> library(swirl)

| Hi! I see that you have some variables saved in your workspace. To keep things

| running smoothly, I recommend you clean up before starting swirl.

| Type ls() to see a list of the variables in your workspace. Then, type

| rm(list=ls()) to clear your workspace.

| Type swirl() when you are ready to begin.

> install\_from\_swirl("R Programming")

|========================================================================| 100%

Downloading: 46 MB

| Course installed successfully!

> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use the same name

| as you did then. If you are new, call yourself something unique.

What shall I call you?

| Please don't use any quotes or other punctuation in your name.

What shall I call you? Basic Building Blocks

| Thanks, Basic Building Blocks. Let's cover a couple of quick housekeeping items

| before we begin our first lesson. First of all, you should know that when you

| see '...', that means you should press Enter when you are done reading and

| ready to continue.

... <-- That's your cue to press Enter to continue

| Also, when you see 'ANSWER:', the R prompt (>), or when you are asked to select

| from a list, that means it's your turn to enter a response, then press Enter to

| continue.

Select 1, 2, or 3 and press Enter

1: Continue.

2: Proceed.

3: Let's get going!

Selection: 1

| You can exit swirl and return to the R prompt (>) at any time by pressing the

| Esc key. If you are already at the prompt, type bye() to exit and save your

| progress. When you exit properly, you'll see a short message letting you know

| you've done so.

| When you are at the R prompt (>):

| -- Typing skip() allows you to skip the current question.

| -- Typing play() lets you experiment with R on your own; swirl will ignore what

| you do...

| -- UNTIL you type nxt() which will regain swirl's attention.

| -- Typing bye() causes swirl to exit. Your progress will be saved.

| -- Typing main() returns you to swirl's main menu.

| -- Typing info() displays these options again.

| Let's get started!

...

| Please choose a course, or type 0 to exit swirl.

1: R Programming

2: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks 2: Workspace and Files

3: Sequences of Numbers 4: Vectors

5: Missing Values 6: Subsetting Vectors

7: Matrices and Data Frames 8: Logic

9: Functions 10: lapply and sapply

11: vapply and tapply 12: Looking at Data

13: Simulation 14: Dates and Times

15: Base Graphics

Selection: 1

| | 0%

| In this lesson, we will explore some basic building blocks of the R programming

| language.

...

|== | 3%

| If at any point you'd like more information on a particular topic related to R,

| you can type help.start() at the prompt, which will open a menu of resources

| (either within RStudio or your default web browser, depending on your setup).

| Alternatively, a simple web search often yields the answer you're looking for.

...==

|==== | 5%

| In its simplest form, R can be used as an interactive calculator. Type 5 + 7

| and press Enter.

> 5+7

[1] 12

| Your dedication is inspiring!

|====== | 8%

| R simply prints the result of 12 by default. However, R is a programming

| language and often the reason we use a programming language as opposed to a

| calculator is to automate some process or avoid unnecessary repetition.

…

|======== | 11%

| In this case, we may want to use our result from above in a second calculation.

| Instead of retyping 5 + 7 every time we need it, we can just create a new

| variable that stores the result.

...

|========= | 13%

| The way you assign a value to a variable in R is by using the assignment

| operator, which is just a 'less than' symbol followed by a 'minus' sign. It

| looks like this: <-

...

|=========== | 16%

| Think of the assignment operator as an arrow. You are assigning the value on

| the right side of the arrow to the variable name on the left side of the arrow.

...

|============= | 18%

| To assign the result of 5 + 7 to a new variable called x, you type x <- 5 + 7.

| This can be read as 'x gets 5 plus 7'. Give it a try now.

> x<-5+7

| Keep working like that and you'll get there!

|=============== | 21%

| You'll notice that R did not print the result of 12 this time. When you use the

| assignment operator, R assumes that you don't want to see the result

| immediately, but rather that you intend to use the result for something else

| later on.

...

|================= | 24%

| To view the contents of the variable x, just type x and press Enter. Try it

| now.

> x

[1] 12

| You're the best!

|=================== | 26%

| Now, store the result of x - 3 in a new variable called y.

