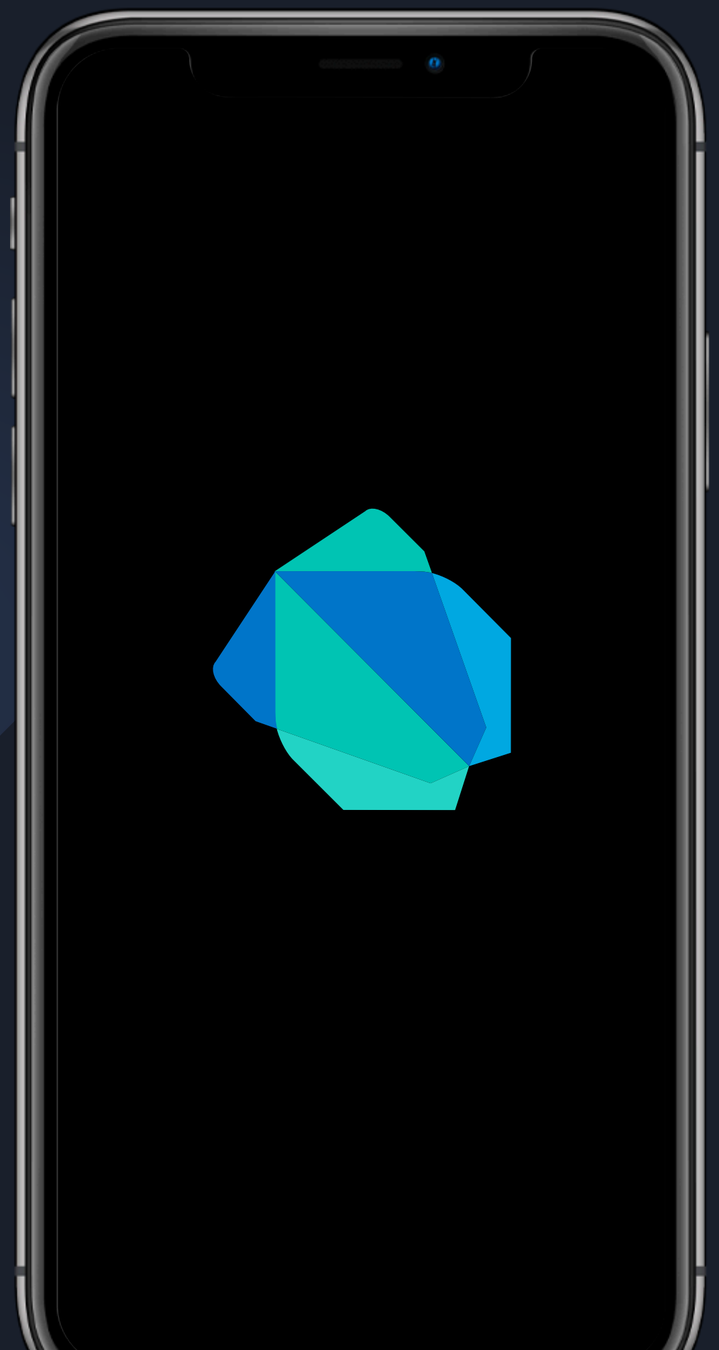
Learn with Rabbil

DART

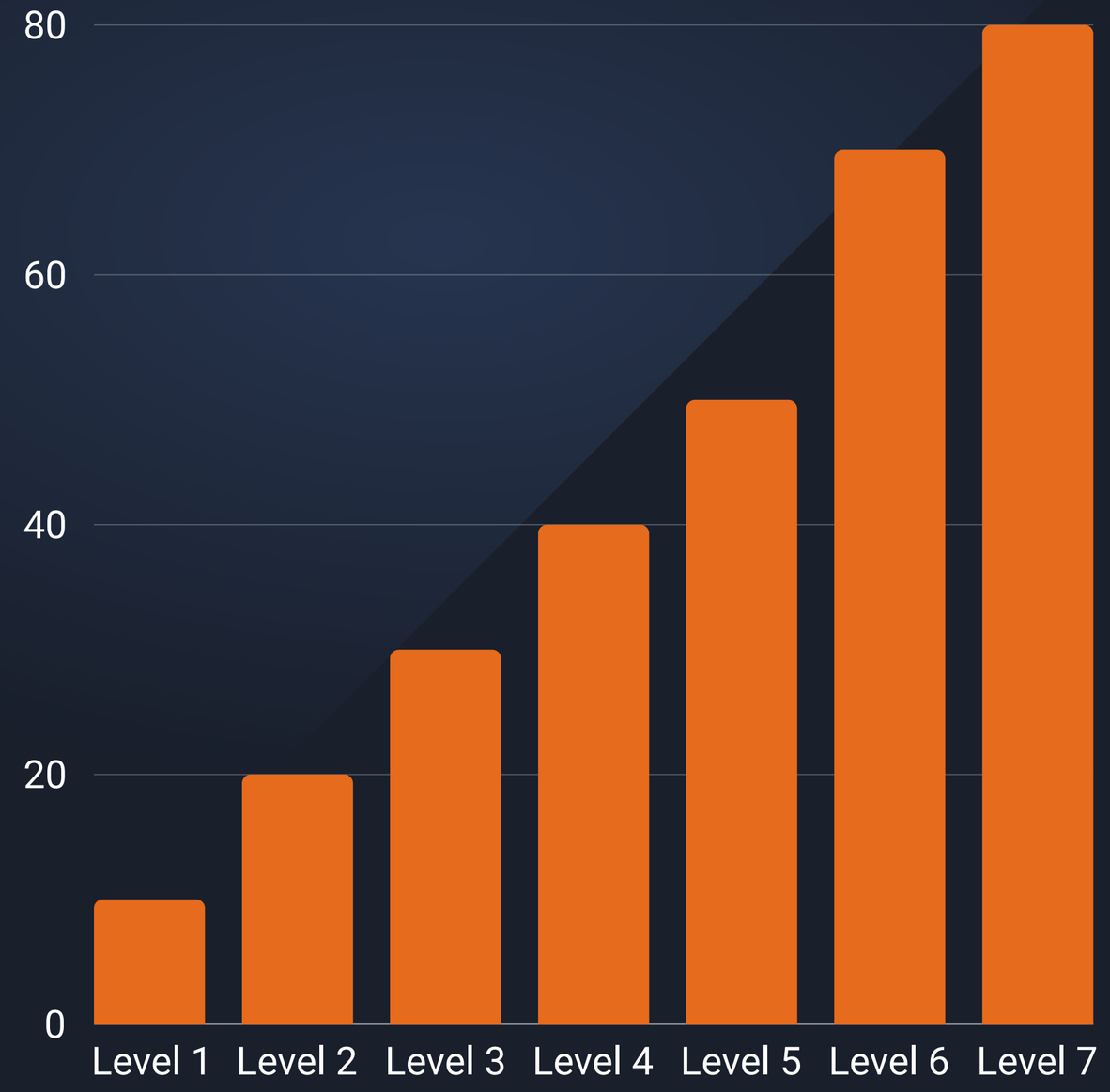
Dart is a client-optimized language
for fast apps on any platform



