ string.printable ➞ 0123456789abcdefghijklmnopqrstuvwx
yzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>
?@[\]^_`{|}~
⑧string.punctuation ➞ !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

STRING OPERATORS

BASIC OPERATORS

The basic operators are + (concatenation) and * (replicator)

Example: 106 →

```
>>> print('Python'+'love')
Pythonlove
>>> 'Python'+'love'
'Pythonlove'
>>> print('Python'*3)
PythonPythonPython
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> 'Python'*3
'PythonPythonPython'
>>> 3*'Python'
'PythonPythonPython'
```

MEMBERSHIP OPERATOR

Membership operators are **in** and **not in**, that we have already studied in previous chapter.

STRING SLICE OPERATOR

The string slice operator **slice[m:n]** has aready been studied in the same chapter.

RANGE OPERATOR

The range operator **range(start, stop, step)** has already been studies in the same chapter.

COMPARISON OPERATOR

Standard comparison operators in Python are relational operator(<, <=, >, >=, ==, !=) that is also applicable for strings. The Python compares two strings throuth relational operators using character by character comparison. Internally the Python compares the strings using Unicode values (called ordinal value). Let us note the most common characters and their ordinal values given in the following table,

Characters	Ordinal Values
'0' to '9'	48 to 57
'A' to 'Z'	65 to 90
'a' to 'z'	92 to 122

Example: 107 →

```
>>> 'a' < 'A'  # Because in Python lowercase letter is
False             higher than uppercase letters
>>> 'abc' > 'ABC'
True
>>> 'aBc' > 'abc'
False
>>> 'abc' == 'ABC'
False
```

Python provides a built-in function ord() that takes a single character and returns the corresponding ordinal Unicode value. The opposite of ord() is char() that takes ord Unicode value and returns character.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 108 →

```
>>> ord('B')
66
>>> chr(66)
'B'
>>> hex(66)
'0x42'
>>> '\u0042'
'B'
```

STRING TRAVERSING

Accessing the individual characters of a string through index for manipulation of the characters, is called traversing the string. Thus, traversing refers to iterating through the elements of a string character by character.

Example: 109 →

```
# Example of string traversal
# File Name: str_traversal1.py | Coded by: A. Nasra
logo = 'Python love'
for ch in logo:
    print(ch,'-',end='')
```

```
# Output of flie str_traversal1.py
P -y -t -h -o -n -  -l -o -v -e -
```

Example: 110 → Write a program to read a string or a number and display it in reverse order.

```
# Reversing a number or string
# File Name: reverse1.py  |  Script by: A. Nasra
a = input('Enter a string or a number: ')
print('Reverse of ',a,'is\u2B95 ',end='')
n = len(a)
for i in range(-1,(-n-1),-1):
    print(a[i],end='')
```

```
# Sample run of reverse1.py
# Sample Run - 1
Enter a string or a number: something great
Reverse of  something great is➞ taerg gnihtemos
# Sample Run - 2
Enter a string or a number: 435628
Reverse of  435628 is➞ 826534
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 111 → Write a program to print the following pattern without using nested loop.

^
^ ^
^ ^ ^
^ ^ ^ ^
^ ^ ^ ^ ^

```
# Program for the pattern | Coded by: A. Nasra
a = '^ '
b = '' # empty string
for i in range(5):
    b = a + b
    print(b)
```

STRING METHODS AND BUILT-IN FUNCTIONS

CAPITALIZE( )

The syntax of **capitalize()** is: **string.capitalize()**

The **capitalize()** function doesn't take any parameter.

The **capitalize()** function returns a string with first letter capitalized. It doesn't modify the old string.

Example: 112 → Capitalize a sentence.

```
# Example of string.capitalize()
# File Name: capitalize.py | Coded by: A. Nasra
string = "python is a passion."
capitalized_string = string.capitalize()
print('Old String: ', string)
print('Capitalized String:', capitalized_string)
```

```
# Sample run of capitalize.py
Old String:  python is a passion.
Capitalized String: Python is a passion.
```

Example: 113 → Non-alphabetic first character.

```
# Example of string.capitalize()
# File Name: capitalize2.py | Coded by: A. Nasra
string = "+ is an operator."
new_string = string.capitalize()
print('Old String:', string)
print('New String:', new_string)
```

```
# Sample run of capitalize2.py
Old String: + is an operator.
New String: + is an operator.
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

COUNT( )

The string **count()** method searches the substring in the given string and returns how many times the substring is present in it.

The syntax of **count()** method is:

string.count(substring, start, stop)

Where,

substring is the string whose count is to be found.

start is the starting index within the string where search starts from. Its default value is 0.

stop ending index within the string where search stops. Its default value is: size of string – 1.

The **count()** method returns the number of occurrences of the substring in the given string.

Example: 114 → Counting occurrence of substring.

```
# Example of string.count()
# File Name: count1.py | Coded by: A. Nasra
string = "To be or not to be that is the question."
# Let substring to be count be 'be'
count = string.count('be')
print("The word \'be\' occurs",count,"times.")
```

```
# Sample run of count1.py
The word 'be' occurs 2 times.
```

Example: 115 → Counting occurrence of substring in a certain part of the string.

```
#Counting of substring within certain part of a string.
#File Name: count2.py | Script by: A. Nasra
string = "A friend in need is friend indeed."
# Let substring to be counted be "i"
count = string.count('i', 2, 28)
print("   The   \'i\'   comes",count,"times   in   the
statement\n",string)
```

```
# Sample run of the file count2.py
 The 'i' comes 5 times in the statement
 A friend in need is friend indeed.
```

FIND( )

The syntax of **find()** method is: **string.find(sub, start, end)**

Where, **sub** is the substring to be searched in the string.

The **start** and **end** (optional) is a search within **string[start : end]**

If substring exists inside the string, it returns the lowest index where substring is found. If substring doesn't exist inside the string, it returns -1.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 116 → Use of **string.find()** method.

```
# Example of string.find()
# File Name: find1.py  |  Program by: A. Nasra
String = input("Enter a line: ")
Sub=input("Enter the substring to find its location:")
x = string.find(sub)
print('The first location of',sub,'in the given line
is:',x)
y = string.find(sub,x+1,len(string))
print('The next location of',sub,'is:',y)
```

```
# Sample run of find1.py file.
# Sample Run - 1
Enter a line: Beauty lies in the eyes of the beholder.
Enter the substring to find its location: be
The first location of be in the given line is: 31
The next location of be is: -1
# Sample Run - 2
Enter a line: beauty lies in the eyes of the beholder.
Enter the substring to find its location: be
The first location of be in the given line is: 0
The next location of be is: 31
```

LOWER( )

The string **lower()** method converts all uppercase characters in a string into lowercase characters and returns it. Its syntax is: **string.lower()** without any parameters. If no uppercase characters exist, it returns the original string.

Example: 117 → Use of **string.lower()** method.

```
>>> strx = 'action SPEAKS lowder than WORDS'
>>> strx.lower()
'action speaks lowder than words'
>>> stry = 'COME 2 THE \u2022'
>>> stry.lower()
'come 2 the •'
```

UPPER( )

The string **upper()** method converts all lowercase characters in a string into uppercase characters and returns it. Its syntax is: **string.upper()**
It doesn't take any parameters.

Example: 118 → Use of **string.upper()**.

```
>>> strx = 'All that glitter is not gold.'
>>> print(strx.upper())
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
ALL THAT GLITTER IS NOT GOLD.
>>> stry = 'Hasta la Vista Baby !!!'
>>> print(stry.upper())
HASTA LA VISTA BABY !!!
```

STRIP( )

The **strip()** removes characters from both left and right based on the argument.

Its syntax is: **string.strip(chars)**, where, **chars** (optional) is a string specifying the set of characters to be removed. If the argument is not provided, all leading and trailing whitespaces are removed from the string.

Example: 119 → Use of **string.strip()**.

```
>>> strx = ' diamond in rough '
>>> print(strx.strip())
diamond in rough
>>> print(strx)
 diamond in rough
>>> strx.strip()
'diamond in rough'
>>> print(strx.strip('di'))
 diamond in rough
>>> print(strx.strip(' di'))
amond in rough
```

LSTRIP( )

The **lstrip()** removes characters from the left based on the argument. Its syntax is: **string.lstrip(char)**

Where, **char** is optional and it specifies the set of characters to be removed. If the **char** argument is not provided, all leading whitespaces are removed from the string.

Example: 120 → Finding first location of substring in a string.

```
>>> strx = '   Knowledge is power. '
>>> strx.lstrip()
'Knowledge is power. '  #Leading space is stripped
>>> strx.lstrip('   Know')
'ledge is power. '
```

RSTRIP( )

The **rstrip()** method returns a copy of the string with trailing characters removed based on the string argument passed.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

The syntax of **rstrip()** is: **string.rstrip(char)**, where **char** is optional specifying the set of characters to be removed. If the **char** argument is not provided, all whitspaces on the right are removed from the string.

Example: 121 → Use of **rstrip()** method.

```
>>> strz = ' Time is money. '
>>> strz.rstrip()
' Time is money.'
>>> strz.rstrip('ey. ')
' Time is mon'
>>> strz.rstrip('money')
' Time is money. '
```

TITLE( )

The **title()** method returns a string with first letter of each word capitalized.
The syntax of **title()** is: **string.title()**
The **title()** method doesn't take any parameters.

Example: 122 → Use of **title()** method.

```
>>> strx = 'the mission impossible'
>>> print(strx.title())
The Mission Impossible
```

REPLACE( )

The **replace()** method returns a copy of the string where all occurrences of a substring is replaced with another substring.
Its syntax is: **str.replace(old, new , count)**
Where, **old** is a substring we want to replace,
new is the substring which would replace the **old** substring, and
count is optional - the number of times we want to replace the **old** substring with the **new** substring
If count is not specified, **replace()** method replaces all occurrences of the old substring with the new substring.

Example: 123 → Use of string.replace() method.

```
>>> strx = 'haste makes waste'
>>> print(strx.replace('makes','don\'t make'))
haste don't make waste
>>>  stry  =  'Two  is  company,  three  is  crowd'>>>
print(stry.replace('o','a',2))
Twa is campany, three is crowd
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

SWAPCASE( )

The string **swapcase()** method converts all uppercase characters to lowercase and all lowercase characters to uppercase characters of the given string, and returns it. The format of **swapcase()** method is: **string.swapcase()**
The **swapcase()** method doesn't take any parameters.
Note: Generally we notice that,

string.swapcase().swapcase() == string

But, it is not necessary in all case.

Example: 124 → Example of **swapcase()**.

```
>>> strx = 'after ALL tomoRRow is ANOTHER DAY.'
>>> strx.swapcase()
'AFTER all TOMOrrOW IS another day.'
>>> print(strx.swapcase())
AFTER all TOMOrrOW IS another day.
```

JOIN( )

The **join()** is a string method which returns a string concatenated with each element of sequence (objects capable of returning its members one at a time, such as string, tuple and list).
The syntax of **join()** is: **string.join(sequence)**
If the sequence contains any non-string values, it raises a *TypeError* exception.

Example: 125 → Use of **string.join()** function.

```
>>> Lnum = ['1','2','3','4']
>>> seperator = '|'
>>> print(seperator.join(Lnum))
1|2|3|4
>>> Tchr = ('A','B','C','D')
>>> joinder = '-1, '
>>> print(joinder.join(Tchr))
A-1, B-1, C-1, D      # D-1 isn't printed, because
sequence starts form 0 index.
>>> Tchr1 = ('a', 'b', 'c', 'd', '')
>>> print(joinder.join(Tchr1))
a-1, b-1, c-1, d-1,
>>> strx = 'abcd '
>>> print(joinder.join(strx))
a-1, b-1, c-1, d-1,
```

ISALNUM( )

The isalnum() method returns True if all characters in the string are alphanumeric (either alphabets or numbers). If not, it returns False.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

The syntax of isalnum() is: string.isalnum()
It doesn't take any parameters.

Example: 126 → Use of **string.isalnum()** function.

```
>>> strx = 'time is money'
>>> strx.isalnum()
False    # whitespace/gap not allowed
>>> stry = '1billion money can\'t bring time'
>>> stry.isalnum()
False    # whitespace/gap not allowed
>>> strz = '1billion'
>>> strz.isalnum()
True
>>> strw = 'one-billion'
>>> strw.isalnum()
False    # special character not allowed
>>> strp = 'onebillion'
>>> strp.isalnum()
True
```

ISALPHA( )

The isalpha() method returns True if all characters in the string are alphabets (Lower or Upper case), otherwise it returns False
The syntax of isalpha() is: string.isalpha()
It doesn't take any parameters.

Example: 127 → Use of **string.isalpha()** function.

```
>>> strx = 'icy_hand'
>>> strx.isalpha()
False
>>> stry = 'IcyHand'
>>> stry.isalpha()
True
>>> strz = 'Icy Hand'
>>> strz.isalpha()
False
>>> strw = '1IcyHand'
>>> strw.isalpha()
False
```

ISDIGIT( )

The **isdigit()** method returns **True** if all characters in a string are digits. Otherwise, it returns **False**.
The syntax of **isdigit()** is: **string.isdigit()**

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

The **isdigit()** doesn't take any parameters.

Example: 128 → Use of **string.isdigit()**.

```
>>> S1 = '11223344556677'
>>> S1.isdigit()
True
>>> S2 = "20 20"
>>> S2.isdigit()
False
```

ISLOWER( )

The **islower()** method returns **True** if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns **False**.
The syntax of **islower()** is: **string.islower()**
The **islower()** method doesn't take any parameters.

Example: 129 → Use of **string.islower()**.

```
>>> S1 = "you are never alone"
>>> S1.islower()
True
>>> S2 = "you never alone."
>>> S2.islower()
True
>>> S3 = 'You are smart.'
>>> S3.islower()
False
```

ISUPPER( )

The string The **isupper()** method returns **True** if all characters in a string are uppercase characters, **False** if any characters in a string are lowercase characters.
The syntax of **isupper()** method is: **string.isupper()**
The **isupper()** method doesn't take any parameters.

Example: 130 → Use of **string.isupper()** method.

```
>>> S1 = 'ALL LIMITATIONS ARE SEFL-IMPOSED.'
>>> S1.isupper()
True
>>> S2 = 'ALL LIMITATIONS are not SELF-IMPOSED'
>>> S2.isupper()
False
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

ISSPACE( )

The **isspace()** method returns: **True** if all characters in the string are whitespace characters; **False** if the string is empty or contains at least one non-printable() character. Characters that are used for spacing are called whitespace characters. For example: tabs, spaces, newline etc.
The syntax of **isspace()** is: **string.isspace()**
The **isspace()** method doesn't take any parameters.

Example: 131 → Use of **string.isspace()** method.

```
>>> S1 = '    \t'
>>> S1.isspace()
True
>>> S2 = ' a '
>>> S2.isspace()
False
>>> S3 = ''
>>> S3.isspace()
False
```

STRING PROGRAMMING EXAMPLES

Example: 132 → Write a program that reads a line and the substring/word, and displays the number of times substring occurs in the line.

```
# Counting the occurence of substring
# File Name: string_pro1.py | Script by: A. Nasra
S1 = input('Write a line: ')
sub = input('Write the word to be counted: ')
n = S1.count(sub)
print('The word',sub,' comes',n, 'times.')
```

```
# Sample run of string_pro1.py
Write a line: jingle bell, jingle bell, jingle all the
way
Write the word to be counted: jingle
The word jingle  comes 3 times.
```

Example: 133 → Write a program that takes a string with multiple words and then capitalizes the first letter of each words.

```
# Capitalizing first letter of each word in a string.
# File Name: cap_word.py | Coded by: A. Nasra
S1 = input("Enter a string with multiple words:\n  ")
print('String with first letter of word apitalized:\n
',S1.title())
```

```
# Sample run of cap_word.py
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Enter a string with multiple words:
   beauty provokes a thief sooner than gold.
String with first letter of word capitalized:
   Beauty Provokes A Thief Sooner Than Gold.
```

Example: 134 → Write a program that test if a word entered by the user is palindrome or not.

```
#A program to capitalize first letter of each word in
a string.
# File Name: palindro.py | Coded by: A. Nasra
S1 = input("Enter a string: ")
for i in range(len(S1)):\
#note that range(n) means 1,2,3,.....,n-1
    if S1[i] != S1[-i-1]:
        print(S1,' isn\'t Palindrome')
        break
else:
    print(S1,' is Palindrome')
```

```
# Sample run of palindro.py
Enter a string: madam
madam  is Palindrome
```

Example: 135 → Write a program that reads a string and then print a string that capitalizes every other letter in the string, e.g. affection becomes aFfEcTiOn

```
# program to capitalize every other letter in a string
# File Name: cap_odd.py {Script by: A. Nasra}
S1 = input('Enter a word: ')
S2 = '' # Empty string
for i in range(len(S1)):
    if i%2 != 0:
        S2 = S2 + S1[i].upper()
    else:
        S2 = S2 + S1[i].lower()
print('Required string is:', S2)
```

```
# Sample run of cap_odd.py
Enter a word: indepencence
Required string is: iNdEpEnCeNcE
```

Example: 136 → Write a program that does the following:

- Takes two inputs: the first an integer and the second a string
- From the input string extract all the digits in order of occurrence. If no digit occurs, set the extracted digits to 0
- Add the integer input and the digits extracted from the input string.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

- Print a string of the form: integer input + string digits = sum
- For example for inputs 13, a1b2c3 → 13 + 123 = 13

```
# File Name: int_str.py | Coded by: A. Nasra
S0 = int(input("Enter an integer: "))
S1 = input("Enter an alphanumeric string: ")
S2 = ''
if S1.isalpha():
    print('Required output = ',S0,'+ 0 =',S0)
else:
    if S1.isalnum(): # Test for alphanumetic string.
        for i in range(len(S1)):
            if S1[i].isdigit():
                S2 = S2+S1[i]
    print('Required output = ',S0,'+',S2,'=',S0 + int(S2))
```

Example: 137 → Write a program to print alternate characters in a string. Input a string of your own choice.

```
# File Name: alt_str.py | Script by: A. Nasra
opt = 'y'
while opt == 'y' or opt =='Y':
    S1 = input('Enter a string: ')
    S2 = '' # Empty string
    for i in range(0,len(S1),2):
        S2 += S1[i]
    print('Required new string is ',S2)
    opt = input('Do you want to do again?(y/n)...')
```

```
# Sample run of alt_str.py
Enter a string: Python Love
Required new string is  Pto oe
Do you want to do again?(y/n)...y
Enter a string: pytho love
Required new string is  ptolv
Do you want to do again?(y/n)...n
```

Example: 138 → Write a program that asks the user for a string and returns an estimate of how many words are in the string. (*Hint*: asks the user for a string and returns an estimate of how many words are in the string.)

```
# Counting number of words in a string
# File Name: word_count.py | Code by: A. Nasra
s1 = input('Enter a sentence: ')
n = s1.count(' ')
print('Sentence entered by you has',n+1,'words')
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run of word_count.py
Enter a sentence: Beauty is, but skin deep.
Sentence entered by you has 5 words
```

Example: 139 → Write a program that asks the user to enter a word and prints out whether that word contains any vowels. If it contains vowels then what and how many?

```
# File Name: vowel_count.py | Coded by: A. Nasra
s1 = input('Enter a word: ')
na = s1.count('a')
ne = s1.count('e')
ni = s1.count('i')
no = s1.count('o')
nu = s1.count('u')
print('Word entered by you conatins:')
if na==0 and ne==0 and ni==0 and no==0 and nu==0:
    print('no vowel.')
if na!=0:
    print('vowel a -',na,'time(s)')
else:
    pass
if ne!=0:
    print('vowel e -',ne,'time(s)')
else:
    pass
if ni!=0:
    print('vowel i -',ni,'time(s)')
else:
    pass
if no!=0:
    print('vowel o -',no,'time(s)')
else:
    pass
if nu!=0:
    print('vowel u -',nu,'time(s)')
```

```
# Sample Run of vowel_count.py
# Sample Run - 1
Enter a word: rhythm
Word entered by you conatins:
no vowel.
# Sample Run - 2
Enter a word: compassion
Word entered by you conatins:
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
vowel a - 1 time(s)
vowel i - 1 time(s)
vowel o - 2 time(s)
```

Example: 140 → Write a program that asks the user to enter a string, then prints out each letter of the string doubled and separated by space. For instance, if the user entered HEY, the output would be: HH EE YY

```
# File Name: double_letter.py {Coded by: A. Nasra}
s1 = input('Enter a word:')
s2 = ''
for i in range(len(s1)):
    s2 = s2 + s1[i]*2 + ' '
print(s2)
```

```
# Sample run of double_letter.py
Enter a word:Python-Prick
PP yy tt hh oo nn -- PP rr ii cc kk
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

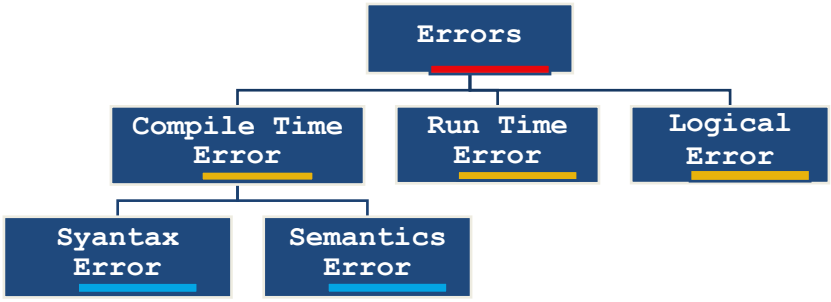
DEBUGGING AND EXCEPTION HANDLING

DEBUGGING

The process of keep on finding errors, rectifying them and recompiling till the program is completely error-free, is called debugging.

ERRORS IN A PROGRAMS

An error, also called bug is a part fo the complete program code, that prevents the program from compiling and running/executing correctly. There are mainly three types of errors as is given below:



COMPILE TIME ERROR

The errors that occur during compile-time are called compile time errors. Two types of errors that fall under this category are –

1. Syntax errors and **2.** Semantics errors

SYNTAX ERROR

The Syntax errors (also called parser errors) are the most basic type of error. They arise when the Python parser is unable to understand a line of code. Syntax errors occur when rules of a programming language is misused or violated. Most syntax errors are type errors, incorrect indentation, or incorrect arguments.

Example: 141 → Ilustration of syntax error.

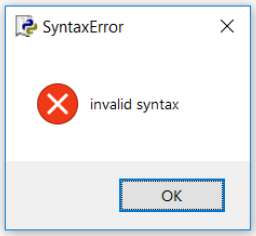
```
# File Name: syntax_error.py
x = int(input('Enter a number: '))
while x%2 == 0:
    print('You have entered an even number.')
else:
    print ('You have entered an odd number.')
```

Notice that the keyword **whille** is misspelled. If we try to run the program, we will get the following error:

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 142 → Illustration of syntax error.

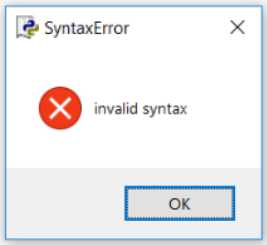
```
# File Name: syntax_error.py
x = int(input('Enter a number: '))
while x%2 == 0:
    print('You have entered an even number.')
else:
    print ('You have entered an odd number.')
```



Example: 143 →

```
# File Name: syntax_error1.py
x = int(input('Enter 1st number: '))
y = int(input('Enter 2nd number: '))
z = int(input('Enter 3rd number: '))
x = y*z
if x = y*z
    print('Your are right.')
```

```
# File Name: syntax_error1.py
x = int(input('Enter 1st number: '))
y = int(input('Enter 2nd number: '))
z = int(input('Enter 3rd number: '))
x = y*z
if x = y*z
    print('Your are right.')
```



SEMANTICS ERROR

The Semantics error occur when syntax of the statements are correct but meaning is not clear or ambiguous. For example, the statement – Ram goes to market. – is syntactically correct and has some meaning, but the statement – Market goes to Ram – is syntactically correct but meaningless.
Similarly, the statement **`x/y = z`** will result in semantics error.

Example: 144 → Illustration of semantics error.

```
>>> a = '21'
>>> b = 'twenty two'
>>> c = a - b
# All the statements are syntactically correct, but the
third statement has no meaning.
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

RUN TIME ERROR

The errors that occur during the execution of a program are called runtime errors. Some runtime errors stop the execution of the program which is then called program 'crashed' or abnormally terminated'. Most runtime errors due to infinite loop or wrong value or using function of the library which is not meant for that.

