Exploring data 2

More on functions

Functions—parameter defaults

When defining a function, you can set default values for some of the parameters. For example, in the `add_one` function, you can set the default value of the `number` input to `0`.

```
add_one <- function(number = 0){
        number + 1 # Value returned by the function
}
```

Now, if someone runs the function without providing a value for `number`, the function will use `0`. If they do provide a value for `number`, the function will use that instead.

```
add_one()     # Uses 0 for `number`
```

```
## [1] 1
```

```
add_one(number = 3:5)    # Uses 5 for `number`
```

```
## [1] 4 5 6
```

Functions—parameters

You could write a function with no parameters:

```
hello_world <- function(){
  print("Hello world!")
}

hello_world()
```

```
## [1] "Hello world!"
```

However, this will be pretty uncommon as you're first learning to write functions.

Functions—parameters

You can include multiple parameters, some with defaults and some without. For example, you could write a function that inputs two numbers and adds them. If you don't include a second value, 1 will be added as the second number:

```
add_two_numbers <- function(first_number, second_number = 1){
  first_number + second_number
}
```

```
add_two_numbers(first_number = 5:7, second_number = 5)
```

```
## [1] 10 11 12
```

```
add_two_numbers(first_number = 5:7)
```

```
## [1] 6 7 8
```

Functions—the `return` function

You can explicitly specify the value to return from the function (use `return` function).

```
add_one <- function(number = 0){
        new_number <- number + 1
        return(new_number)
}
```

If using `return` helps you think about what's happening with the code in your function, you can use it. However, outside of a few exceptions, you usually won't need to do it.

if / else

In R, the `if` statement evaluates everything in the parentheses and, if that evaluates to `TRUE`, runs everything in the braces. This means that you can trigger code in an `if` statement with a single-value logical vector:

```
tell_date <- function(){
  cat("Today's date is: ")
  cat(format(Sys.time(), "%b %d, %Y"))

  todays_wday <- lubridate::wday(Sys.time(),
                                 label = TRUE)
  if(todays_wday %in% c("Sat", "Sun")){
    cat("\n")
    cat("It's the weekend!")
  }
}
```

if / else

```
tell_date()
```

```
## Today's date is: Oct 15, 2020
```

`if / else`

You can add `else if` and `else` statements to tell R what to do if the condition in the `if` statement isn't met.

For example, in the `tell_date` function, we might want to add some code so it will print `"It's almost the weekend!"` on Fridays and how many days until Saturday on other weekdays.

if / else

```r
tell_date <- function(){
  # Print out today's date
  cat("Today's date is: ")
  cat(format(Sys.time(), "%b %d, %Y."), "\n")

  # Add something based on the weekday of today's date
  todays_wday <- lubridate::wday(Sys.time())

  if(todays_wday %in% c(1, 7)){       # What to do on Sat / Sun
    cat("It's the weekend!")
  } else if (todays_wday == c(6)) {  # What to do on Friday
    cat("It's almost the weekend!")
  } else {                           # What to do other days
    cat("It's ", 7 - todays_wday, "days until the weekend.")
  }
}
```

if / else

```
tell_date()
```

```
## Today's date is: Oct 15, 2020.
## It's  2 days until the weekend.
```