Practicals of Mobile Application Development using Flutter
Assignment 1

1. **Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5, between 2000 and 3200 (both included).**

```
void main() {
  for (int i = 2000; i <= 3200; i++) {
    if (i % 7 == 0 && i % 5 != 0) {
      print(i);
    }
  }
}
```

output :

2002

2009

2016

2023

2037

2044

2051

2058

2072

2079

2086

.

.

.

3192

3199

2. Write a program to check if a number is a prime number.

```
is_prime(int number) {
  if (number <= 1) {
    return false;
  }
  for (int i = 2; i < number; i++) {
    if (number % i == 0) {
      return false;
    }
  }
  return true;
}
void main() {
  int number = 4;
  if (is_prime(number)) {
    print("$number is a prime number.");
  } else {
    print("$number is not a prime number.");
  }
}
```

output :

4 is not a prime number.

3. **Write a program that accepts a sentence and calculate the number of letters and digits. Suppose the following input is supplied to the**
   Program: hello world! 123 Then, the output should be: LETTERS 10 DIGITS 3

```
import 'dart:io';
void main() {
  print("Enter a sentence:");
  String input = stdin.readLineSync()!;
  int letters = 0;
  int digits = 0;
  for (int i = 0; i < input.length; i++) {
    int code = input.codeUnitAt(i);
    // A–Z or a–z
    if ((code >= 65 && code <= 90) || (code >= 97 && code <= 122)) {
      letters++;
    }
    // 0–9
    else if (code >= 48 && code <= 57) {
      digits++;
    }
  }
  print("LETTERS $letters");
  print("DIGITS $digits");
}
```

output :

```
Enter a sentence:
hello world! 123
LETTERS 10
DIGITS 3
```

4. **Write a program to calculate squares of even numbers between a given range (like 1 to 30)**

```
import 'dart:io';

void main() {
  print("Enter the range (start and end):");
  int start = int.parse(stdin.readLineSync()!);
  int end = int.parse(stdin.readLineSync()!);
  for (int i = start; i < end; i++) {
    if (i % 2 == 0) {
      print("${i * i}");
    }
  }
}
```

output:

Enter the range (start and end):

1

5

4

16

5. Write a Dart program to accept age input from the user.

If the entered age is less than 18, throw a custom exception.

Properly handle both FormatException (invalid input) and the

custom exception using try-catch.

```
import 'dart:io';

void main() {
  try {
    int age;
    print("Enter your age:");
    age = int.parse(stdin.readLineSync()!);

    if (age < 18) {
      throw AgeException('Age is less than 18');
    } else {
      print("You are eligible.");
    }
  } on FormatException {
    print("Invalid input! Please enter a valid number.");
  } on AgeException catch (e) {
    print(e);
  }
}

class AgeException implements Exception {
  String message;

  AgeException(this.message);
```

```
  @override
  String toString() {
    return "AgeException: $message";
  }
}
```

Output 1:

Enter your age:

17

AgeException: Age is less than 18

Output 2:

Enter your age:

abc

Invalid input! Please enter a valid number.

6. **Create a Dart program to implement a To-Do List application using a List.**

The program should support the following operations:

- **Add new tasks**
- **Remove existing tasks**
- **Mark tasks as completed**
- **Display pending tasks and completed tasks separately**

```
import 'dart:io';

void main() {
  // Create a To-Do List
  List<String> todos = [];
  List<String> completedTodos = [];

  // Choices to be implemented
  // 1. Add new tasks
  // 2. Remove existing tasks
  // 3. Mark tasks as completed
  // 4. Display pending tasks and completed tasks separately

  while (true) {
    print("\nTo-Do List Application");
    print("1. Add new tasks");
    print("2. Remove existing tasks");
    print("3. Mark tasks as completed");
    print("4. Display pending tasks and completed tasks separately");
    print("5. Exit");

    print("Enter your choice: ");
```

```
int choice = int.parse(stdin.readLineSync()!);
switch (choice) {
  case 1:
    String task;
    print("Enter the task to add: ");
    task = stdin.readLineSync()!;
    if (task.isEmpty) {
      print("Task cannot be empty");
    } else {
      todos.add(task);
    }
    break;
  case 2:
    print("Enter the task number to remove from pending tasks: ");
    int taskNum = int.parse(stdin.readLineSync()!);
    if (taskNum < 1 || taskNum > todos.length) {
      print("Invalid task number");
    } else {
      todos.removeAt(taskNum - 1);
    }
    break;
  case 3:
    print("Enter the task number to mark as completed: ");
    int taskNum = int.parse(stdin.readLineSync()!);
    if (taskNum < 1 || taskNum > todos.length) {
      print("Invalid task number");
    } else {
      String completedTask = todos.removeAt(taskNum - 1);
```

```
        completedTodos.add(completedTask);
      }
      break;
    case 4:
      print("\nPending Tasks:");
      if (todos.isEmpty) {
        print("No pending tasks");
      } else {
        for (var i = 0; i < todos.length; i++) {
          print("${i + 1}. ${todos[i]}");
        }
      }
      print("\nCompleted Tasks:");
      if (completedTodos.isEmpty) {
        print("No completed tasks");
      } else {
        for (var i = 0; i < completedTodos.length; i++) {
          print("${i + 1}. ${completedTodos[i]}");
        }
      }
      break;
    case 5:
      exit(0);
    default:
      print("Invalid choice");
  }
 }
}
```

