Programming In Python
CIE-332T

Introduction to Python

- Python is a versatile and high-level programming language that is known for its readability and simplicity.

or

- Python is a high-level, interpreted, and general-purpose programming language. It emphasizes code readability and allows developers to express concepts in fewer lines of code than languages like C++ or Java.

- Python is one of the most popular and widely used programming language, used for set of tasks including console based, GUI based, web programming and data analysis.
- It was created by Guido van Rossum and first released in 1991.

Compilers and Interpreters

Programs written in high-level languages must be translated into machine language to be executed

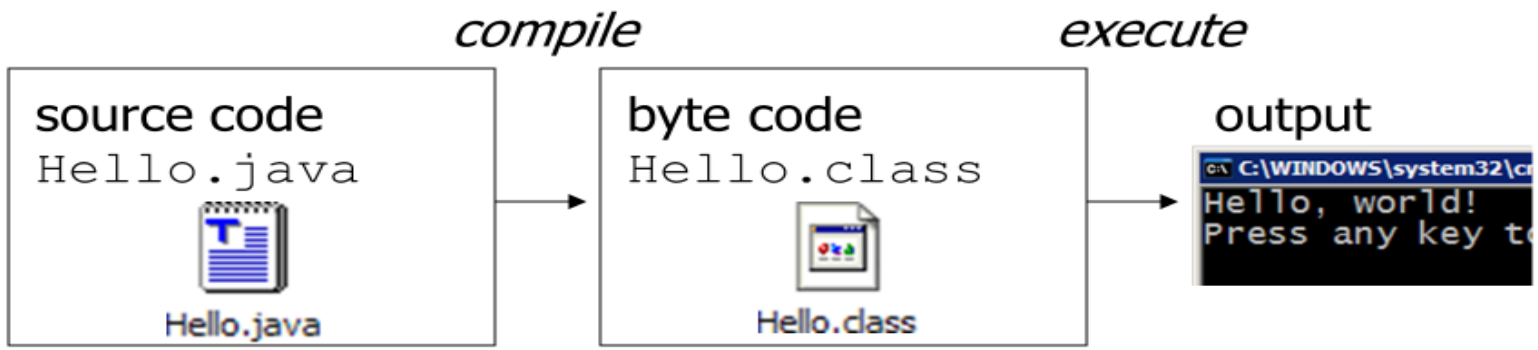
- **Compiler:** A compiler translates the entire source code of a program into machine code or an intermediate code in one go.
- The resulting compiled code is stored in a separate file (e.g., an executable file), which can be executed independently of the original source code.

Interpreter:

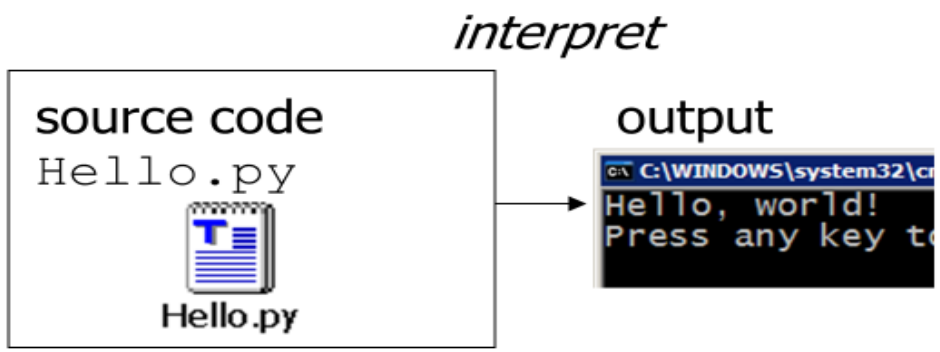
- An interpreter translates the source code line-by-line or statement-by-statement during the program's execution.
- No separate compiled file is generated; the interpreter reads and executes the source code directly.

Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



- Python is instead directly *interpreted* into machine instructions.



Feature of Python

- **Easy to Learn and Use:** Python is easy to learn and use compared with other programming languages. It is developer-friendly and high level programming language.
- **Interpreted Language:** Python is an interpreted language because no need of compilation. This makes debugging easy and thus suitable for beginners.
- **Platform Independent 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.
- **Free and Open Source:** The Python interpreter is developed under an open-source license, making it free to install, use, and distribute.
- **Object-Oriented Language**
- Python supports object oriented language and concepts of classes and objects come into existence.
- **GUI Programming Support**
- Graphical user interfaces can be developed using Python.

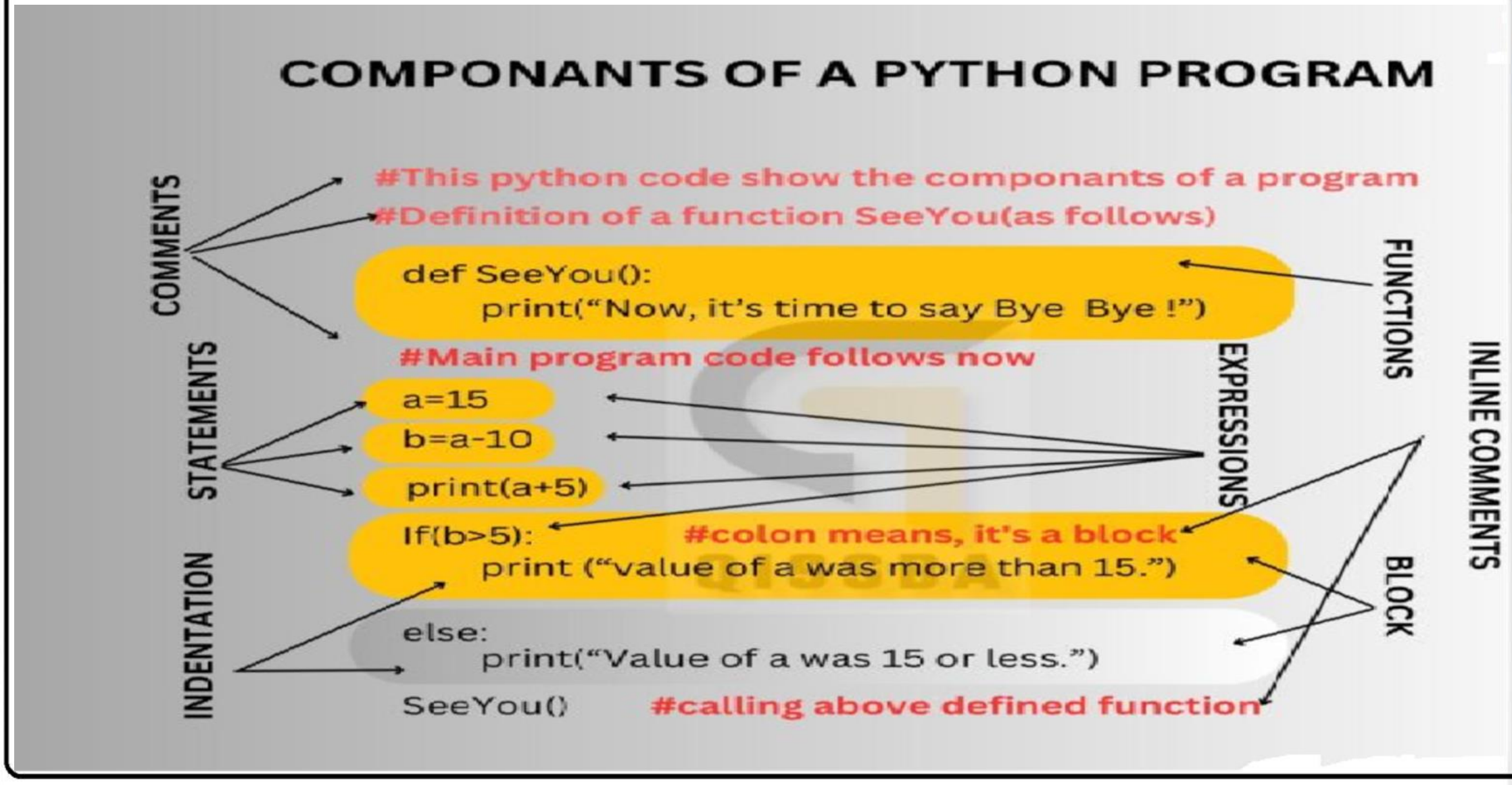
Python Applications

Python is a general purpose programming language that makes it applicable in almost all domains of software development. Python as a whole can be used to develop any type of applications.

