Functions

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Write your own custom functions in R

You will be working on this lab in cloud-based RStudio hosted by IBM Skills Network Labs.

Note that your lab will be reset after about 1 hour inactive window. It's recommended to backup the files you created.

About the Dataset

Imagine you got many movie recommendations from your friends and compiled all of the recommendations in a table, with specific info about each movie.

The table has one row for each movie and several columns

- `name` - The name of the movie
- `year` - The year the movie was released
- `length_min` - The lenght of the movie in minutes
- `genre` - The genre of the movie
- `average_rating` - Average rating on Imdb
- `cost_millions` - The movie's production cost in millions
- `sequences` - The amount of sequences
- `foreign` - Indicative of whether the movie is foreign (1) or domestic (0)
- `age_restriction` - The age restriction for the movie

You can see part of the dataset below

name	year	length_min	genre	average_rating	cost_millions	foreign	age_restriction
Toy Story	1995	81	Animation	8.3	30	0	0
Akira	1998	125	Animation	8.1	10.4	1	14
The Breakfast Club	1985	97	Drama	7.9	1	0	14
The Artist	2011	100	Romance	8	15	1	12
Modern Times	1936	87	Comedy	8.6	1.5	0	10
Fight Club	1999	139	Drama	8.9	63	0	18
City of God	2002	130	Crime	8.7	3.3	1	18
The Untouchables	1987	119	Drama	7.9	25	0	14
Star Wars	1977	121	Action	8.7	11	0	10
American Beauty	1999	122	Drama	8.4	15	0	14
Room	2015	118	Drama	8.3	13	1	14
Dr. Strangelove	1964	94	Comedy	8.5	1.8	1	10
The Ring	1998	95	Horror	7.3	1.2	1	18
Monty Python and the Holy Grail	1975	91	Comedy	8.3	0.4	1	18
High School Musical	2006	98	Comedy	5.2	4.2	0	0
Shaun of the Dead	2004	99	Horror	8	6.1	1	18
Taxi Driver	1976	113	Crime	8.3	1.3	1	14
The Shawshank Redemption	1994	142	Crime	9.3	25	0	16
Interstellar	2014	169	Adventure	8.6	165	0	10
Casino	1995	178	Biography	8.2	50	0	18
The Goodfellas	1990	145	Biography	8.7	25	0	14
Blue is the Warmest Colour	2013	179	Romance	7.8	4.5	1	18
Black Swan	2010	108	Thriller	8	13	0	16
Back to the Future	1985	116	Sci-fi	8.5	19	0	0
The Wave	2008	107	Thriller	7.6	5.5	1	16
Whiplash	2014	106	Drama	8.5	3.3	1	12
The Grand Hotel Budapest	2014	100	Crime	8.1	25.5	0	14
Jumanji	1995	104	Fantasy	6.9	65	0	12
The Eternal Sunshine of the Spotless Mind	2004	108	Drama	8.3	20	0	14
Chicago	2002	113	Comedy	7.2	45	0	12

First, let's identify your current working directory

- In the RStudio Console, run the following code snippet:

```
getwd()
```

then you should see the following result in console:

```
[1] "/resources/rstudio"
```

In the Files panel on the right, if your current directory is not `/resources/rstudio`, you could click `resources` folder and you should find a `rstudio` folder. This will be your current working directory in RStudio.

- Now let's download the dataset:

```
# code to download the dataset
download.file("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0101EN-Coursera/v2/dataset/movies-db.csv", destfile="movies-db.csv")
```

and you should see the dataset gets downloaded into the working directory as `movies-db.csv`

What is a Function?

A function is a re-usable block of code which performs operations specified in the function.

There are two types of functions :

- **Pre-defined functions**
- **User defined functions**

Pre-defined functions are those that are already defined for you, whether it's in R or within a package.

For example, `sum()` is a pre-defined function that returns the sum of its numeric inputs.

User-defined functions are custom functions created and defined by the user.

For example, you can create a custom function to print **Hello World**.

Pre-defined functions

There are many pre-defined functions, so let's start with some simple ones.

