SESSION 3.0

It’s R time to shine!

WELCOME BACK!!

- Over the last few days we've introduced you to quarto and markdown
- We've shown you quarto markdown and jupyter notebooks and how you can write code and text in the same document
- And we've taken you through some programming basics like
  - Variables (saving data under a name you can use later)
  - Data types (like `strings`, `integers`, `floats`, and `booleans`)
  - Functions (a way to `define` a set of instructions and run it multiple times)
  - Control structures (like `loops` and `if` statements)
  - Data structures (like `lists`, `dictionaries`, and `DataFrames`)
  - And more

RECAP QUIZ

- Before we get started, let’s do a quick recap quiz to see how much you remember
- This quiz is a little longer than the previous ones, but it’s still untimed
- And getting things wrong will still remind you of the correct answer
- It’s up in Day 3 on Brightspace

TODAY'S PLAN

- Today we're going to introduce you to R, but we're just going to be looking at how R does things you've already seen in Python
- We'll be looking at
  - Variables
  - Data types
  - Functions
  - Control structures
  - Data structures
- But we'll also be getting you to work in quarto markdown instead of jupyter notebooks
- And we'll be talking you through more about best practices for setting up your projects

SETTING UP YOUR PROJECT

- Over the years various researchers have come up with best practices for setting up your projects
- And they wrapped them up in a set of guidelines called the ‘Best Practices for Scientific Computing’
- But they were too big and no one followed them…
- So they were distilled down into a set of guidelines called the ‘Good Enough Practices for Scientific Computing’ (Wilson 2017)

GOOD ENOUGH PRACTICES FOR SCIENTIFIC COMPUTING

- I really suggest you read the paper (it's up on Brightspace)
- There's a lot in it that may not be relevant to you now, but the principles are good to know
- The main points are
  - Have a clear project structure (set up your folders and files in a way that makes sense)
  - Write code for humans, not computers (use comments and variable names that make sense)
  - Automate repetitive tasks (use functions and loops)
  - Use version control (there's tools like git that can help you keep track of changes to your writing and code)
  - Document everything (write down what you did and why you did it)
  - Collaborate (work with others and share your work)

GOOD ENOUGH PRACTICES FOR SCIENTIFIC COMPUTING

- We can't cover all those points, and honestly, you'll learn them better by doing them and making mistakes
- But things like setting up your projects in a way that makes sense, and writing code for humans are things we can help you with
- Before jumping into R, we're going to talk about how to set up your projects in a way that makes sense
- This means not only setting up your folders and files in a way that makes sense, but also setting up your code thoughtfully
- Installing the things you'll need to run your R code
- But first, lets practice setting up your folders

SETTING UP YOUR FOLDERS

- When you start a new project, you should set up a folder for it, and then set up subfolders for different parts of your project
- For example, you might have a folder for your data, a folder for your code, a folder for your results, and a folder for your writing
- This makes it easier for you to work with things like `paths` and `relative paths` within and across your projects
  - It also makes it easier for you to share your work with others
  - We're going to get you to set up a folder structure for your project now
- Using what you learned about the `explorer` pane on the left of VSCode, create a new folder called "Day 3" in the folder you've been working in all week

SETTING UP YOUR FOLDERS

- Inside the “Day 3” folder, create the following subfolders
  - “data”
  - “src”
  - “output”
- Copy the movies_df.csv file from yesterday into the “data” folder
- And in the “src” folder, create a new file called “intro.qmd”

INSTALLING R

- Before we can start working with R, we need to install it
- You can download R from the CRAN website (https://cran.r-project.org/)
- We've put that link up on Brightspace
- Before you open VSCode, install R on your computer
- As you install you'll be asked if you want to install Rtools, you should say yes
- You'll be asked for admin permissions, so you'll need to call me over when you get there
- Once thats done, open VScode, in the terminal type `R --version` and hit enter
- You should see the version of R you installed

SETTING UP YOUR CODE

- In python we told you that there are packages/libraries/modules that you can use to 'go beyond' the basic functionality of python
- These are things like `pathlib`, `pandas`, `numpy`, `matplotlib`
- And you've learned that we install them from the terminal using `pip install package_name`
- The same thing exists in `R`, but they're called `packages` or `libraries`
- In both languages you need to install the packages before you can `import` them into your code
- In R, you install packages using the `install.packages()` function, but not in the general terminal, in the R terminal

