Plan for the week

● M: Functions and Conditionals (*if* and *if else*)
● W: Loops (*for* and *while*)
● F: Section
  ○ Programming practice
Programming Basics I
Functions
What is a function?

In spreadsheets, we learned about formulas, like \texttt{AVERAGE}, \texttt{MAX}, \texttt{CONCATENATE}, and \texttt{VLOOKUP}.

In R, we learned about \texttt{dplyr}, which has functions, like \texttt{select}, \texttt{filter}, \texttt{arrange}, \texttt{mutate}, and \texttt{group\_by}.

These are all "built-in" functions, meaning they have been written for you.

But you can also write your own functions, to perform tasks unique to your data.
printHelloErin <- function() {
    print("Erin is the best!")
}

Example
Example

```
printHelloErin <- function() {
  print("Erin is the best!")
}
```

You declare a function by writing `printHelloErin <- function()`.

Inside the `{}` brackets, you write code that defines what your function does.

This `printHelloErin` function calls the built-in function `print()`. 

Assignment operator

Declaring your function

Defining your function
Another example

printHelloAnna <- function() {
  print("Anna is REALLY the best!")
}

Abstraction

printHelloErin <- function() {
  print("Erin is the best!");
}

printHelloAnna <- function() {
  print("Anna is REALLY the best!");
}

These functions are REALLY similar.

Can we generalize somehow?

Generalizing code for easy reuse is called abstraction!
Abstraction

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

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

Abstraction is a really important concept in programming. You never want to copy-and-paste code. You want to write general code that can be reused.

name is a parameter, a.k.a. an argument, on which the function depends.

printHelloTA can be called with any string as an input: e.g., printHelloTA("Erin") or printHelloTA("Anna")
Abstraction

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

> paste("Hello, ", "Amy")
Hello, Amy

> printHelloTA("Anna")
Anna is REALLY the best!
Input from the console

```python
readline(prompt = "Enter Name: ")
```

`readline` is a built-in function that gets input from the console and displays it. `prompt = "Enter Name: "` is a named parameter of `readline`. When `enterName()` is called, your console displays the prompt, which in this case we defined as `Enter Name:`, after which you can enter any string.
Input from the console

```javascript
enterName <- function() {
  name <- readline(prompt = "Enter Name: ")
  return(name)
}

> enterName()
Enter Name: Juho
[1] "Juho"
```
Input from the console

```r
enterName <- function() {
  name <- readline(prompt = "Enter Name: ")
  return(name)
}

Run the following in an R Script or R Markdown (not the console)

```r
print(paste("Hello, ", enterName(), "!"))
```

Enter Sarah from the console, and R will display Hello, Sarah!
Function Composition

> printHelloTA(enterName())

What do you think this composition of function calls will do?

Pro-Tip: Always use informative names for functions and variables!
Function Composition

> printHelloTA(enterName())
Enter Name: Will

[1] Will is REALLY the Best!
Local and Global Variables

printNumber <- function() {
  var1 <- 1
  print(var1)
}

var1 is a local variable, a variable declared inside a function. Because it is "local" to the function, it does not exist outside the function.

> print(var1)
[1] Error: object 'var1' not found
> printNumber()
[1] 1
Local and Global Variables

```r
var2 <- 2
printNumber <- function() {
  print(var2)
}
```

`var2` is a **global** variable, a variable declared outside a function. Because it is "global," it can be accessed both inside and outside the function.

```r
> print(var2)
[1] 2
> printNumber()
[1] 2
```
Local and Global Variables

```r
enterName <- function() {
  name <- readline(prompt = "Enter Name: ")
  return(name)
}

name is a local variable, so it cannot be accessed outside enterName

> print(name)
[1] Error: object 'name' not found
```
We learned about making our own functions to perform certain tasks.

Now, suppose we want to perform different tasks under different conditions.

For example, what if we want to do something different for different ranges of numbers? E.g., between -1 and 0, and between 0 and 100.

How can we write functions that make decisions based on these predicates?

Introducing, conditionals!
Conditionals
Logicals

Logicals are one of the basic R data types.

They are either TRUE or FALSE.

In other programming languages, logicals are called booleans.
Relational Operators

When we compare data, we uncover a relationship between them.

A predicate evaluates a relation, as a logical, so either TRUE or FALSE.

3 > 2 is an example of a predicate “David” < “Shivani” is as well

Relational operators build predicates.

The most common relational operators are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>
Logical Operators

We use logical operators to link predicates.

> x <- TRUE
> y <- FALSE

> !x
FALSE

> x && y
FALSE

> x || y
TRUE

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>NOT</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logical Operators

Element-wise AND and OR can be applied to logical vectors.

```r
> x <- c(TRUE, FALSE, 0, 10)
> y <- c(FALSE, TRUE, FALSE, TRUE)
> x & y
FALSE FALSE FALSE TRUE
> x | y
TRUE TRUE FALSE TRUE
```

**NOTE:** 0 is **FALSE**, and all non-zero numeric values are **TRUE**.
Missing Values

Sometimes, a vector might be missing values.

How can we find out if we have such faulty data?

We use `is.na()`, which returns a logical vector of the same length as its input. The entry in the output vector is `FALSE` wherever there is a missing value.

```r
> is.na(c(1, NA, 2))
FALSE  TRUE  FALSE
```
If Statement

We use conditionals to make decisions based on predicate values.

In R, the syntax for conditionals is an if-else statement:

```r
if (predicate) {
    # Something happens
} else {
    # Something else happens
}
```
Example

The clause in an if statement is executed only when the predicate is true.

```python
x <- 1
if (x < 0) {
    # This code is not executed
    print("x is negative")
}
if (x > 0) {
    # This code IS executed
    print("x is positive")
}
```
If-Else Example

The `else` clause is executed when the predicate is false.

```python
x <- 1
if (x < 0) {
    print("x is negative")
} else {
    print("x is greater than or equal to 0")
}
```
If-Else Example

You can include as many else clauses as you like.

```java
if (x > 0) {
    print("x is greater than 0")
} else if (x == 0) {
    print("x is equal to 0")
} else {
    print("x is less than 0")
}
```
Another Example

You can combine predicates in conditionals using logical operators.

```r
x <- 1
if (x > 0 && !is.na(x)) {
    print("x is positive, and x is not NA")
}
```
Nested If-Else Statements

You can also nest if-else statements inside one another:

```java
if (x > 0) {
    if (x < 10) {
        print("x is greater than 0 and less than 10")
    } else {
        print("x is greater than or equal to 10")
    }
}
```
Switch Statement

If you have multiple predicates to test, it might be too much work to write if and else-if statements for each one. In this case, you can use a switch statement:

```r
a <- 10
b <- 10
symbol <- readline(prompt = "Enter an ARITHMETIC OPERATOR: ")
switch(symbol,
      "+" = print(a + b),
      "-" = print(a - b),
      "*" = print(a * b),
      "/" = print(a / b),
      )
```
Repetitive tasks

Today, you saw examples of conditionals.

You can use them to write code to perform a task when a condition is met.

What if you want to execute a task many many times, so long as a condition is met?

You will learn to write loops next time!