PYTHON

Python Introduction

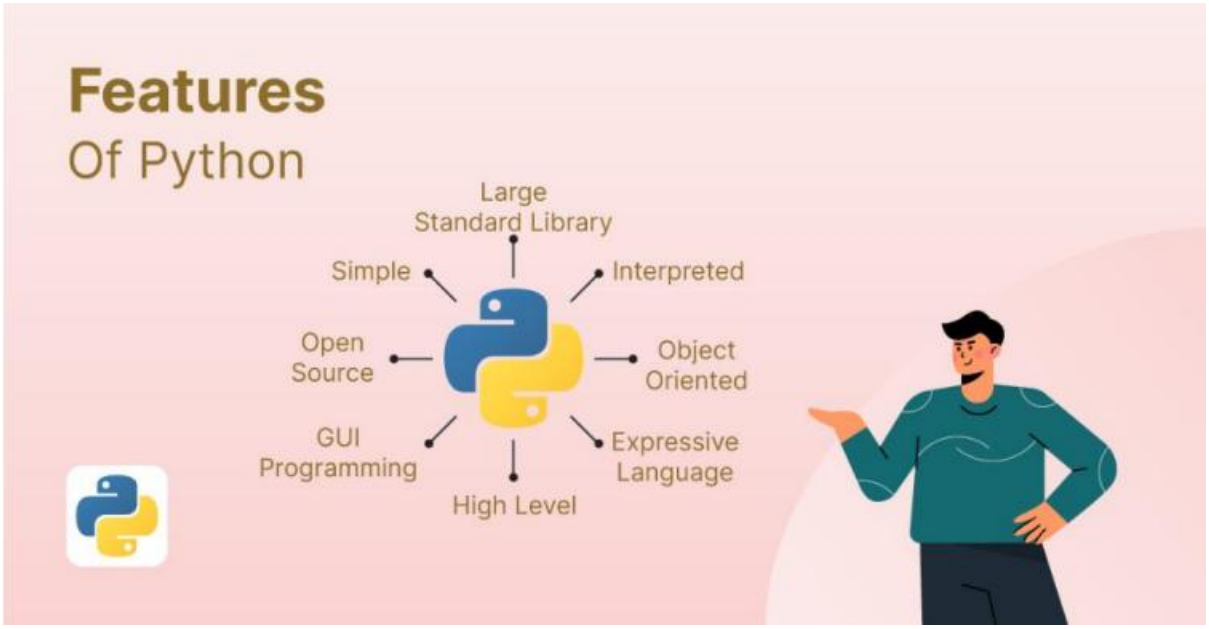
What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Python Features Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don't need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc. In this article we will see what characteristics describe the python programming language.



Features in Python

1. Free and Open Source
2. Easy to code Python is a high-level programming language.
3. Easy to Read As you will see, learning Python is quite simple.
4. Object-Oriented Language One of the key features of Python is Object-Oriented programming.
5. Interpreted Language:

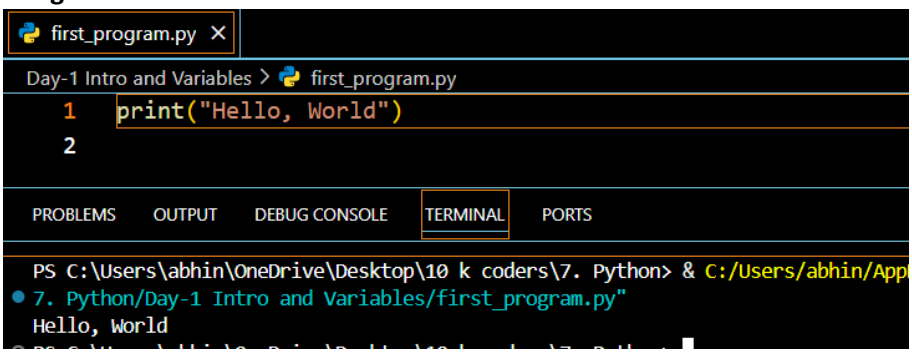
Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Syntax compared to other programming languages

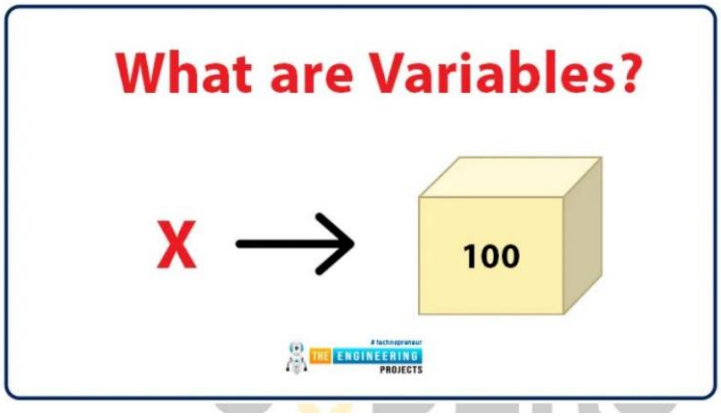
- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

First Program:



Python Variables

Python Variable is containers that store values. Python is not “statically typed”. We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it. A Python variable is a name given to a memory location. It is the basic unit of storage in a program.



Example of Variable in Python

An Example of a Variable in Python is a representational name that serves as a pointer to an object. Once an object is assigned to a variable, it can be referred to by that name. In layman's terms, we can say that Variable in Python is containers that store values.

var = "10KCoders"

Variables Assignment in Python

Here, we will define a variable in python. Here, clearly we have assigned a number, a floating point number, and a string to a variable such as age, salary, and name.

```
# An integer assignment
age = 45

# A floating point
salary = 1456.8

# A string
name = "John"

print(age)
print(salary)
print(name)
```

Output:

```
45
1456.8
John
```

Declaration and Initialization of Variables

Let's see how to declare a variable and how to define a variable and print the variable.

For example:

```
# declaring the var
Number = 100

# display
print( Number)
```

Output:

```
100
```

Redeclaring variables

in Python We can re-declare the Python variable once we have declared the variable and define variable in python already.

For example:

```
# declaring the var
Number = 100

# display
print("Before declare: ", Number)

# re-declare the var
Number = 120.3

print("After re-declare:", Number)
```

Output:

```
Before declare:  100
After re-declare: 120.3
```

Python Assign Values to Multiple Variables Also,

Python allows assigning a single value to several variables simultaneously with "=" operators.

For example:

```
a = b = c = 10

print(a)
print(b)
print(c)
```

Output:

```
10
10
10
```

Assigning different values to multiple variables

Python allows adding different values in a single line with "," operators.

For example

```
a, b, c = 1, 20.2, "GeeksforGeeks"

print(a)
print(b)
print(c)
```

Output:

```
1
20.2
GeeksforGeeks
```

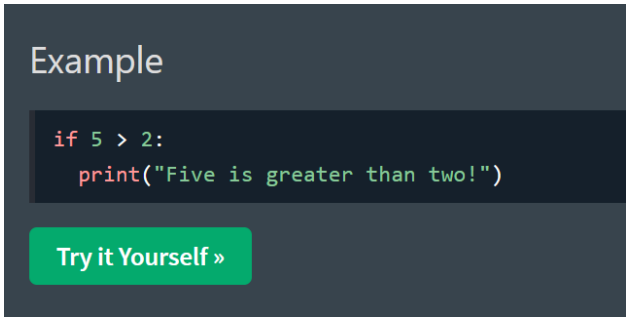
Python Installation:

https://www.tutorialspoint.com/how-to-install-python-in-windows

Python Syntax:

Python Indentation

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



Python Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

Creating a Comment

Comments starts with a #, and Python will ignore them:

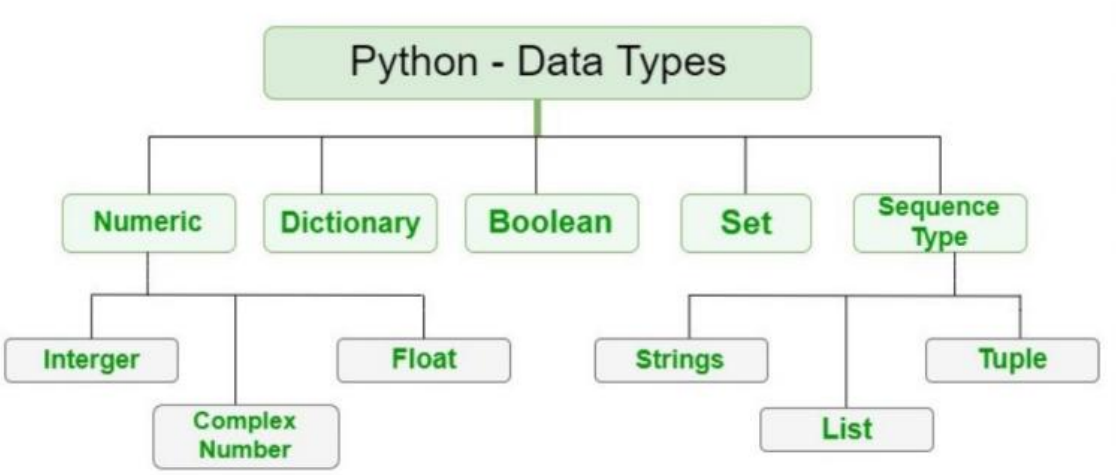
Multiline Comments

Python does not really have a syntax for multiline comments.
To add a multiline comment you could insert a # for each line:

Python Data Types

Python Data Types

Python Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes.



