BGGN 213

Hands-on Lab Session

Class 06

Barry Grant

UC San Diego

http://thegrantlab.org/bggn213

function()

Video Recap:

- Covered the When, Why, What and How of writing your own R functions.

…

Video Recap:

- Covered the **When**, Why, What and How of writing your own R functions.
  - ➡ **When**: When you find yourself doing the same thing 3 or more times with repetitive code consider writing a function.

. . .

Video Recap:

- Covered the When, **Why**, What and How of writing your own R functions.

  ➡ **Why**:

  1. Makes the purpose of the code more clear
  2. Reduces mistakes from copy/paste
  3. Makes updating your code easer
  4. Reduces code duplication and facilitates re-use.

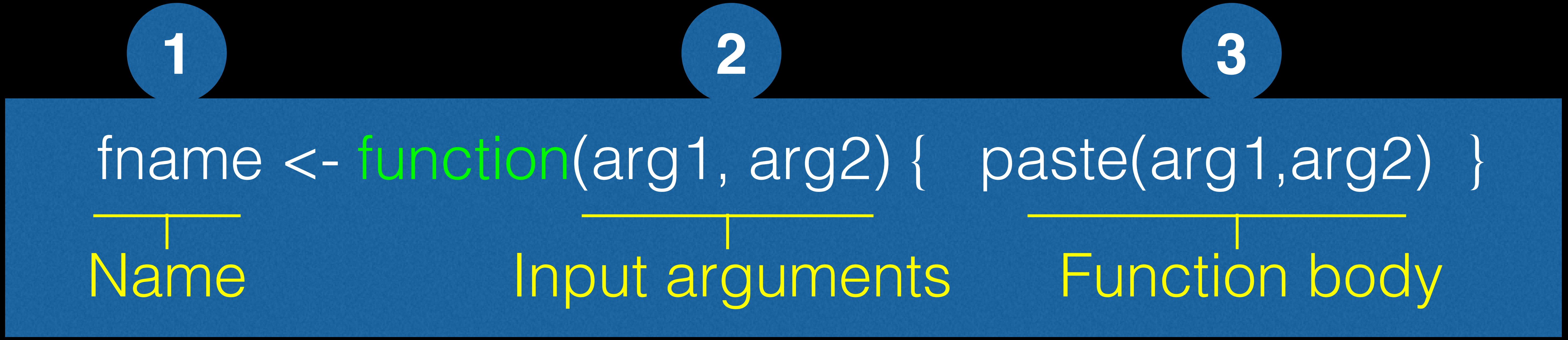
…

Video Recap:

- Covered the When, Why, **What** and How of writing your own R functions.

  ➡ **What**: A function is defined with:

  1. A user selected name,
  2. A comma separated set of input arguments, and
  3. Regular R code for the function body



```
fname <- function(arg1, arg2) {   paste(arg1,arg2)  }
```

. . .

Video Recap:

- **How**: Follow a step-by-step procedure to go from working code snippet to refined and tested function.

1. Start with a simple problem and write a working snippet of code.
2. Rewrite for clarity and to reduce duplication
3. Then, and only then, turn into an initial function
4. Test on small well defined input
5. Report on potential problem by failing early and loudly!

Lab session!

Your turn...

- Write a function grade() to determine an overall grade from a vector of student homework assignment scores dropping the lowest single alignment score.

```
# student 1
c(100, 100, 100, 100, 100, 100, 100, 90)

# student 2
c(100, NA, 90, 90, 90, 90, 97, 80)
```

Lab session!

Your turn...

- Write a function grade() to determine an overall grade from a vector of student homework assignment scores dropping the lowest single alignment score.

```
# student 1
c(100, 100, 100, 100, 100, 100, 100, 90)

# student 2
c(100, NA, 90, 90, 90, 90, 97, 80)

# now grade all students in an example class
url <- "https://tinyurl.com/gradeinput"
```