THE R TERMINAL

- On Day 0 we showed you the terminal in VSCode, and you've used it a few times
- This is a place where you can run code (commands) directly
- On the computers you're using the terminal is running something called PowerShell by default
- This is the windows 'shell'
- If you were on a mac, it would be running something called `bash` which is the mac 'shell'
- you can also run `python` and `R` code in the terminal by typing `python` or `R` and hitting enter

THE R TERMINAL

- In VSCodes, use your mouse to make the terminal bigger
- Then type R and hit enter and watch how the terminal changes
- You should see something like this

R terminal

THE R TERMINAL

- You actually already know some R code
- Try typing `print("Hello World!")` and hitting enter in the terminal
- You should see `Hello World!` printed out with a `[1]` in front of it
- R and Python are very similar in a lot of ways and you'll see that as we go through the day
- As you can see from the instructions on the screen, you can type `q()` to quit the R terminal
- It will ask if you want to save your workspace image, just type `y` and hit enter
- So try quitting the R terminal, then type `R` and hit enter to start it again and print `Hello World!` again

INSTALLING PACKAGES IN R

- To install packages in R, you need the R terminal open because installing packages is a command you have to run in R directly.
- To install a package in R, you use the `install.packages()` function
- And you pass the name of the package you want to install as a `string` to the function
- In order to work with quarto and R you need to install the `quarto` package (which is different from the `quarto` software)
- So in the R terminal, type `install.packages("quarto")` and hit enter
  - This will install the quarto package on your computer, whatch the terminal to see what happens
- You should also install the 'rmarkdown' package by typing `install.packages("rmarkdown")` and hitting enter

INSTALLING PACKAGES IN R

- Ok, you now have the packages installed that we need for this session, but there are others
  - 'tidyverse' (a collection of packages that make working with data easier)
  - 'ggplot2' (a package for making plots)
  - 'psych' (a package for doing some basic statistics)
  - 'lavaan' (a package for doing structural equation modeling)
- Depending on your project you'll use some or all of these, we'll look at some of them throughout the day.
- But for now you can `quit` the R terminal by typing `q()` and hitting enter (and saving your workspace image)

SETTING UP YOUR `.qmd` FILE

- Now that you've installed the quarto and rmarkdown packages, you can start working in R
- Open the "intro.qmd" file you created in the "src" folder
- Remember that we can include `yaml`, `markdown`, and code in a .qmd file
- We're going to start by setting a simple yaml header

````
```{yaml}
---
title: "intro to R"
author: "Your Name"
engine: knitr
---
```
````

SETTING UP YOUR `.qmd` FILE

- This is a really simple yaml header
- The `title` is the title of your document
- The `author` is your name
- The `engine` is the engine that will run your code
- We're using `knitr` because it's the engine that will run R code
- If we were using python code we would use `jupyter` as the engine
- Now I want you to add some markdown to your file starting with a level 1 heading
- Then write a short note about how you feel about learning R

MARKDOWN RECAP

- We took a good look at markdown on Day 0
- And you've answered questions on it in most of the quizzes
- But just to remind you
  - You can use `#` to create headings
  - You can use `-` to create lists
  - You can use `---` to create horizontal rules and to open and close the yaml header
  - You can use `![alt text](path/to/image)` to include images
  - And you use three backticks followed by{languagename} to include code blocks
- In this session you're going to use headings, plain markdown and code blocks

MARKDOWN EXAMPLE

````
```{markdown}

# How I feel about learning R

Nervous, but excited to never open SPSS again

```{{r}}
#| echo: true
#| eval: true
# This is a cell within a cell, cell-ception!
print("Hello World!")
```
```
````

MAKE NOTES

- Before we get into the R code
- Take a few minutes to make some notes in your “intro.qmd” file
- Make a new section with a level 1 heading and list what we did so far
    1. Set up a folder structure (Day 3, data, src, output)
    2. Installed the quarto and rmarkdown packages in R
    a. Opened the R terminal
    b. Installed the quarto package with `install.packages("quarto")`
    c. Installed the rmarkdown package with `install.packages("rmarkdown")`
    3. Set up the “intro.qmd” file in the “src” folder
    a. Added a yaml header
    b. Added a level 1 heading
- Save your file and we’ll move on to the R code

