```
##############################################################################
lab1
# Wartość wielomianu w punkcie przy pomocy schematu Hornera

print("Podaj stopień wielomianu: ")
stopien = int(input())

wspolczynniki = []

for i in range(stopien + 1):
    print("Podaj współczynnik dla x^" + str(stopien - i) + ": ")
    wspolczynniki.append(float(input()))

print("Podaj wartość x0: ")
x = float(input())
wynik = wspolczynniki[0]

for i in range(1, stopien + 1):
    wynik = wynik * x + wspolczynniki[i]

print("Wartość wielomianu dla x0=" + str(x) + ": " + str(wynik))

##############################################################################
lab1
# Dzielenie wielomianu przez dwumian

n = int(input("Podaj stopień wielomianu: "))
wielomian = [0] * (n + 1)
i = n
for x in range(n + 1):
        wielomian[x] = float(input("Podaj współczynnik " + str(i) + " stopnia:
"))
        i -= 1

dwumian = float(input("\nPodaj współczynnik dwumianu: "))

wynik = [0] * (n + 1)
wynik[0] = wielomian[0]

for x in range(n):
    wynik[x + 1] = dwumian * wynik[x] + wielomian[x + 1]

reszta = wynik.pop(n)

wielomian_string = ''
for x in range(n + 1):
    if x == n:
        wielomian_string += str(abs(wielomian[x]))
        break
```

```
    elif x == n - 1:
        wielomian_string += str(abs(wielomian[x])) + 'x'
    else:
        wielomian_string += str(abs(wielomian[x])) + 'x^' + str(n - x)
    wielomian_string += ' - ' if wielomian[x] < 0 else ' + '

dwumian_string = 'x  ' + str(abs(dwumian)) if dwumian < 0 else 'x - ' +
str(dwumian)

print(wielomian_string + ' : ' + dwumian_string + " = " + wielomian_string)
print("Reszta z dzielenia: " + str(reszta))

################################################################################
lab2
# Obliczanie miejsca zerowego równań nieliniowych metodą Bisekcji
#pip install sympy
from sympy import *

x = symbols('x')  # deklaracja symboli funkcji

# deklaracja funkcji. '**' oznacza '^'
f = lambda x: (x + 1) * (x - 1) ** 4

epsilon = 0.001  # ustalamy dokładność błędu

a = float(input("Podaj lewą granicę przedziału: "))
b = float(input("Podaj prawą granicę przedziału: "))
iterations = 0

if f(a) * f(b) >= 0:
    print("Funkcja nie przyjmuje różnych znaków na końcach przedziału.")
else:
    c = (a + b) / 2
    while abs(f(c)) > epsilon:
        if abs(f(c)) <= epsilon:
            break
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        c = (a + b) / 2
        iterations += 1

    print("Wynik: " + str(c))
    print("Liczba iteracji: " + str(iterations) + "\n")

################################################################################
lab2
# Obliczanie miejsca zerowego równań nieliniowych metodą stycznych (Newton)-
```

```
from sympy import *

x = symbols('x')  # deklaracja symboli funkcji

# deklaracja funkcji. '**' oznacza '^'
f = (x + 1) * (x - 1) ** 4

epsilon = 0.0001  # ustalamy dokładność błędu

a = float(input("Podaj lewą granicę przedziału: "))
b = float(input("Podaj prawą granicę przedziału: "))

# sprawdzanie, czy funkcja ma przeciwne znaki w punktach a i b
if f.subs(x, a) * f.subs(x, b) >= 0:
    print("Funkcja nie przyjmuje różnych znaków na końcach przedziału.")
else:
    # znajdowanie punktu startowego
    dx2 = diff(f, x, 2)
    dx = diff(f, x)

    if f.subs(x, a) * dx2.subs(x, a) > 0:
        x0 = a
    else:
        x0 = b

    iterations = 0

    while abs(f.subs(x, x0)) > epsilon:
        xn = x0 - (f.subs(x, x0) / dx.subs(x, x0))
        x0 = xn
        iterations += 1

    print("Wynik: " + str(x0))
    print("Liczba iteracji: " + str(iterations) + "\n")

################################################################################
lab2
# Obliczanie miejsca zerowego równań nieliniowych metodą siecznych
from sympy import *

x = symbols('x')  # deklaracja symboli funkcji

# deklaracja funkcji. '**' oznacza '^'
f = lambda x: (x + 1) * (x - 1) ** 4

epsilon = 0.0001  # ustalamy dokładność błędu

a = float(input("Podaj lewą granicę przedziału: "))
b = float(input("Podaj prawą granicę przedziału: "))
```

```
# sprawdzanie, czy funkcja ma przeciwne znaki w punktach a i b
if f(a) * f(b) >= 0:
    print("Funkcja nie przyjmuje różnych znaków na końcach przedziału.")
else:
    # znajdowanie punktu startowego
    x0 = a
    x1 = b
    x2 = 0
    iterations = 0

    while abs(f(x1)) > epsilon:
        x2 = x1 - ((f(x1) * (x1 - x0)) / (f(x1) - f(x0)))
        x0 = x1
        x1 = x2
        iterations += 1

    print("Wynik: " + str(x2))
    print("Liczba iteracji: " + str(iterations) + "\n")

################################################################################
lab2
    # Obliczanie miejsca zerowego równań nieliniowych metodą falsi
from sympy import *

x = symbols('x')  # deklaracja symboli funkcji

# deklaracja funkcji. '**' oznacza '^'
f = lambda x: (x + 1) * (x - 1) ** 4

epsilon = 0.0001  # ustalamy dokładność błędu

a = float(input("Podaj lewą granicę przedziału: "))
b = float(input("Podaj prawą granicę przedziału: "))

# sprawdzanie, czy funkcja ma przeciwne znaki w punktach a i b
if f(a) * f(b) >= 0:
    print("Funkcja nie przyjmuje różnych znaków na końcach przedziału.")
