````
---
title: 'Worksheet 1: R Basics'
author: "Will Hiott-Sample-knit-pdf-etc."
date: "1.28.2024"
---
_This is the first in a series of worksheets for History 8510 at Clemson University. The goal of these worksheets is simple: practice, practice, practice. The worksheet introduces concepts and techniques and includes prompts for you to practice in this interactive document. When you are finished, you should change the author name (above), knit your document to a pdf, and upload it to canvas._

## What is R?
To start let's define what exactly R is. R is a language and environment for statistical computing and graphics. R provides a variety of statistical and graphical techniques and its very extensible which makes it an ideal language for historians.

## Foundational Concepts
### Values

There are several kinds of variables in R. Numeric, logical, and strings.

R takes inputs and returns an output. So for example, if our input is a number the output will be a number. Simply typing a number, below or in the console, will return a number or a **numeric** value.
```{r}
5
```
Same thing happens if we use a number with a decimal.
```{r}
5.5483
```
Numbers can be used to do arithmetic.
```{r}
5 + 5
```

(@) You try, multiple two numbers.

```{r}
5 * 5
```

(@) Can you multiply two numbers and then divide by the result?

```{r}
20 * 20
```
```{r}
20 * 20 / 4
```

The next type of value is a **string**. Strings are lines of text. Sometimes these are referred to as character vectors. To create a string, you add text between two quotation marks. For example:
```{r}
"Go Tigers"
```

(@) Try to create your own string.
```{r}
"Go Lions"

```

You can't add strings using `+` like you can numbers. But there is a function called `paste()`
````

which concatenates character vectors into one. This function is _very_ useful and one you'll use a lot in a variety of circumstances. Here's what that looks like:

```{r}
paste("Hello", "Clemson Graduate Students")
```

(@) Try it, add two strings together like the above example.
```{r}
paste("Hello", "Y'all")
```

(@) Can you explain what happened in 2-3 sentences?

> Combined two statements and strung them together. Also removed quotation marks.

The last type are **logical** values like `TRUE` and `FALSE`. (Note that these are all caps.)

```{r}
TRUE
```

```{r}
FALSE
```

These logical values are really useful for testing a statement and comparing values to each other. For example:
```{r}
3 < 4
```
3 is indeed less than 4 so the return value is `TRUE`. Here are a few more examples:

R also has relational operators and logical operators. Relational operators test how one object relates to another. We've been using some of these above.

* Equality `==`
* Inequality `!=`
* Less than or Greater than `<` and `>`
* Less than/greater than or equal to `<=` and `>=`

Logical Operators allow you to combine or change the results of comparisons.
* AND `&`
* OR `|`
* NOT `!`

```{r}
5 == 10
3 < 4
5 != 10
5 != 5
3 == 4 | 3
3 != 4 & 3 != 5
```
(@) Explain, what does the code on each line above do?

>equality 5 is not equal to 10
4 is greater than 3
again true that 5 is not the same as 10
5 is that same as 5 so would be true

(@) Create your own comparison.
```{r}
10 > 9
```

Variables

Values are great but they are made so much more powerful when combined with **Variables** which are a crucial building block of any programming language. Simply put, a variable stores a value. For example if I want x to equal 5 I can do that like this:
```{r}
x <- 5
```
`<-` is known as an assignment operator. Technically, you could use `=:` here too but it is considered bad practice and can cause complicated issues when you write more advanced code. So its important to stick with `<-` whenever you are coding in R.

I can also add a string or character vector to a variable.
```{r}
x <- "Go Tigers!"
```

Variable names can be almost anything.
```{r}
MyFavoriteNumber <- 25
```
Variable or object names must start with a letter, and can only contain letters, numbers, `_`, and `.`. You want your object names to be descriptive, so you'll need a convention for multiple words. People do it different ways. I tend to use periods but there are several options:

```
i_use_snake_case
otherPeopleUseCamelCase
some.people.use.periods
And_aFew.People_RENOUNCEconvention
```

Whatever you prefer, be consistent and your future self will thank you when your code gets more complex.