The following are the standard or built-in data types in Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary

Numeric Data Types in Python

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.

Integers – 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.

Float – This value is represented by the float class. It is a real number with a floating-point representation.

Complex Numbers – A complex number is represented by a complex class. It is specified as (real part) + (imaginary part)j .
For example – 2+3j

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

```
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))
```

```
Type of a:  <class 'int'>
Type of b:  <class 'float'>
Type of c:  <class 'complex'>
```

Sequence Data Types in Python

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:

- Python String
- Python List
- Python Tuple

String Data Type

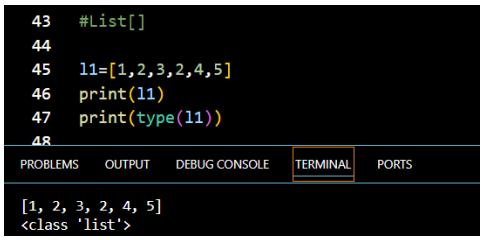
Strings in Python are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote.

Example: This Python code showcases various string creation methods. It uses single quotes, double quotes, and triple quotes to create strings with different content and includes a multiline string. The code also demonstrates printing the strings and checking their data types.

List Data Type

Lists are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating a List in Python Lists in Python can be created by just placing the sequence inside the square brackets[].




```
List = []
print("Initial blank List: ")
print(List)
List = ['GeeksForGeeks']
print("\nList with the use of String: ")
print(List)
List = ["Geeks", "For", "Geeks"]
print("\nList containing multiple values: ")
print(List[0])
print(List[2])
List = [['Geeks', 'For'], ['Geeks']]
print("\nMulti-Dimensional List: ")
print(List)
```

Output:

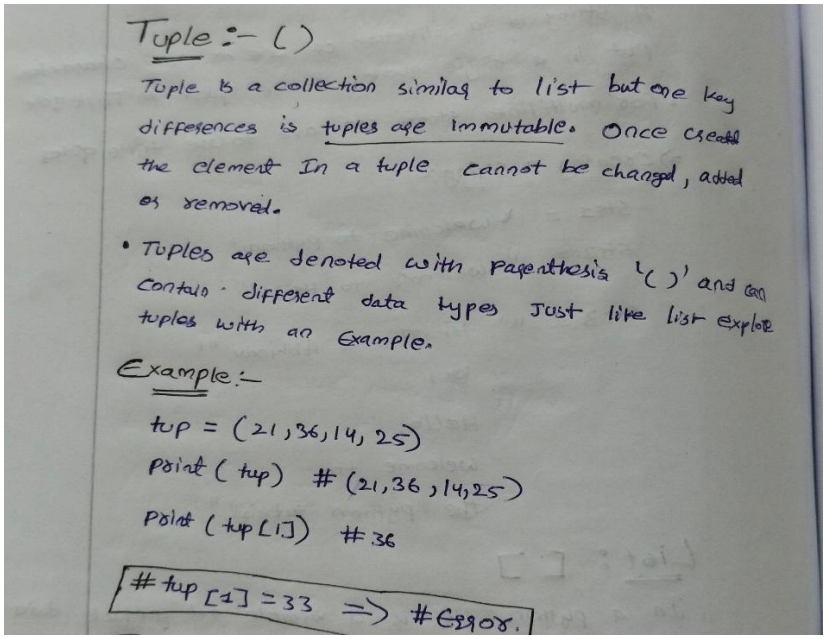
```
Initial blank List:
[]
List with the use of String:
['GeeksForGeeks']
List containing multiple values:
Geeks
Geeks
Multi-Dimensional List:
[['Geeks', 'For'], ['Geeks']]
```

Tuple Data Type

Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by a tuple class.

Creating a Tuple in Python

In Python Data Types, tuples are created by placing a sequence of values separated by a ‘comma’ with or without the use of parentheses for grouping the data sequence



Boolean Data Type in Python

Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by the class bool.

Note – True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

```
print(type(True))
print(type(False))
print(type(true))
```

Output:

```
<class 'bool'>
<class 'bool'>
```

```
Traceback (most recent call last):
  File "/home/7e8862763fb66153d70824099d4f5fb7.py", line 8, in
    print(type(true))
NameError: name 'true' is not defined
```

Set Data Type in Python

In Python Data Types, a Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Create a Set in Python

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a 'comma'. The type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

```
set1 = set()
print("Initial blank Set: ")
print(set1)
set1 = set("GeeksForGeeks")
print("\nSet with the use of String: ")
print(set1)
set1 = set(["Geeks", "For", "Geeks"])
print("\nSet with the use of List: ")
print(set1)
set1 = set([1, 2, 'Geeks', 4, 'For', 6, 'Geeks'])
print("\nSet with the use of Mixed Values")
print(set1)
```

Output:

```
Initial blank Set:
set()
Set with the use of String:
{'F', 'o', 'G', 's', 'r', 'k', 'e'}
Set with the use of List:
{'Geeks', 'For'}
Set with the use of Mixed Values
{1, 2, 4, 6, 'Geeks', 'For'}
```

Dictionary Data Type in Python

A dictionary in Python is an unordered collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element, a Dictionary holds a key: value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a 'comma'

Create a Dictionary in Python

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable.

```
Dict = {}
print("Empty Dictionary: ")
print(Dict)
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print("\nDictionary with the use of Integer Keys: ")
```

Output:

```
Empty Dictionary:
{}
Dictionary with the use of Integer Keys:
{1: 'Geeks', 2: 'For', 3: 'Geeks'}
```

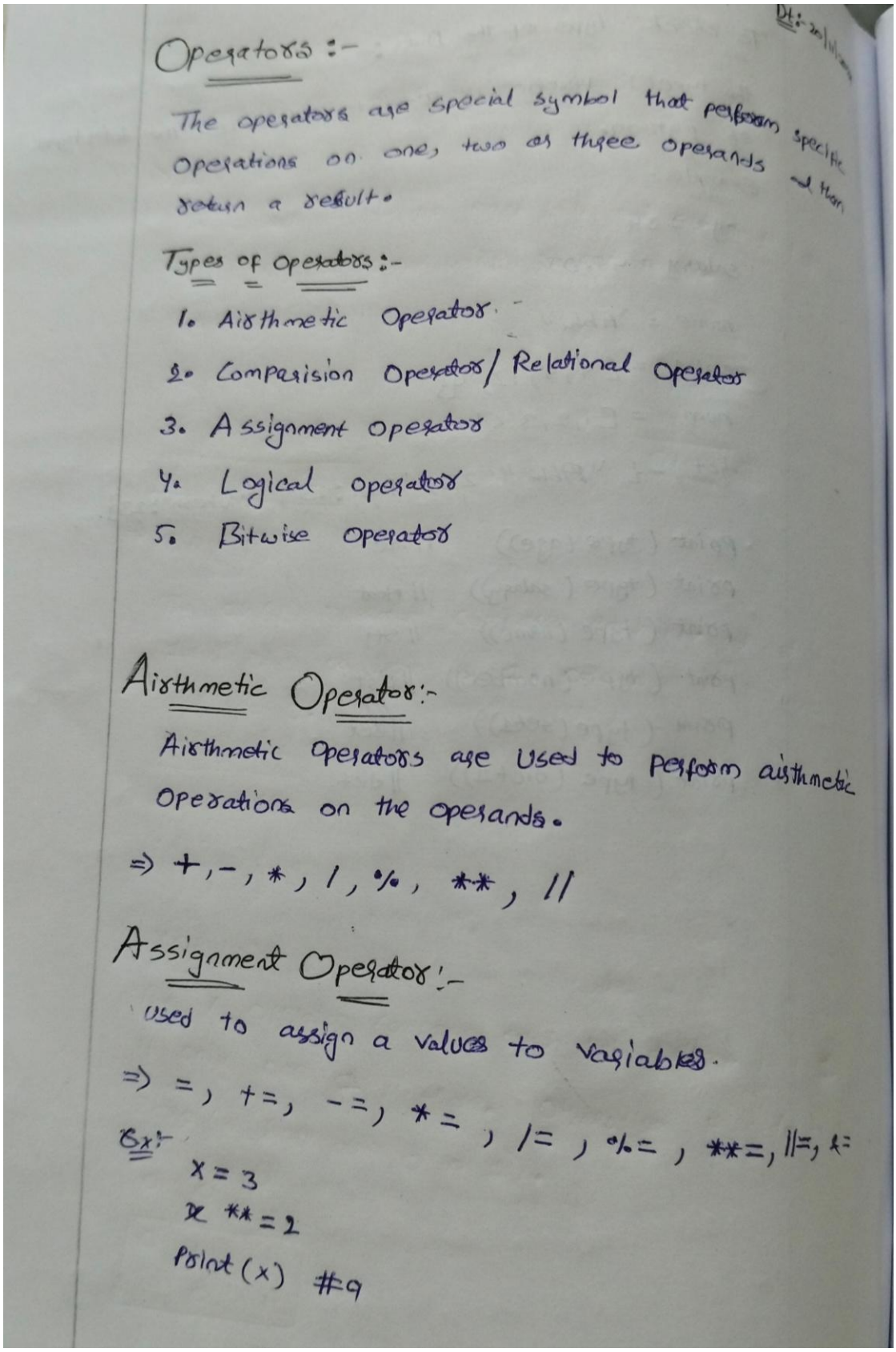
To check type of the Data :-

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

Example :-

```
age = 34
salary = 45.00
name = "Abhi"
set1 = {4, 5, 12, 4}
num = [1, 2, 3, 6]
dict = {"Abhi": 23, "Ravi": 23, "Mukesh": 23}
print (type (age))          // int
print (type (salary))       // float
print (type (name))         // str
print (type (numbers))      // List
print (type (set1))         // set
print (type (dict1))        // dict.
```

Operators



Operators :-

The operators are special symbol that perform specific operations on one, two or three operands and then return a result.

