Week 2 Module 2: Variables

CSCI E-5a: Programming in R

Let's clear the global computing environment:

```
rm( list = ls() )
```

Module Overview and Learning Outcomes

In this module, we'll study the concept of variables, which is a fundamental construct for all programming languages.

- In section 1 we'll learn what variables are, and how we can use them to store values.
- In section 2, we'll learn about the environment, which stores information about variables.
- In section 3, we'll learn about the rules and conventions for naming variables.
- In section 4, we'll learn how to report results using the `cat()` and `formatC()` functions.

At the end of this module, you'll be able to:

- Store the result of a computation into a variable by using the assignment operator.
- Explain how variables are stored in the environment.
- Identify correct and incorrect variable names.
- Report variables using the `cat()` and `formatC()` functions.

We'll also meet two new built-in R functions:

- The `cat()` function
- The `formatC()` function

OK! Let's get started by learning how to use variables to store values.

Section 1: Storing Values

> **Main Idea:** *We can store a value in a variable for future use.*

In this section, we'll learn what variables are, and how we can use them to store values.

Often, when we perform a complicated calculation we would like to store the result in case we have to use this value again.

For instance, we might want to store our result using a symbol such as `x` or `winter.2018.quarterly.sales`.

In R, we can do this by using the *assignment operator*, denoted by `<-`.

For instance, we can associate the symbol `x` with the result of adding 2 and 2:

```
x <- 2 + 2
```

There's a lot going on in this one line of code.

- First, R evaluates the expression on right-hand side and obtains the value 4.
- If the symbol `x` does not currently exist, R will allocate storage for this object, and then store the value 4 in that location.
- If the object `x` already currently exists, then R will overwrite the pre-existing value.

When we create a new variable `x`, this is called *initializing* the variable, because we are creating the object `x` and assigning the first value to it.

If the variable `x` already exists, the assignment operation is called *destructive modification*, because we destroying the old value when we assign the new value.

Be careful: there's no undo for variable assignment!

Thus, once created, the object `x` might be associated with many different values during the course of a computation, and so it is called a *variable*.

Now we can refer to the symbol `x`, and R will return the value that is currently associated with it:

```
x
```

```
## [1] 4
```

This `x` object is a number, and just like any other number we can perform various operations with it.

For instance, we can subtract 1 from it:

```
x - 1
```

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

To evaluate this expression, R first determines the value of `x`, which is currently 4, and then subtracts 1 from it, resulting in the final value of 3.

Notice that when we run this code in the previous example, the variable `x` remains unchanged:

```
x
```

```
## [1] 4
```

So that's what variables are, and how we can use them to store values.

Now let's learn what an "environment" is in R.

Section 2: Environments

Main Idea: *We can use the environment to manage variables.*

In this section, we'll learn about the environment, which stores information about variables.

When we create a variable, we associate a value with a symbol.

Where does this information about the symbol and its associated value get stored?

The R system has what's known as a *global environment*, and this is where your variable definitions are recorded.

To view the contents of the current global environment, go to the Environment panel, which displays all the variables and their associated values.

Try to find the "Environment" panel in the RStudio window, and check to make sure that the variable `x` has the value 4.

It's often nice to start a session with an empty environment, and that way we don't have any extraneous junk in the environment from previous sessions.

To completely clear the global environment of all current variables, run this code:

```
rm( list = ls() )
```

In our course, we always run this code chunk at the beginning of every notebook.

When the environment is completely empty, it's easy to see how variable initialization and modification work.

Let's go back and initialize the variable `x` just like we did before, but now you should pay attention to what happens in the Environment tab.

Since we just cleared the environment, there should be nothing in the display.

Then, when we run this code, the value `x` should appear in the environment, along with its associated value of 4:

```
x <- 2 + 2
```

Do you see a new entry in the Environment panel?

Now let's destructively modify `x`:

```
x <- 6
```

You should verify that `x` is currently associated with the value 6 in the global environment.

So that's how the environment stores information about variables.

Now let's think about how to name variables.

Exercise 2.1: Initializing and modifying a variable

Initialize the variable `y` with the value 7, paying attention to the display in the Environment tab.

Then destructively modify it so that it is associated with the value -2.

Solution

Section 3: Variable names

> **Main Idea:** *There are rules for naming variables.*

In this section, we'll learn about the rules and conventions for naming variables.

Now there are some rules for naming variables – we can't use just any sequence of characters.

A legal variable name may consist of letters, numbers, periods, or underscores, but it may not begin with a number.

In particular, a variable name cannot contain spaces.

Here are some legal variable names:

- `x`
- `winter.2018.quarterly.sales`
- `boston_revenue_data`
- `.CumulativeSum`

Here are some non-legal variable names:

- `2018.sales`
- `Profit@Boston`
- `January Rental Sales`

It's nice to have some sort of convention about how to name variables, because it's easier to read and work with code when there is consistency.

For our course, I will always use lower-case letters and will separate words using a period.

This is the naming convention that base R uses (for the most part), but there are other approaches as well.

The important thing is to be consistent once you've adopted a convention.

