Lambda Expressions in Dart

Functions as Variables

- Key Question: Can Functions Be Variables?
  - Can we treat functions as if they were variables?
  - Can we use functions as arguments or return values?

*In **modern programming languages** like Dart, JavaScript, and Python:*

- **YES!** Functions can be treated as variables
- You can store, pass, and manipulate functions just like any other data

```
// Function AS a variable!
var add = (int x, int y) => x + y;
// Regular variable
var result = 10;
```

In **older languages** like traditional C or early Java:

- **NO!** Functions and variables were completely separate concepts

```
// Function definition
int add(int x, int y) {
    return x + y;
}
// Variable definition
int result = 10;
```

What is a Lambda Expression?

- A **lambda expression** is a function written as a **value** that can be:
  - Stored in variables
  - Passed as arguments
  - Used immediately without naming

- **Basic Syntax in Dart:**
  - => is used for single-line code to return a value.
  - {...} is used to execute multiple lines of code.

```
(parameters) => expression
// OR
(parameters) { statements; }
```

Without Lambda (Verbose):

```
int compareByLength(String a, String b) {
  return a.length.compareTo(b.length);
}

List<String> names = ["Bob", "Alice", "Charlie"];
names.sort(compareByLength);
```

With Lambda (Concise):

```
List<String> names = ["Bob", "Alice", "Charlie"];
names.sort((a, b) => a.length.compareTo(b.length));
```

- We do not have to define a function before using it.

Simple Examples in Three Steps

- **Step 1: Regular function**

```
int square(int x) {
  return x * x;
}
```

- **Step 2: Same function as a lambda expression**
  - We have one line of code that returns a value, so we use the => operator.

```
var square = (int x) { return x * x; }
var square = (int x) => x * x;
```

```
int add(int x, int y) => x + y;
Function add = (int x, int y) => x + y;
```

- The Lambda expression is a value of a Function type.
- These are the same function definitions.

- **Step 3: Using the lambda**

```
print(square(5)); // Output: 25
print(square); // Print function reference
```

We can call lambda functions directly.

```
var lambdaSquare = (int x) => x * x;
print('lambdaSquare(5): ${lambdaSquare(5)}');
```

Basic Lambda Expressions

Functions as Variables

```
var getCurrentDateTime = () => DateTime.now();
var isdouble = (int x) => x * 2;
var makeGreeting = (String name) => "Welcome, \$name!";
```

- Variables (references) that can store a lambda expression
- We use the `()` operator to call the function from the variable.

```
var calculateArea = (double width, double height)
=> width * height;

var getFullName = (Map<String, String> person) =>
      '\${person["first"]} \${person["last"]}';
var person = {"first": "John", "last": "Doe"};
print('getFullName(person): ${getFullName(person)}');
```

- There can be any number of arguments.
- The argument can be an object.

```
// Store functions in a list!
var operations = [add, subtract, multiply];
print('operations[0](8, 3): ${operations[0](8, 3)}');

var calculator = {
  'add': (int a, int b) => a + b,
  'subtract': (int a, int b) => a - b,
};

print('calculator["add"](7, 2):
      ${calculator["add"]!(7, 2)}');
```

- Lambda expressions can be stored in an array or a map.

Conditional Logic

```
var getGrade = (int score) => score >= 90
    ? 'A' : score >= 80  ? 'B' : score >= 70
    ? 'C' : score >= 60  ? 'D' : 'F';
print('getGrade(95): ${getGrade(95)}');
```

- We can make a one-liner with conditional logic in a lambda expression.

Using Functions as Arguments or Return Values

Functions as an Argument

```
int add(int x, int y) => x + y;
var result3 = operatorSelector(add, 10, 20);
print('Result of operatorSelector 3: $result3');
```

- We can use the name of the function (Function type value) as an argument.

```
int operatorSelector(Function func, int a, nt b) {
  return func(a, b);
}
var result = operatorSelector((x, y) => x + y, 10, 20);
print('Result of operatorSelector: $result');
var result2 = operatorSelector((x, y) => x * y, 10, 20);
print('Result of operatorSelector: $result2');
```

- Using a lambda expression is simple and easy to read.

Functions as return values

```
// Function that returns another function as an argument
Function returnFunction(String func) {
  // Return a function based on the input string
  if (func == 'add') {
    return (a, b) => a + b;
  } else if (func == 'multiply') {
    return (a, b) => a * b;
  } else {
    throw Exception('Unknown function type');
  }
}
var addFunction = returnFunction('add');
print('Result of addFunction: ${addFunctio(10, 20)}');
var multiplyFunction = returnFunctio('multiply');
print('Result of multiplyFunction: {multiplyFunction(10, 20)}');
```

Flutter Example: setState() and Event handlers

- We use a lambda expression as an argument to the setState() function.

```
// This is better
setState(() {this.counter++;});
```

Event Handlers

- **Button clicks:**

```
ElevatedButton(
  onPressed: () {                  // Lambda expression here!
    print("Button clicked!");
  },
  child: Text("Click me"),
)
```

- **Bottom Navigation:**

```
BottomNavigationBar(
  onTap: (index) {      // Lambda with parameter
    setState(() {       // Another Lambda expression
      this.currentIndex = index;
    });
  },
  // ...
)
```