Here, we are providing some specific application areas where python can be applied.

- **Web Applications**
- **Desktop GUI Applications**
- **Software Development**
- **Scientific and Numeric**
- **Business Applications**
- **Console Based Application**
- **Audio or Video based Applications**

Basic Structure of a Python Program!



Program structure

Basic Structure of a Python Program!

Expressions: An expression is any legal combination of symbols that represents a value.

An expression represents something which python evaluates and which produces a value.

E.g. 10, x + 5

Statements: A statement is a programming instruction that does something i.e. some action takes place.

A statement executes and may or may not results in a value.

e.g. print(x + 2), y = x + 5, x = 10

Comments : Comments are the additional readable information to clarify the source code.

Comments in Python are the non-executable statements which begin with a hash symbol (#) and generally ends with end of the line.

E.g. #This is a sample python program

you can use triple-quotes (''' or """) multiple lines comments.

Program structure cont…

- **Function :** A function is a code that has a name and it can be reused (executed again) by specifying its name in the program, where needed.

A function begin with ‘def’ statement E.g. seeyou( )

- **Blocks & indentation:** A group of statements which are part of another statement or a function are called block or code-block or suite in python.
- Indentation is used to show blocks in python. Four spaces together mark the next indent-level.

KEYWORDS

- Keywords are the reserved words in Python.
- We cannot use a keyword as a variable name, function name or any other identifier. They are used to de1ne the syntax and structure of the Python language.
- In Python, keywords are case sensitive.
- There are 35 keywords in Python 3.11. This number can vary slightly over the course of time.
- All the keywords except True , False and None are in lowercase and they must be written as they are. The list of all the keywords is given below.

KEYWORDS

False	await	else	import	pass
None	break	except	in	raise
True	class	1nally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Rules for writing identifiers

Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _ . Names like myClass , var_1 and print_this_to_screen , all are valid example.

An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.

Keywords cannot be used as identifiers.

```
global = 1
```

Output

```
File "<interactive input>", line 1  global = 1
                    ^
SyntaxError: invalid syntax
```

Python Identifiers cont..

We cannot use special symbols like !, @, #, $, % etc. in our identifier

Output

```
line 1  a@  = 0
        ^
SyntaxError: invalid syntax
```

5. An identifier can be of any length.

Things to Remember:

- Python is a case-sensitive language. This means, Variable and variable are not the same.
- Always give the identifiers a name that makes sense. While c = 10 is a valid name, writing count = 10 would make more sense, and it would be easier to figure out what it represents when you look at your code after a long gap.

- Multiple words can be separated using an underscore, like this_is_a_long_variable .

Escape Sequence

It is a sequence of characters that, when used inside a character or string, does not represent itself but is converted into another character or series of characters that may be difficult or impossible to express directly, like newline (\n), tab (\t), and so on.

Escape Sequence	Meaning
\’	Single quote
\\’	Double quote
\\	Backslash
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\b	Backspace
\f	Formfeed
\v	Vertical Tab

Escape Sequence cont…

1.Newline (\n): Moves the cursor to the beginning of the next line.

Example: print("Hello,\nWorld!")

Output: Hello,
World!

2.Tab (\t): Inserts a tab character.

Example: print("Hello,\tWorld!")

Output: Hello, World!

3.Backspace (\b): Moves the cursor back one space.

Example: print("Hello,\bWorld!")

Output: Hello,World!

4. Carriage Return (\r): Moves the cursor to the beginning of the line.

Example: print("Hello,\rWorld!")
Output: World!

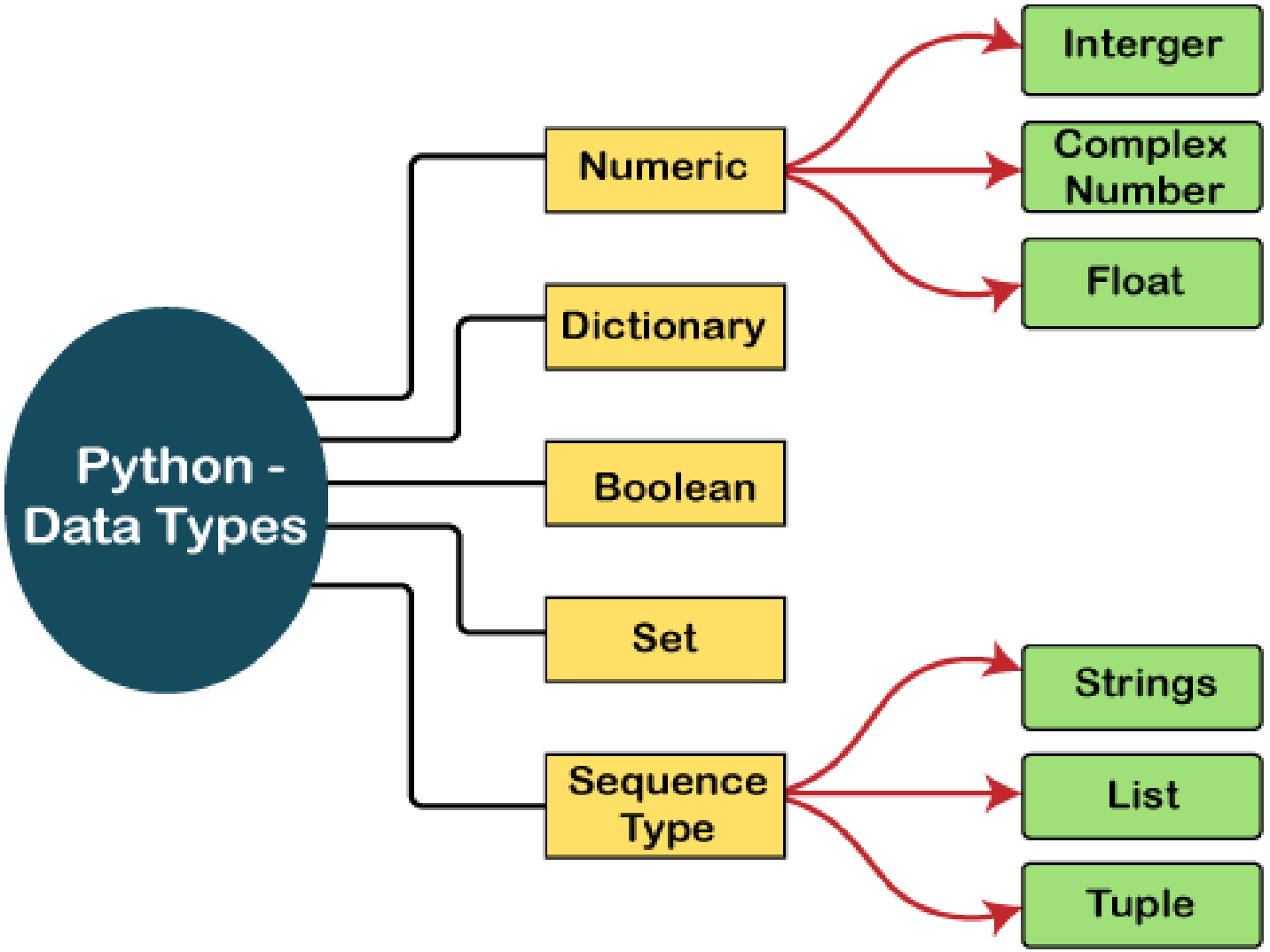
Escape Sequence cont…

5.Backslash (\\): Represents a single backslash character.

Example: print("This is a backslash: \\")

Output: This is a double backslash: \

Data Types


Python - Data Types
Numeric
Interger
Complex Number
Float
Dictionary
Boolean
Set
Sequence Type
Strings
List
Tuple

INTEGER

This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.

Note – type() function is used to determine the type of data type.

Ex- type(1) <class 'int'>

FLOATING-POINT

A floating-point number, or float is a number with a decimal place. 1.0 is a floating-point number, and -2.75 is also

Ex- type(1.0) <class 'float'>

COMPLEX

Complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j*.

