Evaluation Strategies

In this lesson, we will be going over the evaluation strategy Scala uses to evaluate expressions.

We'll cover the following

- The Substitution Model
- Call-by-value (CBV)
- Call-by-Name (CBN)
- Conclusion

The Substitution Model

Scala uses the **substitution model** for evaluating expressions. The idea underlying this model is that evaluation simply reduces expressions to a value.

Let's call the `squareSum` function we created in the first lesson and see how the *substitution model* would evaluate it. To make things interesting we will pass an expression which reduces to a `Double` as one of the parameters.

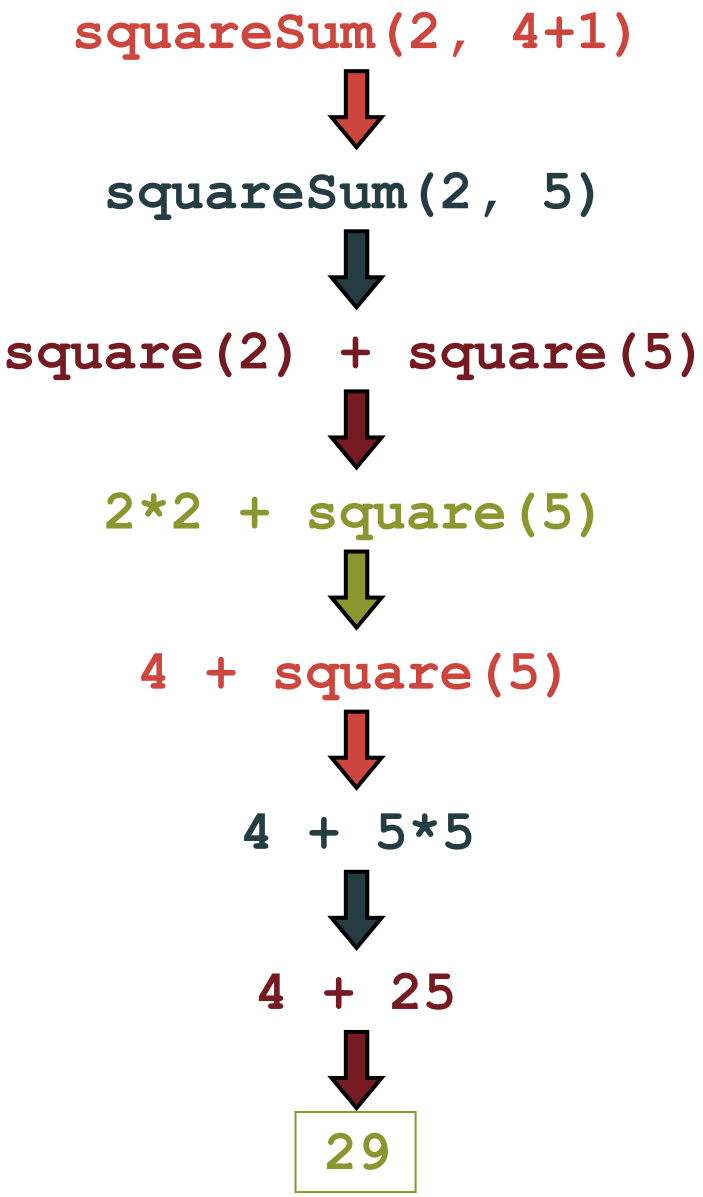
This code requires the following environment variables to execute:

LANG	C.UTF-8

```
def square(x: Double) ={
  x * x
}

def squareSum(x: Double, y: Double) ={
  square(x) + square(y)
}

val total = squareSum(2,4+1) // 4+1 reduces to 5 so this is a valid parameter

// Driver Code
println(total)
```

When you run the code above, you should get **29.0**, but how did this value come to be?

be?



From the flow diagram above, we can see that each expression is reduced until it becomes a value and cannot be further evaluated resulting in the final output of **29**.

The *substitution model* is simply evaluating an expression by reducing it to a value, but how do we know the order in which the expressions are to be evaluated. For this, the *substitution model* is divided into two evaluation strategies: **call-by-value** and **call-by-name**.

