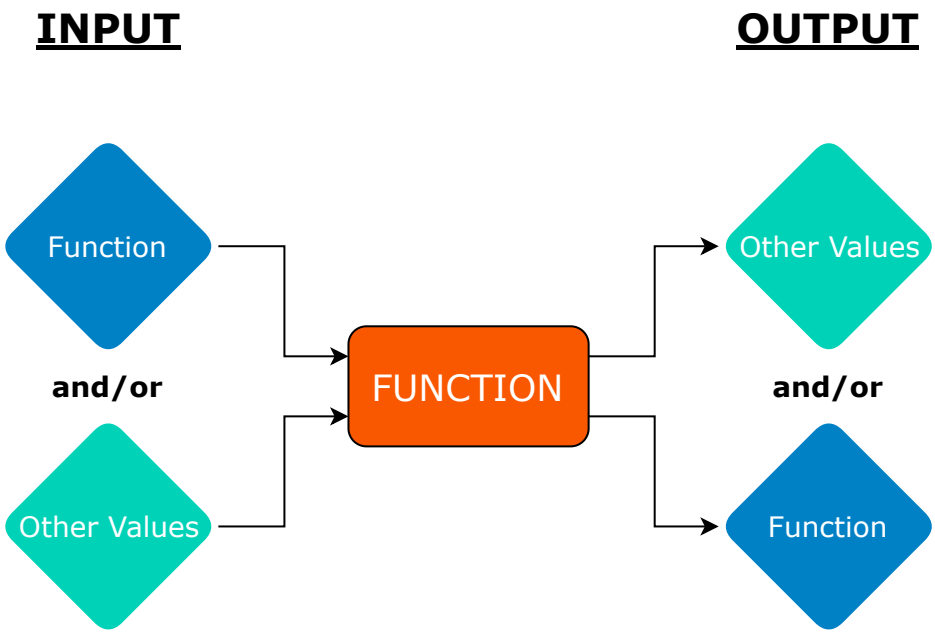
Higher-Order Functions

In this lesson, you will be introduced to higher-order functions and learn their syntax.

We'll cover the following

Introduction

Dart is a true object-oriented language, so even functions are objects and have a type `Function`. Functions are treated like *first-class* values. What this means is that like any other value, a function can be assigned to variables, passed as a parameter to another function, and can also be returned as a result.



Functions that take other functions as parameters or that return functions as results are called **higher-order functions**.

As `Function` is another type just like `num` or `List`, we can create functions which have parameters of type `Function`. This means when calling that created function,

you must pass it another function as an argument.

#

Learning by Example

Let's create a function called `forAll` which takes a list and another function, `f`, as arguments. `forAll` performs `f`'s functionality on every item in the provided list. `forAll` returns a new list with the modified elements of the provided list.

```
List<int> forAll(Function f, List<int> intList){
  var newList = List<int>();
  for(var i = 0; i < intList.length; i ++){
    newList.add(f(intList[i]));
  }
  return newList;
}
```

Since `forAll` returns a list, we have specified its return type as `List<int>`. `newList` is the list to be returned at the end of the function. The value of every item in `intList` is modified according to the functionality of `f` and the modified value is stored in `newList` (**line 4**).

We're going to call `forAll` in the code snippet below. The function we will be passing as an argument will be the `factorial` function created in the previous lesson.

```
List<int> forAll(Function f, List<int> intList){
  var newList = List<int>();
  for(var i = 0; i < intList.length; i ++){
    newList.add(f(intList[i]));
  }
  return newList;
}

// Recursive factorial function
int factorial(int x) {
  if (x == 1) {
    return 1;
  } else {
    return x*factorial(x-1);
  }
}

main() {
  var tester = [1,2,3];
  var result = forAll(factorial, tester);
  print(tester);
  print(result);
}
```

The first list of integers displayed as output is the original list provided to `forAll` (**line 19**). The second list of integers is the list of modified elements returned by `forAll` (**line 20**).

> **Try it Yourself:** Try creating your own function which you can pass to `forAll`. Just remember that the function must have a single parameter of type `int` and also return an integer.

The `forEach` Method

Dart has a built-in method, `forEach`, which has similar functionality to our `forAll` function. `forEach` has a single parameter of type `Function` and is called on a collection type. The functionality of the function passed to `forEach` is applied to every item that `forEach` is called on. However, the function you pass should not have a return value, i.e., it must be `void`.

In the code snippet below, we are passing the built-in `print` method to `forEach` and calling it on a list of integers.

```
main() {
  var tester = [1,2,3];
  tester.forEach(print);
}
```

If we pass our `factorial` function in place of `print` in the code snippet above, we would get an error.

Take Away

By treating functions as first-class values, we can write much shorter code and allow easy scalability.

Imagine having to pass every item in a list to the factorial function individually. When the values in the list change, you would need to extensively modify your code. However, with our `forAll` function we just need to type one line of code in

code. However, with our `toPAll` function we just need to type one line of code in which the only thing that needs to be modified is the name of the list.

Try writing your own higher-order function in the next lesson.