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

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

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.

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.

They are functions that take functions as parameters.

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

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)
}
```

```
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)
```

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

The same effect is achieved with function values.

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