Dart

Dart is a programming language with these features:

Object oriented Language => meaning it not a procedural language, it make use of classes, interfaces, object and other OOP terms.

Open Source => meaning the source code of dart is open, anyone can download it and modify it. So in case you create your own operating system, you can download it and modify it to run on your operating system.

Cross platform => this means Dart can run on all operating systems. Its platform independent, meaning it does not depend on a single operating system, it just like Java.

**=================================================**

Dart synthess:

Dart has a main() method that runs your program when your application start. Unlike Java, this main() method does not take in any command line argument.

void main() {

print("justice");

}

Note= You can create a dart file without a class.

**=================================================**

How To Run Dart File ON CMD

Set up the dart SDK and an environment variable on your system.

Then create a file with the extension .dart with below code in it. Example index.dart

void main(){

print("dart is running");

}

Now, open your cmd and navigate to the directory of this above file and run it with this syntax. dart filename.dart (just type dart followed by the file-Name)

dart index.dart

// result: dart is running

**=================================================**

Variables and data types

Dart support these data types:

* Numbers = dart has two number data types (int and double)
* Strings
* Booleans (Bool)
* Lists
* Maps
* Dynamic = can change TYPE of the variable, & can change VALUE of the variable later in code.

dynamic v = 123; // v is of type int.

v = 456; // changing value of v from 123 to 456.

v = 'abc'; // changing type of v from int to String.

* Var = can't change TYPE of the variable, but can change VALUE of the variable later in code.

var v = 123; // v is of type int.

v = 456; // changing value of v from 123 to 456.

v = 'abc'; // ERROR: can't change type of v from int to String.

* Final = **final:** can't change TYPE of the variable, & can't change VALUE of the variable later in code. (It the same as const)

final v = 123; // v is of type int.

v = 456; // ERROR: can't change value of v from 123 to 456.

v = 'abc'; // ERROR: can't change type of v from int to String.

**=================================================**

Dart Operators

Operators are special symbols for performing operations on its operands.

For example: 2 + 5;

2 and 5 are the operands

“+” is the operator.

Different Types Of Operators IN Dart:

* Assignment Operators

Is an operator that is used to assign a value to a single operand (We mainly use =, +=, -=) Eg:

= Example: Var myName = “justice”

+= (add and assignment) Example: C += A is the same as C = C + A;

-= (subtract and assignment) Example: C -= A is the same as C = C - A;

* Arithmetic Operators

Is for performing Mathematical operation on operands like addition, subtraction, multiplication, division and %module. Eg. ( +, -, %, \* )

2 + 3

* Type test Operator

This operator is used to check the data-type of an object or variable, which return true or false. By using two operators (is and is!)

/\* is (The “is” operator is used to Check if the object have a specific datatype.

Returns True if the object have specific dataType) EG:\*/

int myAge =55;

print(myAge is int);    // result: true (myAGe has a datatype of int)

/\* is! (The “is!” operator is used to Check if the object does not have a specific datatype.

    Returns false if the object have specific dataType) EG: \*/

int myAge =55;

print(myAge is! int);    // result: false (myAGe has a datatype of int)

* Relational Operator

It used to check the relationship between two operands and it returns true or false result. Eg. 3 <= 4;

== (Is equal to)

!= (Not equal to)

>= (greater than or equal to)

<= (Less than or equal to)

< (Less Than)

> (Greater than)

* Logical Operator

Logical operators are used to combine two or more expression to see if it will return true or false (we mainly use: && = AND, ! = NOT, || = OR)

void main() {

 var age = 10;

 var score = 10;

  // the below "!" operator says "if a value is true, make it false"

  // so "age" and "score" has a value of 10. But the "!" will say no, they don't hava a value of 10

  if(!(age ==10 && score ==10)){

    print("age And score is 10");

  }

  else{

    print("age and score is not 10");

  }

  //result: age and score is not 10

}

* Conditional Operator

Conditional operator is used as a short-hand form of writing if-else statement. It also known as “Ternary Operator”;

