Functions and Vectors

INFO 201

Joel Ross

Winter 2017

https://slides.com/joelross/info201w17-vectors/live

Today's Objectives

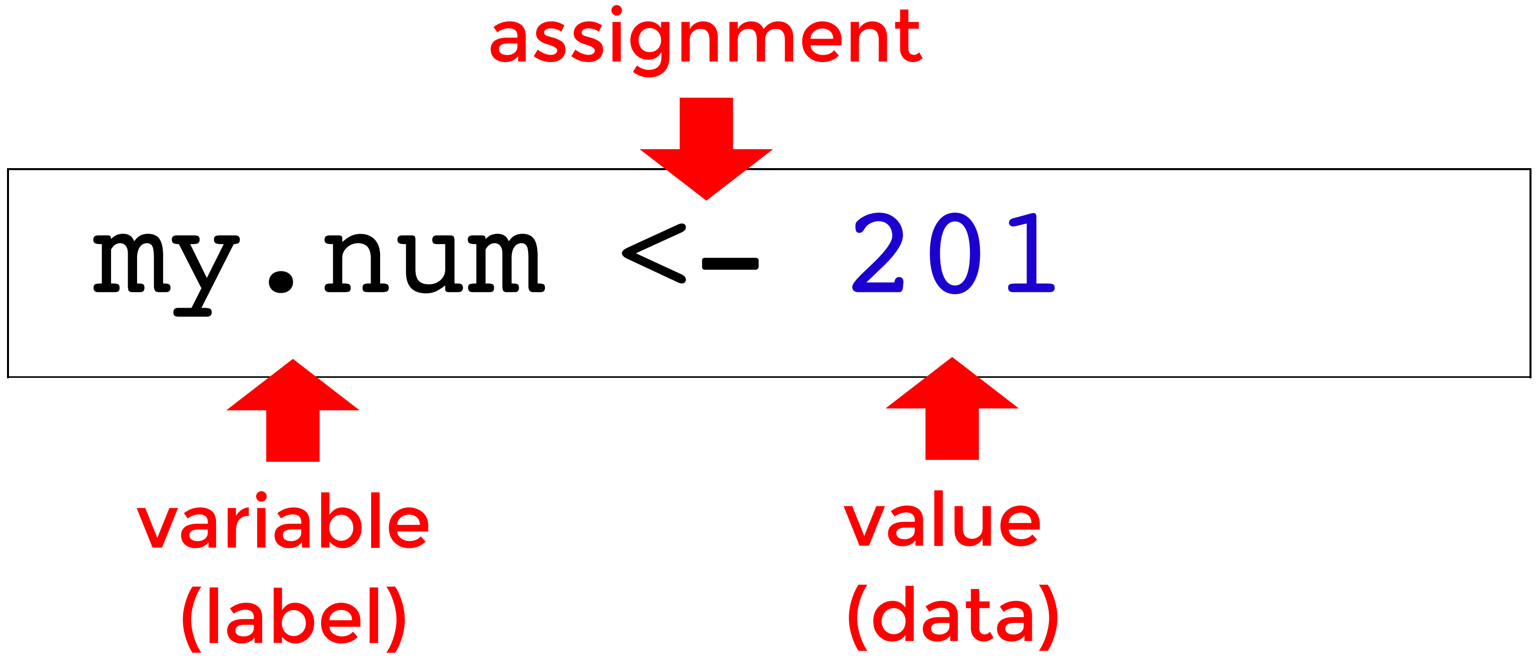
By the end of class, you should be able to

- Confidently use **variables** and **functions**.
- Define and call your own functions
- Store and manipulate data in **vectors**
- Utilize **functions** to manipulate data

RECALL

Variables

A label that refers to a **value** (data)



```
my.num <- 201
```

RECALL

Functions

A named sequence of instructions (lines of code). We **call** a function to do those steps.

```
upcase <- toupper("Hello world")
```

returned result

function name

argument (input value)

*Functions **abstract** computer programs!*

Module 6 exercise-4

New exercise; just copy and paste the code into a new file in your repo!

