Gocourse

Programming in Python

GOCOURSE

www.gocourse.in

2023

Price:**Rs.349/-**

MSME registration no: UDYAM TN-02-0168686

ISBN: 978-93-5906-242-6

GOCOURSE

Website: www.gocourse.in

Email: gocourse00@gmail.com

For online purchase visit our website

PREFACE

Welcome to the world of Python programming! This book is intended for beginners who are interested in learning the basics of the Python programming language.

Python is a popular high-level, interpreted programming language that is widely used for web development, scientific computing, data analysis, artificial intelligence, and many other applications. Its popularity is due to its simplicity, readability, and versatility.

This book is designed to provide you with a solid foundation in Python programming. It covers the basic concepts, syntax, data types, control structures, functions, and object-oriented programming in Python.

Each chapter contains practical examples and exercises to help you understand the concepts and apply them in real-world scenarios. The book assumes no prior programming experience, so even if you are a complete beginner, you can follow along with ease.

By the end of this book, you should be able to write basic Python programs, understand the core concepts of programming, and have a solid foundation for further study or application development.

So, let's dive into the exciting world of Python programming!

ACKNOWLEDGEMENTS

I sincerely thank our professor Mrs. B.Sowmiya for helping me in shaping the book by reviewing and carrying out corrections wherever necessary.

I am deeply grateful to the following individuals who have played a significant role in the creation of this book:

My family, for their unwavering support, encouragement, and love throughout this journey. Thank you for believing in me and always cheering me on.

My friends, for their constant motivation, feedback, and inspiration. Your enthusiasm and encouragement kept me going, and I am grateful for your friendship.

My beta readers, for taking the time to read early drafts of this book and providing me with constructive feedback. Your input was immensely valuable in improving the story and characters.

The writing community, for their support, camaraderie, and invaluable resources. From writing groups to workshops to conferences, I have learned so much from fellow writers and industry professionals.

The team at my publisher, for their hard work, expertise, and commitment to bringing this book to life. Your passion for books and love for storytelling are evident in every aspect of this publication.

Lastly, I want to express my heartfelt appreciation to my readers. Your support, feedback, and enthusiasm for my writing mean the world to me. Thank you for embarking on this literary journey with me.

I am truly grateful to all those who have supported me along the way, and I am honored to share this book with you.

BY Gocourse team

FEEDBACK

**You are most welcome to send the valuable suggestions towards the improvement of
this book. Please feel free to reach us at gocourse00@gmail.com and rsuriya119@gmail.com.**

AUTHOR DETAILS

R.Suriya

- **CEO of Gocourse pvt ltd .**
- **Pursuing underGraduate degree (BCA) at S.I.V.E.T College , Chennai ,Tamil Nadu .**
- **He is a certified Cyber security analyst & Ethical hacker.**

G. Sai Roshni

- **Manager of Gocourse pvt ltd .**
- **Pursuing underGraduate degree (BCA) at S.I.V.E.T College , Chennai ,Tamil Nadu .**
- **She has presented & published papers at International conferences.**

V. Maha lakshmi

- **Author at Gocourse pvt ltd .**
- **Pursuing underGraduate degree (BCA) at S.I.V.E.T College , Chennai ,Tamil Nadu .**

Amirtha Vembu

- **Author at Gocourse pvt ltd .**
- **Pursuing underGraduate degree (BCA) at S.I.V.E.T College , Chennai ,Tamil Nadu**
- **She has presented & published papers at International conferences.**

N.Seema Roselin

- **Author at Gocourse pvt ltd .**
- **Pursuing underGraduate degree (BCA) at S.I.V.E.T College , Chennai ,Tamil Nadu .**
- **She has presented & published papers at International conferences.**

TABLE OF CONTENT

UNIT-1 PYTHON INTRODUCTION

1.1 -Introduction
1.2 -Variables
1.3 -Literal and identifiers
1.4 -Key words
1.5 -Input and output statement
1.6 -Comment statement
1.7 -Operators
1.8 -Data types
1.9 -Type conversion

UNIT-2 WRITING SIMPLE PROGRAMS

2.1 -Write a program to sum any three integers
2.2 -Write a program Add three integer to using input statement
2.3 -Write program to print a String word "welcome to Gocourse" and also using input statement

UNIT-3 CONTROL STRUCTURES

3.1 -Control Structures
3.2 -Selection control statement
3.3 -For loop
3.4 -While loop
3.5 -Break and Continue statement

UNIT-4 LIST

4.1 -list
4.2 -List Traversal
4.3 -List Methods and Operations
4.4 -List membership Test
4.5 -Iterating over list

UNIT-5 FUNCTION

- 5.1 -Function
- 5.2 -Arguments
- 5.3 -Variable scope
- 5.4 -Recursion

UNIT-6 TURTLE GRAPHICS

- 6.1 -Turtle Graphics
- 6.2 -Turtle attributes & methods

UNIT-7 MODULAR DESIGN

- 7.1 -Modular Design
- 7.2 -Python Modules

UNIT-8 FILES

- 8.1 -Files
- 8.2 -Open
- 8.3 -Read
- 8.4 -write

UNIT-9 STRINGS

- 9.1 -String and String Array
- 9.2 -String processing

UNIT-10 EXCEPTIONS

- 10.1 Exception Handling

UNIT-11 DICTIONARIES

- 11.1 -Dictionaries

UNIT-12 SETS

12.1 -sets

UNIT-13 OOPS CONCEPTS

13.1 -Oops concepts
13.2 -Class and Objects
13.3 -Encapsulation
13.4 -Polymorphism
13.5 -Inheritance
13.6 -Data abstraction

LAB PROGRAMS

1.Temperature conversion
2.Preparation of student mark list
3.Find the area of circle,square,rectangle and triangle
4.Fibonacci series
5.Factorial using recursion
6.Counting the even and odd numbers in an array
7.Counting the uppercase and lowercase letters
8.Palindrome checking
9.Sum of all items in a Dictionary
10.Patten construction
11.Copy the file contents
12.Turtle graphics window creation
13.Tower of Hanoi
14.Hangman game

1

INTRODUCTION

1.1 INTRODUCTION

Introduction to Python:

Python is a high-level programming language. It is object oriented . It is simple to learn .

Python language had been used by:

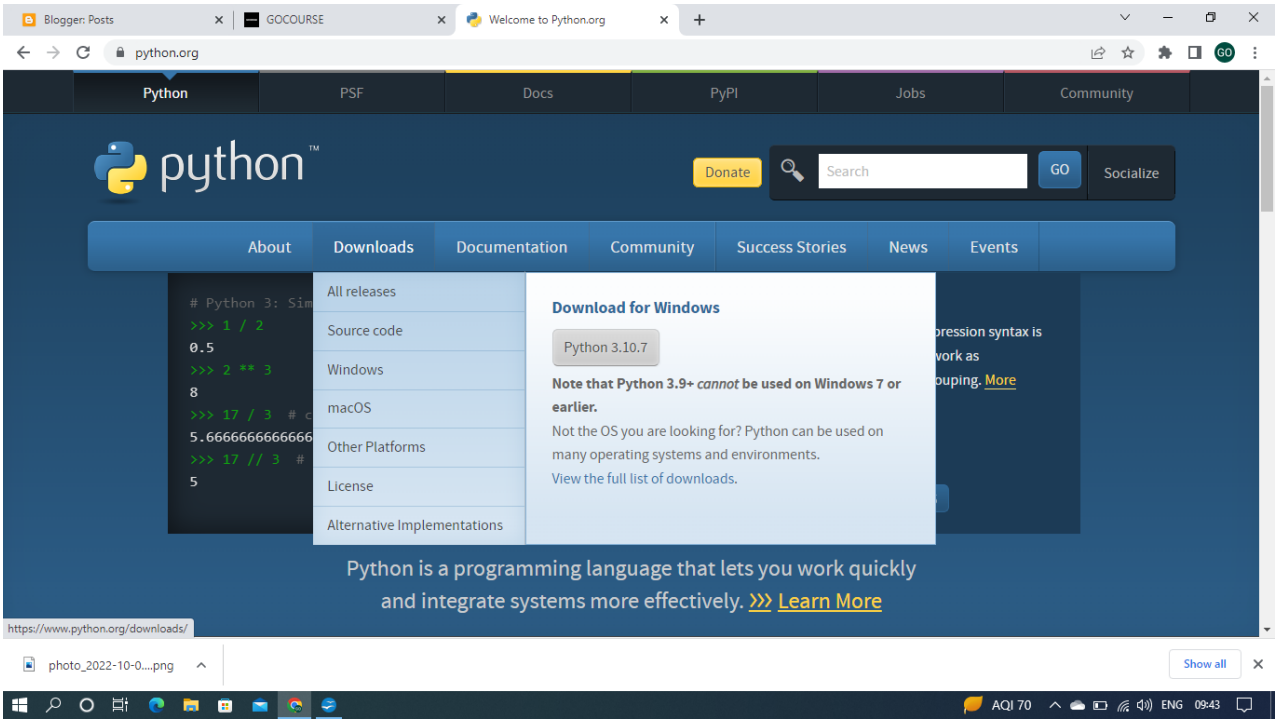
- Google
- You tube
- Blogger

Who Developed Python programming:

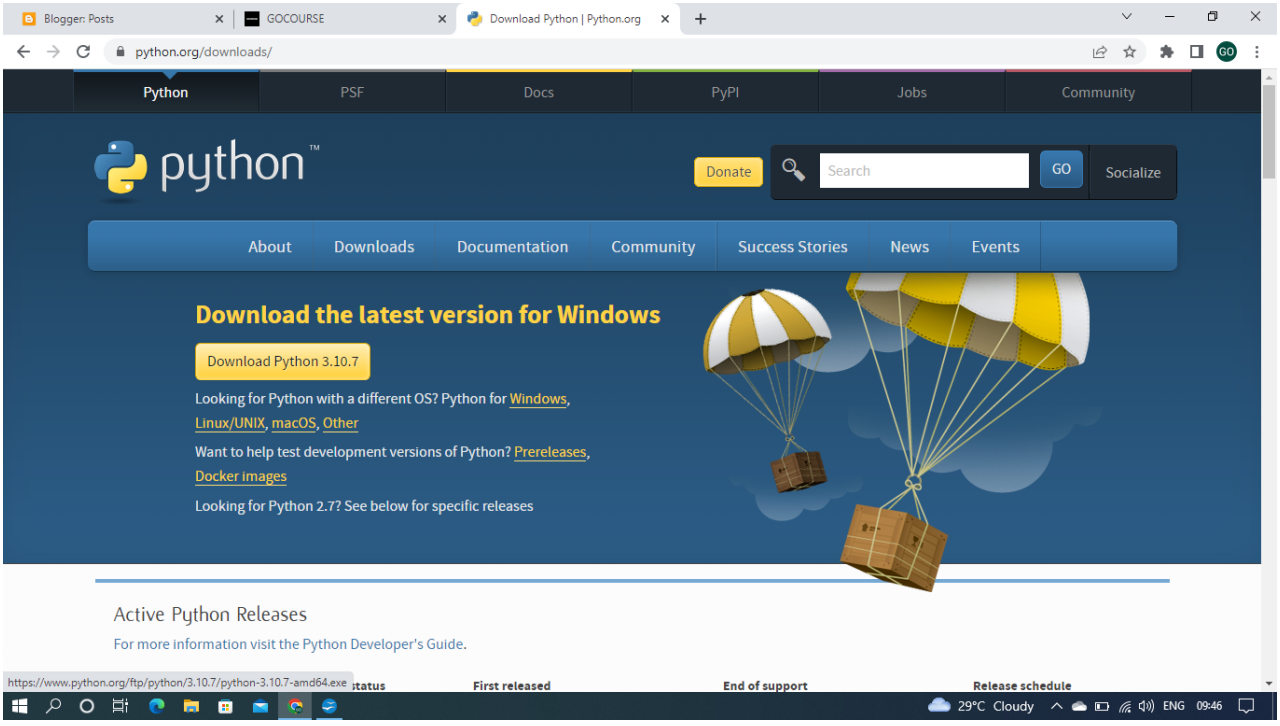
Guido van rossum created Python programming in the 1990s, and the first version was released in 1991. The name Python came from the BBC television show "Monty Python's Flying Circus," which he loved a lot. He also wanted something short, unique, and mysterious for his invention, so he named it Python.

HOW TO DOWNLOAD:

1. It is open source and freely Available at https://www.python.org/

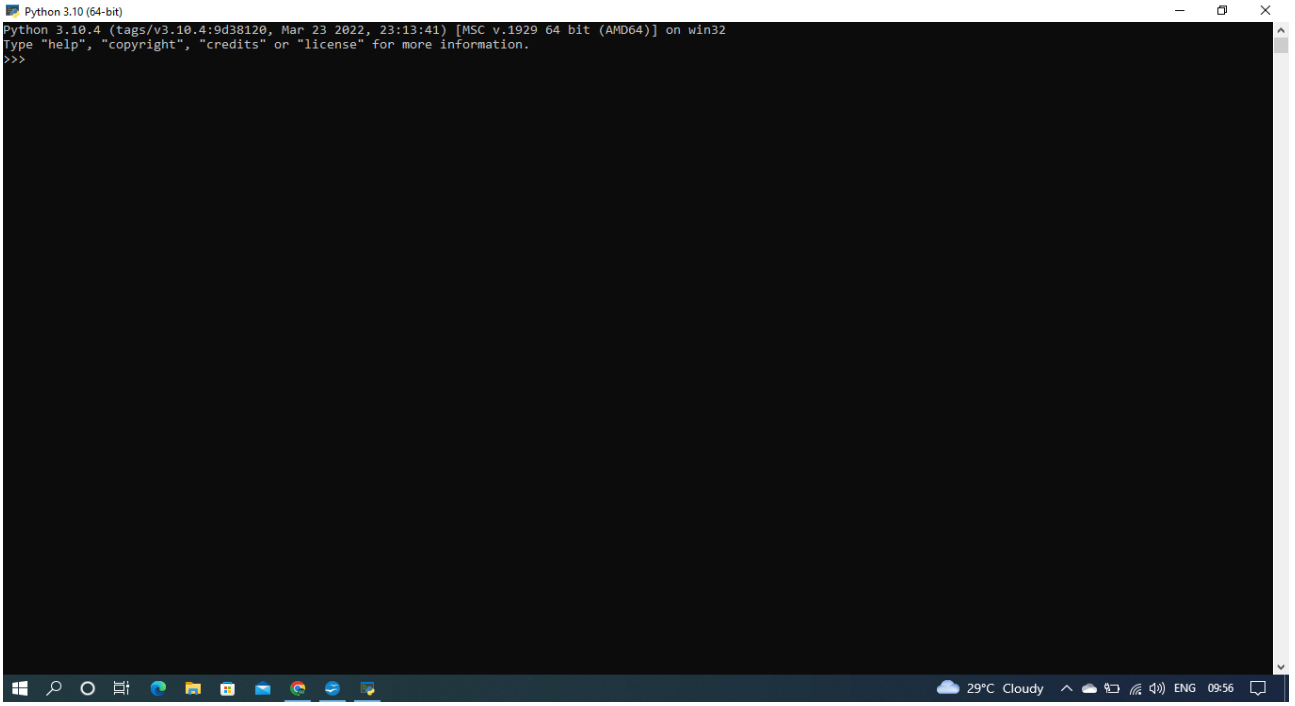


2.Click downloads button



3.Click download python 3.10.7 .Python will be download latest version

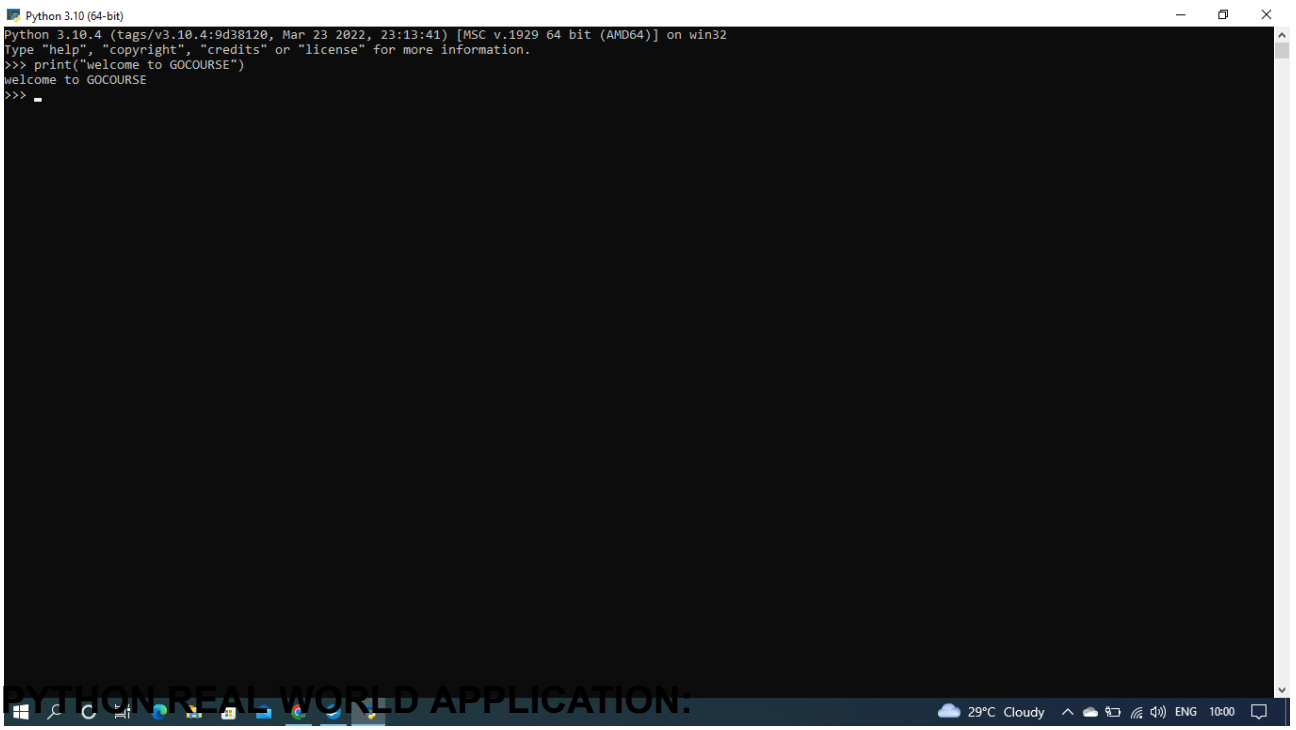
4.After install your computer



It opens python command line screen

EXAMPLE:1

Type print(“welcome to GOCOURSE”)



PYTHON REAL WORLD APPLICATION:

1. Web Development
2. Game Development

Python shell :

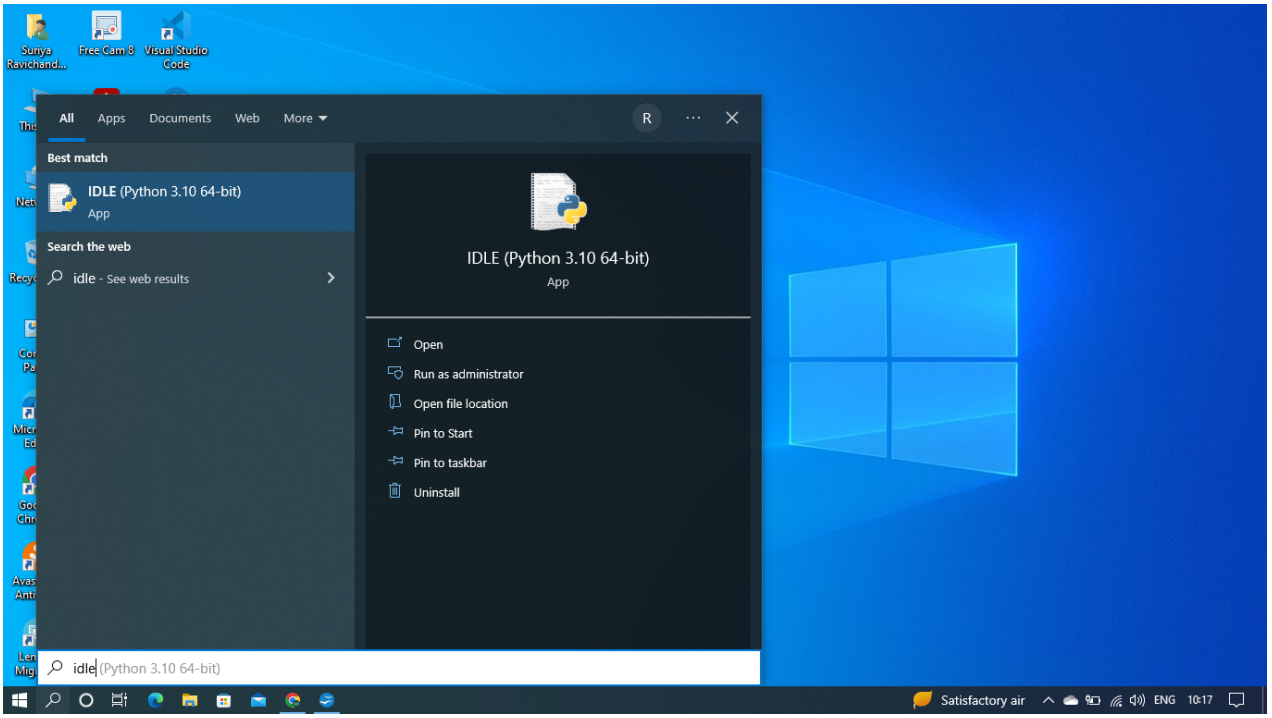
Write python coding / script using python IDLE.

