



Week 3: *Creating Functions*

EMSE 4574: Intro to Programming for Analytics

John Paul Helveston

September 15, 2020

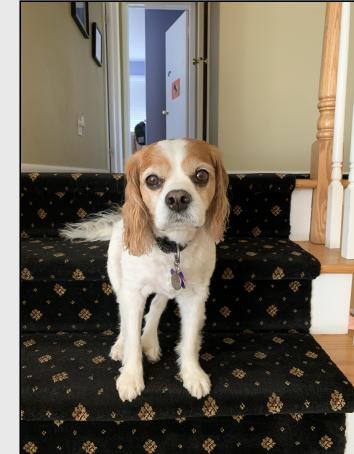
Quiz 2

Go to **#classroom** channel in Slack for link

06 : 00

Thanks for the cute animals:

Olivia Z., Kyara M., Eliese O., Helena R., David R., Carolyne I.



Video on?

It's nice to see your faces :)



If you're okay with it, please turn on your camera - it creates a more engaging discussion environment and an opportunity for us to get to know each other better. Your privacy is important though, and I understand if you wish to keep it off. No pressure.

Week 3: *Creating Functions*

1. Function syntax
2. Local vs global variables
3. Top-down design
4. Coding style

Week 3: *Creating Functions*

1. Function syntax
2. Local vs global variables
3. Top-down design
4. Coding style

Basic function syntax

```
functionName <- function(arguments) {  
  # Do stuff here  
  return(something)  
}
```

Basic function syntax

In English:

"`functionName` is a **function** of **arguments** that does..."

```
functionName <- function(arguments) {  
  # Do stuff here  
  return(something)  
}
```

Basic function syntax

Example:

"squareRoot is a **function** of **n** that...returns the square root of **n**"

```
squareRoot <- function(n) {  
  return(n0.5)  
}
```

```
squareRoot(64)
```

```
## [1] 8
```

`return()` and `cat()` statements

```
isPositive <- function(n) {  
  return(n > 0)  
}
```

```
isPositive <- function(n) {  
  cat(n > 0)  
}
```

`return()` and `cat()` statements

```
isPositive <- function(n) {  
  return(n > 0)  
}
```

`return()` returns back a value

```
isPositive <- function(n) {  
  cat(n > 0)  
}
```

`cat()` prints a value as a string

```
test <- isPositive(7)  
test
```

TRUE

```
test <- isPositive(7)
```

TRUE

```
test
```

Error: object 'test' not found

cat() is short for "concatenating"

```
print_x <- function(x) {  
  cat("The value of x is", x)  
}
```

```
print_x(7)
```

```
## The value of x is 7
```

```
print_x_squared <- function(x) {  
  cat("The value of x is", x, "and the value of x^2 is", x^2)  
}
```

```
print_x_squared(7)
```

```
## The value of x is 7 and the value of x^2 is 49
```

`cat()` adds a space between values by default

```
print_x <- function(x) {  
  cat("The value of x is", x)  
}
```

```
print_x(7)
```

```
## The value of x is 7
```

Modify separator with the `sep` argument:

```
print_x <- function(x) {  
  cat("The value of x is", x, sep = ": ")  
}
```

```
print_x(7)
```

```
## The value of x is: 7
```

02:00

Code tracing practice

Consider these functions:

```
f1 <- function(x) {  
  return(x^3)  
}  
f2 <- function(x) {  
  cat(x^3)  
}  
f3 <- function(x) {  
  cat(x^3)  
  return(x^4)  
}  
f4 <- function(x) {  
  return(x^3)  
  cat(x^4)  
}
```

What will these lines of code produce?

Write your answer down first, *then* run the code to check.

```
f1(2)  
f2(2)  
f3(2)  
f4(2)
```

Week 3: *Creating Functions*

1. Function syntax
2. Local vs global variables
3. Top-down design
4. Coding style

Local objects

All objects inside function are "**local**" - they don't exist in the *global* environment

Example:

```
squareOfX <- function(x) {  
  y <- x^2      # y here is "local"  
  return(y)  
}
```

```
squareOfX(3)
```

```
## [1] 9
```

If you try to call `y`, you'll get an error:

```
y
```

```
Error: object 'y' not found
```

Global objects

Global objects exist in the main environment.