Example: 145 → Illustration of runtime error.

```
# File Name: runtime_error.py
import math
dividend = float(input("Please enter the dividend:"))
divisor = float(input("Please enter the divisor: "))
quotient = dividend / divisor
quotient_rounded = math.round(quotient)
```

```
# Sample run of runtime_error.py
C:/Users/A.Nasra/AppData/Local/Programs/Python/Python
37/runtime_error.py
Please enter the dividend: 12
Please enter the divisor: 67
Traceback (most recent call last):
File C:/.../Python37/runtime_error.py",
line 5, in <module>
  quotient_rounded = math.round(quotient)
AttributeError:module 'math' has no attribute 'round'
```

LOGICAL ERROR

Sometimes, there are no error messages during compile time or run time but program does not provide the correct output as desired. Such errors are called logical errors. This type of error is caused by a mistake in the program's logic. Some of the mistakes giving birth to logic errors may be due to following reasons:

- using the wrong variable name
- indenting a block to the wrong level
- using integer division instead of floating-point division
- getting operator precedence wrong
- making a mistake in a boolean expression
- off-by-one, and other numerical errors

Example: 146 → Program to find the sum of $1^2 + 2^2 + 3^2 + \cdots + 10^2$

```
# File Name: logical_error.py
sum_squares = 0
for i in range(10):
    i_sq = i**2
sum_squares += i_sq
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
print(sum_squares)
```

```
# Sample run of logical_error.py
81        # Not desired output
```

EXCEPTIONS

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

DIFFERENCE BETWEEN ERROR AND EXCEPTION

Error is a bug in the code that causes irregular output or stops a program from executing; whereas an Exception is an irregular unexpected situation occurring during execution on which programmer has no control.

EXCEPTION HANDLING

The code written for dealing with exceptions in a program is known as exception handling.

In Python, exception handling is performed with the help of keywords **`try`** and **`except`**. The code that may generate an exception is written in the **`try`** block and the code for handling exception when the exception is raised is written in **`except`** block, as is given below:

```
    try:
        # code that may generate exception.
    except:
        # code when the exception has occurred.
```

Example: 147 → Program to show the use of exception handling.

```
# File Name: exception1.py | Code by: A. Nasra
# import module sys to get the type of exception
import sys
try:
    entry = eval(input('Enter a number...'))
    print("The entry is", entry)
    print("The reciprocal of",entry,"is",1/entry)
except:
    print("Oops!",sys.exc_info()[0],"occured.")
```

```
# Sample run of exception1.py
# Sample run - 1
Enter a number...0.9999999
The entry is 0.9999999
The reciprocal of 0.9999999 is 1.00000010000001
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run - 2
Enter a number...0.00
The entry is 0.0
Oops! <class 'ZeroDivisionError'> occured.
# Sample run - 3
Enter a number...a
Oops! <class 'NameError'> occured.
# Sample run - 4
Enter a number...3.5E1000
The entry is inf
The reciprocal of inf is 0.0
# Sample run - 5
Enter a number...'78'
The entry is 78
Oops! <class 'TypeError'> occured.
```

BUILT-IN EXCEPTIONS

Python comes with many predefined exceptions, called built-in exceptions. Some most common built-in exceptions are given in the following table:

Name of Exception	
AssertionError	Raised when **assert** statement fails.
AttributeError	Raised when attribute assignment or reference fails.
EOFError	Raised when the **input()** functions hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raise when a generator's **close()** method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when system operation causes system related error.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

OverflowError	Raised when result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by **next()** function to indicate that there is no further item to be returned by iterator.
SyntaxError	Raised by parser when syntax error is encountered.
IndentationError	Raised when there is incorrect indentation.
TabError	Raised when indentation consists of inconsistent tabs and spaces.
SystemError	Raised when interpreter detects internal error.
SystemExit	Raised by **sys.exit()** function.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translating.
ValueError	Raised when a function gets argument of correct type but improper value.
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.

IMPORTANT DEBUGGING TECHNIQUES

Debugging means correction of code so that the cause of errors is removed. For this purpose following techniques are adopted:

1. Carefully spotting the origin of error (The origin of error may be in or before the line of error indicated by the interpreter.)
2. Printing variables' intermediate values

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

3. Code tracing and stepping (It means executing code one line at a time and noting its impact on variables. Code tracing is down with the help of built-in debugging tools, also called debugger.)

USE OF DEBUGGER TOOLS

The debugging tool (or Debugger) is a software that is used to test and debug the code written in any computer language. A debugger can be a separate or integrated with IDE. Python has separate debugger called pdb (*p*rogram/*p*ython *deb*ugger). The Spider IDE has integrated debugger.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

LIST MANIPULATION

DEFINITION OF LIST

List is a sequence of values of any type. These values are called elements or items separated by comma. Elements of list are mutable (It means, the memory address will not change even after we change its values.) and indexed by integers. List is enclosed in square brackets [ ].

CREATING LIST

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas. It can have any number of items and they may be of different types (integer, float, string etc.).

Example: 148 → Creating string

```
L1 = []                    # empty list
L2 = [1, 2, 3]             # list of integers
L3 = [1,"Hello",3.4]       #list with mixed datatypes
L4 = ["mouse",[8,4,6],3.4] # nested list
```

CREATING A LIST FROM EXISTING SEQUENCE

To create a list from the existing sequence (i.e. string, tuple, list) the syntax is,

```
Name_of_list = list(<sequence>)
```

Example: 149 → Creation of list from sequence (i.e. string, list, tuple)

```
>>> S1 = "will power"
>>> L1 = list(S1)
>>> L1
['w', 'i', 'l', 'l', ' ', 'p', 'o', 'w', 'e', 'r']
>>> print(L1)
['w', 'i', 'l', 'l', ' ', 'p', 'o', 'w', 'e', 'r']
>>> T1 = (1,7,2,8,3,6)
>>> L2 = list(T1)
>>> print(L2)
[1, 7, 2, 8, 3, 6]
>>> print(list(T1))
[1, 7, 2, 8, 3, 6]
>>> L0 = ['seven',1,'eight',2,'six']
>>> print(list(L0))
['seven', 1, 'eight', 2, 'six']
```

We can also create a list of single characters or single numbers via keyboard, as given in the following example:

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 150 → Creation of a list by keyboard.

```
>>> L3 = list(input('Enter list elements without space:
'))
Enter list elements without space: 7a8b6c
>>> print(L3)
['7', 'a', '8', 'b', '6', 'c']
```

We notice that the list produced by this method is the list of stings (each element entered by keyboard becomes single string). To enter a list of integers or mixed type, we can use the following method, as given in the following example.

Example: 151 → Creation of a list by keyboard.

```
>>> L4 = eval(input('Enter a list: '))
Enter a list: [3,4,'a',4.5]
>>> print(L4)
[3, 4, 'a', 4.5]
```

Note: Sometimes the **`eval()`** function not work in Python Shell. At that time, script mode can be used.

SIMILARITY BETWEEN STRING AND LIST

List are similar to String in following respect:

1. Function **`len()`** returns the number of items(elements)
2. The index **`list[i]`** returns the item at index **`i`** (the first item is indexed at 0)
3. Slicing **`list[i:j]`** returns a new list, containing the objects at indexes between **`i`** and **`j`** (excluding index **`j`**)
4. The membership operators in and not in works the same way as with strings.
5. Concatenation (**+**) and replication operators(*****) works the same way as with strings.

DIFFERENCE BETWEEN STRING AND LIST

The important difference is that the strings are immutable and list is mutable which is clear from the following example.

Example: 152 → Difference between string and list.

```
>>> L1 = ['N','A','S','H','W','A','R']
>>> S1 = "NASHWAR"
>>> L1[0] = 'M'
>>> L1[0] = 'M'
>>> print(L1)
['M', 'A', 'S', 'H', 'W', 'A', 'R']
>>> S1[0] = 'M'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
    S1[0] = 'M'
TypeError: 'str' object does not support item assignment
```

ACCESSING INDIVIDUAL ELEMENT OF A LIST

The individual element of a list are accessed through their indexes, as is clear from the following example.

Example: 153 → Accessing individual element of a list.

```
>>> L1 = ['h', 'u', 'm', 'a', 'n']
>>> print(L1[0])
h
>>> print(L1[2])
m
>>> print(L1[-1])
n
>>> print(L1[-5])
h
>>> L2 = ["happy",[1,7,2,8,3,6]] # Nested loop
>>> print(L2[0][1])
a
>>> print(L2[1][1],L2[1][3],L2[1][5])
7 8 6
```

TRAVERSING A LIST

Traversing a list means the same as for strings, i.e. accessing and processing each and every element of the list.

Example: 154 → Write a program to find the sum of square of each element of the integer list entered by the user.

```
# sum of squares of each element of an integer list
# File Name: sqrt_list.py | Code by: A. Nasra
sum = 0
L1 = eval(input('Enter an integer list: '))
for i in range(len(L1)):
    sum = sum + L1[i]*L1[i]
print(sum)
```

```
# Sample run of sqrt_list.py
Enter an integer list: [1,1.1,-1,-1.1]
4.42
# Checking: 1**2 + 1.1**2 + (-1)**2 + (-1.1)**2 = 4.42
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

COMPARING LISTS

Two lists are compared with the help of comparison operators (**<, >, ==, !=**) in lexicographical order.

Example: 155 → Illustration of comparison between lists

```
>>> L1 = [1, 2, 8, 9]
>>> L2 = [9, 1]
>>> L3 = [1, 2, 9, 1]
>>> L4 = [1, 2, 9, 10]
>>> L5 = [1, 2, 8, 4]
>>> L1 < L2 ; L1 < L3 ; L1 < L4 ; L1 < L5
True
True
True
False
```

LIST OPERATIONS

Important list operations are – joining lists, replicating lists and slicing lists.

JOINING LISTS

Joining of two or more lists is done by the concatenation operator (+), as is explained in the following examples.

Example: 156 → Illustration of joining of two lists.

```
>>> L1 = [1,2,3,4] ; L2 = [9,8,7]
>>> print('L1 + L2 =', L1 + L2)
L1 + L2 = [1,2,3,4,9,8,7]
>>> print('L1 + L2 + L3 =', L1 + L2 + L3)
L1 + L2 + L3 = [1,2,3,4,9,8,7,1,2,3,4,9,8,7]
```

To concatenate (+) both operands must be lists. The following command will raise **type error**:
List + Number ; List + Complex Number ; List + String

REPEATITION OR REPLICATION OF LISTS

This is done with the help of replication operator (*) as is explained in the following example:

Example: 157 → Illustration of replication of a lists.

```
>>> L1 = ['a', 1,2+3j, 'zing']
>>> print(' Replication of list 2 times:\n', L1*2)
 Replication of list 2 times:
 ['a', 1, (2+3j), 'zing', 'a', 1, (2+3j), 'zing']
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

SLICING THE LISTS

The extraction of subpart of a list is called slicing a list.
Slicing can be best imagined by seeing the index between the elements as shown below. So if we want to access a range, we need two index that will slice that portion from the list.

'c'	'l'	'i'	'c'	'k'	2	's'	'l'	'i'	'p'
0	1	2	3	4	5	6	7	8	9
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Syntax for slice is: **`L2 = L1[start : stop : step]`**
Where, **`start`** is the starting index, **`stop`** is the stopping index-1and **`step`** is the slice step (by default it is 1).

Example: 158 → Illustration of slicing a lists.

```
>>> L1 = ['c','l','i','c','k',2,'s','l','i','p']
>>> L2 = L1[1:7]; L3 = L1[7:1]; L4 = L1[-1:7]
>>> L5 = L1[1:-7]; L6 = L1[-1:-7]; L7 = L1[1:]
>>> L8 = L1[:]
>>>  print('L2 =',  L2,'\nL3 =',L3,'\nL4 =',L4,'\nL5
=',L5,'\nL6 =',L6,'\nL7 =',L7,'\nL8 =',L8)
L2 = ['l', 'i', 'c', 'k', 2, 's']
L3 = []
L4 = []
L5 = ['l', 'i']
L6 = []
L7 = ['l', 'i', 'c', 'k', 2, 's', 'l', 'i', 'p']
L8 = ['c', 'l', 'i', 'c', 'k', 2, 's', 'l', 'i', 'p']
>>> print('L9 =',L1[1:9:2],'\nL10 =',L1[1:9:3],'\nL11
=',L1[::3],'\nL12 =',L1[3::2],'\nL13 = ',L1[::-1])
L9 = ['l', 'c', 2, 'l']
L10 = ['l', 'k', 'l']
L11 = ['c', 'c', 's', 'p']
L12 = ['c', 2, 'l', 'p']
L13 = ['p', 'i', 'l', 's', 2, 'k', 'c', 'i', 'l', 'c']
# Reversal of list
```

LIST FUNCTIONS AND METHODS

The python provides many built-in functions and methods. The syntax for using these functions and method is: **`List_name.Method_name()`**
These methods are discussed below:

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

INDEX( )

The **index()** method searches an element in the list and returns its index/position. However, if the same element is present more than once, **index()** method returns its smallest/first position.

The syntax of **index()** method for list is: **list.index(element)**

Where, **element** is the value to be searched.

The **index()** method returns the **index** of the **element** in the list.

If not found, it raises a **ValueError** exception indicating the element is not in the list.

Example: 159 → Illustration **list.index()** method.

```
>>> L1 = [11, 22, 33, 44, 55]
>>> L1.index(33)
2
>>> L1.index(32)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    L1.index(32)
ValueError: 32 is not in list
```

Example: 160 → Illustration **list.index()** method.

```
>>> L2 = ['thing',(1,3,5),[1.1,2.2],'a',7.8]
>>>     L2.index('thing');          L2.index((1,3,5));
L2.index([1.1,2.2]); L2.index(7.8)
0
1
2
4
```

APPEND( )

The **append()** method adds a single item to the existing list. It doesn't return a new list; rather it modifies the original list. It doesn't return any value.

The syntax of **append()** method is: **list.append(item)**

The **append()** method takes a single **item** and adds it to the end of the list.

The **item** can be numbers, strings, another list, dictionary etc.

Example: 161 → Illustration **list.append()** method.

```
>>> Sub_L = ['maths','phy','chem','bio']
>>> Sub_L.append('cs')
>>> print(Sub_L)
['maths', 'phy', 'chem', 'bio', 'cs']
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 162 → Illustration of appending list to a list.

```
# File name: apnd.py | Script written by: A. Nasra
fruit = ['apple', 'mango', 'guava']
dry_fruit = ['almond', 'cashew']
fruit.append(dry_fruit)
print('Updated fruit list: ', fruit)
```

```
# Sample runoff apnd.py
Updated fruit list:    ['apple', 'mango', 'guava',
['almond', 'cashew']]
```

EXTEND( )

The **extend()** method extends the list by adding all items of a list (passed as an argument) to the end.

The syntax of **extend()** method is: **list1.extend(list2)**

Here, the elements of list2 are added to the end of list1.

The argument may also be a set or a tuple.

The **extend()** method only modifies the original list. It doesn't return any value.

Example: 163 → Illustration of extending a list.

```
# File name: extnd.py | Script written by: A. Nasra
language = ['French', 'English', 'German']
language1 = ['Spanish', 'Portuguese']
language.extend(language1)
# Extended List
print('Entended Language List:\n', language)
```

```
# Sample run of extnd.py
Entended Language List:
 ['French','English','German','Spanish','Portuguese']
```

Example: 164 → Illustration of extending a list.

```
# File Name: extnd1.py | Code by: A. Nasra
L1 = [1, 2, 3, 'l', 'm', 'n']
T1 = (10,20,30)
S1 = {'alpha', 'beta', 'gamma'}
L1.extend(T1)
print('New extended list is:',L1)
L1.extend(S1)
print('Newer extended list is:\n',L1)
```

```
# Sample run of extnd1.py
New extended list is: [1, 2, 3, 'l', 'm', 'n', 10, 20,
30]
Newer extended list is:
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
 [1, 2, 3, 'l', 'm', 'n', 10, 20, 30, 'beta', 'gamma', 'alpha']
```

INSERT( )

The **insert()** method inserts the element to the list at the given index.
The syntax of **insert()** method is: **list.insert(index, element)**
The **insert()** function takes two parameters: (1) **index** - position where element needs to be inserted and (2) **element** - this is the element to be inserted in the list.
The **insert()** method only inserts the element to the list. It doesn't return any value.

Example: 165 → Illustration of inserting an element in a list.

```
>>> swar = ['\u0905','\u0906','\u0908']
>>> print('swar =',swar)
swar =  ['अ', 'आ', 'ई']
>>> swar.insert(2,'\u0907')
>>> print('After inserting swar =',swar)
After inserting swar = ['अ', 'आ', 'इ', 'ई']
```

Example: 166 → Illustration of inserting an element in a list.

```
>>> L1 = [{2,3},[4,'a']]
>>> T1 = (5,6)
>>> L1.insert(0,T1)
>>> print('New L1 =',L1)
New L1 = [(5, 6), {2, 3}, [4, 'a']]
```

POP( )

The **pop()** method takes a single argument (**index**) and removes the element present at that index from the list.
The syntax of **pop()** method is: **list.pop(index)**
The **pop()** method removes the element at the given index and updates the list.
If the **index** passed to the **pop()** method is not in the range, it throws **IndexError: pop index out of range exception.**
If no parameter is passed, the default **index -1** is passed as an argument which removes the last element. The **pop()** method returns the element present at the given index.

Example: 167 → Illustration of pop() method to remove an element from a list.

```
>>> HLL = ['Python','Go','C#','Java','Kotlin']
>>> L_R = HLL.pop(2)
>>> print('Removed HLL is:',L_R)
Removed HLL is: C#
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> print('Updated HLL is:',HLL)
Updated HLL is: ['Python', 'Go', 'Java', 'Kotlin']
```

Example: 168 → Illustration of **pop()** method to remove an element from a list.

```
# File Name: pop.py  |  Code writer: A. Nasra
print('HLL = [Python, Java, Go, Kotlin, C++]')
HLL = ['Python','Java','Go','Kotlin','C++']

# When index is not passed
print('When index is not passed:')
print('Removed Value:',HLL.pop())
print('Updated HLL:',HLL)

# When -1 is passed (Pops Last Element)
print('\nWhen -1 is passed:')
print('Removed Value:',HLL.pop(-1))
print('Updated HLL:',HLL)

# When -3 is passed (Pops Third Last Element)
print('\nWhen -3 is passed:')
print('Removed Value:',HLL.pop(-3))
print('Updated HLL:',HLL)
```

```
# Sample run of pop.py
HLL = [Python, Java, Go, Kotlin, C++]

When index is not passed:
Removed Value: C++
Updated HLL: ['Python', 'Java', 'Go', 'Kotlin']

When -1 is passed:
Removed Value: Kotlin
Updated HLL:  ['Python', 'Java', 'Go']

When -3 is passed:
Removed Value: Python
Updated HLL: ['Java', 'Go']
```

REMOVE( )

The **remove()** method takes a single element as an argument and removes it from the list.

The syntax of **remove()** method is: **list.remove(element)**

If the **element (argument)** passed to the **remove()** method doesn't exist, **valueError** exception is thrown. The **remove()** method only removes the given element from the list. It doesn't return any value.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 169 → Illustration of **remove()** method to remove an element.

```
# File Name: remove1.py | Code by: A. Nasra
print('scientists = [JC Bose, Ramanujan, CV Raman, Kalam]')
scientist = ['JC Bose','Ramanujan','CVaman','Kalam']

# 'Ramanujan' element is removed
print('When Ramanujan is removed.')
scientist.remove('Ramanujan')

#Updated scientist List
print('Updated scientist list:',scientist)
```

```
# Sample run of remove1.py
scientists = [JC Bose, Ramanujan, CV Raman, Kalam]
When Ramanujan is removed.
Updated scientist list: ['JC Bose', 'CV Raman', 'Kalam']
```

Example: 170 → Illustration of **remove()** method to remove an element.

```
# File Name: remove2.py | Code by: A. Nasra
# If a list contains duplicate elements, remove() method removes only the first instance
print('animal = [tiger, dog, dog, lion, dog]')
animal = ['tiger', 'dog', 'dog', 'lion', 'dog']

print('When element dog is removed')
animal.remove('dog')

print('Updated animal list:',animal)
```

```
# Sample run of remove2.py
animal = [tiger, dog, dog, lion, dog]
When element dog is removed
Updated animal list: ['tiger', 'dog', 'lion', 'dog']
```

CLEAR()

The **clear()** method removes all items from the list.

The syntax of **clear()** method is: **list.clear()**

The **clear()** method doesn't take any parameters.

The **clear()** method only empties the given list. It doesn't return any value.

Example: 171 → Illustration of **clear()** method to empty a list.

```
>>> L1 = [(1, 2),['alpha','beta'],{1.1, 2.2}]
>>> L1.clear()
>>> print('L1.clear() =',L1)
L1.clear() = []
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> # Emptying the List Using del
>>> del L1[:]
>>> print('del L1[:] =',L1)
del L1[:] = []
```

COUNT( )

The **count()** method returns the number of occurrences of an element in a list.
The syntax of **count()** method is: **list.count(element)**
The **count()** method takes a single argument and returns the number of occurrences of an element in a list.

Example: 172 → Illustration of **count()** method to count an element in a list.

```
>>> L1 =[123,'abc',1.1,2.2,123,'def',123,'abc','def']
>>> n = L1.count(123)
>>> print('The count of 123 is:',n)
The count of 123 is: 3
>>> n = L1.count('abc')
>>> print('The count of abc is:',n)
The count of abc is: 2
```

Example: 173 → Illustration of **count()** method to count an element in a list.

```
>>> L1 = [1.2,1.3,(1,2),[1,2],{1,2},(1,2),(1,2),[1,2]]
>>> n1 = L1.count((1,2))
>>> n2 = L1.count([1,2])
>>> n3 = L1.count({1,2})
>>> n4 = L1.count(1.4)
>>> print('Count of (1,2) is:',n1,'\nCount of [1,2]
is:',n2,'\nCount of {1,2} is:',n3,'\nCount of 1.4
is:',n4)
Count of (1,2) is: 3
Count of [1,2] is: 2
Count of {1,2} is: 1
Count of 1.4 is: 0
```

SORT( )

The **sort()** method sorts the elements of a given list in Ascending or Descending order. The syntax of **sort()** method is:

list.sort(key=..., reverse=...)

Alternatively, we can also use Python's in-built function **sorted()** for the same purpose. **sorted(list, key=..., reverse=...)**
Simplest difference between **sort()** and **sorted()** is: **sort()** doesn't return any value while, **sorted()** returns an iterable list. By default, **sort()** doesn't require any extra parameters. However, it has two optional parameters:

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

reverse - If true, the sorted list is reversed (or sorted in Descending order)
key - function that serves as a key for the sort comparison.
sort() method doesn't return any value. Rather, it changes the original list.
If you want the original list, use **sorted()**.

Example: 174 → Illustration of **sort()** method to sort an element in a list.

```
# File Name: sort1.py | Coded by: A. Nasra
print('Given list is:')
print(' [Jan,Feb,March,April,May,June]')
month = ['Jan','Feb','March','April','May','June']
month.sort()
print('Sorted list:\n',month)
```

```
# Sample run of sort1.py
Given list is:
 [Jan,Feb,March,April,May,June]
Sorted list:
 ['April', 'Feb', 'Jan', 'June', 'March', 'May']
```

Example: 175 → Display of **sort()** to sort a list in descending order.

```
# File Name: sort2.py | Code: A. Nasra
print('vowel = [e, a, u, o, i]')
vowel = ['e', 'a', 'u', 'o', 'i']
vowel.sort(reverse = True)
print('Sorted list (Descending order):\n', vowel)
```

```
# Sample run of sort2.py
vowel = [e, a, u, o, i]
Sorted list (Descending order):
 ['u', 'o', 'i', 'e', 'a']
```

Example: 176 → Ex. of **sort() with key**.

```
# File Name: sort3.py | Code: A. Nasra
# take second element for sort
def take2nd(elem):
    return elem[1]
def take1st(elem):
    return elem[0]
print('Original list: [(2, 2),(3, 4),(4, 1),(1, 3)]')
point = [(2, 2), (3, 4), (4, 1), (1, 3)]
point.sort(key = take2nd)      # sort list with key
print('Sorted list-1:',point) # print sorted list

point.sort(key = take1st)
print('Sorted list-2:',point)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run of sort3.py
Original list: [(2, 2), (3, 4), (4, 1), (1, 3)]
Sorted list-1: [(4, 1), (2, 2), (1, 3), (3, 4)]
Sorted list-2: [(1, 3), (2, 2), (3, 4), (4, 1)]
```

Example: 177 → Ex. of **sort() with built-in key**.

```
# File Name: sort4.py | Code: A. Nasra
print('Original list: [book, toffee, map, pencil]')
L1 = ['book','toffee','map','pencil']
L1.sort(key = len)  # sort list with built-in function
len
print('Sorted list:', L1)
```

```
# Sample run of sort4.py
Original list: [book, toffee, map, pencil]
Sorted list: ['map', 'book', 'toffee', 'pencil']
```

REVERSE( )

The **reverse()** method reverses the elements of a given list.
The syntax of **reverse()** method is: **list.reverse()**
The **reverse()** function neither takes any argument nor return any value. It only reverses the elements and updates the list.

Example: 178 → Illustration of **revers() .**

```
# File Name: reverse.py | Code: A. Nasra
os = ['Windows', 'macOS', 'Linux', 'Android']
print('Original List:', os)
os.reverse()     # List Reverse
print('Updated List:', os) # updated list
```

```
# Sample run of reverse.py
Original List:['Windows', 'macOS', 'Linux','Android']
Updated List: ['Android', 'Linux', 'macOS','Windows']
```

COPY( )

The **copy()** method returns a copy of the list.
The syntax of **copy()** method is: **new_list = list.copy()**
The **copy()** method doesn't take any parameters. The **copy()** function returns the original list, but it doesn't modify the original list.

Example: 179 → Illustration of **copy() .**

```
# File Name: copy1.py | Script by: A. Nasra
L1 = ['child', 'adolescent', 'adult', 'young']
L2 = L1.copy()
print('Old List:', L1)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
print('New List:', L2)
L2.append('old')          # Updation of New List-L2
print('Old List:', L1) # Old List-L1 is not changed
print('New List:', L2) # New element is added to
                            copied list.
```

```
# Sample run of copy1.py
Old List: ['child', 'adolescent', 'adult', 'young']
New List: ['child', 'adolescent', 'adult', 'young']
Old List: ['child', 'adolescent', 'adult', 'young']
New  List:['child',  'adolescent',  'adult',  'young',
'old']
```

Note: A list can be also copied with the help of = operator but in this way when we change copied list, the original list is also changed. Let us understand this by the following example.

Example: 180 → Illustration of **copy()**.

