**Exercice-11 Classes and Inheritance**

**Q1. Define class and instance**

Class: A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects created from the class will have.

Instance: An instance is a specific object created from a class. Each instance has its own unique data (attributes) but shares the same methods defined in the class.

**Q2. How many constructors can be defined in a ‘Class?’**

In Python, only one constructor (\_\_init\_\_) can be defined in a class. Python does not support multiple constructors like some other languages (e.g., Java). However, you can simulate multiple constructors using default arguments or class methods

**Q3. What is the name of method that initializes instance variable?**

The method that initializes instance variables is called the \_\_init\_\_ method. It is automatically called when a new instance of the class is created.

**Q4. Write the difference between static and instance variable**

|  |  |
| --- | --- |
| **Static Variable** | **Instance Variable** |
| Declared inside the class but outside any method. | Declared inside the \_\_init\_\_ method or other instance methods. |
| Shared by all instances of the class. | Unique to each instance of the class. |
| Accessed using the class name. | Accessed using the instance name. |
| Example: ClassName.variable\_name | Example: self.variable\_name |

**Q5. Write the significance of the word ‘Self’**

Self is a reference to the current instance of the class. It is used to access instance variables and methods within the class. When you create an instance of a class, Python automatically passes the instance as the first argument to all instance methods.

**Q6. Write the significance of super keyword.**

The super() keyword is used to call a method from the parent class (superclass) in inheritance. It is commonly used in the \_\_init\_\_ method of a child class to initialize the parent class’s attributes. Example:

Class Parent:

Def \_\_init\_\_(self, name):

Self.name = name

Class Child(Parent):

Def \_\_init\_\_(self, name, age):

Super().\_\_init\_\_(name) # Calls Parent’s \_\_init\_\_ method

Self.age = age

**Q7. Write the usage of**

**1. issubclass(sub,sup) 2. isinstanc isinstance (obj, class) e (obj, class)**

**3. is operator 4. type**

1.Issubclass(sub, sup):

Checks if sub is a subclass of sup.

Example: issubclass(Child, Parent) returns True if Child inherits from Parent.

2.Isinstance(obj, class):

Checks if obj is an instance of class or a subclass of class.

Example: isinstance(obj, Parent) returns True if obj is an instance of Parent or its subclass.

3.Is operator:

Checks if two variables point to the same object in memory.

Example: a is b returns True if a and b refer to the same object.

4.Type:

Returns the type of an object.

Example: type(10) returns <class ‘int’>.

**Q8. List the types of inheritance**

Single Inheritance: A class inherits from one parent class.

Multiple Inheritance: A class inherits from more than one parent class.

Multilevel Inheritance: A class inherits from a parent class, which in turn inherits from another class.

Hierarchical Inheritance: Multiple classes inherit from a single parent class.

Hybrid Inheritance: A combination of two or more types of inheritance.

**Q9. What is MRO?**

MRO (Method Resolution Order) is the order in which Python looks for a method in a hierarchy of classes. It is determined by the C3 linearization algorithm and ensures that a method is searched in the correct order (e.g., from child to parent classes).

You can view the MRO of a class using the \_\_mro\_\_ attribute or the mro() method.

Example:

Class A: pass

Class B(A): pass

Class C(B): pass

Print(C.\_\_mro\_\_)

# Output: (<class ‘\_\_main\_\_.C’>, <class ‘\_\_main\_\_.B’>, <class ‘\_\_main\_\_.A’>, <class ‘object’>)

**10.How to achieve method overloading in python**

Python does not support method overloading (defining multiple methods with the same name but different parameters) directly. However, you can achieve similar functionality using:

1.Default Arguments:

Def add(a, b=0, c=0):

Return a + b + c

Print(add(1)) # Output: 1

Print(add(1, 2)) # Output: 3

Print(add(1, 2, 3)) # Output: 6

2.Variable-Length Arguments (\*args or \*\*kwargs):

Def add(\*args):

Return sum(args)

Print(add(1, 2, 3)) # Output: 6

3. Using functools.singledispatch (for function overloading):

From functools import singledispatch

@singledispatch

Def func(arg):

Print(“Default:”, arg)

@func.register(int)

Def \_(arg):

Print(“Integer:”, arg)

@func.register(str)

Def \_(arg):

Print(“String:”, arg)

Func(10) # Output: Integer: 10

Func(“Hi”) # Output: String: Hi

**Programs**

**Q1. Write a class called Password\_manager. The class should have a list called old\_passwords that holds the entire user’s past passwords. The last item of the list is the user’s current pass-word. There should be a method called get\_password that returns the current password and a method called set\_password that sets the user’s password. The set\_password method should only change the password if the attempted password is different from all the user’s past passwords. Finally, create a method called is\_correct that receives a string and returns a Boolean True or False depending on whether the string is equal to the current password or not.**

Class Password\_manager:

Def \_\_init\_\_(self):

Self.old\_passwords = [“password1”, “password2”, “current\_password”] # Example old passwords

Def get\_password(self):

Return self.old\_passwords[-1] # Return the current password

Def set\_password(self, new\_password):

If new\_password not in self.old\_passwords:

Self.old\_passwords.append(new\_password)

Print(“Password changed successfully!”)

Else:

Print(“Password must be different from all past passwords.”)

Def is\_correct(self, password):

Return password == self.old\_passwords[-1]

Pm = Password\_manager() # Test the class

Print(“Current Password:”, pm.get\_password())

Pm.set\_password(“new\_password”)

Print(“Is ‘new\_password’ correct?”, pm.is\_correct(“new\_password”))

Print(“This program is executed by 22007-CS-018”)

Output:

Current Password: current\_password  
Password changed successfully!  
Is ‘new\_password’ correct? True  
This program is executed by 22007-CS-018

**Q2. Write a class called Rock\_paper\_scissors that implements the logic of the game Rock-paper-scissors. For this game the user plays against the computer for a certain number of rounds. Your class should have fields for the how many rounds there will be, the current round number, and the number of wins each player has. There should be methods for getting the computer’s choice, finding the winner of a round. You may want more methods.**

Import random

Class Rock\_paper\_scissors:

Def \_\_init\_\_(self, rounds):

Self.rounds = rounds

Self.current\_round = 1

Self.user\_wins = 0

Self.computer\_wins = 0

Def get\_computer\_choice(self):

Return random.choice([“rock”, “paper”, “scissors”])

Def find\_winner(self, user\_choice, computer\_choice):

If user\_choice == computer\_choice:

Return “draw”

Elif (user\_choice == “rock” and computer\_choice == “scissors”) or \

(user\_choice == “paper” and computer\_choice == “rock”) or \

(user\_choice == “scissors” and computer\_choice == “paper”):

Return “user”

Else:

Return “computer”

Def play\_round(self, user\_choice):

Computer\_choice = self.get\_computer\_choice()

Print(f”Round {self.current\_round}: You chose {user\_choice}, Computer chose {computer\_choice}”)

Winner = self.find\_winner(user\_choice, computer\_choice)

If winner == “user”:

Self.user\_wins += 1

Print(“You win this round!”)

Elif winner == “computer”:

Self.computer\_wins += 1

Print(“Computer wins this round!”)

Else:

Print(“It’s a draw!”)