NEVER, NEVER, NEVER call global objects inside functions.

```
print_x <- function(x) {  
  cat(x)  
  cat(n) # n is global!  
}  
  
n <- 5 # Define n in the *global* environment  
  
print_x(5)
```

```
n <- 6  
  
print_x(5)
```

```
## 56
```

Function behavior shouldn't change with the same arguments!

```
## 55
```

Global objects

All objects inside functions should be **arguments** to that function

```
print_x <- function(x, n = NULL) {  
  cat(x)  
  cat(n) # n is local!  
}
```

n <- 5 # Define n in the *global* environment

```
print_x(5)
```

```
## 5
```

```
n <- 6  
print_x(5)
```

```
## 5
```

Use n as argument:

```
print_x(5, n)
```

```
## 56
```

02:00

Code tracing practice

Consider this code:

```
x <- 7
y <- NULL
f1 <- function(x) {
  cat(x^3)
  cat(y, x)
}
f2 <- function(x, y = 7) {
  cat(x^3, y)
}
f3 <- function(x, y) {
  cat(x^3)
  cat(y)
}
f4 <- function(x) {
  return(x^3)
  cat(x^4)
}
```

What will these lines of code produce?

Write your answer down first, *then* run the code to check.

```
f1(2)
f2(2)
f3(2)
f4(2)
```

Break

05 : 00

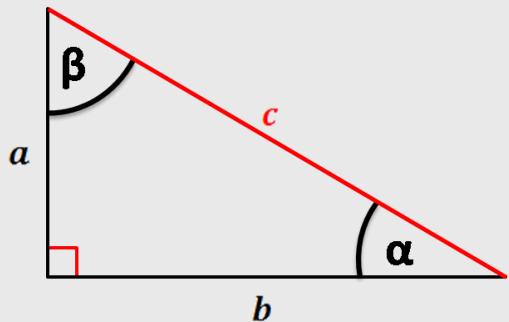
Week 3: *Creating Functions*

1. Function syntax
2. Local vs global variables
3. Top-down design
4. Coding style

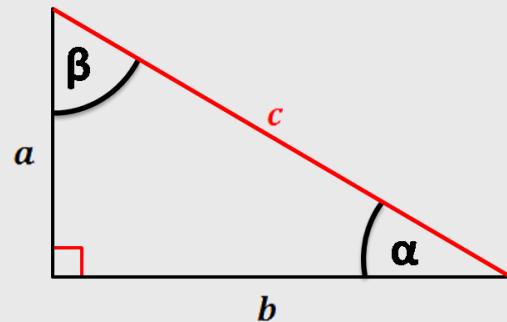
"Top Down" design

1. Break the problem into pieces
2. Solve the "highest level" problem first
3. Then solve the smaller pieces

Example: Given values a and b ,
find the value c such that the
triangle formed by lines of length a ,
 b , and c is a right triangle (in short,
find hypotenuse)



Example: Given values a and b , find the value c such that the triangle formed by lines of length a , b , and c is a right triangle (in short, find hypotenuse)



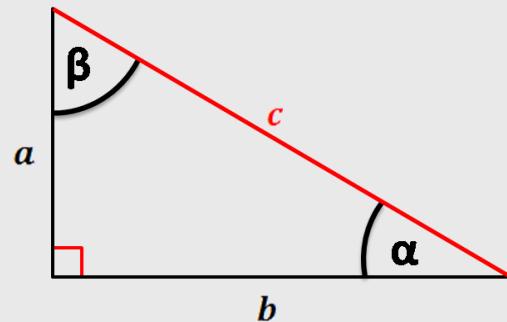
Hypotenuse: $c = \sqrt{a^2 + b^2}$

Break the problem into two pieces:

$$c = \sqrt{x}$$

$$x = a^2 + b^2$$

Example: Given values a and b , find the value c such that the triangle formed by lines of length a , b , and c is a right triangle (in short, find hypotenuse)



Hypotenuse: $c = \sqrt{a^2 + b^2}$

Break the problem into two pieces:

$$c = \sqrt{x}$$

```
hypotenuse <- function(a, b) {  
  return(sqrt(sumOfSquares(a, b)))  
}
```

$$x = a^b + b^2$$

```
sumOfSquares <- function(a, b) {  
  return(a^2 + b^2)  
}
```

