

THE **ULTIMATE** **PYTHON** HANDBOOK



 CodeWithHarry

PREFACE

Welcome to the “Ultimate Python Programming Handbook,” your comprehensive guide to mastering Python programming. This handbook is designed for beginners and anyone looking to strengthen their foundational knowledge of Python, a versatile and user-friendly programming language.

PURPOSE AND AUDIENCE

This handbook aims to make programming accessible and enjoyable for everyone. Whether you're a student new to coding, a professional seeking to enhance your skills, or an enthusiast exploring Python, this handbook will definitely be helpful. Python's simplicity and readability make it an ideal starting point for anyone interested in programming.

STRUCTURE AND CONTENT

The handbook is divided into clear, concise chapters, each focused on a specific aspect of Python:

- **Fundamental Concepts:** Start with the basics, such as installing Python and writing your first program.
- **Practical Examples:** Illustrative examples and sample code demonstrate the application of concepts.
- **Hands-On Exercises:** End-of-chapter exercises reinforce learning and build confidence.
- **Additional Resources:** References to official Python documentation for deeper exploration.

WHY PYTHON?

Python is known for its simplicity and readability, making it perfect for beginners. It is a high-level, interpreted language with a broad range of libraries and frameworks, supporting applications in web development, data analysis, AI, and more. Python's versatility and ease of use make it a valuable tool for both novice and experienced programmers.

ACKNOWLEDGEMENTS

I extend my gratitude to the educators, programmers, and contributors who have shared their knowledge and insights, shaping the content of this handbook. Special thanks to all the students watching my content on YouTube and Python community for maintaining a supportive and inspiring environment for learners worldwide.

CONCLUSION

Learning programming can be both exciting and challenging. The “Ultimate Python Programming Handbook” aims to make your journey smooth and rewarding. Watch my video along with following this handbook for optimal learning. Let this guide be your stepping stone to success in the world of programming.

CONTENTS

PREFACE	1
Purpose and Audience	1
Structure and Content.....	1
Why Python?.....	1
Acknowledgements	1
Conclusion	1
Contents	2
Python programming Handbook	6
What is Programming?	6
What is Python?	6
Features of Python	6
Installation	6
Chapter 1 – Modules, Comments & pip	7
Modules	7
pip	7
Types of Modules	7
Using python as a calculator	7
Comments	7
Types of Comments	7
Chapter 1 – Practice Set.....	9
Chapter 2 – Variables and Datatype	10
Data Types	10
Rules for choosing an identifier	10
Operators in Python	10
type() function and typecasting.....	11
input() Function	11
Chapter 2 – Practice Set.....	12
Chapter 3 – Strings	13
String Slicing.....	13
Slicing With Skip Value	14
String Functions.....	14
Escape Sequence Characters.....	15
Chapter 3 – Practice Set.....	16
Chapter 4 – Lists And Tuples	17
List Indexing	17

List methods.....	17
Tuples in Python	17
Tuple Methods.....	17
Chapter 4 - Practice Set	19
Chapter 5 – Dictionary & Sets	20
Properties of Python Dictionaries.....	20
Dictionary Methods.....	20
Sets in Python.....	20
Properties of Sets.....	21
Operations on sets.....	21
Chapter 5 – Practice Set.....	22
Chapter 6 – Conditional Expression	23
If Else and Elif in Python.....	23
Code example.	23
Relational Operators	24
Logical Operators	24
Elif clause.....	24
Important notes:	24
Chapter 6 – Practice Set.....	25
Chapter 7 – Loops in Python	26
Types of Loops in Python	26
While loop	26
For loop.....	27
range() Function in Python	27
An Example Demonstrating range() function.	27
For Loop with Else.....	27
The Break Statement	27
The Continue Statement.....	28
Pass statement.....	28
Chapter 7 – Practice Set.....	29
Chapter 8 – Functions & Recursions	30
Example and syntax of a function	30
Function call.....	30
Function Definition	30
Types of Functions in Python	30
Functions with Arguments	30
Default Parameter Value	31

Recursion	31
Chapter 8 – Practice Set.....	33
Project 1: Snake, Water, Gun Game.....	34
Chapter 9 – File I/O	35
Type of Files.....	35
Opening a File.....	35
Reading a File in Python.....	35
Other methods to read the file.	36
Modes of opening a file.....	36
Write Files in Python.....	36
With Statement.....	36
Chapter 9 – Practice Set.....	37
Chapter 10 - Object Oriented Programming	38
Class.....	38
Object	38
Modelling a problem in OOPs.....	38
Class Attributes	38
Instance attributes.....	39
self parameter	39
static method	39
__init__() constructor.....	40
Chapter 10 – Practice Set.....	41
Chapter 11 - Inheritance & more on OOPs.....	42
Types of Inheritance.....	42
Single Inheritance	42
Multiple Inheritance	43
Multilevel Inheritance.....	43
super() method	43
class method.....	44
@property Decorators	44
@.getters and @.setters	44
Operator Overloading in Python	44
Chapter 11- Practice set	46
Project 2 – The Perfect Guess	47
Chapter 12 – Advanced Python 1	48
Newly added features in python.....	48
Walrus Operator	48

Types Definitions in Python	48
Advanced Type Hints	48
Match Case	49
Dictionary Merge & Update Operators	49
Exception handling in Python	50
Raising Exceptions	50
try with else clause	50
try with finally	51
If <code>__name__ == '__main__'</code> in python	51
The global keyword	51
enumerate function in python	51
List comprehensions	51
Chapter 12 – Practice set	52
Chapter 13 – Advanced Python 2	53
Virtual enviroinment	53
Installation	53
pip freeze command	53
Lambda functions	53
join method (strings)	54
format method (strings)	54
Map, Filter & Reduce	54
Chapter 13 – Practice Set	56
MEGA Project 1: Jarvis	57
Features	57
Workflow	57
Libraries Used	58
Mega Project 2: Auto Reply AI Chatbot	59
Description	59
Features	59
Workflow	59
Libraries Used	60

PYTHON PROGRAMMING HANDBOOK

WHAT IS PROGRAMMING?

Just like we use Hindi or English to communicate with each other, we use a programming language like Python to communicate with the computer.

Programming is a way to instruct the computer to perform various tasks.

WHAT IS PYTHON?

Python is a simple and easy to understand language which feels like reading simple English. This Pseudo code nature is easy to learn and understandable by beginners.

FEATURES OF PYTHON

- Easy to understand = Less development time
- Free and open source
- High level language
- Portable: Works on Linux / Windows / Mac.
- Fun to work with!

INSTALLATION

Python can be easily installed from python.org. When you click on the download button, python can be installed right after you complete the setup by executing the file for your platform.

**Just install
it like a game**



CHAPTER 1 – MODULES, COMMENTS & PIP

Let's write our very first python program. Create a file called hello.py and paste the below code in it.

```
print("hello world") # print is a function (more later)
```

Execute this file (.py file) by typing python hello.py and you will see Hello World printed on the screen.

