q

W1 PRACTICE

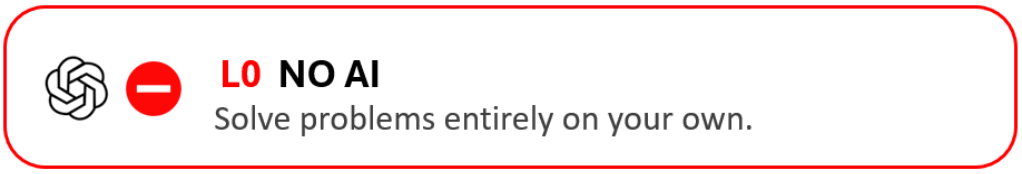
PART 2 – MANIPULATION

⚠ Before this practice

- You need to have completed: **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**



Are you lost?

Read the following documentation to be ready for this practice:

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

REVIEW PART-1

In group of 3 or 4, **review the** self-learning work.

- After discussing with your peer, **update your answers** if you need
- Some groups will present **their outcome** to the classroom

2. Nullable and Non-Nullable Variables

`EXPLAIN` : Explain nullable vs non-nullable variables.

`EXPLAIN` : When is it useful to have nullable variables?

`CODE` : Complete the bellow code to illustrate the concepts:

```
// Declare a nullable integer variable and assign it a null value

// Declare a non-nullable integer variable and assign it a value

// Assign a new value to the nullable variable
```

---

3. Final and const

`EXPLAIN` : Describe the difference between `final` and `const`.

`CODE` : Complete the bellow code to illustrate the concepts:

```
// Declare a final variable and assign it the current date and time

// Can you declare this variable as const? Why?

// Declare a const variable with a integer value

// Can you reassign the value of this final variable? Why?
```

Review, and update your answer regarding your work

EX 1 – Manipulate Types

Are you clear about strings, integer, list, map, set, objects in Dart?
Examine the given data structures and **write the inferred Dart type** for each one (*see example*)

Notes

- *First find by yourself the type*
- *If you need, double check your answer with VSCode.*

-

Data	Dart Type

const studentGrades = {
'Sokan': [90, 85, 88],
'Sokea': [70, 80, 75],
'Hay': [95, 92, 89],
};
	Map<String, List<int>> studentGrades = {
	'Sokan': [90, 85, 75],
	'Sokea': [70, 80, 75],
	'Hay': [95, 92, 89],
	};

const pizzaPrices = {
'margherita': 5.5,
'pepperoni': 7.5,
'vegetarian': 6.5,
};
	Map<String, double> pizzaPrices = {
	"margherita": 5.5,
	"pepperoni":7.5,
	"vegetarian":6.5,
	};

const books = [{'title': '1984', 'author': 'George Orwell'}, {'title': 'Brave New World', 'author': 'Aldous Huxley'}, {'title': 'Fahrenheit 451', 'author': 'Ray Bradbury'},];	List<Map<String, String>> books = [{'title': '1984', 'author': 'George Orwell'}, {'title': 'Brave New World', 'author': 'Aldous Huxley'}, {'title': 'Fahrenheit 451', 'author': 'Ray Bradbury'},];

const company = {
'HR': {'employees': 5, 'budget': 20000},
'IT': {'employees': 10, 'budget': 50000},
'Marketing': {'employees': 7, 'budget': 30000},
};
	Map<String, Map< String, int>> company = {
	'HR': {'employees': 5, 'budget': 20000},
	'IT': {'employees': 10, 'budget': 50000},
	'Marketing': {'employees': 7, 'budget': 30000},
	};

const coordinates = [[1, 2], [3, 4], [5, 6],];	List<List<int>> coordinates = [[1, 2], [3, 4], [5, 6],];
const productDetails = { 'id': 101, 'name': 'Laptop', 'price': 999.99,	Map<String, Object> productDetails = { 'id': 101, 'name': 'Laptop',

'inStock': true, };	'price': 999.99, 'inStock': true, };
const operations = [(int a, int b) => a + b, (int a, int b) => a - b, (int a, int b) => a * b,];	List<int Function(int, int)> operations = [(int a, int b) => a + b, (int a, int b) => a - b, (int a, int b) => a * b,];
const distances = {3.1, 5.5, 10.2, 7.8};	Set<double> distance = {3.1, 5.5, 10.2, 7.8};

const university = {
'departments': [
{
'name': 'Computer Science',
'students': [
{'name': 'Alice', 'age': 22},
{'name': 'Bob', 'age': 24},
]
},
{
'name': 'Mathematics',
'students': [
{'name': 'Charlie', 'age': 21},
{'name': 'Dave', 'age': 23},
]
}
]
};
	Map<String, List<Map<String, Object>>>
	university = {
	'departments': [
	{
	'name': 'Computer Science',
	'students': [
	{'name': 'Alice', 'age': 20},
	{'name': 'Bob', 'age': 24},
	]
	},
	{
	'name': 'Mathematics',
	'students': [
	{'name': 'Charlie', 'age': 21},
	{'name': 'Dave', 'age': 23},
	]
	}
	]
	};

