Functional programming is a programming philosophy based onlambda calculus. Lambda calculus was created by Alonzo Church, the PhD adviser to Alan Turing who is known for his role in cracking the encryption of the Nazi's Enigma machine during World War Two. Functional programming has been a popular approach ever since it helped bring down the Third Reich.

Functional programming concentrates on four constructs:

1. Data (numbers, strings, etc)
2. Variables (function arguments)
3. Functions
4. Function Applications (evaluating functions given arguments and/or data)

By now you're used to treating variables inside of functions as data, whether they're values like numbers and strings, or they're data structures like lists and vectors. With functional programming you can also consider the possibility that you can provide a function as an argument to another function, and a function can return another function as its result.

If you've used functions like sapply() or args() then it's easy to imagine how functions as arguments to other functions can be used. In the case of sapply() the provided function is applied to data, and in the case of args() information about the function is returned. What's rarer to see is a function that returns a function when it's evaluated. Let's look at a small example of how this can work:

```
adder_maker <- function(n){
  function(x){
    n + x
  }
}

add2 <- adder_maker(2)
add3 <- adder_maker(3)

add2(5)
[1] 7
add3(5)
[1] 8
```

In the example above the function adder_maker() returns a function with no name. The function returned adds n to its only argument x.

✓ Complete   Go to next item