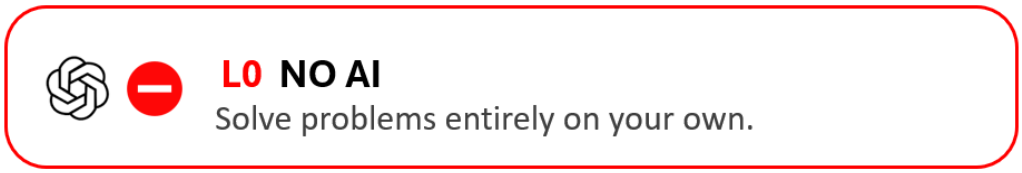
W1 PRACTICE

PART 1 – EXPLORATION

Learning objectives

- Apply type **inference** for variable declarations.
- Handle **nullable** and **non-nullable** variables.
- Differentiate between **final** and **const**.
- Manipulate **strings, lists, and maps.**
- Use **loops** and **conditions** to control flow.
- Define and call functions with positional and **named arguments**, understand **arrow syntax**



How to run Dart code?

You can write you code on VS Code, or using this online editor

- Install Dart and Flutter SDK
- Online Dart Compiler

Resources for this research

To help you complete this handout, use the following resources:

- ✓ Variables
- ✓ Null Safety
- ✓ Built-in types
- ✓ Lists
- ✓ Loops
- ✓ Conditions
- ✓ Functions

EX 1 - Type Inference

EXPLAIN : Explain how Dart infers the type of a variable.

In Dart, when you don't explicitly write the type of a variable, Dart uses **type inference** to figure it out for you.

CODE : Complete the bellow code to illustrate the concepts:

```
// Declare a int variable and let Dart infer its type
var number = 10;
// Define a variable with an explicit String type
String var = "Dart is cool!!!";
```

EX 2 - Nullable and Non-Nullable Variables

EXPLAIN : Explain nullable vs non-nullable variables.

- Non-Nullable cannot hold null and must have a value.
- Nullable(?) can be hold either a value or null.

EXPLAIN : When is it useful to have nullable variables?

Nullable variables are useful when sometimes a value might **not exist yet** or is **optional**.

CODE : Complete the bellow code to illustrate the concepts:

```
// Declare a nullable integer variable and assign it a null value
int? number = null;
// Declare a non-nullable integer variable and assign it a value
int number = 10;
// Assign a new value to the nullable variable
Number = 20;
```

EX 3 – Final and const

EXPLAIN : Describe the difference between final and const.

- final refer to the value is set once at runtime, and it can be assigned to a value isn't known until programs execute.
- Const refer to the value must be a compile-time constant.

CODE : Complete the bellow code to illustrate the concepts:

```
// Declare a final variable and assign it the current date and time
final currentDate = DateTime.now();
// Can you declare this variable as const? Why?
In this situation cannot be declare that variable const because const
requires the value to be compile-time constant but DateTime.now() give the
current system time that runtime.
```

```
// Declare a const variable with a integer value
Const number = 10;
// Can you reassign the value of this final variable? Why?
You cannot be reassigned the value of final variable because final variable
can be assigning only once time.
```

EX 4 - Strings, Lists and Maps

CODE : Complete the bellow code to illustrate the concepts:

Strings:

```
// Declare two strings: firstName and lastName and an integer:age
String firstName = “Keo”;
String lastName = “Hengneitong”
int age = 20;
// Concatenate the 2 strings and the age
final concatenate = “$firstName $lastName and $age”;
// Print result
print(concatenate);
```

Lists:

```
// Create a list of integers
List<int> number = [1, 2, 3, 4, 5];

// Add a number to the list
number.add(6);

// Remove a number from the list
number.remove(3);
// Insert a number at a specific index in the list
number.insert(1, 10);
// Iterate over the list and print each number
for(var number in numbers){
      print(number)
}
```

Maps:

```
// Create a map with String keys and integer values
Map<String, int> score = {
      “student1” : 100,
      “student2” : 90,
      “student3” : 80
};
// Add a new key-value pair to the map
Score[“student4”] = 88;
```

```
// Remove a key-value pair from the map
Score.remove(“student2”);

// Iterate over the map and print each key-value pair
Score.ForEach((key, value){
      Print(“$key: $value”);
});
```

EXPLAIN : When should I use a Map instead of a List?

You should use a **Map** instead of a **List** when you want to **associate values with unique keys** rather than just store an ordered collection.

EXPLAIN : When should I use a Set instead of a List?

You should use a **Set** instead of a **List** when you want a **collection of unique items** and don’t care about order.

EX 5 - Loops and Conditions

CODE : Complete the bellow code to illustrate the concepts:

```
// Use a for-loop to print numbers from 1 to 5
for(int i = 0; i <= 5; i++){
      print(i);
}
// Use a while-loop to print numbers while a condition is true
int j = 1;
while(j <= 5){
      print(j)
}
// Use an if-else statement to check if a number is even or odd
int number=5;
if(number%2 == 0){
      print(“$number is even”);
}else{
      print(“$number is odd!”);
}
```

EX 6 - Functions

EXPLAIN : Compare positional and named function arguments

Positional argument are passed in order and can be optional with [] and named argument are passed by name, order doesn’t matter, and can be optional or required.

EXPLAIN : Explain when and how to use arrow syntax for functions?

arrow function is a shorthand for function that return a single expression.

CODE : Complete the bellow code to illustrate the concepts:

Defining and Invoking a Function:

```
// Define a function that takes two integers and returns their sum
int add(int a, int b){
      return a + b;
}
// Call the function and print the result
Print(add(3,5));
```

Positional vs Named Arguments:

```
// Define a function that uses positional arguments
int getMultiply(int a, int b, [int? c = 1]){
      return a*b*c;
}
// Define another function that uses named arguments with the required
keyword (ex: getArea with rectangle arguments)
int getArea(required int width, required int height){
      Return width * height;
}

// Call both functions with appropriate arguments
int multiply = getMultiply(10, 2);
print(multiply);

int area = getArea(height: 5, width: 2);
print("Area: $area");
```

EXPLAIN : Can positional argument be omitted? Show an example

A positional argument **can only be omitted if declared optional** using []. Otherwise, it must always be supplied.

```
void greet(String name, [int? age]) {
  if (age != null) {
    print("Hello $name, you are $age years old!");
  } else {
    print("Hello $name!");
  }
}
```

EXPLAIN : Can named argument be omitted? Show an example

Named arguments **can be omitted if they are not marked required**. If marked required, they must always be provided.

```
void greet({String? name, int? age}) {
  print("Hello ${name ?? 'Guest'}, Age: ${age ?? 'unknown'}");
}
```

CODE : Complete the bellow code to illustrate the concepts:

Arrow Syntax:

```
// Define a function using arrow syntax that squares a number
int square(int x) => x * x;
// Call the arrow function and print the result
print(square(5));
```