Lista zadań nr 1

Zadanie 1 Utwórz klasę `Coin`. Konstruktor tej klasy powinien przyjmować jeden parametr `denomination` (nominał monety - liczba całkowita) i tworzyć dwa atrybuty publiczne:

- `side` - utrzymujący aktualną stronę monety (wartość domyślna to `"orzeł"`);
- `denomination` - nominał monety inicjalizowany wartością parametru konstruktora.

Zdefiniuj dwie metody:

- `throw()` - losowo "zmienia" stronę monety (`"orzeł"` lub `"reszka"`);
- `__str__()` - wyświetla tekstową reprezentację obiektu (strona monety).

Wykonaj następujące dwa zadania:

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 kolejce program 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 2 Utwórz klasę `Dice` reprezentująca kostkę do gry. Zdefiniuj w tej klasie następujące metody:

- konstrukt klasy powinien tworzyć dwa atrybuty prywatne:
  - `sides` - liczba ścian kostki (inicjalizowany wartością parametru konstruktora);
  - `value` - wylosowana liczba oczek (o wartości początkowej równej np. `None`).
- metoda `roll()` - wykonuje rzut kostką i wynik rzutu przypisuje do atrybutu `value`;
- `get_sides()` - zwraca liczbę ścianek;
- `get_value()` - zwraca liczbę oczek na kostce;

- `__str__()` - wyświetla wartości atrybutów;

Napisz program w którym użytkownik będzie grał 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 kośćmi należącymi do komputera, a następnie pyta gracza czy rzucić jego kośćmi;
- 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 3 Utwórz klasę `Account` reprezentująca saldo konta bankowego. Zdefiniuj w tej klasie następujące metody:

- konstruktor klasy, który tworzy atrybuty `balance` i `name` inicjalizowane wartościami parametrów konstruktora - jeżeli wartość parametru `balance` jest ujemna program powinien zgłaszać błąd (wyjątek `ValueError` z wartością np. `'Saldo początkowe nie może być ujemne.'`)
- metoda `deposit()`, która zwiększa saldo konta o przekazaną jej wartość - metoda powinna zgłaszać błąd gdy wpłacana kwota jest ujemna (wyjątek `ValueError` z wartością np. `'Nie można wpłacić ujemnej kwoty.'`);
- metoda `take()`, która zmniejsza saldo konta o podaną wartość lub zgłasza błąd o braku środków na koncie (wyjątek `ValueError` z wartością np. `'Brak środków na koncie.'`);
- `__str__()` - wyświetla m.in wartość atrybutu `blance` i nazwę konta.

Przetestuj klasę w prostym programie z obsługą wyjątków.

Zadanie 4 Napisz klasę `Matrix`, która reprezentuje macierz o zadanych wymiarach. Zdefiniuj w tej klasie następujące metody i właściwości:

- konstruktor klasy, który powinien tworzyć trzy atrybuty prywatne:
  - `_row` - liczba wierszy macierzy (inicjalizowany wartością parametru konstruktora);
  - `_col` - liczba kolumn macierzy (inicjalizowany wartością parametru konstruktora);
  - `_matrix` - lista list reprezentująca macierz (wypełniana jednakowymi wartościami początkowymi `value` - parametr konstruktora o wartości domyślnej `None`).
- właściwość `size()` zwracająca wymiar macierzy jako krotkę;
- metoda prywatna `_check_index()` która zgłasza wyjątek `IndexError` jeżeli przekazane jej wartości wiersza i kolumny są poza zakresem macierzy;
- `get_cell()` - metoda zwracająca element macierzy o zadanych współrzędnych - metoda powinna korzystać z metody `_check_index()` w celu sprawdzenia poprawności indeksów;
- metoda `set_cell()` pozwalająca zmienić wartość komórki macierzy o zadanych współrzędnych - metoda powinna korzystać z metody `_check_index()` w celu sprawdzenia poprawności indeksów;
- metoda `__str__()` wyświetlająca macierz w klasyczny sposób.

Przetestuj klasę w prostym programie.

Zadanie 5 Napisz klasę `Time`, która reprezentuję obiekt czasu. Klasa powinna posiadać następujące metody i właściwości:

- konstruktor klasy o trzech parametrach: `hour`, `minute`, `second` (o wartościach domyślnych równych 0), który tworzy i inicjalizuje trzy atrybuty publiczne odpowiadające parametrom;
- właściwości, które zarządzają każdym z trzech atrybutów i kontrolują ich poprawność:
  - `hour` - liczba całkowita z zakresu 0 - 23;
  - `minute` - liczba całkowita z zakresu 0 - 59;
  - `second` - liczba całkowita z zakresu 0 - 59.

- metoda `set_time()` - ustawia wskazania godzin, minut i sekund wartościami jej przekazanymi (o wartościach domyślnych równych 0);
- metoda `__repr__()` - zwraca reprezentację tekstową wg. `repr()`;
- metoda `__str__()` - zwraca wskazanie czasu w formacie 12-godzinnym (z sufiksem `PM` lub `AM`).

Przetestuj klasę w prostym programie.

Zadanie 6 Napisz program, który będzie symulował opiekę nad wirtualnym zwierzakiem. W tym celu utwórz klasę `Pet`. Klasa powinna posiadać metody i właściwości opisane poniżej:

- konstruktor klasy powinien inicjalizować trzy publiczne atrybuty obiektu klasy `Pet`: `name`, `hunger`, `tiredness` (zarówno głód jak i znudzenie powinny mieć na początku wartość domyślną 0 - co będzie powodowało, że na początku twój zwierzak będzie w dobrym nastroju);
- za pomocą właściwości kontroluj dostęp i ustawianie atrybutu `name` - minimum trzyliterowy ciąg tekstowy (zawierający tylko litery);
- utwórz prywatną 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 twojego 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()`;
- metoda `__str__()` reprezentując zwierzaka.

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 (skorzystaj z metody `__str__()`) po wprowadzeniu przez użytkownika specjalnej wartości (np. `"xy"`).