Projekt na zaliczenie – Python 2

BUDŻET

Anna Łaba

Celem projektu było stworzenie aplikacji, która ułatwi prowadzenie nadzoru nad planowanymi i aktualnymi wydatkami oraz przychodami. Dane dotyczące wprowadzonych do budżetu przychodów oraz wydatków, a także informacje o zmienionych limitach przechowywane są w pliku tekstowym. Zakłada się, że na każdą z kategorii wydatków można nałożyć limity, które nie mogą zostać przekroczone. Budżet nie może również posiadać ujemnego salda.

1. SPIS KLAS

- Category
- Planned Income
- Actual Income
- Planned Outcome
- Actual Outcome
- Budget
- Analysis
- Budget Operation

2. OPIS KLAS

Klasa Category
Ta klasa reprezentuje kategorie wydatków oraz przychodów. Każda z kategorii może (lecz nie musi) posiadać ustalony priorytet. Priorytety w założeniu ustala się dla kategorii wydatków, ponieważ istnieją wydatki pierwszej kategorii, które powinno się spłacać w pierwszej kolejności.

Klasa Planned Income i Actual Income
Jest to klasa reprezentująca przychody. Obiekty klasy Planned Income posiadają następujące atrybuty: *name, start_date, category, value*, odpowiadające kolejno za nazwę przychodu, datę transakcji, kategorię oraz wartość. Atrybut *start_date* powinien być obiektem klasy Date z modułu **datetime**, a *category* obiektem klasy Category. Wartość przychodu oczywiście nie może być ujemna. Klasa Planned Income wyposażona jest w metody pozwalające zwrócić posiadane przez obiekty atrybuty.

Klasa Actual Income dodatkowo posiada atrybut *planned*, który będąc obiektem klasy Planned Income reprezentuje zaplanowany uprzednio przychód. W sytuacji, gdy zaplanowaliśmy jakiś przychód, który po czasie okazał się być większy lub mniejszy mamy możliwość zaktualizowania budżetu o nowy przychód, pamiętający wartość starego. Klasa ta posiada dwie doatkowe metody: *get_planned()* i *check_with_planned()*, która zwraca różnicę pomiędzy wartością planowanego, a aktualnego przychodu. Oczywiście aktualny wpływ nie może zostać wprowadzony na datę późniejszą niż dzisiejsza.

Klasa Planned Outcome i Actual Outcome
Są to klasy analogiczne do Planned Income oraz Actual Income, posiadają te same atrybuty oraz metody, z tą różnicą, że dotyczą wydatków, a nie przychodów.

Klasa Budget
Jedna z bardziej rozbudowanych klas. Przechowuje w pamięci listy wydatków oraz przychodów, 7 kategorii (obiektów klasy Category) – przychody: zarobki, kredyt, pożyczka; wydatki: czynsz, żywność, rozrywka oraz inne. Przechowuje również limity na kategorie wydatków w słowniku. Klasa ta wyposażona jest w wiele metod:
- show incomes() – printuje dodane do budżetu przychody
- show outcomes() – printuje dodane do budżetu wydatki
- get_limit(category) – zwraca informację o limicie na daną categorię
- check_category(category) – zwraca informację o poniesionych do tej pory wydatkach w danej kategorii
- set_limit(category, limit) – ustala wskazany limit na daną kategorię, limit oczywiście nie może być niższy, niż poniesione dotychczas wydatki w tej kategorii
- add_income(income) – funkcja dodająca przychód do budżetu. *Income* oczywiście musi być obiektem klasy PlannedIncome lub ActualIncome. Jeżeli jest to ActualIncome to przed dodaniem przychodu usuwamy z budżetu powiązany z nim PlannedIncome.
- add_outcome(outcome) – funkcja dodająca wydatek do budżetu. *Outcome* oczywiście musi być obiektem klasy PlannedOutcome lub ActualOutcome. Jeżeli jest to ActualOutcome to przed dodaniem przychodu usuwamy z budżetu powiązany z nim PlannedOutcome. Niemożliwe będzie również dodanie wydatku, który posiada wartość wyższą, niż pozostały budżet do wykorzystania w danej kategorii. Dodatkowo, jeżeli dodawany przez nas wydatek będzie z kategorii o priorytecie równym 1, jednak nie będziemy posiadać wystarczających środków by go pokryć, usunięte zostaną planowane wydatki o mniejszym priorytecie, a użytkownik zostanie poinformowany o tym, które wydatki zostały usunięte
- del_income(income) – odpowiada za usuwanie przychodu
- del_outcome(outcome) – odpowiada za usuwanie wydatku
- check_balance(date) – za pomocą tej funkcji możemy sprawdzić bilans na wskazany dzień.