Types of operators :-

1. Airthmetic Operator
2. Comparision Operator / Relational Operator
3. Assignment operator
4. Logical operator
5. Bitwise operator

Airthmetic Operator :-

Airthmetic Operators are used to perform airthmetic Operations on the operands.

⇒ +, -, *, /, %, **, //

Assignment Operator :-

used to assign a values to variables.

⇒ =, +=, -=, *=, /=, %=, **=, //=, &=

Ex :-

```
x = 3
x **= 2
Print(x)  #9
```

Relational / comparision Operators:-

- Used to compare values. They return a boolean value True or False.
- ==, >, <, !=, >=, <=

Logical Operator:-

Logical operator returns a boolean value by Evaluating boolean Expression.

→ and
→ or
→ not.

Logical And:-

			o/p
⇒	true	true	true
⇒	true	False	False
⇒	False	true	false
⇒	false	false	false

Logical OR:-

			o/p
⇒	true	true	true
⇒	true	false	true
⇒	false	true	true
⇒	false	false	false.

Logical Not:-

	o/p
true	False
False	true

Bitwise Operator:-

Bitwise operators are used to compare (binary) numbers.

→ AND (&)

The operator compare each bit and set it to 1 if both are 1 otherwise it set to 0.

		o/p
0	0	0
0	1	0
1	0	0
1	1	1

Ex:- 6&3

6 = 110

3 = 011

010 → (2)

⇒ OR (|)

The OR (|) operator compares each bit and set it to 1 if one or both is 1. Otherwise it returns 0 set to 0.

		o/p
0	0	0
0	1	1
1	0	1
1	1	1

⇒ XOR (^)

The ^ operator compares each bit and set it to 1 if only one '1' is there. Otherwise it sets to zero (if both are 0 or 1).

		o/p
0	0	0
0	1	1
1	0	1
1	1	0

NOT (~)

The ~ operator inverts each bit (0 becomes 1 and 1 becomes 0)

	o/p
0	1
1	0

Ex:- 3 becomes -4

~~3 = 0 0 1 1~~

Left shift (<<)

The << operator inserts the specified number of 0's from the right and let the same amount of leftmost bits fall off.

Ex:- 3<<2

3 = 0000000000000011

12 = 0000000000001100

Right shift (>>) :-

The >> operator moves each bit the specified number of times to the right. Empty holes at the left are filled with 0's.

Ex:- 8 = 0000 0000 0000 1000

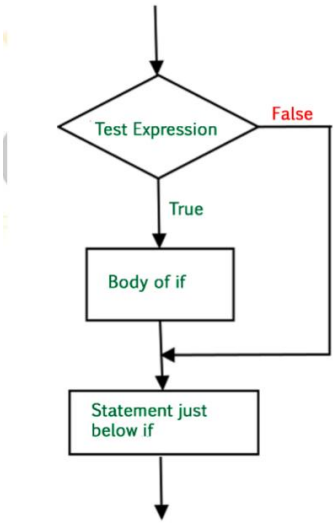
2 = 0000 0000 0000 0010

Conditional statements

If Statement:

The if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not.

Flowchart of If Statement Let's look at the flow of code in the Python If statements



Syntax of If Statement:

```
if condition:
   # Statements to execute if
   # condition is true
```

Example:

```
# if statement example
if 10 > 5:
   print("10 greater than 5")
print("Program ended")
```

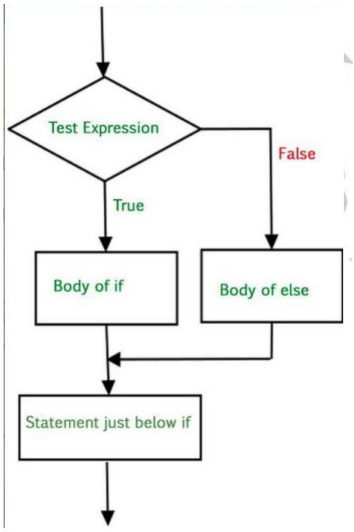
Output:

```
10 greater than 5
Program ended
```

Python If Else Statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But if we want to do something else if the condition is false, we can use the else statement with the if statement Python to execute a block of code when the Python if condition is false

Flowchart of If Else Statement Let's look at the flow of code in an if else Python statement.



Syntax of Python If-Else:

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

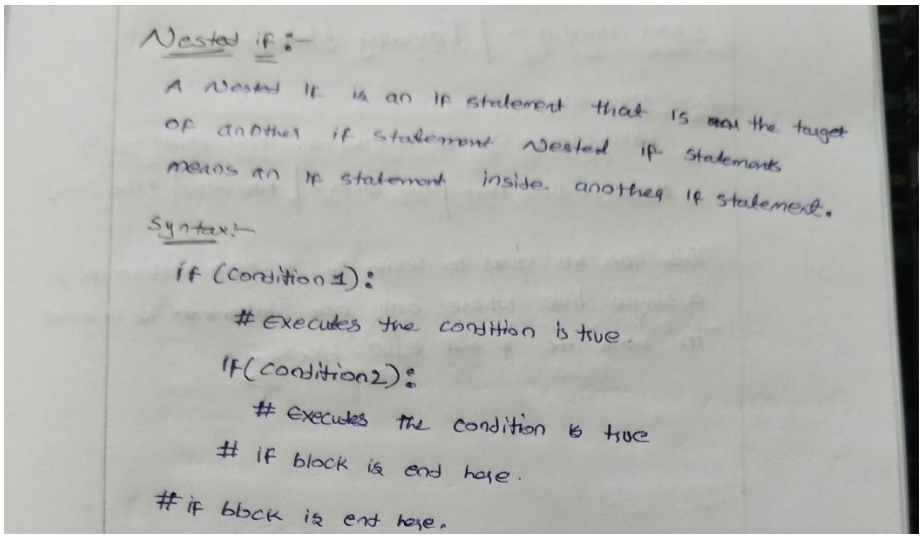
Example:

```
# if..else statement example
x = 3
if x == 4:
    print("Yes")
else:
    print("No")
```

Output: No

Python Nested If Statement

A nested if is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement.



Elif statement :-

Here, a uses can decide among multiple options. The if statements are executed from the top down.

Syntax :-

```
if (condition):
    statement
elif (condition):
    statement
    .
    .
    .
else:
    statement
```

Note :- if statement cannot be empty use pass statement to avoid getting an error.

```
if (condition):
    pass
```

Short Hand if :- / Ternary statement

Ex :- num = 11

print ("even") if num%2==0 else print("odd")

true condition block / true condition / false condition / false block.

This can be used to write the if-else statements in a single line whose only one statement is needed in both the if and else blocks.

LOOPS

For loop:-

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set or a string).

- This is less like the for keyword in other programming language, and works more like an iterator method as found in other object oriented programming languages.

Ex:-
```
# print each fruit in a fruit list
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

Ex:-
```
#Looping through a string
for x in "banana":
    print(x)
```

⇒ **Break:-** with the break statement we can stop the loop before it has looped through all the items.

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

⇒ **Continue:-** it can stop the current iteration of the loop, and continue with the next.

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

The range( ) function:-

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

Examples:-

1.
```
for x in range (6):
    Print (x)
```
↳ end number

O/P: 0, 1, 2, 3, 4, 5

Example-2:-

```
for x in range(2, 6):
    Print(x)
```
↓ start ↳ end.

Example-3:-

```
for x in range (3, 30, 2):
    print(x)
```
↓ start ↓ end ↓ Increment.

Nested Loops:-

- A nested loop is a loop inside a loop.
- The inner loop will be executed one time for each iteration of the "outer loop".

Example:-

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
    for y in fruits:
        print (x, y)
```

While loop:-

While loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition become false, the line immediately after the loop in the program is executed.

Syntax:-

```
While Expression:
    Statement (s)
```

Example:-

```
count = 0
while (count < 3):
    count = count + 1
    Print ("Hello")
```

Example:-

```
for i in range (1, 6, 1):    # 1 2 3 4 5
    for j in range (1, 11, 1): # 1 2 3 4 5 6 7 8 9 10
        Print (i, "x", j, "=", i * j)
```

Formatted String:-

```
name = "Abhi"
Print ("hello {x}")    # hello Abhi
```

↓

formated string

Functions

Python Functions

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

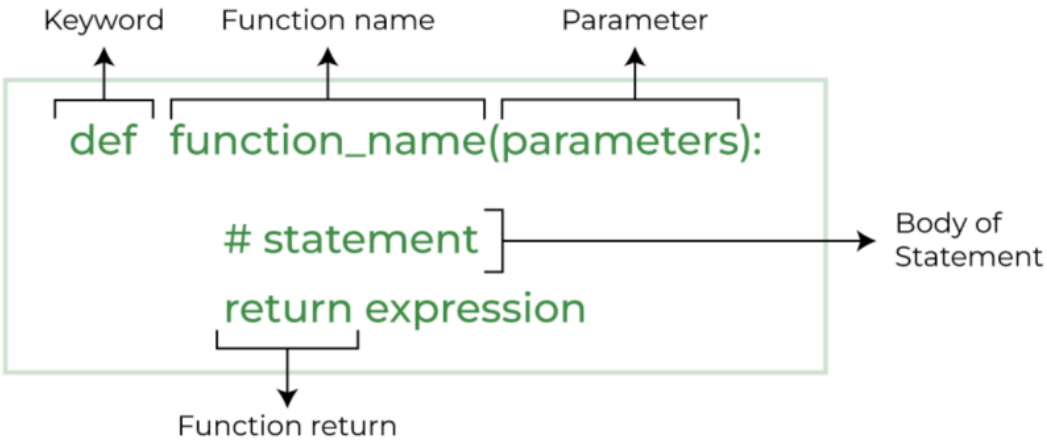
Some Benefits of Using Functions

Increase Code Readability

Increase Code Reusability

Python Function Declaration

The syntax to declare a function is:



Types of Functions in Python

• Below are the different types of functions in Python:

Built-in library function: These are Standard functions in Python that are available to use.