10:00

Think-Pair-Share

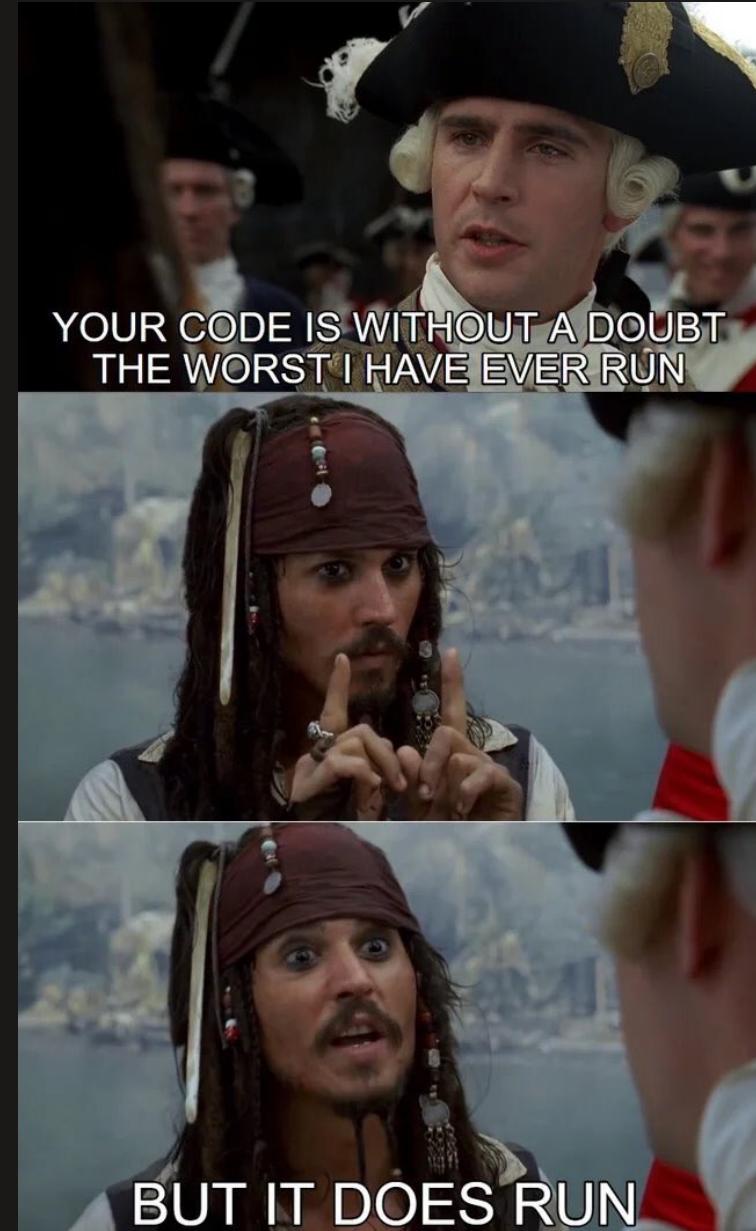
Create a function, `isRightTriangle(a, b, c)` that returns `TRUE` if the triangle formed by the lines of length `a`, `b`, and `c` is a right triangle and `FALSE` otherwise. Use the `hypotenuse(a, b)` function in your solution. **Hint:** you may not know which value (`a`, `b`, or `c`) is the hypotenuse.

```
hypotenuse <- function(a, b) {  
  return(sqrt(sumOfSquares(a, b)))  
}
```

```
sumOfSquares <- function(a, b) {  
  return(a^2 + b^2)  
}
```

Week 3: *Creating Functions*

1. Function syntax
2. Local vs global variables
3. Top-down design
4. Coding style



Style matters!

Which is easier to read?

V1:

```
sumofsquares<-function(a,b)return(a2 + b2)
```

V2:

```
sum_of_squares <- function(a, b) {  
  return(a2 + b2)  
}
```

Style matters!

Which is easier to read?

V1:

```
sumofsquares<-function(a,b)return(a2 + b2)
```

V2: <- **This one is much better!**

```
sum_of_squares <- function(a, b) {  
  return(a2 + b2)  
}
```

Use the "Advanced R" style guide:

<http://adv-r.had.co.nz/Style.html>

Other good style tips on [this blog post](#)

Style guide: Objects

Using = instead of <- for assignment



- Use `<-` for assignment, not `=`
- Put spacing around operators
(e.g. `x <- 1`, not `x<-1`)
- Use meaningful variable names
- This applies to file names too
(e.g. "`hw1.R`" vs. "`untitled.R`")

Style guide: Functions

Generally, function names should be verbs:

```
add()      # Good  
addition() # Bad
```

