What is the difference between built-in control structures and curried function control abstractions?

The curried function requires a function argument of the appropriate type.

Describe currying and its use.

A curried function takes multiple argument lists. It is used for creating control abstractions that feel like native language support.

What is the simplest way to use ScalaTest?

Extend `arg.scalatest.Suite`.

Start method names with "test" and use assertions or exceptions in them.

Invoke `execute()` on it or use the provided Runner application.

Give another strategy for giving custom control abstraction a native feel.

Use the by-name parameter instead of function as final argument list.

```
def byNameAssert(predicate: => Boolean)=
  if (assertionsEnabled && !predicate)
    throw new AssertionError
```

If assertions are off, the predicate will never be evaluated and its side effects will never be seen.

Describe higher-order functions and their uses.

They are functions that take functions as parameters.

They enable the creation of control structures to eliminate duplication of code and simplify client code.

Give custom control abstractions a native feel.

Method calls with one arg can use curlies instead of parens, then you can curry away other args, leaving function as the final parameter.

```
withPrintWriter(file){
  writer => writer.println(new Date)
}
```

Get a reference to a curried function's "second" function.

And its "first" function.

```
def curriedSum(x:Int)(y:Int) = x+y
val onePlus = curriedSum(1)_
onePlus(2) = 3
```

No space is needed before _ because (is not a legal identifier.

```
val onePlus = curriedSum(_:Int)(1)
```

How do you pass method names as values in Scala?

It is not allowed, as it is in dynamic languages.

The same effect is achieved with function values.

Give an example of a loan pattern.

```
def withPrintWriter(file:File)(op:PrintWriter => Unit) {
  val writer = new PrintWriter(file)
  try {
    op(writer)
  } finally {
    writer.close()
    }
}
```