User-defined function: We can create our own functions based on our requirements.

Creating a Function in Python

We can define a function in Python, using the def keyword. We can add any type of functionalities and properties to it as we require. By the following example, we can understand how to write a function in Python. In this way we can create Python function definition by using def keyword.

Syntax:

```
# A simple Python function
def fun():
    print("Welcome to GFG")
```

Calling a Function in Python

After creating a function in Python we can call it by using the name of the functions Python followed by parenthesis containing parameters of that particular function. Below is the example for calling def function Python.

Example:

```
# A simple Python function
def fun():
    print("Welcome to GFG")

# Driver code to call a function
fun()
```

Python Function Arguments

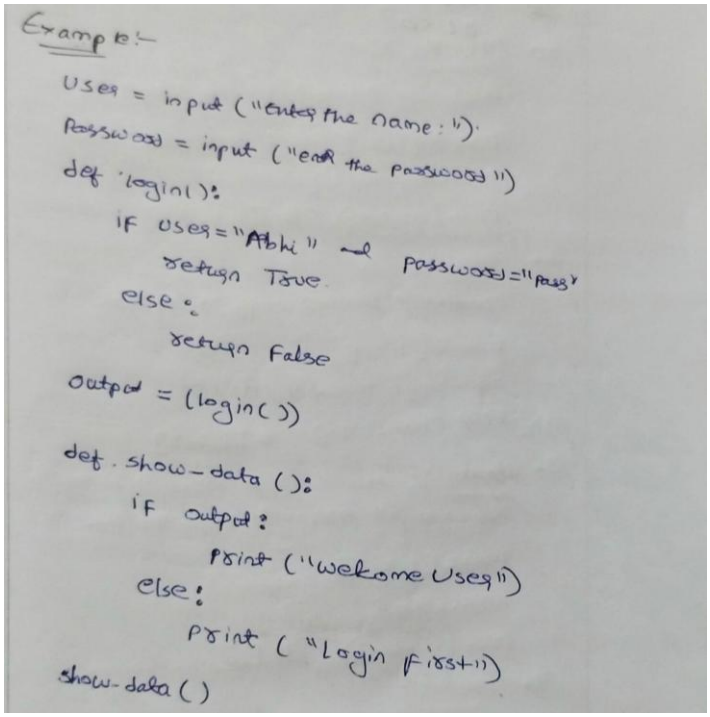
Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

In this example, we will create a simple function in Python to check whether the number passed as an argument to the function is even or odd.

Example:

```
# A simple Python function to check
# whether x is even or odd
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")

# Driver code to call the function
evenOdd(2)
evenOdd(3)
```



Python Default arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

Example: We call myFun() with the only argument.

```
# Python program to demonstrate
# default arguments
def myFun(x, y = 50):
    print("x: ", x)
    print("y: ", y)

# Driver code
myFun(10)
```

Output:

```
x:  10
y:  50
```

Taking input in Python

input () function first takes the input from the user and converts it into a string. The type of the returned object always will be <class 'str'>. It does not evaluate the expression it just returns the complete statement as String.

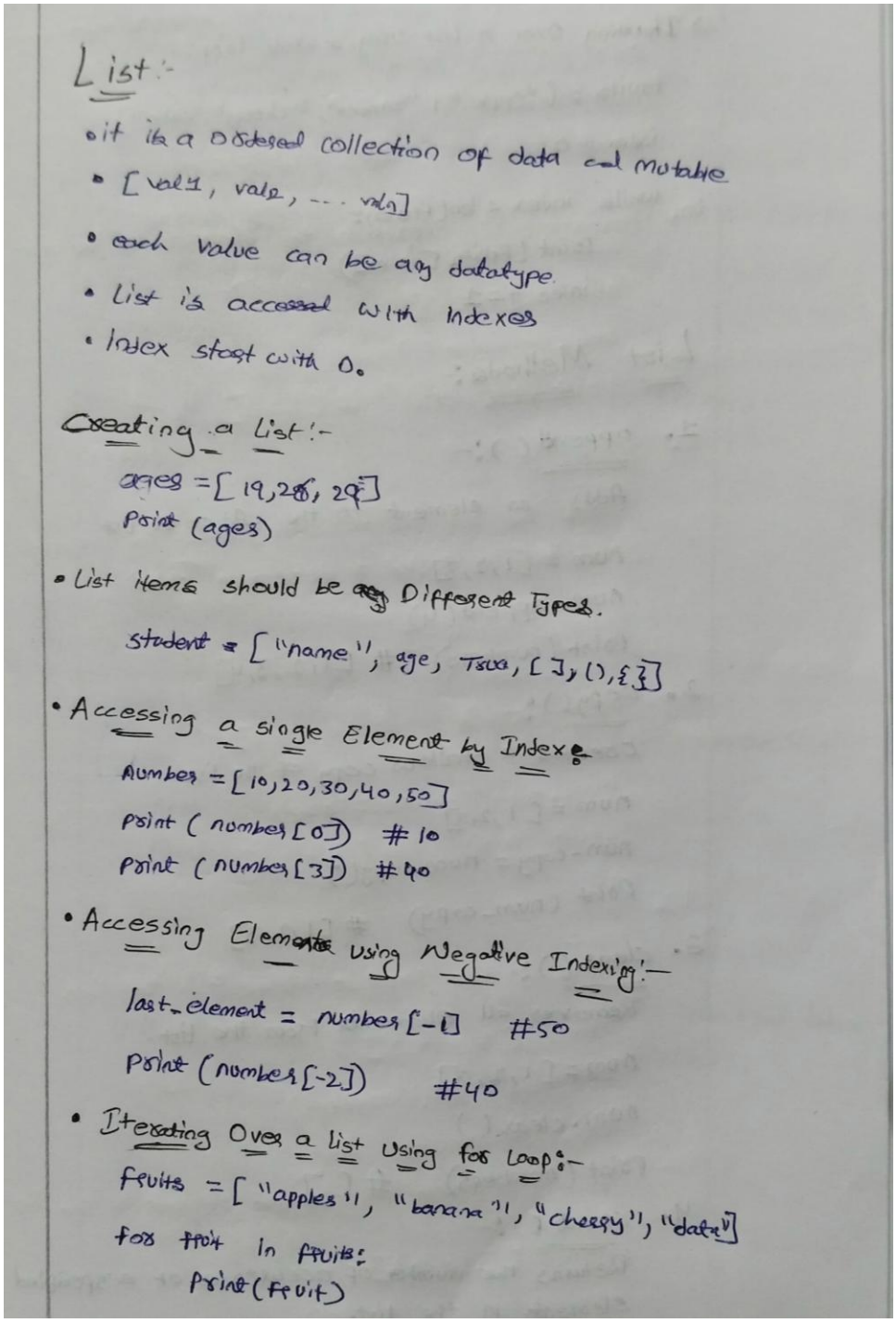
Example-1:

```
# Python program showing
# a use of input()
val = input("Enter your value: ")
print(val)
```

Example-2:

```
username = input("Enter username:")
print("Username is: " + username)
```

List Methods



List :-

- it is a Ordered collection of data and mutable
- [val1, val2, ... valn]
- each value can be any datatype
- List is accessed with indexes
- Index start with 0.

