Lab 5: Intro to Programming

Stat 133, Fall 2016

Gaston Sanchez

Learning Objectives

- Define a function that takes arguments
- Return a value from a function
- Test a function
- Set default values for function arguments
- Conditional structures: if-then-else, switch
- Loops: while loops, for loops

Functions Basics

So far you've been using a number of functions in R. Now it is time to see how you can create and use your own functions. To define a new function in R you use the function `function()`. Actually, you need to specify a name for the function, and then assign `function()` to the chosen name. You also need to define optional arguments (i.e. inputs). And of course, you must write the code (i.e. the body) so the function does something when you use it:

```
# anatomy of a function
some_name <- function(arguments) {
  # body of the function
}
```

Example: From Fahrenheit to Celsius

Let's consider a typical programming example that involves converting fahrenheit degrees into celsius degrees. The conversion formula is $(F - 32) \times 5/9 = C$. Here's some R code to convert 100 fahrenheit degrees into Celsius degrees:

```
# fahrenheit degrees
far_deg <- 100

# convert to celsius
(far_deg - 32) * (5/9)
```

```
## [1] 37.77778
```

What if you want to conver 90 fahrenheit degrees in Celsius degrees? One option would be to rewrite the previous lines as:

```
# fahrenheit degrees
far_deg <- 90

# convert to celsius
(far_deg - 32) * (5/9)
```

```
## [1] 32.22222
```

However, retyping many lines of code can be very boring, tedious, and inefficient. To make your code reusable in a more efficient manner, you will have to write functions.

Writing a simple function

So, how do you create a function? The first step is to write code and make sure that it works. In this case we already have the code that converts a number in Fahrenheit units into Celsius.

The next step is to **encapsulate** the code in the form of a function. You have to choose a name, some argument(s), and determine the output. Here's one example with a function `fahrenheit_to_celsius()`

```
fahrenheit_to_celsius <- function(x) {
  (x - 32) * (5/9)
}

fahrenheit_to_celsius(100)
```

```
## [1] 37.77778
```

If you want to get the conversion of 90 fahrenheit degrees, you just simply execute it again by changing its argument:

```
fahrenheit_to_celsius(90)
```

```
## [1] 32.22222
```

And because we are using arithmetic operators (i.e. multiplication, subtraction, division), the function is also vectorized:

```
fahrenheit_to_celsius(c(90, 100, 110))
```

```
## [1] 32.22222 37.77778 43.33333
```

Sometimes it is recommended to add a default value to one (or more) of the arguments. In this case, we can give a default value of `x = 1`. When the user executes the function without any input, `fahrenheit_to_celsius` returns the value of 1 fahrenheit degree to Celsius degrees:

```
fahrenheit_to_celsius <- function(x = 1) {
  (x - 32) * (5/9)
}

# default execution
fahrenheit_to_celsius()
```

```
## [1] -17.22222
```

Your turn: miles to kilometers

Write a function `miles_to_kms()` that converts miles into kilometers: 1 mile is equal to 1.6 kilometers. Give the argument a default value of 1.

```r
miles_to_kms <- function() {
  # fill in
}
```

Your turn: gallons to liters

Write a function `gallon_to_liters()` that converts gallons to liters: 1 gallon is equal to 3.78541 liters:

```r
# your gallons to liters function
```

Your turn: seconds to years

According to Wikipedia, in 2015 the life expectancy of a person born in the US was 79 years. Consider the following question: Can a newborn baby in USA expect to live for one billion (10^9) seconds?

To answer this question, write a function `seconds2years()` that takes a number in seconds and returns the equivalent number of years. Test the function with `seconds2years(1000000000)`

Conditionals

The most common conditional structure, conceptually speaking, is the **if-then-else** statement. This type of statement makes it possible to choose between two (possibly compound) expressions depending on the value of a (logical) condition.