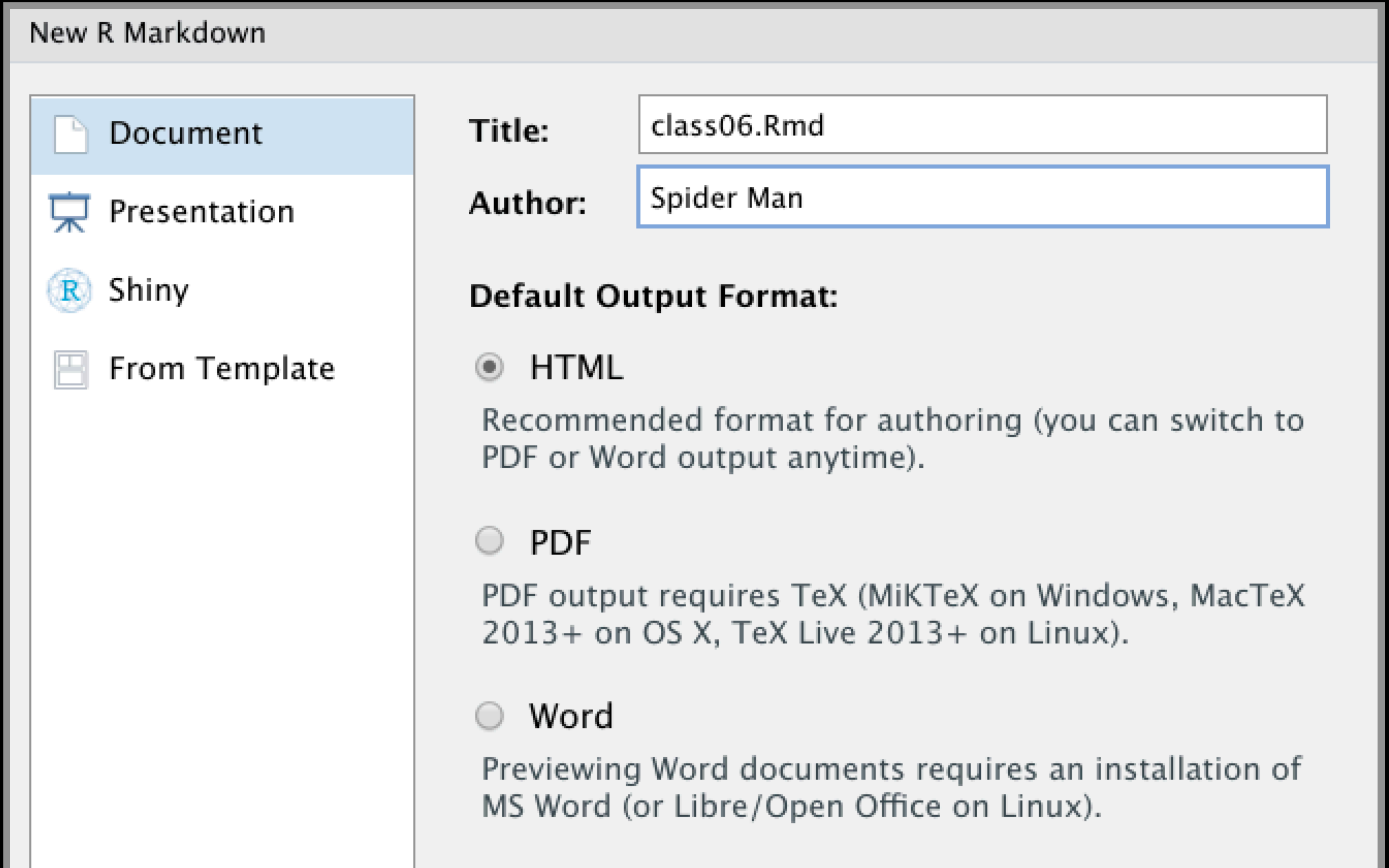
Lab session!

RStudio®

Create a new **Project** for class06

N.B. Open a new **Rmarkdown** document
(Our goal is to make a PDF report with notes and plots)

File > New File > Rmarkdown



New R Markdown

Title: class06.Rmd

Author: Spider Man

Default Output Format:

[✓ HTML]

Recommended format for authoring (you can switch to PDF or Word output anytime).

