LAB-3

AIM : FUNCTION, CLASS and NULLABILITY

Prac-1 : Check for prime

Code :

```
import 'dart:math';

bool isPrime({required int number}) {
  if (number == 2 || number == 3) {
    return true;
  } else {
    for (int i = 2; i <= sqrt(number); i++) {
      if (number % i == 0) {
        return false;
      }
    }
  }
  return true;
}

void main(List<String> arguments) {
  print(isPrime(number: 12));
  print(isPrime(number: 5));
}
```

Output :

```
[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-5\
false
true

[Done] exited with code=0 in 1.061 seconds
```

Prac-2 : reapeatTask using Anonymous function

Code :

```
int repeatTask(int times, int input, int Function(int) task) {
  int result = input;
  for (int i = 0; i <= times; i++) {
    result = task(result);
  }
  return result;
}

void main() {
  print(repeatTask(2, 4, (int x) {
    return x * x;
  }));
}
```

Output :

```
[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-5\
65536

[Done] exited with code=0 in 1.199 seconds
```

Prac-3 : repeatTask using Arrow Function

Code :

```
int repeatTask(int times, int input, int Function(int) task) {
  int result = input;
  for (int i = 0; i <= times; i++) {
    result = task(result);
  }
  return result;
}

void main() {
  int ans = repeatTask(2, 4, (int x) => x * x);
  print(ans);
}
```

Output :

```
[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-5\
65536

[Done] exited with code=0 in 0.933 seconds
```

Prac-4 : Bert and Ernie

Code :

```
class Student {
  final String firstName;
  final String lastName;
  final int grade;

  Student(this.firstName, this.lastName, this.grade);

  void show() {
    print("Firstname = $firstName");
    print("Lastname = $lastName");
    print("Grade = $grade");
  }
}

void main(List<String> arguments) {
  Student bert = Student("Bert", "Scofield", 95);
  Student ernie = Student("Ernie", "Queen", 85);
  bert.show();
  ernie.show();
}
```

Output :

```
[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-6\
Firstname = Bert
Lastname = Scofield
Grade = 95
Firstname = Ernie
Lastname = Queen
Grade = 85

[Done] exited with code=0 in 1.322 seconds
```

Prac-5 : find volume and surface of sphere

Code :

```
class Sphere {
  final double radius;
  static const double pie = 3.1415926535897932;

  const Sphere({required this.radius});

  double getSurface() {
    return (4 * pie * radius * radius);
  }

  double getVolume() {
    return ((4 / 3) * pie * (radius * radius * radius));
  }
}

void main() {
  Sphere sphere = Sphere(radius: 12);
  print(sphere.getSurface());
  print(sphere.getVolume());
}
```

Output :

```
[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-6\
1809.5573684677208
7238.229473870882

[Done] exited with code=0 in 1.215 seconds
```

Prac-6 : get random number or null if return null then print 0

Code:

```
import 'dart:math';

int? getRandomNumberOrNull() {
  int randomNumber = Random().nextInt(2);
  if (randomNumber == 0) {
    return null;
  } else {
    return 42;
  }
}

void main() {
  final result = getRandomNumberOrNull() ?? 0;
  print(result);
}
```

Output:

```
[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-7\
0

[Done] exited with code=0 in 1.308 seconds

[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-7\
42

[Done] exited with code=0 in 1.403 seconds
```

Prac-7 : print name and surname based on user selection using boolean property surnameFirst

Code:

```
class Name {
  final String name;
  String surName = "";
  bool surNameFirst;

  Name(this.name, this.surNameFirst, [this.surName = ""]);

  @override
  String toString() {
    if (surNameFirst) {
      return ("$surName $name");
    } else {
      return (name);
    }
  }
}

void main() {
```

```
  Name micheal = Name("Micheal", true, "Scofield");
  Name oliver = Name("Oliver", false, "Queen");

  print(micheal.toString());
  print(oliver.toString());
}
```

Output:

```
[Running] dart "f:\B.TECH\SEM-5\SDP\lab_3\Chapter-7\
Scofield Micheal
Oliver

[Done] exited with code=0 in 1.101 seconds
```

Lab-3

→ function, class and nullability.

chapter-5

function: function is a simply a block of code that performs specific task

↳ Rather than Reapeting an some code multiple place create one function of that code And call it directly.

↳ All fullfill the functionulity of DRY (Do not reapeat yourself).

Input ⇒ function ⇒ output.

Anotony of function.

```
String compliement (int name) {
        return 'very good $name';
}
```

Return value.

Return Type: it defines what type of function Return A value if your function don't Return anything set type void

function name: you can name function almost Anything you like But you need to Add in lower case

parameters : parameters are the input to the function they go inside the parenthesis after the function name. This example has only one parameter.

