Lista 8

Zadanie 1 (0,75 punktu). Wyjaśnij rolę następującego dekoratora:

```
def cache1(func):
    internal = {}
    def func_with_cache(arg): # założenie: arg jest hashowalny
        if arg not in internal:
            internal[arg] = func(arg)
        return internal[arg]
    return func_with_cache
```

Następnie podaj (w notacji Θ) złożoność czasową następującej funkcji w zależności od `n` $\geqslant 0$:

```
def f(n):
    if n == 0:
        return 1
    return f(n - 1) + f(n - 1)
```

Podaj, jak zmienia się złożoność, gdy funkcja zostaje udekorowana przez `cache1`.

Zadanie 2 (1 punkt). Funkcjom można nadawać atrybuty. Napisz dekorator `history`, który nadaje dekorowanej funkcji atrybut `history` będący (początkowo pustą) listą. Każde wywołanie udekorowanej funkcji dokłada do listy przekazane funkcji argumenty jako parę (`args`, `kwargs`). Zachowanie funkcji nie powinno przy tym ulec zmianie. Przykładowe użycie:

```
@history
def f(x, y, z):
    return x + y + z

print(f(1, 2, 3)) # 6
f("a", "b", z="c")
f(10, z=30, y=20)

print(f.history)
# [((1, 2, 3), {}), (('a', 'b'), {'z': 'c'}), ((10,), {'z': 30, 'y': 20})]
```

Zadanie 3 (1 punkt). Do symulatora gier w kółko i krzyżyk z wykładu dodaj klasę `ControllerOpportunistic` implementującą `Controller`. Kontroler powinien działać według następującego schematu. Jeśli gracz jest w stanie wygrać rozgrywkę pojedynczym ruchem, wykonujemy ten ruch. W przeciwnym wypadku wykonujemy ruch w sposób losowy.

Zadanie 4 (1,25 punktu). Napisz abstrakcyjną klasę bazową `Predicate`, reprezentującą predykat (formułę) na liczbach całkowitych. Klasa powinna mieć metody:

- `matches(self, n)` – abstrakcyjna metoda zwracająca `True` lub `False` w zależności, czy predykat jest prawdziwy dla `n`.
- `filter(self, lst)` – konkretna metoda, która dla listy liczb `lst` zwraca listę takich elementów, dla których predykat jest prawdziwy.

Następnie napisz klasy `LessThan`, `Not` i `And` implementujące `Predicate` i reprezentujące odpowiednio predykaty postaci $n < c$ (gdzie c jest liczbą całkowitą); $\neg P(n)$ (gdzie P jest predykatem); oraz $P(n) \wedge Q(n)$ (gdzie P i Q to predykaty). Ich inicjalizatory napisz tak, aby były zgodne z poniższym przykładowym użyciem:

```
predicate = And(LessThan(5), Not(LessThan(2))) # odpowiada 2 <= n < 5
lst = predicate.filter([0, 2, 6, 2, 4, 9, 4, 6])
print(lst)  # [2, 2, 4, 4]
```