Call-by-value (CBV)

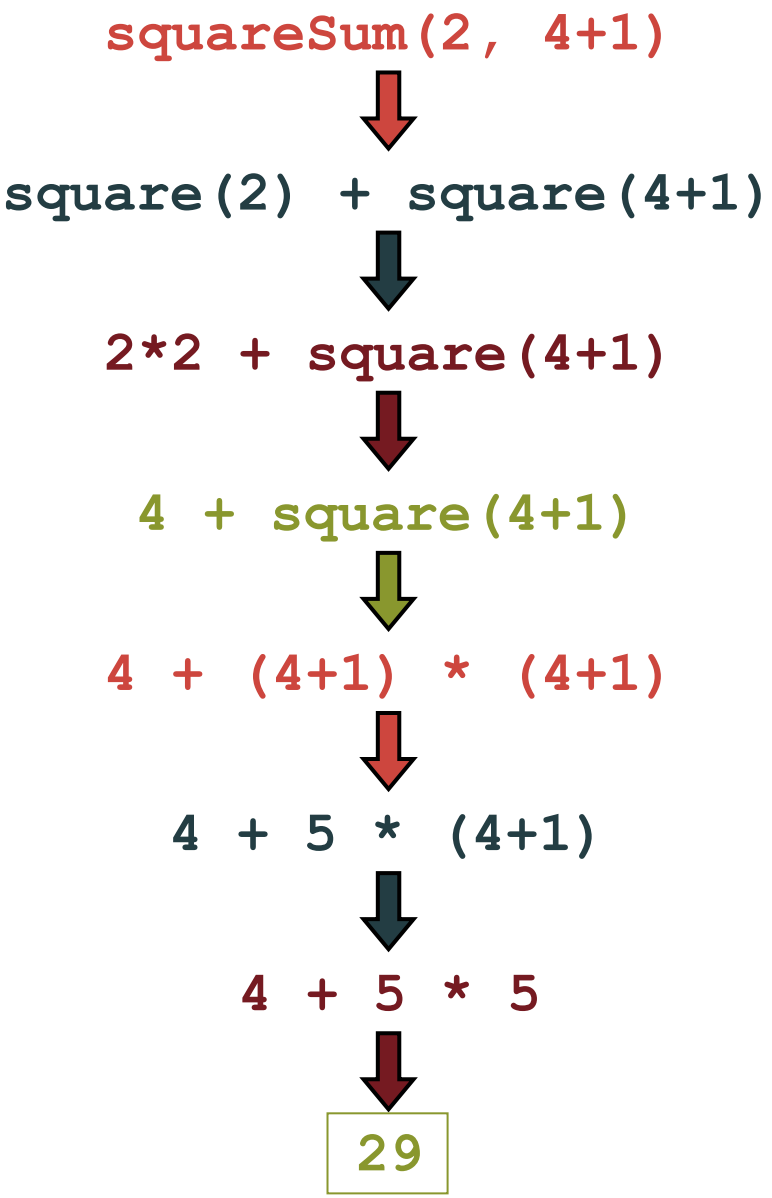
Call-by-value copies the actual arguments of a function to its formal parameters. In our example above, `4+1` is the actual argument which is being copied to the formal parameter `y`. As CBV copies all the arguments, it needs to evaluate them until they reduce to a value which is then used by the function. The flow diagram above is

showing how CBV evaluates the function call `squareSum(2, 4+1)`.

The advantage of CBV is that it evaluates every function argument only once.

Call-by-Name (CBN)

Call-by-name uses the actual argument as is in the function. Let's see how CBN would evaluate `squareSum(2, 4+1)`.



The advantage of CBN is that a function argument is not evaluated if the corresponding parameter is unused in the evaluation of the function body.

Conclusion

Simply put, CBV will evaluate every expression to its final value before calling the function, regardless of if the function body needs it or not. CBN, on the other hand, will only take the expressions required by the body of the function and pass them to the function just as you passed them. It then reduces the expressions to their

to the function just as you passed them. It then reduces the expressions to their final value in the function body.

For instance, we could have a function which takes two parameters, but only uses the first one in its function body.

```
def func(int x, int y) = {
  print(x)
}
```

You then call that function passing it two expressions.

```
func(1+1, 1+2)
```

CBV will start evaluating by:

```
func(2,3)
```

CBN will start evaluating by:

```
print(1+1)
```

In the next lesson, let's see if we can figure out which evaluation strategy is the faster option.