MOBILE APPLICATION DEVELOPMENT

Dart Introduction 1

Week 2

Wake up activity – Know each other

- Introduce yourself (your name and your nickname)
- Share your dream job after graduating from the university.
- Then select your next friend to introduce.

Instructor

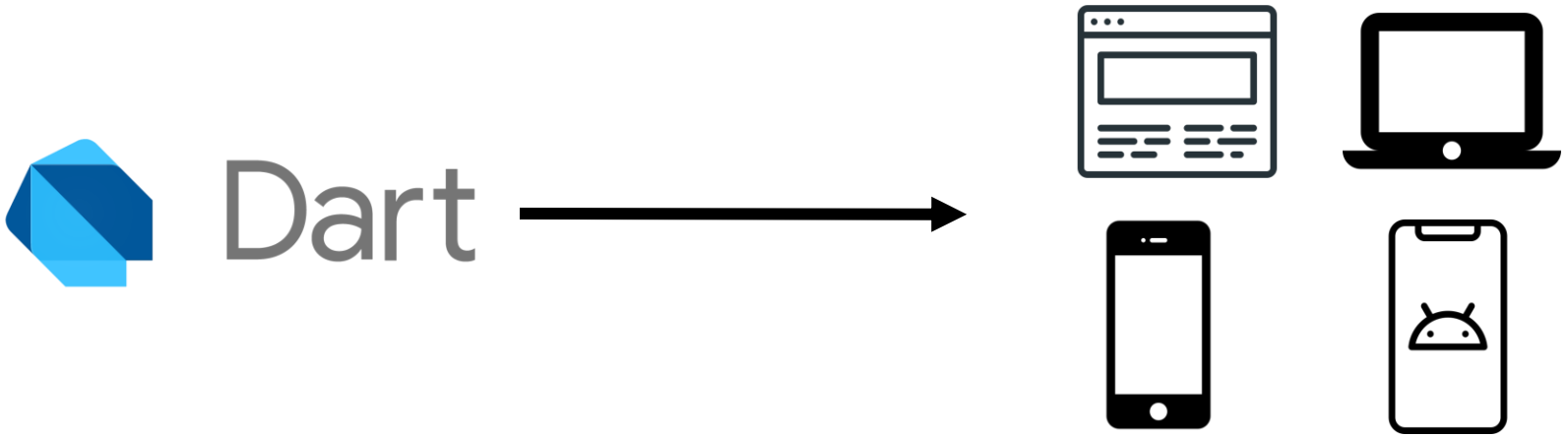
- Aj. App – Akkapon Somjai
- Senior Software Developer at Palo IT
- Discord Username: asomjai
- Email Address : asomjai@palo-it.com

Contents

- Introduction to Dart language
- Write your first program with Dart
- Data Types
- Null Safety
- Standard Input & Output
- Condition Statement
- Looping Statement
- Function

Introduction to Dart language

- Dart is a programming language developed by Google
- Dart is used to create web, mobile, and desktop applications
- Dart code can be compiled to native code which are JavaScript (Web), Java (Android), and Swift (iOS).
- This allows developers to create high-performance, cross-platform mobile apps using a single codebase



Write your first program with Dart

- Create dart_project directory.
- Open it with VSCode.
- Create hello_world.dart file.
- Write the code as below.

```
void main() {
  print("Hello World!");
}
```

- Click on "Run" command above the main() to compile and run.
- Check the result on the "DEBUG CONSOLE" window in VS code.

Checkpoint 1 : Tell me what result you see.

Data Types

- int
- double
- num
- String
- List
- Map
- dynamic

Data Types - int

- "int" stands for "integer," which is a whole number without a decimal point. For example, 1, -5, and 42 are all integers.
- You can also perform arithmetic operations on integers, such as addition, subtraction, multiplication, and division. For example:

```
int a = 10;
int b = 5;
int c = a + b; // c = 15
int d = a - b; // d = 5
int e = a * b; // e = 50
int f = a ~/ b; // f = ?
```