For example – 2+3j

type(2+3j) - < complex>

Example:

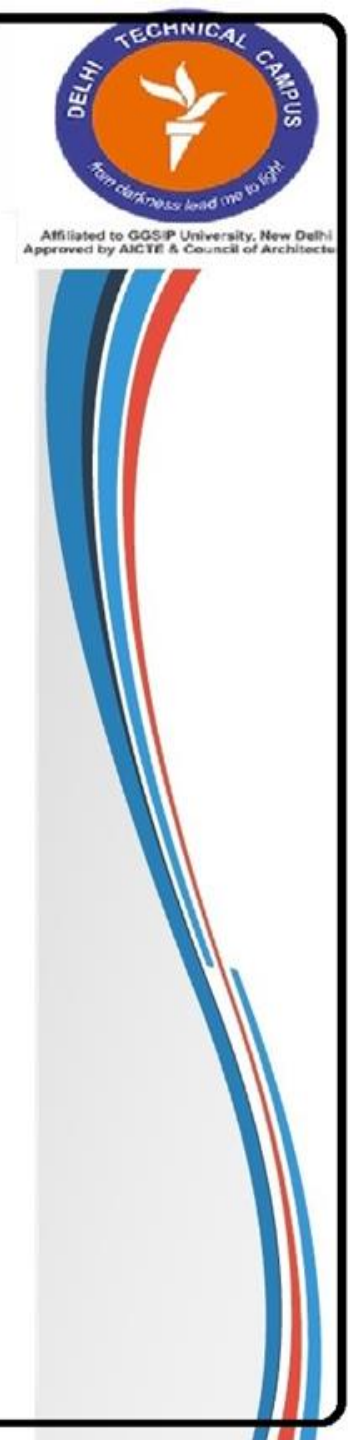
```
a = 5
print("Type of a: ", type(a))

b = 5.0
print("\nType of b: ", type(b))

c = 2 + 4j
print("\nType of c: ", type(c))
```

OUTPUT:

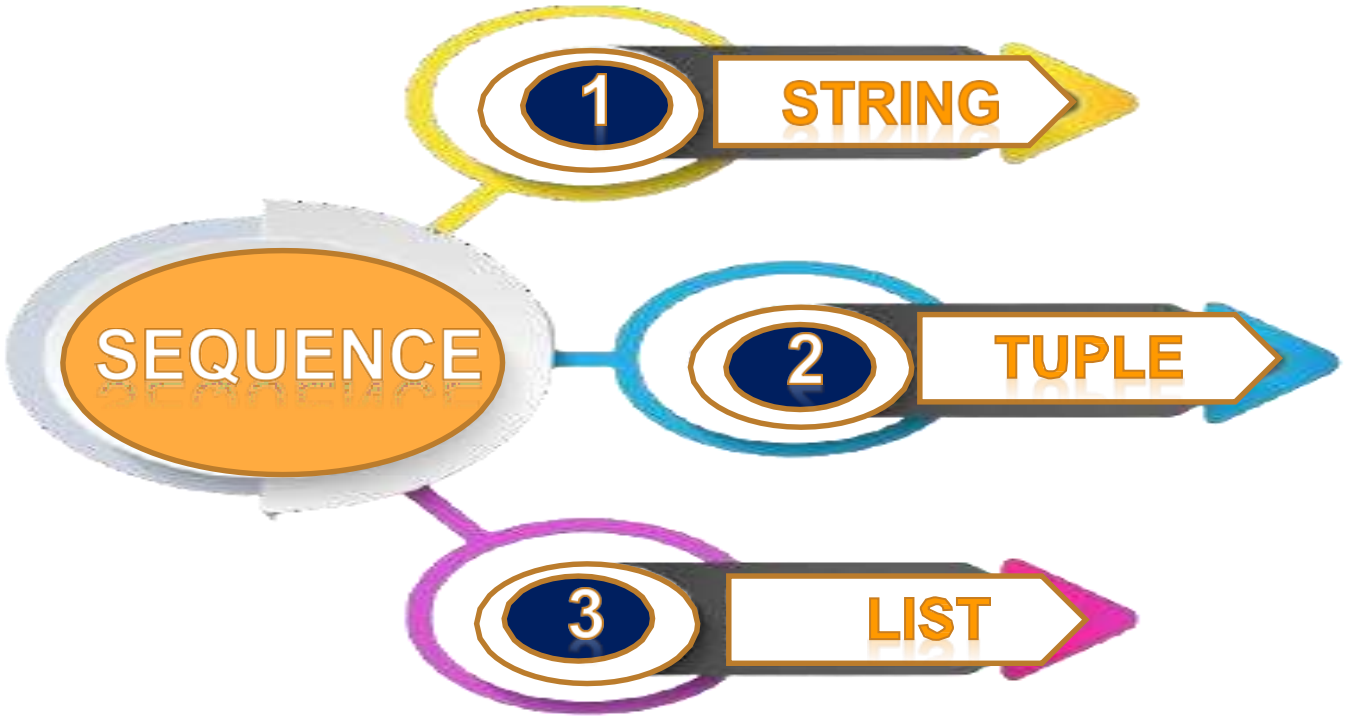
Type of a: <class 'int'>

Type of b: <class 'float'>

Type of c: <class 'complex'>



Sequences allow you to store multiple values in an organized efficient and fashion



String

A string is a collection of one or more characters put in a single quote, double- quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Eg : S1 = “Python"
print(‘Initial String: ‘)
print(S1)

Output: Initial String:
Python

List

- A list is a built-in data type used to store a collection of items.
- Lists are mutable, which means you can modify their contents (add, remove, or change elements) after they are created.
- Lists are defined using square brackets [] and can hold elements of different data types, including other lists.

Ex- L1=[1,2,3,4]

L2=[“Annu”, ”Mini”, 33, 44]

L3=[[1,2,3,4],”Annu”,56]

tuple

- A tuple is a built-in data type used to store an ordered, immutable sequence of elements.
- Tuples are similar to lists, but the key difference is that once a tuple is created, its elements cannot be changed or modified.
- Tuples are defined using parentheses () and can contain elements of different data types.

Ex- t1=(2,3,4,6)

t2=(“Annu”, ”Mini”, 33, 44)

SET

- Set is a built-in data type used to store an unordered collection of unique elements.
- Sets are defined using curly braces {} .
- Sets do not allow duplicate elements, and they are mutable, meaning you can add or remove elements after the set is created.

Ex:

Set1={“BMW”, “Ferrari”, “Tesla”, “Ford”, ”Honda”}

Set2={Honda”,”Yamaha”,”Kawasaki”,20,40}

Dictionary

- Aa dictionary is a built-in data type that represents an unordered collection of key-value pairs.
- Dictionaries are defined using curly braces {} and consist of comma-separated key-value pairs.
- Each key must be unique, and it is associated with a corresponding value.

Ex:

```
# Creating a dictionary with key-value pairs
my_dict = {'name': ‘xyz', 'age': 30, 'city': 'New York'}
```

Entering expression into the interactive shell

- To enter expressions into an interactive shell, you can use the Python interpreter or an interactive development environment (IDE).
- The most common way is to open a terminal or command prompt and run the Python interpreter.

Here are the steps:

- **Open a Terminal or Command Prompt**
- On Windows, you can open the Command Prompt.
- On macOS or Linux, you can open the Terminal.
- **Launch the Python Interpreter:**

Type python or python3 and press Enter. This will start the Python interpreter.

Entering expression into the interactive shell cont…

- **Enter Expressions:**
- Once you are in the Python interpreter, you can enter Python expressions directly. Each expression is evaluated, and the result is displayed immediately.

Eg:

```
>>> 2 + 3
5
>>> x = 10
>>> x * 2
20
```

- **Exit the Interpreter:**

To exit the Python interpreter, you can type exit() or quit()

```
>>> exit()
```

- Alternatively, you can press Ctrl+D (on Unix-like systems) or Ctrl+Z (on Windows) to exit.

Entering expression into the interactive shell cont…

- Interactive shell is also referred to as the Python REPL.
- REPL stands for Read-Eval-Print Loop.
- It is an interactive programming environment that allows users to enter individual commands or expressions, which are then read, evaluated, and the result is printed back to the user.
- or
- In a REPL environment, users can enter code line by line, and each line is immediately executed. The results of the evaluation are displayed in real-time, providing a quick and interactive way to explore programming concepts, test ideas, and learn a programming language.

