Week 7 Module 4: Examples

CSCI 5a: Programming in R

Module Overview and Learning Objectives

In Module 3, we looked at very artifical examples of functions, just to see how R actually works.

In this module, we'll examine some more interesting examples of writing functions.

The best way to learn how to write your own functions is to work through some exercises.

This module is going to have a different format from the other modules in CSCI 5a.

There's no new content in this module, so there's nothing new for you to "learn".

Instead, it consists of a set of practice exercises, and you should really try to solve each exercise before you look at my solution.

Thus, there are no learning objectives for this module, and instead it's just a sequence of challenges for you.

Exercise 1: Quadratic Function

Write a function that takes 4 numeric values `a`, `b`, `c`, and `x`, and returns the value of the quadratic function:

$$f(x) = ax^2 + bx + c$$

Solution

```
# Type your answer here
```

Now let's try it out:

```
quadratic.function( a = 3, b = 7, c = -9, x = 4 )
```

Exercise 2: Sum of the Squares

Exercise Write a function that takes a numeric vector as its input argument and returns the sum of the squares of the elements of the vector.

Solution

```
# Type your answer here
```

Now let's try it out:

```
test.vector <- c(4, 6, -2, 3, 0, 1)

compute.sum.of.squares( test.vector )
```

Exercise 3: Loan Payment

Write a function that takes 3 input arguments: `loan.amount`, `number.of.payments`, and `interest.rate`, and returns the payment amount that will pay off the initial loan amount over the number of payments, given the interest rate.

Solution

```
# Type your answer here
```

Let's test this out:

```
payment.amount( 1000, 3, 0.05 )
```

Check this with the value that we calculated in Lecture 4, Module 3.

Exercise 4: Logical Function

Write a function named `any.false()` that returns `TRUE` if at least one value of the input vector is `FALSE`.

Solution

```
# Type your answer here
```

Let's run this on a simple test vector:

```
any.false( c(FALSE, TRUE, TRUE, TRUE, FALSE) )
```

It's a good idea to test extreme cases:

```
any.false( c(TRUE, TRUE, TRUE, TRUE, TRUE) )
```

Exercise 5: Removing Missing Values

Write a function that removes missing values from a vector. That is, the function takes a vector of any class, and returns a vector consisting of non-missing values in that vector.

You can do this with one line of code.

Solution

```
# Write your answer here
```

Let's try this out:

```
remove.missing.values(
    c( 4, 7, 8, NA, 4, 3, 1, NA, 0, NA )
)
```

Exercise 6: Low-Pass Filter

Write a function that takes two arguments:

- A numeric vector
- A threshold value

The function then returns all the values in the numeric vector that are less than or equal to the threshold value.

Solution

```
# Type your answer here
```

Let's try this with a simple example:

```
filter.example.data <-
    c(5, 7, 2, 4, 1, 4, 2, 8, 7, 5 )
```

Now let's test our `low.pass.filter()` function:

```
low.pass.filter(
    x = filter.example.data,
    threshold.value = 5
)
```

Exercise 7: High-Pass Filter

Write a function that takes two arguments:

- A numeric vector
- A threshold value

The function then returns all the values in the numeric vector that are strictly greater than the threshold value.

Solution

```
# Type your answer here
```

Let's try this with a simple example:

```
filter.example.data <-
    c(5, 7, 2, 4, 1, 4, 2, 8, 7, 5 )
```

Now let's test our `high.pass.filter()` function:

```
high.pass.filter(
    x = filter.example.data,
    threshold.value = 5
)
```

Exercise 8: Reporter Function

Sometimes, we want to write a function to print out information about an R object.

I call these functions "reporter" functions, because they create a report about the R objects.

We've done some simple reporting in Problem Set 7 with a factor.

Notice for this reporter function we don't really care what the return value is; instead, we want the function to *do* something for us (i.e. print out information) rather than perform some calculation and return a value to us.

Write a reporter function that takes an R object and prints out this information:

- First, it states the class of the object.
- Second, it states the length of the R object.
- Third, it reports the number of missing items.
- Finally, if there are any missing items it prints out the locations of the missing items.