```
# File Name: copy2.py | Code by: A. Nasra
old_list = ['fear','anger','sad','joy','disgust']
new_list = old_list
new_list.append('surprise')     # add element to list
print('New List:', new_list )
print('Old List:', old_list )
```

```
# Sample run of copy2.py
New List: ['fear', 'anger', 'sad', 'joy', 'disgust',
'surprise']
Old List: ['fear', 'anger', 'sad', 'joy', 'disgust',
'surprise']
```

MATRIX IMPLIMENTATION USING LIST

A matrix is a two-dimensional data structure. Matrix operation can be implemented using nested list. List inside another list is called nested list.

CREATING A MATRIX

In python, matrix is a nested list. A nested list is created by placing all the items (elements) inside a square bracket [ ] separated by commas. Note that all nested list is not a matrix.

Example: 181 → Write a program to create a matrix.

```
A = [['Monty', 80, 75, 85],
     ['John', 75, 80, 75],
     ['Dave', 80, 80, 80]]
>>> print(A)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
[['Monty', 80, 75, 85], ['John', 75, 80, 75], ['Dave', 80, 80, 80]]
```

CREATING A DYNAMIC MATRIX

Dynamic matrix is created using following syntax:

```
M = [[0 for j in range(c)] for i in range(r)]
```

Where, M is the name of matrix, r is the number of row, c is the number of column, i is the row-index and j is column-index.

Example: 182 → A program to create a 3×3 matrix

```
# File Name: matrix2.py | Code by: A. Nasra
c = 3;  r = 3
print('Enter elements of matrix M:')
M = [[0 for j in range(c)] for i in range(r)]
for i in range(r):
    for j in range(c):
        print('M[',i,'][',j,'] = ', end='')
        M[i][j] = eval(input())
print(M)
```

```
# Sample run of matrix2.py
Enter elements of matrix M:
M[ 0 ][ 0 ] = 'A'
M[ 0 ][ 1 ] = 'num'
M[ 0 ][ 2 ] = '45'
M[ 1 ][ 0 ] = 34
M[ 1 ][ 1 ] = 25
M[ 1 ][ 2 ] = 56.7
M[ 2 ][ 0 ] = 8.7
M[ 2 ][ 1 ] = 14
M[ 2 ][ 2 ] = 86
[['A', 'num', '45'], [34, 25, 56.7], [8.7, 14, 86]]
```

LIST PROGRAMMING EXAMPLES

Example: 183 → Write a program to find the minimum and maximum elements with respective indexes from a list entered by keyboard/user.

```
# File Name: ques1_list.py | Code by: A. Nasra
# To find max and min element with index in a list.
L1 = eval(input('Enter a list: '))
Min = min(L1) ; Max = max(L1)
i_min = L1.index(Min) ; i_max = L1.index(Max)
print('Minimum element:',Min,'at Index:',i_min)
print('Maximum element:',Max,'at Index:',i_max)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run -1 of ques1_list.py
Enter a list: [34.56, 43,43.50,43.65,34,67,67.45]
Minimum element: 34 at Index: 4
Maximum element: 67.45 at Index: 6
# Sample run -2 of ques1_list.py
Enter a list: ['concious','subconcious','unconcious']
Minimum element: concious at Index: 0
Maximum element: unconcious at Index: 2
```

Example: 184 → Write a program to calculate the mean of elements of a list of numbers, entered by keyboard/user.

```
# File Name: ques2_list.py | Code by: A. Nasra
# Program to find the mean of a list of numbers.
L1 = eval(input('Enter a number list:'))
print('Mean = ',sum(L1)/len(L1))
```

```
# Sample run of ques2_list.py
Enter a number list:[98.7, 87.6, 76.5, 65, 54, 43, 32]
Mean =  65.25714285714285
>>> # Verification:
>>> (98.7 + 87.6 + 76.5 + 65 + 54 + 43 + 32)/7
65.25714285714285
```

Example: 185 → Write a program to search for an element in a given list of numbers with the location.

```
# File Name: ques3_list.py | Code by: A. Nasra
L1 = eval(input('Enter a list of numbers:'))
n  = eval(input('Enter an element to find:'))
if n not in L1:
    print(n,'is not found in the list.')
elif n in L1:
    print(n,'is found at index:',L1.index(n))
```

```
# Sample run -1 of ques3_list.py
Enter a list of numbers:[74, 78, 74.1, 78.9, 89, 85]
Enter an element to find:78.9
78.9 is found at index: 3
# Sample run -2 of ques3_list.py
Enter a list of numbers:[74, 78, 74.1, 78.9, 89, 85]
Enter an element to find:98
98 is not found in the list.
```

Example: 186 → Write a program to count the frequency of a given element in a list of numbers.

```
# File Name: ques4_list.py | Script by: A. Nasra
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# program to count the frequency of element in a list.
L1 = eval(input('Enter a List:'))
n  = eval(input('Enter a element:'))
if n not in L1:
    print('Element is not in the list!')
else:
    print(n,'has frequency:',L1.count(n))
```

```
# Sample run -1 of ques4_list.py
Enter a List:[23, 45, 6, 78, 33, 78, 98, 89, 98, 56,
23, 78, 98, 98, 78, 78]
Enter a element:98
98 has frequency: 4
# Sample run -2 of ques4_list.py
Enter a List:['a', 'e', 'i', 'o', 'u', 'e', 'i', 'u',
'e', 'o']
Enter a element:'e'
e has frequency: 3
```

Example: 187 → Given two lists. Write a program that prints "overlapped" if they have at least one member in common otherwise print "seperated".

```
# File Name: ques6_list.py | Code by: A. Nasra
L1 = eval(input('Enter 1st List:'))
L2 = eval(input('Enter 2nd List:'))
for element in L1:
    if element in L2:
        print('Overlapped')
        break
else:
    print('Seperated')
```

```
# Sample run -1 of ques6_list.py
Enter 1st List:[1,2,3,4]
Enter 2nd List:[5,6,7,8,9,4]
Overlapped
#  Sample  run  -2  of  ques6_list.py  Enter  1st
List:[1,2,3,4]
Enter 2nd List:[5,6,7,8,9]
Seperated
# Sample run -3 of ques6_list.py
Enter 1st List:[5,6,7,8,9]
Enter 2nd List:[1,2,3,4]
Seperated
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 188 → Write a program to find the second largest number of a list of numbers entered by the user.

```
# File Name: ques7_list.py | Script by: A. Nasra
# program to find 2nd largest value in a number list.
L1 = eval(input('Enter a number list: '))
L1.sort()        # sorts in ascending order in place.
print('2nd largest number =',L1[-2])
```

```
# Sample run of ques7_list.py
Enter a number list: [23,32,45,45,65,56,11,22,33]
2nd largest number = 56
```

Example: 189 → Write a program that inputs a list of numbers and shifts all the zeros to right and all non-zero to right of the list.

```
# File Name: ques8_list.py | Script by: A. Nasra
L1 = eval(input('Enter a number list: '))
L2, L3 = [], []
for element in L1:
    if element == 0:
        L2.append(element)
    else:
        L3.append(element)
L2.extend(L3)
print('Desired list is:',L2)
```

```
# Sample run of ques8_list.py
Enter a number list: [0,5,4,0,2,3,0,6,0]
Desired list is: [0, 0, 0, 0, 5, 4, 2, 3, 6]
```

Example: 190 → Write a program that inputs two lists and creates a third, that contains all elements of the first followed by all elements of second.

```
# File Name: ques9_list.py | Script by: A. Nasra
L1 = eval(input('Enter 1st list: '))
L2 = eval(input('Enter 2nd list: '))
L1.extend(L2)
print('Desired list is:',L1)
```

```
# Sample run -1 of ques9_list.py
Enter 1st list: [1, 2, 3, 4]
Enter 2nd list: [7, 6, 5, 4, 3, 2, 1]
Desired list is: [1, 2, 3, 4, 7, 6, 5, 4, 3, 2, 1]
# Sample run -2 of ques9_list.py Enter 1st list: [1,
2, 3, 4, 5]
Enter 2nd list: [4,3,2,1]
Desired list is: [1, 2, 3, 4, 5, 4, 3, 2, 1]
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run -3 of ques9_list.py Enter 1st list:
['mercury', 'venus']
Enter 2nd list: ['earth', 'mars', 1, 2, 3, 4]
Desired list is: ['mercury', 'venus', 'earth', 'mars',
1, 2, 3, 4]
```

Example: 191 → Write a program that asks the user to enter a list containing numbers between 1 and 12. Then replace all of the entries in the list that are greater than 10 with 10.

```
# File Name: ques10_list.py | Script by: A. Nasra
L1 = eval(input('Enter a list between 1 and 12: '))
for i in range(len(L1)):
    if L1[i]<0 or L1[i]>11:
        print('Wrong entry! Enter a list between 1 and
12')
        exit()
    elif L1[i]>1 and L1[i]<12:
        if L1[i]>=10:
            L1[i] = 10
print('Desired list is:',L1)
```

```
# Sample run -1 of ques10_list
Enter a list between 1 and 12:
[3,4,10,10,11,4,9,10,7,11]
Desired list is: [3, 4, 10, 10, 10, 4, 9, 10, 7, 10]
# Sample run -2 of ques10_list
Enter a list between 1 and 12: [3, 4, 10,10,12,7, 10]
Wrong entry! Entery a list between 1 and 12
```

Example: 192 → Write a program asks the user to enter a list of strings. Create a new list that consists of those strings woth their first characters removed.

```
# File Name: ques11_list.py | Code by: A. Nasra
L1 = eval(input('Enter a list of strings: '))
L2 = list()
i = 0
for element in L1:
    elem = element[1:len(element)]
    i == L1.index(element)
    L2.insert(i,elem)
    # append() or extend() doesn't work.
L2.reverse()
print('The desired lis is:',L2)
```

```
# Sample run of ques11_list.py
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Enter a list of strings: ['A.', 'Nasra', 'shell', 'GRAM', 'edit']
The desired lis is: ['.', 'asra', 'hell', 'RAM', 'dit']
```

Example: 193 → Write a program to create the following lists:

(a) A list consisting of the integers 0 through 50
(b) A list containing the squares of the integers 1 though 50
(c) A list ['a', 'b', 'ccc', 'dddd', . . .]

```
# File Name: ques12_list.py | Code by: A. Nasra
La = []
for i in range(0,50):
    La.append(i)
print('(a) Desired list is:\n',La,'\n')

Lb = []
for j in range(1,51):
    Lb.append(j*j)
print('(b) Desired list is:\n',Lb,'\n')

S1 =  "abcdefghijklmnopqrstuvwxyz"
letter = list(S1)
Lc = []
for i in range(0,26):
    Lc.append(letter[i]*(i+1))
```

```
# Sample run of ques12_list.py
(a) Desired list is:
 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49]

(b) Desired list is:
 [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169,
196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576,
625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156,
1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849,
1936, 2025, 2116, 2209, 2304, 2401, 2500]

(c) Desired list is:
 ['a',  'bb',  'ccc',  'dddd',  'eeeee',  'ffffff',
'ggggggg',  'hhhhhhhh',  'iiiiiiiii',  'jjjjjjjjjj',
'kkkkkkkkkkk',  'lllllllllll',  'mmmmmmmmmmmmm',
'nnnnnnnnnnnnn',  'ooooooooooooooo',
'pppppppppppppppp',  'qqqqqqqqqqqqqqqq',
'rrrrrrrrrrrrrrrrr', 'sssssssssssssssss',
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
'ttttttttttttttttttt',        'uuuuuuuuuuuuuuuuuuuuu',
'vvvvvvvvvvvvvvvvvvvvv',    'wwwwwwwwwwwwwwwwwwwwww',
'xxxxxxxxxxxxxxxxxxxxxx',
'yyyyyyyyyyyyyyyyyyyyyy',
'zzzzzzzzzzzzzzzzzzzzzz']
```

Example: 194 → Write a program that takes two lists L and M of same size and type and adds their corresponding elements together to form a new list N. for instance, if L = [7, 9, 2] and M = [5, 6, 7] then N should be [12, 15, 9].

```
# File Name: ques13_list.py | Code by: A. Nasra
L = eval(input('Enter 1st list: '))
M = eval(input('Enter 2nd list: '))
N = []
if len(L) == len(M):
    for i in range(len(L)):
        add = L[i] + M[i]
        N.append(add)
    print('N =',N)
else:
    print('Both lists are not of equal length!')
```

```
# Sample run -1 of ques13_list.py
Enter 1st list: [4,5,6,7,8,9]
Enter 2nd list: [6,5,4,3,2,1]
# Sample run -2 of ques13_list.py
Enter 1st list: [4,5,6,7,8,9]
Enter 2nd list: [6,5,4,3,2,1]
N = [10, 10, 10, 10, 10, 10]
# Sample run -3 of ques13_list.py
Enter 1st list: ['SD','DD','A.','1']
Enter 2nd list: ['RAM','RAM','Nasra','st']
N = ['SDRAM', 'DDRAM', 'A.Nasra', '1st']
# Sample run -4 of ques13_list.py
Enter 1st list: [1, 2, 3, 4]
Enter 2nd list: [6, 5, 4, 3, 2, 1]
Both lists are not of equal length!
```

Example: 195 → Write a program to the sum of two given matrices.

```
# File Name: ques14_list.py | Script by: A. Nasra
# Let 2 matrix be A and B of 3x3 size
c = 3;  r = 3
A = [[0 for j in range(c)] for i in range(r)]
B = [[0 for j in range(c)] for i in range(r)]
C = [[0 for j in range(c)] for i in range(r)]
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
print('Enter elements of matrix A:')
for i in range(r):
    for j in range(c):
        print('A[',i,'][',j,'] = ', end='' )
        A[i][j] = eval(input())

print('Enter elements of matrix B:')
for i in range(r):
    for j in range(c):
        print('B[',i,'][',j,'] = ', end='')
        B[i][j] = eval(input())

for i in range(r):
    for j in range(c):
        C[i][j] = A[i][j] + B[i][j]

print('Sum of two matrices entered by you is:')
print(' ',A,'\n+',B,'\n=',C)
```

```
# Sample run of ques14_list.py
Enter elements of matrix A:
A[ 0 ][ 0 ] = 1
A[ 0 ][ 1 ] = 2
A[ 0 ][ 2 ] = 3
A[ 1 ][ 0 ] = 4
A[ 1 ][ 1 ] = 5
A[ 1 ][ 2 ] = 6
A[ 2 ][ 0 ] = 7
A[ 2 ][ 1 ] = 8
A[ 2 ][ 2 ] = 9
Enter elements of matrix B:
B[ 0 ][ 0 ] = 1
B[ 0 ][ 1 ] = 2
B[ 0 ][ 2 ] = 3
B[ 1 ][ 0 ] = 4
B[ 1 ][ 1 ] = 5
B[ 1 ][ 2 ] = 6
B[ 2 ][ 0 ] = 7
B[ 2 ][ 1 ] = 8
B[ 2 ][ 2 ] = 9
Sum of two matrices entered by you is:
  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
+ [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
= [[2, 4, 6], [8, 10, 12], [14, 16, 18]]
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 196 → Write a program to find the product of two matrices or order

```
# File Name: ques15_list.py | Script by: A. Nasra
# A program to find the product of two matrices.
'''The product of A and B will be possible only when
   the No. of col of A == No. of row of B, and the
   resulting matrix will of size(row of A)x(col of B)
   Let 2 matrix be A of size 2x3and B of size 3x3.'''

r1, c1 = 2, 3 ; r2, c2 = 3, 3 ; r3, c3 = 2, 3
A = [[0 for j in range(c1)] for i in range(r1)]
B = [[0 for j in range(c2)] for i in range(r2)]
C = [[0 for j in range(c3)] for i in range(r3)]

print('Enter elements of matrix A:')
for i in range(r1):
    for j in range(c1):
        print('A[',i,'][',j,'] = ', end='' )
        A[i][j] = eval(input())

print('Enter elements of matrix B:')
for i in range(r2):
    for j in range(c2):
        print('B[',i,'][',j,'] = ', end='')
        B[i][j] = eval(input())
for i in range(r3):
    for j in range(c3):
        for k in range(c1): # Note the behaviour of k
            C[i][j] += A[i][k] * B[k][j]
print('Product of two matrices entered by you is:')
print(' ',A,'\nX',B,'\n=',C)
```

```
# Sample run of ques15_list.py
Enter elements of matrix A:
A[ 0 ][ 0 ] = 1
A[ 0 ][ 1 ] = 2
A[ 0 ][ 2 ] = 3
A[ 1 ][ 0 ] = 4
A[ 1 ][ 1 ] = 5
A[ 1 ][ 2 ] = 6
Enter elements of matrix B:
B[ 0 ][ 0 ] = 3
B[ 0 ][ 1 ] = 2
B[ 0 ][ 2 ] = 1
B[ 1 ][ 0 ] = 1
B[ 1 ][ 1 ] = 1
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
B[ 1 ][ 2 ] = 1
B[ 2 ][ 0 ] = 1
B[ 2 ][ 1 ] = 1
B[ 2 ][ 2 ] = 1
Product of two matrices entered by you is:
  [[1, 2, 3], [4, 5, 6]]
X [[3, 2, 1], [1, 1, 1], [1, 1, 1]]
= [[8, 7, 6], [23, 19, 15]]
```

Example: 197 → Write a program to input a matrix and print both diagonal values of the matrix. Note that the diagonal exists only in square matrix. Square matrix is a matrix in which, Number of row = Number of column.

```
# File Name: ques16_list.py | Script by: A. Nasra
# Program to find the diagonal elements of a matrix.
'''Note that the diagonal exists only in square
   matrix, i.e. a matrix in which, No. of = No. of
   column.'''
r = int(input('Enter No. of row in the matrix: '))
c = int(input('Enter No. of col in the matrix: '))
A = [[0 for j in range(c)] for i in range(r)]

if r == c:
    print('Enter elements of matrix:')
    for i in range(r):
        for j in range(c):
            print('A[',i,'][',j,'] = ', end='')
            A[i][j] = eval(input())

    print('The first diagonal is:')
    for i in range(r):
        for j in range(c):
            if i == j:
                print(A[i][j],end=', ')

    print('\nThe second diagonal is:')
    k = c-1
    for i in range(c):
        print(A[i][k],end=', ')
        k = k-1
else:
    print('''There is no diagonal, since
             No. of row is not equal to No. of col''')
```

```
# Sample run of ques16_list.py
Enter No. of row in the matrix: 3
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Enter No. of col in the matrix: 3
Enter elements of matrix:
A[ 0 ][ 0 ] = 1
A[ 0 ][ 1 ] = 2
A[ 0 ][ 2 ] = 3
A[ 1 ][ 0 ] = 4
A[ 1 ][ 1 ] = 5
A[ 1 ][ 2 ] = 6
A[ 2 ][ 0 ] = 7
A[ 2 ][ 1 ] = 8
A[ 2 ][ 2 ] = 9
The first diagonal is:
1, 5, 9,
The second diagonal is:
3, 5, 7,
```

Example: 198 → Write a program to input N×M matrix and find sum of all even numbers in the matrix.

```
# File Name: ques17_list.py | Code by: A. Nasra
M = int(input('Number of row = '))
N = int(input('Number of col = '))
sum = 0
A = [[0 for j in range(N)] for i in range(M)]
print('Enter integers as elements of matrix')
for i in range(M):
    for j in range(N):
        print('A[',i,'][',j,'] = ', end='' )
        A[i][j] = eval(input())
        if A[i][j]%2 == 0:
            sum += A[i][j]
print('Sum of even elements =',sum)
```

```
# Sample run of ques17_list.py
Number of row = 2
Number of col = 3
Enter integers as elements of matrix
A[ 0 ][ 0 ] = 12
A[ 0 ][ 1 ] = 2
A[ 0 ][ 2 ] = 7
A[ 1 ][ 0 ] = 8
A[ 1 ][ 1 ] = 9
A[ 1 ][ 2 ] = 13
Sum of even elements = 22
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 199 → Write a program to print upper triangle matrix and lower triangle matrix for a given matrix

```
# File Name: loup_triangle.py | Code by: A. Nasra
r = 3 ; c = 3
U = [] ; L = []
A = [[0 for j in range(c)] for i in range(r)]
print('Enter integers as elements of matrix')
for i in range(r):
    for j in range(c):
        print('A[',i,'][',j,'] = ', end='' )
        A[i][j] = eval(input())
        if i <= j:
            U.append(A[i][j])
        if i >= j:
            L.append(A[i][j])
print('Upper triangle elements are:',U)
print('Lower triangle elements are:',L)
```

```
# Sample run of loup_traingle.py
Enter integers as elements of matrix
A[ 0 ][ 0 ] = 1
A[ 0 ][ 1 ] = 2
A[ 0 ][ 2 ] = 3
A[ 1 ][ 0 ] = 4
A[ 1 ][ 1 ] = 5
A[ 1 ][ 2 ] = 6
A[ 2 ][ 0 ] = 7
A[ 2 ][ 1 ] = 8
A[ 2 ][ 2 ] = 9
Upper triangle elements are: [1, 2, 3, 5, 6, 9]
Lower triangle elements are: [1, 4, 5, 7, 8, 9]
```

Example: 200 → Write a program to find sum rows-sum and columns-sum of a given matrix.

```
# File Name: col_row_sum.py | Code By: A. Nasra
r = int(input('No. of row: '))
c = int(input('No. of col: '))
M = [[0 for j in range(c)]for i in range(r)]

print('Enter integers as elements of matrix')
for i in range(r):
    for j in range(c):
        print('M[',i,'][',j,'] = ', end='' )
        M[i][j] = eval(input())
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
def rsum(i):
    rtotal = 0
    for j in range(c):
        rtotal += M[i][j]
    print('sum of row-',i,'=',rtotal)

def csum(j):
    ctotal = 0
    for i in range(r):
        ctotal += M[i][j]
    print('sum of col-',j,'=',ctotal)

for i in range(r):
    rsum(i)

for j in range(c):
    csum(j)
```

```
# Sample run of col_row_sum.py
No. of row: 2
No. of col: 3
Enter integers as elements of matrix
M[ 0 ][ 0 ] = 10
M[ 0 ][ 1 ] = 20
M[ 0 ][ 2 ] = 30
M[ 1 ][ 0 ] = 9
M[ 1 ][ 1 ] = 8
M[ 1 ][ 2 ] = 7
sum of row- 0 = 60
sum of row- 1 = 24
sum of col- 0 = 19
sum of col- 1 = 28
sum of col- 2 = 37
```

Example: 201 → Write a program that prints out the two largest and two smallest elements of a list called scores. (Hint: sort the list)

```
# File Name: max_min2.py | Code by: A. Nasra
scores = eval(input('Enter a list of scores: '))
n = len(scores)
scores.sort()
print('Two minimum scores are:',scores[0],'and',
scores[1])
print('Two maximum scores are:',scores[n-1], 'and',
scores[n-2])
```

```
# Sample run of max_min2.py
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Enter a list of scores: [56,67,76,65,89,97,79,43]
Two minimum scores are: 43 and 56
Two maximum scores are: 97 and 89
```

Example: 202 → Write a program to play a simple quiz game.

```
# File name: quiz_list.py | Made by: A. Nasra
ques = ['What is the capital of France? ',
        'What is sum of 2 and 3.3 ? ',
        'What is the capital of India? ',
        'Is [] an empty list? (yes/no) ']
ans = ['Paris','5.3','Delhi','Yes']
right_ans = 0
for i in range(len(ques)):
    guess = input(ques[i])
    if guess.lower() == ans[i].lower():
        print('Correct')
        right_ans = right_ans + 1
    else:
        print('Wrong. The answer is', ans[i])
print('\u2014'*45)
print('You     answered     correctly',right_ans,'out
of',len(ques),'questions')
print('You got',right_ans*100/len(ques),'%.')
print('\u2014'*45)
```

```
What is the capital of France? paris
Correct
What is sum of 2 and 3.3 ? 5
Wrong. The answer is 5.3
What is the capital of India? delhi
Correct
Is [] an empty list? (yes/no) no
Wrong. The answer is Yes
_____________________________________________

You answered correctly 2 out of 4 questions
You got 50.0 %.
_____________________________________________
```

Example: 203 → Write a program that asks the user to enter an integer and creates a list that consists of the factors of that integer.

```
# File Name: factor.py | Code by: A. Nasra
print('This program finds the factors of an integer.')
n = eval(input('Enter an integer: '))
factors = []
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
for i in range(1,n+1):
    if n%i == 0:
        factors.append(i)
print('Factors of',n,'=',factors)
```

```
# Sample run of factor.py
This program finds the factors of an integer.
Enter an integer: 12
Factors of 12 = [1, 2, 3, 4, 6, 12]
```

Example: 204 → Write a program that generates a list L of 50 random numbers between 1 and 100.

```
# File Name: random_list.py | Code by: A. Nasra
from random import randint
L = []
for i in range(50):
    L.append(randint(1,100))
print(L)
```

```
# Sample runoff random_list.py
[7, 13, 50, 43, 29, 95, 2, 75, 70, 43, 62, 93, 11, 65,
1, 45, 56, 86, 93, 33, 21, 19, 52, 40, 10, 62,
50, 75, 42, 19, 30, 27, 1, 94, 60, 96, 84, 39, 89, 93,
74, 96, 75, 12, 77, 79, 92, 76, 70, 75]
```

Example: 205 → Write a Given a list L that contains numbers between 0 and 10 exclusively, create a new list whose first element is how many ones are in L, whose second element is how many twos are in L, and so on.

```
# File Name: ques199_list.py | Code by: A. Nasra
GL = eval(input('Enter a list of integers between 0 and
10 exclisively.\n'))
FL = [] # GL for Given list, FL for frequency list
for i in range(1,11):
    FL.append(GL.count(i))
print('The list as per requirement is:')
print(FL)
```

```
# Sample run of ques199_list.py
Enter a list of integers between 0 and 10 exclusively.
[9,7,5,3,1,3,2,6,5,8,9,7,2,2,3,2]
The list as per requirement is:
[1, 4, 3, 0, 2, 1, 2, 1, 2, 0]
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

TUPLE MANIPULATION

DEFINITION OF TUPLE

Tuple is a sequence of values of any type. These values are called elements or items. Elements of tuple are immutable (non-changeable in place) and indexed by integers. List is enclosed in ( ) brackets.

ADVANTAGES OF TUPLE OVER LIST

- ✓ We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- ✓ Since tuple are immutable, iterating through tuple is faster than with list. S
- ✓ Tuples contain immutable elements, so it be used as key for a dictionary. With list, this is not possible.
- ✓ If we have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

SIMILARITY OF TUPLE WITH LIST

1. Like list, elements of tuples are also indexed in the same manner. They can be positively and negatively indexed. Thus, **`tuple[i]`** returns the element or item at the index **`i`**, so, it is used for accessing individual element of the list.
2. Like list, slicing in tuple is done in the same way. Thus, **`tuple[i,j,n]`** returns every **n^{th}** items from **`i`** to **`j`**.
3. The function **`len(tuple)`** returns the length of the tuple.
4. The membership operators **`in`** and **`not in`** works on tuple like other sequences (i.e. list and string).
5. Concatenation operator (+) and Replication operator (*) in the same way as in any sequence (i.e. series and list).

CREATING TUPLE

A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it. A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

Example: 206 → Creation of tuples.

