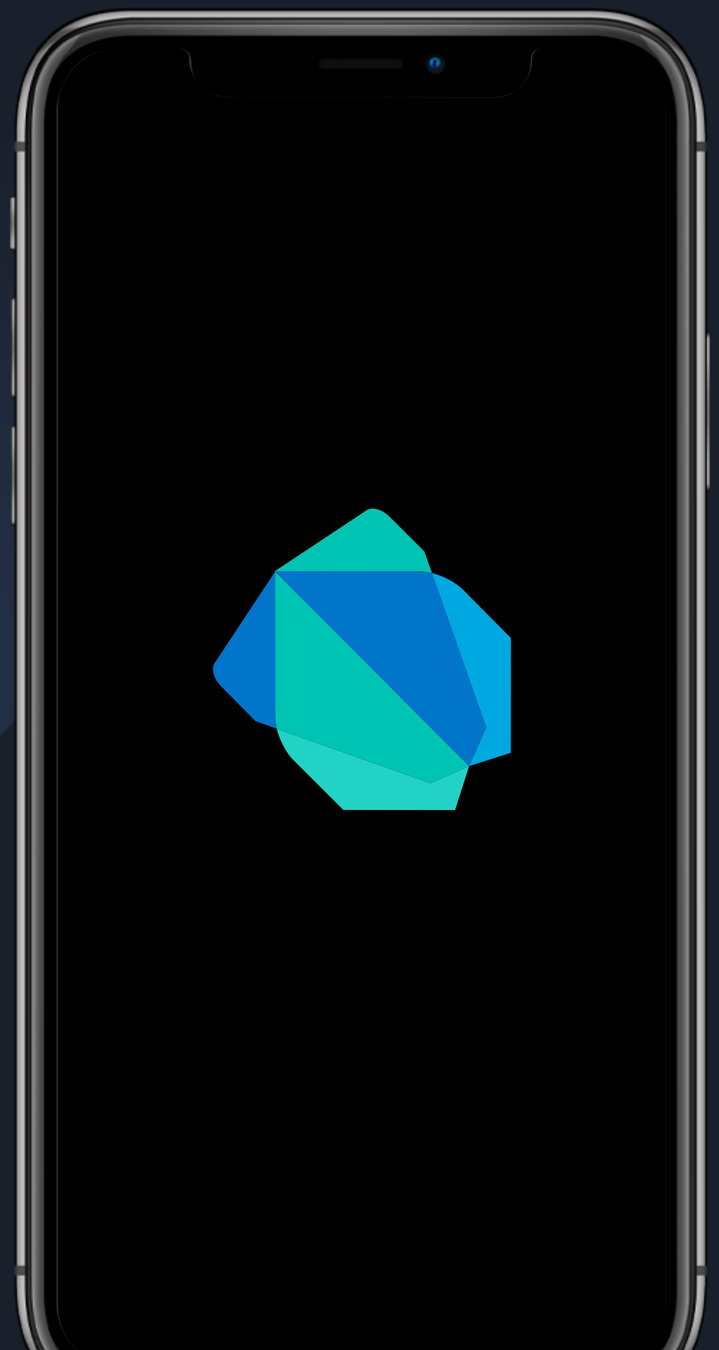
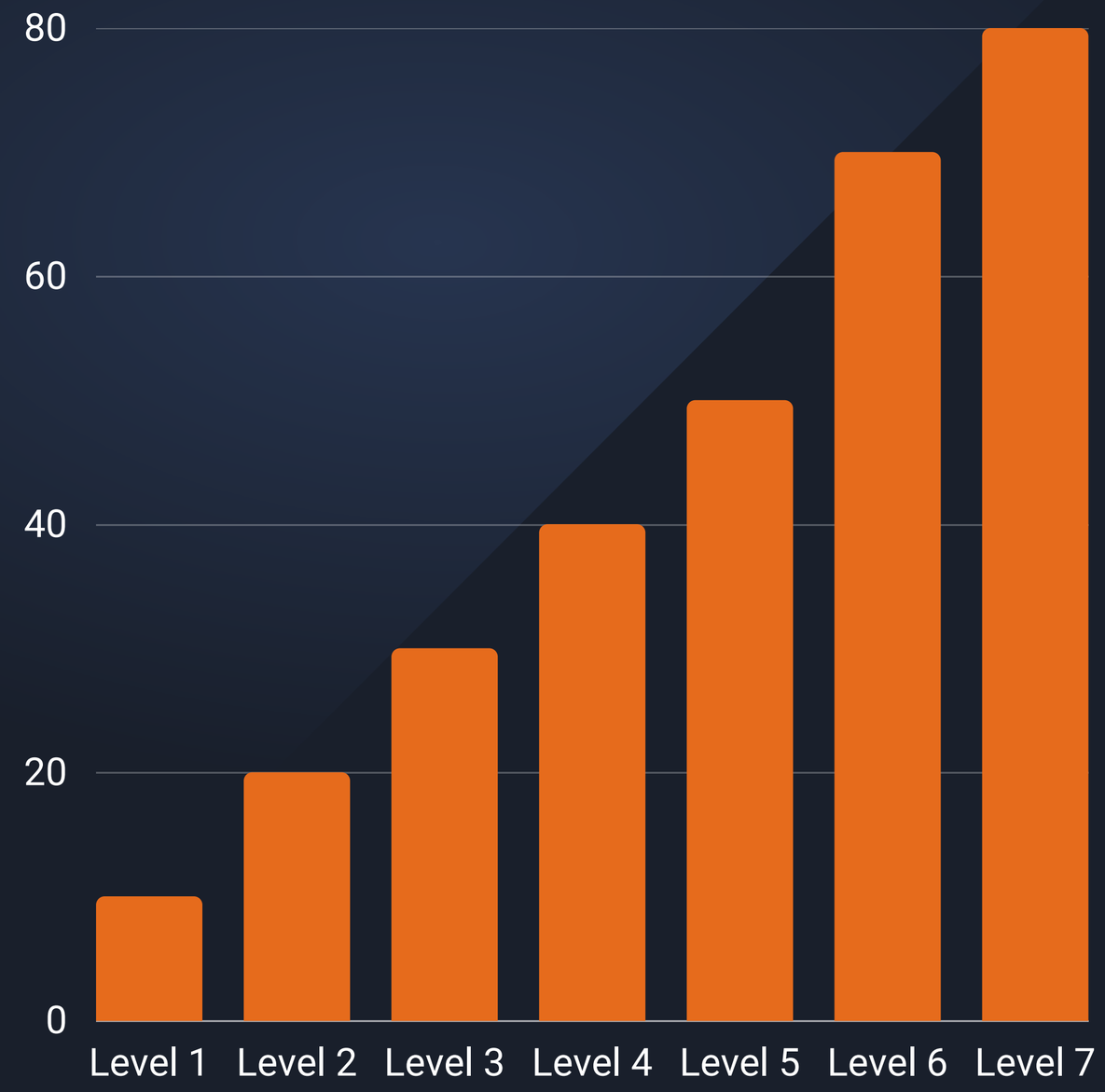
DART

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



YOUR UNDERSTANDING

Will Increase gradually



BEGINNER STAGE

Don’t Dive Deep When You Are Beginner

DART

Environment Setup

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

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');
}
```

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

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 (/* */)

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]

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]	

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.

DART DATA TYPES

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

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);
}
```

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);
}
```

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);
}
```

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]);
}
```

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']);
}
```

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

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}");

}
```

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--);

}
```

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

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.

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.

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.

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.

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);
}
```

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);
}
```

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);
}
```

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);
}
```

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);
}
```