Python script

Script:

A Python script is a text file with a .py extension containing a sequence of Python commands.

The script is executed as a whole, and the entire content is executed sequentially from top to bottom.

Usage: Scripts are typically used for larger programs or projects. They can be run from the command line or executed within an IDE.

Strings

- String is a sequence of characters.
- String may contain alphabets, numbers and special characters.
- Usually strings are enclosed within a single quotes and double quotes.
- Strings is immutable in nature.
- **Example**:

a="hello world"

b="Python"

Inbuilt String functions

- Python mainly contains 3 inbuilt string functions.
- They are
  - len()
  - max()
  - min()
- len()- Find out the length of characters in string
- min()- Smallest value in a string based on ASCII values
- max()- Largest value in a string based on ASCII values

Example for Inbuilt string functions

```
name=input("Enter Your name:")
 print("Welcome", name)

print("Length of your name:",len(name))
print("Maximum value of character in your name",  max(name))
print("Minimum value of character in your name", min(name))
```

OUTPUT :

Enter Your name: PRABHAKIRAN
Welcome PRABHAKIRAN
Length of your name: 11
Maximum value of character in your name R
Minimum value of character in your name A

Strings Concatenation

- The + operator used for string concatenation.

Example:

a=“Hai”

b=“how are you”

c=a+b

print(c)

Output:

Hai how are you

```
>>> a="Hai"
>>> b=' how are you'
>>> c=a+b
>>> print(c)
Hai how are you
```

String Replication-
(* operator)

- Replicating strings with the “*” operator can create multiple copies of a string.
- Example:

```
>>> print("Python"*10)
PythonPythonPythonPythonPythonPythonPythonPythonPythonPython
```

String Slicing

- Slicing operation is used to return/select/slice the particular substring based on user requirements.
- A segment of string is called slice.

- **Syntax**: string_variablename [start:end]

String Slice example

s=“Hello”

```
>>> s="hello"
>>> s[1:4]
'ell'
>>> s[1:]
'ello'
>>> s[:]
'hello'
>>> s[1:100]
'ello'
>>> s[-1]
'o'
>>> s[::]
'hello'
>>> s[:-3]
'he'
```

	H	e	l	l	o
0		1	2	3	4
	-5	-4	-3	-2	-1

String Comparision

- We can compare two strings using comparision operators such as ==, !=, <,<=,>, >=
- Python compares strings based on their corresponding ASCII values.

Example of string comparision

```
str1="green"  str2="black"
print("Is both Equal:", str1==str2)
print("Is str1> str2:", str1>str2)
print("Is str1< str2:", str1<str2)
```

OUTPUT:

Is both Equal: False
Is str1> str2: True
Is str1< str2: False

String functions and methods

len()	min()	max()	isalnum()	isalpha()
isdigit()	islower()	isuppe()	isspace()	isidentifier()
endswith()	startswith()	find()	count()	capitalize()
title()	lower()	upper()	swapcase()	replace()
center()	ljust()	rjust()	center()	isstrip()
rstrip()	strip()			

Converting string functions

captitalize()	Only First character capitalized
lower()	All character converted to lowercase
upper()	All character converted to uppercase
title()	First character capitalized in each word
swapcase()	Lower case letters are converted to Uppercase and Uppercase letters are converted to Lowercase
replace(old,new)	Replaces old string with nre string

Program:

```
str=input("Enter any string:")
print("String Capitalized:", str.capitalize())
print("String lower case:", str.lower())
print("String upper case:", str.upper())
print("String title case:", str.title())
print("String swap case:", str.swapcase())
print("String replace   case:",str.replace("python","python programming"))
```

Output:

```
Enter any string: Welcome to python
String Capitalized: Welcome to python
String lower case: welcome to python
 String upper case: WELCOME TO PYTHON
String title case: Welcome To Python
String swap case: wELCOME TO PYTHON
String replace case: Welcome to python programming
```

Formatting String functions

center(width)	Returns a string centered in a field of given width
ljust(width)	Returns a string left justified in a field of given width
rjust(width)	Returns a string right justified in a field of given width
format(items)	Formats a string

Program:

```
a=input("Enter any string:")
print("Center alignment:", a.center(20))
print("Left alignment:", a.ljust(20))
print("Right alignment:", a.rjust(20))
```

Output:

```
Enter any string:welcome
Center alignment:       welcome
Left alignment: welcome
Right alignment:                welcome
```

Removing whitespace characters

lstrip()	Returns a string with leading whitespace characters removed
rstrip()	Returns a string with trailing whitespace characters removed
strip()	Returns a string with leading and trailing whitespace characters removed

Program

```
a=input("Enter any string:")
print("Left space trim:",a.lstrip())
print("Right space trim:",a.rstrip())
print("Left and right trim:",a.strip())
```

Output:

```
Enter any string:        welcome
Left space trim: welcome
Right space trim:        welcome
Left and right trim: welcome
```

Testing String/Character

isalnum()	Returns true if all characters in string are alphanumeric and there is atleast one character
isalpha()	Returns true if all characters in string are alphabetic
isdigit()	Returns true if string contains only number character
islower()	Returns true if all characters in string are lowercase letters
isupper()	Returns true if all characters in string are uppercase letters
isspace()	Returns true if string contains only whitespace characters.

Program

```
a=input("Enter any string:")
print("Alphanumeric:",a.isalnum())
print("Alphabetic:",a.isalpha())
print("Digits:",a.isdigit())
print("Lowecase:",a.islower())
print("Upper:",a.isupper())
```

Output:

```
Enter any string:python
Alphanumeric: True
Alphabetic: True
Digits: False
Lowecase: True
Upper: False
```

Searching for substring

Endswith()	Returns true if the strings ends with the substring
Startswith()	Returns true if the strings starts with the substring
Find()	Returns the lowest index or -1 if substring not found
Count()	Returns the number of occurrences of substring

Program

```
a=input("Enter any string:")
print("Is string ends with thon:", a.endswith("thon"))
print("Is string starts with good:", a.startswith("good"))
print("Find:", a.find("ython"))
print("Count:", a.count("o"))
```

Output:

```
Enter any string : welcome to python
Is string ends with thon: True
Is string starts with good: False
Find: 12
Count: 3
```

Split Function

The split() method splits a string into a list.

You can specify the separator, default separator is any whitespace.

```
txt = "welcome to the jungle"

x = txt.split()

print(x)
```

Output:

['welcome', 'to', 'the', 'jungle’]

```
txt = "hello# my name is Peter # I am 26 years old"

x = txt.split(“#")

print(x)
```

OUTPUT

['hello', 'my name is Peter', 'I am 26 years old']

Join()

The join() method takes all items in an iterable and joins them into one string.

A string must be specified as the separator.

```
l = ["John", "Peter", "Vicky"]
x = "#".join(l)
print(x)
```

Output:

John#Peter#Vicky

Find()

String find() method returns the lowest index or first occurrence of the substring if it is found in a given string. If it is not found, then it returns -1.

Syntax: str_obj.find(sub, start, end)

Parameters:

sub: Substring that needs to be searched in the given string.
start (optional): Starting position where the substring needs to be checked within the string.
end (optional): End position is the index of the last value for the specified range. It is excluded while checking.
Return: Returns the lowest index of the substring if it is found in a given string. If it's not found then it returns -1.

Ex1 word = ' here for you'
print(word.find('for'))

Output: 5

Ex2 word = 'here for you'
print(word.find('o',2,8))

Output:
6

Replace()

The replace() method replaces a specified phrase with another specified phrase.

Syntex: string.replace(oldvalue, newvalue)

Or

string.replace(oldvalue, newvalue,count)

Examples:

Ex1
```
txt = "one one was a race horse, two two was one too.“
x = txt.replace("one", "three")

print(x)
```

Output:

three three was a race horse, two two was three too.

Ex2.
```
txt = "one one was a race horse, two two was one too."
x = txt.replace("one", "three", 2)
print(x)
```

Output: three three was a race horse, two two was one too.

Operator

An operator is a symbol that performs an operation.

An operator acts on some variables called operands.