> y<-x-3

| You nailed it! Good job!

|===================== | 29%

| What is the value of y? Type y to find out.

> y

[1] 9

| You are quite good my friend!

|======================= | 32%

| Now, let's create a small collection of numbers called a vector. Any object

| that contains data is called a data structure and numeric vectors are the

| simplest type of data structure in R. In fact, even a single number is

| considered a vector of length one.

...

|========================= | 34%

| The easiest way to create a vector is with the c() function, which stands for

| 'concatenate' or 'combine'. To create a vector containing the numbers 1.1, 9,

| and 3.14, type c(1.1, 9, 3.14). Try it now and store the result in a variable

| called z.

> c(1.1, 9, 3.14)

[1] 1.10 9.00 3.14

| Give it another try. Or, type info() for more options.

| Inputting z <- c(1.1, 9, 3.14) will assign the vector (1.1, 9, 3.14) to a new

| variable called z. Including single spaces after the commas in the vector is

| not required, but helps make your code less cluttered and more readable.

> z <- c(1.1, 9, 3.14)

| You are really on a roll!

|=========================== | 37%

| Anytime you have questions about a particular function, you can access R's

| built-in help files via the `?` command. For example, if you want more

| information on the c() function, type ?c without the parentheses that normally

| follow a function name. Give it a try.

> ?c

| Keep up the great work!

|============================ | 39%

| Type z to view its contents. Notice that there are no commas separating the

| values in the output.

> z

[1] 1.10 9.00 3.14

| That's a job well done!

|============================== | 42%

| You can combine vectors to make a new vector. Create a new vector that contains

| z, 555, then z again in that order. Don't assign this vector to a new variable,

| so that we can just see the result immediately.

> c(z,555,z)

[1] 1.10 9.00 3.14 555.00 1.10 9.00 3.14

| Excellent job!

|================================ | 45%

| Numeric vectors can be used in arithmetic expressions. Type the following to

| see what happens: z \* 2 + 100.

> z\*2+10

[1] 12.20 28.00 16.28

| Keep trying! Or, type info() for more options.

| Enter z \* 2 + 100, without assigning the result to a variable. The `\*` symbol

| is used for multiplication and shares a key with the number 8 on many

| keyboards.

> z \* 2 + 100,

Error: unexpected ',' in "z \* 2 + 100,"

> z \* 2 + 100

[1] 102.20 118.00 106.28

| You got it right!

|================================== | 47%

| First, R multiplied each of the three elements in z by 2. Then it added 100 to

| each element to get the result you see above.

...

|==================================== | 50%

| Other common arithmetic operators are `+`, `-`, `/`, and `^` (where x^2 means

| 'x squared'). To take the square root, use the sqrt() function and to take the

| absolute value, use the abs() function.

...

|====================================== | 53%

| Take the square root of z - 1 and assign it to a new variable called my\_sqrt.

> sqrt(z-1)

[1] 0.3162278 2.8284271 1.4628739

| That's not the answer I was looking for, but try again. Or, type info() for

| more options.

| Assign the result of sqrt(z - 1) to a variable called my\_sqrt.

> my\_sqrt<-sqrt(z-1)

| You are really on a roll!

|======================================== | 55%

| Before we view the contents of the my\_sqrt variable, what do you think it

| contains?

1: a vector of length 0 (i.e. an empty vector)

2: a single number (i.e a vector of length 1)

3: a vector of length 3

Selection: 1

| Keep trying!

| Think about how R handled the other 'vectorized' operations:

| element-by-element.

1: a vector of length 0 (i.e. an empty vector)

2: a single number (i.e a vector of length 1)

3: a vector of length 3

Selection: 3

| Great job!

|========================================== | 58%

| Print the contents of my\_sqrt.

> my\_sqrt

[1] 0.3162278 2.8284271 1.4628739

| That's the answer I was looking for.

|============================================ | 61%

| As you may have guessed, R first subtracted 1 from each element of z, then took

| the square root of each element. This leaves you with a vector of the same

| length as the original vector z.

