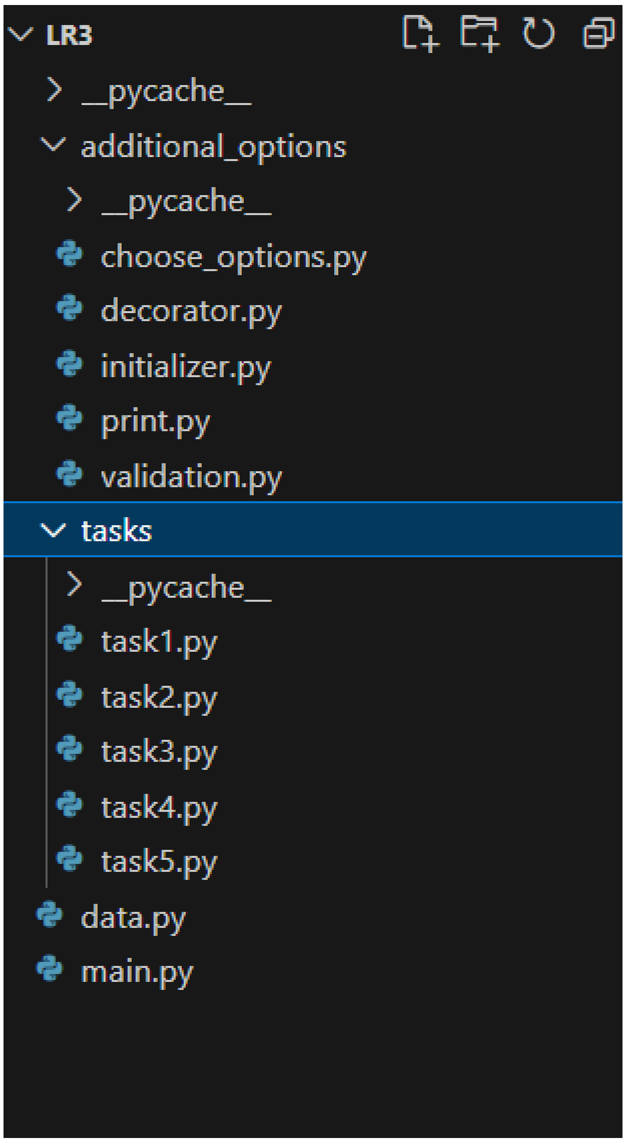
Лабораторная работа №3. Стандартные типы данных, коллекции, функции, модули.

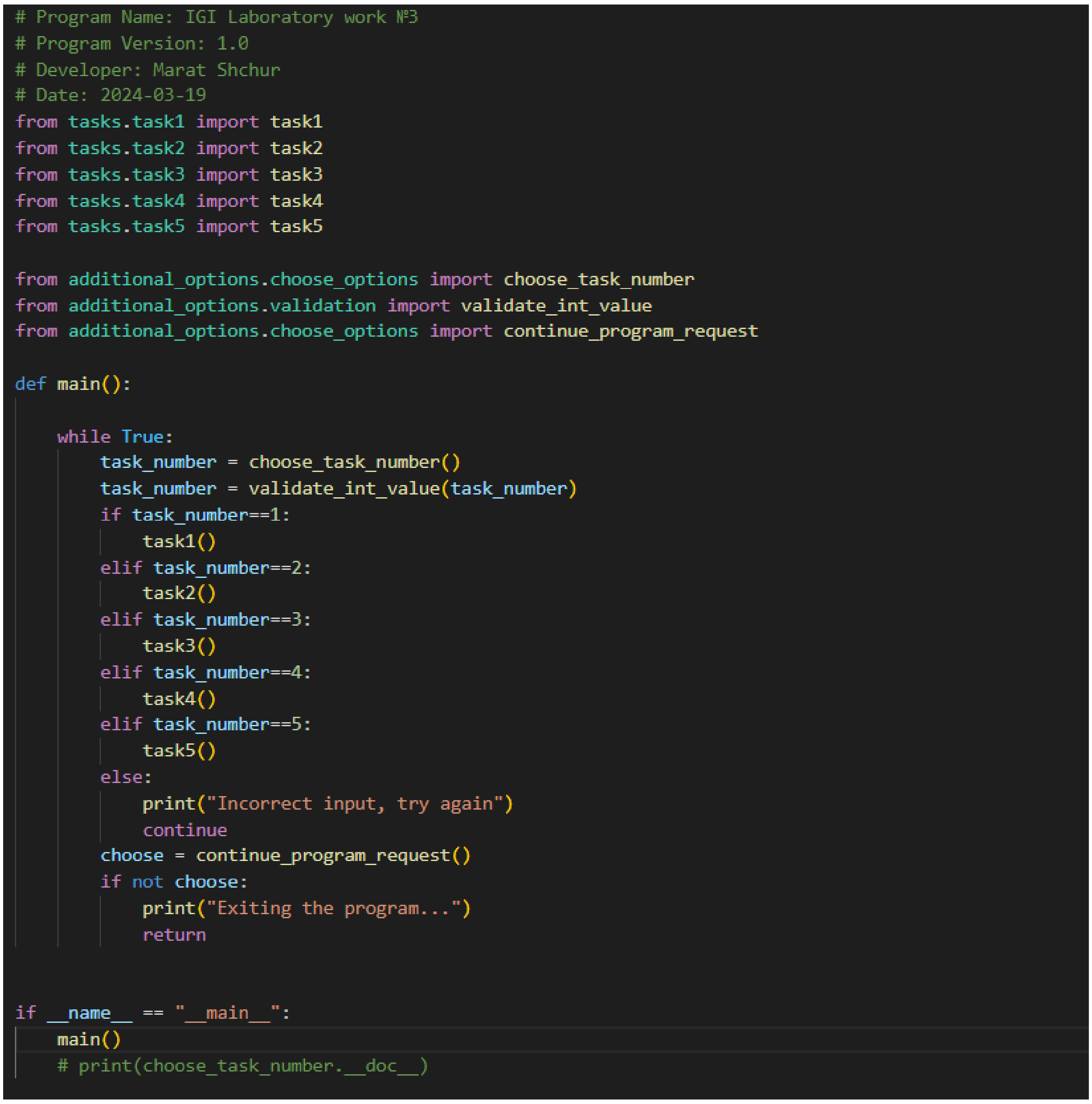
Щур Марат 253501. Репозиторий: https://github.com/maratshchur/IGI_REPO.git

Структура проекта:



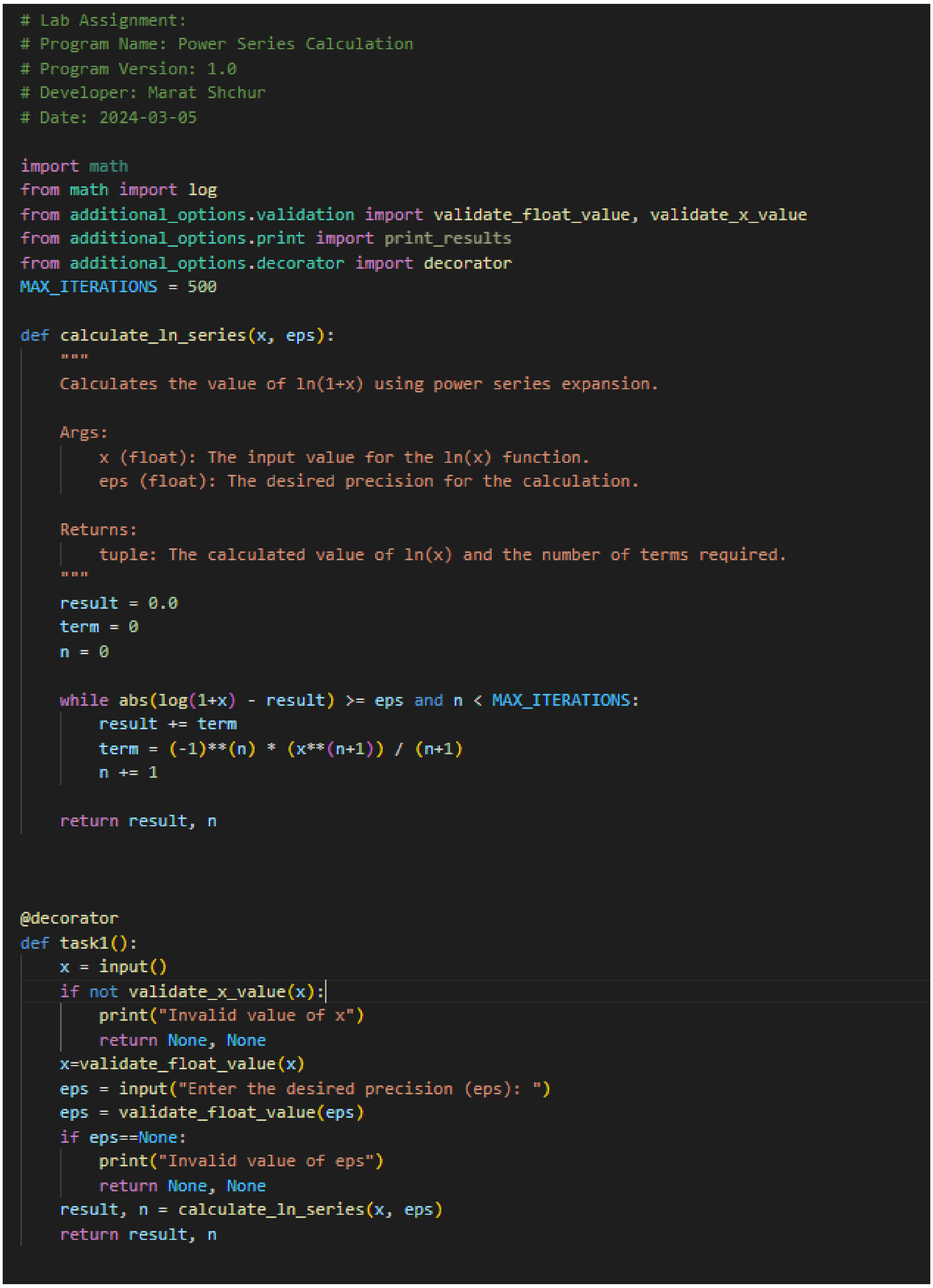
```
LR3
  > __pycache__
  v additional_options
      > __pycache__
      choose_options.py
      decorator.py
      initializer.py
      print.py
      validation.py
  v tasks
      > __pycache__
      task1.py
      task2.py
      task3.py
      task4.py
      task5.py
    data.py
    main.py
```

Вспомогательные функции: choose_options,py, initializer.py, print.py, validation.py *main.py*



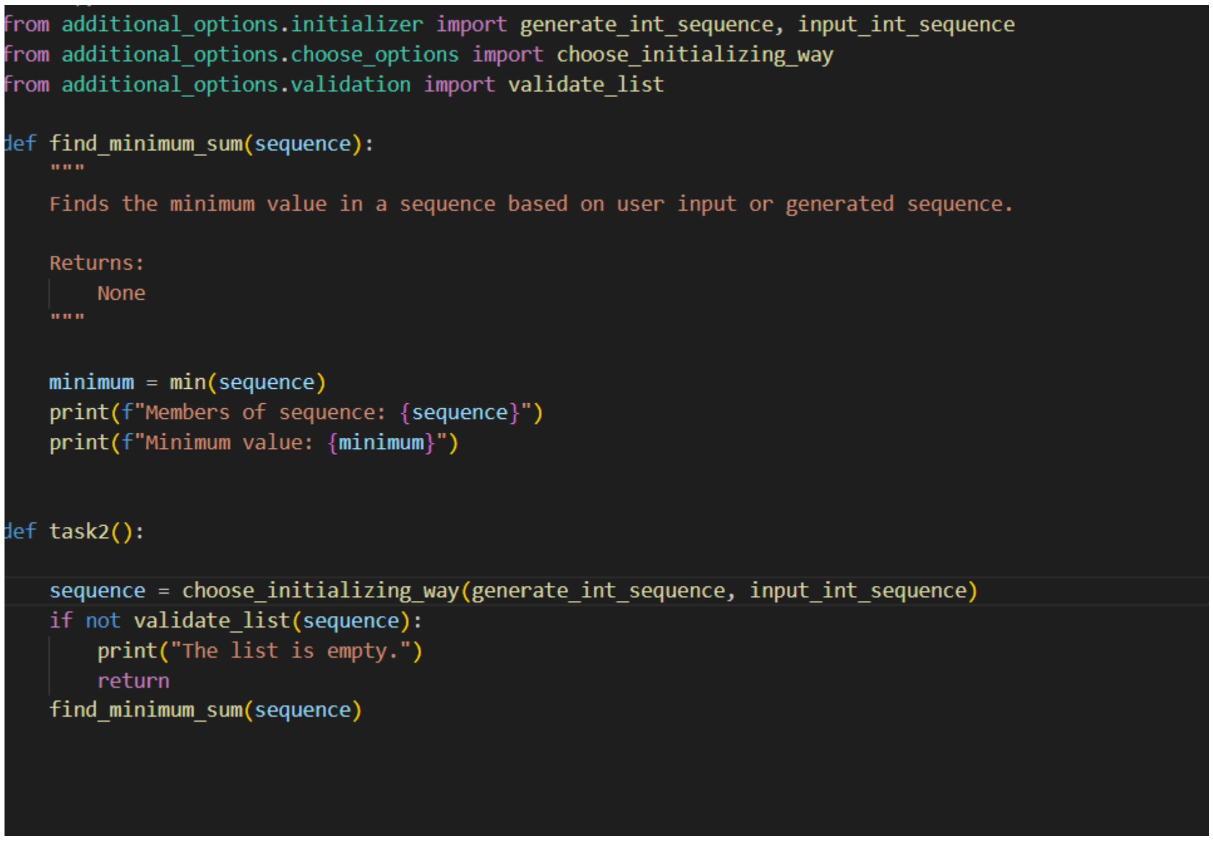
```python