Using the `mean()` function, let's get the average of these three movie ratings:

- **Star Wars (1977)** - rating of 8.7
- **Jumanji** - rating of 6.9
- **Back to the Future** - rating of 8.5

Let's try some pre-defined functions in RStudio. Click `File->New File->R Script`, create a file called `predefined.R`.

- Copy and run the following lines in `predefined.R` to call the predefined `mean()` function

```
ratings <- c(8.7, 6.9, 8.5)
mean(ratings)
```

- We can use the `sort()` function to sort the movies rating in *ascending order*.

```
sort(ratings)
```

- You can also sort by *decreasing* order, by adding in the argument `decreasing = TRUE`.

```
sort(ratings, decreasing = TRUE)
```

Coding Exercise: Get the max (`max()` function) and min (`min()` function) rating value from ratings vector

▶ Click here to see solution

TIPS: How do I learn more about the pre-defined functions in R?

We will be introducing a variety of pre-defined functions to you as you learn more about R. There are just too many functions, so there's no way we can teach them all in one sitting.

But if you'd like to take a quick peek, here's a short reference card for some of the commonly-used pre-defined functions:

R Reference Card

User-defined functions

Now let's move on to user-defined functions, first we can create another R script file called `userdefined.R` to include all the functions we created.

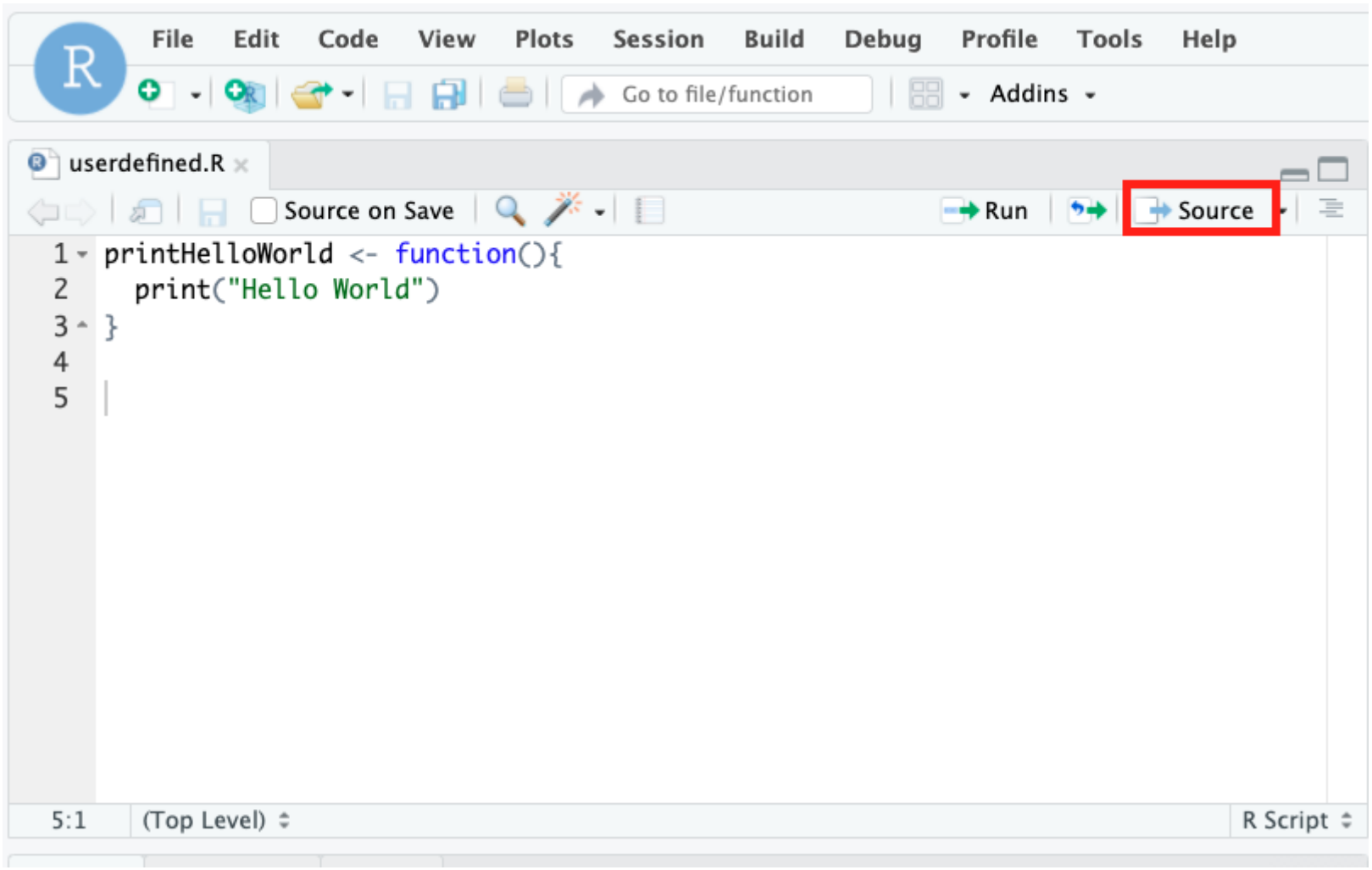
This time, we will write all user-defined functions in the script file, then run them via the console.