* making parameters opptional
-> It is compulsory while calling function to add its parameters But we can make parameters optional by [ ]
example string show(string name, [String? title])
-> if we don't pass value then value considered is NULL

* naming a parameters.
-> Dart Allows named parameters to make the meaning of the parameters, more clear in function calls,
-> To creat a named parameter, you surround it with curly braces instead of square brackets. here's the same function As Above

Example {int max = 0 int? min=10

good function : function that not create any side effect outside of that function is called good function

Example : some function is Accessing and changing global variable value that means they got good functions.

mini excersise
1) write a function named you are wonderful with a string parameter called name. It should return a string using name Say's you're wonderful, Bob"

String youAre wonderfull (String name){
return "you're wonderful, ${
name}";
}

2) Add another int that function called number, people so that the function returns something like "you re wonderfull bob 10 people think to

String youAreWonderfull(String name,
int number people){
return "you're wonderful, ${name}

```
${numberPeople} people thinks so";
```

3. String youAreWonderful(){ required String name, int numberPeople=30}
{
return "you're wonderful, ${name}. ${numberPeople} think so"
}

* Assigning functions to variable

when assigning a value to a variable, functions behave just like other types

```
int number = 1;
string greeting = "hello"
boot is hungry = true;

function multiply = (int a, int b){
        return a * b;
};
```

* passing functions to functions
just as you can write a function to take int or string as a parameter

you can also have function as a parameter

```
void namedFunction (Function anonymous function) {
        // function body
```

* Returning functions from functions

Just as you can pass in functions as input parameter, you can also return them as output

```
Function namedFunction () {
        return () {
            print("hello");
        };
}
```

-> The return value is an anonymous function of type function.

Anonymous functions in foreach loops

```
const numbers = [1, 2, 3];
numbers.forEach (number) {
        final tripled = number * 3;
        print (tripled);
});
```

output
3
6
9

closures and scope

-> Anonymous function in dart act as what are known as closures. The term closure means that the code" closes around" the surrounding scope and therefore has access to variables and functions defined withing that scope.

```
{
    var age = 42;
    (parameters) {
        age = 43;
    }
}
```

scope

another scope

mini exercise

1) string wonderfull = you are wonderful (name);
print (wonderfull);

2) const people = ["chris", "Tiffani", "Pabloe"]
people.foreach (name) {
    print ("you are wonderful, $name");
}

★ Arrow functions

int add (int a, int b) => a+b;
=> you replaced the function's braces and body with an arrow (=>) and left off the return keyword

(parameters) => expression;

★ challenges

1) bool isPrime (int n) {
    if (n == 2 || n == 3) {
        return true;
    }
    if (n == 1) {
        return false;
    }
    for (i = 4; i < n/2; i++) {
        if (n % i == 0) {
            return false
        }
    return true;
}

```
main () {
    print (isPrime(5)); //true
    print (isPrime(68)); //false
```

2)
```
int RepeatTask (int times, int
    input, function (int) task {
        for (int i=0; i<times; i++){
            input= task (input);
        }
    });

print (RepeatTask (4, 2, (int x) {
        return x * x ; //65536
}
```


Chapter-6

class

=> classes are core components of object-oriented programming. They're used to combine data and functions inside a single structure

```
class Myclass {
    var abc = "hello"; // data
    Myclass () ; // constructor
    void mmethod () {
        print (abc) ;
    }
```

Object serialization

serialization is the process of converting a complex data object into a string once the object has been seerialized, it's easy to save that data or transfer it across the network because everything from your app to network and beyond knows how to deal with strings

=> The process of converting a string back int an object of your type called deserialization

* Adding a JSON serialization method

```
string toJSON() {
    return {"id": $id, "name": "$name"};
}
```

- It'll be similar to what you did in toString.

* cascade notation

=> Dart offers a cascade operator (..) that allows you to chain together multiple assignments on the same object without having to repeat the object name.

```
Final user = User()
  ..name = 'Ray'
  ..id = 57
```

=> cascade notation is strictly necessary, but it does make your code just a little tidier when you have to assign along list of properties or repeatedly call a

method that modifies your object

* constructors.

constructors are methods that create, or construct, instances of a class. This is to say, constructors build new objects.

example

```
class Address {
    Address();
    var value = "";
}
```

Including the default Address() constructor is optional

* long-form constructor

In dart the convention is to put the constructor before the property variable and the following generic constructor method at the top of the class body

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

This is known as long-form constructor

* short-form constructor

=> Dart also has a short form constructor where you dont provide a function body, but you want to initialize, prefixed with the this keyword.

```
class User {

  User(this.id, this.name);
    int id = 0;
    string name = "";
}
```

* named constructors
=> dart also has a second type of generative constructor called a named constructor, which yo create by adding an identifier on to the class name.

```
class Name.identifier Name()
```


* Forwarding constructors.

```
User.anonymous() : this(0, "anonymous");
```

This time there's no constructor body, but instead, you follow the name with a color and then forward thee properties on the unnamed constructor.

