

## **PYTHON PROGRAMMING SYLLABUS(KNC-302)**

### **At the end of course, the student will be able to understand**

CO1 To read and write simple Python programs. (K1, K2)

CO2 To develop Python programs with conditionals and loops. (K2, K4)

CO3 To define Python functions and to use Python data structures lists, tuples, dictionaries (K3)

CO4 To do input/output with files in Python (K2)

CO5 To do searching, sorting and merging in Python (K2, K4)

## **DETAILED SYLLABUS**

### **UNIT I**

Introduction: The Programming Cycle for Python, Python IDE, Interacting with Python Programs, Elements of Python, Type Conversion. Basics: Expressions, Assignment Statement, Arithmetic Operators, Operator Precedence, Boolean Expression.

### **UNIT II**

Conditionals: Conditional statement in Python (if-else statement, its working and execution), Nested-if statement and Elif statement in Python, Expression Evaluation & Float Representation. Loops: Purpose and working of loops, while loop including its working, For Loop, Nested Loops, Break and Continue.

### **UNIT III**

Function: Parts of A Function , Execution of A Function , Keyword and Default Arguments ,Scope Rules.

Strings : Length of the string and perform Concatenation and Repeat operations in it. Indexing and Slicing of Strings.

Python Data Structure: Tuples, Unpacking Sequences, Lists, Mutable Sequences, List Comprehension, Sets, Dictionaries

Higher Order Functions: Treat functions as first-class Objects, Lambda Expressions

## UNIT IV

Sieve of Eratosthenes: generate prime numbers with the help of an algorithm given by the Greek Mathematician named Eratosthenes, whose algorithm is known as Sieve of Eratosthenes. File I/O: File input and output operations in Python Programming Exceptions and Assertions.

Modules: Introduction, Importing Modules,

Abstract Data Types : Abstract data types and ADT interface in Python Programming.

Classes : Class definition and other operations in the classes , Special Methods ( such as `_init_`,`_str_`, comparison methods and Arithmetic methods etc.) , Class Example , Inheritance , Inheritance and OOP.

## UNIT V

Iterators & Recursion: Recursive Fibonacci, Tower of Hanoi

Search: Simple Search and Estimating Search Time, Binary Search and Estimating Binary Search Time

Sorting & Merging: Selection Sort, Merge List, Merge Sort, Higher Order Sort

### Text books:

1. Allen B. Downey, ``Think Python: How to Think Like a Computer Scientist``, 2nd edition, Updated for Python 3,Shroff/O'Reilly Publishers, 2016 (<http://greenteapress.com/wp/thinkpython/>)

2. Guido van Rossum and Fred L. Drake Jr, —An Introduction to Python – Revised and updated for Python 3.2, Network Theory Ltd., 2011.

- 3.John V Guttag, —Introduction to Computation and Programming Using Python“, Revised and expanded Edition, MIT Press , 2013
- 4.Robert Sedgewick, Kevin Wayne, Robert Dondero, —Introduction to Programming in Python: An Inter-disciplinary Approach, Pearson India Education Services Pvt. Ltd., 2016.
- 5.Timothy A. Budd, —Exploring Python\|, Mc-Graw Hill Education (India) Private Ltd.,, 2015.
- 6.Kenneth A. Lambert, —Fundamentals of Python: First Programs\|, CENGAGE Learning, 2012.
- 7.Charles Dierbach, —Introduction to Computer Science using Python: A Computational ProblemSolving Focus, Wiley India Edition, 2013.
- 8.Paul Gries, Jennifer Campbell and Jason Montojo, —Practical Programming: An Introduction to Computer Science using Python 3\|, Second edition, Pragmatic Programmers, LLC, 2013. Mapped With : <https://ict.iitk.ac.in/product/python-programming-a-practical-approach/>



# Python Programming

## Unit 1

### (KNC-302)

Prepared By  
Abhishek Kesharwani  
Assistant Professor, UCER Naini, Allahabad

# **UNIT I**

- Introduction of Python
- The Programming Cycle for Python
- Python IDE,
- Interacting with Python Programs,
- Elements of Python, Type Conversion
- Basics: Expressions,
- Assignment Statement,
- Arithmetic Operators
- Operator Precedence,
- Boolean Expression.

# Python Introduction

- **Python** is a general purpose, dynamic, high level, and interpreted programming language. It supports Object Oriented programming approach to develop applications.
- It is simple and easy to learn and provides lots of high-level data structures.
- Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.
- Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.
- Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.

- Python is not intended to work in a particular area, such as web programming. That is why it is known as *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.

# Python Features

## 1) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

## 2) Expressive Language

Python language is more expressive means that it is more understandable and readable.

## 3) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time.

## 4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

## 5) Free and Open Source

Python language is freely available at official web address. The source-code is also available. Therefore it is open source.

## 6) Object-Oriented Language

Python supports object oriented language and concepts of classes and objects come into existence.

## 7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

## 8) Large Standard Library

Python has a large and broad library and provides rich set of module and functions for rapid application development.

## 9) GUI Programming Support

Graphical user interfaces can be developed using Python.

## 10) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

# Python Applications

## 1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing. It also provides Frameworks such as Django, Pyramid, Flask etc to design web based applications

## 2) Desktop GUI Applications

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

## 3) Software Development

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc

## 4) Scientific and Numeric

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

## 5) Business Applications

Python is used to build Bussiness applications like ERP and e-commerce systems. Tryton is a high level application platform.

## 6) Console Based Application

We can use Python to develop console based applications. For example: IPython.

## 7) Audio or Video based Applications

Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

## 8) 3D CAD Applications

To create CAD application Fandango is a real application which provides full features of CAD.

## 9) Enterprise Applications

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

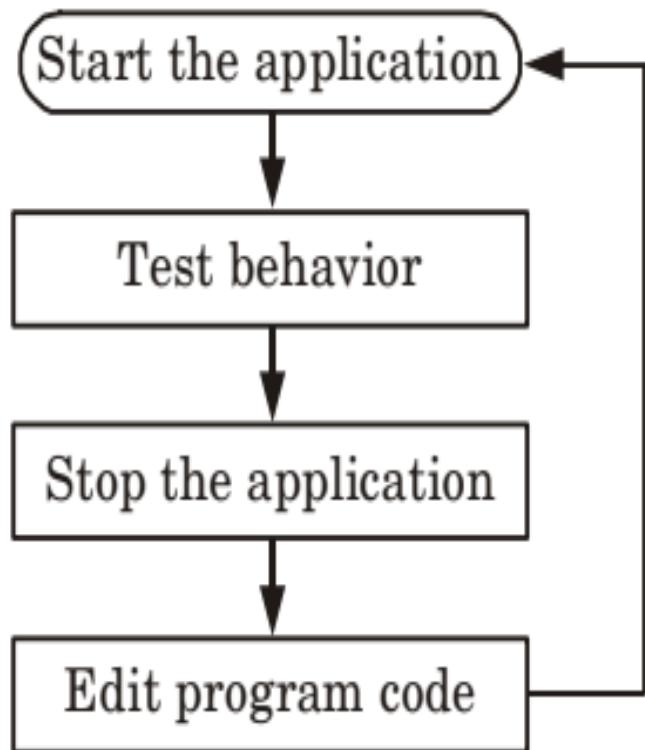
## 10) Applications for Images

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

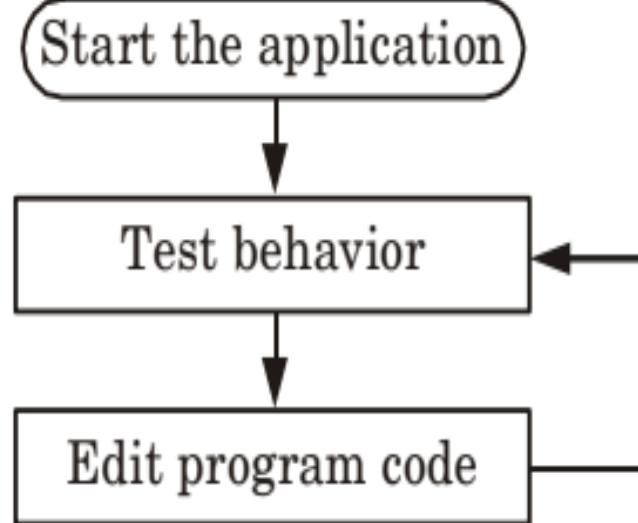
# Programming cycle for Python

- Python's programming cycle is dramatically shorter than that of traditional programming cycle.
- In Python, there are no compile or link steps.
- Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.
- In cases where dynamic module reloading can be used, it is even possible

- To change and reload parts of a running program without stopping it at all.
- Python's impact on the programming cycle is as follows:



(a) Python's programming cycle



(b) Python's programming cycle  
with module reloading

- Since Python is interpreted, there is a rapid turnaround after program changes. And because Python's parser is embedded in Python-based systems, it is easy to modify programs at runtime.

# Integrated Development Environment

1. IDE is a software package that consists of several tools for developing and testing the software.
2. An IDE helps the developer by automating the process.
3. IDEs integrate many tools that are designed for SDLC.
4. IDEs were introduced to diminish the coding and typing errors.
5. Some of the Python IDEs are pycharm,

# PyCharm

PyCharm assists the developers to be more productive and provides smart suggestions. It saves time by taking care of routine tasks, hence increases productivity.

## Features of PyCharm

- i. It has smart code navigation, good code editor, a function for quick refactoring.
- ii. The integrated activities with PyCharm are profiling, testing, debugging, remote development, and deployments.
- iii. PyCharm supports Python web development frameworks, Angular JS, JavaScript, CSS, HTML and live editing functions.

# Spyder

Spyder is widely used for data science works. It is mostly used to create a secure and scientific environment for Python. Spyder Python uses PyQt (Python plug-in) which a developer can add as an extension.

## Features of Spyder :

- i. It has good syntax highlighting and auto code completion features.
- ii. Spyder Python explores and edits variables directly from GUI.
- iii. It performs very well in multi-language editor.

# PyDev

It is an external plug-in for Eclipse and is very popular as Python interpreter.

## Features of PyDev :

- i. PyDev has strong parameters like refactoring, debugging, type hinting, code analysis, and code coverage function.
- ii. PyDev supports tokens browser, PyLint integration, interactive console, remote debugger, Unittest integration, etc.

# IDLE

IDLE is a basic IDE mainly used by beginner level developer.

- i. IDLE Python is a cross-platform IDE, hence it increases the flexibility for users.
- ii. It is developed only in Python in collaboration with Tkinter GUI toolkit.
- iii. The feature of multi-window text editor in IDLE has some great functions like smart indentation, call tips, Python colorizing, and undo option.

# Python Installation on Ubuntu

## 1) Update the APT Repository

```
$ apt-get update
```

## 2) Install Python

```
$ apt-get install python3.6
```

## 3) Verify Python

When we type **python** it shows default installed python that is 2.7.

## 4) Check python –version.

Type **python - - version**. It will show the current version of python.

# Python Variables

- Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.
- Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

# Identifier Naming

- Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.
- The first character of the variable must be an alphabet or underscore ( \_ ).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, \*).

- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive for example my name, and MyName is not the same.

Examples of valid identifiers :

a123, \_n, n\_9, etc.

Examples of invalid identifiers:

1a, n%4, n 9, etc.

# Declaring Variable and Assigning Values

- Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.
- We don't need to declare explicitly variable in Python.
- When we assign any value to the variable that variable is declared automatically.

# Multiple Assignment

- Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

## 1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
```

```
print x
```

```
print y
```

```
print z
```

## **2.Assigning multiple values to multiple variables**

**a,b,c=5,10,15**

**print a**

**print b**

**print c**

# Python Data Types

- Variables can hold values of different data types.
- Python is a dynamically typed language hence we need not define the type of the variable while declaring it.
- The interpreter implicitly binds the value with its type.
- Python enables us to check the type of the variable used in the program.
- Python provides us the **type()** function which returns the type of the variable passed.

A=10

b="Hi Python"

c = 10.5

**print(type(a));**

**print(type(b));**

**print(type(c));**

## **Output:**

<type 'int'>

<type 'str'>

<type 'float'>

# Standard data types

A variable can hold different types of values

Python provides various standard data types that define the storage method on each of them.

- Numbers
- String
- List
- Tuple
- Dictionary

# Numbers

- Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;
- `a = 3 , b = 5 #a and b are number objects`

# Python supports 3 types of numeric data.

- int (signed integers like 10, 2, 29, etc.)
- float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
- complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

# String

- The string can be defined as the sequence of characters represented in the quotation marks.
- In python, we can use single, double, or triple quotes to define a string.
- In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+ " python" returns "hello python".
- The operator \* is known as repetition operator as the operation "Python " \*2 returns "Python Python ".

The following example illustrates the string handling in python.

```
str1 = 'hello Abhishek' #string str1
```

```
str2 = ' how are you' #string str2
```

```
print (str1[0:2]) #printing first two character using slice operator
```

```
print (str1[4]) #printing 4th character of the string
```

```
print (str1*2) #printing the string twice
```

```
print (str1 + str2) #printing the concatenation of str1 and str2
```

# List

- Lists are similar to arrays in C.
- However; the list can contain data of different types.
- The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
- We can use slice [:] operators to access the data of the list.
- The concatenation operator (+) and repetition operator (\*) works with the list in the same way as they were working with the strings.

# Example

```
I = [1, "hi", "python", 2]
```

```
print (I[3:])
```

```
print (I[0:2])
```

```
print (I)
```

```
print (I + I)
```

```
print (I * 3)
```

# Output

- [2]
- [1, 'hi']
- [1, 'hi', 'python', 2]
- [1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
- [1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]

# Tuple

A tuple is similar to the list in many ways.

- Like lists, tuples also contain the collection of the items of different data types.
- The items of the tuple are separated with a comma (,) and enclosed in parentheses ().
- A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

# Example

```
t = ("hi", "python", 2)
```

```
print (t[1:])
```

```
print (t[0:1])
```

```
print (t)
```

```
print (t + t)
```

```
print (t * 3)
```

```
print (type(t))
```

```
t[2] = "hi"
```

# output

- ('python', 2)
- ('hi',)
- ('hi', 'python', 2)
- ('hi', 'python', 2, 'hi', 'python', 2)
- ('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
- <type 'tuple'>
- TypeError: 'tuple' object does not support item assignment

# Dictionary

- Dictionary is an ordered set of a key-value pair of items.
- It is like an associative array or a hash table where each key stores a specific value.
- Key can hold any primitive data type whereas value is an arbitrary Python object.
- The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

# Example

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};  
print("1st name is "+d[1]);  
print("4th name is "+ d[4]);  
print (d);  
print (d.keys());  
print (d.values());
```

# Output

- 1st name is Jimmy
- 4th name is mike
- {1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
- [1, 2, 3, 4]
- ['Jimmy', 'Alex', 'john', 'mike']

# Python Keywords

True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

# Python Literals

Literals can be defined as a data that is given in a variable or constant.

Python support the following literals:

**I. String literals**

**II. Numeric literals**

**III. Boolean literals**

**IV. Special literals.**

# I. String literals

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

**Eg:** "Aman", '12345'

## Types of Strings:

There are two types of Strings supported in Python:

a).**Single line String-** Strings that are terminated within a single line are known as Single line Strings.

**Eg:** text1='hello'

b).**Multi line String-** A piece of text that is spread along multiple lines is known as Multiple line String.

# There are two ways to create Multiline Strings:

1). Adding black slash at the end of each line.

Eg:

```
>>> text1='hello\
user'
>>> text1
'hellouser'
>>>
```

2).Using triple quotation marks:-

Eg:

```
>>> str2="""welcome
to
SSSIT"""
>>> print str2
welcome
to
SSSIT
>>>
```

## **II. Numeric literals:**

### **Int(signed integers)**

Numbers( can be both positive and negative)  
with no fractional part.eg: 100

### **float(floating point)**

Real numbers with both integer and fractional  
part eg: -26.2

### **Complex(complex)**

In the form of  $a+bj$  where a forms the real part  
and b forms the imaginary part of complex  
number. eg: 3.14j

### **III. Boolean literals**

A Boolean literal can have any of the two values:  
True or False.

### **IV. Special literals**

Python contains one special literal i.e., None.  
None is used to specify to that field that is not  
created. It is also used for end of lists in  
Python.

### **V. Literal Collections.**

Collections such as tuples, lists and Dictionary  
are used in Python.

# Python Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands.

- Arithmetic operators
- Relational operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

# Arithmetic operators

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	$x + y$
-	Subtraction: subtracts two operands	$x - y$
*	Multiplication: multiplies two operands	$x * y$
/	Division (float): divides the first operand by the second	$x / y$
//	Division (floor): divides the first operand by the second	$x // y$
%	Modulus: returns the remainder when first operand is divided by the second	$x \% y$

## # Examples of Arithmetic Operator

```
a = 9  
b = 4  
add = a + b  
sub = a - b  
mul = a * b  
div1 = a / b  
div2 = a // b  
mod = a % b  
# print results  
print(add)  
print(sub)  
print(mul)  
print(div1)  
print(div2)  
print(mod)
```

# Output

- 13
- 5
- 36
- 2.25
- 2
- 1

# Relational Operators

OPERATOR	DESCRIPTION	SYNTAX
>	Greater than: True if left operand is greater than the right	$x > y$
<	Less than: True if left operand is less than the right	$x < y$
==	Equal to: True if both operands are equal	$x == y$
!=	Not equal to - True if operands are not equal	$x != y$
>=	Greater than or equal to: True if left operand is greater than or equal to the right	$x >= y$
<=	Less than or equal to: True if left operand is less than or equal to the right	$x <= y$

## # Examples of Relational Operators

```
a = 13
```

```
b = 33
```

```
# a > b is False
```

```
print(a > b)
```

```
# a < b is True
```

```
print(a < b)
```

```
# a == b is False
```

```
print(a == b)
```

```
# a != b is True
```

```
print(a != b)
```

```
# a >= b is False
```

```
print(a >= b)
```

```
# a <= b is True
```

```
print(a <= b)
```

# Logical operators

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

## # Examples of Logical Operator

a = True

b = False

# Print a and b is False

print(a and b)

# Print a or b is True

print(a or b)

# Print not a is False

print(not a)

# Bitwise operators

OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	$x \& y$
	Bitwise OR	$x   y$
~	Bitwise NOT	$\sim x$
^	Bitwise XOR	$x ^ y$
>>	Bitwise right shift	$x >>$
<<	Bitwise left shift	$x <<$

```
a = 60      # 60 = 0011 1100
b = 13      # 13 = 0000 1101
c = 0
c = a & b;    # 12 = 0000 1100
print ("Line 1 - Value of c is ", c)
```

```
c = a | b;    # 61 = 0011 1101
print ("Line 2 - Value of c is ", c)
```

```
c = a ^ b;    # 49 = 0011 0001
print ("Line 3 - Value of c is ", c)
```

```
c = ~a;      # -61 = 1100 0011 -(n+1)
print ("Line 4 - Value of c is ", c)
```

```
c = a << 2;   # 240 = 1111 0000
print ("Line 5 - Value of c is ", c)
```

```
c = a >> 2;   # 15 = 0000 1111
print ("Line 6 - Value of c is ", c)
```

# Assignment operators

OPERATOR	DESCRIPTION	SYNTAX	
=	Assign value of right side of expression to left side operand	x = y + z	
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b	a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b	a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b	a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b	a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b	a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b	a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b	a=a**b

# Special operators

There are some special type of operators like

- Identity operators
- Membership operators

## Identity operators-

**is** and **is not** are the identity operators both are used to check if two values are located on the same part of the memory.

```
a1 = 3
```

```
b1 = 3
```

```
a2 = 'Hello'
```

```
b2 = 'Hello'
```

```
a3 = [1,2,3]
```

```
b3 = [1,2,3]
```

```
print(a1 is not b1) # False
```

```
print(a2 is b2) # True
```

```
print(a3 is b3) # Output is False, since lists are  
mutable.
```

# **Membership operators-**

**in** and **not in** are the membership operators used to test whether a value or variable is in a sequence.

**in** True if value is found in the sequence

**not in** True if value is not found in the sequence

```
x = 'Geeks for Geeks'
```

```
y = {3:'a',4:'b'}
```

```
print('G' in x) # True
```

```
print('geeks' not in x) # True
```

```
print('Geeks' not in x) # False
```

```
print(3 in y) # True
```

```
print('b' in y) # False
```

# Python Comments

Comments in Python can be used to explain any program code. It can also be used to hide the code as well.