R CODE

- R is a `statistical programming language` that is used by a lot of researchers
- It's a little different than python, but it's also very similar
- Where python is a general purpose programming language, R is a language that was *designed* for working with data
- It's good at other stuff, but the underlying structure of the language is built around working with data
- For some people, this makes R *feel* easier to work with than python
- I'm not one of those people, but I can see where they're coming from
- Everything in R builds towards working with data, and the assumptions under it bend in that direction

R COMMUNITY

- R has a really strong community of users
- Specifically R has a really strong community of `data scientists` and `researchers`
- There are online groups like `Stack Overflow` and `Cross Validated` where you can ask questions
- And there are more specialized groups like `R-bloggers` and `R Weekly` that share tips and tricks
- And there are groups like `R-Ladies` and `RLadies Global` that work to increase the participation in R of underrepresented groups

R CODE

- It has a lot of packages that make working with data easier
- And we're going to look at some of them today, but…
- We're going to start by looking at some of the basics of R
- And we're going to do that by showing you how to do things in R that you've already seen in python

WHAT YOU ALREADY KNOW IN R

- Believe it or not, learning some python has already taught you a lot about R
- At the general level, you know about variables, data types, functions, control structures, and data structures
- And python and R handle these things in very similar ways
- So we're going to go through some examples of how you can do things in R that you've already seen in python
- The first thing to note is that comments in R are made with a # just like in python
- And you can use # to comment out code in R just like in python
- Comments really matter!! Don't skimp on them!!

CHARACTERS IN R

- In R, you can create `strings` by wrapping text in either single or double quotes
- So the code:
- `"Hello World!"` and `'Hello World!'`
- Will both create a string with the text `Hello World!`
- And the same reasons for using single or double quotes in python apply in R
- But R calls this type of data `character` instead of `string`

NUMBERS IN R

- R also has `integers` and `floats` like python
- But R just calls both of these `numeric`
- So the code:
  - `1` and `1.0`
  - Will both create a numeric data type with the value `1`

PRINT() IN R

- In R, you can print things to the console using the `print()` function
- So the code:
- `print("Hello World!")`
- Will print `Hello World!` to the console
- So you see that functions in R are called the same way as in python

VARIABLES IN R

- Just as we can assign some data to a variable in python, we can do the same in R
- There's actually 2 ways to do this:
  - `variable = data` (just like in python)
  - `variable <- data`(this is the preferred way in R)
- So the code:
  - `x <- "Hello World!"`
  - Will assign the value `Hello World!` to the variable `x`
- And the code:
  - `print(x)`
  - Will print `Hello World!` to the console

FIRST PRACTICE

- In your “intro.qmd” file, add a new section with a level 1 heading
- Add a code block with the following code
  - `x <- "Hello World!"`
  - `print(x)`
- Save your file and then we’ll try more of the Day 0 quarto commands by getting you to preview your file

PREVIEWING YOUR FILE

- To preview your file, you can use the `quarto preview` command in the terminal
- But not in the R terminal, in the standard terminal
- So from the menu at the top of the screen, select `Terminal` and then `New Terminal`
- This will open a new terminal window at the bottom of the screen
- You'll see the `directory` you're in and a blinking cursor
- You can right click on "intro.qmd" in the explorer pane and select `Copy Path`
- Then type `quarto preview` and paste the path to your file and hit enter
- Cross all the fingers and toes and watch what happens

PREVIEWING YOUR FILE

- If you see the preview of your file in your browser, then you're good to go
- If you see an error message, then you'll need to read it and we'll help you fix
- Just start by making sure that your code looks like this

````
```{r}
# Making a string
x <- "Hello World!"
# Printing the string
print(x)
```
````

PREVIEWING YOUR FILE

- You can leave that preview running and it will update everytime you save your file.
- The preview won't have all the formatting that you'll see when you full render the file
- But it's a good way to check that your code is working and that your general formatting is correct
- Add a new section to the file, maybe with a level 2 heading about previewing your file
  - Remember that the command is `quarto preview` and then the path to your file (you can copy the path from the explorer pane)
- Save your file, take a look at the preview, and then we'll move on to data types

Oh!