Creating a List :-

```
ages = [19, 28, 29]
print (ages)
```

- List items should be Different Types.

```
student = ["name", age, True, [], (), {}]
```

- Accessing a single Element by Index :-

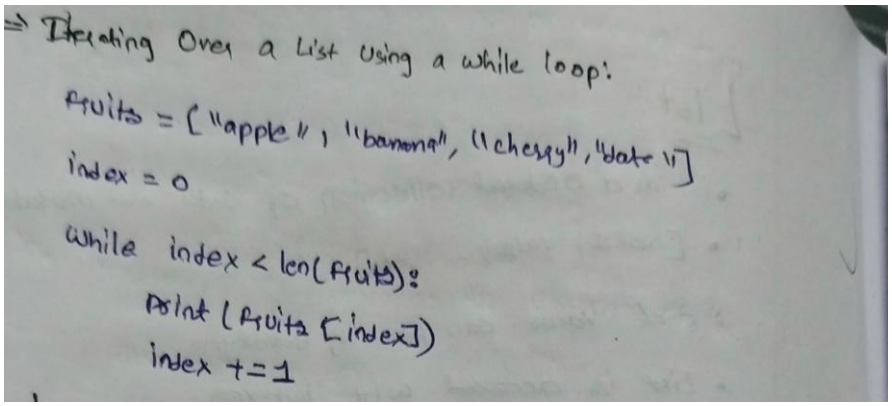
```
numbers = [10, 20, 30, 40, 50]
print (numbers[0])   # 10
print (numbers[3])   # 40
```

- Accessing Elements using Negative Indexing :-

```
last_element = numbers[-1]   #50
print (numbers[-2])          #40
```

- Iterating Over a list using for loop :-

```
fruits = ["apples", "banana", "cherry", "date"]
for fruit in fruits:
    print(fruit)
```



⇒ Iterating Over a List using a while loop :

```
fruits = ["apple", "banana", "cherry", "date"]
index = 0

while index < len(fruits):
    print (fruits[index])
    index += 1
```

List Methods:-

1. append():-

Adds an element to the end of the list

```
num = [1,2,3]
num.append(4)
print(numbers)  # [1,2,3,4]
```

2. copy():-

Creates a shallow copy of the list.

```
num = [1,2,3]
num_copy = num.copy()
print(num_copy)  # [1,2,3]
```

3. clear():-

Removes all elements from the list.

```
num = [1,2,3]
num.clear()
print(numbers)  # []
```

4. count():-

Returns the number of occurences of a specified element in the list.

```
num = [1,2,2,3]
count_of_two = num.count(2)
print(count_of_two)  #2
```

5. Extend() :-

Adds elements of an iterable (like another list) to the end of the list.

```
num = [1,2,3]
num.extend([4,5])
print(numbers)  # [1,2,3,4,5]
```

6. index() :-

Returns the index of the first occurrence of a specified element.

```
num = [1,2,3,2]
index_of_two = numbers.index(2)
print(index_of_two)  #1
```

7. Insert() :-

Inserts an element at a specific position in the list

```
num = [1,2,3]
num.insert(2,3)
print(num)  # [1,2,3,4]
```

8. pop() :-

Removes and returns the element at the specified index (or the last element if no index is specified).

```
num = [1,2,3]
last_ele = num.pop()
Print(last_ele)  #3
Print(num)  # [1,2]
```

9. remove( ) :- removes the first occurrence of a specified element.

```
Num = [1,2,3,2]
num.remove(2)
Print (num)        # [1,3,2]
```

10. reverse( ) :- Reverses the elements of the list.

```
num = [1,2,3]
num.reverse( )
Print (numbers)   # [3,2,1]
```

11. Sort( ) :-

sorts the list in ascending order by default (can also be sorted in descending order)

```
num = [3,1,4,2]
num = sort( )
Print (num)        # [1,2,3,4]

num.sort (reverse=True)
Print (num)        # [4,3,2,1]
```

12. min( ) :-

Returns the smallest element in the list.

```
num = [3,1,4,2]
minimum = min (numbers)
Print (minimum)   #1
```

13. Max( ) :- Returns the largest element in the list

```
num = [3,1,4,2]
maximum = max (numbers)
Print (maximum)   #4
```

List Comprehension:-

List comprehension provides a concise way to create lists in python. It's a shorthand for looping through an iterable and applying an expression to each item.

Example:- Creates a list of squares.

```
squares = [x**2 for x in range(5)]
print(squares)  #[0,1,4,9,16]
```

Example:- Create a list of even numbers.

```
even = [x for x in range(10) if x%2==0]
print(evens)  #[0,2,4,6,8]
```

Nested Lists:-

A nested list is a list that contains other lists as its elements. You can access element of a nested list using indexing.

Example:- simple nested list

```
matrix = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
print(matrix[1][2])  #6
```

Example:- Iterates through a nested list

```
for row in matrix:
    for element in row:
        Print(element, end=" ")
    Print()
```

O/P:-

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Tuple methods

Tuples:-

- Tuple is similar to list (It is ordered collection of objects) but main difference is immutable.

⇒ creating a tuple:-

```
tup1 = (2, "Hello", "Python")
print (tup1)
```

⇒ Tuple characteristics:-

- ordered
- immutable
- Allow duplicates.

⇒ Access Tuple Items:-

- Index value starts with zero (0)

Ex:-

```
Lang = ("Python", 'C', "C++")
Print (lang[0])  #Python
Print (lang[2])  #C++
```

⇒ Tuples cannot be modified:-

It will get an error.

```
Lang[2] = "Java";  # error
```

⇒ Iterate through a tuple:-

```
fruits = ("apple", "banana", "orange")
for fruit in fruits:
    print(fruit).
```

Python Tuple Methods :-

1. count() :- it returns the number of times the specified element appears in the tuple

Ex :-

```
vowels = ('a', 'e', 'i', 'o', 'i', 'u')
count = vowels.count('i')
print(count)  #2
```

2. Index() :- returns the index of specified element in the tuple.

Ex :-

```
index = vowels.index('e')
print(index)  #1
```

Nested Tuples :-

Nested tuples are tuples that contain other tuples as their element. They can be thought as a multidimensional tuples.

Characteristics of Nested Tuples :-

- Immutability
- Accessing Elements
- Fixed structure.

Ex :-

```
nested = (
        (1, 2, 3),
        (4, 5, 6),
        (7, 8, 9)
    )

#Accessing Nested Tuples.
print(nested[0][0])  #1
```

Find a num in Matrix :-

```
matrix = [(1,2,3),(4,5,6),(2,8,9)]
num = 7
exists = False
for i in matrix:
    for j in i:
        if (j == num):
            exists = True

if (exists):
    print("True")
else:
    print("False")
```

Dictionary Methods

Dictionagi

- Stores the element in key value pairs.
- they are enclosed in {} curly braces / Hauge braces.
- key should be always string
- we can store multiple data types in dict.

Example :-

```
thisdict = {
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
    }
print(thisdict)
```

=> Dictionary Item :-

- dictionary items are ordered, changable, and do not allow duplicates.

Ex:-

```
thisdict = {
        "brand" : "Ford",
        "Model" : "Mustang",
        "year" : 1964
    }
print ( thisdict ["brand"])
```

O/P :- Ford.

=> changable

```
thisdict ["year"] = 1965
```

=> Duplicates not allowed

It automatically removes past element.

=> Dictionary length:-

```
print (len (thisdict))   #3
```

=> Accessing Items :-

```
thisdict = {
        "brand" : "Ford",
        "Model" : "Mustang",
        "year" : "1964"
    }
x = thisdict ["Model"]
(or)
x = thisdict.get ("Model")
# Mustang
```

=> Get keys :-

it returns list of keys in the dictionary

Ex:- x = thisdict.keys()

dict_keys (['brand', 'Model', 'year'])

=> Get values :-

It returns a list of all the values in the dictionary

Ex:-

x = thisdict.values()

dict_values (['Ford', 'Mustang', 1964])

=> Get Items :-

It returns each item in a dictionary, as tuples in a list.

Ex:-

x = thisdict.items()

dict_items ([('Brand', 'Ford'), ('Model', 'Mustang'), ('year', 1964)])

=> Update Ditionary :-

The update() method will update the dictionary with the items from the given argument.

Ex:-

```
thisdict = {
        "brand": "Ford",
        "Model": "Mustang",
        "year": 1964
    }
this dict.update ({"year": 2020})
```

Adding Items:-

Ex:- thisdict["color"] = "Red"

Removing Items:-

They are several methods to remove items from a dictionary.

Ex-1:- thisdict.pop("Model")

Ex-2:- thisdict.popItem("Model")

Ex-3:- del thisdict["Model"]

Ex-4:- del thisdict #delete the dictionary completely.

clear():- method empties the dictionary

Ex:- thisdict.clear()

{}

Loop through a dictionary:-

- you can loop through a dictionary by using a for loop.
- when looping through a dictionary, the default value are the keys of the dictionary, but there are methods to return values as well.

