DART – (dartpad.dev)

1. Dart

2. Dart Syntax

3. Comments in Dart

4. Variables

5. Data types

6. String

7. Operators

8. Control Structures

9. Loops

10. Collections

11. Functions

12. Class

1. Dart - Dart is a programming language

2. Dart Syntax –

main() { // always write main before writing the code

print (“Aditya”);

}

3. Comments in Dart

//

shift + ?

4. Variables –

// a-z , A-Z , 0-9(cannot start with number) , \_ , $

main() {

String name = “ Aditya ” ;

print (name);

}

5. Data Types –

I. int –

int age=10;

print(age);

II. double –

double age = 10.10;

print(age);

III. String –

String age= “10.10”;

print (age);

IV. bool –

bool age= true;

print (age);

V. List (layman language = Array) –

List arr = [1,2,3]; List arr = [1,2, ’Aditya’ ];

print(arr); print (arr[2]); // Aditya

VI. Map –

Map arr = { ‘name’ : ‘Aditya’ , ‘city’ : ‘Delhi’ };

(‘name’ Key or index or location identifier of ‘Aditya’)

print (arr[‘name’]); //Aditya

VII. Dynamic

var arr = ‘10’; // Layman language = var is used for int , double , String , Map , List , Array etc. like in JavaScript

print(arr);

6. Strings –

6-A. main(){

var fname=”Aditya is my name”;

var lname= “Royal”;

print(fname+ “ ”+ lname);

print (fname.toLowerCase()); OR str=fname.toLowerCase();

print(str);

print(fname.toUpperCase());

print(str.length);

str.trim(); // removes space from START and END of the string

print(fname.compareTo(lname)); // to compare

print(fname.replaceAll(‘Aditya’, ’Linus’ ));

print(str.split(‘ ’)); // [Linus , is , my , name]

var arr = str.split(‘ ’);

print(arr[1]); // is

var str = “My name is Aditya”;

print(str.substring(3,6)); //nam

print(str.substring(6)); // e is Aditya

}

7. Operators –

7-A. Arithmetic Operator –

+ , – , / , \* , % ,

~/ // gives us whole number – (a~/b)

7-B. Relational Operator –

< , > , <= , >= , == , !=

7-C. Test - is , is!

var a=10 ; var b=20;

print(a is b); // false , print(a!= String) //true

7-D.

++ , -- //Post increment/decrement ad pre increment/decrement

7-E. Assignment Operator

a=a+10; OR a+=10;

7-F. Logical Operator

&& , || , !

7-G. Conditional Operator

?: , ??:

main() {

var a=10;

print(a>20?30:500); // 500

}

7-H. Cascade Notation

..

main() {

var str = “Aditya” ;

print(str..toLowerCase()..substring(2)); // Runs both simultaneously

}

7-I. Bitwise operator –

& , | , ~ , ^ , a << b , a>>b

8. Control Structures –

if ,else , elseif , switch

9. Loops –

I. for (var i=0 ; i<10; i++)

{ print(i); }

II. while(n!=0)

{ n/=10;

print(n);

}

III. do{

print(x);

x++;

} while(x<=10);

IV. for ( in ) , forEach - loop Growable List –

main() { OR main() {

List<int> larr = List(3); List<int> larr = List(3); OR List<int> larr =[10,20,30,40];

larr[0]=10; larr.add(25);

larr[1]=20; larr.add(50);

larr[2]=30; larr[1]=100;

for(int x in larr) { // 10 list.removeAt(1);

print(x); } 20 larr.clear(); // to clear the entire array

larr.forEach((x) => print(x)); 30

}

9-B. Collections –

I. List – Fixed ( below )

Growable ( above )

main() {

List<int> larr = List(3); OR List<int> larr =[10,20,30,40];

larr[0]=10;

larr[1]=20;

larr[2]=30;

print(larr[1]); //20 answer

}

II. Set (does not keep duplicate data , for loop will not work because the data is unordered i.e. indices are not 0,1,2,…

they can be 0,3,10,5,7,… i.e. the indices are random )

Set<int> sarr = Set(); OR Set<int> sarr= Set.from([10,20]);

sarr.add(10);

sarr.add(20);

print(sarr.contains(20)); //true

srr.remove();

sarr.isEmpty();

sarr.length();

sarr.clear();

for(int x in sarr) { //10

print(x); } 20

III. Maps

Map <String , String> marr = Map();