Loading Functions

We can download and *load* **packages** (a.k.a. "**libraries**") of additional functions to call.

```r
# Install `stringr` package (for string funcs)
# Only needs to be done once per machine!
install.packages("stringr")

# Load the package (tell R funcs available for use)
library("stringr") # quotes optional here

sentence <- "The quick brown fox jumped over the lazy dog"

# Use loaded `word()` function
# to get words 2 through 4 of sentence
word(sentence, 2, 4)  # "quick brown fox"
```

https://www.rdocumentation.org/packages/stringr/versions/1.1.0

Writing Functions

By writing our own functions we can:

- Easily reuse algorithms (write less code!)
- Debug one piece of a program at a time
- Abstract an algorithm to focus on the bigger picture

Defining a Function

optional, comma-separated

```
# general syntax
FunctionName <- function(name) {
    # body: instructions (code) go here
}
```

block

CamelCase, without periods!

```
# A function that says hello to someone
SayHello <- function(name) {
    greeting <- paste("Hello", name)
    print(greeting)
}

SayHello("Joel")
```

Function Arguments

Arguments are **variables** (labels) that are assigned values when the function is called.

```
SayHello <- function(name) {
   greeting <- paste("Hello", name)
   print(greeting)
}
                       name <- "Joel" #implicit

SayHello("Joel")
```

Scope

Variables created inside a function (including the arguments) are **local variables**, and so are only available **inside** the function.

```
MakeFullName <- function(first.name, last.name) {
   full.name <- paste(first.name, last.name)
}

MakeFullName("Joel", "Ross")
print(full.name)  #Error! variable not found
```

Read the error message!

Return Values

Functions can **return** a single value as a result. This is different than printing an output.

```
MakeFullName <- function(first.name, last.name) {
   full.name <- paste(first.name, last.name)

   return(full.name)
}


my.full.name <- MakeFullName("Joel","Ross")
```

func to "return" value
This exits our function.

Remember to give the result
a label to use it later!

Module 6 exercise-1

Module 6 exercise-2

Vectors

Vectors

Vectors are *one-dimensional collections* of values that are all stored in a single variable. For example, a "set" of words (strings) or numbers. **Elements** must all be of the same type.

```
# combine 3 dog names into a vector
dogs <- c("Fido", "Spot", "Sparky")

# create a vector of numbers
numbers <- c(1,2,2,3,5,8,13,21,34)  # Fibonacci!



# a vector of the whole numbers from 90 to 100
nineties <- 90:99  # 90  91  92  ...  99
```

combine into vector with c()

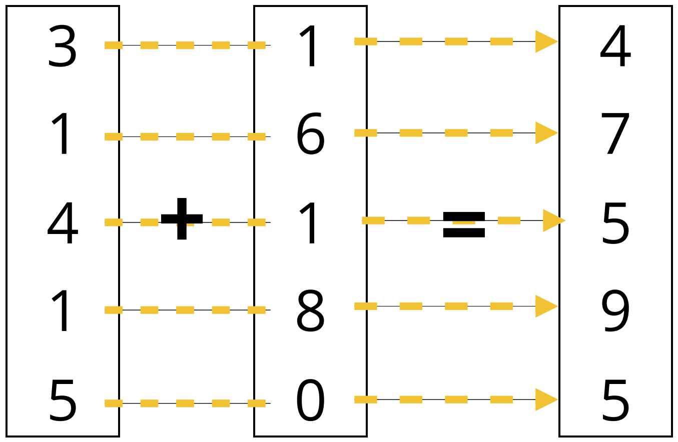
create sequences with : operator

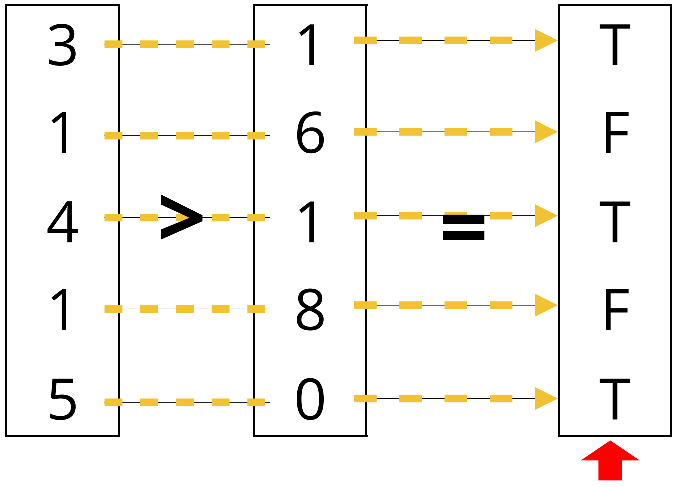
Vectorized Operations

We can use mathematical (**+**, **-**, *****, **/**) or relational (**<**, **>**, **==**) operators on vectors. These operations are applied **member-wise** (1st with 1st, 2nd with 2nd, etc)

```
v1 <- c(3, 1, 4, 1, 5)
v2 <- c(1, 6, 1, 8, 0)

v3 <- v1 + v2  # Add the vectors
v4 <- v1 > v2  # Compare the vectors
```





