## Exception Handling in Python using Try Except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

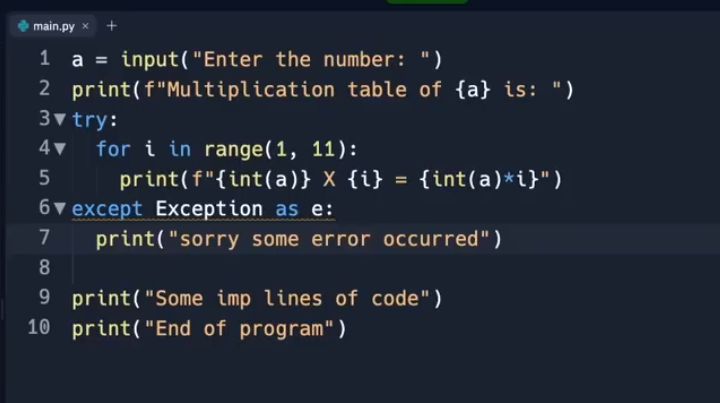
The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

**Example**

The try block will generate an exception, because x is not defined:

try:  
  print(x)  
except:  
  print("An exception occurred")



## Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

### Example

Print one message if the try block raises a NameError and another for other errors:

try:  
  print(x)  
except NameError:  
  print("Variable x is not defined")  
except:  
  print("Something else went wrong")

## Else

You can use the else keyword to define a block of code to be executed if no errors were raised:

### Example

In this example, the try block does not generate any error:

try:  
  print("Hello")  
except:  
  print("Something went wrong")  
else:  
  print("Nothing went wrong")

## Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

### Example

try:  
  print(x)  
except:  
  print("Something went wrong")  
finally:  
  print("The 'try except' is finished")

### Example

Try to open and write to a file that is not writable:

try:  
  f = open("demofile.txt")  
  try:  
    f.write("Lorum Ipsum")  
  except:  
    print("Something went wrong when writing to the file")  
  finally:  
    f.close()  
except:  
  print("Something went wrong when opening the file")

## Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

### Example

Raise an error and stop the program if x is lower than 0:

x = -1  
  
if x < 0:  
  raise Exception("Sorry, no numbers below zero")

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

### Example

Raise a TypeError if x is not an integer:

x = "hello"  
  
if not type(x) is int:  
  raise TypeError("Only integers are allowed")

**1.** Write a Python program to handle a ZeroDivisionError exception when dividing a number by zero.

exception ZeroDivisionError:

Raised when the second argument of a division or modulo operation is zero. The associated value is a string indicating the type of the operands and the operation.

def divide\_numbers(x, y):

try:

result = x / y

print("Result:", result)

except ZeroDivisionError:

print("The division by zero operation is not allowed.")

# Usage

numerator = 100

denominator = 0

divide\_numbers(numerator, denominator)

**2.** Write a Python program that prompts the user to input an integer and raises a ValueError exception if the input is not a valid integer.

exception ValueError:

Raised when an operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception such as IndexError.

**Python Code:**

def get\_integer\_input(prompt):

try:

value = int(input(prompt))

return value

except ValueError:

print("Error: Invalid input, input a valid integer.")

# Usage

n = get\_integer\_input("Input an integer: ")

print("Input value:", n)

**4.** Write a Python program that prompts the user to input two numbers and raises a TypeError exception if the inputs are not numerical.

exception TypeError:

Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.

**Python Code:**

def get\_numeric\_input(prompt):

while True:

try:

value = float(input(prompt))

return value

except ValueError:

print("Error: Invalid input. Please Input a valid number.")

# Usage

n1 = get\_numeric\_input("Input the first number: ")

n2 = get\_numeric\_input("Input the second number: ")

result = n1 \* n2

print("Product of the said two numbers:", result)

**5.** Write a Python program that opens a file and handles a PermissionError exception if there is a permission issue.

ception PermissionError:

Raised when trying to run an operation without the adequate access rights - for example filesystem permissions. Corresponds to errno EACCES, EPERM, and ENOTCAPABLE.

def open\_file(filename):

try:

with open(filename, 'w') as file:

contents = file.read()

print("File contents:")

print(contents)

except PermissionError:

print("Error: Permission denied to open the file.")

# Usage

file\_name = input("Input a file name: ")

open\_file(file\_name)

**6.** Write a Python program that executes an operation on a list and handles an IndexError exception if the index is out of range.

exception IndexError:

Raised when a sequence subscript is out of range. (Slice indices are silently truncated to fall in the allowed range; if an index is not an integer, TypeError is raised.)

**Python Code:**

def test\_index(data, index):

try:

result = data[index]

# Perform desired operation using the result

print("Result:", result)

except IndexError:

print("Error: Index out of range.")

nums = [1, 2, 3, 4, 5, 6, 7]

index = int(input("Input the index: "))

test\_index(nums, index)