IDLE full form (integrated development and learning environment)

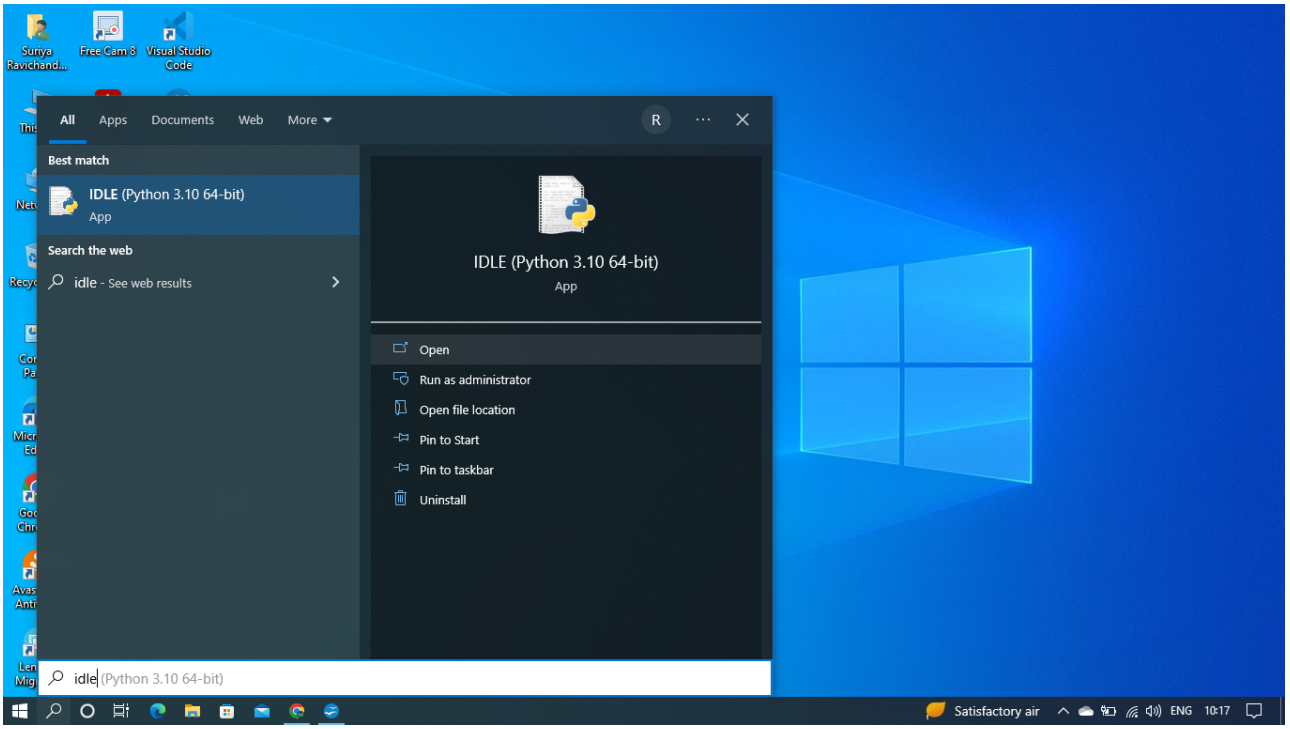
HOW TO OPEN PYTHON IDLE :

To open python idle command line screen, Go to

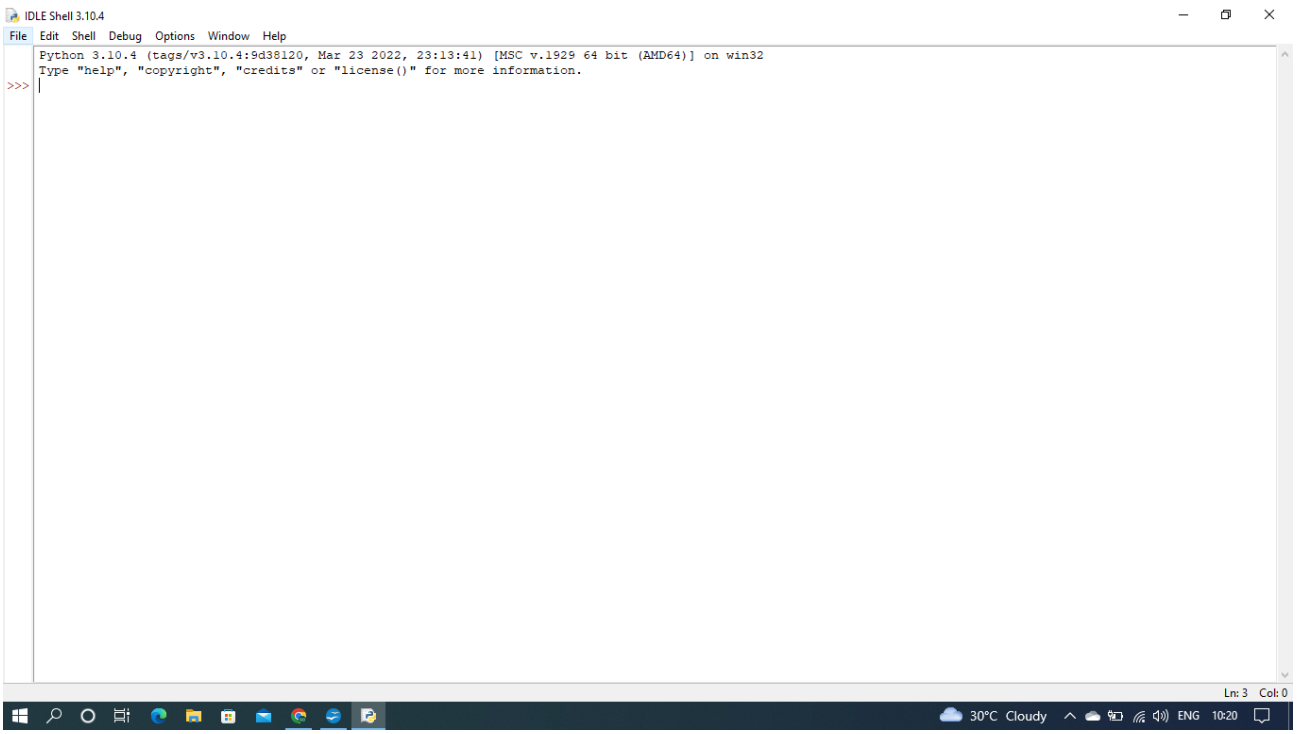
start -> click IDLE (python 3.10 64-bit).



Click open button

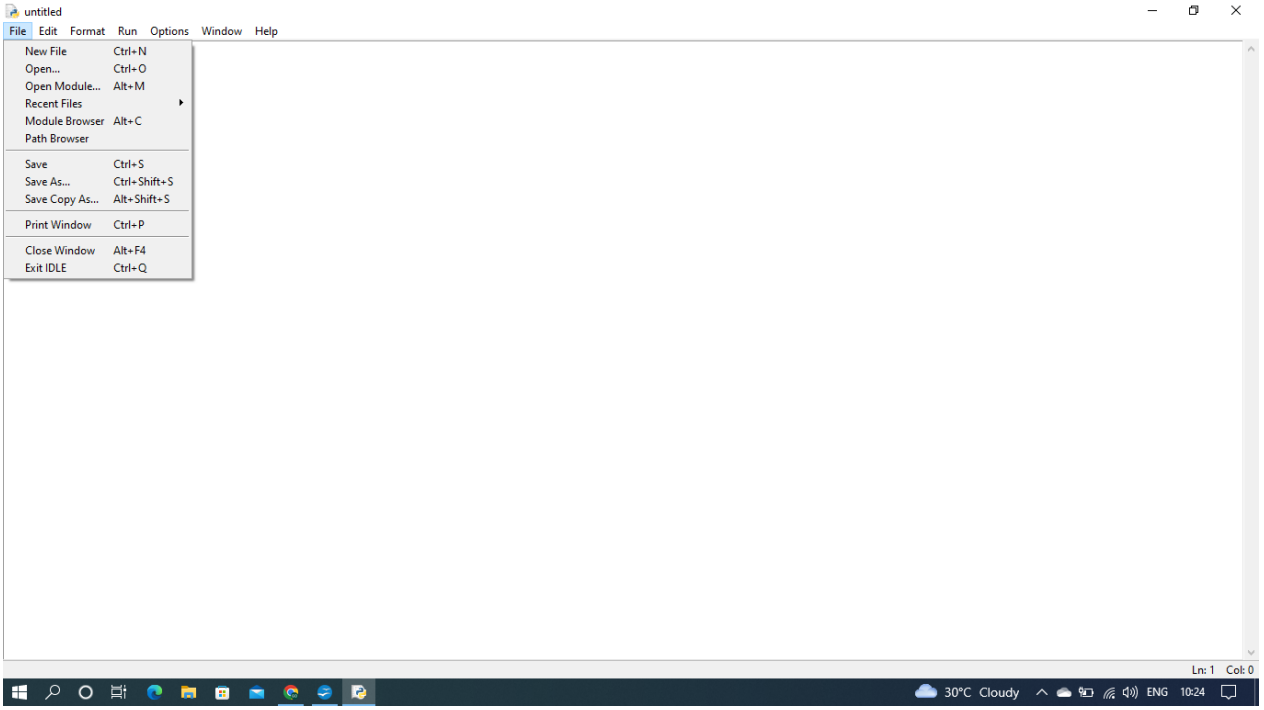


Click to new file

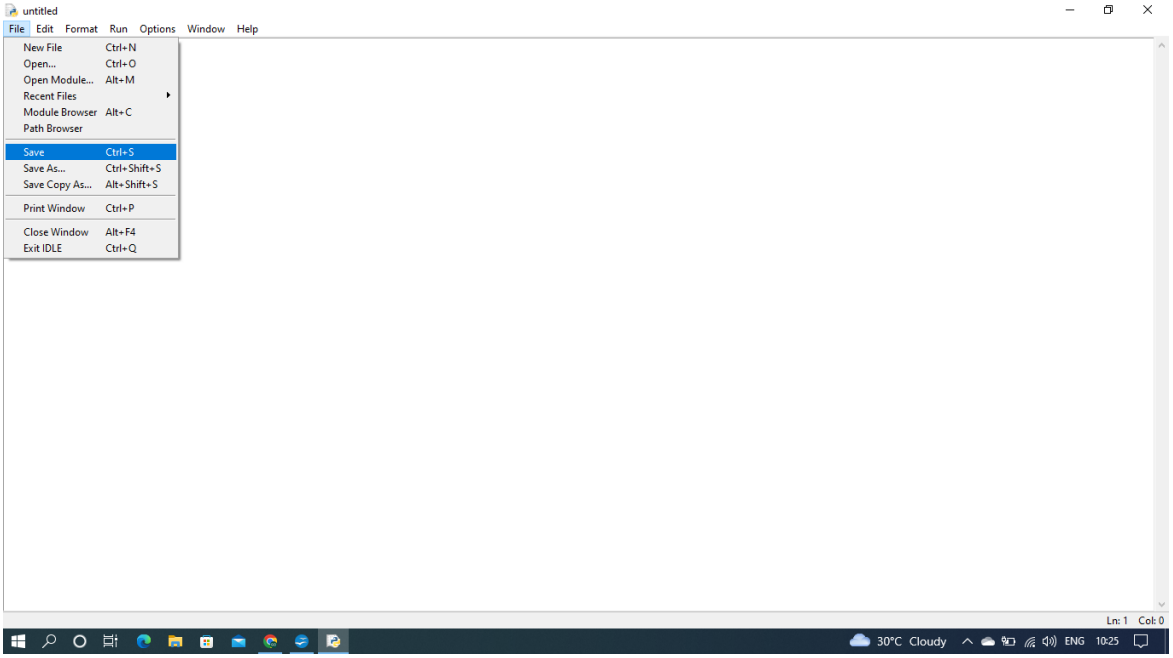


HOW TO SAVE FILE :

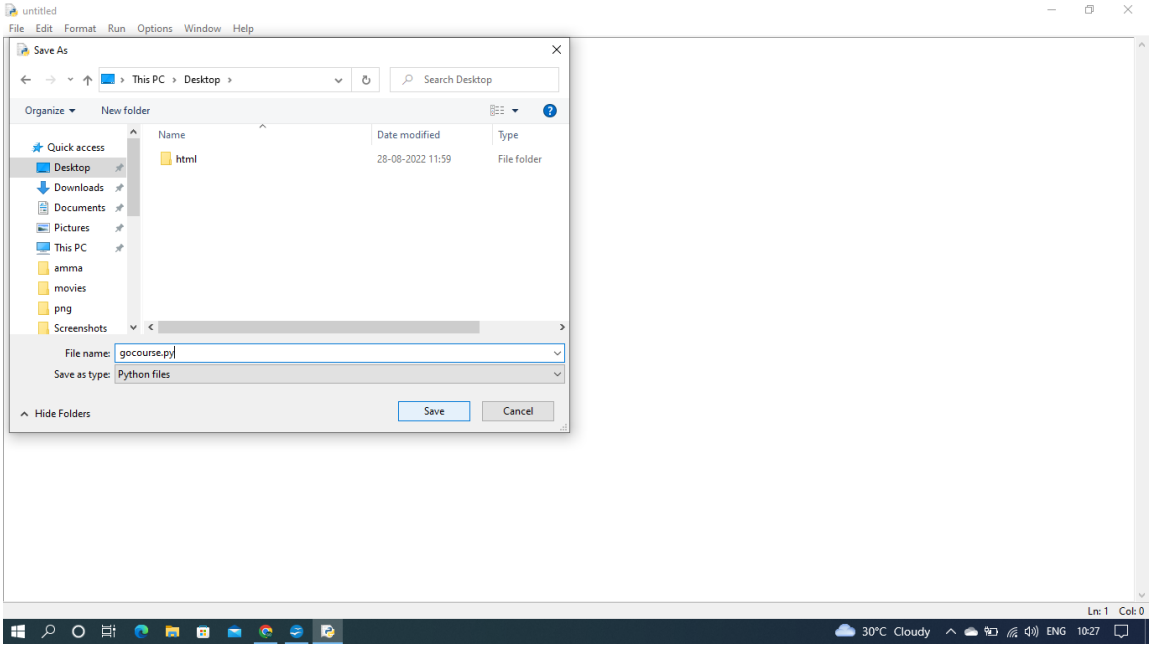
- click file



- click save or ctrl+s

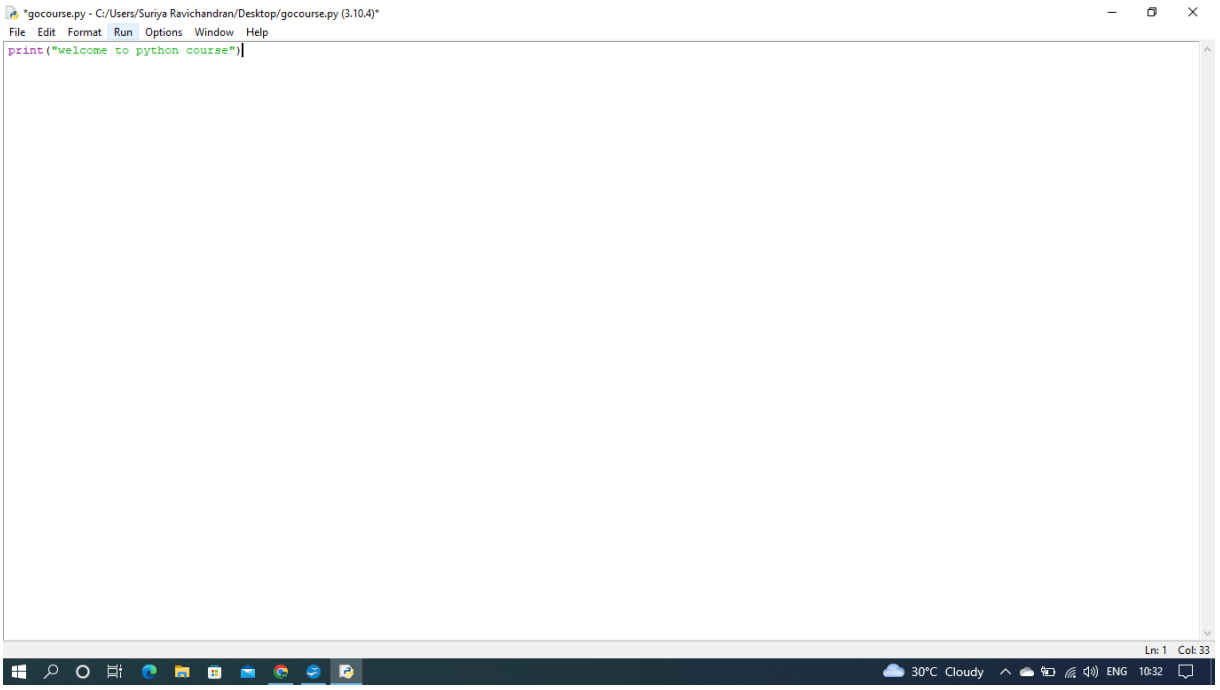


- Select your location extensions filename.py

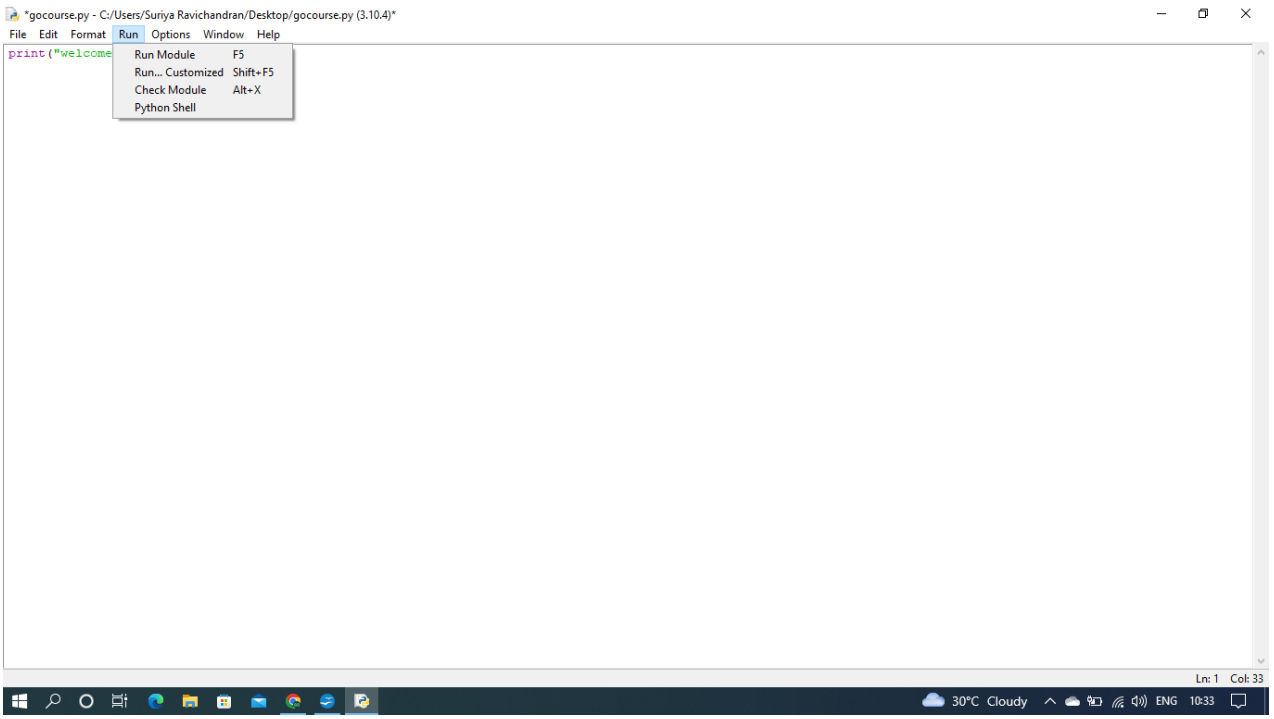


Example: 2

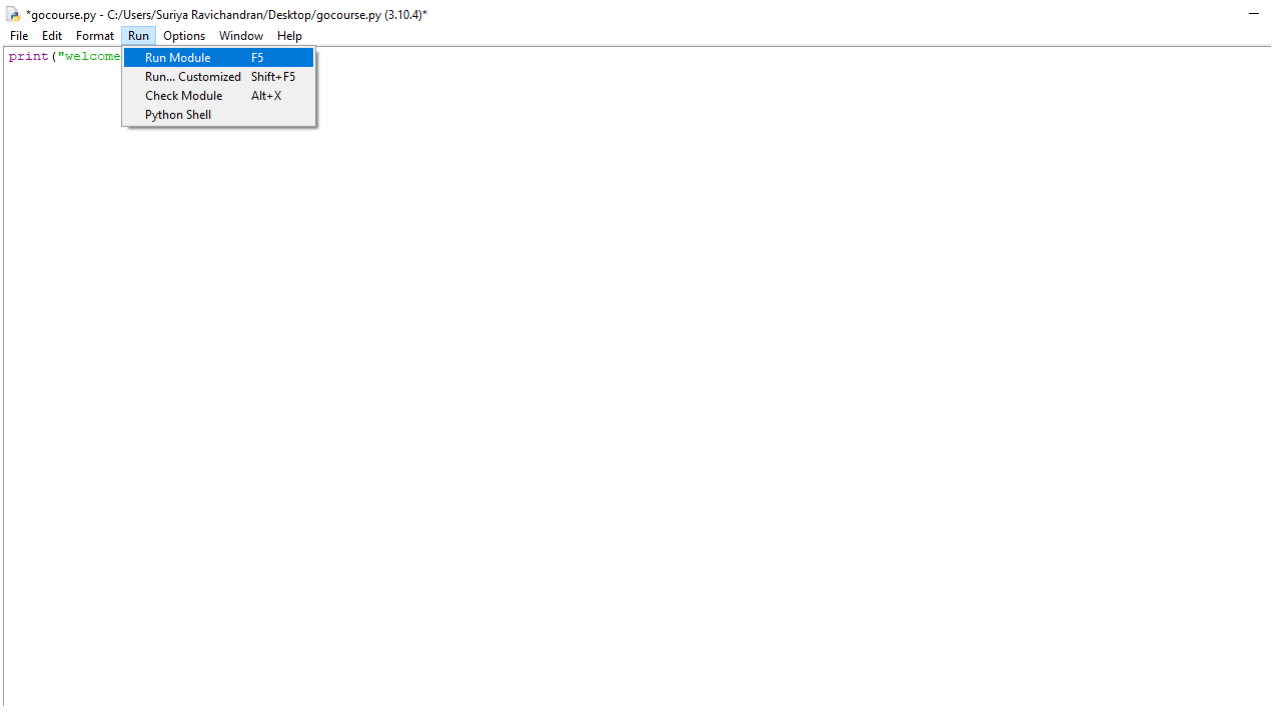
Type : print("welcome to python course")