Data Types - double

- "double" is a data type that represents a number with a decimal point. For example, 3.14, -2.718, and 0.5 are all examples of numbers that can be represented as a double
- You can also perform arithmetic operations on double, such as addition, subtraction, multiplication, and division. For example:

```
double a = 10.5;
double b = 5.5;
double c = a + b; // c = 16.0
double d = a - b; // d = 5.0
double e = a * b; // e = 57.75
double f = a / b; // f = 1.9090909090909092
```

Data Types - num

- "num" is a data type that represents a number (either an integer or a floating-point number). The **num** type is the parent class of both **int** and **double** data types.
- You can use the **num** data type when you don't know if a variable will be an integer or a double, or when you want to perform operations that involve both integers and doubles. For example:

```
num x = 3;
num y = 3.14;
num z = x + y;
```

Data Types - boolean

- "boolean" (or "bool" for short) is a data type that can hold only two values: **true** or **false**.
- Boolean values are often used in programming to represent true or false conditions, such as whether a certain statement is true or false, or whether a certain condition has been met.
- You can create a boolean variable by declaring its name and assigning it a value:

```
bool myBoolean = true;
```

- You can also perform logical operations on boolean variables, such as "and (&&)", "or (||) ", and "not (!)". For example:

```
bool a = true;
bool b = false;
bool c = a && b; // c = false
bool d = a || b;   // d = true
bool e = !a;       // e = false
```

Data Types - String

- a "string" is a sequence of characters that can be used to store and manipulate text. For example, you could use a string to store a person's name, or the contents of a message. You can create a string by enclosing a series of characters in quotation marks, like this:

```
String myString = "Hello, world!";
```

Data Types - String

- You can also use the plus operator(+) to concatenate or add two or more strings:

```
String firstName = "John";
String lastName = "Doe";
String fullName = firstName + " " + lastName;   // John Doe
```

Data Types - String

- You can also use the $ or ${} to inject to a string by other variables.

```
String firstName = "John";
String lastName = "Doe";
String fullName = "$firstName $lastName"; // John Doe

int userId = 1;
String userIdAndfullName = "$userId - $fullName"  // 1 – John Doe
```

Data Types - List

- "List" (also known as an "array" in some other programming languages) is a data type that represents a collection of items. Each item in a list can be of any data type, including other lists.
- You can create a list by declaring its name and assigning it a value, using square brackets **[]** to enclose the items, and separating them by commas:

```
List<int> myList = [1, 2, 3, 4, 5];
```

Data Types - List

- You can also create a list with the **List()** constructor and add items to it using the **add()** method

```
List<String> myList = [];
myList.add("apple");
myList.add("banana");
myList.add("orange");
```

Data Types - List

- You can access the items of a list by their index, which is the position of the item in the list, starting from 0. For example:

```
String firstItem = myList[0]; // firstItem = apple
String secondItem = myList[1]; // secondItem = banana
```

Data Types - Map

- a "Map" is a data type that represents a collection of key-value pairs. A key is an unique identifier that is used to access its corresponding value. The key and value can be of any data type.
- You can create a map by declaring its name and assigning it a value using curly braces **{}**, to enclose the key-value pairs, and separating them by colons **:**

```
Map<String, int> myMap = {
  "apple": 1,
  "banana": 2,
  "orange": 3
};
```

Data Types - Map

- You can also create a map with the **Map()** constructor and add/change items to it using the **[]** operator

```
Map<String, int> myMap = {};
myMap["apple"] = 1;
myMap["banana"] = 2;
myMap["orange"] = 3;
```

Data Types - dynamic

- "dynamic" is a special data type that can hold any value, and the type of the value will be determined at runtime.
- When you declare a variable as dynamic, you can assign any type of value to it and the variable will automatically adapt to the type of the value at runtime.

```
dynamic myVar = "hello";
myVar = 10;
```

Data Types

- Checkpoint 2: Complete this code to display the output as below
- Code:

```
void main() {
   String studentName = "Alexander Mohamad";
   List<int> scores = [8,30,17,18];

   // TODO : Add your code here
}
```

- Output:

  Alexander Mohamad's total score is 73.

