



Unit – 2

Basics of Python Programming

CONDITIONAL AND ITERATIVE STATEMENTS

INTRODUCTION

- Programs are written for the solution to the real world problems.
- A language should have the ability to control the flow of execution so that at different intervals different statements can be executed.
- Structured programming is a paradigm aims at controlling the flow of execution of statements in a program by using control structures.
- A language which supports the control structures is called as structured programming language

TYPES OF CONTROL STRUCTURES

TYPES OF CONTROL STRUCTURES

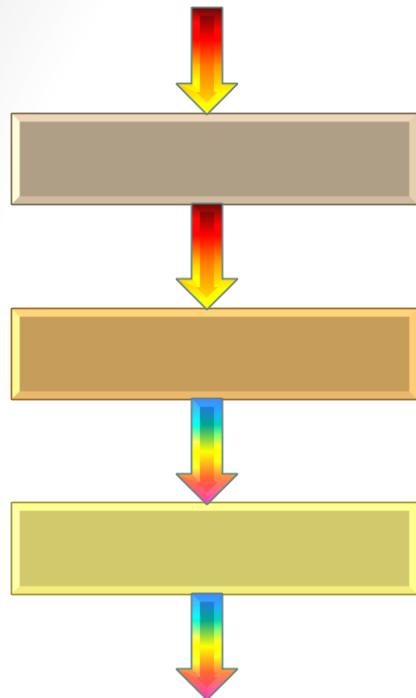
A Structured programming is an important feature of a programming language which comprises following logical structure:

1. SEQUENCE

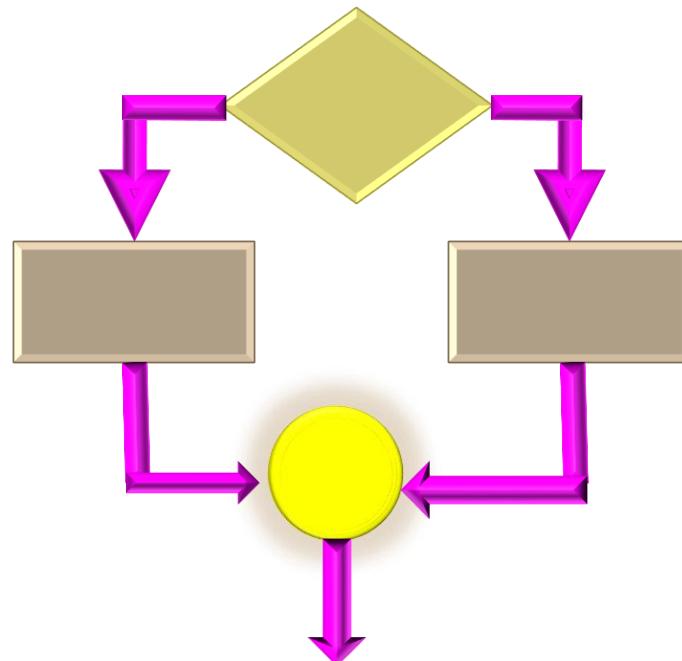
2. SELECTION

3. ITERATION OR LOOPING

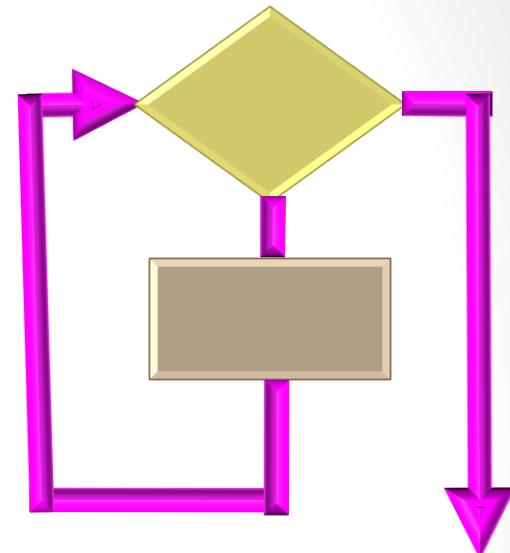
4. BRANCHING OR JUMPING STATEMENTS



SEQUENCE



SELECTION



ITERATION

1. SEQUENCE

Sequence is the default control structure; instructions are executed one after another.

Statement 1

Statement 2

Statement 3

.....

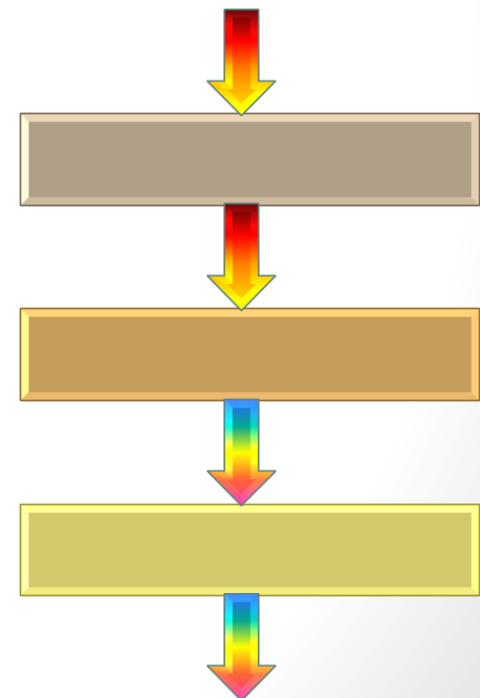
.....

.....

Statement 1

Statement 2

Statement 3

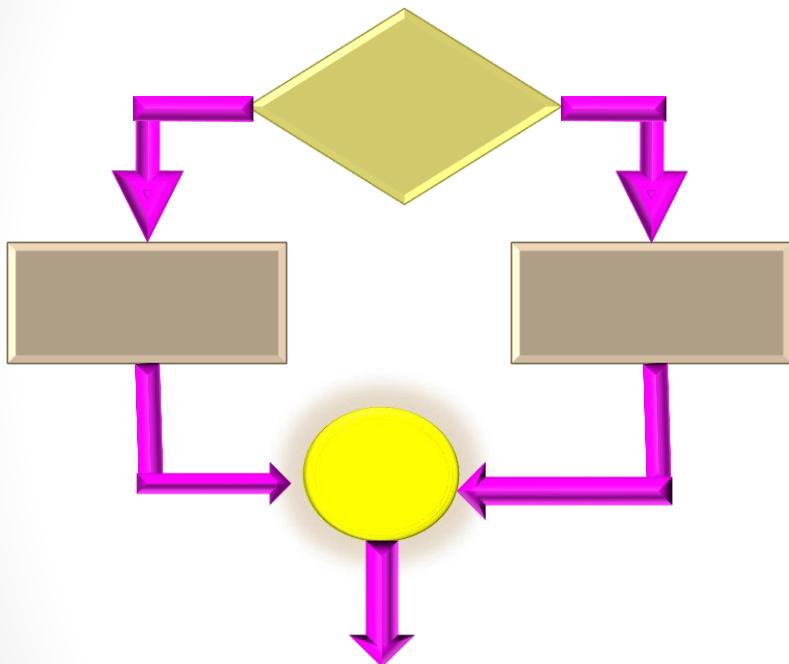


1. SEQUENCE - PROGRAM

Sequence is the default control structure; instructions are executed one after another.

```
# This program adds two numbers
def sum_of_two_no():
    num1 = 1.5
    num2 = 6.3
    sum = float(num1) + float(num2)
    print('The sum is = ', sum)
sum_of_two_no()
```

2. SELECTION



A **selection statement** causes the program control to be transferred to a specific flow based upon whether a certain condition is true or not.

SELECTION

CONDITIONAL CONSTRUCT

if STATEMENT

- The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block.

```
if condition:  
    statement
```

e.g.

: Colon Must

```
num = int(input("enter the number:"))  
if num%2 == 0:  
    print("Number is an even number")
```

CONDITIONAL CONSTRUCT

if-else STATEMENT

- If the condition is true, then the if-block is executed.
Otherwise, the else-block is executed.

