Lista zadań nr 6

Zadanie 1 (2 pkt) Zdefiniuj dwie klasy klasy `Book` i `Shelf`. Klasa `Book` powinna posiadać:

- konstruktor, który powinien definiować i inicjalizować (parametrami konstruktora) dwa atrybuty publiczne `author` i `title`;
- metodę `search()` , która zwraca `True` jeżeli fraza będąc parametrem metody znajduje ciągu tekstowym `author` lub `title`;
- metodę `__str__()` zwracająca tekst postaci np. `Henryk Sienkiewicz: 'Potop'`.

Klasa `Shelf` powinna posiadać:

- konstruktor, który powinien definiować atrybut `shelf` i inicjalizować go pustą listą lub listą z kolejnymi elementami parametru konstruktora o ile jest on podany (o parametrze konstruktora zakładamy, że jest to obiekt literowany , którego iterator zwraca obiekty klasy `Book`);
- metodę `add()`, która pozwala "dodać" książkę do półki (na półce nie mogą być dwie takie same książki);
- metodę `remove()`, która pozwala "usunąć" książkę z półki (według indeksu);
- metodę `find()`, która tworzy i wyświetla listę książek, których ciąg tekstowy `author` lub `title` zawiera fraza będąc parametrem metody - w tym celu wykorzystaj odpowiednią metodę klasy `Book`;
- metodę `show()`, która wyświetla zawartość półki (wraz z numerem indeksu od zera począwszy)
- metodę `sort()`, która "sortuje półkę" według autora lub tytułu - decyduje o tym parametr metody (`"author"` lub `"title"`)

Utwórz program z menu, który pozwoli zarządzać wirtualną półką.
Proponowany podział pracy: pierwsza osoba - klasa `Shelf`, druga osoba - klasa `Book` program do zarządzania wirtualną półką.

Zadanie 2 (2 pkt) Napisz, program w którym zdefiniowane są dwie klasy `Clock` i `Calendar`. Klasa `Clock` powinna zawierać następujące metody

- konstruktor - definiuje trzy atrybuty `hours`, `minutes`, `seconds` inicjalizowane trzema parametrami o wartościach domyślnych równych zero - użyj deskryptorów do odpowiedniego zabezpieczania poprawności wartości atrybutów ;

- `set()` - metoda pozwalająca na ustawienia wartości atrybutów wartościami jej parametrów
- `tick()` - metoda, której każde wywołanie zwiększa upływ czasu o jedną sekundę (powinna odpowiednio aktualizować wartości atrybutów);
- `display()` -metoda, która powinna pokazywać odmierzony czas (odmierzanie odbywa się poprzez wielokrotne wywołanie metody `tick()`) korzystając z funkcji `print()` - format wyświetlacza siedmiosegmentowego;
- `__str__()` - zwraca ciąg tekstowy reprezentujący odmierzony czas;
- `__repr__()` - standardowa implementacja.

Klasa `Calendar` powinna zawierać następujące metody

- konstruktor - definiuje trzy atrybuty `day`, `month`, `year` inicjalizowane trzema parametrami o wartościach domyślnych równych odpowiednio 1, 1 i 1900 - użyj deskryptorów do odpowiedniego zabezpieczania poprawności wartości atrybutów (rok to liczba z zakresu od 0 do 9999);
- `set()` - metoda pozwalająca na ustawienia wartości atrybutów wartościami jej parametrów;
- `passage_of_time()` - metoda, której każde wywołanie zwiększa upływ czasu o jedną dzień (powinna odpowiednio aktualizować wartości atrybutów);
- `is_leap_year()` - właściwość zwracająca `True` gdy rok jest przestępny, a `False` gdy nie jest;
- `__str__()` - zwraca ciąg tekstowy reprezentujący datę;
- `__repr__()` - standardowa implementacja.

Przetestuj klasy w krótkim programie.
Proponowany podział pracy: pierwsza osoba - klasa `Clock` i jej instancje testowe, druga osoba - klasa `Calendar` i jej instancje testowe.