Solution

```
# Type your answer here
```

Let's try this on a numeric vector:

```
numeric.test.vector <-
    c( 5, 7, 8, 3, 4, 1, NA, 7, NA )

reporter.function( numeric.test.vector )
```

Now let's try it on a factor:

```
test.factor <-
    factor(
        c( "Red", "Blue", "Red", "Green", "Blue" )
    )

reporter.function( test.factor )
```

Module Review

As I mentioned at the beginning, there wasn't any new content in this module, so there wasn't anything new for you to "learn".

Instead, it was just a sequence of practice exercises.

I hope you worked through each one yourself before you looked at my answers!

Being able to write your own user-defined functions is a crucial skill to develop if you want to be an R expert.

Solutions to the Exercises

Exercise 1: Quadratic Function

Write a function that takes 4 numeric values `a`, `b`, `c`, and `x`, and returns the value of the quadratic function:

$$f(x) \; = \; a \cdot x^2 + b \cdot x + c$$

Solution

```
# Type your answer here
```

Here's my answer:

```
quadratic.function <- function( a, b, c, x ) {

    return.value <- a*x^2 + b*x + c

    return( return.value )
}
```

Now let's try it out:

```
quadratic.function( a = 3, b = 7, c = -9, x = 4 )
```

```
## [1] 67
```

Exercise 2: Sum of the Squares

Exercise Write a function that takes a numeric vector as its input argument and returns the sum of the squares of the elements of the vector.

Solution

```
# Type your answer here
```

Here's my solution:

```
compute.sum.of.squares <-
    function( numeric.vector ) {

        final.result <- sum( numeric.vector^2 )

        return( final.result )
    }
```

Now let's try it out:

```
test.vector <- c(4, 6, -2, 3, 0, 1)

compute.sum.of.squares( test.vector )
```

```
## [1] 66
```

Exercise 3: Loan Payment

Write a function that takes 3 input arguments: `loan.amount`, `number.of.payments`, and `interest.rate`, and returns the payment amount that will pay off the initial loan amount over the number of payments, given the interest rate.

Solution

```
# Type your answer here
```

Here's my solution:

```
payment.amount <-
    function(
        loan.amount,
        number.of.payments,
        interest.rate ) {

        discount.factor <-
            1/(1 + interest.rate)

        payment.amount <-
            interest.rate * loan.amount /
            (1 - discount.factor^number.of.payments)

        return( payment.amount )
}
```

Let's test this out:

```
payment.amount( 1000, 3, 0.05 )
```

```
## [1] 367.2086
```

Check this with the value that we calculated in Lecture 4, Module 3.

Exercise 4: Logical Function

Write a function named `any.false()` that returns `TRUE` if at least one value of the input vector is `FALSE`.

Solution

```
# Type your answer here
```

Here's a very straightforward solution:

```
any.false <-
    function( input.logical.vector ) {

        return( any( input.logical.vector == FALSE ) )
}
```

Let's run this on a simple test vector:

```
any.false( c(FALSE, TRUE, TRUE, TRUE, FALSE) )
```

```
## [1] TRUE
```

It's a good idea to test extreme cases:

```
any.false( c(TRUE, TRUE, TRUE, TRUE, TRUE) )
```

```
## [1] FALSE
```

Here's a variation of that solution:

```
any.false.2 <-
    function( input.logical.vector ) {

        return( any( ! input.logical.vector ) )
}
```

We will test this using our simple test vector:

```
any.false( c(FALSE, TRUE, TRUE, TRUE, FALSE) )
```

```
## [1] TRUE
```

Let's test the function on the extreme case:

```
any.false( c(TRUE, TRUE, TRUE, TRUE, TRUE) )
```

```
## [1] FALSE
```

Here's a different solution:

```
any.false.3 <-
    function( input.logical.vector ) {

        return( ! all( input.logical.vector ) )
}
```

We will test this using our simple test vector:

```
any.false.3( c(FALSE, TRUE, TRUE, TRUE, FALSE) )
```

```
## [1] TRUE
```

Let's test the function on the extreme case:

```
any.false.3( c(TRUE, TRUE, TRUE, TRUE, TRUE) )
```

```
## [1] FALSE
```

Exercise 5: Removing Missing Values

Write a function that removes missing values from a vector. That is, the function takes a vector of any class, and returns the set of non-missing values in that vector.