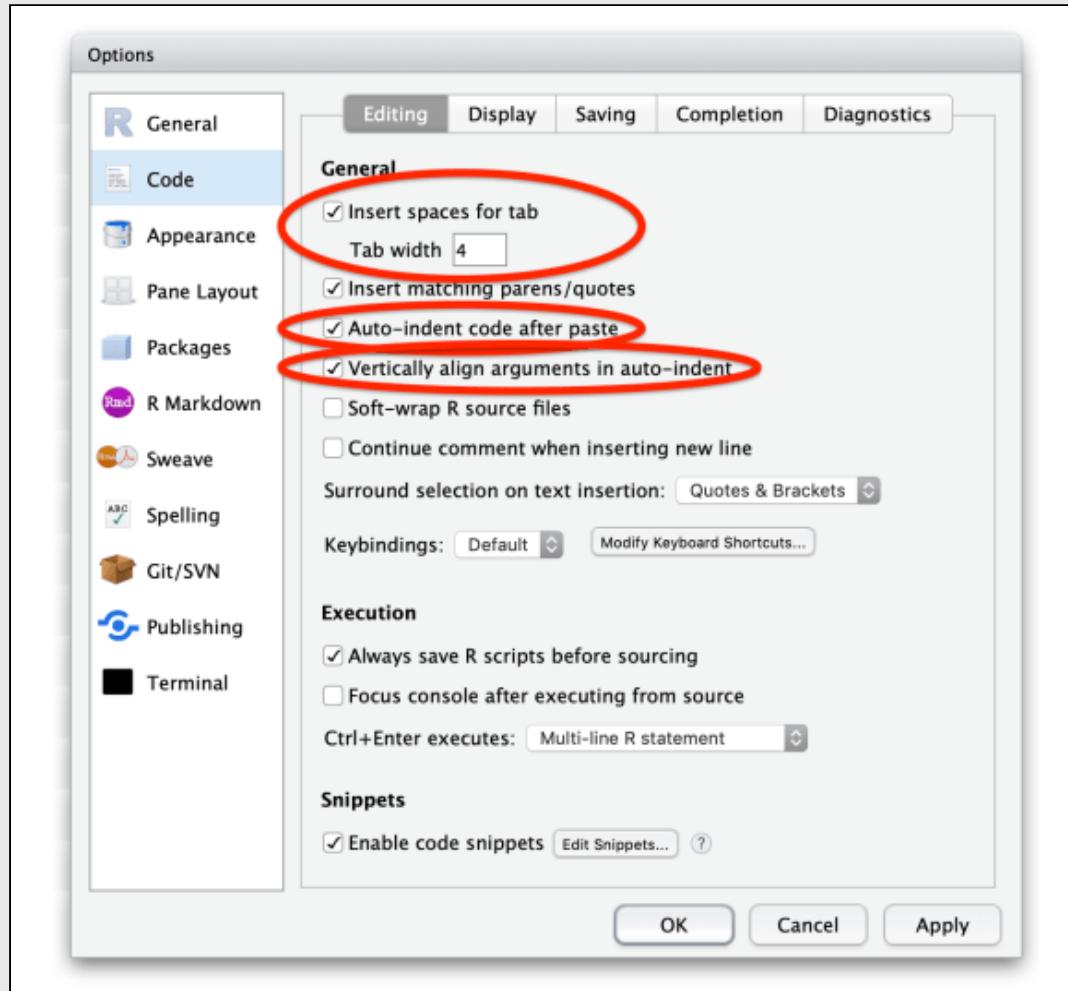
Avoid using the `"."` symbol:

```
get_hypotenuse() # Good  
get.hypotenuse() # Bad
```