Learn with Rabbil

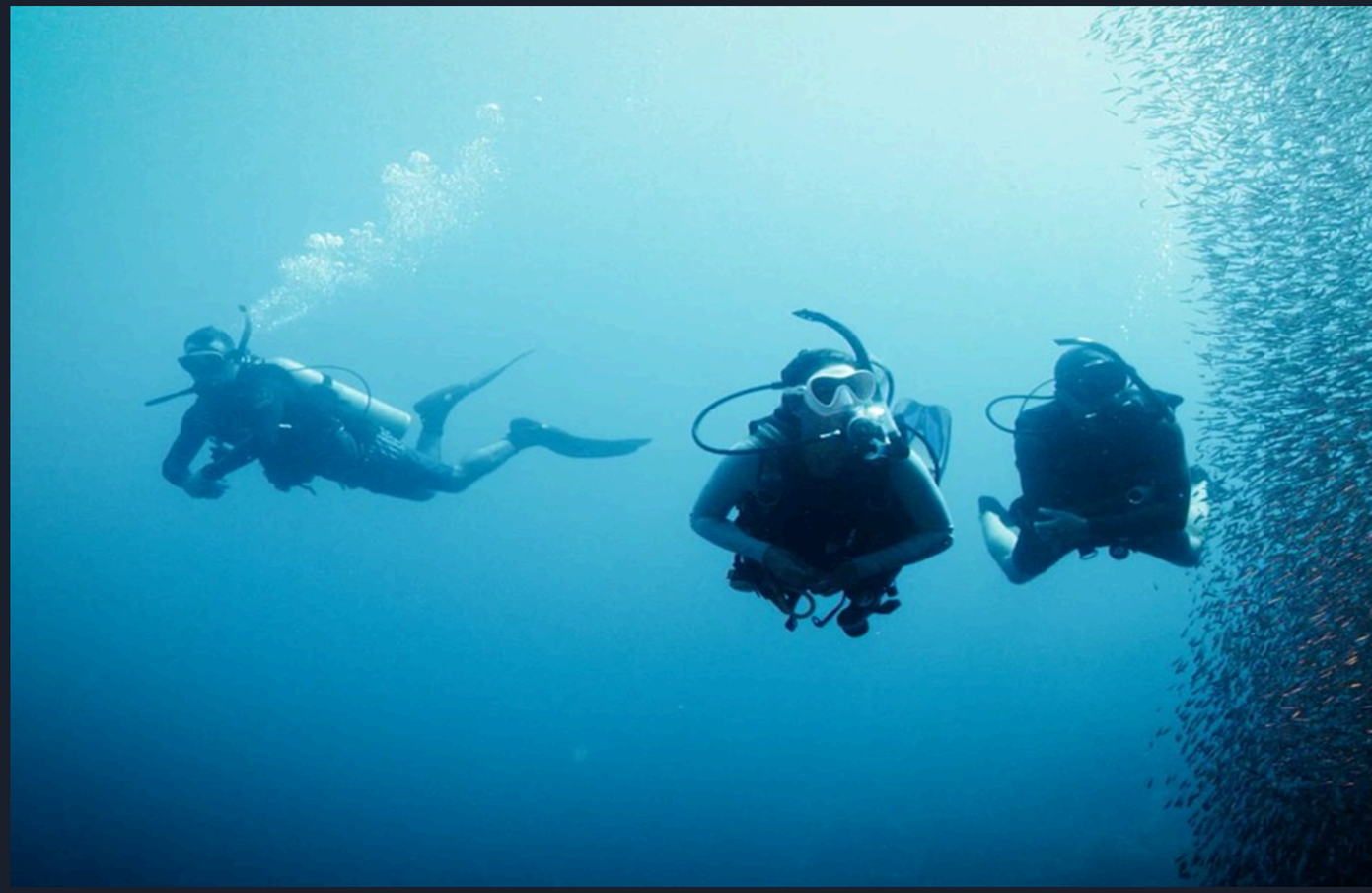
YOUR UNDERSTANDING

Will Increase gradually



BEGINNER STAGE

Don’t Dive Deep When You Are Beginner



Learn with Rabbil

DART

Environment Setup

- https://dartpad.dartlang.org
- https://gekorm.com/dart-windows
- https://www.jetbrains.com/idea/download

Learn with Rabbil

MY FIRST

Dart Program

- The main() function is a predefined method in Dart.
- This method acts as the entry point to the application.

```dart
void main() {
  print('Hello Dart I love You');
}
```

Learn with Rabbil

DART SYNTAX

Syntax is called set of rules for writing programs. Dart has-

- Variables and Operators
- Classes
- Functions
- Expressions and Programming Constructs
- Decision Making and Looping Constructs
- Comments
- Libraries and Packages
- Data structures represented as Collections / Generics

Learn with Rabbil

DART SYNTAX

Whitespace and Line Breaks

Dart ignores spaces, tabs, and newlines that appear in programs.

Dart is Case-sensitive

Dart is case-sensitive. This means that Dart differentiates between uppercase and lowercase characters

Statements end with a Semicolon

Each line of instruction is called a statement. Each dart statement must end with a semicolon

Comments in Dart

Single-line comments (//) Multi-line comments (/* */)

Learn with Rabbil

DART KEYWORDS

abstract [2]	else	import [2]	show [1]
as [2]	enum	in	static [2]
assert	export [2]	interface [2]	super
async [1]	extends	is	switch
await [3]	extension [2]	late [2]	sync [1]
break	external [2]	library [2]	this
case	factory [2]	mixin [2]	throw
catch	false	new	true
class	final	null	try
const	finally	on [1]	typedef [2]

Learn with Rabbil

DART KEYWORDS

continue	for	operator [2]	var
covariant [2]	Function [2]	part [2]	void
default	get [2]	required [2]	while
deferred [2]	hide [1]	rethrow	with
do	if	return	yield [3]
dynamic [2]	implements [2]	set [2]	

Learn with Rabbil

DART VARIABLE

Variable is used to store the value and refer the memory location in computer memory.

