Copyrighted by :

Instagram - @codes.learning

YouTube -- Tech 21

Telegram -- Codeslearningnow

Written by :

Mitasha Manoj Puri

Software Engineer

PYTHON PROGRAMMING BY CODES_LEARNING

Que: What is programming ?

→ Just like we use Hindi or English to comm-unicate 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.

Que: What is python ?

→ Python is a simple and easy to understand language which feels like reading simple english. This pseudo code nature of python makes it easy to learn and understable by begineers.

* Features of python :-
- Easy to understand - needs 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 download button, python can be installed right after you complete the setup by executing the file for your platform.

Just install like a game!

Que: Who is inventor of python ?

→ Python is a simple, easy to understand and very popular programming language.

Python is created by Guido van Rossum and released in (99).

Python is used for:

- Web Development (server side)
- Software Development
- mathematics
- System scripting.

Que: What can python do ?

→ • Python can be used on a server to create web applications.

- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production ready software development.

Que: Why python ?

→ • Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc)

- python has a simple syntax similar to english language.
- Python has syntax that allows developers to write programs with fewer lines.

- python runs on an interpreter systems, meaning that code can be executed as soon as it is written. This means that protyping can be very quick.
- Python can be created 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 identation, using whitespaces, to define scope; such that scope of the loops, function and classes. other programming languages often use curly-brackets for this purpose.

check if you have installed python on your pc's ?

→ 1). To check if you have python installed on a windows pc, search in the start bar for python or run the following on command line: (cmd.exe):

```
c:\users\Your
Name> python --version.
```

2). To check if you have python installed on a linux or mac, then on linux open command line or on mac open the Terminal and type:

```
python --version
```

3). If you find that you do not have python installed on your computer, then you can also download it for free from below website:

https://www.python.org/

Let's move to our very first chapter of python series !!

→ → →

Chapter 1: Modules, Comments and pip.

let's write our first python program:
i.e. Hello World program.

code

create file with name - hello.py

↓

print ("Hello World!")

↓

Execute the file (.py file)

How code runs?

let's see what's going to execute this code at back. firstly we will create a file named as "Hello.py" (you can type any name as per your choice but .py is mandatory) By executing this file, you will see ⟶ Hello World! printed on screen.

I). * Modules :-

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

modules provide an easy way to organize

components into a system by serving as self-contain packages of variables known as namespaces.

* Types of modules :-

Types of modules

1). Built in modules → pre-installed in python → Example – os, abc etc

2). External modules → Need to install using (pip) → example – tensorflow, flask. etc.

* Using python as a calculator :-

yes, you heared it right, we can use python as a calculator by typing

"python" + ↵ on the terminal → This opens REPL.

What is REPL ?

REPL stands for Read Evaluate Print Loop.

II). * Comments :-

comments are used by programmer and it write used something which the programmer does not want to execute.

''' ''' , " # " are used to denote comments.
This can be used to mark author name, date etc.

Example :-

```
#author name = codes.learning
print ("Best channel on youtube")
```

→ Best channel on youtube.

Why ??

Have you seen the line after " # " is not printed on screen.

Types of comments

1) single line → written using " # "

2) multi line → written using '''comment'''

III). * Pip :-

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

Example :- pip install flask

→ It installs flasks module.

Let's practice with some interesting questions

Code-set-01

Q.1. Write a program to print Johny Johny yes papa poem in python.

Q.2. Use REPL and print the table of 2 using it.

Q.3. Install an external module and use it to perform an operation of your interest.

Q.4. Write a python program to print the contents of a directory using os module. Search online for the function which does that.

Q.5. Label the program written in problem 4 with comments.

Yes you can solve this ☺

#1st code:

```
'''
This problem is a solution of
Problem-1 of codes-learning code-set-01!
'''

#This is also a comment just like above line

print('''Johny, Johny,
        Yes papa?
        Eating sugar?
        No papa.
        Telling lies?
        No Papa.
        Open your mouth
        Ha ha ha!
    ''')
```

#2nd code :-

#go to terminal and type python
#REPL get launched and so on stanle continues..

```
2*1
>>2
2*2
>>4
2*3
>>6
```

3rd code :

```
from playsound import playsound
playsound ('D:\\mydata \\Business \\code playground
        \\ Python course with notes\\play.mp3')
```

in brackets you can choose any of your favourite
song from your pc and listout directories of it.

4th code :

```
# Author : codes-learning
# location : mars.
# Date : 23/09/2022

import os
print(os.listdir())
```

5th code :

```
# you can add more comments in above code!
```

***

Chapter 2: Variables and Datatypes.

I). * Variables :-

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

Example :-

```
a = 20
b = "Codes-learning"
c = 71.22
```

What is variable ??

variables are container to store a value

* keywords :-

keywords are the reserved words in python which you cannot use as a variables.

* Identifiers :-

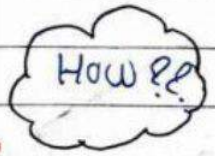
class / function / variable name.

II). * Data types :-

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.



let's see How?

a = 71

write : type (a)
→ int

This is how the data type of a variable is recognize through a word Type.

This identifies a as class <int>

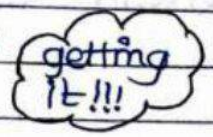
Take 2nd example : b = 88.44 type(b)
ans will be → Identifies b as class < float >

How to define variable name ??
let's see some rules to define variable name :

- A variable name can contain Alphabets, digits & underscore
- A variable name can only start with Alphabets and underscore.
- A variable name can not start with a digit
- No white space is allowed to be used in variable.

let's see which variable name is right?

13codes	☒	Codes_learning	☑
13 – codes	☒	24x7codes	☒
codes_13	☑		



* Operators in python :-

operators in python

1). Arithmatic operators	2). Assignment Operators	3). Comparison operator	4). logical operator
some basic arithmatic operators	operators which assign some value	which compares betn two values.	basic logical operators
+, -, *, /	=, +=, -=	==, >, >=, < ! = etc	and, or, not etc.

* type function and typecasting :-

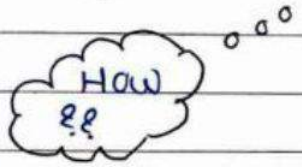
type () function is used to find the data type of a given variable in python.

a = 31
type (a) ⟹ class <int>
which means the data type of a is integer (int)

b = "codes-learning"
type (b) ⇒ class <str>
which means the data type of b is string (str)

Note : string is always in " " double quotation.

Do you want to convert integer variable to a string ?



let's see how to convert a integer into a string and vice versa.

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

str (10) ⇒ " 10 " ⇒ Integer to string conversion
int ("35") ⇒ 35 ⇒ string to integer conversion
float (32) ⇒ 32.0 ⇒ Integer to float conversion

... and so on

codes learning

Now try this one,

str (25) ⇒ .....
try to convert integer into a string.

yes, you get it right answer will be,

str (25) ⇒ "25" (integer shown in " ").

Here note this one, that "25" is a string literal and (without quotation " ") 25 is an integer literal.

* Input function :-

The input () function allows the users to take input from the keyboard as a string.

for example,

a = input ("Enter name")

Here we can write our name :

the screen will look like : (if we write codes learning as our input)

→ a = "Codes learning"

let's take another example,

b = Input ("Enter name")

Here we write name of person : Rahul,

→ b = "Rahul" ← This will be your o/p on screen.

Note :- It is important to note this : the output of the input is always a string (even if an integer is passed).

let's make it clear :

a = (input ("Enter a number")

→ a = "10" → if we put a number as 10 as an input

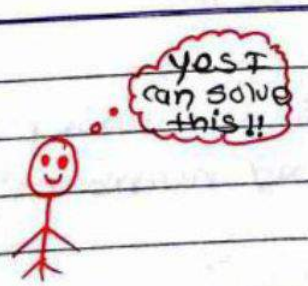
It's time to code !!

Code - set - 02 :

Q.1. Write a python program to add two numbers.

Q.2. Write a python program to find remainder when a number is divided by 2.

Q.3. Write a python program to find average of two numbers entered by the user.



Code - set - 02 :

1 st code :-

```
a = 30
b = 32
print ("The sum of a and b is", a+b)
```

2nd code :-

```
a = 458
z = 15
print ("the remainder when a is divided by z is",
        a%b)
```

3rd code :-

```
a = input ("Enter first number:")
b = input ("Enter second number:")
```

```
a = int (a)
b = int (b)
avg = (a+b) / 2.
print (" The average of a and b is : ", avg)
```

***

Chapter 3 : Strings.

* Strings :-

String is a data type in python.
String is a sequence of characters unclosed in quotes".

We primarily, write a string in three ways :

1). single quoted strings :-

a = 'codes learning'

2). Double quoted strings :-

b = "codes learning"

3). triple quoted strings :-

c = """codes learning"""

* String slicing :-

A string in python can be sliced for getting a part of the string.
consider the following string :

name = "c o d e s" ⇒ length = 5

0	1	2	3	4
(-5)	(-4)	(-3)	(-2)	(-1)

The index in the string starts from 0 to (length -1) in python.

