Programowanie Obiektowe
Instrukcja do laboratorium 1

1 Zasady oceniania

Zadanie	Ocena
Samochód (3.1)	3,0
Konto bankowe (3.2)	3,5
Elektrownie (3.3)	4,0
Ułamek (3.4)	5,0

Warunkiem uzyskania oceny za zadanie n jest wykonanie poprzednich zadań.

W przypadku nie oddania zadania w terminie (tydzień po zajęciach), uzyskana ocena jest zmniejszana o 0,5 za każdy tydzień opóźnienia.

Plik (wkleić cały kod do jednego pliku, podpisać poszczególne zadania za pomocą komentarzy) ze zrobionymi zadaniami proszę przesłać na platformie Moodle. Plik musi mieć nazwe `numeralbumu_lab1.py`. **Plik musi być plikiem tekstowym z rozszerzeniem `.py`**.

UWAGA: Termin oddania zadania jest ustawiony w systemie moodle. W przypadku nie oddania zadania w terminie, uzyskana ocena będzie zmniejszana o 0,5 za każdy zaczęty tydzień opóźnienia. Zadania oddawane później niż miesiąc po terminie ustawionym na moodle są oddawane i rozliczane w trybie indywidualnym na zajęciach lub po umówieniu się z prowadzącym.

UWAGA: W przypadku wysłania zadania w formie niezgodnej z opisem w instrukcji prowadzący zastrzega prawo do wystawienia oceny negatywnej za taką pracę. Przykład: wysłanie `.zip` lub `.pdf` tam, gdzie był wymagany plik tekstowy z rozszerzeniem `.py`.

2 Materiał pomocniczy

2.1 Obiekt

Obiekt jest pewną strukturą, która łączy dane i funkcje przetwarzające te dane. Klasy są szablonami, z których są tworzone obiekty. Przykładowo, zdefiniujemy następującą klasę:

```
1  class Klasa:
2      # Zmienna klasy, która będzie dostępna bezpośrednio
3      # z klasy oraz z każdego obiektu tej klasy
4      zmienna_klasy = 'to jest zmienna klasy Klasa'
5
6      # Konstruktor obiektu (wywoływany przy tworzeniu obiektu
7      def __init__(self):
8          # Zmienna obiektu
9          self.zmienna = 'to jest zmienna obiektu'
10
11     def funkcja(self):
12         # Metoda wypisująca pewien ciąg
13         print(f'Wartość zmiennej to "{self.zmienna}"')
```

Kolejny listing pokazuje jak możemy stworzyć obiekt tej klasy, oraz w jaki sposób uzyskujemy i zmieniamy wartości atrybutów które ten obiekt posiada.

```
>>> ob = Klasa()
>>> ob.zmienna_klasy
'to jest zmienna klasy Klasa'
>>> ob.zmienna
'to jest zmienna obiektu'
>>> ob.funkcja()
Wartość zmiennej to "to jest zmienna obiektu"
>>> ob.zmienna = 'nowa wartosc'
>>> ob.funkcja()
Wartość zmiennej to "nowa wartosc"
```

2.2 Co to jest `self`?

`self` jest odniesieniem do obiektu klasy. Każda metoda klasy, przeprowadzająca pewne działania na obiekcie posiada `self` jako pierwszy argument(przykład: w konstruktorze `__init__` przypisujemy do `self` pewne zmienne). Przykładowo dodawanie możemy zrealizować w sposób następujący:

```
class ExampleAdd:

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add_two_variable(self):
        return self.a + self.b


def add_two_variable(a, b):
    return a + b

>>> obj = ExampleAdd(2,4)
>>> obj.add_two_variable()
6
>>> add_two_variable(2,4)
6
```

W przypadku użycia funkcji *add_two_variable* zawsze jesteśmy zobligowani podać zmienne, natomiast w przypadku metody *add_two_variable* pochodzącej z klasy **ExampleAdd** wystarczy podać do konstruktora raz zmienne i ją wywoływać. Bez użycia `self` nie jesteśmy w stanie ustalić modyfikatorów dostępu pól oraz metod klasy. Słowo kluczowe `self` jest słowem ogólnie przyjętym, dlatego też można (co jest niewskazane) użyć dowolnego słowa.

2.3 Metody magiczne

Metody magiczne to takie metody klasy, które są zdolne do implementowania pewnych operacji. Służą one do przeciążania operatorów. Lista metod magicznych wraz z ich opisem umieszczona jest na stronie z dokumentacją oraz stronie z wyróżnionymi operatorami.