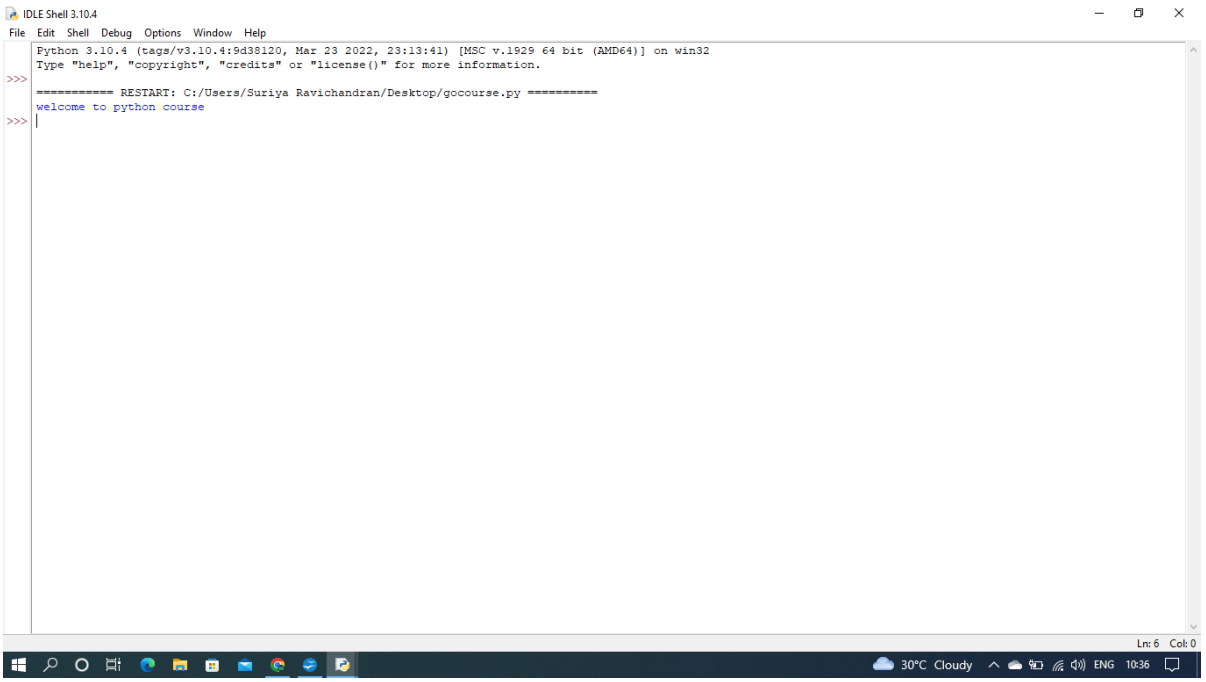
Click to run



Click “Run module” or “f5”



Output screen:



1.2 Variables :

- The memory location known as variables is where data is stored.
- It can hold a value like a container and change its value during execution.
- When assigning a value to a variable, we use the assignment operator (=).

TYPES OF VARIABLES:

- 👉 Integer
- 👉 Long integer
- 👉 Float
- 👉 String

In contrast to other programming languages, Python does not require you to specify the type of value it holds nor does it have a command to create a variable. When a value is first assigned to a variable, it is created.

(E.g): input =10 Here, input is the variable name.

The assignment operator's name is =.

It has been given the value of 10.

RULES TO DECLARE A VARIABLE:

1. Every variable name begins with an underscore or alphabet.
2. Numeric names should not be used to begin variables.
3. The name of a variable can only contain underscore (_) and alphanumeric characters (A-Z, a-z, 0-9).
4. The names of variables are case-sensitive.

RIGHT😊WAY TO DECLARE A VARIABLE:

- Beginning with underscores is permitted (first_variable=10). Beginning with underscores is permitted.

WAYS NOT TO❌ FOLLOW TO DECLARE A VARIABLE:

- (first variable=10), the space between the words.
- Beginning with the number - (78th variable = 10)
- (first-variable=10, first+variable=10) Special characters are not permitted.
- Beginning with the number - (third-firstvariable = 10)

1.3 Literals and Identifiers

LITERALS

Literals can be defined as data that is given in a variable or constant. The values can be a number, character, string or boolean values.

There are four five types of literals:

- Numeric literals
- string literals
- boolean literal
- special literal
- literal collection

NUMERIC LITERAL:

Numeric literals are immutable(can't be changed) and it can be an integer ,float and complex values.

EX:

x=156
y=100.456

STRING LITERAL:

String literals are enclosed within quotes. We can use either single or double quotes.

EX:

'hi','"welcome"

BOOLEAN LITERAL:

Boolean literal can have only two values either true or false.

SPECIAL LITERAL:

Python has only one special literal called "none".

LITERAL COLLECTION:

Literal collection segment to specify values for a collection data type.

IDENTIFIERS

An identifier is any name given to a program element such as variables, functions, class ,variables modules or objects in python .Identifiers may contain letters, digits and underscore, but it should not begin with numbers.

Rules for using python Identifiers:

- An identifier name should not be a keyword.
- An identifier name can begin with a letter or an underscore only.
- An identifier name can contain both numbers and letters along with the underscore(A-Z, 0-9,_)
- An identifier's name in python is case-sensitive .

1.4 KEYWORDS:

- A keyword is an identifier which has some Pre-defined meaning in python.
- Keyword cannot be re-defined

The following are list of Python keywords:

1. and
2. as
3. assert
4. break
5. class
6. continue
7. def
8. del
9. elif
10. else
11. except
12. finally
13. for
14. from
15. if
16. import
17. global
18. in
19. is
20. not
21. or
22. pass
23. raise
24. return
25. try
26. while
27. with
28. yield
29. false
30. none
31. true

1.5 Input & Output statements

Input statement:

- **Input()** keyword used to get information from the user is called a input statement.

Example:

```
name= input("Enter your name")
```

hobby=input("Enter your hobby")

Output statement:

- **print()** keyword used to get output on the screen is called a output statement.

Example:

print(name)

print(hobby)

Program:

name= input("Enter your name")#Abdul Kalam

hobby=input("Enter your hobby")#Invention

print(name)

print(hobby)

Output:

Abdul Kalam

Invention

1.6 Comment statement in Python:

Python comment starts with **Hash(#)** character. Python does not support multiline comments. To add multiline comments insert # in each line.

It helps to explain the code.

Example:

Print("hi") # print() is used to display statement

myname=input("enter the name") #input() is used to read input from the user

Program:

myname=string(input("enter the name"))#input() is used to read input from the user

print(myname)

Output:

Abdul Kalam

1.7 Operators

Operators are used to perform mathematical and logical operations in the program.

1. Arithmetic operators
2. comparison operators
3. assignment operators
4. logical operators

Arithmetic operators

It is used to perform arithmetic operations between values.

- Addition(+)
- Subtraction(-)
- Multiplication(*)
- Divide(/)
- Reminder(%)
- Floor division(//))
- Exponent(**)

Comparison operators

Comparison operator is used to compare two values and return boolean values true/false as result.

- Less than(<)
- Less than or equal to(<=)
- Greater than(>)
- Greater than(>=)
- Equal to(==)
- Not equal (!=)

Assignment operator

It is used to assign values on the right side to the left side variable

OPERATOR	OPERATOR	EQUIVALENT EXPRESSION(m=15)	RESULT
(=)	y=a+b	y=10+20	30
(+=)	m+=10	m=m+10	25
(-=)	m-=10	m=m-10	5
(*=)	m*=10	m=m*10	150
(/=)	m/=10	m=m/10	1.5
(%=)	m%=10	m=m%10	5
(**=)	m**=10	m=m^2	22.5
(//=)	m//=10	m=m//10	1

Logical operator

Logical operator is used to evaluate expressions and to make decisions.

OPERATOR	MEANING	EXAMPLE	RESULT
AND	Logical and	(a<15) and (b>2)	true
OR	Logical or	(a>40) or(b>3)	true
NOT	Logical not	not(5<2)	False

1.8 Data Types

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.

Some built-in Python data types are:

- Numeric
- Sequence
- Dictionary
- Boolean
- Set

Numeric data types:

TYPE	DESCRIPTION	EXAMPLE
Integer	It is a whole number, positive or negative,	-90,45,0-78

	without any fractions.	
float	It is any real number that includes fractions. It can be accurate up to 15 decimal places.	17.990,-235567,0.00897
complex number	It is any number that includes both real and fictitious components.	2+4i

Sequence data types:

TYPE	EXAMPLE
List	“Hi”,’Gocourse
String	List1=[‘python’,’java’, ‘c’]
Tuple	Tuple1=(23.12,’Gocourse’,hi)

Dictionary data types:

Dictionary is a collection that can be changed in any order and is indexed. It prohibits duplicate members. Curly brackets are used in python dictionaries. It is made up of a group of key-value pairs. Those values and keys are distinguished by a colon (:). We can use the key name in the square brackets to get to the items in the dictionary.

Example:

```
mydictionary={                          #create Dictionary
   "websitename":"Gocourse",
   "domainname":"gocourse.in"
}
print(mydictionary)                     #print Dictionary
```

Program:

```
mydictionary={
   "websitename":"Gocourse",
   "domainname":"gocourse.in"
}
print(mydictionary)
```

```
A=mydictionary["domainname"]
print(A)
```

Output:

```
{'websitename': 'Gocourse', 'domainname': 'gocourse.in'}
gocourse.in
```

Boolean data type:

The Python type for this data type is bool, and it comes with true and false as its two default values. When determining whether a given statement is true or false, this data type is utilized.

Set data types:

A collection that is unordered and unindexed in Python is called a set. A built-in function set is used to create the set (). Due to its unique elements and iterable mutability (changeability), it prohibits duplicate members.

Example:

```
myset=set({2023,'Gocourse'})   #creating set
print(myset)                   #print set
```

Program:

```
myset=set({2023,'Gocourse'})
print(myset)
myset.add('education')
print(myset)
myset.remove('education')
print(myset)
```

Output:

```
{'Gocourse', 2023}
{'education', 'Gocourse', 2023}
{'Gocourse', 2023}
```

1.9 Type conversion

The process of converting one data type to another is known as type conversion. Type casting describes this transformation.

Syntax :

<required data type> (expression)

Example:

```
Float1= 38.0098  # float value
int1=int(float1)  #convert to integer
print(int1) #print
```

Program:

```
mynum=38.0098
print("This float number:",mynum)
myconvertnum=int(mynum)
print("Converted number is integer:",myconvertnum)
```

Output:

```
This float number: 38.0098
Converted number is integer: 38
```

REVIEW QUESTIONS

Short questions

1.Define variable
2.Define data type.
3.Write any 4 keywords in python?
4.Define keywords
5.Who developed python programming?

Long questions

1.Explain various types of data types.
2..Explain various types of Operators.

2

WRITING SIMPLE PROGRAM

2.1 -Write a program to sum any three integers

1.Write a program to sum any three integers.

Program:

```
a=10
b=20
c=30
d=a+b+c
print("sum of a,b and c:",d)
```

Output:

```
sum of a,b and c:60
```

2.2 -Write a program Add three integer to using input statement

2. Write a program Add three integer to using input statement

Program:

```
A=int(input("Enter A value"))
B=int(input("Enter B value"))
C=int(input("Enter C value"))
D=A+B+C
print("sum of A,B and C:",D)
```

Output:

```
Enter A value =10
Enter B value =20
Enter C value =30
sum of A,B and C:60
```

2.3 -Write program to print a String word "welcome to Gocourse" and also using input statement

3.Write program to print a String word "welcome to Gocourse" and also using input statement

Program:

```
intro=str(input("Enter given word: "))
print(intro)
```

Output:

```
Enter given word: welcome to Gocourse
welcome to Gocourse
```

3

CONTROL STRUCTURES

3.1 -Control Structures

Control structure is also known as control flow, simply refers to the order in which the instructions are executed in the program. Control statement is the statement which determines the flow of control in a program.

Types of control statements:

There are fundamentally three types of control statement in python .They are

1. Selection statement
2. Iterative statement
3. Sequential statement

1. Selection statement:-

Selection statement is a decision making in a programming language. It can allow you to run the particular block of the code on a decision made by verifying the given condition. Selection statement is also called a conditional statement.

Python supports three types of decision making statements. They are

1. if statement
2. else statement
3. Switch statement

2. Iterative statement:-

Iterative statements are used when some block of code is executed repeatedly until a given condition is satisfied.

Python supports 2 types of loops are,

1. For loop
2. While loop
3. Do while loop

3. Sequential statement:-

Sequential statement is also called a jump statement.

There are:

- break statement
- continue statement

3.2 -Selection control statement

If statement:

It is a selection statement, and a block of code is executed if the given condition is true.

Syntax:

if (condition):

Code to be executed

Program 1:

```
if(9<10):
   print("true")
```

Output:

true

Elif statement:

"Try this condition if the previous conditions were false" is the command that Python uses with the elif keyword.

Syntax:

```
if(condition):
  code to be executed
elif(condition):
  code to be executed
```

Program 2:

```
a=10
b=9
if(a<b):
    print("true")
elif(a>b):
    print("false")
```

Output:

false

Else statement:

A block of code to be executed can be specified with the else statement. when the condition is not true.

Syntax:

```
If(condition):

  Code to be executed

else:

  Code to be executed
```

Program:

```
subject1=int(input("Enter subject1 mark:"))
subject2=int(input("Enter subject2 mark:"))
total=subject1+subject2
percentage=total/2
if(percentage >90):
    print("Good mark & well tried")
else:
    print("better luck next time")
```

Output:

```
Enter subject1 mark:80
Enter subject2 mark:75
better luck next time
```

3.3 -For loop

In Python, the for loop is used for iterating over a sequence (list, tuple, string, dictionary, set, etc.).

Syntax:

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

Example 1:

1.Print each Mobile in a Mobile list

Program:

```
mobile=["vivo","oppo","nokia","poco"]
for x in mobile:
    print(x)
```

Output:

vivo
oppo
nokia
poco

Example 2:

2. For loop through a string

Program:

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

Output:

G
o
c
o
u
r
s
e

Example 3:

A number sequence is returned by the range() and range() functions. The value at the end is enclosed in range brackets and starts at 0 and is increased by 1

Program:

```
for x in range(10):
    print(x)
```

Output:

```
0
1
2
3
4
5
6
7
8
9
```

3.4 -While loop

The while loop through a block of code as long as a specified condition is true.

Syntax:

```
while(condition)

{

}
```

Example 1:

```
i = 0
while i < 10:
  print(i)
  i += 1
```

Output:

```
0
1
2
3
4
5
6
7
8
9
```

3.5 -Break and Continue statement

break statement:

The break statement transfers control to the statement immediately following the loop, which it is used to terminate.

Example for break statement in "for loop":

```
for x in "GOCOURSEwebsite":
 if x=="w":
   break
 print(x)
print("for loop Example for break statement")
```

Output:

G

O

C

O

U

R

S

E

Example for break statement in for loop

Example for break statement in "while loop":

```
i = 1
while i < 10:
  print(i)
  if (i == 5):
    break
  i += 1
```

Output:

1

2

3

4

5

Continue Statement:

In Python, the continue statement is used to skip over the current iteration of a loop (either ‘for’ or ‘while’ loop) and move on to the next iteration.

Example for continue statement in "for loop":

```
for x in "GOCOURSEwebsite":
 if x=="w":
   continue
 print(x)
print("Example for continue statement in for loop")
```

Output:

G

O

C

O

U

R

S

E

e

b

s

i

t

e

Example for continue statement in "while loop"

```
i = 0
while i < 10:
  i += 1
  if i == 5:
    continue
  print(i)
```

Output:

1

2

3

4

6

7

8

9

10

REVIEW QUESTIONS

Short questions

1.Define control structure
2.Define for loop with an example.
3.Write a program to while loop?

Long questions

1.Explain About control structure,selection statement,looping statement.
2..Explain break and continue statement.

4

LIST

4.1 list

A list is an ordered data structure with elements enclosed within square brackets and separated by commas.

Example:

```
Mylist=['Pythonbook', 'Javabook']
Print(Mylist)
```

Output:

['Pythonbook', 'Javabook']

4.2 List Traversal

List Traversal:

- List traversal is the sequential access to list elements.
- Index numbers, or List index, are used to access the list's elements.
- There are 2 types of indexing:
    - Positive indexing
    - Negative indexing

Positive indexing:

To get to the element from the beginning (from left to right), positive indexing is used. Index numbers begin with zero.

List index number	Mylist[0]	Mylist[1]	Mylist[2]	Mylist[3]	Mylist[4]
Element	Bike	Car	Bus	bicycle	tricycle

Example:

```
Mylist=['bike', 'car', 'bus','bicycle',’tricycle’]
Print(Mylist [0])
```

Print(Mylist [2])

Output:

bike
bus

Negative indexing:

To get to the element from the end (right to left), negative indexing is used.
-1 alludes first last thing, - 2 alludes second last thing, - 3 alludes third keep going, etc.

List index number	Mylist[-1]	Mylist[-2]	Mylist[-3]	Mylist[-4]	Mylist[-5]
Element	Python	Java	C++	Html	Css

Example:

```
Mylist=['Python','Java','C++','Html',’Css’]
Print (Mylist [-1])
Print (Mylist [-2])
Print (Mylist [-3])
Print (Mylist [-4])
Print (Mylist [-5])
```

Output:

Css
Html
C++
Java
Python

4.3 List Methods and Operations

Working with a list is made easier by the built-in methods that Python provides.
The dot operator is used to access list methods.

Syntax:

Listname.method()

method()	Description
append()	used to add element at the end of the list
clear()	Removes all the element in the list
copy()	Returns copy of the list
extend()	Adds element at the end of the list

insert()	Adds element at the specified index of the list
pop()	Remove element at the specified index
remove()	Remove the item with specified value
reverse()	Reverse the order of the list
sort()	Sort the list

Example:

```
mylist=['bike','car']
mylist.append('bus')
print(mylist)
```

Output:

['bike', 'car', 'bus']

Example:

```
mylist=['bike','car','bus']
mylist.pop(2)
print(mylist)
```

Output:

['bike', 'car']

4.4 List membership Test

Checking the membership test and iterating over the list are two other list operations. Using the keyword "in," we can verify that an element is present in the list. It returns a true or false boolean value.

Example: 1

```
Mylist=['apple', 'banana', 'guava','kiwi']
Print ('apple' in Mylist)
```

Output:

True

Example: 2

```
Mylist=['apple','banana','guava','kiwi']
Print ('orange' in Mylist)
```

Output:

False

4.5 Iterating over list

You can iterate through each list item using loops.

Example:

```
#remove duplicates number
number_list=[1,2,5,5,3,2,7,4,7,6]
new_list=[]
for i in sorted(number_list):
    if i not in new_list:
        new_list.append(i)
print(new_list)
```

Output:

```
[1, 2, 3, 4, 5, 6, 7]
```

REVIEW QUESTIONS

Short questions

1.Define list
2.Define method and write any 4 methods in list ?
3.Define membership test.

Long questions

1.Explain about list methods and operations.
2..Explain iterating over the list and write a program to remove duplicate numbers in the number list ?

5
FUNCTION

5.1 Function

- A function is a named part of a program with smaller size which is used to perform a specific task.
- It can be called any number of times.
- The Def keyword is used to define a function.
- Function block begins with the keyword Def followed by the function name and parentheses(()) ends with colon and followed by block of statements.

Creating a function

```
def my_function():
print("hello world")
```

Calling a function

```
def my_function():
print("hello world")

my_function()
```

PROGRAM ROUTINES:

In python,program routine are called as "function"
Function is majorly classified into two types:

- Built-in function
- User-defined function

Built-in Function:

Built-in functions are built into an application and can be used by end users.
Python has several built-in functions,such as:

- print()
- input()
- len()

- clear()
- copy()

User defined function:
Users can create their own function called user defined function.

ADDITIONALLY ON FUNCTIONS

- Value returning function
- non-value returning function

👉Generally, "return" statement is used values to the function call.

Value returning function:
In this type of function it return back some values to the function call.

Example:
```
def myfunction(number):
   return 10*number
print(myfunction(10))
print(myfunction(20))
print(myfunction(30))
print(myfunction(40))
print(myfunction(50))
```

Output:
100
200
300
400
500

Non value returning function:
This type of function does not return back any values to the function call.

