**Design Patterns. Iterators, Generators, Decorators**

1. **Singleton**

Se da următoarea clasa:

**class** **PresedinteRomania**:

**def** \_\_str\_\_(self):

**return** "Eu sunt presedintele Romaniei"

**def** say\_hello(self):

**return** f'Salut! {self}'



In momentul de fata, dacă încercăm să creăm mai multe obiecte din clasa aceasta, vom putea:

a = **PresedinteRomania**()

b = **PresedinteRomania**()

print(f'ID(a) = {id(a)}')

print(f'ID(b) = {id(b)}')

print(f'Acelasi obiect? {a is b}')



Scopul acestui exercițiu este sa modificăm clasa de mai sus folosind DP Singleton pentru a obține mereu același obiect:

* Vom folosi functia `\_\_new\_\_` (adevăratul constructor din Python)
* Vom tine singurul obiect pe clasa (cls), și îl vom crea doar la prima apelare a lui \_\_new\_\_
* La orice alta apelare, vom returna obiectul deja existent

1. **Factory Pattern**

Acest pattern ne permite sa creăm un obiect dintr-o clasa folosind o alta clasa (fabrica). Fabrica are posibilitatea de a crea obiecte din mai multe clase (de obicei aceste clase sunt siblings, adica mostenesc de la acelasi parinte).

Vom implementa următoarele clase:

* English/French/Spanish Translator – clase care știu sa traduca cuvinte din română în limba specificata
  + translations va fi un dicționar cu acele cuvinte, exemplu `{ “masina”: “car” }` – se poate hardcoda în clasa
  + localize va fi o funcție care pentru un parametru de intrare, ne va da traducerea lui în acea limba (exemplu `input(“masina”)` returneaza “car”)
* TranslatorFactory – clasa care are o singura metoda (preferabil statica sau de clasa) numita get\_translator(language) – in functie de parametrul language, returnează un translator object.

factory = TranslatorFactory()

english\_trans = factory.get\_translator('en')

spanish\_trans = factory.get\_translator('es')

print(f'In engleza zicem {english\_trans.localize("masina")}')

print(f'In spaniola zicem {spanish\_trans.localize("masina")}')



1. **Generators**

Implementați un generator pentru loteria 6/49 și noroc:

* Primele 6 apelări către generator vor da cate un numar intre 1 si 49 (inclusiv
* Ultima apelare va da un singur număr de “noroc” format din 7 cifre

1. **Context Managers**

Se da un fisier text hello.txt, care contine un text scurt. Deschideți fișierul și citiți conținutul, folosind urmatoarele 2 metode:

* try - finally
  + Fișierul se deschide înainte de try, folosind functia open
  + În interiorul try citim conținutul folosind functia read
  + În finally se închide fișierul
* with (context manager)
  + Se va observa ca pentru with nu mai este nevoie sa inchidem noi manual fișierul, pentru ca context managerul face asta pentru noi.

1. **Decorators**

Implementați următorii decoratori:

* timeit – decorator care măsoară timpul de execuție al unei funcții
* logger – decorator care printeaza argumentele de intrare, și rezultatul unei funcții

1. **Decorators extra**

Implementați o clasă User, cu următoarele cerințe:

– constructorul va primi nume, email, parola, și data nașterii

– o metoda login, care va primi email și parola ca parametrii; dacă acestea sunt corecte, userul va fi marcat ca logat

– o metoda get\_info, care va returna toate informațiile despre user DOAR DACĂ acesta este logat, folosind un decorator `@require\_login`

– o metoda logout, fara params, care delogheaza userul.

Se va testa apelarea metodei get\_info inainte de logare, dupa logare, dupa delogare, și după încă o logare.