In order to slice a string, we use the following syntax :

st = name [ind-start : ind-end]

↓ first index included ↓ last index is included.

st [0:3] returns " cod " ie. characters from 0 to 3.
st [1:3] returns " od " ie 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 corresponds to the length (-2) index.

Note :-

one important point to remember that the last index of a string always (-1).

i.e. in our above code element s has negative index as -1, so the last character of the string should always have negative index as (-1).

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 mostly used functions to perform operations on or manipulate strings are :

1). len() function :-
This function returns the length of the string.

len ("codes") ⟶ returns 5.

2). string.endswith ("ry") :-
This function tells whether the variable string ends with the string "ry" or not. If string is "codes" it returns true for "ry" since codes ends with ry.

3). string.count ("c") :-
counts the total number of occurance of any character.

4). string.capitalize () :-
This function capitalizes the first character of a given string.

5). string.find () :-
This function finds a word and returns the index of first occurance of that word in the string.

6). string.replace (oldword, newword) :-
This function replaces the oldword with the newword in the entire string.

Escape sequence characters :
sequence of characters after backslash '\'
i.e. \ escape sequence character.

Escape sequence character comprises of more than one characters but represents one characters when used within the strings.

Example :-

\n → newline
\t → Tab
\' → single quote
\\ → backslash etc.

Let's sum up the chapter of strings by some practical examples :-

1). finding type of a variable by type () function

```
#b = "codes 's"  #--> use this if you have
                      single quotes in string.
#b = 'codes"s'

b = '''code"s and code's'''

print (b)

# print (type(b))
```

2). slicing of string :-

```
#greeting = "Good Evening,"
#Name = "codes-learning"
# print (type (name))

#concatenating two strings
#c = greeting + name
# print (c)
name = "codes-learning"
#print (name[4])
#name [3] = "k" --> Does not words

#print (name [1:4]).
#print (name [:4]) # is same as name[0:4]
# print (name [1:]) #is same as name [1:5].
#c = name [-4:-1] #is same as name [1:4].
# print (c)

name = "codes-learning is best channel"
#d = names [0::3]
d = name [:0:-1].
print (d)
```

3). string functions :-

```
story = "once upon a time there was a good boy"

# string functions
# print (len (story))
# print (story.endswith("boy"))
# print (story.count("c")
# print (story.capatitalize())
# print (story.find("upon")
print (story.replace("boy", "girl"))
```

4). Escape sequence of string :-

```
story = "Ram is a good.\n He\t is vellry good"
print(story)
```

It's time for some interesting questions!

Code-set-03

Q.1. Write a python program to display a user entered name followed by Good evening using input() function.

Q.2. Write a python program to fill in a letter template given, below with name and date

```
letter = """ Dear <|NAME|>,
                You are selected!
                <|DATE|> """
```

Q.3. Write a program to detect spaces in a string

Q.4. Write a python python program to format the following letter using escape sequence characters:

```
letter = " Dear, codes-learning, This python
           course is nice.. Thanks!!!"
```

Q.5. Replace the double spaces from problem 3 with single spaces.

codes for practice :- code-set-03

```
# 1st code :-
name = input("Enter your name \n")
print("Good evening, " +name)


# 2nd code :-
letter = '''Dear <|NAME|>,
Greetings from ABC coding house, I am happy
to tell you about your selection
You are selected!
Have a great day Ahead!
Thanks and regards,
Bill
Date : <|DATE|>
'''
name =input("Enter your Name \n")
date = input("Enter date \n")
letter = letter.replace("<|NAME|>",name)
letter = letter.replace("<|DATE|>", date)
print(letter)


# 3rd code :
st = "This is a string with double spaces"
doubleSpaces = st.find("  ")
print (doubleSpaces)
```

#4th code :

```
letter = "Dear codes_learning, This python
          course is nice! Thanks !!!"
print( letter)

formated_letter = "Dear codes_learning, \n It
                   This python course is nice! \n Thanks!!!"
print (formatted_letter)
```

#5th code :

```
st = "This is a string with double spaces ok"

st = st.replace ("  ", "   ")
print (st)
```

***

Chapter 4 : Lists and Tuples.

Lists :-

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

fruits = ["Apple", "Akash", "Rohan", 7, false]

list Indexing :-

A list can be indexed just like a string.

li = [7, 9, "codes-learning"]

li[0] = 7
li[1] = 9
li[70] = Error
li[0:2] = [7, 9] ⇒ list slicing

lists are created using square brackets:

create a list :

```
this_list = ["apple", "banana", "cherry"]
print (this_list)
```

output :

apple, banana, cherry.

List items :-

List items are ordered, changeable, and allow duplicate values.
List index are indexed, the first item has index [0] and second item has [1] etc.

Ordered :-

When we say that lists are ordered, it means that items have a defined order, and that order will not change.
If you add new items to a list, the new items will be placed at the end of the list.

changeable :-

lists are changeable, meaning that we can change, add and remove items in a list after it has been created.

Allows Duplicates :-

since lists are indexed, lists can have items with the same value:

```
this_list = ["apple", "banana", "cherry", "banana"]
print (this_list)
```

output :
['apple', 'banana', 'cherry', 'banana']

list length :-

To determine how many items a list has, use len () function.

② Example :-

```
this_list = ["apple", "banana", "cherry"]
print (len (this_list))
```

output :-

len () function counts the no. of items.

→ 3

③. Lists items - Data Types :-

lists items can be of any data type.

```
list_1 = ["apple", "banana", "cherry"]
list_2 = [1,5,7,9,3]
list_3 = [True, false, false]
```

output:

```
['apple', 'banana', 'cherry']
[1,5,7,9,3]
[True, false, false]
```

A list can contain different data types:

A list with strings, integers and boolean values:

list1 = ["abc", 34, True, 40, "male"]

output:
~~list1~~ ['abc', 34, True, 40, 'male']

type() :-
from python's perspective, lists are defined as objects with the data type 'list':

<class 'list'>

Example:

4. mylist = ["apple", "banana", "cherry"]
print (type (mylist))

output:
<class 'list'>

How to make a list with list() constructor :-

5. thislist = list (("apple", "banana", "cherry"))
note the double round-brackets).
print (thislist)

output:
['apple', 'banana', 'cherry'].

How to access lists items :-

lists items are indexed and you can access them by referring to the index number.

6. thislist = ["apple", "banana", "cherry"]
print (thislist [1])

output:
banana.

Note :-
- The first item has index 0.
- The last item has index -1 and -2 refers to second last item (in negative indexing)

Range of indexes :-

range is nothing but indexes by specifying where to start and where to end range.

7. thislist = ["apple", "banana", "cherry", "orange" "kiwi", "melon", "mango"]
print (thislist [2 : 5])

output:
['cherry', 'orange', 'kiwi']

Note :
- The search will start at index 2 (included) and end at index 5 (not included)

8. this list = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print (this list [:4])

output:
['apple', 'banana', 'cherry', 'orange']

note :
- Remember that index 0 is first item and index 4 is fifth item.
- the item in the index 4 is not included.

9. print (this list [2 :])

output:
['cherry', 'orange', 'kiwi', 'melon', 'mango']

note :
- This will return items from index 2 to the end.

10. print(this list [-4 : -1])

output:
['orange', 'kiwi', 'melon']

Note :
- Negative indexing means starting from end of list.
- above example returns items from index -4 to -1.

How to check if item is present in list or not?

To determine if a specified item is present in a list use the in keyword:

(11).
```
#check if "apple" is present in list:
thislist = ["apple", "banana", "cherry"].
if "apple" in thislist:
    print("yes, 'apple' is in fruits list")
```

output:

Yes, 'apple' is in the fruit lists.

Change item value:

(12).
```
#to change value of first item.
thislist = ["apple", "banana", "cherry"].
thislist[0] = "mango".
print(thislist)
```

output:

['mango', 'banana', 'cherry'].

Insert items in a list :-

we use insert() method, to insert a new list item, without replacing any of existing values.

(13). thislist = ["apple", "banana", "cherry"].
thislist.insert (2, "watermelon")
print (thislist)

output:
['apple', 'banana', 'watermelon'].

Add items in the list :-
To add an item at the end of the list, we use append () method.

(14). thislist = ["apple", "banana", "cherry"]
thislist. append ("orange").
print (thislist)

output:
['apple', 'banana', 'cherry', 'orange']

Remove items from the lists :-
remove () method removes specified item.

(15). #to remove banana.
thislist = ["apple", "banana", "cherry"].
thislist. remove ("banana").
print (thislist).

output:
['apple', 'cherry'].

Tuple :-

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built in data types in python used to store collections of data, other 3 list, set and dictionary all with different qualities.

A tuple is a collection which is ordered and unchangeable.

Tuples are written in round brackets.

① #to create a tuple :

```
thistuple = ("apple", "mango", "banana")
print (thistuple)
```

output :-

```
('apple', 'mango', 'banana')
```

Tuple items :-

Tuple items are ordered, unchangeable and allow duplicate values.