(@) You try, create a variable and assign it a number:

```{r}
a <-"i.may.give.periods.a.try"
```

(@) Can you assign a string to a variable?

```{r}
x <-10
y <-20
```

Once we've assigned a variable we can use that variable to run calculations just like we did with raw numbers.
```{r}
x <- 25
x * 5
```

R can also handle more complex equations.
```{r}
(x + x)/10
(x + x * x) - 100
```

And we could store the output of a calculation in a new variable:

```{r}
My.Calculation <- (x + x)*10
```

(@) You try. Assign a number to x and a number to y. Add those two numbers together.
```{r}
x <-5
y <-5
x + y
```

(@) Can you take `x` and `y` and multiply the result by 5?
```{r}
(x + y) * 5
```

(@) Try creating two variables with names other than `x ` and `y`. Descriptive names tend to be more useful. Can you multiply the contents of your variables?
```{r}
Hardin <- 50
Hall <-50
Hardin * Hall
```

(@) Try creating two variables that store strings. Can you concatenate those two variables together?
```{r}
Tillman <- 2 + 2
Chapel <- 9 - 1
Tillman * Chapel
```

Vectors

If we have a lot of code and rely on just variables, we're going to have a lot of variables. That's where **vectors** come into play. Vectors allow you to store multiple values. All variables in R are actually already vectors. That's why when R prints an output, there is a `[1]` before it. That means there is one item in that vector.

```{r}
myvalue <- "George Washington"
myvalue
```
In this instance "George Washington" is the only item in the variable myvalue. But we could add more. To do that we use the `c()` function which combines values into a vector.
```{r}
myvalue <- c("George Washington", "Franklin Roosevelt", "John Adams")
myvalue
```
You'll notice that the output still only shows `[1]` but that doesn't mean there is only one item in the list. It simply means George Washington is the first. If we use `length()` we can determine the number of items in this vector list.
```{r}
length(myvalue)
```
We could get the value of the 2nd or 3rd item in that list like this:
```{r}
myvalue[2]
```
We could also create a vector of numbers:
```{r}
my.numbers <- c(2, 4, 6, 8)
```

And we could then do calculations based on these values:

```{r}
my.numbers * 2
```
(@) Explain in a few sentences, what happened in the code above?

>multiplied consectutively, so 2 * 2, Sum * 4, etc.

Lets try something slightly different.
```{r}
my.numbers[3] * 2
```
(@) Explain in a few sentences, what happened in the code above?

>That simply lists the 3rd numeric in the printed results as 12.

(@) You try, create a list of five items and store it in a descriptive variable.
```{r}
our.numbers <- c(1, 3, 5, 7)
our.numbers * 1
```

Built In Functions

R also has **built in functions**. We've already used a couple of these: `paste()` and `c()`. But there are others, like `sqrt()` which does what you think it does, finds the square root of a number.
```{r}
sqrt(1000)
```

Many functions have options that can be added to them. For example, the `round()` function allows you to include an option specifying how many digits to round to.

You can run it without that option and it'll use the default:
```{r}
round(15.492827349)
```

Or we can tell it to round it to 2 decimal places.
```{r}
round(15.492827349, digits = 2)
```

How would you know what options are available for each function in R? Every function and package in R comes with **documentation** or a **manual** that is built into R studio and can be pulled by by typing a question mark in front of the function in your console. These packages will commonly give you examples of how to use the function and syntax for doing so. They are incredibly useful.

We can ask R to pull up the documentation like this:
```{r}
?round()
```

(@) Now you try, find the documentation for the function `signif()`.
```{r}
?signif()
```

In real life, you typically you wouldn't want to store this code in your script file. You probably don't need to pull up the documentation for the function every time you run that piece of code. But for the purposes of this worksheet we're adding it to our `.Rmd` document.

(@) Use the console to find the documentation for `floor()`? Try it and then tell me, what does that function do?

>floor takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x.

Data Frames & Packages

R is the language of choice for most data scientists and that is because of its powerful suite of data analysis tools. Some are built into R, like `floor()` which we looked at above. But others come from packages that you have to install. Most programming languages have some sort of package system although every language calls it a slightly different thing. Ruby has gems, python has eggs, and php has libraries. In R these are called packages or libraries and they are hosted by the Comprehensive R Archive Network or more commonly, CRAN. CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. You didn't know it but you already used CRAN when you looked up the documentation for `floor()` above. If you go to the CRAN webpage and look at the [list of available R packages](https://cran.r-project.org/web/packages/available_packages_by_name.html) you'll see just how many there are!

So there are many packages but there are also some that are indispensable. You'll use some packages over and over for this class. We'll get into some of those in the next two worksheets but for now lets look at one basic package for data.

Lets start with the `tibble()` package which allows you to create and work with **data frames**. What is a data frame? Think of it as a spreadsheet. Each column can contain data - including numbers, strings, and logical values.