Example:
```
def myfunction(number):
   print(5*number)
(myfunction(10))
(myfunction(20))
(myfunction(30))
(myfunction(40))
(myfunction(50))
```

Output:

50
100
150
200
250

5.2 Arguments

- Information can be passed into functions as "Arguments".
- Arguments are specified after the function name ,inside the parenthesis().
- Arguments are classified into two types:

    👉Actual argument
    👉Formal arguments

- Arguments are simply a variable or value inside the parenthesis.
- More than one argument can be specified inside the parenthesis by separating it with a comma.

ACTUAL ARGUMENTS

Arguments specified inside the parenthesis of the function call are called actual arguments.
Syntax:
Function_name(actual arguments)

FORMAL ARGUMENTS

Arguments specified inside the parenthesis of the function header are called formal arguments.
Syntax:
def function_name(formal arguments)

Example:

```
def personalinfo(name,age,hobby): #formal arguments
    print("name:",name)
    print("age:",age)
    print("hobby:",hobby)
personalinfo('sam',53,'tenis') #actual arguments
```

Output:

```
name: sam
age: 53
hobby: tenis
```

5.3 Variable scope

- Variables are just containers for storing data values.
- The location where we can find and access variables when required is called scope variable.
- Variable scope is a determination of the lifetime of the variable.
- Scope variables determine the accessibility and visibility of a variable.
- There are four types of scope variable available in python:

    👉 Local scope variable
    👉 Enclosing scope variable
    👉 Global scope variable

👉 Built-in scope variable

LOCAL SCOPE VARIABLE

A variable created within a function can only be used within that function and is part of its local scope.

Example:

```
def myfunction():
    myval=220
    print("Only within the declared functions can a local variable be accessed.",myval)
print("Using a local variable outside of the function is impossible.")
myfunction()
```

Output:

Using a local variable outside of the function is impossible.
Only within the declared functions can a local variable be accessed. 220

ENCLOSING SCOPE VARIABLE

For the enclosed scope, we need to define an outer function enclosing the inner function. In the enclosed scope ,a variable defined and used in the outer function can not be used directly in the inner function on the same name. If you define the same variable in the inner function which also exists in the outer function after the use of the variable, the python will generate an error message.

GLOBAL SCOPE VARIABLE

A global variable is one that is created in the main Python code and is part of the global scope.

Any global or local scope can be used to access global variables.

Example:

```
x = 200

def myfunc():
  print(x)

myfunc()

print(x)
```

Output:

200
200

BUILT-IN SCOPE VARIABLE

The built-in scope is the widest scope in python. All the reserved names defined in python built-in modules have a built-in scope. When the python doesn't find an identifier in its local, enclosing or global scope, it looks in the built-in scope to see if it's defined there. If it's not present in them,it produces an error message.

5.4 -Recursion

- Recursion can be defined as the process of defining something in terms of itself.
- It is a process in which a function calls itself directly or indirectly.
- A complicated function can be split down into smaller sub-problems utilizing recursion.
- Recursion functions are challenging to debug.
- A lot of memory and time is taken for expensive uses.

Example:

```
#factorial using recursion

print("program to calculate factorial of a number")

number=int(input("enter the number"))

def fact(number):

    if(number<1):

        print("enter only positive values")

        exit()

    elif(number==1):

        return number

    else:

        return number*fact(number-1)

print(fact(number))
```

Output:

```
program to calculate factorial of a number

enter the number5

120
```

REVIEW QUESTIONS

Short questions

1.Define Function
2.What are the types of arguments?

Long questions

1.Explain About Variable scope
2..Explain Recursion with an example program?

6

TURTLE GRAPHICS

6.1 Turtle graphics:

Python's popular turtle graphics feature lets you use a turtle cursor to draw pictures and shapes on the screen in a fun and easy way.

To execute a turtle program you follow five steps:

1. Import the turtle library
2. Create a drawing board(Window)
3. Create a turtle
4. Use turtle methods to perform some operation
5. Run turtle.done().

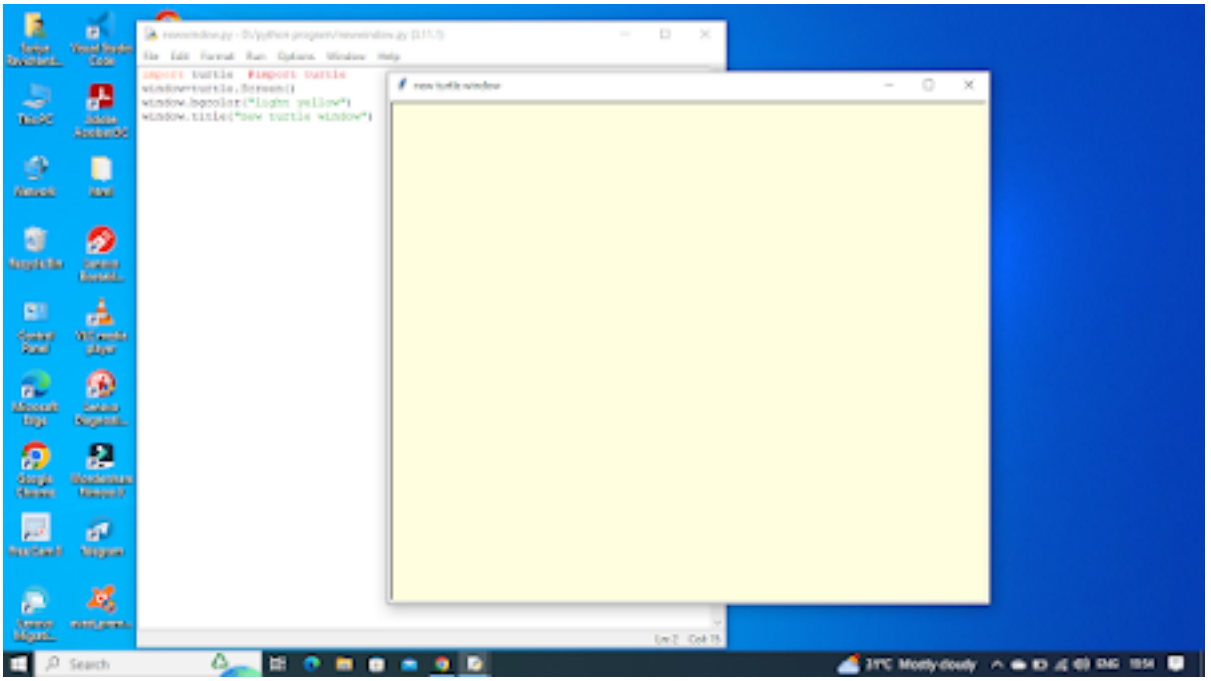
Example:

To create window and importing turtle

Program:

```
import turtle  #import turtle
window=turtle.Screen() #create window
window.bgcolor("light yellow")
window.title("new turtle window")
```

Output:



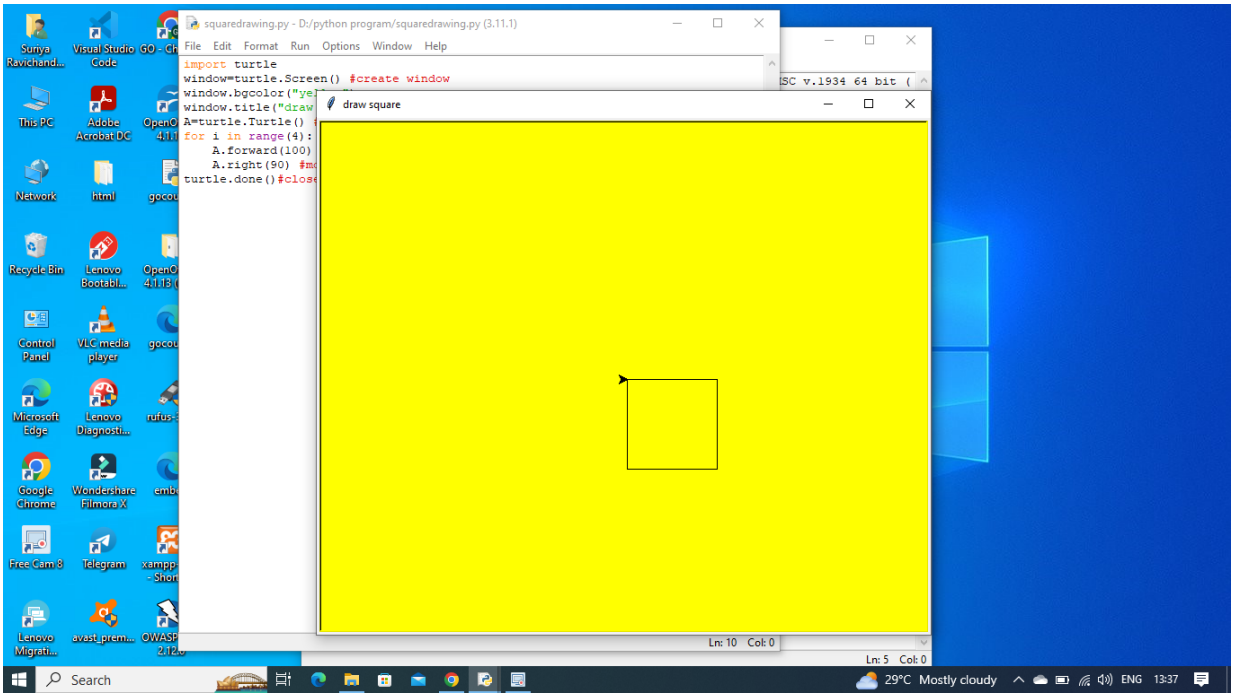
Example 2:

To Draw a square in turtle window

Program:

```
import turtle
window=turtle.Screen() #create window
window.bgcolor("yellow")
window.title("draw square")
A=turtle.Turtle() #create turtle
for i in range(4):
    A.forward(100) #move turtle forward
    A.right(90) #move turtle right
turtle.done()#close turtle
```

Output:



Example 3:

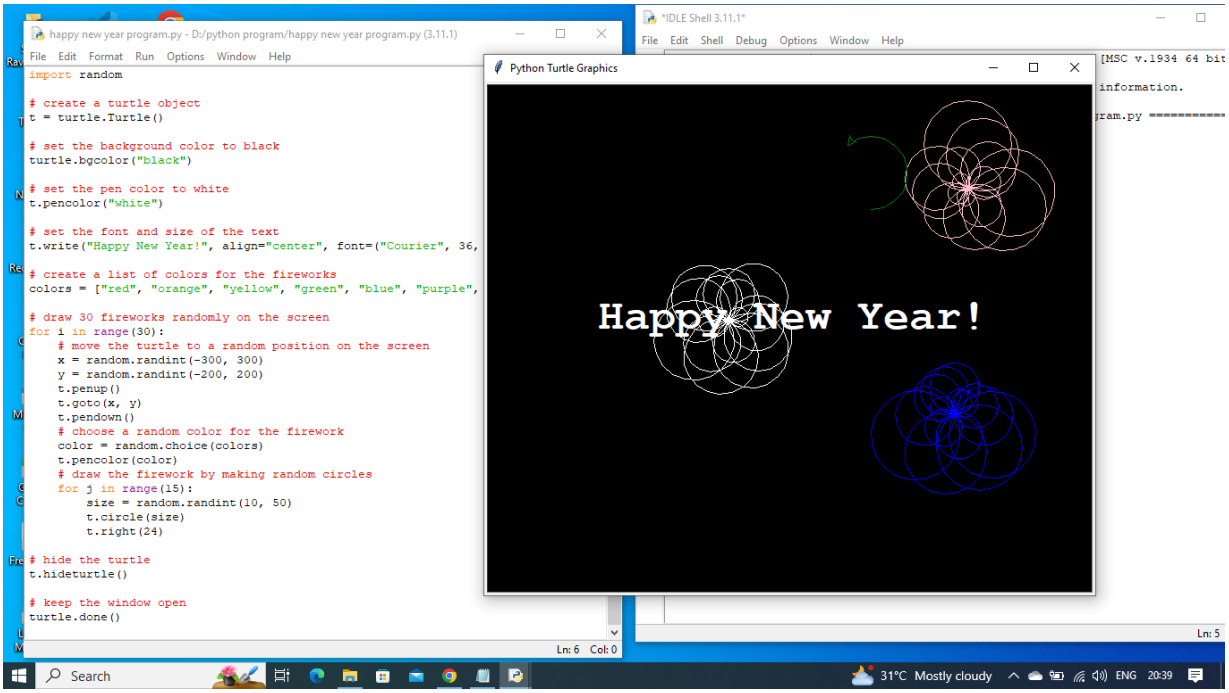
To create new year wishes in turtle

Program:

```
import turtle
import random
# create a turtle object
t = turtle.Turtle()
# set the background color to black
turtle.bgcolor("black")
# set the pen color to white
t.pencolor("white")
# set the font and size of the text
t.write("Happy New Year!", align="center", font=("Courier", 36, "bold"))
# create a list of colors for the fireworks
colors = ["red", "orange", "yellow", "green", "blue", "purple", "pink", "white"]
# draw 30 fireworks randomly on the screen
for i in range(30):
    # move the turtle to a random position on the screen
    x = random.randint(-300, 300)
    y = random.randint(-200, 200)
    t.penup()
    t.goto(x, y)
    t.pendown()
    # choose a random color for the firework
    color = random.choice(colors)
    t.pencolor(color)
    # draw the firework by making random circles
    for j in range(15):
        size = random.randint(10, 50)
        t.circle(size)
```

```
        t.right(24)
# hide the turtle
t.hideturtle()
# keep the window open
turtle.done()
```

Output:



6.2 Turtle attributes & methods

Python's turtle graphics give you a lot of options for controlling the turtle cursor and drawing various shapes and patterns on the screen.

Turtle attributes:

- position() -provides a tuple of coordinates as the current position of the turtle.
- heading() -returns the current heading angle of the turtle in degrees.
- color() -returns a tuple of RGB values representing the turtle's current color.
- pensize() -returns the current pen size of the turtle.
- speed() -returns the current speed of the turtle.

Turtle methods:

1. forward(distance): This option propels the turtle forward by a predetermined amount.
2. backward(distance): This option sends the turtle back the specified amount of time.

3. right(angle): This option moves the turtle to the right at the specified degree angle.
4. left(angle): Moves the turtle to the left at the specified degree angle.
5. penup() : raises the turtle's pen so that it can't draw while it moves.
6. pendown():puts the pen down on the turtle so it can draw while it moves.
7. setposition(x, y):sets the position of the turtle to the specified x and y coordinates.
8. setheading(angle) :sets the heading angle of the turtle to the specified degree.
9. dot(size, color) :draws a dot of the specified size and color at the turtle's current position.
10. circle(radius, extent):draws a circle with the radius and extent you specify (the percentage of the circle to be drawn).
11. begin_fill():begins to fill the space that the turtle's movements have enclosed.
12. end_fill():stops occupying the space that the turtle's movements have enclosed.

REVIEW QUESTIONS

Short questions

1.Define Turtle graphics.

Long questions

1.Explain turtle methods and attributes

7

MODULAR DESIGN

7.1Modular design:

In Python, modular design means breaking up a large program into smaller, independent modules that can be made, tested, and managed independently, making the program as a whole easier to manage and comprehend.

The idea of modules and packages can be used to put a modular design into action in Python. A package is a collection of modules arranged in a directory hierarchy, whereas a module is a single file containing Python definitions and statements.

Implementing modular design in Python:

Make a list of your program's various features and organize them into distinct modules.

Give each module a clear goal and a limited scope. It will be simpler to test and debug this way.

Make sure the purpose of your modules is clear by giving them meaningful names.

Instead of importing the entire package, import only the modules that are necessary for your program.

Before incorporating your modules into your main program, test them separately in the "main" block.

Type of modular design:

1,Functions: Functions can help you break up your code into smaller, easier-to-manage pieces of logic. Because functions can be called from other parts of your program, it is simple to reuse code and organize your code.

2.Modules: You can group functions, classes, and variables that are related together in a single file using modules. This makes it easier to reuse code across different parts of your program and helps keep your code organized.

3.Packages: Your code can be organized into modules organized in a hierarchical fashion using packages. When you need to meaningfully group related modules in large projects, this is helpful.

4.Classes: Classes permit you to characterize complex information structures and typify conduct inside them. This is helpful for making reusable parts that can be utilized all through your program.

5.Mixins: Mixins add functionality to a class without having to subclass it, allowing code to be reused across multiple classes. Code maintainability and duplication can both benefit from this.

Difference between inheritance and mixin in python

Inheritance	mixin
1.Inheritance is a way to create a new class that is a modified version of an existing class. The new class inherits all the properties and methods of the existing class, and can also have its own properties and methods. 2.This allows you to reuse code and extend functionality.	1.Mixins, on the other hand, are a way to add functionality to a class without creating a new inheritance hierarchy. A mixin is a class that defines some behavior, but is not meant to be instantiated on its own. 2. Instead, it is meant to be mixed in with other classes to provide additional functionality.

7.2 Python module:

A module is a file in Python that contains functions, definitions, and statements that can be imported into another Python script. A technique for organizing code into logical, reusable parts. You can break up your code into smaller, easier-to-manage files with modules, making it simpler to update, test, and debug your code.

To make a module in Python, you essentially have to characterize your capabilities, classes, and factors in a .py record.

To create a module:

Open a new file with a .py extension.
Define functions, classes, and variables in the file.
Save the file with a meaningful name (this will be the name of the module).

Example:

module file (example_module.py)

Program:

```
def greet(name):
    print(f"Hello, {name}!")

def add_numbers(a, b):
    return a + b

greeting = "Welcome to my module!"
```

To use a module:

Import the module into your Python script using the import keyword.
Access the functions, classes, and variables defined in the module using the module name and dot notation.

Example :

script (example_script.py)

Program:

```
import example_module

example_module.greet("Students")

result = example_module.add_numbers(3, 4)
print(result)

print(example_module.greeting)
```

Output:

```
Hello, Students
7
Welcome to my module!
```

REVIEW QUESTIONS

Short questions

1.Define modular design.

Long questions

1.Explain about python modules and give an example.
2.Explain the difference between inheritance and mixin.

8

FILES

8.1 FILE:

A file is a collection of characters or bytes represented using a code format (ASCII or UTF or Unicode).These files are called as text files such as notepad files,word files, programming language source programs, etc. Some files are stored as a sequence of bits. These types of files are called binary files such as image, video, audio, object, executable files.

Operations in files

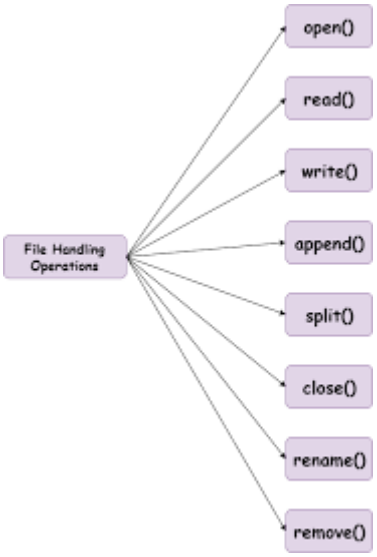


Fig:8.1 Operations in files

8.2 OPEN:

If we want to process a file for either read or write operation, the file must be opened first. The opening of a file is done with the help of open() method.

Function:

Open()

syntax:

file_object=open (file_name[,'mode'] [, buffering])

The open() method returns a file_object with allocated buffer size when the specified file is successfully opened on the specified mode.The file_object refers the allocated buffer for file processing operations.

CLOSE:

The close method used to clear the allocated buffer and unwritten information and release the file object. No more read and write operation can be done on the file object after that .

Function:

Close()

Syntax:

file_object.close()

The python produces an error message, if we try to close an unopened or unassigned file object.

Program to file open and close:

```
# Open a file in read mode
file = open('file.txt', 'r')

# Read the content of the file
content = file.read()

# Print the content of the file
print(content)

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

8.3 READ:

The read method is used to read the entire content or specified number of characters from the opening.

Keyword:

read ()

Syntax:

file_object.read([size])

The optional parameter 'size' specifies the number of characters to be read at a time. If we omit the parameter, the entire content of the file will be read.

Program:

```
# Open the file for reading
file = open("example.txt", "r")

# Read the contents of the file
contents = file.read()

# Print the contents to the console
print(contents)

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

8.4 WRITE:

The write method writes a specified text to the file. If we open the file in 'W' , it creates a new file on the name specified and returns the file object reference. If the specified file is already existing, the content will be erased.

Function:

Write()

Syntax:

file_object.write('string')

The write() method accepts a string as a parameter and writes the given string value into the specified file. After successful write operation is over, it returns the number of characters written into the file.

Program:

```
# Open the file for writing
```

```
file = open("example.txt", "w")

# Write some text to the file
file.write("Hello, world!\n")

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

REVIEW QUESTIONS

Short questions

1.Define files.

Long questions

1.Explain open, close and write .

9

STRINGS

9.1 STRING

- In computer programming, a string is a sequence of characters. The variable is initialized with the string Python Programming.
- The string is a collection of alphanumeric letters and special characters.
- A string is a collection of alphabets ('a' to 'z' and 'A' to 'Z'),numbers (0 to9),spaces and spe6 characters.
- string value is enclosed by single or double or triple quotes
- The flexibility of using different quotes is to use the other quotes as part of the string value.
- The main restriction is same quote should be used at the beginning and end of the string to denote the string value.

string usage:

Strings are used for storing text/characters.

For example:

"Hello World" is a string of characters.

Types of string:

The string data types are:

- CHAR
- VARBINARY
- BLOB
- TEXT
- SET
- ENUM

Creating Strings:

- A string in python can be created using single or double or even triple quotes
- String in single quotes cannot hold any other single quotes character in it, because the compiler will not recognise where to start and end the string.
- To overcome this problem you have to use double quotes.
- String which contains double quotes should be define within triple quotes
- Defining strings within triple quotes also allows creation of multiline strings.

Single quotes program:

```
# Define a string using single quotes
my_string = 'Jack'

# Print the string to the console
print(my_string)
```

Output:

Jack

Double quotes program:

```
# Prompt the user to enter a string
input_str = input("Enter your name: ")

# Print the string to the console
print("Your name is :", input_str)
```

Output:

Enter your name: Siva

Your name is :Siva

Triple quotes program:

```
# Define a multiline string using triple quotes
my_string = """My name is:Suriya."""