EX 2 – Manipulate final and const

In this exercise, you need to decide which variable can be declared as **const** or **final**.

```dart
// 1 - startText
String iLike = 'I like pizza';

// 2 - toppings
String toppings = 'with tomatoes';
toppings += " and cheese";

// 3 - message
String message = '$iLike $toppings';
print(message);

// 4 - rbgColors
List<String> rbgColors = ['red', 'blue', 'green'];
print(rbgColors);

// 5 - weekDays
List<String> weekDays = ['monday', 'tuesday', 'wednesday'];
weekDays.add('thursday');
print(weekDays);

// 6 - scores
List<int> scores = [45,35,50];
scores = [45,35,50, 78];
print(scores);
```

Guess which variables can be declared as **const**, **final** or **var**, and explain your choices.

Notes

- *Read here to understand the concepts.*
- *Prefer const over final over var.*

	VAR, FINAL, CONST?	**WHY**
iLike	CONST	*Because this variable never changes*
toppings	VAR	*Because this variable can change but in still keep as the String type*
message	FINAL	*Because this variable can ADD MORE STRING now change the string*
rbgColors	CONST	*Because this variable can't changes the element colors*
weekDays	FINAL	*Because this variable is initial afterword (at compile time)*
score	FINAL	*Because this can't change*

EX 3 – Filter a list

Instructions

- You are given a list of integers representing the scores of students in an exam.
- A score of 50 or higher is considered passing.
- Write a Dart program that filters and returns a list of students and the number of students who passed the exam.

Constraints

- You must use the **where** function with a proper **anonymous function** to filter the original list
- More information here

Examples

INPUT

[45, 78, 62, 49, 85, 33, 90, 50]

OUTPUT

[78, 62, 85, 90, 50]

5 students passed

Code:

```
void main(){
    List<int> scores = [45, 78, 62, 49, 85, 33, 90, 50];
    print(scores.where((scorePass) => scorePass >= 50).toList());
}
```

EX 4 – Manipulate maps

Given the following **map** of pizza prices:

```
const pizzaPrices = {
  'margherita': 5.5,
  'pepperoni': 7.5,
  'vegetarian': 6.5,
};
```

Write a program to calculate the total for a given order.

```
print(pizzaPrices.values.toList().reduce((a, b) => a + b));  // it can error when the map of pizza is nothing
print(pizzaPrices.values.toList().fold<double>(0.0, (a, b) => a + b)); // it set to 0.0 when the map of pizz is nothing
```

For example, given the following order:

```
const orders = ['margherita', 'pepperoni'];
```

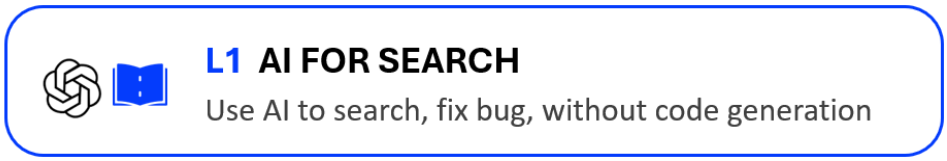
```
void main(){
    const pizzaPrices = {
        'margherita': 5.5,
        'pepperoni': 7.5,
        'vegetarian': 6.5,
    };
    const orders = ["margherita", "pepperoni", "1"]; // ordered


    final checkPizza = orders.where((pizzaOrdered) => !pizzaPrices.containsKey(pizzaOrdered)).toList(); // store pizza that not in the menu as a list
     if(checkPizza.isNotEmpty){
        print('$checkPizza pizza is not on the menu'); // print the pizza that not in the menu
     }else{
        final total = orders.map((pizzaOrdered) => pizzaPrices[pizzaOrdered] ?? 0.0).fold<double>( 0.0, (sumTotalItems, prices) => sumTotalItems + prices); //create a new list as map from orders and check it in the pizzaPrices if it null using 0.0 if not using .fold method to calculate the total value and store in the total.

        print('Total: \$${total.toStringAsFixed(0)}');
    }
}
```

The program should print:

```
Total: $13
```

If a pizza is not on the menu, the program should print:

```
pineapple pizza is not on the menu
```