As such, we can have a nice separation from code definitions/implementation from code execution.

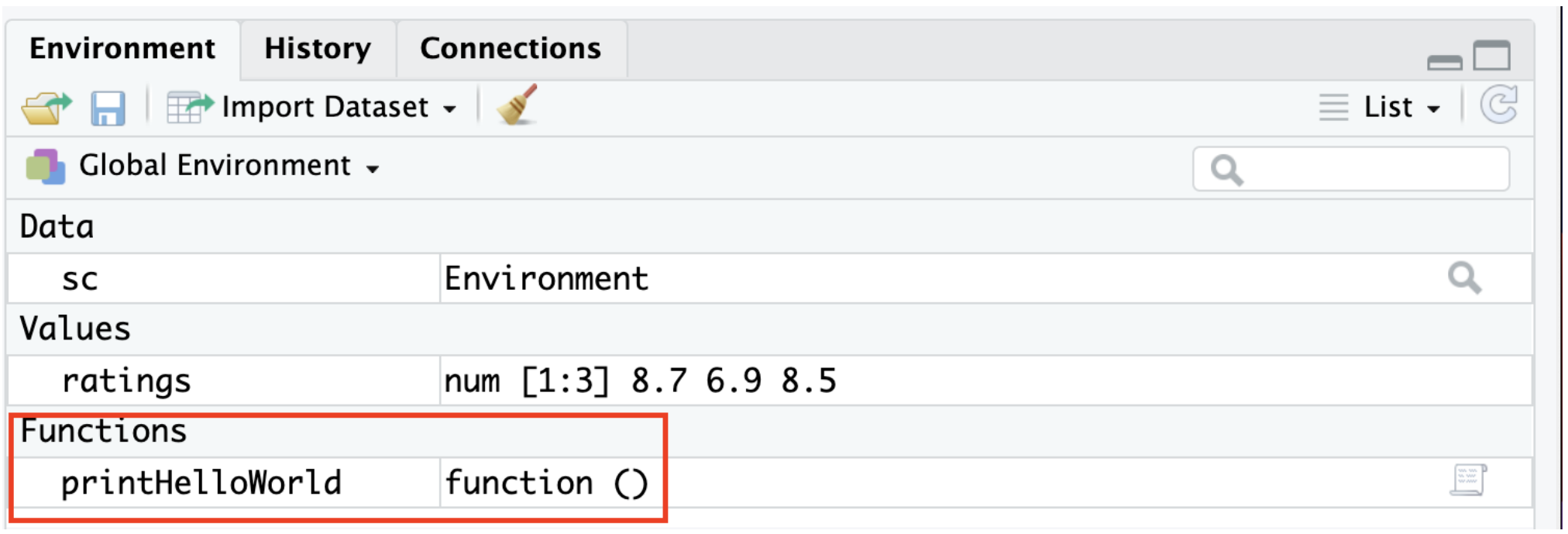
- Let's start with a simple print function, now copy the following function in `userdefined.R`

```
printHelloWorld <- function(){
  print("Hello World")
}
```

Let's create the function object (but not calling it yet) by clicking the `Source` button. Also make sure the last line is an empty line.