- The variable cannot contain special characters such as whitespace, mathematical symbol, runes, Unicode character, and keywords.
- The first character of the variable should be an alphabet([A to Z],[a to z]). Digits are not allowed as the first character.
- Variables are case sensitive. For example, - variable age and AGE are treated differently.
- The special character such as #, @, ^, &, * are not allowed expect the underscore(_) and the dollar sign($).
- The variable name should be retable to the program and readable.

Learn with Rabbil

DART DATA TYPES

- Number
- Strings
- Boolean
- Lists
- Maps
- Runes
- Symbols

Learn with Rabbil

DART NUMBER

Integer

Integer values represent the whole number or non-fractional values.

Double

Double value represents the floating number or number with the large decimal points.

```
main() {
  var intNumber = 10;
  var doubleNumber=10.10;
  print(intNumber);
  print(doubleNumber);
}
```

Learn with Rabbil

DART STRING

- A string is the sequence of the character. If we store the data like – name, address, special character, etc.
- It is signified by using either single quotes or double quotes.

```
main() {
  var myStringSingle = 'This is a single quotes string';
  var myStringDouble = "This is a double quotes string";
  print(myStringSingle);
  print(myStringDouble);
}
```

Learn with Rabbil

DART BOOLEAN

- The Boolean type represents the two values - true and false.
- The bool keyword uses to denote Boolean Type.
- The numeric values 1 and 0 cannot be used to represent the true or false value.

```
main() {
  var negative = false;
  bool positive = true;
  print(negative);
  print(positive);
}
```

Learn with Rabbil

DART LISTS

- The list is a collection of the ordered objects (value). The concept of list is similar to an array.
- An array is defined as a collection of the multiple elements in a single variable.
- The elements in the list are separated by the comma enclosed in the square bracket[].

```
main() {
  var list = [1,2,3];
  print(list[0]);
}
```