Whatever you do, make your variable names highly descriptive.

For instance, suppose you are conducting a study on a psychiatric disorder, and you want to record the age of a patient when they had their first episode.

One choice for the variable name would be `age.at.first.diagnosis`, and I like this because it's clear, easy to read, and can be understood without any effort.

Many people will however name this variable something like `afd`, abbreviating "age at first diagnosis".

The problem with a name such as `afd` is that it is cryptic and cannot be easily understood, and thus makes your code hard to read.

You might think that this is not a big deal, but if you have multiple variables with such abbreviated names it can quickly become difficult to remember everything.

Remember, you might need to modify or adjust your code in the future, and 3 months from now you might not find it so easy to recall what all the variable names stand for.‘

So those are the rules and conventions for naming variables.

Now let’s see how to report our results.

Section 4: Reporting Variables

> **Main Idea:** *We can use standard methods for reporting values.*

In this section, we’ll learn how to report results using the `cat()` and `formatC()` functions.

When you perform a computation, it’s important to be able to report the result of that computation clearly and precisely.

So far, we’ve been doing simple arithmetic calculations, and it’s easy to just type in an expression, have R evaluate it, and then display the result.

Here’s an example: R has a built-in constant called `pi`, which has the value of the famous mathematical constant.

We can display the value of π:

```
pi
```

```
## [1] 3.141593
```

For our course, I’m going to call this “directly displaying” the result.

Directly displaying a result is quick and convenient, and it’s fine when we’re working through a computation together.

However, for more formal presentations, this isn’t so desirable, because there’s no way for a reader to know what the number represents without being intimately familiar with your code.

Instead, we will use a different approach.

The `cat()` function can print out a message:

```
cat( "Hi Mom!" )
```

```
## Hi Mom!
```

What’s remarkable about the `cat()` function is that we can give it multiple input arguments, and it will format and print them.

Let’s use the `cat()` function to print out the value of `pi`, using the built-in constant:

```
cat( "pi =", pi)
```

```
## pi = 3.141593
```

This is a big improvement, but it's still a little unfortunate because there are too many decimal places.

We can control the number of decimal places using a function called `formatC()`.

The `formatC()` function has a lot of features, and we won't investigate all of them.

For our purposes you need to specify three arguments for the `formatC()` function:

- First, the number that you want to display.
- Next, you need to set the format using `format = "f"`.
- Finally, you can specify the number of decimal places by using the `digits` argument.

Let's format `pi` with 4 decimal places, and display the result directly:

```
formatC(
    pi,
    format = "f",
    digits = 4
)
```

```
## [1] "3.1416"
```

You can see that the number is now displayed with only 4 decimal places.

Now let's combine this with the `cat()` statement:

```
cat(
    "pi =",
    formatC(
        pi,
        format = "f",
        digits = 4
    )
)
```

```
## pi = 3.1416
```

Whenever you report a final value for a problem on a problem set or an exam, you should always use this approach for reporting the value.

You can also use the `formatC()` function to report numeric values in scientific notation by setting the `format` option to "e":

```
cat(
    "123.456 =",
    formatC(
        123.456,
        format = "e",
        digits = 4
    )
)
```

```
## 123.456 = 1.2346e+02
```

There's a lot more to the `formatC()` function than what I've shown here, and this is only an introduction to this very powerful function.

If you need to do more sophisticated formatting, you should check out the documentation for the `formatC()` function.

So that's how to report results using the `cat()` and `formatC()` functions.

Now let's review what we've learned in this module.

Exercise 2.2: Reporting numeric values

Report the value of `pi` using a `cat()` statement, displaying this value with 2 decimal places.

Solution

Module Review

In this module, we studied the concept of variables, which is a fundamental construct for all programming languages.

- In section 1 we learned what variables are, and how we can use them to store values.
- In section 2, we learned about the environment, which stores information about variables.
- In section 3, we learned about the rules and conventions for naming variables.
- In section 4, we saw how to report results using the `cat()` and `formatC()` functions.

Now that you've completed this module, you should be able to:

- Store the result of a computation into a variable by using the assignment operator.
- Explain how variables are stored in the environment.
- Identify correct and incorrect variable names.
- Report variables using the `cat()` and `formatC()` functions.

We also met two new built-in R functions:

- The `cat()` function
- The `formatC()` function

All right! That's it for MOdule 2: Variables.

Now let's move on to Module 3: Baseball Batting Statistics.

Solution to the Exercise

Exercise 2.1: Initializing and modifying a variable

Initialize the variable `y` with the value 7, paying attention to the display in the Environment tab.

Then destructively modify it so that it is associated with the value -2.

Solution

First, let's initialize the variable `y` with the value 7.

```
y <- 7
```

Now let's destructively modify the variable, replacing the associated value with 2.

```
y <- 2
```

Exercise 2.2

Report the value of `pi` using a `cat()` statement, displaying this value with 2 decimal places.

Solution

```
cat(
    "pi =",
    formatC(
        pi,
        format = "f",
        digits = 2
    )
)
```

```
## pi = 3.14
```