If an operator acts on single variable , it is called unary operators

If an operator acts on two variables, then it is called binary operator

If an operator acts on three variables, then it is called ternary operator.

Types of operators in Python

- Arithmetic Operators
- Assignment Operators
- Unary minus operator
- Relational Operators
- Logical Operators
- Boolean Operators
- Bitwise Operators

Arithmetic operator

- Arithmetic operators are basic mathematical operations.

Operator	Meaning	Example	Result
+	Addition	C=12+1	C=13
-	Subtraction	C=12-1	C=11
*	Multiplication	C=12*1	C=12
/	Division	C=12/1	C=12.0
//	Floor division	C=12//10	1
%	Modulus	C=12%10	C=2
**	Exponentiation	C=10**2	C=100

Example of Arithmetic Operator

```
print("Arithmetic Operator")
a=10
b=5
print("Addition:",a+b)
print("Subtraction:",a-b)
print("Multiplication:",a*b)
print("Division:",a/b)
print("Floor Division:",a//b)
print("Modulus:",a%b)
print("Exponent",a**b)
```

Output:

```
Arithmetic Operator
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Floor Division: 2
Modulus: 0
Exponent 100000
```

Relational operator

- Relational operators are also called as Comparison operators
- It is used to compare values.
- It either returns True or False according to

Operator	Meaning	Example	Result
>	Greater than	5>6	False
<	Less than	5<6	True
==	Equal to	5==6	False
!=	Not equal to	5!=6	True
>=	Greater than or equal to	5>=6	False
<=	Less than or equal to	5<=6	True

Example of Relational Operator

```
print("Relational Operator")

a=10
b=5
print(a>b)
print(a<b)
print(a==b)
print(a!=b)
print(a>=b)
print(a<=b)
```

Output:

```
Relational Operator
True
False
False
True
True
False
```

Logical operator

- Logical operator are typically used with Boolean(logical) values.
- They allow a program to make a decision based on multiple condition.

Operator	Meaning	Example	Result
and	True if both the operands are true	10<5 and 10<20	False
or	True if either of the operands is true	10<5 or 10<20	True
not	True if operands is false (complements the operand)	not (10<20)	False

Example of Logical Operator

```
print("Logical Operator")
print(10<5 and 10<20)
print(10<5 or 10<20)
  print(not(10<20))
```

Output:

```
Logical Operator

False
True
False
```

Bitwise operator

- Bitwise operators act on operands as if they are string of binary digits.
- It operates bit by bit.

Operator	Meaning	Example
&	Bitwise AND	a & b
|	Bitwise OR	a | b
~	Bitwise NOT	a ~ b
^	Bitwise XOR	a ^ b
>>	Bitwise right shift	a >> 2
<<	Bitwise left shift	a << 2

Assignment operator

- Assignment operators are used to assign values to variables.

Operator	Meaning	Example
=	Assign a value	a=5
+=	Adds and assign the result to the variable	a+=1 (a=a+1)
-=	Subtracts and assign the result to the variable	a-=1 (a=a-1)
*=	Multiplies and assign the result to the variable	a*=5 (a=a*5)
/=	Division and assign the result to the variable	a/= (a=a/5)
//=	Floor division and assign the result to the variable	a//=5(a=a//5)
%=	Find modulus and assign the result to the variable	a%=5 (a=a%5)
**=	Find Exponentiation and assign the result to the variable	a**=5 (a=a**5)

Operator	Meaning	Example
&=	Find Bitwise AND and assign the result to the variable	a&=5(a=a&5)
|=	Find Bitwise OR and assign the result to the variable	a|=5(a=a|5)
^=	Find Bitwise XOR and assign the result to the variable	a^=5(a=a^5)
>>=	Find Bitwise right shift and assign the result to the variable	a>>=5 (a=a>>5)
<<=	Find Bitwise left shift and assign the result to the variable	a<<=5 (a=a<<5)

Special operator

- Python offers some special operators like identity operator and the membership operator.
- **Identity Operator:**

– **is** and **is not** are the identity operator

Operator	Meaning	Example
is	True if the operands are identical and occupy same memory location.	a is true
is not	True if the operands are not identical	a is not true

Example of Identity Operator

```
a1=5  b1=5
a2="Hello"
b2="Hello"

a3=[1,2,3]
b3=[1,2,3]

print(a1 is not b1)
print(a2 is b2)
print(a2 is b3)
print(a3 is b3)
```

Output:

```
False
True
False
False
```

- **Membership Operators:**

– **in** and **not in** are the membership operators.

Operator	Meaning	Example
in	True if value/ variable is found in the sequence	5 in a
not in	True if value/ variable is not found in the sequence	5 not in a

Example of Membership Operator

```
a="Hello world"
b={1,"a","b",2}

print("H" in a)
print("hello" in a )
print(1 in b)
print("b" in b)
Print(“c” not in b)
```

Output:

True
False
True
True
True

Precedence of Operators in Python

Operator	Description
( )	Parentheses (grouping)
**	Exponentiation
~x	Bitwise nor
+x,-x	Positive , negative (unary +,-)
*,/,//,%	Multiplication , division , floor division , remainder
+,-	Addition , subtraction
&	Bitwise and
^	Bitwise XOR
|	Bitwise OR
<,<=,>,>=,<>,!=,== ,is,isnot	Comparisons (Relational operators),identity operators
not x	Boolean NOT
And	Boolean AND
or	Boolean OR

Highest

Lowest

CONTROL FLOW - BRANCHING

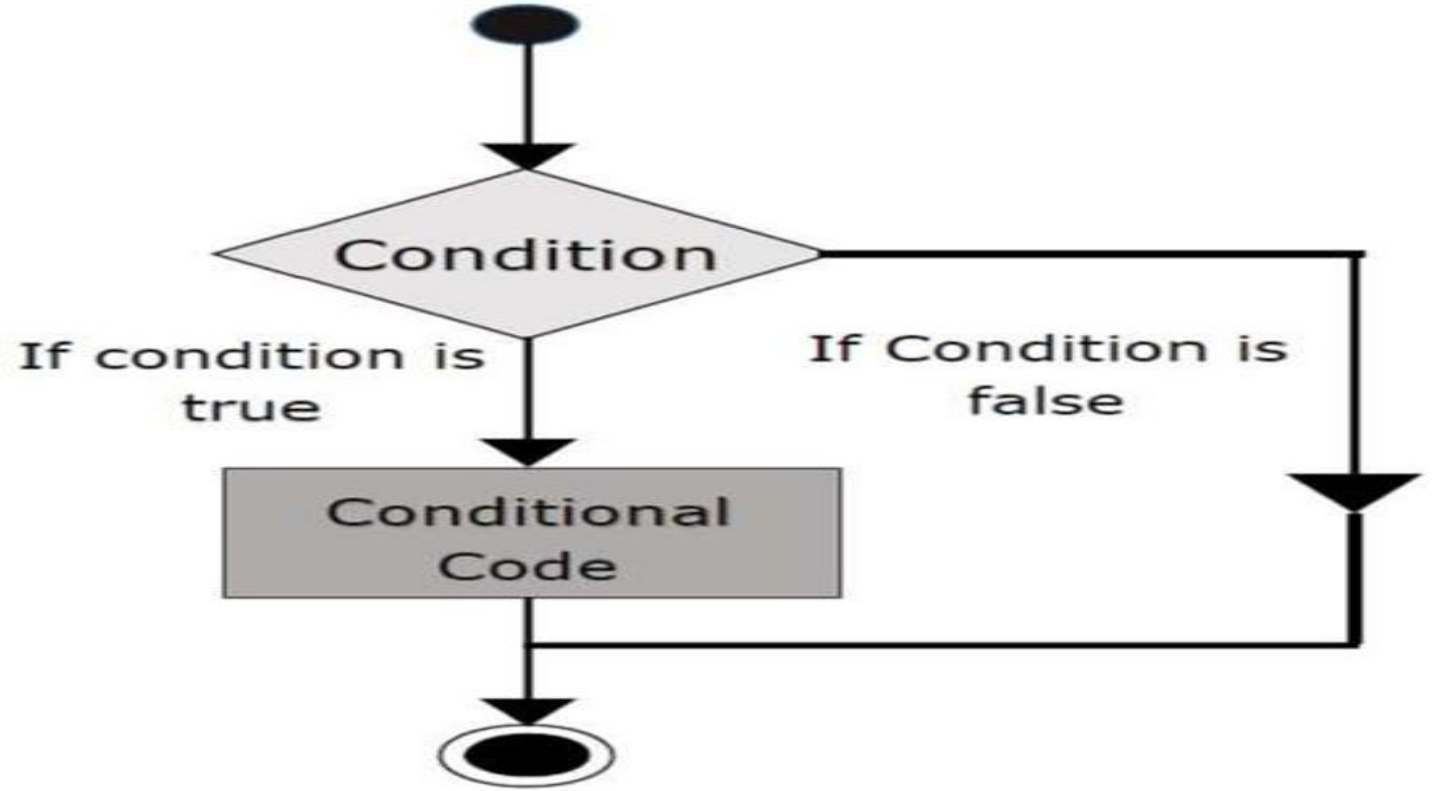
```
if <condition>:
<expression>
<expression>
...
```

```
if <condition>:
<expression>
<expression>
...
else:
<expression>
<expression>
...
```

