# Lista zadań nr 5

**Zadanie 1** (2 pkt) Napisz kod źródłowy w Pythonie definiujący dwie klasy `Point` i `Circle`. Klasa `Point` powinna być oparta na klasie bazowej `object`. Konstruktor tej klasy powinien definiować dwa atrybutu `x` i `y` będące współrzędnymi punktu. Wykorzystaj właściwości, aby zabezpieczyć atrybuty przed przypisaniem im niewłaściwych typów danych. Zdefiniuj w klasie `Point` metodę `distance_from_origin()` zwracającą odległość punktu od środka układu współrzędnych. Klasa ta powinna także zawierać własne metody specjalne `__str__()`, `__rep__()` oraz `__eq__()` służące odpowiednio do: wyświetlania reperacji w postaci łańcucha znaków (np. wyświetlenie za pomocą funkcji `print()` punktu `p = Point(1,2)` powinno skutkować pojawieniem się ciągu: `(1,2)`), wyświetlania tzw. reprezentacji obiektu klasy `Point` - metoda `__rep__()`, porównywania dwóch obiektów klasy `Point`.

Klasa `Circle` powinna być oparta na klasie bazowej `Point`. Klasa `Circle` powinna mieć zmodyfikowany konstruktor bazujący na konstruktorze klasy `Point` (wykorzystaj funkcję `super()`) ze definiowanym dodatkowym atrybutem `radius`. Wykorzystaj właściwości, aby zabezpieczyć atrybut `radius` przed przypisaniem mu niewłaściwego typu danych (promień koła musi być liczbą dodatnią). W klasie `Circle` zdefiniuj następujące metody: `edge_distance_from_origin()` - zwraca odległość brzegu koła od środka układu współrzędnych; `area()` – zwraca pole koła; `circumference()` – zwraca obwód koła; `__eq__()` – służąca do porównywania dwóch obiektów klasy `Circle` (metoda powinna wywoływać za pomocą funkcji `super()` metodę klasy bazowej `__eq__()` i modyfikować ją); `__str__()` i `__rep__()`) – powinny zwracać ciąg tekstowy postaci np. `Circle(3,1,2)` dla wartości atrybutów `radius = 3`, `x = 1` i `y = 2` (metoda `__str__()` może np. wywoływać metodę `__rep__()`).

W definicjach obu klas możesz wykorzystać moduł `math`.

**Proponowany podział pracy:** pierwsza osoba - klasa `Circle` i jej instancje, druga osoba - klasa `Point` i jej instancje.

**Zadanie 2** (1 pkt) Zmodyfikuj klasę `Point` z zadania 1 taki sposób, aby obsługiwała poniższe operacje, gdzie `a`, `b` i `c` to obiekty klasy `Point`, a $n$ jest liczbą:

- a = b + c (`Point.__add__()`);
- a += b (`Point.__iadd__()`);
- a = b - c (`Point.__sub__()`);
- a -= b (`Point.__isub__()`);

- a = b * $n$ (`Point.__mul__()`);
- a *= $n$ (`Point.__imul__()`)
- a = b / $n$ (`Point.__truediv__()`);
- a /= $n$ (`Point.__itruediv__()`)
- a = b // $n$ (`Point.__floordiv__()`);
- a //= $n$ (`Point.__ifloordiv__()`).

**Zadanie 3** (1 pkt) Zaprojektuj klasę `Fraction`, która posiada:

- konstruktor klasy, który powinien definiować dwa atrybuty `numerator` (licznik) i `denominator` (mianownik) - liczby całkowite inicjalizwowane parametrami konstruktora. Konstruktor powinien odpowiednio uproszczać licznik i mianownik, a znak ułamka powinien przechowywać licznik.
- metody do obsługi podstawowych operatorów porównywania;
- metody do obsługi operatora dodawania, odejmowania, mnożenia i dzielenia
- metodę `__str__()`, `__repr__()` oraz `__float__()` oraz `__getitem__()`.

Napisz krótki program testujący zdefiniowaną klasę.

Spróbuj zabezpieczyć atrybuty `numerator` i `denominator` przed inicjalizacją nieodpowiednimi danymi (oczywiście będzie to wymagało zmian w szczegółach implemnetacyjnych klasy).

