Лабораторна робота №4.

Тема. Вивчення основ використання винятків в Python.

Мета. Ознайомлення з використанням та створенням винятків в Python.

Хід роботи.

1. Використання інструкції *raise*, перехоплення та обробка винятків.

   Розробіть функцію з назвою raise_func, яка при виклику відразу буде збуджувати (викликати) виняток *KeyError*. Розробіть іншу функцію except_func, яка буде містити виклик функції raise_func та в якій буде реалізовано перехоплення та обробку винятку. З розробленими функціями провести наступні дії:

   a. запустити функцію except_func на виконання і пересвідчитись що здійснюється обробка винятку;

   b. змінити виняток в функції raise_func на *IndexError* і запустити функцію except_func на виконання і пересвідчитись чи здійснюється обробка винятку;

   c. в функції raise_func залишити два винятки *IndexError* та *KeyError*. Запустити функцію except_func на виконання і пересвідчитись чи здійснюється обробка винятків;

   d. доповнити функції так, щоб було забезпечино виведення на екран додаткової інформації, якщо виняток збуджується та обробляється або його не вдалося перехопити та обробити.

   e. розробити власний клас виняток та функцію myexcept_raise_func яка при її виклику відразу буде збуджувати розробленй виняток. Розширте інструкцію *try* в функції except_func для того щоб здійснювалося перехоплення всіх винятків, які згадувалися в задачі та при їх обробці виводилися на екран представлення екземплярів винятків та додаткова інформація, що передана як аргументи.

2. Використання інструкції assert.

Інструкція assert в Python використовується для збужденя винятків на етапі відлагодження програм. Насправді це скорочена форма типового шаблону використання інструкції raise і це умовна інструкція raise. Проста форма *assert expression*, еквівалентна до:

```
if __debug__:
    if not expression: raise AssertionError
```

Розширена форма *assert expression1, expression2*, еквівалентна до:

```
if __debug__:
    if not expression1: raise AssertionError(expression2)
```

Інструкція *assert* дозволяє додати в програму код для її відлагодження. В розширеній формі *expression1* – вираз, який повинен повертати значення *True* або *False*. Якщо вираз *expression1* поверне значення *False*, інструкція *assert* збуджує виняток *AssertionError* з переданим йому *expression1*. Наприклад:

```
def write_data(file,data):
    assert file, “write_data: файл не визначений!”
```

Інструкція *assert* не повинна містити код програми, який забезпечує безпомилкову роботу програми, тому що цей код не буде виконуватися інтерпретатором, в оптимізованому режимі роботи (таких режим вмикається при запуску інтерпретатора з ключем -O). Зокрема, буде помилкою використовувати інструкцію *assert* для перевірки того що вводить користувач. Інструкцію *assert* використовують для перевірки умов, які завжди повинні бути істинними а якщо така умова порушується то ця ситуація розглядається як помилка в програмі а не як помилка користувача. Наприклад, якщо функція write_data() призначена для взаємодії з користувачем то інструкцію *assert* потрібно замінити на звичайну умовну інструкцію.

Розробіть модуль test_except.py для перевірки працездатності функції except_func.