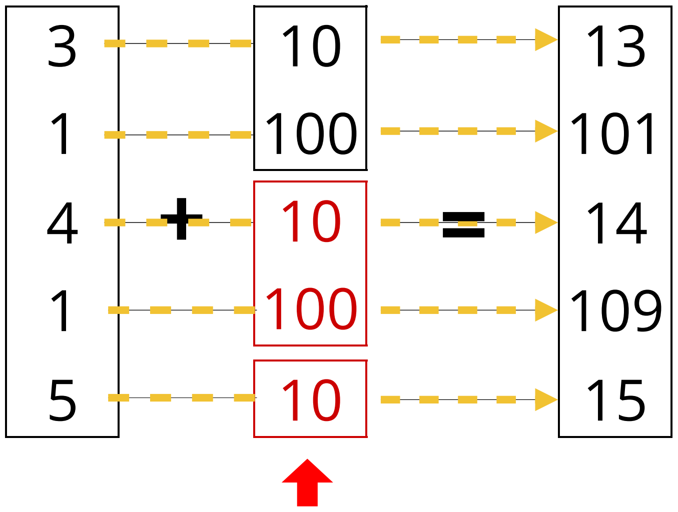
vector of booleans!

Recycling

If one vector is shorter than another, then the elements of the shorter vector are **recycled** (reused):

```
v1 <- c(3, 1, 4, 1, 5)
v2 <- c(10, 100)

v3 <- v1 + v2  # Add the vectors
```



will cause a **warning** if we throw away elements, but still works

Vectors and Scalars?

What do you think will happen if we add a **vector** and a **scalar** (a single number)?

```
# create vector of numbers 0 to 5
v1 <- 0:5  # equivalent to c(0, 1, 2, 3, 4, 5)

result <- v1 + 201  #add scalar to vector
print(result)
```

EVERYTHING IS A VECTOR

Everything is a Vector

Literals (single numbers or values) are really just vectors with a single element in them.

```r
# Create a vector of length 1 in a variable x
x <- 201  # equivalent to `x <- c(201)`

# R states the vector index (1) in the console
print(x)  # [1] 201

identical(201, c(201))  # TRUE
```

