# Week 6: Vectors

## EMSE 6574, Section 11

John Helveston
September 30, 2019

# Announcements

2nd tutor session on Sat. this week (not Fri.)

Quiz 3 next week, Exam 1 week after that

Quiz 2 review

# Iteration review: loop over a sequence

```
seq(5)
```

```
## [1] 1 2 3 4 5
```

## `for` loop

```
for (i in seq(5)) {
    cat(i, '\n')
}
```

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

## `while` loop

```
i <- 1
while (i <= 5) {
    cat(i, '\n')
    i <- i + 1
}
```

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

# Search for something in a sequence

```
seq(5)
```

```
## [1] 1 2 3 4 5
```

Count of **even** numbers in sequence:

## `for` loop

```
count <- 0
for (i in seq(5)) {
    if (i %% 2 == 0) {
        count <- count + 1
    }
}
count
```

```
## [1] 2
```

## `while` loop

```
count <- 0
i <- 1
while (i <= 5) {
    if (i %% 2 == 0) {
        count <- count + 1
    }
    i <- i + 1
}
count
```

```
## [1] 2
```

# This week: Vectors!

Note: We've been dealing with vectors all along!

```
x <- 1
x
```

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

```
is.vector(x)
```

```
## [1] TRUE
```

```
length(x)
```

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

# The "concatenate" function

The universal vector generator: `c()`

```
# Numeric vectors
x <- c(1, 2, 3)
x
```

```
## [1] 1 2 3
```

```
# Character vectors
y <- c('one', 'two', 'three')
y
```

```
## [1] "one"   "two"   "three"
```

```
# Logical vectors
y <- c(TRUE, FALSE, TRUE)
y
```

```
## [1]  TRUE FALSE  TRUE
```

# Other ways to make a vector

1) Sequences (we saw these last week):

```
seq(1, 5)
```

```
## [1] 1 2 3 4 5
```

```
1:5
```

```
## [1] 1 2 3 4 5
```

2) Repeating a value:

```
rep(5, 10)
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5
```

```
rep("snarf", 5)
```

```
## [1] "snarf" "snarf" "snarf" "snarf" "snarf"
```

# Repeating a vector

```
x <- rep(c("foo", "snarf"), 3)
x
```

```
## [1] "foo"   "snarf" "foo"   "snarf" "foo"   "snarf"
```

```
length(x)
```

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

Note the difference when adding the `each` argument:

```
x <- rep(c("foo", "snarf"), each = 3)
x
```

```
## [1] "foo"   "foo"   "foo"   "snarf" "snarf" "snarf"
```

```
length(x)
```

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

# Elements in vectors must be the same type

If a vector has a *single* character element, R makes everything a **character**:

```
c(1, 2, "3")
```

```
## [1] "1" "2" "3"
```

```
c(TRUE, FALSE, "TRUE")
```

```
## [1] "TRUE"  "FALSE" "TRUE"
```

If a vector has numeric and logical elements, R makes everything a **number**:

```
c(1, 2, TRUE, FALSE)
```

```
## [1] 1 2 1 0
```

If a vector has integers and floats, R makes everything a **float**:

```
c(1L, 2, pi)
```

```
## [1] 1.000000 2.000000 3.141593
```

# Doing math on numeric vectors

```
x <- 1:10
x
```

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

```
x + 5
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15
```

```
x / 2
```

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
x + x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x - x
```

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

# Practice: Think-Pair-Share

3 minutes - no typing!

```
f1 <- function(x) {
    m = x
    n = x + 4
    m = m + 5
    return(c(m, n))
}
```

```
f2 <- function(x) {
    return(c(x, x / 2))
}
```

What will this return?

```
x <- c(1, 2, 3)
y <- c(TRUE, FALSE, 7)
f1(x)
f2(y)
```

```
f1(x)
```

```
## [1] 6 7 8 5 6 7
```

```
f2(y)
```

```
## [1] 1.0 0.0 7.0 0.5 0.0 3.5
```

# Using vectors instead of a loop: Summation

```
x <- 1:10
x
```

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

**Using a loop**:

```
total <- 0
for (item in x) {
    total <- total + item
}
total
```

```
## [1] 55
```

**Using the `sum()` function**:

```
sum(x)
```

```
## [1] 55
```

# Other useful "summary" functions

```
x
```

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

```
length(x)
```

```
## [1] 10
```

```
prod(x)
```

```
## [1] 3628800
```

```
mean(x)
```

```
## [1] 5.5
```

```
median(x)
```

```
## [1] 5.5
```

```
max(x)
```

```
## [1] 10
```

```
min(x)
```

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

# Using vectors instead of a loop

Example: Get the remainder for each value in a vector:

```
x <- c(3.1415, 1.618, 2.718)
x
```

```
## [1] 3.1415 1.6180 2.7180
```

**Using a loop**:

```
remainder <- c()
for (i in x) {
    remainder <- c(remainder, i %% 1)
}
remainder
```

```
## [1] 0.1415 0.6180 0.7180
```

**Using a vector**:

```
remainder <- x %% 1
remainder
```

```
## [1] 0.1415 0.6180 0.7180
```

# Get elements in a vector: Use brackets [ ]

```
x <- seq(1, 10)
x
```

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

```
x[1] # Returns the first element
```

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

```
x[3] # Returns the third element
```

```
## [1] 3
```

```
x[1:3]  # Returns the first three elements
```

```
## [1] 1 2 3
```

```
x[c(2, 7)] # Returns the 2nd and 7th elements
```

```
## [1] 2 7
```

```
x[length(x)] # Returns the last element
```

```
## [1] 10
```

# Use negative integers to *remove* elements

```
x <- seq(1, 10)
x
```

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

```
x[-1] # Returns everything except the first element
```

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

```
x[-1:-3]  # Returns everything except the first three elements
```

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

```
x[-c(2, 7)] # Returns everything except the 2nd and 7th elements
```

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

```
x[-length(x)] # Returns everything except the last elements
```

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

# You can name vector elements

```
x <- seq(5)
x
```

```
## [1] 1 2 3 4 5
```

1) Use the `names()` function:

```
names(x) <- c('a', 'b', 'c', 'd', 'e')
x
```

```
## a b c d e
## 1 2 3 4 5
```

2) Use the `c()` function:

```
y <- c('a' = 1, 'b' = 2, 'c' = 3, 'd' = 4, 'e' = 5)
y
```

```
## a b c d e
## 1 2 3 4 5
```

# You can use names to extract elements in a

```
x
```

```
## a b c d e
## 1 2 3 4 5
```

```
x['a']
```

```
## a
## 1
```

```
x[c('a', 'c')]
```

```
## a c
## 1 3
```

# Logical indices

```
x <- seq(10, 1)
x
```

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

Create a logical vector:

```
x > 5
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Use logical vector in brackets to filter elements:

```
x[x > 5]
```

```
## [1] 10  9  8  7  6
```

# Practice: Think-Pair-Share

4 minutes - no typing!

```
f <- function(x) {
    for (i in seq(length(x))) {
        x[i] <- x[i] + sum(x) + max(x)
    }
    return(x)
}
x <- c(1, 2, 3)
```

What will this return?

```
f(x)
```

```
## [1] 10 27 70
```

# Comparing vectors

Check if 2 vectors are the same:

```
x <- c(1, 2, 3)
y <- c(1, 2, 3)
```

```
x == y
```

```
## [1] TRUE TRUE TRUE
```

To check if *all* elements are the same, use `all()`:

```
all(x == y)
```

```
## [1] TRUE
```

# Comparing vectors

To check if *any* elements are the same, use `any()`:

```
a <- c(1, 2, 3)
b <- c(-1, 2,-3)
a == b
```

```
## [1] FALSE  TRUE FALSE
```

```
all(a == b)
```

```
## [1] FALSE
```

```
any(a == b)
```

```
## [1] TRUE
```

# `all()` vs. `identical()`

```
x <- c(1, 2, 3)
y <- c(1, 2, 3)
names(x) <- c('a', 'b', 'c')
names(y) <- c('one', 'two', 'three')
```

```
all(x == y) # Only compares the elements
```

```
## [1] TRUE
```

```
identical(x, y) # Also compares the **names** of the elements
```

```
## [1] FALSE
```

```
names(y) <- c('a', 'b', 'c')
identical(x, y)
```

```
## [1] TRUE
```

# Practice: Think-Pair-Share

10 minutes

Re-write `isPrime(n)` from last week, but **without loops!**

Remember, `isPrime(n)` 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`

Loop solution:

```
isPrime <- function(n) {
    if (n <= 1) { return(FALSE) }
    if (n == 2) { return(TRUE) }
    for (i in seq(2, n-1)) {
        if (n %% i == 0) {
            return(FALSE)
        }
    }
    return(TRUE)
}
```

# 5 minute break - stand up, move around,

5 minutes

# Vectorized operations

```
x1 <- c(1, 2, 3)
x2 <- c(4, 5, 6)
```

```
# Addition
x1 + x2 # Returns (1+4, 2+5, 3+6)
```

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

```
# Subtraction
x1 - x2 # Returns (1-4, 2-5, 3-6)
```

```
## [1] -3 -3 -3
```

```
# Multiplicattion
x1 * x2 # Returns (1*4, 2*5, 3*6)
```

```
## [1]  4 10 18
```

```
# Division
x1 / x2 # Returns (1/4, 2/5, 3/6)
```

```
## [1] 0.25 0.40 0.50
```

# Vectorized operations

## Be careful to match dimensions!

```
x1 <- c(1, 2, 3, 4)
x2 <- c(4, 5)
```

```
x1 + x2
```

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

Single-value vector operations

```
x1 <- c(1, 2, 3)
x2 <- c(4)
```

```
x1 + x2
```

```
## [1] 5 6 7
```

# Sorting

```
a = c(2, 4, 6, 3, 1, 5)
a
```

```
## [1] 2 4 6 3 1 5
```

```
sort(a)
```

```
## [1] 1 2 3 4 5 6
```

```
sort(a, decreasing = TRUE)
```

```
## [1] 6 5 4 3 2 1
```

Use `order()` to get the index values of the sorted order:

```
order(a)
```

```
## [1] 5 1 4 2 6 3
```

This does the same thing as `sort(a)`:

```
a[order(a)]
```

```
## [1] 1 2 3 4 5 6
```

# Group practice - no loops! 20 minutes

1) `reverse(x)`

Write a function that returns the vector in reverse order. You cannot use the `rev()` function. Test cases:

- `reverse(c(5, 1, 3)) == c(3, 1, 5)`
- `reverse(c('a', 'b', 'c')) == c('c', 'b', 'a')`
- `reverse(c(FALSE, TRUE, TRUE)) == c(TRUE, TRUE, FALSE)`
- `reverse(seq(10)) == seq(10, 1, -1)`

2) `middleValue(a)`

Write a function that takes a vector of numbers `a` and returns the value of the middle element (or the average of the two middle elements). Test cases:

- `middleValue(c(0,0,0)) == 0`
- `middleValue(c(1,2,3)) == 2`
- `middleValue(c(4,5,6,7,8)) == 6`
- `middleValue(c(5,3,8,4)) == mean(c(3,8))`
- `middleValue(c(4,5,6,7)) == mean(c(5,6))`

# Group practice - no loops! 20 minutes

1) `dotProduct(a, b)`

The "dot product" of two vectors is the sum of the products of the corresponding terms. So the dot product of the vectors `c(1,2,3)` and `c(4,5,6)` is `(1*4) + (2*5) + (3*6)`, or `4 + 10 + 18 = 32`. Write a function that takes two vectors and returns the dot product of those vectors. If the vectors are not equal length, ignore the extra elements in the longer vector. Test cases:

- `dotProduct(c(1,2,3), c(4,5,6)) == 32`
- `dotProduct(c(1,2), c(4,5,6)) == 14`
- `dotProduct(c(2,3,4), c(-7,1,9)) == 25`
- `dotProduct(c(0,0,0), c(-7,1,9)) == 0`

2) `alternatingSum(a)`

Write a function that takes a vector of numbers `a` and returns the alternating sum, where the sign alternates from positive to negative or vice versa. Test cases:

- `alternatingSum(c(5,3,8,4)) == (5 - 3 + 8 - 4)`
- `alternatingSum(c(1,2,3)) == (1 - 2 + 3)`
- `alternatingSum(c(0,0,0)) == 0`
- `alternatingSum(c(-7,5,3)) == (-7 - 5 + 3)`