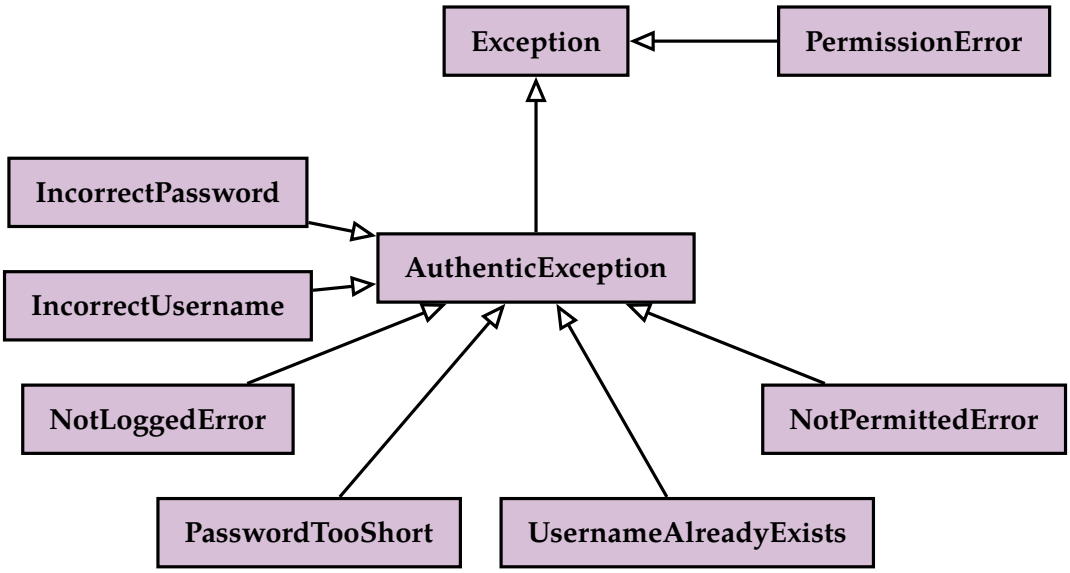
Lista zadań nr 3

Zadanie 1 (3 pkt) Napisz program, który symuluje prosty system uwierzytelniania i autoryzacji. W module `authorization_system` utwórz klasy:

- `User` - która przechowuje nazwę użytkownika i zaszyfrowane hasło. Klasa powinna zawierać metody:
  - `__init__()` - tworzy i incjializuje atrybutu: `username` - nazwa użytkownika podana podczas konkretyzacji obiektu; `password` - hasło (złożone z nazwy użytkownika i hasła) podane podczas konkretyzacji obiektu, ale zaszyfrowane przez `_encrypt_password()`; `is_logged` - o początkowej wartości `False`;
  - `_encrypt_password()` - metoda zmieniająca hasło (podane jako argument metody) i nazwę użytkownika na zaszyfrowaną wersję, którą zwraca (wykorzytaj moduł `hashlib` i np. funkcję skrótu `SHA-256`);
  - `check_password()` - metoda sprawdzająca hasło (podane jako argument metody) z hasłem przechowywanym w atrybucie i zwracająca `True` lub `False`.
- prostą klasę wyjątku `AuthenticException` z parometrowym konstruktorem, który tworzy i incjalizuje dwa atrybuty `username` i `user` - o wartości domniemanej `None`.
- prostą hierarichę klas wyjątków o pustych ciałach (zob poniższy diagram UML)



- klasę `Authenticator` - klasa kontrolująca użytkowników. Klasa powinna zawierać metody:

- ◇ `__init__()` - tworzy i incjializuje pustym słownikiem atrybut `users`;
- ◇ metoda `add_user`, która umożliwia dodanie użytkownika (o podanej nazwie i haśle) do słownika `user` pod warunkiem, że w słowniku nie ma takiego użytkownika (w przeciwnym wypadku następuje podsinienie wyjątku `UsernameAlreadyExists`) oraz jego hasło ma więcej niż 7 znaków (w przeciwnym wypadku następuje podsinienie wyjątku `PasswordTooShort`);
- ◇ metoda `login`, która służy do logowania (gdy użytkownik jest zalogowany jago atrybut `is_logged` przyjmuje wartość `True`), metoda podnosi wyjątek `IncorrectUsername`, gdy taki użytkownik nie ma konta (nie ma go w słowniku `users`) lub wyjątek `IncorrectPassword`, gdy hasło użytkownika jest niepoprawne - gdy logowanie się powiedzie metoda powinna też zwrócić wartość `True`;
- ◇ metoda `is_logged_in` - zwraca odpowiednio `True` lub `False` gdy dany użytkownik jest lub nie jest zalogowany;

- klasę `Authorizor`, która mapuje uprawnienia dla użytkowników. Klasa powinna zawierać metody:
  - ◇ `__init__()` tworzy i incjializuje pustym słownikiem atrybut `permissions` oraz atrybut `authenticator` inicjalizowany parametrem konstruktora;
  - ◇ metoda `add_permission`, która pozwala dodać do słownika nowe uprawnienie jako klucz z wartością będącą pustym zbiorem, jeżeli uprawnienie już istnieje metoda podnosi wyjątek `PermissionError`
  - ◇ metoda `permit_user`, która umożliwia przypisanie danemu użytkownikowi podanemu jako argument metody odpowiedniego uprawnienia (drugi argument metody). Metoda powinna podnosić w odpowiednich miejscach wyjątki `PermissionError` oraz `IncorrectUsername`.
  - ◇ metoda `check_permission`, która pozwala sprawdzić czy podany użytkownik posiada wskazane uprawnianie. Metoda powinna podnosić odpowiednie wyjątki: `NotLoggedError` - gdy użytkownik nie jest zalogowany, `PermissionError` - gdy nie ma takiego uprawnienia, `NotPermittedError` - gdy użytkownik nie ma podanego uprawnienia.