MORE WITH `characters`

- We can `concatenate characters` in R, but it's a little different than in python
- In R, you use the `paste()` function to concatenate `characters`
- Like so:

```r
# Making some characters
x <- "Hello"
y <- "World!"
# Concatenating the characters
z <- paste(x, y)
# Printing the concatenated characters
print(z)
```

```
[1] "Hello World!"
```

PASTE() IN R

- The `paste()` function in R is a little different than the `+` operator in python
- In R, you can pass multiple `characters` to the `paste()` function and it will concatenate them
- You can also pass a `sep` argument to the `paste()` function to specify what you want to separate the strings with
- So the code:
  - `z <- paste(x, y, sep = " ")`
  - Will concatenate the strings `x` and `y` with a space between them
- And the code:
  - `print(z)`
  - Will print `Hello World!` to the console

MORE PRACTICE

- You may have also noticed that we assigned the result of the `paste()` function to a variable
- In your "intro.qmd" file, you could add a new code block that gets the computer to say hi to you instead of the whole world

```
# Save a perfectly normal name
name <- "Grampton St. Rumpterfrabble"
# Concatenate the greeting
greeting <- paste("Hello", name, sep = " ")
print(greeting)
```

```
[1] "Hello Grampton St. Rumpterfrabble"
```

- You may notice that this is a little less flexible that the `f-string` in python
- But it's still a good way to work with strings

DATA TYPES IN R

- We've seen that R has `numeric` and `character` data types
- But theres others
- And we can see what type of data we have by using the `class()` function

```
1 # printing some classed
2 print(class(1))
```

```
[1] "numeric"
```

```
1 print(class(1.0))
```

```
[1] "numeric"
```

```
1 print(class("Hello World!"))
```

```
[1] "character"
```

```
1 print(class(TRUE))
```

```
[1] "logical"
```

DATA TYPES IN R

- There are also things like `lists` and `dataframes` in R
- The R list is similar to the python list, we just make it with `list()` instead of `[]`
- But you rarely see the R list in practice
- R uses something called a `vector`, which is like a list but it can only contain one type of data

```
# Making a list
my_list <- list(1, 2, 3, "Hello World!", TRUE)
# Making a vector
my_vector <- c(1, 2, 3, 4, 5)

print(class(my_list))
```

```
[1] "list"
```

```
print(class(my_vector))
```

```
[1] "numeric"
```

VECTORS

- The `vector` is much more common in R than the `list`
- It does a lot more than the `list`, even though it's a little more restrictive
- The fact that the vector can only contain one type of data is actually a good thing
- Think of a column in a data table, it can only contain one type of data
- Vectors form the basis of a lot of data structures in R
- They're used a lot in R so it's good to know about them

INDEXING VECTORS

- Vectors are made using the `c()` function, and we pass the data we want to put in the vector to the function
- We can also access elements in a vector in R using the same square brackets as in python
- This is just like slicing in python but....
- In R, the first element in a vector is element 1, not element 0 (R is a 1-indexed language)

```r
# Accessing elements in a vector
# making a vector
my_v <- c("a", "b", "c", "d", "e")
# accessing the first element
print(my_v[1])
```

```
[1] "a"
```

```r
# accessing the last element
print(my_v[5])
```

```
[1] "e"
```

INDEXING VECTORS

- You can also access multiple elements in a vector by passing a vector of indexes to the square brackets
- So the code:
  - `print(my_v[c(1, 3, 5)])`
  - Will print the first, third, and fifth elements of the vector `my_v`

INDEXING VECTORS

- But negative indexes in R are a little different than in python
- In R, a negative index will remove the element at that index
- So the code:
  - `print(my_v[-1])`
  - Will print all the elements of the vector `my_v` *except* the first element
- And the code:
  - `print(my_v[-c(1, 3, 5)])`
  - Will print all the elements of the vector `my_v` *except* the first, third, and fifth elements

SLICING VECTORS

- You can also slice vectors in R
- Just like in python, you can use the colon `:` to slice a vector

```
1  # Slicing a vector
2  print(my_v[1:3])
```

```
[1] "a" "b" "c"
```