# Program Name: IGI Laboratory work №3
# Program Version: 1.0
# Developer: Marat Shchur
# Date: 2024-03-19
from tasks.task1 import task1
from tasks.task2 import task2
from tasks.task3 import task3
from tasks.task4 import task4
from tasks.task5 import task5

from additional_options.choose_options import choose_task_number
from additional_options.validation import validate_int_value
from additional_options.choose_options import continue_program_request

def main():

    while True:
        task_number = choose_task_number()
        task_number = validate_int_value(task_number)
        if task_number==1:
            task1()
        elif task_number==2:
            task2()
        elif task_number==3:
            task3()
        elif task_number==4:
            task4()
        elif task_number==5:
            task5()
        else:
            print("Incorrect input, try again")
            continue
        choose = continue_program_request()
        if not choose:
            print("Exiting the program...")
            return


if __name__ == "__main__":
    main()
    # print(choose_task_number.__doc__)
```

1. *Составить программу для вычисления значения функции arcsin с помощью разложения функции в степенной ряд. Задать точность вычислений eps.*



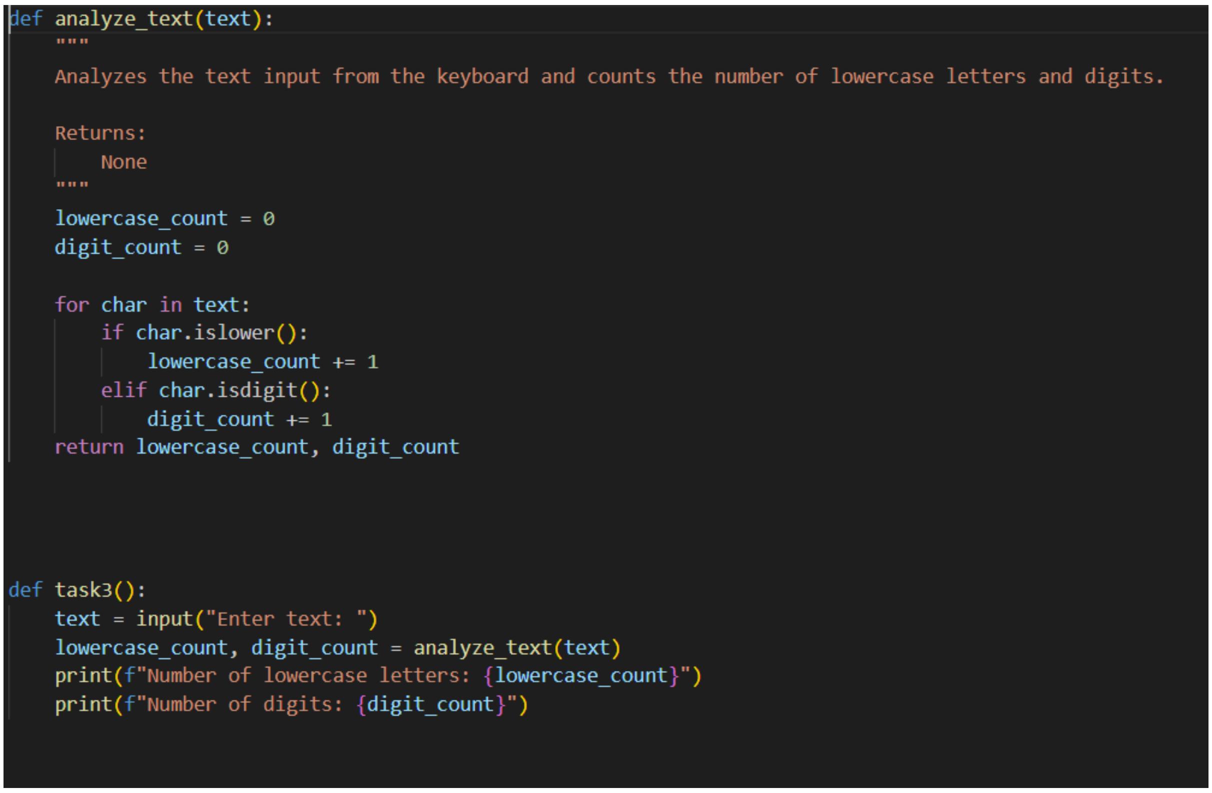
```python
# Lab Assignment:
# Program Name: Power Series Calculation
# Program Version: 1.0
# Developer: Marat Shchur
# Date: 2024-03-05

import math
from math import log
from additional_options.validation import validate_float_value, validate_x_value