Standard Input & Output

- The process of reading input data from an external source (such as a user or a file) and writing output data to an external destination (such as a screen or a file).
- In Dart, standard input and output is typically performed using the stdin and stdout streams, which are part of the **dart:io** library.
- To write the output, we can use **print()** or using the stdout stream's writeln**()** method to display the result on the screen.
- as we did in the first program to display the result on the screen. To read input from the user, you can use the **stdin** stream's **readLineSync()** method. This method reads a line of text from the input stream and returns it as a string:

```
import 'dart:io';

void main() {
  print("What is your name?");
  String? name = stdin.readLineSync();
  print("Hello, $name!");
}
```

Standard Input & Output

- To run the code that receive the input by stdin.**readLineSync**(), we need to execute the script through dart command in terminal:

dart <file-name>

Example:
dart example_io.dart

Null Safety - The problem with null

In programming, null refers to the absence of a value. When a variable is declared but not initialized, its value is null. This can lead to null reference exceptions, which occur when an application tries to call a method or access a property of a null object.

```
String name = null;     // This is NOT allowed by compiler
```

Null Safety – Nullable Type

Dart introduced null safety feature to prevent these types of errors by clearly distinguishing between nullable and non-nullable types. A nullable type is a type that can have a value of null, while a non-nullable type cannot. For example, the **String** type is non-nullable and cannot have a value of null, whereas **String?** is nullable and can have a value of null.

```
String? name = null;     // This is allowed by compiler
```

Null Safety – Operators

To help developers work with nullable types, Dart introduced several new operators, such as the null-aware operator (**?.**) and the null-coalescing operator (**??**). These operators allow developers to safely access properties and methods of nullable variables without the risk of null reference exceptions.

Example Code:

```
// Receive the input from the user which we don't know that is null or not.
String? studentScore = stdin.readLineSync();
int studentScoreInt = int.parse(studentScore ?? "0");

// The scoreDigits will be 0 if studentScore is null
int scoreDigits = studentScore?.length ?? 0;
```

Null Safety

Checkpoint 3: Write source code to receive the student name and his/her score from standard input (**readLineSync()**) then display student name and his score in the terminal as below:

Please answer the questions below:
1.What is student name?
Akkapon
2.How many score does he/she have?
76
Result: Akkapon has 76 scores.

```
Please answer the questions below:
1.What is student name?
Akkapon
2.How many score does he/she have?
76
Akkapon has 76 scores
```

Remark: Akkapon and 76 are input from typing in the terminal.

Condition Statement

- if-else
- Ternary
- switch

Condition Statement – if-else

- if-else statements are used to make decisions in a program
- The if-else statement is used to execute a block of code if a certain condition is true and another block of code if the condition is false.
- The condition is a boolean expression that evaluates to either true or false.

```
if (condition) {
  // code to be executed if condition is true
} else {
  // code to be executed if condition is false
}
```

Condition Statement – if-else

- It's also possible to nest multiple if-else statements, allowing for more complex decisions to be made

```
if(condition1){
  // code to be executed if condition1 is true
}  else if (condition2){
  // code to be executed if condition1 is false and condition2 is true
} else {
  // code to be executed if both condition1 and condition2 are false
}
```

Condition Statement - Comparison Operators

- Comparison operators are used in conditional expressions to compare two values
- Common comparison operators include:
  - == (equal to)
  - != (not equal to)
  - > (greater than)
  - < (less than)
  - >= (greater than or equal to)
  - <= (less than or equal to)