: Colon Must

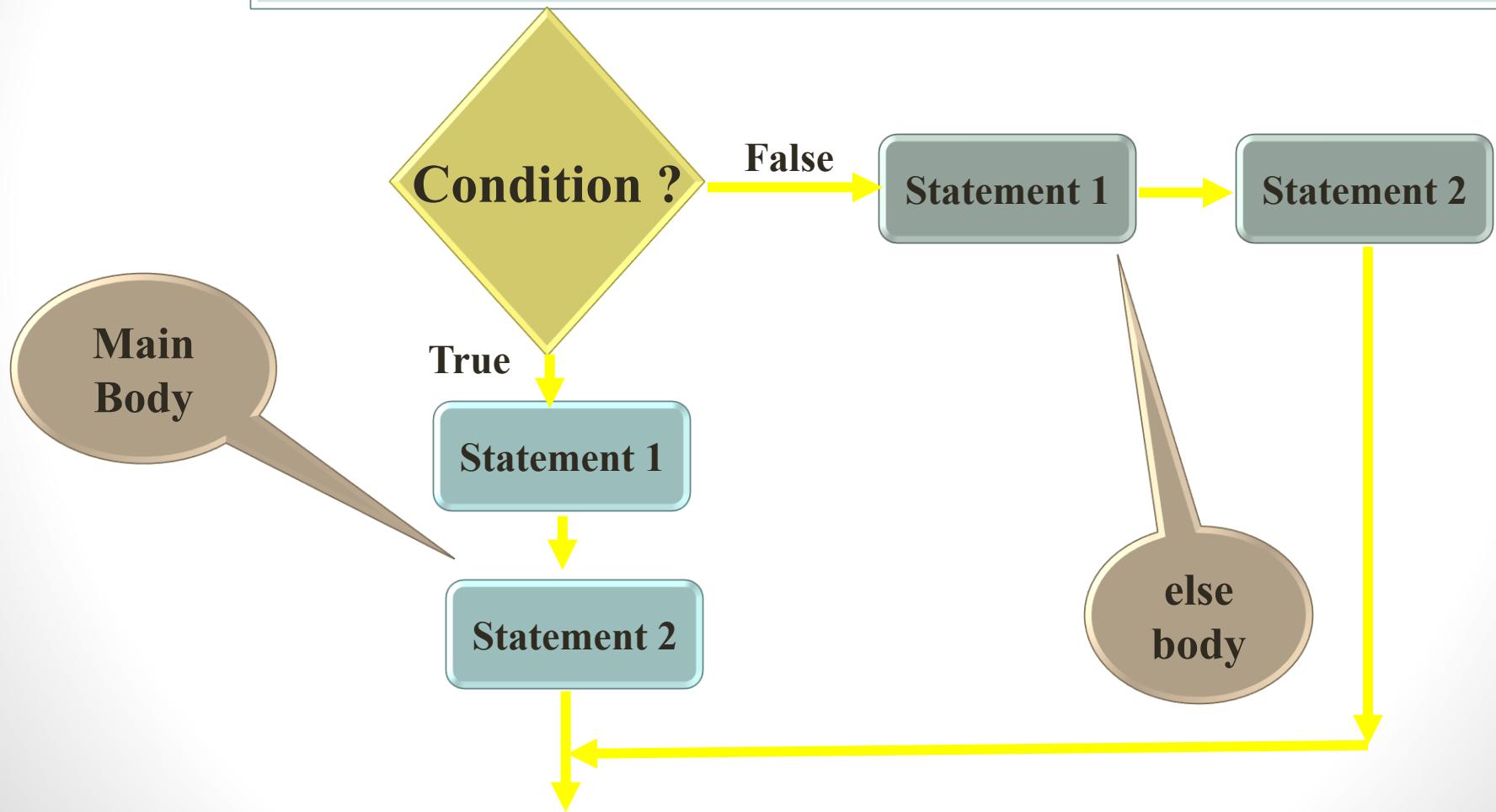
```
if expression:  
    statement  
else:  
    statement
```

e.g.

```
age = int (input("Enter your age: "))  
if age>=18:  
    print("You are eligible to vote !!");  
else:  
    print("Sorry! you have to wait !!");
```

CONDITIONAL CONSTRUCT – if else STATEMENT

FLOW CHART



CONDITIONAL CONSTRUCT

elif STATEMENT

- The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

if expression:

statement

elif:

statement

elif:

statement

else:

statement

CONDITIONAL CONSTRUCT

elif STATEMENT

e.g.

```
number = int(input("Enter the number?"))

if number==10:
    print("The given number is equals to 10")
elif number==50:
    print("The given number is equal to 50");
elif number==100:
    print("The given number is equal to 100");
else:
    print("The given number is not equal to 10, 50 or 100");
```

PROGRAM LIST ON if CONSTRUCT

BELOW AVERAGE PROGRAMS

- 1. Write a PYTHON program that reads a value of n and check the number is zero or non zero value.**

- 2. Write a PYTHON program to find a largest of two numbers.**

- 3. Write a PYTHON program that reads the number and check the no is positive or negative.**

- 4. Write a PYTHON program to check entered character is vowel or consonant.**

AVERAGE PROGRAMS

1. Write a PYTHON program to evaluate the student performance

If % is ≥ 90 then Excellent performance

If % is ≥ 80 then Very Good performance

If % is ≥ 70 then Good performance

If % is ≥ 60 then average performance

else Poor performance.

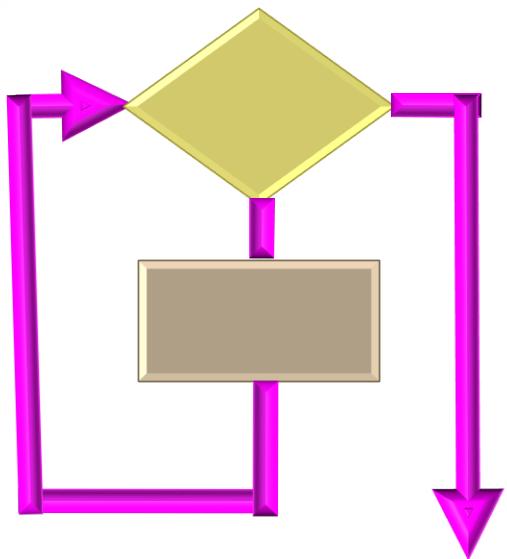
2. Write a PYTHON program to find largest of three numbers.
3. Write a PYTHON program to find smallest of three numbers

ABOVE AVERAGE PROGRAMS

- 1. Write a PYTHON program to check weather number is even or odd.**
- 2. Write a PYTHON program to check a year for leap year.**
- 3. A company insures its drivers in the following cases:**
 - If the driver is married.**
 - If the driver is unmarried, male and above 30 years of age.**
 - If the driver is unmarried, female and above 25 years of age.**

In all the other cases, the driver is not insured.
Write a PYTHON program to determine whether the driver is insured or not

3. ITERATION OR LOOPING



ITERATION

- Loops can execute a block of code number of times until a certain condition is met.
- OR**
- The iteration statement allows instructions to be executed until a certain condition is to be fulfilled.
 - The iteration statements are also called as loops or Looping statements.

3. ITERATION OR LOOPING

Python provides two kinds of loops as,

while loop

for loop

while loop

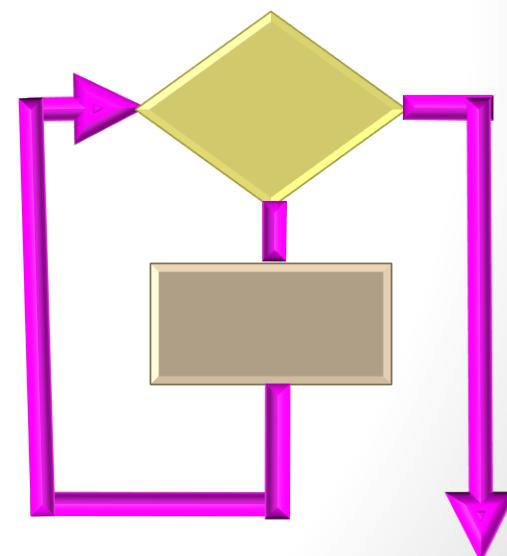
A while loop allows general repetition based upon the repeated testing of a condition

Syntax -

*while condition:
body statements*

: Colon Must

The loop iterates while the expression evaluates to true, when expression becomes false the loop terminates.