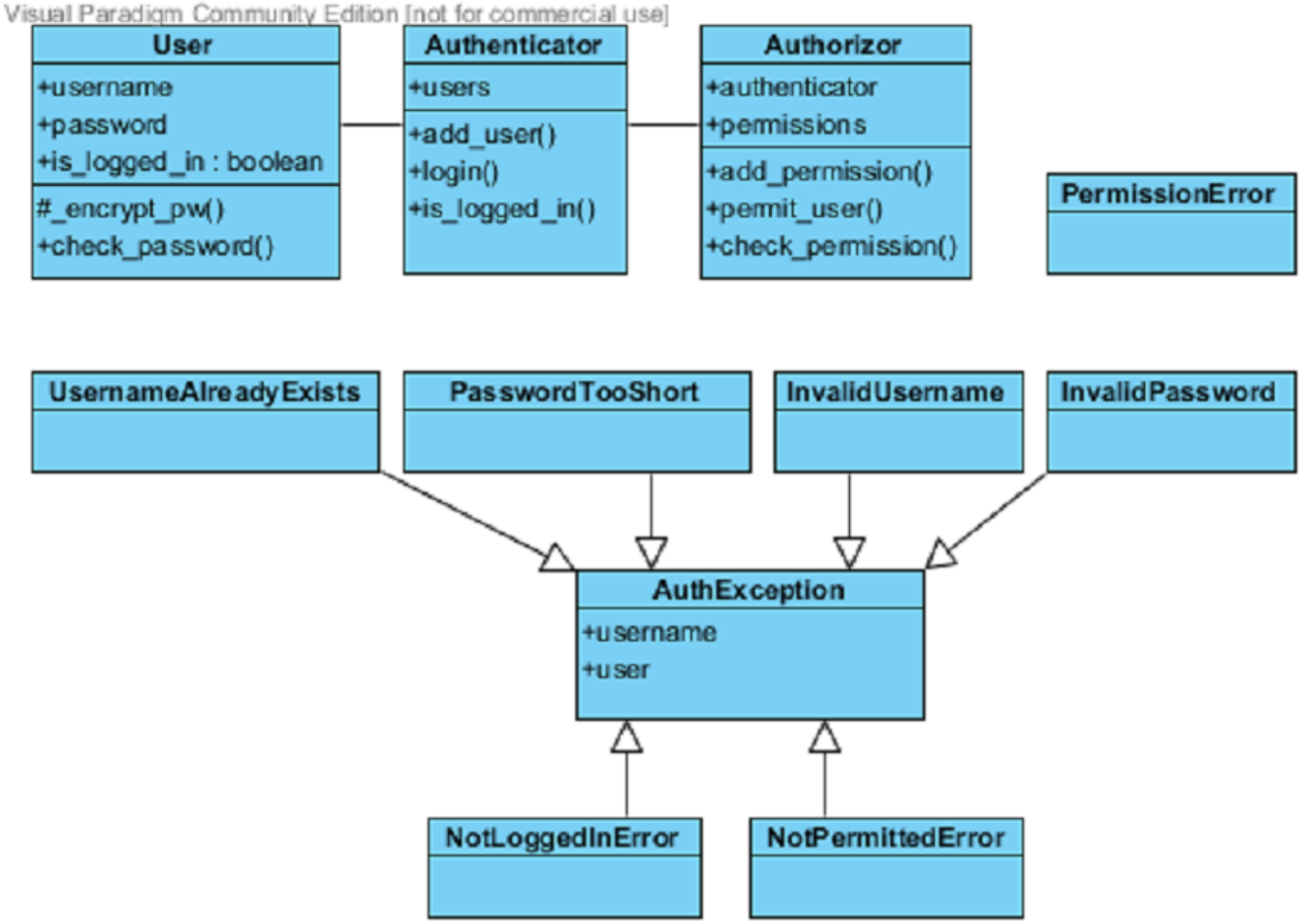
Для виконання наступних завдань необхідно ознаймитися з наступною інформацією (“Python 3 Object-oriented Programming Second Edition” by Dusty Phillips ст. 114-123)

Автентифіка́ція — процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора. З позицій інформаційної безпеки автентифікація є частиною процедури надання доступу для роботи в інформаційній системі, яка йде наступною після ідентифікації і передує авторизації. Один із способів автентифікації в інформаційній системі полягає у попередній ідентифікації на основі користувацького ідентифікатора «логіна» (англ. login — реєстраційного імені користувача) і пароля — певної конфіденційної інформації, знання якої передбачає володіння певним ресурсом в мережі. Отримавши введений користувачем логін і пароль, комп'ютер (програма) порівнює їх зі значенням, яке зберігається в спеціальній захищеній базі даних і, у випадку успішної автентифікації проводить авторизацію з подальшим допуском користувача до роботи в системі. (https://uk.wikipedia.org/wiki/Автентифікація).

Авторизація - керування рівнями та засобами доступу до певного захищеного ресурсу, як в фізичному розумінні (доступ до кімнати готелю за карткою), так і в галузі цифрових технологій (наприклад, автоматизована система контролю доступу) та ресурсів системи залежно від ідентифікатора і пароля користувача або надання певних повноважень (особі, програмі) на виконання деяких дій у системі обробки даних. З позицій інформаційної безпеки авторизація є частиною процедури надання доступу для роботи в інформаційній системі, після ідентифікації та автентифікації. (https://uk.wikipedia.org/wiki/Авторизація).

Програма(система), яка використовує автентифікацію та авторизацію може підтримуваим наступний сценарій роботи з програмою. Для того щоб отримати можливість працювати з програмою потрібно пройти реєстрацію. Реєстрація проводиться шляхом введення імені користувача та пароля. В процесі реєстрації, якщо ім'я, яке було введено вже використовує інший користувач то буде запропоновано обрати інше ім'я. Так само пароль, який не відповідає вимогам безпеки (довжина, символи) також пропонується змінити. Після успішної реєстрації користувач може працювати з системою але перед цим йому потрібно пройти автентифікацію, іншими словами, ввійти в систему ввівши ім'я та пароль. У випадку помилок чи невідповідностей у введених імені та паролю користувачу пропонується зробити ще одну спробу увійти. Тільки після успішного входу користувач отримує можливість працювати з системою але і тут його можуть чекати несподіванки бо в залежності від того, які права має користувач то тільки ці дії він і може виконувати в системі. Один користувач може тільки читати дані, інший може ще і змінювати ці дані а може бути користувач, який може здійснювати не тільки читання і внесення змін а може навіть додати чи видалити користувача з системи.