Condition Statement – Ternary

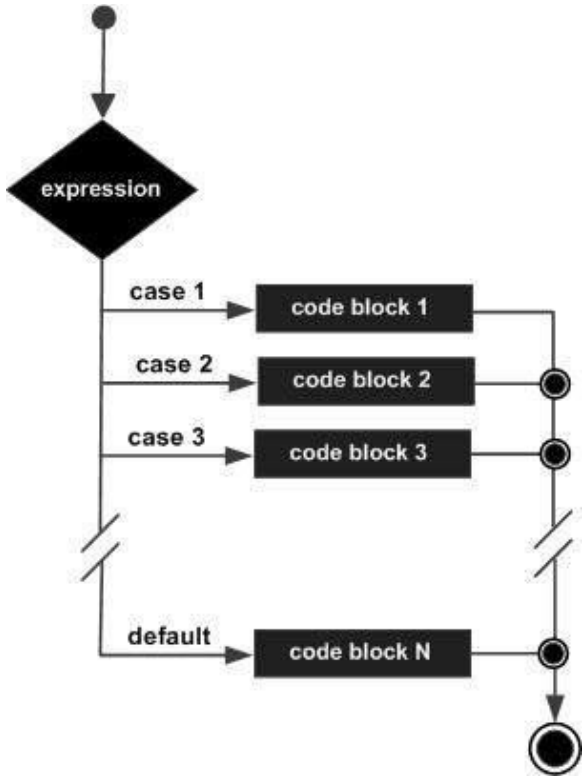
- A ternary expression in Dart is a shorthand way of writing an "if-else" statement. It is a compact way to express a simple decision-making logic.
- The syntax of a ternary expression is as follows:

```
bool isHeStudent;
If(score > 49){
    isHeStudent = true;
} else {
    isHeStudent = false;
}

// Equal to:
bool isStudentPassed = score > 49 ? true : false
```

Condition Statement – Switch Case

- In Dart, switch-case statements are a simplified version of the nested if-else statements.



Condition Statement – Switch Case

```
//example of switch case statement using integers
int age = 25;
switch (age) {
  case 18:
    print("You are 18 years old");
    break;
  case 25:
    print("You are 25 years old");
    break;
  default:
    print("Age not specified");
```

Condition Statement

- Checkpoint 4: Extend the script from the checkpoint 3 to check score and give him/her a grade as the condition below
  - If score is higher than or equal to 80, student will get A grade
  - if score is higher than or equal to 70, student will get B grade
  - if score is higher than or equal to 60, student will get C grade
  - if score is higher than or equal to 50, student will get D grade
  - Otherwise, the student will get E grade
- Hint : You can use **int.parse()** to convert String from the standard input into integer value.

- Example Output:

Please answer the questions below:

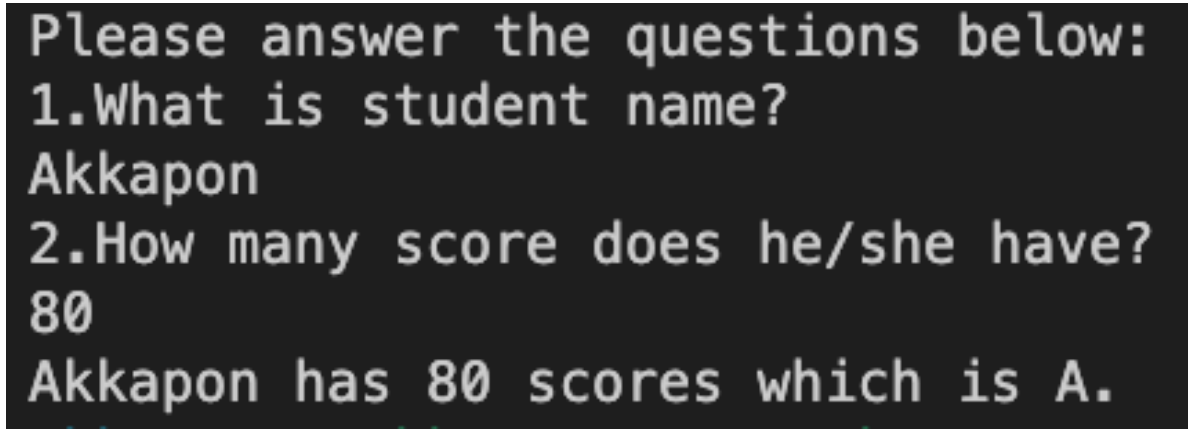
1.What is student name?

Akkapon

2.How many score does he/she have?

