Test Driven Development Practice

Test driven development is a way to write a program by *writing the tests first*.

This means you should identify you test cases and **write your unit or integration tests before you start writing any code**.

For this project, we will design a Python web service by using test driven development.

Context

Cipher

A cipher is an algorithm which encrypts a message by shuffling its characters. One famous example of cipher is called Caesar Cipher which works by shifting the letters of a string.

By example, a Caesar Cipher which shifts characters by 1 would replace ‘a’ by ‘b’, ‘b’ by ‘c’, ‘c’ by ‘d’ and so on. Finally, ‘z’ would loop back and be replaced by ‘a’.

A Caesar Cipher can also be given a key to shift by more than one. By example, a cipher with key-3 would shift ‘a’ to ‘d’, ‘b’ to ‘e’ and so on.

Here is a full example of a cipher function:

text = “This is a secret I want to hide”
key = 15

output → “Iwxh xh p htrgti X lpci id wxst”

Web service

A web service is a feature or function available at a specific URL which returns data, but not necessarily html, by example it could return a json object. Web services are one way to allow a mobile application to access some features offered by a server.

The project

For this class project, you will have to implement the logic of a web service providing four different functionalities.

Launching the web service

To launch the web service, all you need to do is **run server.py** through PyCharm. The features of the service will then be available at the following addresses:

- `localhost:8000/encrypt`
- `localhost:8000/decrypt`
- `localhost:8000/random_key`
- `localhost:8000/random_encrypt`

Writing a test

You are required to write unit tests **before you implement your features**.

First read the documentation about each feature on the next page.

Once you are ready to implement a feature, carefully choose *meaningful* input and expected output and implement a unit test in the **tests.py** file.

Once you are done, run the test. It should fail since you did not implement the feature yet.

Writing a feature

You do not need to implement anything related to the server for this assignment. To implement a feature, all you need to do is open the **features.py** file and implement the function corresponding to that feature. The server will handle dispatching requests to the function.

When implementing the functions:

1) **Do not change the function name**
2) **Do not change the argument names**
3) **Implement the function so it passes your tests**

If you wrote tests correctly, making sure your tests pass should be enough to make sure the feature is bug-free.

Documentation

encrypt

url: once you launch your server on your computer, you can access this web service on http://localhost:8000/encrypt

The above endpoint should take two query arguments: **text** and **key**.

So, a request to this service would look like this.

http://localhost:8000/encrypt?text=hello&key=3

Expected output:

khoor

requirements: This web service must give access to a cipher as described on page 1.

To implement the feature, open the **features.py** file and implement the **encrypt** function. It should take **text** and **key** as argument and encrypt the text by shifting each character by **key**.

Only letters should be shifted, other characters such as spaces and punctuation should remain the same.

Capitalization must be preserved. By example, with a key of 3, ‘a’ will be shifted to ‘d’, but ‘A’ will be shifted to ‘D’.

Since the input comes from a web request, **all arguments will come in as strings**. By example the key will be given as ‘3’, not as 3. You will have to handle type casts.

decrypt

url: once you launch your server on your computer, you can access this web service on http://localhost:8000/decrypt

The above endpoint should take two query arguments: **secret** and **key**.

So, a request to this service would look like this.

http://localhost:8000/decrypt?secret=khoor&key=3

Expected output:

hello

requirements: This web service must give access to a decipherer. That is a function which does the exact **opposite of encrypt**

To implement the feature, open the **features.py** file and implement the **decrypt** function. It should take **secret** and **key** as argument and decrypt the text by unshifting each character by **key**.

Only letters should be shifted, other characters such as spaces and punctuation should remain the same.

Capitalization must be preserved. By example, with a key of 3, ‘d’ will be shifted to ‘a’, but ‘D’ will be shifted to ‘A’.

Again, since the input comes from a web request, **all arguments will come in as strings**. By example the key will be given as ‘3’, not as 3. You will have to handle type casts.

random_key

url: once you launch your server on your computer, you can access this web service on http://localhost:8000/random_key

The above endpoint should take no query argument

So, a request to this service would look like this.

http://localhost:8000/random_key

Expected output:

A random number from 1 to 25, ex: 7

requirements: This web service allows to randomly pick an encryption key.

To implement the feature, open the **features.py** file and implement the **random_key** function. You can and should import the Python **random** library.

random_encrypt

url: once you launch your server on your computer, you can access this web service on http://localhost:8000/random_encrypt

The above endpoint should take a **text** to be encrypted as argument

So, a request to this service would look like this.

http://localhost:8000/random_encrypt?text=hello

Expected output:

{'secret': 'khoor', 'key': 3}

requirements: This web service randomly picks an encryption key for the user and returns the encrypted secret and the chosen key as **json**.

To implement the feature, open the **features.py** file and implement the **random_encrypt** function. You can and should import the Python **json** library.