while loop

while loop

E.g. -

i=1

while i<=10:

print(i, end=' ')

i+=1

while with else

Syntax -

: Colon Must

While *condition*:

body statements

executes the statements until sequences are exhausted

else:

statements

executes these statements when while loop is completed

while loop with else

E.g. -

i=1

while i<=10:

print(i, end=' ')

i+=1

else:

print("Program Ends")

while loop - Programs

Class work / Home Work

while loop - programs

BELOW AVERAGE PROGRAMS

- 1. Write a PYTHON program to print the natural numbers up to n**

- 2. Write a PYTHON program to print even numbers up to n**

- 3. Write a PYTHON program to print odd numbers up to n**

- 4. Write a PYTHON program to print sum of natural numbers up to n**

while loop - programs

AVERAGE PROGRAMS

- 1. Write a PYTHON program to print sum of odd numbers up to n**

- 2. Write a PYTHON program to print sum of even numbers up to n**

- 3. Write a PYTHON program to print natural numbers up to n in reverse order.**

- 4. Write a PYTHON program to print Fibonacci series up to n**

- 5. Write a PYTHON program find a factorial of given number**

while loop - programs

ABOVE AVERAGE PROGRAMS

- 1. Write a PYTHON program to check the entered number is prime or not**

- 2. Write a PYTHON program to find the sum of digits of given number**

- 3. Write a PYTHON program to check the entered number is palindrome or not**

- 4. Write a PYTHON program to reverse the given number.**

while loop - programs

ABOVE AVERAGE PROGRAMS

- 1. Write a PYTHON program to print the multiplication table**

- 2. Write a PYTHON program to print the largest of n numbers**

- 3. Write a PYTHON program to print smallest of n numbers**

for loop

- Python's **for-loop** syntax is a more convenient alternative to a while loop when iterating through a series of elements. The for-loop syntax can be used on any type of iterable structure, such as a list, tuple str, set, dict, or file

Syntax -

for element in iterable:
body statements

: Colon Must

for loop executes a code block multiple times

for loop

- Example

```
# Iterating over a String  
print("String Iteration")
```

```
s = "Geeks"  
for i in s:  
    print(i)
```

*Output –
String Iteration*

*G
e
e
k
s*

for loop with else

Syntax -

for element in iterable:

body statements

executes the statements until sequences are exhausted

else:

statements

executes these statements when for loop is completed

: Colon Must

for loop with else

- Example

tuple_ = (3, 4, 6, 8, 9, 2, 3, 8, 9, 7)

for value in tuple_:

if value % 2 != 0:

print(value)

else:

print("These are the odd numbers present in the tuple")

for loop - range keyword

- The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Syntax -

range(start, stop, step)

e.g.

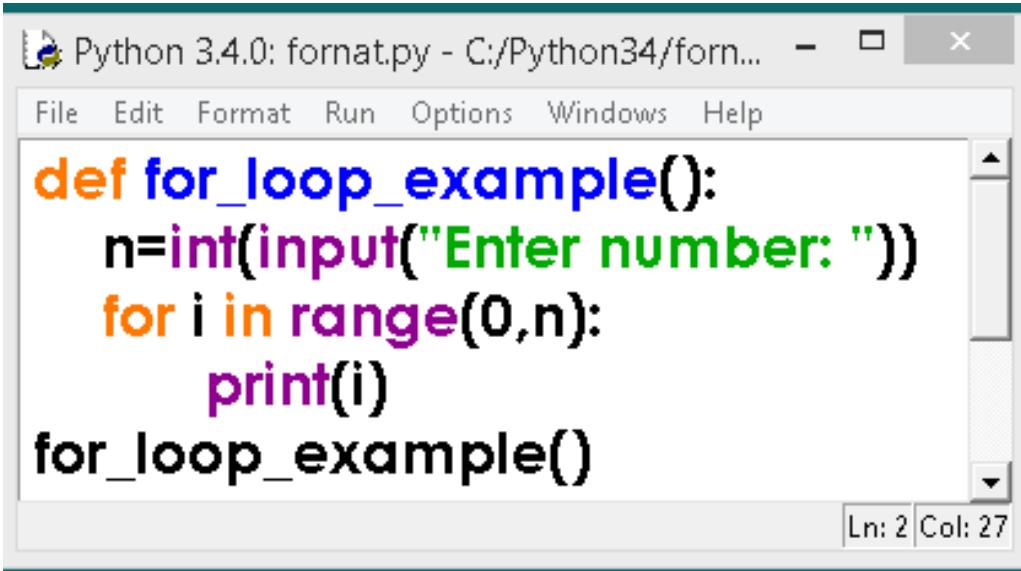
```
for n in range(3,6):  
    print(n)
```

OR

```
x = range(3, 6)  
for n in x:  
    print(n)
```

for LOOP - range KEYWORD

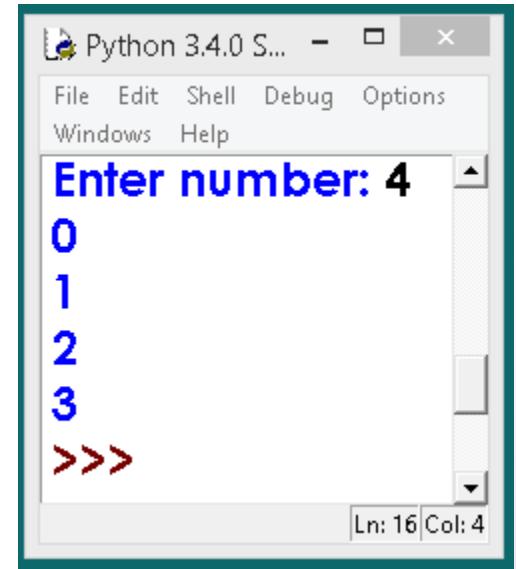
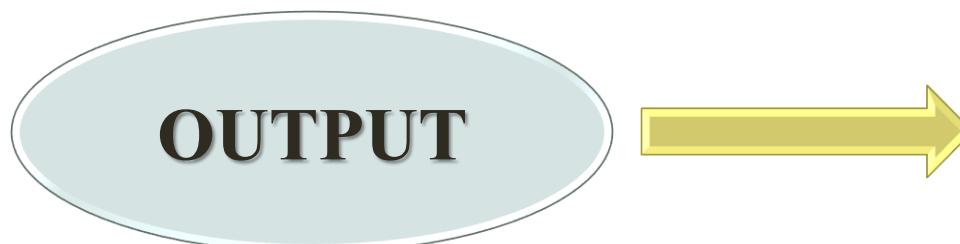
#Generating series of numbers



Python 3.4.0: fornat.py - C:/Python34/fornat.py

```
def for_loop_example():
    n=int(input("Enter number: "))
    for i in range(0,n):
        print(i)
for_loop_example()
```

Ln: 2 Col: 27



Python 3.4.0 S...

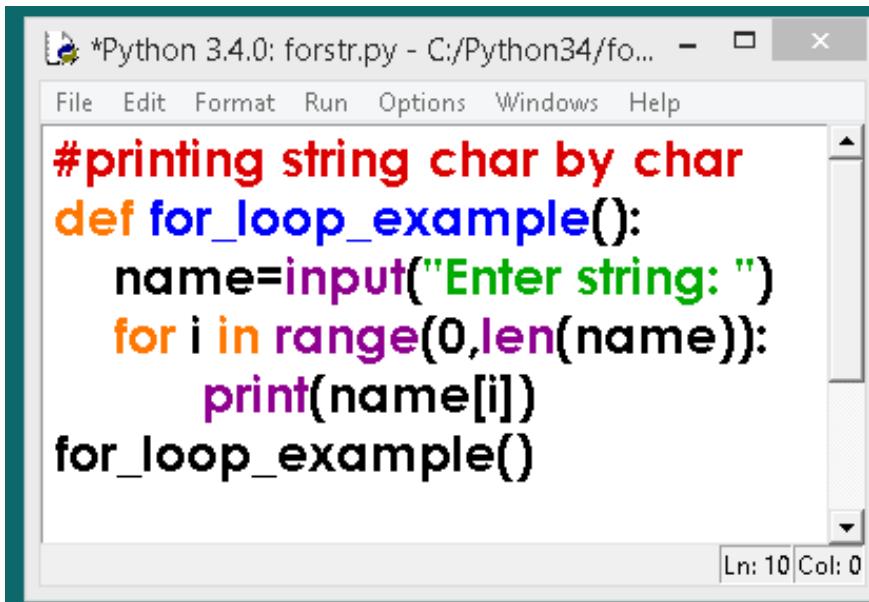
```
Enter number: 4
0
1
2
3
>>>
```

Ln: 16 Col: 4

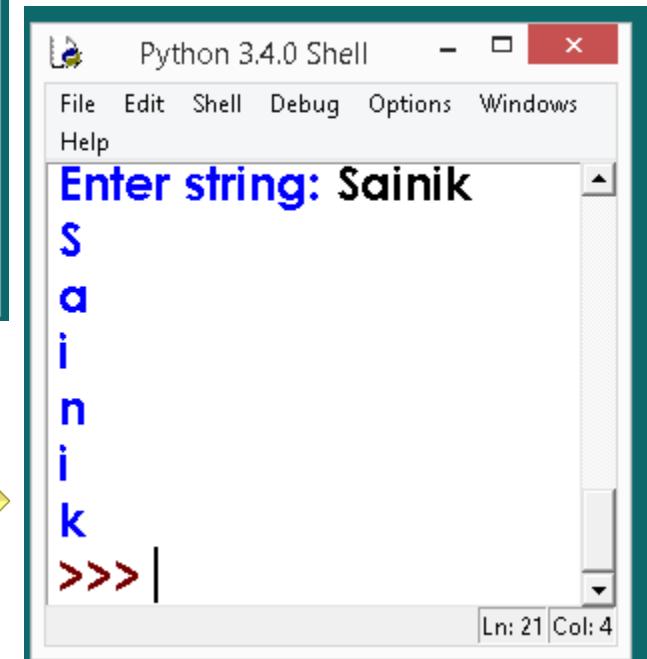
for LOOP – len() FUNCTION

for LOOP - range KEYWORD and len()