```
>>> T1 = () # empty tuple
>>> T2 = (1, 3, 5, 7) # tuple of integers
>>> T3 = (5.7, 'Hi Tup', 5) # tuple of mixed datatype
>>> T4 = (4,'tup',(1,2,3),[9,8,7]) # nested tuple
>>> T5 = 5,'Tup',1,0,7 # tuple without parentheses
>>> T6 = 7,  #single element tuple
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> T7 = (7,)#single element tuple,(7) is int not tuple
>>> print(' T1 =',T1,'\n','T2 =',T2,'\n','T3 =', T3,
    '\n','T4 =',T4,'\n','T5 =' ,T5, '\n', 'T6=', T6,
    '\n','T7 =' ,T7)
 T1 = ()
 T2 = (1, 3, 5, 7)
 T3 = (5.7, 'Hi Tup', 5)
 T4 = (4, 'tup', (1, 2, 3), [9, 8, 7])
 T5 = (5, 'Tup', 1, 0, 7)
 T6 = (7,)
 T7 = (7,)
```

CREATING TUPLE FROM EXISTING SEQUENCE

We can create a tuple from the sequence, with the help of built-in function **`tuple()`** according to the syntax: **`T = tuple(sequence)`**

Example: 207 → Creation of tuples from existing tuple.

```
>>> T1 = tuple('LKCC')
>>> T2 = tuple(['L', 'K', 'C', 'C'])
>>> T3 = tuple((1, 2, 3, 4))
>>> print(' T1 =',T1,'\n','T2 =',T2,'\n', 'T3 =', T3)
 T1 = ('L', 'K', 'C', 'C')
 T2 = ('L', 'K', 'C', 'C')
 T3 = (1, 2, 3, 4)
```

Example: 208 → Creation of tuples by user.

```
# File Name: tuple_by_user.py | By: Lallu
# Tuple creation by user
T4 = tuple(input('Enter elements of tuple: '))
T5 = tuple(input('Enter tuple elements: '))
T6 = eval(input('Enter tuple: '))
print('1st tuple is:', T4)
print('2nd tuple is:', T5)
print('3rd tuple is:', T6)
```

```
# Sample run of tuple_by_user.py
Enter elements of tuple: LKCC
Enter tuple elemets: LKCC lkcc
Enter tuple: (1, 2, 3, 4, 5)
1st tuple is: ('L', 'K', 'C', 'C')
2nd tuple is: ('L','K','C','C',' ','l', 'k','c', 'c')
3rd tuple is: (1, 2, 3, 4, 5)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

ACCESSING TUPLE

We can use the index operator `[]` to access an item in a tuple where the index starts from 0. The index must be an integer, so we cannot use float or other types. This will result into TypeError.
Likewise, nested tuple are accessed using nested indexing, as shown in the example below.

Example: 209 → Accessing elements of tuples.

```
# File name: access_tuple.py | Code by: A. Nasra
print('\u26AF'*28)
T1 = (2, 3.5, 2 + 3j, 7, 12345)
print('T1 =', T1)
print('T1[0] =', T1[0], '\tT1[2] =', T1[2])
print('~'*52)
T2 = (12345, [7, 6, 5, 4],(1.1, 2.2, 3.3), 'zing')
print('T2 =', T2)
print('T2[1] =',T2[1], '\nT2[2] =',T2[2])
print('T2[1][3] =' ,T2[1][3],'\nT2[2][1] =', T2[2][1])
print('T2[3][3] =', T2[3][3])
print('\u26AF'*28)
```

```
# Sample run of access_tuple.py
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
T1 = (2, 3.5, (2+3j), 7, 12345)
T1[0] = 2    T1[2] = (2+3j)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
T2 = (12345, [7, 6, 5, 4], (1.1, 2.2, 3.3), 'zing')
T2[1] = [7, 6, 5, 4]
T2[2] = (1.1, 2.2, 3.3)
T2[1][3] = 4
T2[2][1] = 2.2
T2[3][3] = g
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

TUPLE OPERATIONS

The most common operations with tuples are joining tuples, replicating tuple and slicing the tuple.

JOINING TUPLES

The concatenation operator (**+**) is used to join two or more tuples. We can easily understand this by the following example.
The **+** operator results into a new tuple, because tuples are immutable.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 210 → Program to join two or more tuples.

```
>>> T1 = (1,5,9,4) ; T2 = (2,5,7,8)
>>> print('T1 + T2 =', T1+T2)
T1 + T2 = (1, 5, 9, 4, 2, 5, 7, 8)
>>> T3 = ((1,2), 'LKCC', 3.5)
>>> print('T1 + T3 =', T1 + T3)
T1 + T3 = (1, 5, 9, 4, (1, 2), 'LKCC', 3.5)
```

REPLICATING TUPLES

We can also **repeat** the elements in a tuple for a given number of times using the replication operator (*) . The replication operations result into a new tuple.

Example: 211 → Program replicate (repeat) a tuples.

```
>>> T1 = ('more','study')*3
>>> print('T1 =',T1)
T1 = ('more', 'study','more','study','more','study')
>>> T2 = ('more','study')
>>> T3 = T2*3
>>> print('T3 =',T3)
T3 = ('more','study','more','study','more',)
```

SLICING TUPLES

The tuple slices are the subpart of a given tuple. Slice is obtained by using the colon (:) operator, according to the format given below.

```
T2 = T1(start, stop, step)
```

Where, **T1** is given tuple;
T2 is sliced tuple (subpart of given tuple);
start is the starting index (included);
stop is the ending index (excluded) and
step is the interval of index (by default it is 1).

If the value of index is out of range, Python raises **IndexError** exception.

Example: 212 → Program for slicing a tuples.

```
>>> T1 = (77, 88, 66, 11, 99, 22)
>>> print('T3 =', T1[2:4])
T3 = (66, 11)
>>> print('T3 =', T1[1:-2])
T3 = (88, 66, 11)
>>> print('T2 =', [3:-3])
T2 = ()
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

TUPLE MEMBERSHIP OPERATION

We can use membership operator **in** and **not in** to test whether a particular item is in a tuple or not in a tuple.

Example: 213 → Use of membership operator in list.

```
>>> T1 = (12, 21, 23, 32, 34, 43)
>>> print(32 in T1)
True
>>> print( 42 not in T1)
True
>>> print(42 in T1)
False
```

CHANGING A TUPLE

We know the like lists, tuples are not mutable. This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.
We can also assign a tuple to different values (reassignment). Item of mutable element can be changed.

Example: 214 → Program for changing a tuples.

```
# File Name: mutable_tuple.py | Code by: A. Nasra
print('''We cannot change an element of tuple(immutable)but item of mutable element can be changed.''')
T1 = (4, 2, 3, [6, 5])
print('T1 =', T1)
T1[3][0] = 9
print('T1 =', T1)

print('\u2014'*50)
T2 = ('L','K','C','C','F','O','R','U')
print('Tuple before reasignment\n T2 =',T2)
T2 = ('L','K','M','C','F','O','R','U')
print('Now, after reassignmnet\n T2 =',T2)
```

```
# Sample run of mutable_tuple.py
We cannot change an element of tuple(immutable)
but item of mutable element can be changed.
T1 = (4, 2, 3, [6, 5])
T1 = (4, 2, 3, [9, 5])
——————————————————————————————————————————————
Tuple before reasignment
 T2 = ('L', 'K', 'C', 'C', 'F', 'O', 'R', 'U')
Now, after reassignmnet
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
T2 = ('L', 'K', 'M', 'C', 'F', 'O', 'R', 'U')
```

Example: 215 → Program for changing a tuples in elegant way.

```
>>> T1 = (1,2,3)
>>> T2 = (1,4,3)
>>> T1, T2 = T2, T1
>>> print('T1 =', T1, '\nT2 =', T2)
T1 = (1, 4, 3)
T2 = (1, 2, 3)
```

COMPARING TUPLES

Two lists are compared with the help of comparison operators (`<, >, ==, !=`) in lexicographical order.
Note: For two tuples to be equal, they must have same number of elements and matching values.

Example: 216 → Illustration of comparison between lists

```
>>> T1 = (1, 2, 8, 9)
>>> T2 =(9,1)
>>> T3 =(1,2,9,1)
>>> T4 =(1,2,9,10)
>>> T5 =(1,2,8,4)
>>> T1 < T2 ; T1 < T3 ; T1 < T4 ; T1 < T5
True
True
True
False
```

PACKING AND UNPACKING TUPLES

Creating a tuple from a set of values is called packing, while its reverse, that is finding individual values from the tuple is called unpacking. Unpacking is done according to the following syntax.

```
variable1, variable2, variable3 = Tuple_Name
```

Example: 217 → Illustration of packing and unpacking of tuple.

```
# File Name: unpack_tuple.py | Code by: A. Nasra
x = ("LKCC", 20, "Education")     # tuple packing
(company, emp, profile) = x     # tuple unpacking
print('company:',company)
print('emp:',emp)
print('profile:',profile)
```

```
# Sample run of unpack_tuple.py
company: LKCC
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
emp: 20
profile: Education
```

DELETING A TUPLE

As we know that a tuple is immutable, so the elements cannot be deleted or removed, but deleting the entire tuple is possible using del statement.

Example: 218 → Illustration of deletion of a tuple.

```
>>> planets = ('mercury','venus','earth','marsh')
>>> del planets
>>> planets
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module> planets
NameError: name 'planets' is not defined
>>> # from the above error it's clear that
>>> # the tuple planets is removed.
```

TRAVERSING A TUPLE

Traversing a tuple means accessing and processing each element of the tuple.

Example: 219 → Illustration of traversing a tuple.

```
# File Name: traverse_tuple.py | Code by: A. Nasra
name = eval(input('Enter a tuple of some trees: '))
for element in name:
    print('Hello',element,'tree.')
```

```
# Sample run of traverse_tuple.py
Enter a tuple of some trees: ('apple','mango','banana')
Hello apple tree.
Hello mango tree.
Hello banana tree.
```

Example: 220 → Illustration of traversing a tuple.

```
# File Name: traverse_tuple2.py
name = eval(input('Enter a tuple of some trees: '))
for i in range(len(name)):
    print(i,'-',name[i])
```

```
# Sample run of traverse_tuple2.py
Enter a tuple of some trees: ('banana','neem','fig')
0 - banana
1 - neem
2 - fig
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

TUPLE FUNCTIONS AND METHODS

LEN( )

The **len()** method returns the length of a tuple'.

Syntax: **len(tuple_name)**

Example: 221 → Finding the length of a tuple.

```
>>> T1 =(7,'wonders', 7.8, 3+2j, 987123)
>>> print('Length of tuple is:', len(T1))
Length of tuple is: 5
```

COUNT( )

The **count()** method returns the number of occurrences of an element in a list.

The syntax of **count()** method is: **tuple.count(element)**

The **count()** method takes a single argument and returns the number of occurrences of an element in a list.

Example: 222 → Illustration of **count()** method to count an element in a tuple.

```
>>> T1 =(123,'abc',1.1,2.2,123,'def',123,'abc','def')
>>> n = T1.count(123)
>>> print('The count of 123 is:',n)
The count of 123 is: 3
>>> n = T1.count('abc')
>>> print('The count of abc is:',n)
The count of abc is: 2
```

INDEX( )

The **index()** method searches an element in the tuple and returns its index/position. However, if the same element is present more than once, **index()** method returns its smallest/first position.

The syntax of **index()** method for list is: **tuple.index(element)**

Where, **element** is the value to be searched.

The **index()** method returns the **index** of the **element** in the tuple.

If not found, it raises a **ValueError** exception indicating the element is not in the list.

Example: 223 → Illustration of **tuple.index()** method.

```
>>> T1 = [11, 22, 33, 44, 55]
>>> T1.index(33)
2
>>> T1.index(32)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    T1.index(32)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
ValueError: 32 is not in tuple
```

MAX( )

The **max()** method returns an element from a tuple, which has maximum value.
Its syntax is: **max(tuple_name)**

Example: 224 → Illustration of **max(tuple_name)** method with tuple.

```
>>> T1 = (45,78,87,98,55,86.68)
>>> max(T1)
98
```

MIN( )

The **min()** method returns an element from a tuple, which has maximum value.
Its syntax is: **min(tuple_name)**

Example: 225 → Illustration of **min(tuple_name)** method with tuple.

```
>>> T1 = (45,78,87,98,55,86.68)
>>> min(T1)
45
```

TUPLE( )

The **tuple()** is a constructor method. It converts a sequence (string, list, set, dictionary) into a tuple.
Its syntax is: **tuple(sequence_name)**
If no argument is given, it returns an empty tuple.

Example: 226 → Illustration of **tuple(sequence_name)** method.

```
# File Name: tuple_conversion.py | By: A. Nasra
T1 = tuple()        # empty tuple
print('T1 =', T1)

# creating a tuple from a list
T2 = tuple([2, 8, 6])
print('T2 =', T2)

# creating a tuple from a string
T3 = tuple('LKCC')
print('T3=',T3)

# creating a tuple from a dictionary
T4 = tuple({1: 'one', 2: 'two',3:'three'})
print('T4=',T4)
```

```
# Run of file tuple_conversion.py
T1 = ()
T2 = (2, 8, 6)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
T3= ('L', 'K', 'C', 'C')
T4= (1, 2, 3)
```

ENUMERATE( )

The **enumerate()** method adds counter to a sequence and returns the numerated object.

Tts syntax is: **enumerate(sequence, start=n)**

By default the start is 0, that is if start is omitted the enumeration start with 0.

We can convert enumerate objects to list and tuple using list() and tuple() method respectively.

Example: 227 → Illustration of **enumerate(sequence, start=n).**

```
# File Name: enum_tuple.py | By: A. Nasra
grocery = ('pulse', 'rice', 'atta', 'corn')
E1 = enumerate(grocery)
print(type(E1))
print(list(E1))

# changing the default counter
E1 = enumerate(grocery, 7)
print(list(E1))
```

```
# Run of enum_tuple.py
<class 'enumerate'>
[(0, 'pulse'), (1, 'rice'), (2, 'atta'), (3, 'corn')]
[(7, 'pulse'), (8, 'rice'), (9, 'atta'), (10, 'corn')]
```

Example: 228 → Illustration of **enumerate(sequence, start=n).**

```
# File Name: enum_tuple1.py | By: A. Nasra
temples = ('Badrinath', 'Konark', 'Somnath', Sanchi')
for item in enumerate(temples):
  print(item)

print('\n')
for count, item in enumerate(temples):
  print(count, item)

print('\n')
# changing default start value
for count, item in enumerate(temples, 100):
  print(count, item)
```

```
# Sample run of enum_tuple1.py
(0, 'Badrinath')
(1, 'Konark')
(2, 'Somnath')
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
(3, 'Sanchi')

0 Badrinath
1 Konark
2 Somnath
3 Sanchi

100 Badrinath
101 Konark
102 Somnath
103 Sanchi
```

SUM( )

The **sum()** function adds the items of a sequence and returns the sum.
The syntax of **sum()** is: **sum(sequence, start)**
Where, **sequence** may be a list, tuple, dict etc whose item's sum is to be found. Normally, items of the sequence should be numbers; **start** (optional) is the value that is added to the sum of items of the sequence. The default value of start is **0** (if omitted)
The **sum()** function returns the sum of start and items of the given sequence.

Example: 229 → Illustration of **sum()** method.

```
>>> T1 = (1,2,3,4,5,6,7,8,9,10)
>>> sum(T1)
55
>>> T2 = (1.1, 2.2, 3.3, 4,5,6,7,7)
>>> sum(T2)
35.6
>>> T3 = (1.23, 2.2, 3,5, 7+2j, 8+2j)
>>> sum(T3)
(26.43+4j)
>>> T4 = (5, 7, 9, 11)
>>> sum(T4, 8)
40
>>> T5 = (5, 10, 20) ; T6 = (5, 0, -10)
>>> sum(T5, sum(T6))
30
```

SORTED( )

The **sorted()** method sorts the elements of a given sequence in ascending or descending order.
Its syntax is: **sorted(sequence, key, reverse)**
It takes three parameters:
1. **sequence** - (string, tuple, list) or collection (set, dictionary, frozen set) or

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

any other sequence.

2. **reverse** – (Optional) If true, the sorted list is reversed (or sorted in Descending order)
3. **key** – (Optional) function that serves as a key for the sort comparison sorted() method returns a sorted list from the given sequence.

Example: 230 → Sorting the list.

```
# File Name: sorted_tuple1.py | By: A. Nasra
sense = ('see', 'smell', 'taste', 'hear', 'touch')
print('Given tuple is:\nsense =', sense)
print('Sorting in alphabetical order:\n',
      sorted(sense))
print('Sorting in reverse order:\n',
      sorted(sense, reverse=True))
print('Sorting as per length of string:\n',
      sorted(sense, key=len))
```

```
# Sample run of sorted_tuple1.py
Given tuple is:
sense = ('see', 'smell', 'taste', 'hear', 'touch')
Sorting in alphabetical order:
 ['hear', 'see', 'smell', 'taste', 'touch']
Sorting in reverse order:
 ['touch', 'taste', 'smell', 'see', 'hear']
Sorting as per length of string:
 ['see', 'hear', 'smell', 'taste', 'touch']
```

Example: 231 → Sorting the list.

```
# File Name: sorted_tuple2.py | Cd. by: A. Nasra
# taking first element to sort
def X(elm):
    return elm[0]
# random tuple
random = ((2, 2), (3, 4), (4, 1), (1, 3))
# sort tuple with key
sortedTup = sorted(random, key=X)
# print tuple
print('Sorted list:', sortedTup)

# taking 2nd element to sort
def Y(elm):
    return elm[1]
sortedTup = sorted(random, key=Y)
print('Sorted list:', sortedTup)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run of sorted_tuple2.py
Sorted list: [(1, 3), (2, 2), (3, 4), (4, 1)]
Sorted list: [(4, 1), (2, 2), (1, 3), (3, 4)]
```

TUPLE PROGRAMMING EXAMPLES

Example: 232 → Write a Python function secondLargest(T) which takes as input a tuple T and return the second largest element in the tuple.

```
# File Name: ques1_tuple.py | Cd by: A. Nasra
print('''Call the function secondLargest(T)
where, T is a tuple.''')
def secondLargest(T):
    T2 = sorted(T)
    print('Second largest value = ', end='')
    return T2[len(T)-2]
```

```
# Sample run of ques1_tuple.py
Call the function secondLargest(T)
where, T is a tuple.
>>> secondLargest((23,45,23,78,96,54))
Second largest value = 78
```

Example: 233 → Write a function getPower(x) that returns a tuple containing x, x^2, x^3 and x^4.

```
# File Name: ques2_tuple.py | Cd: A. Nasra
print(''' Call the function getPowers(x),
 where, x may be integer, float or complex number.''')
def getPowers(x):
    T = (x, x**2, x**3, x**4)
    print('Tuple = ', end='')
    return T
```

```
# Sample run of ques2_tuple.py
Call the function getPowers(x),
 where, x may be integer, float or complex number.
>>> getPowers(2)
Tuple = (2, 4, 8, 16)
>>> getPowers(2.5)
Tuple = (2.5, 6.25, 15.625, 39.0625)
>>> getPowers(1+1j)
Tuple = ((1+1j), 2j, (-2+2j), (-4+0j))
```

Example: 234 → Write a program that receives the index of the tuple entered by the user and returns the corresponding value.

```
# File Name: ques3_tuple.py | Cd: A. Nasra
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
T = eval(input('Enter a tuple T = '))
print(''' Call the function value(i)
 where, i is the index of the tuple T''')
def value(i):
    if i<0 or i>=len(T):
        print('Value of i is out of range')
    else:
        print('value(',i,') = ', end='')
        return T[i]
```

```
# Sample run of ques3_tuple.py
Enter a tuple T = (75.8, 95, 'wow!', 4+7j, 557649987)
 Call the function value(i)
 where, i is the index of the tuple T
>>> value(3)
value( 3 ) = (4+7j)
>>> value(2)
value( 2 ) = 'wow!'
>>> value(4)
value( 4 ) = 557649987
>>> value(5)
Value of i is out of range
```

Example: 235 → Write a program that interactively creates a nested tuple to store the marks in three subjects for five students, i.e. tuple will look somewhat like: Marks((46, 46, 41), (34, 39, 37), (37, 31, 40), (50, 54, 40), (20, 30, 40))
Also, add a function that computes total marks and average marks obtained by each student.

```
# File Name: ques4_tuple.py | Cd. by: A. Nasra
r = int(input('No. of students: '))
c = int(input('No. of subjects: '))
M = [[0 for j in range(c)]for i in range(r)]
N = []
for i in range(r):
    for j in range(c):
        print('Student-',i+1,'Marks  in  Sub-',j+1,'=
',end='')
        M[i][j] = eval(input())

for i in range(r):
    N.append(tuple(M[i]))
print('Marks  of',r,'students  in',c,'sub  are:\n',
tuple(N))

for i in range(r):
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
    s = sum(tuple(M[i]))
    print('Student-',i+1,'|  Total  Marks=',s,'|  Avg.
Marks =',round(s/c,2))
```

```
# Sample run of ques4_tuple.py
No. of students: 2
No. of subjects: 3
Student- 1 Marks in Sub- 1 = 12
Student- 1 Marks in Sub- 2 = 21
Student- 1 Marks in Sub- 3 = 34
Student- 2 Marks in Sub- 1 = 44
Student- 2 Marks in Sub- 2 = 33
Student- 2 Marks in Sub- 3 = 32
 Marks of 2 students in 3 sub are:
 ((12, 21, 34), (44, 33, 32))
Student- 1 | Total Marks= 67 | Avg. Marks = 22.33
Student- 2 | Total Marks= 109 | Avg. Marks = 36.33
```

Example: 236 → Write a program that returns the length of the shortest and longest string in the tuple of strings entered by the user.

