**Python Lab 3 Nikita Halaiko**

**Ćwiczenie 1: Konto bankowe**

class BankAccount:

def \_\_init\_\_(self, balance=0):

self.balance = balance

def deposit(self, amount):

self.balance += amount

def withdraw(self, amount):

if self.balance >= amount:

self.balance -= amount

return True

return False

# Przykład użycia:

account = BankAccount(100)

account.deposit(50)

print(account.balance) # 150

print(account.withdraw(200)) # False

print(account.balance) # 150

**Ćwiczenie 2: Liczba zespolona**

import math

class Zespolona:

def \_\_init\_\_(self, re, im):

self.re = re

self.im = im

def modul(self):

return math.sqrt(self.re\*\*2 + self.im\*\*2)

@staticmethod

def dodaj(z1, z2):

return Zespolona(z1.re + z2.re, z1.im + z2.im)

@staticmethod

def mnoz(z1, z2):

return Zespolona(z1.re \* z2.re - z1.im \* z2.im, z1.re \* z2.im + z1.im \* z2.re)

# Przykład użycia:

z1 = Zespolona(3, 4)

z2 = Zespolona(1, 2)

print(z1.modul()) # 5.0

z3 = Zespolona.dodaj(z1, z2)

print(z3.re, z3.im) # 4 6

z4 = Zespolona.mnoz(z1, z2)

print(z4.re, z4.im) # -5 10

**Ćwiczenie 3: Ułamek**

from math import gcd

class Ulamek:

def \_\_init\_\_(self, licznik, mianownik):

self.licznik = licznik

self.mianownik = mianownik

def skroc(self):

dzielnik = gcd(self.licznik, self.mianownik)

self.licznik //= dzielnik

self.mianownik //= dzielnik

@staticmethod

def dodaj(u1, u2):

licznik = u1.licznik \* u2.mianownik + u2.licznik \* u1.mianownik

mianownik = u1.mianownik \* u2.mianownik

return Ulamek(licznik, mianownik)

# To samo co metody subtract(), multiply() i dzielenia().

# Przykład użycia:

u1 = Ulamek(1, 2)

u2 = Ulamek(2, 3)

u3 = Ulamek.dodaj(u1, u2)

u3.skroc()

print(u3.licznik, u3.mianownik) # 7 6

**Ćwiczenie 4: Ciąg Fibonacciego**

import timeit

import matplotlib.pyplot as plt

# Funkcja iteracyjna

def fibonacci\_iteracyjnie(n):

if n <= 1:

return n

a, b = 0, 1

for \_ in range(2, n + 1):

a, b = b, a + b

return b

# Funkcja rekurencyjna

def fibonacci\_rekurencyjnie(n):

if n <= 1:

return n

return fibonacci\_rekurencyjnie(n - 1) + fibonacci\_rekurencyjnie(n - 2)

# Porównanie czasu działania

czasy\_iteracyjne = []

czasy\_rekurencyjne = []

zakres = range(30)

for n in zakres:

czas\_iteracyjny = timeit.timeit(lambda: fibonacci\_iteracyjnie(n), number=10)

czas\_rekurencyjny = timeit.timeit(lambda: fibonacci\_rekurencyjnie(n), number=10)

czasy\_iteracyjne.append(czas\_iteracyjny)

czasy\_rekurencyjne.append(czas\_rekurencyjny)

# Wykres

plt.plot(zakres, czasy\_iteracyjne, label="Iteracyjna")

plt.plot(zakres, czasy\_rekurencyjne, label="Rekurencyjna")

plt.xlabel("n")

plt.ylabel("Czas [s]")

plt.legend()

plt.title("Porównanie czasu działania funkcji Fibonacciego")

plt.show()

**Ćwiczenie 5: Domowe Zoo**

class Zwierze:

def \_\_init\_\_(self, imie, wiek, gatunek):

self.imie = imie

self.wiek = wiek

self.gatunek = gatunek

def przedstaw\_sie(self):

print(f"Jestem {self.gatunek}. Mam na imię {self.imie}, mam {self.wiek} lat.")