The syntax is: conditionToCheck ? if-statemtExperation-Block : else-StatementExperation-Block;

void main(){

    // check if 10 is greater than 15 and assign "Greater" or "Smaller" to the "result" variable

    var result = 10 > 15 ? "Greater" : "Smaller";

    print(result);  // result: smaller

    // this can be written in if-else statement as.....

    var result;

    if(10 > 15){

      result = "Greater";

    }

    else{

      result = "smaller";

    }

     print(result);  // result: smaller

}

**=================================================**

Type Casting IN Dart

Type-casting is a way of converting one datatype to another.

How to convert String-number to Integer

Use the num.parse() to convert a string-number to Number. If the value passed into it is not a string number, there will be error. EX:

void main() {

  /\* num.parse() convert a String number to an integer.

  if the value passed to it is not a String-number, there will be an error

  \*/

  print(num.parse("42")); // result: 42

  print(num.parse("justice")); // result: this will be an error

}

**=================================================**

Assert() method

The assert method is used to check if condition is true or false. It mainly used for debugging purposes

assert(condition, optionalMessage);

void main(){

  printAge();

}

 void printAge(){

 int num1 = 5;

 int num2 = 5;

 // if num1 is not equal to num2 the program will throw an error. but if it’s equal it will not

// so if the assert() method returns false, the program will throw an error

 assert(num1==num2, "num1 must be equal to num2");

 }

**=================================================**

List

List is a way of storing data in a list of order. In other programming language Array is used as a list.

List format: List<dataType\_of\_this\_list\_element> myList = [element1, element2];

void main(){

  // ========== HOW TO CREATE EMPTY LIST ==========

   List<String> myEmptyList =[];

    // ========== HOW TO ADD ELEMENT TO THE END OF A LIST==========

    // list has add() method which will add element at the end of a list

 List<String> myNewColors =["red", "yellow"];

 myNewColors.add("white");

 print(myNewColors);

   /\* Result:=>

[red, yellow, white]

\*/

  // ========== HOW TO REMOVE ELEMENT FROM A LIST ==========

   List<String> newColors =["red", "yellow"];

   newColors.remove("yellow"); // remove "yellow" from the list

  print(newColors);

   /\* Result:=>

  [red]

\*/

   // ========== HOW TO Map Through A List==========

 List<String> myColors =["red", "yellow"];

   myColors.map((myColorsElement){  //"myColorsElement" represent each single elemnet in mycolors eg. "red" and "yellow"

   // whatever code in this block will be run on each element in the list

         print(myColorsElement); // print all the element

   }).toList();

   /\* Result:=>

yellow

red

\*/

  // ========== HOW TO PRINT THE LAST ELEMENT OF A LIST TO THE FIRST ==========

 List<String> colors =["red", "yellow", "pink", "blue"];

/\* first assign "i" the last index of the list length, then stop the loop when it's equals to 0 which represent the first element in the list.

 Then decrease "i" since its starting from the end of the list \*/

for(int i=colors.length-1; i>=0; i--){

  print(colors[i]);

}

/\* Result:=>

blue

pink

yellow

red

\*/

   // ========== HOW TO CREATE LIST THAT CONTAIN LIST==========

 // from below the list contains another list.

List<List<int>> myListOfLists = [[1, 2, 3], [4, 5, 6]];

// you need to first map through the parent list, then get the inner list and map through it too to get it element

myListOfLists.map((myInnerList){

   myInnerList.map((innerListElement) {

     print(innerListElement);

   }).toList();

}).toList();

/\* Result=>

1

2

3

4

5

6

\*/

// ============ HOW TO CREATE A LIST REPRESENTATION OF MAP OR CONVERT LIST TO MAP============

 List<String> sports = ['cricket', 'football', 'tennis', 'baseball'];

 // the key will be the index, and the value will be the element

 Map<int, String> map = sports.asMap();

 print(map); // {0: cricket, 1: football, 2: tennis, 3: baseball}

// ====== HOW TO SKIP ELEMENTS IN A lIST =======

// skip(index) means ignore all the element before this index, and then print the index element and the element after it

 List<String> colors =["red", "yellow", "pink", "blue"];

 var bb =colors.skip(2);

 print(bb);

// Result: (pink, blue)

}

**=================================================**

**Null Safety**

By default, all dart variables and methods are non-nullable, meaning they can’t be or return null.