# Print the string to the console
print(my_string)
```

Output:

My name is:Suriya

STRING AND STRING ARRAY

- String is a data type in python ,which is used to handle arrays of characters.

- String is a sequence of Unicode characters that may be a combination of letters, numbers, or special symbols enclosed within single ,double or even triple quotes.
- strings are immutable ,it means,once you define a string,it cannot be changed during execution.
- String Array can be defined as the capacity of a variable to contain more than one string value at the same time, which can be called and accessed at any time in the program during the execution process.
- A String Array is an Array of a fixed number of String values. A String is a sequence of characters. Generally, which means the value of the string can not be changed.
- The String Array works similarly to other data types of Array. In Array, only a fixed set of elements can be stored.
- The key difference between Array and String is that an Array is a data structure.

Program:

```
# Create an array of strings
string_array = ["apple", "banana", "cherry", "elderberry"]

# Print each string in the array to the console
for string in string_array:
    print(string)
```

Output:

```
apple
banana
cherry
elderberry
```

9.2 STRING PROCESSING

String processing is a very broad topic, but in Python, there are many built-in methods and functions that make it easy to work with strings.

PYTHON STRING METHODS:

- String Slicing
- String Striping
- String Splitting
- String Reversing
- String length
- Uppercase and lowercase
- Replacing Substring

- Palindrome check

String slicing:

In Python, the process of extracting a portion of a string by specifying a range of indices is known as string slicing.

Program:

```
# Define a string variable

string = "Hello, world!"

# Extract the first five characters of the string

substring = string[0:5]

print("Substring:", substring)

# Extract the last six characters of the string

substring = string[-6:]

print("Substring:", substring)

# Extract every other character of the string

substring = string[::2]

print("Substring:", substring)
```

Output:

```
Substring: Hello

Substring: world!

Substring: Hlo ol!
```

String stripping:

In Python, the process of stripping a string of its leading and/or trailing characters is known as string stripping.

Program:

```
# Define a string variable with leading/trailing whitespace

string = "    Hello, world!    "

# Strip leading/trailing whitespace from the string

stripped_string = string.strip()

# Print the stripped string

print("Stripped string:", stripped_string)
```

Output:

Stripped string: Hello, world!

String Splitting:

The split() method in Python allows you to divide a string into a list of substrings. The split() method splits the original string wherever it finds a separator, taking as its argument a separator string.

Program 1:

```
string = "Hello World"

words = string.split(" ")

print(words)
```

Output:

['Hello', 'World']

Program 2:

```
string = "apple,banana,orange"

fruits = string.split(",")

print(fruits)
```

Output:

['apple', 'banana', 'orange']

String Reversing:

There is no built-in function for reversing the given string in Python. By cutting that number of steps backward, -1, it is done.

Program:

```
string = "Hello World"

reversed_string = string[::-1]

print(reversed_string)
```

Output:

```
dlroW olleH
```

String Length:

The length of the given string can be determined using the len() function.

Program:

```
# string_length.py

string = "Hello World"

length = len(string)

print(length)
```

Output:

```
11
```

Uppercase and Lowercase:

The upper() function encases the given string in upper case, and the lower() function encases it in lower case.

Program:

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

# convert to uppercase and print
```

```
uppercase_string = string.upper()

print("Uppercase string: ", uppercase_string)

# convert to lowercase and print

lowercase_string = string.lower()

print("Lowercase string: ", lowercase_string)
```

Output:

```
Enter a string: Hello World

Uppercase string:  HELLO WORLD

Lowercase string:  hello world
```

Replacing substring:

Using the replace() method, you can replace a substring within a string in Python. Two arguments are accepted by the replace() method: the replacement substring and the substring that will be replaced.

Program:

```
string = "Hello World"

new_string = string.replace("World", "Python")

print(new_string)
```

Output:

```
Hello Python
```

Palindrome check:

By comparing a string to its opposite in Python, you can determine whether or not it is a palindrome. A word, phrase, number, or other sequence of characters that reads the same way forward and backward is known as a palindrome.

Program:

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

reverse_string = string[::-1]
```

```
if string == reverse_string:
    print(string, "is a palindrome")
else:
    print(string, "is not a palindrome")
```

Output 1:

Enter a string: racecar

racecar is a palindrome

Output 2:

Enter a string: python

python is not a palindrome

REVIEW QUESTIONS

Short questions

1.What is uppercase and lowercase?
2.what is palindrome check?
3.what is string?
4.what are the types of strings?

Long questions

1.Explain about string and String processing give some examples .

10

EXCEPTION HANDLING

10.1 Exception handling:

- In Python, an exception is an error that occurs during the execution of a program. When an exception occurs, Python stops the normal flow of execution and jumps to the nearest exception handler. If no exception handler is found, the program terminates and prints an error message.

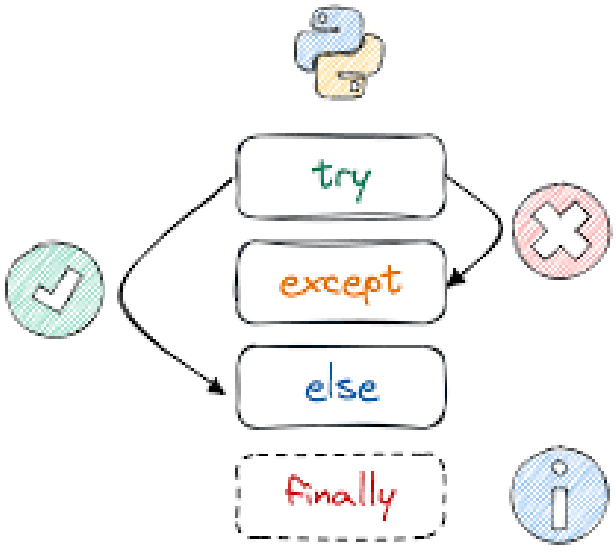


Fig 10.1 Exception handling mechanism

Exception handling step:

1. Use a try block to enclose the code that might raise an exception.
2. Use one or more except blocks to handle the specific exception(s) that might be raised.

3. Optionally, use a finally block to execute code that should be run whether or not an exception was raised.
4. Optionally, raise a new exception using the raise keyword if the situation warrants it.

Structure of exception handling:

```
try:
    # Code that might raise an exception
    ...
except ExceptionType1:
    # Code to handle ExceptionType1
    ...
except ExceptionType2:
   # Code to handle ExceptionType2
    ...
except:
    # Code to handle all other exceptions
    ...
finally:
    # Code to be executed regardless of whether an exception was raised or not
```

Example program:

```
try:
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))
    result = num1 / num2
   print("The result is:", result)
except ValueError:
   print("Please enter a valid integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
except Exception as e:
    print("An error occurred:", e)
finally:
    print("Program completed.")
```

Output :

#ZeroDivisionError

Enter a number: 10
Enter another number: 0
Cannot divide by zero.
Program completed.
#Value error
Enter a number: abc
Please enter a valid integer.
Program completed.
#correct output
Enter a number: 10
Enter another number: 2
The result is: 5.0
Program completed.

REVIEW QUESTIONS

Long questions

1.Write about Exception handling in python and give examples ?

11

DICTIONARY

11.1 Dictionary

- Dictionary is a collection of key values used to store data values.
- It is an unordered data structure stored as a key and value pair.
- It is a built-in type denoted by dict.
- It is a mutable data type.
- Dictionary is enclosed by curly braces{}
- Key and value pair are separated by comma.(,)
- The separator colon(:)is used between key and the value.
- The key used to map the value.
- Dictionary cannot have two items with the same key.

Methods in dictionary:

Clear() - It removes all the elements from the dictionary.

Copy() - Returns a copy of the dictionary.

Get() - It returns the values of the specified key.

Items() - It returns a list containing a tuple for each key value pair.

Keys()-Returns a list containing a dictionary of keys.

Pop() - Remove the elements with specified keys.

Popitem() - Remove the last inserted key-value pair.

Update() - Updates the dictionary with specified key-value pairs.

Values() - Returns a list of all the values.

Create a dictionary:

curly braces{} can be used to create a dictionary in Python.

Syntax:

```
# create an empty dictionary

my_dict = {}
```

Example:

Dictionary program to accessing elements , adding elements and removing elements in python

Program:

```
# create a dictionary

my_dict = {'apple': 2, 'banana': 1, 'orange': 3}


# accessing elements in the dictionary

print(my_dict['apple'])   # output: 2

print(my_dict.get('banana'))  # output: 1

print(my_dict.get('grape'))   # output: None


# adding elements to the dictionary

my_dict['grape'] = 4

print(my_dict)   # output: {'apple': 2, 'banana': 1, 'orange': 3, 'grape': 4}


# updating an element in the dictionary

my_dict['orange'] = 5

print(my_dict)   # output: {'apple': 2, 'banana': 1, 'orange': 5, 'grape': 4}


# removing an element from the dictionary

del my_dict['banana']

print(my_dict)   # output: {'apple': 2, 'orange': 5, 'grape': 4}
```

Output:

2

1

None

{'apple': 2, 'banana': 1, 'orange': 3, 'grape': 4}

{'apple': 2, 'banana': 1, 'orange': 5, 'grape': 4}

{'apple': 2, 'orange': 5, 'grape': 4}

REVIEW QUESTIONS

Long questions

1.Explain about the Dictionary and give examples ?

12
SETS

12.1 SETS

In python,a set is a type of collection in data type. A set is a mutable (changeable),unordered data. It doesn't allow duplicate values. Sets are useful in doing set operation.The provided with operations like ,intersection,union, difference and symmetric. Items in set are enclosed by curly brackets { }. Each item is separated by comma (,).

Syntax:

Identifier = { item1, item2, item3}

CREATING A SETS

- A set is created by placing all the elements separated by comma within a pair of curly brackets.
- Since sets are unordered and unindexed ,you cannot be sure in which order element appears.
- The items can be integer ,float, string or tuple.
- It allows the heterogeneous (different type) type of values in one set.
- It automatically eliminates the duplicate items.
- The set( ) function can also be used to create sets in python.

Syntax:

Set_Variable = {E1 , E2 , E3 .....En}

SET METHODS IN PYTHON

The in-built methods of set are used to:

- create
- insert
- remove

- delete the set
- set items.

Example:

Using all method program in set

Program:

```
# create an empty set

my_set = set()

# add items to the set

my_set.add('apple')

my_set.add('banana')

my_set.add('orange')

# print the set

print('Set after adding items:', my_set)

# remove an item from the set

my_set.remove('banana')

# print the set

print('Set after removing an item:', my_set)

# delete the entire set

del my_set

# try to print the set (will result in a NameError)

print(my_set)
```

Output:

```
Set after adding items: {'orange', 'banana', 'apple'}

Set after removing an item: {'orange', 'apple'}
```

```
Traceback (most recent call last):
  File "set_program.py", line 16, in <module>
    print(my_set)
NameError: name 'my_set' is not defined
```

SET OPERATIONS

- The set class supports mathematical set operations like **union,intersection,difference** and **symmetric** .
- These operations can be performed either using in-built functions or using operators.
- The operators work only on set data structure.
- The function works on any iterable data structure like list and tuple.

1.Union Operation

Union operation combines the elements of sets single list after eliminating the repeated items .The union operation can be done using single pipe operation (|).

Syntax:

```
NewSet = Set1 | Set2  | Set3.
```

Example:

```
# define sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}
# perform the union operation using the | operator
union_set = set1 | set2
print(union_set)
```

Output:

```
{1, 2, 3, 4, 5, 6, 7}
```

2.Union Function

The method union() is used to do union operation.

Syntax:

```
NewSer = Set.union( Set2 ,Set3...)
```

Example:

```
# define two sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}
```

```
# perform the union operation using the union() method
union_set = set1.union(set2)
print(union_set)
```

Output:

{1, 2, 3, 4, 5, 6, 7}

3.Intersection Operation

The intersection operation is to get the elements commonly available in two sets The intersection operation can be done using the logical 'and' (&) operator.

Syntax:

```
NewSet = Set1 & Set2
```

Example:

```
# define two sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

# perform the intersection operation using the & operator
intersection_set = set1 & set2
print(intersection_set)
```

Output:

{3, 4, 5}

4.Intersection Function

The function intersection () of set class is used to identify the common elements among the given sets The function accepts more than one parameter.The parameter of the function can be or list or tuple.

Syntax:

```
NewSet = Set1 intersect(Set2 , Set3 , Set4...)
```

Example:

```
# define two sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

# perform the intersection operation using the intersection() method
intersection_set = set1.intersection(set2)
print(intersection_set)
```

Output:

{3, 4, 5}

5.Difference Operations

The difference operation gets the set of elements only in the first set, not in the second set. This operation can be performed using the minus(-) operator.

Syntax:

NewSer = Set1 - Set2

Example:

```
# define two sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

# perform the difference operation using the - operator
diff_set = set1 - set2
print(diff_set)
```

Output:

{1, 2}

6.DIfference Function

The method difference () gets the set of elements only in the invoking set but not in sets passed as parameters.

Syntax:

NewSet = Set1 difference (Set2 , Set3 , Set4 ...)

Example:

```
# define two sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

# perform the difference operation using the difference() method
diff_set = set1.difference(set2)
print(diff_set)
```

Output:

{1, 2}

7.Symmetric Difference Operation

The symmetric difference operation is to get the elements of two sets after excluding the common elements .This operation can be performed using the exponential operator.

Syntax:

newset= set1 ^ set2

Example:

```
# define two sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}
# perform the symmetric difference operation using the ^ operator
sym_diff_set = set1 ^ set2
print(sym_diff_set)
```

Output:

{1, 2, 6, 7}

REVIEW QUESTIONS

Short questions

1.Define sets.

Long questions

1.Explain the set method and write an example program .
2.Explain set operation and types.

13

OOPS CONCEPTS

13.1 OOPS concepts in python

In Python, object-oriented Programming (OOPs) is programming that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

Who is the father of OOPs?

Alan Kay coined the term “object oriented programming” in 1966 or 1967.

What is the first OOPS language?

Simula (1967) is generally accepted as being the first language with the primary features of an object-oriented language.

What are the popular OOPS languages?

1. C++
2. PYTHON
3. RUBY
4. JAVA.

Why is OOPs important?

Benefits of OOPs:

The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost.OOP language allows to break the program into the bit-sized problems that can be solved easily (one object at a time).

What is OOPs example?

An Object is one of the Java OOPs concepts which contains both the data and the function, which operates on the data. For example – chair, bike, marker, pen, table, car, etc.

Main Concepts of Object-Oriented Programming (OOPs)

Fig:13.1 Object-Oriented Programming (OOPs)

13.2 Class and Objects

1. CLASS

In python a class is a type of object oriented programming. It is a container for dta and functions. A Class is like an object constructor, or a "blueprint" for creating objects. To create a class, use the keyword class. It is a Collection of objects called classes. It is a logical entity.

Syntax: Defining Class

```
class Classname:
#statement1
......
......
#statement n
```

2. OBJECT

Object is an instance of a class. you can create multiple objects with the same behavior, instead of repeating. Object is also a copy of the class

Example 1:

1.write a program with class and object python.

Program:

```
class bike:
  name="hero"
  model=2022
  def function1(self):
    print("bike name and model")
    print(self.name)
    print(self.model)
obj=bike()
obj.function1()
```

Output:

```
bike name and model
hero
2022
```

13.3 Encapsulation

The practice of keeping fields within a class private, then providing access to those fields via public methods. Encapsulation is a protective barrier that keeps the data and code safe within the class itself. We can then reuse objects like code components or variables without allowing open access to the data system-wide.Binding (or wrapping) code and data together into a single unit are known as encapsulation.

Program:

```
class company:
   def __init__(self, name, salary, working_days):
      self.name = name
      self.salary = salary
      self.working_days = working_days
   def demofunc(self):
      print("worker name "+self.name)
      print("salary amount Rs",+self.salary)
      print("working days",+self.working_days)
c1 = company("Steve",10000, 30)
c2 = company("Chris",9000, 10)
c3 = company("Mark",20000, 65)
c4 = company("Kate",6000, 5)
c1.demofunc()
c2.demofunc()
```

```
c3.demofunc()
c4.demofunc()
```

Output:

```
worker name Steve
salary amount Rs 10000
working days 30
worker name Chris
salary amount Rs 9000
working days 10
worker name Mark
salary amount Rs 20000
working days 65
worker name Kate
salary amount Rs 6000
working days 5
```

13.4 Polymorphism

It is the feature of the 'one name many forms' method that can perform with different tasks.Polymorphism allows us to characterize methods in the child class that have similar names as the strategies in the parent class. One form of polymorphism is method polymorphism. That’s when the code itself implies different meanings. The other form is method overriding.

The types of polymorphism are given as

- Compile time polymorphism
- Run-time polymorphism

Program:

```
class bikename:
  name="hero"
  def function1(self):
    print("bikename class executed :")
    print(self.name)
class bikemodel(bikename):
     model=2022
     def function2(self1):
         print("bikemodel class executed:")
         print(self1.model)
obj=bikename()
obj1=bikemodel()
obj.function1()
obj1.function2()
```

Output:

bikename class executed
hero
bikemodel class executed
2022

13.5 INHERITANCE

A special feature of Object-Oriented Programming in Java, Inheritance lets programmers create new classes that share some of the attributes of existing classes. Using Inheritance lets us build on previous work without reinventing the wheel.It is the process of creating a new class from existing class. Base class is parent class and child class is sub class.

The types of inheritance are:

- Single Inheritance
- Multi Level Inheritance
- Hierarchical Inheritance
- multiple Inheritance

Single inheritance:

By allowing a derived class to inherit properties from a single parent class, single inheritance makes it possible to reuse code and add new features to code that already exists.

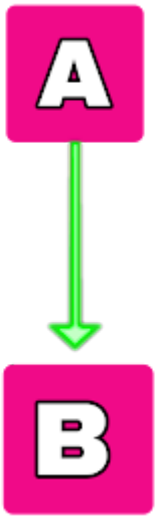


Fig:13.2 Single Inheritance

Example:

single inheritance

```
# Base class
class Parent:
    def func1(self):
        print("This is parent class.")

# Derived class
class Child(Parent):
    def func2(self):
        print("This is child class.")

# object creating
obj = Child()
obj.func1()
obj.func2()
```

Output:

```
This is parent class.
This is child class.
```

Multiple inheritance:

Multiple inheritances refer to the situation in which a class can be derived from more than one base class. In multiple inheritances, the derived class inherits all base class characteristics.

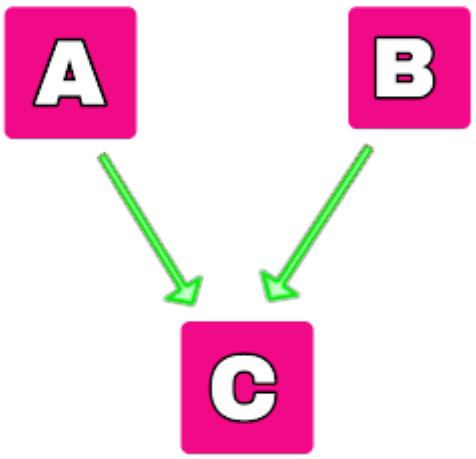


Fig:13.3 Multiple Inheritance

Example:

```
# multiple inheritance

# Base class1
class student:
    Studentname = ""
    def student(self):
        print(self.studentname)

# Base class2
class department:
    departmentname = ""

    def department(self):
        print(self.departmentname)

# Derived class
class college(student, department):
    def student(self):
        print("department :", self.departmentname)
        print("student :", self.studentname)

# object creating
s1 = college()
s1.departmentname = "computer science"
s1.studentname = "Ram"
s1.student()
```

Output:

```
department : computer science
student : Ram
```

Multilevel inheritance:

Features of the base class and the derived class are further inherited into the new derived class in multilevel inheritance. A relationship depicting a child and a grandfather is similar to the following figure.

Fig:13.4 Multilevel Inheritance

Example:

```
# multilevel inheritance
# Base class
class name:

    def __init__(self, name1):
        self.name1 = name1

# Intermediate class
class Hobby(name):
    def __init__(self, Hobby1, name1):
        self.Hobby1 = Hobby1

        # invoking constructor of name class
        name.__init__(self, name1)

# Derived class
class age(Hobby):
    def __init__(self, age1, Hobby1, name1):
        self.age1 = age1

        # invoking constructor of Hobby class
        Hobby.__init__(self, Hobby1, name1)

    def print_name(self):
        print(' name :', self.name1)
        print("Hobby :", self.Hobby1)
```

```
        print("age :", self.age1)

#  Driver code
s1 = age(20, 'hockey', 'Ram')
s1.print_name()
```

Output:

```
name : Ram
Hobby : hockey
age : 20
```

Hierarchical Inheritance:

Hierarchical inheritance refers to the process by which a single base is used to create multiple derived classes. There are two derived classes and a parent class in this program.

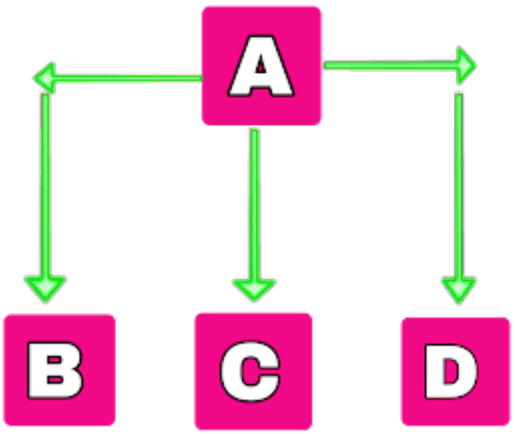


Fig:13.5 Hierarchical Inheritance

Example:

```
# Hierarchical inheritance
# Base class
class Parent:
    def func1(self):
        print("best program language")

# Derived class1
class Child1(Parent):
    def func2(self):
```

```
        print("python")

# Derived class2
class Child2(Parent):
    def func3(self):
        print("java")

# object creating
obj1 = Child1()
obj2 = Child2()
obj1.func1()
obj1.func2()
obj2.func1()
obj2.func3()
```

Output:

```
best program language
python
best program language
java
```

Hybrid inheritance:

Hybrid inheritance is a type of inheritance that combines two or more types of inheritance, such as multiple inheritance and hierarchical inheritance, in a single class hierarchy. In Python, you can implement hybrid inheritance by using a combination of multiple inheritance and hierarchical inheritance.

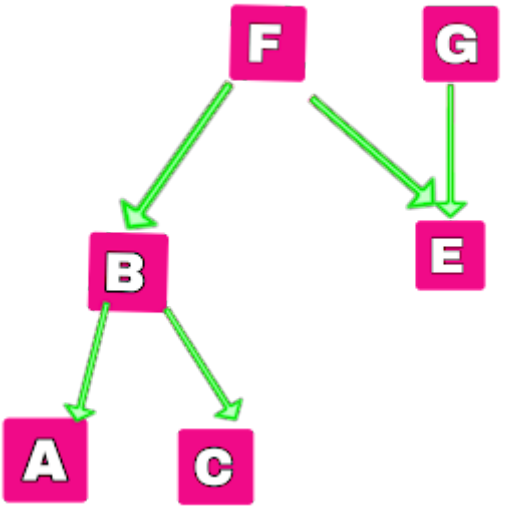


Fig:13.6 Hybrid Inheritance

Example:

```
# hybrid inheritance
class website:
    def func1(self):
        print("Gocourse pvt ltd.")

class msg1(website):
    def func2(self):
        print("Best online tutorial platform. ")

class msg2(website):
    def func3(self):
        print("Easy to learn.")


class msg3(msg1, website):
    def func4(self):
        print("Become a certified programmer in python.")