MODULES

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs.

PIP

Pip is the package manager for python. You can use pip to install a module on your system.

```
pip install flask # Installs Flask Module
```

TYPES OF MODULES

There are two types of modules in Python.

1. Built in Modules (Preinstalled in Python)
2. External Modules (Need to install using pip)

Some examples of built in modules are os, random etc.

Some examples of external modules are tensorflow, flask etc.

USING PYTHON AS A CALCULATOR

We can use python as a calculator by typing "python" + ↵ on the terminal.

This opens **REPL** or Read Evaluate Print Loop.

COMMENTS

Comments are used to write something which the programmer does not want to execute. This can be used to mark author name, date etc.

TYPES OF COMMENTS

There are two types of comments in python.

1. Single Line Comments: To write a single line comment just add a '#' at the start of the line.

```
# This is a Single-Line Comment
```

2. Multiline Comments: To write multi-line comments you can use '#' at each line or you can use the multiline string ('''' ''')

```
"""This is an amazing  
example of a Multiline  
comment!"""
```

CodeWithHarry

CHAPTER 1 – PRACTICE SET

1. Write a program to print Twinkle twinkle little star poem in python.
2. Use REPL and print the table of 5 using it.
3. Install an external module and use it to perform an operation of your interest.
4. Write a python program to print the contents of a directory using the os module.
Search online for the function which does that.
5. Label the program written in problem 4 with comments.

CodeWithHarry

CHAPTER 2 – VARIABLES AND DATATYPE

A variable is the name given to a memory location in a program. For example.

```
a= 30      # variables = container to store a value.  
b= "harry" # keywords = reserved words in python  
c= 71.22   # identifiers = class/function/variable name
```

DATA TYPES

Primarily these are the following data types in Python:

1. Integers
2. Floating point numbers
3. Strings
4. Booleans
5. None

Python is a fantastic language that automatically identifies the type of data for us.

```
a= 71      # identifies a as class <int>  
b=88.44    # identifies b as class <float>  
name= "harry" # identifies name as class <str>
```

RULES FOR CHOOSING AN IDENTIFIER

- A variable name can contain alphabets, digits, and underscores.
- A variable name can only start with an alphabet and underscores.
- A variable name can't start with a digit.
- No while space is allowed to be used inside a variable name.

Examples of a few variable names are: harry, one8, seven, _seven etc.

OPERATORS IN PYTHON

Following are some common operators in python:

1. Arithmetic operators: +, -, *, / etc.
2. Assignment operators: =, +=, -= etc.
3. Comparison operators: ==, >, >=, <, != etc.
4. Logical operators: and, or, not.

TYPE() FUNCTION AND TYPECASTING.

type() function is used to find the data type of a given variable in python.

```
a = 31
type(a) # class <int>

b = "31"
type(b) # class <str>
```

A number can be converted into a string and vice versa (if possible)

There are many functions to convert one data type into another.

```
str(31)    => "31"    # integer to string conversion
int("32")  => 32      # string to integer conversion
float(32)  => 32.0    # integer to float conversion
```

... and so, on

Here "31" is a string literal and 31 a numeric literal.

INPUT () FUNCTION

This function allows the user to take input from the keyboard as a string.

```
A = input ("enter name") # if a is "harry", the user entered harry
```

It is important to note that the output of input is always a string (even is a number is entered).

CHAPTER 2 – PRACTICE SET

1. Write a python program to add two numbers.
2. Write a python program to find remainder when a number is divided by z.
3. Check the type of variable assigned using input () function.
4. Use comparison operator to find out whether 'a' given variable a is greater than 'b' or not. Take a = 34 and b = 80
5. Write a python program to find an average of two numbers entered by the user.
6. Write a python program to calculate the square of a number entered by the user.

CodeWithHarry

CHAPTER 3 – STRINGS

String is a data type in python.

String is a sequence of characters enclosed in quotes.

We can primarily write a string in these three ways.

```
a = 'harry'      # Single quoted string
b = "harry"      # Double quoted string
c = '''harry'''  # Triple quoted string
```

STRING SLICING

A string in python can be sliced for getting a part of the strings.

Consider the following string:

Name= "harry" => Length = 5

h	a	r	r	y
0	1	2	3	4
(-5)	(-4)	(-3)	(-2)	(-1)

The index in a string starts from 0 to (length - 1) in Python. In order to slice a string, we use the following syntax:

sl = name[ind_start:ind_end]

first index included

last index is not included

sl [0:3] returns "har" ———> characters from 0 to 3

sl [1:3] returns "ar" ———> characters from 1 to 3

Negative Indices: Negative indices can also be used as shown in the figure above. -1 corresponds to the (length - 1) index, -2 to (length - 2).

SLICING WITH SKIP VALUE

We can provide a skip value as a part of our slice like this:

```
word = "amazing"
word[1: 6: 2] # "mzn"
```

Other advanced slicing techniques:

```
Word = "amazing"
Word = [:7] # word [0:7] - 'amazing'
Word = [0:] # word [0:7] - 'amazing'
```

STRING FUNCTIONS

Some of the commonly used functions to perform operations on or manipulate strings are as follows. Let us assume there is a string 'str' as follows:

```
str = 'harry'
```

Now when operated on this string 'str', these functions do the following:

1. len () function – This function returns the length of the strings.

```
str = "harry"
print(len(str)) # Output: 5
```

2. String.endswith("rry") – This function tells whether the variable string ends with the string "rry" or not. If string is "harry", it returns true for "rry" since Harry ends with rry.

```
str = "harry"
print(str.endswith("rry")) # Output: True
```

3. string.count("c") – counts the total number of occurrences of any character.

```
str = "harry"
count = str.count("r")
print(count) # Output: 2
```

4. the first character of a given string.

```
str = "harry"
capitalized_string = str.capitalize()
print(capitalized_string) # Output: "Harry"
```

5. string.find(word) – This function finds a word and returns the index of first occurrence of that word in the string.

```
str = "harry"
```

```
index = str.find("rr")  
print(index) # Output: 2
```

6. `string.replace (old word, new word)` – This function replace the old word with new word in the entire string.

```
str = "harry"  
replaced_string = str.replace("r", "l")  
print(replaced_string) # Output: "hally"
```

ESCAPE SEQUENCE CHARACTERS

Sequence of characters after backslash "\" → **Escape Sequence characters**

Escape Sequence characters comprise of more than one character but represent one character when used within the strings.

Example: \n, \t, \', \\ etc.