So, to make a variable or method nullable, meaning to be or return null, you have to use the null-safety operator “?” on it data-Type or before the variable or method.

You have to also use ! to check if a particular variable has a null value. If the value is null, dart will throw and exception

**Example:**

// Non-nullable ==> means it must have a value before using it (can't be null).

String myName;

print(myName);  // this will be an error since is Non-nullable

// Nullable ==> means it can be null before using it

String? email;

print(email); // this print null, without causing any error

String school = null; // it will be error, you're assigning null value to a Variable that must not be null

String? church =null; // Correct no error, because it marked nullable to accept null values

String getName(){

  return null;  //error: the method is non-nulllable it must not return null

}

String? getTown(){

   return null; // correct: the method is marked nullable

}

List<String> colors1 =[null]; // error

List<String> colors2 =null;  //error

List<String?> newcolors =[null]; // correct

List<String>? colors =null;  //correct

Use the “!” to check if a variable has a null value

void main(List<String> args) {

  // make the variable nullable but don't assign any value to it

  String? firstName;

  // use the "!" to check if the <firstName> has a value of null

  // if the value is not null, dart will not throw an exception

  //But if it has a value of null, dart will throw an exception like:

  // Null check operator used on a null value

  print(firstName!);

  /\* result:

  Unhandled exception:

Null check operator used on a null value

\*/

}

**=================================================**

**String interpolation**

To concatenate a variable together with a string, use the String interpolation symbol $ or ${}.

If it only a single variable you want to print together with the string, use single dollar sign “$”

If you want to perform an expression together with a String, use the string interpolation symbol “${}” together with any arithmetic operators like +, \* or –

**Example:**

main() {

  String firstName = "Justice Ankomah";

  String lastName = "Ankomah";

// Use a just a dollar sign if you want to print a single variable

  print("my first Name is: $firstName");

// results: my first Name is: Justice Ankomah

  // if it an expression then use the string interpolation symbol ${}

  print("my full-name is: ${firstName + ' ' + lastName}");

  // my full-name is: Justice Ankomah Ankomah

**=================================================**

**Escape dart special characters**

Special characters are some characters that dart will not print them out to the system console.

So, to escape or false dart to print these characters, use single back-slashes “\” or the Raw-String symbol “r”

NOTE:: The back-slash needs to come in-front of the special character you want to escape.

Example: (using back-slash)

// escape special characters by using backslash \.

// THE back-slash should come Infront of the special-characters you want to escape,

  print("this is \"justice\" ");

  //result: this is "justice"

  // OR

  print("this is \\justice\\ ");

  //result: this is \justice\

**Example: (using Raw-String symbol “r”)**

**Note:** the raw-String symbol “r” needs to come in-front of the String that contains the special characters you want to escape.

main() {

  // The Raw-String symbol “r” must come in-front-of the string that contains the special characters

  String churchName = r"my church name is: $Methodist$";

  print(churchName);

  // results:my church name is: $Methodist$

  // Or

  print(r"my name is: \justice\");

  //results: my name is: \justice\

}

**=================================================**

**CaseCade Operator (..)**

The casecade operator “..” just helps to avoid repeatedly typing of codes that needs to be perform on specific object. So, it simply saves you from repetitive typing.