Learn with Rabbil

DART MAPS

- The maps type is used to store values in key-value pairs. Each key is associated with its value.
- The key and value can be any type. In Map, the key must be unique, but a value can occur multiple times.
- The Map is defined by using curly braces ({}), and comma separates each pair.

```
main() {
  var student = {'name': 'Joseph',  'age':25, 'Branch': 'Computer Science'};
  print(student['name']);
}
```

Learn with Rabbil

DART OPERATORS

(Arithmetic Operators)

Sr.	Operator Name	Description	Example
1.	Addition(+)	It adds the left operand to the right operand.	a+b will return 30
2.	Subtraction(-)	It subtracts the right operand from the left operand.	a-b will return 10
3	Divide(/)	It divides the first operand by the second operand and returns quotient.	a/b will return 2.0
4.	Multiplication(*)	It multiplies the one operand to another operand.	a*b will return 200
5.	Modulus(%)	It returns a reminder after dividing one operand to another.	a%b will return 0
6.	Division(~/)	It divides the first operand by the second operand and returns integer quotient.	a/b will return 2
7.	Unary Minus(-expr)	It is used with a single operand changes the sign of it.	-(a-b) will return -10

Learn with Rabbil

DART OPERATORS

(Arithmetic Operators)

```dart
main() {
  var n1 = 10;
  var n2 = 5;

  print("n1+n2 = ${n1+n2}");
  print("n1-n2 = ${n1-n2}");
  print("n1*n2 = ${n1*n2}");
  print("n1/=n2 = ${n1/n2}");
  print("n1%n2 = ${n1%n2}");

}
```

Learn with Rabbil

DART OPERATORS

(Unary Operators)

Sr.	Operator Name	Description	Example
1.	++(Prefix)	It increment the value of operand.	++x
2.	++(Postfix)	It returns the actual value of operand before increment.	x++
3.	--(Prefix)	It decrement the value of the operand.	--x
4.	--(Postfix)	It returns the actual value of operand before decrement.	x--

DART OPERATORS

(Unary Operators)

```
main() {

  var x = 30;
  print(x++);

  var y = 25;
  print(++y);

  var z = 10;
  print(--z);

  var u = 12;
  print(u--);

}
```

Learn with Rabbil

DART OPERATORS

(Assignment Operator)

Operators	Name
= (Assignment Operator)	It assigns the right expression to the left operand.
+=(Add and Assign)	It adds right operand value to the left operand and resultant assign back to the left operand. For example - a+=b → a = a+b → 30
-=(Subtract and Assign)	It subtracts right operand value from left operand and resultant assign back to the left operand. For example - a-=b → a = a-b → 10
*=(Multiply and Assign)	It multiplies the operands and resultant assign back to the left operand. For example - a*=b → a = a*b → 200
/=(Divide and Assign)	It divides the left operand value by the right operand and resultant assign back to the left operand. For example - a%=b → a = a%b → 2.0
~/=(Divide and Assign)	It divides the left operand value by the right operand and integer remainder quotient back to the left operand. For example - a%=b → a = a%b → 2
%=(Mod and Assign)	It divides the left operand value by the right operand and remainder assign back to the left operand. For example - a%=b → a = a%b → 0

Learn with Rabbil

DART OPERATORS

(Relational Operator)

Sr.	Operator	Description
1.	>(greater than)	a>b will return TRUE.
2.	<(less than)	a<b will return FALSE.
3.	>=(greater than or equal to)	a>=b will return TRUE.
4.	<=(less than or equal to)	a<=b will return FALSE.
5.	==(is equal to)	a==b will return FALSE.
6.	!=(not equal to)	a!=b will return TRUE.

Learn with Rabbil

DART OPERATORS

(Type Test Operators)

Sr.	Operator	Description
1.	as	It is used for typecast.
2.	is	It returns TRUE if the object has specified type.
3.	is!	It returns TRUE if the object has not specified type.

Learn with Rabbil

DART OPERATORS

(Logical Operators)

Sr.	Operator	Description
1.	&&(Logical AND)	It returns if all expressions are true.
2.	||(Logical OR)	It returns TRUE if any expression is true.
3.	!(Logical NOT)	It returns the complement of expression.

Learn with Rabbil

DART OPERATORS

(Bitwise Operators)

Sr.	Operators	Description
1.	&(Binary AND)	It returns 1 if both bits are 1.
2.	|(Binary OR)	It returns 1 if any of bit is 1.
3.	^(Binary XOR)	It returns 1 if both bits are different.
4.	~(Ones Compliment)	It returns the reverse of the bit. If bit is 0 then the compliment will be 1.
5.	<<(Shift left)	The value of left operand moves left by the number of bits present in the right operand.
6.	>>(Shift right)	The value of right operand moves left by the number of bits present in the left operand.

Learn with Rabbil

DART CONSTANTS

Dart Constant is defined as an immutable object

- Which means it can't be changed or modified during the execution of the program.
- Once we initialize the value to the constant variable, it cannot be reassigned later.