from additional_options.print import print_results
from additional_options.decorator import decorator
MAX_ITERATIONS = 500

def calculate_ln_series(x, eps):
    """
    Calculates the value of ln(1+x) using power series expansion.

    Args:
        x (float): The input value for the ln(x) function.
        eps (float): The desired precision for the calculation.

    Returns:
        tuple: The calculated value of ln(x) and the number of terms required.
    """
    result = 0.0
    term = 0
    n = 0

    while abs(log(1+x) - result) >= eps and n < MAX_ITERATIONS:
        result += term
        term = (-1)**(n) * (x**(n+1)) / (n+1)
        n += 1

    return result, n



@decorator
def task1():
    x = input()
    if not validate_x_value(x):
        print("Invalid value of x")
        return None, None
    x=validate_float_value(x)
    eps = input("Enter the desired precision (eps): ")
    eps = validate_float_value(eps)
    if eps==None:
        print("Invalid value of eps")
        return None, None
    result, n = calculate_ln_series(x, eps)
    return result, n
```

2. *Организовать цикл, который принимает целые числа и суммирует каждое второе из них.*



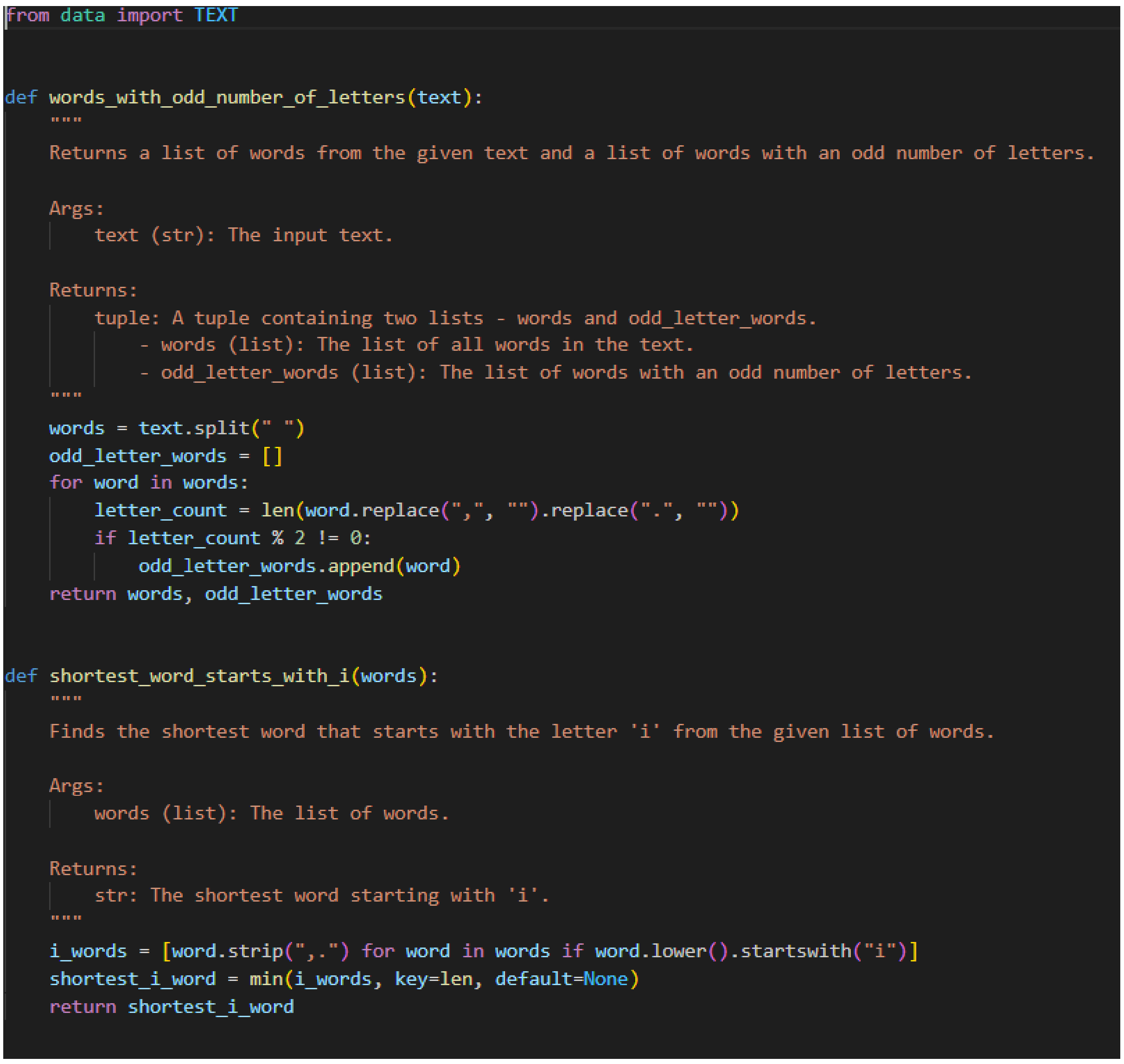
```python
from additional_options.initializer import generate_int_sequence, input_int_sequence
from additional_options.choose_options import choose_initializing_way
from additional_options.validation import validate_list

def find_minimum_sum(sequence):
    """
