Chapter 29: Procedure Calls

Parameter	Info
IdentifierName	The name of the procedure to call.
arguments	A comma-separated list of arguments to be passed to the procedure.

Section 29.1: This is confusing. Why not just always use parentheses?

Parentheses are used to enclose the arguments of *function calls*. Using them for *procedure calls* can cause unexpected problems.

Because they can introduce bugs, both at run-time by passing a possibly unintended value to the procedure, and at compile-time by simply being invalid syntax.

Run-time

Redundant parentheses can introduce bugs. Given a procedure that takes an object reference as a parameter...

```
Sub DoSomething(ByRef target As Range)
End Sub
```

...and called with parentheses:

```
DoSomething (Application.ActiveCell) 'raises an error at runtime
```

This will raise an "Object Required" runtime error #424. Other errors are possible in other circumstances: here the `Application.ActiveCell` `Range` object reference is being *evaluated* and passed by value **regardless** of the procedure's signature specifying that `target` would be passed `ByRef`. The actual value passed `ByVal` to `DoSomething` in the above snippet, is `Application.ActiveCell.Value`.

Parentheses force VBA to evaluate the value of the bracketed expression, and pass the result `ByVal` to the called procedure. When the type of the evaluated result mismatches the procedure's expected type and cannot be implicitly converted, a runtime error is raised.

Compile-time

This code will fail to compile:

```
MsgBox ("Invalid Code!", vbCritical)
```

Because the expression `("Invalid Code!", vbCritical)` cannot be *evaluated* to a value.

This would compile and work:

```
MsgBox ("Invalid Code!"), (vbCritical)
```

But would definitely look silly. Avoid redundant parentheses.

Section 29.2: Implicit Call Syntax

```
ProcedureName
```

```
ProcedureName argument1, argument2
```

Call a procedure by its name without any parentheses.

Edge case

The `Call` keyword is only required in one edge case:

```
Call DoSomething : DoSomethingElse
```

`DoSomething` and `DoSomethingElse` are procedures being called. If the `Call` keyword was removed, then `DoSomething` would be parsed as a *line label* rather than a procedure call, which would break the code:

```
DoSomething: DoSomethingElse 'only DoSomethingElse will run
```

Section 29.3: Optional Arguments

Some procedures have optional arguments. Optional arguments always come after required arguments, but the procedure can be called without them.

For example, if the function, `ProcedureName` were to have two required arguments (`argument1`, `argument2`), and one optional argument, `optArgument3`, it could be called at least four ways:

```
' Without optional argument
result = ProcedureName("A", "B")

' With optional argument
result = ProcedureName("A", "B", "C")

' Using named arguments (allows a different order)
result = ProcedureName(optArgument3:="C", argument1:="A", argument2:="B")

' Mixing named and unnamed arguments
result = ProcedureName("A", "B", optArgument3:="C")
```

The structure of the function header being called here would look something like this:

```
Function ProcedureName(argument1 As String, argument2 As String, Optional optArgument3 As String)
As String
```

The `Optional` keyword indicates that this argument can be omitted. As mentioned before - any optional arguments introduced in the header **must** appear at the end, after any required arguments.

You can also provide a *default* value for the argument in the case that a value isn't passed to the function:

```
Function ProcedureName(argument1 As String, argument2 As String, Optional optArgument3 As String =
"C") As String
```

In this function, if the argument for `c` isn't supplied it's value will default to `"C"`. If a value *is* supplied then this will override the default value.

Section 29.4: Explicit Call Syntax

```
Call ProcedureName
```

```
Call ProcedureName(argument1, argument2)
```

The explicit call syntax requires the `Call` keyword and parentheses around the argument list; parentheses are redundant if there are no parameters. This syntax was made obsolete when the more modern implicit call syntax was added to VB.

Section 29.5: Return Values

To retrieve the result of a procedure call (e.g. `Function` or `Property Get` procedures), put the call on the right-hand side of an assignment:

```
result = ProcedureName
result = ProcedureName(argument1, argument2)
```

Parentheses must be present if there are parameters. If the procedure has no parameters, the parentheses are redundant.