The Dart constant can be defined in the following two ways.

- Using the final keyword
- Using the const keyword

```
main() {
  final a = 10;
  print(a);
}
```

```
main() {
  const a = 10;
  print(a);
}
```

Learn with Rabbil

DART CONSTANTS

Dart Constant is defined as an immutable object

- Which means it can't be changed or modified during the execution of the program.
- Once we initialize the value to the constant variable, it cannot be reassigned later.

The Dart constant can be defined in the following two ways.

- Using the final keyword
- Using the const keyword

```
main() {
  final a = 10;
  print(a);
}
```

```
main() {
  const a = 10;
  print(a);
}
```

Learn with Rabbil

LIST PROPERTIES

Property	Description
first	It returns the first element case.
isEmpty	It returns true if the list is empty.
isNotEmpty	It returns true if the list has at least one element.
length	It returns the length of the list.
last	It returns the last element of the list.
reversed	It returns a list in reverse order.
Single	It checks if the list has only one element and returns it.

FIXED LENGTH LIST

- The fixed-length lists are defined with the specified length.
- We cannot change the size at runtime.

```
void main(){
    const myList = [25, 63, 84];
    print(myList);
    //can't add item to fixed const list
    myList.add(96);
    print(myList);
}
```

Learn with Rabbil

GROWABLE LIST

- The list is declared without specifying size is known as a Grow able list.
- The size of the Grow able list can be modified at the runtime.

```
void main(){
    var myList = [25, 63, 84];
    print(myList);
    //add item to growable list
    myList.add(96);
    print(myList);
}
```

LIST INSERT

Dart provides four methods which are used to insert the elements into the lists. These methods are given below.

- add()
- addAll()
- insert()
- insertAll()

```dart
void main() {
  var myList = [1,3,5,7,9];
  print(myList);
  myList.addAll([11,13,15]);
  print(myList);
}
```

```dart
void main() {
  var myList = [1,3,5,7,9];
  print(myList);
  myList.add(11);
  print(myList);
}
```

```dart
void main() {
  var myList = [1,3,5,7,9];
  print(myList);
  myList.insert(2,10);
  print(myList);
}
```

```dart
void main() {
  var myList = [1,3,5,7,9];
  print(myList);
  myList.insertAll(2,[0,0,0]);
  print(myList);
}
```

Learn with Rabbil

UPDATING LIST

list_name[index] = new_value

```
void main() {
  var list1 = [10,15,20,25,30];
  print(list1);

  list1[3] = 55;
  print(list1);
}
```

REMOVING LIST ELEMENTS

- remove() removeAt() removeLast() removeRange()

```
void main() {
  var list1 = [10,15,20,25,30];
  print(list1);
  list1.removeLast();
  print(list1);
}
```

```
void main() {
  var list1 = [10,15,20,25,30];
  print(list1);
  list1.remove(20) ;
  print(list1);
}
```

```
void main() {
  var list1 = [10,15,20,25,30];
  print(list1);
  list1.removeAt(3) ;
  print(list1);
}
```

Learn with Rabbil

DART SET

- The Dart Set is the unordered collection of the different values of the same type
- It has much functionality, which is the same as an array, but it is unordered.
- Set doesn't allow storing the duplicate values.
- Set must contain unique values.

ADD ELEMENT INTO SET

- The Dart provides the two methods add() and addAll() to insert an element into the given set.

```
void main() {
  var names = <String>{"James","Ricky", "Devansh","Adam"};
  names.add("Jonathan");
  print(names);
}
```

```
void main() {
  var names = <String>{"James","Ricky", "Devansh","Adam"};
  names.addAll({"A","B"});
  print(names);
}
```

ACCESS THE SET ELEMENT

- Dart provides the elementAt() method, which is used to access the item by passing its specified index position.

```
void main() {
  var names = <String>{"James","Ricky", "Devansh","Adam"};
  var x = names.elementAt(3);
  print(x);
}
```

DART REMOVE ALL SET ELEMENT

- We can remove entire set element by using the clear() methods.

```
void main() {
  var names = <String>{"James","Ricky", "Devansh","Adam"};
  names.clear();
  print(names);
}
```

Learn with Rabbil

DART SET PROPERTIES

Properties	Explanations
first	It is used to get the first element in the given set.
isEmpty	If the set does not contain any element, it returns true.
isNotEmpty	If the set contains at least one element, it returns true
length	It returns the length of the given set.
last	It is used to get the last element in the given set.
hashcode	It is used to get the hash code for the corresponding object.
Single	It is used to check whether a set contains only one element.

