Tuesday: Introduction, asserTrue(); Python Modules

Introduction

So yesterday we introduced Object Oriented Programing, and wrote some unit tests to verify that our code works. Today we will be seeing how to handle user input. We will cover some error handling and we will finish up by working with some python modules.

Warm Up

1. How do you create a class in Python?
2. You are building a website for handling train ticketing . Using BDD list all the behaviors you expect to have in your application.
3. What is a test case, in Python and how is it created ?
4. Describe the testing process used in TDD.
5. Write a test method for a function that returns `"Hello World!"`

`assertTrue` method

Now we want to check if a contact object actually exists.

```
def test_contact_exists(self):
        '''
        test to check if we can return a Boolean  if we cannot find the contact.
        '''

        self.new_contact.save_contact()
        test_contact = Contact("Test","user","0711223344","test@user.com") # new contact
        test_contact.save_contact()

        contact_exists = Contact.contact_exist("0711223344")

        self.assertTrue(contact_exists)
```

Created a test case to test the `contact_exist` method that takes in a phone number. We then use the `assertTrue` method provided by the *TestCase* class to check if the return value is `True`

Run the test and confirm it fails then. Let us write the code to pass the test

```
    @classmethod
    def contact_exist(cls,number):
        '''
        Method that checks if a contact exists from the contact list.
        Args:
            number: Phone number to search if it exists
        Returns :
            Boolean: True or false depending if the contact exists
        '''
        for contact in cls.contact_list:
            if contact.phone_number == number:
                    return True

        return False
```

Here we created the class method `contact_exist` that loops through all the saved contacts and checks if any matches the number passed in. Then it returns a boolean value.

Display all contacts

contact_test.py

```
def test_display_all_contacts(self):
        '''
        method that returns a list of all contacts saved
        '''

        self.assertEqual(Contact.display_contacts(),Contact.contact_list)
```

We create a test case that test to check if we receive the list of the saved contacts. Run the test and confirm that it fails. Now lets create the code to make the test pass.

contact.py

```
    @classmethod
    def display_contacts(cls):
        '''
        method that returns the contact list
        '''
        return cls.contact_list
```

Here we create another class method `display_contacts` that returns the `contact_list` list.

Pyperclip module

The next feature we would like to implement, is that we would like to copy a contacts email address. We will copy that number to the machine's clipboard and paste it wherever we want.

We will use a third party module called **pyperclip**. Pyperclip will allow us to copy and paste items to our clipboard.

We first need to install it. Remember we use a tool called **pip** that installs third party modules from the *python package index*.

```
$ python3.6 -m pip install pyperclip
```

The command illustrates how to download a third party module.

Let us write a test for this behavior

```
import pyperclip
...............

    def test_copy_email(self):
        '''
        Test to confirm that we are copying the email address from a found contact
        '''

        self.new_contact.save_contact()
        Contact.copy_email("0712345678")

        self.assertEqual(self.new_contact.email,pyperclip.paste())
```

First at the top of our test module we import our new pyperclip module.

In the test method we first confirm that the contact exists then we call on the class method `Contact.copy_email()` and pass in the contact number.

pyperclip.paste()

In our `assertEqual` method we use `pyperclip.paste()`, and compare it with the contact object email.

`pyperclip.paste()` method returns whatever is copied on the machine's clipboard at that time.

Run the test to confirm that it fails.

```
import pyperclip
......................

@classmethod
    def copy_email(cls,number):
        contact_found = Contact.find_by_number(number)
        pyperclip.copy(contact_found.email)
```

Here we first import pyperclip at the top of the document.

Then we get the email of the contact by first getting the contact object by using our `find_by_number` method. We then use the `pyperclip.copy()` and pass in the contact email address.

pyperclip.copy()

`pyperclip.copy()` method allows us to copy passed in items to the machines clipboard

We use it here to copy the email to the clipboard.

Contact-list

Now let us implement the class we created to develop our contact-list application.

Create a new file in the Contact-List Folder and name it *run.py*.

#! shebang

The shebang line in any script determines the script's ability to be executed like an standalone executable without typing python3.6 beforehand in the terminal. It is always the first line in any runnable python script.

It also helps anyone looking at your application to know what version of python the application uses.

run.py

```
#!/usr/bin/env python3.6
```

The shebang starts with `#!` and points to the python Interpreter the application is using.