Self.current\_round += 1

Def display\_score(self):

Print(f”Score: You {self.user\_wins} – Computer {self.computer\_wins}”)

# Test the class

Game = Rock\_paper\_scissors(3)

Game.play\_round(“rock”)

Game.play\_round(“paper”)

Game.play\_round(“scissors”)

Game.display\_score()

Print(“This program is executed by 22007-CS-018”)

Output:

Round 1: You chose rock, Computer chose scissors  
You win this round!  
Round 2: You chose paper, Computer chose rock  
You win this round!  
Round 3: You chose scissors, Computer chose paper  
You win this round!  
Score: You 3 – Computer 0  
This program is executed by 22007-CS-018

**Q3. Define a Person with attributes name, age and gender and a method display\_ details function. Then define three different classes namely Pilot, Teacher and a Doctor that inherits Person class, add to these classes respective attributes and methods. In the table given below attributes of different profession are mentioned, you add at least two services given by each of these individual. Name the type of inheritance that should be employed in this case.**

**PILOT**

**Teacher**

**Doctor**

**Airline**

**Subject**

**Hospital**

**Hours (No. of Hours in Air)**

**College**

**Experience**

**Home Airport**

**No. of Students**

**Specialization**

**Destination Airport**

**Workload**

**Daily Patients**

**Flight Size**

**Semester**

**Timings**

**Services**

Class Person:

Def \_\_init\_\_(self, name, age, gender):

Self.name = name

Self.age = age

Self.gender = gender

Def display\_details(self):

Print(f”Name: {self.name}, Age: {self.age}, Gender: {self.gender}”)

Class Pilot(Person):

Def \_\_init\_\_(self, name, age, gender, airline, home\_airport):

Super().\_\_init\_\_(name, age, gender)

Self.airline = airline

Self.home\_airport = home\_airport

Def services(self):

Print(f”{self.name} provides flight services and safety checks.”)

Class Teacher(Person):

Def \_\_init\_\_(self, name, age, gender, subject, college):

Super().\_\_init\_\_(name, age, gender)

Self.subject = subject

Self.college = college

Def services(self):

Print(f”{self.name} teaches {self.subject} and conducts exams.”)

Class Doctor(Person):

Def \_\_init\_\_(self, name, age, gender, hospital, specialization):

Super().\_\_init\_\_(name, age, gender)

Self.hospital = hospital

Self.specialization = specialization

Def services(self):

Print(f”{self.name} provides medical services and consultations.”)

# Test the classes

Pilot = Pilot(“John”, 35, “Male”, “ABC Airlines”, “JFK”)

Teacher = Teacher(“Alice”, 40, “Female”, “Math”, “XYZ College”)

Doctor = Doctor(“Bob”, 45, “Male”, “City Hospital”, “Cardiology”)

Pilot.display\_details()

Pilot.services()

Teacher.display\_details()

Teacher.services()

Doctor.display\_details()

Doctor.services()

Print(“This program is executed by 22007-CS-018”)

Output**:**

Name: John, Age: 35, Gender: Male  
John provides flight services and safety checks.  
Name: Alice, Age: 40, Gender: Female  
Alice teaches Math and conducts exams.  
Name: Bob, Age: 45, Gender: Male  
Bob provides medical services and consultations.  
This program is executed by 22007-CS-018

**Q4. Write a program with traits to the A class, which are then inherited into the child class, class B. Mr. XYZ have his own traits but also inherits the traits from class B. Write a function in XYZ class that display all the traits of Mr.XYZ. Mention the type of inheritance should be employed here.**

Class A:

Def \_\_init\_\_(self):

Self.trait\_a = “Trait from A”

Class B(A):

Def \_\_init\_\_(self):

Super().\_\_init\_\_()

Self.trait\_b = “Trait from B”

Class XYZ(B):

Def \_\_init\_\_(self):

Super().\_\_init\_\_()

Self.trait\_xyz = “Trait from XYZ”

Def display\_traits(self):

Print(self.trait\_a)

Print(self.trait\_b)

Print(self.trait\_xyz)

# Test the class

Xyz = XYZ()

Xyz.display\_traits()

Print(“This program is executed by 22007-CS-018”)

Output:

Trait from A  
Trait from B  
Trait from XYZ  
This program is executed by 22007-CS-018

**Q6. Write a python program where a child inherits the attributes and function from both father and mother. Add additional attributes and function for the child class. Call all the methods and display all the attributes of the child class along with the inherited using the child class object. Mention the type of inheritance should be employed here.**

# Father class

Class Father:

Def \_\_init\_\_(self):

Self.trait\_father = “Kindness” # Trait from Father

Def father\_method(self):

Print(“Father’s method: Teaches discipline.”)

# Mother class

Class Mother:

Def \_\_init\_\_(self):

Self.trait\_mother = “Compassion” # Trait from Mother

Def mother\_method(self):

Print(“Mother’s method: Teaches love and care.”)

# Child class inherits from both Father and Mother

Class Child(Father, Mother):

Def \_\_init\_\_(self):

Super().\_\_init\_\_() # Calls Father’s \_\_init\_\_ (due to MRO)

Self.trait\_child = “Curiosity” # Trait from Child

Def child\_method(self):

Print(“Child’s method: Learns and explores.”)

Def display\_traits(self):

Print(“Traits inherited from Father:”, self.trait\_father)

Print(“Traits inherited from Mother:”, self.trait\_mother)

Print(“Traits of Child:”, self.trait\_child)

# Create an instance of Child

Child = Child()

# Call methods and display traits

Child.display\_traits()

Child.father\_method()

Child.mother\_method()

Child.child\_method()

Print(“This program is executed by 22007-CS-018”)

Output:

Traits inherited from Father: Kindness  
Traits inherited from Mother: Compassion  
Traits of Child: Curiosity  
Father’s method: Teaches discipline.  
Mother’s method: Teaches love and care.  
Child’s method: Learns and explores.  
This program is executed by 22007-CS-018

**Q7. A person has two kids, each kid have their own attributes and the inherited. Display the details of both kids along with the inherited ones. Mention the type of inheritance should be employed here.**

# Parent Class

Class Person:

Def \_\_init\_\_(self, name, age):

Self.name = name

Self.age = age

Def display(self):

Print(f”Name: {self.name}, Age: {self.age}”)

# Child Class

Class Kid(Person):

Def \_\_init\_\_(self, name, age, hobby):

Super().\_\_init\_\_(name, age)

Self.hobby = hobby

Def display(self):

Super().display()

Print(f”Hobby: {self.hobby}”)

Kid1 = Kid(“Alice”, 10, “Drawing”) # Creating instances of Kid

Kid2 = Kid(“Bob”, 8, “Dancing”)

# Displaying details of both kids

Kid1.display()

Kid2.display()

Print(“This program is executed by 22007-CS-018”)

Output:

Name: Alice, Age: 10  
Hobby: Drawing  
Name: Bob, Age: 8  
Hobby: Dancing  
This program is executed by 22007-CS-018

**Q8. Write a python program that illustrates how the constructors are called in a multi level inheritance.**

# Base Class

Class GrandParent:

Def \_\_init\_\_(self):

Print(“GrandParent Constructor Called”)

# Intermediate Class

Class Parent(GrandParent):

Def \_\_init\_\_(self):

Super().\_\_init\_\_()

Print(“Parent Constructor Called”)

# Derived Class

Class Child(Parent):

Def \_\_init\_\_(self):

Super().\_\_init\_\_()

Print(“Child Constructor Called”)

# Creating an instance of Child

Child = Child()