...

|============================================= | 63%

| Now, create a new variable called my\_div that gets the value of z divided by

| my\_sqrt.

> my\_div<-z/my\_sqrt

| Great job!

|=============================================== | 66%

| Which statement do you think is true?

1: my\_div is a single number (i.e a vector of length 1)

2: The first element of my\_div is equal to the first element of z divided by the first element of my\_sqrt, and so on...

3: my\_div is undefined

Selection: 2

| You got it right!

|================================================= | 68%

| Go ahead and print the contents of my\_div.

> my\_div

[1] 3.478505 3.181981 2.146460

| That's the answer I was looking for.

|=================================================== | 71%

| When given two vectors of the same length, R simply performs the specified

| arithmetic operation (`+`, `-`, `\*`, etc.) element-by-element. If the vectors

| are of different lengths, R 'recycles' the shorter vector until it is the same

| length as the longer vector.

...

|===================================================== | 74%

| When we did z \* 2 + 100 in our earlier example, z was a vector of length 3, but

| technically 2 and 100 are each vectors of length 1.

...

|======================================================= | 76%

| Behind the scenes, R is 'recycling' the 2 to make a vector of 2s and the 100 to

| make a vector of 100s. In other words, when you ask R to compute z \* 2 + 100,

| what it really computes is this: z \* c(2, 2, 2) + c(100, 100, 100).

...

|========================================================= | 79%

| To see another example of how this vector 'recycling' works, try adding c(1, 2,

> library(swirl)

| Hi! I see that you have some variables saved in your workspace. To keep things

| running smoothly, I recommend you clean up before starting swirl.

| Type ls() to see a list of the variables in your workspace. Then, type

| rm(list=ls()) to clear your workspace.

| Type swirl() when you are ready to begin.

> install\_from\_swirl("R Programming")

|========================================================================| 100%

Downloading: 46 MB

| Course installed successfully!

> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use the same name

| as you did then. If you are new, call yourself something unique.

What shall I call you?

| Please don't use any quotes or other punctuation in your name.

What shall I call you? Basic Building Blocks

| Thanks, Basic Building Blocks. Let's cover a couple of quick housekeeping items

| before we begin our first lesson. First of all, you should know that when you

| see '...', that means you should press Enter when you are done reading and

| ready to continue.

... <-- That's your cue to press Enter to continue

| Also, when you see 'ANSWER:', the R prompt (>), or when you are asked to select

| from a list, that means it's your turn to enter a response, then press Enter to

| continue.

Select 1, 2, or 3 and press Enter

1: Continue.

2: Proceed.

3: Let's get going!

Selection: 1

| You can exit swirl and return to the R prompt (>) at any time by pressing the

| Esc key. If you are already at the prompt, type bye() to exit and save your

| progress. When you exit properly, you'll see a short message letting you know

| you've done so.

| When you are at the R prompt (>):

| -- Typing skip() allows you to skip the current question.

| -- Typing play() lets you experiment with R on your own; swirl will ignore what

| you do...

| -- UNTIL you type nxt() which will regain swirl's attention.

| -- Typing bye() causes swirl to exit. Your progress will be saved.

| -- Typing main() returns you to swirl's main menu.

| -- Typing info() displays these options again.

| Let's get started!

...

| Please choose a course, or type 0 to exit swirl.

1: R Programming

2: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks 2: Workspace and Files

3: Sequences of Numbers 4: Vectors

5: Missing Values 6: Subsetting Vectors

7: Matrices and Data Frames 8: Logic

9: Functions 10: lapply and sapply

11: vapply and tapply 12: Looking at Data

13: Simulation 14: Dates and Times

15: Base Graphics

Selection: 1

| | 0%

| In this lesson, we will explore some basic building blocks of the R programming

| language.

...

|== | 3%

| If at any point you'd like more information on a particular topic related to R,

| you can type help.start() at the prompt, which will open a menu of resources

| (either within RStudio or your default web browser, depending on your setup).

| Alternatively, a simple web search often yields the answer you're looking for.

...==

|==== | 5%

| In its simplest form, R can be used as an interactive calculator. Type 5 + 7

| and press Enter.