↙ ↘ ↙ ↘
newline Tab Singlequote backslash

CHAPTER 3 – PRACTICE SET

1. Write a python program to display a user entered name followed by Good Afternoon using input () function.
2. Write a program to fill in a letter template given below with name and date.

```
letter = '''
Dear <|Name|>,
You are selected!
<|Date|>
'''
```

3. Write a program to detect double space in a string.
4. Replace the double space from problem 3 with single spaces.
5. Write a program to format the following letter using escape sequence characters.

```
letter = "Dear Harry, this python course is nice. Thanks!"
```

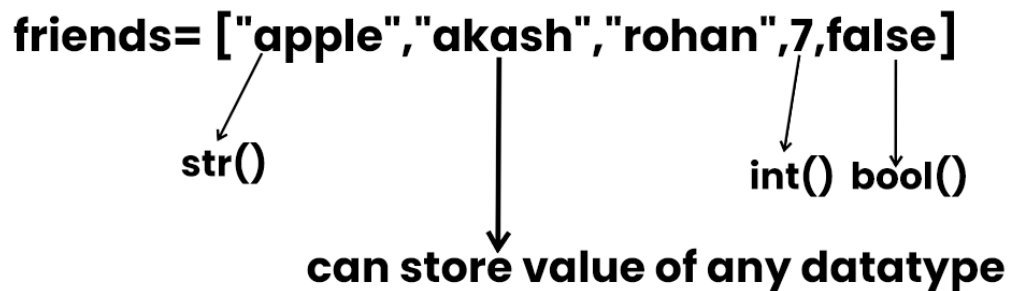
CHAPTER 4 – LISTS AND TUPLES

Python lists are containers to store a set of values of any data type.

friends= ["apple","akash","rohan",7,false]

str() **int()** **bool()**

can store value of any datatype



LIST INDEXING

A list can be indexed just like a string.

```
l1 = [7,9,"harry"]  
l1[0] # 7  
l1[1] # 9  
l1[70] # error  
l1[0:2] # [7,9] #list slicing
```

LIST METHODS.

Consider the following list:

```
l1 = [1,8,7,2,21,15]
```

- l1.sort(): updates the list to [1,2,7,8,15,21]
- l1.reverse(): updates the list to [15,21,2,7,8,1]
- l1.append(8): adds 8 at the end of the list
- l1.insert(3,8): This will add 8 at 3 index
- l1.pop(2): Will delete element at index 2 and return its value.
- l1.remove(21): Will remove 21 from the list.

TUPLES IN PYTHON

A tuple is an immutable data type in python.

```
a = () # empty tuple  
a = (1,) # tuple with only one element needs a comma  
a = (1,7,2) # tuple with more than one element
```

TUPLE METHODS

Consider the following tuple.

```
a = (1, 7, 2)
```

- `a.count(1)`: a count (1) will return number of times 1 occurs in a.
- `a.index(1)` will return the index of first occurrence of 1 in a.

CodeWithHarry

CHAPTER 4 - PRACTICE SET

1. Write a program to store seven fruits in a list entered by the user.
2. Write a program to accept marks of 6 students and display them in a sorted manner.
3. Check that a tuple type cannot be changed in python.
4. Write a program to sum a list with 4 numbers.
5. Write a program to count the number of zeros in the following tuple:

```
a = (7, 0, 8, 0, 0, 9)
```

CodeWithHarry

CHAPTER 5 – DICTIONARY & SETS

Dictionary is a collection of keys-value pairs.

Syntax:

```
a = {
    "key": "value",
    "harry": "code",
    "marks": "100",
    "list": [1, 2, 9]
}

print(a["key"]) # Output: "value"
print(a["list"]) # Output: [1, 2, 9]
```

PROPERTIES OF PYTHON DICTIONARIES

1. It is unordered.
2. It is mutable.
3. It is indexed.
4. Cannot contain duplicate keys.

DICTIONARY METHODS

Consider the following dictionary.

```
a={"name":"harry"
   "from":"india"
   "marks":[92,98,96]}
```

- a.items(): Returns a list of (key,value)tuples.
- a.keys(): Returns a list containing dictionary's keys.
- a.update({"friends":}): Updates the dictionary with supplied key-value pairs.
- a.get("name"): Returns the value of the specified keys (and value is returned eg."harry" is returned here).

More methods are available on docs.python.org

SETS IN PYTHON.

Set is a collection of non-repetitive elements.

```
s = set() # no repetition allowed!
s.add(1)
s.add(2) # or set ={1,2}
```

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

PROPERTIES OF SETS

1. Sets are unordered => Element's order doesn't matter
2. Sets are unindexed => Cannot access elements by index
3. There is no way to change items in sets.
4. Sets cannot contain duplicate values.

OPERATIONS ON SETS

Consider the following set:

```
s = {1,8,2,3}
```

- `len(s)`: Returns 4, the length of the set
- `s.remove(8)`: Updates the set s and removes 8 from s.
- `s.pop()`: Removes an arbitrary element from the set and return the element removed.
- `s.clear()`: empties the set s.
- `s.union({8,11})`: Returns a new set with all items from both sets. {1,8,2,3,11}.
- `s.intersection({8,11})`: Return a set which contains only item in both sets {8}.



$$\begin{aligned}R2 &= A \cap B \\ R1 + R2 + R3 &= A \cup B \\ R1 + R3 &= A \Delta B \\ R1 &= A - B \\ R3 &= B - A\end{aligned}$$

CHAPTER 5 – PRACTICE SET

1. Write a program to create a dictionary of Hindi words with values as their English translation. Provide user with an option to look it up!
2. Write a program to input eight numbers from the user and display all the unique numbers (once).
3. Can we have a set with 18 (int) and '18' (str) as a value in it?
4. What will be the length of following set s: $\rightarrow 3$

```
s = set()
s.add(20)
s.add(20.0)
s.add('20') # length of s after these operations?
```

5. `s = {}`
What is the type of 's'? $\rightarrow \emptyset$
6. Create an empty dictionary. Allow 4 friends to enter their favorite language as value and use key as their names. Assume that the names are unique.
7. If the names of 2 friends are same; what will happen to the program in problem 6?
8. If languages of two friends are same; what will happen to the program in problem 6?
9. Can you change the values inside a list which is contained in set S?

```
s = {8, 7, 12, "Harry", [1,2]}
```

No

CHAPTER 6 – CONDITIONAL EXPRESSION

Sometimes we want to play PUBG on our phone if the day is Sunday.

Sometimes we order Ice Cream online if the day is sunny.

Sometimes we go hiking if our parents allow.

All these are decisions which depend on a condition being met.

In python programming too, we must be able to execute instructions on a condition(s) being met.

This is what conditionals are for!

IF ELSE AND ELIF IN PYTHON

If else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax:

```
if (condition1): # if condition1 is True
    print ("yes")

elif(condition2): # if condition2 is True
    print("no")

else:             # otherwise
    print("maybe")
```

CODE EXAMPLE.

```
a=22
if(a>9):
    print("greater")
else:
    print("lesser")
```

Quick Quiz: Write a program to print yes when the age entered by the user is greater than or equal to 18.

RELATIONAL OPERATORS

Relational Operators are used to evaluate conditions inside the if statements. Some examples of relational operators are:

`==`: equals.