Learn with Rabbil

DART MAP

- Dart Map is an object that stores data in the form of a key-value pair.
- Each value is associated with its key, and it is used to access its corresponding value.
- Both keys and values can be any type.
- In Dart Map, each key must be unique, but the same value can occur multiple times.
- Dart Map can be defined in two methods.
- Using Map Literal
- Using Map Constructor

DART MAP USING MAP LITERAL

- To declare a Map using map literal, the key-value pairs are enclosed within the curly braces "{}" and separated by the commas.

```
void main() {
  var student = {'name':'Tom','age':'23'};
  print(student);
}
```

ADDING VALUE AT RUNTIME

- To declare a Map using map literal, the key-value pairs are enclosed within the curly braces "{}" and separated by the commas.

```
void main() {
  var student = {'name':' tom', 'age':23};
  student['course'] = 'B.tech';
  print(student);
}
```

USING MAP CONSTRUCTOR

- To declare the Dart Map using map constructor can be done in two ways.
- First, declare a map using map() constructor. Second, initialize the map.

```
void main() {
  var student = new Map();
  student['name'] = 'Tom';
  student['age'] = 23;
  student['course'] = 'B.tech';
  student['Branch'] = 'Computer Science';
  print(student);
}
```

MAP PROPERTIES

Properties	Explanation
Keys	It is used to get all keys as an iterable object.
values	It is used to get all values as an iterable object.
Length	It returns the length of the Map object.
isEmpty	If the Map object contains no value, it returns true.
isNotEmpty	If the Map object contains at least one value, it returns true.

MAP METHODS

Learn with Rabbil

- addAll() - It adds multiple key-value pairs of other. The syntax is given below.

```
void main() {
  Map student = {'name':'Tom','age': 23};
  print(student);
  student.addAll({'dept':'Civil','email':'tom@xyz.com'});
  print(student);
}
```

- Map.clear() - It eliminates all pairs from the map.

```
void main() {
  Map student = {'name':'Tom','age': 23};
  print(student);
  student.addAll({'dept':'Civil','email':'tom@xyz.com'});
  student.clear();
  print(student);
}
```

MAP METHODS

- remove() - It removes the key and its associated value if it exists in the given map.

```
void main() {
  Map student = {'name':'Tom','age': 23};
  print(student);
  student.remove('age');
  print(student);
}
```

Learn with Rabbil

DART CONTROL FLOW STATEMENT

The control statements or flow of control statements are used to control the flow of Dart program. In Dart, Control flow statement can be categorized mainly in three following ways.

- Decision-making statements
- Looping statements
- Jump statements

DART DECISION-MAKING STATEMENTS

The Decision-making statements allow us to determine which statement to execute based on the test expression at runtime. Dart provides following types of Decision-making statement.

- If Statement
- If-else Statements
- If else if Statement
- Switch Case Statement

IF STATEMENT

If statement allows us to a block of code execute when the given condition returns true.

```
void main() {
   var n = 35;
   if (n<40){
     print("The number is smaller than 40");
   }
}
```

IF-ELSE STATEMENTS

In Dart, if-block is executed when the given condition is true. If the given condition is false, else-block is executed.

```
void main() {
 var x = 20;
 var y = 30;
 if(x > y){
  print("x is greater than y");
 } else {
  print("y is greater than x");
 }
}
```

Learn with Rabbil

IF ELSE-IF STATEMENT

- Dart if else-if statement provides the facility to check a set of test expressions and execute the different statements.
- It is used when we have to make a decision from more than two possibilities.

```
void main() {
 var marks = 74;
 if(marks > 85)
 {
  print("Excellent");
 }
 else if(marks>75)
 {
  print("Very Good");
 }
 else if(marks>65)
 {
  print("Good");
 }
 else
 {
  print("Average");
 }
}
```

SWITCH CASE STATEMENT

- Dart Switch case statement is used to avoid the long chain of the if-else statement.
- It is the simplified form of nested if-else statement.

```
void main() {
 int n = 3;
 switch (n) {
  case 1:
   print("Value is 1");
   break;
  case 2:
   print("Value is 2");
   break;
  case 3:
   print("Value is 3");
   break;
  case 4:
   print("Value is 4");
   break;
  default:
   print("Out of range");
   break;
 }
}
```

Learn with Rabbil

DART LOOPING STATEMENTS

Dart Loop is used to run a block of code repetitively for a given number of times or until matches the specified condition.

- Dart for loop
- Dart for…in loop
- Dart while loop
- Dart do-while loop

DART FOR LOOP

The for loop is used when we know how many times a block of code will execute.

```
for(int i=0; i<=10; i++){
 print(i);
}
```

DART FOR... IN LOOP OVER LIST

The for...in loop is slightly different from the for loop. It only takes dart object or expression as an iterator and iterates the element one at a time.

```
var list1 = [10,20,30,40,50];
for(var i in list1) {
   print(i);
}
```

DART FOR... IN LOOP OVER MAP (JSON ARRAY)

```
var student = [
{'name':'Rain', 'age':36},
{'name':'Rupom', 'age':31},
{'name':'Rifat', 'age':18}
];

for(var i in student) {
   var name=i['name'];
   print(name);
}
```

DART FOR... IN LOOP OVER SET

```
void main() {
 var names ={"James","Ricky", "Devansh","Adam"};
 for(var i in names) {
    print(i);
 }
}
```