Create a tuple with one item :-

To create a tuple with only one item, you have to add a comma after item.

② #one item in tuple :-

```
thistuple = ("apple",)  #comma (,) is mandatory
print (type (thistuple))
```

output :-

```
<class 'tuple'>
```

note : If we don't use (,) here so type of tuple is "str".

③ # to make a tuple with tuple() constructor :-

```
thistuple = tuple (("apple", "banana", cherry")).
# note double round brackets.
print (thistuple)
```

output :-

('apple', 'banana', 'cherry').

TO Access Tuple items :-

You can access tuple items by referring to index number, inside square brackets.

④ # print second item in the tuple.

```
thistuple = ("apple","banana","cherry").
print( thistuple [1])
```

output :

banana. #first item has index 0.

⑤ negative indexing :-

Negative indexing means start from the last element (item)

to print last item of the tuple :-

```
thistuple = ("apple", "banana", "cherry").
print ( thistuple [-2])
```

output :-

banana #-1 refers to last item.
#-2 refers to second last item.

Range of indexes :-

range means to specifying where to start and where to end.

When specifying a range, the return value will be a new tuple with specified items.

(6) # To Return third, fourth and fifth item :-

```
thistuple = ("apple", "banana", "cherry", "orange"
             "kiwi", "melon", "mango")
print (thistuple [2:5])
```

output :-

```
('cherry', 'orange', 'kiwi')
```

Note :-

- The search will start at index 2 (included) and end at index 5 (not included).

(7) # negative index :

```
thistuple = ("apple", "banana", "cherry", "orange",
             "kiwi", "melon", "mango")
print (thistuple [-4:-1])
```

output :-

```
('orange', 'kiwi', 'melon')
```

Tuple methods

- count()
  - Returns the number of times specified value occurs in a tuple.
- index()
  - Searches the tuple for a specified value and returns position of where it was found.

It's the time with some interesting questions.

Code-set-04.

Q.1. Write a program to store seven fruits in a list entered by the user.

Q.2. Write a program to accept marks of 6 students and display them in a sorted manner.

Q.3. check that a tuple cannot be changed in python.

Q.4. Write a program to sum a list with 4 numbers.

Q.5. Write a program to count the number of zeros in following tuple:

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

TRY TO SOLVE !!!

#1st code :-

```
f1 = input("Enter fruit number 1 : ")
f2 = input("Enter fruit number 2 : ")
f3 = input("Enter fruit number 3 : ")
f4 = input("Enter fruit number 4 : ")
f5 = input("Enter fruit number 5 : ")
f6 = input("Enter fruit number 6 : ")
f7 = input("Enter fruit number 7 : ")

myfruitList = [f1, f2, f3, f4, f5, f6, f7]
print (myfruitList)
```

#2nd code :

```
m1 = int(input("Enter marks of student number 1: "))
m2 = int(input("Enter marks of student number 2: "))
m3 = int(input("Enter marks of student number 3: "))
m4 = int(input("Enter marks of student number 4: "))
m5 = int(input("Enter marks of student number 5: "))
m6 = int(input("Enter marks of student number 6: "))
mylist = [m1, m2, m3, m4, m5, m6]
mylist.sort()
print(mylist)
```

```
#3rd code:
a = (2, 4, 5, 3, 2)
a[0] = 45.


# 4th code:

a = [2, 4, 56, 7].

print (a[0] + a[1] + a[2] + a[3])
print (sum (a)).


#5th code:

#creating a tuple using ()

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

print (a.count (0)).
```

***

Chapter 5 : Dictionary & Sets.

Dictionary :-

Dictonaries are used to store data values in key : value pairs.

A dictionary is a collection which is ordered, changeable and does not allow duplicates *.

dictionary is written with curly brackets, and have keys and values.

① #create and print a dictionary:-

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

output :-

{'brand': 'ford', 'model' : 'Mustang', 'year': 1964}

Dictionary length :-

To determine how many items a dictionary uses, use the len () function.

② # to print number of items in the dictionary:-

```
print(len(thisdict))
thisdict = {"brand": "ford", "model" : "Mustang",
            "year": 1964 , "year" : 2020 }
print (len(thisdict))
```

output :-

3

Data types :-

The values in dictionary items can be of any data type :

(3). thisdict = { "brand" : "ford", "electric" : false, "year : 1964 , "colours" : ["red," white", "blue"]. }

print (thisdict)

here a dictionary contains a list data type along with string, int and boolean.

output :-

{ 'brand' : 'ford', 'electric' : false, 'year' : 1964, 'colours' : ['red', 'white', 'blue'] }.

type () :-

dictionaries are defined as objects with data type 'dict':

<class 'dict'>

(4). # print the data type of a dictionary :-

thisdict = { "brand" : "ford", "model" : "Mustang", "year" : 1964 }.

print (type (thisdict))

output :

<class 'dict'>

Accessing items :-

You can access items of a dictionary by referring to its key name, inside square brackets :

(5) # get the value of "model" key :

thisdict = { "brand": "ford", "model" : Mustang" }.

x = thisdict["model"]

output :-

Mustang.

Dictionary methods.

1. clear ()
Removes all the elements from dictionary.

2. copy ()
Returns a copy of the dictionary

3. get()
Returns value of specified key.

4. items ()
Returns a list containing tuple for each key value pair

5. keys ()
Returns a list containing dictionary's keys.

6. pop()
Removes element with specified key.

7. setdefault()
Returns value of specified key, If key does not exist: insert key, with specified value.

8. update ()
updates dictionary with specified key value pairs.

9. values()
Returns a list of all values in the dictionary

Python sets :-

A set is a collection which is both unordered and unindexed.

sets are written with curly brackets. this are used to store multiple items in a single variable.

① #create a set :
thisset = { "apple ", "banana", "cherry "}.
print (thisset).
note : the set list is unordered, the items will appear in a random order.
output :-
{ 'cherry ', 'apple ', 'banana '}.

Note :-
- sets are unordered, so you cannot be sure in which order the items will appear.
- sets items are unordered, unchangeable and do not allow duplicate values.
- once a set is created, you cannot change its items, but you can add new items.

② # Duplicate values will be ignored :-
thisset = { "apple ", "banana ", "cherry ", "apple "}.
print (thisset)
output :-
{ 'banana ', 'cherry ', 'apple '}

Set methods

1. add()
Adds an element to the set

2. clear()
Removes all elements from set.

3. copy()
Returns a copy of the set.

4. difference()
Returns a set containing difference between two/more sets.

5. discard()
Remove the specified item.

6. Intersection_update()
Returns items in this set that are not present in other, specified sets.

7. intersection()
Returns a set that is intersection of two sets.

8. pop()
Removes an element from the set.

9. remove()
Removes specified element.

10. union()
Return a set containing union of set.

11. update()
update set with union of set & others.

11. issubset()
Returns whether another set contains this set/not

12. issuperset()
Returns whether this set contains another set/not.

Operations on sets :-

consider following set :

$S = \{1, 8, 2, 3\}$.

1). Len(S) : Returns 4, as length of set is 4

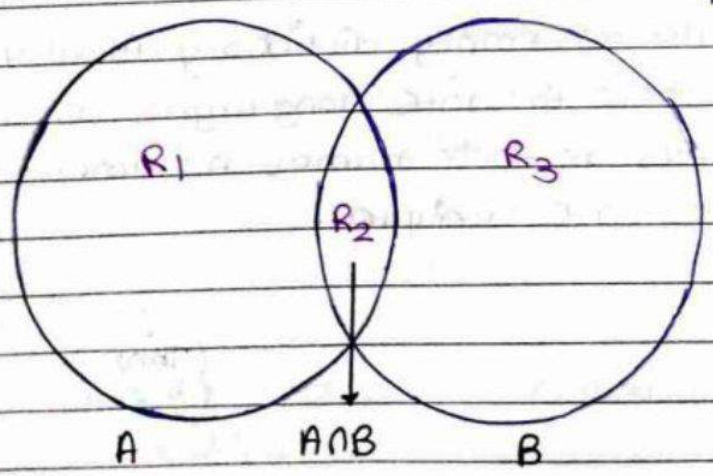
2). s.remove (8) : Update set s and removes 8 from set S.

3). s. pop() : Removes an arbitrary element from set and returns the element removed.

4). s.clear() : Empties the set s.

5). s.union ({8, 11}) : Returns a new set with all items from both sets. $\Rightarrow \{1, 8, 2, 3, 11\}$

6). s.intersection ({8, 11}) : Returns a set which contains only items in both sets. $\Rightarrow \{8\}$



$R_2 \Leftrightarrow A \cap B$

$R_1 + R_2 + R_3 \Leftrightarrow A \cup B$

$R_1 + R_3 \Leftrightarrow A \triangle B$

$R_1 \Rightarrow A - B$

$R_3 \Rightarrow B - A$.

It's the time to practice with some interesting questions !!

Code-set-05

Q.1. Write a program to create a dictionary of hindi words values as their english translation. provide user with an option to look it up!