**Zadanie 4** (1 pkt) Zaimplementuj klasę `Polynomial`, która będzie reprezentować wielomian o współczynnikach rzeczywistych, z odpowiednim konstruktorem, operacjami dodawania, odejmowania, mnożeniem (wykorzystaj tzw. iloczyn Cauchy'ego), mnożenia przez skalar oraz wypisywaniem (postaci np. $5.5x^8 + 6x^2 + 6$). Obiekty tej klasy powinny być wywoływalne i dla danego argumentu powinny zwracać wartość wielomianu w tym punkcie. Przetestuj zaprojektowaną klasę.

**Zadanie 5** (1 pkt) Zaprojektuj klasę `QuadraticEquation` reprezentującą równanie kwadratowe, która posiada:

- konstruktor klasy, który powinien definiować trzy atrybuty publiczne `a`, `b` i `c` - liczby rzeczywiste inicjalizwowane parametrami konstruktora (wartość atrybutu `a` musi być różna od zera). Wykorzystaj właściwości aby zapewnić poprawne wartości dla atrybutów;

- właściwość `delta`, która zwraca wartość delty dla równania kwadratowego;
- metodę `roots()`, która zwraca krotkę zawierającą pierwiastki równania kwadrartowego $ax^2 + bx + c = 0$ (krotka powinna zawierać $2$, $1$ lub $0$ elementów);
- metedę `factored_form()`, która wyświetla ciąg tekstowy reprezentujący postać iloczynową równania kwadratowego lub ciąg tekstowy: `'postać iloczynowa nie istnieje'`;
- metodę `__call__()`, która oblicza wartość wyrażenia $ax^2 + bx + c$ dla podanego $x$ i zwraca ją - instancja klasy będzie wywoływalna;
- metodę `__add__()` pozwalającą na dodawanie dwóch obiektów klasy;
- metodę `__sub__()` pozwalającą na odejmowanie dwóch obiektów klasy;
- metodę `__mul__()` pozwalającą na mnożenie równania przez skalar;
- metodę `__str__()` zwracająca równanie kwadratowe w postaci $y = ax^2 + bx + c$;
- metodę `__repr__()` zwracająca równanie w następującej postaci `QuadraticEquation(a, b, c)`.

Zaimplementuj i wykorzystaj klasę `Menu` do zarządzania programem wykorzystującym klasę `QuadraticEquation`.

**Zadanie 6** (1 pkt) Zaprojektuj i zaimplementuj klasę `Permutation` reprezentująca permutacje zbioru $\{0, 1, 2, \ldots, n-1\}$ dla $n > 0$.

- Instancja klasy `Permutation` powinna być inicjalizowana listą wartości permutacji listy (`list(range(n))`). W przypadku nieoprawnej listy inicjalizującej powinien zostać podniesiony wyjątek `ValueError`.
- Przeciąż operator mnożenia (`__mul__()`) jako operator składania dwóch permutacji. Próba składania permutacji o rożnych długościach powinno skutkować podniesieniem wyjątku `TypeError`.
- Przeciąż operator potęgowania jako składanie Permutacji samej ze sobą - wykorzystaj algorytm szybkiego potęgowania.
- Przeciąż operator negacji `__neg__()`, aby `-p` oznaczało permutację odwrotną do `p`.

- Instancje klasy powinny być wywoływalne. W tym celu zaimplementuj metodę `__call__()` tak, aby działając na permutacjip dla dowolnej sekwecji `s` (tj. `p(s)`) otrzymywać listę elementów sekwencji `s` przepermutowaną permutacją `p`. Próba działania permutacji na sekwencji o niezgodnej długości powinna kończyć się podniesieniem wyjątku `ValueError`.
- Zaimplementuj standardowe metody `__repr__()` i `__str__()`.

