Week 6: *Vectors*

EMSE 4574: Intro to Programming for Analytics

John Paul Helveston

October 06, 2020

Week 6: *Vectors*

1. Making vectors

2. Vector operations

3. Slicing vectors

4. Comparing vectors

5. Lists

Week 6: *Vectors*

1. Making vectors

2. Vector operations

3. Slicing vectors

4. Comparing vectors

5. Lists

Note: We've been using vectors already!

```
x <- 1
x
```

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

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

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

```
length(x)
```

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

The universal vector generator: c()

Numeric vectors

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

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

Character vectors

```r
y <- c('one', 'two', 'three')
y
```

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

Logical vectors

```r
z <- c(TRUE, FALSE, TRUE)
z
```

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

Elements in vectors must be the same type

Type hierarchy:

- `character` > `numeric` > `logical`

- `double` > `integer`

Coverts to characters:

```
c(1, "foo", TRUE)
```

```
## [1] "1"    "foo"  "TRUE"
```

Coverts to numbers:

```
c(7, TRUE, FALSE)
```

```
## [1] 7 1 0
```

Coverts to double:

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

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

Other ways to make a vector

Sequences (we saw these last week):

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

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

```
1:5
```

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

Repeating a value:

```
rep(5, 3)
```

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

```
rep("snarf", 3)
```

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

Repeating a vector

Repeating a sequence

```
x <- rep(seq(1, 3), 3)
x
```

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

```
length(x)
```

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

Note what the each argument does:

```
x <- rep(seq(1, 3), each = 3)
x
```

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

```
length(x)
```

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

Quick code tracing

What will each of these lines produce?

```
rep(c(TRUE, FALSE, "TRUE"), 2)
```

```
seq(FALSE, 3)
```

```
rep(c(seq(3), seq(2)), each = 2)
```

Week 6: *Vectors*

1. Making vectors

2. Vector operations

3. Slicing vectors

4. Comparing vectors

5. Lists

Math on vectors is done **by element**

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

```
x + 2
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12
```

```
x - 2
```

```
##  [1] -1  0  1  2  3  4  5  6  7  8
```

```
x * 2
```

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

```
x / 2
```

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

Math on vectors is done **by element**

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

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

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

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

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

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

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

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

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

If dimensions don't match, R "wraps" the vector

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

```r
x1 + x2
```

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

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

```r
x1 + x2
```

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

Most R functions work on vectors

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

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

```r
round(x, 2)
```

```
## [1] 3.14 1.62 2.72
```

```r
sqrt(x)
```

```
## [1] 1.772428 1.272006 1.648636
```

Works with your own functions too:

```r
isEven <- function(x) {
    return((x %% 2) == 0)
}
```

```r
x <- c(1, 4, 5, 10)
isEven(x)
```

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

Using vectors instead of a loop: **Summation**

Example: Sum the integers from 1 to 10

Summing with a loop:

```r
x <- seq(1, 10)
total <- 0
for (item in x) {
    total <- total + item
}
total
```

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

Use a *summary function* on the vector:

```r
sum(x)
```

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

"Summary" functions take a vector and return one value

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

```
length(x)
```

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

```
sum(x)
```

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

```
prod(x)
```

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

```
min(x)
```

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

```
max(x)
```

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

```
mean(x)
```

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

```
median(x)
```

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

Quick code tracing

Consider this function:

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

What will this return?

```
x <- c(1, 3)
f(x)
```

```
y <- c(TRUE, FALSE, 1)
f(y)
```

Break

05:00

Week 6: *Vectors*

1. Making vectors

2. Vector operations

3. Slicing vectors

4. Comparing vectors

5. Lists

Use brackets [ ] to get elements from a vector

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

Indices start at 1:

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

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

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

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

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

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

Slicing with a vector of indices:

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

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

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

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

Use negative integers to *remove* elements

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

```r
x[-1] # Drops the first element
```

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

```r
x[-1:-3] # Drops the first three elements
```