FUNCTION PARTS

return_type - It can be any data type such as void, integer, float, etc. The return type must be matched with the returned value of the function.

func_name - It should be an appropriate and valid identifier.

parameter_list - It denotes the list of the parameters, which is necessary when we called a function.

return value - A function returns a value after complete its execution.

```
int myFunc(int a, int b){
 int c;
 c = a+b;
 return c;
}

 void main() {
  var res=myFunc(10,20);
  print(res);
 }
```

FUNCTION PARTS

return_type - It can be any data type such as void, integer, float, etc. The return type must be matched with the returned value of the function.

func_name - It should be an appropriate and valid identifier.

parameter_list - It denotes the list of the parameters, which is necessary when we called a function.

return value - A function returns a value after complete its execution.

```
int myFunc(int a, int b){
  int c;
  c = a+b;
  return c;
}

void main() {
   var res=myFunc(10,20);
   print(res);
}
```

DEFINING A FUNCTION

- A function can be defined by providing the name of the function with the appropriate parameter and return type.
- A function contains a set of statements which are called function body.

```
myFunc(){


}
```

CALLING A FUNCTION

After creating a function, we can call or invoke the defined function inside the main() function body

```
myFunc(){

}
void main() {
 myFunc();
}
```

PASSING ARGUMENTS TO FUNCTION

When a function is called, it may have some information as per the function prototype is known as a parameter (argument).

```
myFunc(String Name){
 print(Name);
}
void main() {
 myFunc("RABBIL");
}
```

FUNCTION RETURN & RETURN TYPE

- It can be any data type such as void, integer, float, etc. The return type must be matched with the returned value of the function.

- A function returns a value after complete its execution.

```
int myFunc(int a, int b){
 int c;
 c = a+b;
 return c;
}
```

```
void main() {
 var res=myFunc(10,20);
 print(res);
}
```

Learn with Rabbil

THE MAIN() FUNCTION

- The main() function is the top-level function of the Dart.
- It is the most important and vital function of the Dart programming language.
- The execution of the programming starts with the main() function.
- The main() function can be used only once in a program.

```
void main() {


}
```

Learn with Rabbil

FUNCTION RECURSION

- Dart Recursion is the method where a function calls itself as its subroutine.
- It is used to solve the complex problem by dividing it into sub-part.
- A function which is called itself again and again or recursively, then this process is called recursion

```
myFunc(){
 print("I am called recursion ");
 myFunc();
}
```

```
void main() {
 myFunc();
}
```

Learn with Rabbil

DART OBJECT-ORIENTED CONCEPTS

Dart is an object-oriented programming language, and it supports all the concepts of object-oriented programming such as classes, object, inheritance, mixin, and abstract classes.

- Class
- Object
- Inheritance
- Polymorphism
- Interfaces
- Abstract class

Learn with Rabbil

DART CLASS

- Dart classes are defined as the blueprint of the associated objects.
- A Class is a user-defined data type that describes the characteristics and behavior of it.
- To get all properties of the class, we must create an object of that class.

```
class MyClass {
 var myName="Rabbil Hasan";
}

void main() {
  var MyClassObj=new MyClass();
  print(MyClassObj.myName);
}
```

Learn with Rabbil

ACCESSING VARIABLE FROM CLASS

```
class MyClass {
  var myName="Rabbil Hasan";
}

void main() {
   var MyClassObj=new MyClass();
   print(MyClassObj.myName);
}
```

ACCESSING FUNCTION FROM CLASS

```
class MyClass {
 addTwoNumber(var a,var b) {
  var c=a+b;
  print(c);
 }
}

void main() {
  var MyClassObj=new MyClass();
  MyClassObj.addTwoNumber(10,20);
}
```

Learn with Rabbil

ACCESSING STATIC VARIABLE FROM CLASS

```
class MyClass {
  static var myName="Rabbil Hasan";
}

void main() {
  print(MyClass.myName);
}
```

ACCESSING STATIC FUNCTION FROM CLASS

```
class MyClass {
 static addTwoNumber(var a,var b) {
  var c=a+b;
  print(c);
 }
}

void main() {
  MyClass.addTwoNumber(10,20);
}
```

Learn with Rabbil

CLASS CONSTRUCTOR

A constructor is a different type of function which is created with same name as its class name.
The constructor is used to initialize an object when it is created.

- Constructor has no return type
- Constructor can have parameter
- Constructor execute automatically

```
class MyClass {
 MyClass(){
   print("I am a constructor");
 }
}

void main() {
  new MyClass();
}
```

DART THIS KEYWORD

- The this keyword is used to refer the current class object.
- It indicates the current instance of the class, methods, or constructor.

```
class student {
  var name="Rabbil";
  fun(){
    print(this.name);
  }
}
void main() {
  var obj=new student();
  obj.fun();
}
```

DART INHERITANCE

- Dart inheritance is defined as the process of deriving the properties and characteristics of another class.
- It provides the ability to create a new class from an existing class.
- It is the most essential concept of the oops(Object-Oriented programming approach).
- We can reuse the all the behavior and characteristics of the previous class in the new class.