Print(“This program is executed by 22007-CS-018”)

Output:

GrandParent Constructor Called  
Parent Constructor Called  
Child Constructor Called  
This program is executed by 22007-CS-018

**Q9. Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest. Write a python program that overrides the get\_interest\_rate functionality in the Bank class for each of the Banks. Mentioned the type of inheritance should be employed here.**

# Base Class

Class Bank:

Def get\_interest\_rate(self):

Return “Unknown”

# Derived Class 1

Class SBI(Bank):

Def get\_interest\_rate(self):

Return “8%”

# Derived Class 2

Class ICICI(Bank):

Def get\_interest\_rate(self):

Return “7%”

# Derived Class 3

Class AXIS(Bank):

Def get\_interest\_rate(self):

Return “9%”

# Creating instances of each bank

Sbi = SBI()

Icici = ICICI()

Axis = AXIS()

# Displaying interest rates

Print(“SBI Interest Rate:”, sbi.get\_interest\_rate())

Print(“ICICI Interest Rate:”, icici.get\_interest\_rate())

Print(“AXIS Interest Rate:”, axis.get\_interest\_rate())

Print(“This program is executed by 22007-CS-018”)

Output:

SBI Interest Rate: 8%  
ICICI Interest Rate: 7%  
AXIS Interest Rate: 9%  
This program is executed by 22007-CS-018

**Q10. Write a python using id and type operator on class instance**

# Define a simple class

Class MyClass:

Def \_\_init\_\_(self, value):

Self.value = value

# Create an instance of the class

Obj = MyClass(10)

# Using id() to get the memory address of the object

Print(“Memory address of obj:”, id(obj))

# Using type() to get the type of the object

Print(“Type of obj:”, type(obj))

Print(“This program is executed by 22007-CS-018”)

Output:

Memory address of obj: <memory\_address>  
Type of obj: <class ‘\_\_main\_\_.MyClass’>  
This program is executed by 22007-CS-018

Note: The <memory\_address> will be a unique identifier representing the memory location of the object obj. The actual value will vary each time you run the program.

**Exercice-12 Virtual Environment and Packages**

**Q1. Define Virtual Environment**

A virtual environment is an isolated Python environment that allows you to manage dependencies for a specific project separately from other projects. It ensures that the packages and versions used in one project do not conflict with those in another.

**Q2. Define Module**

A module is a single Python file containing Python code, such as functions, classes, and variables. It can be imported and used in other Python scripts.

**Q3. What is the value of \_\_name\_\_ attribute in the module itself and when the module is imported?**

When a module is run as the main program, the value of \_\_name\_\_ is “\_\_main\_\_”.

When a module is imported into another script, the value of \_\_name\_\_ is the name of the module (i.e., the filename without the .py extension).

**Q4. What is the use of ‘as’ keyword when importing modules?**

The as keyword is used to provide an alias (alternative name) for a module or object when importing it. This is useful to avoid naming conflicts or to shorten long module names.

Example:

Import numpy as np

**Q5. Define package**

A package is a collection of related modules organized in a directory hierarchy. It must contain a special file called \_\_init\_\_.py to be recognized as a package.

**Q6. What is the need of package?**

Packages help organize and structure Python code into reusable and maintainable components. They allow for better modularity and avoid naming conflicts between modules.

**Q7. What is the need of \_\_init\_\_ file?**

The \_\_init\_\_.py file is used to mark a directory as a Python package. It can also be used to initialize package-level variables or execute code when the package is imported.

**Q8. What is the need of virtual Environment?**

A virtual environment is needed to:

Isolate project dependencies.

Avoid conflicts between different versions of the same package.

Ensure reproducibility of the project across different systems.

**Q9. List two tools to create virtual environment**

Venv (built into Python 3)

Virtualenv (third-party tool)

**Q10. What is the use pip command?**

The pip command is used to install, upgrade, and manage Python packages and dependencies.

**Q11. List five pip commands options**

Pip install <package>: Installs a package.

Pip uninstall <package>: Uninstalls a package.

Pip freeze: Lists installed packages and their versions.

Pip list: Lists all installed packages.

Pip show <package>: Displays information about a specific package.

**Q12. Write the steps to create, activate and deactivate virtual environment using command window.**

Create a virtual environment:

Bash

Python -m venv myenv

Replace myenv with the desired name for your virtual environment.

Activate the virtual environment:

On Windows:

Bash

Myenv\Scripts\activate

On macOS/Linux:

Bash

Source myenv/bin/activate

Deactivate the virtual environment:

Bash

Deactivate

**Q13. Write the steps to create virtual environment in ‘Pycharm’**

Open your project in PyCharm.

Go to File > Settings (or PyCharm > Preferences on macOS).

Navigate to Project: <project\_name> > Python Interpreter.

Click the gear icon and select Add.

Choose Virtualenv Environment and specify the location for the virtual environment.

Click OK to create the virtual environment

**Q14. Write the steps to install packages for a particular virtual environment using command window.**

Activate the virtual environment (see Q12 for activation steps).

Use the pip install command to install the required package:

Bash

Pip install <package\_name>

**Q15. Write the steps to install packages for a particular virtual environment using ‘Pycharm’**

Open your project in PyCharm.

Go to File > Settings (or PyCharm > Preferences on macOS).

Navigate to Project: <project\_name> > Python Interpreter.

Click the + button to add a new package.

Search for the package you want to install and click Install Package.

**Programs**

**Q1. Create a package called college with three different modules, where each module holds a separate class namely student, teacher and computer laboratory. Define these classes with attributes and methods. Import these modules by using import statement into the packages namely admission where student takes the admission, employed where faculty employed for a particular subject and test packages where a student takes his online test.**

College/

\_\_init\_\_.py

Student.py

Teacher.py

Computer\_laboratory.py

Admission/

\_\_init\_\_.py

Admission\_process.py

Employed/

\_\_init\_\_.py

Employment\_process.py

Test/

\_\_init\_\_.py

Test\_process.py

**College/student.py:**

Class Student:

Def \_\_init\_\_(self, name, roll\_no):

Self.name = name

Self.roll\_no = roll\_no

Def display(self):

Print(f”Student Name: {self.name}, Roll No: {self.roll\_no}”)

Print(“This program is executed by 22007-CS-018”)

**College/teacher.py:**

Class Teacher:

Def \_\_init\_\_(self, name, subject):

Self.name = name

Self.subject = subject

Def display(self):

Print(f”Teacher Name: {self.name}, Subject: {self.subject}”)

Print(“This program is executed by 22007-CS-018”)

**College/computer\_laboratory.py**

Class ComputerLaboratory:

Def \_\_init\_\_(self, lab\_name, capacity):

Self.lab\_name = lab\_name

Self.capacity = capacity

Def display(self):

Print(f”Lab Name: {self.lab\_name}, Capacity: {self.capacity}”)

Print(“This program is executed by 22007-CS-018”)

**Admission/admission\_process.py:**

Import college.student as student

S = student.Student(“John Doe”, 101)

s.display()

print(“This program is executed by 22007-CS-018”)

output:

Student Name: John Doe, Roll No: 101

This program is executed by 22007-CS-018

**employed/employment\_process.py:**

import college.teacher as teacher

t = teacher.Teacher(“Dr. Smith”, “Computer Science”)

t.display()

print(“This program is executed by 22007-CS-018”)

Output:

Teacher Name: Dr. Smith, Subject: Computer Science

This program is executed by 22007-CS-018

**test/test\_process.py:**

import college.computer\_laboratory as lab

l = lab.ComputerLaboratory(“CS Lab”, 50)

l.display()

print(“This program is executed by 22007-CS-018”)

Output:

Lab Name: CS Lab, Capacity: 50  
This program is executed by 22007-CS-018

**Q2. Create a package called college with three different modules, where each module holds a separate class namely student, teacher and computer laboratory. Define these classes with attributes and methods. Import these modules by using from statement into the packages namely admission where student takes the admission, employed where faculty employed for a particular subject and test packages where a student takes his online test.**

**Admission/admission\_process.py:**

From college.student import Student

S = Student(“John Doe”, 101)

s.display()

print(“This program is executed by 22007-CS-018”)

Output:

Student Name: John Doe, Roll No: 101  
This program is executed by 22007-CS-018

**employed/employment\_process.py:**

From college.teacher import Teacher

T = Teacher(“Dr. Smith”, “Computer Science”)

t.display()

print(“This program is executed by 22007-CS-018”)

Output:

Teacher Name: Dr. Smith, Subject: Computer Science  
This program is executed by 22007-CS-018

**test/test\_process.py:**

From college.computer\_laboratory import ComputerLaboratory

L = ComputerLaboratory(“CS Lab”, 50)

l.display()

print(“This program is executed by 22007-CS-018”)

Output:

Lab Name: CS Lab, Capacity: 50  
This program is executed by 22007-CS-018

**Q3. Create a package called college with three different modules, where each module holds a separate class namely student, teacher and computer laboratory. Define these classes with attributes and methods. Save the module with the same name as the class name. Import these modules by using import/from statement into the packages namely admission where student takes the admission, employed where faculty employed for a particular subject and test packages where a student takes his online test. Explain how to access the access the classes if the module name and the class name are same.**

Directory Structure:

Copy

College/

\_\_init\_\_.py

Student.py

Teacher.py

ComputerLaboratory.py

Admission/

\_\_init\_\_.py

Admission\_process.py

Employed/

\_\_init\_\_.py

Employment\_process.py

Test/

\_\_init\_\_.py

Test\_process.py

Code:

**College/Student.py:**

Class Student:

Def \_\_init\_\_(self, name, roll\_no):

Self.name = name

Self.roll\_no = roll\_no

Def display(self):

Print(f”Student Name: {self.name}, Roll No: {self.roll\_no}”)

Print(“This program is executed by 22007-CS-018”)

**Admission/admission\_process.py:**

From college.Student import Student

S = Student(“John Doe”, 101)

s.display()

print(“This program is executed by 22007-CS-018”)

Output:

Student Name: John Doe, Roll No: 101  
This program is executed by 22007-CS-018

**Q4. Write the steps to install numpy and scikit-learn (specific version) into the virtual environment using command windows.**

Activate the virtual environment:

Bash

Myenv\Scripts\activate # On Windows

Source myenv/bin/activate # On macOS/Linux

Install specific versions of numpy and scikit-learn:

Bash

Pip install numpy==1.21.0

Pip install scikit-learn==0.24.2

The output will look something like this:

Collecting numpy==1.21.0

Downloading numpy-1.21.0-cp39-cp39-win\_amd64.whl (14.0 MB)

Installing collected packages: numpy

Successfully installed numpy-1.21.0

Collecting scikit-learn==0.24.2

Downloading scikit\_learn-0.24.2-cp39-cp39-win\_amd64.whl (7.2 MB)

Installing collected packages: scikit-learn

Successfully installed scikit-learn-0.24.2

**Q5. Write the steps to install numpy and scikit-learn (specific version) for a particular project using ‘pycharm’**

Open your project in PyCharm.

Go to File > Settings (or PyCharm > Preferences on macOS).

Navigate to Project: <project\_name> > Python Interpreter.

Click the + button to add a new package.

Search for numpy and specify the version (e.g., 1.21.0), then click Install Package.

Repeat the process for scikit-learn (e.g., 0.24.2).

The output in the PyCharm terminal will look something like this:

Installing package ‘numpy’ with version ‘1.21.0’

Collecting numpy==1.21.0

Downloading numpy-1.21.0-cp39-cp39-win\_amd64.whl (14.0 MB)

Installing collected packages: numpy

Successfully installed numpy-1.21.0

Installing package ‘scikit-learn’ with version ‘0.24.2’

Collecting scikit-learn==0.24.2

Downloading scikit\_learn-0.24.2-cp39-cp39-win\_amd64.whl (7.2 MB)

Installing collected packages: scikit-learn

Successfully installed scikit-learn-0.24.2

**Exercice-13 local and global Variable**

**Q1. Define a global variable**

A global variable is a variable defined outside of any function or block of code. It can be accessed and modified throughout the entire program, including inside functions.

**Q2. Define a local variable**

A local variable is a variable defined inside a function or block of code. It can only be accessed and modified within the scope of that function or block.

**Q3. Write the use of global and nonlocal variable**

Global keyword: Used to declare that a variable inside a function is a global variable. This allows the function to modify the global variable.

Nonlocal keyword: Used to declare that a variable inside a nested function is not local to that function but belongs to the nearest enclosing scope (not the global scope).

**Q4. In a local scope a local variable name is same as global variable name, then which variables hides which other variable.**

If a local variable has the same name as a global variable, the local variable hides the global variable within the local scope. The global variable remains unchanged outside the local scope

**Q5. When UnboundedLocalError is thrown.**

UnboundLocalError is thrown when a local variable is referenced before it is assigned a value within the local scope

**Programs**

**Q1. Write a program using global keyword.**

# Global variable

X = 10

Def modify\_global():

Global x # Declare x as global

X = 20 # Modify the global variable

Modify\_global()

Print(“Global variable x after modification:”, x)

Print(“This program is executed by 22007-CS-018”)

Output:

Global variable x after modification: 20  
This program is executed by 22007-CS-018

**Q2. Write a program using global and variables.**

# Global variable

X = 10

Def local\_scope():

X = 5 # Local variable with the same name as global variable

Print(“Local variable x:”, x)

Local\_scope()

Print(“Global variable x:”, x)

Print(“This program is executed by 22007-CS-018”)

Output:

Local variable x: 5  
Global variable x: 10  
This program is executed by 22007-CS-018

**Q3. Write a program using nonlocal keyword.**

Def outer\_function():

X = 10 # Enclosing scope variable

Def inner\_function():

Nonlocal x # Declare x as nonlocal

X = 20 # Modify the enclosing scope variable

Print(“Inner function x:”, x)

Inner\_function()

Print(“Outer function x:”, x)

Outer\_function()

Print(“This program is executed by 22007-CS-018”)

Output:

Inner function x: 20  
Outer function x: 20  
This program is executed by 22007-CS-018

**Q4. Write a python program where first function swaps the global variable with the local variable and the next function return the cube root of the new global variable value.**

Import math

# Global variable

Global\_var = 8

Def swap\_global\_local():

Local\_var = 27 # Local variable

Global global\_var

Global\_var, local\_var = local\_var, global\_var # Swap values

Print(“After swap – Global variable:”, global\_var, “Local variable:”, local\_var)

Def cube\_root\_global():

Return math.pow(global\_var, 1/3) # Return cube root of global variable

Swap\_global\_local()

Print(“Cube root of the new global variable:”, cube\_root\_global())

Print(“This program is executed by 22007-CS-018”)

Output:

After swap – Global variable: 27 Local variable: 8  
Cube root of the new global variable: 3.0  
This program is executed by 22007-CS-018

**Exercice-14 Math and Datetime Module**

**Q1. List five trigonometric functions of math module**

Math.sin(x) – Returns the sine of x (in radians).