Then you can see a `printHelloWorld` function object in workspace, which means R interprator creates a function object for us to call, and it can be called from both console and script files.



Let's call it in console:

```
printHelloWorld()
```

and you should see the following result in console output

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

As you can see, the `printHelloWorld()` function has no inputs or arguments, but what if you want the function to provide some **output** based on some **inputs**?

- Let's take a look at an `add` function, copy the following lines into `userdefined.R` and

click `Source` icon to create the `add` function object.

```
add <- function(x, y) {
    x + y
}
```

Remember to click the `Source` button again everytime you made updates to the script file.

- Call the `add` function in console

```
add(3, 4)
```

As you can see above, you can create functions with the following syntax to take in inputs (as its arguments), then provide some output.

```
f <- function(<arguments>) {
    Do something
    Do something
    return(some_output)
}
```

Explicitly returning outputs in user-defined functions

In R, the last line in the function is automatically inferred as the output the function.

You can also explicitly tell the function to return an output.

```
add <- function(x, y){
    return(x + y)
}
```

It's good practice to use the `return()` function to explicitly tell the function to return the output so please update the previous `add` function with a `return()` function.

Using IF/ELSE statements in functions

The `return()` function is particularly useful if you have any IF statements in the function, when you want your output to be dependent on some condition.

- Copy a `isGoodRating` function into `userdefined.R` and run the script file:

```
isGoodRating <- function(rating){
    #This function returns "NO" if the input value is less than 7. Otherwise it returns "YES".

    if(rating < 7){
        return("NO") # return NO if the movie rating is less than 7

    }else{
        return("YES") # otherwise return YES
    }
}
```

- You can call `isGoodRating` in console with different inputs to cover the two logic branches:

```
isGoodRating(6)
isGoodRating(9.5)
```

Setting default argument values in your custom functions

You can a set a default value for arguments in your function. For example, in the `isGoodRating()` function, what if we wanted to create a threshold for what we consider to be a good rating?

Perhaps by default, we should set the threshold to 7.

- Update the `isGoodRating` by using a default threshold value 7:

```
isGoodRating <- function(rating, threshold = 7){
    if(rating < threshold){
        return("NO") # return NO if the movie rating is less than the threshold
    }else{
        return("YES") # otherwise return YES
    }
}
```

- Call the `isGoodRating` function again:

```
isGoodRating(6)
isGoodRating(10)
```

Notice how we did not have to explicitly specify the second argument (threshold), but we could specify it. Let's say we have a higher standard for movie ratings, so let's bring our threshold up to 8.5:

```
isGoodRating(8, threshold = 8.5)
```

Great! Now you know how to create default values. Note that if you know the order of the arguments, you do not need to write out the argument, as in:

```
isGoodRating(8, 8.5)
```

Coding Practice: Write a is bad rating function to print YES if rating is under 5 and print NO if rating is above 5

▶ Click here to see solution

Using functions within functions

Using functions within functions is no big deal. In fact, you've already used the `print()` and `return()` functions.

So let's try making our `isGoodRating()` more interesting.

Let's create a function that can help us decide on which movie to watch, based on its rating. We should be able to provide the name of the movie, and it should return **NO** if the movie rating is below 7, and **YES** otherwise.

First, in the console, let's read the movies data into workspace so that all functions could use it

```
my_data <- read.csv("movies-db.csv")
head(my_data)
```

and you should see the head in console result and `my_data` in the Environment panel.

Next, do you remember how to check the value of the **average_rating** column if we specify a movie name?
Here's how.

- Run the following code in console:

```
akira <- my_data[my_data$name == "Akira", "average_rating"]
akira
```

Now, let's put this all together into a function, that can take any **moviename** and return a **YES** or **NO** for whether or not we should watch it.