Moduł zakończ stworzeniem instancji klasy `Authenticator` oraz `Authorizor` (argumentem konstruktora tej drugiej instancji jest oczywiście pierwszy obiekt).

W głównym programie utwórz odpowiednie instancje obiektów reprezentacyjnych użytkowników i nadaj im uprawnienia (np. testowania i/lub zmieniania programów).

Stwórz klasę `Editor`, która zawiera podstawowy interfejs menu pozwalający niektórym użytkownikom zmienić lub testować program. Wspomniana klasa powinna zawierać metody:

- `__init__()` - tworzy dwa atrybutu: `username` wartości `None` oraz `options` o wartosści self.options = {"a": self.login, "b": self.test, "c": self.change, "d": self.quit}.
- `login()` - metoda pobierająca od użytkownika nazwę i hasło oraz wywołująca odpowiednią metodę `login()` stworzonej instancji klasy `Authenticator` wraz z obsługą wyjątków;
- `is_permitted()` - metoda sprawdzająca czy użytkownik jest zalogowany i ma odpowiednie uprawnienia (wywołuje metodę `check_permission` i obsługuje odpowiednie wyjątki);
- `test()` - metoda imitująca testowanie hipotetycznego programu (korzysta z metody `is_permitted()`);
- `change()` - metoda imitująca zmienianie hipotetycznego programu (korzysta z metody `is_permitted()`);
- `quit()` - metoda kończąca działanie głównego programu;
- `run()`, która zapewnia pobranie od użytkownika odpowiedniego klucza i odczytanie (wraz z wywołaniem) odwadniającej mu wartości słownika `options`.

Główny program powinien tworzyć instancje klasy `Editor` i wywoływać metodę `run()`.

Zadanie 2 (1 pkt) Napisz nieskończony generator produkujący kolejne dodatnie liczby całkowite, następnie napisz nieskończony generator kolejnych kwadratów dodatnich liczb całkowitych - wykorzystaj poprzedni generator. Zaprojektuj i napisz funkcję `select()`, która tworzy n-elementową listę wartości dowolnego obiektu iterowalnego - przetestuj funkcję na zdefiniowanych wcześniej nieskończonych generatorach (w definicji użyj funkcje `iter()` oraz `next()`). Zdefiniuj generator produkujący trójelementowe krotki zawierające tzw. trójki pitagorejskie tzn. krotki postaci (a, b, c) gdzie $a^2 + b^2 == c^2$ - wykorzystaj wyrażenie generatora (załóż, że $a < b < c$). Wyświetl 15 pierwszych trójek korzystając z funkcji `select()`.

Zadanie 3 (1 pkt) Napisz klasę implementującą iterator ciągu Fibonacciego, który zwraca kolejne wyrazy ciągu mniejsze od $n > 0$. Wykonaj to samo zadanie pisząc odpowiedni generator. Następnie, utwórz iterator (na podstawie nieskończonego generatora ciągu Fibonacciego) zwracający liczby Fibonacciego od F_{100000} do F_{100020} i zapisz te liczby do pliku tekstowego. Ile cyfr ma liczba F_{100000}?

Zadanie 4 (1 pkt) Napisz generator `gen_time`, który produkuje kolejne sekwencję czasu w postaci krotki `(godziny, minuty, sekundy)`. Generator powinien przyjmować w postaci krotek czasu czas startowy, czas końcowy i krok czasu. Zamiast zwykłych krotek możesz skorzystać z krotek nazwanych. Przykładowe działanie:

```
>>> for time in gen_time((8, 10, 00), (10, 50, 15), (0, 15, 12) ):
    print(time)


(8, 10, 0)
(8, 25, 12)
(8, 40, 24)
(8, 55, 36)
(9, 10, 48)
(9, 26, 0)
(9, 41, 12)
(9, 56, 24)
(10, 11, 36)
(10, 26, 48)
(10, 42, 0)
>>>
```

Zadanie 5 (1 pkt) Napisz "generator" kolejnych liczb pierwszych. Jak dużą liczbę pierwszą potrafisz wygenerować? Zapisz 10000 początkowych liczb pierwszych do pliku.

Zadanie 6 (1 pkt) Napisz rekurencyjną funkcję generatora, która pozwoli przekształcać zagnieżdżona sekwencję na postać jednowymiarową w postaci listy wartości (pomijaj ciągi tekstowe) tj. na przykład
$([1,'kot'],3,(4,5,[7,8,9])) \rightarrow [1,'kot',3,4,5,6,7,8,9]$.