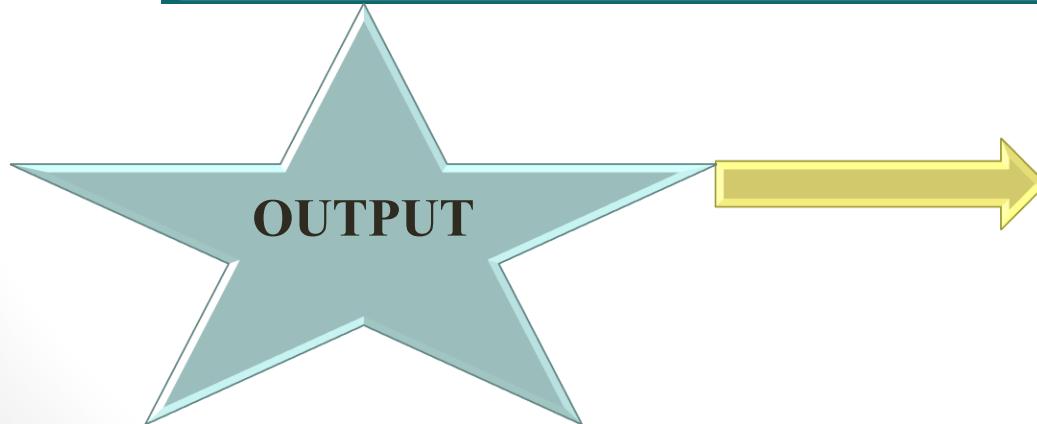
print string character by character



```
#printing string char by char
def for_loop_example():
    name=input("Enter string: ")
    for i in range(0,len(name)):
        print(name[i])
for_loop_example()
```



```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Enter string: Sainik
S
a
i
n
i
k
>>>
```



for loop - Programs

for loop - Programs

1. Write a PYTHON program to print the natural numbers up to n
2. Write a PYTHON program to print even numbers up to n
3. Write a PYTHON program to print odd numbers up to n
4. Write a PYTHON program that prints 1 2 4 8 16 32 ... n^2
5. Write a PYTHON program to sum the given sequence
 $1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!$

for loop - Programs

6. Write a PYTHON program to compute the cosine series
$$\cos(x) = 1 - x^2 / 2! + x^4 / 4! - x^6 / 6! + \dots x^n / n!$$
7. Write a short PYTHON program to check weather the square root of number is prime or not.
8. Write a PYTHON program to produce following design

A B C

A B C

A B C

for loop - Programs

9. Write a PYTHON program to produce following design

A

A B

A B C

A B C D

A B C D E

If user enters n value as 5

10. Write a PYTHON program to produce following design

A B C D E

A B C D

A B C

A B

A

(If user enters n value as 5)

for loop - Programs

11. Write a PYTHON program to produce following design

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

If user enters n value as 5

12. Write a PYTHON program to produce following design

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

If user enters n value as 5

4. BRANCHING OR JUMPING STATEMENTS

Python has an unconditional branching statements and they are,

1. break STATEMENT

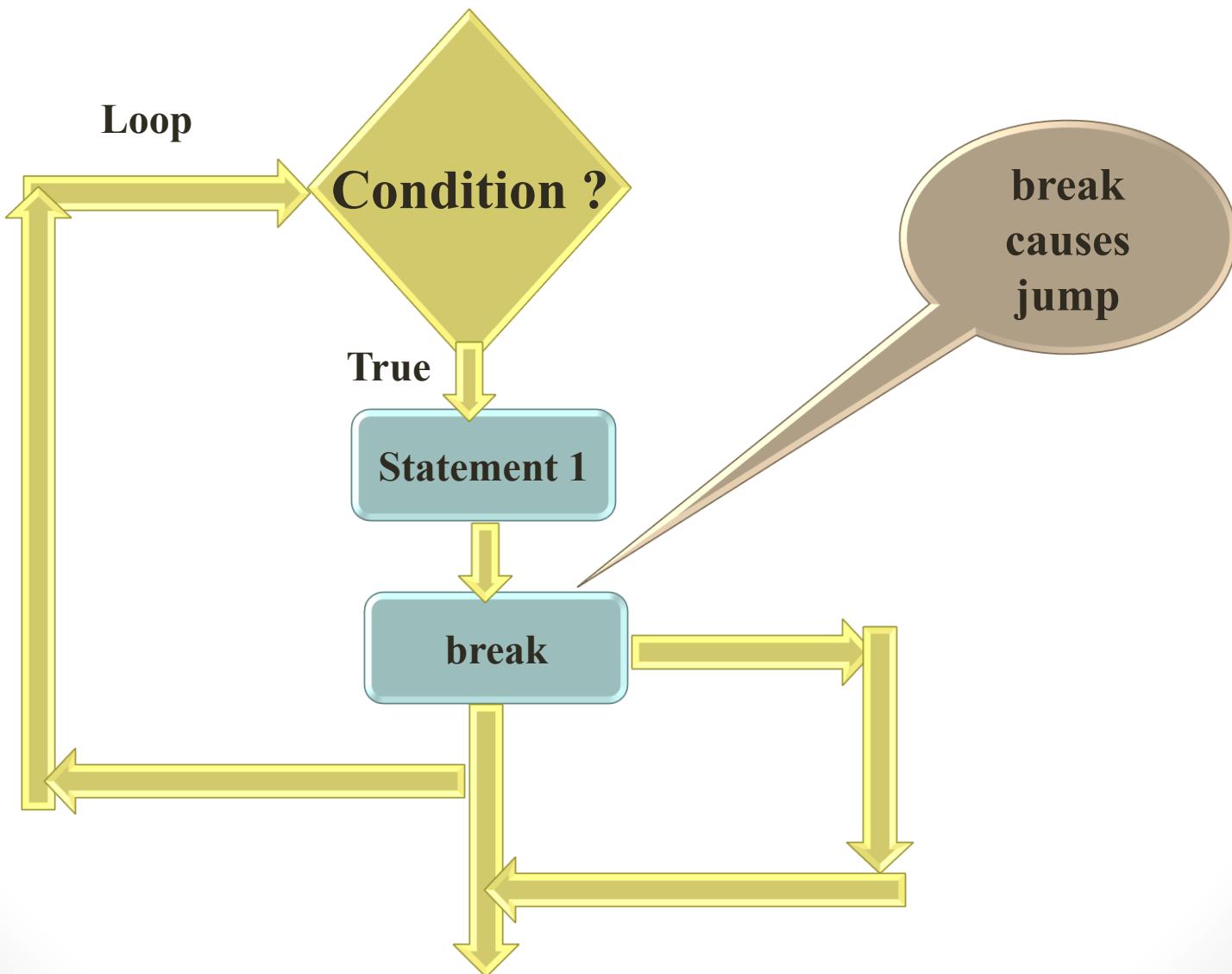
2. continue STATEMENT

3. pass STATEMENT

1. break STATEMENT

- **Break can be used to unconditionally jump out of the loop.**
- **It terminates the execution of the loop.**
- **Break can be used in while loop and for loop.**
- **Break is mostly required, when because of some external condition, we need to exit from a loop.**

1. break STATEMENT



1. break STATEMENT

```
for string in "Python Loops":
```

```
    if string == 'L':
```

```
        break
```

```
    print('Current Letter: ', string)
```

Output:

Current Letter: P

Current Letter: y

Current Letter: t

Current Letter: h

Current Letter: o

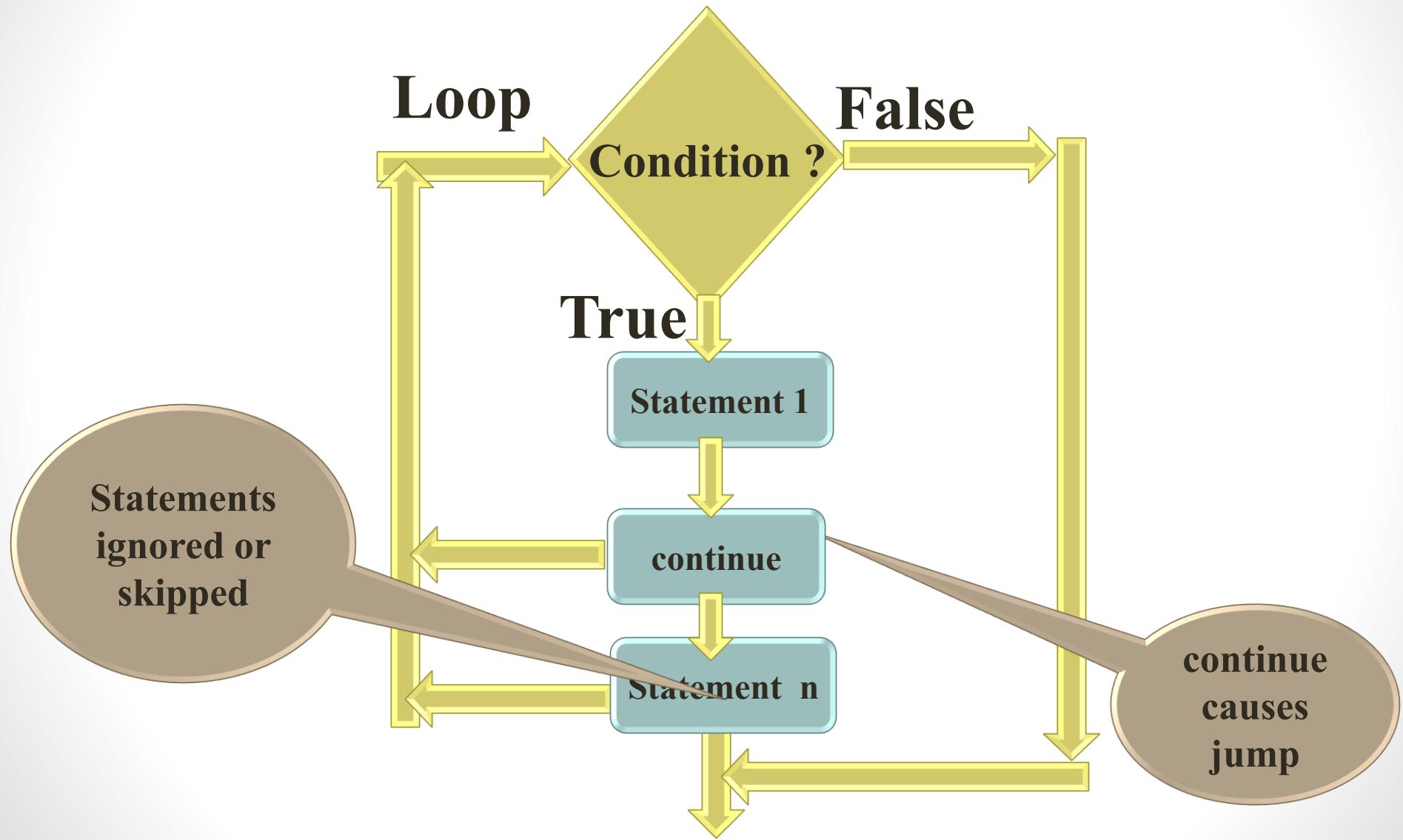
Current Letter: n

Current Letter:

2. continue STATEMENT

- The **continue** statement in Python returns the control to the beginning of the while loop.
- The **continue** statement **skip** all the remaining statements in the current iteration of the loop and **moves the control back to the top of the loop.**
- The **continue** statement can be used in both while and for loops.

2. continue STATEMENT



2. continue STATEMENT

for string in "Python Loops":

if string == "o" or string == "p" or string == "t":

continue

print('Current Letter:', string)

Output:

Current Letter: P

Current Letter: y

Current Letter: h

Current Letter: n

Current Letter:

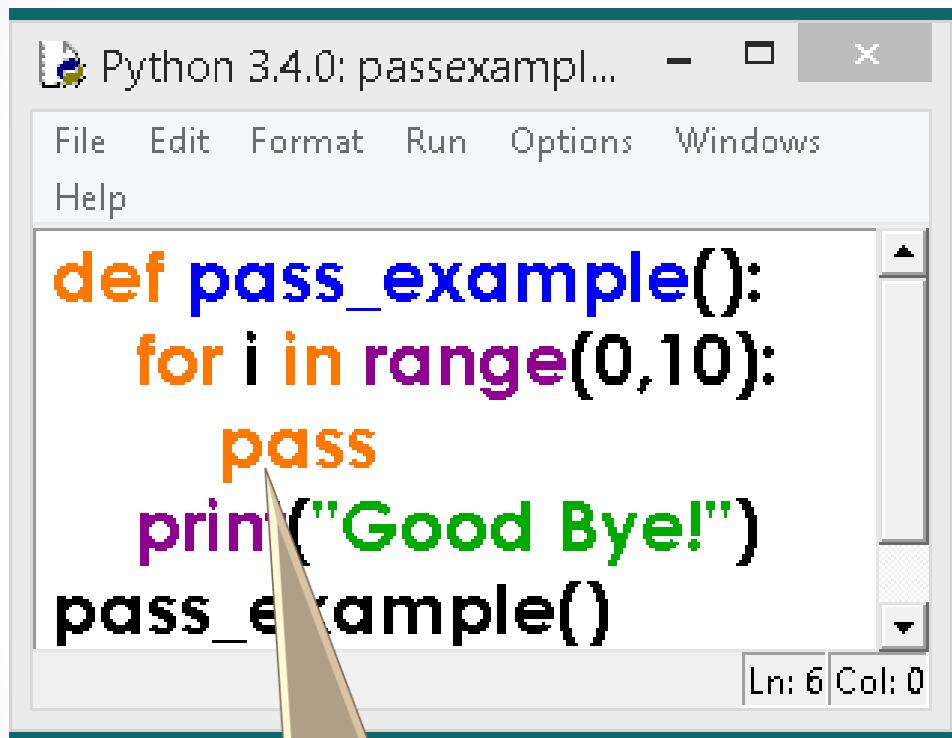
Current Letter: L

Current Letter: s

3. pass STATEMENT

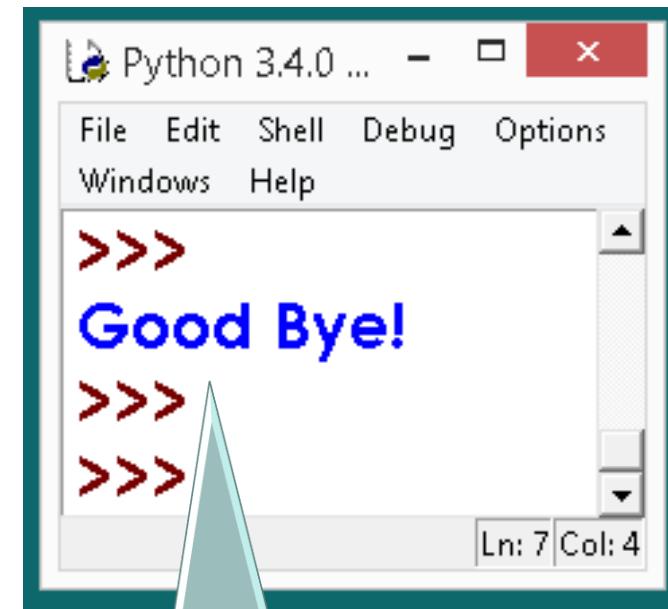
- The **pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.**
- The **pass statement is a *null* operation; nothing happens when it executes.**
- The **pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):**

pass STATEMENT



```
Python 3.4.0: passexampl... - □ ×  
File Edit Format Run Options Windows  
Help  
def pass_example():  
    for i in range(0,10):  
        pass  
    print("Good Bye!")  
pass_example()  
Ln: 6 Col: 0
```

pass in loop



```
Python 3.4.0 ... - □ ×  
File Edit Shell Debug Options  
Windows Help  
>>>  
Good Bye!  
>>>  
>>>  
Ln: 7 Col: 4
```

pass in loop has
no output

String

String

- Python string is the collection of the characters surrounded by **single quotes**, **double quotes**, or **triple quotes**.
- E.g.

str = "Hi Python!"

