For-Loops
What is a For-Loop?

- A for-loop lets you run the same piece of code many times
- Prevents you from having to copy-and-paste your code
- It’s also less error prone than rewriting your code multiple times!
For-Loop Example: Printing Numbers

- Let’s say that you want to print the numbers from 1-5 in R
For-Loop Example: Printing Numbers

- Let’s say that you want to print the numbers from 1-5 in R
- Without for-loops, you’d have to do something like this:

```r
print(1)
print(2)
print(3)
print(4)
print(5)
```
For-Loop Example: Printing Numbers

- Let’s say that you want to print the numbers from 1-5 in R
- With for-loops, you can simply do:

```r
for (i in 1:5) {
  print(i)
}
```
For-Loop Example: Printing Numbers

- `1:5` creates a list of numbers from 1 to 5, similar to `c(1, 2, 3, 4, 5)`
- The for-loop creates a variable `i` and iterates across each value from 1 to 5, setting `i` to each of these values and then printing it

```r
for (i in 1:5) {
    print(i)
}
```
For-Loop Example: Printing Numbers

- Now let’s try another example. Let’s use a for-loop to find the sum of the numbers from 1 to 1000
For-Loop Example: Summing Numbers

- The code creates a variable `sum` with an initial value of 0.
- The for-loop then adds the values from 1 to 1000 to `sum`

```
sum <- 0
for (i in 1:1000) {
    sum <- sum + i
}
```
For-Loop Example: Summing Numbers

- Make sure that you create the variable sum outside of the for-loop! Why would the example below not work?

```r
for (i in 1:1000) {
    sum <- 0
    sum <- sum + i
}
```
For-Loop Example: Summing Numbers

- If you have `sum <- 0` inside of the for-loop, the value of `sum` will be reset to 0 on each iteration of the for-loop.
- The end result would be $0 + 1000 = 1000$

```r
for (i in 1:1000) {
    sum <- 0
    sum += i
}
```
For-Loop Example: Non-Numerics

- You can also use for-loops on lists of non-numeric values

```r
for (name in c("Chelsea", "Bruce", "Ashley")) {
  print(name)
}
```

- This code will print “Chelsea”, “Bruce”, and “Ashley”, in that order
Conditionals
What is a Conditional?

A conditional is an “if... else...” statement. In computer science, conditionals are often represented as if-else clauses:

```plaintext
if(Something is true) {
    # Something happens
} else {
    # Something else happens
}
```
If-Statement Example

x <- 5

if(x > 3) {
    print("X is greater than 3");
}


x <- 5

if(x > 3) {
    print("X is greater than 3");
} else {
    print("X is less than or equal 3");
}
If-Else Example

```c
if(x > 3) {
    print("X is greater than 3");
}
else if(x == 3) {
    print("X is equal to 3");
}
else {
    print("X is less than to 3");
}
```
Conditions in R

- `if(true)`
- `if(x >= 2)`
- `if(x == 2 && y > 5)`
- `if(isRaining || temperature > 80)`
- `if(str == "CS100 Is the Best")`
Conditions in R

- if(true)
  - This is always true
- if(x >= 2)
  - X is greater or equal to 2
- if(x == 2 && y > 5)
  - If x is equal to 2 AND y > 5
- if(isRaining || temperature > 80)
  - If isRaining = true OR temperature is greater than 80
- if(x == “CS100”)
  - X is the same string, “CS100”.


Functions
What is a function?

A function is a named section of a program that does a specific task. We’ve already seen some built-in functions that R has.

sort()

print()

mean()

today()
Function Example

printHelloNina <- function() {
    print("Nina is the Best!")
}

Function Example

printHelloAlex <- function() {
    print("Alex is REALLY the Best!")
}

Can We Use Abstraction?

printHelloAlex <- function() {
    print("Alex is REALLY the Best!");
}

printHelloNina <- function() {
    print("Nina is the Best!");
}

These functions are REALLY similar, can we generalize them?

In computer science, writing general code that can be re-used is called abstraction!
Abstract Function

printHelloTA <- function(name) {
    print(paste(name, "is REALLY the Best!"));
}

Abstraction is a really important concept in Computer Science. You don’t want to copy-and-paste code. You want to write good code that can be reused.

In this case, printHelloTA can be called with any TA name- “Will”, “Anna”, “Joon”, “Ben”, “Erin”, etc.
More Functions!

Remember Studio 4?

accuracy <- function(predictions, ground_truth) {
  mean(predictions == ground_truth)
}

predictions and ground_truth are parameters (a.k.a. arguments) being sent to the function.

Inside accuracy, the built-in function mean() is being called to evaluate the average of how many times the predicted titanic survival matches the actual ground_truth.
Function With Console Input

```
enterString <- function() {
    n <- readline(prompt = "Enter: ")
    return(n)
}

print(paste("Hello,", enterString(), "!"))
```
Function With Console Input

```r
enterString <- function() {
  n <- readline(prompt="Enter: ")
  return(n)
}

print(paste("Hello, ", enterString(), ", !"))

readline(prompt="Enter: ") is a built-in function that gets input from the console and prints the prompt.
```
Function With Console Input

```r
enterString <- function() {
    n <- readline(prompt="Enter: ")
    return(n)
}

print(paste("Hello, ", enterString(), "!"))

readline(prompt="Enter: ") is a named parameter. If you are passing a lot of parameters to a function, names parameters makes your code easier to understand.
```
Function With Console Input

```r
enterString <- function() {

    n <- readline(prompt="Enter: ")

    return(n)  # n is a local variable
}

print(paste("Hello, ", enterString(), "!"))
```

return(n) has the function return a value when it is called. Since n is a local variable it can’t be referenced outside the enterString function.
Function With Console Input

enterString <- function() {
  n <- readline(prompt="Enter: ")
  return(n) # n is a local variable
}

print(paste("Hello, ", enterString(), ")

paste() is another built-in R function that combines strings.
So paste("Hello, ", "Alex") = Hello, Alex
You Can Also Combine Functions!

`printHelloTA(enterString())`

What do you think will happen?
Choose Your Own Adventure in R?

print("You meet a monster in the CIT. Do you run away or fight? (run away/fight)")

option <- enterString()

if(option == "run away") {
  print("You run away! Ahh!")
  for (i in 1:5) {
    print("AHHH!!!")
  }
}
Choose Your Own Adventure in R?

} else if( option == "fight") {

    print("You fight! Hi-yah!")

} else {

    print("This isn’t even an option!")

}
Choose Your Own Adventure!

For the rest of class, create an R script with your own choose-your-own adventure story using for loops, conditionals and functions.

You must have at least:

- Two for-loops
- Two sets of if-statements
- Three different custom functions

Email your script to cs0100tas@cs.brown.edu when you’re finished!
Happy Coding!