> 5+7

[1] 12

| Your dedication is inspiring!

|====== | 8%

| R simply prints the result of 12 by default. However, R is a programming

| language and often the reason we use a programming language as opposed to a

| calculator is to automate some process or avoid unnecessary repetition.

...12

|======== | 11%

| In this case, we may want to use our result from above in a second calculation.

| Instead of retyping 5 + 7 every time we need it, we can just create a new

| variable that stores the result.

...total<-5+7

|========= | 13%

| The way you assign a value to a variable in R is by using the assignment

| operator, which is just a 'less than' symbol followed by a 'minus' sign. It

| looks like this: <-

...

|=========== | 16%

| Think of the assignment operator as an arrow. You are assigning the value on

| the right side of the arrow to the variable name on the left side of the arrow.

...

|============= | 18%

| To assign the result of 5 + 7 to a new variable called x, you type x <- 5 + 7.

| This can be read as 'x gets 5 plus 7'. Give it a try now.

> x<-5+7

| Keep working like that and you'll get there!

|=============== | 21%

| You'll notice that R did not print the result of 12 this time. When you use the

| assignment operator, R assumes that you don't want to see the result

| immediately, but rather that you intend to use the result for something else

| later on.

...

|================= | 24%

| To view the contents of the variable x, just type x and press Enter. Try it

| now.

> x

[1] 12

| You're the best!

|=================== | 26%

| Now, store the result of x - 3 in a new variable called y.

> y<-x-3

| You nailed it! Good job!

|===================== | 29%

| What is the value of y? Type y to find out.

> y

[1] 9

| You are quite good my friend!

|======================= | 32%

| Now, let's create a small collection of numbers called a vector. Any object

| that contains data is called a data structure and numeric vectors are the

| simplest type of data structure in R. In fact, even a single number is

| considered a vector of length one.

...

|========================= | 34%

| The easiest way to create a vector is with the c() function, which stands for

| 'concatenate' or 'combine'. To create a vector containing the numbers 1.1, 9,

| and 3.14, type c(1.1, 9, 3.14). Try it now and store the result in a variable

| called z.

> C(1.1, 9, 3.14)

Error in C(1.1, 9, 3.14) : object not interpretable as a factor

> c(1.1, 9, 3.14)

[1] 1.10 9.00 3.14

| Give it another try. Or, type info() for more options.

| Inputting z <- c(1.1, 9, 3.14) will assign the vector (1.1, 9, 3.14) to a new

| variable called z. Including single spaces after the commas in the vector is

| not required, but helps make your code less cluttered and more readable.

> z <- c(1.1, 9, 3.14)

| You are really on a roll!

|=========================== | 37%

| Anytime you have questions about a particular function, you can access R's

| built-in help files via the `?` command. For example, if you want more

| information on the c() function, type ?c without the parentheses that normally

| follow a function name. Give it a try.

> ?c

| Keep up the great work!

|============================ | 39%

| Type z to view its contents. Notice that there are no commas separating the

| values in the output.

> z

[1] 1.10 9.00 3.14

| That's a job well done!

|============================== | 42%

| You can combine vectors to make a new vector. Create a new vector that contains

| z, 555, then z again in that order. Don't assign this vector to a new variable,

| so that we can just see the result immediately.

> c(z,555,z)

[1] 1.10 9.00 3.14 555.00 1.10 9.00 3.14

| Excellent job!

|================================ | 45%

| Numeric vectors can be used in arithmetic expressions. Type the following to

| see what happens: z \* 2 + 100.

> z\*2+10

[1] 12.20 28.00 16.28

| Keep trying! Or, type info() for more options.

| Enter z \* 2 + 100, without assigning the result to a variable. The `\*` symbol

| is used for multiplication and shares a key with the number 8 on many

| keyboards.

> z \* 2 + 100,

Error: unexpected ',' in "z \* 2 + 100,"

> z \* 2 + 100

[1] 102.20 118.00 106.28

| You got it right!

|================================== | 47%

| First, R multiplied each of the three elements in z by 2. Then it added 100 to

| each element to get the result you see above.