7. Develop a Dart program to simulate basic bank account operations.

Create a BankAccount class that uses encapsulation to protect

account balance.

Implement the following methods:

● deposit() to add amount to the balance

● withdraw() to deduct amount (handle insufficient balance condition)

● checkBalance() to display the current balance

● Ensure the balance variable is declared as private.

```
import 'dart:io';


class BankAccount {
  // encapsulated private balance variable
  double _balance = 0.0; // private variable

  // Method to check current balance
  void checkBalance() {
    print("Current Balance: ₹${_balance}");
  }

  // Method to deposit amount
  void deposit(double amount) {
    if (amount > 0) {
      _balance += amount;
      print("Deposited: ₹${amount}");
      checkBalance();
    } else {
      print("Deposit amount must be positive.");
    }
```

```
  }

  // Method to withdraw amount
  void withdraw(double amount) {
    if (amount > 0) {
      if (amount <= _balance) {
        _balance -= amount;
        print("Withdrawn: ₹${amount}");
        checkBalance();
      } else {
        print("Insufficient balance.");
      }
    } else {
      print("Withdrawal amount must be positive.");
    }
  }
}

void main() {
  BankAccount account = BankAccount();

  while (true) {
    print("Bank Account Operations:");
    print("1. Deposit");
    print("2. Withdraw");
    print("3. Check Balance");
    print("4. Exit");
```

```
    print("Enter your choice: ");
    int choice = int.parse(stdin.readLineSync()!);

    switch (choice) {
      case 1:
        print("Enter amount to deposit: ");
        double depositAmount = double.parse(stdin.readLineSync()!);
        account.deposit(depositAmount);
        break;
      case 2:
        print("Enter amount to withdraw: ");
        double withdrawAmount = double.parse(stdin.readLineSync()!);
        account.withdraw(withdrawAmount);
        break;
      case 3:
        account.checkBalance();
        break;
      case 4:
        exit(0);
      default:
        print("Invalid choice.");
    }
  }
}
```

8. **Create a Dart program to calculate salaries for different types of employees.**

● Define a base class Employee and derive two subclasses: PermanentEmployee ContractEmployee

● Override the calculateSalary() method in each subclass to compute salary according to the employee type.

```
class Employee {
  double calculateSalary() {
    return 0.0;
  }
}

class PermanentEmployee extends Employee {
  double basicSalary;
  double hra; // House Rent Allowance
  double pf; // Provident Fund

  PermanentEmployee(this.basicSalary, this.hra, this.pf);

  @override
  double calculateSalary() {
    return basicSalary + hra - pf;
  }
}

class ContractEmployee extends Employee {
  double hourlyRate;
  int hoursWorked;

  ContractEmployee(this.hourlyRate, this.hoursWorked);
```

```
  @override
  double calculateSalary() {
    return hourlyRate * hoursWorked;
  }
}

void main() {
  PermanentEmployee permEmp = PermanentEmployee(50000, 10000, 5000);
  ContractEmployee contractEmp = ContractEmployee(200, 160);

  print("Permanent Employee Salary: ₹${permEmp.calculateSalary()}");
  print("Contract Employee Salary: ₹${contractEmp.calculateSalary()}");
}
```

9. **Given Code**

```
class Student {
int id;
String name;
Student(int id, String name) {
id = id;
name = name;
}
}
void main() {
Student s = Student(1, "Rahul");
print(s.name);
}
```

Tasks -

- **Does the code produce output or error?**
- **If error → identify and rectify it**
- **Add functionality to:**
  - **Display both id and name**
  - **Add a method display()**

```
class Student {
  int? id;
  String? name;

  Student(int id, String name) {
    this.id = id;
    this.name = name;
  }
```

```
  void display() {
    print("ID: $id, Name: $name");
  }
}

void main() {
  Student s = Student(1, "Rahul");
  print(s.name);
  s.display();
}
```

1. Does the code produce output or error?

The original code produces an error because the constructor parameters `id` and `name` are shadowing the instance variables. As a result, the instance variables are not being initialized properly.

2. If error → identify and rectify it

The error can be rectified by using `this` keyword to refer to the instance variables inside the constructor.

