Lista zadań nr 1

Zadanie 1 (zadanie na start - prosta symulacja gry "SAPER") Napisz program, który będzie symulował popularną grę "Saper" dostępną dla systemu operacyjnego Windows.
Możesz wykorzystać proponowany schemat programu wykorzystujący funkcje. W module `minesweeper` umieść następujące funkcje:

- Funkcja `get_number(a,b, text)` powinna pobierać od użytkownika liczbę całkowitą z zakresu od a do b i zawracać ją (funkcja nie może zwrócić błędnych danych). Parametr `text` odpowiada za wyświetlenie podpowiedzi. Wykorzystaj tę funkcję do pobrania rozmiaru planszy oraz liczby min. Liczba min powinna być liczbą z zakresu od 10 do $(m-1) \times (n-1)$ (gdzie $m \times n$ to rozmiar planszy, natomiast $8 \leqslant m \leqslant 30$ oraz $8 \leqslant n \leqslant 24$).

- Funkcja `lay_mines()` generuje zbiór losowych współrzędnych min na planszy i zwraca go.

- Funkcja `number_of_neighboring_mines()` dla każdego pola (współrzędne pola podane jako argument) liczy ile sąsiadów to miny i zwraca tę liczbę.

- Funkcja `create_board()` generuję planszę (tablicę dwuwymiarową), gdzie pola wypełnione są liczbami oznaczającymi liczbę min na polach sąsiednich (od 0 do 8 - zera można nie wyświetlać) oraz minami (np. liczby 9). Funkcja powinna zwracać tak utworzoną tablicę.

- Rekurencyjna funkcja `reveal_fields()` odkrywająca pola planszy dla podanego przez użytkownika pola.

- Funkcja `print_board()` wyświetlająca aktualny wygląd planszy. Możesz skorzystać z funkcji `chr()` zwracającej jednoznakowy łańcuch znaków, którego liczbowy kod Unicode zostaje podany jako argument (funkcja działa odwrotnie do funkcji `ord()`). Jako argumenty funkcji `chr()` możesz wykorzystać liczby: 9552, $9553, 9556, 9559, 9562, 9565, 9568, 9571, 9574, 9577, 9580$.

W głównym pliku zaimportuj moduł `minesweeper` i umieść napisz w nim następującą funkcję:

- Głowna funkcja programu `sapper()`, sterująca całą rozgrywką. Funkcja powinna korzystać z wyżej wymienionych funkcji, pobierać od użytkownika współrzędne pola do sprawdzenia, rejestrować aktualny stan gry i kończyć grę gdy użytkownik odkryje wszystkie pola poza minami lub trafi na minę

Zadanie 2 (1 pkt) Utwórz klasę `Coin`. Bezparametrowy konstruktor tej klasy powinien tworzyć atrybut (`side`) utrzymujący aktualną stronę monety. Zdefiniuj dwie metody: `throw()` - losowo "zmienia" stronę monety, `show_side()` - zwraca wartość atrybutu `side`.

a) Utwórz kilka obiektów klasy `Coin` i wywołaj ich metody oraz wykonaj symulację piętnastu rzutów monetą.

b) Napisz program, który będzie symulował pewną grę hazardową. Program powinien tworzyć trzy instancje klasy `Coin` reprezentujące monety o nominałach 1 zł, 2 zł i 5 zł. Początkowe saldo gry powinno być równe 0 zł. W każdej kolejceprogram powinien wykonywać rzut trzema monetami i dodawać do salda ich wartości, jeżeli n danej monecie wypadnie orzeł. Gra powinna się kończyć wtedy, gdy saldo będzie równe lub większe 20 zł. Saldo równe dokładnie 20 zł oznacza wygraną, a większe przegraną. Wykonaj np. 100 symulacji gry i zlicz przegrane i wygrane.