HOMEWORK 1 – Write a robot simulator

A robot factory's test facility needs a program to verify robot movements.
The robots have three possible movements:

- turn right
- turn left
- advance

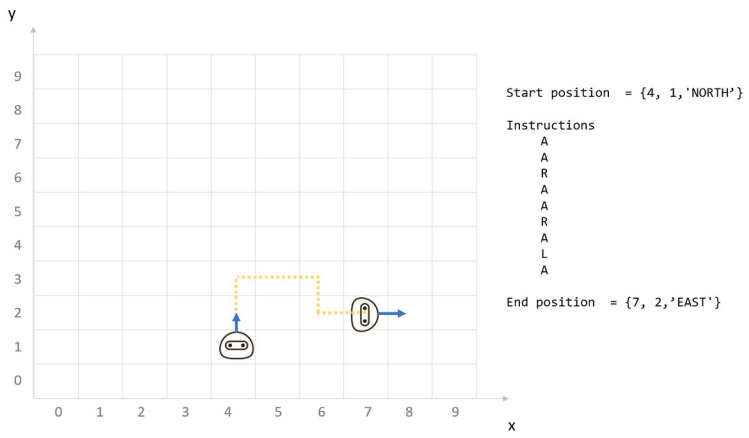
Robots are placed on a hypothetical infinite grid, facing a particular direction (NORTH, EAST, SOUTH, OR WEST) at a set of {X, Y} coordinates, e.g., {3,8}, with coordinates increasing to the north and east.

The robot then receives a number of instructions, at which point the testing facility verifies the robot's new position, and in which direction it is pointing.

As example

- the string "RAALAL" means:
    1. Turn right
    2. Advance twice
    3. Turn left
    4. Advance once
    5. Turn left yet again

- Say a robot starts at {7, 3} facing north.
- Then running this stream of instructions should leave it at {9, 4} facing west.



Note

- You are free to decide how to structure your data in Dart language
- Try to use as much as possible functions to separate your logic

Code:

```
enum Direction { north, east, south, west } // to access each element use .name, .index, .values
void main() {
  // Initial position {7, 3} and facing north
  int x = 7;
  int y = 3;
  Direction direction = Direction.north;
  // Example instruction sequence
  const instructions = "RAALAL";

  // Process the instructions and get the final position and direction
  for (int i = 0; i < instructions.length; i++ ){
    String movement = instructions[i];

  if(movement == "R"){
    direction = Direction.values[(direction.index +1) % 4];
  }else if(movement == "L"){
    direction = Direction.values[(direction.index +3) % 4];
  }else if (movement == "A") {
    if(direction == Direction.north) y++;
    else if( direction == Direction.east) x++;
    else if (direction == Direction.west) x--;
    else if ( direction == Direction.south) y--;
    // the below comment is in advance way generate by AI review by me (still not clear)
    // final movements = {
    // Direction.north: (0, 1),
    // Direction.east: (1, 0),
    // Direction.south: (0, -1),
    // Direction.west: (-1, 0),
    // };
     // final (dx, dy) = movements[direction]!;
```

```
      // x += dx;
      // y += dy;
    }else print("Invalid Instruction");
  }

  // Print the final position and direction
  print("Final position: $x, $y");
  print("Facing: $direction.name");
}
```

HOMEWORK 2 – Matching Brackets

Instructions

Given a string containing brackets [], braces {}, parentheses (), or any combination thereof, verify that any and all pairs are matched and nested correctly. Any other characters should be ignored.

Examples

INPUT	OUTPUT
{what is (42)}?	Balanced
[text}	Not balanced
{[[(a)b]c]d}	Balanced

Code:

```
String eleminateMatchSymbols(String symbols){ // using recursive
  final eleminated = symbols.replaceAll("()", "").replaceAll("[]",
"").replaceAll("{}", "");
  return (eleminated == symbols) ? symbols :
eleminateMatchSymbols(eleminated); // if eleminated == symbols calling the
function again and again until it not equal
}
void main (){
  // const input = "{what is (42)}?";
  const input = "[text}";
  // const input = "{[[(ab)]c]d}";
  const validSymbols = [
    "{", "(", "[", "}", ")", "]"
  ];

  // var stackOpens = input.split("").where((comparedSymbol) =>
validSymbols.contains(comparedSymbol)).join(); // for simple way but still
need to call the function recursive
  final checkSymboles =
eleminateMatchSymbols(input.split("").where((comparedSymbol) =>
```

```
validSymbols.contains(comparedSymbol)).join()); // in advance way
  print(checkSymboles.isEmpty ? "balanced" : "not Balanced");
}
```