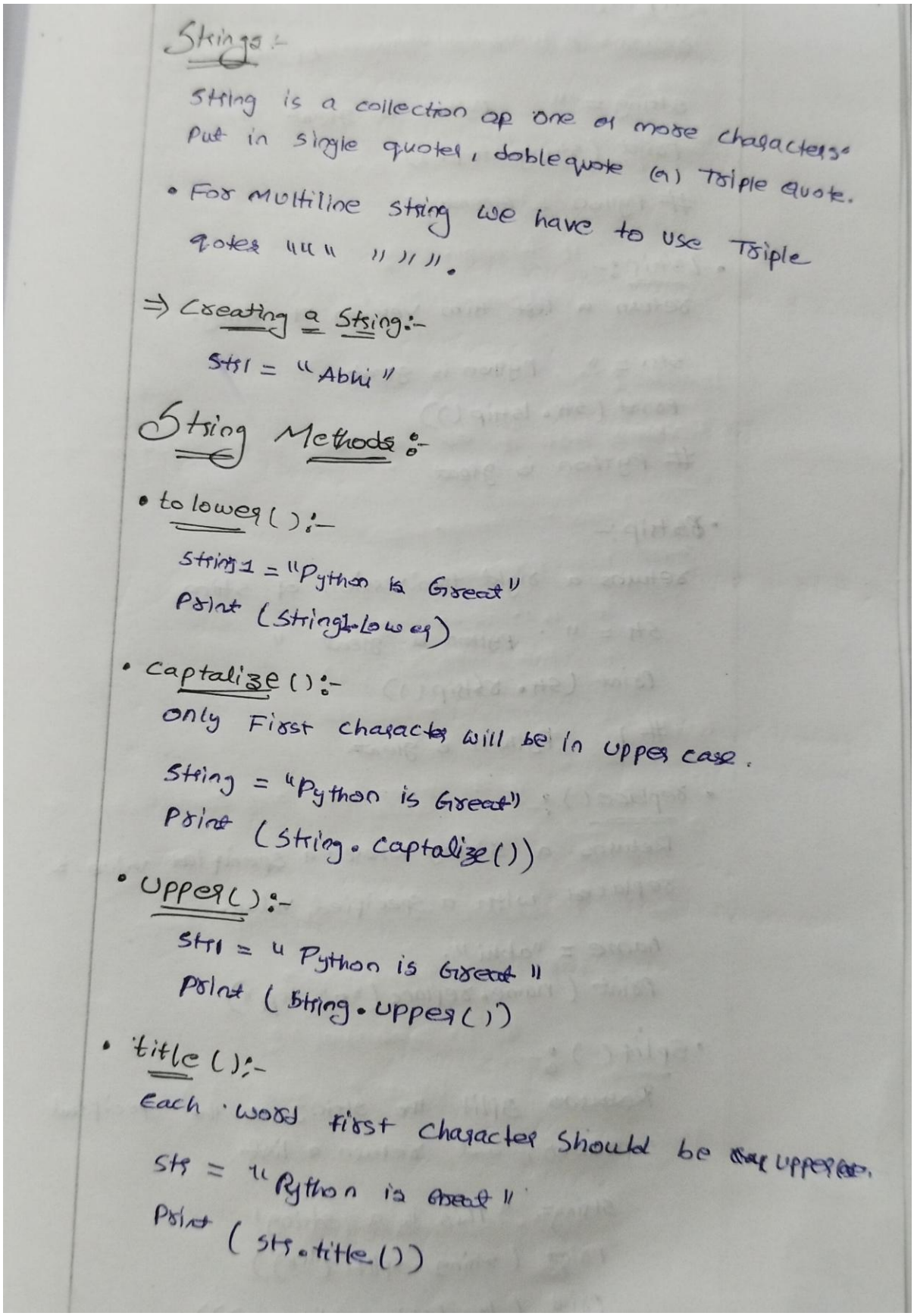
Ex:-

```
for x in thisdict:
    print(x)
    # Brand Model Year
```

Ex-2:-

```
for x in thisdict:
    print(thisdict[x])
# Ford Mustang 1964
```

String and string methods:



Strings :-

String is a collection of one or more characters put in single quotes, double quote (or) Triple quote.

- For Multiline string we have to use Triple quotes """ """ """.

⇒ Creating a String :-

```
Str1 = "Abhi"
```

String Methods :-

- to lower() :-

```
string1 = "Python is Great"
Print (string1.lower)
```

- captalize() :-

only First character will be in upper case.

```
String = "python is Great"
Print (String.captalize())
```

- Upper() :-

```
Str1 = " Python is Great "
Print (String.upper())
```

- title() :-

Each word first character should be upper.

```
Str = " Python is Great "
Print (Str.title())
```

- strip():-
returns a trimmed version of the string.

```
string = "    Python is great   "
print (string.strip())
# Python is Great.
```

- lstrip:-
return a left trim version of string.

```
str = "    python is great   "
print (str.lstrip())
# Python is great   "
```

- rstrip:-
returns a right trim version of string

```
str = "   Python is great   "
print (str.rstrip())
#    Python is great.
```

- replace():-
Returns a string where a specified value is replaced with a specified value.

```
name = "abhi";
print ( name.replace ("a", "A"))
```

- Split():-
~~Returns~~ split the string at the specified seperator, and return a list.

```
string= " This is a string"
print ( string .split (" "))
print ( len (string .split(" ")))
```

join():-

Joins the elements of an iterable to the end of the string.

```
String = "This is a string"
Print ("-".join(string.split(" ")))
```

Find():-

Searches the string for a specified value and returns the position of where it was found.

returns the first occurence of the index.

Ex:-

```
String = "this is string"
Print (string.Find("t"))   #0
```

Index():-

Searches the string for a specified value and returns the position of where it was found.

Ex:-

```
string = "String"
Print (string.index("s"))
```

startswith():- Returns true if the string starts with the specified value.

Ex:-

```
txt = "Hello, welcome to my world"
x = txt.startswith("Hello")
print (x)
```

endswith():- returns true if the string ends with the specified value.

```
name = Abhi
Print (name.endswith("i"))  #true.
```

isdigit():-

Returns true if all characters in the string are digit.

Ex:-

```
Password = "secret_123"
for i in Password:
    if i.isdigit():
        print ("valid", i)
```

isalpha():-

Returns true if all characters in the string are alpha

Ex:-

```
Password = "secret-123"
for i in password:
    if i.isalpha():
        print ("valid", i)
```

isalnum():-

Returns true if all characters are combination of alpha and numeric.

Ex:-

```
Password = "12345 alphabets"
for i in password:
    if i.isalnum():
        print ("valid", i)
```

isnumeric():-

Ex:-

```
txt.isnumeric()
```

isspace():-

return true if all characters in the string are whitespace.

Ex:-

```
txt = "   "
x = txt.isspace()
print(x).
```

swapcase:-

swapcase, lowercase become uppercase and vice versa.

Ex:-

```
name = "Abhi"
print(name.swapcase())
# aBHI
```

Zfill():-

Fills the string with a specified number of 0 values at the beginning.

Ex:-

```
txt = "50"
x = txt.zfill(10)
print(x)
# 0000000050
```

rjust:-

Returns a right justified version of a string.

Ex:-

```
txt = "banana"
x = txt.rjust(20)
print(x)  # banana.
```

ljust:- Returns a left justified version of a string.

Ex:-

```
txt = "banana"
x = txt.ljust(10)
print(x)  # banana
```

center():-

Returns a centered string.

Ex:-

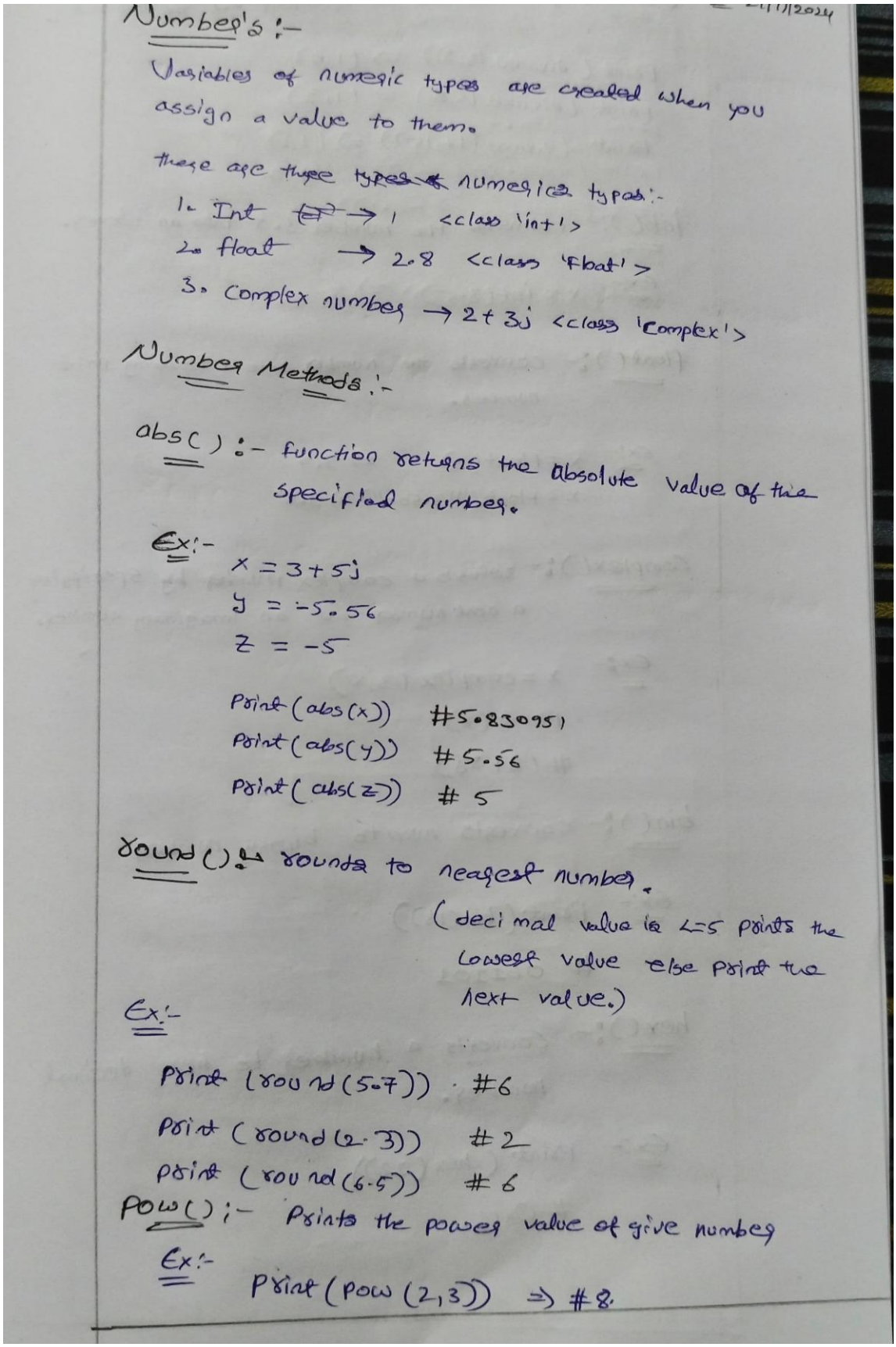
```
txt = "banana"
x = txt.center(10)
print(x)
```

O/P:- banana.