...

|==================================== | 50%

| Other common arithmetic operators are `+`, `-`, `/`, and `^` (where x^2 means

| 'x squared'). To take the square root, use the sqrt() function and to take the

| absolute value, use the abs() function.

...

|====================================== | 53%

| Take the square root of z - 1 and assign it to a new variable called my\_sqrt.

> sqrt(z-1)

[1] 0.3162278 2.8284271 1.4628739

| That's not the answer I was looking for, but try again. Or, type info() for

| more options.

| Assign the result of sqrt(z - 1) to a variable called my\_sqrt.

> my\_sqrt<-sqrt(z-1)

| You are really on a roll!

|======================================== | 55%

| Before we view the contents of the my\_sqrt variable, what do you think it

| contains?

1: a vector of length 0 (i.e. an empty vector)

2: a single number (i.e a vector of length 1)

3: a vector of length 3

Selection: 1

| Keep trying!

| Think about how R handled the other 'vectorized' operations:

| element-by-element.

1: a vector of length 0 (i.e. an empty vector)

2: a single number (i.e a vector of length 1)

3: a vector of length 3

Selection: 3

| Great job!

|========================================== | 58%

| Print the contents of my\_sqrt.

>

> my\_sqrt()

Error in my\_sqrt() : could not find function "my\_sqrt"

> my\_sqrt

[1] 0.3162278 2.8284271 1.4628739

| That's the answer I was looking for.

|============================================ | 61%

| As you may have guessed, R first subtracted 1 from each element of z, then took

| the square root of each element. This leaves you with a vector of the same

| length as the original vector z.

...

|============================================= | 63%

| Now, create a new variable called my\_div that gets the value of z divided by

| my\_sqrt.

> my\_div<-(z/my\_sqrt)

| Not quite right, but keep trying. Or, type info() for more options.

| Enter my\_div <- z / my\_sqrt. The spaces on either side of the `/` sign are not

| required, but can often improve readability by making code appear less

| cluttered. In the end, it's personal preference.

> my\_div<-z/my\_sqrt

| Great job!

|=============================================== | 66%

| Which statement do you think is true?

1: my\_div is a single number (i.e a vector of length 1)

2: The first element of my\_div is equal to the first element of z divided by the first element of my\_sqrt, and so on...

3: my\_div is undefined

Selection: 2

| You got it right!

|================================================= | 68%

| Go ahead and print the contents of my\_div.

> my\_div

[1] 3.478505 3.181981 2.146460

| That's the answer I was looking for.

|=================================================== | 71%

| When given two vectors of the same length, R simply performs the specified

| arithmetic operation (`+`, `-`, `\*`, etc.) element-by-element. If the vectors

| are of different lengths, R 'recycles' the shorter vector until it is the same

| length as the longer vector.

...

|===================================================== | 74%

| When we did z \* 2 + 100 in our earlier example, z was a vector of length 3, but

| technically 2 and 100 are each vectors of length 1.

...

|======================================================= | 76%

| Behind the scenes, R is 'recycling' the 2 to make a vector of 2s and the 100 to

| make a vector of 100s. In other words, when you ask R to compute z \* 2 + 100,

| what it really computes is this: z \* c(2, 2, 2) + c(100, 100, 100).

...

|========================================================= | 79%

| To see another example of how this vector 'recycling' works, try adding c(1, 2,

| 3, 4) and c(0, 10). Don't worry about saving the result in a new variable.

> c(1,2,3,4)+c(0,10)

[1] 1 12 3 14

| Great job!

|=========================================================== | 82%

| If the length of the shorter vector does not divide evenly into the length of

| the longer vector, R will still apply the 'recycling' method, but will throw a

| warning to let you know something fishy might be going on.

...

|============================================================= | 84%

| Try c(1, 2, 3, 4) + c(0, 10, 100) for an example.

> c(1, 2, 3, 4) + c(0, 10, 100)

[1] 1 12 103 4

Warning message:

In c(1, 2, 3, 4) + c(0, 10, 100) :

longer object length is not a multiple of shorter object length

| Nice work!

|=============================================================== | 87%

| Before concluding this lesson, I'd like to show you a couple of time-saving

| tricks.

