Programowanie obiektowe w języku Python

L1

Celem tej listy jest zaznajomienie z paradygmatami programowania zorientowanego obiektowo w języku Python. Zadania ilustrują koncepcje klas, obiektów, konstruktorów i modyfikatorów dostępu.

Z1 – Zwierze

Napisz klasę o nazwie `Zwierze`, która będzie zawierała trzy pola — `imie` (string), `wiek` (int), `rodzaj` (string). Obiekt tej klasy skonstruujesz z trzema argumentami, które zostaną przypisane do odpowiednich pól klasy.

Przykład użycia:

```
z1 = Zwierze('Azor', 5, 'Pies')
assert z1.imie == 'Azor'
assert z1.wiek == 5
assert z1.rodzaj == 'Pies'
```

Z2 – Kolory

Napisz program zawierający klasę o nazwie `Colors` z metodą `to_hex`, która zamienia nazwę koloru na odpowiadającą mu wartość HEX. Prywatnym polem tej klasy powinien być słownik zawierający mapowanie nazwy koloru na jego wartość HEX.

```
przykladowe_kolory = {
    "Red" : "#e6194B",
    "Green" : "#3cb44b",
    "Yellow" : "#ffe119",
    "Blue" : "#4363d8",
    "Orange" : "#f58231",
    "Purple" : "#911eb4",
    "Cyan" : "#42d4f4",
    "Magenta" : "#f032e6",
    "Lime" : "#bfef45",
    "Pink" : "#fabebe",
    "Teal" : "#469990",
    "Lavender" : "#e6beff",
    "Brown" : "#9A6324",
    "Beige" : "#fffac8",
    "Maroon" : "#800000",
    "Mint" : "#aaffc3",
    "Olive" : "#808000",
    "Apricot" : "#ffd8b1",
    "Navy" : "#000075",
    "Grey" : "#a9a9a9",
    "White" : "#ffffff",
    "Black" : "#000000",
}
```

Przykład użycia:

```
c = Colors()
assert c.to_hex("Black") == "#000000"
assert c.to_hex("TeaL") == "#469990"
assert c.to_hex("YELLOW") == "#ffe119"
```

Z3 – Prostokat

Stwórz nowy projekt i napisz program z klasą o nazwie `Prostokat`, której obiekt skonstruujesz (funkcja `__init__()`) z wartościami `długości(a)` i `szerokości(b)`.
Klasa ta powinna zawierać metodę (funkcję) o nazwie:

- **`pole()`**, która zwraca obliczone pole prostokąta
- **`obwod()`**, która zwraca obwód prostokąta.

Z4 – Ciąg geometryczny

Stwórz klasę `CiagGeometryczny` zawierającą poniższe metody:

- **`konstruktor`**`(self, a1, q, n)`: wymaga podania trzech danych przy inicjalizacji (`a1` — pierwszy wyraz ciągu, `q` — iloraz, `n` — początkowa liczba wyrazów ciągu tworzona podczas konstrukcji obiektu)
- **`add`**`(self)`: która dodaje kolejny wyraz ciągu,
- **`print`**`(self)`: która wypisuje wszystkie przechowywane wyrazy.
- **`rozmiar`**`(self)`: która zwróci liczbę przechowywanych wyrazów ciągu.

Dla przypomnienia wzór na kolejny wyraz ciągu geometrycznego: $a_n = q \cdot a_{n-1}$

Napisz testy jednostkowe z użyciem słowa kluczowego `assert`.

Z5 – Czas

Stwórz nowy projekt i napisz klasę `Czas` służącą do zapamiętania okresu tj. liczby godzin i minut.
Klasa ta powinna mieć następujące metody publiczne:

• `konstruktor(self, godzin, minut)` (funkcja `__init__()`) z parametrami będącymi liczbą godzin i minut, gdy zostanie podany jeden parametr należy potraktować go jako łańcuch znaków na podstawie którego można ustalić wartość godzin i minut np. ”12 h 58 min” (funkcja `split()` pozwala dzielić napis według określonego separatora: https://www.w3schools.com/python/ref_string_split.asp)
• `dodaj(self, inny)` której wynikiem jest nowy obiekt klasy `Czas` będący sumą bieżącego i podanego jako parametr obiektu Czas (dodaj odpowiednie metody)
• `odejmij(self, inny)` analogicznie jak dodaj, tyle że odejmowanie,
• `pomnoz(self, ile)` wynikiem ma być okres pomnożony podaną liczbę razy.

Metody specjalne:
• `__str__(self)` której wynikiem jest łańcuch znaków opisujący dany okres, np. ”29 h 19 min”.

Pozostałe pola klasy powinny być prywatne.

*Operację dodawania, odejmowania i mnożenia obsłuż za pomocą operatorów +, -, * (funkcje specjalne* `__add__()`, `__sub__()`, `__mul__()`).

Przykładowe operacje:

```
czas1 = Time("12 h 58 min")
czas2 = Time("12 h", "58 min")
czas3 = Time(12, 58)


#czas6 przechowuje godzinę i minuty aktualnie pobrane z systemu operacyjnego
czas6 = Time()

#czas4 i czas5 będą przechowywać tą samą wartość
czas4 = czas1.dodaj(czas2)
czas5 = czas1 + czas2
```