Example:-

```
String = "Python"
res = ""
for i in String:
    if i == i.upper():
        res = res + i.lower()
    else:
        res += i.upper()
Print(res).
```

Number and Number methods:



Numbers :-

Variables of numeric types are created when you assign a value to them.

these are three types of numerical types :-

1. Int → 1 <class 'int'>
2. float → 2.8 <class 'float'>
3. Complex number → 2+3j <class 'complex'>

Number Methods :-

abs() :- function returns the absolute value of the specified number.

Ex :-

```
x = 3+5j
y = -5.56
z = -5

Print(abs(x))   #5.830951
Print(abs(y))   #5.56
Print(abs(z))   #5
```

round() :- rounds to neagest number.
(decimal value is <=5 prints the lowest value else print the next value.)

Ex :-

```
Print(round(5.7))   #6
Print(round(2.3))   #2
print(round(6.5))   #6
```

pow() :- Prints the powers value of give number

Ex :-

```
Print(pow(2,3))  => #8.
```

divmod():- it returns the quotient and the remainder.

print (divmod (3,3)) => (1,0)
print (divmod (5,3)) => (1, 2)
print (divmod (17,15)) => (1,2)

int():- converts the number 3.5 into an integer.

Ex:- x = int(3.5) => 3

float():- converts the number into a floating point number.

Ex:- x = float(3) => 3.0
x = float("3.500") => 3.5

Complex():- returns a complex number by specifying a real number and an imaginary number.

Ex:-
x = complex(3,5)
print(x)
#(3+5j)

bin():- converts num to binary number.

Ex:- Print (bin(13))
0b1101

hex():- Converts a number to hexa decimal number.

Ex:- print (hex(22))
#0x16

oct():- converts a number into the oct number

Ex:-

```
print (oct(16))
#0020
```

Aggregate Functions:-

max():- returns the max value of list or tuple

Ex:-

```
a = (2,3,4,5,7,8,9)
print (max(a))
#9
```

Min():- returns the minimum value

```
a = (2,3,4,5,6,7,8)
print (min(a))
#2
```

Sum():- returns the sum value.

```
print (sum(2,10,8,3,7))
#30
```

Avg():-

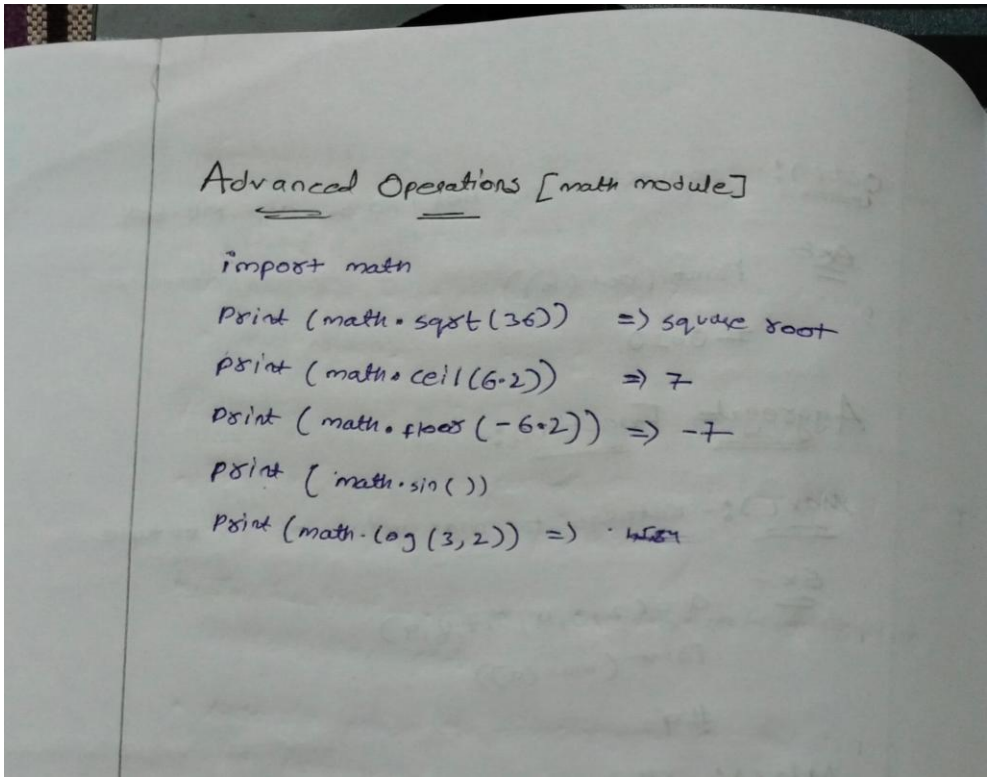
Example:-(without methods)

```
tup = (1,2,3,4)
max = 0

for i in tup:
    if i > max:
        max = i
print (max)
```

(or)

```
max = tup[0]
for i in tup:
    if i > max:
        max = i
print (max)
```



Advanced Operations [math module]

```
import math
Print (math.sqrt(36))      => square root
Print (math.ceil(6.2))     => 7
Print (math.floor(-6.2))   => -7
Print (math.sin())
Print (math.log(3,2))      =>
```

Modules:

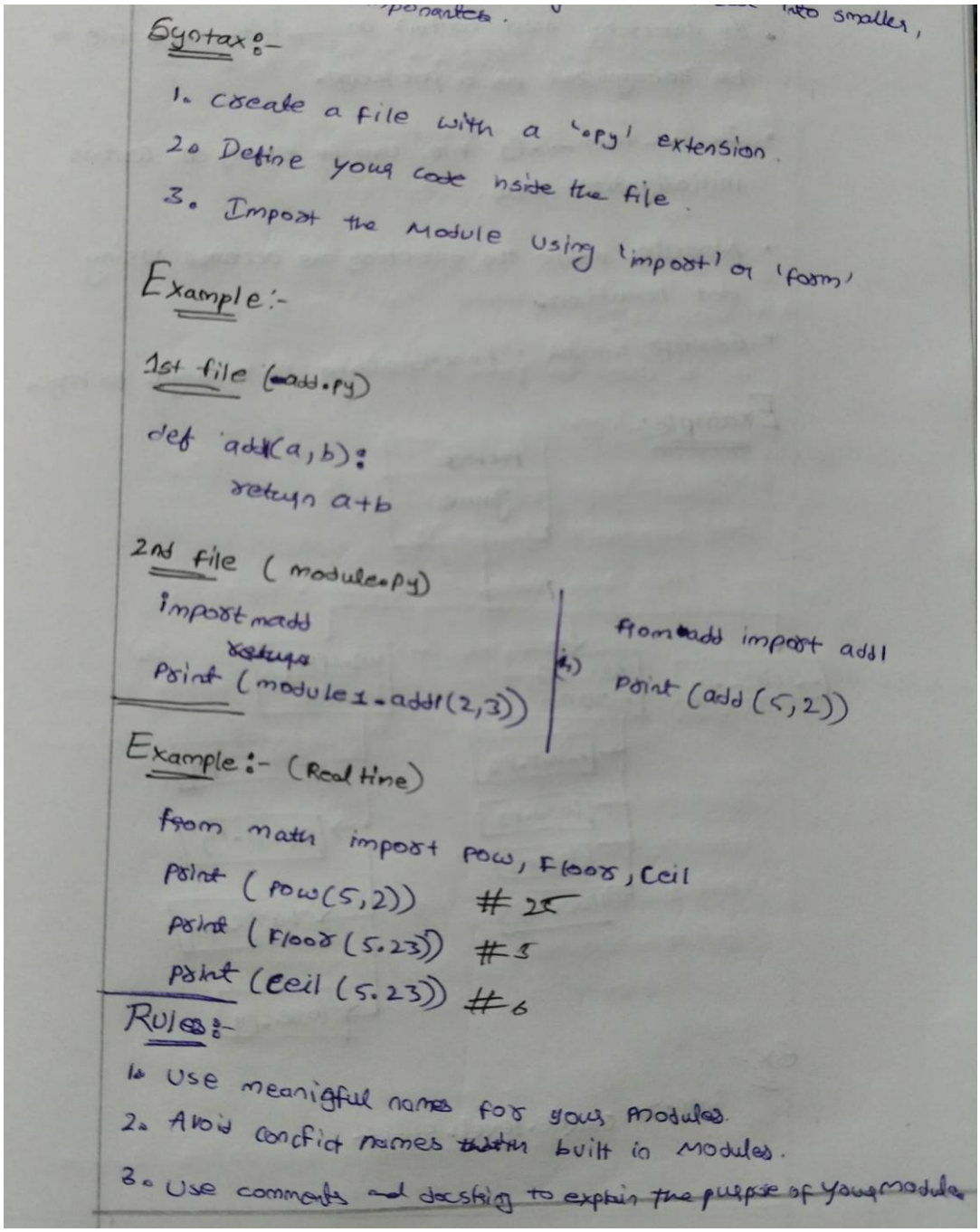
Python Module is a file that contains built-in functions, classes,**its** and variables. There are many **Python modules**, each with its specific work.

In this article, we will cover all about Python modules, such as How to create our own simple module, Import Python modules, From statements in Python, we can use the alias to rename the module, etc.

