**Exception Handling in R**

In R, errors and exceptions can be handled using functions like:

* try()
* tryCatch()
* withCallingHandlers()

Each method has its own use case, and I'll explain them with examples.

**1. Using try()**

try() allows you to attempt an operation and catch an error if it occurs, preventing the program from stopping.

**Example: Handling an Error with try()**

result <- try(log("text"), silent = TRUE)

print(result)

**Explanation**

* log("text") will generate an error because logarithm functions expect numeric input.
* try(log("text"), silent = TRUE) prevents R from stopping execution and instead returns an error object.
* The output will be:

Error in log("text") : non-numeric argument to mathematical function

but the program continues running.

**2. Using tryCatch()**

tryCatch() is a more powerful and structured way to handle exceptions. It allows you to define specific actions for different types of errors.

**Example: Handling Multiple Exceptions with tryCatch()**

safe\_division <- function(a, b) {

tryCatch({

result <- a / b

return(result)

}, warning = function(w) {

message("Warning occurred: ", w)

return(NA)

}, error = function(e) {

message("Error occurred: ", e)

return(NA)

}, finally = {

message("Execution complete.")

})

}

# Test cases

print(safe\_division(10, 2)) # Normal case

print(safe\_division(10, 0)) # Error: Division by zero

print(safe\_division("text", 2)) # Error: Non-numeric input

**Explanation**

* tryCatch() has handlers for:
  + error: Catches errors like division by zero or incorrect input.
  + warning: Catches warnings (not triggered here but can be added).
  + finally: Executes regardless of success or failure.
* Outputs:

Execution complete.

[1] 5

Error occurred: Error in a/b : non-numeric argument to binary operator

Execution complete.

[1] NA

**3. Using withCallingHandlers()**

This is useful when you want to **capture warnings** without stopping execution.

**Example: Handling Warnings with withCallingHandlers()**

handle\_warning <- function() {

withCallingHandlers({

x <- log(-1) # This generates a warning

print(x)

}, warning = function(w) {

message("Caught a warning: ", w)

invokeRestart("muffleWarning") # Prevents warning from being displayed

})

}

handle\_warning()

**Explanation**

* log(-1) generates a warning because logarithm is not defined for negative numbers.
* withCallingHandlers() captures the warning and displays a custom message without stopping execution.
* invokeRestart("muffleWarning") prevents the warning from showing in the console.

**Comparison Table**

| **Function** | **Use Case** | **Stops Execution?** | **Handles Errors?** | **Handles Warnings?** |
| --- | --- | --- | --- | --- |
| try() | Catching simple errors | No | ✅ | ❌ |
| tryCatch() | Handling specific errors, warnings, and finally block | No | ✅ | ✅ |
| withCallingHandlers() | Handling warnings without stopping execution | No | ❌ | ✅ |

**Examples:**

read\_file\_safe <- function(filename) {

tryCatch({

data <- read.csv(filename)

print("File read successfully!")

return(data)

}, error = function(e) {

message("Error: Could not read the file. Check if the file exists.")

return(NULL)

})

}

# Test cases

read\_file\_safe("non\_existent\_file.csv") # File doesn't exist

**Handling User Input Errors**

safe\_square\_root <- function(x) {

tryCatch({

result <- sqrt(x)

return(result)

}, error = function(e) {

message("Error: Invalid input. Please enter a numeric value.")

return(NA)

})

}

# Test cases

print(safe\_square\_root(16)) # Valid input

print(safe\_square\_root(-1)) # Error: Negative number

print(safe\_square\_root("abc")) # Error: Non-numeric input

**Handling API Request Errors**

library(httr)

safe\_api\_call <- function(url) {

tryCatch({

response <- GET(url)

if (status\_code(response) != 200) stop("API request failed!")

return(content(response, "text"))

}, error = function(e) {

message("Error: API request unsuccessful.")

return(NULL)

})

}

# Test cases

safe\_api\_call("https://jsonplaceholder.typicode.com/posts/1") # Valid API call

safe\_api\_call("https://invalid-url.com") # Invalid URL

**Handling Vector Index Errors**

safe\_vector\_access <- function(vec, index) {

tryCatch({

return(vec[index])

}, error = function(e) {

message("Error: Invalid index. Please check the vector length.")

return(NULL)

})

}

# Test cases

vec <- c(1, 2, 3, 4, 5)

print(safe\_vector\_access(vec, 2)) # Valid index

print(safe\_vector\_access(vec, 10)) # Invalid index

**Handling Errors in Loop Execution**

If one iteration fails, the loop should continue.

values <- c(4, 0, 8, "text", 16)

for (val in values) {

result <- tryCatch({

print(10 / as.numeric(val))

}, error = function(e) {

message("Error: Invalid calculation for value: ", val)

})

}

## ****Handling Function Execution Timeouts****

If a function takes too long, we can stop it.

safe\_long\_running <- function() {

tryCatch({

withTimeout({

Sys.sleep(5) # Simulating long computation

print("Completed!")

}, timeout = 3) # Timeout after 3 seconds

}, error = function(e) {

message("Error: The function took too long to execute.")

})

}

safe\_long\_running()

| **Example** | **Use Case** |
| --- | --- |
| **File Read Errors** | Prevents crashes when a file is missing |
| **User Input Errors** | Handles invalid inputs like negative numbers or strings |
| **Division by Zero** | Catches division by zero errors |
| **API Request Errors** | Handles network failures |
| **Function Timeouts** | Stops execution if it takes too long |
| **Vector Index Errors** | Prevents out-of-bounds errors |
| **Loop Execution Errors** | Ensures loops continue running even if one iteration fails |