**Quiz:**

You can hand-write the following for “And”: &

Hint: When working these out, it can be helpful to substitute the value for the variable.

**Slide 12:**

mySqrt <- function(n) {

return(n^0.5)

}

**Slide 14:**

isPositive <- function(n) {

return(n > 0)

}

isPositive <- function(n) {

cat(n > 0)

}

**Slide 36:**

hypotenuse <- function(a, b) {

return(sqrt(sumOfSquares(a, b)))

}

sumOfSquares <- function(a, b) {

return(a^2 + b^2)

}

**Slide 53:**

# -----------------------------------------------------------------------------

Ones's Digit

# Stub Solution

This step is not required but is quite useful. We'll just write a tiny version (or "stub") of our solution which does not really try to solve the problem, instead returning some bogus answer. Why? This lets us run our test code, at least, and watch it successfully determine that our "stub" does not actually work properly. Here is what we mean:

onesDigit <- function(x) {

return(3)

}

# Now run the tests:

onesDigit(123) == 3

onesDigit(7890) == 0

onesDigit(6) == 6

onesDigit(-54) == 4

But this is just what we expected! Well, mostly. Curiously, upon inspection we see that we passed the first test. By sheer luck, when we returned 3, we passed the first test

# Solve, Test, Repeat

In this case, it's actually quite simple: the x %% y is the remainder when x is divided by y, and the 1's digit is nothing more than the remainder when we divide a number by 10 (right?). And so we get this:

onesDigit <- function(x) {

return(x %% 10) # first attempt!

}

<Run the tests>

Darn. But at least we passed the first three tests. So it's a start. So it looks like we works for positives and not for negatives. Why not? Think about how the remainder function works differently for positives and for negatives and you should (hopefully) realize our mistake.

Regardless of whether you see why, it should be easy to see what the fix is (well, at least one fix, among several options): just take the absolute value so we're only dealing with positives! So we get:

onesDigit <- function(x) {

return(abs(x) %% 10)

}

PASSED!

# -----------------------------------------------------------------------------

Ten's Digit

Stub Solution

tensDigit <- function(x) {

return(3)

}

tensDigit(456) == 5

tensDigit(23) == 2

tensDigit(1) == 0

tensDigit(-7890) == 9

# Solve, Test, Repeat

Say that after some thought we think (x %/% 10) might be a solution here. So, let's try it:

tensDigit <- function(x) {

return(x %/% 10) # our first attempt

}

<Run tests>

Darn. We failed on the first test. But why? Well, let’s print it out:

tensDigit(456)

And we get:

45

Ahhh! Now we see what went wrong. We meant to only get that 5, but we got more than we bargained for! We got all the digits except the one's digit. So by dividing x by 10, what we really did was shift x one digit to the right. Re-read that last sentence and think about it.

Now, after shifting to the right, the old ten's digit has become the new one's digit. Right? Yes!!!! So now we have a plan to fix our code:

tensDigit <- function(x) {

return(onesDigit(x %/% 10))

}

Hooray!

But wait...what happens with we call tensDigit(-11)?

For one thing, we realize our test cases were incomplete (whoops), since they did not expose this

bug. Figure it out, and fix both our code and our test cases!

tensDigit <- function(x) {

return(onesDigit(abs(x) %/% 10))

}

**Slide 55:**

# -----------------------------------------------------------------------------

eggCartons

Stub solution

eggCartons <- function(eggs) {

return(eggs)

}

eggCartons(0) == 0

eggCartons(1) == 1

eggCartons(12) == 1

eggCartons(13) == 2

eggCartons(24) == 2

eggCartons(25) == 3

This feels like a division problem, right?

eggCartons <- function(eggs) {

return(eggs / 12)

}

eggCartons <- function(eggs) {

return(ceiling(eggs / 12))

}

Another approach:

eggCartons <- function(eggs) {

return((eggs + 11) %/% 12)

}

# -----------------------------------------------------------------------------

militaryTimeToStandardTime

Stub solution:

militaryTimeToStandardTime <- function(militaryTime) {

return(12)

}

militaryTimeToStandardTime(0) == 12

militaryTimeToStandardTime(1) == 1

militaryTimeToStandardTime(11) == 11

militaryTimeToStandardTime(12) == 12

militaryTimeToStandardTime(13) == 1

militaryTimeToStandardTime(23) == 11

militaryTimeToStandardTime <- function(militaryTime) {

return(1 + (militaryTime-1) %% 12)

}