# Exercise: Error Handling

The following problem descriptions **do not require** **submissions** to the Judge System.

**Ask** **your** **questions** here: [https://www.slido.com](https://www.slido.com/) by entering the course code **#python-advanced**

## Numbers Dictionary

You are provided with the following code:

numbers\_dictionary = {}  
  
line = input()  
  
**while** line != **"Search"**:  
 number\_as\_string = line  
 number = int(input())  
 numbers\_dictionary[number\_as\_string] = number  
  
line = input()  
  
**while** line != **"Remove"**:  
 searched = line  
 print(numbers\_dictionary[searched])  
  
line = input()  
  
**while** line != **"End"**:  
 searched = line  
 **del** numbers\_dictionary[searched]

print(numbers\_dictionary)

* On the first several lines, until you receive the command **"**Search**"**, you will receive on **separate lines** the **number as a text** and the **number as an integer**
* When you receive **"**Search**"** on the next several lines until you receive the command **"**Remove**"**, you will be given the **searched number as a text,** and you need to **print it on the console**
* When you receive **"**Remove**"** on the next several lines until you receive **"**End**"**, you will be given the **searched number as a text,** and you need to **remove** it from the dictionary
* In the end, you need to **print** what is left from the **dictionary**

There is some **missing code** in the solution, and some errors **may occur**. Complete the code, so the following errors are handled:

* Passing **non-integer** type to the variable number
* Searching for a **non-existent** number
* Removing a **non-existent** number

Print appropriate **messages** when an error has occurred. The messages should be:

* **"The variable number must be an integer"**
* **"Number does not exist in dictionary"** - for non-existing keys

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| one  1  two  2  Search  one  Remove  two  End | 1  {'one': 1} |
| one  two  Search  Remove  End | The variable number must be an integer  {} |
| one  1  Search  one  Remove  two  End | 1  Number does not exist in dictionary  {'one': 1} |

## Email Validator

You will be given some **emails** until you receive the command **"**End**"**. Create the following custom exceptions to validate the emails:

* NameTooShortError - raise it when the name in the email is **less than or equal to 4** ("**peter**" will be the name in the email "**peter@gmail.com**")
* MustContainAtSymbolError - raise it when there is **no** **"@"** in the email
* InvalidDomainError - raise it when the **domain** of the email is **invalid** (valid domains are: **.com, .bg, .net, .org**)

When an error is encountered, **raise** it with an appropriate **message**:

* NameTooShortError **- "Name must be more than 4 characters"**
* MustContainAtSymbolError - **"Email must contain @"**
* InvalidDomainError **- "Domain must be one of the following: .com, .bg, .org, .net"**

***Hint:*** use the following syntax to add a message to the Exception: MyException("Exception Message")

If the current email is **valid,** print **"**Email is valid**"** and read the next one.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| abc@abv.bg  End | Traceback (most recent call last):  File ".\email\_validator.py", line 20, in <module>  raise NameTooShortError("Name must be more than 4 characters")  \_\_main\_\_.NameTooShortError: Name must be more than 4 characters |
| peter@gmail.com  petergmail.com  End | Email is valid  Traceback (most recent call last):  File ".\email\_validator.py", line 18, in <module>  raise MustContainAtSymbolError("Email must contain @")  \_\_main\_\_.MustContainAtSymbolError: Email must contain @ |
| peter@gmail.hotmail  End | Traceback (most recent call last):  File ".\email\_validator.py", line 22, in <module>  raise InvalidDomainError("Domain must be one of the folowing: .com, .bg, .org, .net")  \_\_main\_\_.InvalidDomainError: Domain must be one of the folowing: .com, .bg, .org, .net |

## Password Validator

You will receive **passwords** as input on new lines, until the command "**Done**". Your task is to **validate** if the passwords are **strong** by applying the following **validations**:

* Each password should be at least **8 characters long**, otherwise, **PasswordTooShortError** should be **raised**.
* If the password consists of only **digits**, only **letters**, or only **special** **characters**, **PasswordTooCommonError** should be **raised**.
* Each password should have at least **1** **special** **character**, otherwise, **PasswordNoSpecialCharactersError** should be **raised**. The **special** **characters** are "**@**", "**\***", "**&**", and "**%**".
* If the password contains **empty** **spaces**, **PasswordContainsSpacesError** should be **raised**.

When an error is encountered, **raise** it with an appropriate **message**:

* **PasswordTooShortError** **- "Password must contain at least 8 characters"**
* **PasswordTooCommonError** **-"Password must be a combination of digits, letters, and special characters"**
* **PasswordNoSpecialCharactersError** - **"Password must contain at least 1 special character"**
* **PasswordContainsSpacesError** **-"Password must not contain empty spaces"**

If the current password is **valid,** print **"**Password is valid**"** and read the next one.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1234qwer@  Done | Password is valid |
| Qazxwj21  Done | Traceback (most recent call last):  File ".\password\_validator.py", line 65, in <module>  raise PasswordNoSpecialCharactersError('Password must contain at least 1 special character')  PasswordNoSpecialCharactersError: Password must contain at least 1 special character |
| Password  Done | Traceback (most recent call last):  File ".\password\_validator.py", line 67, in <module>  raise PasswordTooCommonError('Password must be a combination of digits, letters, and special characters')  PasswordTooCommonError: Password must be a combination of digits, letters, and special characters |
| zjL2k 1#@  Done | Traceback (most recent call last):  File ".\password\_validator.py", line 66, in <module>  raise PasswordContainsSpacesError('Password must not contain empty spaces')  PasswordContainsSpacesError: Password must not contain empty spaces |
| 12345q#  Done | Traceback (most recent call last):  File ".\password\_validator.py", line 57, in <module>  raise PasswordTooShortError('Password must contain at least 8 characters')  PasswordTooShortError: Password must contain at least 8 characters |