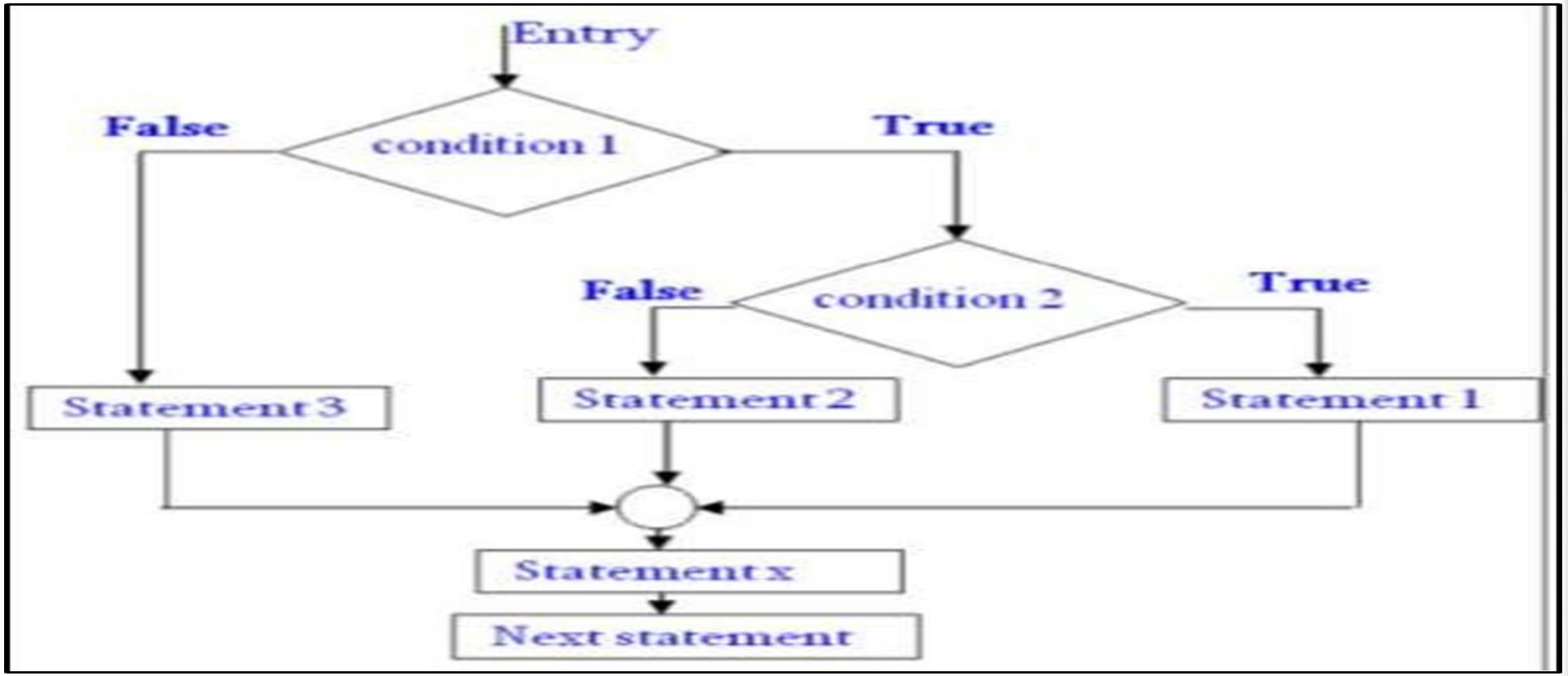
```
if <condition>:
<expression>
<expression>
...
elif <condition>:
<expression>
<expression>
...
else:
<expression>
<expression>
...
```

- `<condition>` has a value `True` or `False`
- evaluate expressions in that block if `<condition>` is `True`

CONTROL FLOW – If statement



Nested if … else Statements



CONTROL FLOW – If statement

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if(x == y):
    print("x and y are equal")
elif(y!=0):
    print("therefore,  x  /  y  is  not
infinite")
elif(x < y):
     print("x is smaller")
else:
    print("y is smaller")
print("thanks!")
```

What if x = y here?

get a SyntaxError

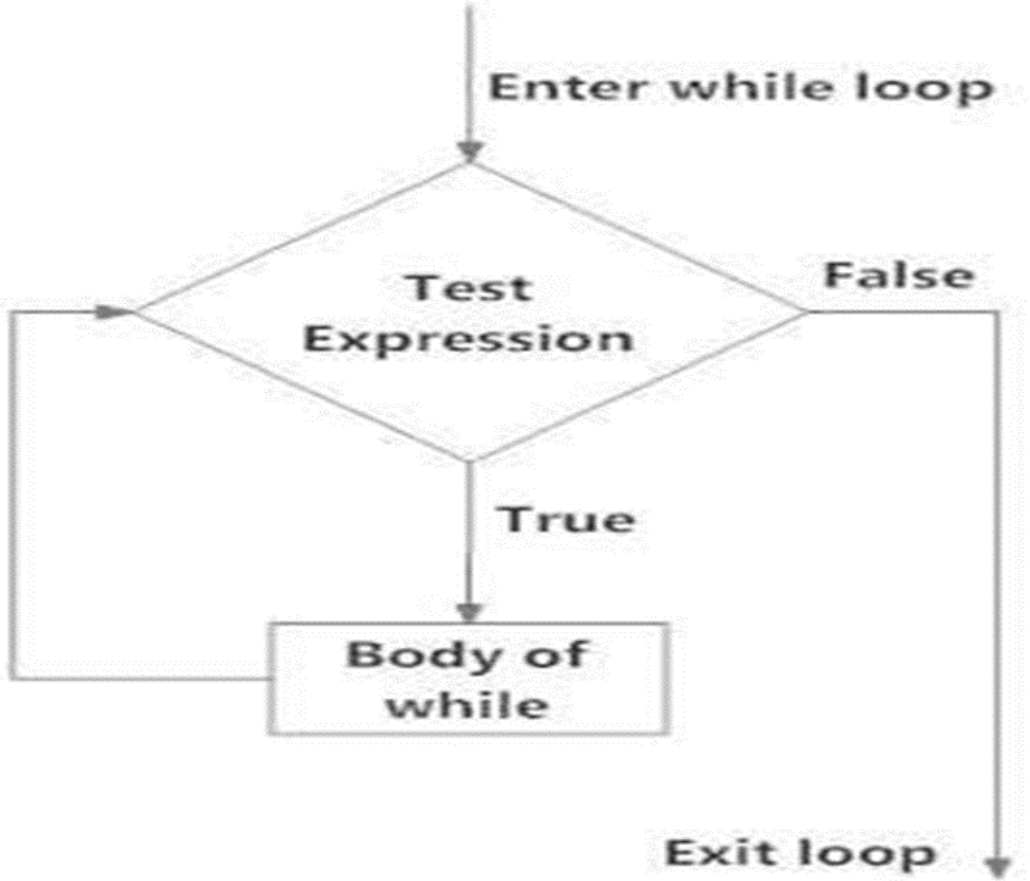
CONTROL STATEMENT (Looping Statement)

- Program statement are executed sequentially one after another. In some situations, a block of code needs of times.
- These are repetitive program codes, the computers have to perform to complete tasks. The following are the loop structures available in python.

- **while statement**
- **for loop statement**
- **Nested loop staement**

While Loop

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.



While Loop Example

