Ruby For Programmers

This cheat-sheet accompanies the Bitwise Courses course on *Ruby For Programmers* by Huw Collingbourne.

Blocks

A Ruby block may be regarded as a sort of nameless function or method and its most frequent use is to provides a means of iterating over items from a list or range of values. Blocks may either be delimited by curly brackets `{` and `}` or by the keywords `do` and `end`. Variables declared at the start of a block between upright bars such as `|i|` can be treated like the arguments to a named method and are called 'block parameters'.

Blocks are often used as iterators with items from a collection or range passed into the block and assigned to the block parameters. This block iterates over the items in an array:

```
[1,2,3].each { |i|
   puts(i)
}
```

Output:

```
1
2
3
```

This block iterates over a range of value from 1 to 3:

```
(1..3).each do |i|
   puts(i)
end
```

Output:

```
1
2
3
```

A block may have multiple block parameters:

```
[[1,2,3],[3,4,5],[6,7,8]].each{
  |a,b,c|
    puts( "#{a}, #{b}, #{c}" )
}
```

Output:

```
1, 2, 3
3, 4, 5
6, 7, 8
```

Files

Ruby provides classes dedicated to handling IO – Input and Output. Chief among these is a class called, unsurprisingly, `IO`. The `IO` class lets you open and close IO 'streams' (sequences of bytes) and read and write data to and from them.

```
IO.foreach("testfile.txt") {|line| print( line ) }
```

The code above opens a text file named *"testfile.txt"* and the `foreach` method of the `IO` class passes one line at a time into the block delimited by curly brackets. Each line is assigned, one by one to the block parameter `line` and it is then displayed on screen by the `print` method.

Opening and Closing Files

While some standard methods open and close files automatically, often, when processing the contents of a file, you will need to open and close the file explicitly. You can open a file using either the `new` or the `open` method. You must pass two arguments to one of those methods – the file name and the file 'mode' – and it returns a new `File` object. The `File` class is a subclass of `IO`.

The File modes may be either integers which are defined by operating-system-specific constants or strings. The mode generally indicates whether the file is be opened for reading ("r"), writing ("w") or reading and writing ("rw"). This is the list of available string modes:

Mode	Meaning
`"r"`	Read-only, starts at beginning of file (default mode).
`"r+"`	Read-write, starts at beginning of file.
`"w"`	Write-only, truncates existing file to zero length or creates a new file for writing.
`"w+"`	Read-write, truncates existing file to zero length or creates a new file for reading and writing.
`"a"`	Write-only, starts at end of file if file exists, otherwise creates a new file for writing.
`"a+"`	Read-write, starts at end of file if file exists, otherwise creates a new file for reading and writing.
`"b"`	(DOS/Windows only) Binary file mode (may appear with any of the key letters listed above).

This is how open a file, *"myfile.txt"*, for writing ("w"). When a file is opened for writing, a new file will be created if it doesn't already exist. I use `puts()` to write six strings to the file, one string on each of six lines. Finally I close the file.

```
f = File.new("myfile.txt", "w")
f.puts( "I", "wandered", "lonely", "as", "a", "cloud" )
f.close
```

You can test for the existence of a file or directory using the `exist?` method (including the `?` at the end) of the `File` class, like this:

```
if File.exist?( "C:\\" ) then
   puts( "Yup, you have a C:\\ directory" )
else
   puts( "Eeek! Can't find the C:\\ drive!" )
end
```