In R (as in many other languages) the if-then-else statement has the following structure:

```r
if (condition) {
  # do something
} else {
  # do something else
}
```

As you can tell, the `if` clause works like a function: `if(condition)`. Likewise, braces are used to group one or more expressions. If the condition to be evaluated is true, then just the expressions inside the first pair of braces are executed. If the condition is false, then the expressions inside the second pair of braces are executed.

```r
x <- 1

if (x > 0) {
  print("positive")
} else {
  print("not positive")
}
```

```
## [1] "positive"
```

Loops

Loops are used when you want to perform a given task many times. R provides three differnt types of loop constructs:

- `for` loop
- `while` loop
- `repeat` loop

To illustrate the different types of loop, let's consider the following geometric series:

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$$

The series can be expressed in summation notation as:

$$\sum_{i=0} \frac{1}{2^i}$$

The first 5 term of the series could be obtained like this:

```
# first five terms of the series
1 / (2^0)
1 / (2^1)
1 / (2^2)
1 / (2^3)
1 / (2^4)
```

The sum of those five terms could be compute in R as follows:

```
(1 / (2^0)) + (1 / (2^1)) + (1 / (2^2)) + (1 / (2^3)) + (1 / (2^4))
```

```
## [1] 1.9375
```

Writing code as above is very tedious and error prone. Moreover, we are performing the same operations several times which is the ideal situation to use a loop.

For loop

When you want to perform an action a specified number of times you can use a `for` loop.

Here is how to write a loop for the geometric series up to the first 11 terms, i.e. when $i = 10$

```
# geometric series with a for loop
series1 <- 0

for (i in 0:10) {
  series1 <- series1 + (1 / (2^i))
  print(series1)
}
```

```
## [1] 1
## [1] 1.5
## [1] 1.75
## [1] 1.875
## [1] 1.9375
## [1] 1.96875
## [1] 1.984375
## [1] 1.992188
## [1] 1.996094
## [1] 1.998047
## [1] 1.999023
```

```
series1
```

```
## [1] 1.999023
```

While Loop

Often you will want to perform a loop until some condition is satisfied, or as long as a condition is satisfied. In that case, a **while** loop may be more appropriate.

For the summation series, we can implement a `while` loop until i becomes 11 (i.e. iterate as long as $i < 11$):

```
# geometric series with a while loop
series2 <- 0

i <- 0
while (i <= 10) {
  series2 <- series2 + (1 / (2^i))
  print(series2)
  i <- i + 1
}
```

```
## [1] 1
## [1] 1.5
## [1] 1.75
## [1] 1.875
## [1] 1.9375
## [1] 1.96875
## [1] 1.984375
## [1] 1.992188
## [1] 1.996094
## [1] 1.998047
## [1] 1.999023
```

```
series2
```

```
## [1] 1.999023
```

Repeat Loop

A **repeat** loop will repeatedly evaluate a set of expressions until it is told to stop. This implies using a stop condition with an `if()` statement, and the use of `break` to stop the loop.

For the geometric series, the stop condition is reached when $i == 11$:

```
# geometric series with a while loop
series3 <- 0

i <- 0
repeat {
  series3 <- series3 + (1 / (2^i))
  print(series3)
  i <- i + 1
  if (i == 11) break
}
```

```
## [1] 1
## [1] 1.5
## [1] 1.75
## [1] 1.875
## [1] 1.9375
## [1] 1.96875
## [1] 1.984375
## [1] 1.992188
## [1] 1.996094
## [1] 1.998047
## [1] 1.999023
```

```
series3
```

```
## [1] 1.999023
```

References

- Web: Advanced R: Functions (by Hadley Wickham)
- Tutorial: R novice inflammation: Functions (by Software Carpentry)

Solutions

```
# miles to kms
miles_to_kms <- function(x = 1) {
  x * 1.6
}
```

```r
# gallons to liters
gallons_to_liters <- function(x = 1) {
  x * 3.78541
}


# secs to years
seconds2years <- function(x = 1) {
  x / (365 * 24 * 60 * 60)
}
```