Data types and variables

The data types in dart are :

- Integer
- Double
- Map
  It has key value pair , written inside curly braces {}
- String
- Boolean
  Value is either true or false
- List
  It is same as array but have to denote type of value inside <>.

The syntax for the data types are :

- int variable_name = 21;
- bool variable_name = true/false;
- Map <string,dynamic> var_name = {
  “Name” : “Ritik”,
  “Age” : 13 };
  Dynamic means value can be of any data type , and key is string
- List <int> var_name =[1,2,3,4];
- String var_name = “Anuj”;
- Double var_name = 2.3;

Syntax to print variable :

print(var_name);

Pre built methods for List

Dart consists of pre built methods .

Syntax :

Var_name of list .method();

Methods : add(),addAll(),sort().....etc

Eg :

```
List <string> fruits : [“apple”,”Mango”,”Banana”];
fruits.add(“avocado”);
fruits.addAll([“grapes”,”berries”]);
bool val = fruits.contains(“mango”);
Print (val); //this returns either true or false
```

fruits.where() is one of the important method of List . It returns a element if there exist that element or return empty , to convert it into list we simply use .toList()

```
Eg : var value = fruits.where((element) => element ==”mango”).toList();
     print(value);
```

CODE :

```
void main() {
  int age = 21;
  String name ="Nisha";
  String address ="Dhangadhi";
  Map <String,dynamic> data ={
    "name":name,
    "age":age,
    "address":address
  };
  print(data["age"]); //by this we can print only specific value of key
  print(data.keys); //to print all keys only
   print(data.keys.toList()); //to print all keys only in list
  print(data.values.toList()); //to print all values only in list
  print(data.containsValue('age')); //returns either true or false
 }
```

If we want to print any certain key value only from map then,
SYNTAX:
Map <String,dynamic> data = {
“Name”: “Nisha”,
“Age” : 20}:
print(data[“AE

Final and Const keywords

We can't change the value of const and final variable.

Final —- runtime,can reside inside a class
Final variable is only assigned when accessed i.e when we define final variable then it is not assigned with the value but when we try to access then it is assigned

Const —-compile time,can’t reside inside a class
Constant variable are assigned whether we access it or not i.e when we define const variable Memory is allocated or we can say the variable is assigned with the value , whether we use it or Not.
Const is used inside class by only making it static.

If-else condition

CODE :

```
void main() {
 int age = 110;
 String vote = voteFunc(age);
 print(vote);
}

String voteFunc(int age)
{
  if(age>=18 && age <100)
  {
    return "You can vote";
  }
  else if(age>100)
  {
    return "You cannot vote";
  }
  else
  {
    return "Invalid age ";
  }
}
```

Ternary Condition

CODE :

```
void main() {
 int age = 110;
 String vote = voteFunc(age);
 print(vote);
}

String voteFunc(int age)
{
  String result = age>=18 && age <100 ? "You can vote": age>100 ? "You cannot vote"
: "Invalid age ";
  return result ;
}
```

For Loop and While Loop ,Switch Case

Same as c , c++

Break and Continue

CODE :

```
void main()
{
  int i ;
 for (i =1;i<=10 ;i++)
 {
   if(i%2 == 0)
   {
     print( i*2) ;
   }
   if( i == 5)
   {
     break ; // i ko value 5 vaye paxi it will break
    }
 }
  print ("Loop is stopped");
}
```

CODE :

```
void main()
{
  int i ;
 for (i =1;i<=10 ;i++)
 {
   if( i == 5)
   {
     continue;
    }
   if(i%2 != 0)
   {
     print( i*2) ;
   }

 }
  print ("Loop is stopped");
}
```

```
Output :

2
6
14
18
Loop is stopped
```

:

Function

We can call function inside class using object or making the function static.

SYNTAX TO CREATE OBJECT :

Class_name obj = new Class_name();

SYNTAX TO MAKE FUNCTION STATIC :

Static return_type function_name()
{
– – – –
– – – –
}

CODE :

```
void main() {
 Mul  obj = new Mul(); //creating obj
  print (obj.multiple(2)); //calling the function
  print (Mul.addition(3)); //calling static function
}
class Mul {
 int multiple(int x)
 {
   return (2*x);
 }
  static int  addition(int y) //defining static function
  {
    return (2+y);
  }
}
```

We also have inbuilt function for string like map and list.

Positional Parameters

parameters must be provided in the correct order when calling a function or constructor .

CODE:

```
void main() {
 print(user("Narad",23,"Mahendranagar"));
}
Map user( String name ,int age, String address) // position is important while calling
{
  return(
    {
  "name" : name,
  "age":age,
  "address":address
      });
}
```

Named Parameters

If parameter haru dhaerai xan vanay it will be difficult using the concept of positional parameter so we use named parameter.
Named parameter uses curly braces :
Syntax:
Return_type Function_name ({String name,int age})
{
– – – – – —
– – – – – –
}

The `required` keyword in Dart makes sure that a **named parameter** must be given a value when calling a function or constructor. Without it, the parameter becomes optional, and you might accidentally forget to provide it.