Діаграма класів для реалізації такої системи може бути наступна:



Діаграма класів містить ієрархію класів – винятків, які якраз будуть використовуватися для реалізації багаточесельних перевірок введених даних. Оскільки всі класи призначені для вирішення однієї і тієї ж задачі то систему доцільно реалізовувати у вигляді одного модуля auth.py. В цьому модулі вартує також створити екземпляри класів Authorizor та Authenticator що спростить використання системи в подальшому. Наступні рядки демонструють тестування модуля auth.py.

```
>>> import auth # імпорт модуля з класами User, Authenticator, Authorizor та екземплярами класів Authenticator, Authorizor
>>> auth.authenticator.add_user("Arman", "Armanpassword") # виклик методу add_user для додавання користувача (ім'я та пароль)
>>> auth.authorizor.add_permission("paint") # виклик методу add_permission для додавання права "paint"
```

```
>>> auth.authorizor.check_permission("paint", "Arman") # виклик методу для перевірки прав для користувача
Traceback (most recent call last):
  File "C:/Users/Oles/Desktop/auth/test_auth.py" , line 4, in <module>
    auth.authorizor.check_permission("paint", "Arman")
  File "C: \Users\Oles\Desktop\auth\auth.py", line 74, in check_permission
    raise NotLoggedInError(username)
auth.NotLoggedInError: ('Arman' , None) # результат виклику збудження винятку, який вказує що користувач не ввійшов в систему
>>> auth.authenticator.is_logged_in("Arman") # виклик методу для перевірки чи користувач ввійшов систему
False
>>> auth.authenticator.login("Arman", "Armanpassword") # виклик методу для входу в систему за допомогою імені та пароля
True
>>> auth.authorizor.check_permission("paint", "Arman") # виклик методу для перевірки права для користувача
Traceback (most recent call last):
  File "C:/Users/Oles/Desktop/auth/test_auth.py" , line 7, in <module>
    auth.authorizor.check_permission("paint", "Arman")
  File "C: \Users\Oles\Desktop\auth\auth.py", line 81, in check_permission
    raise NotPermittedError(username)
auth.NotPermittedError: ('Arman' , None) # результат виклику збудження винятку, який вказує що користувач не має вказаних прав
>>> auth.authorizor.check_permission("mix", "Arman") # виклик методу для перевірки права для користувача
Traceback (most recent call last):
  File "C: \Users\Oles\Desktop\auth\auth.py", line 76, in check_permission
    perm_set = self.permissions[perm_name]
KeyError: 'mix'
During handling of the above exception, another exception occurred:
Traceback (most recent call last):
  File "C:/Users/Oles/Desktop/auth/test_auth.py" , line 8, in <module>
    auth.authorizor.check_permission("mix", "Arman")
  File "C: \Users\Oles\Desktop\auth\auth.py", line 78, in check_permission
    raise PermissionError("Permission does not exist")
auth.PermissionError: Permission does not exist # результат виклику збудження винятку, який вказує що таких прав не існує
>>> auth.authorizor.permit_user("mix", "Arman") # виклик методу для додавання права користувачу
Traceback (most recent call last):
  File "C: \Users\Oles\Desktop\auth\auth.py", line 64, in permit_user
    perm_set = self.permissions[perm_name]
KeyError: 'mix'
During handling of the above exception, another exception occurred:
Traceback (most recent call last):
  File "C:/Users/Oles/Desktop/auth/test_auth.py" , line 9, in <module>
    auth.authorizor.permit_user("mix", "Arman")
  File "C: \Users\Oles\Desktop\auth\auth.py", line 66, in permit_user
    raise PermissionError("Permission does not exist")
auth.PermissionError: Permission does not exist # результат виклику збудження винятку, який вказує що таких прав не існує
>>> auth.authorizor.permit_user("paint", "Arman") # виклик методу для додавання права користувачу
>>> auth.authorizor.check_permission( "paint", "Arman") # виклик методу для перевірки прав користувача
True
```

Модуль auth.py можна використати як окрему частину в будь якій програмі, яка потребує функцій автентифікації та авторизації. Для цього потрібно імпортувати цей модуль, скористатися методами класів Authorizor та Authenticator та так реалізувати обробку винятків, щоб результати цієї обробки відповідали потребам та вимогам.

3. Розробіть сценарій аутентифікації та авторизації користувачів програми обліку нерухомості або гравців у грі «Морський бій». Розробіть також список можливих дозволів для роботи з обраною програмою. Опис оформити у вигляді текстового файлу account_description.txt в якому обов’язково потрібно вказати в яких випадках і які винятки будуть використовуватися при реалізації.
4. Розробіть модуль auth.py для аутентифікації та авторизації користувачів.
5. Розробіть модуль auth_account.py, який буде використовувати модуль auth.py і в якому буде реалізовано реєстрацію користувачів, вхід в систему та надання прав користувачеві. Рекомендується спочатку зареєструвати користувача з максимальними правами, а потім тільки цей користувач зможе надавати відповідні дозволи іншим користувачам.

Результати виконання лабораторної роботи потрібно представити у cms і вони повинні містити наступні складові:

func.py – до кінця заняття.
test_except.py – до кінця заняття
account_description.txt – до завершення граничного терміну
auth.py – до завершення граничного терміну
auth_account.py – до завершення граничного терміну

Література:

1. Python 3 Object-oriented Programming Second Edition , Dusty Phillips, 2015
Chapter 4. Expecting the Unexpected