# object creating
object = msg3()
object.func1()
object.func2()
object.func3()
object.func4()
```

Output:

```
Gocourse pvt ltd.
Best online tutorial platform.
Easy to learn
Become a certified programmer in python
```

13.6 DATA ABSTRACTION

In Python, Data Abstraction hiding internal details and showing functionality is known as Data Abstraction. Data abstraction or Information hiding refers to providing only essential information to the outside world and hiding their background details. A class that comprises at least one abstract method is known as the abstraction class. Abstraction class can be acquired by the subclass and the unique method gets its definition in the subclass.

An abstraction class can be helpful when we are planning huge capacities. An abstraction class is additionally useful to give the standard interface to various executions of parts.

Example:

```
class BankAccount:
    def __init__(self):
        self.__balance = 0
    def deposit(self, amount):
        self.__balance += amount
    def withdraw(self, amount):
        if amount > self.__balance:
            print("Insufficient balance")
        else:
            self.__balance -= amount
    def get_balance(self):
        return self.__balance
# Create an object of the BankAccount class
my_account = BankAccount()
# Deposit some money into the account
my_account.deposit(1000)
# Try to withdraw more money than is in the account
my_account.withdraw(2000)
# Withdraw some money from the account
my_account.withdraw(500)
```

```
# Get the current balance of the account

balance = my_account.get_balance()

# Print the current balance

print("Current balance:", balance)
```

Output:

```
Insufficient balance
Current balance: 500
```

IMPORTANT QUESTION

Short Answer questions

1.Define OOPs concepts.

Long Answer questions

1.Explain about OOPs concepts and types.
2.Explain Inheritances and type.

LAB PROGRAMS

1.Temperature conversion

To develop a python program to convert the given temperature from fahrenheit to celsius or vice-versa depending on user choice

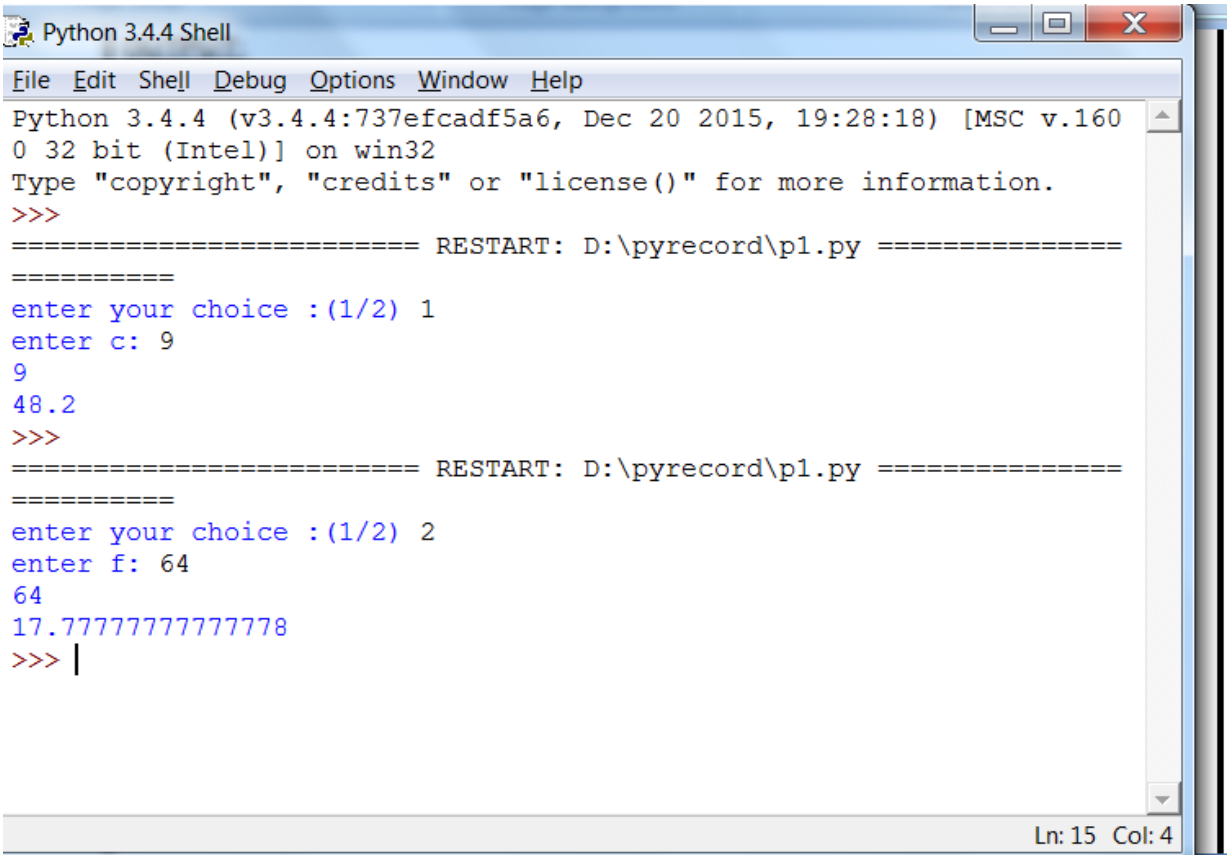
ALGORITHM:

Step 1: Start
Step 2: Read the input choice
Step 3: if choice is “1”, celsius is converted into fahrenheit using formula f = 9*c / 5+32 Step 4: if choice is “2”, fahrenheit is converted into celsius using formula c=(f-32)*5/9 Step 5: print the result
Step6: Stop

Program:

```
choice=int(input("Enter your choice: 1 or 2"))
if (choice==1):
    c=int(input("Enter the celsius value:"))
    print(c)
    f=9*c/5+32
    print(f)
elif(choice==2):
    f=int(input("Enter the fahrenheit values:"))
    print(f)
    c=(f-3)*5/9
    print(c)
else:
    print("Enter the correct choice!")
```

Output:


Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.160
0 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\pyrecord\p1.py ==========
==========
enter your choice :(1/2) 1
enter c: 9
9
48.2
>>>
========== RESTART: D:\pyrecord\p1.py ==========
==========
enter your choice :(1/2) 2
enter f: 64
64
17.77777777777778
>>>
Ln: 15 Col: 4

2.Preparation of student mark list

To develop a program to calculate the total mark, percentage and grade of the student.

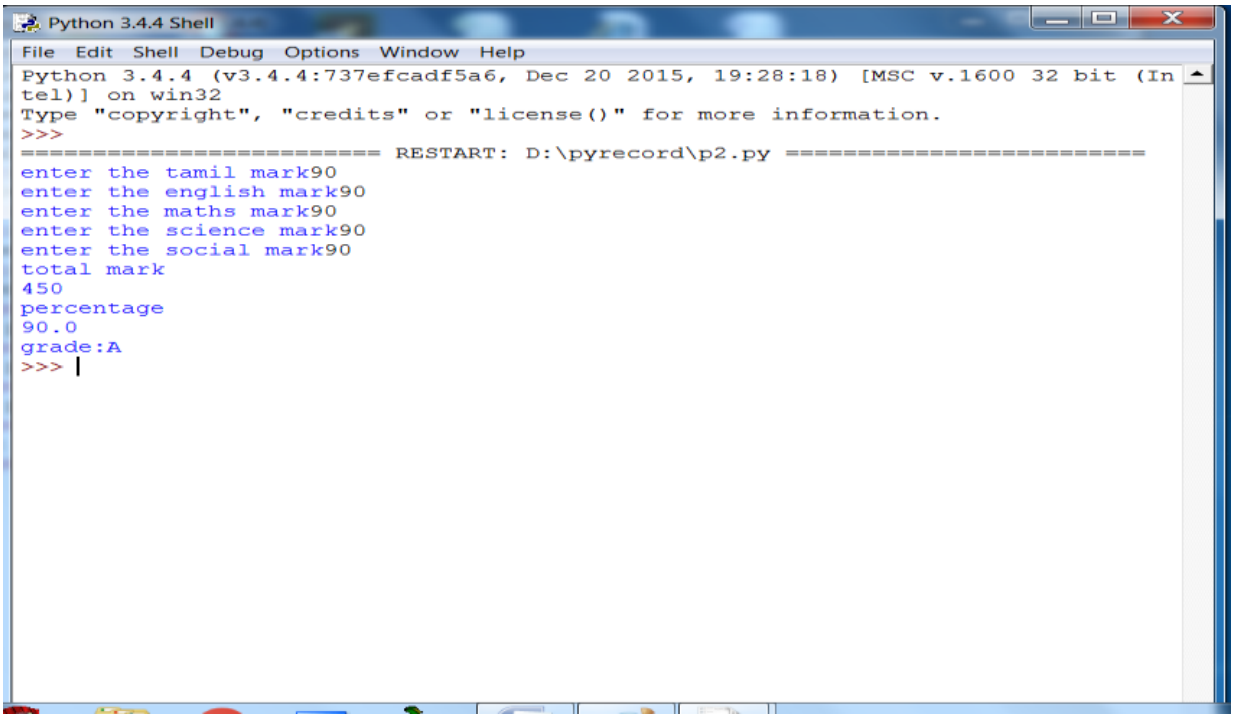
ALGORITHM:

Step 1: Start
Step 2: Get the five subject marks as input
Step 3: calculate the total, percentage and grade
Step 4: print the total, percentage and grade
Step 5: stop

PROGRAM:

```
tamil=int(input("enter the tamil mark"))
english=int(input("enter the english mark"))
maths=int(input("enter the maths mark"))
science=int(input("enter the science mark"))
social=int(input("enter the social mark"))
print("total mark")
total=tamil+english+maths+science+social
print(total)
print("percentage")
percentage=total/5
print(percentage)
if percentage >=80:
   print("grade:A")
elif percentage >=70:
   print("grade:B")
elif percentage >=60:
   print("grade:C")
elif percentage >=40:
   print("grade:D")
else:
   print("grade:E")
```

OUTPUT:


Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\pyrecord\p2.py ==========
enter the tamil mark90
enter the english mark90
enter the maths mark90
enter the science mark90
enter the social mark90
total mark
450
percentage
90.0
grade:A
>>>

3.Find the area of circle,square,rectangle and triangle

To develop a program to find the area of circle, square, rectangle and triangle by accepting input parameters from the user.

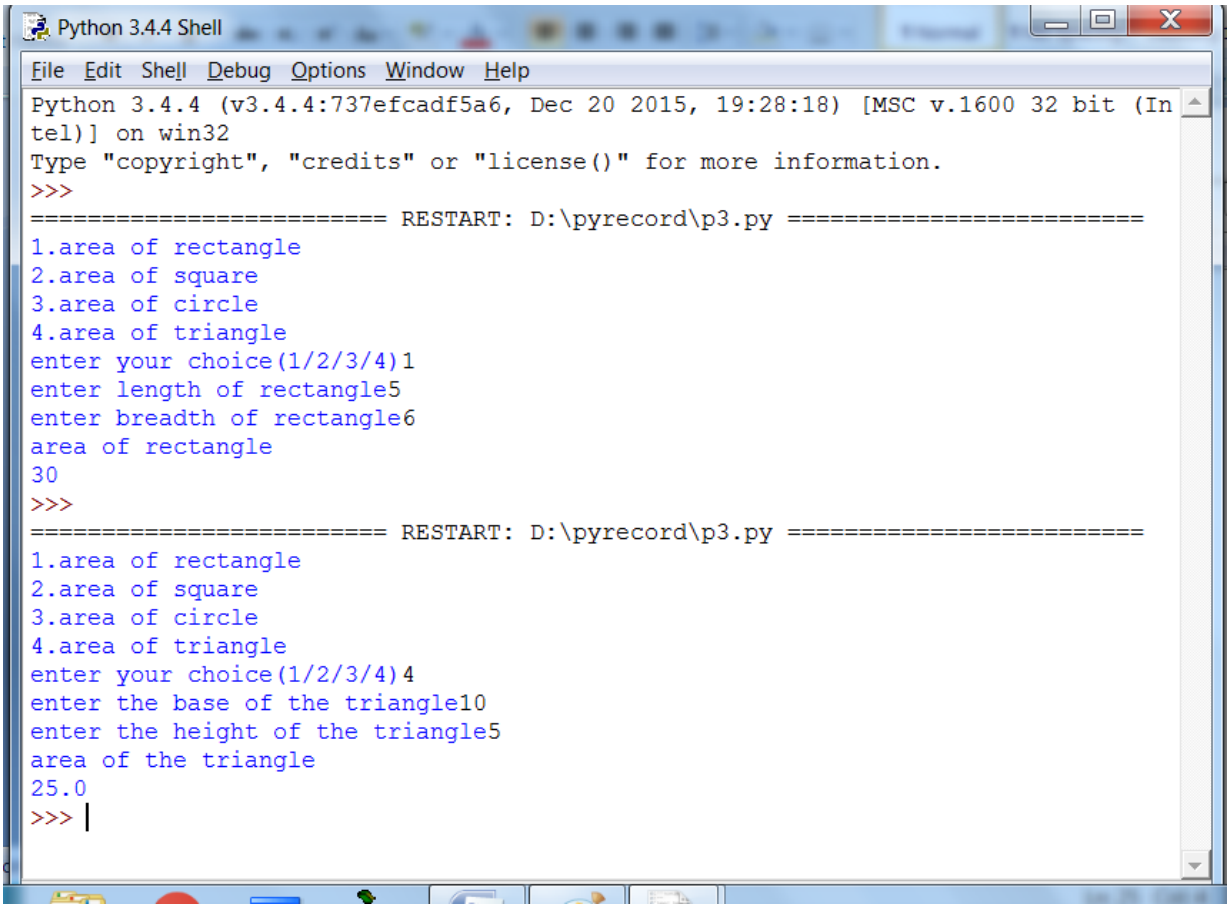
ALGORITHM:

Step 1: start
Step 2: Get the input values
Step 3: As per user choice, calculate the area of the rectangle, circle, square and triangle.
Step 4: Use the appropriate formulae:
rectangle= l * b
circle= 3.14 * r * r
square=side * side
Triangle =½ *base*height
Step 5: print the result
Step 6: stop

PROGRAM:

```
print("1. area of rectangle")
print("2. area of square")
print("3. area of circle")
print("4. area of triangle")
choice=int(input("enter your choice(1/2/3/4)"))
if(choice==1):
   l=int(input("enter length of rectangle"))
   b=int(input("enter breadth of rectangle"))
   area=l*b
   print("area of rectangle")
   print(area)
elif(choice==2):
   s=int(input("enter side of the square"))
   area=s*s
   print("area of the square")
   print(area)
elif(choice==3):
   r=int(input("enter the circle"))
   area=3.14*r*r
   print("area of the circle")
   print(area)
elif(choice==4):
    r=int(input("enter the base of the triangle"))
    h=int(input("enter the height of the triangle"))
    area=1/2*r*h
   print("area of the triangle")
   print(area)
```

OUTPUT:


Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\pyrecord\p3.py ==========
1.area of rectangle
2.area of square
3.area of circle
4.area of triangle
enter your choice(1/2/3/4)1
enter length of rectangle5
enter breadth of rectangle6
area of rectangle
30
>>>
========== RESTART: D:\pyrecord\p3.py ==========
1.area of rectangle
2.area of square
3.area of circle
4.area of triangle
enter your choice(1/2/3/4)4
enter the base of the triangle10
enter the height of the triangle5
area of the triangle
25.0
>>>

4.Fibonacci series

To develop a program to display the first n terms of fibonacci series

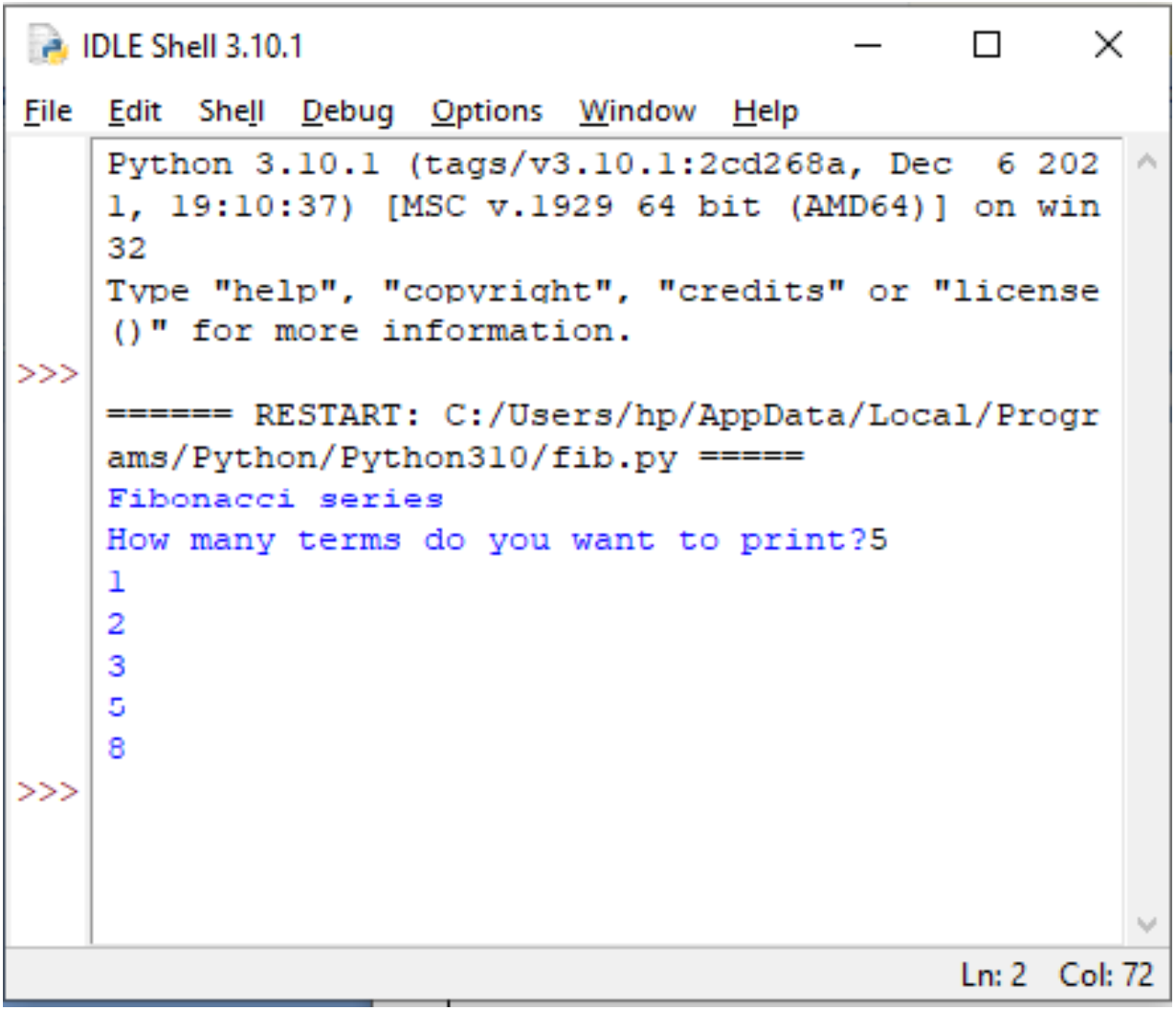
ALGORITHM:

Step 1: Start
Step 2: Get the length of series as input
Step 3: Calculate the fibonacci value upto length
Step 4: Print the result
Step 5: Stop

PROGRAM:

