Other Methods for Exception Handling

In this lesson, we mention some other functions that can be useful while handling exceptions.

We'll cover the following

Function to Handle Exceptions

Other methods can be used to handle exceptions in R.

Here is a minimal list of functions that can help programmers in handling errors and warnings in code:

- `warning(...)` — to generate warnings
- `stop(...)` — to generate errors
- `suppressWarnings(expression)` — to evaluate the given expression and ignore any warnings
- `tryCatch(...)` — we have already studied this in detail here

Let's have a look at each one of them in detail.

Warning

The function `warning()` generates a warning message specified by the programmer. Suppose we are writing a program that divides two numbers however, we know that if a number is divided by 0, it returns `INF`. In such a case it is better to give a warning as this might be an unintentional mistake by the programmer:

```
division <- function(myNumber1, myNumber2)
{
  if(myNumber2 == 0)
```

```
  {
    warning("Division by 0")
  }

  return(myNumber1 / myNumber2)
}

# Driver Code
division(1, 0)

division(6, 1)
```

Stop

The `stop()` function generates **error** messages. This means that it will halt code execution. This function is particularly useful if we want the code to be *error free*. If we want to force the code to execute only when it completely meets our expectations, we can use this function.

For example, we want our program to divide only when none of the inputs is a 0:

```
division <- function(myNumber1, myNumber2)
{
  if(myNumber1 == 0)
  {
    stop("Division by 0")
  }
  if(myNumber2 == 0)
  {
    stop("Division by 0")
  }
  return(myNumber1 / myNumber2)
}

# Driver Code
division(1, 0)

division(6, 1)
```

In the code above, notice that the last statement **line number 17** is not executed even though it was error free because the program was already **stopped** due to a custom defined error in the previous statement **(Line number 15)**.

Suppress Warnings

There are times when a function issues a warning but as a programmer, we are

certain that our inputs are correct. We are certain that the warning will be

produced due to a conscious choice that will not produce erroneous results. In this case, we can wrap the function call in `suppressWarnings()`.

Let's revisit our matrix creating example, where the length of the vector being input into the matrix was not a sub-multiple or multiple of the number of rows.

```
myMatrix <- suppressWarnings(matrix(c(1:10), nrow = 4, byrow = TRUE))
print(myMatrix)
```

Here, we knew that the length of the vector is not a sub-multiple of the number of rows, but we chose not to display any warnings.

In the next lesson, we have a small quiz for you to test your understanding of exception handling.