```
# File Name: ques5_tuple.py | Cd by: A. Nasra
T1 = eval(input('Enter a tuple of strings below:\n '))
T2 = sorted(T1,key=len)
length = len(T2)
print('Shortest String:',T2[0])
print('Longest String :',T2[length-1])
```

```
# Sample run of ques5_tuple.py
Enter a tuple of strings below:
('teacher','actor','nun','scientist','artist','chef')
Shortest String: nun
Longest String : scientist
```

Example: 237 → Write a program to calculate the Standard Deviation for the numeric samples collected in a tuple namely Samples, according to the following formula:

$$S_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{N}}$$

```
# File Name: ques6_tuple.py | Cd by: A. Nasra
from math import sqrt
sample = eval(input('Enter a tuple of Numeric Sample
below:\n '))
N = len(sample)
mean = sum(sample)/N
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
m_sum = 0
for elm in sample:
    m_sum = m_sum - elm
sd = sqrt((m_sum)**2/N)
print("Standard Deviation =",round(sd,2))
```

```
# Sample run of ques5_tuple.py
Enter a tuple of Numeric Sample below:
 (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Standard Deviation = 17.39
```

Example: 238 → Write a program to enter any two tuples and interchange the tuple values.

```
# File Name: ques6_tuple.py | Cd by: A. Nasra
T1 = eval(input('Enter 1st tuple: '))
T2 = eval(input('Enter 2nd tuple: '))
T1, T2 = T2, T1
print('Swapped tuples are:')
print('T1 =', T1)
print('T2 =', T2)
```

```
# Sample run of ques6_tuple.py
Enter 1st tuple: ('all','students','passed')
Enter 2nd tuple: (17, 'got', 'above',95)
Swapped tuples are:
T1 = (17, 'got', 'above', 95)
T2 = ('all', 'students', 'passed')
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

DICTIONARIES

DEFINITION OF DICTIONARY

A dictionary is mutable, unordered collections of elements in the form of a **key:value** pairs that associate keys to values, enclosed within braces (curly brackets). The keys are unique within a dictionary. Dictionaries are also called *associative arrays* or *mappings* or *hashes*.

Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by **keys**, which can be any immutable type; strings and numbers can always be a keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. We can't use lists as keys, since lists can be modified in place.

CHARACTERISTICS OF A DICTIONARY

- ✓ Dictionary is an unordered collection of **key:value** pairs.
- ✓ Dictionary is not a sequence, because it not indexed like string, list or tuple.
- ✓ Dictionary is indexed by keys.
- ✓ Keys in dictionary are unique within the dictionary.
- ✓ Dictionary is mutable, like list. we can change the value of a certain key in place.
- ✓ Internally, the **key:value** pairs of a dictionary are stored as mapping. Keys are associated with values with the help of some internal function called hash-function.

CREATION OF DICTIONARY

Creating a dictionary is as simple as placing items (**key:value** pairs) inside curly braces {} separated by comma. In addition to it, there are other various ways for creating the dictionary, as is described in following lines.

Example: 239 → Program to create dictionary.

```
# File Name: dict_creation1.py | Cd by: A. Nasra
D1 = {} # empty dictionary
# dictionary with int keys
D2 = {1: 'Au', 2: 'Ag', 3:'Cu'}
# dictionary with str keys
D3 = {'Au':'Gold', 'Ag':'Silver', 'Cu':'Copper'}
# dictionary with mixed keys
D4 = {'name': 'nasra', 1: [2, 4, 3]}
print('Empty dictionary:', D1)
print('Dictionary with int keys:',D2)
print('Dictionary with str keys:\n',D3)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
print('Dictionary with mixed keys:',D4)
```

```
# Sample run of dict_creation1.py
Empty dictionary: {}
Dictionary with int keys: {1: 'Au', 2: 'Ag', 3: 'Cu'}
Dictionary with str keys:
 {'Au': 'Gold', 'Ag': 'Silver', 'Cu': 'Copper'}
Dictionary with mixed keys: {'name': 'nasra', 1:[2, 4,
3]}
```

The **dict()** constructor can also be used to create dictionaries directly from sequences of key-value pairs.

Example: 240 → Program to create dictionary with the help of **dict()**.

```
>>> L1 = [('a', 1), ('b', 2), ('c', 3), ('d',4)]
>>> D1 = dict(L1) ; print(D1)
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> T1 = (('a', 1), ('b', 2), ('c', 3), ('d',4))
>>> print(dict(T1))
{'a': 1, 'b': 2, 'c': 3, 'd': 4}

>>> L1 = [['a', 1], ['b', 2], ['c', 3], ['d',4]]
>>> print(dict(L1))
{'a': 1, 'b': 2, 'c': 3, 'd': 4}

T1 = (['a', 1], ['b', 2], ['c', 3], ['d',4])
>>> print(dict(T1))
{'a': 1, 'b': 2, 'c': 3, 'd': 4}

L1 = [('a', 1), ('b', 2), ('c', 3), ('d',4)]
>>> print(dict(L1))
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

When the keys are simple strings, it is sometimes easier to specify pairs using keyword arguments with the help of **dict()** constructor.

Example: 241 → Program to create dictionary with the help of **dict()**.

```
>>> dict(angle=2, triangle=3, rectangle=4, pentagon=5)
{'angle':2, 'triangle':3, 'rectangle':4, 'pentagon':5}
>>> dict(angle=2.1, triangle=3.2, rectangle=4.4)
{'angle': 2.1, 'triangle': 3.2, 'rectangle': 4.4}

>>> profile = dict(name='Shahi', salary=200000,
age=30)
>>> print(profile)
{'name': 'Shahi', 'salary': 200000, 'age': 30}
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

We can also create a dictionary by specifying set of keys separately and set of values separately, with the help of **zip()** method.

Example: 242 → Program to create dictionary with the help of **dict()**.

```
>>> marks = dict(zip(('name', 'game', 'marks'),
('nasra', 'kabaddi', 96)))
>>> print(marks)
{'name': 'nasra', 'game': 'kabaddi', 'marks': 96}
>>> marks = dict(zip(['name', 'game',],['nasra',
'kabaddi', 96]))
>>> print(marks)
{'name': 'nasra', 'game': 'kabaddi', 'marks': 96}

>>> marks = dict(zip(('name', 'game', 'marks'),
['nasra', 'kabaddi',  96]))
>>> marks
{'name': 'nasra', 'game': 'kabaddi', 'marks': 96}

>>> marks = dict(zip(['name', 'game','marks'],
('nasra','kabaddi', 96)))
>>> marks
{'name': 'nasra', 'game': 'kabaddi', 'marks': 96}

>>> print(dict(zip([1,2,3],[97, 99, 95])))
```

Dictionary comprehension is an elegant and concise way to create new dictionary from a sequence in Python.
Dict comprehension consists of an expression pair **(key: value)** followed by **for** statement inside curly braces **{}**.

Example: 243 → Program to create dictionary with the help of **dict()**.

```
>>> cube = {x: x**3 for x in (1, 2, 3, 4, 5, 6, 7)}
>>> print(cube)
{1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343}

>>> tripling = {x: x*3 for x in ('a', 'b', 'c', 'd')}
>>> print(tripling)
{'a': 'aaa', 'b': 'bbb', 'c': 'ccc', 'd': 'ddd'}

>>> L1 = (1, 2, 3, 4, 5, 6, 7)
>>> square = {x:x**2 for x in L1}
>>> print(square)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}

>>> square = {x/2:x**2 for x in L1}
>>> square
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
{0.5: 1, 1.0: 4, 1.5: 9, 2.0: 16, 2.5: 25, 3.0: 36,
3.5: 49}
```

For creation of a dictionary by the keyboard i.e. by the user, we can use **eval()** method.

Example: 244 → Creation of dictionary by user.

```
>>> d = eval(input('Enter a dictionary: '))
Enter a dictionary: {1:2, 2:5, 3:10, 4:17}
>>> print(d)
{1: 2, 2: 5, 3: 10, 4: 17}
```

ACCESSING ELEMENTS OF A DICTIONARY

ACCESSING BY KEY

For accessing the element of a dictionary, the corresponding key is needed.

Example: 245 → Program to access the element of a dictionary.

```
>>> flowers = {1:'Rose',2:'Lily',3:'Tulip',4:'Lotus'}
>>> print(flowers[2])
Lily
>>> flowers[4]
'Lotus'
>>> flowers[5]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    flowers[5]
KeyError: 5
```

ACCESSING ALL KEYS OR ALL VALUES SIMULTANEOUSLY

For accessing all keys of a dictionary, we use the function **keys()**, and for accessing all values of a dictionary, we use the function **values()**.

Example: 246 → Accessing all keys or all values of a dictionary.

```
>>> colors = {1:'violet', 2:'indigo', 3:'blue',
          4:'green', 5:'yellow', 6:'orange', 7:'red'}
>>> colors.keys()
dict_keys([1, 2, 3, 4, 5, 6, 7])
>>> colors.values()
dict_values(['violet', 'indigo', 'blue', 'green',
'yellow', 'orange', 'red'])
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

ADDING OR CHANGING ELEMENTS OF A DICTIONARY

We know that dictionaries are mutable, so we can add new items (**key: value** pair) to a dictionary or change the value of existing items using assignment operator.
If the key is already present, value gets updated, else a new **key: value** pair is added to the dictionary.

Example: 247 → Changing or adding element in a dictionary.

```
>>> D1 = {'eyes':'vision', 'ears':'sound',
          'nose':'smell', 'tongue':'taste'}
>>> D1['ears'] = 'listen' # updation
>>> print(D1)
{'eyes': 'vision', 'ears': 'listen', 'nose': 'smell',
'tongue': 'taste'}
>>> D1['skin'] = 'touch' # adding new element
>>> print(D1)
{'eyes': 'vision', 'ears': 'listen', 'nose': 'smell',
'tongue': 'taste', 'skin': 'touch'}
```

DELETING ELEMENTS FROM A DICTIONARY

The **pop()** method removes a particular item with the provided key and returns the value. If we try to remove an item which in not in dictionary, the **Key Error** exception is raised.
The **del** keyword removes individual items or the entire dictionary itself. If we try to remove a item which in not in dictionary, the **Key Error** exception is raised.
The **popitem()** can also be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the **clear()** method.

Example: 248 → Removing an element from a dictionary.

```
# File Name: del_dict.py | Cd. by: A. Nasra
cubes = {1:1, 2:8, 3:27, 4:64, 5:125, 6:216}
print(cubes.pop(4)) # remove a particular item
print(cubes)

print(cubes.popitem()) # remove an arbitrary item
print(cubes)

del cubes[5]  # delete a particular item
print(cubes)

cubes.clear() # remove all items
print(cubes)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
cub = {1:1, 2:8, 3:27, 4:64, 5:125, 6:216}
print(cub.pop(6,'Not found!'))
print(cub.pop(7,'Not found!'))
```

```
# Sample run of del_dict.py
64
{1: 1, 2: 8, 3: 27, 5: 125, 6: 216}
(6, 216)
{1: 1, 2: 8, 3: 27, 5: 125}
{1: 1, 2: 8, 3: 27}
{}
216
Not found!
```

TRAVERSAL OF A DICTIONARY

Traversal of a dictionary means to access and process each element of a dictionary.

Example: 249 → Traversal of a dictionary.

```
# File Name: traverse_dict.py | Cd. by: A. Nasra
DS = {'number':7, 'string':'abc', 'list':[1,2],
      'tuple':(3,4),     'dictionary':{1:2,     2:5},
'set':{6,7}}
for key in DS:
    print(DS[key], 'is an example of', [key])
```

```
# Sample run of traverse_dict.py
7 is an example of ['number']
abc is an example of ['string']
[1, 2] is an example of ['list']
(3, 4) is an example of ['tuple']
{1: 2, 2: 5} is an example of ['dictionary']
{6, 7} is an example of ['set']
```

NESTED DICTIONARY

In Python, a nested dictionary is a dictionary inside a dictionary. It's a collection of dictionaries into one single dictionary. Key points to remember about nested dictionary are as follows:

1. Nested dictionary is an unordered collection of dictionary
2. Slicing Nested Dictionary is not possible.
3. We can shrink or grow nested dictionary as need.
4. Like Dictionary, it also has key and value.
5. Dictionary are accessed using key.

Nested dictionary is created as given below:

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 250 → Nested dictionary creation.

```
>>> people = {1:{'name':'Liza','age':26,'sex':'fem'},
              2:{'name':'Albi','age':52,'sex':'male'}}
>>> print(people)
{1: {'name': 'Liza', 'age': 26, 'sex': 'female'},
 2: {'name': 'Albi', 'age': 52, 'sex': 'male'}}
```

To access element of a nested dictionary, we use indexing `[]` syntax in Python.

Example: 251 → Accessing elements of a nested dictionary.

```
# File Name: nes_dict_access.py | Cd. by: A. Nasra
emp = {1: {'name': 'Albi', 'age': 53, 'sex': 'Male'},
       2:{'name':'Zeni','age': 32, 'sex': 'Female'}}
print(emp[1]['name'])
print(emp[2]['name'])
print(emp[1]['sex'],'and',emp[2]['sex'])
```

```
# Sample run of nes_dict_access.py
Albi
Zeni
Male and Female
```

Example for updating or adding a new element in the dictionary is given in the following example.

Example: 252 → Changing and adding new elements in a nested dictionary.

```
# File Name: nes_dict_updt.py | Cd. by: A. Nasra
emp = {1: {'name': 'Albi', 'age': 53, 'sex': 'Male'},
       2: {'name': 'Zeni', 'age': 32, 'sex': 'Female'}}
# Updation of an element
emp[2]['name'] = 'Romi'
print('Updated dict is:',emp)
# Adding a new element
emp[3] = {'name':'Priti','age':28,'sex':'female'}
print('Dict with new element is:\n',emp)
```

```
# Sample run of nes_dict_updt.py
Updated dict is: {1: {'name':'Albi', 'age':53, 'sex':
'Male'}, 2:{'name':'Romi','age':32, 'sex': 'Female'}}
Dict with new element is:
 {1: {'name': 'Albi', 'age': 53, 'sex': 'Male'},
  2: {'name': 'Romi', 'age': 32, 'sex': 'Female'},
  3: {'name': 'Priti', 'age': 28, 'sex': 'female'}}
```

In Python, we use **`del`** statement to delete elements from nested dictionary.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 253 → Deleting elements from a nested dictionary.

```
# File Name: nes_dict_del.py | Cd. by: A. Nasra
emp = {1:{'name':'John', 'age':27, 'sex':'Male'},
       2:{'name':'Marie', 'age':22, 'sex':'Female'},
       3:{'name':'Luna', 'age':24, 'sex':'Female'},
       4:{'name':'Peter', 'age':29, 'sex':'Male'}}
del emp[3]['sex'] # deleting particular element
print('Dict emp after del:\n',emp)
del emp[3]  # deleting entire key:value pair
print('Dict emp after del:\n',emp)
```

```
# sample run of nes_dict_del.py
Dict emp after del:
 {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
  2: {'name': 'Marie', 'age': '22', 'sex': 'Female'},
  3: {'name': 'Luna', 'age': '24'},
  4: {'name': 'Peter', 'age': '29', 'sex': 'Male'}}
Dict emp after del:
 {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
  2: {'name': 'Marie', 'age': '22', 'sex': 'Female'},
  4: {'name': 'Peter', 'age': '29', 'sex': 'Male'}}
```

Using the for loops, we can iterate through each elements in a nested dictionary

Example: 254 → Iteration through a nested dictionary.

```
# File name: nes_dict_iterate.py | Cd. by: A. Nasra
emp = {1: {'Name': 'John', 'Age': 27, 'Sex': 'Male'},
       2: {'Name':'Marie', 'Age':22, 'Sex':'Female'},
       3: {'Name':'Romil', 'Age': 31, 'Sex':'Eemale'}}
for emp_id, info in emp.items():
    print("\nEmployee's ID:", emp_id)
    for key in info:
        print(key + ':', info[key])
```

```
# Sample run of nes_dict_iterate.py
Employee's ID: 1
Name: John
Age: 27
Sex: Male

Employee's ID: 2
Name: Marie
Age: 22
Sex: Female
Employee's ID: 3
Name: Romil
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Age: 31
Sex: Eemale
```

PRETTY PRINGTING OF A DICTIONARY

When a dictionary has a large number of elements, it becomes clumsy and reading is not easy. Presenting a such dictionary in readable format is called pretty printing. There are some certain ways of pretty printing, that are explained in the following examples.

Example: 255 → Pretty printing of a dictionary.

```
# File Name: pretty_print1.py
table = {'Simmi': 4127, 'Jacob': 4098, 'Albino': 7678}
h1 = "NAME" ;  h2 = "ID-NUMBER"
print(f' {h1:10} | {h2:5}')
print('\u2014'*23)
for name, phone in table.items():
    print(f' {name:10} | {phone:5}')
print('-'*23)
```

```
# Sample run of pretty_print1.py
NAME       | ID-NUMBER
-----------------------

 Simmi     |  4127
 Jacob     |  4098
 Albino    |  7678
-----------------------
```

Example: 256 → Pretty printing of a dictionary.

```
# File Name: pretty_print2.py
table = {'Simmi': 4127, 'Jacob': 4098, 'Albino': 7678}
h1 = "NAME" ; h2 = "PHONE-NO"
print(' {0:10} {1:5}'.format(h1, h2))
print('\u2014'*21)
for name, phone in table.items():
    print(' {0:10} {1:5d}'.format(name, phone))
```

```
# Sample run of pretty_print2.py
NAME       PHONE-NO
---------------------

 Simmi      4127
 Jacob      4098
 Albino     7678
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 257 → Pretty printing of a dictionary.

```
# File Name: pretty_print3.py
import json
table = {'Simmi': 4127, 'Jacob': 4098, 'Albino': 7678}
print(json.dumps(table, indent = 2))
```

```
# Sample run of pretty_print3.py
{
  "Simmi": 4127,
  "Jacob": 4098,
  "Albino": 7678
}
```

DICTIONARY MEMBERSHIP TEST

We can test if a key is in a dictionary or not using the keyword **in**. Notice that membership test is *for keys only*, not for *values*.

Example: 258 → Membership test for a dictionary.

```
>>> population = {'Maharashtra': 112372972,
 'Bihar': 103804637, 'WB': 91347736,
 'MP': 72597565, 'Tamil Nadu': 72138958}
>>> 'Maharashtra' in population
True
>>> 'Tamil Nadu' not in population
False
>>> 'WB' not in population
False
>>> 'MP' in population
True
```

BUILT IN FUNCTIONS AND METHODS WITH DICTIONARY

LEN( )

The len() function returns the number of items (length) of an object.
Its syntax is: **len(s)**
Where **s** is a sequence (string, bytes, tuple, list, or range) or a collection (dictionary, set or frozen set)
Failing to pass an argument or passing an invalid argument will raise a **TypeError** exception.

Example: 259 → Length or size of a dictionary.

```
>>> inventor = {'mobile':'Martin Cooper',
                'insulin':'Banting & Charles',
                'television':' J L Baird',
                'chocolate bar':'F L Cailler ',
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
                    'bicycle':'K. Macmillan',
                    'computer':'Charles Babbage',
                    'dynamite':'Alfred Nobel'}
>>> print(len(inventor))
7
```

SORTED( )

The sorted() method sorts the elements of a given iterable in ascending or descending order.

Its syntax is: **sorted(iterable, key, reverse)**

Where, **iterable** is a sequence (string, tuple, list) or a collection (set, dictionary, frozen set) or any other iterator; **reverse** is optional and if true, the sorted list is reversed (or sorted in Descending order), and **key** is optional function that serves as a key for the base of sorting.

Example: 260 → Sorting of a dictionary.

```
# File Name: sorted_dict1.py | Cd. by: A. Nasra
D = {'a':1, 'c':3, 'e':5, 'g':7, 'i':9, 'k':2,
     'm':4, 'o':6, 'q':8, 's':1, 'u':3, 'w':5}
DS = sorted(D, reverse=True)
for k in DS:
    print(' ', k,'\t' ,D[k])
```

```
# Sample run of sorted_dict1.py
  w   5
  u   3
  s   1
  q   8
  o   6
  m   4
  k   2
  i   9
  g   7
  e   5
  c   3
  a   1
```

Example: 261 → Sorting of a dictionary.

```
# File Name: sorted_dict2.py | Cd. by: A. Nasra
inventors = {'Mobile':'Martin Cooper',
             'Insulin':'Banting & Charles',
             'Television':'J L Baird',
             'Chocolate Bar':'F L Cailler',
             'Bicycle':'K. Macmillan',
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
              'Computer':'Charles Babbage',
              'Dynamite':'Alfred Nobel'}
h1 = "INVENTION" ; h2 = "INVENTOR"
print(f'{h1:15} {h2}')
print('\u2014'*34)
for k in sorted(inventors):
    print(f'{k:15} {inventors[k]}')
print('\u2014'*34)
```

```
# Sample run of sorted_dict2.py
INVENTION       INVENTOR
__________________________________

Bicycle         K. Macmillan
Chocolate Bar   F L Cailler
Computer        Charles Babbage
Dynamite        Alfred Nobel
Insulin         Banting & Charles
Mobile          Martin Cooper
Television      J L Baird
__________________________________
```

Example: 262 → Sorting of a dictionary.

```
# File Name: sorted_dict2.py | Cd. by: A. Nasra
inventors = {'Mobile':'Martin Cooper',
             'Insulin':'Banting & Charles',
             'Television':'J L Baird',
             'Chocolate Bar':'F L Cailler',
             'Bicycle':'K. Macmillan',
             'Computer':'Charles Babbage',
             'Dynamite':'Alfred Nobel'}
h1 = " INVENTION" ; h2 = " INVENTOR"
print(f'{h1:15} {h2}')
print('\u2014'*34)
for k in sorted(inventors, key=len):
    print(f' {k:15} {inventors[k]}')
print('\u2014'*34)
```

```
# Sample run of sorted_dict2.py
INVENTION       INVENTOR
__________________________________

 Mobile          Martin Cooper
 Insulin         Banting & Charles
 Bicycle         K. Macmillan
 Computer        Charles Babbage
 Dynamite        Alfred Nobel
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Television      J L Baird
Chocolate Bar   F L Cailler
_____________________________________
```

CLEAR( )

The **clear()** method removes all items from the dictionary and makes the dictionary empty. Its syntax is: **dict.clear()**

The **clear()** method neither take any parameters nor nor return any value (i.e., returns **None**).

Example: 263 → Illustration of **clear()** method for a dictionary.

```
>>> Android = {3.2:'Honycomb', 4.1:'Jelly Bean',
               4.4:'KitKat', '5.x':'Lollypop',
              '6.x':'Marshmallow', '7.x':'Nought',
              '8.x':'Oreo', '9.x':'Pie'}
>>> Android.clear()
>>> print('Android =', Android)
Android = {}
```

We can also empty the dictionary by assigning it as empty dictionary.
In the above example, we must note that all elements of the dictionary named as Android are removed, dictionary itself is not removed, it is only emptied. Dictionary can be removed by applying the clear() method on the empty dictionary, as is given in the following example.

Example: 264 → Illustration of **clear()** method for a dictionary.

```
# File Name: del_dict1.py
Android = {3.2:'Honycomb', 4.1:'Jelly Bean',
           4.4:'KitKat', '5.x':'Lollypop',
           '6.x':'Marshmallow', '7.x':'Nought',
           '8.x':'Oreo', '9.x':'Pie'}
Android = {}
print('Android =', Android)
print('Again applying clear() method on empty dict.')
print('Android =', Android.clear())
```

```
# Sample run of del_dict1.py
Android = {}
Again applying clear() method on empty dict.
Android = None
```

COPY( )

They **copy()** method returns a shallow copy of the dictionary.
The syntax of **copy()** is: **dict.copy()**

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

The **copy()** method doesn't take any parameters and returns a shallow copy of the dictionary. It doesn't modify the original dictionary, when any change is done in the copied dictionary.

Example: 265 → Illustration of copy() method for a dictionary.

```
# File Name: copy_dict1.py | Cd. by: A. Nasra
Origin1 ={0:'Zero',(0,0):'OriginPt', '\u221D':'Alpha'}
Origin2 = Origin1.copy()
print('Origin1 =', Origin1)
print('Origin2 =', Origin2)
print('\u2014'*57)
Origin2[0] = '\u0905'
print('Origin1 =', Origin1)
print('Origin2 =', Origin2)
```

```
# Sample run of copy_dict1.py
Origin1 = {0: 'Zero', (0, 0): 'OriginPt', '∝': 'Alpha'}
Origin2 = {0: 'Zero', (0, 0): 'OriginPt', '∝': 'Alpha'}
———————————————————————————————————————————————————
Origin1 = {0: 'Zero', (0, 0): 'OriginPt', '∝': 'Alpha'}
Origin2 = {0: 'अ', (0, 0): 'OriginPt', '∝': 'Alpha'}
———————————————————————————————————————————————————
```

A copy of a dictionary can also be obtained with the help of = (assignment operator), but it modifies the original dictionary when any change is made in assigned dictionary. All these points are crystal clear from the following example.