Strings +ve indexing and splitting

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[:] = 'HELLO'

str[1] = 'E'

str[0:] = 'HELLO'

str[2] = 'L'

str[:5] = 'HELLO'

str[3] = 'L'

str[:3] = 'HEL'

str[4] = 'O'

str[0:2] = 'HE'

str[1:4] = 'ELL'

num-1

Strings +ve indexing and splitting

```
# Given String  
str = "JAVATPOINT"  
# Start 0th index to end  
print(str[0:])  
# Starts 1th index to 4th index  
print(str[1:5])  
# Starts 2nd index to 3rd index  
print(str[2:4])  
# Starts 0th to 2nd index  
print(str[:3])  
#Starts 4th to 6th index  
print(str[4:7])
```

```
JAVATPOINT  
AVAT  
VA  
JAV  
TPO
```

Strings –ve indexing and splitting

`str = "HELLO"`

H	E	L	L	O
-5	-4	-3	-2	-1

`str[-1]` = 'O'

`str[-3:-1]` = 'LL'

`str[-2]` = 'L'

`str[-4:-1]` = 'ELL'

`str[-3]` = 'L'

`str[-5:-3]` = 'HE'

`str[-4]` = 'E'

`str[-4:]` = 'ELLO'

`str[-5]` = 'H'

`str[::-1]` = 'OLLEH'

num-1

Strings –ve indexing and splitting

```
str = 'JAVATPOINT'
```

```
print(str[-1])
```

```
print(str[-3])
```

```
print(str[-2:])
```

```
print(str[-4:-1])
```

```
print(str[-7:-2])
```

Reversing the given string

```
print(str[::-1])
```

```
print(str[-12])
```

```
T  
I  
NT  
OIN  
ATPOI
```

```
TNIOPTAVAJ
```

```
IndexError: string index out of range
```

String Functions

- Python split() method splits the string into a comma separated list.
- It separates string based on the separator delimiter.
- It allows you to split a string into substrings based on a specified separator.

e.g.

```
string = "Hello World"
```

```
result = string.split()
```

```
print(result)
```

o/p - ['Hello', 'World']

String Functions

Strings in python are surrounded by either single quotation marks, or double quotation marks. String is immutable.

Example

```
a="Hello" or a='Hello'
```

➤ Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

```
a = """Welcome to the TY-C  
DYP CET, Kolhapur  
2024-25 batch."""
```

```
print(a)
```

Or three single quotes:

```
a = '''Welcome to the TY-C  
DYP CET, Kolhapur  
2024-25 batch.'''
```

```
print(a)
```

- Square brackets can be used to access elements of the string.

Example:

A=“Hello”

Print(A[0])

- To get the length of a string, use the **len()** function.
- To check if a certain phrase or character is present in a string, use the keyword **in**.
- To check if a certain phrase or character is NOT present in a string, use the keyword **not in**.

➤ **Slicing** : return a range of characters

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example:

A=“Hello World”

Print(A[1:5])

Print(A[:5])

Print(A[5:])

Use negative indexes to start the slice from the end of the string.

Example:

A=“Hello World”

Print(A[-8:-2])

- The **upper()** method returns the string in upper case.
- The **lower()** method returns the string in lower case.
- The **strip()** method removes any whitespace from the beginning or the end.
- The **replace()** method replaces a string with another string.
- The **split()** method splits the string into substrings if it finds instances of the separator.
- To concatenate, or combine, two strings use the + operator.

```
s=" Hello World "
print(type(s))
print(s[3])
print(s[0:4])
print(s[:6])
print(s[0:])
print(s[0:10:2])
print("Length of string is: ",len(s))
print('l' in s)    #True - 'l' character is present in s string
print('s' in s)    #False - 's' character is not present in s string
print('s' not in s)    #True - 's' character is not present in s string
print(s.lower())      #lowercase
print(s.upper())      #uppercase
print(s.strip())      # removes white space at begining and end
print(s.replace('H','S')) #replace 'H' charatcter with 'S' in other string
print(s)
print(s.split(" "))  # split with white space
print(s.count('l'))  #count the occurance of 'l' character
s1="Welcome to python programming."
s2=s+s1
print(s2)
print(len(s2))
```

String methods

Method	Description
<u>capitalize()</u>	Converts the first character to upper case
<u>casefold()</u>	Converts string into lower case
<u>center()</u>	Returns a centered string
<u>count()</u>	Returns the number of times a specified value occurs in a string
<u>encode()</u>	Returns an encoded version of the string
<u>endswith()</u>	Returns true if the string ends with the specified value
<u>expandtabs()</u>	Sets the tab size of the string
<u>find()</u>	Searches the string for a specified value and returns the position of where it was found

<u>format()</u>	Formats specified values in a string
<u>format_map()</u>	Formats specified values in a string
<u>index()</u>	Searches the string for a specified value and returns the position of where it was found
<u>isalnum()</u>	Returns True if all characters in the string are alphanumeric
<u>isalpha()</u>	Returns True if all characters in the string are in the alphabet
<u>isascii()</u>	Returns True if all characters in the string are ascii characters
<u>isdecimal()</u>	Returns True if all characters in the string are decimals

<u>isdigit()</u>	Returns True if all characters in the string are digits
<u>isidentifier()</u>	Returns True if the string is an identifier
<u>islower()</u>	Returns True if all characters in the string are lower case
<u>isnumeric()</u>	Returns True if all characters in the string are numeric
<u>isprintable()</u>	Returns True if all characters in the string are printable
<u>isspace()</u>	Returns True if all characters in the string are whitespaces
<u>istitle()</u>	Returns True if the string follows the rules of a title
<u>isupper()</u>	Returns True if all characters in the string are upper case

<u>join()</u>	Joins the elements of an iterable to the end of the string
<u>ljust()</u>	Returns a left justified version of the string
<u>lower()</u>	Converts a string into lower case
<u>lstrip()</u>	Returns a left trim version of the string
<u>maketrans()</u>	Returns a translation table to be used in translations
<u>partition()</u>	Returns a tuple where the string is parted into three parts
<u>replace()</u>	Returns a string where a specified value is replaced with a specified value
<u>rfind()</u>	Searches the string for a specified value and returns the last position of where it was found

<u>rindex()</u>	Searches the string for a specified value and returns the last position of where it was found
<u>rjust()</u>	Returns a right justified version of the string
<u>rpartition()</u>	Returns a tuple where the string is parted into three parts
<u>rsplit()</u>	Splits the string at the specified separator, and returns a list
<u>rstrip()</u>	Returns a right trim version of the string
<u>split()</u>	Splits the string at the specified separator, and returns a list
<u>splitlines()</u>	Splits the string at line breaks and returns a list
<u>startswith()</u>	Returns true if the string starts with the specified value

<u>strip()</u>	Returns a trimmed version of the string
<u>swapcase()</u>	Swaps cases, lower case becomes upper case and vice versa
<u>title()</u>	Converts the first character of each word to upper case
<u>translate()</u>	Returns a translated string
<u>upper()</u>	Converts a string into upper case
<u>zfill()</u>	Fills the string with a specified number of 0 values at the beginning

String Functions

- Try other functions at a time of practical.....

List

List

- In Python, the sequence of various data types is stored in a list.
- A list is a collection of different kinds of values or items.
- Since Python lists are mutable, we can change their elements after forming.
- The comma (,) and the square brackets [enclose the List's items] serve as separators.
- Lists written in Python are identical to dynamically scaled arrays defined in other languages, such as ArrayList in Java and Vector in C++.
- A list is a collection of items separated by **commas** and denoted by the symbol **[]**.

List

```
# a simple list
```

```
list1 = [1, 2, "Python", "Program", 15.9]
```

```
list2 = ["Amy", "Ryan", "Henry", "Emma"]
```

```
# printing the list
```

```
print(list1)
```

```
print(list2)
```

```
# printing the type of list
```

```
print(type(list1))
```

```
print(type(list2))
```

```
[1, 2, 'Python', 'Program', 15.9]  
['Amy', 'Ryan', 'Henry', 'Emma']  
< class ' list '>  
< class ' list '>
```

Characteristics of Lists

- The lists are in **order**.
- The list element can be accessed via the **index**.
- The **mutable** (modify) type of List
- The rundowns are changeable sorts.
- The number of **various elements** can be stored in a list.