Klasa Analysis
Ta klasa odpowiada za analizę budżetu. Tworząc ją powinniśmy podać budżet (klasa Budget), który chcemy analizować. Wyposażona jest w następujące funkcje:

- ***get balance(date)*** – funkcja zwraca bilans na wskazany dzień (jeżeli wartość planowanych transakcji została zaktualizowana (zmiana z planned na actual), to wezmą pod uwagę wzięte te aktualne)

- *get_planned_balance(date)* – funkcja zwraca bilans na dany dzień (jeżeli wartość planowanych transakcji została zaktualizowana (zmiana z planned na actual), to wezmą pod uwagę wzięte te planowane)

- *plot_balance(start, end)* – funkcja przedstawia na wykresie zmiany w planowanym oraz aktualnym bilansie. Przyjmuje dwa argument: *start* i *end*, odpowiadające za początkową i końcową datę analizowanego okresu. Podane argumenty powinny być klasy Date z modułu **datetime**.

Klasa Budget Operation

Pozwala na sprawne przeprowadzanie operacji na utworzonym budżecie. Przy tworzeniu obiektu tej klasy należy podać ścieżkę dostępu do pliku tekstowego, w którym zapisywane będą przeprowadzone dotychczas na budżecie operacje. Oprócz tego posiada dwa atrybuty, jeden klasy Budget, drugi klasy Analysis. Posiada również dwie funkcje:

- ***decoder()*** – funkcja ta pozwala na odczytanie ze wskazanego pliku tekstowego przeprowadzonych dotychczas operacji. Kategorie dotyczące wydatków zapisane są pod zmienną *catoutcomes*, natomiast te dotyczące przychodów – *catincomes*. Stworzone zostały, by ułatwić odczytywanie zapisanych w pliku informacji. Plik tekstowy otwieramy wyłącznie do odczytu, a operacje zapisane w nim są każda w nowej linijce. Pierwsze słowo podanych sekwencji odpowiada nazwie operacji, dlatego też w zależności od pierwszego argumentu listy *line* (utworzonej po zesplitowaniu każdej z przechowywanych linii), program wykona inna operację. Stworzone jest to z myślą, by przy każdym uruchomieniu budżetu nie tracić przeprowadzonych dotychczas operacji.

- *initialize()* – odpowiada za wszystkie możliwe do wykonania na budżecie operacje. Tutaj również kategorie dotyczące wydatków zapisane są pod zmienną *catoutcomes*, natomiast te dotyczące przychodów – *catincomes*. Tym razem plik tekstowy zostaje otworzony w celu dopisywania do niego nowych informacji. Początkowo przy uruchomieniu użytkownikowi ukazuje się menu z 11 możliwymi do wykonania czynnościami.

1. Sprawdzenie limitów w kategoriach wydatków
2. **Ustawienie nowego limitu w wybranej kategorii**
3. Sprawdzenie wprowadzonych przychodów
4. Sprawdzenie wprowadzonych wydatków
5. **Dodanie nowego przychodu**
6. **Dodanie nowego wydatku**
7. **Usunięcie wybranego przychodu**
8. **Usunięcie wybranego wydatku**
9. Sprawdzenie bilansu na wskazany dzień
10. Przedstawienie zmian w bilansie na wykresie
11. Wyjście

Wytłuszczone operacje ulegają zapisowi w pliku tekstowym.

Funkcja działa do momentu wybrania opcji „end” z menu głównego. Użytkownik wprowadza swój wybór poprzez wpisanie jednej z cyfr 1-11 w terminalu. W zależności od wybranej opcji,

może zostać poproszony o podanie dodatkowych informacji (np. wybór kategorii, data, nazwa i wartość transakcji). Przy wprowadzaniu przychodów i wydatków każdorazowo zostanie zapytany, czy chce zaktualizować wartość którejś z uprzednio zaplanowanych transakcji (jeżeli wartość faktycznej transakcji różni się od tej planowanej). W zależności od wyboru Y/N, zostanie poproszony jedynie o wskazanie powiązanej z aktualną transakcji oraz nowej wartości, lub wszystkich niezbędnych informacji do wprowadzenia nowej transakcji. Przy usuwaniu transakcji, ponownie wyświetlą się wszystkie dotychczas wprowadzone transakcje, a użytkownik zostanie poproszony w wskazanie numerem, którą z transakcji chce usunąć.

Dodatkowo, w osobny pliku znajdują się również testy jednostkowe opisanych klas.

3. MODUŁ DATETIME

Przy tworzeniu projektu skorzystałam z biblioteki **datetime**, dzięki której możliwe są operacje na datach. Konkretne funkcje, z których skorzystałam, to przede wszystkim funkcja:

date(year, month, day),

która reprezentuje datę zgodną z kalendarze gregoriańskim. Daty zapisane w ten sposób jesteśmy w stanie porównywać między sobą (która jest wcześniejsza, która późniejsza) za pomocą zwykłych operatorów >, <, ≥, ≤.

Oprócz tego, skorzystałam również z funkcji **timedelta(),** której podajemy konkretną liczbę dni, a następnie możemy o tyle zwiększyć wskazaną datę.