Eric Pitman Summer Workshop in Computational Science

4. Writing Functions
The function $f$ generates an output $(Y)$, given an input $(X)$. 
Functions

A piece of code that can be called again and again

To call it, specify:
- Function name
- Input values

It may return an output value
Functions

- We use native R functions all the time! Examples: `class()`, `str()`, `summary()`

- You can also write your own.
Function Syntax in R

functionName = function(inputs) {
    # do something
    # return the result
}

Name of function

Input parameter(s)

Declaration (start of function)

End of function
Function Syntax in R

toFahrenheit = function(celsius) {
  f = (9/5) * celsius + 32; # calculate
  return(f); # return the result
}
Caveats

caveat
/ˈkavət, kævət/ noun

a warning or proviso of specific stipulations, conditions, or limitations.
“there are a number of caveats which concern the validity of the assessment results”
synonyms: warning, caution, admonition, monition, red flag, alarm bells; More
About return()

NotToFahrenheit <- function(celsius) {
  f = (9/5) * celsius + 32;
  g = 97; # R will return the last computed value
}

About return()

NotToFahrenheit <- function(celsius) {
  f = (9/5) * celsius + 32;
  return(f);
  g = 97; # Do we ever execute this line?
}

Accepted inputs?

ToFahrenheit <- function(celsius) {
    f = (9/5) * celsius + 32;
    return(f);
}

Q: Do I need to write a loop to call the function with multiple values?
Accepted inputs?

ToFahrenheit <- function(celsius) {
  f = (9/5) * celsius + 32;
  return(f);
}

Q: Do I need to write a loop to call the function with multiple values?

A: NOPE! These operations accept vectors.
Calling `toFahrenheit()`

celsius = c(20:25);  # define input temperatures

toFahrenheit = function(celsius) {
  f = (9/5) * celsius + 32;  # perform the conversion
  return(f);
}

# call the function to convert temperatures to Fahrenheit:
toFahrenheit(celsius);
[1] 68.0  69.8  71.6  73.4  75.2  77.0
Control Structures
Control Structures: if/else

• Make a logical test
• Perform operations based on the outcome

```plaintext
if (condition is true)
{
    # do something
}
```
Control Structures: if/else

age = 21;

if (age >= 17) {
    print("You can drive!");
}

else if (age >= 16) {
    print("You are almost old enough to drive!");
}

else {
    print("You are not old enough to drive.");
}
If ($age \geq 17$)

- If (age $\geq 17$)
  - Print “You are old enough to drive!”
- Else if (age $\geq 16$)
  - Print “You are almost old enough to drive!”
- Else
  - Print “You are too young to drive.”
Iteration

The repetition of a process or utterance.

- Repetition of a mathematical or computational procedure applied to the result of a previous application, typically as a means of obtaining successively closer approximations to the solution of a problem.
- A new version of a piece of computer hardware or software.

Plural noun: Iterations

Origin:
Late Middle English: from Latin *itertio* (n-), from the verb *iterare* (see *iterate*).
Control Structures: Iteration

- What if we want to call a function over and over?
- Other languages use loops.
- R can do this with a single line of code!
- Use it on native R functions, or functions you wrote yourself.
Definitions

• **Functional languages** consider functions first class citizens.
  – Functions can be assigned to variables, stored in lists, passed as arguments, and returned from calls to other functions.

• **Vectorized operations** execute as precompiled C code, hiding their loops. They are fast—even on vectors.
Meet the apply() family

- by(): with factors
- apply(): returns vector
- sapply(): returns vector or matrix
- lapply()
- mapply()
- rapply()
- ...
- ...
apply(X, MARGIN, FUN,...)

Returns a vector obtained by applying a function to margins of a matrix:
Control Structures: by()

- What if we want to call a function several times, on several *groups* of data?
- We can use a single line of R code:

  \[
  \text{by}(\text{data, group, function})
  \]
Group and operate: by()

by(data-to-operate-on, factor-to-group-by, function)

<table>
<thead>
<tr>
<th>data</th>
<th>function(gp1)</th>
<th>res1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

function(gp2)

res2
Group and operate: `by()`

```
> by(data$Height, data$Hand, max)
```

Vector to operate on

Factor for grouping

Function

<table>
<thead>
<tr>
<th>Height</th>
<th>Weight</th>
<th>Age</th>
<th>Hand</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>120</td>
<td>16</td>
<td>L</td>
</tr>
<tr>
<td>75</td>
<td>160</td>
<td>17</td>
<td>R</td>
</tr>
<tr>
<td>60</td>
<td>118</td>
<td>16</td>
<td>R</td>
</tr>
</tbody>
</table>
Group and operate: `by()`

```r
> by(data$Height, data$Hand, max)
  L  R
68  75
```

<table>
<thead>
<tr>
<th>Height</th>
<th>Weight</th>
<th>Age</th>
<th>Hand</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>120</td>
<td>16</td>
<td>L</td>
</tr>
<tr>
<td>75</td>
<td>160</td>
<td>17</td>
<td>R</td>
</tr>
<tr>
<td>60</td>
<td>118</td>
<td>16</td>
<td>R</td>
</tr>
</tbody>
</table>
Compute summaries and means of data, grouping by Species:

```R
<workshop>/examples/by-example.R
```
iris and by()

```r
> by(iris[,c(1:3)], Species, colMeans)

Species: setosa
Sepal.Length  Sepal.Width  Petal.Length
  5.006        3.428        1.462

Species: versicolor
Sepal.Length  Sepal.Width  Petal.Length
  5.936        2.770        4.260

Species: virginica
Sepal.Length  Sepal.Width  Petal.Length
  6.588        2.974        5.552
```
Discriminate and operate: `sapply()`

```r
> lis = list(37, "a", NA, 4.78)
> sapply(lis, class)
```

List to operate on: `lis`
Function: `class(lis)`

<table>
<thead>
<tr>
<th>lis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>NA</td>
</tr>
<tr>
<td>4.78</td>
</tr>
</tbody>
</table>
Discriminate and operate: `sapply()`

```r
> lis = list(37, "a", NA, 4.78)
> sapply(lis, class)

"integer" "character" "logical" "numeric"
```
Tips: Writing Functions

- Use an editor window (not the command line) to compose functions
- Try out one line at a time, and test!
- Start with the simplest case and build.
- Comment your function to indicate:
  - input
  - output
  - purpose
Student Dataset Example

Remember our own dataset:

firstInitial, lastInitial, school, height, htUnit, age, handed, gender

Let's write functions that:

- Convert heights to a uniform unit
- List initials of students that are old enough to drive
Interlude

Complete function exercises.

Open in the RStudio source editor:

<workshop>/exercises/4-exercises-functions.R
Interlude++

Function reading assignment:

“How to write and debug an R function”:

https://vidia.ccr.buffalo.edu/resources/686