`length()` IN R

- You can also get the length of a vector in R using the `length()` function
- So the code:
  - `print(length(my_v))`
  - Will print the length of the vector `my_v` to the console
- We can use this to get the last element of a vector

```
1 # Getting the last element of a vector
2 print(my_v[length(my_v)])
```

```
[1] "e"
```

PRACTICE WITH VECTORS

- In your “intro.qmd” file, add a new section with a level 1 heading
- Add a code block with the following code
  - `my_v <- c("a", "b", "c", "d", "e")`
  - `print(my_v[1])`
  - `print(my_v[5])`
  - `print(my_v[c(1, 3, 5)])`
  - `print(my_v[-1])`
  - `print(my_v[-c(1, 3, 5)])`
  - `print(my_v[1:3])`
  - `print(my_v[length(my_v)])`

NAMED VECTORS

- R doesn't have a dictionary data type like python
- But it does have something called a `named vector`
- This is a vector where each element has a name so it's really similar to a dictionary

```r
# Making a named vector
my_named_vector <- c("a" = 1, "b" = 2, "c" = 3, "d" = 4, "e" = 5)
# Accessing elements in a named vector
print(my_named_vector["a"])
```

```
a
1
```

```r
print(my_named_vector)
```

```
a b c d e
1 2 3 4 5
```

NAMED VECTORS

- You'll notice that the name and the value in the named vector are separated by an `=` rather than a `:`
- But other than that we can still think about them like `key-value` pairs
- And we can access the values in the named vector using the names.
- So the code:
  - `print(my_named_vector["a"])`
  - Will print the value `1` to the console
- And the code:
  - `print(my_named_vector)`
  - Will print the whole named vector to the console

NAMING A VECTOR

- We use the `c()` function to make both vectors and named vectors
- But we can also turn a vector into a named vector using the `names()` function
- The `syntax` for this is a little different to python but it's not too hard to get used to

```r
# Making a vector
my_v <- c(1, 2, 3, 4, 5)
# Naming the vector
names(my_v) <- c("a", "b", "c", "d", "e")
# Accessing elements in a named vector
print(my_v["a"])
```

```
a
1
```

```r
print(my_v["e"])
```

```
e
5
```

NAMING A VECTOR

- You'll notice that we use the `names()` function to name the vector
- Passing the vector that we want named as an argument to the `names()` function
- Then we use the `assignment` operator `<-`
- Then we create a vector of names that we want to use
- Another example might be if we had a vector of ages and we wanted to name them

```r
# Creating a vector
scores <- c(95, 85, 75)
# Assigning names to the vector elements
names(scores) <- c("Alice", "Bob", "Charlie")
# Printing the named vector
print(scores)
```

```
  Alice     Bob Charlie
     95      85      75
```

```r
# Accessing elements by name
print(scores["Alice"])
```

```
Alice
   95
```

NAMED VECTORS

- You can also access multiple elements in a named vector by passing a vector of names to the square brackets
- So the code:
  - `print(my_v[c("a", "c", "e")])`
  - Will print the first, third, and fifth elements of the named vector `my_v`

PRACTICE WITH NAMED VECTORS

- In your “intro.qmd” file, add a new section with a level 1 heading
- Add a code block with the following code
  - `my_named_vector <- c("a" = 1, "b" = 2, "c" = 3, "d" = 4, "e" = 5)`
  - `print(my_named_vector["a"])`
  - `print(my_named_vector)`
  - `my_v <- c(1, 2, 3, 4, 5)`
  - `names(my_v) <- c("a", "b", "c", "d", "e")`
  - `print(my_v["a"])`
  - `print(my_v["e"])`

PRACTICE WITH NAMED VECTORS

- Then make another code block with the following code
  - `scores <- c(95, 85, 75)`
  - `names(scores) <- c("Alice", "Bob", "Charlie")`
  - `print(scores)`
  - `print(scores["Alice"])`

MULTIDIMENSIONAL DATA

- We mentioned earlier that R is a language that was designed for working with data
- This is apparent when we look at how R builds up to working with `dataframes`
- The next data structure we're going to look at is the `matrix` which is like a 2D vector
- But what does that mean?

