How to write a function

How should you write a function?

1. Start with a simple problem.
2. Get some working code to solve your simplified problem.
3. Rewrite the code to use temporary variables.
4. Finally turn the code into a function using the function template.

1. Start with a simple problem: Know how you want to do what

- A simple problem should be concrete and you should know how to solve it.
- If you have a complicated task to perform, it might make sense to break it down in multiple tasks and thus, multiple functions.
- Example:

  Estimate the ROI for this company

Do NOT think first of the function template (`function(){…}`).

$$ROI = \frac{Total\ Revenue - Total\ Cost}{Total\ Cost}\ x\ 100$$

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
…	…	…	…	…

1. Start with a simple problem: Know how you want to do what

- A simple problem should be concrete and you should know how to solve it.
- If you have a complicated task to perform, it might make sense to break it down in multiple tasks and thus, multiple functions.
- Example:

  Estimate the ROI for a project

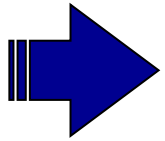
Do NOT think first of the function template (`function(){…}`).

$$ROI = \frac{Total\ Revenue - Total\ Cost}{Total\ Cost}\ x\ 100$$

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
…	…	…	…	…

1. Start with a simple problem: Use toy data

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
…	…	…	…	…



PurchAmount	Cost
3	1
4	2
5	3
6	4
7	5

Create a toy dataset which only includes the focal variables, i.e. `PurchAmount` and `Cost`, and use values for which you can easily check if the processing worked correctly. We can easily calculate with this toy data the sum of `PurchAmount` (25) and `Cost` (15). The ROI should be 10/15 * 100 = 67

```
simdata <- data.table(PurchAmount=3:7,Cost=1:5)
```

2. Get some working code

```
(sum(simdata[,PurchAmount], na.rm=T)-
        sum(simdata[,Cost], na.rm=T))/
        sum(simdata[,Cost], na.rm=T)*
        100
```

Simple code to calculate ROI (i.e. (revenue – cost)/cost)

3. Rewrite the code using temporary variables

```
(sum(simdata[,PurchAmount], na.rm=T)-
        sum(simdata[,Cost], na.rm=T))/
        sum(simdata[,Cost], na.rm=T)*
        100
```

```
x <- simdata
r <- "PurchAmount"
c <- "Cost"

(sum(x[,r],na.rm=T)-
        sum(x[,c],na.rm=T))/
        sum(x[,c],na.rm=T)*
        100
```

Rewrite the code to use temporary variables

3. Rewrite the code using temporary variables

```
(sum(simdata[,PurchAmount], na.rm=T)-
        sum(simdata[,Cost], na.rm=T))/
        sum(simdata[,Cost], na.rm=T)*
        100
```

```
x <- simdata
r <- "PurchAmount"
c <- "Cost"

(sum(x[,r],na.rm=T)-
        sum(x[,c],na.rm=T))/
        sum(x[,c],na.rm=T)*
        100
```

Rewrite the code to use temporary variables

4. Turn your code into a function (1/4)

Choose an appropriate name for the function

```
ROI <- function(){

}
```

4. Turn your code into a function (2/4)

Define the input arguments for the function

```
ROI <- function(x,r,c){

}
```

4. Turn your code into a function (3/4)

```
ROI <- function(x,r,c){
        (sum(x[,r], na.rm=T)-
        sum(x[,c], na.rm=T))/
        sum(x[,c], na.rm=T)*100
}
```

Add the code with the temporary variables

4. Turn your code into a function (4/4)

```
ROI <- function(x,r,c){
        (sum(x[,r], na.rm=T)-
        sum(x[,c], na.rm=T))/
        sum(x[,c], na.rm=T)*100
}
```

Test the function with your toy dataset

```
ROI(simdata, "PurchAmount", "Cost")
```

OUTPUT:

```
[1] 66.66667
```

4. Turn your code into a function (4/4)

```
ROI <- function(x,r,c){
        (sum(x[,r], na.rm=T)-
        sum(x[,c], na.rm=T))/
        sum(x[,c], na.rm=T)*100
}
```

Test the function with your toy dataset

```
ROI(simdata, "PurchAmount", "Cost")
```

OUTPUT:

```
[1] 66.66667
```

Apply your function

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

```
ROI_1 <- ROI(myData,"PurchAmount","Cost")
```

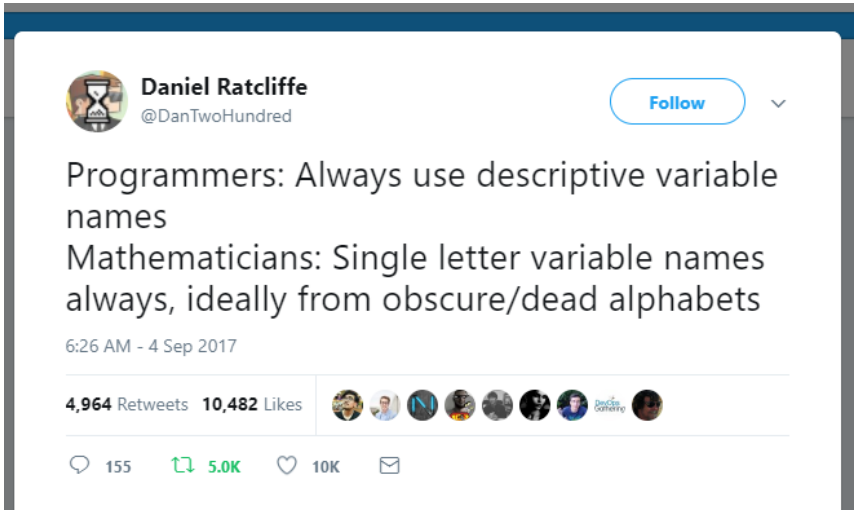
Apply the function to the real dataset

OUTPUT:

```
[1] 115.7332
```

You know now the procedure to write a function…
… but what makes a good function?

- It solves a problem correctly.
- It performs a single operation.
- It is understandable (to other users), e.g. use adequate naming for the variables processed and the function itself.



https://twitter.com/DanTwoHundred/status/904697259345969152

Understandability: How NOT to name functions

```
baz <- foo(bar, qux)
```

Anyone any clue what this function actually does?
No way, without context this line of code is meaningless…

1 Probably a new `data.frame` is created

2 The old `date.frame` is probably rearranged

```
df2 <- arrange(df, qux)
```

Using adequate names helps us to increase our understanding.

But still, we don't know enough to use the function without more information.

3 Same applies to our `ROI` function, which we have written previously, probably the naming was not perfect

Understandability: How NOT to name functions

```
baz <- foo(bar, qux)
```

Anyone any clue what this function actually does?
No way, without context this line of code is meaningless…

1 Probably a new `data.frame` is created

2 The old `date.frame` is probably rearranged

```
df2 <- arrange(df, qux)
```

Using adequate names helps us to increase our understanding.

But still, we don't know enough to use the function without more information.

3 Same applies to our `ROI` function, which we have written previously, probably the naming was not perfect

Understandability:
How to name functions

Generally use verbs (since functions are “doing” something) and be descriptive of what is done.

```
#super bad
myFirstAwesomeFunction()

#bad
firstPurchases()

#good
removeFirstPurchases()
```