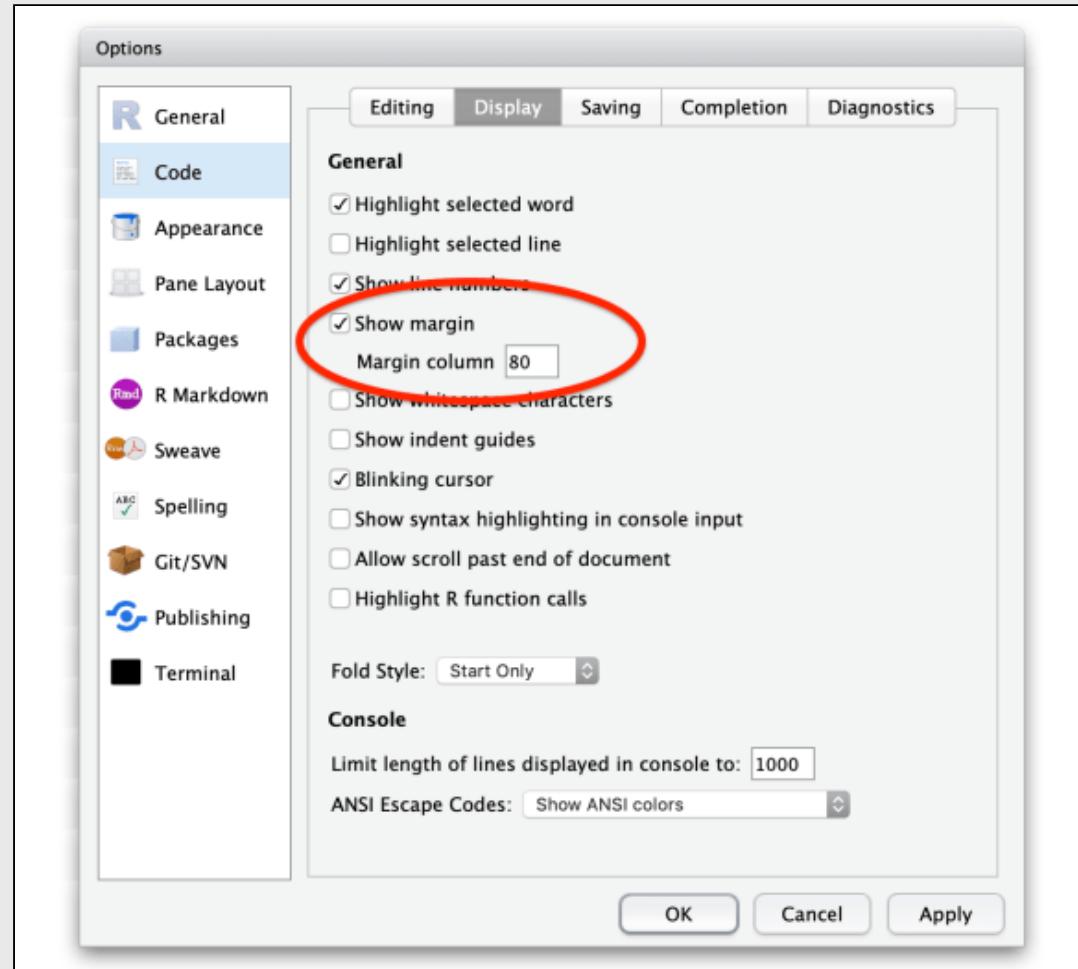
Use curly braces, with indented code inside:

```
sum_of_squares <- function(a, b) {  
  return(a^2 + b^2)  
}
```

Indent by 4 spaces



Set line length to 80



15:00

Think-Pair-Share

onesDigit(x): Write a function that takes an integer and returns its ones digit.

Tests:

- `onesDigit(123) == 3`
- `onesDigit(7890) == 0`
- `onesDigit(6) == 6`
- `onesDigit(-54) == 4`

tensDigit(x): Write a function that takes an integer and returns its tens digit.

Tests:

- `tensDigit(456) == 5`
- `tensDigit(23) == 2`
- `tensDigit(1) == 0`
- `tensDigit(-7890) == 9`

Hint #1:

You may want to use `onesDigit(x)` as a helper function for `tensDigit(x)`

Hint #2:

The mod operator (`%%`) "chops" a number
and returns everything to the *right*

```
123456 %% 1
```

```
## [1] 0
```

```
123456 %% 10
```

```
## [1] 6
```

The integer divide operator (`%/%`) "chops" a
number and returns everything to the *left*

```
123456 %/% 1
```

```
## [1] 123456
```

```
123456 %/% 10
```

```
## [1] 12345
```

15:00

Think-Pair-Share

`eggCartons(eggs)`: Write a function that reads in a non-negative number of eggs and prints the number of egg cartons required to hold that many eggs. Each egg carton holds one dozen eggs, and you cannot buy fractional egg cartons.

- `eggCartons(0) == 0`
- `eggCartons(1) == 1`
- `eggCartons(12) == 1`
- `eggCartons(25) == 3`

`militaryTimeToStandardTime(n)`: Write a function that takes an integer between 0 and 23 (representing the hour in military time), and returns the same hour in standard time.

- `militaryTimeToStandardTime(0) == 12`
- `militaryTimeToStandardTime(3) == 3`
- `militaryTimeToStandardTime(12) == 12`
- `militaryTimeToStandardTime(13) == 1`
- `militaryTimeToStandardTime(23) == 13` / 37

HW 3

- Use the template
- Use Polya's problem solving technique:
 1. Understand the problem
 2. Devise a plan
 3. Carry out the plan
 4. Check your work
- Try out the autograder (Saurav will DM you your password on Slack)