Math.cos(x) – Returns the cosine of x (in radians).

Math.tan(x) – Returns the tangent of x (in radians).

Math.asin(x) – Returns the arc sine of x (in radians).

Math.atan(x) – Returns the arc tangent of x (in radians).

**Q2. List five mathematical functions of math module**

Math.sqrt(x) – Returns the square root of x.

Math.pow(x, y) – Returns x raised to the power of y.

Math.floor(x) – Returns the floor of x (the largest integer less than or equal to x).

Math.ceil(x) – Returns the ceiling of x (the smallest integer greater than or equal to x).

Math.fabs(x) – Returns the absolute value of x

**Q3. List three constants in math module**

Math.pi – The mathematical constant π (pi).

Math.e – The mathematical constant e (Euler’s number).

Math.tau – The mathematical constant τ (tau), which is equal to 2π.

**Q4. What is the use date class?**

The date class in the datetime module is used to represent dates (year, month, and day) without any time information.

**Q5. List five functions of date class**

Date.today() – Returns the current local date.

Date.year – Returns the year of the date.

Date.month – Returns the month of the date.

Date.day – Returns the day of the date.

Date.weekday() – Returns the day of the week as an integer (Monday is 0, Sunday is 6).

**Q6. What is the use datetime class?**

The datetime class in the datetime module is used to represent both dates and times (year, month, day, hour, minute, second, and microsecond).

**Q7. List five functions of datetime class**

Datetime.now() – Returns the current local date and time.

Datetime.year – Returns the year of the datetime.

Datetime.month – Returns the month of the datetime.

Datetime.day – Returns the day of the datetime.

Datetime.strftime(format) – Returns a string representing the datetime in the specified format.

**Q8. What is the use timedelta class?**

The timedelta class in the datetime module is used to represent the difference between two dates or times.

**Q9. List five functions of timedelta**

Timedelta.days – Returns the number of days.

Timedelta.seconds – Returns the number of seconds.

Timedelta.microseconds – Returns the number of microseconds.

Timedelta.total\_seconds() – Returns the total duration in seconds.

Timedelta supports arithmetic operations like addition, subtraction, etc.

**Programs**

**Q1. Write a python program to use 10 mathematical functions.**

Import math

# Using 10 mathematical functions

Print(“Square root of 16:”, math.sqrt(16))

Print(“2 raised to the power of 3:”, math.pow(2, 3))

Print(“Floor of 3.7:”, math.floor(3.7))

Print(“Ceiling of 3.2:”, math.ceil(3.2))

Print(“Absolute value of -5:”, math.fabs(-5))

Print(“Factorial of 5:”, math.factorial(5))

Print(“GCD of 12 and 18:”, math.gcd(12, 18))

Print(“Logarithm of 100 (base 10):”, math.log10(100))

Print(“Sine of 90 degrees:”, math.sin(math.radians(90)))

Print(“Cosine of 60 degrees:”, math.cos(math.radians(60)))

Print(“This program is executed by 22007-CS-018”)

Output:

Square root of 16: 4.0  
2 raised to the power of 3: 8.0  
Floor of 3.7: 3  
Ceiling of 3.2: 4  
Absolute value of -5: 5.0  
Factorial of 5: 120  
GCD of 12 and 18: 6  
Logarithm of 100 (base 10): 2.0  
Sine of 90 degrees: 1.0  
Cosine of 60 degrees: 0.5  
This program is executed by 22007-CS-018

**Q2. Write a python program for given angles in degrees 00, 300,600, 900, 1200, 1800, 2100, 2400, and 3600.**

1. **Calculates the sine and cosine of above angles**

**B. Create a list that contains the converted angles values into radient.**

**C. Finds floor value of sine of above angles**

**D. Finds Ceiling value of cosine of above angles**

**E. Finds absolute value of cosec of above angles**

Import math

Angles = [0, 30, 60, 90, 120, 180, 210, 240, 360]

# A. Calculate sine and cosine

Print(“Sine and Cosine of angles:”)

For angle in angles:

Radians = math.radians(angle)

Print(f”Angle: {angle}°, Sine: {math.sin(radians):.2f}, Cosine: {math.cos(radians):.2f}”)

# B. Convert angles to radians

Radians\_list = [math.radians(angle) for angle in angles]

Print(“Angles in radians:”, radians\_list)

# C. Floor value of sine

Floor\_sine = [math.floor(math.sin(math.radians(angle))) for angle in angles]

Print(“Floor value of sine:”, floor\_sine)

# D. Ceiling value of cosine

Ceil\_cosine = [math.ceil(math.cos(math.radians(angle))) for angle in angles]

Print(“Ceiling value of cosine:”, ceil\_cosine)

# E. Absolute value of cosec

Abs\_cosec = [abs(1 / math.sin(math.radians(angle))) if math.sin(math.radians(angle)) != 0 else “Undefined” for angle in angles]

Print(“Absolute value of cosec:”, abs\_cosec)

Print(“This program is executed by 22007-CS-018”)

Output:

Sine and Cosine of angles:  
Angle: 0°, Sine: 0.00, Cosine: 1.00  
Angle: 30°, Sine: 0.50, Cosine: 0.87  
Angle: 60°, Sine: 0.87, Cosine: 0.50  
Angle: 90°, Sine: 1.00, Cosine: 0.00   
Angle: 120°, Sine: 0.87, Cosine: -0.50  
Angle: 180°, Sine: 0.00, Cosine: -1.00  
Angle: 210°, Sine: -0.50, Cosine: -0.87  
Angle: 240°, Sine: -0.87, Cosine: -0.50  
Angle: 360°, Sine: 0.00, Cosine: 1.00  
Angles in radians: [0.0, 0.5235987755982988, 1.0471975511965976, 1.5707963267948966, 2.0943951023931953, 3.141592653589793, 3.665191429188092, 4.1887902047863905, 6.283185307179586]  
Floor value of sine: [0, 0, 0, 1, 0, 0, -1, -1, 0]  
Ceiling value of cosine: [1, 1, 1, 0, 0, -1, -1, 0, 1]  
Absolute value of cosec: [‘Undefined’, 2.0, 1.1547005383792517, 1.0, 1.1547005383792517, ‘Undefined’, -2.0, -1.1547005383792517, ‘Undefined’]  
This program is executed by 22007-CS-018

**Q3. Write a Python script to display the following**

**a) Current date and time**

**b) Current year**

**c) Month of year**

**d) Week number of the year**

**e) Weekday of the week**

**f) Day of year**

**g) Day of the month**

**h) Day of week**

From datetime import datetime

Now = datetime.now()

Print(“Current date and time:”, now)

Print(“Current year:”, now.year)

Print(“Month of year:”, now.month)

Print(“Week number of the year:”, now.isocalendar()[1])

Print(“Weekday of the week:”, now.strftime(“%A”))

Print(“Day of year:”, now.timetuple().tm\_yday)

Print(“Day of the month:”, now.day)

Print(“Day of week:”, now.weekday()) # Monday is 0, Sunday is 6

Print(“This program is executed by 22007-CS-018”)