Q.2. Write a program to input eight numbers from the user and display all unique numbers (once)

Q.3. can we have a set with 18 (int) and "18" (str) as a values in it ?

Q.4. What will be length of following set s ?

```
s = set()
s.add(20)
s.add(20.0)
s.add("20")
```

⇒ length of s after these operation

Q.5. create an empty dictionary. Allow 4 friends to enter their favourite languages as values and use keys as their names. Assume that the names are unique.

Think & solve

Code-set-05.

#1st code :-

```
myDict = {"Pankha" : "fan", "Dabba" : "Box",
          "Vastu" : "Item"}
print ("options are", myDict.keys ())
a = input ("Enter the Hindi Word \n")

#print ("The meaning of your word is :", myDict [a])

#Below line will not throw an error if key is not
 present in the dictionary.
print ("The meaning of your word is :", myDict.get(a))
```

2nd code :-

```
num1 = int(input ("Enter number 1\n"))
num2 = int(input ("Enter number 2 \n"))
num3 = int (input ("Enter number 3 \n"))
num4 = int (input (" Enter number 4\n"))
num5 = int(input("Enter number 5\n"))
num6 = int (input("Enter number 6\n"))
num7 = int (input ("Enter number 7 \n"))
num8 = int (input("Enter number 8\n"))

s = {num1, num2, num3, num4, num5, num6, num7,
     num8}.
print (s)
```

3rd code :-

```
s = { 18, "18", 18.1 }.
print (s).
```

4th code :-

```
s = { 20, 20.0, "20" }
print (s)
print (len (s))
```

5th code :-

```
favLang = { }
a = input ("Enter your favourite language Mitasha \n")
b = input ("Enter your favourite language Manoj \n")
c = input ("Enter your favourite language Mamta \n")
d = input ("Enter your favourite language sushmita \n")

favLang ['Mitasha'] = a
favLang ['Manoj'] = b
favLang ['Mamta'] = c
favLang ['sushmita'] = d

print (favLang).
```

Chapter 6 : Conditional Expression

Conditional Expressions :-

python has conditional expressions (sometimes called a "ternary operator") you can write operation like if statement in one line with conditional expressions.

x if condition else y.

The condition is evaluated first. If condition is True, x is evaluated and value is returned, and if condition is False, y is evaluated and value is returned.

① #check given number is even or not.

```
a = 1
result = 'even'
if a % 2 == 0
   else 'odd'
print (result).
```

output:

odd.

logical conditions and if else statements :-

- Equals : a == b
- Not equals : a != b
- Less than : a < b
- less than or equal to : a <= b
- Greater than : a > b
- Greater than or equal to : a >= b

If statement is written by using if keyword.

(2). # If statement :-

```
a = 33
b = 200
if b > a :
    print ("b is greater than a")
```

output :-

b is greater than a

Indentation :-

python relies on indentation (white space at beginning of a line) to define scope in the code.
* other programming languages often use curly brackets for this purpose. *

(3) # without indentation (if statement) will raise an error.

```
a = 33
b = 200
if b > a :
print ("b is greater than a")
```

Here no white space at begining of line i.e. identation so, this will raise an error.

(4). # Elif statement :-

"Elif keyword in python way of saying "if previous conditions were not true, then try to this condition".

```
a = 33
b = 33
if b > a :
    print ("b is greater than a")
elif a == b :
    print ("a and b are equal")
```

identation → print ("b is greater than a")

identation → print ("a and b are equal")

output :-

a and b are equal.

⑤ # Else statement :-

else keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a :
    print ("b is greater than a")
elif a == b :
    print ("a and b are equal")
else :
    print ("a is greater than b")
```

output :-

a is greater than b.

⑥ # And keyword :-

And keyword is a logical operator, and it is used to combine conditional statements.

Test if a greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a :
    print("Both conditions are True").
```

Output :-

Both conditions are True.

(7). # OR keyword :-

or keyword is a logical operator, and is used to combine conditional statements.

Test if a is greater than b OR if a is greater than c:

```
a = 200
b = 33
c = 500
if a > b or a > c :
    print("At least one of the conditions is True")
```

Output :-

At least one of the conditions is True.

(8). # Nested if :-

if statements inside if statements this is called nested if statements.

```
x = 41
if x > 10 :
    print ("Above ten,")
    if x > 20 :
     print ("and also above 20!")
    else :
      print ("but not above 20")
```

output :-

Above ten,
and also above 20!

(g) # pass statement :-

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200
if b > a :
    pass
```

having an empty if statement like this, would raise an error without pass statement.

It's time to practice !!

Code-set-06 :

Q.1. Write a program to find greatest of four numbers entered by the user.

Q.2. Write a program to find out whether a student is pass or fail, if it requires total 40 % and at least 33% in each subject to pass. Assume 3 subjects and take marks an input from the user.

Q.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 this spams.

Q.4. Write a program which find out whether a given name is present in a list or not.

Q.5. Write a program to calculate grade of a student from this his marks from the following scheme :

90 - 100 → EX
80 - 90 → A
70 - 80 → B
60 - 70 → C
50 - 60 → D
<50 → F

TRY TO SOLVE BY YOURSELF

Code - set - 06 :

```
#1 st code :-
num1 = int(input("Enter number1 : "))
num2 = int(input("Enter number 2 : "))
num3 = int(input("Enter number 3 : "))
num4 = int(input("Enter number 4: "))

if (num1 > num4) :
    f1 = num1
else :
    f1 = num4.

if (num2 > num3) :
    f2 = num2
else :
    f2 = num3

if (f1>f2) :
   print( str(f1) + " is greatest")
else :
   print (str (f2)+" is greatest ")


#2nd code :
sub1 = int(input("Enter first subject marks \n"))
sub2 = int(input("Enter second subject marks \n"))
```

```
sub3 = int(input("Enter third subject marks \n"))

if (sub1 < 33 or sub2 < 33 or sub3 < 33):
    print("You are fail because you have less than
          33 % in one of the subjects")
elif (sub1 + sub2 + sub3)/3 < 40 :
    print ("You are fail due to total percentage
          less than 40")
else :
    print ("Congratulations! You passed the exam")


# 3rd code :-

text = input ("Enter the text \n")

if ("make a lot of money" in text):
    spam = True
elif ("buy now" in text):
    spam = True.
elif ("click this" in text):
    spam = True :
elif ("subscribe this" in text):
    spam = True
else :
    spam = false.

if (spam):
    print ("this text is spam")
```

```
else :
    print ("This text is not spam").


# 4th code :
names = ["mitasha", "khushi", "mamta", "shivani", "manoj"]
name = input ("Enter the name to check In")

if name in names :
    print ("Your name is present in the list")
else :
    print ("Your name is not present in list")


# 5th code :-

marks = int (input("Enter Your marks In"))
if marks >= 90 :
    grade = "Ex"
elif marks >= 80 :
    grade = "A"
elif marks >= 70 :
    grade = "B"
elif marks >= 60 :
    grade = "C"
elif marks >= 50 :
    grade = "D"
else :
    grade = "F"
print ("Your grade is " + grade)
```

***

Chapter 7 : Functions & Recursions.

* Functions :-

A function is a block of code when only runs when it is called.

You can pass data, known as parameters into a function.

A function can return data as a result.

① #creating a function :

In python a function is defined using def keyword :

```
def my-function () :
    print ("Hello from a function")
```

② # calling a function :-

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

```
def my-function ():
    print ("Hello from a function")
my-function ()
```

output :

Hello from a function.

* Arguments :-

information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parantheses. You can add as many arguments, as you want, just seprate them with a comma.

(3) # Example on Arguments :

```
def my-function (fname):
    print (fname + "Refsnes")

my-function ("Email")
my-function ("Tobias")
my-function ("linus")
```

output :-

```
Email Refsnes
Tobias Refnes
Linus Refsnes.
```

Note :-

- Arguments are often shortened to args in python documentations.

Parameters and arguments can be used for same thing?

Answer : yes, the terms parameter and argument can be used for same thing i.e. information that are passed into a function.

From a function's perspective :

A parameter is a variable listed inside the paranthe-ses in the function defination.

An argument is the value that is sent to the function when it is called.

④. # Number of arguments :-

By default, a function must be called with correct number of arguments.

for example - If your function expects 2 arguments, you have to call function with 2 arguments, not more not less.

```
def my-function (fname, lname):
    print (fname + " " + lname)
my-function ("Emil", "Refsnes")
```

output :-

Emil Refsnes.

⑤. # Arbitrary Arguments, *args :-

If you don't have any idea how many arguments that will be passed into your, function, add a * before parameter name in the function definition.

```
def my-function (*kids) :
    print (" The youngest child is " + kids [2])
my-function ("Emil", "Tobias", "Linus")
```

output :-

The youngest child is Linus.

6. # Keyword arguments :-

You can also send arguments with key = value syntax.

This way the order of arguments does not matter.