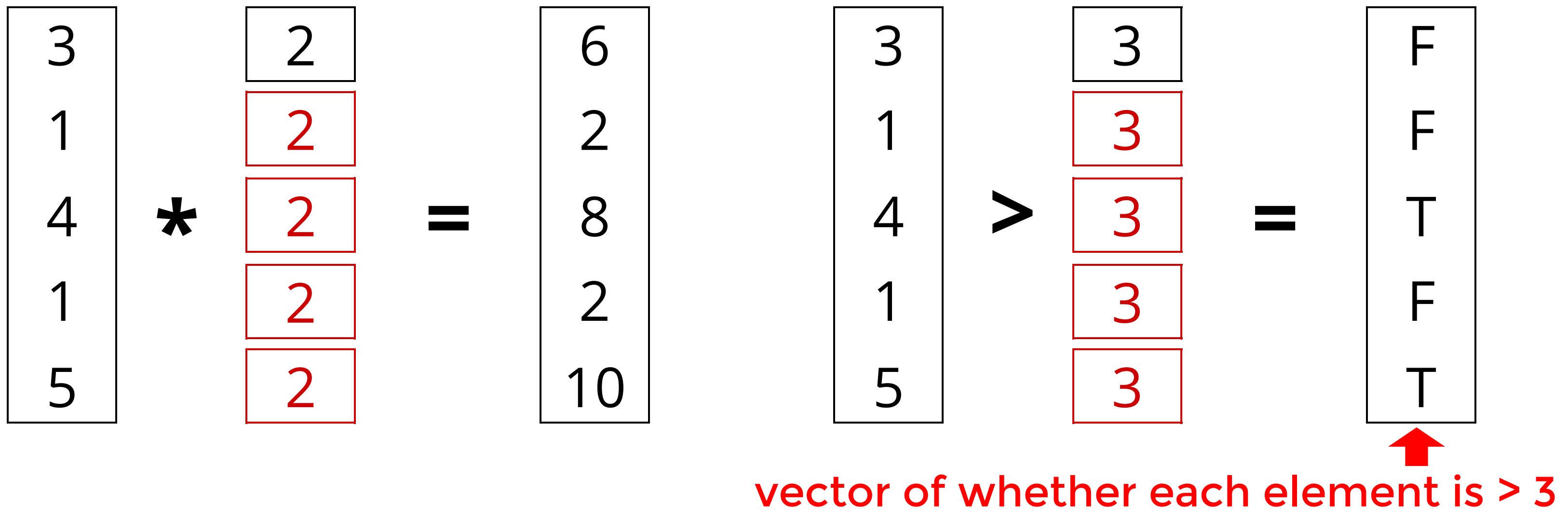
Everything is a Vector

Using "scalars" (single values) as operands causes the **recycling** behavior so that the operand is applied to every element in the vector

```
vec <- c(3, 1, 4, 1, 5)

doubled <- vec * 2  # double the elements, double your fun

above.three <- vec > 3  # compare the elements to 3
```



Vectorized Functions

Most functions work with vectors (as we've been doing!), applying to each individual element.

```
# Create a vector of colors
colors <- c("red", "green", "blue")

# Make uppercase
upper.case <- toupper(letters)  # [1] RED  GREEN  BLUE

# Create a vector of 10 random numbers (uniform)
random <- runif(10)

# Round to 2 decimal places
rounded <- round(random, 2)
```

Vectorized functions is what makes R so efficient with large data sets (*better than loops!*)

Module 7 exercise-4

New exercise; just copy and paste the code into a new file in your repo!

Vector Indices

We can refer to each element in a vector by its **index** (which "position") it is in the set.

```
vowels <- c('a','e','i','o','u')
```

index	
1	a
2	e
3	i
4	o
5	u

Bracket Notatoin

We access individual values in a vector by using **bracket notation**, putting the **index** of the element inside brackets after the vector name (start at index 1).

```
vowels <- c('a','e','i','o','u')

first.vowel <- vowels[1]  # "a"
print(first.vowel)  # [1] "a"

fourth.vowel <- vowels[4]  # "o"
print(fourth.vowel)  # [1] "o"
```

printed index of ***resulting*** vector

```
# Can also use variables inside the brackets
last.index <- length(vowels)  # num elements = length
vowels.last <- vowels[last.index]  # "u"
```

Multiple Indices

We can make our "position vector" have multiple elements to extract a **subset** of elements.

```
# Create a `colors` vector
colors <- c('red', 'green', 'blue', 'yellow', 'purple')

# Vector of indices to extract
indices <- c(1,3,4)

# Retrieve the colors at those indices
extracted <- colors[indices]
print(extracted)  # [1] "red"    "blue"   "yellow"


# Specify the index array anonymously
others <- colors[c(2, 5)]
print(others)  # [1] "green"  "purple"


# Retrieve values in positions 2 through 5
colors[2:5]  # [1] "green"  "blue"   "yellow" "purple"
```

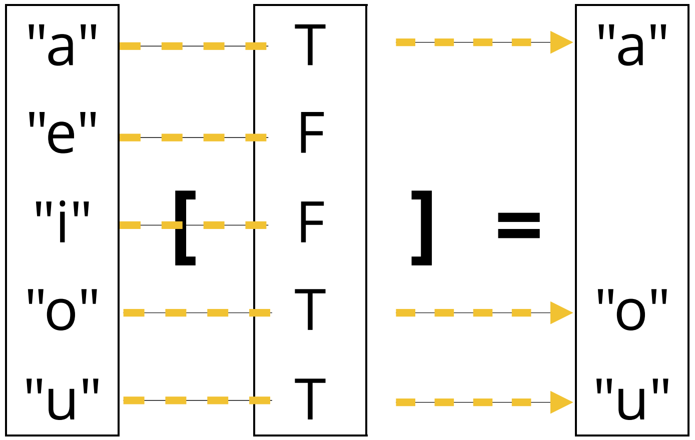
Vector Filtering

We can use a vector of **logical** values (`TRUE`, `FALSE`) inside the brackets instead of a position vector. This will extract every element that corresponds with `TRUE`.

```
vowels <- c('a','e','i','o','u')

# Vector of elements to extract
filter <- c(TRUE, FALSE, FALSE, TRUE, TRUE)

# Extract every element in an index that is TRUE
vowels[filter]  # [1] "a" "o" "u"
```