Przykładowe metody magiczne:

- `__add__(self, b)` - arytmetyczna operacja dodawania (przeciążenie operatora '+')
- `__mul__(self, b)` - arytmetyczna operacja mnożenia (przeciążenie operatora '*')
- `__eq__(self, b)` - operacja porównania obiektu (przeciążenie operatora '==')
- `__len__(self)` - długość obiektu
- `__str__(self)` - ciąg znaków reprezentujących obiekt

- `__repr__(self)` - reprezentacja obiektu

Przykład różnicy między `__repr__(self, b)`, a `__str__(self, b)`:

```
>>> import numpy as np
>>> vec = np.array([1,2,3])
>>> repr(vec)
'array([1, 2, 3])'
>>> str(vec)
'[1 2 3]'
```

2.4 Przeciążenia operatorów

Przeciążenie operatorów jest operacją pozwalającą na przechwytywanie obiektów danej instancji w celu wykonania na nich działania. W przypadku braku przeciążenia operatora wystąpi błąd.

Przykład:

```
>>> obj = Example(1, 2, 3)
>>> obj_other = Example(4, 5, 6)
>>> obj + obj_other
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'Example' and 'Example'
```

W celu dodania dwóch obiektów tej samej klasy powinniśmy przeciążyć operator dodawania. Poniżej zamieszczono klasę posiadającą jedno pole w której przeciążono operator dodawania:

```
class ExampleAdding:
    def __init__(self, a):
        self.pole_publiczne = a

    def __add__(self, other):
        return ExampleAdding(self.pole_publiczne + other.pole_publiczne)
```

Za pomocą listingu 2.4 przedstawiono przykładowe dodawanie dwóch obiektów w których przeciążono operator dodawania:

```
>>> obj = ExampleAdding(4)
>>> obj_other = ExampleAdding(8)
>>> obj_created = obj + obj_other
>>> obj_created.pole_publiczne
12
```

3 Zadania do wykonania

3.1 Samochód i droga

Stwórz klasę o nazwie "Samochod", która będzie miała atrybuty takie jak marka, model, rok produkcji oraz prędkość maksymalna. Dodaj konstruktor, który będzie przyjmował wartości tych atrybutów, a także destruktor, który będzie wyświetlał informację o zniszczeniu obiektu.

Stwórz klasę o nazwie "Droga", która będzie miała atrybuty takie jak rodzaj oraz maksymalna prędkość. Dodaj konstruktor, który będzie przyjmował wartości tych atrybutów.

Dodatkowo do klasy "Samochod", dodaj metodę o nazwie "jedź", która będzie przyjmowała wartość prędkości oraz obiekt klasy "Droga"i wypisywała komunikat o tym, że samochód jedzie z podaną prędkością oraz o ile km/h przekracza maksymalną prędkość w zależności od obiektu "Droga".

Przykład:

```
>>> moj_samochod = Samochod("Ferrari", "250 GTO", 2019)
>>> moja_droga = Droga("Autostrada", 140)
>>> moj_samochod.jedz(200, moja_droga)
Samochód marki Ferrari, model 250 GTO z 2019 roku jedzie z prędkością 200 km/h.
Przekraczasz maksymalną prędkość o 60 km/h na drodze rodzaju Autostrada!
```

3.2 Konto bankowe

Stwórz klasę o nazwie "KontoBankowe", która będzie miała atrybuty takie jak numer konta, imię właściciela, nazwisko właściciela oraz saldo. Dodaj konstruktor, który będzie przyjmował wartości tych atrybutów oraz destruktor, który będzie wyświetlał informację o usunięciu obiektu.

Dodatkowo, dodaj metody o nazwach "wpłata"i "wypłata", które będą dodawać lub odejmować od salda odpowiednią kwotę. Metoda "wpłata"powinna przyjmować kwotę wpłaty, a metoda "wypłata"powinna sprawdzać, czy na koncie jest wystarczająca ilość środków, aby dokonać wypłaty i wypłacać tylko wtedy, gdy saldo na koncie jest większe niż kwota wypłaty.

