BIRZEIT UNIVERSITY

Department of Electrical and Computer Engineering

Linux Lab (ENCS 313)

Project #2: Data Validation System

Motivation

In some cases, you want to check that a given data is valid, before doing some operations on it(for example storing it to database). So instead of validating every data manually, it's better to create a general validation rules that validate any data, then f the validation rules pass, your code will keep executing normally ; however, if validation fails a validation error messages will be shown to describe which fields have failed to pass.

Problem

Given A python dictionary that have multi-level data. Your job is to build a validation rules for the data in that dictionary.

Simple Example that passes the validation rules

```
data = {
 "name": "Maher",
 "age": 23
}

rules = {
 "name": "string",
 "age": "number",
}
```

We can see from the rules that the name field is required, and the age field must be a number. Moreover we can notice from the data the name is a string, and the age is a number, so the data will pass the validation rules.

Simple Example that fails with the validation rules

```
data = {
 "name": "Maher",
 "age": "Maher"
}
rules = {
 "name": "string",
 "age": "number",
}
```

We can notice here that the age is not not a number(The age in our data is “Maher”, and it should be a number as the rules array says). Therefore an appropriate error message will be returned to indicate the fields the have errors. I.e.

```
errors = [
 "The age field must be a number "
]
```

More Details

For example given the following data that you want to validate,
<<NOTE: This is just an example data, your code should work with any data>>

```
data = {
 "title": "Book",
 "author": {
   "name": "Maher",
   "dob": "11-01-1996",
   "email": "maher@birzeit.edu",
   "co_authors": ["Monica", "Ziad"]
 },
 "pages": 50,
 "creation_date": "15/12/2015"
}
```

You can see that this dictionary has the following **attributes**.

- title
- author.name
- author.dob
- author.email
- author.co_authers
- pages
- creation_date

Note that you can go as deep as levels of the dictionary by adding a dot between the levels.

Your job, is to provide the following validations for any given dictionary.

1) Number
   a) Explanation:
      The field under validation must be numeric.
   b) Default error Message format:
      The **:attributes** field must be a number.
      You have to replace the “:attributes” with the name of the validated attribute.
   c) Example of error message:
      The age field must be a number

2) String
    a) Explanation:
       The field under validation must be String.
    b) Default error Message format:
       The **:attributes** field must be a string
    c) Example of error message:
       The name field must be a string
3) Required
    a) Explanation:
       The field under validation must be present in the input data and not empty string.
    b) Default error Message format:
       The **:attributes** field is required.
    c) Example of error message:
       The name field is required
4) Email
    a) Explanation:
       The field under validation must a valid email.
    b) Default error Message format:
       The **:attributes** field must be a valid email.
5) Phone Number
    a) Explanation:
       The field under validation must a valid mobile number (i.e. +xxxxxxxxxxxx, example +970599123456).
    b) Default error Message format:
       The **:attributes** field must be a valid mobile number.
6) Date
    a) Explanation:
       The field under validation must a valid date number (i.e. dd-mm-yyyy)
    b) Default error Message format:
       The **:attributes** field must be a valid date.

7) Min:*value*
    a) Explanation:
       the field under validation must be a number with a minimum value specified in the *value* as a parameter.
    b) Default error Message format:
       The :attributes field must be at least :value
    c) Example

```
 date = {
  "age": 10
 }
rules = {
  "age": 'min:15'
 }
```

       So an error message "The age filed must be at least 15" will be returned.
8) Max:*value*
    a) Explanation:
       the field under validation must be a number with a maximum value specified in the *value* as a parameter.

    b) Default error Message format:
       The :attributes field must be at max :value

9) Array
    a) Explanation:
       the field under validation must be an array(list).
    b) Default error Message format:
       The :attributes field must be an array

10) in:x,y,x…
    a) Explanation:
       The field under validation must be included in the given list of values.
    b) Default error message:
       The :attributes field must be one of these x,y,z ...
    c) Example

```
date = {
 "gender": "B"
}
rules = {
 "gender": 'in:M,F'
}
```

So an error message “The gender field must be one of these M,F ”

Cascading Attributes

Sometimes you may wish to add validation rules based on more complex conditional logic. So you can cascading them as the following:

```
rules = {
 "dob" : "required|date",
 "pages": "required|integer|min:5|max:20",
 "auther.name" : "required|string",
}
```

For example the pages field is required, and must be an integer, with min value as 5, and max value of 20.

Custom Attributes Mapping.

In some cases, you don’t like to original names of the attributes, For example you want to change dob to date of birth. Or “auther.name” to “Name of the Author”. Etc. So you can specify new names for the attributes to be displayed in the error message.

```
customAttributes = {
 "dob" : "Date of Birth",
 "auther.name" : "Author Name",
}
```

Note that other fields will stay with the original attribute name.

<u>Custom Error Messages</u>.

In some case, you don't like the default error message, for example, "The age field must be an integer" you want to display is as "Please provide a valid number for the age". So your code should support that.

```
customErrorMessages = {
       "age.number" : "Please provide a valid number for the age",
       "author.name.required" : "The author name must not be empty",
}
```

<u>What should you implement?</u>

Your code should provide a class called **DataValidator** which has the following constructor:

```
def __init__(data, rules, customAttributes = {}, customErrorMessage = {})
```

Which will take the data to be validated, and the validation rules. Moreover, it has two optional other parameters which are customAttributes and custom Error Messages.

And it will have the following methods:

1) getErrorMessages(), which will return an array of the error messages. Ex:

```
[
 "The name field must be a string",
 "The age must must be at least 12",
 "The dob must be a valid date",
 #etc
]
```

2) isValidData(), will return true if all the data is valid, false otherwise.
3) getFaildAttributes(), will return an array of failed attributes

<u>Another Thing</u>

Please prepare a set of data, and rules for all cases, and submit them with you project. Moreover, it would be better if you read about **"unit tests"** and to try to use them in the project. Doing unit tests for the cases will be considered as a bonus :")
You can check this link for more details
https://www.geeksforgeeks.org/unit-testing-python-unittest/

Notes:

- Students must work on groups of 2 only. Groups of 1 student are not allowed.
- You can use any built-in python functions even if they are not in the lab manual.
- Deadline: 11/5/2019
- Please submit your project as a reply to this message.