**Przykładowe wywołania w sesji interaktywnej:**

```
>>> p = Permutation([1,0,4,2,3,5])
>>> p
Permutation([1, 0, 4, 2, 3, 5])
>>> print(p)
[1, 0, 4, 2, 3, 5]
>>> q = Permutation([3,4,0,1,5,2])
>>> p * q
Permutation([2, 3, 1, 0, 5, 4])
>>> -p
Permutation([1, 0, 3, 4, 2, 5])
>>> -q
Permutation([2, 3, 5, 0, 1, 4])
>>> p * -p
Permutation([0, 1, 2, 3, 4, 5])
>>> q ** 0
Permutation([0, 1, 2, 3, 4, 5])
>>> q ** 1
Permutation([3, 4, 0, 1, 5, 2])
>>> q ** 2
Permutation([1, 5, 3, 4, 2, 0])
>>> q ** 5
Permutation([2, 3, 5, 0, 1, 4])
>>> p("abcdef")
['b', 'a', 'd', 'e', 'c', 'f']
>>> q(['ala', 2, 'kot', (3,4), [1,2], {3,'a'}])
['kot', (3, 4), {'a', 3}, 'ala', 2, [1, 2]]
```

**Zadanie 7** (1 pkt) Zaprojektuj i zaimplementuj klasę `Quaterion` reprezentującą kwaterniony - obiekty (liczby) postaci $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ (tzw. postać algebraiczna kwaternionu), gdzie $a, b, c, d \in \mathbb{R}$ natomiast **i**, **j**, **k** są pewnymi obiektami (jednostkami urojonymi) podobni do $i$ w liczbach zespolonych gdyż zachodzi związek $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$. Dodawania i mnożenie kwaternionów w postaci algebraicznej, wykonuje się jak na wielomianach trzech zmiennych **i**, **j**, **k** z tym, że mnożenie jednostek **i**, **j**, **k** z uwzględnieniem ich kolejności określa tabla (mnożenie kwaternionów nie jest przemienne) Instancję klasy

| $\times$ | $1$ | **i** | **j** | **k** |
|---|---|---|---|---|
| $1$ | $1$ | **i** | **j** | **k** |
| **i** | **i** | $-1$ | **k** | $-\mathbf{j}$ |
| **j** | **j** | $-\mathbf{k}$ | $-1$ | **i** |
| **k** | **k** | **j** | $-\mathbf{i}$ | $-1$ |

`Quaterion` powinno się dać inicjalizować w następujący sposób:

- przez liczbę rzeczywistą (dowolny typ, który ma być konwertowany na `float`);
- przez liczbę zespoloną;
- przez cztery liczby rzeczywiste (dowolny typ, który ma być konwertowany na `float`).

Wywołanie konstruktora z inną liczbą augmentów lub argumentami niewłaściwego typu powinno skutkować podniesieniem wyjątku `ValueError`.

Przeciąż operatory dodawania, odejmowania, mnożenia i dzielenia (również dla liczb całkowitych, zespolonych i rzeczywistych). Zaimplementuj odpowiednie metody lustrzane dla wspominanych działań (pamiętaj że mnożenie dzielenie i odejmowanie kwaternionów nie jest przemienne!). Zaimplementuj metodę `conjugate()` zwracającą kwaternion sprzężony do danego. Zaimplementuj metodę `__str__()` zwracającą ciąg tekstowy reprezentujący kwaternion w postaci algebraicznej oraz standardową metodę `__repr__()`.

**Przykładowe wywołania w sesji interaktywnej:**

```
>>> x = Quaternion(4,3,1,-2)
>>> x
Quaternion(4.0,3.0,1.0,-2.0)
>>> print(x)
4.0+3.0i+1.0j-2.0k
>>> x.r
4.0
>>> x.j
```

```
1.0
>>> x.k
-2.0
>>> y = Quaternion(3 + 2j)
>>> y
Quaternion(3.0,2.0,0.0,0.0)
>>> z = Quaternion(-7)
>>> z
Quaternion(-7.0,0.0,0.0,0.0)
>>> x + y
Quaternion(7.0,5.0,1.0,-2.0)
>>> x * 2
Quaternion(8.0,6.0,2.0,-4.0)
>>> 2 * x
Quaternion(8.0,6.0,2.0,-4.0)
>>> 1 + y
Quaternion(4.0,2.0,0.0,0.0)
>>> y + 5
Quaternion(8.0,2.0,0.0,0.0)
>>> y / 3
Quaternion(1.0,0.6666666666666666,0.0,0.0)
>>> 3 / y
Quaternion(0.6923076923076923,-0.46153846153846156,0.0,0.0)
>>> x - y
Quaternion(1.0,1.0,1.0,-2.0)
>>> x - 3
Quaternion(1.0,3.0,1.0,-2.0)
```