Zadanie 3 (1 pkt) Utwórz klasę `Die` reprezentująca kostkę do gry. Konstrukt klasy powinien tworzyć dwa atrybuty chronione: `sides` - liczba ścian kostki, `value` - wylosowana liczba oczek. Pierwszy atrybut powinien być inicjalizowany wartością parametru konstruktora, drugi mieć wartość początkową równą np. `None`. Zdefiniuj w klasie metody: `roll()` - wykonuje rzut kostką i wynik rzutu przypisuje do atrybutu`value`, `get_sides()` - zwraca liczbę ścianek, `get_value` - zwraca liczbę oczek na kostce. Napisz program w którym użytkownik będzie grała w popularną grę "Oczko" ale z wykorzystaniem dwóch sześciennych kostek. Gracz będzie rzucał kostkami starając się uzyskać większą liczbę punktów od ukrytych punktów komputera, ale nie większą niż 21. Sugestie dotyczące gry:

- każda kolejka gry powinna być obiegiem pętli powtarzanej dotąd, aż gracz zrezygnuje z rzucania lub gdy suma punktów będzie większa niż 21;
- na początku każdej kolejki gracz powinien być pytany o to czy chce kontynuować rzucanie i sumować punkty;
- w każdej kolejce program powinien symulować rzut dwiema sześciennymi kostkami. Najpierw niech rzuca kostką należącą do komputera, a następnie pyta gracza czy rzucić jego kostką;
- podczas wykonywania pętli program powinien sumować punkty gracza i komputera;
- liczba punktów komputera powinna być ukryta do czasu zakończenia pętli;

- po zakończeniu pętli program powinien ujawnić punkty komputera - gracz wygrywa, jeżeli uzyskał więcej punktów od komputera, ale nie więcej niż 21.

Zadanie 4 (1 pkt) Utwórz moduł zawierający klasę `Account`. Konstruktor klasy powinien tworzyć chroniony atrybut `balance`, który jest inicjalizowany wartością parametru konstruktora. Zdefiniuj w klasie metody: `pay()` - zwiększa saldo konta o podaną wartość, `take()` - zmniejsza saldo konta o podaną wartość lub informuje o braku środków, `show_balance()` - zwraca wartość chronionego atrybutu `blance`. Napisz metodę specjalną `__str__()`. Przetestuj moduł w prostym programie.

Zadanie 5 (1 pkt) Utwórz moduł zawierający klasę `Smartphone`. Konstruktor klasy powinien tworzyć i inicjalizować trzy prywatne atrybuty: `manufacturer`, `model` i `price`. Zdefiniuj w tej klasie metody ustawiające i wyświetlające wartości atrybutów. Zaimplementuj metodę `__str__()`. Napisz program, który zapisuje do pliku binarnego o nazwie *phones.dat* kilka obiektów klasy `Smartphone` oraz program odczytujący dane z pliku *phones.dat*. Wykorzystaj moduł `pickle` do peklowania/odpeklowywania obiektów klasy `Smartphone` do/z pliku.

Zadanie 6 (1 pkt) Wyobraź sobie, że jesteś *obcym*, który posiada swój kosmiczny statek (może, akurat czujesz się dość obco na tych zajęciach, więc nie będzie to aż tak trudne do wyobrażenia...). Jak każdy zaawansowany statek floty międzygalaktycznej twój statek potrafi wytwarzać potrzebne mu elementy z dostępnej materii (właściwie cały statek jest jednym wielkim przetwornikiem materii), kiedy akurat są potrzebne i niszczyć je kiedy są zbędne (tzn. znów przerabiać na bliżej nieokreśloną materię). Załóżmy, że wyruszasz na standardowy patrol międzyplanetarny. Podczas tej podróży będziesz potrzebował zróżnicowanej ilości silników o określonej mocy. Ruszając z portu kosmicznego potrzebujesz dwóch małych silników manewrowych o mocy równej 50, następnie potrzebujesz dwóch silników o mocy 500 aby rozpędzić statek w celu przejścia w hiperprędkość. Następnie, statek potrzebuje dwóch silników do podróży z hiperprędkością o mocy 400000 każdy. Zaprojektuj klasę `RocketEngine` reprezentując silnik statku. Zdefiniuj w klasie dwa atrybuty statyczne `count` o wartości początkowej 0 oraz `all_power` - całkowita moc silników statku o wartości początkowej 0. Napisz w klasie następujące metody:

- konstruktor klasy (`__init__()`) - powinien tworzyć i inicjalizować trzy atrybuty `name`, `power` oraz `working` (o wartości domyślnej `False`), a także zwiększać o 1 wartość atrybutu `count`;