Example: 266 → Illustration of copy of a dictionary by assignment operator (=).

```
# File Name: copy_dict2.py | Cd. by: A. Nasra
Origin1={0:'Zero',(0,0):'Origin Pt', '\u221D':'Alpha'}
Origin2 = Origin1
print('Origin1 =', Origin1)
print('Origin2 =', Origin2)
print('\u2014'*57)
Origin2[0] = 'Alif'
print('Origin1 =', Origin1)
print('Origin2 =', Origin2)
```

```
# Sample run of copy_dict2.py
Origin1 = {0: 'Zero', (0, 0): 'OriginPt', '∝': 'Alpha'}
Origin2 = {0: 'Zero', (0, 0): 'OriginPt', '∝': 'Alpha'}
———————————————————————————————————————————————————
Origin1 = {0: 'Alif', (0, 0): 'OriginPt', '∝': 'Alpha'}
Origin2 = {0: 'Alif', (0, 0): 'OriginPt', '∝': 'Alpha'}
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

GET( )

The get() method returns the value for the specified key if key is in dictionary.
Its syntax is: **dict.get(key, value)**
The get() method takes maximum of two parameters: **key** to be searched in the dictionary; **value** (optional) to be returned if the **key** is not found. The default value is **None**.
The **get()** method returns:
the value for the specified **key** if **key** is in dictionary.
None if the **key** is not found and **value** is not specified.
value if the **key** is not found and **value** is specified.

Example: 267 → Illustration of **get()** method for a dictionary.

```
# File Name: get_dict.py | Cd. by: A. Nasra
Temp = {'Aug':142, 'Sep':109, 'Oct':104, 'Nov':44}
# value is provided
print('Temp in Oct:', Temp.get('Oct'))
print('Temp in Nov:', Temp.get('Nov'))
# value is not provided
print('Temp in Jan:', Temp.get('Jan'))
# value is provided
print('Temp in Jan:', Temp.get('Jan', 20))
```

```
# Sample run of get_dict.py
Temp in Oct: 104
Temp in Nov: 44
Temp in Jan: None
Temp in Jan:  20
```

The major difference between **get()** method and **dict[key]** to access elements is that the get() method returns a default value if the **key** is missing. However, if the **key** is not found when we use **dict[key]**, **KeyError exception** is raised.

ITEMS( )

The **items()** method returns a view object that displays a list of dictionary's (key, value) tuple pairs.
Its syntax is: **dictionary.items()**
The **items()** method is similar to dictionary's **viewitems()** method in Python 2.7. The **items()** method doesn't take any parameters.

Example: 268 → Illustration of **items()** method for a dictionary.

```
>>> TrigVal = {'sin0':0, 'sin30':1/2,
          'sin45':'1/\u221A2','sin60':'\u221A3/2',
          'sin90':1}
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> TrigVal.items()
dict_items([('sin0', 0), ('sin30', 0.5), ('sin45',
'1/√2'), ('sin60', '√3/2'), ('sin90', 1)])
>>> for i in TrigVal.items():
    print(i)
('sin0', 0)
('sin30', 0.5)
('sin45', '1/√2')
('sin60', '√3/2')
('sin90', 1)
```

How items() works when a dictionary is modified – is illustrated in the following example. If the list is updated at any time, the changes are reflected on to the view object itself, as shown in the program below.

Example: 269 → Illustration of **`items()`** method for a dictionary.

```
days = { '8 March': 'International Women\'s Day',
         '7 April': 'World Health Day',
         '5 June': 'World Environment Dan',
         '1 December': 'World AIDS Day'}
Items = days.items()
print('Original Items:\n', Items)
# delete an item from dictionary
print('\u2014'*58)
del days['8 March']
print('Updated Items:\n', Items)
```

```
Original Items:
 dict_items([('8 March', "International Women's Day"),
('7 April', 'World Health Day'), ('5 June', 'World
Environment Dan'), ('1 December', 'World AIDS Day')])
————————————————————————————————————————————
Updated Items:
 dict_items([('7 April', 'World Health Day'), ('5
June', 'World Environment Dan'), ('1 December', 'World
AIDS Day')])
```

KEYS( )

The **`keys()`** method returns a view object that displays a list of all the keys in the dictionary. Its syntax is: **`dict.keys()`**

The **`keys()`** doesn't take any parameters and returns a view object that displays a list of all the keys. When the dictionary is changed, the view object also reflect these changes.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 270 → Illustration of **keys()** for a dictionary.

```
>>> depth = {11033:'Pacific Ocean', 9460:'Atlantic
             Ocean',7542:'Indian Ocean'}
>>> print(depth.keys())
dict_keys([11033, 9460, 7542])
>>> depth[45000] = 'Southern Ocean'
>>> print(depth)
{11033: 'Pacific Ocean', 9460: 'Atlantic Ocean', 7542:
'Indian Ocean', 45000: 'Southern Ocean'}
>>> print(depth.keys())
dict_keys([11033, 9460, 7542, 45000])
```

VALUES( )

The **values()** method returns a view object that displays a list of all the values in the dictionary. Its syntax is: **dictionary.values()**
The values() method doesn't take any parameters.

Example: 271 → Illustration of **values()** for a dictionary.

```
>>> crop = {'Rabi':('Wheat', 'Gram', 'Pea', 'Barley'),
        'Kharif':('Rice', 'Jowar', 'Bajra', 'Maize'),
        'Zaid':('Cucumber', 'Pumpkin', 'Watermelon')}
>>> print(crop.values())
dict_values([('Wheat', 'Gram', 'Pea', 'Barley'),
('Rice', 'Jowar', 'Bajra', 'Maize'), ('Cucumber',
'Pumpkin', 'Watermelon')])
```

How **values()** works when dictionary is modified, has been exemplified in the following example. If the dictionary is updated at any time, the changes are reflected on to the view object itself, as shown in the above program.

Example: 272 → Illustration of **values()** for a dictionary.

```
# File Name: values_dict.py | Cd by: A. Nasra
crop = {'Rabi':('Wheat', 'Gram', 'Pea', 'Barley'),
        'Kharif':('Rice', 'Jowar', 'Bajra', 'Maize'),
        'Zaid':('Cucumber', 'Pumpkin', 'Watermelon')}
print('Original crop is:\n', crop.values())
del crop['Zaid']
print('\u2014'*55)
print('Updated crop is:\n', crop.values())
```

```
# Sample run of values_dict.py
Original crop is:
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
 dict_values([('Wheat',  'Gram',  'Pea',  'Barley'),
('Rice',  'Jowar',  'Bajra',  'Maize'),  ('Cucumber',
'Pumpkin', 'Watermelon')])
_____________________________________________________

Updated crop is:
 dict_values([('Wheat',  'Gram',  'Pea',  'Barley'),
('Rice', 'Jowar', 'Bajra', 'Maize')])
```

UPDATE( )

The **update()** method updates the dictionary with the elements from the another dictionary object or from an iterable of key/value pairs.

The **update()** method adds element(s) to the dictionary if the key is not in the dictionary. If the key is in the dictionary, it updates the key with the new value.

Its syntax is: **dict.update([other])**

If **update()** is called without passing parameters, the dictionary remains unchanged. It doesn't return any value (returns **None**).

Example: 273 → Illustration of **update()** for a dictionary.

```
# File Name: update_dict1.py
import json
missile = {'Astra':80, 'Akash':35, 'Trishul':9,
           'Prithvi':80, 'Agni-VI':12000}
print('Original dictionary named missile:')
print(json.dumps(missile, indent = 2))
m1 = {'Prithvi':600}
m2 = {'Dhanush':600}
# update the value of existing key 'Prithvi'
missile.update(m1)
print('Existing key\'s value is updated:')
print(json.dumps(missile, indent = 2))
# adding element with new key:value pair
missile.update(m2)
print('Updating by adding new key:value pair:')
print(json.dumps(missile, indent = 2))
```

```
# Sample run of update_dict1.py
Original dictionary named missile:
{
  "Astra": 80,
  "Akash": 35,
  "Trishul": 9,
  "Prithvi": 80,
  "Agni-VI": 12000
}
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Existing key's value is updated:
{
  "Astra": 80,
  "Akash": 35,
  "Trishul": 9,
  "Prithvi": 600,
  "Agni-VI": 12000
}
Updating by adding new key:value pair:
{
  "Astra": 80,
  "Akash": 35,
  "Trishul": 9,
  "Prithvi": 600,
  "Agni-VI": 12000,
  "Dhanush": 600
}
```

Example: 274 → Illustration of **update()** for a dictionary.

```
# File Name: update_dict2.py
d1 = {'x': 2}
d1.update(y = 3, z = 0)
print(d1)
d2 = {'x':2}
d2.update([('y',3),('z', 4)])
print(d2)
d3 = {1:1, 2:4, 3:9}
d3.update([(5,25),(6,36)])
print(d3)
d4 = {'x':1}
d4.update(y = 'so what')
print(d4)
```

```
# Sample run of update_dict2.py
{'x': 2, 'y': 3, 'z': 0}
{'x': 2, 'y': 3, 'z': 4}
{1: 1, 2: 4, 3: 9, 5: 25, 6: 36}
{'x': 1, 'y': 'so what'}
```

DICTIONARY PROGRAMMING EXAMPLES

List is a sequence of values of any type. These values are

Example: 275 → Write a program that repeatedly asks the user to enter product names and prices. Store all of these in a dictionary whose keys are the product names and whose values are the prices. When the user is done entering products

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

and prices, allow them to repeatedly enter a product name and print the corresponding price or a message if the product is not in the dictionary.

```
# File Name: ex275_dict.py | Code by: A. Nasra
import sys
from json import dumps
opt2 = 'y'
opt = 'y'
product = {}
while opt == 'y' or opt =='Y':
    prd = str(input('Product name: '))
    pri = float(input('Price = '))
    product[prd] = pri
    opt = input('Enter more product?(y/n)...')
print('\u2014'*30)
print('Dictionary is:\n', dumps(product, indent = 2))
print('\u2014'*30)
while opt2 == 'y' or opt2 == 'Y':
    k = str(input('Enter product name: '))
    if k in product:
        print('Price of', k, '=', product[k])
    else:
        print(k, 'is not in the dictionary.')
    opt2 = input('Price of more product?(y/n)...')
    if opt2 == 'n' or opt2 == 'N':
        sys.exit(0)
```

```
# Sample run of ex275_dict.py
Product name: cup
Price = 45.75
Enter more product?(y/n)...y
Product name: diary
Price = 25
Enter more product?(y/n)...y
Product name: oil
Price = 123.80
Enter more product?(y/n)...y
Product name: almirah
Price = 700
Enter more product?(y/n)...n
——————————————————————————————
Dictionary is:
 {
  "cup": 45.75,
  "diary": 25.0,
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
  "oil": 123.8,
  "almirah": 700.0
}
_____________________________________________
Enter product name: oil
Price of oil = 123.8
Want price of more product?(y/n)...y
Enter product name: plate
plate in not in the dictionary.
Want price of more product?(y/n)...n
```

Example: 276 → Write a program that takes a value and checks whether the given value is part of given dictionary or not. If it is, it should print the corresponding key otherwise print an error message.

```
# ex276_dict.py | Cd by: A. Nasra
from json import dumps
dict1  =  {0:'Zero',  1:'one',  2:'two',  3:'three',
4:'four', 5:'five'}
print('Let given dictionary be\n', dumps(dict1, indent
= 3))

opt = 'y'
while opt == 'y' or opt == 'Y':
    val = str(input('Enter the value: '))
    for k in dict1:
        if dict1[k] == val:
            sw = 1
            print(val,'is a part of dict and exits
                  at',k)
            break
        else:
            sw = 0
    if sw == 0:
        print('not found')
    opt = str(input('Check more values?(y/n)...'))
```

```
# Sample run of ex276_dict.p
Let the given dictionary be
 {
   "0": "Zero",
   "1": "one",
   "2": "two",
   "3": "three",
   "4": "four",
   "5": "five"
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
}
Enter the value: one
one is a part of dict and exits at 1
Check more values?(y/n)...y
Enter the value: zero
Check more values?(y/n)...n
```

Example: 277 → Repeatedly ask the user to enter a team name and the how many games the team won and how many they lost. Store this information in a dictionary where the keys are the team names and the values are lists of the form [wins, losses].
Using the dictionary created above, allow the user to enter a team name and print out the team's winning percentage.

```
# ex277_dict.py | Code by: A. Nasra
from sys import exit
opt = 'y'
d = {}
n = int(input('No. of teams = '))
for i in range(n):
    name = str(input('Enter name of team: '))
    win = int(input('No. of games won = '))
    loss = int (input('No. of games lossed = '))
    d[name] = [win, loss]
    print('\u2014'*30)

while opt == 'y' or opt =='Y':
    name = str(input('Enter name of team- '))
    if name not in d:
        print(name, 'is not in the dictionary!')
    else:
        print('Winning %age of', name, 'is = ',
              d[name][0]*100/(d[name][0]+d[name][1]))
    opt     =     str(input('Do     you     want     to
continue?(y/n)...'))
if opt != 'y' or 'Y':
    exit()
```

```
# Sample run of ex277_dict.py
No. of teams = 4
Enter name of team: Dallas Cowboys
No. of games won = 502
No. of games lossed = 374
------------------------------------
Enter name of team: Green Bay Packers
No. of games won = 737
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
No. of games lossed = 562
-----------------------------------

Enter name of team: Chicago Bears
No. of games won = 749
No. of games lossed = 479
-----------------------------------

Enter name of team: Miami Dolphins
No. of games won = 445
No. of games lossed = 351
-----------------------------------

Enter name of team- Chicago Bears
Winning %age of Chicago Bears is =  60.99348534201955
Do you want to continue?(y/n)...y
Enter name of team- Miami Dolphins
Winning %age of Miami Dolphins is =  55.904522613065325
Do you want to continue?(y/n)...n
```

Example: 278 → Write a program that uses a dictionary that contains ten user names and passwords. The program should ask the user to enter their username and password. If the username is not in the dictionary, the program should indicate that the person is not a valid user of the system. If the username is in the dictionary, but the user does not enter the right password, the program should say that the password is invalid. If the password is correct, then the program should tell the user that they are now logged in to the system.

```
# ex278_dict.py
# ex278_dict.py
dic ={'jackroy':'Jack123@d', 'abhi':'9038110152',
      'Superboy':'1234567','Gamehack':'Fsdgdsgds',
      'Kim':'Kimstar','Pokemon':'Pokemon4ever',
      'advade':'a12378965','pdateios.com':'rosie124',
      'bahogbelat':'iyot', 'cmonster':'cruz123456'}

un = str(input("Enter User Name: "))
if un not in dic:
    print('You are not Valid User!')
else:
    pw = str(input('Enter Password; '))
    if pw != dic[un]:
        print('Passawrod is Invalid.')
    else:
        print('Now you are logged in to System.')
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 279 → Write a program to find the Numeral Sum of a word according to following dictionary.

pts = {'A':1, 'B':2, 'C':3, 'D':4, 'E':5, 'F':6, 'G':7, 'H':8, 'I':9, 'J':1, 'K':2, 'L':3, 'M':4, 'N':5, 'O':6, 'P':7, 'Q':8, 'R':9, 'S':1, 'T':2, 'U':3, 'V':4, 'W':5, 'X':6, 'Y':7, 'Z':8, 'a':1, 'b':2, 'c':3, 'd':4, 'e':5, 'f':6, 'g':7, 'h':8, 'i':9, 'j':1, 'k':2, 'l':3, 'm':4, 'n':5, 'o':6, 'p':7, 'q':8, 'r':9, 's':1, 't':2, 'u':3, 'v':4, 'w':5, 'x':6, 'y':7, 'z':8, ' ':0}

```
# File Name: ex279_dict.py
pts = {'A':1, 'B':2, 'C':3, 'D':4, 'E':5, 'F':6, 'G':7,
       'H':8, 'I':9, 'J':1, 'K':2, 'L':3, 'M':4, 'N':5,
       'O':6, 'P':7, 'Q':8, 'R':9, 'S':1, 'T':2, 'U':3,
       'V':4, 'W':5, 'X':6, 'Y':7, 'Z':8, 'a':1, 'b':2,
       'c':3, 'd':4, 'e':5, 'f':6, 'g':7, 'h':8, 'i':9,
       'j':1, 'k':2, 'l':3, 'm':4, 'n':5, 'o':6, 'p':7,
       'q':8, 'r':9, 's':1, 't':2, 'u':3, 'v':4, 'w':5,
       'x':6, 'y':7, 'z':8, ' ':0}

word = str(input('Enter a word: '))
score = sum([pts[c] for c in word])
print('The Numeral-Sum of',word,'=', score)
```

```
# Sample run of ex279_dict.py
Enter a word: Mohan Das Karam Chand Gandhi
The Numeral-Sum of Mohan Das Karam Chand Gandhi = 102
```

Example: 280 → Write a program to create a function that takes year as parameters and returns the detail of Miss World from India (Details must return – name, DOB, won at the age). Take help from the following table.

Year	Name of Miss World	Date of birth	At The Age
1951	First ever Miss World contest		
1966	Reita Faria	23-08-1943	23
1994	Aishwarya Rai	01-11-1973	21
1997	Diana Hayden	01-05-1973	24
1999	Yukta Mookhey	07-10-1977	22
2000	Priyanka Chopra	18-07-1982	18
2017	Manushi Chhillar	14-05-1997	20

```
# File Name: ex280_dict.py
mw = {1966:['Reita Faria', '23-08-1943', 23],
      1994:['Aishwarya Rai', '01-11-1973', 21],
      1997:['Diana Hayden', '01-05-1973', 24],
      1999:['Yukta Mookhey', '07-10-1977', 22],
      2000:['Priyanka Chopra', '18-07-1982', 18],
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
      2017:['Manushi Chhillar',   '14-05-1997', 20]}
print('Call the function ms(year) to get detail of Miss
World from India.')
def ms(year):
    if year >=1966 and year<=2017:
        print('Name:', mw[year][0], '\nDOB:',
              mw [year][1],'\nMiss World Title at
              the age of:', mw[year][2])
    elif year<1966:
        print('No Miss World was selected from India.')
    elif year>2017:
        print('Miss World event to be conducted.')
    else:
        print('Call the function properly.')
```

```
# Sample run of ex280_dict.py
Call the function ms(year) to get detail of Miss World
from India.
>>> ms(2019)
Miss World event to be conducted.
>>> ms(1950)
No Miss World was selected from India.
>>> ms(1994)
Name: Aishwarya Rai
DOB: 01-11-1973
Miss World Title at the age of: 21
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

SET MANIPULATION

DEFINITION OF SET

A set is an unordered collection of immutable and unique items (no duplicates). However, the set itself is mutable. We can add or remove items from it. Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

CREATION OF SETS

A set is created by placing all the items (elements) inside curly braces **{}**, separated by comma or by using the built-in function **set()**.
It can contain any number of items of different or same types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

Example: 281 → Creation of Sets

```
>>> A = {1, 3, 5, 7, 11}              # Set of integers
>>> B = {'a', 'e', 'i', 'o', 'u'} # characters
>>> C = {2, 2.1, 'Hi', (1, 2, 3)} # mixed data type
>>> print(' A =', A, '\n B =', B, '\n C =', C)
 A = {1, 3, 5, 7, 11}
 B = {'o', 'i', 'e', 'u', 'a'}
 C = {'Hi', 2, 2.1, (1, 2, 3)}
```

Example: 282 → Creation of more Set

```
>>> # Set can't keep duplicate elements
>>> A = {3, 6, 8, 4, 6, 9}
>>> print('A =', A)
A = {3, 4, 6, 8, 9}
>>> # Set can't keep mutable data
>>> B = {1, 2, [3, 4, 5]}
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    B = {1, 2, [3, 4, 5]}
TypeError: unhashable type: 'list'
# We can make a set from any sequence.
>>> B = set([1,2,3])
>>> print('B =', B)
B = {1, 2, 3}
>>> C = set((1, 2, 3, 4)) ; print('C =', C)
C = {1, 2, 3, 4}
>>> D = set({2, 4, 6, 8, 10})
>>> print('D =', D)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
D = {2, 4, 6, 8, 10}
>>> E = set('giggle') ; print('E =', E)
E = {'g', 'i', 'e', 'l'}
>>> F = set('no one knows')
>>> print('F =', F)
F = {'o', 'w', ' ', 's', 'n', 'k', 'e'}
>>> G = set({1:9, 2:25, 3:16})
>>> print("G =", G)
G = {1, 2, 3}
>>> H = set({1:9, 2:25, 3:16}.values())
>>> print("H =", H)
H = {16, 9, 25}
```

Since empty curly braces **{}** make an empty dictionary so, to make an empty set) we must use the **set()** function without any argument.

CHANGING THE ELEMENTS OF A SET

We know that sets are mutable and unordered. Therefore, indexing have no meaning and so we cannot access or change an element of set using indexing or slicing.

We can add single element using the **add()** method and multiple elements using the **update()** method. The **update()** method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

Example: 283 → Adding elements to Sets

```
>>> A = {0, 1}
>>> A.add(-1)
>>> print('A =', A)
A = {0, 1, -1}
>>> A.update([4,5], {1,6,8})
>>> print('A =', A)
A = {0, 1, 4, 5, 6, 8, -1}
```

REMOVING THE ELEMENTS FROM A SET

A particular item can be removed from set using the methods, **discard()** and **remove()**.

The only difference between the two is that, while using **discard()** if the item does not exist in the set, it remains unchanged, but **remove()** will raise an error in such condition.

Example: 284 → Removing/discarding elements from a set.

```
>>> S = {1, 0, -1, 'a', 'm', 'z'}
>>> S.discard('m') ; print('S =', S)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
S = {0, 1, 'a', 'z', -1}
>>> S.remove(-1) ; print('S =', S)
S = {0, 1, 'a', 'z'}
>>> # discarding and removing an element
>>> # not present in the set.
>>> P = {0, 1, -1, 9, -9, 7, -7}
>>> P.discard(6) ; print('P =', P)
P = {0, 1, 7, 9, -9, -7, -1}
>>> P.remove(8) ; print('P =',P)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    P.remove(8) ; print('P =',P)
KeyError: 8
```

We can remove and return the removed item with the help of **pop()** method. Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary. We can also remove all items from a set with the help of **clear()** method.

Example: 285 → Pop an element and clear set.

```
>>> A = set('other-world')
>>> print('A =', A)
A = {'o', 'w', 'd', 'e', 'h', 'l', 't', 'r', '-'}
>>> A.pop()
'o'
>>> print('A =', A)
A = {'w', 'd', 'e', 'h', 'l', 't', 'r', '-'}
>>> A.pop()
'w'
>>> A.clear()
>>> print('A =', A)
A = set()
```

MEMBERSHIP TEST FOR SET

We can test if an item exists in a set or not, using the keyword **in**.

Example: 286 → Membership test for a set.

```
>>> A = {123,321,567,765,657,756}
>>> 367 in A
False
>>> 367 not in A
True
>>> 567 in A
True
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

ITERATION THROUGH A SET

With the help of a loop (generally for loop) we can iterate through a set

Example: 287 → Iteration through a set.

```
# File Name: iterate_set.py
B = set()
A = set("indian")
for ele in A:
    ele1 = ele.upper()
    B.add(ele1)
print('B =', B)
```

```
# Sample run of iterate_set.py
B = {'N', 'D', 'I', 'A'}
```

BULT-IN FUNCTIONS WITH SET

Built-in functions like **enumerate()**, **sum()**, **len()**, **max()**, **min()**, **sorted()**, etc. are commonly used with set to perform diverse jobs.

ENUMERATE( )

The **enumerate()** method adds counter to an iterable and returns the enumerate object.

The syntax of enumerate() is: **enumerate(iterable, start=0)**

Where, **iterable** is a sequence, an iterator, or objects that supports iteration, **start** begins counting from this number. Default value for **start** is 0.

The enumerate objects returned by the **enumerate()** method can be converted to list and tuple using **list()** and **tuple()** method respectively.

Example: 288 → Illustration of **enumerate()** for a set.

```
# File Name: enum_set.py
gem_set = {'Sapphire','Ruby','Emerald','Opel'}
print('gem_set =',gem_set)
print('Enumerated gem_set is:')
e1 = enumerate(gem_set)
for elm1 in e1:
    print(elm1)
print('~'*27)
e2 = enumerate(gem_set, 91)
print('New Enumerated gem_set is:')
for elm2 in e2:
    print(elm2)
print('gem_set converted to list:')
print(list(enumerate(gem_set)))
print('gem_set converted to tuple:')
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run of enum_set.py
print(tuple(enumerate(gem_set,start=5)))
gem_set = {'Opel', 'Ruby', 'Emerald', 'Sapphire'}
Enumerated gem_set is:
(0, 'Opel')
(1, 'Ruby')
(2, 'Emerald')
(3, 'Sapphire')
~~~~~~~~~~~~~~~~~~~~~~~~~~
New Enumerated gem_set is:
(91, 'Opel')
(92, 'Ruby')
(93, 'Emerald')
(94, 'Sapphire')
gem_set converted to list:
[(0,'Opel'),(1,'Ruby'),(2,'Emerald'),(3,'Sapphire')]
gem_set converted to tuple:
((5,'Opel'),(6,'Ruby'),(7,'Emerald'),(8,'Sapphire'))
```

Example: 289 → Pop an element and clear set.