```
print("Fibonacci series")
num1=0
num2=1
n=int(input("How many terms do you want to print?"))
count=1
while (count<=n):
   num3=num1+num2
   print(num3)
   num1=num2
   num2=num3
   count=count+1
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/fib.py =====
Fibonacci series
How many terms do you want to print?5
1
2
3
5
8
>>>
Ln: 2 Col: 72

5.Factorial using recursion

Program to find factorial of the given number using recursive function

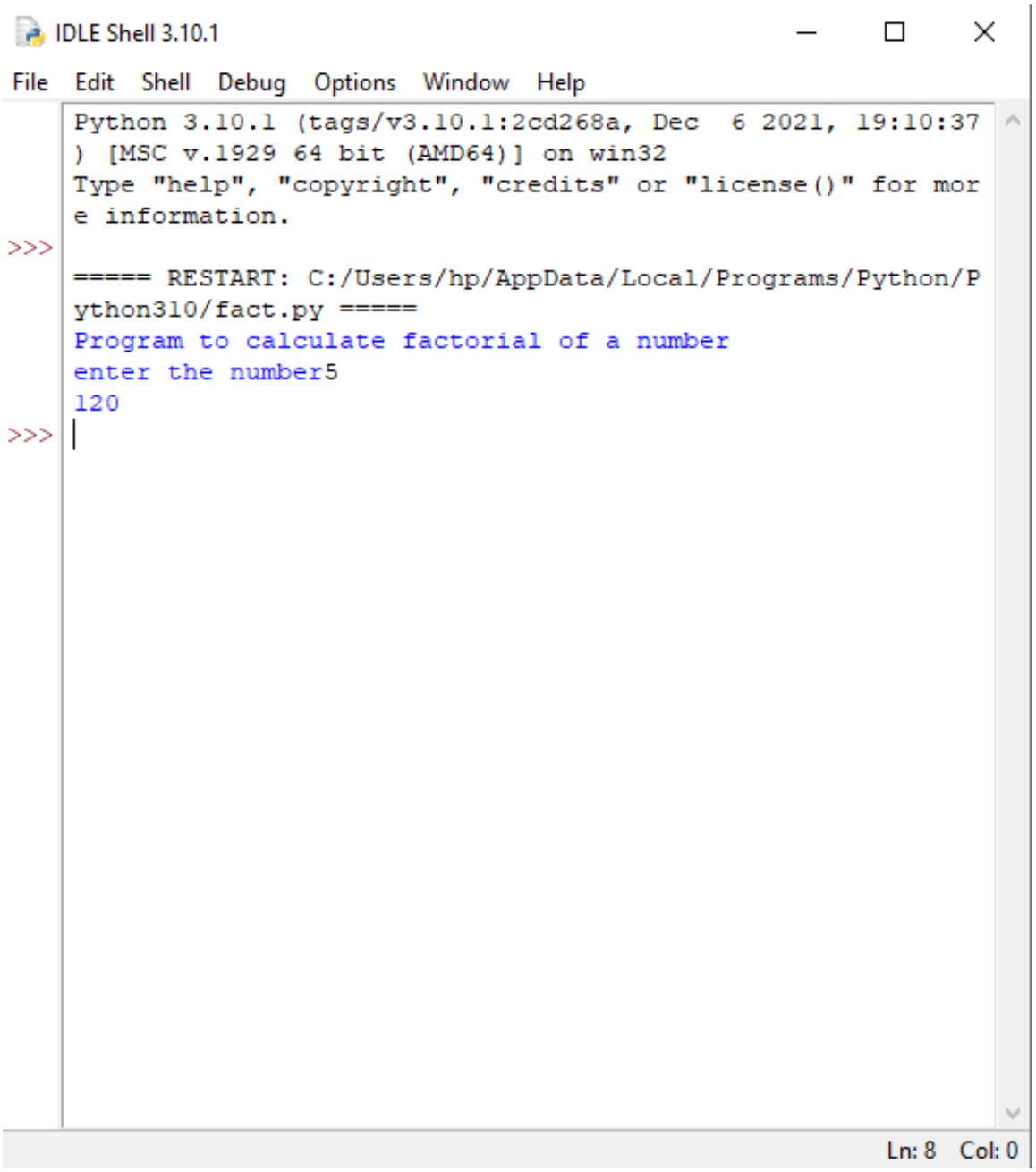
ALGORITHM:

Step 1: Start
Step 2: Get the number to calculate factorial
Step 3: Calculate the factorial value using recursion function
Step 4: Print the result
Step 5: Stop

PROGRAM:

```
print("Program to calculate factorial of a number")
number=int(input("enter the number"))
def fact (number):
  if(number <1):
    print("Enter only positive values")
    exit()
  elif (number==1):
    return number
  else:
    return number*fact(number-1)
    print(fact(number))
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37
) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for mor
e information.
>>>
===== RESTART: C:/Users/hp/AppData/Local/Programs/Python/P
ython310/fact.py =====
Program to calculate factorial of a number
enter the number5
120
>>>
Ln: 8 Col: 0

6.Counting the even and odd numbers in an array

To develop the program to count the number of even and odd number from array of n numbers

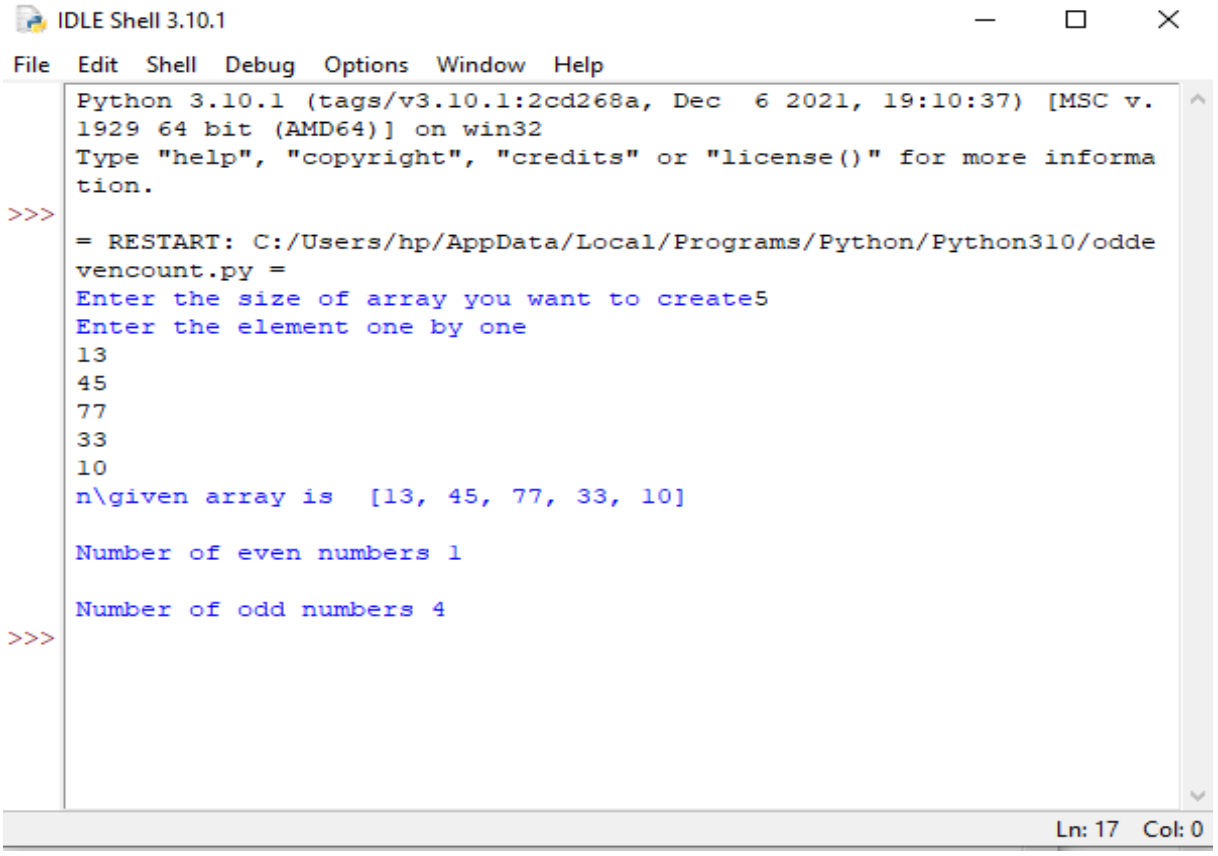
ALGORITHM:

Step 1: Start
Step 2: Get the input values
Step 3: Calculate the number of even number and odd number
Step 4: Print the odd number count and even number count
Step 5: Stop

PROGRAM:

```
arraysize=int(input("Enter the size of array you want to create"))
myarray=[]
print("Enter the element one by one ")
for i in range (0,arraysize):
    x =int(input())
    myarray.append(x)
print("n\given array is ",myarray)
evencount=0
oddcount=0
for i in myarray:
  if((i%2)==0):
    evencount=evencount+1
  else:
      oddcount=oddcount+1
print("\nNumber of even numbers",evencount)
print("\nNumber of odd numbers",oddcount)
```

OUTPUT


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.
1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
= RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/odde
vencount.py =
Enter the size of array you want to create5
Enter the element one by one
13
45
77
33
10
n\given array is [13, 45, 77, 33, 10]
Number of even numbers 1
Number of odd numbers 4
>>>
Ln: 17 Col: 0

7.Counting the uppercase and lowercase letters

To develop a program to find uppercase and lowercase letter in the given string

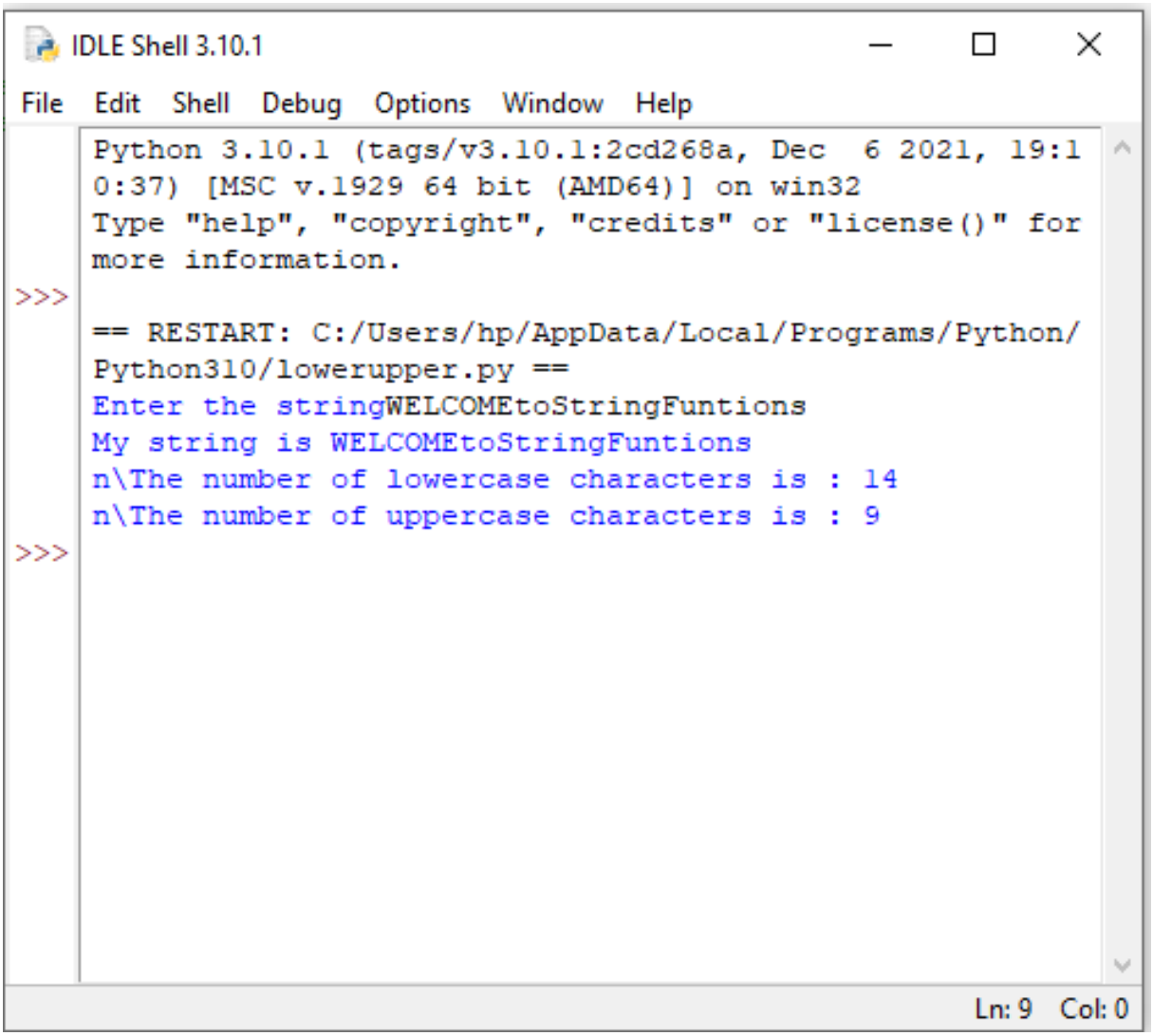
ALGORITHM:

Step 1:Start
Step 2:Get the input string
Step 3:Calculate the uppercase and lowercase letter using is upper and islower function
Step 4:Print the total uppercase and lowercase letter
Step 5:Stop

PROGRAM:

```
mystring=input("Enter the string")
print("My string is",mystring)
upper=0
lower=0
for i in mystring:
  if(i.isupper()):
     upper=upper+1
  elif(i.islower()):
     lower=lower+1
print("n\The number of lowercase characters is :",lower)
print("n\The number of uppercase characters is :",upper)
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/lowerupper.py ==
Enter the stringWELCOMEtoStringFuntions
My string is WELCOMEtoStringFuntions
n\The number of lowercase characters is : 14
n\The number of uppercase characters is : 9
>>>
Ln: 9 Col: 0

8.Palindrome checking

Program find to the given string is palindrome or not

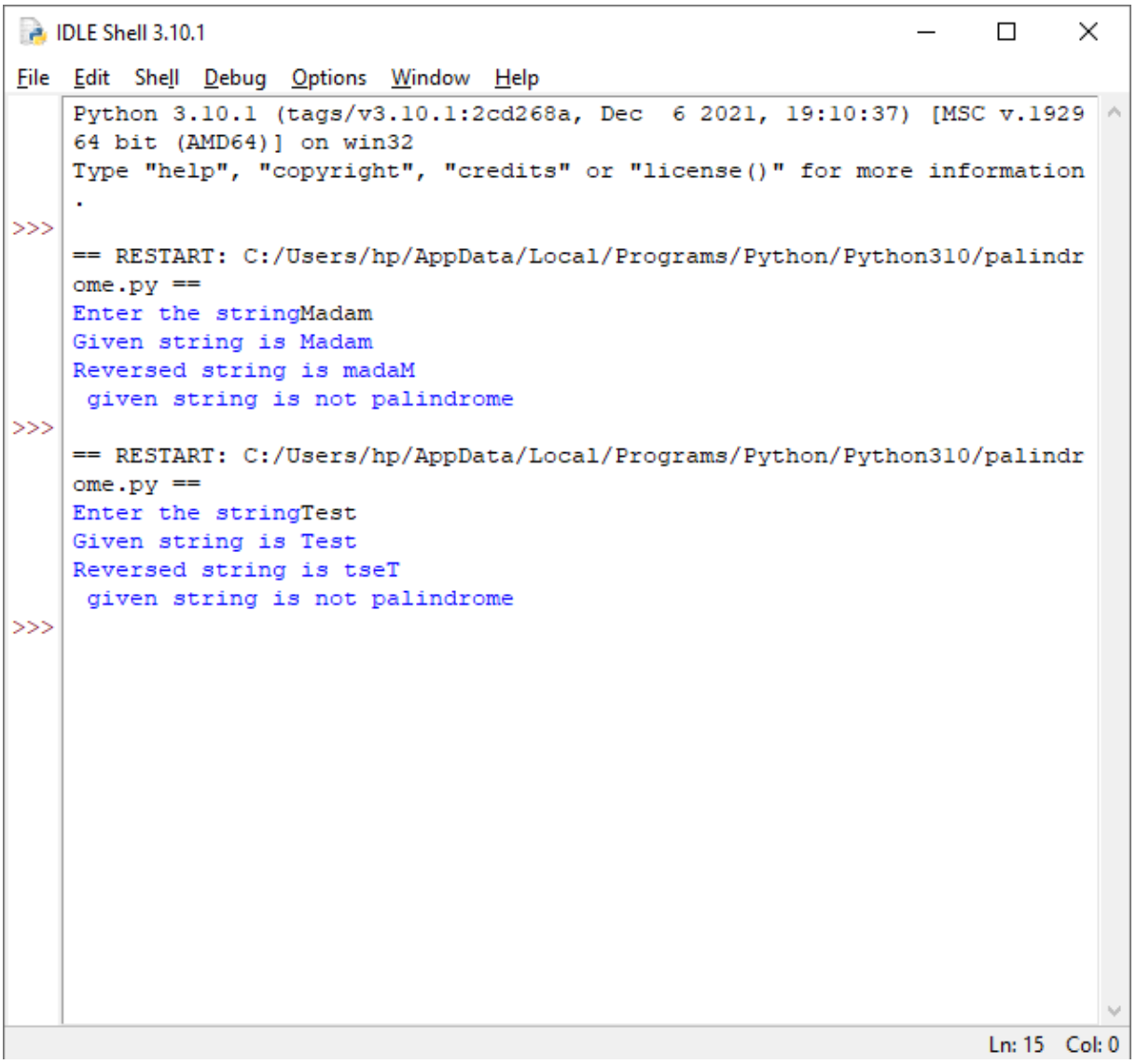
ALGORITHM:

Step 1: Start
Step 2: Get the string as input
Step 3: Reverse the given string using reverse () function & compare with original string
Step 4: If both are equal then it is palindrome otherwise not palindrome
Step 5:Stop

PROGRAM:

```
str=input("Enter the string")
print("Given string is",str)
strreverse=str[::-1]
print("Reversed string is",strreverse)
if(str==strreverse):
   print(" given string is palindrome")
else:
   print(" given string is not palindrome")
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information
.
>>>
== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/palindr
ome.py ==
Enter the stringMadam
Given string is Madam
Reversed string is madaM
given string is not palindrome
>>>
== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/palindr
ome.py ==
Enter the stringTest
Given string is Test
Reversed string is tseT
given string is not palindrome
>>>
Ln: 15 Col: 0

9.Sum of all items in a Dictionary

To Develop a program to find the sum of all values in a Dictionary

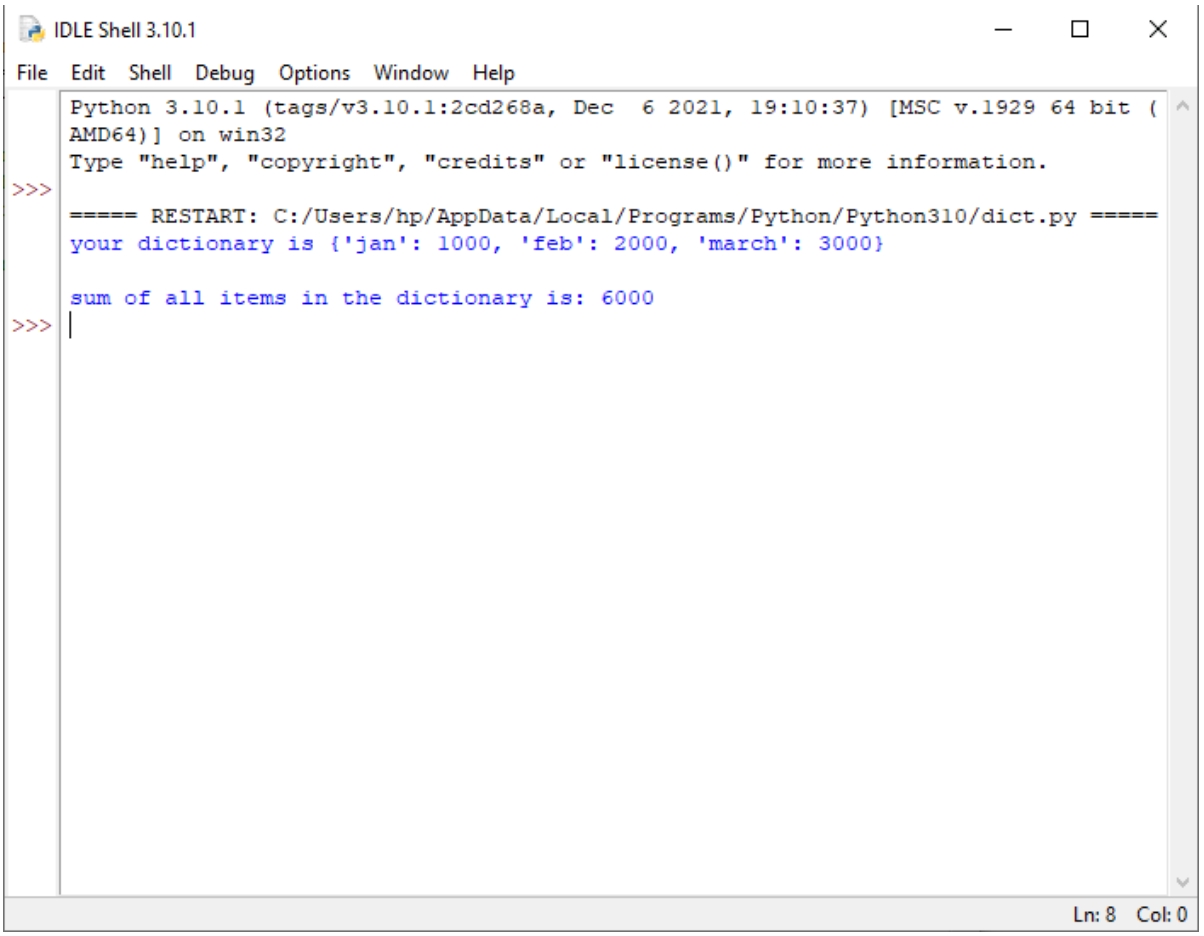
ALGORITHM:

Step 1: Start
Step 2: Get the input values for dictionary
Step 3: Add all the values is dictionary using for loop
Step 4: Print the result
Step 5: Stop

PROGRAM:

```
def sum(mydict):
  tot=0
  for i in mydict:
    tot=tot+mydict[i]
return tot
mydict={'jan':1000,
    'feb':2000,
    'march':3000}
print("your dictionary is",mydict)
print("\nsum of all items in the dictionary is:",sum(mydict))
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/dict.py =====
your dictionary is {'jan': 1000, 'feb': 2000, 'march': 3000}
sum of all items in the dictionary is: 6000
>>>
Ln: 8 Col: 0

10.Patten construction

To develop a program to construct a pattern using a nested loop

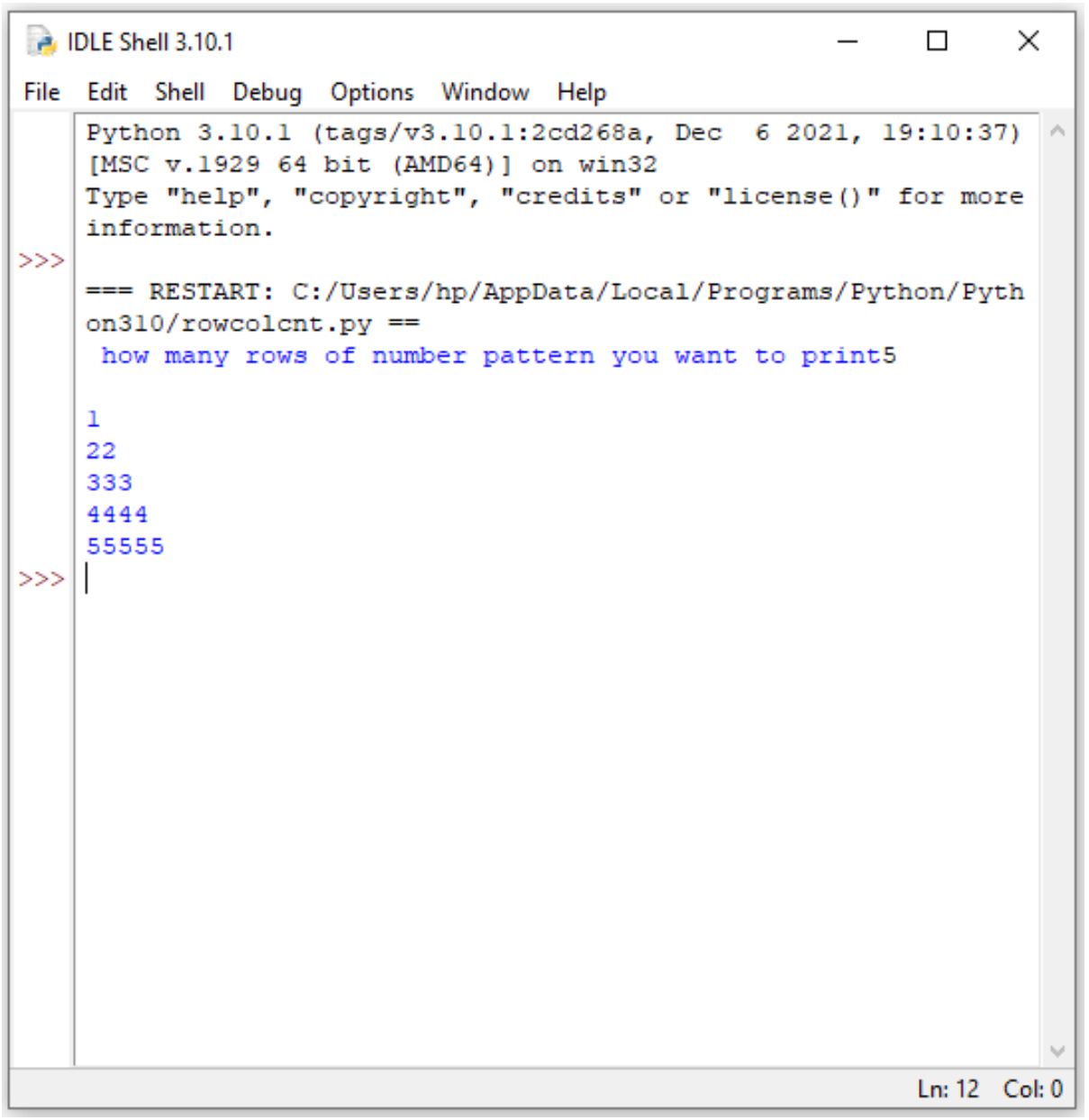
ALGORITHM:

Step 1: Start
Step 2:Get the input of row of pattern
Step 3: Use nested loop for row and columns to print the values
Step 4:Stop

PROGRAM:

```
count=int(input(" how many rows of number pattern you want to print"))
for i in range(count): #outer loop, indicates row count
    for j in range(i): #inner loop, indicates column count
       print(i,end="")
 print(" ")
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/rowcolcnt.py ==
how many rows of number pattern you want to print5
1
22
333
4444
55555
>>>
Ln: 12 Col: 0

11.Copy the file contents

To develop a program to read a file content and copy only the contents at odd line into a new file.

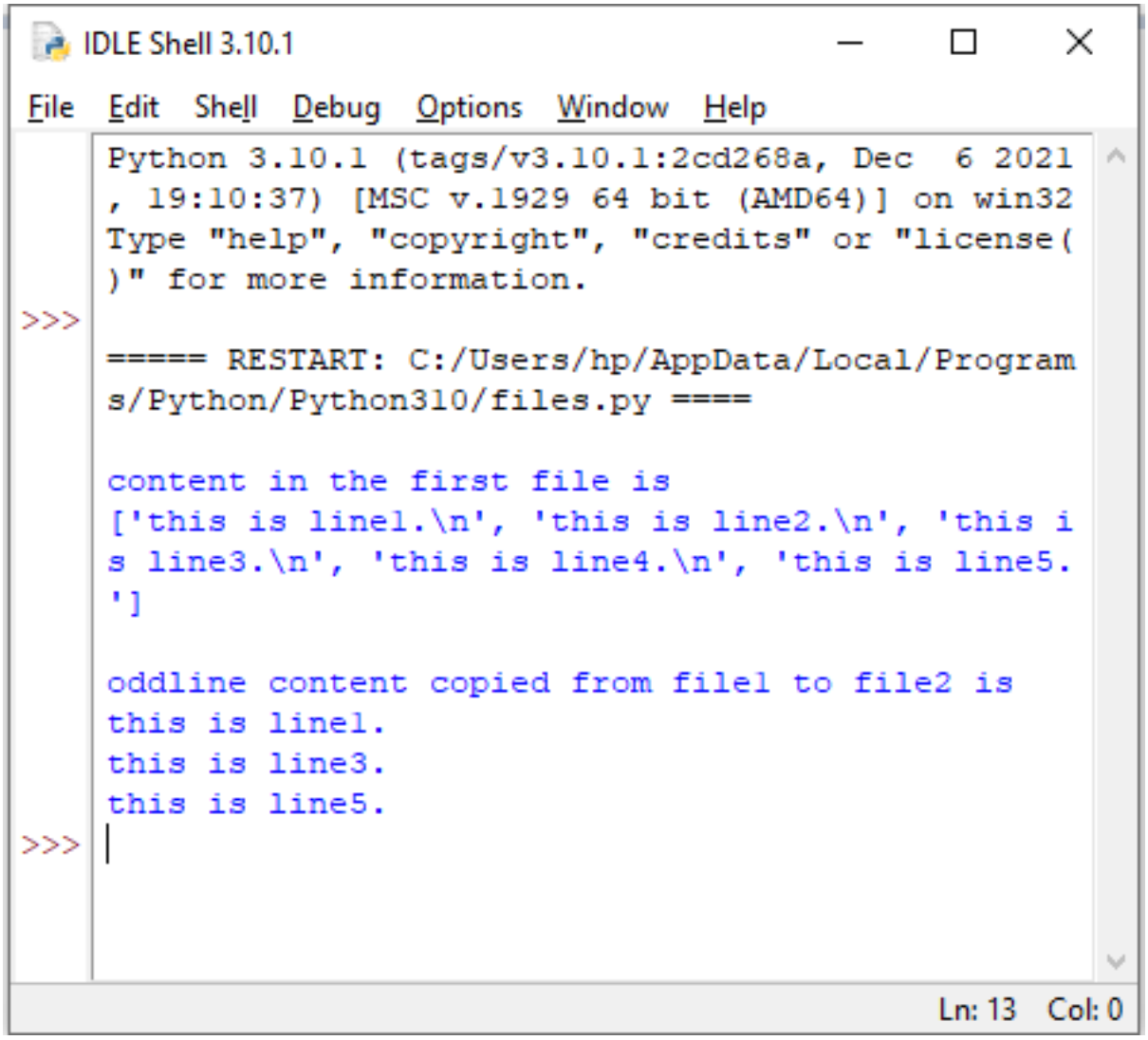
ALGORITHM:

Step 1: Start
Step 2: Create two text files “file1” and “file2” in read and write modes
Step 3: Read contents from “file1”
Step 4: copy the “file1”contents to “ file2
Step 5: open “file2” in readmode
Step 6: read and print “file2” contents
Step 7: Close the “file1” and “file2”
Step 8: Stop

PROGRAM:

```
#open a file named file1.txt in read mode
file1=open('D:\\file1.txt','r')
#open a file named file2.txt in write mode
file2=open('D:\\file2.txt','w')
#read the contents of file line by line
content1=file1.readlines()
print("\ncontent in the first file is")
print(content1)
for i in range(0,len(content1)+1):
    if(i%2 != 0):
       file2.write(content1[i-1])
    else:
       pass
# open file in read mode
file2=open('D:\\file2.txt','r')
#read file contents
content2=file2.read()
print("\noddline content copied from file1 to file2 is ")
print(content2)
#close the files
file1.close()
file2.close()
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021
, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license(
)" for more information.
>>>
===== RESTART: C:/Users/hp/AppData/Local/Program
s/Python/Python310/files.py ====
content in the first file is
['this is line1.\n', 'this is line2.\n', 'this i
s line3.\n', 'this is line4.\n', 'this is line5.
']
oddline content copied from file1 to file2 is
this is line1.
this is line3.
this is line5.
>>>
Ln: 13 Col: 0

12.Turtle graphics window creation

To develop a program to create a turtle graphics window with specific size

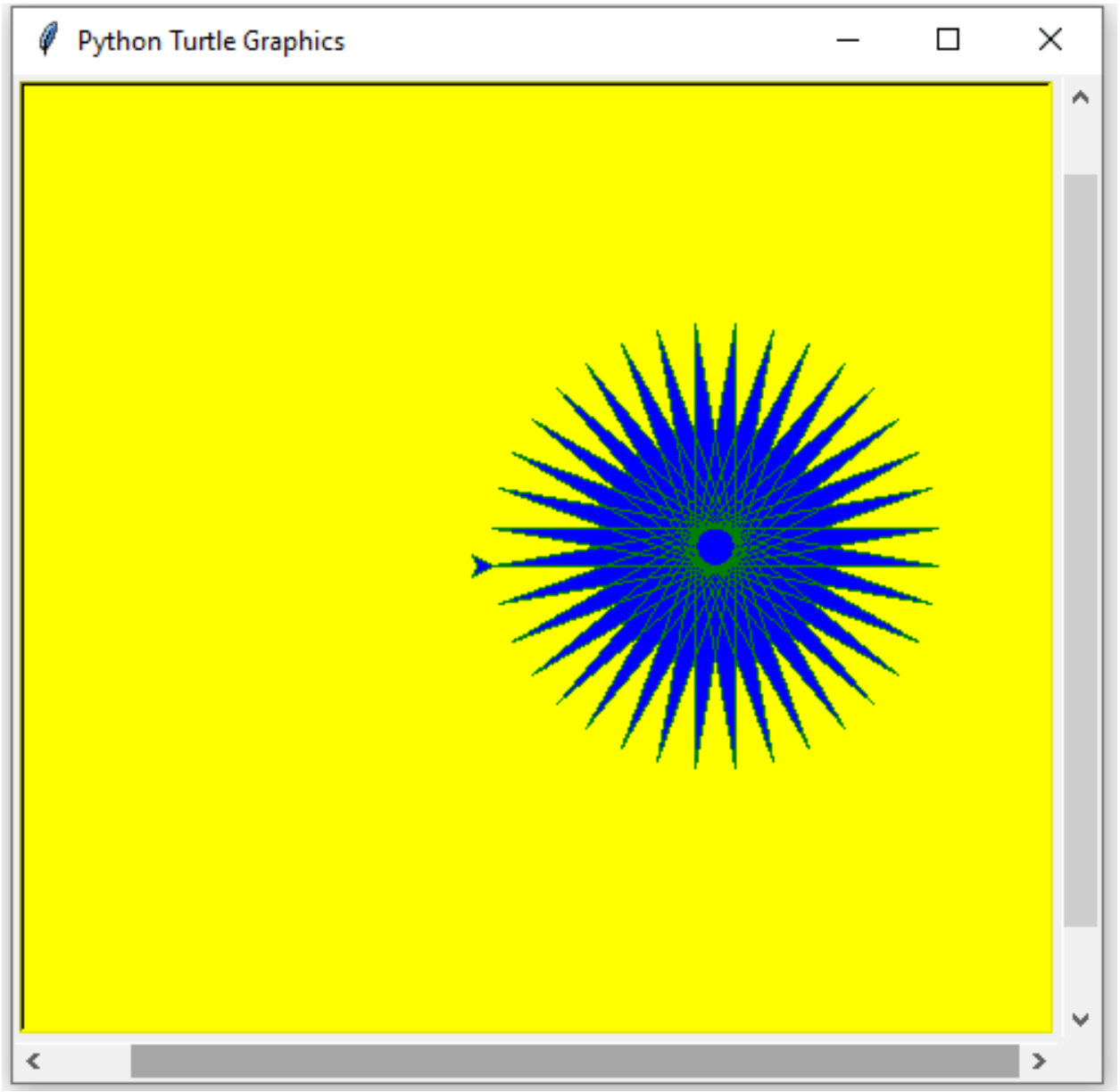
ALGORITHM:

Step 1:Start
Step 2:Import turtle and draw the image using pre-define function
Step 3:Using for loop to draw using functions such as a width(),screensize(), circle(),right()
Step 4:Print the result
Step 5:Stop

PROGRAM:

```
import turtle
turtle.screensize(canvwidth=500, canvheight=500, bg="yellow")
turtle.color('green', 'blue')
turtle.begin_fill()
while True:
   turtle.forward(200)
   turtle.left(170)
   if abs(turtle.pos()) < 1:
      break
turtle.end_fill()
turtle.done()
```

OUTPUT:



13.Tower of Hanoi

To develop a program for towers of hanoi using recursion

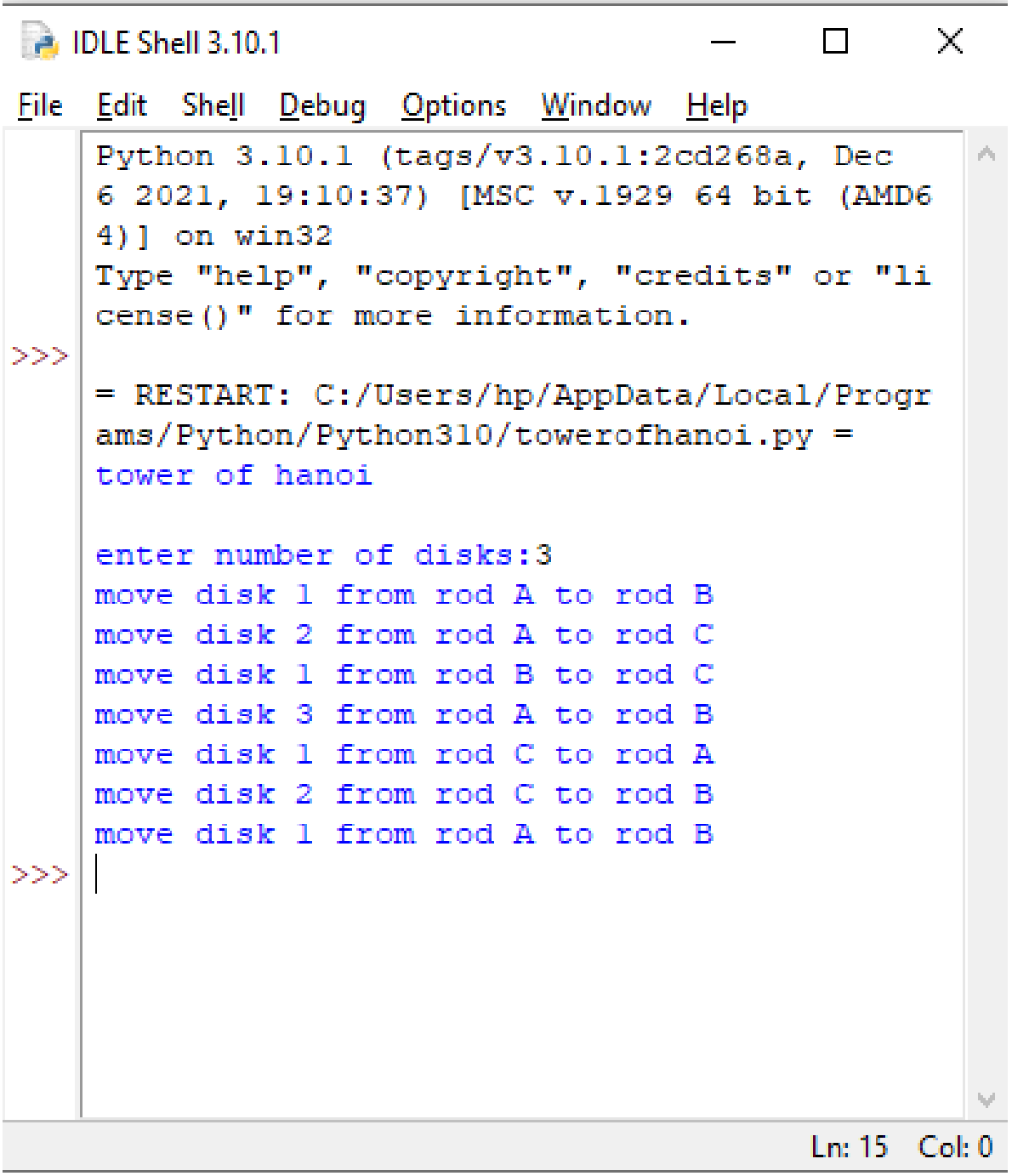
ALGORITHM:

Step 1: Start
Step 2: Let the three towers be the rodfrom, rodto, rodaux.
Step 3: Read the number of disks, n from the user.
Step 4: Move n-1 disks from source to rodaux.
Step 5: Move nth disk from source to rodto.
Step 6: Move n-1 disks from rodaux to rodto.
Step 7: Repeat Steps 3 to 5, by decrementing n by 1.
Step 8: Stop

PROGRAM:

```
print("tower of hanoi")
def towerofhanoi(n,rodfrom,rodto,rodaux):
    if n==1:
        print("move disk 1 from rod",rodfrom,"to rod",rodto)
        return
    towerofhanoi(n-1,rodfrom,rodaux,rodto)
    print("move disk",n,"from rod",rodfrom,"to rod",rodto)
    towerofhanoi(n-1,rodaux,rodto,rodfrom)
numberofdisk=int(input('\nenter number of disks:'))
towerofhanoi(numberofdisk,'A','B','C')
```

OUTPUT:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/towerofhanoi.py =
tower of hanoi
enter number of disks:3
move disk 1 from rod A to rod B
move disk 2 from rod A to rod C
move disk 1 from rod B to rod C
move disk 3 from rod A to rod B
move disk 1 from rod C to rod A
move disk 2 from rod C to rod B
move disk 1 from rod A to rod B
>>>
Ln: 15 Col: 0

14.Hangman game

To develop a code for Hangman game

ALGORITHM:

Step 1: start
Step 2: assign secret word, set allowed error count to 7
Step 3: start guessing the characters till allowed error count > 0 else go to Step7
Step 4: if guessed character is correct, guess next character
Step 5: decrement the allowed error count
Step 6: go to step 3
Step 7: If characters are guessed, print ‘word found’ else print ‘word not found’
Step 8: stop

PROGRAM:

```
word = "secret"
allowed_errors = 7
guesses = []
done = False
while not done:
    for letter in word:
        if letter.lower() in guesses:
            print(letter, end=" ")
        else:
            print("_",end=" ")
print(" ")
done=True
guess = input("allowed errors left{allowed_errors},Next Guess:")
guesses.append(guess.lower())
if guess.lower() not in word.lower():
    allowed_errors -= 1
if allowed_errors == 0:
    break
done = True
for letter in word:
    if letter.lower() not in guesses:
```

```
        done = False
    if done:
        print("You found the word! It was {word}!")
    else:
        print("Game over! The word was {word}!")
```

OUTPUT: