**Name: Kanak Agrawal**

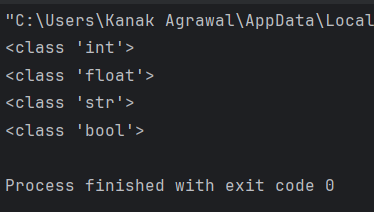
**PYTHON**

1. **DATATYPE**

**CODE:**

# int  
a = 5  
print(type(a))  
  
# Float  
a=10.5  
print(type(a))  
  
# String  
a="Kanak"  
print(type(a))  
  
# Boolean  
a = True  
print(type(a))

**OUTPUT:**

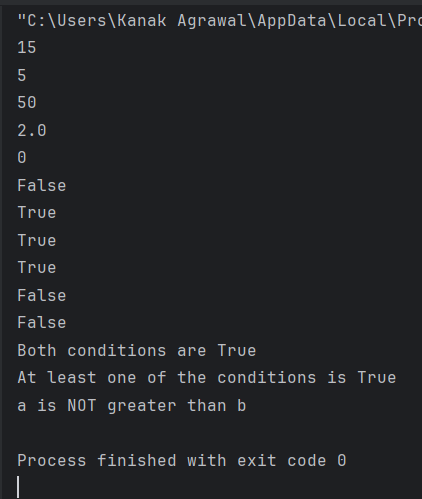
****

1. **OPERATOR**

**CODE:**

# arthimatic Operator  
  
# +  
print(10 + 5)  
  
# -  
print(10 - 5)  
  
# \*  
print(10 \* 5)  
  
# /  
print(10 / 5)  
  
# %  
print(10 % 5)  
  
  
# Comparision Operator  
  
# ==  
x = 5  
y = 3  
print(x == y)  
  
# !=  
x = 5  
y = 3  
print(x != y)  
  
# >  
x = 5  
y = 3  
print(x > y)  
  
# >=  
x = 5  
y = 3  
print(x >= y)  
  
# <  
x = 5  
y = 3  
print(x <= y)  
  
# <=  
x = 5  
y = 3  
print(x <= y)  
  
  
#logical Operator  
  
# AND  
a = 200  
b = 33  
c = 500  
if a > b and c > a:  
 print("Both conditions are True")  
  
  
# OR  
a = 200  
b = 33  
c = 500  
if a > b or a > c:  
 print("At least one of the conditions is True")  
  
  
# NOT  
a = 33  
b = 200  
if not a > b:  
 print("a is NOT greater than b")

**OUTPUT:**

****

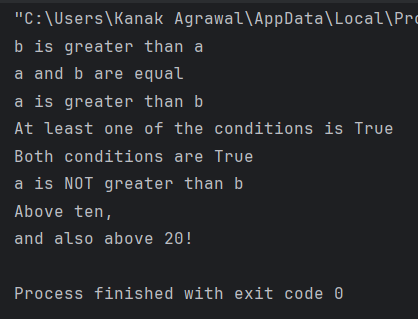
1. **CONTROL STRUCTURE**

**IF ELSE STATEMENT**

**CODE:**

# if  
a = 77  
b = 100  
if b > a:  
 print("b is greater than a")  
  
  
# if\_elif  
  
a = 33  
b = 33  
if b > a:  
 print("b is greater than a")  
elif a == b:  
 print("a and b are equal")  
  
  
  
# if \_elif\_else  
  
a = 100  
b = 77  
if b > a:  
 print("b is greater than a")  
elif a == b:  
 print("a and b are equal")  
else:  
 print("a is greater than b")  
  
  
# if else with OR  
  
a = 200  
b = 33  
c = 500  
if a > b or a > c:  
 print("At least one of the conditions is True")  
  
  
# if else with AND  
  
a = 200  
b = 33  
c = 500  
if a > b and c > a:  
 print("Both conditions are True")  
  
  
# if else with NOT  
  
a = 33  
b = 200  
if not a > b:  
 print("a is NOT greater than b")  
  
  
# nested if loop  
  
x = 41  
  
if x > 10:  
 print("Above ten,")  
 if x > 20:  
 print("and also above 20!")  
 else:  
 print("but not above 20.")

**OUTPUT:**

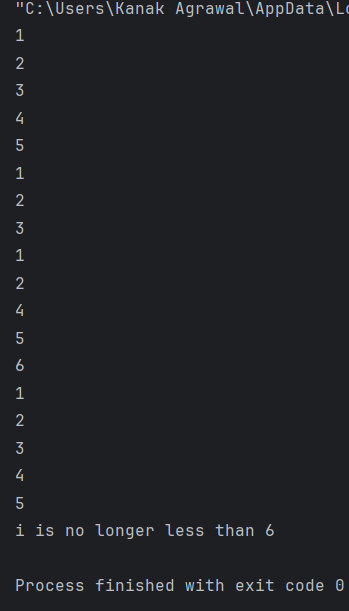
****

**WHILE LOOP**

**CODE**

# While Loop  
  
i = 1  
while i < 6:  
 print(i)  
 i += 1  
  
  
# While loop with break Statment  
  
i = 1  
while i < 6:  
 print(i)  
 if i == 3:  
 break  
 i += 1  
  
  
# While loop with continue statement  
  
i = 0  
while i < 6:  
 i += 1  
 if i == 3:  
 continue  
 print(i)  
  
  
# while loop with else statement  
  
  
i = 1  
while i < 6:  
 print(i)  
 i += 1  
else:  
 print("i is no longer less than 6")

**OUTPUT**

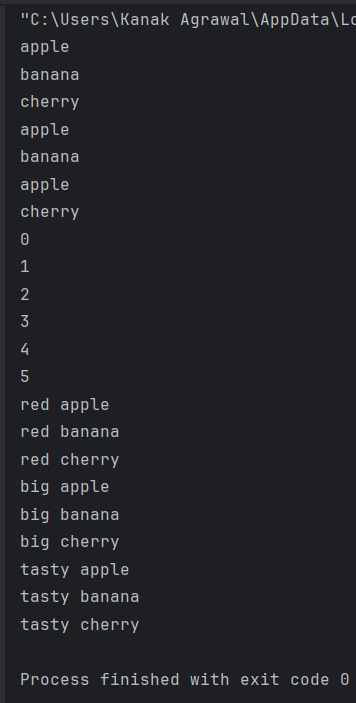
****

**FOR LOOP**

**CODE**

# For loop  
  
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
 print(x)  
  
  
# for loop with break statement  
  
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
 print(x)  
 if x == "banana":  
 break  
  
  
# for loop with continue statement  
  
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
 if x == "banana":  
 continue  
 print(x)  
  
  
# for loop with range  
  
for x in range(6):  
 print(x)  
  
  
# nested for loop  
  
adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]  
  
for x in adj:  
 for y in fruits:  
 print(x, y)  
  
  
# for loop with pass statement  
  
for x in [0, 1, 2]:  
 pass

**OUTPUT**

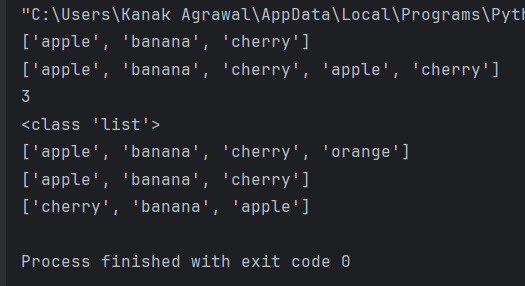
****

1. **LIST**

**CODE:**

# list  
thislist = ["apple", "banana", "cherry"]  
print(thislist)  
  
# duplicate list  
thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)  
  
# lenght of list  
thislist = ["apple", "banana", "cherry"]  
print(len(thislist))  
  
# Type of list  
mylist = ["apple", "banana", "cherry"]  
print(type(mylist))  
  
# list method append  
fruits = ['apple', 'banana', 'cherry']  
fruits.append("orange")  
print(fruits)  
  
# list method count  
fruits = ['apple', 'banana', 'cherry']  
x = fruits.count("cherry")  
print(fruits)  
  
# list method reverse  
fruits = ['apple', 'banana', 'cherry']  
fruits.reverse()  
print(fruits)

**OUTPUT:**

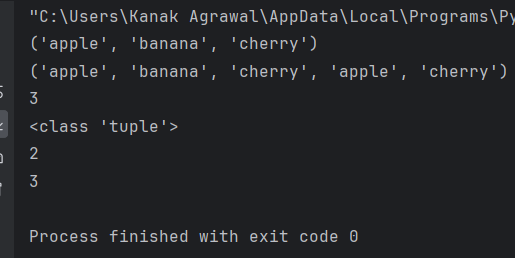
****

1. **TUPLE**

**CODE:**

# Tuple  
thistuple = ("apple", "banana", "cherry")  
print(thistuple)  
  
# duplicate tuple  
thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)  
  
# length of tuple  
thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))  
  
# Type of tuple  
mytuple = ("apple", "banana", "cherry")  
print(type(mytuple))  
  
# count tuple method  
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)  
x = thistuple.count(5)  
print(x)  
  
# index tuple method  
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)  
x = thistuple.index(8)  
print(x)

**OUTPUT:**

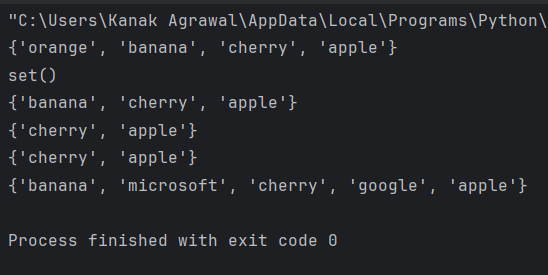
****

1. **SET AND METHODS**

**CODE:**

# set and set method  
  
# set add method  
fruits = {"apple", "banana", "cherry"}  
fruits.add("orange")  
print(fruits)  
  
# clear method  
fruits = {"apple", "banana", "cherry"}  
fruits.clear()  
print(fruits)  
  
# copy method  
fruits = {"apple", "banana", "cherry"}  
x = fruits.copy()  
print(x)  
  
# pop method  
fruits = {"apple", "banana", "cherry"}  
fruits.pop()  
print(fruits)  
  
  
# remove method  
fruits = {"apple", "banana", "cherry"}  
fruits.remove("banana")  
print(fruits)  
  
  
#update method  
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
x.update(y)  
print(x)

**OUTPUT:**

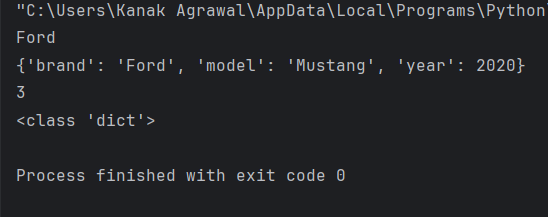
****

1. **DICTIONARY**

**CODE:**

# dictionary  
  
# dictionary item  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
print(thisdict["brand"])  
  
  
# duplicates not allowed in dictionary  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964,  
 "year": 2020  
}  
print(thisdict)  
  
  
# length of dictionary  
print(len(thisdict))  
  
  
# datatype of dictionary  
thisdict = {  
 "brand": "Ford",  
 "electric": False,  
 "year": 1964,  
 "colors": ["red", "white", "blue"]  
}  
  
  
# type of dictionary  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
print(type(thisdict))

**OUTPUT:**

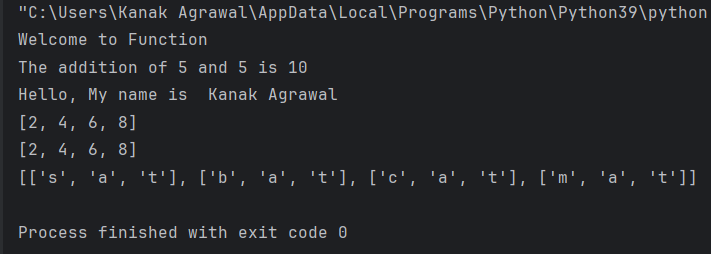
****

1. **FUNCTION**

**CODE:**

# calling a function  
funct()  
  
# python function with parameters  
# adding two number  
def add(num1:int,num2:int):  
 num3=num1+num2  
 return num3  
  
num1=5  
num2=5  
ans=add(num1,num2)  
print(f"The addition of {num1} and {num2} is {ans}")  
  
  
# python function with arguments  
def name(fname,lname):  
 print("Hello, My name is ", fname+" "+lname)  
  
  
name("Kanak","Agrawal")

**OUTPUT:**

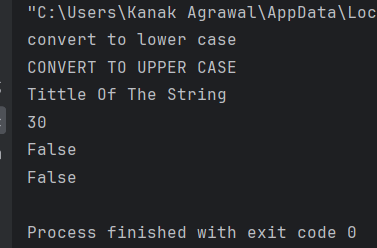
****

1. **STRING FUNCTION**

**CODE:**

# String Function  
  
# Concat String  
  
# Lower Case  
string = "ConveRT TO LOwer Case"  
print(string.lower())  
  
# Upper case String  
string = "Convert to upper case"  
print(string.upper())  
  
# Tittle string  
string="Tittle of the string"  
print(string.title())  
  
# Lenght of the string  
string = "Hello my name is kanak agrawal"  
print(len(string))  
  
# is alpha  
print(string.isalpha())  
  
#is decimal  
print(string.isdecimal())

**OUTPUT:**

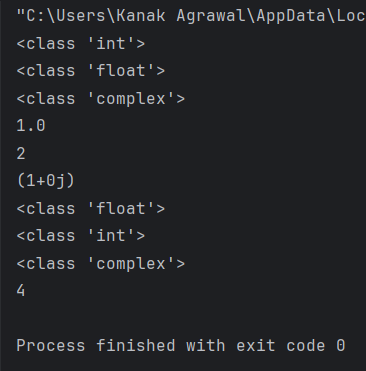
****

1. **NUMBER FUNCTION**

**CODE:**

# number Function  
# int,float,complex  
  
x = 1  
y = 2.8  
z = 1j  
print(type(x))  
print(type(y))  
print(type(z))  
  
# convert from int to float:  
a = float(x)  
  
# convert from float to int:  
b = int(y)  
  
# convert from int to complex:  
c = complex(x)  
  
print(a)  
print(b)  
print(c)  
  
print(type(a))  
print(type(b))  
print(type(c))  
  
  
# Random Number  
import random  
print(random.randrange(1, 10))

**OUTPUT:**

****

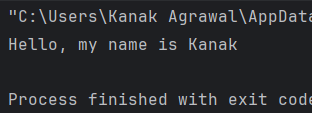
1. **OOPS**

**CLASS AND OBJECT**

**CODE:**

# creating a class and a object  
  
class Person:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 def greet(self):  
 print("Hello, my name is " + self.name)  
  
 # Create a new instance of the Person class and assign it to the variable person1  
  
  
person1=Person("Kanak",22)  
person1.greet()

**OUTPUT:**

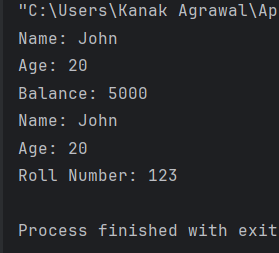
****

1. **ACCESS SPECIFIER**

**CODE:**

# Access Specifier  
  
# public specifier  
class Student:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 def display(self):  
 print("Name:", self.name)  
 print("Age:", self.age)  
  
  
s = Student("John", 20)  
s.display()  
  
  
# private specifier  
class BankAccount:  
 def \_\_init\_\_(self, account\_number, balance):  
 self.\_\_account\_number = account\_number  
 self.\_\_balance = balance  
  
 def display\_balance(self):  
 print("Balance:", self.\_\_balance)  
  
  
b = BankAccount(1234567890, 5000)  
b.display\_balance()  
  
  
# protected Modifier  
class Person:  
 def \_\_init\_\_(self, name, age):  
 self.\_name = name  
 self.\_age = age  
  
 def \_display(self):  
 print("Name:", self.\_name)  
 print("Age:", self.\_age)  
  
  
class Student(Person):  
 def \_\_init\_\_(self, name, age, roll\_number):  
 super().\_\_init\_\_(name, age)  
 self.\_roll\_number = roll\_number  
  
 def display(self):  
 self.\_display()  
 print("Roll Number:", self.\_roll\_number)  
  
  
s = Student("John", 20, 123)  
s.display()

**OUTPUT:**

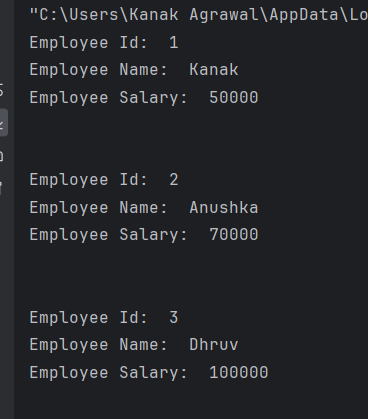
****

1. **CONSTRUCTOR**

**CODE:**

# Constructor  
  
class Employee:  
 def \_\_init\_\_(self,id,name,salary):  
 self.id = id  
 self.name = name  
 self.salary = salary  
  
 def emp\_details(self):  
 print("Employee Id: ", self.id)  
 print("Employee Name: ", self.name)  
 print("Employee Salary: ", self.salary)  
 print("\n")  
  
emp1 = Employee(1,"Kanak",50000)  
emp2 = Employee(2,"Anushka",70000)  
emp3 = Employee(3,"Dhruv",100000)  
  
emp1.emp\_details()  
emp2.emp\_details()  
emp3.emp\_details()

**OUTPUT:**

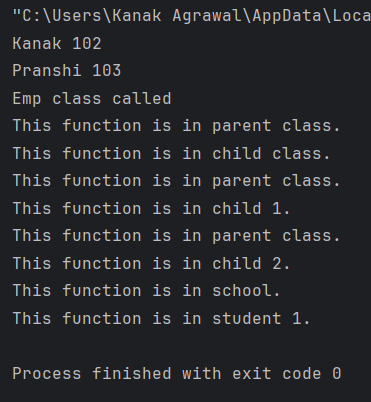
****

1. **INHERITANCE AND TYPES OF INHERITANCE**

**CODE:**

# Inheritance  
  
# creating a parent class  
class Person(object):  
  
 # Constructor  
 def \_\_init\_\_(self, name, id):  
 self.name = name  
 self.id = id  
  
 def Display(self):  
 print(self.name, self.id)  
  
  
emp = Person("Kanak", 102) # An Object of Person  
emp.Display()  
  
  
# creating a child class  
class Emp(Person):  
  
 def Print(self):  
 print("Emp class called")  
  
  
Emp\_details = Emp("Pranshi", 103)  
  
# calling parent class function  
Emp\_details.Display()  
  
# Calling child class function  
Emp\_details.Print()  
  
  
# Types of inheritance  
# single inheritance  
class Parent:  
 def func1(self):  
 print("This function is in parent class.")  
  
# Child class  
class Child(Parent):  
 def func2(self):  
 print("This function is in child class.")  
  
object = Child()  
object.func1()  
object.func2()  
  
# Heirrichal inheritance  
class Parent:  
 def func1(self):  
 print("This function is in parent class.")  
  
# Parent class  
class Child1(Parent):  
 def func2(self):  
 print("This function is in child 1.")  
  
# child class  
class Child2(Parent):  
 def func3(self):  
 print("This function is in child 2.")  
  
object1 = Child1()  
object2 = Child2()  
object1.func1()  
object1.func2()  
object2.func1()  
object2.func3()  
  
  
# Hybrid Inheritance  
class School:  
 def func1(self):  
 print("This function is in school.")  
  
class Student1(School):  
 def func2(self):  
 print("This function is in student 1. ")  
  
  
class Student2(School):  
 def func3(self):  
 print("This function is in student 2.")  
  
class Student3(Student1, School):  
 def func4(self):  
 print("This function is in student 3.")  
  
object = Student3()  
object.func1()  
object.func2()

**OUTPUT:**

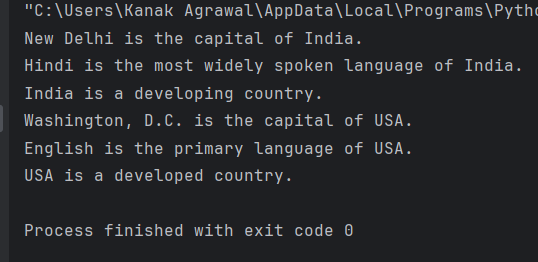
****

1. **POLYMORPHISMS**

**CODE:**

# polymorphism  
  
class India():  
 def capital(self):  
 print("New Delhi is the capital of India.")  
  
 def language(self):  
 print("Hindi is the most widely spoken language of India.")  
  
 def type(self):  
 print("India is a developing country.")  
  
  
class USA():  
 def capital(self):  
 print("Washington, D.C. is the capital of USA.")  
  
 def language(self):  
 print("English is the primary language of USA.")  
  
 def type(self):  
 print("USA is a developed country.")  
  
  
obj\_ind = India()  
obj\_usa = USA()  
for country in (obj\_ind, obj\_usa):  
 country.capital()  
 country.language()  
 country.type()

**OUTPUT:**

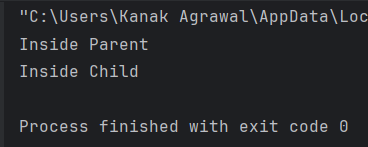
****

1. **METHOD OVERRIDDING**

**CODE:**

# Method overridding  
  
# Defining a parent class  
class Parent():  
 # producer  
 def \_\_init\_\_(self):  
 self.value = "Inside Parent"  
 # showing a parents method  
 def show(self):  
 print(self.value)  
   
# Defining a child class  
class Child(Parent):  
 # Constructor  
 def \_\_init\_\_(self):  
 self.value = "Inside Child"  
 # showing the child's method  
 def show(self):  
 print(self.value)  
  
  
obj1 = Parent()  
obj2 = Child()  
obj1.show()  
obj2.show()

**OUTPUT:**

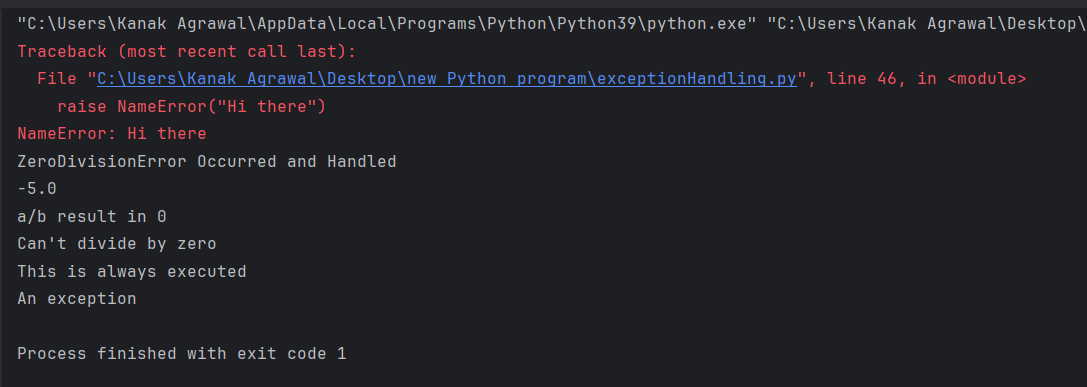
****

1. **EXCEPTION HANDLING**

**CODE:**

# Exception Handling  
# try and expect  
  
def funct(a):  
 if a < 4:  
 b = a / (a - 3)  
 print("Value of b = ", b)  
  
  
try:  
 funct(3)  
 funct(5)  
except ZeroDivisionError:  
 print("ZeroDivisionError Occurred and Handled")  
except NameError:  
 print("NameError Occurred and Handled")  
  
  
# try else except  
def AbyB(a , b):  
 try:  
 c = ((a+b) / (a-b))  
 except ZeroDivisionError:  
 print ("a/b result in 0")  
 else:  
 print (c)  
AbyB(2.0, 3.0)  
AbyB(3.0, 3.0)  
  
  
# Finally  
try:  
 k = 5 // 0  
 print(k)  
  
except ZeroDivisionError:  
 print("Can't divide by zero")  
  
finally:  
 print('This is always executed')  
  
  
  
# raising a exception  
try:  
 raise NameError("Hi there")  
except NameError:  
 print ("An exception")  
 raise

**OUTPUT:**

****

1. **MODULES**

**CODE:**

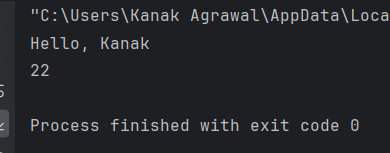
# python Module  
  
def greeting(name):  
 print("Hello, " + name)  
  
person1 = {  
 "name": "Kanak",  
 "age": 22,  
 "country": "Agrawal"  
}

**User define modules**

**CODE:**

# user define module  
  
import mymodule  
mymodule.greeting("Kanak")  
  
  
import mymodule  
a = mymodule.person1["age"]  
print(a)

**OUTPUT:**

****