How can I write my first tryCatch() function in R?

Authored by

stats writer

June 29, 2024

A tryCatch() function in R is used to handle errors and exceptions that may occur during the execution of a program. To write your first tryCatch() function, you need to first specify the code that you want to execute within the try block. This code will be evaluated and if an error occurs, it will be caught by the catch block. The catch block allows you to define the actions to be taken in case of an error, such as printing an error message or returning a default value. It is important to carefully structure your tryCatch() function to ensure efficient error handling and prevent program crashes.

Write Your First tryCatch() Function in R

You can use a tryCatch() function in R to return the value of some expression or produce a custom message if a warning or error is encountered.

This function uses the following basic syntax:

```
my_function <- function(x, y){
tryCatch(
#try to do this
{
#some expression
},
#if an error occurs, tell me the error
error=function(e) {
message('An Error Occurred')
print(e)
},
```

```
#if a warning occurs, tell me the warning
warning=function(w) {
message('A Warning Occurred')
print(w)
return(NA)
}
)
}
```

The following examples shows how to use a tryCatch() function in practice.

Example: Create a tryCatch() Function in R

Suppose we create the following tryCatch() function that attempts to take the log of one value and then divide by a second value.

If an error occurs, we will produce the message "An Error Occurred" and then print the error in R.

If a warning occurs, we will produce the message "A Warning Occurred", print the warning in R, and then return an NA value.

If neither an error or warning occurs, we'll simply return

the result of the function.

```
log_and_divide <- function(x, y){
tryCatch(
{
result = log(x) / y
return(result)
},
error=function(e) {
message('An Error Occurred')
print(e)
},
warning=function(w) {
message('A Warning Occurred')
print(w)
return(NA)
}
)
}
```


Let's run this function in different scenarios.

Scenario 1: No error or warning occurs.

The following code shows how to use the function in a

scenario where no error or warning occurs.

```
#run function
log_and_divide(10, 2)

1.151293
```

Since no error or warning occurs, the function simply returns the result of the expression, which turns out to be 1.151293.

Scenario 2: An error occurs.

The following code shows how to use the function in a scenario where an error occurs:

```
#run function
log_and_divide(10)

An Error Occurred
<simpleError in doTryCatch(return(expr), name, parentenv, handler):
argument "y" is missing, with no default>
```

Since we only provided one argument to the function,

we receive the message that "An Error Occurred" and we also see the exact error produced by R.

Scenario 3: A warning occurs.

The following code shows how to use the function in a scenario where a warning occurs:

```
#run function
log_and_divide(-10, 2)

A Warning Occurred
<simpleWarning in log(x): NaNs produced>
NA
```

Since we provided a negative value for the first argument, R is unable to calculate the log of a negative value so we receive the message that "A Warning Occurred", we see the exact warning produced by R, and the function returns NA as a result.

Additional Resources