3. Add functionality to:

- Display both id and name
- Add a method display()

10. Given Code

```
void main() {
int marks = 85;
if (marks > 90) {
print("A");
} else if (marks > 75) {
print("B");
} else {
print("C");
}
}
```

Tasks-

- **Predict output**
- **Identify logical issue**
- **Fix grading logic**
- **Add functionality: Accept marks from user.**

```
import 'dart:io';

void main() {
  print("Enter your marks: ");
  int? marks = int.parse(stdin.readLineSync()!);

  if (marks > 90 && marks <= 100) {
    print("A");
  } else if (marks > 75 && marks <= 90) {
    print("B");
  } else {
```

```
    print("C");
  }
}
```

1. Predict output

The original code will output "B" for marks = 85.

2. Identify logical issue

The logical issue is that the grading logic does not handle marks greater than 100 or less than 0.

3. Fix grading logic

The grading logic has been fixed to include checks for valid marks (0-100).

4. Add functionality: Accept marks from user.

The code now accepts marks as input from the user.

11. Given Code -

```
class Product {
String name;
double price;
Product(this.name, this.price);
}
void main() {
Product p = Product("Laptop", -50000);
print(p.price);
}
```

Tasks -

- **Is this logically correct?**
- **Add validation using exception handling**
- **Add functionality: Apply 10% discount if price > 30000**

```
class Product {
  String? name;
  double? price = 0.0;

  Product(name, price) {
    if (price < 0) {
      throw Exception("Price cannot be negative");
    }
    this.name = name;
    this.price = price;
  }

  // Apply 10% discount if price > 30000
  void applyDiscount() {
```

```
    if (price! > 30000) {
      price = price! * 0.9;
    }
  }
}

void main() {
  try {
    Product p = Product("Laptop", 40000);
    p.applyDiscount();
    print(p.price);
  } catch (e) {
    print(e);
  }
}
```

1. Is this logically correct?

The original code is not logically correct because it allows the creation of a Product with a negative price.

2. Add validation using exception handling

The constructor has been modified to throw an exception if the price is negative.

3. Add functionality: Apply 10% discount if price > 30000

A method `applyDiscount` has been added to the Product class to apply a 10% discount if the price is greater than 30000.

12. Given Code -

```
class Person {
String name;
Person(this.name);
}
void main() {
Person p1 = Person("Amit");
Person p2 = Person("Amit");
print(p1 == p2);
}
```

Tasks-

- **Predict output**
- **Explain why**
- **Override equality operator**
- **Add functionality: compare by name**

```
class Person {
  String name;
  Person(this.name);
}

void main() {
  Person p1 = Person("Amit");
  Person p2 = Person("Amit");
  print(p1 == p2);
  // Override equality operator
  bool areEqual = p1.name == p2.name;
  print(areEqual);
```

```
  // Add functionality: compare by name
  if (p1.name == p2.name) {
    print("Both persons have the same name.");
  } else {
    print("Persons have different names.");
  }
}

/**
  1. Predict output
    The output will be `false`.
  2. Explain why
   In Dart, the `==` operator checks for reference equality by default, meaning it checks whether both references point to the same object in memory. Since `p1` and `p2` are two different instances of the `Person` class, even though they have the same name, they are not the same object, hence the output is `false`.
```

13. Given Code -

```
void main() {
int day = 1;
switch (day) {
case 1:
print("Monday");
case 2:
print("Tuesday");
break;
}
}
```

Tasks -

- **Predict output**
- **Explain Dart switch behavior**
- **Fix the logic**
- **Add functionality: handle all weekdays**

```
void main() {
  int day = 1;
  print("1.Monday");
  print("2.Tuesday");
  print("3.Wednesday");
  print("4.Thursday");
  print("5.Friday");
  print("6.Saturday");
  print("7.Sunday");
  switch (day) {
    case 1:
      print("Monday");
```

```
      break;
    case 2:
      print("Tuesday");
      break;
    case 3:
      print("Wednesday");
      break;
    case 4:
      print("Thursday");
      break;
    case 5:
      print("Friday");
      break;
    case 6:
      print("Saturday");
      break;
    case 7:
      print("Sunday");
      break;
    default:
      print("Invalid day");
  }
}
```

1. Predict output

The output will be:

Monday

Tuesday

2. Explain Dart switch behavior

In Dart, the switch statement does not have implicit fall-through behavior like some other languages (e.g., C, Java). However, if there is no `break` statement at the end of a case, the execution will continue to the next case. In this example, since there is no `break` after case 1, it falls through to case 2 and prints both "Monday" and "Tuesday".

3. Fix the logic

To fix the logic, a `break` statement has been added after the print statement in case 1 to prevent fall-through.

4. Add functionality: handle all weekdays

The switch statement can be extended to include cases for all weekdays (1 to 7)