Eric Pitman Summer Workshop in Computational Science

4. Writing Functions
Functions

A function generates an output (Y), given an input (X).
Control Structures: if/else

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

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

```python
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 >= 17)

Print “You are old enough to drive!”

else if (age >= 16)

Print “You are almost old enough to drive!”

else

Print “You are too young to drive.”
A function $f$ takes an input, $x$, and returns an output $f(x)$.

It's like a machine that converts an input into an output.
Functions

Function: 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
```r
functionName = function(inputs) {
    # do something
    # return the result
}
```
Functions in R

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

Name of function

Input parameter(s)

Declaration (start of function)

Output value

End of function
Functions in R

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


Functions in R

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 for Iteration

- In other languages we write loops
- But R is a (functional, vector) language
- We can operate on multiple data subsets with one line of code!

apply()
by()
Control Structures: apply() Family

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

\[ \text{apply(vector, function) \)]
Control Structures: `sapply()`

```r
> lis = c("a", "b", "c", "d")
> sapply(lis, class)

  a                     b                     c                     d
"character" "character" "character" "character"
```

<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>
</tbody>
</table>
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:

\[
\text{by(data, group, function)}
\]
Control Structures: by()

by(data-to-operate-on, data-to-group-by, function)
iris and `by()`

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>

Compute summaries and means of data, grouping by Species:

`<workshop>/examples/by-example.R`
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