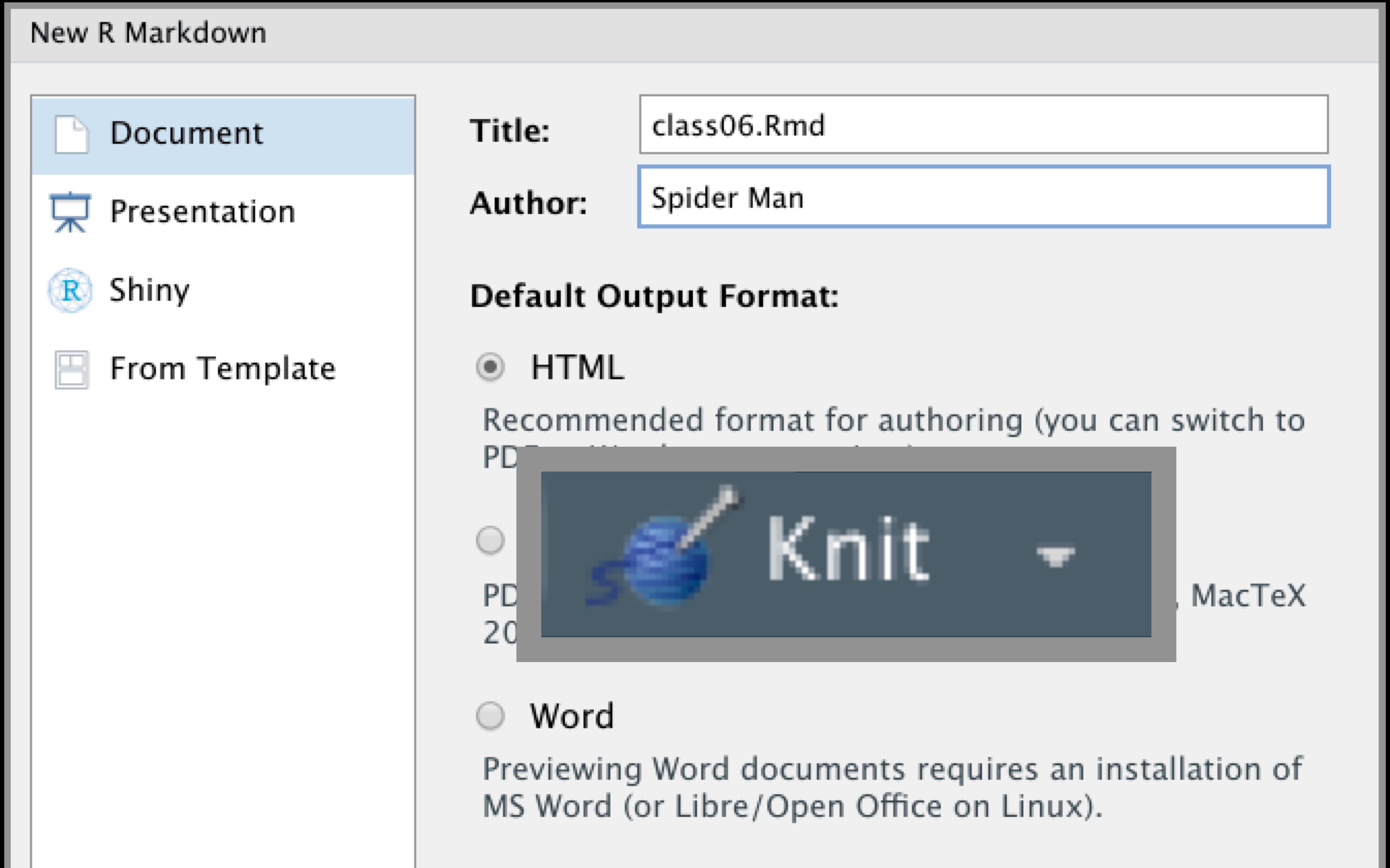
PDF

PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

Word

Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

File > New File > Rmarkdown



New R Markdown

Title: class06.Rmd

Author: Spider Man

Default Output Format:

[✓ HTML]