else:
    # obliczenia
    xn = (a * f(b) - b * f(a)) / (f(b) - f(a))
    iterations = 0

    while abs(f(xn)) > epsilon:
        x1 = (xn * f(a) - a * f(xn)) / (f(a) - f(xn))
        x2 = (xn * f(b) - b * f(xn)) / (f(b) - f(xn))
        if f(a) * f(xn) < 0:
            xn = x1
        else:
```

```
            xn = x2
        iterations += 1

    print("Wynik: " + str(xn))
    print("Liczba iteracji: " + str(iterations) + "\n")

################################################################################
lab3
# kwadratury metodą trapezów prostych
from sympy import *

x = symbols('x')  # deklaracja symboli funkcji

# deklaracja funkcji. '**' oznacza '^'
f = x ** 2

a = float(input("Podaj dolną granicę przedziału: "))
b = float(input("Podaj górną granicę przedziału: "))

# obliczenia
wyn = ((b - a) / 2) * (f.subs(x, a) + f.subs(x, b))
print("Wynik to: " + str(wyn))

psi = max(f.subs(x, a), f.subs(x, b))
err = (1.0 / 12.0) * (b - a) ** 3 * diff(f, x, 2).subs(x, psi)
print("Błąd to: " + str(err))

################################################################################
lab3
# kwadratury metodą simpsona prosta
from sympy import *

x = symbols('x')  # deklaracja symboli funkcji

# deklaracja funkcji. '**' oznacza '^'
f = x ** 2

a = float(input("Podaj dolną granicę przedziału: "))
b = float(input("Podaj górną granicę przedziału: "))

# obliczenia
wyn = ((b - a) / 6) * (f.subs(x, a) + f.subs(x, b) + 4 * f.subs(x, (a + b) /
2))
print("Wynik to: " + str(wyn))

psi = max(f.subs(x, a), f.subs(x, b))
err = (1.0 / 90.0) * pow((b - a) / 2, 5) * diff(f, x, 4).subs(x, psi)
print("Błąd to: " + str(err))
```

```
##############################################################################
lab4
# kwadratury metodą simpsona złożona
from sympy import *
x = symbols('x')

f = x ** 2

a = float(input("Podaj dolną granicę przedziału: "))
b = float(input("Podaj górną granicę przedziału: "))
n = int(input("Podaj liczbę przedziałów (musi być parzysta): "))

if n % 2 != 0:
    print("Liczba przedziałów musi być parzysta")
    exit()

h = (b - a) / n

wyn = f.subs(x, a) + f.subs(x, b)
xk = a + h
for i in range(1, n):
    if i % 2 == 0:
        wyn += 2 * f.subs(x, xk)
    else:
        wyn += 4 * f.subs(x, xk)
    xk += h
wyn = ((1.0 / 3.0) * h) * wyn  # wynik ze wzoru

print("Wynik to: " + str(wyn))

e = (a + b) / 2  # wybieramy e do liczenia błędu

r = ((((b - a) * pow(h, 4)) / 180.0) * diff(f, x, 4).subs(x, e)) * (-1.0)

print("Błąd to: " + str(r))

print("Wartość całki to: " + str(wyn + r))

##############################################################################
lab4
# kwadratury metodą trapezów złożna
from sympy import *

x = symbols('x')

f = x ** 2

a = float(input("Podaj dolną granicę przedziału: "))
b = float(input("Podaj górną granicę przedziału: "))
```

```
n = int(input("Podaj liczbę podziałów: "))

h = (b - a) / n

wyn = (f.subs(x, b) + f.subs(x, a)) / 2
xk = a + h
for i in range(1, n):
    wyn += f.subs(x, xk)
    xk += h
wyn = h * wyn
print("Wynik to: " + str(wyn))

e = 0
if f.subs(x, a) > f.subs(x, b):
    e = a
else:
    e = b

r = ((((b - a) * pow(h, 2)) / 12.0) * diff(f, x, 2).subs(x, e)) * (-1.0)
print("Błąd to: " + str(r))

print("Całka to: " + str(wyn + r))

################################################################################
lab5
# Obliczanie wielomianu interpolacyjnego metoda Lagrange
from sympy import *

x = symbols('x')
wenz = []

n = int(input("Podaj stopień wielomianu interpolacyjnego: "))

for i in range(n + 1):
    a = float(input("Podaj współrzędną x" + str(i) + ": "))
    b = float(input("Podaj współrzędną y" + str(i) + ": "))
    wenz.append([a, b])

lambd_up = []
lambd_down = []

for i in range(len(wenz)):
    fup = 1
    fdown = 1
    for k in range(len(wenz)):
        if i != k:
            fup *= x - wenz[k][0]
            fdown = (wenz[i][0] - wenz[k][0]) * fdown
```

```
    lambd_up.append(fup)
    lambd_down.append(fdown)

funk = 0
for iter in range(len(wenz)):
    funk += wenz[iter][1] * (lambd_up[iter] / lambd_down[iter])

wyn = simplify(funk)
print(wyn)

################################################################################
```