`>=`: greater than/ equal to.

`<=`: lesser than/ equal to.

LOGICAL OPERATORS

In python logical operators operate on conditional statements. For Example:

- `and` – true if both operands are true else false.
- `or` – true if at least one operand is true or else false.
- `not` – inverts true to false & false to true.

ELIF CLAUSE

`elif` in python means [else if]. An if statements can be chained together with a lot of these `elif` statements followed by an `else` statement.

```
if (condition1):  
    #code  
elif (condition2):    # this ladder will stop once a condition in an if or  
elif is met.  
    #code  
elif(condition3):  
    #code  
else:  
    #code
```

IMPORTANT NOTES:

1. There can be any number of `elif` statements.
2. Last `else` is executed only if all the conditions inside `elifs` fail.

CHAPTER 6 – PRACTICE SET

1. Write a program to find the greatest of four numbers entered by the user.
2. Write a program to find out whether a student has passed or failed if it requires a total of 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.
3. A spam comment is defined as a text containing following keywords: “Make a lot of money”, “buy now”, “subscribe this”, “click this”. Write a program to detect these spams.
4. Write a program to find whether a given username contains less than 10 characters or not.
5. Write a program which finds out whether a given name is present in a list or not.
6. Write a program to calculate the grade of a student from his marks from the following scheme:
90 – 100 => Ex
80 – 90 => A
70 – 80 => B
60 – 70 => C
50 – 60 => D
<50 => F
7. Write a program to find out whether a given post is talking about “Harry” or not.

CHAPTER 7 – LOOPS IN PYTHON

Sometimes we want to repeat a set of statements in our program. For instance: Print 1 to 1000.

Loops make it easy for a programmer to tell the computer which set of instructions to repeat and how!

TYPES OF LOOPS IN PYTHON

Primarily there are two types of loops in python.

- while loops
- for loops

also study for i--

We will look into these one by one.

WHILE LOOP

Syntax:

```
while (condition): # The block keeps executing until the condition is true
    #Body of the loop
```

In while loops, the condition is checked first. If it evaluates to true, the body of the loop is executed otherwise not!

If the loop is entered, the process of [condition check & execution] is continued until the condition becomes False.

Quick Quiz: Write a program to print 1 to 50 using a while loop.

Example:

```
i = 0
while i < 5: # print "Harry" - 5 times!
    print("Harry")
    i = i + 1
```

Note: If the condition never become false, the loop keeps getting executed.

Quick Quiz: Write a program to print the content of a list using while loops.

```
# Decremental for loop example
for i in range(10, 0, -1):
    # Starts from 10, decrements by -1, and stops at 1
    print(i)
```

FOR LOOP

A for loop is used to iterate through a sequence like list, tuple, or string [iterables]

Syntax:

```
l = [1, 7, 8]
for item in l:
    print(item) # prints 1, 7 and 8
```

RANGE FUNCTION IN PYTHON

The range() function in python is used to generate a sequence of number.

We can also specify the start, stop and step-size as follows:

```
range(start, stop, step_size)
# step_size is usually not used with range()
```

AN EXAMPLE DEMONSTRATING RANGE () FUNCTION.

```
for i in range(0,7): # range(7) can also be used.
    print(i) # prints 0 to 6
```

FOR LOOP WITH ELSE

An optional else can be used with a for loop if the code is to be executed when the loops exhausts.

Example:

```
l= [1,7,8]
for item in l:
    print(item)
else:
    print("done") # this is printed when the loop exhausts!
```

Output:

```
1
7
8
done
```

THE BREAK STATEMENT

'break' is used to come out of the loop when encountered. It instructs the program to – exit the loop now.

Example:

```
for i in range (0,80):  
    print(i)      # this will print 0,1,2 and 3  
    if i==3  
        break
```

THE CONTINUE STATEMENT

‘continue’ is used to stop the current iteration of the loop and continue with the next one. It instructs the Program to “skip this iteration”.

Example:

```
for i in range(4):  
    print("printing")  
    if i == 2:  # if i is 2, the iteration is skipped  
        continue  
    print(i)
```

PASS STATEMENT

pass is a null statement in python.

It instructs to “do nothing”.

Example:

```
l = [1,7,8]  
for item in l:  
    pass          # without pass, the program will throw an error
```

```
# Decremental while loop example  
i = 10 # Initialize starting value  
while i > 0:  
    # Run the loop until i is greater than 0  
    print(i)  
    i -= 1 # Decrement i by 1
```

CHAPTER 7 – PRACTICE SET

1. Write a program to print multiplication table of a given number using for loop.
2. Write a program to greet all the person names stored in a list 'l' and which starts with S.
`l = ["Harry", "Soham", "Sachin", "Rahul"]`
3. Attempt problem 1 using while loop.
4. Write a program to find whether a given number is prime or not.
5. Write a program to find the sum of first n natural numbers using while loop.
6. Write a program to calculate the factorial of a given number using for loop.
7. Write a program to print the following star pattern.

```
*  
***  
***** for n = 3
```

8. Write a program to print the following star pattern:

```
*  
**  
*** for n = 3
```

9. Write a program to print the following star pattern.

```
* * *  
* * for n = 3  
* * *
```

10. Write a program to print multiplication table of n using for loops in reversed order.

No, Python does not have a native do...while

In a do...while loop, the code block is guaranteed to execute at least once before the condition is checked. You can mimic this behavior in Python by using a while loop with a break statement, like this:

```
# Simulate do...while loop  
while True:  
    user_input = input("Enter a number  
greater than 10: ")  
    if int(user_input) > 10:  
        print("You entered:", user_input)  
        break  
    print("Try again!") # This is executed if  
the condition is not met
```

CHAPTER 8 – FUNCTIONS & RECURSIONS

A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a program to keep track on which piece of code is doing what!

A function can be reused by the programmer in a given program any number of

EXAMPLE AND SYNTAX OF A FUNCTION

The syntax of a function looks as follows:

```
def func1():  
    print('hello')
```

This function can be called any number of times, anywhere in the program.

FUNCTION CALL

Whenever we want to call a function, we put the name of the function followed by parentheses as follows:

```
func1() # This is called function call.
```

FUNCTION DEFINITION

The part containing the exact set of instructions which are executed during the function call.

Quick Quiz: Write a program to greet a user with “Good day” using functions.

TYPES OF FUNCTIONS IN PYTHON

There are two types of functions in python:

- Built in functions (Already present in python)
- User defined functions (Defined by the user)

Examples of built in functions includes len(), print(), range() etc.

The func1() function we defined is an example of user defined function.

FUNCTIONS WITH ARGUMENTS

A function can accept some value it can work with. We can put these values in the parentheses.

A function can also return value as shown below:

```
def greet(name):
    gr = "hello"+ name
    return gr

a = greet ("harry")
# a will now contain "hello harry"
```

DEFAULT PARAMETER VALUE

We can have a value as default as default argument in a function.