+ve indexing Lists

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5
---	---	---	---	---	---

List[0] = 0

List[0:] = [0,1,2,3,4,5]

List[1] = 1

List[:] = [0,1,2,3,4,5]

List[2] = 2

List[2:4] = [2, 3]

List[3] = 3

List[1:3] = [1, 2]

List[4] = 4

List[:4] = [0, 1, 2, 3]

List[5] = 5

-ve indexing Lists

List = [0, 1, 2, 3, 4, 5]



list = [0, 1, 2, 3, 4, 5]

`print(list[-1])`

`print(list[-3:])`

`print(list[:-1])`

`print(list[-3:-1])`



5

[3, 4, 5]

[0, 1, 2, 3, 4]

[3, 4]

Iterating a List

```
list = ["John", "David", "James", "Jonathan"]
```

```
for i in list:
```

*# The `i` variable will iterate over the elements of the List and
contains each element in each iteration.*

```
print(i)
```

```
John  
David  
James  
Jonathan
```

Adding Elements to the List

The **append()** function in Python can add a new item to the List.

insert() methods can also add values to a list

```
l = []
```

#Number of elements will be entered by the user

```
n = int(input("Enter the number of elements in the list:"))
```

#for loop to take the input

```
for i in range(0,n):
```

The input is taken from the user and added to the list as the item

```
l.append(input("Enter the item:"))
```

```
print("printing the list items..")
```

traversal loop to print the list items

```
for i in l:
```

```
print(i, end = " ")
```

```
Enter the number of elements in the  
list:2
```

```
Enter the item:32
```

```
Enter the item:56
```

```
printing the list items.. 32 56
```

Removing Elements from the List

The **remove()** function in Python can remove an element from the List

The list elements can also be deleted by using the **del** keyword

```
list = [0,1,2,3,4]
```

```
print("printing original list: ");
```

```
for i in list:
```

```
    print(i,end=" ")
```

```
list.remove(2)
```

```
print("\n printing the list after the removal of first element...")
```

```
for i in list:
```

```
    print(i,end=" ")
```

```
printing original list:
```

```
0 1 2 3 4
```

```
printing the list after the removal of first element...
```

```
0 1 3 4
```

Concatenation

It concatenates the list mentioned on either side of the operator.

```
list1 = [12, 14, 16, 18, 20]
```

```
list2 = [9, 10, 32, 54, 86]
```

```
# concatenation operator +
```

```
l = list1 + list2
```

```
print(l)
```

```
[12, 14, 16, 18, 20, 9, 10, 32, 54, 86]
```

Repetition

The redundancy administrator empowers the rundown components to be rehashed on different occasions

```
list1 = [12, 14, 16, 18, 20]
```

```
# repetition operator *
```

```
l = list1 * 2
```

```
print(l)
```

```
[12, 14, 16, 18, 20, 12, 14, 16, 18, 20]
```

List methods

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

```
l=[10,30,20,50,40,60]
print("Type of Object is: ",type(l))
l.append(70)
print("70 is appended: ",l)
l.insert(2,90)
print("90 is insterted:",l)
l1=[100,300,400,900]
l.extend(l1)
print("Extended List is: ",l)
l.sort()
print("Sorted List is: ",l)
l.reverse()
print("Reverse List is: ",l)
l2=l.copy()
print("Copied List is:",l2)
l.pop()
print("Last element is popped: ",l)
l.remove(900)
print("900 is removed: ",l)
l.append(900)
l.append(900)
print("900 occurance is: ",l.count(900))
```

List Functions

- Try other functions at a time of practical.....

Tuple

Tuple

- Tuples are an **immutable data type**, meaning their elements **cannot be changed** after they are generated.
- Each element in a tuple has a **specific order** that will never change because tuples are ordered sequences.
- All the objects-also known as "elements"-must be separated by a **comma**, enclosed in **parenthesis ()**.
- E.g.

t1 = (4, "Python", 9.3)

print("Tuple with different data types: ", t1)

Count () Method

```
# Creating tuples  
T1 = (0, 1, 5, 6, 7, 2, 2, 4, 2, 3, 2, 3, 1, 3, 2)  
T2 = ('python', 'java', 'python', 'Tpoint', 'python', 'java')  
# counting the appearance of 3  
res = T1.count(2)  
print('Count of 2 in T1 is:', res)
```

Count of 2 in T1 is: 5

index() Method

- The **index(start index, search element)** function returns the first instance of the requested element from the Tuple.
- Start: (Optional) the index that is used to begin the final (optional) search

```
Tuple_data = (0, 1, 2, 3, 2, 3, 1, 3, 2)
```

```
# getting the index of 3
```

```
res = Tuple_data.index(3)
```

```
print('First occurrence of 1 is', res)
```

First occurrence of 1 is 2

First occurrence of 1 after 4th index is: 6

```
# getting the index of 3 after 4th index
```

```
res = Tuple_data.index(3, 4)
```

```
print('First occurrence of 1 after 4th index is:', res)
```

Tuple method

Method	Description
<u>count()</u>	Returns the number of times a specified value occurs in a tuple
<u>index()</u>	Searches the tuple for a specified value and returns the position of where it was found

- The **count()** method returns the number of times a specified value appears in the tuple.

tuple.count(value)

- The **index()** method finds the first occurrence of the specified value.
- The index() method raises an exception if the value is not found.
- To join two or more tuples you can use the + operator.
- If you want to multiply the content of a tuple given number of times, use the * operator.

Tuple Functions

- Try other functions at a time of practical.....

Set

Set

- A Python set is the collection of the **unordered items**.
- Each element in the set must be **unique, mutable**, and the sets **remove the duplicate elements**.
- Sets are mutable which means **we can modify** it after its creation.
- Unlike other collections in Python, there is **no index attached** to the elements of the set, i.e., we cannot directly access any element of the set **by the index**.
- However, we can print them all together, or we can get the list of elements by looping through the set.

Creating Set – using curly braces

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}
```

```
print(Days)
```

```
print(type(Days))
```

```
print("looping through the set elements ... ")
```

```
for i in Days:
```

```
    print(i)
```

Output -

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}
```

Creating Set – using set()

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
```

```
print(Days)
```

```
print(type(Days))
```

```
print("looping through the set elements ... ")
```

```
for i in Days:
```

```
    print(i)
```

Output -

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}
```

Adding items to the set

```
Months = set(["January", "February", "March", "April", "May", "June"])
print("\\nprinting the original set ... ")
print(months)
```

```
print("\\nAdding other months to the set... ");
Months.add("July");
Months.add ("August");
```

```
print("\\nPrinting the modified set... ");
print(Months)
print("\\nlooping through the set elements ... ")
for i in Months:
    print(i)
```

```
Months.update(["July", "August", "September", "October"]);
print("\\nprinting the modified set ... ")
print(Months);
```

Removing items from the set

```
months = set(["January", "February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(months)
```

```
print("\nRemoving some months from the set... ");
months.discard("January");
months.discard("May");
print("\nPrinting the modified set... ");
print(months)
print("\nlooping through the set elements ... ")
for i in months:
    print(i)
```

```
months.remove(" February ");
months.remove(" March ");
print("\nPrinting the modified set... ");
print(months)
```

clear() to remove all the items from the set

```
Months = set(["January", "February", "March", "April", "May", "June"])
```

```
print("\nprinting the original set ... ")
```

```
print(Months)
```

```
print("\nRemoving all the items from the set...");
```

```
Months.clear()
```

```
print("\nPrinting the modified set...")
```

```
print(Months)
```

Union operation

```
Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday", "Sunday"}
```

```
Days2 = {"Friday", "Saturday", "Sunday"}
```

```
print(Days1|Days2) #printing the union of the sets
```

```
Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}
```

```
Days2 = {"Friday", "Saturday", "Sunday"}
```

```
print(Days1.union(Days2)) #printing the union of the sets
```

Set operations

- Try other operations such as intersection, difference, symmetric difference at a time of practical.....

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

Method	Shortcut	Description
<u>isdisjoint()</u>		Returns whether two sets have a intersection or not
<u>issubset()</u>	<=	Returns whether another set contains this set or not
	<	Returns whether all items in this set is present in other, specified set(s)
<u>issuperset()</u>	>=	Returns whether this set contains another set or not
	>	Returns whether all items in other, specified set(s) is present in this set
<u>pop()</u>		Removes an element from the set
<u>remove()</u>		Removes the specified element
<u>symmetric_difference()</u>	^	Returns a set with the symmetric differences of two sets

Method	Shortcut	Description
<u>symmetric_difference_update()</u>	$\wedge=$	Inserts the symmetric differences from this set and another
<u>union()</u>	$ $	Return a set containing the union of sets
<u>update()</u>	$ =$	Update the set with the union of this set and others

Frozen Set

Frozen Set

- In Python, a frozen set is an **immutable version** of the built-in set data type.
- It is similar to a set, but its **contents cannot be changed** once a frozen set is created.
- Frozen set objects are **unordered collections of unique elements**, just like sets.
- They can be used the same way as sets, except they cannot be modified.
- One of the main advantages of using frozen set objects is that they are hashable, meaning they can be used as keys in dictionaries or as elements of other sets.
- Frozen set objects support many of the assets of the same operation, such as union, intersection, Difference, and symmetric Difference.

Creating frozen Set

```
fs = frozenset([1,2,3,4,5])
```

```
print(type(fs))
```

```
print("\nprinting the content of frozen set...")
```

```
for i in fs:
```

```
    print(i);
```

Output -

```
<class 'frozenset'>
```

```
printing the content of frozen set...
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

Set Functions

- Try other set functions at a time of practical.....

Dictionary

Dictionary

- Dictionaries are a useful data structure for storing data in Python because they are capable of imitating real-world data arrangements where a **certain value exists for a given key**.
- The data is stored as **key-value pairs** using a Python dictionary.
- This data structure is **mutable**
- The components of dictionary were made using keys and values.
- Keys must only have **one component**.
- Values can be of **any type**, including integer, list, and tuple.
- Dictionary entries are **ordered** as per latest versions

Creating the Dictionary

```
Employee = {"Name": "Johnny", "Age": 32, "salary":26000, "Company":"TCS"}
```

```
print(type(Employee))
```

```
print("printing Employee data .... ")
```

```
print(Employee)
```

Output –

```
<class 'dict'>
```

```
printing Employee data ....
```

```
{'Name': 'Johnny', 'Age': 32, 'salary': 26000, 'Company': TCS}
```

Creating the Dictionary – dict()

```
Employee = dict({"Name": "Johnny", "Age": 32, "salary":26000, "Company":"TCS"})  
print(type(Employee))  
print("printing Employee data .... ")  
print(Employee)
```

Output –

```
<class 'dict'>  
printing Employee data ....  
{'Name': 'Johnny', 'Age': 32, 'salary': 26000, 'Company': 'TCS'}
```

Accessing the dictionary values

```
Employee = {"Name": "Dev", "Age": 20, "salary":45000,"Company":"WIPRO"}  
print(type(Employee))  
print("printing Employee data .... ")  
print("Name : " Employee["Name"])  
print("Age : " Employee["Age"])  
print("Salary : " Employee["salary"])  
print("Company : " Employee["Company"])
```

Dictionary methods

Method	Description
<u>clear()</u>	Removes all the elements from the dictionary
<u>copy()</u>	Returns a copy of the dictionary
<u>fromkeys()</u>	Returns a dictionary with the specified keys and value
<u>get()</u>	Returns the value of the specified key
<u>items()</u>	Returns a list containing a tuple for each key value pair
<u>keys()</u>	Returns a list containing the dictionary's keys
<u>pop()</u>	Removes the element with the specified key
<u>popitem()</u>	Removes the last inserted key-value pair
<u>setdefault()</u>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<u>update()</u>	Updates the dictionary with the specified key-value pairs
<u>values()</u>	Returns a list of all the values in the dictionary

Dictionary functions

- Try other functions at a time of practical.....

Functions

Functions

- A collection of related assertions that carry out a mathematical, analytical, or evaluative operation is known as a function.
- An assortment of proclamations called Python Capabilities returns the specific errand.
- Python functions are necessary for intermediate-level programming and are easy to define.
- Function names meet the same standards as variable names do.
- The objective is to define a function and group-specific frequently performed actions.
- Instead of repeatedly creating the same code block for various input variables, we can call the function and reuse the code it contains with different variables.

Syntax

```
def function_name( parameters ):  
    # code block
```

e.g.

```
def input( n1, n2):  
    print("number 1 is: ", n1)  
    print("number 2 is: ", n2)
```

```
print( "Call to function" )  
input(50,30)
```

Default Arguments in function

e.g.

```
# Python code to demonstrate the use of default arguments
# defining a function
def function( n1, n2 = 20 ):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)

# Calling the function and passing only one argument
print( "Passing only one argument" )
function(40) //for second argument takes default arg value

# Now giving two arguments to the function
print( "Passing two arguments" )
function(50,30) //override the default value
```

Lambda Functions

Lambda Functions

- Lambda Functions in Python are **anonymous functions**, implying they **don't have a name**.
- A lambda function **can take n number of arguments** at a time.
- **But it returns only one argument** at a time.
- **lambda keyword** is used.
- **lambda expressions appear to be one-line representations of functions**
- **Syntax –**

lambda arguments: expression

Example

```
add = lambda num: num + 4  
  
print( add(6) )
```

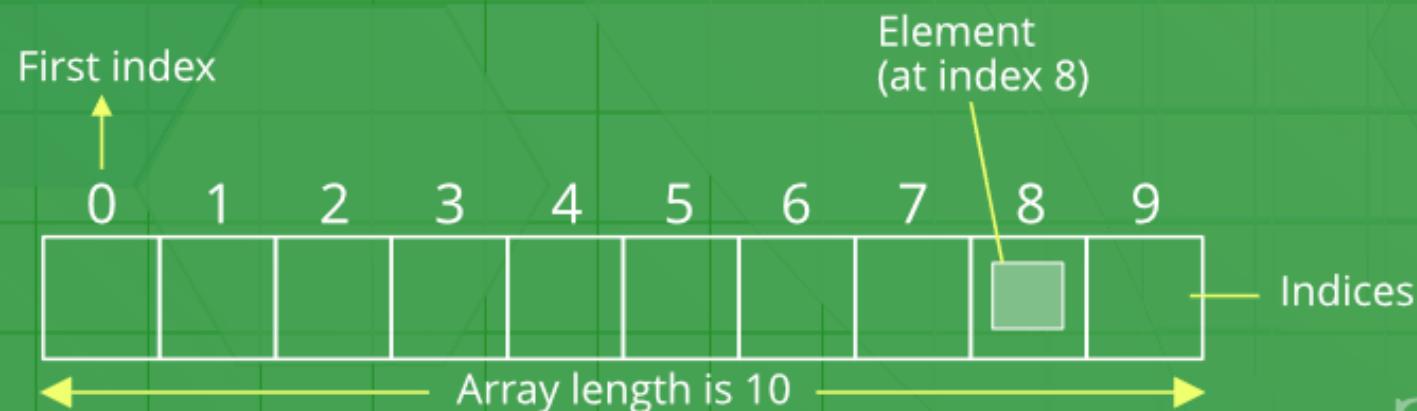
e.g. normal function

```
def add( num ):  
  
    return num + 4  
  
print( add(6) )
```

Array

Array

- An array is a collection of items stored at contiguous memory locations.
- The idea is to store multiple items of the same type together.



Creating the array

- Array in Python can be created by importing an **array module** as –

import array as arr

- To create the arry the following function is used as -

array(data_type, value_list)

where,

data type is the data type of array values

and

value list is the actual element of the array

Data Type Constants

Type Code	C Type	Python Type	Minimum Size In Bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

Creating the array

```
import array as arr
```

```
a = arr.array('i', [1, 2, 3]) //integer array
```

```
print("The new created array is : ", end=" ")
```

```
for i in range(0, 3):
```

```
    print(a[i], end=" ")
```

```
b = arr.array('d', [2.5, 3.2, 3.3]) //double array
```

```
print("\nThe new created array is : ", end=" ")
```

```
for i in range(0, 3):
```

```
    print(b[i], end=" ")
```

Inserting element in the array

```
import array as arr

a = arr.array('i', [1, 2, 3])      //integer array

print("The new created array is : ", end=" ")

for i in range(0, 3):

    print(a[i], end=" ")

a.insert(1, 4)                  //inserting element 4 at the index 1

print("Array after insertion : ", end=" ")

for i in (a):

    print(i, end=" ")
```

Array operations

- Try other functions such as removing elements, slicing array, searching element, counting, reversing, etc. at a time of practical.....