Lets load the `tibble()` package. If you have this package installed this line of code will work. If not, you'll get an error that says something like: `## Error in library("tibble"): there is no package called 'tibble'`. If that's the case (it probably is), then no worries - we can install it. To install a package run `install.packages("tibble")` in your console and R will download and install the package. (Remember this isn't code that we want or need to store in our document, so run it in your console not in the `.Rmd` document.)

```{r}
library(tibble)
```

The tibble package will allow you to create dataframes. Lets use some built in data in R to demonstrate this. This dataset contains the measurements in centimeters of sepal and petal width and length. Lets start by figuring out what kind of data iris is:

```{r}
typeof(iris)
```

We can see that iris is a list. But we want to use the tibble package to turn it into a dataframe. Thats very easy, we can simply run:

```{r}
as_tibble(iris)
```

That is useful. But we can also create our own data frame from vectors. That means we can create the values in each field. For example:

```{r}
tibble(
x = 1:5,
y = 1,
z = x * 2 + y
)
```

What is going on here? Each line within this function (x,y,z) creates a new column and determines the values in each row. So the first line creates five rows with values sequentially from 1 to 5. The second column contains only the value 1 in all five rows.

(@) What is going on in row z? Can you explain the logic here?

>For x there is a list of sequential numerics
For y each numeral is a one
For z is a compination of multiplication and addition so for the last line 5 * 2 + 1 = 11

Loading Dataframes from Packages

Lets also install the data package that I've created for our class. This package contains a variety of historical datasets that you can use to complete your assignments this semester. However, this package is not on CRAN. That's okay. We can still install it from GitHub. To do that we'll need to use the `devtools` package. We'll install devtools and then use it to install our class's package which is hosted on GitHub.

First, install the `devtools` package.

Our class's package is called `DigitalMethodsData` and its hosted on [GitHub](https://github.com/regan008/DigitalMethodsData). Use the directions on the repository page and what you learned about packages above to install this package.

After installing the package, we should load it:
```{r}
library("DigitalMethodsData")

```

What kind of datasets are available in this package? We can use the help documentation to find out.

Run `help(package="DigitalMethodsData")` in your console to pull up a list of datasets included in this package.

For demonstration purposes we'll use the `gayguides` data here. Pull up the help documentation for this dataset. What does the documentation tell you abou the scope of this dataset?

>The scope from the documentation listes it as one data frame with 14 variable and some 60698 observations.
gayguides {DigitalMethodsData} R Documentation
Bob Damron's Address Books
Description

[Package DigitalMethodsData version 0.0.0.1 Index

To use a dataset included in this package we first need to load it. We can do so like this:

```{r}
data(gayguides)
```

Notice that you now have a loaded dataset in your environment pane. It shows us that there are 60,698 observations (rows) of data and 14 variables (columns).

Let's now look at our data. You can use the `head()` function to return the first part of an object or dataset.
```{r}
head(gayguides)
```
That gives us the first 6 rows of data. Its really useful if you just want to peak into the dataset but don't want to print out all 60k+ rows.

(@) The default is to print six rows of data. Can you modify the above code to print out the first 10 rows?
```{r}
list(gayguides)
```

```
Maybe not but got something else with first ten list.

(@) Can you find the last 6 rows of data?
```{r}
tail(letters)
```

(@) Reflect on the previous two prompts. How did you figure these out?

>I asked the wrong question and it gave me the last letters of the alphabet.

The `str()` function is another very useful function for understanding a dataset. It compactly displays the internal structure of an R object.
```{r}
str(gayguides)
```
There is a ton of useful info here. First, we see that this is a data.frame and that it has 60698 obs. of 14 variables. Second, we see the names of all the variables (columns) in the dataset. Lastly, it shows us what type of data is contained in each variable. For example, the variable `state` is a chr or character vector while `Year` (note the capitalization) is numeric.

The `$` operator allows us to get a segment of the data.
```{r}
head(gayguides$title)
```
While that's useful, this particular dataset happens to be sorted by title. So the first rows are all labeled some version of B.A. beach. That's not very useful. What if we want to see the 500th item in this list? Well, we can pull that up like this:
```{r}
gayguides$title[500]
```

(@) Break this down. Whats going on here? What do each of the elements of this code mean? (gayguides, $, title, and [500])

>Collection, title name, and 500th entry.

(@) The Year variable contains the year (numeric) of each entry. Can you find the earliest year in this dataset? (Using code, don't cheat and use the documentation!)
```{r}
gayguides$Year[1]
```
Seens like this should be earlier???

(@) How about the latest year?
```{r}
gayguides$Year[500]
```
Was the last year 1983?

(@) Add another dataset from `DigitalMethodsData`. How many observations are included in this dataset? Run through some of the examples from above to learn about the dataset.
```{r}
data(judges)
head(judges)
str(judges)
head(judges$title)

```

(@) What did you learn about the dataset? What can you tell me about it?

>List names, birth date, state, gender, race
```

Court type, name, title, president, party,
All official type information
However, unless I missed it, does not list religion or marital status, more personal matters, but influential in casting votes, perhaps from a legal historian perspective.

CONGRATS! You've completed your first worksheet for Digital Methods II. That wasn't so bad, right?