If we specify name = “stranger” in the line containing def, this value is used when no argument is passed.

Example:

```
def greet(name = "stranger"):
    # function body
greet() # name will be "stranger" in function body (default)
greet("harry") # name will be "harry" in function body (passed)
```

RECURSION

Recursion is a function which calls itself.

It is used to directly use a mathematical formula as function.

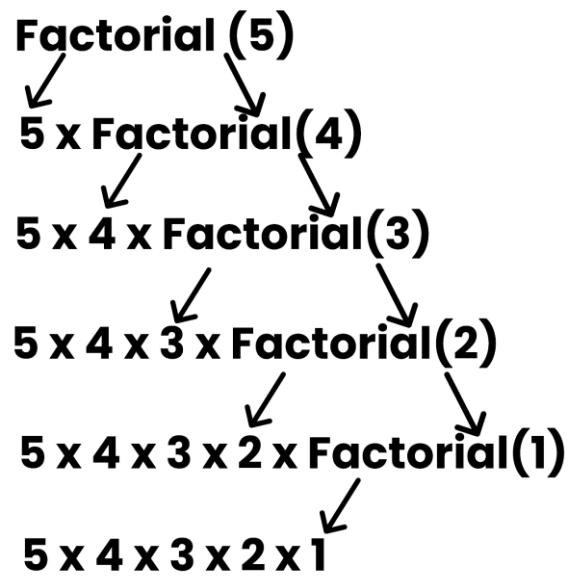
Example:

```
factorial(n) = n x factorial (n-1)
```

This function can be defined as follows:

```
def factorial(n)
    if i == 0 or i==1: # base condition which doesn't call the function
    any further
    return 1
else:
    return n*factorial(n-1) # function calling itself
```


This works as follows:



The programmer needs to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep calling itself. Recursion is sometimes the most direct way to code an algorithm.

CHAPTER 8 – PRACTICE SET

1. Write a program using functions to find greatest of three numbers.
2. Write a python program using function to convert Celsius to Fahrenheit.
3. How do you prevent a python print() function to print a new line at the end.
4. Write a recursive function to calculate the sum of first n natural numbers.
5. Write a python function to print first n lines of the following pattern:

** - for n = 3
*

6. Write a python function which converts inches to cms.
7. Write a python function to remove a given word from a list and strip it at the same time.
8. Write a python function to print multiplication table of a given number.

PROJECT 1: SNAKE, WATER, GUN GAME

We all have played snake, water gun game in our childhood. If you haven't, google the rules of this game and write a python program capable of playing this game with the user.

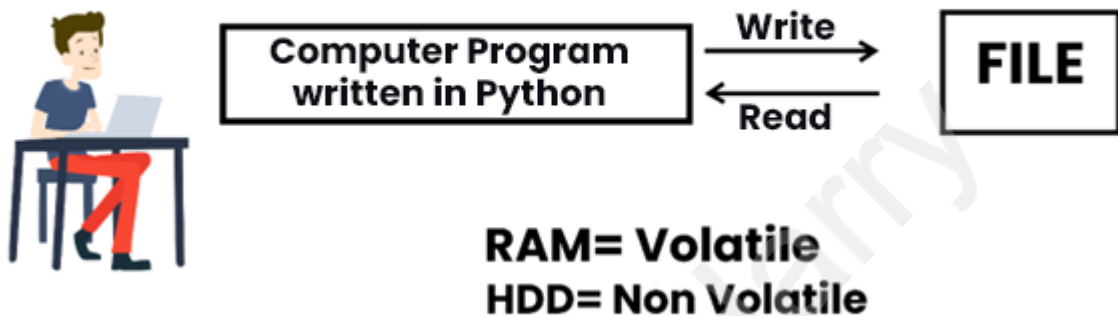
CodeWithHarry

CHAPTER 9 – FILE I/O

The random-access memory is volatile, and all its contents are lost once a program terminates. In order to persist the data forever, we use files.

A file is data stored in a storage device. A python program can talk to the file by reading content from it and writing content to it.

Programmer



TYPE OF FILES.

There are 2 types of files:

1. Text files (.txt, .c, etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating, and deleting files.

OPENING A FILE

Python has an `open()` function for opening files. It takes 2 parameters: filename and mode.

```
# open("filename", "mode of opening(read mode by default)")
open("this.txt", "r")
```

READING A FILE IN PYTHON

```
# Open the file in read mode
f = open("this.txt", "r")
# Read its contents
text = f.read()
# Print its contents
print(text)
```

```
# Close the file
f.close()
```

OTHER METHODS TO READ THE FILE.

We can also use `f.readline()` function to read one full line at a time.

```
f.readline() # Read one line from the file.
```

MODES OF OPENING A FILE

r – open for reading

w - open for writing

a - open for appending

+ - open for updating.

‘rb’ will open for read in binary mode.

‘rt’ will open for read in text mode.

WRITE FILES IN PYTHON

In order to write to a file, we first open it in write or append mode after which, we use the python’s `f.write()` method to write to the file!

```
# Open the file in write mode
f = open("this.txt", "w")
# Write a string to the file
f.write("this is nice")
# Close the file
f.close()
```

WITH STATEMENT

The best way to open and close the file automatically is the with statement.

```
# Open the file in read mode using 'with', which automatically closes the
file
with open("this.txt", "r") as f:
    # Read the contents of the file
    text = f.read()

# Print the contents
print(text)
```

CHAPTER 9 – PRACTICE SET

1. Write a program to read the text from a given file 'poems.txt' and find out whether it contains the word 'twinkle'.
2. The game() function in a program lets a user play a game and returns the score as an integer. You need to read a file 'Hi-score.txt' which is either blank or contains the previous Hi-score. You need to write a program to update the Hi-score whenever the game() function breaks the Hi-score.
3. Write a program to generate multiplication tables from 2 to 20 and write it to the different files. Place these files in a folder for a 13 – year old.
4. A file contains a word "Donkey" multiple times. You need to write a program which replace this word with ##### by updating the same file.
5. Repeat program 4 for a list of such words to be censored.
6. Write a program to mine a log file and find out whether it contains 'python'.
7. Write a program to find out the line number where python is present from ques 6.
8. Write a program to make a copy of a text file "this. txt"
9. Write a program to find out whether a file is identical & matches the content of another file.
10. Write a program to wipe out the content of a file using python.
11. Write a python program to rename a file to "renamed_by_ python.txt."

CHAPTER 10 - OBJECT ORIENTED PROGRAMMING

Solving a problem by creating object is one of the most popular approaches in programming. This is called object-oriented programming.

This concept focuses on using reusable code (DRY Principle).

CLASS

A class is a blueprint for creating object.



Syntax:

```
class Employee: # Class name is written in pascal case
    # Methods & Variables
```

OBJECT

An object is an instantiation of a class. When class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user. – **Abstractions & Encapsulation!**

MODELLING A PROBLEM IN OOPS

We identify the following in our problem.