Przykład:

```
>>> moje_konto = KontoBankowe("123456789", "Jan", "Kowalski", 1000.0)
>>> moje_konto.wpłata(500.0)
Wpłata na konto o numerze 123456789 została wykonana. Aktualne saldo: 1500
>>> print(moje_konto.saldo)
1500
>>> moje_konto.wypłata(1500.0)
Wypłata z konta o numerze 123456789 została wykonana. Aktualne saldo: 0
>>> print(moje_konto.saldo)
0
Konto o numerze 123456789 należące do Jan Kowalski zostało usunięte.
```

3.3 Elektrownie

Stwórz klasę "EnergiaOdnawialna", która będzie reprezentować źródło energii odnawialnej. Klasa powinna mieć atrybuty nazwa (nazwa źródła), moc (moc w megawatach) oraz lokacja (lokalizacja źródła). Klasa powinna także mieć konstruktor, który przyjmuje argumenty nazwa, moc i lokacja i ustawia je jako atrybuty obiektu.

Zaimplementuj metodę `get_info`, które zwróci informacje o elektrowni. Dodatkowo zaimplementuj metodę magiczną `__eq__` do porównań. Rozszerz klasę o metody związane z wyświetlaniem.

Przykład:

```
>>> elektrownia_wiatrowa = EnergiaOdnawialna("Wiatr", 50, "Niemcy")
>>> elektrowania_sloneczna = EnergiaOdnawialna("Slonce", 30, "Polska")
>>> elektrownia_wiatrowa.get_info()
Źródło: Wiatr, Moc: 50 MW, Lokalizacja: Niemcy
>>> elektrowania_sloneczna.get_info()
Źródło: Slonce, Moc: 30 MW, Lokalizacja: Polska
>>> elektrowania_hybrydowa = elektrownia_wiatrowa + elektrowania_sloneczna
>>> elektrowania_hybrydowa.get_info()
Źródło: Wiatr, Slonce, Moc: 80 MW, Lokalizacja: Polska, Niemcy
>>> print(elektrownia_wiatrowa)
Źródło: Wiatr, Moc: 50 MW
>>> print(elektrownia_wiatrowa == elektrowania_sloneczna)
False
```

3.4 Ułamek

Napisać klase, reprezentującą ułamek (fraction) w postaci $\frac{a}{b}$, gdzie a jest licznikiem, b mianownikiem. W wyniku ma powstać kod klasy spełniający poniższe warunku oraz kod demonstrujący działanie zaimplementowanej klasy.

Powstała klasa ma spełniać następujące warunki:

- Tworzymy obiekt z dwóch liczb (licznik i mianownik). Przy tworzeniu klasy ułamek ma zostać skrócony (można użyć metody `gcd` z modułu `math`). Przy próbie podania mianowniku 0, wyrzucamy wyjątek `ZeroDivisionError`.

  Przykład:

```
>>> Fraction(3, 4)
Fraction(3, 4)
>>> Fraction(6, 12)
Fraction(1, 2)
```

- Dwie metody do wypisywania obiektów naszej klasy: `__repr__` i `__str__`. Reprezentacja musi zwracać polecenia którym można utworzyć nasz obiekt, a zamiana na string pokazywać postać ułamkową (patrz przykład).

  Przykład:

```
>>> f = Fraction(3, 4)
>>> print(repr(f))
Fraction(3, 4)
>>> print(f) # Niejawnie wywołane __str__
3/4
>>> f2 = Fraction(5, 4)
>>> print(f2) # W przypadku gdy mamy ułamek nie właściwy to oddzielamy część całą
1 1/4
```

- Obiekty klasy muszą wspierać podstawowe operacje arytmetyczne (przeciążenia operatorów): dodawanie, odejmowanie, mnożenie, dzielenie, wartość bezwzględna. Przy wykonaniu operacji ma powstać nowy obiekt. Wszystkie operacje wykonujemy tylko na obiektach naszej klasy (nie trzeba implementować mnożenia przez liczbę i t.d.).

  Przykład:

```
>>> Fraction(1, 4) + Fraction(2, 4)
Fraction(3, 4)
```

- Obiekty tej klasy można porównywać (6 operatorów do przeciążenia):

  Przykład:

```
    >>> Fraction(1, 4) < Fraction(2, 4)
    True
    >>> Fraction(1, 4) == Fraction(2, 4)
    False
```

- Klasa musi mieć zaimplementowane metody rzutowania na inne typy danych (`float`, `int`, `bool`).
- Klasa musi posiadać implementacje metody `__round__`, żeby umożliwić zaokrąglanie jednocześnie z rzutowaniem na `float`.