class Pies(Zwierze):

def \_\_init\_\_(self, imie, wiek):

super().\_\_init\_\_(imie, wiek, "pies")

class Chomik(Zwierze):

def \_\_init\_\_(self, imie, wiek):

super().\_\_init\_\_(imie, wiek, "chomik")

class DomoweZoo:

def \_\_init\_\_(self):

self.zwierzatka = []

def dodaj(self, zwierze):

self.zwierzatka.append(zwierze)

def wypisz\_zwierzaki(self):

for zwierze in self.zwierzatka:

zwierze.przedstaw\_sie()

# Przykład użycia:

zoo = DomoweZoo()

zoo.dodaj(Pies("Reksio", 5))

zoo.dodaj(Chomik("Puszek", 2))

zoo.wypisz\_zwierzaki()

**Ćwiczenie 6: Klasa Kot**

class Kot:

def \_\_init\_\_(self, imie):

self.imie = imie

self.lubiane\_zabawki = set()

self.nielubiane\_zabawki = set()

def przedstaw\_sie(self):

print(f"Cześć, jestem kotem o imieniu {self.imie}!")

def dodaj\_lubiana\_zabawke(self, zabawka):

if zabawka in self.nielubiane\_zabawki:

self.nielubiane\_zabawki.remove(zabawka)

self.lubiane\_zabawki.add(zabawka)

def dodaj\_nielubiana\_zabawke(self, zabawka):

if zabawka in self.lubiane\_zabawki:

self.lubiane\_zabawki.remove(zabawka)

self.nielubiane\_zabawki.add(zabawka)

def wypisz\_zabawki(self):

print(f"Lubię te zabawki: {self.lubiane\_zabawki}. Nie lubię tych: {self.nielubiane\_zabawki}.")

# Przykład użycia:

kot = Kot("Mruczek")

kot.przedstaw\_sie()

kot.dodaj\_lubiana\_zabawke("piłeczka")

kot.dodaj\_nielubiana\_zabawke("myszka")

kot.wypisz\_zabawki()

**Ćwiczenie 7: Kot i zabawki**

import random

class Kot:

def \_\_init\_\_(self, imie):

self.imie = imie

self.lubiane\_zabawki = set()

self.nielubiane\_zabawki = set()

def przejdz\_obok\_zabawki(self, zabawka):

decyzja = random.choice(["obojętność", "polubić", "znienawidzić"])

if decyzja == "polubić":

self.dodaj\_lubiana\_zabawke(zabawka)

elif decyzja == "znienawidzić" and zabawka in self.lubiane\_zabawki:

self.dodaj\_nielubiana\_zabawke(zabawka)

def dodaj\_lubiana\_zabawke(self, zabawka):

self.nielubiane\_zabawki.discard(zabawka)

self.lubiane\_zabawki.add(zabawka)

def dodaj\_nielubiana\_zabawke(self, zabawka):

self.lubiane\_zabawki.discard(zabawka)

self.nielubiane\_zabawki.add(zabawka)

def wypisz\_zabawki(self):

print(f"Kot {self.imie} lubi: {self.lubiane\_zabawki}. Nie lubi: {self.nielubiane\_zabawki}.")

# Symulacja

def symulacja\_kota(kot, lista\_zabawek, iteracje):

for \_ in range(iteracje):

zabawka = random.choice(lista\_zabawek)

kot.przejdz\_obok\_zabawki(zabawka)

kot.wypisz\_zabawki()

# Przykład użycia:

lista\_zabawek = ["piłka", "mysz", "piórko", "sznurek", "kocimiętka"]

mruczek = Kot("Mruczek")

symulacja\_kota(mruczek, lista\_zabawek, 100)

**Ćwiczenie 8: Liczba zespolona (szczegółowe rozwiązanie)**

import math

# Klasa reprezentująca liczbę zespoloną

class Zespolona:

def \_\_init\_\_(self, re, im):

"""

Inicjalizacja liczby zespolonej z częściami rzeczywistą i urojoną.