Recommended format for authoring (you can switch to PD

PD

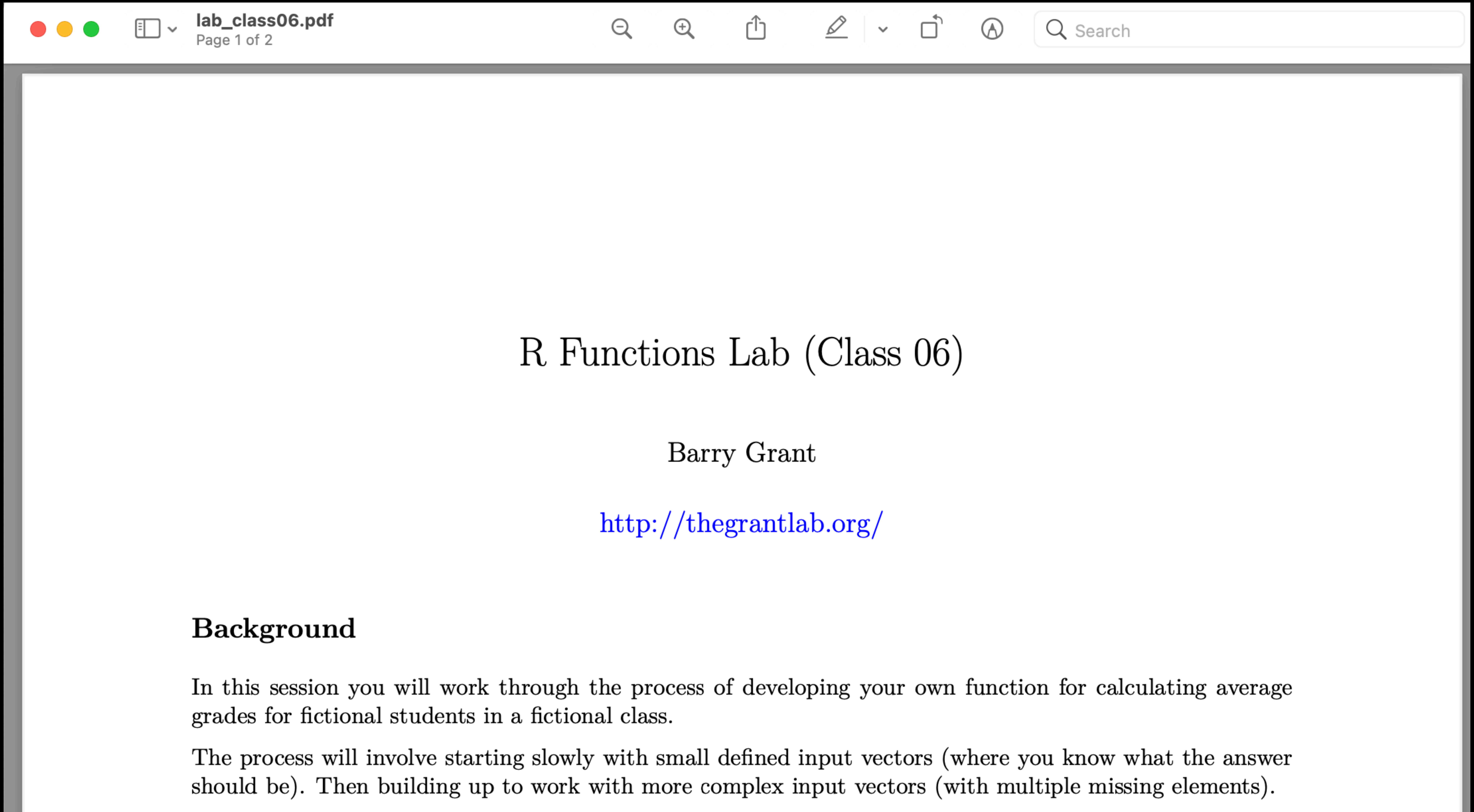
20

, MacTeX

Word

Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

Lab sheet



R Functions Lab (Class 06)

Barry Grant

http://thegrantlab.org/

Background

In this session you will work through the process of developing your own function for calculating average grades for fictional students in a fictional class.

The process will involve starting slowly with small defined input vectors (where you know what the answer should be). Then building up to work with more complex input vectors (with multiple missing elements).

And some homework….

Do it Yourself!

```
# Can you improve this analysis code?
library(bio3d)
s1 <- read.pdb("4AKE")  # kinase with drug
s2 <- read.pdb("1AKE")  # kinase no drug
s3 <- read.pdb("1E4Y")  # kinase with drug

s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s1, chain="A", elety="CA")

s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b

plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
plotb3(s2.b, sse=s2.chainA, typ="l", ylab="Bfactor")
plotb3(s3.b, sse=s3.chainA, typ="l", ylab="Bfactor")
```

Suggested steps for writing your functions

1. Start with a simple problem and get a working snippet of code
2. Rewrite to use temporary variables (e.g. x, y, df, m etc.)
3. Rewrite for clarity and to reduce calculation duplication
4. Turn into an initial function with clear useful names
5. Test on small well defined input and (subsets of) real input
6. Report on potential problem by failing early and loudly!
7. Refine and polish

Side-Note: What makes a good function?

- Correct
- Understandable (remember that functions are for humans and computers)
- Correct + Understandable = **Obviously correct**
- Use sensible names throughout. What does this code do?

```
baz <- foo(df, v=0)
df2 < replace_missing(df, value=0)
```

- Good names make code understandable with minimal context. You should strive for self-explanatory names

Recap From Last Time:

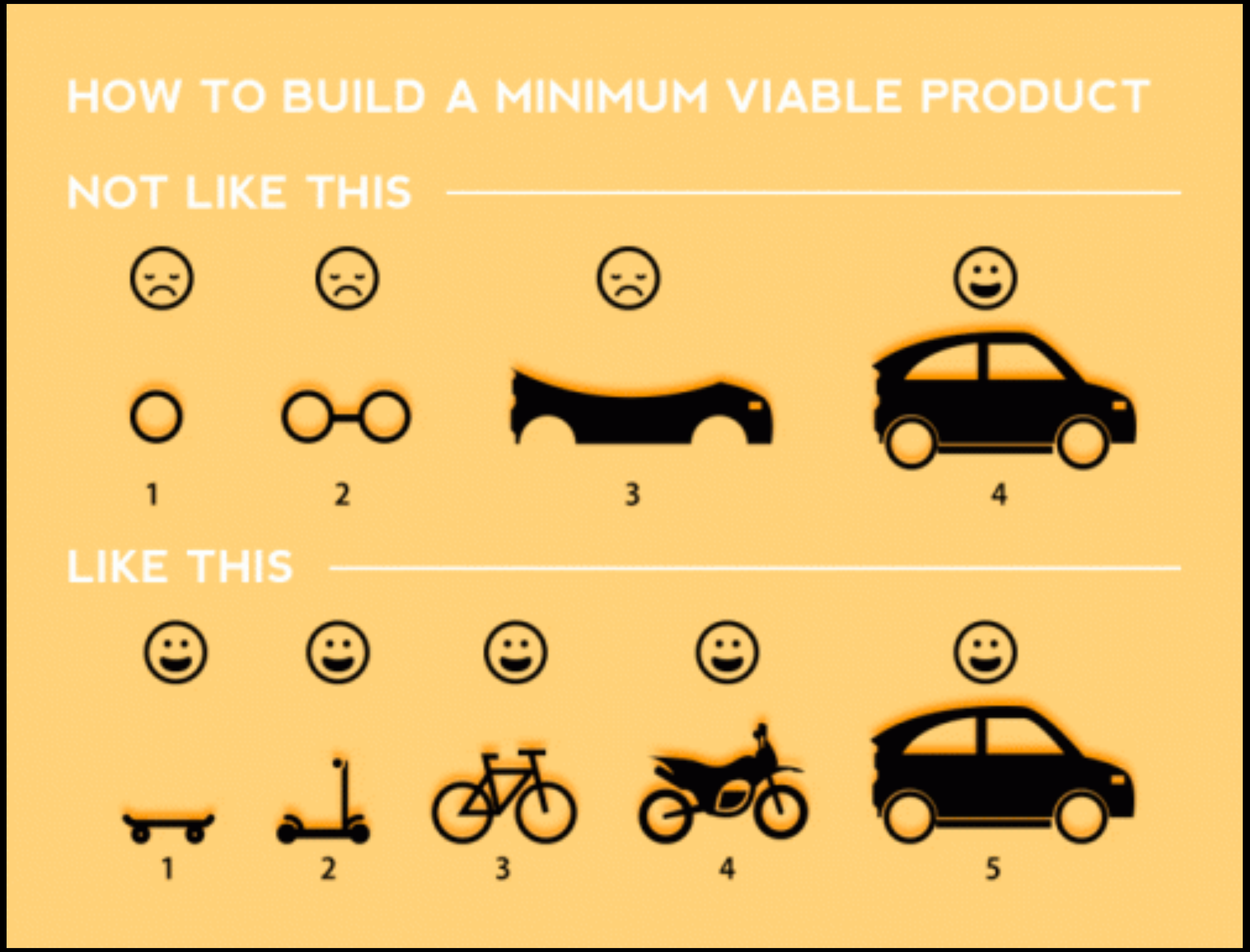
➡ **<u>How</u>**: Follow a step-by-step procedure to go from working code snippet to refined and tested function.

1. Start with a simple problem and write a working snippet of code.
2. Rewrite for clarity and to reduce duplication
3. Then, and <u>only then</u>, turn into an initial function
4. Test on small well defined input
5. Report on potential problem by failing early and loudly!

…

Recap…

1. Start with a simple problem and write a working snippet of code.



[Image credit: Spotify development team]

Build that skateboard before you build the car.

A limited but functional thing is very useful and keeps the spirits high.

Suggested steps for writing your functions

1. Start with a simple problem and get a working snippet of code
2. Rewrite to use temporary variables (e.g. x, y, df, m etc.)
3. Rewrite for clarity and to reduce calculation duplication
4. Turn into an initial function with clear useful names
5. Test on small well defined input and (subsets of) real input
6. Report on potential problem by failing early and loudly!
7. Refine and polish

BGGN 213

Hands-on Lab Session

Class 06

Barry Grant

UC San Diego

http://thegrantlab.org/bggn213