```
# File Name: enum2_set.py
gem_set = {'Sapphire','Ruby','Emerald','Opel'}
print('gem_set =', gem_set)
print('\u2014'*49)
for count, item in enumerate(gem_set):
  print(count, item)

print('\u2014'*13)
# changing default start value
for count, item in enumerate(gem_set, 11):
  print(count, item)
```

```
# Sample run of enum2_set.py
gem_set = {'Emerald', 'Sapphire', 'Opel', 'Ruby'}
_________________________________________________

0 Emerald
1 Sapphire
2 Opel
3 Ruby
_____________

11 Emerald
12 Sapphire
13 Opel
14 Ruby
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

LEN( )

The **len()** function returns the number of items (length) of an object.
The syntax of **len()** is: **len(s)**
Where s is a sequence (string, bytes, tuple, list, or range) or a collection (dictionary, set or frozen set)
Failing to pass an argument or passing an invalid argument will raise a **TypeError** exception.

Example: 290 → Illustration of **len()** method for a set.

```
# File Name: len1_set.py
testSet = set()
print(testSet, 'length is', len(testSet))

testSet = {3, 55, 777}
print(testSet, 'has length', len(testSet))
```

```
# Sample run of len1_set.py
set() length is 0
{777, 3, 55} has length 3
```

MAX( )

The max() method returns the largest element in an iterable or largest of two or more parameters.
Different syntaxes of max() are:
max(iterable, *iterables[,key, default])
max(arg1, arg2, *args[, key])
where **iterable** is a sequence (tuple, string), collection (set, dictionary) or an iterator object whose largest element is to be found.
***iterables** (Optional) is any number of **iterables** whose largest is to be found.
key (Optional) is a key function where the iterables are passed and comparison is performed based on its return value.
default (Optional) is a default value if the given iterable is empty
arg1 is mandatory first object for comparison (could be number, string or other object)
arg2 is mandatory second object for comparison (could be number, string or other object)
***args** (Optional) are other objects for comparison

Example: 291 → Illustration of max() method for sets.

```
>>> print('Maximum is:', max({1, 3, 2, 5, 4}))
Maximum is: 5
>>> S = (23, 34, 43, 32)
>>> print('Maximum is:', max(S))
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Maximum is: 43
```

Example: 292 → Illustration of max() method for sets.

```
# File Name: max_set.py
def sumDigit(num):
    sum = 0
    while(num):
        sum += num % 10
        num = int(num / 10)
    return sum

# using max(arg1, arg2, *args, key)
print('Maximum is:', max({990, 311, 2156, 54, 45},
key=sumDigit))

# using max(iterable, key)
numSet = {19, 299, 2601, 721, 53, 13, 7}
print('Maximum is:', max(numSet, key=sumDigit))
```

```
# Sample run of max_set.py
Maximum is: 990
Maximum is: 299
```

Example: 293 → Illustration of max() method for sets.

```
>>> S1 = {12, 34, 56, '-', 17}
>>> S2 = {123,345,143,234,432,129}
>>> S3 = {'a',1,'bb',2,'ccc',3,'dddd'}
>>> print('Maximum is', max(S1, S2, S3, key = len))
Maximum is {'a', 2, 1, 3, 'ccc', 'bb', 'dddd'}
```

MIN( )

The **min()** acts in the same way as the **max()** method, only we have to change **min** in place of **max** in the above description of **max()** method. The same follows for the examples of of **min()**.

SUM( )

The **sum()** function adds the items of an iterable and returns the sum.

Its syntax is: **sum(iterable, start)**

Where, **iterable** is a list, tuple, dict etc whose item's sum is to be found. Normally, items of the iterable should be numbers; **start** (optional) is the value that is added to the sum of items of the iterable. The default value of start is 0 (if not given). The **sum()** function returns the sum of **start** and items of the given **iterable.**

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 294 → Pop an element and clear set.

```
>>> S1 = {123,345,143,234,432,129}
>>> print('Sum of elements of S1 =', sum(S1))
Sum of elements of S1 = 1406
>>> print('Sum of elements of S1 =', sum(S1,100))
Sum of elements of S1 = 1506
```

SORTED( )

The sorted() method sorts the elements of a given iterable in a specific order - Ascending or Descending.

Its syntax is: **sorted(iterable[, key][, reverse])**

Where **iterable** is a sequence (string, tuple, list) or collection (set, dictionary, frozen set) or any iterator, and **reverse** (Optional) sorts in descending order (or reverses), if true. The **key** (Optional) is a function that serves as a key for the sort comparison

Example: 295 → Illustration of **sorted()** method for a set.

```
>>> A = {'pen', 'pencil', 'book', 'eraser', 'notes',
        'videos'}
>>> print(sorted(A))
['book', 'eraser', 'notes', 'pen', 'pencil', 'videos']
>>> B = {789, 987, 567, 986, 777}
>>> print(sorted(B))
[567, 777, 789, 986, 987]
>>> C = {789, 987, 567, 986, 777}
>>> print(sorted(C, reverse = True))
[987, 986, 789, 777, 567]
>>> D = {'trust', 'belief', 'certain', 'sure',
        'character'}
>>> print(sorted(D, key=len))
['sure', 'trust', 'belief', 'certain', 'character']
```

Example: 296 → Illustration of **sorted()** method for a set.

```
# File Name: sorted_set.py
# take last letter for sorting
def lastLetter(word):
    w = word[len(word)-1]
    return w

strSet = {'select', 'all', 'from', 'table', 'where'}
print('strSet:',strSet)
sortedSet = sorted(strSet, key=lastLetter)
print('Sorted Set:', sortedSet)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
print('''NOTE\u27F6Here, sorting is done according to
      the last letter of the elements (words) of the
      strSet''')
```

```
# Sample run of sorted_set.py
strSet: {'all', 'select', 'where', 'table', 'from'}
Sorted Set: ['where', 'table', 'all', 'from', 'select']
NOTE⟶Here, sorting is done according to the last
      letter of the elements (words) of the strSet
```

SET METHODS AND OPERATORS

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

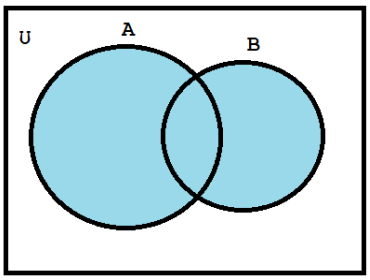
UNION( ) AND | OPERATOR

Union of **A** and **B** is a set of all elements from both sets. Union is performed using **|** operator or using the method **union()**.

Example: 297 → Illustration of union of two sets.

```
# File Name: union_set.py
A = {3, 4, 7, 8}
B = {2, 1, 9}
C = {'a', 'b', 'y', 'z'}

print('A \u222A B =', A | B)
print('A \u222A B =', A.union(B))
print('\u2014'*54)
print('A \u222A B \u222A C =', A | B |C)
print('A \u222A B \u222A C =', A.union(B, C))
```



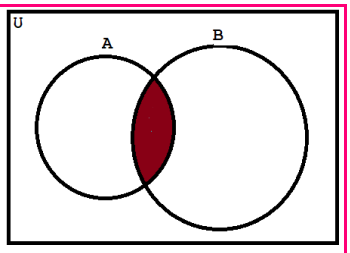

```
# Sample run of union_set.py
A U B = {1, 2, 3, 4, 7, 8, 9}
A U B = {1, 2, 3, 4, 7, 8, 9}
————————————————————————————————————————
A U B U C = {1, 2, 3, 4, 'y', 7, 8, 9, 'a', 'z', 'b'}
A U B U C = {1, 2, 3, 4, 'y', 7, 8, 9, 'a', 'z', 'b'}
```

INTERSECTION( ) AND & OPERATOR

Intersection of **A** and **B** is a set of elements that are common in both sets. Intersection is executed by using **&** operator or by using the method **intersection()**.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 298 → Illustration of intersection of two sets.




```
# File Name: union_set.py
A = {3, 4, 7, 8}
B = {2, 1, 9}
C = {'a', 'b', 'y', 'z'}

print('A \u2229 B =', A & B)
print('A \u2229 B =', A.intersection(B))
print('\u2014'*54)
print('A \u2229 B \u222A C =', A & B |C)
print('A \u2229 B \u222A C =', A.intersection(B, C))
```

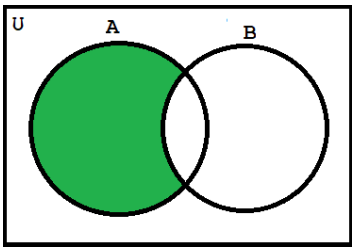
```
# Sample run of intersection_set.py
A ∩ B = set()
A ∩ B = set()
------------------------------------------------------

A ∩ B ∪ C = {'b', 'z', 'y', 'a'}
A ∩ B ∪ C = set()
```

DIFFERENCE( ) AND – OPERATOR

Difference of **A** and **B** **(A - B)** is a set of elements that are in **A** but not in **B**. Similarly, **B - A** is a set of element in **B** but not in **A**. Difference is achieved using **-** operator or using the method **difference()**.

Example: 299 → Illustration of difference of sets.




```
# File Name: difference_set.py
A = {3, 4, 7, 8}
B = {2, 1, 9}
C = {'a', 1, 'y', 7}

print('A \u2014 B =', A - B)
print('A \u2014 B =', A.difference(B))
print('\u2014'*23)
print('(A \u2014 B) \u2014 C =', (A - B) - C)
print('(A \u2014 B) \u2014 C =',
      (A.difference(B)).difference(C))
print('\u2014'*23)
print('C \u2014 A =', C - A)
print('C \u2014 A =', C.difference(A))
```

```
# Sample run of difference_bet.py
A – B = {8, 3, 4, 7}
A – B = {8, 3, 4, 7}
-----------------------

(A – B) – C = {8, 3, 4}
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
(A – B) – C = {8, 3, 4}
_____________________________

C – A = {'y', 1, 'a'}
C – A = {'y', 1, 'a'}
```

DIFFERENCE_UPDATE( )

The **difference_update()** updates the set calling difference_update() method with the difference of sets.

Its syntax is: **A.difference_update(B)**

Here, **A** and **B** are two sets. The difference_update() updates set **A** with the set difference of **A-B**. The **difference_update()** returns **None** which indicates that the object (set) is mutated.

If we run the code **outcome = A.difference_update(B)**

Then **outcome** will be **None**; **A** will be equal to **A – B**, and **B** will be unchanged.

Example: 300 → Illustration of **difference_update()** method of set.

```
# File Name: diff_update_set.py
A = {'l', 'k', 'p', 'c'}
B = {'c', 'f', 'p'}

outcome = A.difference_update(B)

print('A =', A) # Actually A becomes A - B
print('B =', B)
print('outcome =', outcome)
```

SYMMETRIC_DIFFERENCE( ) AND ^ OPERATOR

Symmetric Difference of **A** and **B** is a set of elements in both **A** and **B** except those that are common in both, i.e. **(A – B)∪(B – A)**

Symmetric difference is performed using **^** operator. Same can be accomplished using the method **symmetric_difference()**.

Example: 301 → Illustration of **symmetric_difference()**.

```
# File Name: symmetric_diff_set.py
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

print('Symmetric Diff =', (A - B)|(B - A))
print('Symmetric Diff =', A ^ B)
print('Symmetric Diff =', A.symmetric_difference(B))
print('Symmetric Diff =', B.symmetric_difference(A))
Symmetric Diff = {1, 2, 3, 6, 7, 8}
Symmetric Diff = {1, 2, 3, 6, 7, 8}
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Symmetric Diff = {1, 2, 3, 6, 7, 8}
Symmetric Diff = {1, 2, 3, 6, 7, 8}
```

SYMMETRIC_DIFFERENCE_UPDATE( )

The **symmetric_difference_update()** method updates the set calling the **symmetric_difference_update()** with the symmetric difference of sets.

Its syntax is: **A.symmetric_difference_update(B)**

Note that the **symmetric difference** of two sets is the set of elements that are in either of the sets but not in both.

The **symmetric_difference_update()** takes a single argument (set) and returns **None** (meaning, absence of a return value). It only updates the set calling this method.

Example: 302 → Illustration of **symmetric_difference_update()**.

```
# File Name: symmetric_diff_update_set.py
A = {1, 2, 3, 99, 98, 97}
B = {1, 2, 97, 4, 5}

outcome = A.symmetric_difference_update(B)

print('A =', A)
print('B =', B)
print('outcome =', outcome)
```

```
# Sample run of symmetric_diff_update_set.py
A = {99, 3, 98, 4, 5}
B = {1, 2, 97, 4, 5}
outcome = None
```

ISDISJOINT( )

The **isdisjoint()** method returns **True** if two sets are disjoint sets, and returns False if not, if sets are not disjoint.

The syntax of isdisjoint() is: **set_A.isdisjoint(set_B)**

The **isdisjoint()** method takes a single argument (a set).

We can also pass an iterable (list, tuple, dictionary and string) to **disjoint()**.

The **isdisjoint()** method will automatically convert iterables to set and checks whether the sets are disjoint or not.

The **isdisjoint()** method returns **True** if two sets are disjoint sets, and return **False** if two sets are not disjoint sets. Mathematically, two sets are disjoint if their intersection is empty set.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 303 → Illustration of **isdisjoint1.py**.

```
# File Name: isdisjoint1_set.py
A = {2, 3, 4, 5}
B = {6, 7, 8}
C = {5, 6, 7}

print('Are A and B disjoint?', A.isdisjoint(B))
print('Are A and C disjoint?', A.isdisjoint(C))
if A & B == set():
    print('''Since, A \u2229 B = empty_set.
Therefore, A and B are disjoint''')
```

```
# Sample run of isdisjoint1_set.py
Are A and B disjoint? True
Are A and C disjoint? False
Since, A ∩ B = empty_set.
Therefore, A and B are disjoint
```

Example: 304 → Illustration of **isdisjoint()**.

```
# File Name: isdisjoint2_set.py
A = {'a', 'b', 'c'}
B = ['b', 'c', 'd']
C = 'e12f'
D = (3, 4, 'g', 5, 6)
E ={1 : 'a', 2 : 'b'}
F ={'a' : 1, 'b' : 2}

print('Are A and B disjoint?', A.isdisjoint(B))
print('Are A and C disjoint?', A.isdisjoint(C))
print('Are A and D disjoint?', A.isdisjoint(D))
print('Are A and E disjoint?', A.isdisjoint(E))
print('Are A and F disjoint?', A.isdisjoint(F))
```

```
# Sample run of isdisjoint2_set.py
Are A and B disjoint? False
Are A and C disjoint? True
Are A and D disjoint? True
Are A and E disjoint? True
Are A and F disjoint? False
```

ISSUBSET( )

The **issubset()** method returns **True** if all elements of a set are present in another set (passed as an argument). If not, it returns **False**.

The syntax of issubset() is: **set_A.issubset(set_B)**

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

The **issubset()** returns **True** if **A** is a subset of **B**, **False** if **A** is not a subset of **B.**

Example: 305 → Illustration of **issubset()**.

```
# File Name: issubset_set.py
A = {1, 2, 3}
B = {1, 2, 3, 4, 5}

print(A.issubset(B))
print(B.issubset(A))
print('\u2014'*6)
print(A.issubset(A))
print({3, 4}.issubset(B))
```

```
# Sample run of issubset_set.py
True
False
——————
True
True
```

ISSUPERSET( )

Set **A** is said to be the superset of set **B** if all elements of **B** are in **A**.
The **issuperset()** method returns **True** if a set has every elements of another set (passed as an argument). If not, it returns **False**.
Its syntax is: **set_A.issuperset(B)**
The **issuperset()** returns **True** if **A** is a superset of **B**, **False** if **A** is not a superset of **B**.

Example: 306 → Illustration of **issuperset()**.

```
# File Name: issuperset_set.py
A = {1, 2, 3, 4, 5}
B = {1, 2, 3}

print(A.issuperset(B))
print(B.issuperset(A))
print(A.issuperset(A))
```

```
# Sample run of issuperset_set.py
True
False
True
```

UPDATE( )

The **update()** adds elements from a set (passed as an argument) to the set (calling the **update()** method).

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Its syntax is: **A.update(B)**

Where, **A** and **B** are two sets. The elements of set **B** are added to the set **A**.

The **update()** method takes a single argument (a set).

We can also add elements of other iterables or sequence (like tuple, list, dictionary etc.) to the set, by using following syntax:

set.update(set(sequence) or **set.update(sequence)**

This method returns **None** (meaning, absence of a return value).

Example: 307 → Illustration of **update()**.

```
# File Name: update1_set.py
A = {'a', 'b', 'c'}
B = {1, 2, 3, 4}

output = A.update(B)
print('A =',A)
print('B =',B)
print('output =',output)
```

```
# Sample run of update1_set.py
A = {1, 2, 3, 4, 'c', 'b', 'a'}
B = {1, 2, 3, 4}
output = None
```

Example: 308 → Illustration of **update()**.

```
# File Name: update2_dict.py
A = {1, 2}              # number_set to be updated
B = {1, 3, 'a', 'b'}
A.update(B)             # updation of set A with set B.
print('Updated set A with set B:', A)
print('\u2014'*50)
A = {1, 2}
B = (2, 3, 'z')         # tuple
A.update(B)             # updation of set A with tuple B
print('Updated set A with tuple B:', A)
print('\u2014'*50)
A = {1, 2}
B = [1, 'a', 2]         # list
A.update(B)             # updation of set A with list B
print('Updated set A with list B:', A)
print('\u2014'*50)
A = {1, 2}
B = "ghagh"             # string
A.update(B)             # updation of set A with list B
print('Updated set A with string B:', A)
```

```
# Sample run of update2_set.py
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Updated set A with set B: {1, 2, 3, 'b', 'a'}
Updated set A with tuple B: {1, 2, 3, 'z'}
Updated set A with list B: {1, 2, 'a'}
Updated set A with string B: {1, 2, 'h', 'a', 'g'}
```

ADD( )

The **add()** method adds a given element to a set. If the element is already present, it doesn't add any element. Its syntax is: **set.add(elem)**
It takes only one parameter elem to the set. The **add()** method itself returns **None**. Actually it changes the set by creating elements given as parametet.

Example: 309 → Illustration of **add()** method.

```
>>> A = set()              # Empty set
>>> A.add(1)               # adding 1 to set A
>>> print('A =', A)
A = {1}
>>> A.add(-1)              # adding -1 to set changed set A
>>> print('A =',A)
A = {1, -1}
>>> A.add('lkcc')          # adding string
>>> print('A =', A)
A = {'lkcc', 1, -1}
>>> A.add((0, 1))          # adding tuple
>>> print('A =', A)
A = {'lkcc', 1, (0, 1), -1}
```

DISCARD( )

The **discard()** method removes a specified element from the set (if present). Its syntax is: **set.discard(x)**
This method takes a single parameter (element **x)** and removes it from the set (if present). This method returns **None** (meaning, absence of a return value), it only updates the **set**.

Example: 310 → Pop an element and clear set.

```
>>> numSet = {2, 3, 4, 5}
>>> numSet.discard(4)
>>> print('numSet =', numSet)
numSet = {2, 3, 5}
>>> print(numSet.discard(2))
None
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> print('numSet =', numSet)
numSet = {3, 5}
>>> numSet.discard(3)
>>> numSet.discard(5)
>>> print('numSet =', numSet)
numSet = set()
```

POP( )

The pop() method removes an arbitrary element from the set and returns the element removed. Its syntax is: **set.pop()**

It doesn't take any arguments and returns an arbitrary (random) element from the set. Also, the set is updated and will not contain the element (which is returned). If the set is empty, **TypeError** exception is raised.

Example: 311 → Illustration of pop() method for a set.

```
# File Name: pop_set.py
A ={'a', 'b', 'c', 'd'}
print('Popped Value is:', A.pop())
print('A = ', A)
A.pop() ; A.pop() ; A.pop()
print('A =', A)
```

```
# Sample run of pop_set.py
Popped Value is: a
A =  {'b', 'c', 'd'}
A = set()
```

CLEAR( )

The **clear()** method removes all elements from the set.

Its syntax is: **set.clear()**

This method doesn't take any parameters and return None i.e. does not return any value, it only empties the set.

Example: 312 → Pop an element and clear set.

```
>>> A = {2, 4, 8, 32, 256, 1024}
>>> A.clear()
>>> print('After A.clear(), A =', A)
After A.clear(), A = set()
>>> print(A.clear())
None
```

COPY( )

The copy() method returns a shallow copy of the set.

A set can be copied using **=** operator in Python. The problem with copying the set in this way is that if we modify the set, the original set is also modified.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

The syntax of **copy()** is: **set.copy()**
It doesn't take any parameters, neither it returns any value, it only modifies the given set.

Example: 313 → Pop an element and clear set.

```
# File Name: copy_set.py
N = {1, 2, 3, 4}
newN = N
newN.add(0)           # Changing the set newN
print('newN =', newN)
print('N =', N)       # Original N is also changed.
print('\u2014'*22)
N1 = {2, 3, 4}
newN1 = N1.copy()
newN1.add(1)          # Changing the set newN1
print('newN1 =', newN1)
print('N1 =', N1)     # Original N1 is also changed.
```

```
# Sample run of copy_set.py
newN = {0, 1, 2, 3, 4}
N = {0, 1, 2, 3, 4}
——————————————————————

newN1 = {1, 2, 3, 4}
N1 = {2, 3, 4}
```

FROZEN SET

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets.
Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary.
Frozensets can be created using the function **frozenset().**
The frozen set supports the The frozen set supports the methods like, **copy()**, **difference()**, **intersection()**, **isdisjoint()**,**issubset()**, **issuperset()**, **symmetric_difference()** and **union()**. Being immutable it does not have method that add or remove elements.

Example: 314 → Frozenset examples.

```
>>> A = frozenset([1, 2, 3, 4])
>>> B = frozenset([3, 4, 5, 6])
>>> A.isdisjoint(B)
False
>>> A.difference(B)
frozenset({1, 2})
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
>>> A | B
frozenset({1, 2, 3, 4, 5, 6})
>>> A.add(3)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    A.add(3)
AttributeError: 'frozenset' object has no attribute
'add'
```

PROGRAMMING EXAMPLES OF SET

A dictionary is mutable, unordered collections of elements in the form of a

Example: 315 → Frozenset examples.

Example: 316 → Frozenset examples.

Example: 317 → Frozenset examples.

Example: 318 → Frozenset examples.

Example: 319 → Frozenset examples.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

SORTING ALGORITHMS

DEFINITION OF SORTING

A Sorting is defined as the arrangement of elements in an ordered sequence. From the following figures, concepts of sorting can be understood easily.

There are many algorithm of that can be applied to a group of unsorted groups, among them important are:

(1) Selection Sort (2) Bubble Sort (3) Insertion Sort
(4) Heap Sort (5) Quick Sort (6) Merge Sort

BUBBLE SORT

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example: 320 → Understanding bubble sort.

```
Pass-1: (i is 1)
[5 1 4 2 8] ➡ [1 5 4 2 8], Here, algorithm compares
the first two elements, and swaps since 5 > 1.
[1 5 4 2 8] ➡ [1 4 5 2 8], Swap since 5 > 4
[1 4 5 2 8] ➡ [1 4 2 5 8], Swap since 5 > 2
[1 4 2 5 8] ➡ [1 4 2 5 8], Now, since these elements
are already in order 8 > 5, algorithm does not swap
Pass-2:  (i is 2)
[1 4 2 5 8] ➡ [1 4 2 5 8]
[1 4 2 5 8] ➡ [1 2 4 5 8], Swap since 4 > 2
[1 2 4 5 8] ➡ [1 2 4 5 8]
[1 2 4 5 8] ➡ [1 2 4 5 8]
Now, the array is already sorted, but our algorithm
does not know if it is completed. The algorithm needs
one whole pass without any swap to know it is sorted.
Pass-3:  (i is 3)
[1 2 4 5 8] ➡ [1 2 4 5 8]
[1 2 4 5 8] ➡ [1 2 4 5 8]
[1 2 4 5 8] ➡ [1 2 4 5 8]
[1 2 4 5 8] ➡ [1 2 4 5 8]
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

IMPLIMENTATION OF BUBBLE SORT

Example: 321 → Implementation of bubble sort in Python.