Zadanie 3 (1 pkt) Napisz program, który służy do symulacji tzw. *problemu kolekcjonera kuponów*.
Podstawowy wariant tego problemu można opisać następująco:

1. mamy n urn;

2. do rozważanych urn wrzucamy kolejno kule;

3. wybór każdej urny jest jednakowo prawdopodobny oraz kolejne wybory są wykonywane niezależnie.

Należy wyznaczyć liczbę rzutów T_n, po której w każdej urnie znajdzie się co najmniej jedna kula. Program powinien umożliwiać wielokrotne przeprowadzanie symulacji. Zdefiniuj klasę `Container` reprezentująca pojedynczą urnę; klasę `Simulation`, która wykona symulacje dla n urn (niech n będzie z zakresu 2 - 100 - wykorzystaj właściwości); klasę `Menu`, która pozwoli na wielokrotne przeprowadzenie symulacji dla danych podanych przez użytkownika i wyświetla odpowiednie menu.

Zadanie 4 (1 pkt) Zdefiniuj klasę `SingleLinkedList`, która reprezentuje listę jednokierunkową z dowiązaniami. W tym celu zdefiniuj klasę `SingleLinkedNode`, która reprezentuje pojedynczy węzeł listy jednokierunkowej. Klasa `SingleLinkedNode` powinna zawierać następujące metody:

- dwuparametrowy konstruktor, który inicjalizuje wartościami parametrów dwa atrybuty: `value` (wartość przechowywana w węźle), `next` (instancja kolejnego węzła);
- metoda `__repr__()` zaimplementowana w klasyczny sposób.

Klas `SingleLinkedList` powinna posiadać następujące metody:

- bezparametrowy konstruktor, który tworzy dwa atrybuty (o początkowych wartościach równych `None`) `begin` i `end` reprezentujące pierwszy i ostatni węzeł listy;
- metoda `push()` - dołącza nowy element na końcu listy;
- metoda `pop()` - usuwa ostatni element listy i zwraca go;
- metoda `unshift()` - usuwa pierwszy element listy i zwraca go;
- metoda `remove()` - znajduje pasujący element (przechowujący daną wartość), a następnie usuwa go z listy;
- metoda `first()` - zwraca pierwszy element listy;
- metoda `find()` - wyszukuje i zwraca element o zadanej wartości;
- metoda `flast()` - zwraca ostatni element listy;
- metoda `count()` - liczy i zwraca liczbę wszystkich elementów listy;

- metoda `get()` - zwraca element listy o zadanym indeksie lub wzbudza wyjątek `IndexError` gdy nie ma tyle elementów na liście lub indeks jest ujemny (elementy listy powinny być liczone za każdym razem w celu wyznaczania elementu o zadanym indeksie);
- metoda `show_list()` - wyświetla kolejne elementy listy od początku do końca.

Przetestuj klasę w prostym programie.

Zadanie 5 (1 pkt) Zdefiniuj klasę `DoubleLinkedList`, która reprezentuje dwukierunkową listę z dowiązaniami. W tym celu zdefiniuj klasę `DoubleLinkedNode`, która reprezentuje pojedynczy węzeł listy dwukierunkowej. Klasa `DoubleLinkedNode` powinna zawierać następujące metody:

- konstruktor o trzech parametrach, który inicjalizuje wartościami tych parametrów trzy atrybuty: `value` (wartość przechowywana w węźle), `next` (instancja kolejnego węzła), `previous` (instancja przedniego węzła);
- metoda `__repr__()` zaimplementowana w klasyczny sposób.