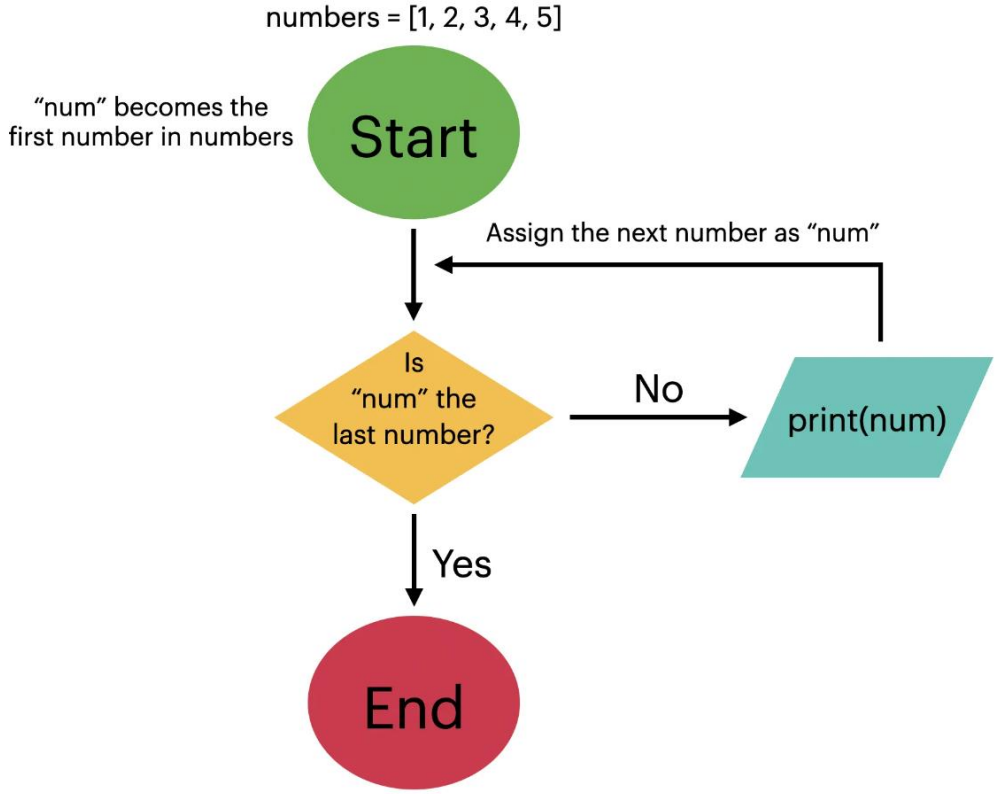
80

Akkapon has 80 scores which is A grade.



Looping Statement - Concept

- Looping statements are a way for a computer program to repeat a certain action multiple times. Think of it like a person doing a task multiple times, but instead of a person, it's the computer doing it.

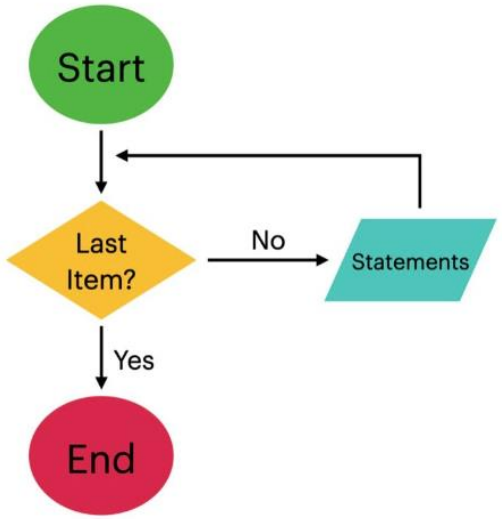


Looping Statement

- for
- for-in
- while
- do-while

Looping Statement – for Loop

- The for loop is used to execute a block of code a specific number of times
- The for loop has three parts: the initialization, the condition, and the increment/decrement
- The syntax for a for loop is as follows:




```
for (int i = 0; i < 10; i++) {
  // statements to be executed
}
```

