Exercise: Error Handling

Problems for exercise and homework for the [**Python Advanced Course @SoftUni**](https://softuni.bg/trainings/4106/python-advanced-may-2023)**.**

# 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**](mailto: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](mailto:abc@abv.bg) | Traceback (most recent call last):  File ".\email\_validator.py", line 20, in <module>  raise NameTooShort("Name must be more than 4 characters")  main .NameTooShort: Name must be more than 4 characters |
| [peter@gmail.com](mailto:peter@gmail.com) petergmail.com | 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](mailto:peter@gmail.hotmail) | 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 |