"""

self.re = re

self.im = im

def modul(self):

"""

Oblicza moduł liczby zespolonej: √(a² + b²)

"""

return math.sqrt(self.re\*\*2 + self.im\*\*2)

@staticmethod

def dodaj(z1, z2):

"""

Oblicza sumę dwóch liczb zespolonych.

z1 = a+bi, z2 = c+di

Wynik: (a+c) + (b+d)i

"""

return Zespolona(z1.re + z2.re, z1.im + z2.im)

@staticmethod

def mnoz(z1, z2):

"""

Oblicza iloczyn dwóch liczb zespolonych.

z1 = a+bi, z2 = c+di

Wynik: (ac - bd) + (ad + bc)i

"""

return Zespolona(z1.re \* z2.re - z1.im \* z2.im, z1.re \* z2.im + z1.im \* z2.re)

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

"""

Reprezentacja tekstowa liczby zespolonej w formacie a+bi.

"""

return f"{self.re} + {self.im}i"

# Przykład użycia

z1 = Zespolona(3, 4)

z2 = Zespolona(1, 2)

print("Liczba zespolona z1:", z1)

print("Liczba zespolona z2:", z2)

print("Moduł z1:", z1.modul())

print("Suma z1 + z2:", Zespolona.dodaj(z1, z2))

print("Iloczyn z1 \* z2:", Zespolona.mnoz(z1, z2))

**Ćwiczenie 9: Ułamek (szczegółowe rozwiązanie)**

from math import gcd

# Klasa reprezentująca ułamki

class Ulamek:

def \_\_init\_\_(self, licznik, mianownik):

"""

Inicjalizacja ułamka z licznikiem i mianownikiem.

"""

if mianownik == 0:

raise ValueError("Mianownik nie może być równy 0!")

self.licznik = licznik

self.mianownik = mianownik

def skroc(self):

"""

Skraca ułamek do postaci nieskracalnej.

"""

dzielnik = gcd(self.licznik, self.mianownik)

self.licznik //= dzielnik

self.mianownik //= dzielnik

@staticmethod

def dodaj(u1, u2):

"""

Oblicza sumę dwóch ułamków.

"""

licznik = u1.licznik \* u2.mianownik + u2.licznik \* u1.mianownik

mianownik = u1.mianownik \* u2.mianownik

wynik = Ulamek(licznik, mianownik)

wynik.skroc()

return wynik

@staticmethod

def odejmij(u1, u2):

"""

Oblicza różnicę dwóch ułamków.

"""

licznik = u1.licznik \* u2.mianownik - u2.licznik \* u1.mianownik

mianownik = u1.mianownik \* u2.mianownik

wynik = Ulamek(licznik, mianownik)

wynik.skroc()

return wynik

@staticmethod

def mnoz(u1, u2):

"""

Oblicza iloczyn dwóch ułamków.

"""

licznik = u1.licznik \* u2.licznik

mianownik = u1.mianownik \* u2.mianownik

wynik = Ulamek(licznik, mianownik)

wynik.skroc()

return wynik

@staticmethod

def dziel(u1, u2):

"""

Oblicza iloraz dwóch ułamków.

"""

licznik = u1.licznik \* u2.mianownik

mianownik = u1.mianownik \* u2.licznik

wynik = Ulamek(licznik, mianownik)

wynik.skroc()

return wynik

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

"""

Reprezentacja tekstowa ułamka w formacie licznik/mianownik.

"""

return f"{self.licznik}/{self.mianownik}"

# Przykład użycia

u1 = Ulamek(1, 2)

u2 = Ulamek(2, 3)

print("Ułamek u1:", u1)

print("Ułamek u2:", u2)

print("Suma u1 + u2:", Ulamek.dodaj(u1, u2))

print("Różnica u1 - u2:", Ulamek.odejmij(u1, u2))

print("Iloczyn u1 \* u2:", Ulamek.mnoz(u1, u2))

print("Iloraz u1 / u2:", Ulamek.dziel(u1, u2))