```
# program to display numbers from 1 to 5

# initialize the variable
i = 1
n = 5
while i <= n:
print(i)
    i = i + 1
```

For loop statement

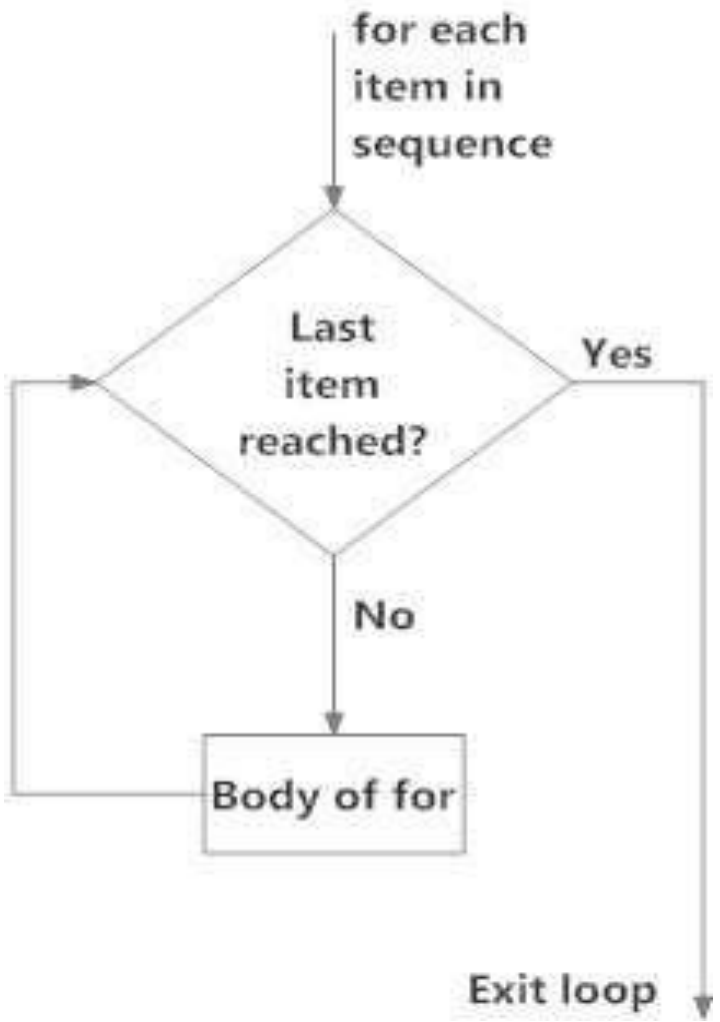
.

❖Thefor loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax

```
for val in sequence:
 Body of for loop
```

For loop flow chart



Addition of number using for loop

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4]
sum = 0
for val in numbers:
    sum = sum+val
print("The sum is", sum)
```

OUTPUT

The sum is 37

#To print sum of all numbers from 1 to a given number.

```
a =int (input("enter the end value"))
sum = 0
for i in range(1,a+1):
        sum+=I
print(sum)
```

enter the end value5
15

Nested For loop

```
n = 5;
print("Pattern for n = 5 is,")
for x in range(1,n+1):
    for y in range(x):
        print("*",end=" ")
    print()
```

Output:

Pattern for n = 5 is,

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

Note: The end parameter in the print function is used to add any string. At the end of the output of the print statement in python.

By default, the print function ends with a newline.

Passing the whitespace to the end parameter (end=‘ ‘) indicates that the end character has to be identified by whitespace and not a newline.

```
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

output

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

```
multiplicand = int(input('Enter the multiplicant       : '))
multiplier = int(input('Enter the maximum multiplier : ‘))
i=0
while i<=multiplier:
        print(f"{multiplicand} * {i} = {multiplicand * i}")
        i += 1
```

Output

```
Enter the multiplicant       : 5
Enter the maximum multiplier : 10
5 * 0 = 0
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Q1: Write a Python program that takes a student's score as input and prints their corresponding grade based on the following criteria:

90 or above: A
80-89: B
70-79: C
60-69: D
Below 60: F

Q2: Write a Python program that checks whether a given year is a leap year or not.

Q3Write a Python program that takes three numbers as input and prints the largest one.

Q4: Write a Python program that takes an integer as input and prints whether it's even or odd.

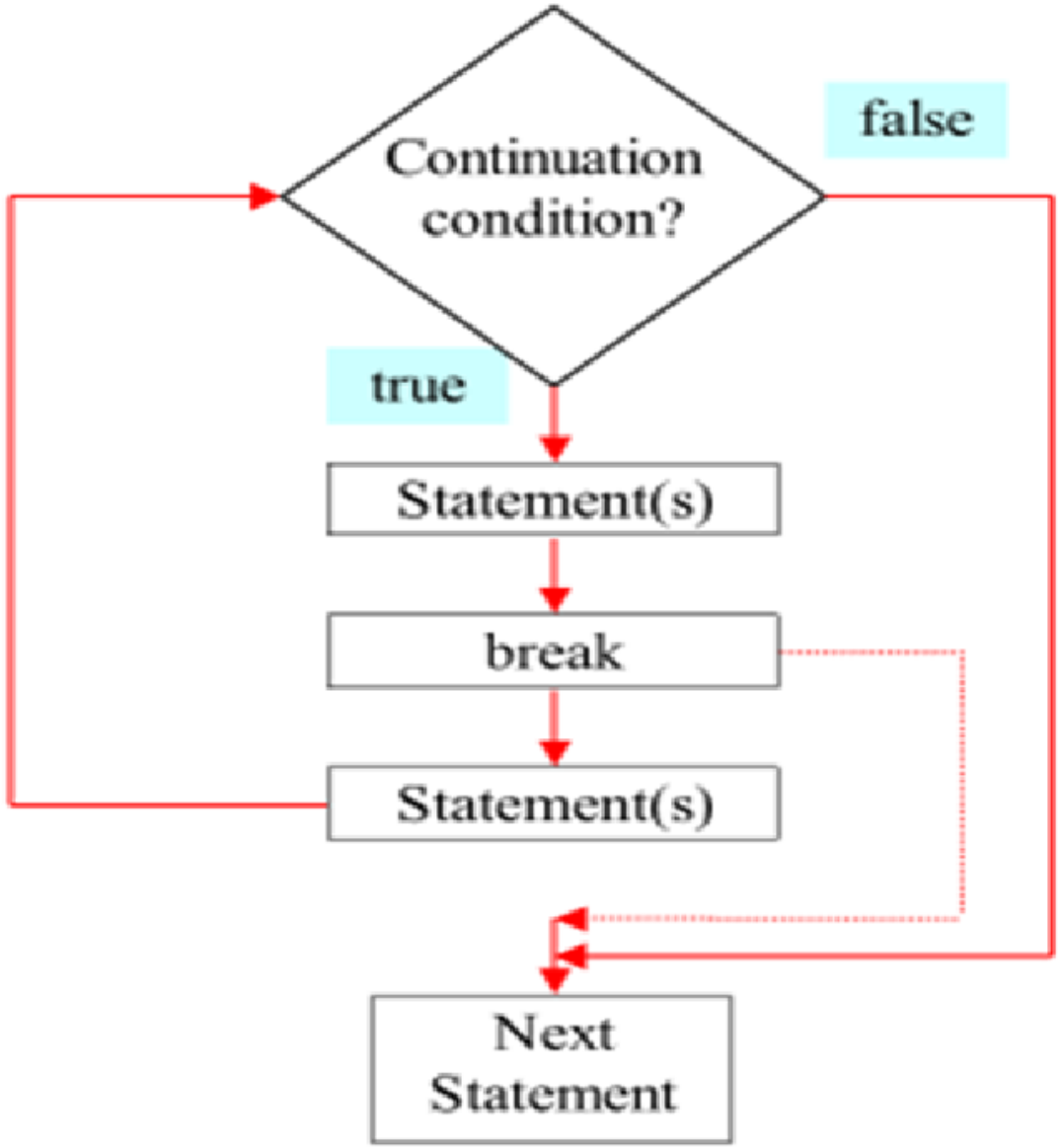
break and continue Statements

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only innermost loop!