Output:

Current date and time: 2023-10-25 14:30:00  
Current year: 2023  
Month of year: 10  
Week number of the year: 43  
Weekday of the week: Wednesday  
Day of year: 298  
Day of the month: 25  
Day of week: 2  
This program is executed by 22007-CS-018

**Q4. Write a Python program to convert a string to datetime.**

**Sample String : Jan 1 2014 2:43PM**

**Expected Output : 2014-07-01 14:43:00**

From datetime import datetime

# Sample string

Date\_string = “Jan 1 2014 2:43PM”

# Convert string to datetime

Date\_object = datetime.strptime(date\_string, “%b %d %Y %I:%M%p”)

Print(“Converted datetime:”, date\_object)

Print(“This program is executed by 22007-CS-018”)

Output:

Converted datetime: 2014-01-01 14:43:00  
This program is executed by 22007-CS-018

**Q5. Write a python that calculates from your date of birth of date**

**1. Number of Year you lived**

**2. Number of Months you lived**

**3. Number of days you lived**

From datetime import datetime

# Input date of birth

Dob\_input = input(“Enter your date of birth (YYYY-MM-DD): “)

Dob = datetime.strptime(dob\_input, “%Y-%m-%d”)

# Current date

Now = datetime.now()

# Calculate difference

Delta = now – dob

# Extract years, months, and days

Years = delta.days // 365

Remaining\_days = delta.days % 365

Months = remaining\_days // 30

Days = remaining\_days % 30

Print(f”You have lived: {years} years, {months} months, and {days} days.”)

Print(“This program is executed by 22007-CS-018”)

Output:  
Enter your date of birth (YYYY-MM-DD): 1990-05-15  
You have lived: 33 years, 5 months, and 10 days.  
This program is executed by 22007-CS-018

**Q6. Write a python that calculates number of days to your next birthday.**

From datetime import datetime

# Input date of birth

Dob\_input = input(“Enter your date of birth (YYYY-MM-DD): “)

Dob = datetime.strptime(dob\_input, “%Y-%m-%d”)

# Current date

Now = datetime.now()

# Next birthday

Next\_birthday = datetime(now.year, dob.month, dob.day)

# If birthday has already passed this year, set it to next year

If next\_birthday < now:

Next\_birthday = datetime(now.year + 1, dob.month, dob.day)

# Calculate difference

Delta = next\_birthday – now

Print(f”Days until your next birthday: {delta.days}”)

Print(“This program is executed by 22007-CS-018”)

Output:

Enter your date of birth (YYYY-MM-DD): 1990-05-15  
Days until your next birthday: 202  
This program is executed by 22007-CS-018

**Q7. Write a python that calculates on which week day your next birthday will happen**

From datetime import datetime

# Input date of birth

Dob\_input = input(“Enter your date of birth (YYYY-MM-DD): “)

Dob = datetime.strptime(dob\_input, “%Y-%m-%d”)

# Current date

Now = datetime.now()

# Next birthday

Next\_birthday = datetime(now.year, dob.month, dob.day)

# If birthday has already passed this year, set it to next year

If next\_birthday < now:

Next\_birthday = datetime(now.year + 1, dob.month, dob.day)

# Get weekday

Weekday = next\_birthday.strftime(“%A”)

Print(f”Your next birthday will be on a {weekday}.”)

Print(“This program is executed by 22007-CS-018”)

Output:

Enter your date of birth (YYYY-MM-DD): 1990-05-15  
Your next birthday will be on a Monday.  
This program is executed by 22007-CS-018

**Q8. Write that calculates the retirement age of person. A person retires from the service if he attains the age of 58 or 26 years into the service whichever is earlier. Give the following input from the keyboard**

**1. Date of Birth**

**2. Date of Joining the Service**

From datetime import datetime

# Input date of birth and joining date

Dob\_input = input(“Enter your date of birth (YYYY-MM-DD): “)

Dob = datetime.strptime(dob\_input, “%Y-%m-%d”)

Join\_input = input(“Enter your date of joining (YYYY-MM-DD): “)

Join\_date = datetime.strptime(join\_input, “%Y-%m-%d”)

# Retirement conditions

Retirement\_age = 58

Service\_years = 26

# Calculate retirement date based on age

Retirement\_by\_age = datetime(dob.year + retirement\_age, dob.month, dob.day)

# Calculate retirement date based on service years

Retirement\_by\_service = datetime(join\_date.year + service\_years, join\_date.month, join\_date.day)

# Determine which retirement date is earlier

Retirement\_date = min(retirement\_by\_age, retirement\_by\_service)

Print(f”Your retirement date is: {retirement\_date.strftime(‘%Y-%m-%d’)}”)

Print(“This program is executed by 22007-CS-018”)

Output:

Enter your date of birth (YYYY-MM-DD): 1985-08-20  
Enter your date of joining (YYYY-MM-DD): 2010-06-15  
Your retirement date is: 2043-06-15  
This program is executed by 22007-CS-018

**Q9. If a Student joins the College on X day and leaves the college on Y day. Calculate the Numbers of years, months and days a student studies in the college.**

From datetime import datetime

# Input joining and leaving dates

Join\_input = input(“Enter the joining date (YYYY-MM-DD): “)

Join\_date = datetime.strptime(join\_input, “%Y-%m-%d”)

Leave\_input = input(“Enter the leaving date (YYYY-MM-DD): “)

Leave\_date = datetime.strptime(leave\_input, “%Y-%m-%d”)

# Calculate difference

Delta = leave\_date – join\_date

# Extract years, months, and days

Years = delta.days // 365

Remaining\_days = delta.days % 365

Months = remaining\_days // 30

Days = remaining\_days % 30

Print(f”The student studied for: {years} years, {months} months, and {days} days.”)

Print(“This program is executed by 22007-CS-018”)

Output:

Enter the joining date (YYYY-MM-DD): 2018-09-01  
Enter the leaving date (YYYY-MM-DD): 2023-05-15  
The student studied for: 4 years, 8 months, and 14 days.  
This program is executed by 22007-CS-018

**15-Exception Handling**

**Q1. Define Exception**

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. It is typically an error that occurs at runtime.

**Q2. Define Exception Handling**

Exception handling is the process of responding to exceptions in a program. It allows the program to continue executing even after encountering an error, by providing a way to handle the error gracefully.

**Q3. List the keywords related to handle exceptions**

The keywords related to exception handling in Python are:

Try , Except, Finally, Raise, Else

**Q4. Write the syntax for exception handling**

Try:

# Code that may raise an exception

Except ExceptionType:

# Code to handle the exception

Finally:

# Code that will always execute

**Q5. List five predefined exceptions**

ZeroDivisionError

ValueError

NameError

TypeError

IndexError

**Q6. Write the syntax to create a user-defined exception**

Class CustomError(Exception):

Pass

**Q7. Write the use of raise keyword**

The raise keyword is used to explicitly throw an exception in Python. It can be used to raise built-in exceptions or user-defined exceptions.

**Q8. Write the use of following blocks**

1. **Try b. except c. error d. finally**

Try: The try block contains the code that may raise an exception.

Except: The except block contains the code that handles the exception.

Error: This is not a block but refers to the type of exception being handled.

Finally: The finally block contains code that will always execute, regardless of whether an exception occurred or not.