**Example:**

void main() {

  /\*

  List fruits = [];

    fruits.length = 0;

    fruits.add("lime");

    fruits.add("orange");

    fruits.shuffle();

     INSTEAD OF REPEATING ABOVE "fruits" OBJECT-name ON ALL IT OPERATORS YOU CAN DO WHAT IS BELOW:S

  \*/

  // Create a list of fruits, Then instead of typing "fruits" all the time to perform some operation

  // Simply user the .. cascade operator. (so the .. below represent the "fruit")

  // Note = you have to close only the last operation with semi-colon

  List fruits = []

    ..length = 0

    ..add("lime")

    ..add("orange");

  print(fruits);

  // Result: [lime, orange]

}

**=================================================**

**The toString() method In A class**

If you have a class and you override the toString() method, it will return such string method when the instance/object of the class is only printed.

Why? when you create an object of a class and you print it to the console. It only returns the address of that object in the memory location.

For example:

A a =new A();

Print(a); // it will print the address of the “a” object in the memory like: adadads.A or Instance of A

So the toString() method in a class will only override the return of address location when the instance/object of the class is printed to the console.

**Example:**

// Create a class

class Name {

  String name = "justice";

  int age = 23;

// override the toString method and return some String when the object/instance of this class is printed

  @override

  String toString() {

    // am returning the above class properties

    return this.name + " " + this.age.toString();

  }

}

void main() {

  // create object/instance of the above class

  Name name = new Name();

  /\*

  If you print an object/instance of a class and that class override the toString method,

  it will invoke the toString method implicitly instead of returning the instance address of the class.

  \*/

  print(name);

  // So because the below "Name" class override the toString methods the result will be: justice 23

 // if the "Name" class didn't override the toString method, then the result will be: Instance of 'Name'

}

**=================================================**

**DateTime**

The DateTime class provides methods to work with date and times.

void main() {

  //=== GET THE CURRENT DATE AND TIME

  // use DateTime.now() to get the current date and time. it returns an object of DateTime

  DateTime cur\_DateTime = DateTime.now();

  print(cur\_DateTime);

  // result: 2022-02-02 08:48:26.605217

//====== Get The Number of A Particular Day or a Month

// Use DateTime.<month-Name or Day Name> to get the number of that month or a day. it returns an int

// Eg:

  int dayNumber = DateTime.sunday;

  print(dayNumber);

  // result: 7

  int monthNumber = DateTime.december;

  print(monthNumber);

  // result: 12

  //==== ADD & SUBTRACT: DAYS, MINUTES, SECONDS, MILLISECONDS FROM A PARTICULAR DATE

  // The Duration() class gives you access to work with days, hours, minute, seconds so...

  // Use the <DateTime>.add(Duration(key:value)) to add days, hours, minute, seconds to a particular dateTime

  // Use the <DateTime>.subtract(Duration(key:value)) to subtract days, hours, minute, seconds from a particular datetime

  DateTime oldDateTime = DateTime.parse("2022-02-02 00:00:00");

  // ADD 2 days to the above oldDateTime

  DateTime newDate = oldDateTime.add(Duration(days: 2));

  print(newDate);

  // result: 2022-02-04 00:00:00.000

  // two days have been added to the above oldDateTime

  // subtract 2 days from the above oldDateTime

  DateTime subDate = oldDateTime.subtract(Duration(days: 2));

  print(subDate);

  // result: 2022-01-31 00:00:00.000

  // two days have been subtracted from the above oldDateTime

  // === GET THE YEAR, DAY, MONTH, HOUR, MINUTS, SECONDS of a particular date

  // use <particular-DateTime>.T the "T" can be year, month, day, hour, minute, seconds

  print(oldDateTime.year);

  // result: 2022

  // ==== CUSTOME DATETIME

  // Use the DateTime.parse(<datetime-here>); And pass in your own date & time.

  // you can seperate the date and the time with "T" eg: 2022-02-02T10:02:24.5 the "T" will tell dart that the string after the date is time

  // Or you can simple use space to seperate the date and time if you don't want to use "T" eg: 2022-02-02 10:02:24.500

  // Eg.