Parent Class:

A class which is inherited by the other class is called superclass or parent class. It is also known as a base class.

Child Class:

A class which inherits properties from other class is called the child class. It is also known as the derived class or subclass.

```dart
class Father {
  var FatherTitle="Islam";
  FatherAsset(){
    print("House,Land");
  }
}
class Son extends Father {
  SonsAsset(){
    print(FatherTitle);
  }
}

void main() {
  var obj=new Son();
  obj.SonsAsset();
}
```

METHOD OVERRIDING

- When we declare the same method in the subclass, which is previously defined in the superclass is known as the method overriding.

- The subclass can define the same method by providing its own implementation, which is already exists in the superclass.

- The method in the superclass is called method overridden, and method in the subclass is called method overriding.

```
class Father {
  var FatherTitle="Islam";
  FatherAsset(){
    print("House,Land");
  }
}
class Son extends Father {
  FatherAsset(){
    print("House,Land,Gold");
  }
}

void main() {
  var obj=new Son();
  obj.FatherAsset();
}
```

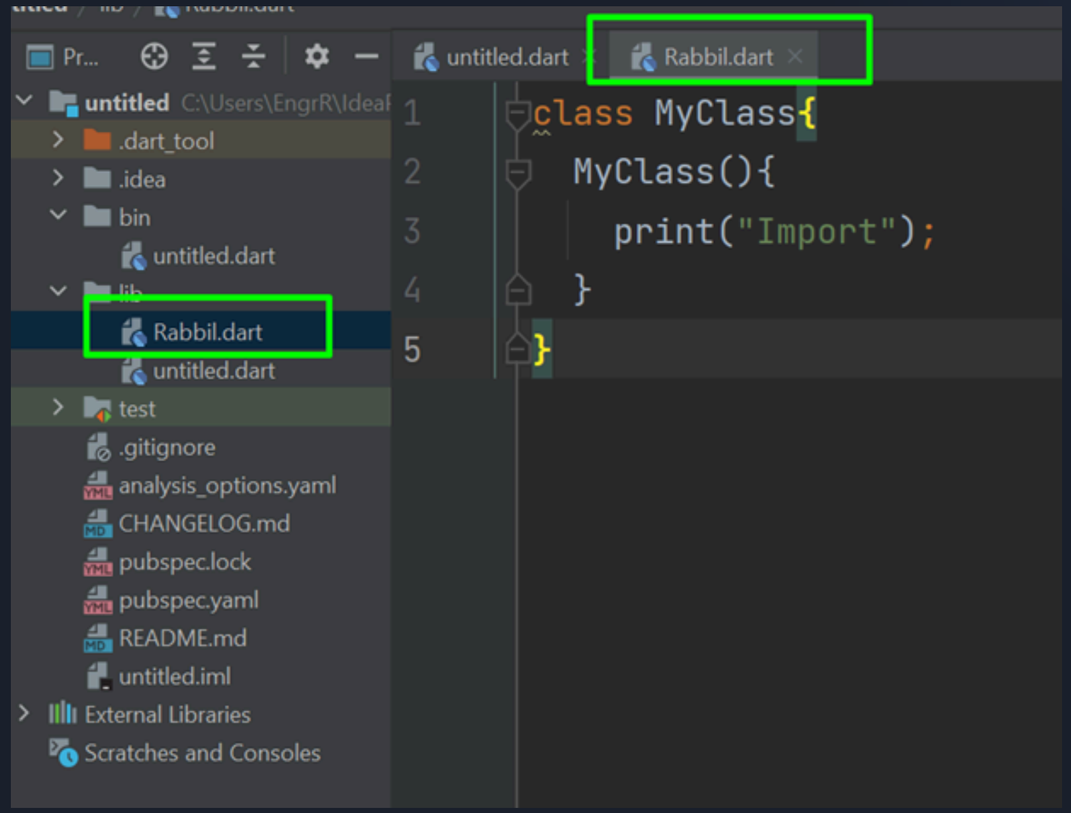
DART ABSTRACT CLASSES

- Abstract classes are the classes in Dart that has one or more abstract method.
- Abstraction is a part of the data encapsulation where the actual internal working of the function hides from the users.
- They interact only with external functionality.
- We can declare the abstract class by using the abstract keyword.
- There is a possibility that an abstract class may or may not have abstract methods.

```dart
abstract class Father {
  var FatherTitle="Islam";
  FatherAsset(){
    print("House,Land");
  }
}
class Son extends Father {
  FatherAsset(){
    print("House,Land,Gold");
  }
}

void main() {
  var obj=new Son();
  obj.FatherAsset();
}
```

DART IMPORT CODE FORM EXTERNAL FILES




```dart
import 'package:untitled/Rabbil.dart';
void main() {
 var obj= MyClass();
}
```

DART DEBUGGING

Learn with Rabbil

WHY DEBUGGING IS IMPORTANT

- To find out my mistake
- To test/check my code , that works well
- To understand complex program flow
- To works with complex action part by part
- To improve my code