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

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

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

```r
x[-length(x)] # Drops the last element
```

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

Slicing with logical indices

```
x <- seq(1, 20, 3)
x
```

```
## [1]  1  4  7 10 13 16 19
```

Create a logical vector based on some condition:

```
x > 10
```

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

Slice x with logical vector - only TRUE indices will be returned:

```
x[x > 10]
```

```
## [1] 13 16 19
```

You can also use `which()` to find indices

```
x <- seq(1, 20, 3)
x
```

```
## [1]  1  4  7 10 13 16 19
```

Use `which()` around a condition to get the indices where condition is TRUE:

```
which(x > 10)
```

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

```
x[which(x > 10)]
```

```
## [1] 13 16 19
```

You can name vector elements

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

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

1) Add names with the `names()` function:

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

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

2) Create a named vector:

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

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

Using names to slice a vector

```
x
```

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

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

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

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

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

Sorting vectors with `sort()`

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

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

```r
sort(a)
```

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

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

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

`order()` returns the indices of the sorted vector

```
a
```

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

```
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
```

Quick code tracing

Consider this function:

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

What will this code return?

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

Think-Pair-Share

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

- `reverse(c(5, 1, 3))` should return `c(3, 1, 5)`

- `reverse(c('a', 'b', 'c'))` should return `c('c', 'b', 'a')`

- `reverse(c(FALSE, TRUE, TRUE))` should return `c(TRUE, TRUE, FALSE)`

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.

- `alternatingSum(c(5,3,8,4))` should return `(5 − 3 + 8 − 4)`

- `alternatingSum(c(1,2,3))` should return `(1 − 2 + 3)`

- `alternatingSum(c(0,0,0))` should return `0`

- `alternatingSum(c(−7,5,3))` should return `(−7 − 5 + 3)`

Week 6: *Vectors*

1. Making vectors

2. Vector operations

3. Slicing vectors

4. Comparing vectors

5. Lists

Comparing vectors

Check if 2 vectors are the same:

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

```r
x == y
```

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

Comparing vectors with all() and any()

all(): Check if *all* elements are the same

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

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

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

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

any(): Check if *any* elements are the same

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

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

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

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

all() vs. identical()

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

all() only compares the element *values*:

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

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

identical() compares *values* and *names*:

```r
identical(x, y)
```

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

```r
names(y) <- names(x)
identical(x, y)
```

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

Think-Pair-Share

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`

(If you're stuck, go to the next slide for a hint)

Hint

Loop solution:

```r
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)
}
```

Week 6: *Vectors*

1. Making vectors

2. Vector operations

3. Slicing vectors

4. Comparing vectors

5. Lists

Lists are like vectors that can store anything

Vectors force things to one type:

```
c(1, "foo", TRUE)
```

```
## [1] "1"      "foo"   "TRUE"
```

Lists store any type:

```
list(1, "foo", TRUE)
```

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

Elements in lists can be any object

List of vectors:

```
list(c(1, 2, 3), c("foo", "bar"), TRUE)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "foo" "bar"
##
## [[3]]
## [1] TRUE
```

List of vector and function:

```
vector <- c(1, 2, 3)
square <- function(x) {
    return(x^2)
}

list(vector, square)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## function(x) {
##     return(x^2)
## }
## <environment: 0x7fd430049040>
```

Slice list with indices or names

Slice with index using `[[]]`

```r
x <- list(
  c(1, 2, 3),
  c("foo", "bar"),
  TRUE)
```

```r
x[[1]]
```

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

```r
x[[2]]
```

```
## [1] "foo" "bar"
```

Slice with name using `` `[[]] `` or `$`

```r
x <- list(
  numbers = c(1, 2, 3),
  chars   = c("foo", "bar"),
  logical = TRUE)
```

```r
x[['numbers']]
```

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

```r
x$numbers
```

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

HW 6

- This is the last HW due before before the midterm (check the schedule).

- Next week is Quiz 4 - also the last quiz before the midterm.

- Midterm is during class period on Oct. 20.