ASSIGNMENT 17.1

Task 1
Write a simple program to show inheritance in scala.

Solution#1

The program defines two classes below. The base class Employee which has name and salary as member variables. Class engineer is a derived class which extends(inherits)the properties of base class Employee. The inherited class Engineer will have properties of the base class (name,salary) and will also extend the property to add another property called bonus. Also in the example below it shows that derived class can override the method of the base class.

```
class Employee(val name: String, val salary: Int) {


  def what_is_your_type = println("I am an employee")

}

class Engineer(name: String, salary: Int) extends Employee(name, salary) {

  val bonus: Int = 1000

  override def what_is_your_type = println("I am an engineer")


}
```

We instantiate these two classes in the main method and run it, The output is shown below

```
def main(args: Array[String]): Unit = {

  val a: Employee = new Employee("Ashok", 20000)
  println("Name="+a.name+" salary="+a.salary)
  a.what_is_your_type
  val b: Engineer = new Engineer("Mohit", 40000)
  println("Name="+b.name+" salary="+b.salary + " bonus="+b.bonus)
  b.what_is_your_type


}
```

Following is the output of the program

```
Name=Ashok salary=20000
I am an employee
Name=Mohit salary=40000 bonus=1000
I am an engineer
```

We see that derived class inherits the properties of the base class (name,salary) and also extends it to add bonus as property. We can see that we can print those values from the derived class. Secondly we also see that calling what_is_your_type method of the derived class prints "I am an Engineer" which means the method of the same name in base class was overridden by the derived class.

Task 2
Write a simple program to show multiple inheritance in scala

Solution#2
Multiple inheritance where a class derives its properties from more than one classes can be achieved using 'traits' in Scala. Traits are unusual in that a class can incorporate as many of them as desired, like java interfaces, but they can also contain behavior, like classes.
In some way we can assume trait as abstract class with the additional property that is used in multiple inheritance. In case of multiple inheritance if the same method exists in the
Also, like both classes and interfaces, traits can introduce new methods. But unlike either, the definition of that behavior isn't checked until the traitis actually incorporated as part of a class.
In case of same method name between various traits, the one which comes after
'**with**' gets precedence

We define a base class 'bird' below with a method named as sound.
We define two trait classes which extend the properties of the base class 'bird'.
Class 'flyingbird' defines additional method called 'canfly' and class 'runningbird' defines additional method called 'canrun'. Both traits override the sound method of the base class to demonstrate which one gets called later.
Finally we define a class called 'flyingrunningbird' which extends from both the above traits.

```
class bird{
  def sound=println("sound of bird")
}
trait  flyingbird extends bird
{

  override def sound= println("sound of the flying bird")
  def canfly=println("I am always flying")

}

trait  runningbird extends bird
{

  override def sound= println("sound of the running bird")
  def canrun=println("I can run")
}

class flyingrunningbird extends flyingbird with runningbird
{

}
```

```
def main(args: Array[String]): Unit = {


  val x:flyingrunningbird = new flyingrunningbird
  x.sound
  x.canfly
  x.canrun

}
```

The output of the program is below. First line of output shows that 'sound' method of the 'runningbird' gets called and takes precedence as it comes after the with keyword. Second and third line of the output show that class 'flyingrunningbird' has properties of both the traits it extends from

```
sound of the running bird
I am always flying
I can run
```

Task 3
Write a partial function to add three numbers in which one number is constant and two numbers can be passed as inputs and define another method which can take the partial function as input and squares the result.

Solution#3

```
def main(args: Array[String]): Unit = {


  // define the sum function of 3 numbers
  def sum(x:Int,y:Int,z:Int)=x+y+z

  //define a partially applied function where first parameter is fixed to value 1
  def fixsum = sum(1,_:Int,_:Int)

  // define another function which takes partial function as input and sqaures the
//result
  def sumsquare(f:(Int,Int)=>Int,x:Int,y:Int)= f(x,y)*f(x,y)


  println("the fixsum(1,5) is " + fixsum(1,5))
  println("the sumsquare(1,5) is " + sumsquare(fixsum,1,5))

}
```

Below is the output of the program

```
the fixsum(1,5) is 7
the sumsquare(1,5) is 49
```

Task 4
Write a program to print the prices of 4 courses of Acadgild:
Android App Development -14,999 INR
Data Science - 49,999 INR
Big Data Hadoop & Spark Developer – 24,999 INR
Blockchain Certification – 49,999 INR
using match and add a default condition if the user enters any other course.

Solution#4

Below is the code for the above task. We define a function to print the prices based on the provide course name

```
def print_course_fee(x: String) = {
  x match {

    case "Android App Development" => println("14999 INR")
    case "Data Science" => println("49,999 INR")
    case "Big Data Hadoop & Spark Developer" => println("24,999 INR")
    case "Blockchain Certification" => println("49,999 INR")
    case _ => println("Invalid course value")

  }


}
```