- Noun → **Class** → **Employee**
- Adjective → **Attributes** → **name, age, salary**
- Verbs → **Methods** → **getSalary(), increment()**

CLASS ATTRIBUTES

An attribute that belongs to the class rather than a particular object.

Example:

```
class Employee:
    company = "Google" # Specific to Each Class
harry = Employee() # Object Instatiation
harry.company
Employee.company = "YouTube" # Changing Class Attribute
```

INSTANCE ATTRIBUTES

An attribute that belongs to the Instance (object). Assuming the class from the previous example:

```
harry.name = "harry"
harry.salary = "30k" # Adding instance attribute
```

Note: Instance attributes, take preference over class attributes during assignment & retrieval.

When looking up for harry.attribute it checks for the following:

- 1) Is attribute present in object?
- 2) Is attribute present in class?

SELF PARAMETER

self refers to the instance of the class. It is automatically passed with a function call from an object.

```
harry.getSalary() # here self is harry
# equivalent to Employee.getSalary(harry)
```

The function getSalary() is defined as:

```
class Employee:
    company = "Google"
    def getSalary(self):
        print("Salary is not there")
```

STATIC METHOD

Sometimes we need a function that does not use the self-parameter. We can define a static method like this:

```
@staticmethod # decorator to mark greet as a static method
def greet():
    print("Hello user")
```


__INIT__() CONSTRUCTOR

__init__() is a special method which is first run as soon as the object is created.

__init__() method is also known as constructor.

It takes 'self' argument and can also take further arguments.

For Example:

```
class Employee:
    def __init__(self, name):
        self.name=name
    def getSalary(self):
        ...

harry = Employee("Harry")
```

CHAPTER 10 – PRACTICE SET

1. Create a class “*Programmer*” for storing information of few programmers working at Microsoft.
2. Write a class “*Calculator*” capable of finding square, cube and square root of a number.
3. Create a class with a class attribute a; create an object from it and set ‘a’ directly using ‘object.a = 0’. Does this change the class attribute?
4. Add a static method in problem 2, to greet the user with hello.
5. Write a Class ‘Train’ which has methods to book a ticket, get status (no of seats) and get fare information of train running under Indian Railways.
6. Can you change the self-parameter inside a class to something else (say “harry”). Try changing self to “slf” or “harry” and see the effects.

CHAPTER 11 - INHERITANCE & MORE ON OOPS

Inheritance is a way of creating a new class from an existing class.

Syntax:

```
class Employee: # Base class
    # Code

class Programmer(Employee): # Derived or child class
    # Code
```

We can use the method and attributes of 'Employee' in 'Programmer' object.

Also, we can overwrite or add new attributes and methods in 'Programmer' class.

TYPES OF INHERITANCE

- Single inheritance
- Multiple inheritance
- Multilevel inheritance

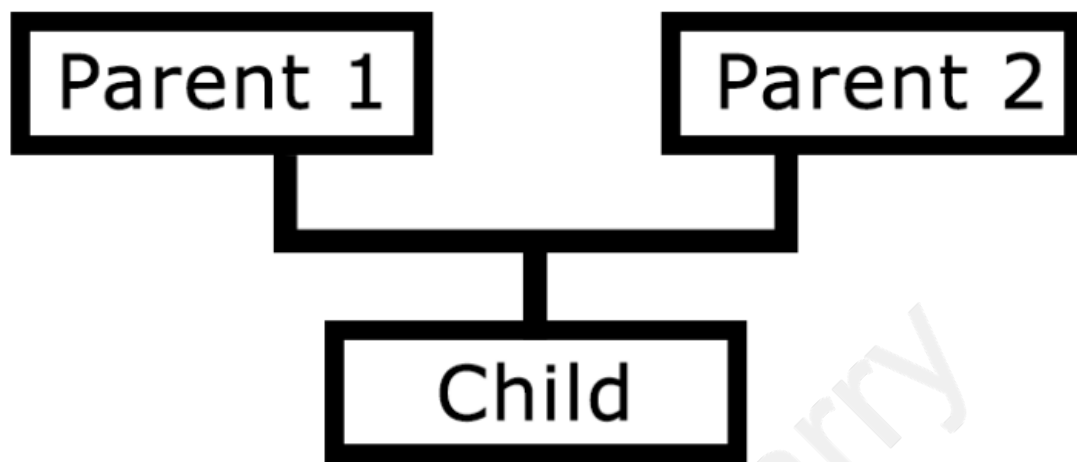
SINGLE INHERITANCE

Single inheritance occurs when child class inherits only a single parent class.



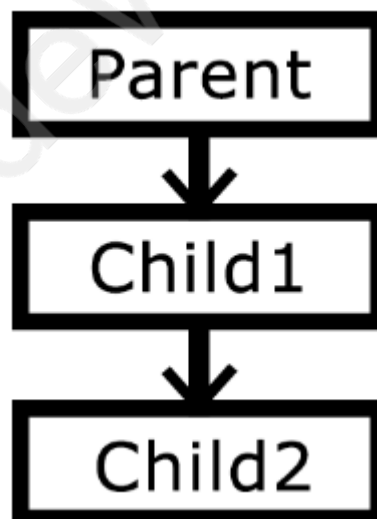
MULTIPLE INHERITANCE

Multiple Inheritance occurs when the child class inherits from more than one parent classes.



MULTILEVEL INHERITANCE

When a child class becomes a parent for another child class.



SUPER() METHOD

super() method is used to access the methods of a super class in the derived class.

```
super().__init__()
# __init__() Calls constructor of the base class
```

CLASS METHOD

A class method is a method which is bound to the class and not the object of the class.

`@classmethod` decorator is used to create a class method.

Syntax:

```
@classmethod
def(cls, p1, p2):
```

@PROPERTY DECORATORS

Consider the following class:

```
class Employee:
    @property
    def name(self):
        return self.ename
```

If `e = Employee()` is an object of class employee, we can print `(e.name)` to print the ename by internally calling `name()` function.

@.GETTERS AND @.SETTERS

The method name with '`@property`' decorator is called getter method.

We can define a function + `@ name.setter` decorator like below:

```
@name.setter
def name (self, value):
    self.ename = value
```

OPERATOR OVERLOADING IN PYTHON

Operators in Python can be overloaded using dunder methods.

These methods are called when a given operator is used on the objects.

Operators in Python can be overloaded using the following methods:

```
p1+p2 # p1.__add__(p2)
p1-p2 # p1.__sub__(p2)
p1*p2 # p1.__mul__(p2)
p1/p2 # p1.__truediv__(p2)
p1//p2 # p1.__floordiv__(p2)
```

Other dunder/magic methods in Python:

```
str__() # used to set what gets displayed upon calling str(obj)
```

```
__len__() # used to set what gets displayed upon calling.__len__() or  
len(obj)
```

CodeWithHarry

CHAPTER 11- PRACTICE SET

1. Create a class (2-D vector) and use it to create another class representing a 3-D vector.
2. Create a class 'Pets' from a class 'Animals' and further create a class 'Dog' from 'Pets'. Add a method 'bark' to class 'Dog'.
3. Create a class 'Employee' and add salary and increment properties to it.