You can do this with one line of code.

Solution

```
# Write your answer here
```

Here's my solution:

```
remove.missing.values <-
    function( x ) {

        return( x[ ! is.na( x ) ] )
    }
```

Let's try this out:

```
remove.missing.values(
    c( 4, 7, 8, NA, 4, 3, 1, NA, 0, NA )
)
```

```
## [1] 4 7 8 4 3 1 0
```

Exercise 6: Low-Pass Filter

Write a function that takes two arguments:

- A numeric vector
- A threshold value

The function then returns all the values in the numeric vector that are less than or equal to the threshold value.

Solution

```
# Type your answer here
```

Here's my solution:

```
low.pass.filter <-
    function(x, threshold.value ) {

        logical.indexing.vector <-
            (x <= threshold.value)

        filtered.vector <-
            x[ logical.indexing.vector ]

        return( filtered.vector )
    }
```

I could have also written this:

```
low.pass.filter <-
    function(x, threshold.value ) {

        return( x[ x <= threshold.value ] )
    }
```

Let's try this with a simple example:

```
filter.example.data <-
    c(5, 7, 2, 4, 1, 4, 2, 8, 7, 5 )
```

Now let's test our `low.pass.filter()` function:

```
low.pass.filter(
    x = filter.example.data,
    threshold.value = 5
)
```

```
## [1] 5 2 4 1 4 2 5
```

Exercise 7: High-Pass Filter

Write a function that takes two arguments:

- A numeric vector
- A threshold value

The function then returns all the values in the numeric vector that are strictly greater than the threshold value.

Solution

```
# Type your answer here
```

Here's my solution:

```
high.pass.filter <-
    function(x, threshold.value ) {

        logical.indexing.vector <-
            (x > threshold.value)

        filtered.vector <-
            x[ logical.indexing.vector ]

        return( filtered.vector )
    }
```

I could have also written this:

```
high.pass.filter <-
    function(x, threshold.value ) {

        return( x[ x > threshold.value ] )
    }
```

Let's try this with a simple example:

```
filter.example.data <-
    c(5, 7, 2, 4, 1, 4, 2, 8, 7, 5 )
```

Now let's test our `high.pass.filter()` function:

```
high.pass.filter(
    x = filter.example.data,
    threshold.value = 5
)
```

```
## [1] 7 8 7
```

Exercise 8: Reporter Function

Sometimes, we want to write a function to print out information about an R object.

I call these functions "reporter" functions, because they create a report about the R objects.

We've done some simple reporting in Problem Set 7 with a factor.

Notice for this reporter function we don't really care what the return value is; instead, we want the function to *do* something for us (i.e. print out information) rather than perform some calculation and return a value to us.

Write a reporter function that takes an R object and prints out this information:

- First, it states the class of the object.
- Second, it states the length of the R object.
- Third, it reports the number of missing items.
- Finally, if there are any missing items it prints out the locations of the missing items.

Solution

```
# Type your answer here
```

Here's my solution:

```
reporter.function <-
    function( x ) {

        cat( "Object class:", class( x ), "\n" )

        cat( "Object length:", length( x ), "\n" )

        number.of.missing.items <-
            sum( is.na( x ) )

        cat( "Number of missing items:",
             number.of.missing.items, "\n" )

        if( number.of.missing.items >= 1 ) {

            missing.item.locations <-
                which( is.na( x ) )

            cat( "Missing item locations:",
                 missing.item.locations, "\n" )
        }
    }
```

Let's try this on a numeric vector:

```
numeric.test.vector <-
    c( 5, 7, 8, 3, 4, 1, NA, 7, NA )

reporter.function( numeric.test.vector )
```

```
## Object class: numeric
## Object length: 9
## Number of missing items: 2
## Missing item locations: 7 9
```

Now let's try it on a factor:

```
test.factor <-
    factor(
        c( "Red", "Blue", "Red", "Green", "Blue" )
    )

reporter.function( test.factor )
```

```
## Object class: factor
## Object length: 5
## Number of missing items: 0
```

Notice that in this case the function doesn't print out the locations of the missing items because there aren't any.