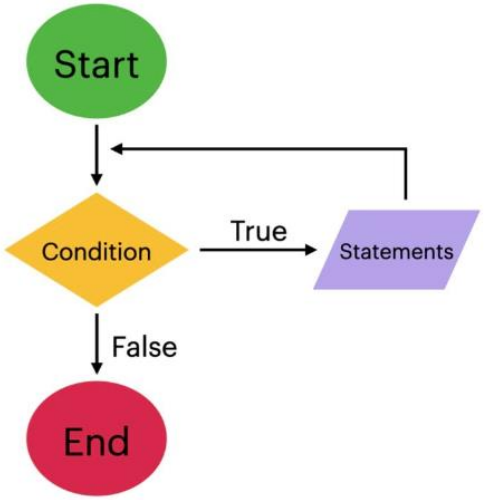
Looping Statement – for-in

- The for-in loop is a type of loop in Dart that allows you to iterate over a collection of items, such as a list or an array
- The for-in loop is a concise and easy to use looping construct that can make code more readable and efficient.

```
var fruits = ["Apple", "Banana", "Orange"];
for (var fruit in fruits) {
  print(fruit);
}
```

Looping Statement – while Loop

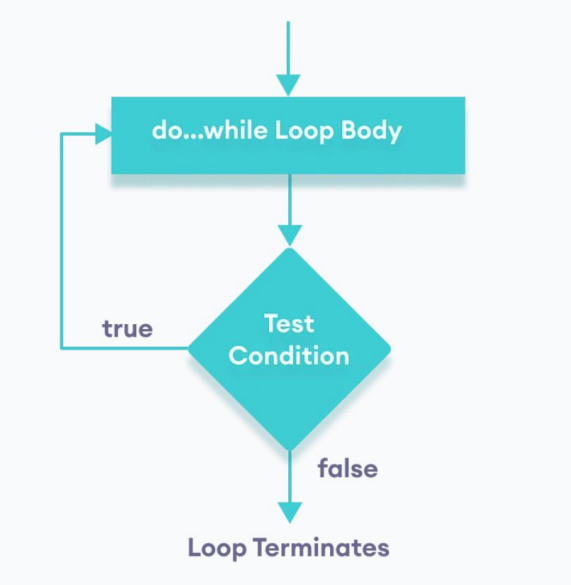
- The while loop is used to execute a block of code while a certain condition is true
- The syntax for a while loop is as follows:




```
while (condition) {
  // statements to be executed
}
```

Looping Statement – do-while Loop

- The do-while loop is similar to the while loop, but the code inside the loop is executed at least once before the condition is checked
- The syntax for a do-while loop is as follows:




```
do {
  // code to be executed
} while (condition);
```

Looping Statement

- Checkpoint 5: Refactor the source code from checkpoint 2 by using for, for in, while or do-while to traverse in each item of List then sum the total and print the same out as we did in checkpoint 2
- Example Output : Alexander Mohamad's total score is 73.
- Hint : You can call **scores.length** to get the size of list

Function

- Functions are blocks of code that can be executed multiple times
- Functions are a fundamental concept in programming, used to organize and reuse code
- A function in Dart is defined using the **void** or **FunctionType** keyword, followed by the name of the function, a set of parentheses, and a set of curly braces

```
void functionName() {
  // code to be executed
}
```

Function - Arguments

- Functions can take parameters, which are variables that are passed to the function when it is called
- The values passed to the function when it is called are called arguments

```
void functionName(int parameter1, String parameter2) {
  // code to be executed
}
```

Function – Named parameters

- Named parameters are a way to provide more meaningful names to the parameters when calling a function in Dart
- Named parameters can make code more readable and easier to understand. It is also more flexible in case that you change the order of parameter.
- To specify a named parameter, the parameter name is preceded by the **@required** keyword and followed by a colon

```
void printName({@required String name, int age}) {
   print("$name age is $age");
}
printAge(age: 25,name:"Alex");
```

Function – Returning Data

- Functions can also return a value using the **return** keyword
- The syntax for defining a function with a return value is as follows:

```
int functionName(){
  return 1;
}
```

Function - Calling

- The syntax for calling a function with arguments is as follows:

```
// Calling the function without storing returned value
functionName(1, "argument");

// Declare a variable to store returned value from the function
int returnValue = functionName(1, "argument");
```