## 1) Single Line Comment:

In case user wants to specify a single line comment, then comment must start with #

Eg:

```
# This is single line comment.
```

## **2) Multi Line Comment:**

Multi lined comment can be given inside triple quotes.

**eg:**

```
''' This  
    Is  
    Multipline comment'''
```

# Operator Precedence

This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

# Operator Precedence

addition operator ← lower precedence      higher precedence → multiplication operator

10 + 20 \* 30



multiply will happen first  
as \* has higher precedence

10 + 60

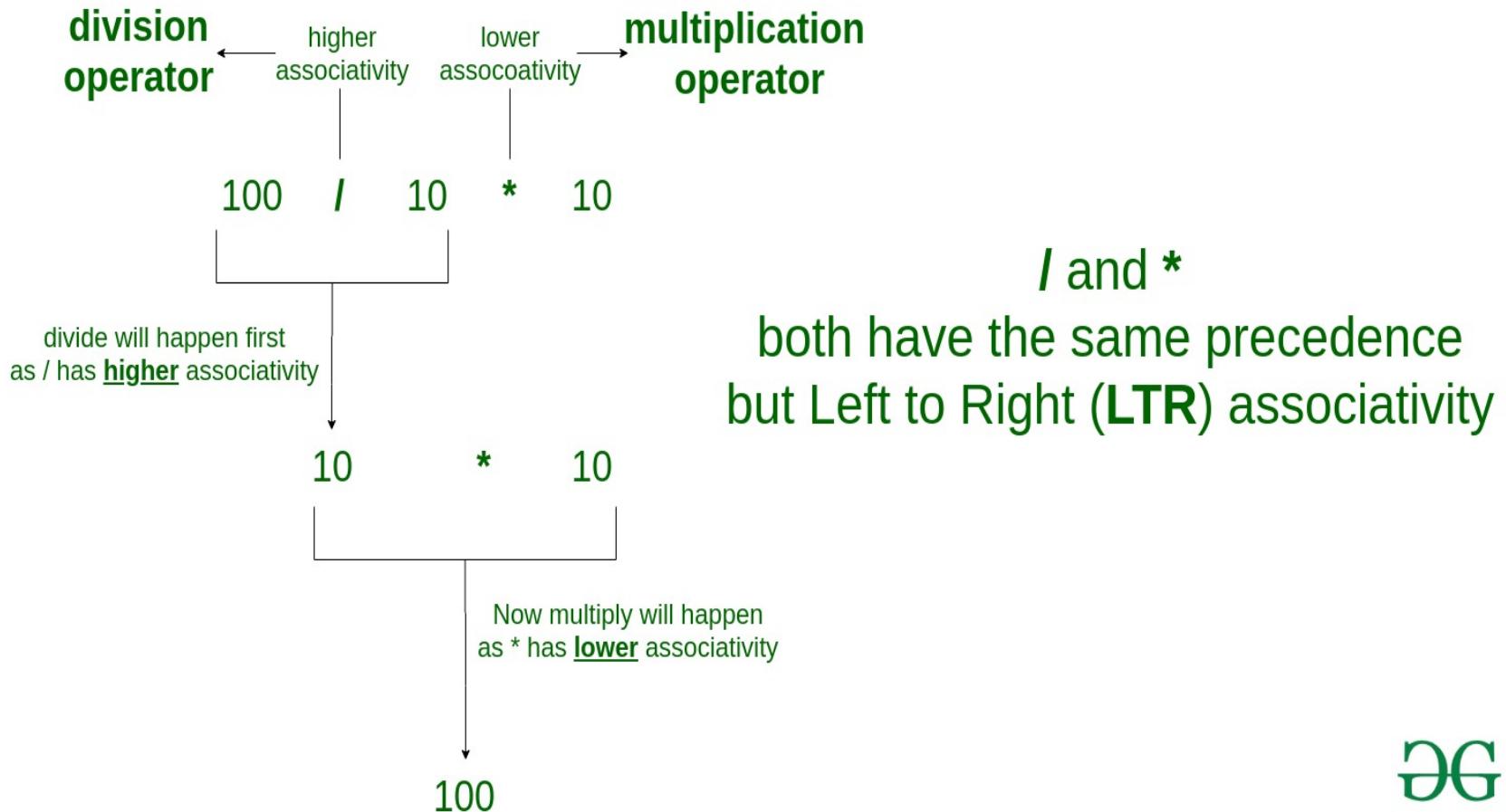


Now addition will happen  
as + has lower precedence

# Operator Associativity

If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be Left to Right or from Right to Left.

# Operator Associativity



Operators Precedence and Associativity are two main characteristics of operators that determine the evaluation order of sub-expressions in absence of brackets.

Evaluate

$$100 + 200 / 10 - 3 * 10$$

# Operator Precedence and Associativity.

1 Divide '/' will happen first  
It has **higher precedence** than + and -  
It has the same precedence as \* but **higher associativity**

3 Addition '+' will happen third  
It has **lower precedence** than / and \*  
It has the same precedence as - but **higher associativity**

4 Subtract '-' will happen last  
It has **lower precedence** than / and \*  
It has the same precedence as + but **lower associativity**

2 Multiply '\*' will happen second  
It has **higher precedence** than + and -  
It has the same precedence as / but **lower associativity**

/ and \*

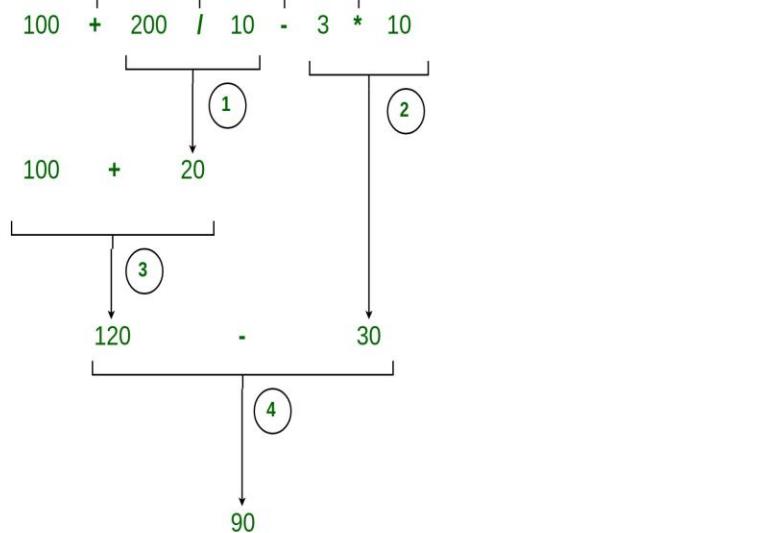
both have the same precedence but Left to Right (**LTR**) associativity

+ and -

both have the same precedence but Left to Right (**LTR**) associativity

/ and \*

have the higher precedence than + and -



Operator	Description	Associativity
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	

<code>== !=</code>	Relational is equal to/is not equal to	left-to-right
<code>is, is not in, not in</code>	Identity Membership operators	left-to-right
<code>&amp;</code>	Bitwise AND	left-to-right
<code>^</code>	Bitwise exclusive OR	left-to-right
<code> </code>	Bitwise inclusive OR	left-to-right
<code>not</code>	Logical NOT	right-to-left
<code>and</code>	Logical AND	left-to-right
<code>or</code>	Logical OR	left-to-right

	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
--	---	---------------



# Python Programming

## Unit 2

### (KNC-302)

Prepared By  
Abhishek Kesharwani  
Assistant Professor, UCER Naini, Allahabad

## **UNIT II**

- Conditionals: Conditional statement in Python
- if-else statement, its working and execution
- Nested-if statement and Elif statement in Python
- Expression Evaluation
- Float Representation
- Loops: Purpose and working of loops
- While loop including its working
- For Loop, Nested Loops,
- Break and Continue.

# Python Loops

- The flow of the programs written in any programming language is sequential by default.
- Sometimes we may need to alter the flow of the program.
- The execution of a specific code may need to be repeated several numbers of times.

# Python for loop

The **for loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

**for iterating\_var in sequence:**

**statement(s)**

# The range() Function

- To loop through a set of code a specified number of times, we can use the range() function,
- 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.

## Example

```
for x in range(6):  
    print(x)
```

- Note that `range(6)` is not the values of 0 to 6, but the values **0 to 5**.
- The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter:
- `range(2, 6)`, which means values from 2 to 6 (but not including 6):

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

## Example

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):  
    print(x)
```

i=1

n=int(input("Enter the number up to which you want to print the natural numbers?"))

**for i in range(0,10):**

**print(i,end = ' ')**

**Output:**

0 1 2 3 4 5 6 7 8 9

# printing the table of the given number

i=1

```
num = int(input("Enter a number:"))
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i))
```

# Nested for loop in python

- Python allows us to nest any number of for loops inside a for loop.
- The inner loop is executed n number of times for every iteration of the outer loop.

## Syntax

```
for iterating_var1 in sequence:  
    for iterating_var2 in sequence:  
        #block of statements
```

#Other statements

# Pattern

```
n=int(input("enter the number of rows"))

for i in range (0,n):
    for j in range (0,i+1):
        print("*",end=" ")
    print()
```

# Output

```
D:\python cs\programs>python forloop3.py
enter the number of rows5
*
*
*
*
*
*   *
* * *
* * * *
* * * * *
```

# Using else statement with for loop

- Unlike other languages like C, C++, or Java, python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted.
- Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

# Example

```
for i in range(0,5):  
    print(i)  
else:print("for loop completely exhausted, since  
there is no break.");
```

# Output

0

1

2

3

4

for loop completely exhausted, since there is no  
break.

# Example

```
for i in range(0,5):
    print(i)
    break;
else:print("for loop is exhausted");
print("The loop is broken due to break statemen
      t...came out of loop")
```

## Output:

0

The loop is broken due to break  
statement...came out of loop

# The break Statement

With the break statement we can stop the loop before it has looped through all the items:

## Example

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    print(x)
```

```
    if x == "banana":
```

```
        break
```

```
D:\python cs\programs>python forloop4.py
apple
banana
```

```
D:\python cs\programs>
```

# The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

# Output

```
C:\Users\My Name>python demo_for_continue.py
apple
cherry
```

# Print each fruit in a fruit list

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:  
    print(x)
```

# Looping Through a String

```
for x in "banana":  
    print(x)
```

# Example

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
```

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in adj:
```

```
    for y in fruits:
```

```
        print(x, y)
```

```
C:\Users\My Name>python demo_for_nested.py
```

```
red apple
```

```
red banana
```

```
red cherry
```

```
big apple
```

```
big banana
```

```
big cherry
```

```
tasty apple
```

```
tasty banana
```

```
tasty cherry
```

# Python while loop

- The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true.

**while expression:**

**statements**

# Example

i=1;

**while** i<=10:

**print(i);**

    i=i+1;

# Using else with Python while loop

- Python enables us to use the while loop with the while loop also.
- The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

# Example

Consider the following example.

```
i=1;
```

```
while i<=5:
```

```
    print(i)
```

```
    i=i+1;
```

```
else:print("The while loop exhausted");
```

# Output

1

2

3

4

5

The while loop exhausted

# Python Pass

- In Python, pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty.
- It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

# Example

```
for i in [1,2,3,4,5]:  
    if i==3:  
        pass  
        print "Pass when value is",i  
    print i,
```

# Example

```
D:\python cs\programs>python pass.py
```

```
Pass when value is 3
```

12345



# Python Programming

## Unit 3

### (KNC-302)

Prepared By  
Abhishek Kesharwani  
Assistant Professor, UCER Naini, Allahabad

# **UNIT III**

- Function: Parts of A Function
- Execution of A Function
- Keyword and Default Arguments
- Scope Rules.
- Strings
- Length of the string and perform Concatenation and Repeat operations in it.
- Indexing and Slicing of Strings.

# Python Functions

- A function is a block of code which only runs when it is called.
- You 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:

Example

```
def my_function():
    print("Hello from a function")
```

# Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():
    print("Hello from a function")
my_function()
```

# Parameters

- Information can be passed to functions as parameter.
- Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

```
def my_function(fname):  
    print(fname + " Welcome")
```

```
my_function("Abhishek")
```

# Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without parameter, it uses the default value:

Example

```
def my_function(country = "Norway"):  
    print("I am from " + country)
```

```
my_function("Sweden")  
my_function("India")  
my_function()
```

```
C:\Users\My Name>python demo_function_param2.py
```

```
I am from Sweden
```

```
I am from India
```

```
I am from Norway
```

# Passing a List as a Parameter

- You can send any data types of parameter to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

# Example

```
def my_function(food):  
    for x in food:  
        print(x)
```

```
fruits = ["apple", "banana", "cherry"]
```

```
my_function(fruits)
```

```
C:\Users\My Name>python demo_function_param3.py
```

```
apple
```

```
banana
```

```
cherry
```

# Return Values

To let a function return a value, use  
the return statement:

Example

```
def my_function(x):  
    return 5 * x
```

```
print(my_function(3))  
print(my_function(5))  
print(my_function(9))
```

# Call by reference in Python

- In python, all the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.
- However, there is an exception in the case of mutable objects since the changes made to the mutable objects like string do not revert to the original string rather, a new string object is made, and therefore the two different objects are printed.

# Mutable vs Immutable Objects in Python

- Every variable in python holds an instance of an object.
- There are two types of objects in python i.e. **Mutable** and **Immutable objects**.
- Whenever an object is instantiated, it is assigned a unique object id.
- The type of the object is defined at the runtime and it can't be changed afterwards. However, its state can be changed if it is a mutable object.

- NOTE: mutable objects can change their state or contents and immutable objects can't change their state or content.

**Immutable Objects** : These are of in-built types like **int, float, bool, string, unicode, tuple**. In simple words, an immutable object can't be changed after it is created.

**Mutable Objects** : These are of type **list , dict , set** . Custom classes are generally mutable.

## Example 1 Passing mutable Object (List)

```
def change_list(l):  
    l.append(20)  
    l.append(30)  
    print("list inside function = ",l)
```

```
list1 = [10,30,40,50]
```

```
change_list(list1)  
print("list outside function = ",list1)
```

# Output

list inside function = [10, 30, 40, 50, 20, 30]

list outside function = [10, 30, 40, 50, 20, 30]

## Example 2 Passing immutable Object (String)

```
def change_string (str):  
    str = str + " Hows you";  
    print("printing the string inside function :",str);
```

```
string1 = "Hi I am there"
```

```
change_string(string1)
```

```
print("printing the string outside function :",string1)
```

# Output

printing the string inside function : Hi I am there  
Hows you

printing the string outside function : Hi I am  
there

# Types of arguments

There may be several types of arguments which can be passed at the time of function calling.

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

# Required Arguments

- required arguments are those arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition.
- If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

# Example

```
def simple_interest(p,t,r):  
    return (p*t*r)/100  
  
p = float(input("Enter the principle amount? "))  
r = float(input("Enter the rate of interest? "))  
t = float(input("Enter the time in years? "))  
  
print("Simple Interest: ",simple_interest(p,r,t))
```

# Keyword arguments

- Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.
- The name of the arguments is treated as the keywords and matched in the function calling and definition.
- If the same match is found, the values of the arguments are copied in the function definition.

# Example

```
def simple_interest(p,t,r):  
    return (p*t*r)/100  
  
print("Simple Interest:  
    ",simple_interest(t=10,r=10,p=1900))
```

## Output:

Simple Interest: 1900.0

# Default Arguments

- Python allows us to initialize the arguments at the function definition.
- If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

# Example

```
def printme(name,age=22):  
    print("My name is",name,"and age is",age)  
printme(name = "john")
```

## Output:

My name is john and age is 22

# Variable length Arguments

- Python provides us the flexibility to provide the comma separated values which are internally treated as tuples at the function call.
- However, at the function definition, we have to define the variable with \* (star) as \*<variable - name >.

# Example

```
def printme(*names):
    print("type of passed argument is ",type(names))
    print("printing the passed arguments...")
    for name in names:
        print(name)
printme("john","David","smith","nick")
```

# Output

```
type of passed argument is <class 'tuple'>
```

```
printing the passed arguments...
```

```
john
```

```
David
```

```
smith
```

```
nick
```

# Python Built-in Functions

- The Python built-in functions are defined as the functions whose functionality is pre-defined in Python.
- The python interpreter has several functions that are always present for use. These functions are known as Built-in Functions.

# Python String

- In python, strings can be created by enclosing the character or the sequence of characters in the quotes.
- Python allows us to use single quotes, double quotes, or triple quotes to create the string.

**str = "Hi Python !"**

Here, if we check the type of the variable str  
using a python script

**print(type(str)),**

**then it will print string (str).**

python doesn't support the character data type  
instead a single character written as 'p' is  
treated as the string of length 1.