OR

Map <String, String> marr = {

“name” : “Aditya” ,

“city” : “Chandigarh” // “keys” : “values”

};

OR

marr[‘email’] = [aditya@chandigarh.com](mailto:aditya@chandigarh.com);

for(String key in marr.keys){ // name

print(key); city

} email

for(String key in marr.values){ //Aditya

print(key); Chandigarh

}; [aditya@chandigarh.com](mailto:aditya@chandigarh.com)

OR

marr.forEach((key,value) => print(key + “ - ” + value)); // name – Aditya

city – Chandigarh

email – [aditya@chandigarh.com](mailto:aditya@chandigarh.com)

marr.remove(“name”);

marr.length;

marr.clear();

marr.isEmpty;

marr.containsKey(“name”); // True

marr.containsValue(“Chandigarh”); // True

11. Functions –

I. Basic Rules

Non-Parametrised Function –

I-A. main() {

getNumber();

}

getNumber() {

print(“Hello”); // Hello

}

I-B. main() { print(getNumber()); }

getNumber(){} //null

Parametrised Function –

I-C. main(){

getNumber(10,20);

}

getNumber(int a, int b){

print(a\*b);

}

I-D. main() {

var x= getNumber(10,20);

print(x);

}

getNumber(int a, int b){

print(a\*b);

return a+b;

}

// 200

30

II. Expressions – FAT Arrow

main(){

var x= getNumber(10,20);

print(x\*2);

}

getNumber(int a, int b){ OR getNumber(int a, int b) => (a\*b)+10; (only valid for 1 line statement)

var y= (a\*b)+10; // 420

return y ; // 420

}

III. Parameter

DONE ABOVE EXPRESSIONS

IV. Optional

main(){

getNumber(10,30);

}

getNumber(int x, [int y=10 , int z=20]) {

print(x+y);

}

V. Named Parameter

main(){

getNumber(10 , z : 30 , y : 20);

}

getNumber(int x , {int y, int z}){

print(x); // 10

print(y); 20

print(z); 30

var num= (x+y)\*z;

print(num); //900

}

12. OOPS , Class , Object , Constructor , Hiearchy ( class2 extends class1)

(make object of class 2 and call class 1)

I. main(){

var obj =new class1();

print(obj.name); // Aditya

obj.fun1(); Hello1 Aditya

}

class class1{

var name = “Aditya”;

fun1(){

print(‘Hello1 ’ + name);

}

}

I. main(){

var obj1 = class1();

obj1.name=”Aditya”

obj1.fun1(); // Hello1 Aditya

var obj2= class1();

obj2.name=”Royal”

obj2.fun1(); // Hello1 Royal

}

class class1{

var name;

fun1(){

print(‘Hello1 ’ + name);

}

}

12 . B – Constructor

I. Default –

main(){

var obj1 = class1();

obj1.fun1(); // Hello

} Aditya

class class1{

class1(){

print (‘Hello’);

}

fun1(){

print(‘Aditya’);

}

}

II. Parametrised –

main(){

var obj1 = class1(“Aditya”);

obj1.fun1(); // I am Aditya

}

class class1{

var name;

class1(name){

this.name=name;

}

fun1(){

print(“I am” + name);

}

}

12-C. Polymorphism –

main(){

var obj =new hr();

obj.fun1();

}

class emp{

fun1(){

print(“emp\_fun1”);

} }

class hr extends emp{

fun1(){

super.fun1(); // emp\_fun1

print(“hr\_emp1”); hr\_fun1

}}

12 – C. Abstract –

// to make mandatory use of a function of parent class in child class

// abstract function

// No object to be made of abstract class

// abstract class contains normal functions and variables

main() {

var obj = new hdfc();

print(obj.id\_proof());

}

abstract class rbi{

var name = “Aditya”;

id\_proof();

test(){

}

}

class hdfc extends rbi {

id\_proof(){

}

}

12 – D. Interface –

Using implements you can inherit more than 1 class

main(){

var obj =new class3();

obj.fun1();

}

class class1{

fun1(){

print(“Class1\_Fun1”);

} }

class class2{

fun2(){

print(“Class2\_Fun1”);

}}

class class3 implements class1,class2{

fun1(){

print(“Class2\_Fun1”);

fun2(){

print(“Class2\_Fun1”);

}}