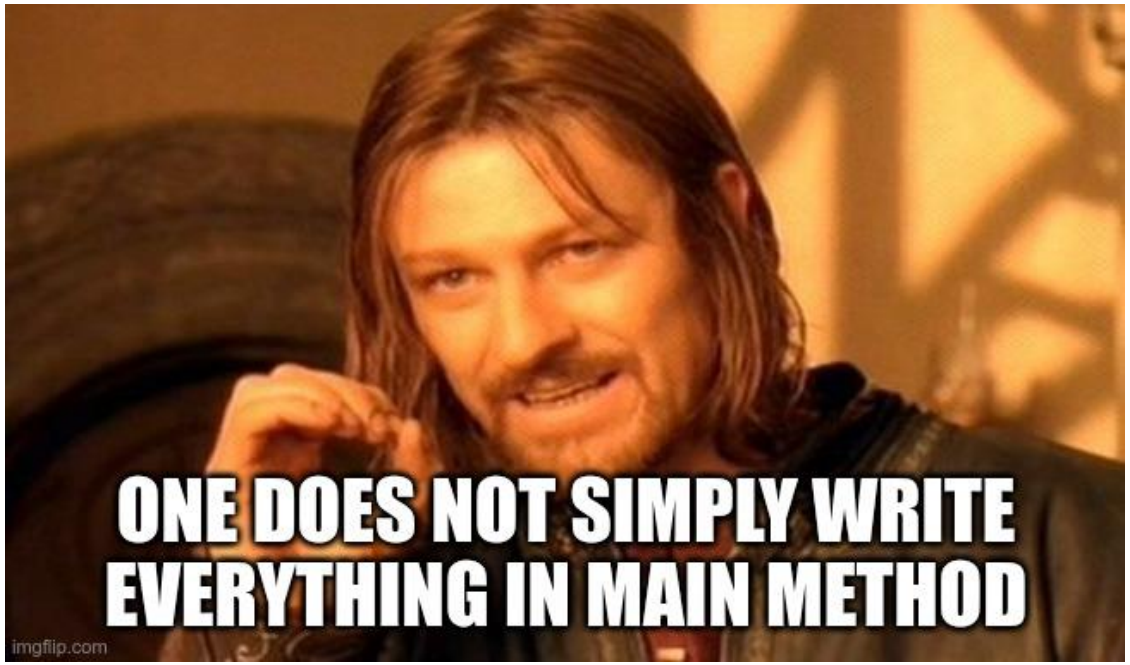
Function – Anonymous Function or Lambda Function

- You also can declare a function as a variable and it can be called any where in the same scope.

```
void main(){
  Function printHelloWorld = () => print("Hello World!");
  printHelloWorld();
}
```

Function – Best Practice

- Should be small
- Should do just one thing
- Should have fewer arguments
- Should not have side effects



Dart Effective

We encourage you to read more detail about best practices on this page.
https://dart.dev/guides/language/effective-dart

Function

- Checkpoint 6: Refactor code from checkpoint 5 to separate do-while, while, for loop into a function in which receive a list of score as an argument and return the total score as integer value. Then use the total score to display the output as same as checkpoint 5
- Example Output : Alexander Mohamad's total score is 73.

How to submit checkpoints

- The checkpoint file name should be in checkpoint<X>.dart format
- Create a folder with <studentid>-lecture-2 name then put all source code then zip a file.
- Upload into lecture 2 on the it onlearn platform.
- The due date of submission is Fri, Jan 20.
- Feel free to ask any questions via Discord channel.

Homework: Temperature Conversion

- Create a function called **convertToFahrenheit(double celsius)** that takes in a double value representing a temperature in Celsius and returns the equivalent temperature in Fahrenheit.
- The formula to convert Celsius to Fahrenheit is F = ((9/5) * C) + 32
- Create a new function called **printTemperatureTable()** that uses a for-in loop to iterate from 0 to 100 in increments of 10 and prints a table of temperatures in both Celsius and Fahrenheit.
- The table should have 2 columns, one for Celsius and the other for Fahrenheit, and should be labeled accordingly.

Homework: Temperature Conversion

- Call the **printTemperatureTable()** function in **main()** to test your implementation
- The table should look something like this:

```
Celsius        Fahrenheit
  0             32.0
 10             50.0
 20             68.0
...
100            212.0
```