



## Week 5: *Iteration*

🏛️ EMSE 4571: Intro to Programming for Analytics

👤 John Paul Helveston

📅 February 11, 2022

# Quiz 3

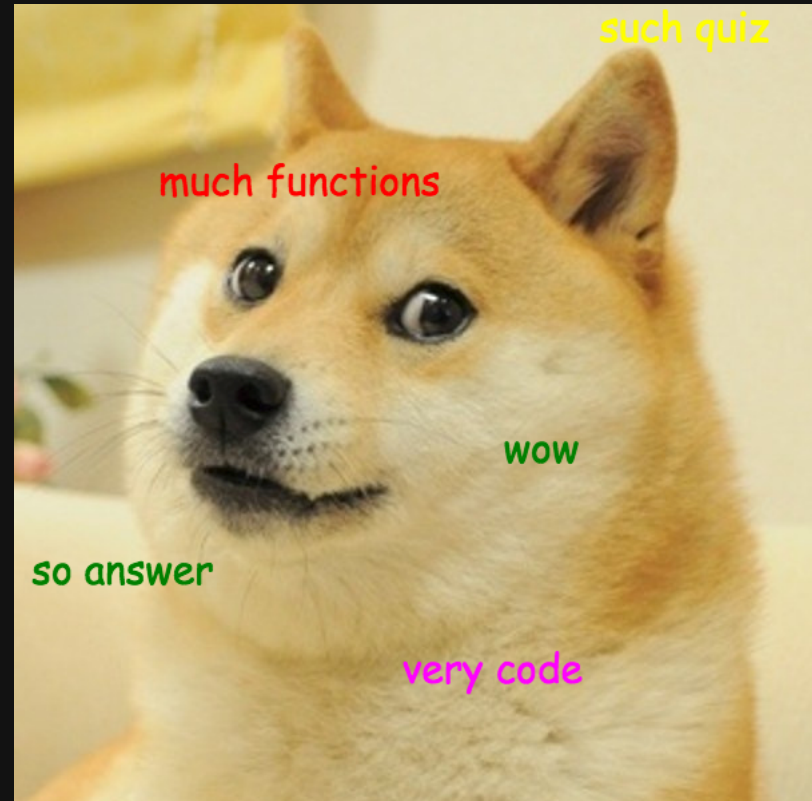
05:00

Go to **#class** channel in Slack for quiz link

Open RStudio first!

## Rules:

- You may use your notes and RStudio
- You may **not** use any other resources (e.g. the internet, your classmates, etc.)



# Notes on common problems in homeworks

Use `almostEqual()` in test cases with numbers

This could fail on you:

```
stopifnot(getTheCents(2.45) == 45)
```

Instead, use:

```
stopifnot(almostEqual(getTheCents(2.45), 45))
```

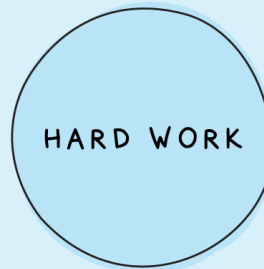
# Notes on common problems in homeworks

## Check your full script for errors

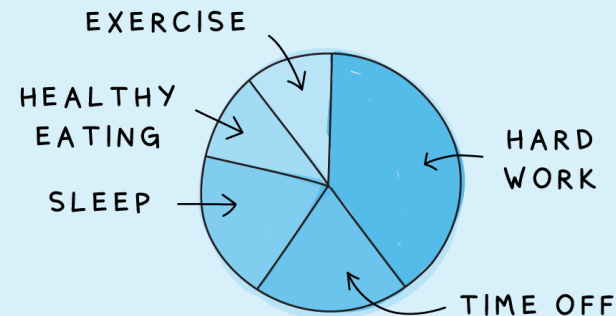
- Restart R and run your whole code from the top
- **Sequence matters:** Have you called a function before defining it?

# Reconsidering productivity

WHAT I THOUGHT WOULD  
MAKE ME PRODUCTIVE



WHAT ACTUALLY DOES



@LIZ ANDMOLLIE

# Week 5: *Iteration*

1. for loops

2. breaking and skipping

**BREAK**

3. while loops

# Week 5: *Iteration*

1. **for loops**

2. breaking and skipping

**BREAK**

3. while loops

# "Flow Control"

Code that alters the otherwise linear flow of operations in a program.

Last week:

- `if` statements
- `else` statements

This week:

- `for` loops
- `while` loops
- `break` statements
- `next` statements

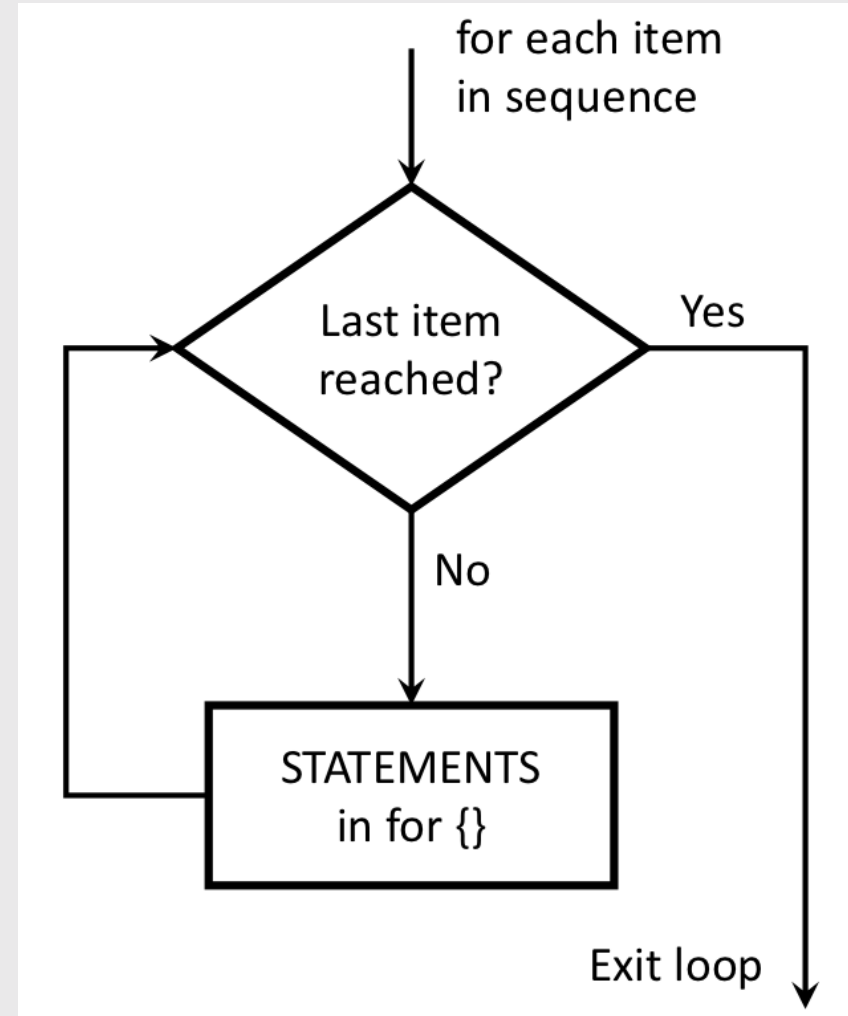


# The **for** loop

Basic format:

```
for (item in sequence) {  
    # Do stuff with item  
    # Loop stops after last item  
}
```

Flow chart:



# Making a sequence

(Side note: these are vectors...that's next week - read ahead!)

Two ways to make a sequence:

1. Use the `seq()` function

```
seq(1, 10)
```

```
#> [1] 1 2 3 4 5 6 7 8 9 10
```

```
seq(1, 10, by = 2)
```

```
#> [1] 1 3 5 7 9
```

2. Use the `:` operator (step size = 1)

```
1:10
```

```
#> [1] 1 2 3 4 5 6 7 8 9 10
```

# Quick code tracing

What will this function print?

```
for (i in 1:5) {  
  if ((i %% 2) == 0) {  
    cat('--')  
  } else if ((i %% 3) == 0) {  
    cat('----')  
  }  
  cat(i, '\n')  
}
```

# Quick code tracing

What will this function print?

```
n <- 6
for (i in seq(n)) {
  cat('|')
  for (j in seq(1, n, 2)) {
    cat('*')
  }
  cat('|', '\n')
}
```

# Your turn

15:00

1) `sumFromMToN(m, n)`: Write a function that sums the total of the integers between `m` and `n`.

**Challenge:** Try solving this without a loop!

- `sumFromMToN(5, 10) == (5 + 6 + 7 + 8 + 9 + 10)`
- `sumFromMToN(1, 1) == 1`

2) `sumOfOddsFromMToN(m, n)`: Write a function that sums every *odd* integer between `m` and `n`.

- `sumOfOddsFromMToN(4, 10) == (5 + 7 + 9)`
- `sumOfOddsFromMToN(5, 9) == (5 + 7 + 9)`

# Week 5: *Iteration*

1. for loops

2. breaking and skipping

BREAK

3. while loops



# Breaking out of a loop

Force a loop to stop with `break`

**Note:** `break` doesn't require `()`

```
for (val in 1:5) {  
    if (val == 3) {  
        break  
    }  
    cat(val, '\n')  
}
```

```
1  
2
```

# Quick code tracing

02:00

What will this code print?

```
for (i in 1:3) {  
  cat('|')  
  for (j in 1:5) {  
    if (j == 3) {  
      break  
    }  
    cat('*')  
  }  
  cat('|', '\n')  
}
```



# Skipping iterations

Skip to the next iteration in a loop with `next`

**Note:** `next` doesn't require `()`

```
for (val in 1:5) {  
    if (val == 3) {  
        next  
    }  
    cat(val, '\n')  
}
```

```
1  
2  
4  
5
```

# Quick code tracing

02:00

What will this code print?

```
for (i in 1:3) {  
  cat('|')  
  for (j in 1:5) {  
    if (j == 3) {  
      next  
    }  
    cat('*')  
  }  
  cat('|', '\n')  
}
```

# Your turn

15:00

`sumOfOddsFromMToNMax(m, n, max)`: Write a function that sums every *odd* integer from `m` to `n` until the sum is less than the value `max`. Your solution should use both `break` and `next` statements.

- `sumOfOddsFromMToNMax(1, 5, 4) == (1 + 3)`
- `sumOfOddsFromMToNMax(1, 5, 3) == (1)`
- `sumOfOddsFromMToNMax(1, 5, 10) == (1 + 3 + 5)`

*Break*

05 : 00

# Week 5: *Iteration*

1. for loops

2. breaking and skipping

BREAK

3. while loops

## Lame joke time:

A friend calls her programmer roommate and said, "while you're out, buy some milk"...

...and she never returned home.

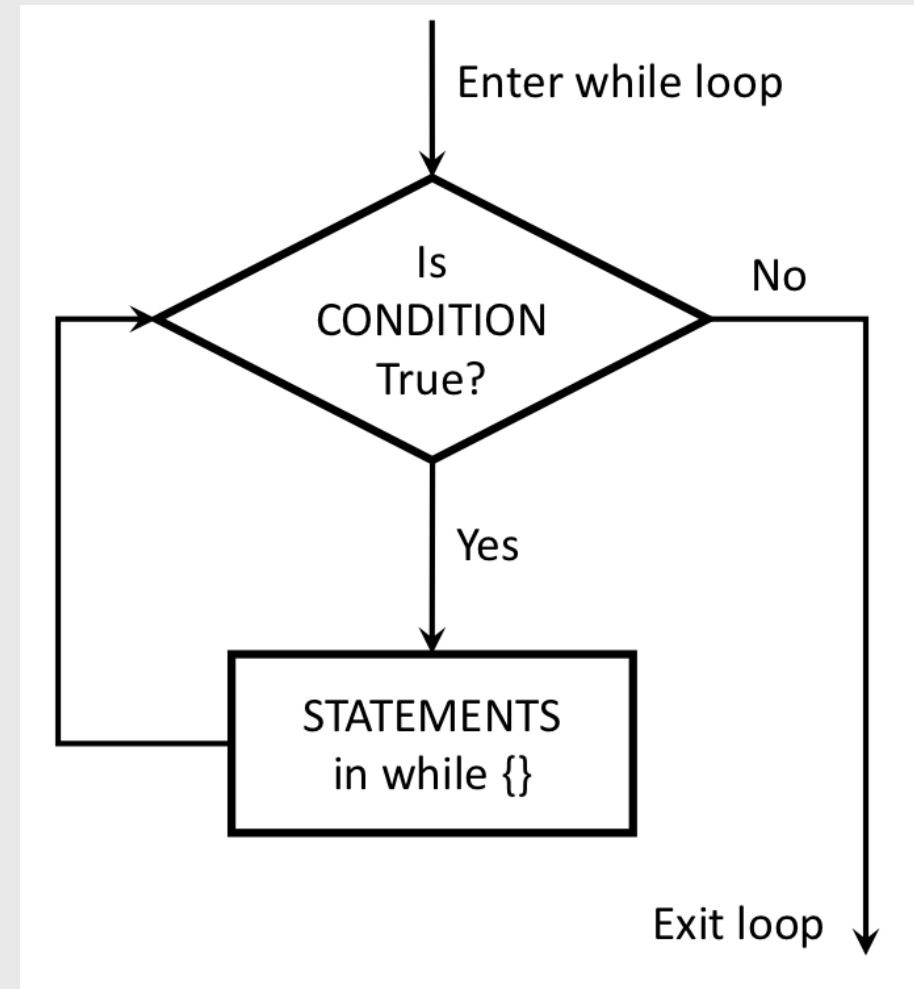


# The **while** loop

Basic format:

```
while (CONDITION) {  
    # Do stuff here  
  
    # Update condition  
}
```

Here's the general idea:



# Quick code tracing

02:00

Consider this function:

```
f <- function(x) {  
  n <- 1  
  while (n < x) {  
    cat(n, '\n')  
    n <- 2*n  
  }  
}
```

What will this code print?

```
f(5)  
f(10)  
f(50)  
f(60)  
f(64)
```

# for vs. while

Use **for** loops when the number of iterations is **known**.

1. Build the sequence
2. Iterate over it

```
for (i in 1:5) { # Define the sequence
  cat(i, '\n')
}
```

```
#> 1
#> 2
#> 3
#> 4
#> 5
```

Use **while** loops when the number of iterations is **unknown**.

1. Define stopping condition
2. Iterate until condition is met

```
i <- 1
while (i <= 5) { # Define stopping
  condition
  cat(i, '\n')
  i <- i + 1 # Update condition
}
```

```
#> 1
#> 2
#> 3
#> 4
```



# Your turn: Write functions

15:00

## 1) `isMultipleOf4Or7(n)`

Write a function that returns **TRUE** if `n` is a multiple of 4 or 7 and **FALSE** otherwise.

- `isMultipleOf4Or7(0) == FALSE`
- `isMultipleOf4Or7(1) == FALSE`
- `isMultipleOf4Or7(4) == TRUE`
- `isMultipleOf4Or7(7) == TRUE`
- `isMultipleOf4Or7(28) == TRUE`

## 2) `nthMultipleOf4Or7(n)`

Write a function that returns the `nth` positive integer that is a multiple of either 4 or 7.

- `nthMultipleOf4Or7(1) == 4`
- `nthMultipleOf4Or7(2) == 7`
- `nthMultipleOf4Or7(3) == 8`
- `nthMultipleOf4Or7(4) == 12`
- `nthMultipleOf4Or7(5) == 14`
- `nthMultipleOf4Or7(6) == 16`

# Your turn

20:00

**isPrime(n)**: Write a function that takes a non-negative integer, **n**, and returns **TRUE** if it is a prime number and **FALSE** otherwise. Here's some test cases:

- **isPrime(1) == FALSE**
- **isPrime(2) == TRUE**
- **isPrime(7) == TRUE**
- **isPrime(13) == TRUE**
- **isPrime(14) == FALSE**

**nthPrime(n)**: Write a function that takes a non-negative integer, **n**, and returns the **nth** prime number, where **nthPrime(1)** returns the first prime number (2). Hint: use the function **isPrime(n)** as a helper function!

- **nthPrime(1) == 2**
- **nthPrime(2) == 3**
- **nthPrime(3) == 5**
- **nthPrime(4) == 7**
- **nthPrime(7) == 17**

## HW 5

- Trickier turtles
- Read about [Happy Numbers](#)