DATA IN ONE DIMENSION

- A vector is a 1D data structure
- Like a single columns or row in a table
- We only ‘read’ the data in one direction depending on how we’re thinking about it.
- For example, if we want to know the mean age, we add up all the ages in the column and divide by the number of people
- But no other information is needed
- Similarly, if we want to know everything we have on participant 3, we just look at the third row of the table

DATA IN TWO DIMENSIONS

- A matrix is a 2D data structure
- Like a table, it has height and width so to speak
- Can can think about them as stacking vectors on top of each other
- Or lining them up next to each other (if we're thinking about columns)
- This is much more like what you're used to seeing in a spreadsheet
- But if you think back to our pandas example, where we made a lot of lists, had to zip them into dictionaries, and *then* made a dataframe
- You can see that having a 2D data structure is a lot easier

MAKING A MATRIX

- Like vectors, a matrix can only contain one type of data
- So we can't have words and numbers in the same matrix, which again, may seem like a problem but it's really not, it's a feature
- We can make a matrix in R multiple ways:
  - `matrix()` function
  - `cbind()` and `rbind()` functions
- Lets start with the `cbind()` function

MAKING A MATRIX WITH CBIND()

- The `cbind()` function in R is used to combine vectors into a matrix by standing them next to each other
- So we're 'column' binding the vectors together

```r
# making 3 vectors
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)
v3 <- c(7, 8, 9)
# making a matrix with cbing
m <- cbind(v1, v2, v3)
# printing the matrix
print(m)
```

```
     v1 v2 v3
[1,]  1  4  7
[2,]  2  5  8
[3,]  3  6  9
```

MAKING A MATRIX WITH RBIND()

- The `rbind()` function in R is used to combine vectors into a matrix by stacking them on top of each other
- So we're 'row' binding the vectors together

```r
# making 3 vectors
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)
v3 <- c(7, 8, 9)
# making a matrix with rbind
m <- rbind(v1, v2, v3)
# printing the matrix
print(m)
```

```
   [,1] [,2] [,3]
v1    1    2    3
v2    4    5    6
v3    7    8    9
```

BINDING VECTORS

- So we now know that we can bind together vectors to make a matrix
- And we can do so either by stacking them on top of each other or by standing them next to each other
- Thinking as rows and columns
- But we can also just make a matrix directly using the `matrix()` function
- The `matrix()` function takes a vector and the number of rows and columns we want in the matrix

MAKING A MATRIX WITH MATRIX()

- The `matrix()` function in R is used to create a matrix from a vector
- The `matrix()` function takes a vector and the number of rows and columns we want in the matrix
- So the code:
  - `m <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3)`
  - Will create a 3x3 matrix with the numbers 1 to 9 in it

```r
# making a matrix with the matrix function
m <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3)
# printing the matrix
print(m)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

INDEXING MATRICES

- Just like with vectors, we can access elements in a matrix using the square brackets
- But we need to pass two indexes to the square brackets
- The first index is the row we want to access and the second index is the column we want to access
- So the code:
  - `print(m[1, 1])`
  - Will print the element in the first row and first column of the matrix `m`
- And the code:
  - `print(m[c(1, 3), c(1, 3)])`
  - Will print the elements in the first and third rows and the first and third columns of the matrix `m`

INDEXING MATRICES

```
1 # Accessing elements in a matrix
2 print(m[1, 1])
```

```
[1] 1
```

```
1 print(m[c(1, 3), c(1, 3)])
```

```
     [,1] [,2]
[1,]    1    7
[2,]    3    9
```

PRACTICE WITH MATRICES

- In your “intro.qmd” file, add a new section with a level 1 heading
- Add a code block where you make a matrix with the numbers 1 to 9 and 3 rows and 3 columns
- Choose a method to make the matrix and then print it
- Then make another cell and try a different method to make the matrix and print it

OPERATIONS WITH MATRICES AND VECTORS

- One of the things that makes R so powerful is that it can do operations on matrices and vectors
- So we can add, subtract, multiply, and divide matrices and vectors
- And we can do this element-wise meaning that we can take the first element of one matrix and add it to the first element of another matrix

OPERATIONS WITH MATRICES AND VECTORS

```r
# Vector operations
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)

# Multiplying vectors
print(v1 * v2)
```

```
[1]  4 10 18
```

```r
# Adding vectors
print(v1 + v2)
```

```
[1] 5 7 9
```

```r
v3 = v1 + v2
print(v3)
```

```
[1] 5 7 9
```

NAMING MATRICES

- Just like with vectors, we can name the rows and columns of a matrix
- We can do this using the `rownames()` and `colnames()` functions