  DateTime customeDateTime = DateTime.parse("2022-02-02T10:02:24.5");

  print(customeDateTime);

  // result: 2022-02-02 10:02:24.500

// ==== CHECK DIFFERENCE BETWEEN TWO DATETIME

// use the <fIRST\_DateTime>.difference(<SECOND\_DateTime>) to check the difference between two dates.

// it actually returns the time difference in Duration, does either 1 day = 24hrs, 3 days = 72hrs, 2 minute or 5 seconds difference

// So the return type is Duration

  DateTime todayDatetime = DateTime.parse("2022-02-02 10:02:24.5");

  DateTime yesterDayDatetime = DateTime.parse("2022-02-01 10:02:24.5");

  print(yesterDayDatetime.difference(todayDatetime));

  // result: -24:00:00.000000 meaning it just 1 day difference

//  === CHECK THE DIFFERENCE IN: days, hours, minutes, seconds, or inMicroseconds

  // The difference() method returns an object of the Duration of the associated date

  // So you can use that object to check the difference in: days, hours, minutes, seconds, or inMicroseconds

  Duration difference = yesterDayDatetime.difference(todayDatetime);

  // difference in days:

  print(difference.inDays);

  // result: -1

  // difference in inHours:

  print(difference.inHours);

  // result: -24

//==== CHECK THE Negative Value DIFFERNCE OF TWO DATETIME

  DateTime oneDatetime = DateTime.parse("2022-02-02 10:02:24.5");

  DateTime twoDayDatetime = DateTime.parse("2022-02-01 10:02:24.5");

  Duration diff = yesterDayDatetime.difference(todayDatetime);

  print(diff); // result: -24:00:00.000000

  // check if the difference is Negative

  print(diff.isNegative);

  // true

}

**=================================================**

**Unite testing**

Unit testing is a way of testing classes, single methods to see if it produces the expected outcome. For example, let say you have a method that you return true when age is greater than 18 and false when age is less than 18.

You can use unit test to test the method.

**Example:**

Inside Age.dart

// create a class with a method that returns true if a number is greater  than

//or equal to 18 and false if is less than 18

class Age {

  int? age;

  Age({this.age});

  bool checkAge() {

    if (age! >= 18) {

      return true;

    } else {

      return false;

    }

  }

}

// Inside test/checkage\_test.dart

import 'package:test/test.dart';

import '../index.dart';

/\*

install the "test" package.

Then create a folder/package called "test". This will be the folder to group all your test

Then create a dart file checkage\_test.dart in the above "test" package.

The file name should be what-to-check\_test.dart

\*/

void main() {

  // Test single function

// the test() method takes two argument, 1: description, 2: function to test

  test(

      // write a little description of what the test is about

      "if given age is greater than 18, return true else false", () {

    Age age = new Age(age: 129);

    // expect("run a function", "what result do you expect the function to be")

    expect(age.checkAge(), true);

  });

// Group different kinds of test together in the group() method

  group("check", () {

    test("if given age is greater than 18, return true else false", () {

      Age age = new Age(age: 39);

      expect(age.checkAge(), true);

    });

    test("if given age is greater than 18, return true else false", () {

      Age age = new Age(age: 129);

      expect(age.checkAge(), true);

    });

  });

}

// then open the teminal and run to check the test

// run: dart test test/checkage\_test.dart

// result: +3: All tests passed!

**=================================================**

**Exceptions**

Exception is simply unexpected errors that happens while a program is running.

Exception happens when a particular function/method throws an error to a piece of code that calls it. It then the responsibility of the piece of code that calls the function to which the exception was thrown from to handle it, or throw it again to the chain method that calls the piece of code.

so, exceptions occurs when it is thrown, and then handled by being caught.

Note: When an exception happens in some line of code in try block, let say in line 3, the code will jump straight to the catch block and never return to the try block to execute line 4 and the rest of the code in the try-block.