```
def my-function (child3, child2, child1 ):
    print("The youngest child is " + child3).

my-function (child1 = " Emil ", child 2 = "Tobias",
            child 3 = "linus").
```

output :-

The youngest child is linus.

Note :-

The phrase keyword Arguments are often shortened to kwargs in Python documentations.

Types of functions in python.

1). Built-in functions

Build in functions are already present in python.

2). User defined functions

User defined functions means defined by the user

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

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

Recursion :-

Recursion means a defined function can call itself. Recursion is a common mathematical and programming concept.

It is directly used as a mathematical formula as a function for 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
        return 1        call function any further
    else :
        return n * factorial (n-1) → function calling
                                     itself.
```

How this works ?

Answer: factorial (3) → function called.

↓

3 x factorial (2)

3 x [2 x factorial (1)]

3 x 2 x 1 → function returned.

The developer should be very careful with recursion as it can be quite easy to slip into a writing function which never terminates or one that uses excess amounts of memory or processor power.

(7) # Recursion Example :

```
def tri-recursion(k) :
    if (k>0) :
        result = k + tri-recursion (k-1)
        print (result)
    else :
        result = 0
    return result
print ("\n\n Recursion Example Results ")
tri-recursion (6)
```

Output :-

```
Recursion Example Results
1
3
6
10
15
21
```

It's time to practice with some interesting questions !!

Code – set – 07

Q.1. Write a program using function to find greatest of these three numbers.

Q.2. Write a python program using function to convert celsius to fahrenheit.

Q.3. How do you prevent a python print () function to print a new line at the end.

Q.4. Write a Recursive function to calculate the sum of first n natural numbers.

Q.5. Write a python function to print reset first n lines of following pattern :

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

→ for n = 3

Q.6. Write a python program function to remove a given word from a list and strip it at the same time.

I can solve this

code-set-07

```
#1st code :-
def maximum (num1,num2, num3):
    if (num1 > num2):
        if (num1 > num3):
            return num1
        else :
            return num3
    else :
        if (num2 > num3):
            return num2
        else :
            return num3

m = maximum (13,55,2)
print ("The value of the maximum is " + str(m))


# 2nd code :-

def farh (cel):
    return (cel * (9/5)) +32

c = 0
f = farh(c)
print ("farhenheit temperature is "+ str(f))
```

3rd code :

```
print ("Hello ", end = "    ")
print ("my", end = "    ")
print ("Name ", end = "    ")
print (" is ", end = "    ")
print ("mitasha", end = "    ")
```

4th code :-

```
# n! = (n-1)! * n
# sum (n) = sum (n-1) + n.
```

5th code :-

```
n = 3
for i in range (n):
    print ("*" * (n-i))
#this will print * n-i times.
```

6th code :-

```
def remove-and-split (string, word) :
    newstr = string. replace (word, " ")
    return newstr. strip ()
this = "    Rahul is a good boy    "
n = remove-and-split (this, "Rahul ")
print (n)
# print (this)
# print ( this. strip ())
```

***

Chapter 8 : Loops in python

Python loops :-

Python has two primitive loop commands:

- While loop.
- for loop.

The while loop :-

With the while loop we can execute a set of statements as long as a condition is true.

① #Example

```
i = 1
while i < 6 :
    print (i)
    i += 1
```

output :-

1
2
3
4
5

Note :-

remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which is we set to 1.

- The break statement :-

With the break statement we can stop the loop even if the while condition is true :

(2) # Example on break statement

```
i = 1
while i < 6 :
    print (i)
    if (i == 3) :
        break
i += 1
```

output :-

```
1
2
3
```

- The continue statement :-

With the continue statement we can stop the current iteration, and continue with next.

(3) # Example on continue statement :

```
i = 0
while i < 6 :
    i += 1
```

```
if i==3 :
    continue
print (i)
```

Output :

```
1
2
4
5
6
```

#Note :
#Here number 3 is missing in the result.

- The else statement :-

With the else statement we can run a block of code once when the condition no longer is true :

(4) #Example on else statement :

```
i=1
while i<6 :
    print (i)
    i+=1
else :
    print ("i is no longer less than 6")
```

Output :-

```
1
2
3
4
5
i is no longer less than 6.
```

- Python for loops :-

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

With the for loop we can execute a set of statement, once for each item in a list, tuple and set etc.

(5) # Example on for loop :-

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

output :

```
apple
banana
cherry
```

- Note :-

The for loop does not require an indering variable to set beforehand.

- Looping through a string :-

Even strings are iterable objects, they contain a sequence of characters :

(6) # Example on loop :-

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

Output :-

b
a
n
a
n
a

* **The range function** :-

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

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

7. #Example on range () function :-

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

Output :-

0
1
2
3
4
5

Note :-

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

(8) # Example using start parameter :-

```
for x in range (2,6):
    print (x)
```

output :-

```
2
3
4
5.
```

(9) # Increment sequence with 3 :

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

output :-

```
2
5
8
11
14
17
20
23
26
29
```

(10) # else in for loop :-

The else keyword in for loop specifies a block of code to be executed when the loop is finished :

```
for x in range (6):
    print (x)
else :
    print ("finally finished!")
```

output :-

```
0
1
2
3
4
5
finally finished!
```

Note :-

The else block will not be executed if the loop is stopped by a break statement

(11) # break statement :-

```
for x in range (6):
    if x==3 : break
    print (x)
else :
    print ("finally finished!")
```

Output :-

0
1
2

- 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".

(12). # print each adjective for every fruit.

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
 for x in adj :
     for y in fruits :
         print (x, y)
```

output :

```
red   apple
red   banana
red   cherry
big   apple
big   banana
big   cherry
tasty apple
tasty banana
tasty cherry
```

Now its time to practice !!

Code-set-08

Q.1. Write a program to print multiplication table of a given number using for loop.

Q.2. Write a program to greet all person names stored in a list ll and which starts with s.

ll = ["Mitasha", "khushi", "shivani", "Mamta"]

Q.3. Write a program to find whether a given number is prime or not.

Q.4. Write a program to calculate the factorial of a given number using for loop.

Q.5. Write a program to print following star pattern :

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

for n = 3.

You can solve this

Code - set - 08

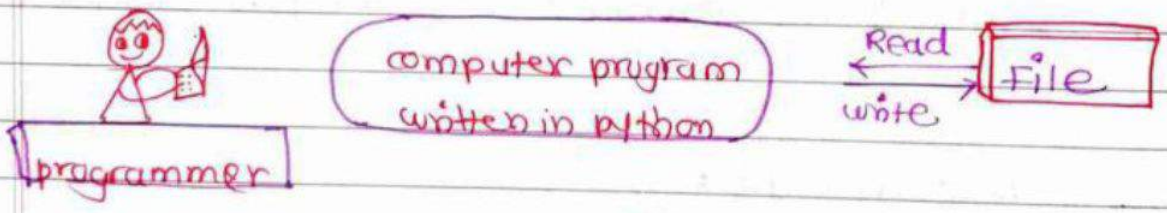
```
#1 st code :-

num = int (input ("Enter the number "))
for i in range (1, 11):
    #print (str (num) + " x " +str (i) +"= " +str(i*num)
    print (f "{num} x {i} = {num* i} ")


# 2nd code :

l1 = [ "Mitasha", "khushi ", "shivani ", "Mamta"]

for name in l1 :
    if name. startswith ("s"):
        print ("Hello ", +name)


#3rd code :

num= int (input ("Enter the number :"))
prime = True

for i in range (2, num) :
    if (num % i == 0):
        prime = False
        break
```

```
if prime :
    print ("This number is prime")
else :
    print ("This number is not prime")


# 4th code :-

num= int (input ("Enter the number : "))
factorial = 1
for i in range (1, num+1) :
        factorial = factorial *i
print ("The factorial of {num} is {factorial}")