    Finds the minimum value in a sequence based on user input or generated sequence.

    Returns:
        None
    """

    minimum = min(sequence)
    print(f"Members of sequence: {sequence}")
    print(f"Minimum value: {minimum}")


def task2():

    sequence = choose_initializing_way(generate_int_sequence, input_int_sequence)
    if not validate_list(sequence):
        print("The list is empty.")
        return
    find_minimum_sum(sequence)
```

3. *В строке, вводимой с клавиатуры, подсчитать количество цифр.*

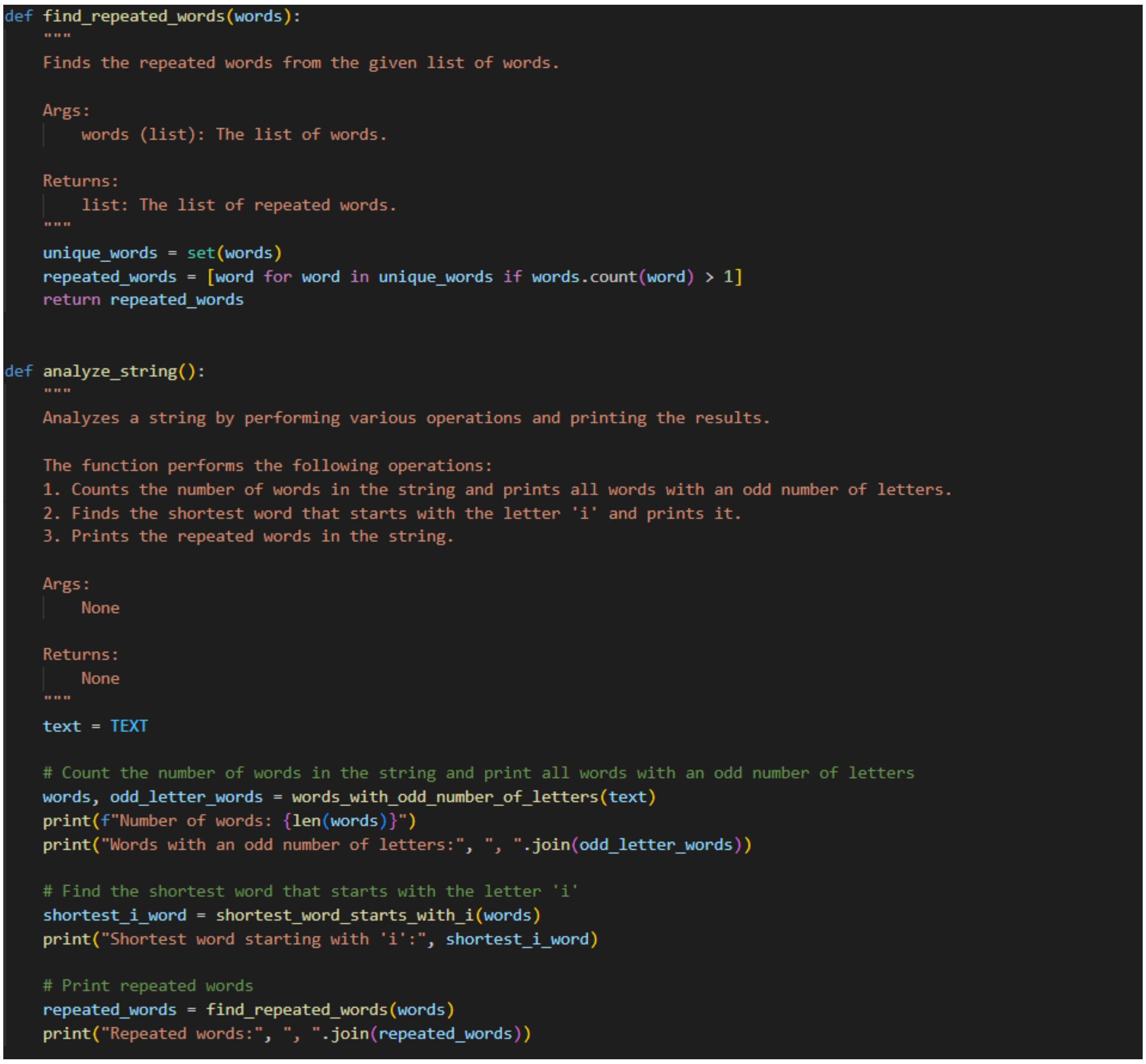


```python
def analyze_text(text):
    """
    Analyzes the text input from the keyboard and counts the number of lowercase letters and digits.

    Returns:
        None
    """
    lowercase_count = 0
    digit_count = 0

    for char in text:
        if char.islower():
            lowercase_count += 1
        elif char.isdigit():
            digit_count += 1
    return lowercase_count, digit_count




def task3():
    text = input("Enter text: ")
    lowercase_count, digit_count = analyze_text(text)
    print(f"Number of lowercase letters: {lowercase_count}")
    print(f"Number of digits: {digit_count}")
```

4. *Дана строка текста, в которой слова разделены пробелами и запятыми. В соответствии с заданием своего варианта составьте программу для анализа строки, инициализированной в коде программы:*
«So she was considering in her own mind, as well as she could, for the hot day made her feel very sleepy and stupid, whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.»
