HEALTH
UNIVERSITY OF UTAH

R Syntax & Data Structures

Andrew Redd, PhD.

R Bootcamp 2021

~~This will be short, I promise.~~

Year’s past feedback said this needed to be much longer.

The absolute most important part of syntax … Comments

Comments

Comments are started by the # character and continue to the end of the line.

```
# Comments use the '#' character

1 + 1 # But they don't have to be the whole line
```

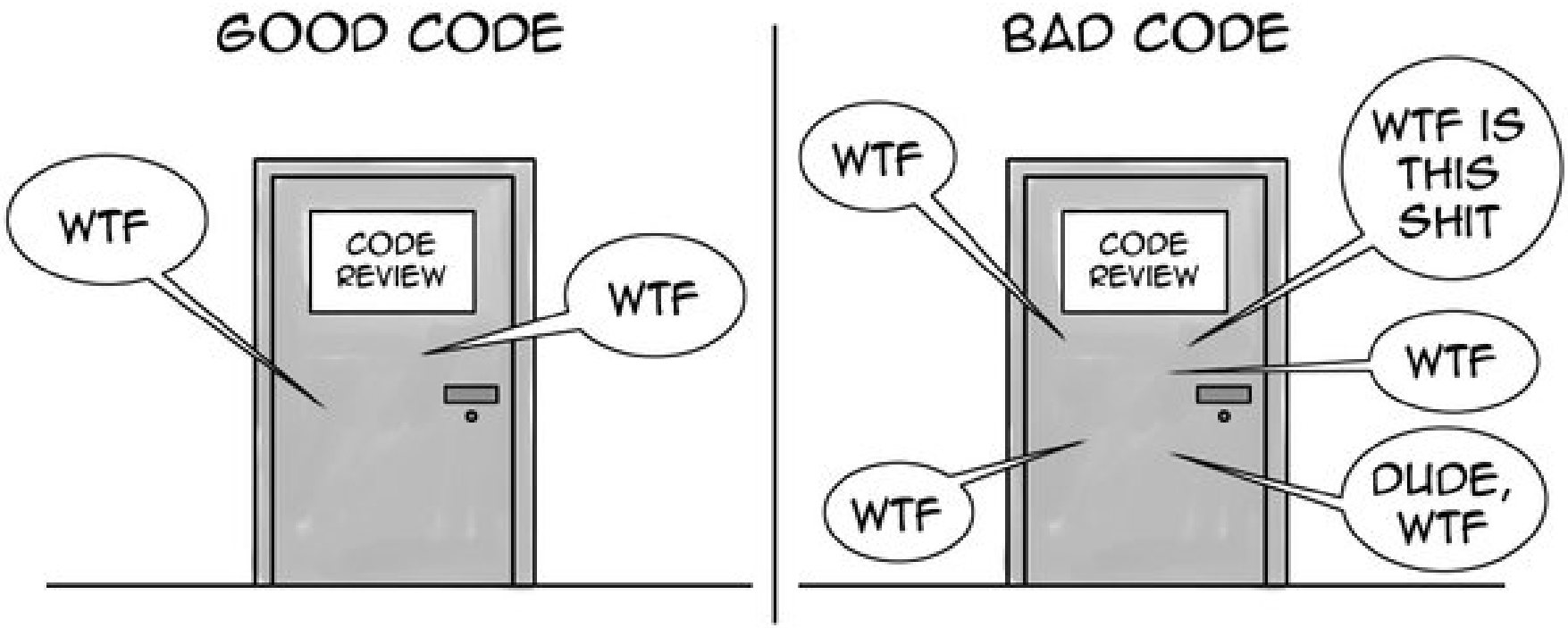
```
## [1] 2
```

There is no multi-line comment in R

Special Comments

Some comments have special meaning.

- `#'` Roxygen documenting comment
- `#!` sometimes configuration comments
- `#<` hey I'm talking about this comment.



THE ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE

What’s in a name … Variables

Assignments

- `a <- 1` assigns the value `1` to the variable `a`.
- `a = 1` does the same, *generally*.
  - The “generally” is why this is also generally discouraged.

Variables

Variables “can” include:

- **letters** (case-sensitive)
- **numbers**
  - as long as it does not start with a number
- period
  - if it starts with a period it is ‘hidden’
- underscore, but may not start with underscore

Getting fancy

- But if you want to be fancy use back ticks ``` and you can do whatever you want.

```
`A variable name that is actually informative` <-
    "A string that is not"
`A variable name that is actually informative`
```

```
## [1] "A string that is not"
```


GOOD IDEA

Good Name, Bad name

Good Name

- `TIU.document.data`
- `fit_model`
- \base model without longitudinal variables``

Bad Name

- `x`
- `y`
- `lm`
- `summary`

Functions

Functions

Functions provide reusable functionality.

Examples:

- `print()`
- `c()`, short for combine
- `cat()`, concatenate
- `lm()`, linear model
- `plot()`, plot data

Calling functions

Call a function with the function name followed by parentheses.

```
print("Hello Bootcamp")
```

```
## [1] "Hello Bootcamp"
```

Aside: Printing

The `print` function is often not called explicitly but rather implicitly. If a value is stored in a variable calling just that variable will print the value out.

```
hb <- "Hello Bootcamp"
hb
```

```
## [1] "Hello Bootcamp"
```

Arguments

In `print("Hello Bootcamp")` the `"Hello Bootcamp"` is an argument.