# 5th code :

n = 4

for i in range (4):
    print ("*" * (i+1))
```

Chapter 9 : File I/O

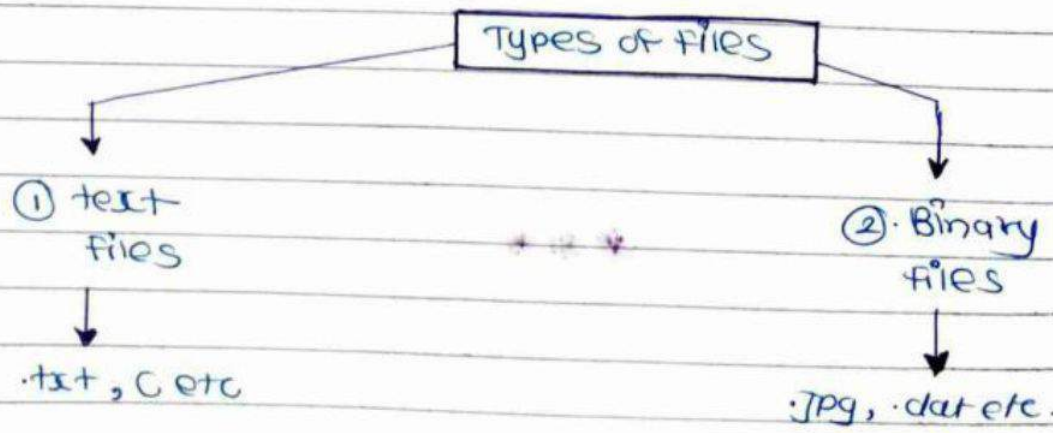
The random access memory is volatile and all its contents are lost once a program terminates. In order to persist the data forever, we use files.

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.



RAM - Volatile
HDD - Non Volatile.

- Types of files :-



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.

① open ("this.txt", "r")

↓ mode of opening (read mode).

↓ file name.

↓ Built in function.

* Reading a file :-

②
```
f = open ("this.txt", "r") → open file in r mode.
text = f.read () → Read contents of file.
print (text) → prints its contents.
f.close () → close the file.
```

We can also specify the number of character in read () function : f.read (2)

↓

Reads only first two characters.

Other methods to read the file :-

we can also use f.readline() function to read an full line at a time.

(3) f.readline() → Reads one line from file.

* Modes of opening a file :-

r	w	a	+
open for reading	open for writing	open for appending	open for updating.

"rb" will open for read in binary mode.
"rt" will open for read in text mode.

* Writing 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!

④ f = open ("this.txt", "w")
f.write ("This is nice") → can be called multiple times.
f.close ()

- With statement :-

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

⑤.
```
with open ("this.txt") as f :
    f.read()
```
↓

no need to write f.close () as it is done automatically.

⑥. # Example on opening a file :-

demofile.txt

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

output :

This file is for testing purposes.

Note :-

If the file is located in a different location, you will have to specify the file path, like above.

(7) # Return only 5 characters of file :-

```
f = open ("demofile.txt", "r")
print ( f.read (5)).
```

output :-

```
C:\Users\My Name > python demo-file-open2.py
Hello.
```

(8) # Return two lines of file :-

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

output :-

```
Hello ! welcome to demofile.txt
This file is for testing purpose.
```

(9) # loop through file line by line :

```
f = open ("demofile.txt", "r")
for x in f :
    print (x)
```

output :-

```
- ~ ~ ~ ~ ~ ~ - - - -
```

this return all the contents of file as it is.

It's time to practice !!

Code-set-09

Q.1. Write a program to read the text from a given file 'poems.txt' and find out whether it continous the word 'twinkle'.

Q.2. Write a program to generate multiplication tables from 2 to 20 and write it to different files. place these files in a folder for 13 year old.

Q.3. A file contains a word "Donkey" multiple times you need to write a program with which replaces this word with ###### by updating same file.

Q.4. Repeat program 3 for a list of such words to be censored.

Q.5. Write a program to find out whether a file identical & makes content of another file.

Think & solve!

Code-set-09

```
# 1 st code :

f = open ('poems.txt')
t = f.read ()
if 'twinkle' in t :
    print ("Twinkle is present")
else :
    print ("Twinkle is not present")
f.close ()


# 2nd code :

for i in range (2,21):
    with open(f"tables/multiplication_table_of_{i}.
            txt", "w") as f:
        for j in range (1,11) :
            f.write (f"{i} x {j} = {i*j}")
            if j! =10 :
                f.write ('\n')


# 3rd code :-

with open( "sample.txt") as f:
    content = f.read ()
```

```
content = content.replace ("donkey ", " $%^@$^#")

with open ( "sample.txt", "w") as f:
    f.write (content)


# 4th code :-

words = [ "donkey", "fat", "ugly"]

with open("sample.txt") as f:
        content = f.read()

for word in words :
        content = content.replace (word, "$%^@$^#")
        with open ("sample.txt", "w") as f:
            f.write (content)


# 5th code :-

file 1 = "log.txt"
file 2 = "this.txt"

with open (file1) as f:
        f1 = f.read ()
with open (file2) as f:
        f2 = f.read ()
if f1 == f2 :
    print ("files are identical")
else :
    print ("files are not identical")
```

Chapter 10 : Object Oriented Programming

- OOP :-

solving a problem by creating objects is one of the most popular approaches in programming. This is called as object oriented programming.

This concept focuses on using reusable code.

- Class :-

A class is a blueprint of creating objects.

class ⟹ object installation ⟹ object

① #syntax of class:

```
Class Employee:
    #methods & variables
```

② #create a class :

create a class named Myclass, with property x :

```
class Myclass :
  x = 5
print ( Myclass).
```

output:

```
<class '__main__.Myclass'>
```

- Object :-

Almost everything in python is an object, with its properties and methods.

Now we can use class named Myclass to create objects :

③ # create an object :

create an object named p1, and print value of x:

```
p1 = Myclass ()
print(p1.x)
class Myclass :
    x = 5
```

output:

```
5
```

An object is an installation of class. When class is defined, a template (info) is defined. memory is allocated only after object instantation.

object of a given class can invoke methods available to it without revealing the implementation details to the user → Abstraction & Encapsulation

* Modeling 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 :

(4). class Employee :
company = "Google" → specific to each class.

mitasher = Employee () → object instantiation
mitasha.company
Employee.company = "youtube" → changing class attribute.

* Instance Attributes :-

An attribute that belongs to the instance (object) Assumming class from previous example :

(5). mitasha.name = "Mitasha"
mitasha.salary = "50 k" → Adding instance attributes

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

* self Parameter :-

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

mitasha.getsalary () → Here, self is mitasha
↓
Equivalent to Employee.getsalary (mitasher)

The function getsalary is defined as:

(6)
```
class Employee:
    company = "Google"
    def getsalary (self):
        print ("salary is not there")
```

* static method :-

sometimes we need a function that doesn't use the self parameter.

We can define static method like this :

(7)
```
@staticmethod
def greet():
    print ("Hello User")
```
→ decorator to mark great as a static method.

* __init__() constructor :-

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

__init__() method is also known as constructor.

It takes self argument and can also take further arguments.

(8)
```
# Example on init() method.
class Employee:
    def __init__(self.name):
        self.name = name
```

```
def getsalary (self) :
        . . .

mitasha = Employee ("Mitasha")
```

↳ object can be instanticated using constructor like this !

(g). # Example

```
class Person :
    def__init__(self, name, age):
        self.name = name
        self.age = age
p1 = person ("Rahul", 42)
print (p1.name)
print (p1.age)
```

Output :-

Rahul
42.

Note :-

The __init__() function is called automatically every time the class is being used to create a new object.

* Object methods :-

methods in objects are functions that belong to the object.

(g). # Example on method

```
class person :
    def __init__(self, name, age):
```

```
    self.name = name
    self.age = age.

  def myfunc(self):
      print("Hello my name is + self.name)

p1 = Person("Ram", 36)
p1.myfunc()
```

output :

Hello my name is Ram.

It's time to practice !!

Code- set - 10

Q.1. create a class programmer for storing information of two programmers working at microsoft.

Q.2. Write a class calculator capable of finding square, cube and squareroot of a number.

Q.3. create a class with a class attribute a; create an object from it and set a directory using object a = 0, Does this change class attribute?

Q.4. Add a static method in problem 2 to greet the user with hello.

Q.5. Write a class Train which has methods to book a ticket, get status (no. of seats) and get fare information of trains running under Indian Railways.

You can solve this!!

Code-set-10

```
#1 st code :

class programmer :
    company = "Microsoft"
    def __init__(self, name, product):
        self.name = name
        self.product = product.
    def getInfo (self):
        print (f"The name of the {self.company}
            programmer is {self.name} and
            product is {self.product}")

mitasha = programmer ("mitasha", "Skype")
Ram = programmer ("Ram", "Github")
mitasha.getInfo ()
ram.getinfo ()


#2nd code :-

class calculator :
    def __init__(self, num):
        self.number = num
    def square (self):
        print (f" The value of {self.number}
            square is {self.number**2}")
```

```
def squareroot (self):
        print(f" The value of {self.number} square
            root is {self.number **0.5} ")
def cube (self):
        print(f" The value of {self.number} cube is
            {self.number **3} ")

a = calculator (g)
a.square ()
a.squareroot ()
a.cube ()
```

#3rd code :-

```
class sample :
        a = "Ram"

obj = sample ()
obj.a = "shaam"
# sample.a = " shaam"
print (sample.a)
print (obj.a)
```

#4th code :

```
class calculator :
        def __init__(self, num):
            self.number = num.
```

```
def square(self):
    print(f"The value of {self.number} square is
        {self.number **2}")

def squareroot(self):
    print(f"The value of {self.number} square
     root is {self.number **0.5}")

def cube(self):
    print(f"The value of {self.number} cube is
        {self.number **3}")
@staticmethod
  def greet():
      print("***** Hello there welcome to best calculator
              of world *****")
a = calculator(g)
a.greet()
a.square()
a.squareroot()
a.cube()
```

```
# 5th code :-

class Train :
    def __init(self, name, fare, seats):
        self.name = name
        self.fare = fare
        self.seats = seats.
```