Klas `DoubleLinkedList` powinna posiadać następujące metody:

- bezparametrowy konstruktor, który tworzy dwa atrybuty (o początkowych wartościach równych `None`) `begin` i `end` reprezentujące pierwszy i ostatni węzeł listy;
- metoda `push()` - dołącza nowy element na końcu listy;
- metoda `pop()` - usuwa ostatni element listy i zwraca go;
- metoda `unshift()` - usuwa pierwszy element listy i zwraca go;
- metoda `remove()` - znajduje pasujący element (przechowujący daną wartość), a następnie usuwa go z listy;
- metoda `first()` - zwraca pierwszy element listy;
- metoda `find()` - wyszukuje i zwraca element o zadanej wartości;
- metoda `last()` - zwraca ostatni element listy;
- metoda `count()` - liczy i zwraca liczbę wszystkich elementów listy;
- metoda `get()` - zwraca element listy o zadanym indeksie lub wzbudza wyjątek `IndexError` gdy nie ma tyle elementów na liście lub indeks jest ujemny (elementy listy powinny być liczone za każdym razem w celu wyznaczania elementu o zadanym indeksie);

- metoda `show_list()` - wyświetla kolejne elementy listy od początku do końca.

Przetestuj klasę w prostym programie.

Zadanie 6 (1 pkt) Zdefiniuj klasę `Stack`, która reprezentuje stos. W tym celu zdefiniuj klasę `StackNode`, która reprezentuje pojedynczy element stosu. Klasa `StackNode` powinna zawierać następujące metody:

- dwuparametrowy konstruktor, który inicjalizuje wartościami parametrów dwa atrybuty: `value` (wartość przechowywana w węźle), `next` (instancja kolejnego węzła);
- metoda `__repr__()` zaimplementowana w klasyczny sposób.

Klas `DoubleLinkedList` powinna posiadać następujące metody:

-
- bezparametrowy konstruktor, który tworzy atrybut `top` (o początkowej wartości równej `None`), który reprezentuje element znajdujący się na szczycie stosu;
- metoda `push()` - umiesz nową wartość na górze stosu;
- metoda `pop()` - "zdejmuje" element który jest aktualnie na górze stosu i zwraca wartość tego elementu;
- metoda `top()` - zwraca element znajdujący się na szczycie stosu;
- metoda `count()` - liczy i zwraca liczbę wszystkich elementów stosu.

Napisz funkcję `hanoi()`, która w sposób rekurencyjny rozwiązuje tzw. problem wież z Hanoi. Funkcja powinna przyjmować trzy argumenty - trzy instancje klasy stos (jeden zawierający n elementów i dwa puste), które reprezentują trzy wieże. Wywołanie funkcji powinno spowodować przeniesienie elementów z pierwszego stosu na trzeci stos z pomocą drugiego stosu (oczywiście po przeniesieniu kolejność elementów na trzecim stosie powinna być taka sama jak na stosie pierwszym przed wywołaniem funkcji).

Zadanie 7 Zdefiniuj klasę `Graph`, która reprezentuje graf nieskierowany reprezentowany za pomoca tzw. list sąsiedztwa - w tym celu wykorzystaj słownik, którego kluczami będą wierzchołki grafu (np. litery), a wartościami zbiory wierzchołków sąsiadujących. Klasa `Graph` powinna posiadać następujące metody:

- jednoparametrowy konstruktor, który jako parametr przyjmuje wspominany wyżej słownik (domyślna wartość parametru to `None`) i tworzyć atrybut prywatny `graph_dict` inicjalizowany przekazanym słownik lub słownikiem pustym.

- metoda `vertices()` - zwraca listę wierzchołków grafu;
- metoda `edges()` - zwraca listę krawędzie grafu (dwuelementowych zbiorów);
- metoda `add_vertex()` - dodaje do grafu wierzchołek przekazany jako jej argument;
- metoda `add_edge()` - dodaje do grafu krawędź (dwuelementowa lista, krotka lub zbiór) przekazaną jako jej argument;
- metoda `__str__()` - zwraca ciąg tekstowy, który zawiera wypisane wierzchołki i krawędzie grafu.

Przetestuj klasę w prostym programie.