Scope

In this lesson, you will learn about lexical scoping.

We'll cover the following

Lexical Scope

Lexical scope is the range of functionality of a variable so that it may only be called from within the block of code in which it is defined.

Dart is a lexically scoped language, which means that the scope of variables is determined statically, simply by the layout of the code.

Each set of curly brackets (`{}`) acquires its own new scope while inheriting from the scope in which it was declared. With lexical scoping, descendant scopes will access the most recently declared variable of the same name.

Remember in the previous lesson when `nestedFunction` was outside the scope of `main()`? This was because it was declared in the scope of `outerFunction`.

> You can follow the curly brackets outwards to see if a variable is in scope.

Visibility

When it comes to what is and isn't visible in a block of code, there are two rules.

Rule #1: When you define something inside a block of code, it is only visible from within that block of code.

Rule #2: The definitions inside a block of code, shadow definitions of the same names outside the block of code.

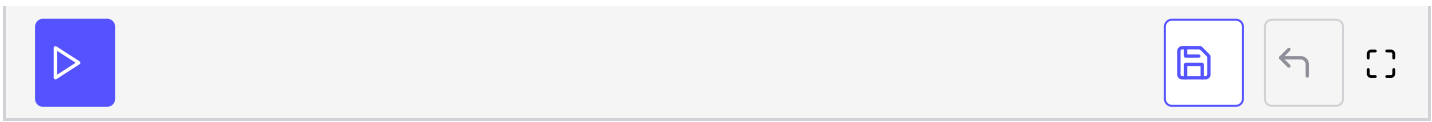
For instance, if I define a variable outside of a function, I can define a variable with the same name inside the function as well, without it affecting the outside variable. The outer variable will be shadowed by the inner. Let's look at an example below.

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

main() {
  var amIVisible = 0;

  void result() {
    var amIVisible = square(3);
    print("Variable Inside Block: $amIVisible");
  }

  result();
  print("Variable Outside Block: $amIVisible");
}
```



While we are printing a variable with the same name, i.e., `amIVisible`, the variables are actually different, hence, we get two different values in the output.

In the next lesson, you will apply the concepts you learned in this lesson and write your own nested function.