**Programs**

**Q1. Write a program to catch ZeroDivisionError, ValueError, and NameError in three separate except blocks.**

Try:

Num = int(input(“Enter a number: “))

Result = 10 / num

Print(“Result:”, result)

Print(variable) # This will raise NameError

Except ZeroDivisionError:

Print(“Error: Division by zero is not allowed.”)

Except ValueError:

Print(“Error: Invalid input. Please enter a valid number.”)

Except NameError:

Print(“Error: Variable not defined.”)

Print(“This program is executed by 22007-CS-018”)

Output:

Enter a number: 0  
Error: Division by zero is not allowed.  
This program is executed by 22007-CS-018

**Q2. Write a program to catch ZeroDivisionError, ValueError, and NameError and handle all these exceptions in the same manner.**

Try:

Num = int(input(“Enter a number: “))

Result = 10 / num

Print(“Result:”, result)

Print(variable) # This will raise NameError

Except (ZeroDivisionError, ValueError, NameError):

Print(“An error occurred. Please check your input.”)

Print(“This program is executed by 22007-CS-018”)

Output:  
Enter a number: abc  
An error occurred. Please check your input.  
This program is executed by 22007-CS-018

**Q3. Write a program with error block and finally block.**

Try:

Num = int(input(“Enter a number: “))

Result = 10 / num

Print(“Result:”, result)

Except ZeroDivisionError:

Print(“Error: Division by zero is not allowed.”)

Finally:

Print(“This block will always execute.”)

Print(“This program is executed by 22007-CS-018”)

Output:  
Enter a number: 0  
Error: Division by zero is not allowed.  
This block will always execute.  
This program is executed by 22007-CS-018

**Q4. A, B, and C are Exception classes where class C inherits B, which in turn inherited from A. Write a Python script that handles exceptions in the order A, B, and C and another Python script that handles exceptions in the order C, B, and A.**

**Program 1: Handling exceptions in the order A, B, and C**

Class A(Exception):

Pass

Class B(A):

Pass

Class C(B):

Pass

Try:

Raise C()

Except A:

Print(“Caught exception of type A”)

Except B:

Print(“Caught exception of type B”)

Except C:

Print(“Caught exception of type C”)

Print(“This program is executed by 22007-CS-018”)

Output:

Caught exception of type A  
This program is executed by 22007-CS-018

**Program 2: Handling exceptions in the order C, B, and A**

Class A(Exception):

Pass

Class B(A):

Pass

Class C(B):

Pass

Try:

Raise C()

Except C:

Print(“Caught exception of type C”)

Except B:

Print(“Caught exception of type B”)

Except A:

Print(“Caught exception of type A”)

Print(“This program is executed by 22007-CS-018”)

Output:

Caught exception of type C  
This program is executed by 22007-CS-018

**Q5. Write a Python program where try block is given inside another try block.**

Try:

Num = int(input(“Enter a number: “))

Try:

Result = 10 / num

Print(“Result:”, result)

Except ZeroDivisionError:

Print(“Error: Division by zero is not allowed.”)

Except ValueError:

Print(“Error: Invalid input. Please enter a valid number.”)

Print(“This program is executed by 22007-CS-018”)

Output:

Enter a number: 0  
Error: Division by zero is not allowed.  
This program is executed by 22007-CS-018

**Q6. Write a Python program to raise an exception.**

Try:

Age = int(input(“Enter your age: “))

If age < 0:

Raise ValueError(“Age cannot be negative.”)

Print(“Your age is:”, age)

Except ValueError as e:

Print(“Error:”, e)

Print(“This program is executed by 22007-CS-018”)

Output:

Enter your age: -5  
Error: Age cannot be negative.  
This program is executed by 22007-CS-018

**Q7. Create a Game called GuessTheCorrectNumber**

Class NumberTooSmallError(Exception):

Pass

Class NumberTooBigError(Exception):

Pass

Correct\_number = 50

While True:

Try:

Guess = int(input(“Guess the number: “))

If guess < correct\_number:

Raise NumberTooSmallError(“Your guess is too small.”)

Elif guess > correct\_number:

Raise NumberTooBigError(“Your guess is too big.”)

Else:

Print(“Congratulations! You guessed the correct number.”)

Break

Except NumberTooSmallError as e:

Print (e)

Except NumberTooBigError as e:

Print (e)

Print(“This program is executed by 22007-CS-018”)

Output:

Guess the number: 30  
Your guess is too small.  
Guess the number: 60  
Your guess is too big.  
Guess the number: 50  
Congratulations! You guessed the correct number.  
This program is executed by 22007-CS-018

**16-Multithreading**

**Q1. Define Multithreading**

Multithreading is a programming concept where multiple threads of execution run concurrently within a single process. Each thread can perform a separate task, allowing for parallel execution and improved performance in certain scenarios.

**Q2. Define Thread**

A thread is the smallest unit of execution within a process. It is a sequence of instructions that can be scheduled independently by the operating system. Multiple threads within a process share the same memory space.

**Q3. List the different ways of creating a thread in Python**

Using the threading.Thread class and passing a target function.

By subclassing the threading.Thread class and overriding the run() method.

**Q4. Write the use of the following methods of the Thread class**

1. Start(): Starts the thread’s activity by invoking the run() method.
2. Run(): Contains the code that is executed when the thread starts. It can be overridden in a subclass.

**Q5. List the advantages of using threads**

Improved performance for I/O-bound tasks.

Efficient utilization of CPU resources.

Simplified modeling of concurrent tasks.

Responsive user interfaces in GUI applications.

**Q6. List the disadvantages of using threads**

Complexity in debugging and testing.

Potential for race conditions and deadlocks.

Increased memory usage.

Overhead of thread creation and context switching.

**Q7. Define Synchronization**

Synchronization is the coordination of multiple threads to ensure that they access shared resources in a controlled manner, preventing race conditions and ensuring data consistency.

**Q8. List the problems which cannot be solved by synchronization**

Deadlocks.Live locks. Starvation. Priority inversion.

**Q9. Define Inter-thread Communication**

Inter-thread communication is a mechanism that allows threads to communicate with each other, typically through shared variables or message passing, to coordinate their activities.

**Q10. List five methods of the Condition class**

Wait(): Releases the lock and waits until notified.

Notify(): Wakes up one thread waiting on the condition.

Notify\_all(): Wakes up all threads waiting on the condition.

Acquire(): Acquires the underlying lock.

Release(): Releases the underlying lock.

**Q11. Write the use of following methods of condition class**

1. **Wait b. notify c. acquire d. release**
2. Wait(): Releases the lock and waits until another thread notifies it.
3. Notify(): Wakes up one thread that is waiting on the condition.
4. Acquire(): Acquires the lock associated with the condition.
5. Release(): Releases the lock associated with the condition.