Continue:

- The continue statement skips some statements inside the loop.
- Loop will restart when continue is encountered.

The break Keyword



Example of Break Statement

```
for i in range(0,5):
    if i == 3:
        break

    print(i)
```

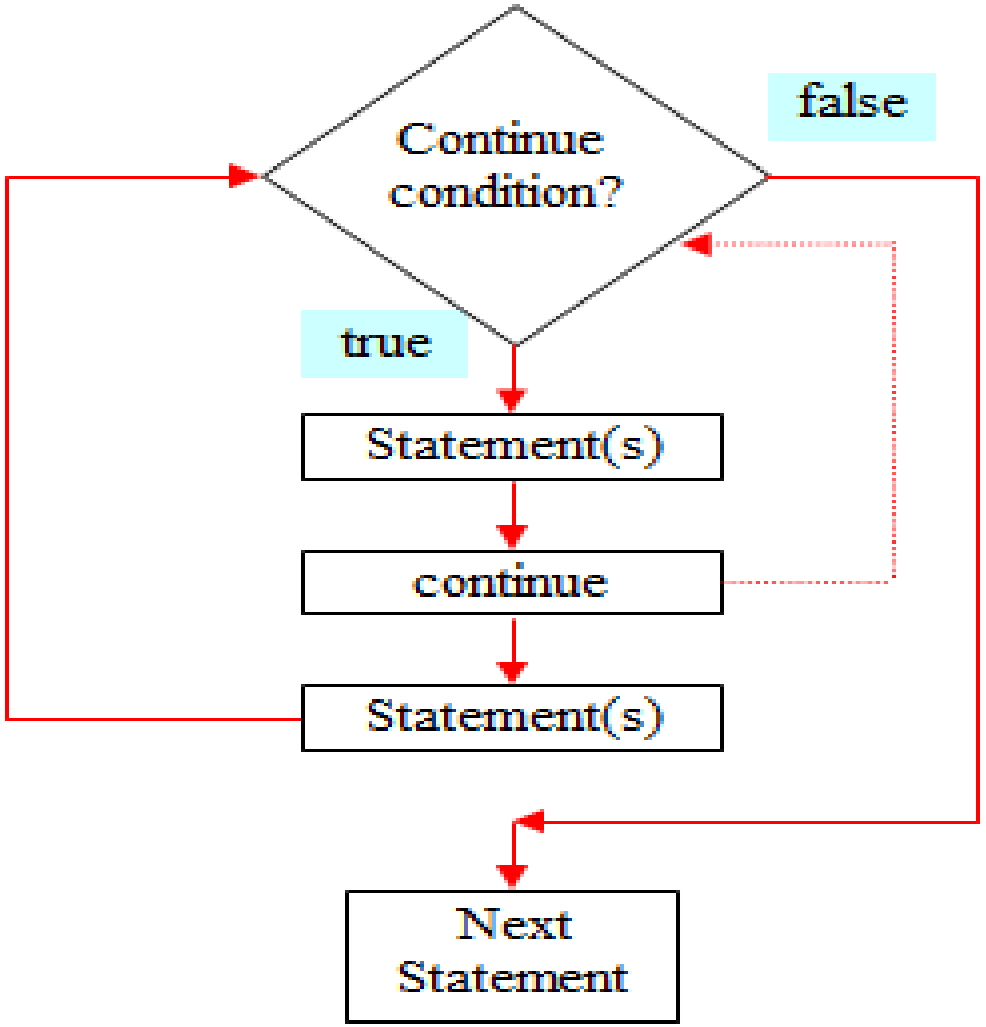
Output:

0

1

2

The continue Keyword



Example of continue Statement

```
for i in range(0,5):
    if i == 3:
        continue
    print(i)
```

Output:

0

1

2

4

Pass Statement

In Python, the **pass** statement is a null operation.
It doesn't do anything, and it acts as a placeholder where syntactically a statement is required but you don't want any code to execute.

Example:
```
def some_function():
    pass  # Placeholder for the function implementation

for item in my_list:
    # do something with each item
    pass  # Placeholder for future code

if condition:
    # Handle the condition
else:
    # Placeholder for the else block
    pass
```

For VS while LOOPS

`for` loops

- **know** number of iterations
- can **end early** via `break`
- uses a **counter**
- **can rewrite** a `for` loop using a `while` loop

`while` loops

- **unbounded** number of iterations
- can **end early** via `break`
- can use a **counter but must initialize** before loop and increment it inside loop
- **may not be able to rewrite** a `while` loop using a `for` loop

Importing 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.

- Modules in Python provides us the flexibility to organize the code in a logical way.
- Import in Python is similar to #include header_file in C/C++. Python modules can get access to code from another module by importing the file/function using import.

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

#displayMsg prints a message to the name being passed.

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

In this example, we will create a module named as **file.py** which contains a function that contains a code to print some message on the console.

Loading the module in our python code

- **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.
- we can use the functionality of any python source file by importing that file as the module into another python source file.

Example

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

Output:Enter your name? Aakash
Hi Aakash

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>

Example: create a module as **calculation.py :**

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

```
from calculation import summation
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
 print("Sum = ",summation(a,b))
```

Output:

```
Enter the first number10
 Enter the second number20
 Sum =    30
```

The from...import statement is always better to use if we know the attributes to be imported from the module in advance.
It doesn't let our code to be heavier.
We can also import all the attributes from a module by using *.
Syntax:

from <module> import *

example:

```
from calculation import *
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
 print("Sum = ",summation(a,b))
print(“product= ", multiplication(a,b))
print(“division = ", divide(a,b))
```

Output:

```
Enter the first number5
Enter the second number7
Sum =  12
product=  35
division =  0.7142857142857143
```

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.

The syntax to rename a module is given below.

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

```
import calculation as cal
 a = int(input("Enter a"))
b = int(input("Enter b"))
print("Sum = ",cal.summation(a,b))
```

Output:

```
Enter a=10
 Enter b=20
Sum = 30
```

Import Python Built-In Modules

1. **math module:** It is a built in module in python .To use mathematical functions under this module, we have to import the module as:

import math

#The math.sqrt() method returns the square root of a given number.

>>>math.sqrt(100)

10.0

>>>math.sqrt(3) 1.7320508075688772

'''The ceil() function approximates the given number to the smallest integer, greater than or equal to the given floating point number. The floor() function returns the largest integer less than or equal to the given number'''

>>>math.ceil(4.5867) 5

>>>math.floor(4.5687) 4

“’The math.pow() method receives two float arguments, raises the first to the second and returns the result. In other words, pow(2,3) is equivalent to 2**3.’”

>>>math.pow(2,4)

16

#Returns the absolute value of x

>>> import math

>>> math.fabs(-5.5)

5.5

“’The math module contains functions for calculating various trigonometric ratios for a given angle. The functions (sin, cos, tan, etc.) need the angle in radians as an argument.’”

>>> math.sin(270)

-0.1760459464712114

Random Module

The random module provides access to functions that support many operations.

Perhaps the most important thing is that it allows us to generate

random numbers.

1. random.randint()

Randint accepts two parameters: a lowest and a highest number.

- **import random**
- **print (random.randint(0, 5))**
- **This will output either 0,1, 2, 3, 4 or 5.**

2. random.random()

- Genereate random number from 0.01 to 1.If we want a larger number, we can multiply it.

import random

print(random.random())

Output:0.33222497054616607

import random

print(random.random()*100)

output: 33.222497054616607

Random Module

randrange()
generate random numbers from a specified range and also allowing rooms for steps to be included**.**
Syntax :
random.randrange(start(opt),stop,step(opt))

```
import random
# Using randrange() to generate numbers from 0-100  print ("Random number from 0-100 is : ",end="")
 print (random.randrange(100))
# Using randrange() to generate numbers from 50-100
 print ("Random number from 50-100 is : ",end="")
 print (random.randrange(50,100))
# Using randrange() to generate numbers from 50-100  # skipping 5
print ("Random number from 50-100 skip 5 is : ",end="")
print (random.randrange(50,100,5))
```

OUTPUT:

```
Random number from 0-100 is : 27
  Random number from 50-100 is : 48
 Random number from 50-100 skip 5 is : 80
```