Write a method 'salaryAfterIncrement' method with a @property decorator with a setter which changes the value of increment based on the salary.

4. Write a class 'Complex' to represent complex numbers, along with overloaded operators '+' and '*' which adds and multiplies them.
5. Write a class vector representing a vector of n dimensions. Overload the + and * operator which calculates the sum and the dot(.) product of them.
6. Write __str__() method to print the vector as follows:

```
7i + 8j +10k
```

Assume vector of dimension 3 for this problem.

7. Override the __len__() method on vector of problem 5 to display the dimension of the vector.

PROJECT 2 – THE PERFECT GUESS

We are going to write a program that generates a random number and asks the user to guess it.

If the player's guess is higher than the actual number, the program displays "Lower number please". Similarly, if the user's guess is too low, the program prints "higher number please" When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hint: Use the random module.

CodeWithHarry

CHAPTER 12 – ADVANCED PYTHON 1

NEWLY ADDED FEATURES IN PYTHON

Following are some of the newly added features in Python programming language

WALRUS OPERATOR

The walrus operator (`:=`), introduced in Python 3.8, allows you to assign values to variables as part of an expression. This operator, named for its resemblance to the eyes and tusks of a walrus, is officially called the "assignment expression."

```
# Using walrus operator
if (n := len([1, 2, 3, 4, 5])) > 3:
    print(f"List is too long ({n} elements, expected <= 3)")

# Output: List is too long (5 elements, expected <= 3)
```

In this example, `n` is assigned the value of `len([1, 2, 3, 4, 5])` and then used in the comparison within the `if` statement.

TYPES DEFINITIONS IN PYTHON

Type hints are added using the colon (`:`) syntax for variables and the `->` syntax for function return types.

```
# Variable type hint
age: int = 25

# Function type hints
def greeting(name: str) -> str:
    return f"Hello, {name}!"

# Usage
print(greeting("Alice")) # Output: Hello, Alice!
```

ADVANCED TYPE HINTS

Python's typing module provides more advanced type hints, such as `List`, `Tuple`, `Dict`, and `Union`.

You can import `List`, `Tuple` and `Dict` types from the typing module like this:

```
from typing import List, Tuple, Dict, Union
```

The syntax of types looks something like this:

```
from typing import List, Tuple, Dict, Union

# List of integers
numbers: List[int] = [1, 2, 3, 4, 5]

# Tuple of a string and an integer
person: Tuple[str, int] = ("Alice", 30)

# Dictionary with string keys and integer values
scores: Dict[str, int] = {"Alice": 90, "Bob": 85}

# Union type for variables that can hold multiple types
identifier: Union[int, str] = "ID123"
identifier = 12345 # Also valid
```

These annotations help in making the code self-documenting and allow developers to understand the data structures used at a glance.

MATCH CASE

Python 3.10 introduced the match statement, which is similar to the switch statement found in other programming languages.

The basic syntax of the match statement involves matching a variable against several cases using the case keyword.

```
def http_status(status):
    match status:
        case 200:
            return "OK"
        case 404:
            return "Not Found"
        case 500:
            return "Internal Server Error"
        case _:
            return "Unknown status"

# Usage
print(http_status(200)) # Output: OK
print(http_status(404)) # Output: Not Found
print(http_status(500)) # Output: Internal Server Error
print(http_status(403)) # Output: Unknown status
```

DICTIONARY MERGE & UPDATE OPERATORS

New operators `|` and `|=` allow for merging and updating dictionaries.

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged = dict1 | dict2
print(merged) # Output: {'a': 1, 'b': 3, 'c': 4}
```

You can now use multiple context managers in a single `with` statement more cleanly using the parenthesised context manager

```
with (
    open('file1.txt') as f1,
    open('file2.txt') as f2
):
    # Process files
```

EXCEPTION HANDLING IN PYTHON

There are many built-in exceptions which are raised in python when something goes wrong.

Exception in python can be handled using a `try` statement. The code that handles the exception is written in the `except` clause.

```
try:
    # Code which might throw exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exception to catch like below:

```
try:
    # Code
except ZeroDivisionError:
    # Code
except TypeError:
    # Code
except:
    # Code # All other exceptions are handled here.
```

RAISING EXCEPTIONS

We can raise custom exceptions using the `'raise'` keyword in python.

TRY WITH ELSE CLAUSE

Sometimes we want to run a piece of code when `try` was successful.

```
try:
    # Somecode
except:
    # Somecode
else:
    # Code          # This is executed only if the try was successful
```

TRY WITH FINALLY

Python offers a ‘finally’ clause which ensures execution of a piece of code irrespective of the exception.

```
try:
    # Some Code
except:
    # Some Code
finally:
    # Some Code          # Executed regardless of error!
```

IF __NAME__ == '__MAIN__' IN PYTHON

‘__name__’ evaluates to the name of the module in python from where the program is ran.

If the module is being run directly from the command line, the ‘__name__’ is set to string “__main__”. Thus, this behaviour is used to check whether the module is run directly or imported to another file.

THE GLOBAL KEYWORD

‘global’ keyword is used to modify the variable outside of the current scope.

ENUMERATE FUNCTION IN PYTHON

The ‘enumerate’ function adds counter to an iterable and returns it

```
for i,item in list1:
    print(i,item)  # Prints the items of list 1 with index
```

LIST COMPREHENSIONS

List Comprehension is an elegant way to create lists based on existing lists.

```
list1 = [1,7,12,11,22,]
list2 = [i for item in list 1 if item > 8]
```

CHAPTER 12 – PRACTICE SET

1. Write a program to open three files 1.txt, 2.txt and 3.txt if any these files are not present, a message without exiting the program must be printed prompting the same.
2. Write a program to print third, fifth and seventh element from a list using enumerate function.
3. Write a list comprehension to print a list which contains the multiplication table of a user entered number.
4. Write a program to display a/b where a and b are integers. If $b=0$, display infinite by handling the 'ZeroDivisionError'.
5. Store the multiplication tables generated in problem 3 in a file named Tables.txt.

CHAPTER 13 – ADVANCED PYTHON 2

VIRTUAL ENVIRONMENT

An environment which is same as the system interpreter but is isolated from the other Python environments on the system.

INSTALLATION

To use virtual environments, we write:

```
pip install virtualenv # Install the package
```

We create a new environment using:

```
virtualenv myprojectenv # Creates a new venv
```

The next step after creating the virtual environment is to activate it.

We can now use this virtual environment as a separate Python installation.

