Lista zadań nr 2

Zadanie 1 (2 pkt) Napisz program, który będzie symulował bardzo prosty notatnik. Utwórz plik modułu, który zawiera klasy: `Note` i `Notebook`. Konstruktor pierwszej klasy powinien tworzyć i inicjalizować atrybuty `text` i `tag` (dane przekazane przez parametry konstruktora) atrybut `date` (automatyczna data tworzenia notatki) oraz atrybut `ID` (automatyczny numer notatki w danym notatniku - inicjalizowany przez odpowiedni atrybut klasy zliczający liczbę już utworzonych instancji klasy). Klasa `Note` powinna posiadać także metodę `match`, która zwraca `True` lub `False` jeżeli `text` lub `tag` zawiera przekazany do tej metody ciąg tekstowy. Konstruktor klasy `Notebook` powinien tworzyć i inicjalizować pustą listą - atrybut `notes`. Klasa `Notebook` powinna posiadać następujące metody: `new_note()` - pozwala na dodanie obiektu klasy `Note` do notatnika (listy `notes`); `modify_text()` - pozwalająca na zmianę tekstu notatki o podanym `ID`; `modify_tag()` - pozwalająca na zmianę tekstu etykiety notatki o podanym `ID`; `search()` - zwraca listę notatek zawierających szukaną frazę (w tekście lub etykiecie notatki). Główny program powinien importować klasy, które są przedstawione wyżej i definiować nową klasę `Menu`. Konstruktor klasy `Menu` powinien tworzyć następujące atrybuty: `notebook` - inicjalizowany obiektem klasy `Notebook`, `options` - inicjalizowany słownikiem: `{"1": self.show_notes, "2": self.search_notes, "3": self.add_note, "4": self.modify_note, "5": self.quit}`. We wspominanej klasie zdefiniuj metody: `show_menu()` - wyświetlająca menu notatnika; `run()` - zapewniająca pobranie odpowiedniego klucza i odczytanie odwadniającej mu wartości słownika `options`; metody odpowiadające wartościom słownika `options` tzn. `show_notes()`, `search_notes()` itd. (łatwo wywnioskować jak mają działać te metody). **Uwaga:** Metoda `search_notes()` powinna wyświetlać znalezioną listę notatek (zawierających szukaną frazę) za pomocą wywołania metody `show_notes()` (jak?).

Zadanie 2 (2 pkt) W module `figure_module` zdefiniuj klasę `Figure`. Konstruktor klasy `Figure` powinien definiować i inicjalizować (wartościami domyślnymi) dwa atrybuty `colour` i `is_filled` (wartość `True` lub `False`). W klasie powinny być także zdefiniowane metody specjalne `__str__()` oraz `__repr__()`. Następnie zdefiniuj dwie klasy `Circle` i `Rectangle` dziedziczące po klasie `Figure`. Konstruktor klasy `Circle` powinien wywoływać konstruktor klasy bazowej i definiować dodatkowy atrybut `radius` - wykorzystaj właściwość, aby zabezpieczyć atrybut przed ustawieniem niepoprawnych danych (wykorzystaj właściwości). W klasie powinny być zdefiniowane także właściwości zwracające wartość pola, obwodu i średnice - `area`, `perimeter`, `diameter`. Klasa `Circle` powinna też posiadać metody specjalne `__str__()` oraz `__repr__()` - tam gdzie to uzasadnione wykorzystaj wywołanie metody klasy bazowej. Konstruktor klasy

`Rectangle` powinien wywoływać konstruktor klasy bazowej i definiować dodatkowe atrybuty `width` i `height` - wykorzystaj właściwości, aby zarządzać tymi atrybutami. Wyposaż tę klasę w odpowiednie właściwości i metody specjalne analogiczne jak w klasie `Circle` (zamiast średnicy rozważ przekątną - `diagonal`). Przetestuj klasy `Circle` i `Rectangle`. **Uwaga:** Metoda `__repr__()` w klasach `Circle` i `Rectangle` powinna być tylko dziedziczona po klasie bazowej i powinna działać dobrze dla wszystkich wspominanych klas (wykorzystaj specjalny atrybut instancji `__dict__` - słownik przechowujący nazwy atrybutów powiązane z ich wartościami).

Zadanie 3 (2 pkt) Napisz program, który tworzy obiekt klasy `Person`. Klasa posiada atrybuty: `name`, `surname`, `age`. Program pozwala na uzupełnienie wartości atrybutów danymi podanymi z klawiatury. Wiek musi być liczbą całkowitą w zakresie od 0 do 130, a imię i nazwisko muszą posiadać minimum 3 znaki - wykorzystaj w tym celu właściwości. Klasa `Person` powinna definiować metodę `__str__()`. Następnie, zaimplementuj dwie klasy `Student` i `Employee`, oparte na klasie `Person`. W klasie `Student` dodaj atrybuty `field_of_study` i `student_book` - słownik, którego klucze to nazwy przedmiotów, a wartości to oceny. W klasie `Employee` dodaj atrybut `job_title` i `skills` - lista kluczowych umiejętności. Zaimplementuj odpowiednie metody pozwalające na uzupełnianie i wyświetlanie wartości atrybutów w obu klasach potomnych.

Zadanie 4 (2 pkt) Zdefiniuj klasę `Rectangle` z dwoma atrybutami: `length` i `height` - długości boków. Klasa powinna posiada następujące metody:

- `__init__()`;
- `area()` - zwraca pole;
- `__str__()`;
- `__repr__()`.

Zdefiniuj klasę `Cuboid` dziedziczącą po klasie `Rectangle` i mającą dodatkowy atrybut `width` oraz metody:

- `__init__()` - wywołuje konstruktor klasy bazowej;
- `area()` - zwraca pole powierzchni prostopadłościanu (wykorzystaj odpowiednią metodę klasy bazowej);
- `volume()` - metoda ma zwracać objętość prostopadłościanu (wykorzystaj odpowiednią metodę klasy bazowej);
- `__str__()`;

- `__repr__()`.

Inicjalizacja atrybutów instancji klas powinna odbywać się poprzez wartości jej parametrów.

Napisz program, w której wczytasz dane z pliku tekstowego - dane czytane do końca pliku. W kolejnych wierszach dane: liczba 1 lub 2 (1-prostokąt, 2-prostopadłościan), a następnie oddzielone spacjami, w przypadku prostokąta długość i wysokość zaś w przypadku prostopadłościanu długość, wysokość i szerokość. Wypisz na ekranie monitora typ figury, parametry ją charakteryzujące i pole powierzchni, a dla prostopadłościanu także objętość. Zastosuj obsługę wyjątków. Zdefiniuj własną klasę `InvalidData` dziedzicząca po `Exception`. Wykorzystaj tę klasę do obsługi sytuacji wyjątkowych: ujemne lub zerowe długości boków/krawędzi. Obsłuż błędy `IO` i błąd złego typu danych.