**Programs**

**Q1. Create three threads where each thread is implemented with three functions.**

**First thread calculates the square of numbers from 1 to 10.**

**Second thread calculates the square root of numbers from 11 to 20.**

**Third thread calculates the cube root of numbers from 21 to 30.**

Import threading

Import math

Def square\_numbers():

For I in range(1, 11):

Print(f”Square of {i} is {i\*i}”)

Print(“This program is executed by 22007-CS-018”)

Def square\_root\_numbers():

For I in range(11, 21):

Print(f”Square root of {i} is {math.sqrt(i):.2f}”)

Print(“This program is executed by 22007-CS-018”)

Def cube\_root\_numbers():

For I in range(21, 31):

Print(f”Cube root of {i} is {i\*\*(1/3):.2f}”)

Print(“This program is executed by 22007-CS-018”)

# Create threads

Thread1 = threading.Thread(target=square\_numbers)

Thread2 = threading.Thread(target=square\_root\_numbers)

Thread3 = threading.Thread(target=cube\_root\_numbers)

# Start threads

Thread1.start()

Thread2.start()

Thread3.start()

# Wait for all threads to finish

Thread1.join()

Thread2.join()

Thread3.join()

Output:

Square of 1 is 1  
Square of 2 is 4  
Square of 3 is 9  
Square of 4 is 16  
Square of 5 is 25  
Square of 6 is 36  
Square of 7 is 49  
Square of 8 is 64  
Square of 9 is 81  
Square of 10 is 100  
This program is executed by 22007-CS-018  
Square root of 11 is 3.32  
Square root of 12 is 3.46  
Square root of 13 is 3.61  
Square root of 14 is 3.74  
Square root of 15 is 3.87  
Square root of 16 is 4.00  
Square root of 17 is 4.12  
Square root of 18 is 4.24  
Square root of 19 is 4.36  
Square root of 20 is 4.47  
This program is executed by 22007-CS-018  
Cube root of 21 is 2.76  
Cube root of 22 is 2.80  
Cube root of 23 is 2.84  
Cube root of 24 is 2.88  
Cube root of 25 is 2.92  
Cube root of 26 is 2.96  
Cube root of 27 is 3.00  
Cube root of 28 is 3.04  
Cube root of 29 is 3.07  
Cube root of 30 is 3.11  
This program is executed by 22007-CS-018

**Q2. Create three threads where each thread is implemented with three functions in a class.**

**First thread calculates the square of numbers from 1 to 10.**

**Second thread calculates the square root of numbers from 11 to 20.**

**Third thread calculates the cube root of numbers from 21 to 30.**

Import threading

Import math

Class SquareThread(threading.Thread):

Def run(self):

For I in range(1, 11):

Print(f”Square of {i} is {i\*i}”)

Print(“This program is executed by 22007-CS-018”)

Class SquareRootThread(threading.Thread):

Def run(self):

For I in range(11, 21):

Print(f”Square root of {i} is {math.sqrt(i):.2f}”)

Print(“This program is executed by 22007-CS-018”)

Class CubeRootThread(threading.Thread):

Def run(self):

For I in range(21, 31):

Print(f”Cube root of {i} is {i\*\*(1/3):.2f}”)

Print(“This program is executed by 22007-CS-018”)

# Create threads

Thread1 = SquareThread()

Thread2 = SquareRootThread()

Thread3 = CubeRootThread()

# Start threads

Thread1.start()

Thread2.start()

Thread3.start()

# Wait for all threads to finish

Thread1.join()

Thread2.join()

Thread3.join()

Output**:**

Square of 1 is 1  
Square of 2 is 4  
Square of 3 is 9  
Square of 4 is 16  
Square of 5 is 25  
Square of 6 is 36  
Square of 7 is 49  
Square of 8 is 64  
Square of 9 is 81  
Square of 10 is 100  
This program is executed by 22007-CS-018  
Square root of 11 is 3.32  
Square root of 12 is 3.46  
Square root of 13 is 3.61  
Square root of 14 is 3.74  
Square root of 15 is 3.87  
Square root of 16 is 4.00  
Square root of 17 is 4.12  
Square root of 18 is 4.24  
Square root of 19 is 4.36  
Square root of 20 is 4.47  
This program is executed by 22007-CS-018  
Cube root of 21 is 2.76  
Cube root of 22 is 2.80  
Cube root of 23 is 2.84  
Cube root of 24 is 2.88  
Cube root of 25 is 2.92  
Cube root of 26 is 2.96  
Cube root of 27 is 3.00  
Cube root of 28 is 3.04  
Cube root of 29 is 3.07  
Cube root of 30 is 3.11  
This program is executed by 22007-CS-018

**Q3. Write a python program to avoid race condition.**

Import threading

# Shared resource

Counter = 0

Lock = threading.Lock()

Def increment\_counter():

Global counter

For \_ in range(100000):

With lock:

Counter += 1

# Create threads

Thread1 = threading.Thread(target=increment\_counter)

Thread2 = threading.Thread(target=increment\_counter)

# Start threads

Thread1.start()

Thread2.start()

# Wait for threads to finish

Thread1.join()

Thread2.join()

Print(f”Final counter value: {counter}”)

Print(“This program is executed by 22007-CS-018”)

Output:

Final counter value: 200000  
This program is executed by 22007-CS-018

**Q4. Write a python program to give the solution for producer-consumer problem in the multi-threaded environment.**

Import threading

Import time

Import random

# Shared buffer

Buffer = []

Buffer\_size = 5

Lock = threading.Lock()

Condition = threading.Condition(lock)

Def producer():

For I in range(10):

With condition:

While len(buffer) == buffer\_size:

Condition.wait()

Item = random.randint(1, 100)

Buffer.append(item)

Print(f”Produced: {item}”)

Condition.notify()

Time.sleep(random.random())

Def consumer():

For I in range(10):

With condition:

While len(buffer) == 0:

Condition.wait()

Item = buffer.pop(0)

Print(f”Consumed: {item}”)

Condition.notify()

Time.sleep(random.random())

# Create threads

Producer\_thread = threading.Thread(target=producer)

Consumer\_thread = threading.Thread(target=consumer)

# Start threads

Producer\_thread.start()

Consumer\_thread.start()

# Wait for threads to finish

Producer\_thread.join()

Consumer\_thread.join()

Print(“This program is executed by 22007-CS-018”)

Output:

Produced: 42  
Consumed: 42  
Produced: 87  
Consumed: 87  
Produced: 15  
Consumed: 15  
Produced: 63  
Consumed: 63  
Produced: 29  
Consumed: 29  
Produced: 91  
Consumed: 91  
Produced: 50  
Consumed: 50  
Produced: 78  
Consumed: 78  
Produced: 36  
Consumed: 36  
Produced: 22  
Consumed: 22  
This program is executed by 22007-CS-018

**Q5. Write a python program to provide synchronize access to the global variable balance representing the balance in your father bank account where three threads namely father, mother and you trying to depositing an amount, checking the balance and you performing the withdrawal through an ATM respectively.**

Import threading

# Shared resource

Balance = 1000

Lock = threading.Lock()

Def deposit(amount):

Global balance

With lock:

Balance += amount

Print(f”Deposited {amount}. New balance: {balance}”)

Def check\_balance():

Global balance

With lock:

Print(f”Current balance: {balance}”)

Def withdraw(amount):

Global balance

With lock:

If balance >= amount:

Balance -= amount

Print(f”Withdrew {amount}. New balance: {balance}”)

Else:

Print(“Insufficient balance”)

# Create threads

Father\_thread = threading.Thread(target=deposit, args=(500,))

Mother\_thread = threading.Thread(target=check\_balance)

You\_thread = threading.Thread(target=withdraw, args=(200,))

# Start threads

Father\_thread.start()

Mother\_thread.start()

You\_thread.start()

# Wait for threads to finish

Father\_thread.join()

Mother\_thread.join()

You\_thread.join()

Print(“This program is executed by 22007-CS-018”)

Output:

Deposited 500. New balance: 1500  
Current balance: 1500  
Withdrew 200. New balance: 1300  
This program is executed by 22007-CS-018