# Strings indexing and splitting

**str = "HELLO"**



**str[0] = 'H'**

**str[1] = 'E'**

**str[2] = 'L'**

**str[3] = 'L'**

**str[4] = 'O'**

**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'**

# Reassigning strings

- Updating the content of the strings is as easy as assigning it to a new string.
- The string object doesn't support item assignment i.e., A string can only be replaced with a new string since its content can not be partially replaced.
- Strings are immutable in python.

## Example 1

```
str = "HELLO"
```

```
str[0] = "h"
```

```
print(str)
```

## Output:

```
TypeError: 'str' object does not support item  
assignment
```

## Example 2

```
str = "HELLO"
```

```
print(str)
```

```
str = "hello"
```

```
print(str)
```

## Output:

HELLO

hello

# String Operators

## Operator    Description

+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.

in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

# Raw String

- `s = 'Hi\nHello'`
- `print(s)`

**output**

- Hi
- Hello

- `raw_s = r'Hi\nHello'`
- `print(raw_s)`

**Output:** Hi\nHello

# Built-in String functions

- Python **capitalize()** method converts first character of the string into uppercase without altering the whole string.
- It changes the first character only and skips rest of the string unchanged.

# Python Formatting operator

- Python allows us to use the format specifiers used in C's printf statement.
- The format specifiers in python are treated in the same way as they are treated in C.

# Example

Integer = 10;

Float = 1.290

String = "Ayush"

```
print("Hi I am Integer ... My value is %d\nHi I am  
float ... My value is %f\nHi I am string ... My v  
alue is %s"%(Integer,Float,String));
```

## Output:

Hi I am Integer ... My value is 10

Hi I am float ... My value is 1.290000

Hi I am string ... My value is Ayush

# Python String Count() Method

- It returns the number of occurrences of substring in the specified range.
- It takes three parameters, first is a substring, second a start index and third is last index of the range.
- Start and end both are optional whereas substring is required.

# Example

```
str = "Hello Javatpoint"  
str2 = str.count('t')  
# Displaying result  
print("occurences:", str2)
```

## Output:

occurences: 2

# Example

```
str = "ab bc ca de ed ad da ab bc ca"  
oc = str.count('a', 3)  
# Displaying result  
print("occurences:", oc)
```

## Output:

occurences: 5

# Example

```
str = "ab bc ca de ed ad da ab bc ca 12 23 35 62"
```

```
oc = str.count('2')
```

```
# Displaying result
```

```
print("occurences:", oc)
```

## Output:

```
occurences: 3
```

# Python String endswith() Method

Python `endswith()` method returns true if the string ends with the specified substring, otherwise returns false.

## Signature

`endswith(suffix[, start[, end]])`

## Return Type

It returns a boolean value either True or False.

# Example

```
str = "Hello this is United."
```

```
isends = str.endswith(".")
```

```
print(isends)
```

**Output:**

True

A simple example which returns true because it ends with dot (.).

# Example

```
str = "Hello this is javatpoint."  
isends = str.endswith("is",10)  
# Displaying result  
print(isends)
```

## Output:

False

Note: Here, we are providing start index of the range from where method starts searching.

# Example

```
str = "Hello this is javatpoint."  
isends = str.endswith("is",0,13)  
# Displaying result  
print(isends)
```

## Output:

True

**Note:** It returns true because third parameter stopped the method at index 13

# Python String expandtabs() Method

- Python **expandtabs()** method replaces all the characters by sepecified spaces.
- By default a single tab expands to 8 spaces which can be overridden according to the requirement.
- We can pass 1, 2, 4 and more to the method which will replace tab by the these number of space characters.

## Signature

`expandtabs(tabsize=8)`

**tabsize** : It is optional and default value is 8.

# Example

```
str = "Welcome \t to \t the \t Javatpoint."
```

```
str2 = str.expandtabs(1)
```

```
str3 = str.expandtabs(2)
```

```
str4 = str.expandtabs(4)
```

```
print(str2)
```

```
print(str3)
```

```
print(str4)
```

## **Output:**

```
Welcome to the Javatpoint.
```

```
Welcome to the Javatpoint.
```

```
Welcome to the Javatpoint.
```

# Python String find() Method

Python **find()** method finds substring in the whole string and returns index of the first match. It returns -1 if substring does not match.

## Signature

```
find(sub[, start[, end]])
```

## Parameters

**sub** : substring

**start** : start index a range

**end** : last index of the range

## Return Type

If found it returns index of the substring, otherwise -1.

# Example

```
str = "Welcome to the Javatpoint."
```

```
str2 = str.find("the")
```

```
print(str2)
```

**Output:**

11

# Python String `format()` Method

- Python **format()** method is used to perform format operations on string.
- While formatting string a delimiter {} (braces) is used to replace it with the value.
- This delimiter either can contain index or positional argument.

# Example

```
str = "Java"
```

```
str2 = "C#"
```

```
str3 = "{} and {} both are programming language
```

```
    s".format(str,str2)
```

```
print(str3)
```

## Output:

Java and C# both are programming languages

# Example

```
val = 10  
print("decimal: {0:d}".format(val));  
# display decimal result  
print("hex: {0:x}".format(val));    # display hexadecimal result  
print("octal: {0:o}".format(val));  # display octal result  
print("binary: {0:b}".format(val)); # display binary result
```

## Output:

decimal: 10

hex: a

octal: 12

binary: 1010

# Example

```
val = 100000000
print("decimal: {:.}{}".format(val)); # formatting float value
print("decimal: {:.%}{}".format(56/9));
print("decimal: {:.3%}{}".format(56/9));
```

## Output

decimal: 100,000,000

decimal: 622.222222%

decimal: 622.222%

# Python String isalnum() Method

Python **isalnum()** method checks whether the all characters of the string is alphanumeric or not. A character which is either a letter or a number is known as alphanumeric. It does not allow special chars even spaces.

## Signature

- `isalnum()`

## Parameters

- No parameter is required.

## Return

- It returns either True or False.

# Example

```
str = "Welcome123"  
str3 = "Welcome 123"  
str2 = str.isalnum()  
str4 = str3.isalnum()  
print(str2)  
print(str4)
```

## Output:

True

False

# Example

```
str = "123"
```

```
str3 = "2.50"
```

```
str2 = str.isdecimal()
```

```
str4 = str3.isdecimal()
```

```
print(str2)
```

```
print(str4)
```

**Output:**

True

False

# Python String **islower()** Method

Python string **islower()** method returns True if all characters in the string are in lowercase. It returns False if not in lowercase.

## Signature

- `islower()`

## Parameters

- No parameter is required.

## Return

- It returns either True or False.

# Example

```
str = "United College"  
str2 = str.islower()  
print(str2)
```

**Output:**

False

# Example

```
str = "hi, my contact is 876562...."
```

```
str2 = str.islower()
```

```
print(str2)
```

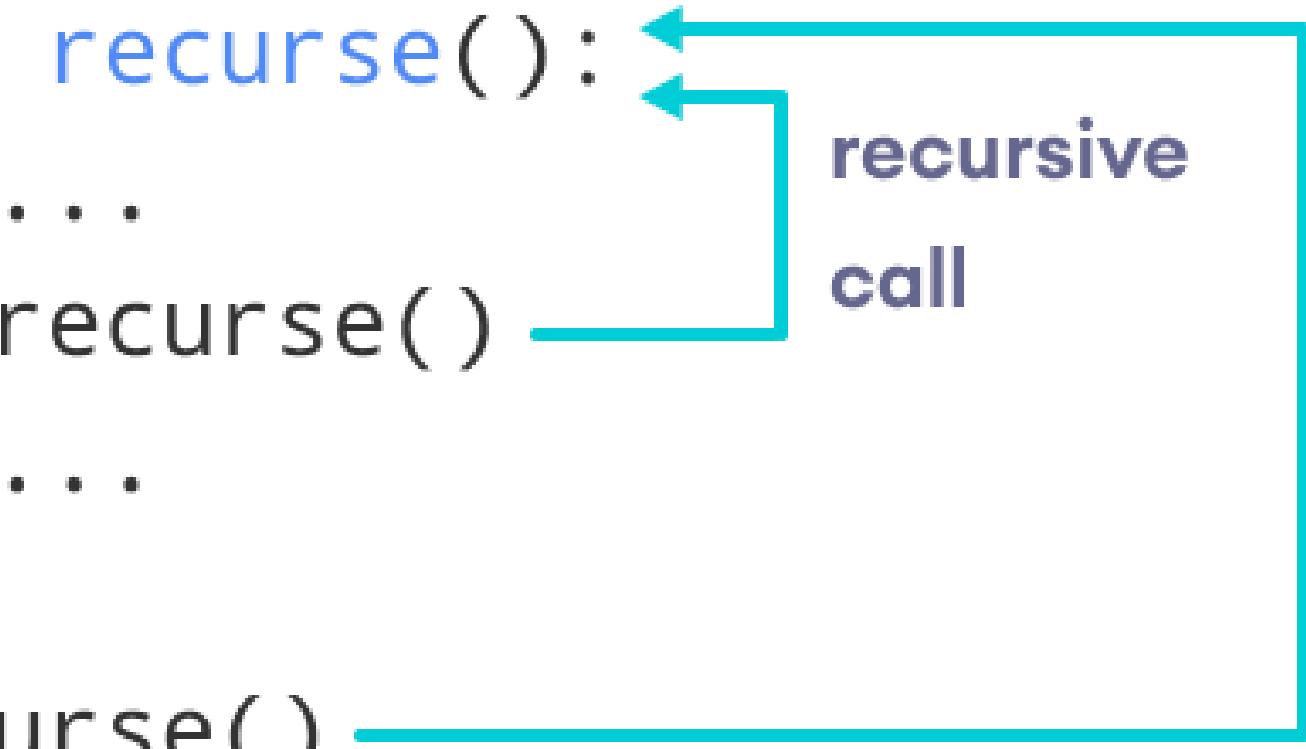
output

True

# Recursive Function

- In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

```
def recurse():  
    ...  
    recurse()  
    ...  
  
re recurse()
```



recursive  
call

# Example of a recursive function

```
def factorial(x):
    if x == 1 or x==0:
        return 1
    else:
        return (x * factorial(x-1))
```

```
num = 3
print("The factorial of", num, "is", factorial(num))
```

## Output

The factorial of 3 is 6

# Advantages of Recursion

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

# Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

# Fibonacci series

In mathematical terms, the sequence  $F_n$  of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with seed values  $F_0 = 0$  and  $F_1 = 1$ .

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, .....

# Fibonacci Number in Python

```
def Fibonacci(n):
    if n < 0:
        print("Incorrect input")
        # First Fibonacci number is 0
    elif n == 1:
        return 0
    # Second Fibonacci number is 1
    elif n == 2:
        return 1
    else:
        return Fibonacci(n - 1) + Fibonacci(n - 2)

print(Fibonacci(int(input("Enter the
number")))))
```



# Python Programming

## Unit 3

### (KNC-302)

Prepared By  
Abhishek Kesharwani  
Assistant Professor, UCER Naini, Allahabad

# **UNIT III**

- Python Data Structure:
- Tuples,
- Unpacking Sequences,
- Lists
- Mutable Sequences,
- List Comprehension,
- Sets
- Dictionaries
- Higher Order Functions:
- Functions as first-class Objects,
- Lambda Expressions

## What is List

- List is ordered and allows duplicate entries as well.
- Lists in python need not be homogeneous, which means it can contain different data types like integers, strings and other collection data types.
- It is mutable in nature and allows indexing to access the members in a list.
- The items in the list are separated with the comma (,) and enclosed with the square brackets [].

**A list can be defined as follows.**

```
L1 = ["John", 102, "USA"]
```

```
L2 = [1, 2, 3, 4, 5, 6]
```

```
L3 = [1, "Ryan"]
```

# Why it is important

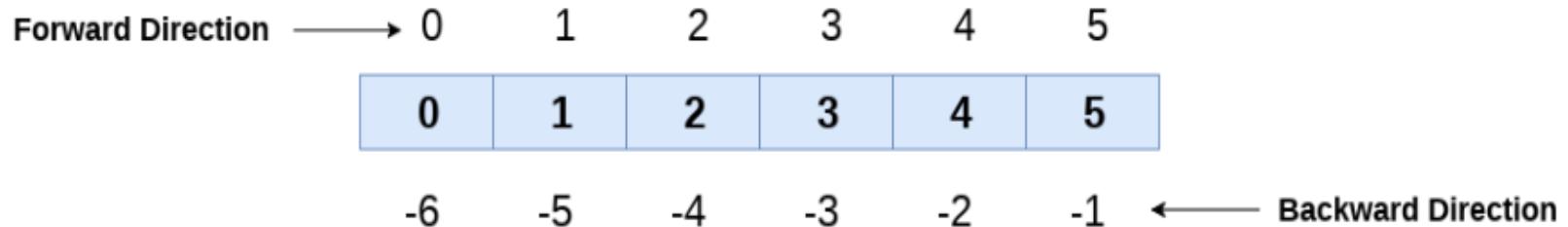
- Lists are small but essential element in data science.
- No need to worry about size and type of the list.
- List can be used to solve complex problems with reduced lines of codes.

# List indexing and splitting

- The elements of the list can be accessed by using the slice operator [ : ].
- The index starts from 0 and goes to length - 1.
- Unlike other languages, python provides us the flexibility to use the negative indexing also.
- The negative indices are counted from the right.

# Positive and Negative indexing

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



# Splitting

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

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
----------	----------	----------	----------	----------	----------

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

# Find the Output of the Program

```
l1=[1,2,3,6,7]
print(l1[3:])
print(l1[1:3])
print(l1[:3])
```

# Output

- [6, 7]
- [2, 3]
- [1, 2, 3]

# List Built-in functions

SN	Function	Description
1	<code>len(list)</code>	It is used to calculate the length of the list.
2	<code>max(list)</code>	It returns the maximum element of the list.
3	<code>min(list)</code>	It returns the minimum element of the list.
4	<code>list(seq)</code>	It converts any sequence to the list.

# List built-in methods

S. No	Function	Description
1	<a href="#">list.append(obj)</a>	The element represented by the object obj is added to the list.
2	<a href="#">list.clear()</a>	It removes all the elements from the list.
3	<a href="#">list.count(obj)</a>	It returns the number of occurrences of the specified object in the list.
4	<a href="#">list.insert(index, obj)</a>	The object is inserted into the list at the specified index.
5	<a href="#">list.pop(obj=list[-1])</a>	It removes and returns the last object of the list.
6	<a href="#">list.remove(obj)</a>	It removes the specified object from the list.
7	<a href="#">list.reverse()</a>	It reverses the list.
8	<a href="#">list.sort([func])</a>	It sorts the list by using the specified compare function if given.
9	<a href="#">list.extend(seq)</a>	The sequence represented by the object seq is extended to the list.

# Find the Output of the Program

```
l1=[1,2,3]
l1.append(0)
print(l1)
l1.insert(2,7)
print(l1)
print(l1.pop(2))
print(l1)
l1.remove(1)
print(l1)
l1.reverse()
print(l1)
l1.sort()
print(l1)
l1.sort(reverse=True)
print(l1)
```

# Output

- [1, 2, 3, 0]
- [1, 2, 7, 3, 0]
- 7
- [1, 2, 3, 0]
- [2, 3, 0]
- [0, 3, 2]
- [0, 2, 3]
- [3, 2, 0]

# Python List Operations

Consider a List l1 = [1, 2, 3, 4], and l2 = [5, 6, 7, 8]

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	<code>l1*2 = [1, 2, 3, 4, 1, 2, 3, 4]</code>
Concatenation	It concatenates the list mentioned on either side of the operator.	<code>l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]</code>
Membership	It returns true if a particular item exists in a particular list otherwise false.	<code>print(2 in l1) prints True.</code>

Iteration	The for loop is used to iterate over the list elements.  for i in l1: print(i)	<b>Output</b>  1 2 3 4
Length	It is used to get the length of the list	len(l1) = 4

```
if __name__ == '__main__':
    N = int(input())
    L=[]
    for i in range(N):
        x=input().split(" ")
        command=x[0]
        if command=="insert":
            L.insert(int(x[1]),int(x[2]))
        if command=="remove":
            L.remove(int(x[1]))
        if command=="append":
            L.append(int(x[1]))
        if command=="sort":
            L.sort()
        if command=="pop":
            if(len(L)!=0):
                L.pop()
        if command=="reverse":
            L.reverse()
        if command=="print":
            print(L)
```

# Introduction to the list unpacking

```
colors = ['red', 'blue', 'green']
```

To assign the first, second, and third elements of the list to variables, you may assign individual elements to variables like this:

```
red = colors[0]
```

```
blue = colors[1]
```

```
green = colors[2]
```

```
red, blue, green = colors
```

If you use a fewer number of variables on the left side, you'll get an error. For example:

```
colors = ['red', 'blue', 'green']
```

```
red, blue = colors
```

Error

```
colors = ['cyan', 'magenta', 'yellow', 'black']  
cyan, magenta, *other = colors
```

```
print(cyan)  
print(magenta)  
print(other)
```

Here the first and second elements of the colors list to the cyan and magenta variables. And it assigns the last elements of the list to the other variable

# Python Tuple

- Python Tuple is used to store the sequence of immutable python objects.
- Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the value of the items stored in the tuple can not be changed.

# Example

```
tuple1=(11,12,1,4,56,32)
count=0;
for i in tuple1:
    print("tuple1[%d]=%d"%(count,i))
    count=count+1
```

# Output

tuple1[0]=11

tuple1[1]=12

tuple1[2]=1

tuple1[3]=4

tuple1[4]=56

tuple1[5]=32

# Tuple indexing and splitting

- The indexing and slicing in tuple are similar to lists. The indexing in the tuple starts from 0 and goes to `length(tuple) - 1`.
- The items in the tuple can be accessed by using the slice operator.
- Python also allows us to use the colon operator to access multiple items in the tuple.

# Tuple indexing and splitting

`Tuple = ( 0, 1, 2, 3, 4, 5 )`

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
----------	----------	----------	----------	----------	----------

`Tuple[0] = 0`

`Tuple[0:] = (0, 1, 2, 3, 4, 5)`

`Tuple[1] = 1`

`Tuple[:] = (0, 1, 2, 3, 4, 5)`

`Tuple[2] = 2`

`Tuple[2:4] = (2, 3)`

`Tuple[3] = 3`

`Tuple[1:3] = (1, 2)`

`Tuple[4] = 4`

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

`Tuple[5] = 5`

# Basic Tuple operations

Operator	Description	Example
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	$T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)$
Concatenation	It concatenates the tuple mentioned on either side of the operator.	$T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)$
Membership	It returns true if a particular item exists in the tuple otherwise false.	<code>print (2 in T1) prints True.</code>

Iteration	The for loop is used to iterate over the tuple elements.	<pre>for i in T1:     print(i)</pre> <p><b>Output</b></p> <p>1 2 3 4 5</p>
Length	It is used to get the length of the tuple.	<pre>len(T1) = 5</pre>

# Python Tuple inbuilt functions

SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple.
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

# List VS Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [ ].	The literal syntax of the tuple is shown by the ( ).
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	The list Is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

# Python Set

- The set in python can be defined as the unordered collection of various items enclosed within the curly braces.
- The elements of the set can not be duplicate. The elements of the python set must be immutable.
- 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 a set

```
Days = {"Monday", "Tuesday", "Wednesday", "T  
hursday", "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'}
```

```
<class 'set'>
```

```
looping through the set elements ...
```

```
Friday
```

```
Tuesday
```

```
Monday
```

```
Saturday
```

```
Thursday
```

```
Sunday
```

```
Wednesday
```

# **set() method**

```
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)
```

# Python Set operations

## Adding items to the set

- Python provides the add() method which can be used to add some particular item to the set.

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

```
Months.add("July");
```

```
Months.add("August");
```

To add more than one item in the set, Python provides the **update()** method.

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

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

print("\nupdating the original set ... ")
Months.update(["July", "August", "September", "October"]);

print(Months)
```

# Removing items from the set

- Python provides **discard()** method which can be used to remove the items from the set.
- Python also provide the **remove()** method to remove the items from the set.
- We can also use the **pop()** method to remove the item. However, this method will always remove the last item.
- Python provides the **clear()** method to remove all the items from the set.
- To add more than one item in the set, Python provides the **update()** method.

## Difference between discard() and remove()

- If the key to be deleted from the set using discard() doesn't exist in the set, the python will not give the error. The program maintains its control flow.
- On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the python will give the error.

# Example

```
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")
```

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

```
Months.remove("January");
Months.remove("May");
print("\\nPrinting the modified set...");
print(Months)
Months.pop();
print("\\nPrinting the modified set...");
print(Months)
Months.clear()
print("\\nPrinting the modified set...")
print(Months)
```

# Union of two Sets

- The union of two sets are calculated by using the or (|) operator.
- The union of the two sets contains the all the items that are present in both the sets.

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

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

```
print(Days1 | Days2)
```

**Output:**

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

# union() method

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

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

```
print(Days1.union(Days2))
```

```
#printing the union of the sets
```

**Output:**

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

# Intersection of two sets

- The & (intersection) operator is used to calculate the intersection of the two sets in python.
- The intersection of the two sets are given as the set of the elements that common in both sets.

# using & operator

```
set1 = {"Ayush", "John", "David", "Martin"}
```

```
set2 = {"Steve", "Milan", "David", "Martin"}
```

```
print(set1&set2)
```

#prints the intersection of the two sets

**Output:**

```
{'Martin', 'David'}
```

# using intersection() method

```
set1 = {"Ayush", "John", "David", "Martin"}  
set2 = {"Steave", "Milan", "David", "Martin"}  
print(set1.intersection(set2))
```

#prints the intersection of the two sets

## Output:

```
{'Martin', 'David'}
```

# Difference of two sets

- The difference of two sets can be calculated by using the subtraction (-) operator.
- The resulting set will be obtained by removing all the elements from set 1 that are present in set 2.

# using subtraction ( - ) operator

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

```
Days2 = {"Monday", "Tuesday", "Sunday"}
```

```
print(Days1-  
      Days2) #{"Wednesday", "Thursday" will be pri  
      nted}
```

**Output:**

```
{'Thursday', 'Wednesday'}
```

# using difference() method

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

```
Days2 = {"Monday", "Tuesday", "Sunday"}
```

```
print(Days1.difference(Days2))
```

```
# prints the difference of the two sets Days1 and  
Days2
```

## Output:

```
{'Thursday', 'Wednesday'}
```

# Set comparisons

- Python allows us to use the comparison operators i.e., `<`, `>`, `<=`, `>=`, `==` with the sets by using which we can check whether a set is subset, superset, or equivalent to other set.
- The boolean true or false is returned depending upon the items present inside the sets.

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

```
Days2 = {"Monday", "Tuesday"}
```

```
Days3 = {"Monday", "Tuesday", "Friday"}
```

#Days1 is the superset of Days2 hence it will print true.

```
print (Days1>Days2)
```

#prints false since Days1 is not the subset of Days2

```
print (Days1<Days2)
```

#prints false since Days2 and Days3 are not equivalent

```
print (Days2 == Days3)
```

## **Output:**

True

False

False

# FrozenSets

- The frozen sets are the immutable form of the normal sets, i.e., the items of the frozen set can not be changed and therefore it can be used as a key in dictionary.
- The elements of the frozen set can not be changed after the creation.
- We cannot change or append the content of the frozen sets by using the methods like add() or remove().

```
Frozenset = frozenset([1,2,3,4,5])
print(type(Frozenset))
print("\nprinting the content of frozen set...")
for i in Frozenset:
    print(i);
Frozenset.add(6)
```

# Output

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

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

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
'frozenset' object has no attribute 'add'
```

# Python Dictionary

- Dictionary is used to implement the key-value pair in python.
- The dictionary is the data type in python which can simulate the real-life data arrangement where some specific value exists for some particular key.
- A dictionary is the collection of key-value pairs where the value can be any python object whereas the keys are the immutable python object, i.e., Numbers, string or tuple.

# Creating the dictionary

- The dictionary can be created by using multiple key-value pairs enclosed with the small brackets () and separated by the colon (:).
- The collections of the key-value pairs are enclosed within the curly braces {}.

```
Employee = {"Name": "John", "Age": 29, "salary":  
    :25000,"Company":"GOOGLE"}
```

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

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

```
print(Employee)
```

# Output

- <class 'dict'>
- printing Employee data ....
- {'Age': 29, 'salary': 25000, 'Name': 'John',  
'Company': 'GOOGLE'}

# Accessing the dictionary values

```
Employee = {"Name": "John", "Age": 29, "salary":  
    :25000,"Company":"GOOGLE"}
```

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

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

```
print("Name : %s" %Employee["Name"])
```

```
print("Age : %d" %Employee["Age"])
```

```
print("Salary : %d" %Employee["salary"])
```

```
print("Company : %s" %Employee["Company"])
```

# Output

<class 'dict'>

printing Employee data ....

Name : John Age : 29 Salary : 25000 Company :  
GOOGLE

# Updating dictionary values

- The dictionary is a mutable data type, and its values can be updated by using the specific keys.

## Example to update the dictionary values.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"  
Company":"GOOGLE"}  
print(type(Employee))  
print("printing Employee data .... ")  
print(Employee)  
print("Enter the details of the new employee....");  
Employee["Name"] = input("Name: ");  
Employee["Age"] = int(input("Age: "));  
Employee["salary"] = int(input("Salary: "));  
Employee["Company"] = input("Company:");  
print("printing the new data");  
print(Employee)
```

# Deleting elements using del keyword

```
Employee = {"Name": "John", "Age": 29, "salary":25  
000,"Company":"GOOGLE"}
```

```
del Employee["Name"]
```

```
del Employee["Company"]
```

```
print("printing the modified information ")
```

```
print(Employee)
```

```
print("Deleting the dictionary: Employee");
```

```
del Employee
```

```
print("Lets try to print it again ");
```

```
print(Employee)
```

# Output

- printing the modified information
- {'Age': 29, 'salary': 25000}
- Deleting the dictionary: Employee Lets try to print it again Traceback (most recent call last): File "list.py", line 13, in <module>  
print(Employee)
- NameError: name 'Employee' is not defined

# Iterating Dictionary

**for loop to print all the keys of a dictionary**

```
Employee = {"Name": "John", "Age": 29, "salary":25  
000,"Company":"GOOGLE"}
```

```
for x in Employee:  
    print(x)
```

**Output**

Name

Company

Salary

Age

**for loop to print the values of the dictionary by using values() method.**

```
Employee = {"Name": "John", "Age": 29, "salary":  
    :25000,"Company":"GOOGLE"}
```

```
for x in Employee.values():  
    print(x);
```

**Output:**

GOOGLE

25000

John

29

#for loop to print the items of the dictionary by using items() method.

```
Employee = {"Name": "John", "Age": 29, "salary":  
:25000,"Company":"GOOGLE"}
```

```
for x in Employee.items():  
    print(x);
```

**Output:**

('Name', 'John')

('Age', 29)

('salary', 25000)

('Company', 'GOOGLE')

# Properties of Dictionary keys

1. In the dictionary, we can not store multiple values for the same keys. If we pass more than one values for a single key, then the value which is last assigned is considered as the value of the key.
2. In python, the key cannot be any mutable object. We can use numbers, strings, or tuple as the key but we can not use any mutable object like the list as the key in the dictionary.

# Built-in Dictionary functions

len(dict)	It is used to calculate the length of the dictionary.
str(dict)	It converts the dictionary into the printable string representation.
type(variable)	It is used to print the type of the passed variable.

# Dictionary setdefault() Method

- Python setdefault() method is used to set default value to the key.
- It returns value, if the key is present. Otherwise it insert key with the default value.
- Default value for the key is None.

# Example

```
coursefee = {'B,Tech': 400000, 'BA':2500, 'B.COM  
':50000}
```

```
p = coursefee.setdefault('BCA',100000)
```

```
print("default",p)
```

```
print(coursefee)
```

# Dictionary update() Method

- Python update() method updates the dictionary with the key and value pairs.
- It inserts key/value if it is not present.
- It updates key/value if it is already present in the dictionary.
- It also allows an iterable of key/value pairs to update the dictionary. like: update(a=10,b=20) etc.

# Example

```
inventory = {'Fan': 200, 'Bulb':150, 'Led':1000}  
inventory.update({'cooler':50})  
inventory.update(Switch=300,case=200)  
print("Updated inventory:",inventory)
```

Updated inventory:

```
{'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 50,  
'Switch': 300, 'case': 200}
```

# List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

## Example:

- Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.
- Without list comprehension you will have to write a for statement with a conditional test inside.

# Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
```

```
newlist = [x for x in fruits if "a" in x]
```

```
print(newlist)
```

# Output

```
['apple', 'banana', 'mango']
```

# Syntax

```
newlist =  
[expression for item in iterable if  
condition == True]
```

# Example

```
fruits = ["apple", "banana", "cherry", "kiwi",  
"mango"]
```

```
newlist = [x for x in fruits if x != "apple"]
```

```
print(newlist)
```

# Example

```
fruits = ["apple", "banana", "cherry", "kiwi",  
"mango"]
```

```
newlist = [x for x in fruits]
```

```
print(newlist)
```

# Example

You can use the range() function to create an iterable:

```
newlist = [x for x in range(10)]
```

# Example

```
newlist = [x for x in range(10) if x < 5]
```

# Example

```
fruits = ["apple", "banana", "cherry", "kiwi",  
"mango"]
```

```
newlist = [x if x != "banana" else "orange" for x  
in fruits]
```

```
print(newlist)
```

# Python Lambda Functions

- Python allows us to not declare the function in the standard manner, i.e., by using the def keyword.
- Rather, the anonymous functions are declared by using lambda keyword.
- However, Lambda functions can accept any number of arguments, but they can return only one value in the form of expression.

## **lambda arguments : expression**

### **Example 1**

**x = lambda a:a+10**

a is an argument and a+10

is an expression which got evaluated and returned.

**print("sum = ",x(20))**

### **Output**

sum = 30

## Example 2

Multiple arguments to Lambda function

```
x = lambda a,b:a+b
```

```
# a and b are the arguments and a+b is the expression which gets evaluated and returned.
```

```
print("sum = ",x(20,10))
```

**Output:**

```
sum = 30
```

## Example 2

Use of lambda function with filter

#program to filter out the list which contains number divisible by 3 numbers

List = {1,2,3,4,10,123,22}

Oddlist = list(filter(lambda x:(x%3 == 0),List))

**print(Oddlist)**

### Output:

[3, 123]

## Example 3

```
List = {1,2,3,4,10,123,22}
```

```
new_list = list(map(lambda x:x*3,List))
```

```
print(new_list)
```

## Output:

```
[3, 6, 9, 12, 30, 66, 369]
```



# Python Programming

## Unit 4

### (KNC-302)

Prepared By  
Abhishek Kesharwani  
Assistant Professor, UCER Naini, Allahabad

## **UNIT IV**

- Sieve of Eratosthenes: generate prime numbers with the help of an algorithm given by the Greek Mathematician named Eratosthenes, whose algorithm is known as Sieve of Eratosthenes.
- File I/O:
- File input and output operations in Python Programming
- Exceptions and Assertions.

# Sieve of Eratosthenes

Algorithm to find all the prime numbers less than or equal to a given integer  $n$  by Eratosthenes' method

1. Create a list of consecutive integers from 2 to  $n$ :  $(2, 3, 4, \dots, n)$ .
2. Initially, let  $p$  equal 2, the first prime number.
3. Starting from  $p^2$ , count up in increments of  $p$  and mark each of these numbers greater than or equal to  $p^2$  itself in the list. These numbers will be  $p(p+1), p(p+2), p(p+3)$ .

# Sieve of Eratosthenes

4. Find the first number greater than  $p$  in the list that is not marked. If there was no such number, stop. Otherwise, let  $p$  now equal this number (which is the next prime), and repeat from step 3.

### Example:

Let us take an example when  $n = 50$ . So we need to print all numbers smaller than or equal to 50.

We create a list of all numbers from 2 to 50.

	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	50

According to the algorithm we will mark all the numbers which are divisible by 2.

	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	50

	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	50

Now we move to our next unmarked number 3 and mark all the numbers which are multiples of 3 and are greater than or equal to the square of it.

	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	50

We move to our next unmarked number 5 and mark all multiples of 5 and are greater than or equal to the square of it.

	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	50

So the prime numbers are the unmarked ones: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

# Sieve of Eratosthenes

```
def sieve(n):
    global primes
    primes=[True]*(n+1)
    primes[0]=False
    primes[1]=False

    for j in range(2,n+1):
        if primes[j]==False:
            continue
        for i in range (j*j,n+1,j):
            primes[i]=False
```

```
n=int(input('input n'))
```

```
sieve(n)
```

```
for i in range (2,n+1):
```

```
    if primes[i]:
```

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

# Output

input n 100

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59  
61 67 71 73 79 83 89 97

# Python File Open

- File handling is an important part of any web application.
- Python has several functions for creating, reading, updating, and deleting files.

The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; filename, and mode.

# Create a New File

To create a new file in Python, use the `open()` method, with one of the following parameters:

- "`x`" - Create - will create a file, returns an error if the file exist
- "`a`" - Append - will create a file if the specified file does not exist
- "`w`" - Write - will create a file if the specified file does not exist

There are four different methods (modes) for opening a file:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

## Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the **default values**, you do not need to specify them.

**Note:** Make sure the file exists, or else you will get an error.

# Opening a file

- Python provides the `open()` function which accepts two arguments, file name and access mode in which the file is accessed.
- The function returns a file object which can be used to perform various operations like reading, writing, etc.

# Example

```
#opens the file file.txt in read mode
```

```
fileptr = open("file.txt","r")
```

```
if fileptr:
```

```
    print("file is opened successfully")
```

# The close() method

- Once all the operations are done on the file, we must close it through our python script using the close() method.
- Any unwritten information gets destroyed once the close() method is called on a file object.

# Reading the file

- To read a file using the python script, the python provides us the read() method.
- The read() method reads a string from the file. It can read the data in the text as well as binary format.

# Example

```
fileptr = open("file.txt","r");
```

```
content = fileptr.read();
```

```
print(type(content))
```

```
print(content)
```

```
fileptr.close()
```

# Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

## Example

- Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
```

```
print(f.read(5))
```

# Looping through the file

```
fileptr = open("file.txt","r");
```

```
for i in fileptr:
```

```
    print(i) # i contains each line of the file
```

By looping through the lines of the file, you can  
read the whole file, line by line:

# Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

**"a"** - Append - will append to the end of the file

**"w"** - Write - will overwrite any existing content

# Delete a File

To delete a file, you must import the OS module, and run its `os.remove()` function:

## Example

Remove the file "demofile.txt":

```
import os  
os.remove("demofile.txt")
```

# Check if File exist

```
import os  
if os.path.exists("ak123"):  
    os.remove("demofile.txt")  
else:  
    print("The file does not exist")
```

To avoid getting an error, you might want to check if the file exists before you try to delete it:

# Delete Folder

To delete an entire folder, use the `os.rmdir()` method:

## Example

Remove the folder "myfolder":

```
import os
```

```
os.rmdir("myfolder")
```

# File Pointer positions

- Python provides the `tell()` method which is used to print the byte number at which the file pointer exists.

## Modifying file pointer position

- In the real world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.
- `seek()` method which enables us to modify the file pointer position externally.

# Example

```
fileptr = open("file2.txt","r")
```

```
print("The filepointer is at byte :",fileptr.tell())
content = fileptr.read();
print("After reading, the filepointer is at:",fileptr.tell())
```

# Output

The filepointer is at byte : 0

After reading, the filepointer is at 26

# Modifying file pointer position

```
# open the file file2.txt in read mode  
fileptr = open("file2.txt","r")
```

```
#initially the filepointer is at 0  
print("The filepointer is at byte :",fileptr.tell())
```

```
#changing the file pointer location to 10.  
fileptr.seek(10);
```

```
#tell() returns the location of the fileptr.  
print("After reading, the filepointer is at:",fileptr.tell())
```

# Output

The filepointer is at byte : 0

After reading, the filepointer is at 10

# Exceptions

- An exception can be defined as an abnormal condition in a program resulting in the disruption in the flow of the program.
- Whenever an exception occurs, the program halts the execution, and thus the further code is not executed.
- Therefore, an exception is the error which python script is unable to tackle with.
- Python provides us with the way to handle the Exception so that the other part of the code can be executed without any disruption.

# Common Exceptions

- 1. ZeroDivisionError:** Occurs when a number is divided by zero.
- 2. NameError:** It occurs when a name is not found. It may be local or global.
- 3. IndentationError:** If incorrect indentation is given.
- 4. IOError:** It occurs when Input Output operation fails.
- 5. EOFError:** It occurs when the end of the file is reached, and yet operations are being performed.

# Exception handling in python

- If the python program contains suspicious code that may throw the exception, we must place that code in the try block.
- The try block must be followed with the except statement which contains a block of code that will be executed if there is some exception in the try block.

try

{ Run this code }

except

{ Run this code if an exception occurs }

# Syntax

**try:**

#block of code

**except Exception1:**

#block of code

**except Exception2:**

#block of code

#other code

**try**

{ Run this code }

**except**

{ Run this code if an exception occurs }

**else**

{ Run this code if no exception occurs }

# try except else

- We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.

# Example

**try:**

```
a = int(input("Enter a:"))
```

```
b = int(input("Enter b:"))
```

```
c = a/b;
```

```
print("a/b = %d" %c)
```

**except** Exception:

```
print("can't divide by zero")
```

**else:**

```
print("Hi I am else block")
```

# Output

Enter a:10

Enter b:2

a/b = 5

Hi I am else block

# Points to remember

- We can declare multiple exceptions in the except statement since the try block may contain the statements which throw the different type of exceptions.
- We can also specify an else block along with the try-except statement which will be executed if no exception is raised in the try block.
- The statements that don't throw the exception should be placed inside the else block.

# Example

**try:**

#this will throw an exception if the file doesn't exist.

```
fileptr = open("file.txt","r")
```

**except** IOError:

```
    print("File not found")
```

**else:**

```
    print("The file opened successfully")
```

```
fileptr.close()
```

# The finally block

- We can use the finally block with the try block in which, we can place the important code which must be executed before the try statement throws an exception.

**try**

{ Run this code }

**except**

{ Run this code if an exception occurs }

**else**

{ Run this code if no exception occurs }

**finally**

{ Always run this code }

# Example

```
try:  
    a = int(input("Enter a:"))  
    b = int(input("Enter b:"))  
    c = a/b;  
    print("a/b = %d"%c)  
  
except:  
    print("can't divide by zero")  
  
else:  
    print("Hi I am else block")  
  
finally:  
    print("finally block is always  
executed")
```

# Declaring multiple exceptions

**try:**

#block of code

**except** (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>)

#block of code

**else:**

#block of code

# Example

**try:**

a=10/0;

**except** ArithmeticError,StandardError:

**print** "Arithmetic Exception"

**else:**

**print** "Successfully Done"

# Raising exceptions

An exception can be raised by using the raise clause in python. The syntax to use the raise statement is given below.

**syntax**

**raise Exception\_class,<value>**

# Example

**try**:

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

**if** age<18:

```
    raise ValueError;
```

**else**:

```
    print("the age is valid")
```

**except** ValueError:

```
    print("The age is not valid")
```

# Example

**try:**

```
a = int(input("Enter a?"))
```

```
b = int(input("Enter b?"))
```

**if** b **is** 0:

```
    raise ArithmeticError
```

**else:**

```
    print("a/b = ",a/b)
```

**except** ArithmeticError:

```
    print("The value of b can't be 0")
```

# Output

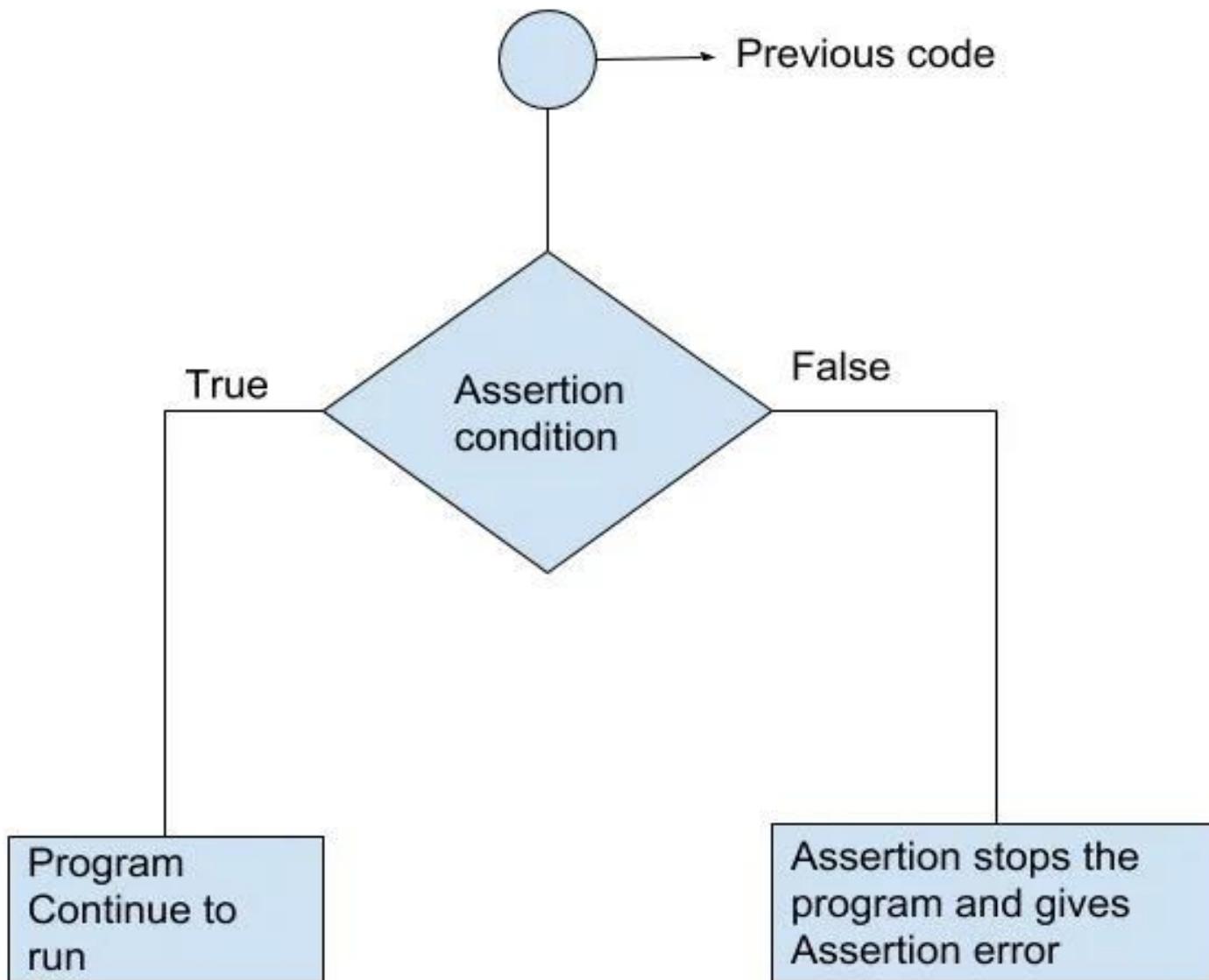
Enter a 10

Enter b 0

The value of b can't be 0

# Assertions

- Assertions are statements that assert or state a fact confidently in your program.
- For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.
- Assertions are simply boolean expressions that check if the conditions return true or not.
- If it is true, the program does nothing and moves to the next line of code.
- However, if it's false, the program stops and throws an error.



# Python assert Statement

- Python has built-in assert statement to use assertion condition in the program.
- assert statement has a condition or expression which is supposed to be always true.
- If the condition is false assert halts the program and gives an AssertionError.

# Syntax for using Assert in Pyhton:

- assert <condition>
- assert <condition>,<error message>

```
def avg(marks):  
    assert (len(marks) != 0,"List is empty.")  
    return sum(marks)/len(marks)
```

```
mark2 = [55,88,78,90,79]  
print("Average of mark2:",avg(mark2))
```

```
mark1 = []  
print("Average of mark1:",avg(mark1))
```

Average of mark2: 78.0

AssertionError: List is empty.

# Key Points to Remember

- Assertions are the condition or boolean expression which are always supposed to be true in the code.
- assert statement takes an expression and optional message.
- assert statement is used to check types, values of argument and the output of the function.
- assert statement is used as debugging tool as it halts the program at the point where an error occurs.



# Python Programming

## Unit 4

### (KNC-302)

Prepared By  
Abhishek Kesharwani  
Assistant Professor, UCER Naini, Allahabad

## **UNIT IV**

- Modules: Introduction,
- Importing Modules,
- Abstract Data Types : Abstract data types and ADT interface in Python Programming.
- Classes : Class definition and other operations in the classes
- Special Methods ( such as `_init_`,`_str_`, comparison methods and Arithmetic methods etc.)
- Class Example , Inheritance , Inheritance and OOP.

# Python Modules

- A python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module.
- We may have a runnable code inside the python module.
- Modules in Python provides us the flexibility to organize the code in a logical way.
- To use the functionality of one module into another, we must have to import the specific module.

# Loading the module in our python code

Python provides two types of statements as defined below.

- The import statement
- The from-import statement

# The import statement

- The import statement is used to import all the functionality of one module into another. Here, we must notice that we can use the functionality of any python source file by importing that file as the module into another python source file.
- We can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times, it has been imported into our file.

- The syntax to use the import statement is given below.

**import module1,module2,..... module n**

Let's create the module named as **file.py**.

#displayMsg prints a message to the name being passed.

```
def displayMsg(name)  
print("Hi "+name);
```

```
import file;  
name = input("Enter the name?")  
file.displayMsg(name)
```

## Output:

```
Enter the name John  
Hi John
```

# The from-import statement

- Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific attributes of a module.
- This can be done by using from? import statement. The syntax to use the from-import statement is given below.

```
from < module-name> import <name 1>, <name  
2>..,<name n>
```

# calculation.py

```
def summation(a,b):  
    return a+b  
  
def multiplication(a,b):  
    return a*b;  
  
def divide(a,b):  
    return a/b;
```

# Main.py

```
from calculation import summation
#it will import only the summation() from calculation.py
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b)) #we do not need
to specify the module name while accessing summation()
```

# Output

Enter the first number 10

Enter the second number 20

Sum = 30

NOTE: We can also import all the attributes from a module by using \*.

`from <module> import *`

# Renaming a module

Python provides us the flexibility to import some module with a specific name so that we can use this name to use that module in our python source file.

## syntax

```
import <module-name> as <specific-name>
```

# dir() function

- The dir() function returns a sorted list of names defined in the passed module.
- This list contains all the sub-modules, variables and functions defined in this module.

## Example

```
import json
```

```
List = dir(json)
```

```
print(List)
```

# Output

```
['JSONDecoder', 'JSONEncoder', '__all__',  
 '__author__', '__builtins__', '__cached__',  
 '__doc__', '__file__', '__loader__', '__name__',  
 '__package__', '__path__', '__spec__',  
 '__version__', '_default_decoder',  
 '_default_encoder', 'decoder', 'dump', 'dumps',  
 'encoder', 'load', 'loads', 'scanner']
```

# Python OOPs Concepts

- Python is an object-oriented programming language.
- It allows us to develop applications using an Object Oriented approach.
- In Python, we can easily create and use classes and objects.

Major principles of object-oriented programming system are given below.

- Object
- Class
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation

# Object

- The object is an entity that has state and behavior.
- It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.
- Everything in Python is an object, and almost everything has attributes and methods.
- All functions have a built-in attribute `__doc__` which returns the doc string defined in the function source code.

# Class

- It is a logical entity that has some specific attributes and methods.

For example:

if you have an employee class then it should contain an attribute and method, i.e. an email id, name, age, salary, etc.

# Method

- The method is a function that is associated with an object.
- In Python, a method is not unique to class instances.
- Any object type can have methods.

# Inheritance

- Inheritance is the most important aspect of object-oriented programming which simulates the real world concept of inheritance.
- It specifies that the child object acquires all the properties and behaviors of the parent object.
- By using inheritance, we can create a class which uses all the properties and behavior of another class.

- The new class is known as a derived class or child class, and the one whose properties are acquired is known as a base class or parent class.
- It provides re-usability of the code.

# Polymorphism

- Polymorphism contains two words "poly" and "morphs". Poly means many and Morphs means form, shape.
- By polymorphism, we understand that one task can be performed in different ways.

# Encapsulation

- Encapsulation is also an important aspect of object-oriented programming.
- It is used to restrict access to methods and variables.
- In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

# Data Abstraction

- Data abstraction and encapsulation both are often used as synonyms.
- Both are nearly synonym because data abstraction is achieved through encapsulation.
- Abstraction is used to hide internal details and show only functionalities.
- Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does.

# Creating classes in python

- In python, a class can be created by using the keyword class followed by the class name.

## Syntax

```
class ClassName:
```

```
    #statement_suite
```

A class contains a statement suite including fields, constructor, function, etc. definition.

# Example

```
class Employee:
```

```
    id = 10;
```

```
    name = "ayush"
```

```
    def display (self):
```

```
        print(self.id,self.name)
```

NOTE: self is used as a reference variable which refers to the current class object. It is always the first argument in the function definition

# Creating an instance of the class

- A class needs to be instantiated if we want to use the class attributes in another class or method.
- A class can be instantiated by calling the class using the class name.

# Example

```
class Employee:  
    id = 10;  
    name = "John"  
    def display (self):  
        print("ID: %d \nName:  
%s"%(self.id,self.name))  
emp = Employee()  
emp.display()
```

# Output

ID: 10

Name: ayush

# Python Constructor

A constructor is a special type of method (function) which is used to initialize the instance members of the class.

Constructors can be of two types.

- Parameterized Constructor
- Non-parameterized Constructor

# Creating the constructor in python

- Constructor definition is executed when we create the object of this class.
- Constructors also verify that there are enough resources for the object to perform any start-up task.
- `__init__` simulates the constructor of the class. This method is called when the class is instantiated.
- It is mostly used to initialize the class attributes.
- Every class must have a constructor, even if it simply relies on the default constructor.

# Example

```
class Employee:  
    def __init__(self,name,id):  
        self.id = id;  
        self.name = name;  
    def display (self):  
        print("ID: %d \nName: %s"%(self.  
id,self.name))  
emp1 = Employee("John",101)  
emp2 = Employee("David",102)  
emp1.display();  
emp2.display();
```

# Output

ID: 101

Name: John

ID: 102

Name: David

# Python Non-Parameterized Constructor

```
class Student:  
    def __init__(self):  
        print("This is non parameterized constructor")  
  
    def show(self,name):  
        print("Hello",name)  
st1 = Student()  
st1.show("John")
```

# Output

This is non parametrized constructor

Hello John

# Parameterized Constructor Example

```
class Student:  
    # Constructor - parameterized  
    def __init__(self, name):  
        print("This is parametrized constructor")  
        self.name = name  
    def show(self):  
        print("Hello",self.name)  
student = Student("John")  
student.show()
```

# Output

This is parametrized constructor

Hello John

# Python In-built class functions

SN	Function	Description
1	getattr(obj, name, default)	It is used to access the attribute of the object.
2	setattr(obj, name, value)	It is used to set a particular value to the specific attribute of an object.
3	delattr(obj, name)	It is used to delete a specific attribute.
4	hasattr(obj, name)	It returns true if the object contains some specific attribute.

# Example

```
class Student:  
    def __init__(self,name,id,age):  
        self.name = name;  
        self.id = id;  
        self.age = age  
s = Student("John",101,22)  
print(getattr(s,'name'))  
setattr(s,"age",23)  
print(getattr(s,'age'))  
print(hasattr(s,'id'))  
delattr(s,'age')  
print(s.age)
```

# Output

John

23

True

AttributeError: 'Student' object has no attribute  
'age'

```
class Student:  
    def __init__(self,name,roll,marks):  
        self.name=name  
        self.roll=roll  
        self.marks=marks  
s1=Student("Rahul",1,65)  
print(getattr(s1,'name'))  
print(hasattr(s1,'roll'))  
setattr(s1,'roll',3)  
print(s1.roll)  
delattr(s1,'roll')  
print(s1.roll)
```

# Built-in class attributes

SN	Attribute	Description
1	<code>__dict__</code>	It provides the dictionary containing the information about the class namespace.
2	<code>__doc__</code>	It contains a string which has the class documentation
3	<code>__name__</code>	It is used to access the class name.
4	<code>__module__</code>	It is used to access the module in which, this class is defined.
5	<code>__bases__</code>	It contains a tuple including all base classes.

# Example

```
class Student:  
    def __init__(self,name,id,age):  
        self.name = name;  
        self.id = id;  
        self.age = age  
    def display_details(self):  
        print("Name:%s, ID:%d, age:%d"%(se  
lf.name,self.id))  
s = Student("John",101,22)  
print(s.__doc__)  
print(s.__dict__)  
print(s.__module__)
```

# Output

None

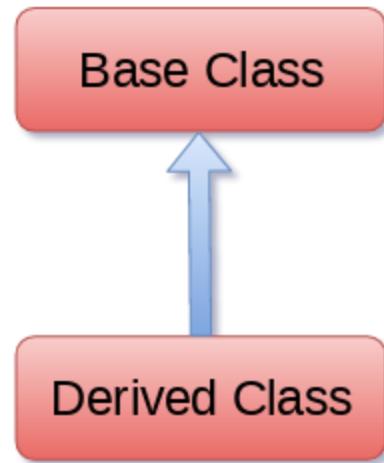
```
{'name': 'John', 'id': 101, 'age': 22}
```

\_\_main\_\_

# Python Inheritance

- Inheritance is an important aspect of the object-oriented paradigm.
- Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.
- In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class.

# Syntax



```
class derived-class(base class):  
    <class-suite>
```

A class can inherit multiple classes by mentioning all of them inside the bracket.

## Syntax

**class** derive-class

(<base **class** 1>, <base **class** 2>, ..... <base **class** n>):

<**class** - suite>

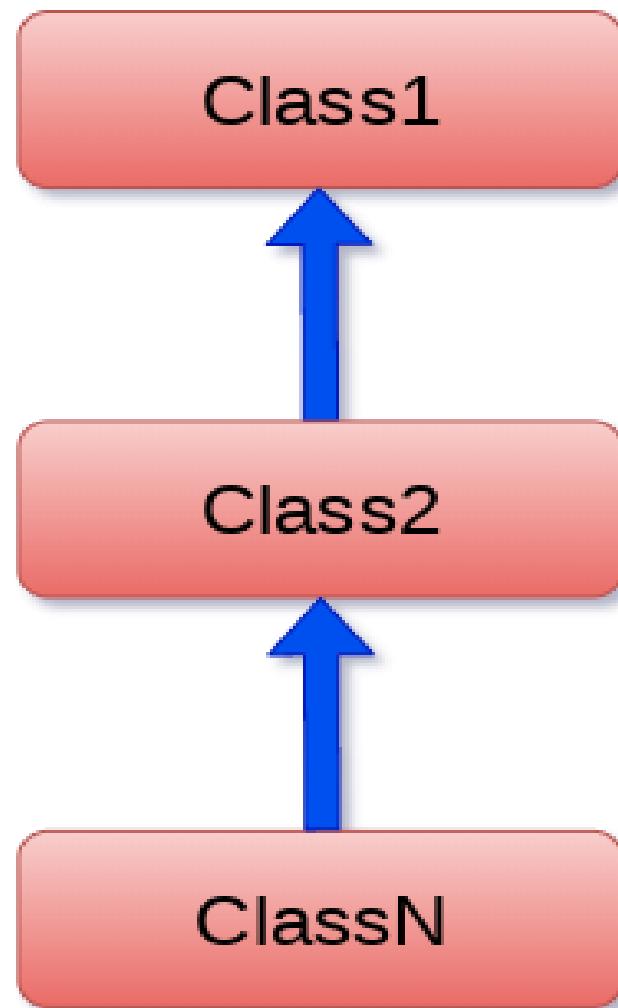
# Example

```
class Animal:  
    def speak(self):  
        print("Animal Speaking")  
#child class Dog inherits the base class Animal  
class Dog(Animal):  
    def bark(self):  
        print("dog barking")  
d = Dog()  
d.bark()  
d.speak()
```

# Python Multi-Level inheritance

- Multi-Level inheritance is possible in python like other object-oriented languages.
- Multi-level inheritance is archived when a derived class inherits another derived class.
- There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.

# Multi-Level inheritance



# Syntax

**class** class1:

  <**class**-suite>

**class** class2(class1):

  <**class** suite>

**class** class3(class2):

  <**class** suite>

# Example

```
1. class Animal:  
2.     def speak(self):  
3.         print("Animal Speaking")  
4. #The child class Dog inherits the base class Animal  
5. class Dog(Animal):  
6.     def bark(self):  
7.         print("dog barking")  
8. #The child class Dogchild inherits another child class Dog  
9. class DogChild(Dog):  
10.    def eat(self):  
11.        print("Eating bread...")  
12.d = DogChild()  
13.d.bark()  
14.d.speak()  
15.d.eat()
```

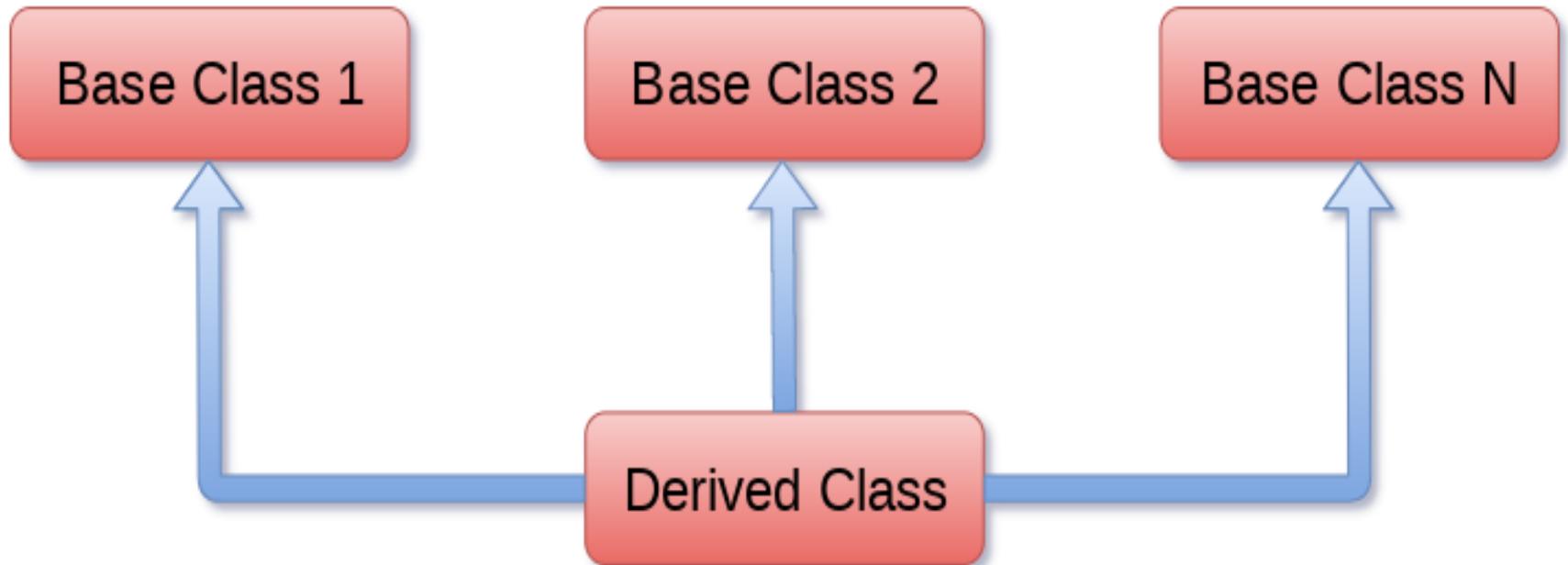
# Output

dog barking

Animal Speaking

Eating bread..

# Python Multiple inheritance



# Syntax

```
class Base1:  
  <class-suite>
```

```
class Base2:  
  <class-suite>
```

.

.

.

```
class BaseN:  
  <class-suite>
```

```
class Derived(Base1, Base2, ..... BaseN):  
  <class-suite>
```

# Example

```
class Calculation1:  
    def Summation(self,a,b):  
        return a+b  
class Calculation2:  
    def Multiplication(self,a,b):  
        return a*b  
class Derived(Calculation1,Calculation2):  
    def Divide(self,a,b):  
        return a/b  
d = Derived()  
print(d.Summation(10,20))  
print(d.Multiplication(10,20))  
print(d.Divide(10,20))
```

# issubclass(sub,sup) method

- The `issubclass(sub, sup)` method is used to check the relationships between the specified classes.
- It returns true if the first class is the subclass of the second class, and false otherwise.

# Example

```
class Calculation1:  
    def Summation(self,a,b):  
        return a+b;  
class Calculation2:  
    def Multiplication(self,a,b):  
        return a*b;  
class Derived(Calculation1,Calculation2):  
    def Divide(self,a,b):  
        return a/b;  
d = Derived()  
print(issubclass(Derived,Calculation2))  
print(issubclass(Calculation1,Calculation2))
```

# Output

- True
- False

# Method Overriding

- When the parent class method is defined in the child class with some specific implementation, then the concept is called method overriding.
- We may need to perform method overriding in the scenario where the different definition of a parent class method is needed in the child class.

# Example

```
class Animal:
```

```
    def speak(self):
```

```
        print("speaking")
```

```
class Dog(Animal):
```

```
    def speak(self):
```

```
        print("Barking")
```

```
d = Dog()
```

```
d.speak()
```

# Output

- Barking

# Data abstraction in python

- Abstraction is an important aspect of object-oriented programming.
- In python, we can also perform data hiding by adding the double underscore (\_\_\_\_) as a prefix to the attribute which is to be hidden.
- After this, the attribute will not be visible outside of the class through the object.

```
class Employee:  
    __count = 0;  
    def __init__(self):  
        Employee.__count = Employee.__count+1  
    def display(self):  
        print("The number of employees",Employee  
              .__count)  
    emp = Employee()  
    emp2 = Employee()  
    try:  
        print(emp.__count)  
    finally:  
        emp.display()
```

# Output

- The number of employees 2
- AttributeError: 'Employee' object has no attribute '\_\_count'



# Python Programming

## Unit 5

### (KNC-302)

Prepared By  
Abhishek Kesharwani  
Assistant Professor, UCER Naini, Allahabad

## **UNIT V**

- Iterators & Recursion:
- Recursive Fibonacci
- Tower of Hanoi
- Search: Simple Search and Estimating Search Time
- Binary Search and Estimating Binary Search Time
- Sorting & Merging:
- Selection Sort,
- Merge List,
- Merge Sort
- Higher Order Sort

# Recursion

- Recursive Fibonacci
- Tower Of Hanoi

# Fibonacci series

In mathematical terms, the sequence  $F_n$  of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with seed values  $F_0 = 0$  and  $F_1 = 1$ .

- The Fibonacci numbers are the numbers in the following integer sequence.
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, .....

# Fibonacci Number in Python

```
def Fibonacci(n):
    if n < 0:
        print("Incorrect input")
        # First Fibonacci number is 0
    elif n == 1:
        return 0
    # Second Fibonacci number is 1
    elif n == 2:
        return 1
    else:
        return Fibonacci(n - 1) + Fibonacci(n - 2)

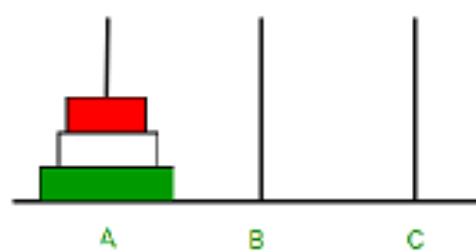
print(Fibonacci(int(input("Enter the number"))))
```

# Tower of Hanoi

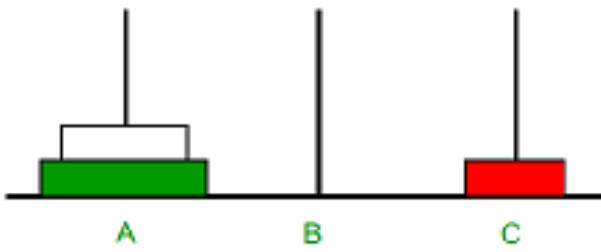
Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- 1) Only one disk can be moved at a time.
- 2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- 3) No disk may be placed on top of a smaller disk.

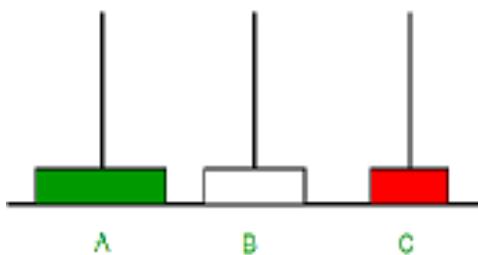
3 Disk



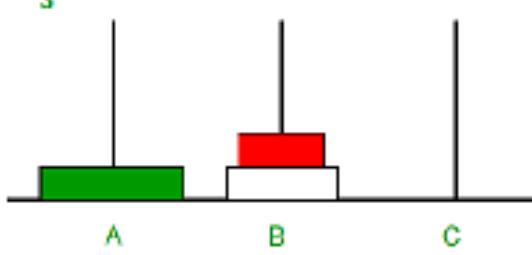
1



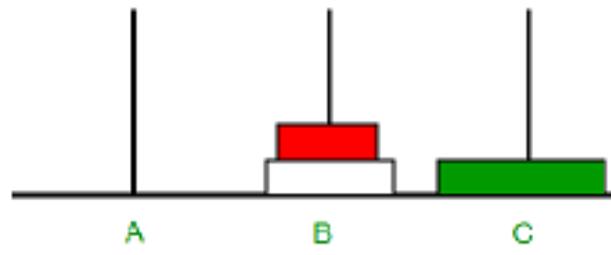
2



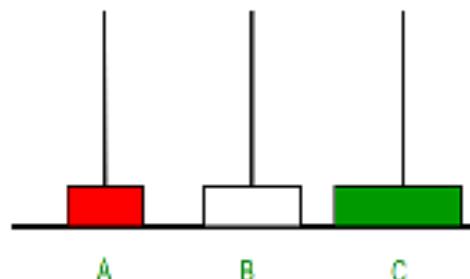
3



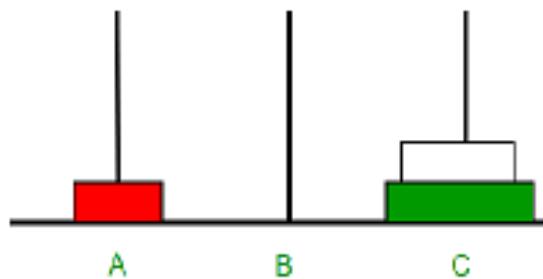
4



5



6



7



# Python Program of Tower of Hanoi

```
def TowerOfHanoi(n, source, destination, auxiliary):
    if n == 1:
        print("Move disk 1 from source", source, "to destination", destination)
        return
    TowerOfHanoi(n - 1, source, auxiliary, destination)
    print("Move disk", n, "from source", source, "to destination", destination)
    TowerOfHanoi(n - 1, auxiliary, destination, source)

# Driver code
n = 3
TowerOfHanoi(n, 'A', 'C', 'B')
# A, C, B are the name of rods
```

# Output

Move disk 1 from source A to destination C

Move disk 2 from source A to destination B

Move disk 1 from source C to destination B

Move disk 3 from source A to destination C

Move disk 1 from source B to destination A

Move disk 2 from source B to destination C

Move disk 1 from source A to destination C

# Python Iterators

- An iterator is an object that contains a countable number of values.
- An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
- Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods `__iter__()` and `__next__()`

**Return an iterator from a tuple, and print each value:**

```
mytuple = ("apple", "banana", "cherry")
```

```
myit = iter(mytuple)
```

```
print(next(myit))
```

```
print(next(myit))
```

```
print(next(myit))
```

**Strings are also iterable objects,  
containing a sequence of characters:**

```
mystr = "banana"
```

```
myit = iter(mystr)
```

```
print(next(myit))
```

# Create an Iterator

To create an object/class as an iterator you have to implement the methods `__iter__()` and `__next__()` to your object.

- The `__iter__()` method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself.
- The `__next__()` method also allows you to do operations, and must return the next item in the sequence.

```
class MyNumbers:  
    def __iter__(self):  
        self.a = 1  
        return self
```

```
    def __next__(self):  
        x = self.a  
        self.a += 1  
        return x
```

```
myclass = MyNumbers()
```

```
myiter = iter(myclass)
```

```
print(next(myiter))
```

1

2

3

4

5

# Search

# Python Program for Linear Search

A simple approach is to do linear search

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.

**Example:**

Find 'J'



# Code For Linear Search

```
def search(arr, x):  
    for i in range(len(arr)):  
        if arr[i] == x:  
            return i  
    return -1
```

NOTE: The time complexity of Linear Search is  $O(n)$

# Python Program for Binary Search

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half sub array after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

```
def binarySearch(arr, l, r, x):
    while l <= r:
        mid = (l + r) //2;
        # Check if x is present at mid
        if (arr[mid] == x):
            return mid
            # If x is greater, ignore left half
        elif (arr[mid] < x):
            l = mid + 1
            # If x is smaller, ignore right half
        else:
            r = mid - 1
            # If we reach here, then the element was not
            present
    return -1
```

```
arr = [2, 3, 4, 10, 40]
x = 10
length=len(arr) - 1
result = binarySearch(arr, 0, length, x)
```

# Sorting & Merging

# Selection Sort Algorithm

- Selection sort is conceptually the most simplest sorting algorithm. This algorithm will first find the **smallest** element in the array and swap it with the element in the **first** position, then it will find the **second smallest** element and swap it with the element in the **second** position, and it will keep on doing this until the entire array is sorted.
- It is called selection sort because it repeatedly **selects** the next-smallest element and swaps it into the right place.

# Algorithm

Step 1 – Set MIN to location 0

Step 2 – Search the minimum element in the list

Step 3 – Swap with value at location MIN

Step 4 – Increment MIN to point to next element

Step 5 – Repeat until list is sorted



For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.



So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in a loop:



We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.



After two iterations, two least values are positioned at the beginning in a sorted manner.

10

14

27

33

35

19

42

44

10

14

27

33

35

19

42

44

10

14

19

33

35

27

42

44

10

14

19

33

35

27

42

44

10

14

19

27

35

33

42

44

10

14

19

27

35

33

42

44

10

14

19

27

35

33

42

44

10

14

19

27

33

35

42

44

10

14

19

27

33

35

42

44

# Complexity Analysis of Selection Sort

- Selection Sort requires two nested for loops to complete itself.
- Hence for a given input size of  $n$ , following will be the time and space complexity for selection sort algorithm:
  - Worst Case Time Complexity [ Big-O ]:  $O(n^2)$
  - Best Case Time Complexity [Big-omega]:  $\Omega(n^2)$
  - Average Time Complexity [Big-theta]:  $\Theta(n^2)$
  - Space Complexity:  $O(1)$

```
A = [64, 25, 12, 22, 11]
```

```
for i in range(len(A)):
```

```
    min_idx = i
```

```
    for j in range(i+1, len(A)):
```

```
        if A[min_idx] > A[j]:
```

```
            min_idx = j
```

```
        A[i], A[min_idx] = A[min_idx], A[i]
```

```
print ("Sorted array")
```

```
for i in range(len(A)):
```

```
    print(A[i])
```

# Merge Sort Algorithm

- Merge Sort follows the rule of **Divide and Conquer** to sort a given set of numbers/elements, recursively, hence consuming less time.
- Merge sort runs in  $O(n * \log n)$  time in all the cases.
- Before jumping on to, how merge sort works and it's implementation, first lets understand what is the rule of **Divide and Conquer**?

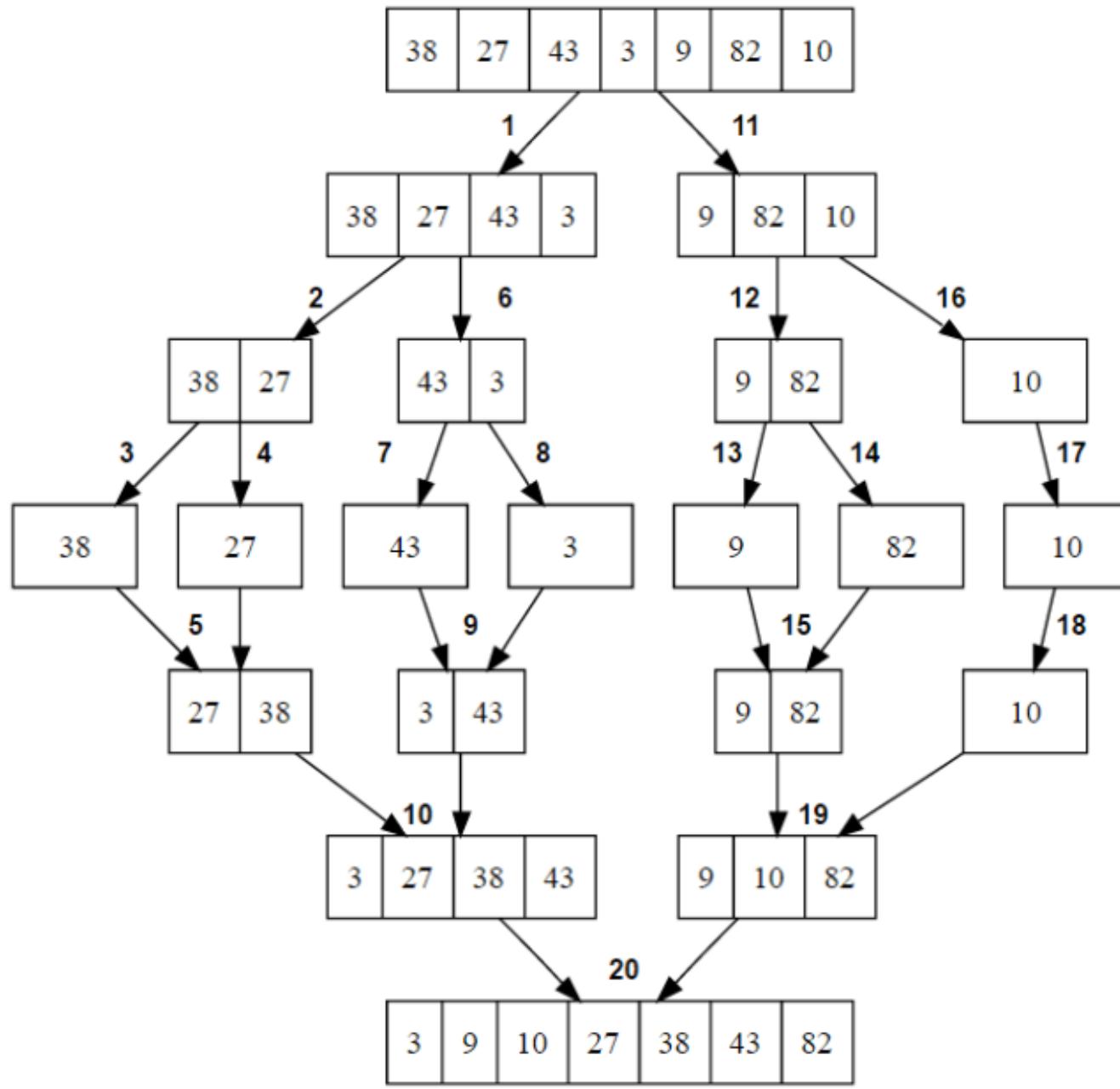
# Divide and Conquer

- If we can break a single big problem into smaller sub-problems, solve the smaller sub-problems and combine their solutions to find the solution for the original big problem, it becomes easier to solve the whole problem.
- In **Merge Sort**, the given unsorted array with  $n$  elements, is divided into  $n$  sub-arrays, each having **one** element, because a single element is always sorted in itself. Then, it repeatedly merges these sub-arrays, to produce new sorted sub-arrays, and in the end, one complete sorted array is produced.

The concept of Divide and Conquer involves three steps:

- **Divide** the problem into multiple small problems.
- **Conquer** the subproblems by solving them. The idea is to break down the problem into atomic subproblems, where they are actually solved.
- **Combine** the solutions of the subproblems to find the solution of the actual problem.

<https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/visualize/>



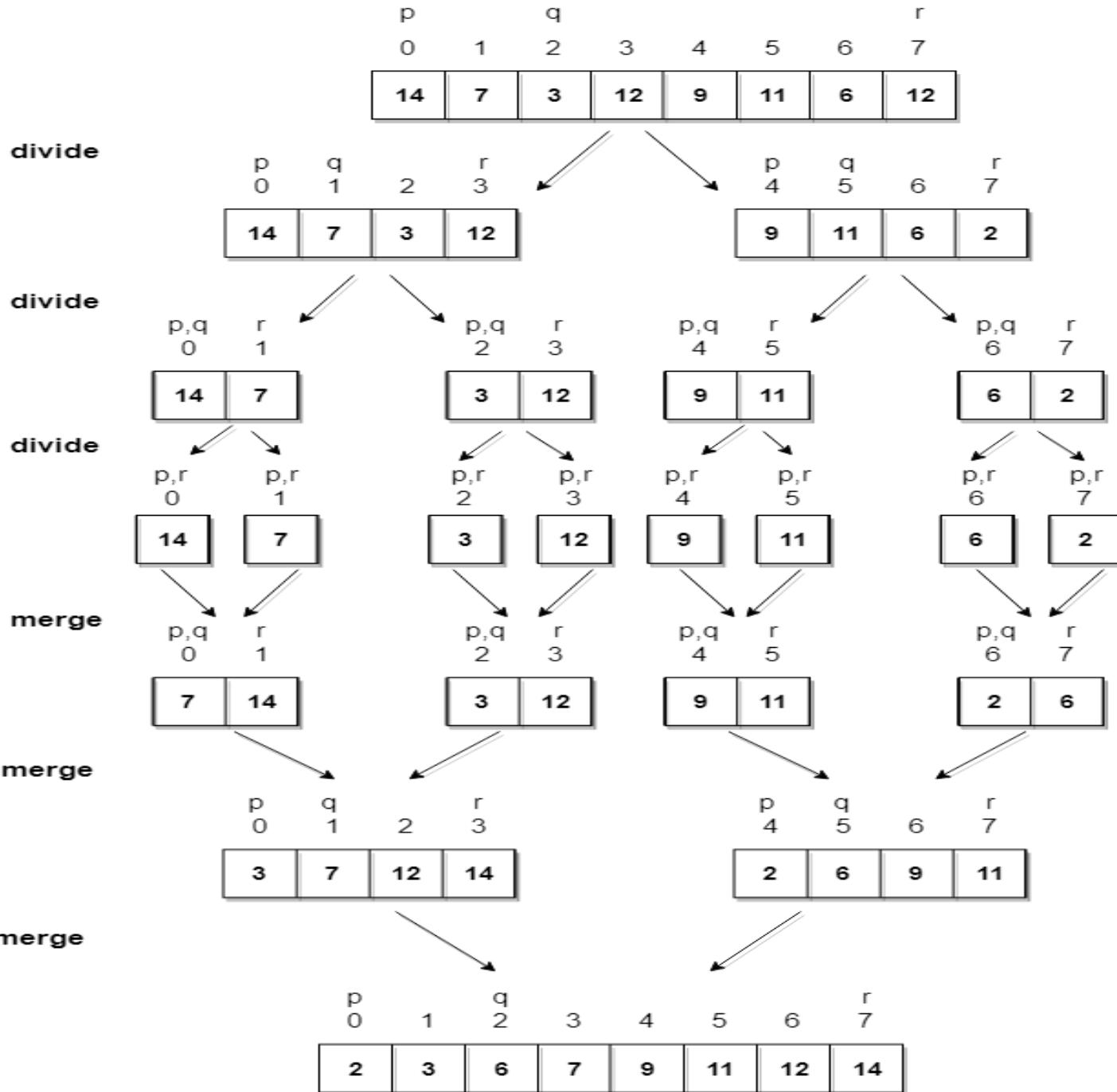
# Pseudo Code

MERGE-SORT( $A, p, r$ )

```
1  if  $p < r$ 
2     $q = \lfloor (p + r) / 2 \rfloor$ 
3    MERGE-SORT( $A, p, q$ )
4    MERGE-SORT( $A, q + 1, r$ )
5    MERGE( $A, p, q, r$ )
```

MERGE( $A, p, q, r$ )

```
1   $n_1 \leftarrow q - p + 1$ 
2   $n_2 \leftarrow r - q$ 
3  create arrays  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$ 
4  for  $i \leftarrow 1$  to  $n_1$ 
5    do  $L[i] \leftarrow A[p + i - 1]$ 
6  for  $j \leftarrow 1$  to  $n_2$ 
7    do  $R[j] \leftarrow A[q + j]$ 
8   $L[n_1 + 1] \leftarrow \infty$ 
9   $R[n_2 + 1] \leftarrow \infty$ 
10  $i \leftarrow 1$ 
11  $j \leftarrow 1$ 
12 for  $k \leftarrow p$  to  $r$ 
13   do if  $L[i] \leq R[j]$ 
14     then  $A[k] \leftarrow L[i]$ 
15        $i \leftarrow i + 1$ 
16     else  $A[k] \leftarrow R[j]$ 
17        $j \leftarrow j + 1$ 
```



# Merge List

```
test_list1 = [3, 4, 7, 8, 101]
test_list2 = [1, 5, 65, 95, 111,123,143,197]
size_1 = len(test_list1)
size_2 = len(test_list2)
res = []
i, j = 0, 0
while i < size_1 and j < size_2:
    if test_list1[i] < test_list2[j]:
        res.append(test_list1[i])
        i += 1
    else:
        res.append(test_list2[j])
        j += 1
res = res + test_list1[i:] + test_list2[j:]
print ("The combined sorted list is : " ,res)
```

# Merge Sort

```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * (n1)
    R = [0] * (n2)
    for i in range(0, n1):
        L[i] = arr[l + i]
    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
    i = 0 # Initial index of first subarray
    j = 0 # Initial index of second subarray
    k = l # Initial index of merged subarray
```

```
while i < n1 and j < n2:  
    if L[i] <= R[j]:  
        arr[k] = L[i]  
        i += 1  
    else:  
        arr[k] = R[j]  
        j += 1  
    k += 1  
while i < n1:  
    arr[k] = L[i]  
    i += 1  
    k += 1  
while j < n2:  
    arr[k] = R[j]  
    j += 1  
    k += 1
```