* factory constructors
-> A factory constructors provides more flexibility in how you create your objects. A generative constructor can only create a new instance of the class itself. However, factory constructors can return existing instance of the class, or even subclasses of it.

example

```
factory User fromJson(map <String,
                Object> json) {
  final User id = json['id'] as int;
  final username = json['name'] as
  String return
```

```
user(id: uerid, name:
}      username);
```

```
Final map = {'id': 10, 'name': 'ab'};
Final ab = User.FromJson(map);
```

Chapter - 7

null

Null means "no value" or "absence of value"

```
int abc = null;
```

=> A value of type 'null' cant be assigned to a variable of type 'int'

Nullable types

-> A nullable type can contain the null value in addition to its own data type

for example:

```
int? myint = null;
```


* Null aware assignment operator (??=)

for example

```
fonsize ??= 200
```

if fontsize is null then it will be assigned 20.0 but otherwise it will be assigned 20.0 but it returns it's value. The ??= operator it returns combines the null check with the assignment

* late keyword

late keyword is used to declare a variable or field that will be initialized at a later time. it is used to declare a non-nullable variable that is not initialized at a time of declaration.

challange
Bert and Ernie

1) class Student {

Student ({firstname = "vineet",
"lastname = "Joshi",
grade = 100}); }

final string ! firstName ;
final string ? lastName ;
final int grade ;

void show Details () {
print ('student name': ${firstname} ${lastname});
print ("grade : " ${grade});
}
}

void main () {

var Bert = new student ("Bert",
"oppenheimer", 98);

var Ernie = new student ("Ernie",
"queen", 85);
}

2) find volume and surface

class spheres {
static final PI = 3.14 ;
int ? radius ;
double ? _volume ;
double ? _surface ;
spheres ({this radius}) : _volume =
(4/3) * PI * pow (radius *, 3) ;
_surface : (4 * (PI) * pow (radius,
2)) ;
double get volume => _volume ;
double get surface => _surface

void main () {
var sp = spheres (radius : 10) ;
print (sp.surface) ;
print (sp.volume) ;
}

output.

1256.6
4186.6667

☆ IF null operator (??)

One very convenient way to handle null values is use the double questionmark (??) also known as if-null operator. This operator say's If the value of this test isn't null, then use it; otherwise go with the value on the right.

Example

```
String? message?
final text = message ?? "Error";
```

☆ null aware assignment operator (??=)

In the example above you had two variables: message and text. however, another common situation is when you have a single variable that you want to update if it's value is null

☆ null aware access operator

Earlier with an Integer. is negative, you show that trying to access the is Negative property when an integer was null caused a no such method Error! There's also an operator for null safty when accessing object members.

```
int? age;
print (age?.isNegative);

print (age?.toDouble());
```

☆ Null assertion operator.

Sometimes Dart isn't sure whether a nullable variable is null or not, but you know it's not, dart is smart and all, but machine don't rule the world yet.

operator (!) or sometimes mor generally reffered to as the bang operator

```
String nonNullableString = myNullable
        string!;
```

★ Null aware cascade operator

In chapter 6 you learned about the cascade operator (..), which allows you to call multiple methods or set multiple properties on the same object.

```
class user {
  String? name; int? id; }

User user = User()
..name = "Ray"
..id = 42;

user? user;
```

Challang-1 Random nothings

code:

```
import 'dart:math';

int? getRandomNumberOrNull() {
  int randomNumber = Random().nextInt(2);
  if (randomNumber == 0) {
    return null;
  } else {
    return 42;
  }
}

void main() {
  final result = getRandomNumberOrNull() ?? 0;
  print(result);
}
```

output 0 42 42 42 0 42

challang-2: Naming customs

code:

```
class Name {
  final String name;
  String surName = "";
  bool surNameFirst;

  Name(this.name, this.surnameFirst,
      {this.surName = ""});

  @Override
  String toString() {
    if (surNameFirst) {
      return ("$surName $name");
    } else {
    } return (name);
```

```
    }
}

void main() {

Name michael = Name ("Michael",
                true, "scofield");
Name oliver = Name ("Oliver", fal
                "queen");
print (michael.toString());
print (oliver.toString());
} Output ↓
Scofield Michael
Oliver
```