- `start()` - powinna zwiększać wartość atrybutu klasy `all_power` o wartość atrybutu instancji `power` jeżeli tylko `working` jest ustawiony na `False` i zmieniać jego

wartość na `True`;

- `stop()` - powinna zmniejszać wartość atrybutu klasy `all_power` o wartość atrybutu instancji `power` jeżeli tylko `working` jest ustawiony na `True` i zmieniać jego wartość na `False`;
- `__str__()` - zwraca ciąg tekstowy zawierający podstawowe dane silnika;
- `__del__()` - "destruktor" powinien zmniejszać o 1 wartość atrybutu `count` (gdy będziemy używać polecenia `del`);
- `status()` - statyczna metoda, która wyświetla wartości atrybutów statycznych `count` i `all_power`.

Przeprowadź symulację lotu statku kosmicznego, który manewruje, rozpędza się, przechodzi w hiperpredkość, zwalania i znów manewruje cumując w porcie. Śledź aktualną moc i liczbę silników. Po wyjściu z hiperprędkości zbędne silniki powinny być systematycznie dematerializowane.

Zadanie 7 (1 pkt) Napisz program, który "symuluje" bardzo prosty telewizor, tworząc go jako obiekt. Użytkownik powinien mieć możliwość wprowadzenia numeru kanału oraz zwiększania bądź zmniejszania głośności. Zastosuj mechanizm zapewniający utrzymanie numeru kanału i poziomu głośności w właściwych zakresach - wykorzystaj właściwości.

Zadanie 8 (2 pkt) Napisz program, który będzie symulował opiekę nad wirtualnym zwierzakiem. W tym celu utwórz klasę `Pet`. Konstruktor klasy powinien inicjalizować trzy publiczne atrybuty obiektu klasy `Pet`: `name`, `hunger`, `tiredness`. Za pomocą właściwości kontroluj dostęp i ustawianie atrybutu `name` - minimum trzyliterowy ciąg tekstowy (zawierający tylko litery). Zarówno głód jak i znudzenie powinny mieć na początku wartość domyślną 0 (co będzie powodowało, że na początku twój zwierzak będzie w dobrym nastroju). Utwórz chronioną metodę o nazwie `_passage_of_time()`, która zwiększa (o jeden) poziom głodu i znudzenia zwierzaka. Załóż, że czas mija dla zwierzaka tylko gdy ten coś robi (tj. mówi, bawi się lub je). Utwórz właściwość `mood` – reprezentująca nastrój zwierzaka. Zwierzak jest: szczęśliwy (suma poziomu głodu i znudzenia < 5), zadowolony (suma poziomu głodu i znudzenia w zakresie od 5 do 10), podenerwowany (suma poziomu głodu i znudzenia w zakresie od 11 do 15) i wściekły (suma poziomu głodu i znudzenia > 15). Utwórz metodę `talk()`, która informuje jaki jest nastój zwierzaka. Metoda `eat()` powinna zmniejszać poziom głodu zwierzaka o liczbę przekazaną poprzez parametr `food` (o wartości domyślnej

równej 4). Metoda `play()` powinna zmniejszać poziom znudzenia zwierzaka o liczbę przekazaną poprzez parametr `fun` (o wartości domyślnej równej 4). Każda z trzech metod (`talk()`,`eat()`,`play()`) obiektu klasy `Pet` powinna wywoływać prywatną metodę `_passage_of_time()`. Na koniec napisz główną funkcję programu o nazwie `main()`, w której nastąpi konkretyzacja obiektu klasy `Pet` i pojawi się menu służące do opieki nad zwierzakiem.

Pozwól użytkownikowi na określenie ilości pożywienia podawanego zwierzakowi i czasu poświęconego na zabawę z nimi. Wartości te powinny wpływać na szybkość spadku poziomu głodu i znudzenia u zwierzaka. Utwórz w programie mechanizm pozwalający na pokazanie dokładnych wartości atrybutów obiektu. Zrealizuj to poprzez wyświetlenie obiektu po wprowadzeniu przez użytkownika specjalnej wartości (np. `"xy"`). Wykorzystaj metodę specjalną `__str__()` do klasy `Pet`.

Proponowany podział pracy: pierwsza osoba - klasa `Pet`; druga osoba - program symulujący opiekę nad wirtualnym zwierzakiem (może być w formie osobnej klasy).