```
def mergeSort(arr, l, r):
    if l < r:
        m = (l + (r - 1)) // 2
        mergeSort(arr, l, m)
        mergeSort(arr, m + 1, r)
        merge(arr, l, m, r)
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print("Given array is")
for i in range(n):
    print("%d" % arr[i]),
mergeSort(arr, 0, n - 1)
print("\n\nSorted array is")
for i in range(n):
    print("%d" % arr[i])
```

**United College of Engineering and Research, Prayagraj**  
**Department of Computer Science and Engineering**  
**Lecture Plan**

Name of Course	<b>PYTHON PROGRAMMING</b>				
AKTU Course Code	<b>(KNC-302)</b>				
Branch	<b>Computer Science</b>				
Semester	<b>3</b>				
Section	<b>A</b>				
Total Number of Students	<b>68</b>				
Name of Faculty	<b>Mr. Abhishek Kesharwani</b>				
Number of Lecture Proposed	<b>34</b>				

S. No	Unit No	Topic	CO	No of Lectures Required	No of Student present	Actual Date of Completion	T L M	Signature of HOD
1	1	Python IDE Interacting with Python Programs	1	1				
2		Elements of Python Type Conversion		1				
3		Basics: Expressions Assignment Statement		1				
4		Arithmetic Operators		1				
5		Operator Precedence		1				
6		Boolean Expression		1				
<b>No of Lectures Required to complete Unit 1</b>				<b>6</b>		<b>No of Lectures Taken:</b>		
7	2	Conditional statement in Python if-else statement Nested-if Statement		1				
8		Elif statement in Python		1				
9		Expression Evaluation		1				
10		Loops While loop For Loop		1				
11		Break and Continue		1				
<b>No of Lectures Required to complete Unit 2</b>				<b>5</b>		<b>No of Lectures Taken:</b>		

12	3	Function	3	1				
13		Execution of a Function						
14		Keyword and Default Arguments						
15		Scope Rules.						
16		Strings : Length of the string						
17		Different string function						
18		Indexing and Slicing of Strings.						
19		Python Data Structure: Tuples						
20		Unpacking Sequences, Lists						
21		Mutable Sequences, List Comprehension						
22		Sets, Dictionaries						
23		Lambda Expressions		1				
<b>No of Lectures Required to complete Unit 3</b>				9	<b>No of Lectures Taken:</b>			
24	4	Sieve of Eratosthenes: generate prime number	4	2				
25		File I/O: File input and output operations in Python Programming		1				
26		Exceptions and Assertions		1				
27		Modules: Introduction, Importing Modules		1				
28		Abstract Data Types in Python		1				
29		Classes: Class definition		1				
30		other operations in the classes		1				
31		Class Example, Inheritance and OOP.		1				
<b>No of Lectures Required to complete Unit 4</b>				9	<b>No of Lectures Taken:</b>			
32	5	Recursive Fibonacci		1				
33		Tower of Hanoi		1				
34		Search: Linear & Binary Search		1				
35		Sorting & Merging: Selection Sort		1				
36		Merge List & Merge Sort		1				
<b>No of Lectures Required to complete Unit 5</b>				5	<b>No of Lectures Taken:</b>			

Teaching Learning Methods					
TLM1	Chalk and Talk	TLM4	Problem solving	TLM7	GD
TLM2	PPT	TLM5	Programming	TLM8	Case Study
TLM3	Tutorial	TLM6	Lab Demo	TLM9	Seminar

Text Books & References					
1	Allen B. Downey, "Think Python: How to Think Like a Computer Scientist", 2nd edition, Updated for Python 3, Shroff/O'Reilly Publishers, 2016 ( <a href="http://greenteapress.com/wp/thinkpython/">http://greenteapress.com/wp/thinkpython/</a> )				
2	Guido van Rossum and Fred L. Drake Jr, —An Introduction to Python – Revised and updated for Python 3.2, Network Theory Ltd., 2011.				
3	John V Guttag, —Introduction to Computation and Programming Using Python“, Revised and expanded Edition, MIT Press , 2013				
4	Robert Sedgewick, Kevin Wayne, Robert Dondero, —Introduction to Programming in Python: An Interdisciplinary Approach, Pearson India Education Services Pvt. Ltd., 2016.				
5	Timothy A. Budd, —Exploring Python®, Mc-Graw Hill Education (India) Private Ltd., 2015.				
6	Kenneth A. Lambert, —Fundamentals of Python: First Programs®, CENGAGE Learning, 2012.				
7	Charles Dierbach, —Introduction to Computer Science using Python: A Computational ProblemSolving Focus, Wiley India Edition, 2013.				

PYTHON PROGRAMMING					
Course Outcome ( CO )			Bloom's Knowledge Level (KL)		
<b>At the end of course , the student will be able to understand</b>					
CO 1	To read and write simple Python programs.				K <sub>1</sub> , K <sub>2</sub>
CO 2	To develop Python programs with conditionals and loops.				K <sub>2</sub> , K <sub>4</sub>
CO 3	To define Python functions and to use Python data structures — lists, tuples, dictionaries				K <sub>3</sub>
CO 4	To do input/output with files in Python				K <sub>2</sub>
CO 5	To do searching ,sorting and merging in Python				K <sub>2</sub> , K <sub>4</sub>

Faculty Instructor

Course Coordinator

Lecture Plan Incharge

Programme Coordinator

Head of Department

**B. TECH.**  
**(SEM III) THEORY EXAMINATION 2019-20**  
**PYTHON PROGRAMMING**

*Time: 3 Hours*

Total Marks: 100

**Note:** 1. Attempt all Sections. If require any missing data; then choose suitably.

## SECTION A

- 1.** Attempt *all* questions in brief. **2 x 10 = 20**

Qno.	Question	Marks	CO
a.	What is the difference between list and tuples in Python?	2	CO3
b.	In some languages, every statement ends with a semi-colon (;). What happens if you put a semi-colon at the end of a Python statement?	2	CO2
c.	Mention five benefits of using Python.	2	CO4
d.	How is Python an interpreted language?	2	CO2
e.	What type of language is python?	2	CO1
f.	What are local variables and global variables in Python?	2	CO1
g.	What is the difference between Python Arrays and lists?	2	CO3
h.	Define ADTinterface.	2	CO4
i.	Define floor division with example.	2	CO5
j.	Differentiate Fruitful functions and void functions.	2	CO3

## SECTION B

- 2. Attempt any *three* of the following:** **3 x 10 = 30**

Qno.	Question	Marks	CO
a.	Explain iterator. Write a program to demonstrate the tower of Hanoi using function.	10	CO5
b.	Discuss function in python with its parts and scope. Explain with example. (Take Simple calculator with add, subtract, Division and Multiplication).	10	CO3
c.	Discuss ADT in python.How to define ADT? Write code for a student information.	10	CO4
d.	Explain all the Conditional statement in Python using small code example.	10	CO2
e.	What is Python? How Python is interpreted? What are the tools that help to find bugs or perform static analysis? What are Python decorators?	10	CO1

### SECTION C

- 3. Attempt any *one* part of the following:** **1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Write short notes with example: The Programming Cycle for Python, Elements of Python, Type Conversion in Python, Operator Precedence, and Boolean Expression.	10	CO1
b.	How memory is managed in Python? Explain PEP 8. Write a Python program to print even length words in a string.	10	CO1

**4. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Explain Expression Evaluation & Float Representation with example. Write a Python Program for How to check if a given number is Fibonacci number.	10	CO2
b.	Explain the purpose and working of loops. Discuss Break and continue with example. Write a Python program to convert time from 12 hour to 24-hour format.	10	CO2

**5. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Explain higher order function with respect to lambda expression. Write a Python code to Count occurrences of an element in a list.	10	CO3
b.	Explain Unpacking Sequences, Mutable Sequences, and List Comprehension with example. Write a program to sort list of dictionaries by values in Python – Using lambda function.	10	CO3

**6. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Discuss File I/O in python. How to perform open, read,write, and close into a file? Write a Python program to read a file line-by-line store it into a variable.	10	CO4
b.	Discuss Exceptions and Assertions in python. How to handle Exceptions with Try-Finally? Explain 5 Built-in Exceptions with example.	10	CO4

**7. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Discuss and differentiate Iterators & Recursion. Write a program for Recursive Fibonacci series.	10	CO5
b.	Discuss Sorting & Merging. Explain different types of sorting with example. Write a Python Program for Sieve of Eratosthenes.	10	CO5

**B. TECH.**  
**(SEM III) THEORY EXAMINATION 2019-20**  
**PYTHON PROGRAMMING**

**Time: 3 Hours****Total Marks: 100****Note:** 1. Attempt all Sections. If require any missing data; then choose suitably.**SECTION A****1. Attempt all questions in brief.****2 x 10 = 20**

Qno.	Question	Marks	CO
a.	What is the difference between list and tuples in Python?	2	CO3
b.	In some languages, every statement ends with a semi-colon (;). What happens if you put a semi-colon at the end of a Python statement?	2	CO2
c.	Mention five benefits of using Python.	2	CO4
d.	How is Python an interpreted language?	2	CO2
e.	What type of language is python?	2	CO1
f.	What are local variables and global variables in Python?	2	CO1
g.	What is the difference between Python Arrays and lists?	2	CO3
h.	Define ADTinterface.	2	CO4
i.	Define floor division with example.	2	CO5
j.	Differentiate Fruity functions and void functions.	2	CO3

**SECTION B****2. Attempt any three of the following:****3 x 10 = 30**

Qno.	Question	Marks	CO
a.	Explain iterator. Write a program to demonstrate the tower of Hanoi using function.	10	CO5
b.	Discuss function in python with its parts and scope. Explain with example. (Take Simple calculator with add, subtract, Division and Multiplication).	10	CO3
c.	Discuss ADT in python. How to define ADT? Write code for a student information.	10	CO4
d.	Explain all the Conditional statement in Python using small code example.	10	CO2
e.	What is Python? How Python is interpreted? What are the tools that help to find bugs or perform static analysis? What are Python decorators?	10	CO1

**SECTION C****3.. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Write short notes with example: The Programming Cycle for Python, Elements of Python, Type Conversion in Python, Operator Precedence, and Boolean Expression.	10	CO1
b.	How memory is managed in Python? Explain PEP 8. Write a Python program to print even length words in a string.	10	CO1

**4. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Explain Expression Evaluation & Float Representation with example. Write a Python Program for How to check if a given number is Fibonacci number.	10	CO2
b.	Explain the purpose and working of loops. Discuss Break and continue with example. Write a Python program to convert time from 12 hour to 24-hour format.	10	CO2

**5. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Explain higher order function with respect to lambda expression. Write a Python code to Count occurrences of an element in a list.	10	CO3
b.	Explain Unpacking Sequences, Mutable Sequences, and List Comprehension with example. Write a program to sort list of dictionaries by values in Python – Using lambda function.	10	CO3

**6. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Discuss File I/O in python. How to perform open, read, write, and close into a file? Write a Python program to read a file line-by-line store it into a variable.	10	CO4
b.	Discuss Exceptions and Assertions in python. How to handle Exceptions with Try-Finally? Explain 5 Built-in Exceptions with example.	10	CO4

**7. Attempt any one part of the following:****1 x 10 = 10**

Qno.	Question	Marks	CO
a.	Discuss and differentiate Iterators & Recursion. Write a program for Recursive Fibonacci series.	10	CO5
b.	Discuss Sorting & Merging. Explain different types of sorting with example. Write a Python Program for Sieve of Eratosthenes.	10	CO5

Roll No: 

--	--	--	--	--	--	--	--	--	--

**B.TECH**  
**(SEM III) THEORY EXAMINATION 2020-21**  
**PYTHON PROGRAMMING**

**Time: 3 Hours****Total Marks: 100****Note:** 1. Attempt all Sections. If require any missing data; then choose suitably.**SECTION A****1. Attempt all questions in brief.**

Q no.	Question	Marks	CO
a.	What is the use of “raise” statement? Describe with an example.	2	5
b.	Write a recursive Python function “rprint” to print all elements in a list in reverse. <code>rprint([]) prints nothing</code> <code>rprint([1,2,3]) prints 3 2 1</code>	2	3
c.	Describe the behavior of “range (s, e)” in Python.	2	3
d.	Explain the use of “with” construct in Python with an example program.	2	5
e.	Which of the following statements produce an error in Python? <code>x, y, z = 1,2,3 # s1</code> <code>a, b = 4,5,6 # s2</code> <code>u = 7,8,9 # s3</code> (List all the statements that have error.)	2	1
f.	Explain the role of precedence with an example.	2	1
g.	How do you read an input from a user in Python to be used as an integer in the rest of the program? Explain with an example.	2	5
h.	Consider the program:  <code>x = ['12', 'he l o', 456]</code> <code>x[0] *= 3</code> <code>x[1][1] = 'bye'</code>  Explain why this program generates an error.	2	1
i.	What is the output of the following program?  <code>(lambda x, y : y - 2*x)(1, 11)</code>	2	5
j.	Explain the use of <code>lt</code> function in a class Python?	2	5

Roll No: 

--	--	--	--	--	--	--	--	--	--

**SECTION B****2. Attempt any three of the following:**

Q no.	Question	Marks	CO
a.	<p>Write a Python function <code>removekth(s, k)</code> that takes as input a string <code>s</code> and an integer <code>k&gt;=0</code> and removes the character at index <code>k</code>. If <code>k</code> is beyond the length of <code>s</code>, the whole of <code>s</code> is returned. For example,</p> <pre>removekth("PYTHON", 1) returns "PTHON" removekth("PYTHON", 3) returns "PYTON" removekth("PYTHON", 0) returns "YTHON" removekth("PYTHON", 20) returns "PYTHON"</pre>	10	5
b.	<p>Write a Python function <code>average</code> to compute the average of a list of numbers. The function must use <code>try-except</code> to handle the case where the input list is empty. Further, in that case the average for the empty list should be set to 0.0 using the <code>except</code> block.</p>	10	3
c.	<p>Describe the differences between a linear search and a binary search?</p>	10	5
d.	<p>Write a function <code>lessthan(lst, k)</code> to return list of numbers less than <code>k</code> from a list <code>lst</code>. The function must use list comprehension. Example: <code>lessthan([1, -2, 0, 5, -3], 0)</code> returns <code>[-2, -3]</code></p>	10	4
e.	<p>Write a program <code>factors(N)</code> that returns a list of all positive divisors of <code>N</code> (<code>N&gt;=1</code>). For example:</p> <pre>factors(6) returns [1, 2, 3, 6] factors(1) returns [1] factors(13) returns [1, 13]</pre>	10	2

**SECTION C****3. Attempt any one part of the following:**

Q no.	Question	Marks	CO
a.	<p>How can you create Python file that can be imported as a library as well as run as a standalone script?</p>	10	5
b.	<p>Describe the difference between  <code>import library</code>  and  <code>from library import *</code>  when used in a python program. Here <code>library</code> is some python library .</p>	10	5

Roll No: 

--	--	--	--	--	--	--	--	--	--

**4. Attempt any *one* part of the following:**

Q no.	Question	Marks	CO
a.	<p>Write a function <code>makePairs</code> that takes as input two lists of equal length and returns a single list of same length where k-th element is the pair of k-th elements from the input lists. For example,</p> <pre>makePairs([1,3,5,7],[2,4,6,8])           returns [(1,2),(3,4),(5,6),(7,8)] makePairs([],[])           returns []</pre>	10	2
b.	Show an example where both <b>Keyword</b> arguments and <b>Default</b> arguments are used for the same function in a call. Show both the definition of the function and its call.	10	4

**5. Attempt any *one* part of the following:**

Q no.	Question	Marks	CO
a.	Explain why Python is considered an interpreted language.	10	1
b.	<p>What is short circuit evaluation? What is printed by the following Python program?</p> <pre>a = 0 b = 2 c = 3 x = c or a print(x)</pre>	10	1

Roll No: 

--	--	--	--	--	--	--	--	--	--

**6. Attempt any one part of the following:**

Q no.	Question	Marks	CO
a.	<p>Write a Python program, triangle(N), that prints a right triangle having base and height consisting of N * symbols as shown in these examples:</p> <pre>triangle(3) prints: * ** ***</pre> <p>triangle(5) prints: * ** *** **** *****</p>	10	3
b.	<p>Write a Python program, countSquares(N), that returns the count of perfect squares less than or equal to N (N&gt;1). For example:</p> <pre>countSquares(1)      returns 1                       # Only 1 is a perfect square &lt;= 1 countSquares(5)      returns 2                       # 1, 4 are perfect squares &lt;= 5 countSquares(55)     returns 7                       # 1, 4, 9, 16, 25, 36, 49 &lt;= 55</pre>	10	4

**7. Attempt any one part of the following:**

Q no.	Question	Marks	CO
a.	<p>Write a Python function, alternating(lst), that takes as argument a sequence lst. The function returns True if the elements in lst are <b>alternately odd and even, starting with an even number</b>. Otherwise it returns False. For example:</p> <pre>alternating([10, 9, 9, 6]) returns False alternating([10, 15, 8]) returns True alternating([10]) returns True alternating([]) returns True alternating([15, 10, 9]) returns False</pre>	10	4
b.	<p>Write a Python function, searchMany(s, x, k), that takes as argument a sequence s and integers x, k (k&gt;0). The function returns True if there are <i>at most k occurrences of x in s</i>. Otherwise it returns False. For example:</p> <pre>searchMany([10, 17, 15, 12], 15, 1) returns True searchMany([10, 12, 12, 12], 12, 2) returns False searchMany([10, 12, 15, 11], 17, 18) returns True</pre>	10	3



PAPER ID-411310

Printed Page: 1 of 2

Subject Code: KNC302

Roll No: 

--	--	--	--	--	--	--	--	--	--	--	--

**BTECH**  
**(SEM III) THEORY EXAMINATION 2021-22**  
**PYTHON PROGRAMMING**

**Time: 3 Hours****Total Marks: 50****Note:** Attempt all Sections. If you require any missing data, then choose suitably.**SECTION A**

- 1.
- Attempt all questions in brief.**

**2\*5 = 10**

Qno	Questions	CO
(a)	Explain the Programming Cycle for Python in detail.	1
(b)	What will be the output of the following Python code? <pre>i = 0 while i&lt; 3:     print(i) i += 1 else:     print(0)</pre>	2
(c)	What will be the output of the following Python code? <pre>def cube(x):     return x * x * x x = cube(3) print x</pre>	3
(d)	How do we define an Interface for an ADT?	4
(e)	How do you perform a search in Python?	5

**SECTION B**

- 2.
- Attempt any three of the following:**

**5\*3 = 15**

Qno	Questions	CO
(a)	What do you mean by Python IDE? Explain in detail.	1
(b)	How can you randomize the items of a list in place in Python?	2
(c)	Explain Tuples and Unpacking Sequences in Python Data Structure.	3
(d)	What are File input and output operations in Python Programming?	4
(e)	Solve the Tower of Hanoi problem for n= 3 disk and show all the steps.	5

**SECTION C**

- 3.
- Attempt any one part of the following:**

**5\*1 = 5**

Qno	Questions	CO
(a)	Write a program in Python to execute the Selection sort algorithm.	5
(b)	Explain why python is considered an interpreted language.	1

- 4.
- Attempt any one part of the following:**

**5 \*1 = 5**

Qno	Questions	CO
(a)	Write a Python program to construct the following pattern, using a nested for loop.  * * * * * * * * * * * * * * *	2



PAPER ID-411310

Printed Page: 2 of 2

Subject Code: KNC302

Roll No:

--	--	--	--	--	--	--	--	--	--	--	--

**BTECH**  
**(SEM III) THEORY EXAMINATION 2021-22**  
**PYTHON PROGRAMMING**

---

	* * * *	
	* * *	
	* *	
	*	
(b)	Write a program to produce Fibonacci series in Python.	4

5. **Attempt any one part of the following:** 5\*1 = 5

Qno	Questions	CO
(a)	Write a Python program to change a given string to a new string where the first and last chars have been exchanged.	3
(b)	Write a Python program to add an item in a tuple.	3

6. **Attempt any one part of the following:** 5\*1 = 5

Qno	Questions	CO
(a)	How to create and import a module in Python?	4
(b)	Explain the algorithm Sieve of Eratosthenes used in Python Programming.	4

7. **Attempt any one part of the following:** 5\*1 = 5

Qno	Questions	CO
(a)	Write a Recursive function in python BinarySearch(Arr,l,R,X) to search the given element X to be searched from the List Arr having R elements, where l represent slower bound and R represents the upper bound.	5
(b)	Explain the terms Merge List and Merge Sort in Python Programming.	5

### **Python Important Questions for AKTU 4<sup>th</sup> Semester**

#### **Unit-I**

**I Introduction:** The Programming Cycle for Python, Python IDE, Interacting with Python Programs, Elements of Python, Type Conversion.

**Basics:** Expressions, Assignment Statement, Arithmetic Operators, Operator Precedence, Boolean Expression.

1. What type of language is python?
2. What are local variables and global variables in Python?
3. In some languages, every statement ends with a semi-colon (;). What happens if you put a semi-colon at the end of a Python statement?
4. Define floor division with example.
5. Mention five benefits of using Python.
6. How is Python an interpreted language?
7. Describe Arithmetic Operators, Assignment Operators, Comparison Operators, Logical Operators and Bitwise Operators in detail with examples.
8. Explain the Identifiers, Keywords, Statements, Expressions, and Variables in Python programming language with examples.
9. Explain the basic data types available in Python with examples.
10. What is operator Precedence?

#### **Unit-II**

**II Conditionals:** Conditional statement in Python (if-else statement, its working and execution), Nested-if statement and Elif statement in Python, Expression Evaluation & Float Representation.

**Loops:** Purpose and working of loops , While loop including its working, For Loop , Nested Loops , Break and Continue.

1. Explain the purpose and working of loops.
2. Discuss Break and continue with example.
3. What is break, continue and pass in Python?
4. Explain Expression Evaluation & Float Representation with example.
5. Write a program factors(N) that returns a list of all positive divisors of N (N>=1). For example: factors(6) returns [1,2,3,6] factors(1) returns [1] factors(13) returns [1,13]
6. Write a function makePairs that takes as input two lists of equal length and returns a single list of same length where k-th element is the pair of k-th elements from the input lists. For example, makePairs([1,3,5,7],[2,4,6,8]) returns [(1,2),(3,4),(5,6),(7,8)] makePairs([],[]) returns []
7. Write Python program to swap two numbers without using Intermediate/Temporary variables. Prompt the user for input.
8. Write a program that accepts a sentence and calculate the number of digits, uppercase and lowercase letters.
9. Write Python code to sort a sequence of names according to their alphabetical order without using sort() function.
10. Write Python Program to reverse a number and find the Sum of digits in the reversed number. Prompt the user for input.
11. Write Python code to check if a given year is a leap year or not.
12. Write Python program to find the GCD of two positive numbers.
13. Write Python code to determine whether the given string is a Palindrome or not using slicing.
14. Write Python program to add two matrices and find the transpose of the resultant matrix.
15. Input five integers (+ve and -ve). Write Python code to find the sum of negative numbers, positive numbers and print them. Also, find the average of all the numbers and numbers above average.

### Unit-III

**III Function:** Parts of A Function, Execution of A Function, Keyword and Default Arguments, Scope Rules.

**Strings:** Length of the string and perform Concatenation and Repeat operations in it. Indexing and Slicing of Strings.

**Python Data Structure :** Tuples , Unpacking Sequences , Lists , Mutable Sequences , List Comprehension , Sets , Dictionaries

**Higher Order Functions:** Treat functions as first class Objects , Lambda Expressions

1. Write a Python function to find the Max of three numbers.

2. Write a Python function to sum all the numbers in a list.

Sample List: (8, 2, 3, 0, 7)

Expected Output: 20

3. Write a Python function to multiply all the numbers in a list.

Sample List: (8, 2, 3, -1, 7)

Expected Output: -336

4. Write a Python program to reverse a string.

Sample String: "1234abcd"

Expected Output: "dcba4321"

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

6. Write Python code to find Mean, Variance and Standard Deviation for a list of numbers.

7. Write a Python function to check whether a number falls in a given range.

8. Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.

Sample String: 'The quick Brown Fox'

Expected Output:

No. of Upper case characters: 3

No. of Lower case Characters: 12

9. Write a Python function that takes a list and returns a new list with unique elements of the first list.

Sample List: [1, 2, 3, 3, 3, 3, 4, 5]

Unique List: [1, 2, 3, 4, 5]

10. Write a Python function that takes a number as a parameter and check the number is prime or not.

11. Note: A prime number (or a prime) is a natural number greater than 1 and that has no positive divisors other than 1 and itself.

12. Write a Python program to print the even numbers from a given list.

Sample List : [1, 2, 3, 4, 5, 6, 7, 8, 9]

Expected Result : [2, 4, 6, 8]

13. Write a Python function to check whether a number is perfect or not.

Example: The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and  $1 + 2 + 3 = 6$ .

Equivalently, the number 6 is equal to half the sum of all its positive divisors:  $(1 + 2 + 3 + 6) / 2 = 6$ . The next perfect number is  $28 = 1 + 2 + 4 + 7 + 14$ . This is followed by the perfect numbers 496 and 8128.

14. Write a Python function that checks whether a passed string is palindrome or not.

15. Explain Unpacking Sequences, Mutable Sequences, and List Comprehension with example.

16. Write a Python program to convert time from 12 hour to 24-hour format.

17. Write a Python code to Count Occurrences of an element in a list.

18. Explain Unpacking Sequences, Mutable Sequences, and List Comprehension with example.

19. Write a program to sort list of dictionaries by values in Python – Using lambda function.

20. Discuss the relation between tuples and lists, tuples and dictionaries in detail.

21. Illustrate the following Set methods with an example.

a) intersection()

b) union()

c) issubset()

d) difference ()

e) update()

f) discard()

22. Write a function less\_than(lst, k) to return list of numbers less than k from a list lst. The function must use list comprehension. Example: lessthan([1, -2, 0, 5, -3], 0) returns [-2, -3]
23. Explain the use of join () and split () string methods with examples.
24. Describe why strings are immutable with an example.
25. Show an example where both Keyword arguments and Default arguments are used for the same function in a call. Show both the definition of the function and its call.
26. Write Python program to count the total number of vowels, consonants and blanks in a String.
27. Explain higher order function with respect to lambda expression.

## Unit-IV

**IV Sieve of Eratosthenes:** generate prime numbers with the help of an algorithm given by the Greek Mathematician named Eratosthenes, whose algorithm is known as Sieve of Eratosthenes.

**File I/O :** File input and output operations in Python Programming

**Exceptions and Assertions**

**Modules :** Introduction , Importing Modules ,

**Abstract Data Types:** Abstract data types and ADT interface in Python Programming.

**Classes:** Class definition and other operations in the classes, Special Methods (such as `_init_`, `_str_`, comparison methods and Arithmetic methods etc.), Class Example, Inheritance, Inheritance and OOP.

1. Explain and Write a Python Program for Sieve of Eratosthenes.
2. Discuss File I/O in python. How to perform open, read, write, and close into a file?
3. How can you create Python file that can be imported as a library as well as run as a standalone script?
4. Describe the difference between import library and from library import \* when used in a python program. Here library is some python library.
5. What are modules and packages in Python?
6. Discuss Exceptions and Assertions in python.
7. Write a Python program to read a file line-by-line store it into a variable.
8. How to handle Exceptions with Try-Finally? Explain 5 Built-in Exceptions with example.
9. Write a Python program to read an entire text file. -
10. Write a Python program to read first n lines of a file. -
11. Write a Python program to append text to a file and display the text.
12. Write a Python program to read last n lines of a file.
13. Write a Python program to read a file line by line and store it into a list.
14. Write a Python program to read a file line-by-line store it into a variable.
15. Write a Python program to read a file line-by-line store it into an array.
16. Write a Python program that takes a text file as input and returns the number of words of a given text file. -
17. Write a Python program to count the frequency of words in a file.

## Unit-V

**V Iterators & Recursion:** Recursive Fibonacci , Tower Of Hanoi

**Search :** Simple Search and Estimating Search Time , Binary Search and Estimating Binary Search Time

**Sorting & Merging:** Selection Sort , Merge List , Merge Sort , Higher Order Sort

1. How do you create a class in Python?
2. How does inheritance work in python? Explain it with an example.
3. Discuss and differentiate Iterators and Recursion.
4. Write a program for Recursive Fibonacci series.
5. Discuss Sorting & Merging. Explain different types of sorting with example.
6. Write a Python function average to compute the average of a list of numbers. The function must use try-except to handle the case where the input list is empty. Further, in that case the average for the empty list should be set to 0.0 using the except block.
7. Describe the differences between a linear search and a binary search?