Пример идеального решения домашнего задания

Задание 1:

```
# 1. Вычислить число с заданной точностью *d*
#     Пример:
#     - при d = 0.001, π = 3.141.    10**-1 ≤ d ≤10**(-10)
import math
from decimal import Decimal

def calc_pi(epsilon):
    '''
    Для вычисления используется ряд Лейбница pi/4 = Sum((-1)**n/(2 * n + 1), n = 0..inf)
    '''
    n = 2
    pi = 4*[1, -1/3]
    dif = 1
    while dif > epsilon:
        pi.append(4*(-1)**n/(2 * n + 1))
```

```
        dif = abs(sum(pi[:-1]) - sum(pi[:-2]))
        n += 1

    return math.floor(sum(pi)*int(1/epsilon))*epsilon

def gauss_Pi (d):
    accuracy = 1
    pi_Gauss = 48*math.atan(1/18)+32*math.atan(1/57)-20*math.atan(1/239)  #формула Гаусса
для вычисления числа Пи
    number_of_digits = int(len(str(d).split(".")[1]))
    print(number_of_digits)
    while number_of_digits > 0:
        accuracy *= 10
        number_of_digits -= 1
    return int(pi_Gauss * accuracy) / accuracy

def chudnovskii_series(n) -> Decimal:
    pi = Decimal(0)
    delta = Decimal(10005).sqrt() / Decimal(4270934400)
    measure = len(str(n)) - 2
    for i in range(0, measure + 1):
        multiplier = (13591409 + 545140134 * i)
        if i != 0:
            delta *= -Decimal((6 * i - 5) * (2 * i - 1) * (6 * i - 1) / Decimal(26680 * 640320 * 640320 *
i * i * i))
        multiplier *= delta
        pi += multiplier
    return str(pi ** -1)[:measure+2]
```

Задание 2:

```python
# 2. Задайте натуральное число N.
# Напишите программу, которая составит список простых множителей числа N.
# Пример:
# n = 17 -> [17]
# n = 3628800 -> [2,3,5,7]

from typing import List
import math
def check_is_prime_number(n: int) -> bool:
    '''
    Согласно теореме Вильсона, если n — простое число,
    то[(n–1)! + 1] делится на n (без остатка)
    '''
    n_prev = int(n) - 1
    factorial_n_prev = math.factorial(n_prev)
    if (factorial_n_prev + 1) % int(n) == 0:
        return True
    else:
        return False

def find_primes(n: int) -> List[int]:
    primes = []
    for num in range(2, n+1):
        if not n%num and check_is_prime_number(num):
            primes.append(num)
    return primes
```

Задание 3:

```
# 3. Задайте последовательность чисел. Напишите программу, которая выведет список неповторяющихся элементов исходной последовательности.
# [1,2,3,4,3,2,1] -> [1,2,3,4]

def get_unique_list(lst):
    unique = []
    for element in lst:
        if element not in unique:
            unique.append(element)
    return unique
```

Задание 4:

```
# 4. Задана натуральная степень n.
# Сформировать случайным образом список коэффициентов (значения от 0 до 100)
# многочлена и записать в файл многочлен степени n.
#     Пример:
#     - n=2 => 2*x² + 4*x + 5 = 0 или x² + 5 = 0 или 10*x² = 0
#     - n =3 => 5*x*3 + 18*x**2 + 7*x + 19 = 0 - коэффициенты = [5,18,7,19]
from random import randint
```

```python
from typing import List, Optional


def get_int(input_string:str,
        min_num: Optional[int] = None,
        max_num: Optional[int] = None) -> int:
    '''
    Asks user for integer until it's done
    Args:
        input_string - text of request,
        min_num - minimum of range,
        max_num - maximum of range
    Returns:
        integer
    '''
    while True:
        try:
            num = int(input(input_string))
            if min_num and num < min_num:
                raise ValueError
            elif max_num and num > max_num:
                raise ValueError
            return num
        except ValueError:
            print('Неправильный ввод. Повторите попытку')

def get_koef_list() -> List[int]:
    '''
    Create list of koefs where first is not None

    Returns:
        List[int]
    '''
    nums = [randint(1,100)] + [randint(0,100) for _ in range(n)]
    return nums

def write_to_file(string_to_write: str) -> None:
    '''
    Write argument to text.txt
```

```
    Args:
        string_to_write - str
    '''
    with open('text.txt', mode='a') as file:
        file.write(string_to_write + '\n')


if __name__ == '__main__':
    n = get_int('Введите натуральную степень n: ')
    koef_list = get_koef_list()
    degree_list = [i for i in range(n, -1, -1)]
    expression = ''
    for koef, degree in zip(koef_list, degree_list):
        if koef:
            expression += f'{koef}'
            if degree:
                expression += f'*x'
                if degree > 1:
                    expression += f'**{degree}'
            expression+= ' + '
    expression = expression[:-2] + '= 0'
    write_to_file(expression)
```

Задание 5:

```python
# 5. Даны два файла, в каждом из которых находится запись многочлена.
# Задача - сформировать файл, содержащий сумму многочленов.

from typing import List
from task4 import write_to_file


def read_file(filename: str) -> str:
    '''
    Read first line from file

    Args:
        filename - name of file to open
    '''
```

```
    with open(filename) as file:
        return file.readline()


def split_polinom(polinom: str) -> List[str]:
    '''
    Get list of lists with koefs and degrees

    Args:
        polinom - looks like 15*x**2 + 12*x + 86 = 0

    Returns:
        list looks like [['15', '**2'], ['12', '*x'], ['86']]
    '''
    polinom = polinom[:-4].split(' + ')
    for i, element in enumerate(polinom):
        polinom[i] = element.strip().split('*x')
    return polinom


def sum_list_polinoms(polinom1: List[str],
                      polinom2: List[str]) -> List[str]:
    '''
    Go to maxlen polinom and add koef of element with
    element of another polinom if it exists

    Args:
        polinom1, polinom2 - lists of koefs and degrees

    Returns:
        sum_result_polinom - List[str]
    '''
    if len(polinom1) > len(polinom2):
        polinom1, polinom2 = polinom2, polinom1
    result_polinom = []
    for element in polinom2:
        filter_element = list(filter(lambda el: element[-1] in el, polinom1))
        if filter_element:
            filter_element = filter_element.pop()
```

```
            element[0] = str(int(element[0]) + int(filter_element[0]))
        result_polinom.append(element)
    return result_polinom


def unite_list_polinom_to_str(polinom_list: List[str]) -> str:
    '''
    Unite list of coefs and degrees into string

    Args:
        polinom_list - list of coefs and degrees

    Returns:
        string
    '''
    expression = ''
    for i, element in enumerate(result_polinom):
        expression += '*x'.join(element)
        if i != len(result_polinom):
            expression += ' + '
        if len(element) == 1:
            expression = expression[:-3]
    expression += ' = 0'
    return expression
```