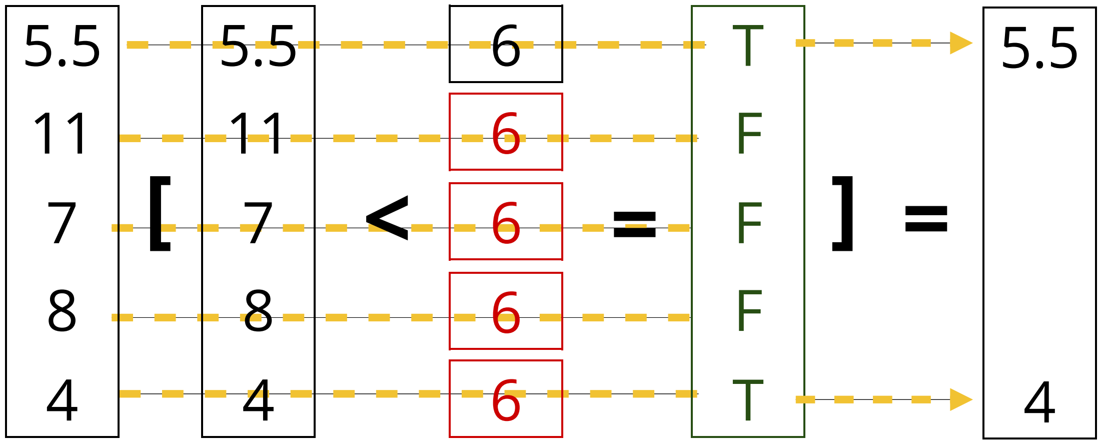
Vector Filtering

When combined with **relational operators** and **recycling**, we can use this approach to **filter** for vector items that meet a criteria!

```
shoe.sizes <- c(5.5, 11, 7, 8, 4)

# A boolean vector that indicates if a shoe size is less than 6
shoe.is.small <- shoe.sizes < 6  # T, F, F, F, T

# Use the `shoe.is.small` vector to select large shoes
small.shoes <- shoe.sizes[shoe.is.small]  # returns 5.5, 4
```



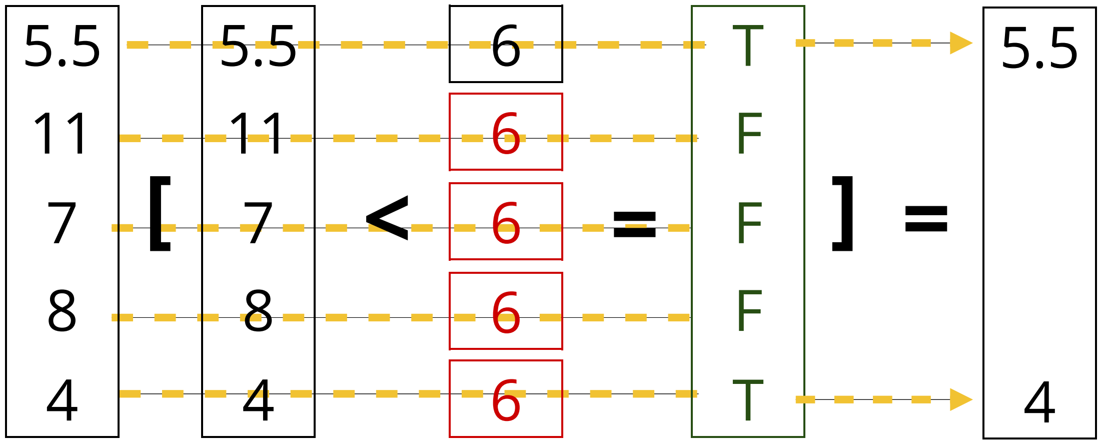
Vector Filtering

When combined with **relational operators** and **recycling**, we can use this approach to **filter** for vector items that meet a criteria!

```
shoe.sizes <- c(5.5, 11, 7, 8, 4)


# Select shoe sizes that are smaller than 6
small.shoes <- shoe.sizes[shoe.sizes < 6]  # returns 5.5, 4
```

combine into one line (anonymous variables)



Module 7 exercise-2

Action Items!

- Be comfortable with **module 0 - 7** by Tues
- Assignment 2 due ***Tuesday before class***

Tuesday: Data frame (pre-read: **modules 8-9**)