Why use `required`?

- It forces the user to provide important values.
- Prevents errors caused by missing values.

CODE :

```
void main() {
 print(user(name:"Narad",age:23,address:"mahendranagar"));
}
Map user( { required String name ,required int age,required String address})
{
  return(
    {
  "name" : name,
  "age":age,
  "address":address
      });
}
```

Default Parameters

To make parameter default , it must be made named parameter and use curly braces .
Default value does not work with positional parameters.

If user ley kunai value deyaena vanay tyo case maa default value use hunay vayo.

CODE:

```
void main() {
 double length = 4.0;
  double breadth = 2;
 print(areaRec(length:length, breadth:breadth)); //passing named parameters
}
  //making parameter named and set default value for length

double areaRec({double length = 1, required double breadth})
{
  return(length * breadth);
}
```

Constructors

We cannot have parameterized and default constructor in the same class .

Here to use global variables of the class inside the parametarized constructor or other functions , we use This Pointer .

Named Constructor

Syntax :

Class Class_name {

Class_name.namedConstructor()
{
– – – – – – – –
– – – – – –
}
};
Void main()
{
Class_name obj = new Class_name.namedConstructor();
– – –
– – –
}; // Here namedConstructor is the name of the named constructor which can be of any name according to user

CODE : Here in this code we haven't created obj because , we are not further using it.

```
class Firstclass{
  int a = 2;
  int b = 3;
  Firstclass(int x, int y) //parameterized constructor
  {
    this.a = x;
    this.b = y;
    print (x*y);
  }
  Firstclass.sum() //named constructor
  {
    print(this.a+this.b);
  }
}
void main()
{
  Firstclass(2,2); //calling parameterized constructor
  Firstclass.sum(); //calling named constructor
}
```

Using underscore before the name of any variables inside class makes them private.

Getters and Setters

CODE:

```
class Maths{
  int _den = 0;
  int _num = 0;
  //customized setter functions
  void set setnumerator(int val)
  {
    _num = val * 6;
  }
  void set setdenominator(int val)
  {
    _den = val * 6;
  }
  //customized getter functions
  int get getdenominator{
    return _den;
  }
  int get getnumerator{
    return _num;
  }
}
void main() {
  Maths obj = new Maths();
  obj._den=2; //setter
  print(obj._den);//getter
  obj.setnumerator =5; //setter function called
  obj.setdenominator = 4;
  print(obj.getdenominator);//getter function called
  print(obj.getnumerator);
}
```

The difference between getter,setter and customized getter,setter is that
In getter and setter we simply get and set the values , but in customized getter and setter we can make any type of changes before setting and getting .

In the above code there we first multiplied the value with 6 and then we set it .

Inheritance

Types :
1.single
2.Hierarchical – more children have one parent
3.Multilevel

SYNTAX :
class child_class extends parent_class
{
— – – –
— – – - –
}

constructor in parent class must be called before child class so we use **Super().**
SYNTAX:

class child_class extends parent_class{
chlid_class(parameter for child,parameter for parent) : super(parameter of parent)
{
– – – – – –
– – – – – –
}
In case of parameterized constructor its mandatory to write but not needed in case of default constructor.

CODE :

```
class Animal {
  Animal( String kingdom)
  {
    print("The kingdom of animal is $kingdom");
  }
}
class Cow extends Animal{
  Cow(String kingdom,String group) : super(kingdom)
  {
    print("Cow is $group");
  }
}
void main() {
   Cow obj =  Cow("Vertibrate","Mammel");
}
```

Method overriding

Here i want the function of same name in parent should be called after the function in child then ,

SYNTAX :

```
class parent{
 void  fun_name()
{
– – –
– – –
}
}
class child extends parent {
 Void fun_name()
{
– – – – –
– – – – –
super.fun_name();
}
}
Void main ()
{
Child obj = new child();
obj.fun_name();
}
```

Abstract Class

Abstract methods can only be made inside abstract class .
An object of abstract class cannot be made .
Abstract methods only contain name of function and parentheses.

CODE:

```
abstract class Electronics{ //abstract class
  void watch(); //Abstract method
  void play()
  {
    print ("playing games in electronic device");
  }
}
class Mobile extends Electronics{
  void watch()
  {
    print("Watching movie in electronic device");
  }
}
void main()
{
  Mobile obj = new Mobile();
  obj.watch();
}
```

Implements vs Extends

- If we use implements then we have to override all functions of parent in child.
- If we use implements we can have more than one parent for single child.

Mixins

When we have similar functions in two classes then we can use mixins.

Syntax :

```
class one with function1,function2{
– – –
– – –
}
class two with function1,function2{
– – –
– – –
}
mixin function1()
{
– – –
– – –
}
mixin function2()
{
– – –
– – –
}
```