## Rotate Matrix [Solve with AI]

You are given the following code:

**def**rotate\_matrix(matrix):  
 matrix\_length = len(matrix)  
  
 **for**i *in* range(matrix\_length):  
 **for**j *in* range(i, matrix\_length):  
 matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]  
  
 **for**i *in* range(matrix\_length):  
 matrix[i].reverse()

mtrx = []  
  
***while True***:line = input().split() ***if not*** line: ***break*** mtrx.append(line)

rotate\_matrix(mtrx)  
  
**for**row *in* mtrx:  
 print(\*row, sep=' ')

On the following lines, until there is an **empty** **line**, you receive numbers, divided by **space**, representing each matrix **row**.

The **rotate\_matrix** function accepts the **matrix** as a parameter and **rotates** it **90 degrees clockwise** (**to the right**).

* The provided code contains **errors** that must be **fixed**. You should **refactor** the existing **code** without reconstructing the entire **algorithm**.

Implement **error** **handling** during the following stages:

* Verify the **matrix** contains only **integers**, otherwise, **MatrixContentError** should be **raised.**
* Ensure the input is an **N** x **N** (**2D** **matrix**), otherwise, **MatrixSizeError** should be **raised**.

When an **error** is encountered, **raise** it with an appropriate **message**:

* **MatrixContentError - "The matrix must consist of only integers"**
* **MatrixSizeError - "The size of the matrix is not a perfect square"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3  4 5 6  7 8 9 | 7 4 1  8 5 2  9 6 3 |
| 1 2 3 4  5 6 7 8 | Traceback (most recent call last):  File ".\rotate\_matrix.py", line 34, in <module>  raise MatrixSizeError("The size of the matrix is not a perfect square")  MatrixSizeError: The size of the matrix is not a perfect square |
| 7 8  9 k | Traceback (most recent call last):  File ".\rotate\_matrix.py", line 39, in <module>  raise MatrixContentError("The matrix must consist of only integers")  MatrixContentError: The matrix must consist of only integers |

## Online Banking [Solve with AI]

On the first line, you will receive your **bank** **account** details, separated by a **comma** and a **space**, indicating your **PIN code**, initial **balance**, and **age**. Subsequently, you will receive a series of **commands** until the command "**End**":

**"Send Money#{money}#{pin\_code}"**

* + You should send **money** to your friend in need. Before the **transaction**, you must go through several **validations**:
    - The money to be sent must be **less than or equal** to the initial balance, otherwise **MoneyNotEnoughError** should be **raised**.
    - The given **PIN code** must **match** the initial one, otherwise, **PINCodeError** should be **raised**.
    - To perform online transactions, you must be **18 or older**, otherwise, **UnderageTransactionError** should be **raised**.
* If the transaction is **successful**, print on the console:
  + "**Successfully sent {amount\_of\_money} money to a friend**"
  + "**There is {amount\_of\_money} money left in the bank account**"
  + The **amount of money** must be **formatted** to the **second** **decimal** **place**

**"Receive Money#{money}"**

* At the end of the month, you receive your **salary**. You **invest** half of the money in the stock market and the other half goes **directly** into the **bank** **account**:
  + If the given money is a **negative number**, **MoneyIsNegativeError**, should be **raised**.
* If the operation is **successful**, print on the console:
  + "**{amount\_of\_money} money went straight into the bank account**"
  + The **amount of money** must be **formatted** to the **second** **decimal** **place**

When an **error** is encountered, **raise** it with an appropriate **message**:

* **MoneyNotEnoughError** **- "Insufficient funds for the requested transaction"**
* **PINCodeError** **- "Invalid PIN code"**
* **UnderageTransactionError** **- "You must be 18 years or older to perform online transactions"**
* **MoneyIsNegativeError - "The amount of money cannot be a negative number"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9999, 3000, 18  Send Money#1500#9999  Receive Money#2000  End | Successfully sent 1500.00 money to a friend  There is 1500.00 money left in the bank account  1000.00 money went straight into the bank account |
| 5545, 20000, 40  Send Money#15000#5455  End | Traceback (most recent call last):  File ".\online\_banking.py", line 32, in <module>  raise PINCodeError('Invalid PIN code')  PINCodeError: Invalid PIN code |
| 2289, 1000, 17  Send Money#100#2289  End | Traceback (most recent call last):  File ".\online\_banking.py", line 35, in <module>  raise UnderageTransactionError('You must be 18 years or older to perform online transactions')  UnderageTransactionError: You must be 18 years or older to perform online transactions |
| 1234, 10000, 21  Send Money#10001#1234  End | Traceback (most recent call last):  File ".\online\_banking.py", line 29, in <module>  raise MoneyNotEnoughError('Insufficient funds for the requested transaction')  MoneyNotEnoughError: Insufficient funds for the requested transaction |
| 1111, 7000, 50  Receive Money#-500  End | Traceback (most recent call last):  File ".\online\_banking.py", line 46, in <module>  raise MoneyIsNegativeError('The amount of money cannot be a negative number')  MoneyIsNegativeError: The amount of money cannot be a negative number |