```
def getstatus (self):
    print ("******")
    print (f"The name of train is {self.name}")
    print (f"The seats available in train are {self.seats}")
    print ("******")

def fareinfo (self):
    print (f "The price of the ticket is : RS {self.fare}")

def bookTicket (self):
    if (self.seats > 0)
        print (f "your ticket has been booked!
                your seat number is {self.seats}")
        self.seats = self.seats -1
    else :
        print ("sorry this train is full ! kindly try in tatkal")

def cancelTicket (self, seatNo):
    pass

intercity = Train ("Intercity Express : 14015", 90, 2)
intercity.getstatus ()
intercity.bookTicket ()
intercity.bookTicket ()
intercity.bookTicket ()
intercity.getstatus ()
```

* * *

Chapter 11 : Inheritance & more on OOPs.

- Inheritance :-

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

syntax :-

```
Class Employee :          → Base class / parent class.
    # code
      ...
class Programmer (Employee) : → Derived / child class
    # code
```

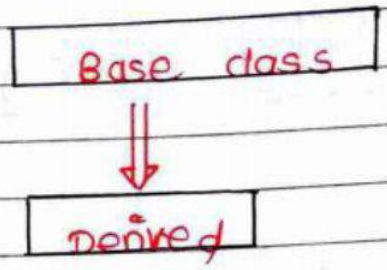
We can use the methods and attributes of Employee in programmer object.
Also, we can overwrite or add new attributes and methods in programmer class.

Types of inheritance

1. single inheritance
2. multiple inheritance
3. multilevel inheritance

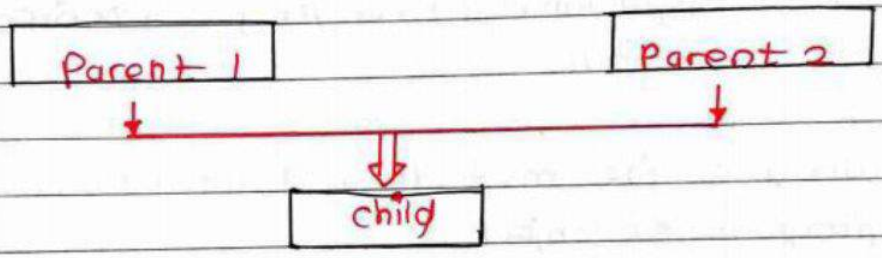
1. single inheritance :-

single Inheritance occurs when child class inherits only a single parent / base class.



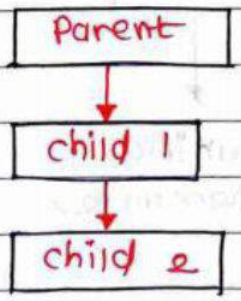
2). Multiple inheritance :-

multiple inheritance occurs when the child class inherits from more than one parent class.



3). 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 a derived class.

super() --init--()
↳ calls constructor of base class.

★ Class methods :-

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

@classmethod decorator is used to create a class method.

Syntax to create class method :

```
@classmethod
 def (cls, p1, p2) :
      # code ...
```

- @property decorators :-

```
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 / call 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 overloadding 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 :

$P_1 + P_2 \rightarrow P_1$ --add-- (P_2)

$P_1 - P_2 \rightarrow P_1$ --sub--(P_2)

$P_1 * P_2 \rightarrow P_1$ --mult--(P_2)

$P_1 / P_2 \rightarrow P_1$ --truediv--(P_2)

$P_1 // P_2 \rightarrow P_1$ --floordiv--(P_2)

① # create a parent class :-

```
class person :
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname
    def printname (self):
print (self.firstname, self.lastname)
x = Person ("John", "Doe")
x.printname ()
```

output :

John
Doe.

② # create a child class :-

```
class student (person):
    pass
```

Note :-

- Use the pass keyword when you do not want to add any other properties or methods to class.

③

```
class person :
    def __init__(self, fname, lname)
        self.firstname = fname
        self.lastname = lname
    def printname (self):
        print (self.firstname, self.lastname)
class student (person) :
    pass
x = student ("Ram", "sharma")
x.printname ()
```

output:-

Ram sharma.

④. # super () method :-

```
class person :
    def __init__ (self, fname, lname) :
        self. firstname = fname
        self. lastname = lname.
    def printname (self) :
        print (self. firstname, self. lastname)
class student ( person) :
    def __init__(self, fname, lname):
        super ( ). __init__ (fname, lname)
x = student ( "classmate", "Notebook")
x. printname ( )
```

output :

Classmate Notebook.

Note :-

By using super() function, you dont have to use a name of parent element, it will automatically inherit methods and properties of its parent.

It's time to practice !!

Code - set - 11

Q.1. Create a class C-2 vector and use it to create another class representing a 3-D vector.

Q.2. Create a class pets from a class Animals and further create class Dog from pets. Add a method barks to class Dog.

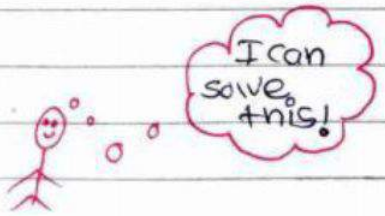
Q.3. Write a class complex to represent complex numbers, along with overloaded operators "+" and "*" which adds and multiplies them.

Q.4. Write a class vector representing a vector of a "n" dimension. overload the "+" and "*" operators which calculates the sum and dot product of them.

Q.5. Write --str--() method to print vector as follows :

$$7\hat{i} + 8\hat{j} + 10\hat{k}$$

Assume vector of dimension 3 for this problem.



Code-set-11

```
# 1st code :-

class c2dvec :
    def__init__(self, i, j):
        self.icap = i
        self.jcap = j
    def __str__(self):
        return f"{self.icap}i + {self.jcap}j"

class c3dvec (c2dvec):
    def __init__(self, i, j, k):
        super().__init__(i, j)
        self.kcap = k
    def __str(self):
        return f"{self.icap}i + {self.jcap}j + {self.kcap}k

v2d = c2dVec(1,3)
v3d = c3dVec(1,9,7)
print(v2d)
print(v3d)

'''

# 2nd code :-

class Animals :
    animalType = "Mammal"
```

```
class Pets :
    colour = "White"

class Dog :
    @staticmethod
    def bark ():
        print ("Bow bow!")

d = Dog.()
d.bark.()


    # 3rd code :-

class complex :
    def __init__ (self, r, i):
        self.real = r
        self.imaginary = i

    def __add (self, c):
        return complex (self.real + c.real, self.imaginary
                                            + c.imaginary)
    def __mul __ (self, c):
        mulReal = self.real * c.real - self.imaginary * c.imagin
        mulImg = self.real * c.imaginary + self.imaginary * c.real
        return complex (mulReal, mulImg)

    def __str__ (self):
        if self.imaginary < 0 :
            return f"{self.real} - {-self.imaginary}i"
```

```
else:
    return f"{self.real} + {self.imaginary}i"

c1 = Complex(1,-4)
c2 = Complex(331,-37)
print(c1 + c2)
print(c1 * c2)


# 4th code :-

class vector:
    def__init__(self, vec):
        self.vec = vec

    def__str__(self):
        str1 = " "
        index = 0
        for i in self.vec:
            str1 += f"{i}a{index} + "
            index + = 1
        return str1[:-1]

    def__add__(self, vec2):
        newlist = []
        for i in range(len(self.vec)):
            newlist.append(self.vec[i]+vec2.vec[i])
        return Vector(newlist)
```

```
def __mul__(self, vec2):
    sum = 0
    for i in range (len(self.vec)):
        sum += self.vec[i] = vec2.vec[i]
    return sum

v1 = Vector([1, 4, 6])
v2 = Vector([1, 6, 9])
print (v1 + v2)
print (v1 * v2)


# 5th code:

class vector :
    def __init__(self, vec):
        self.vec = vec
    def __str__(self):
        return f"{self.vec[0]}i + {self.vec[1]}j
                + {self.vec[2]}k"

v1 = vector ([1, 4, 6])
v2 = vector ([1, 6, 9])
print(v1)
print (v2)
```

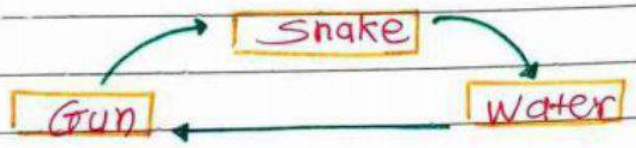
***

Project_fun_01 : snake, water, gun game.

Que: Write a program (python) capable of playing this game with user (snake, water & gun)

- We all have played snake, water, gun game in our childhood.
- If you haven't, google the rules of this game, and play with us!