PIP FREEZE COMMAND

‘pip freeze’ returns all the package installed in a given python environment along with the versions.

```
pip freeze > requirements .txt
```

The above command creates a file named ‘requirements.txt’ in the same directory containing the output of ‘pip freeze’.

We can distribute this file to other users, and they can recreate the same environment using:

```
pip install -r requirements.txt
```

LAMBDA FUNCTIONS

Function created using an expression using ‘lambda’ keyword.

Syntax:

```
lambda arguments:expressions  
# can be used as a normal function
```

Example:

```
square = lambda x:x*x  
square(6) # returns 36  
sum = lambda a,b,c:a+b+c
```

```
sum(1,2,3) # returns 6
```

JOIN METHOD (STRINGS)

Creates a string from iterable objects.

```
l = ["apple", "mango", "banana"]  
result = ", and, ".join(l)  
print(result)
```

The above line will return “apple,and,mango,and,banana”.

FORMAT METHOD (STRINGS)

Formats the values inside the string into a desired output.

```
template.format(p1,p2...)
```

Syntax:

```
"{} is a good {}".format("harry", "boy") #1.  
("{} is a good {o}".format("harry", "boy") #2.  
  
# output for 1:  
# harry is a good boy  
  
# output for 2:  
# boy is a good harry
```

MAP, FILTER & REDUCE

Map applies a function to all the items in an input_list.

Syntax.

```
map(function, input_list)  
# the function can be lambda function
```

Filter creates a list of items for which the function returns true.

```
list(filter(function))  
# the function can be lambda function
```

Reduce applies a rolling computation to sequential pair of elements.

```
from functools import reduce  
val=reduce (function, list1)  
# the function can be lambda function
```

If the function computes sum of two numbers and the list is [1,2,3,4]



CHAPTER 13- PRACTICE SET

1. Create two virtual environments, install few packages in the first one. How do you create a similar environment in the second one?
2. Write a program to input name, marks and phone number of a student and format it using the format function like below:

“The name of the student is Harry, his marks are 72 and phone number is 99999888”
3. A list contains the multiplication table of 7. write a program to convert it to vertical string of same numbers.

7
14
.
.
.

4. Write a program to filter a list of numbers which are divisible by 5.
5. Write a program to find the maximum of the numbers in a list using the reduce function.
6. Run pip freeze for the system interpreter. Take the contents and create a similar virtualenv.
7. Explore the ‘Flask’ module and create a web server using Flask & Python.

MEGA PROJECT 1: JARVIS - VOICE-ACTIVATED VIRTUAL ASSISTANT

Jarvis is a voice-activated virtual assistant designed to perform tasks such as web browsing, playing music, fetching news, and responding to user queries using OpenAI's GPT-3.5-turbo model.

FEATURES

- Voice Recognition
- Utilizes the speech_recognition library to listen for and recognize voice commands.
- Activates upon detecting the wake word "Jarvis."
- Text-to-Speech
- Converts text to speech using pyttsx3 for local conversion.
- Uses gTTS (Google Text-to-Speech) and pygame for playback.
- Web Browsing.
- Opens websites like Google, Facebook, YouTube, and LinkedIn based on voice commands.
- Music Playback
- Interfaces with a musicLibrary module to play songs via web links.
- News Fetching
- Fetches and reads the latest news headlines using NewsAPI.
- OpenAI Integration
- Handles complex queries and generates responses using OpenAI's GPT-3.5-turbo.
- Acts as a general virtual assistant similar to Alexa or Google Assistant.
- Activates upon detecting the wake word "Jarvis."
- Text-to-Speech

WORKFLOW

1. Initialization
2. Greets the user with "Initializing Jarvis...."
3. Wake Word Detection
4. Listens for the wake word "Jarvis."
5. Acknowledges activation by saying "Ya."
6. Command Processing.
7. Processes commands to determine actions such as opening a website, playing music, fetching news, or generating a response via OpenAI.
8. Speech Output.
9. Provides responses using speak function with either pyttsx3 or gTTS.
10. Greets the user with "Initializing Jarvis...."
11. Wake Word Detection
12. Acknowledges activation by saying "Ya."

13. Processes commands to determine actions such as opening a website, playing music, fetching news, or generating a response via OpenAI.

LIBRARIES USED

- speech_recognition
- webbrowser
- pyttsx3
- musicLibrary
- requests
- openai
- gTTS
- pygame
- os

CodeWithHarry

MEGA PROJECT 2: AUTO-REPLY AI CHATBOT

DESCRIPTION

This project automates the process of interacting with a chat application, specifically designed to analyze chat history and generate humorous responses using OpenAI's GPT-3.5-turbo model. The virtual assistant, named Naruto, is a character that roasts people in a funny way, based on the chat history.

FEATURES

14. Automated Chat Interaction
15. Uses pyautogui to perform mouse and keyboard operations, interacting with the chat application without manual intervention.
16. Chat History Analysis
17. Copies chat history from the chat application and analyzes it to determine if the last message was sent by a specific user (e.g., "Rohan Das").
18. Humorous Response Generation
19. Integrates with OpenAI's GPT-3.5-turbo model to generate funny, roast-style responses based on the analyzed chat history.
20. Clipboard Operations
21. Utilizes pyperclip to copy and paste text, facilitating the retrieval and insertion of chat messages.
22. Uses pyautogui to perform mouse and keyboard operations, interacting with the chat application without manual intervention.
23. Copies chat history from the chat application and analyzes it to determine if the last message was sent by a specific user (e.g., "Rohan Das").
24. Humorous Response Generation
25. Integrates with OpenAI's GPT-3.5-turbo model to generate funny, roast-style responses based on the analyzed chat history.

WORKFLOW

- Initialization and Setup
- Click on the Chrome icon to open the chat application.
- Wait for a brief period to ensure the application is open and ready for interaction.
- Chat History Retrieval
- Periodically select and copy chat history by dragging the mouse over the chat area and using the copy shortcut.
- Retrieve the copied text from the clipboard.
- Message Analysis

- Analyze the copied chat history to check if the last message is from a specific user (e.g., "Rohan Das").
- If the last message is from the target user, send the chat history to OpenAI's GPT-3.5-turbo to generate a humorous response.
- Copy the generated response to the clipboard.
- Send Response
- Click on the chat input area and paste the generated response.
- Press 'Enter' to send the response.
- Wait for a brief period to ensure the application is open and ready for interaction.
- Chat History Retrieval
- Retrieve the copied text from the clipboard.
- Message Analysis
- Analyze the copied chat history to check if the last message is from a specific user (e.g., "Rohan Das").
- Generate Response
- Copy the generated response to the clipboard.
- Send Response

LIBRARIES USED

1. pyautogui: For automating mouse and keyboard interactions.
2. time: For adding delays between operations.
3. pyperclip: For clipboard operations.
4. openai: For interacting with OpenAI's GPT-3.5-turbo model.