```r
# Naming the rows and columns of a matrix
rownames(m) <- c("row1", "row2", "row3")
colnames(m) <- c("score1", "score2", "score3")
# Printing the matrix
print(m)
```

```
     score1 score2 score3
row1      1      4      7
row2      2      5      8
row3      3      6      9
```

NAMING MATRICES

- You'll notice that we use the `rownames()` and `colnames()` functions to name the rows and columns of the matrix
- We pass the matrix we want to name as an argument to the function
- Then we use the `assignment` operator `<-`
- Then we create a vector of names that we want to use
- And we can access the elements in the matrix using the names

FUNCTIONS AND ASSIGNMENT

- You'll have seen by now that some of the functions, like `rownames()` and `colnames()` and `names()` work differently in R than in python
- In python we would `return` the result of the function and assign it to a variable
- But when working with the attributes of a dataframe we use the = operator to assign the output to that attribute
- Like when we renamed the columns of a dataframe in pandas
- `df.columns = ["new_name1", "new_name2"]`
- This is the same in R, we use the assignment operator `<-` in the same way
- But the rownames and colnames are functions that give us direct access to setting the row and column names
- Like the `attributes` are wrapped up in a function that lets us change them

DATAFRAMES

- The `dataframe` is the most common data structure in R
- It's like a matrix but it can contain different types of data
- It's like a table in a spreadsheet
- It's more useful to social scientists than a matrix because it can contain different types of data
- And it's what we use to work with data in R
- We can make a dataframe in R using the `data.frame()` function
- The `data.frame()` function takes vectors and combines them into a dataframe

MAKING A DATAFRAME

- The `data.frame()` function in R is used to create a dataframe from vectors
- The `data.frame()` function takes vectors and combines them into a dataframe

```r
# Making a dataframe
df <- data.frame(name = c("Alice", "Bob", "Charlie"), age = c(25, 30, 35), is_human = c(TRUE, TRUE,
# Printing the dataframe
print(df)
```

```
     name age is_human
1   Alice  25     TRUE
2     Bob  30     TRUE
3 Charlie  35    FALSE
```

MAKING A DATAFRAME

- You'll notice that we use the `data.frame()` function to make the dataframe
- We pass the vectors that we want to combine into the dataframe to the `data.frame()` function
- And we use the assignment operator `<-` to assign the dataframe to a variable
- And we can access the elements in the dataframe using the names

INDEXING DATAFRAMES

- Just like with matrices, we can access elements in a dataframe using the square brackets
- But we need to pass two indexes to the square brackets

```
# Accessing elements in a dataframe
print(df[1, 1])
```

```
[1] "Alice"
```

```
print(df[1, "name"])
```

```
[1] "Alice"
```

```
print(df["name"])
```

```
     name
1   Alice
2     Bob
3 Charlie
```

SAVING DATA

- We can save data in R using the `write.csv()` function
- The `write.csv()` function takes a data, like a matrix or a dataframe and a file path as arguments
- And it writes the dataframe to a csv file at the file path

```
# Saving a dataframe
write.csv(df, file = "data/df.csv")
```

PRACTICE WITH DATAFRAMES

- In your “intro.qmd” file, add a new section with a level 1 heading
- Add a code block where you make a dataframe with the names of the participants, their ages, and whether they’re human
- Keep the dataframe small, maybe 3 rows and 3 columns
- Then print the dataframe
- Then make another code block where you save the dataframe to a csv file in the “data” folder you made earlier.

PRACTICE WITH DATAFRAMES

- Take your time with this, it's a lot of new stuff
- In the next section we're going to look at some other elements of coding in R
  - Like loops and conditionals (if statements) and functions
  - And we're going to look at some of the packages that make working with data in R easier
- But for now, just try to get the dataframe made and saved to a csv file

REFERENCES

Wilson, Jennifer AND Cranston, Greg AND Bryan. 2017. “Good Enough Practices in Scientific Computing.” *PLOS Computational Biology* 13 (6): 1–20. https://doi.org/10.1371/journal.pcbi.1005510.