→




```
import random
# snake water Gun or Rock paper scissors :
def game win (comp, you):
    # If two values are equal, declare a tie !!!
    if comp == you :
        return None

# check for all possibilities when computer chose s
    elif comp == 's' :
        if you == 'w' :
            return false
        elif you == 'g' :
            return True

# check for all possibilities when computer chose w
    elif comp == 'w' :
        if you == 'g' :
```

```
        return false
    elif you == 's':
        return True

# check for all possibilities when computer chose g
elif comp == 'g':
    if you == 's':
        return false
    elif you == 'w':
        return True

print("comp Turn: snake(s) Water(w) or Gun (g)?")
randNo = random.randint(1,3)
if randNo == 1:
    comp = 's'
elif randNo == 2:
    comp = 'w'
elif randNo == 3:
    comp = 'g'

you = input("your Turn: snake(s) Water(w) or Gun(g)
a = gamewin(comp, you)

print(f"computer chose {comp}")
print(f"You chose {you}")

if a == None:
    print("The game is a tie!")
elif a:
    print("you win!")
```

```
else :
    print ("you loose!")
```

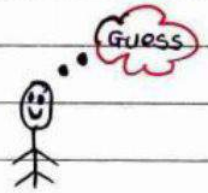
* * *

Project – fun-2 : The perfect Guess.

Que: write a program that generates a random number and asks the user to guess it. If the players 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 correct number, the program displays number of guesses the player used to arrive at the number.

Hint: (use the random module)

→



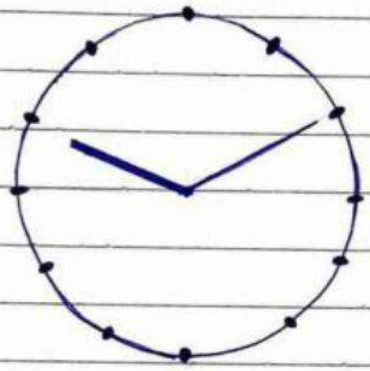

```
import random
randNumber = random.randint(1,100)
userGuess = None
guesses =0

While (userGuess != randNumber):
  userGuess = int(input("Enter your guess:"))
  guesses += 1
  if (userGuess == randNumber):
     print ("You guessed it right!")
  else :
       if (userGuess > randNumber):
```

```
        print("You guessed it wrong! Enter a smaller number")
    else:
        print("You guessed it wrong! Enter a larger number")

print(f"you guessed the number in {guesses} guesses")
with open("hiscore.txt","r") as f:
    hiscore = int(f.read())

if (guesses < hiscore):
    print("You have just broken high score!")
    with open("hiscore.txt","w") as f:
        f.write(str(guesses))
```

***

Project_fun_3 : Alarm Clock

que: Write a simple python program for a command line alarm clock that plays a random youtube video when it goes off.



```
"""" Alarm clock
-------------------
"""

import datetime
import os
import time
import random
import webbrowser
 // If video URL file does not exist, create one
if not
os.path.isfile("youtube_alarm_videos.txt"):
    print('creating "youtube_alarm_videos.txt"....')
    with
    open("youtube_alarm_videoes.txt","w") as alarm_file:
```

```
alarm_file.write("https://www.youtube.com/watch?
                v=anMGulzvo74")
def
check_alarm_input(alarm_time):
    """ checks to see if the user has entered in a
valid alarm time """
        if alarm_time[0]<24 and alarm_time[0]>=0:
            return True

        if len(alarm_time)==2 :
        //[Hour:Minute] format

            if alarm_time[0]<24 and
                    alarm_time[0]>==0 and
                    \alarm_time[1]<60 and
                     alarm_time[1]>=0 :
            return true

        elif len(alarm_time)==3 :
    // [Hour:Minute:second] format

        if alarm_time[0]<24 and
            alarm_time[0]>=0 and \
             \alarm_time[1]<60 and
              alarm_time[1]>=0 and
               \alarm_time[2]<60 and
                alarm_time[2]>=0 :
        return True
    return false.
```

```
// Get user input for alarm time.
    print ("set a time for the alarm
     (Ex:06:30 or 18:30:00)")
    while True:
        alarm_input = input (">>")
    try:
        alarm_time = [int (n) for n in alarm_input
                        .split (":")]
        if
        check_alarm_input (alarm_time):
            break
        else:
            raise ValueError
    except ValueError:
        print ("ERROR: Enter time in HH:MM or
                HH:MM:SS format")

// convert the alarm time from [H:M] or [H:M:S]
to seconds
seconds_hms = [3600, 60, 1]
// Number of seconds in an Hour, Minute and Second
alarm_seconds = sum ([a*b for a,b in zip
                (seconds_hms[:len(alarm_time)],
                alarm_time)])

// Get the current time of day in seconds
now = datetime.datetime.now()
current_time_seconds = sum ([a*b for a,b in
                zip (seconds_hms, [now.hour,
                now.minute, now.second])])
```

```
// calculate the number of seconds until alarm
goes off

time_diff_seconds = alarm_seconds - current_time_sec

// If time difference is negative,
  set alarm for next day

if time_diff_seconds < 0:
        time_diff_seconds += 86400
// number of seconds in a day.
// Display amount of time until alarm goes off
print ("Alarm set to go off in %s" %
        datetime.timedelta(seconds = time_diff_seconds)
// sleep until alarm goes off time.sleep(time_diff
                                        _seconds)

// Time for the alarm to go off
  print ("Wake Up!")
// Load list of possible video URLs
  with
    open ("youtube_alarm_videos.txt", "r")
     as alarm_file:
      videos = alarm_file.readlines()
// open a random video from list.
webbrowser.open (random.choice)
```

***

Project_fun_4 : student Library

Que: Implement a student library system using oop's where students can borrow a book from the list of books.

- Create a seprate library and student class.
- Your program must be menu driven.

You are free to choose methods and attributes of your choice to implement this functionality.

→

```
class Library :
    def __init__(self, listOfBooks):
        self.books = listOfBooks

    def displayAvailableBooks(self):
        print("Books present in this library are:")
        for book in self.books :
            print("* " + book)

    def borrowBook (self, bookName):
        if bookName in self.books :
            print(f "You have been issued {bookname}.
                    Please keep it safe and return in 30 days.
            self.books.remove(bookName)
            return True
        else :
```

```
        print ("Sorry, This book is either not available
                or has already been issued to someone
                else. Please wait until book is available")
        return false

    def returnBook (self, bookName):
        self. books. append(bookName)
        print ("Thanks for returning this book! Hope you
                enjoyed reading it. Have a great day ahead")

class Student :
    def requestBook (self):
        self. book = input("Enter the name of book you
                            want to borrow")
        return self.book

    def returnBook (self):
        self.book = input ("Enter the name of book you
                            want to return")
        return self. book

if __name__ == "__main__":
    centralLibrary = Library ([ "Algorithms", "Django", "clrs",
                                "python Notes"])
    student = Student()
    # centralLibrary. displayAvailableBooks()
    while (True):
        welcomeMsg = ''' \n ====Welcome to Central Library==
        Please choose an option :
```

```
1. list all the books
2. Request a book
3. Add / Return a book
4. Exit the library
'''
print (welcome mgg)
a = input ("Enter a choice : "))
if a == 1 :
    centraLibrary. displayAvailableBooks())
elif a == 2 :
    centraLibrary. borrowBook (student. requestBook())
elif a == 3 :
    centralibrary. returnBook (student. returnBook())
elif a == 4 :
    print ("Thanks for choosing central Library.
            Have a great day ahead!")
    exit ()
else :
    print ("Invalid choice!")
```

***

Project_fun_5 Indian flag

que: Draw indian flag (tricolour-orange, white and green) with python.

→

```
import turtle
from turtle import *

# screen for output
screen = turtle.Screen()

# Defining a turtle Instance
t = turtle.Turtle()
speed(0)

# initialy penup()
t.penup()
t.goto(-400, 250)
t.pendown()

# Orange Rectangle
# white Rectangle
t.colour("orange")
t.begin-fill()
```

```
t.forward (200)
t.right (90)
t.forward (167)
t.right (90)
t.forward (200)
t.end_fill ()
t.left (90)
t.forward (167)

# Green Rectangle
t.colour ("green")
t.begin_fill ()
t.forward (167)
t.left (90)
t.forward (200)
t.left (90)
t.forward (167)
t.end_fill ()

# Big Blue Circle
t.penup ()
t.goto (70, 0)
t.pendown ()
t.colour ("navy")
t.begin_fill ()
t.circle (70)
t.end_fill ()
```

```
# Big white circle
t.penup()
t.goto(60, 0)
t.pendown()
t.colour("white")
t.begin_fill()
t.circle(60)
t.end_fill()

# mini Blue circles
t.penup()
t.goto(-57, -8)
t.pendown()
t.colour("navy")
for i in range(24):
    t.begin_fill()
    t.circle(3)
    t.end_fill()
    t.penup()
    t.forward(15)
    t.right(15)
    t.pendown()

# Small Blue circle
t.penup()
t.goto(20, 0)
t.pendown()
t.begin_fill()
t.circle(20)
t.end_fill()
```

```
# Spokes
 t.penup ()
 t.goto (0,0)
 t.pendown ()
 t.pensize (2)
 for i in range (24):
        t.forward (60)
        t.backward (60)
        t.left (15)

# to hold the
# output window
 turtle.done ()
```

***

Thank You !!