```
# File Name: bubble_sort.py
# Python program for implementation of Bubble Sort
from winsound import Beep#duration=1000 ms,freq=440 Hz
def bubbleSort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):  # i indicates the pass
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    print("Sorted array is:")
    Beep
    for i in range(len(arr)):
        print(arr[i], end='   ')
```

```
# Sample run of bubble_sort.py
>>> L = [5, 1, 6, 2, 8, 3]
>>> bubbleSort(L)
Sorted array is:
1   2   3   5   6   8

>>> bubbleSort(list('something'))
Sorted array is:
e   g   h   i   m   n   o   s   t

>>> T = list((1, 4, 7, 2, 3, 5))
>>> bubbleSort(T)
Sorted array is:
1   2   3   4   5   7

>>> S = list({3, 9, 6, 4, 11, 54, 23})
>>> bubbleSort(S)
Sorted array is:
3   4   6   9   11   23   54

>>> dic = {1:'L', 2:'K', 3:'C', 4:'c'}
>>> bubbleSort(list(dic.values()))
Sorted array is:
C   K   L   c
```

NUMBER OF OPERATIONS IN BUBBLE SORT

The number of operation in any sort is the measurement of efficiency in aspect of CPU time.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

If in the above program of bubble sort, we take number of elements in the list N, we can see that the number of comparisons (swapping) in each loop (pass) will be as given below:

Pass	Comparisons
1	N - 1
2	N - 2
3	N - 3
4	N - 4
& co.	& going on less
N - 3	3
N - 2	2
N - 1	1

Thus, total number of comparisons is

$= (N-1) + (N-2) + (N-3) + (N-4) + \cdots\cdots + (2) + (1)$

$= (N + N + N + N\ times) - (1 + 2 + 3 + \cdots\cdots + N)$

$= N^2 - \sum N = N^2 - \frac{1}{2}N(N+1) = \frac{N(N-1)}{2} \cong N^2$

Actually, the number of swapping very much depends on the element of sequence. Therefore,

$$No.of\ operations \cong No.of\ comparisons + No.of\ swapping$$

- **Best Case**: All the elements of the sequence is already sorted.

$$No.of\ operations \cong No.of\ comparisons + No.of\ swapping$$
$$\cong N^2 + 0 \cong N^2$$

- **Worst Case**: All elements are in opposite order.

$$No.of\ operations \cong No.of\ comparisons + No.of\ swapping$$
$$\cong N^2 + N^2 \cong 2N^2$$

Example: 322 → Calculation of No. of operations in bubble sort.

```
# File Name: bubble_op.py | No. of operations
def bubbleSort(arr):
    n = len(arr)
    SumS = 0     # SumS is sum of no. of swapping
    for i in range(n):
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                SumS = SumS + 1

    print("Sorted array is:")
    for i in range(n):
        print(arr[i], end='  ')
    print("\nNo. of Operations =", SumS + n*(n+1)/2)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
# Sample run of bubble_op.py
>>> arr = [3, 2, 1]
>>> arr1 = [1, 2, 3]
>>> bubbleSort(arr)
Sorted array is:
1  2  3
No. of Operations = 9.0
>>> bubbleSort(arr1)
Sorted array is:
1  2  3
No. of Operations = 6.0
>>> bubbleSort([15, 6, 13, 22, 3, 52, 2])
Sorted array is:
2  3  6  13  15  22  52
No. of Operations = 40.0
```

INSERTION SORT

Insertion sort is a technique of sorting in which every iteration moves an element from unsorted portion to sorted portion until all the elements are sorted in the list.

Example: 323 → Understanding insertion sort.

```
Let the list to be sorted be: [5 1 4 2 8]
[5  1 4 2 8] ➡ [1  5  4 2 8], Initially, element 5 is
taken in sorted portion. Now, 2nd element 1 is
compared to 5 and inserted before 5 in sorted portion
[1 5  4 2 8] ➡ [1 4 5  2 8], 4 is inserted ...
[1 4 5  2 8] ➡ [1 2 4 5  8], 2 is inserted ...
[1 4 2 5  8] ➡ [1 4 2 5 8], 8 is inserted in the last
of sorted portion.
NOTE: At every iteration the number of comparison for
inserting the element from unsorted portion to sorted
portion is increasing by 1.
```

IMPLIMENTATION OF INSERTION SORT

Example: 324 → Implementation of bubble sort in Python.

```
# File Name: insertion_sort.py
def insertSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j]:
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
            arr[j+1] = arr[j]
            j -= 1
            arr[j+1] = key

    print("Sorted array is:")
    for i in range(len(arr)):
        print(arr[i], end='  ')
```

```
# Sample run of insertion_sort.py
>>> t = [5, 3, 4, 7, 6]
>>> insertSort(t)
Sorted array is:
3  4  5  6  7
```

NUMBER OF OPERATIONS IN INSERTION SORT

The number of operations in insertion sort depends on

- the number of comparisons in the inner loop, and
- the number of exchanges (insertion) in the inner loop.

If in the above program of insertion sort, we take number of elements in the list N, we can see that the number of comparisons (swapping) in each loop (pass) will be as given below:

Pass	Comparisons (at most)
1	1
2	2
3	3
4	4
& co.	& so on
N - 3	N - 3
N - 2	N - 2
N - 1	N - 1

Thus, total number of comparisons is

$$= (1) + (2) + (3) + (4) + \cdots\cdots + (N-2) + (N-1)$$

$$= \sum (N-1) = \frac{1}{2}N(N+1) \cong N^2$$

If we carefully view the program of insertion sort, we find that

- during the 1st iteration of outer loop, no. of exchange = 1 at most
- during the 2nd iteration of outer loop, no. of exchange = 2 at most
- during the 3rd iteration of outer loop, no. of exchange = 3 at most
- …
- During the (N-1)th iteration of outer loop, no. of exchange = (N-1) at most.

Thus, maximum no. of exchanges

$$= (1) + (2) + (3) + (4) + \cdots\cdots + (N-2) + (N-1)$$

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

$= \sum(N-1) = N/2 \times (N-1) < N^2$

We must note that the number of comparisons and exchanges depends on the sequence to be sorted.

- **Best Case**: All the elements of the sequence is already sorted.
  Outer for loop runs for (N – 1) times
  Inner while loop exits the loop in the very first comparison
  Thus, Total no. of operations = 1 + 1 + 1 + … + (N – 1) times = **N – 1**
- **Worst Case**: All elements are in opposite order.
  Outer for loop runs (N – 1) times
  Inner while loop runs for (N – 2) times
  Thus, Total no. of operations = 1+2+3+...+(N-1) = (N-1) × N/2 ≅ $\mathbf{N^2}$

Example: 325 → Calculation of No. of operations in bubble sort.

```
# File Name: insertion_op.py
def insertSort(arr):
    n = len(arr)
    for i in range(1, n):
        insum = 0
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
            arr[j+1] = key
            insum += 1

    print("Sorted array is:")
    for i in range(len(arr)):
        print(arr[i], end='  ')
    print('\nNo. of operations\u2245',n*(n-1)/2+insum)
```

```
# Sample run of insertion_op.py
>>> insertSort([5, 4, 6, 3, 7])
Sorted array is:
3  4  5  6  7
No. of operations ≅ 10.0
```

APPLICATION OF BUBBLE SORT

1. Bubble sort is applied to sort the datasheet which stored on tape of cassette.
2. In computer graphics, bubble sort is popular for its capability to detect a very small error.
3. It is used in a polygon filling algorithm.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

APPLICATION OF INSERTION SORT

1. Insertion sort is a preferred choice for sorting small datasheet.
2. Sorting of answer-sheets according to the roll-number of students.
3. Sorting of playing cards is done using insertion sort.

PROGRAMMING EXAMPLES OF BUBBLE AND INSERTION SORT

Example: 326 → Let election be a dictionary where key : value pairs are in the form of name : votes received. Write a program that sorts the contents of the Election dictionary and creates two lists that stores the sorted data. List A[i] will contain the name of the candidate and List B[i] will contain the votes received by candidate of List A[i]. print the name of candidates with votes received in descending order of the number of votes received.

```
# File Name: ques1_sort.py
import json
Election = {} # empty dictionary
A = []; B = []
n = int(input('Enter the Number of party: '))
for i in range(1, n+1):
    print('Name of Candidate-',i, end=' \u27A1 ')
    name = str(input())
    vote = int(input('Votes Received \u27A1 '))
    Election[name] = vote
    A.append(name)
    B.append(vote)
print('Election dictionary is:')
print(json.dumps(Election, indent = 4))
print('A[i] = ', json.dumps(A, indent = 3))
print('B[i] = ', json.dumps(B, indent = 3))

B2 = sorted(B, reverse = True)
for i in range(len(B2)):
    for key in Election:
        if B2[i] == Election[key]:
            print(key, '\t\t',B2[i])
```

```
Enter the Number of party: 4
Name of Candidate- 1 ➡ Sonia Gandhi
Votes Received ➡ 106935942
Name of Candidate- 2 ➡ Mamata Banerjee
Votes Received ➡ 20378052
Name of Candidate- 3 ➡ Narendra Modi
Votes Received ➡ 171660230
Name of Candidate- 4 ➡ Jayalalithaa
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Votes Received ➞ 18111579
Election dictionary is:
{
     "Sonia Gandhi": 106935942,
     "Mamata Banerjee": 20378052,
     "Narendra Modi": 171660230,
     "Jayalalithaa": 18111579
}
A[i] =  [
    "Sonia Gandhi",
    "Mamata Banerjee",
    "Narendra Modi",
    "Jayalalithaa"
]
B[i] =  [
    106935942,
    20378052,
    171660230,
    18111579
]
Narendra Modi           171660230
Sonia Gandhi            106935942
Mamata Banerjee         20378052
Jayalalithaa            18111579
```

Example: 327 → Write a program to read a list containing 3-digit integers only. Then write an insertion sort function that sorts the list on the basis of one's digit of all elements. That is, if given list is:

```
[378, 410, 285, 106]
```

Then the sorted list (as per above conditions) should be:

```
[410, 285, 106, 387]
```

```
# File Name: ques2_sort.py
def insertSort(arr):
    D = {}
    for elm in arr:
        D[elm%10] = elm
    L = list(D.keys())
    for i in range(len(L)):
        key = L[i]
        j = i-1
        while j >=0 and key < L[j]:
            L[j+1] = L[j]
            j -= 1
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
            L[j+1] = key

    print("Sorted array is:")
    for elm in L:
        print(D[elm], end='    ')
```

```
# Sample run of ques2_sort.py
>>> insertSort([321, 543, 652])
Sorted array is:
321    652    543
>>> insertSort([325, 543, 652])
Sorted array is:
652    543    325
```

Example: 328 → Write a program to read a list containing 3-digit integer only, that sorts the list on the basis of one's digit of all elements. That is, if given list is: `[378, 410, 285, 106]`
Then the sorted list (as per above conditions) should be:
`[410, 285, 106, 387]`

```
# File Name: ques3_sort.py
# Method-1
L1 = eval(input('Enter a integer list having 3 digits
only:\n '))
D = {}
for elm in L1:
    D[elm%10] = elm
L = sorted(D.keys())

print("Sorted array (by Method-1) is:")
for elm in L:
    print(' ',D[elm], end='    ')
# Method-2
def unit(x):
    return x%10

Sorted = sorted(L1, key = unit)
print("\nSorted array (by Method-2) is:\n", Sorted)
```

```
# Sample run of ques3_sort.py
Enter a integer list having 3 digits only:
 [543, 981, 452, 767, 786]
Sorted array (by Method-1) is:
  981      452      543      786      767
Sorted array (by Method-2) is:
 [981, 452, 543, 786, 767]
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 329 → Write a program to perform sorting on a given list of strings, on the basis of length of strings. That is, the smallest-length string should be the first string in the list and the largest-length string should be the last string in the sorted list.

```
# File Name: ques4_sort.py
L = eval(input('Enter a list of strings:\n '))
SL = sorted(L, key = len)
print('The sorted list according the length of string
is:')
print(' ',SL)
```

```
# Sample run of ques4_sort.py
Enter a list of strings:
 ['Handsome', 'is', 'who', 'does', 'handsome']
The sorted list according the length of string is:
  ['is', 'who', 'does', 'Handsome', 'handsome']
```

Example: 330 → Write a program that sorts a list of tuple-elements in descending order of Points using Bubble sort the tuple-elements of the list contain following information about different players:

(PlayerNo, Playname, Points)

Sample content of the list before sorting:

```
[(103, Ritika, 3001), (104, John, 2819),
 (101, Razia, 3451), (105, Tarandeep, 2971)]
```

After sorting the list would be like:

```
[(101, Razia, 3451), (103, Ritika, 3001),
 (105, Tarandeep, 2971) (104, John, 2819)]
```

```
# File Name: ques5_sort.py
def insertSort(arr):
    p = []
    for elm in arr:
        m = list(elm)
        p.append(m)
    for i in range(1, len(p)):
        key = p[i][2]
        j = i-1
        while j >=0 and key < p[j][2]:
            p[j+1][2] = p[j][2]
            j -= 1
            p[j+1][2] = key
    print("Sorted list is:")
    for i in range(len(arr)-1,0,-1):
        print(p[i])
```

```
# Sample run of ques5_sort.py
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

```
Sorted list is:
[105, 'Tarandeep', 3451]
[101, 'Razia', 3001]
[104, 'John', 2971]
```

Example: 331 → Write a program that sorts a list of tuple-elements in descending order of Points. The tuple-elements of the list contain following information about different players: (PlayerNo, Playname, Points)

Sample content of the list before sorting:

```
[(103, Ritika, 3001), (104, John, 2819),
 (101, Razia, 3451), (105, Tarandeep, 2971)]
```

After sorting the list would be like:

```
[(101, Razia, 3451), (103, Ritika, 3001),
 (105, Tarandeep, 2971) (104, John, 2819)]
```

```
# File Name: ques6_sort.py
L = [(103, 'Ritika', 3001), (104, 'John', 2819),
     (101, 'Razia', 3451), (105, 'Tarandeep', 2971)]

def third(elm):
    return elm[2:]

s = sorted(L, key = third, reverse = True)
print('Sorted List as required is:')
for elm in s:
    print(elm)
```

```
# Sample run of ques6_sort.py
Sorted List as required is:
(101, 'Razia', 3451)
(103, 'Ritika', 3001)
(105, 'Tarandeep', 2971)
(104, 'John', 2819)
```

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

CONCEPT OF STATES AND TRANSITIONS

INTRODUCTION

It indicates how programs can be thought of in terms of states and transitions, and how ordinary imperative programs can be transformed into functional ones.

IMPORTANT TERMINOLOGY

A finite state machine, states, inputs and outputs

STATE

The state is defined as a set of information sufficient to determine the future possible behaviours of a system. In other words, the state tells us how the system can change. It also can tell us something about what has happened in the past, if we know it to have been started in a particular initial state.
Each state is represented by a circle or oval.

TRANSITION

A transition is defined as a marked change from one state to another in rewponse to some external input and or stimulus.
Each transition is representation by an arrow (not necessarily straight arrow) with the level of input (label representing stimulus).

TRANSITION RELATION

A transition relation shows how the system has progressed from one state to another. It is shown by ⇒ as:
`S1 ⇒ S2` is a transition between states **`(S1,S2)`**, where **`S1`** is called predecessor state and **`S2`** is called successor state.

STATE-TRANSITION DIAGRAM

State Transition Diagrams are a type of directed graph, in which the graph nodes represent states and labels on the graph edges represent actions (external input or stimulus). The change of state is marked as a transition.
A state-transition diagram is simply abbreviated as STD or State Diagram.

DIFFERECNE BETWEEN STD AND FLOWCHART

A flowchart illustrates processes that are executed in the system that change the state of objects. A state transition diagram shows the actual changes in state, not the processes or commands that created those changes.

UTILITY OF STATE-TRANSITION DIAGRAM

- The state-transition diagram (STD) is used in Theory of Automata.
- It is useful to understand any automated machine.

Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

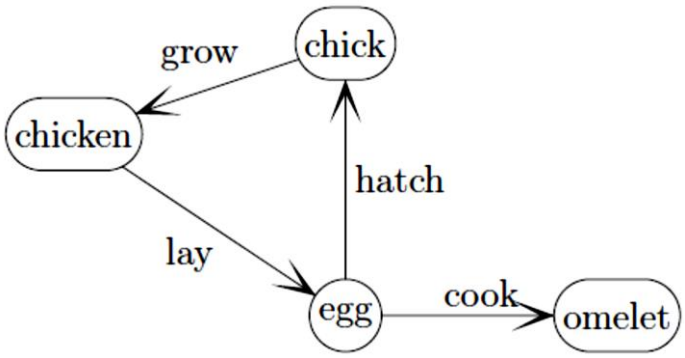
- STD is used in Object Oriented Modelling.
- STD is used in Unified Modelling Language (UML).
- State transition diagram is used in software testing.

AUTOMATON

An automaton is an extremely simple model of a computer or a program. Automata are defined by state transition diagrams. An automaton examines an input string character by character and either say "yes" (accept the string) or "no" (reject the string).

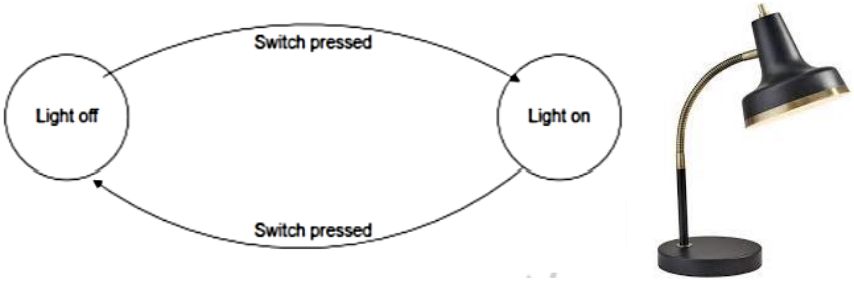
EXAMPLES OF STDs

Example: 332 → Make an state transition diagram for the life cycle of a chicken.



System: **Life cycle of a chicken.**
States: **chick, chicken, egg, omelette.**
Actions: **grow, lay, hatch, cook.**
Transitions: **chick ⇒ chicken, chicken ⇒ egg, egg ⇒ chick, egg ⇒ omelet**

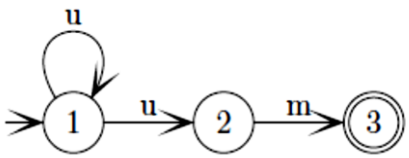
Example: 333 → Make the state transition diagram for a simple desk-lamp.



System: **Desk lamp.**
States: **Light off, Light on.**
Actions: **Switch pressed.**
Transitions: **Light off ⇒ Light on, Light on ⇒ Light off**

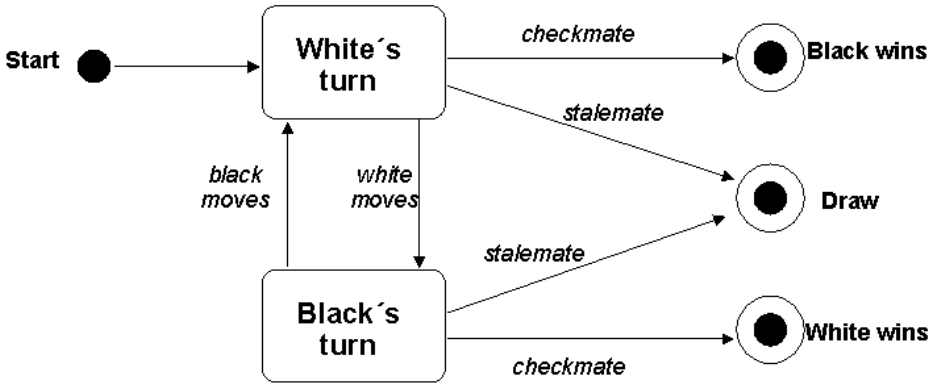
Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

Example: 334 → Make an STD to create a pattern of um, uum, uuum, uuuum, and so on.



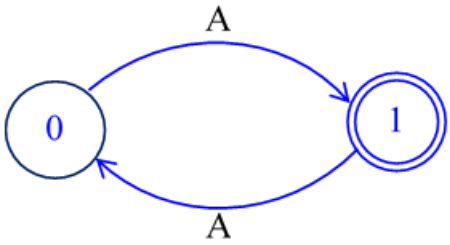
System: **Pattern of um, uum, uuum, uuuum, etc.**
States: **1, 2, 3.**
Actions: **u, m.**
Transitions: **1 ⇒ 1, 1 ⇒ 2, 2 ⇒ 3,**

Example: 335 → Display the STD for the Chess game.



System: **The chess game.**
States: **White's turn, Balck's turn, Three final states – Black wins, White wins, Draw.**
Actions: **black moves, white moves, checkmate, stalemate**
Transitions: **White's turn ⇒ Black's turn,**
White's turn ⇒ Black wins,
White's turn ⇒ Draw,
Black's turn ⇒ White's turn,
Black's turn ⇒ White wins,
Black's turn ⇒ Draw,

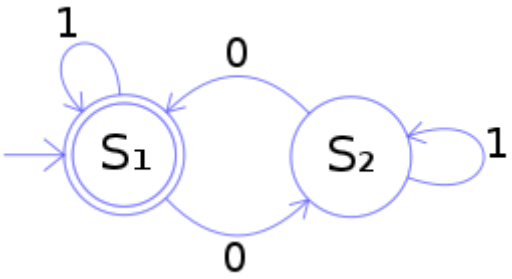
Example: 336 → Make a state transition diagram that accepts A, AAA, AAAAA, and so on, i.e. any string containing odd number of A, i.e. A^{2k+1} (where, k = 0, 1, 2, 3, …., up to infinity)



Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

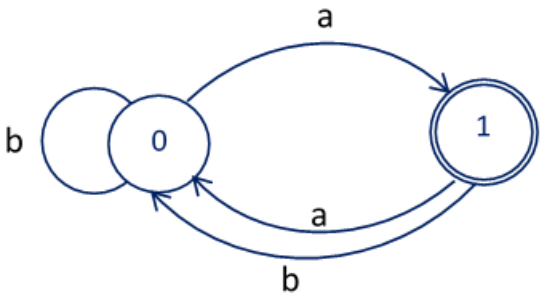
System: **Pattern of string A^{2k+1}, i.e. A, AAA, AAAAA, etc**
States: **0 and 1. (1 is the end state)**
Actions: **A.**
Transitions: **0 ⇒ 1, 1 ⇒ 0.**

Example: 337 → State transition diagram for an automaton for the binary number system (bin).



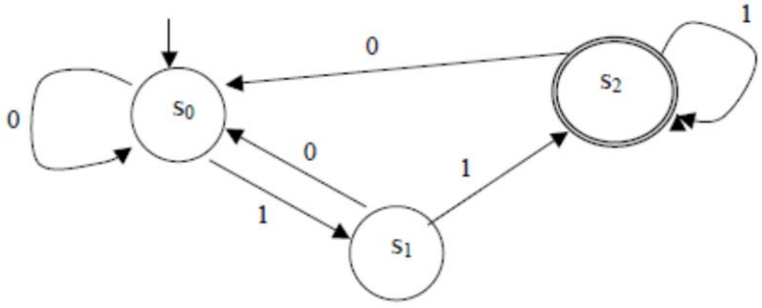
System: **Binary Number System (BIN)**
States: **S_1 and S_2.**
Actions: **0, 1.**
Transitions: **$S_1 \Rightarrow S_1$, $S_1 \Rightarrow S_2$, $S_2 \Rightarrow S_2$, $S_2 \Rightarrow S_1$.**

Example: 338 → Make the STD for the automaton that accepts any string ending with an odd number of a's.



System: **An automaton accepting any string ending with an odd number of a's**
States: **0 and 1. (1 is the end state)**
Actions: **a and b.**
Transitions: **0 ⇒ 0, 0 ⇒ 1, 1 ⇒ 0**

Example: 339 → Consider the finite-state automaton A:



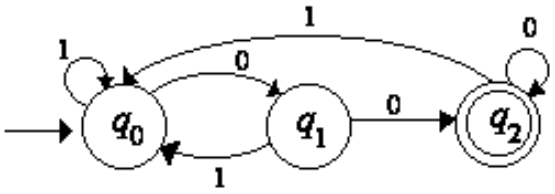
Author of This Tutorial:
Er. Akhlaque Nasra
B. Tech, Computer Science [AMIE (INDIA)]

To what states does *A* go if symbols of the following strings are input to *A* in sequence, starting from the initial state?

(a) 11 (b) 0101 (c) 011011 (d) 00110

Answers: (a) S_2 (the final or accepting state).
(b) S_1 (c) S_2 (d) S_0 (starting state)

Example: 340 → Consider the finite-state automaton **M** shown as the diagram below.

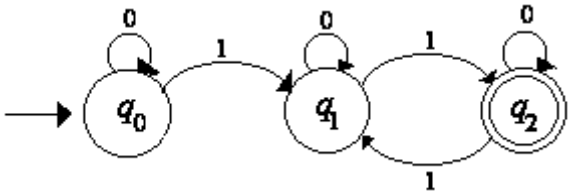


(a) What state does **M** end up in if **M** reads the following input strings and starts from the start state?
(i) 10 (ii) 10100 (iii) 110010 (iv) 1101000 (v) 1010011

(b) Which of the strings in part (a) are accepted by **M**?

Answers: (a) (i) q_1 (ii) q_2 (iii) q_1 (iv) q_2 (v) q_0
(b) The strings 10100 and The strings 10100 and 1101000 are accepted by **M** as both strings result in **M** being in state q_2 at the end of the input are accepted by **M** as both strings result in **M** being in state q_2 at the end of the input

Example: 341 → Consider the finite-state automaton ***M*** defined by the state diagram shown below:



(a) What are the states of ***M***?
(b) What is the start state of ***M***?
(c) What are the accept states of ***M***?
(d) Find $T(q_2, 1)$. ***T*** stands for transition.
(e) Create the transition table of ***M***.

Answers: (a) States of ***M*** are: q_0, q_1, q_2 (b)The start state of ***M*** is q_0.
(c) The accepting state of ***M*** is: q_2
(d) $T(q_2, 1) = q_1$ (as there is an arrow from q_2 to q_1 labeled 1.)
(e) The transition table of ***M*** is :

State \ Input	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_1