Named

Arguments can be named such as in

```
plot(data=iris, x=Sepal.Length, y=Petal.Length)
```

Use the `=` to specify the name of any argument

Unnamed

Or they can be unnamed as in the previous example.

Arguments

Mixed

You can mix named and unnamed

```
cat("Hello", "Bootcamp", sep=" ")
```

```
## Hello Bootcamp
```

Creating Functions

Create a function with with the `function` keyword followed by parentheses for the arguments.

```
hello_world <- function(who = 'world', how = 'hello'){
    print(paste(how, who))
}
```

Variadic Arguments

Often functions use the special argument `...`, which is a placeholder for as many arguments as you wish.

Example

```
print
```

```
## function (x, ...)
## UseMethod("print")
## <bytecode: 0x000000001512b530>
## <environment: namespace:base>
```

Print takes arguments `x` and `...` which is used for parameters to affect printing.

`print()` is also an example of a method which we will discuss later.

Operators

Unary

- -, +
- ! logical not,
- ? help,
- ?? help search,
- ~ formula,

Math

- +, - add/subtract,
- *, / multiply/divide,
- ^, ** exponents,
- %% modulus,
- %/% integer division,

Matrix

- : sequence,
- %*% matrix product,
- %o% outer product,
- %x% Kronecker product,

Logical

- <, >, ==, <=, >=, !=
- &/| vectorized ‘and’/‘or’,
- &&/|| singular ‘and’/‘or’,

Precedence

1. : :,
2. $, @
3. ^
4. -x, +x Unary minus/plus
5. :
6. All %%operators 7, *, /
7. +, –
8. <, >, <=, >=, ==, !=
9. !, &, &&, |, ||
10. ~
11. ->, ->>, <-, <<-, =

Types

Vectors

- In R everything is a vector
- Sequential vectors can be created with `:`

```
a <- 1:5
```

- Create longer vectors by combining smaller vectors

```
b <- c(a, 6, 7, 8)
```

Class

An object's class given information about what the class is supposed to be or do.

The `class()` function gives the object class.

```
class(1:2)
```

```
## [1] "integer"
```

```
class(pi)
```

```
## [1] "numeric"
```

```
class('Hello Bootcamp')
```

```
## [1] "character"
```

```
class(print)
```

```
## [1] "function"
```

```
class(iris)
```

```
## [1] "data.frame"
```

Basic Types, i.e. Mode

- integer
- numeric
- logical
- character
- list (i.e. anything)
- *language*

```
mode(pi)
```

```
## [1] "numeric"
```

```
mode(iris)
```

```
## [1] "list"
```

```
mode(print)
```

```
## [1] "function"
```

Vector Operations

most operations expect and operate on vectors.

```
a <- 1:5
a^2
```

```
## [1]  1  4  9 16 25
```

When given a vector that doesn't match the length of the other elements are repeated.

```
a <- 1:5
a^(1:2)
```

```
## Warning in a^(1:2): longer object length is not a multiple of shorter object
## length
```

```
## [1]  1  4  3 16  5
```

If the larger is not a multiple of the shorter a warning will be given.

Indexing (2-types)

- [- subsetting
  - preserves class
  - multiple indices
- [[- extraction
  - no guarantee of class
  - single index

```
# A list holds anything
a <- list(first=1L, second=2, FALSE)
```

```
# by position
class(a[1])
```

```
## [1] "list"
```

```
# by name
class(a['second'])
```

```
## [1] "list"
```

```
# by position
class(a[[1]])
```

```
## [1] "integer"
```

```
# by name
class(a[['second']])
```

```
## [1] "numeric"
```

$ Indexing

lists may also be indexed by the $ sign.

But be careful not all vectors can be.

```
a$first
```

```
## [1] 1
```

```
a$second
```

```
## [1] 2
```

This comes in useful for data.

Data Types

Data for our purpose is:

- A (named) list of variables,
  - where all variables have the same length;
- Rectangular with and rows as observations.
  - columns as variables `ncol(iris)`=5
  - and rows as observations `nrow(iris)`=150

Example Data: Iris

`iris`

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa

Creating Data

```
data.frame(
    x = 1:4,
    y = rnorm(4) #< random normals
)
```

x	y
1	0.3532997
2	-0.2620877
3	-1.5670904
4	0.0536385

We will discuss tibbles in the Tidyverse slides.

SPECIAL

The Pipe %>%

The pipe is a newer but very handy tool in R.

Use it to tie multiple short statements together into a single complex statement that is easy to understand and use. It comes from the `magrittr` package but it is more common to use it through the tidyverse package, which will be covered in detail later.

```
library(tidyverse)
iris %>% #< take iris data
    filter(Species=='setosa') %>% #< perform a filter
    nrow() #< count the rows.
```

```
## [1] 50
```

Next Up

Packages (04-Packages.html)