- Open `userdefined.R` script file, add a new `watchMovie` function and click `Source` to run the file

```
watchMovie <- function(data, moviename){
    rating <- data[data["name"] == moviename,"average_rating"]
    return(isGoodRating(rating))
}
```

- Call the watchMovie function in console

```
watchMovie(my_data, "Akira")
```

and you should see `YES` meaning we should watch this movie for its high rating.

Make sure you take the time to understand the function above. Notice how the function expects two inputs: `data` and `moviename`, and so when we use the function, we must also input two arguments.

But what if we only want to watch really good movies? How do we set our rating threshold that we created earlier?

Here's how, update the `watchMovie` function:

```
watchMovie <- function(data, moviename, my_threshold){
    rating <- data[data$name == moviename,"average_rating"]
    return(isGoodRating(rating, threshold = my_threshold))
}
```

Now our `watchMovie` takes three inputs: `data`, `moviename`, and `my_threshold`, let's call it from the console:

```
watchMovie(my_data, "Akira", 7)
```

What if we want to still set our default threshold to be 7?

- Here's how we can do it, update the `watchMovie` function with a default input:

```
watchMovie <- function(data, moviename, my_threshold = 7){
    rating <- data[data[,1] == moviename,"average_rating"]
    return(isGoodRating(rating, threshold = my_threshold))
}
```

- Then call it with only two inputs.

```
watchMovie(my_data,"Akira")
```

As you can imagine, if we assign the output to a variable, the variable will be assigned to `YES`

```
is_watch <- watchMovie(my_data, "Akira")
is_watch
```

While the `watchMovie` is easier to use, I can't tell what the movie rating actually is. How do I make it `print` what the actual movie rating is, before giving me a response? To do so, we can simply add in a `print` statement before the final line of the function.

We can also use the built-in `paste()` function to concatenate a sequence of character strings together into a single string.

- Now update the `watchMovie` function to print the movie name and actual rating

```
watchMovie <- function(moviename, my_threshold = 7){
    rating <- my_data[my_data[,1] == moviename,"average_rating"]

    memo <- paste("The movie rating for", moviename, "is", rating)
    print(memo)

    return(isGoodRating(rating, threshold = my_threshold))
}
```

- Then call the updated `watchMovie` function in the console

```
is_watch <- watchMovie("Akira")
is_watch
```

Coding Exercise: update the `watchMovie` function to use the mean rating of all movies as the threshold

▶ Click here to see solution

Global and local variables

So far, we've been creating variables within functions, but did you notice what happens to those variables outside of the function?

- Let's try to see what `memo` returns:

```
watchMovie <- function(moviename, my_threshold = 7){
    rating <- my_data[my_data[,1] == moviename,"average_rating"]

    memo <- paste("The movie rating for", moviename, "is", rating)
    print(memo)

    isGoodRating(rating, threshold = my_threshold)
}
```

- Run the updated function in console:

```
watchMovie("Akira")
memo
```

We got an error: `object 'memo' not found`. Why?

It's because all the variables we create in the function remain within the function. In technical terms, this is a **local variable**, meaning that the variable assignment does not persist outside the function. The `memo` variable only exists within the function.

But there is a way to create **global variables** from within a function -- where you can use the global variable outside of the function. It is typically *not* recommended that you use global variables, since it may become harder to manage your code, so this is just for your information.

To create a **global variable**, we need to use this syntax: `x <<- 1`

Here's an example of a global variable assignment, create a `myFunction` in `userdefined.R` file:

```
myFunction <- function(){
    y <<- 3.14
    return("Hello World")
    }
```

- Call `myFunction` in console

```
myFunction()
```

- Now you should access variable `y` globally:

```
y
```

Author(s)

Hi! It's Aditya Walia, the author of this lab. I hope you found R easy to learn! There's lots more to learn about R but you're well on your way. Feel free to connect with me if you have any questions.

Other Contributor(s)

Yan Luo

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-03-02	2.0	Yan	Added coding tasks