...

|================================================================ | 89%

| Earlier in the lesson, you computed z \* 2 + 100. Let's pretend that you made a

| mistake and that you meant to add 1000 instead of 100. You could either re-type

| the expression, or...

...

|================================================================== | 92%

| In many programming environments, the up arrow will cycle through previous

| commands. Try hitting the up arrow on your keyboard until you get to this

| command (z \* 2 + 100), then change 100 to 1000 and hit Enter. If the up arrow

| doesn't work for you, just type the corrected command.

> z\*2+1000

[1] 1002.20 1018.00 1006.28

| All that hard work is paying off!

|==================================================================== | 95%

| Finally, let's pretend you'd like to view the contents of a variable that you

| created earlier, but you can't seem to remember if you named it my\_div or

| myDiv. You could try both and see what works, or...

...

|====================================================================== | 97%

| You can type the first two letters of the variable name, then hit the Tab key

| (possibly more than once). Most programming environments will provide a list of

| variables that you've created that begin with 'my'. This is called

| auto-completion and can be quite handy when you have many variables in your

| workspace. Give it a try. (If auto-completion doesn't work for you, just type

| my\_div and press Enter.)

>

> my\_div

[1] 3.478505 3.181981 2.146460

| Keep up the great work!

|========================================================================| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes

2: No

Selection: 1

What is your email address? sabestinjackson@gmail.com

What is your assignment token? gBoicBHQE1mL4wWq

Grade submission succeeded!

| You are really on a roll!

| You've reached the end of this lesson! Returning to the main menu...| 3, 4) and c(0, 10). Don't worry about saving the result in a new variable.

> c(1,2,3,4)+c(0,10)

[1] 1 12 3 14

| Great job!

|=========================================================== | 82%

| If the length of the shorter vector does not divide evenly into the length of

| the longer vector, R will still apply the 'recycling' method, but will throw a

| warning to let you know something fishy might be going on.

...

|============================================================= | 84%

| Try c(1, 2, 3, 4) + c(0, 10, 100) for an example.

> c(1, 2, 3, 4) + c(0, 10, 100)

[1] 1 12 103 4

Warning message:

In c(1, 2, 3, 4) + c(0, 10, 100) :

longer object length is not a multiple of shorter object length

| Nice work!

|=============================================================== | 87%

| Before concluding this lesson, I'd like to show you a couple of time-saving

| tricks.

...

|================================================================ | 89%

| Earlier in the lesson, you computed z \* 2 + 100. Let's pretend that you made a

| mistake and that you meant to add 1000 instead of 100. You could either re-type

| the expression, or...

...

|================================================================== | 92%

| In many programming environments, the up arrow will cycle through previous

| commands. Try hitting the up arrow on your keyboard until you get to this

| command (z \* 2 + 100), then change 100 to 1000 and hit Enter. If the up arrow

| doesn't work for you, just type the corrected command.

> z\*2+1000

[1] 1002.20 1018.00 1006.28

| All that hard work is paying off!

|==================================================================== | 95%

| Finally, let's pretend you'd like to view the contents of a variable that you

| created earlier, but you can't seem to remember if you named it my\_div or

| myDiv. You could try both and see what works, or...

...

|====================================================================== | 97%

| You can type the first two letters of the variable name, then hit the Tab key

| (possibly more than once). Most programming environments will provide a list of

| variables that you've created that begin with 'my'. This is called

| auto-completion and can be quite handy when you have many variables in your

| workspace. Give it a try. (If auto-completion doesn't work for you, just type

| my\_div and press Enter.)

>

> my\_div

[1] 3.478505 3.181981 2.146460

| Keep up the great work!

|========================================================================| 100%

| Would you like to receive credit for completing this course on Coursera.org?

1: Yes

2: No

Selection: 1

What is your email address? sabestinjackson@gmail.com

What is your assignment token? gBoicBHQE1mL4wWq

Grade submission succeeded!

| You are really on a roll!

| You've reached the end of this lesson! Returning to the main menu...