To make the program now executable we go to the terminal

```
$ chmod +x run.py
```

Now we can open the program with

```
$ ./run.py
```

In our `run.py` we want to implement the features we have just have been testing.

After our shebang line let us import the `Contact` class

```
from contact import Contact
```

Once we import the contact class let's now create functions that implement the behaviours we have just created.

1. Creating a Contact

```
def create_contact(fname,lname,phone,email):
    '''
    Function to create a new contact
    '''
    new_contact = Contact(fname,lname,phone,email)
    return new_contact
```

Here we create a function called `create_contact()`, that takes in four arguments. We then create a new contact object and return it.

2. Save contacts

```
def save_contacts(contact):
    '''
    Function to save contact
    '''
    contact.save_contact()
```

We create a `save_contacts` function that takes in a contact object and then calls the `save_contact` method to save the contact.

3. Delete contact

```
def del_contact(contact):
    '''
    Function to delete a contact
    '''
    contact.delete_contact()
```

We create a `del_contact` function that takes in a contact object and then calls the `delete_contact()` method on the contact object deleting it from the contact list.

4. Finding a contact

```
def find_contact(number):
    '''
    Function that finds a contact by number and returns the contact
    '''
    return Contact.find_by_number(number)
```

We create a function `find_contact` that takes in a number and calls the `Contact` class method `find_by_number` that returns the contact.

5. Check if a contact exists

```
def check_existing_contacts(number):
    '''
    Function that check if a contact exists with that number and return a Boolean
    '''
    return Contact.contact_exist(number)
```

The function `check_existing_contacts` takes in a number as an argument and calls the class method `contact_exist` which returns a boolean.

6. Displaying all contacts

```
def display_contacts():
    '''
    Function that returns all the saved contacts
    '''
    return Contact.display_contacts()
```

Task

Create a function that handles the behavior of copy the email to the clipboard for the `copy_email` class method

Main function

We have defined the functions that implement the behaviors. Now let us call them.

```
def main():
    print("Hello Welcome to your contact list. What is your name?")
            user_name = input()

            print(f"Hello {user_name}. what would you like to do?")
            print('\n')

            while True:
                    print("Use these short codes : cc - create a new contact, dc - display contact
s, fc -find a contact, ex -exit the contact list ")

                    short_code = input().lower()

                    if short_code == 'cc':
                            print("New Contact")
                            print("-"*10)

                            print ("First name ....")
                            f_name = input()

                            print("Last name ...")
                            l_name = input()

                            print("Phone number ...")
                            p_number = input()

                            print("Email address ...")
                            e_address = input()


                            save_contacts(create_contact(f_name,l_name,p_number,e_address)) # creat
e and save new contact.
                            print ('\n')
                            print(f"New Contact {f_name} {l_name} created")
                            print ('\n')

                    elif short_code == 'dc':

                            if display_contacts():
                                    print("Here is a list of all your contacts")
                                    print('\n')
                                    for contact in display_contacts():
                                            print(f"{contact.first_name} {contact.last_name} .....
{contact.phone_number}")

                                    print('\n')
                            else:
                                    print('\n')
                                    print("You dont seem to have any contacts saved yet")
                                    print('\n')

                    elif short_code == 'fc':

                            print("Enter the number you want to search for")

                            search_number = input()
                            if check_existing_contacts(search_number):
                                    search_contact = find_contact(search_number)
                                    print(f"{search_contact.first_name} {search_contact.last_name}"
)

                                    print('-' * 20)

                                    print(f"Phone number.......{search_contact.phone_number}")
                                    print(f"Email address.......{search_contact.email}")
                            else:
                                    print("That contact does not exist")

                    elif short_code == "ex":
                            print("Bye .......")
                            break
                    else:
                            print("I really didn't get that. Please use the short codes")
```

Here we created a main function that calls all the other function. It has a while loop, where we offered users a shortcut of commands, they could use in our application.

We then prompt the user for input which we transform to **lowercase** and strip any extra spaces.

We then use a series of `if`-`elif`-`else` statements to check user input and call the functions we have just defined depending on the user's choice.

Task

With your pair implement the remaining behaviors : 1. Deleting a contact. 2. Copying a contact email.

Calling the `main` function

Now we want the main function to run only if it was run from this module. To do this we use at the bottom of our `run.py` file:

```
if __name__ == '__main__':

    main()
```

Now we can run our script using `./run.py` and interact with our program from the command line.