9. Functions

swirl Team

06-26-2022

9. Functions

Author: swirl Team

Acknowledgements: R Language Concepts and code questions are used here from the swirl package. https://www.r-project.org/nosvn/pandoc/swirl.html

By writing functions, you can gain serious insight into how R works. As John Chambers, the creator of R, once said: To understand computations in R, two slogans are helpful:

1. Everything that exists is an object.
2. Everything that happens is a function call.

You're about to write your first function! Just like you would assign a value to a variable with the assignment operator, you assign functions in the following way:

Example 1

function_name <- function(arg1, arg2){ Manipulate arguments in some way Return a value }

arg2 represent the arguments of your function. You can manipulate the arguments you specify within the function. After sourcing the function, you can use the function by typing:

function_name(value1, value2)

Below we will create a function called boring_function. This function takes the argument `x` as input, and returns the value of x without modifying it. Delete the pound sign in front of the x to make the function work! Be sure to save this script and type submit() in the console after you make your changes.

```
boring_function <- function(x) {
      x
}
```

```
boring_function("Test3")
```

```
## [1] "Test3"
```

Example 2

You're free to implement the function my_mean however you want, as long as it returns the average of all of the numbers in `my_vector`.

Hint #1: sum() returns the sum of a vector. Ex: sum(c(1, 2, 3)) evaluates to 6

Hint #2: length() returns the size of a vector. Ex: length(c(1, 2, 3)) evaluates to 3

Hint #3: The mean of all the numbers in a vector is equal to the sum of all of the numbers in the vector divided by the size of the vector.

Note for those of you feeling super clever: Please do not use the mean() function while writing this function. We're trying to teach you something here!

Be sure to save this script and type submit() in the console after you make your changes.

```
my_mean <- function(my_vector) {
  sum_v = sum(my_vector)
  size_v = length(my_vector)
  sum_v/size_v
}
```

```
my_mean(c(8,10,12))
```

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

Example 3

Let me show you an example of a function I'm going to make up called increment(). Most of the time I want to use this function to increase the "by" where "number" is the digit I want to increment and "by" is the amount I want to increment "number" by. I've written the function below.

```
increment <- function(number, by = 1){
    number + by
}
```

```
increment(5)
```

```
## [1] 6
```

```
increment(5,2)
```

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

If you take a look in between the parentheses you can see that I've set "by" equal to 1. This means that the "by" argument will have the default value of 1.

I can now use the increment function without providing a value for "by": increment(5) will evaluate to 6.

However if I want to provide a value for the "by" argument I still can! The expression: increment(5, 2) will evaluate to 7.

Example 4

You're going to write a function called "remainder." remainder() will take two arguments: "num" and "divisor" where "num" is divided by "divisor" and the remainder is returned. Imagine that you usually want to know the remainder when you divide by 2, so set the default value of "divisor" to 2. Please be sure that "num" is the first argument and "divisor" is the second argument.

Hint #1: You can use the modulus operator %% to find the remainder. Ex: 7 %% 4 evaluates to 3.

Remember to set appropriate default values! Be sure to save this script and type submit() in the console after you write the function.

```
remainder <- function(num, divisor) {
  num %% divisor
}
```

```
remainder(59, 7)
```

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