What is Python Module

A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code.

Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.



Packages:

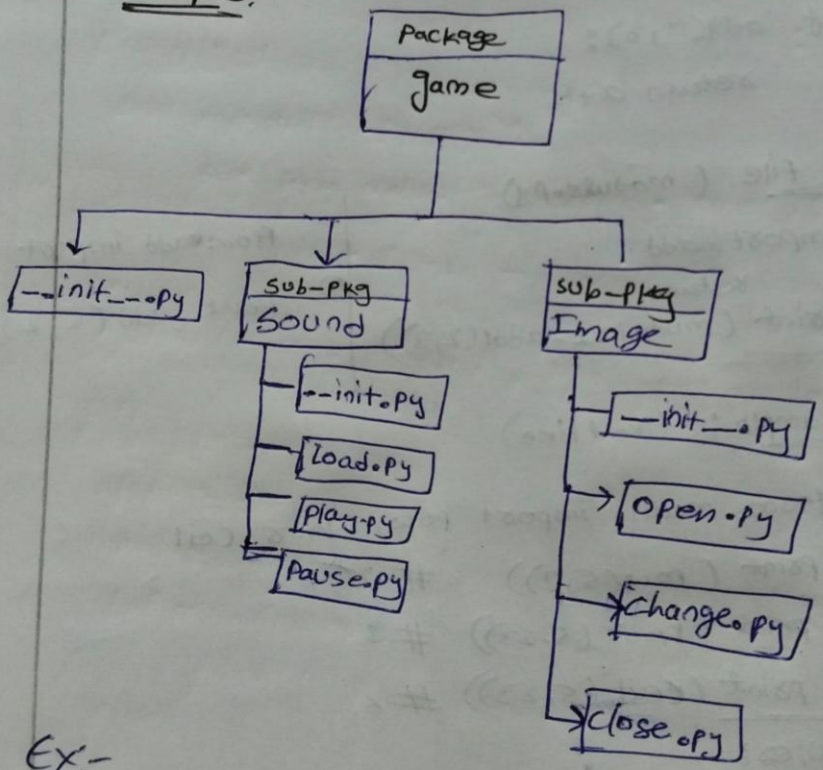
Package:-

A Package is a collection of modules organised in a dictionary.

Rules:-

- The directory must contain an __init__.py file to be recognized as a package.
- The __init__.py file can be empty or contain initialization code.
- Modules inside the directory are accessed using dot notation.
- Group similar functionalities in a single package.

Example:-



Ex:-

```
import Game.level.start
```

Example:- Startwith() method functionality

```
S = "Python is COOL"
chas = "py"
def start_with(S, chas):
    temp = ""
    for i in range(len(chas)):
        temp += S[i]

    if temp == chas:
        return True
    else:
        return False

Print(Start_with(S, chas))
```

Special Symbols used for Passing arguments:-

- *args (Non-keyword Argument)
- **kwargs (keyword Arguments)

*args:- it is used to pass a non-keyworded, variable-length argument list. (to access multiple values)

Example:1:-

```
def myFun(*argv):
    for arg in argv:
        Print(arg)

myFun('Hello', 'welcome', 'to', 'GFG').
```

O/P:-

```
Hello
welcome
to
GFG
```

Example-2:-

```
def myFun(arg1, *argv):
    print("1st Arg:", arg1)
    for arg in argv:
        print(arg)

myFun('Hello', 'welcome', 'to', 'GFG')
```

O/P:-

1st Arg: Hello
welcome
to
GFG

** kwargs:-

It is used to pass a keyworded, variable-length argument list.

Example-1:-

```
def myFun(**kwargs):
    for key, value in kwargs.items():
        print("%s == %s" %(key, value))

myFun(first='Geeks', mid='for', last='Geeks')
```

O/P:-

first == Geeks
mid == for
last == Geeks.

File Handling:

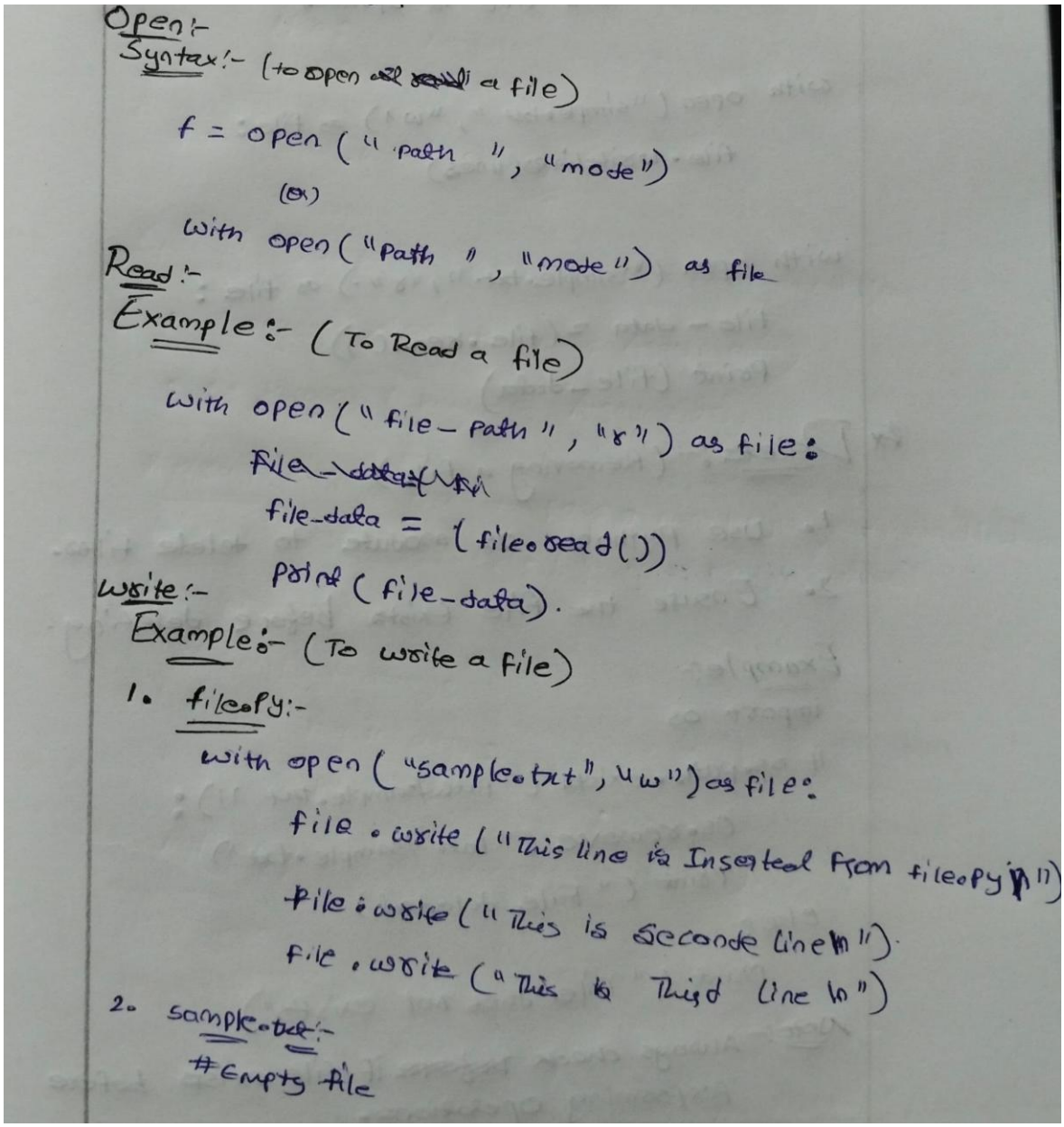
file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.

Advantages of File Handling in Python: Versatility **,** Flexibility**,** User – friendly and Cross-platform

File Handling

The key function for working with files in Python is the open() function.
The open() function takes two parameters; *filename*, and *mode*.
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



Read Lines :-

- readline() :- reads only one line
- readlines() :- reads all lines as a page

Update :-

Example :-

```
with open ("simple.txt", "r") as file:
    lines = file.readlines()
lines[1] = "This line is updated"

with open ("simple.txt", "w") as file:
    file.writelines(lines)

with open ("simple.txt", "r") as file:
    file_data = (file.read())
    Print (file_data)
```

Delete :- (Removing a file)

1. Use Python's os module to delete files.
2. Ensure the file exists before deleting.

Example :-

```
import os
if os.path.exists ("Path/sample.txt"):
    os.remove ("Path:/sample.txt")
    print ("File deleted")
else:
    print ("File does not exists")
```

Note :- Always check ~~before~~ if file exists before performing operations.

⇒ If we want to remove a particular line in a file just update the written line with empty string.

Example :-

```
with open ("path/simple.txt", "r") as file:
    lines = file.readlines()
Lines[i] = ""

with open ("path/simple.txt", "w") as file:
    file.writelines (lines)

with open ("path/simple.txt", "r") as file:
    file_data = (file.read())
    print (file_data)
```

File Paths :-

Relative path :- Refers to the file location relative to the script.

Ex :- 'subfolder/file.txt'

Absolute path :- Refers to the complete file location

Ex :- 'C:/Users/Name/Documents/file.txt'.

Example :- (Read function)

```
def read (f_name, m):
    with open (f_name, m) as file:
        temp = file.read()
        return temp.

print (read ("sample.text", "r"))
```

TASK :- Create a file perform CURD operations.

Regular Expressions