а) определить число слов, длина которых равна 3 символа;
б) найти слова, у которых количество гласных равно количеству согласных и их порядковые номера;
в) вывести слова в порядке убывания их длин

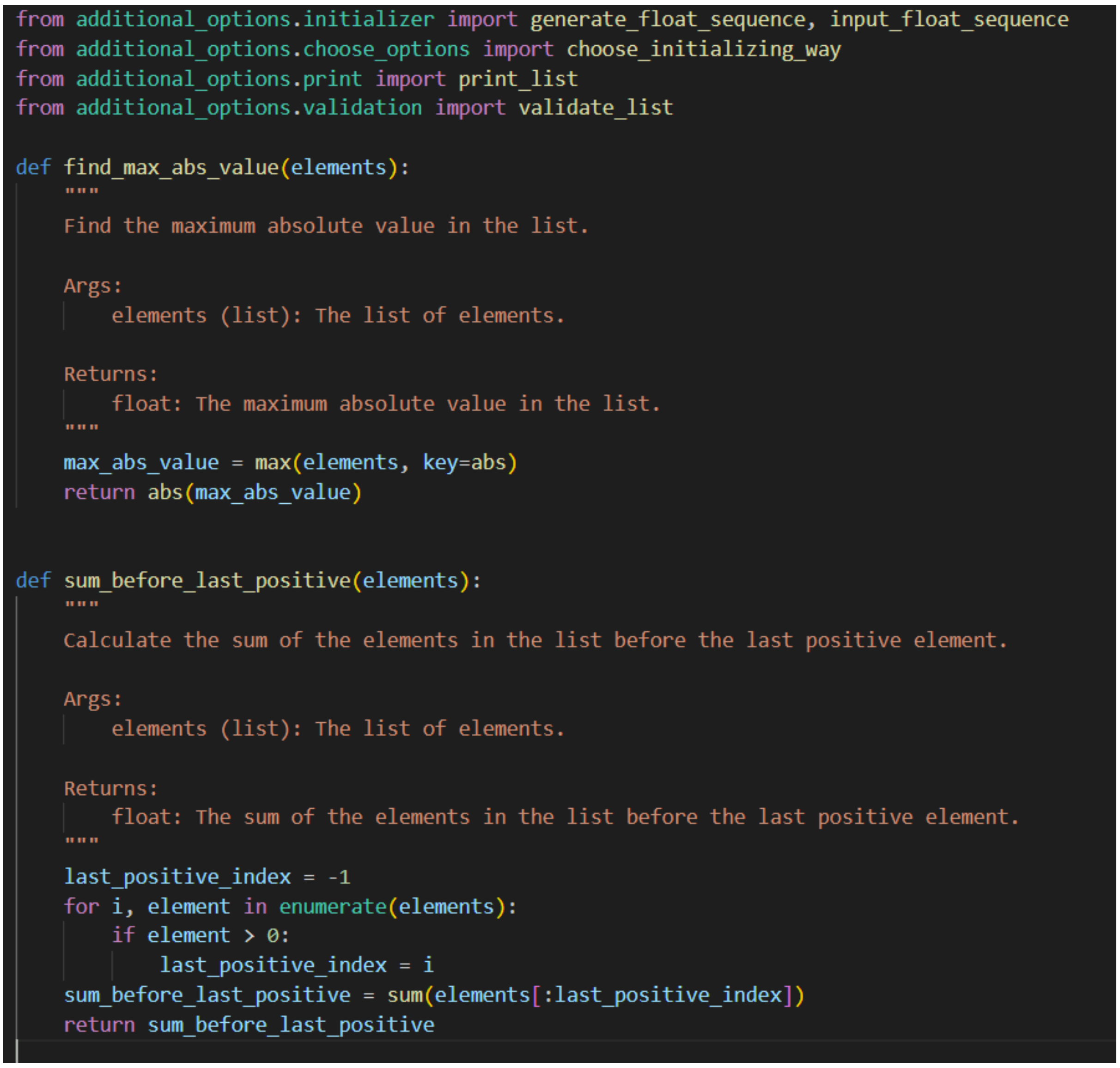


```python
from data import TEXT


def words_with_odd_number_of_letters(text):
    """
    Returns a list of words from the given text and a list of words with an odd number of letters.

    Args:
        text (str): The input text.

    Returns:
        tuple: A tuple containing two lists - words and odd_letter_words.
            - words (list): The list of all words in the text.
            - odd_letter_words (list): The list of words with an odd number of letters.
    """
    words = text.split(" ")
    odd_letter_words = []
    for word in words:
        letter_count = len(word.replace(",", "").replace(".", ""))
        if letter_count % 2 != 0:
            odd_letter_words.append(word)
    return words, odd_letter_words


def shortest_word_starts_with_i(words):
    """
    Finds the shortest word that starts with the letter 'i' from the given list of words.

    Args:
        words (list): The list of words.

    Returns:
        str: The shortest word starting with 'i'.
    """
    i_words = [word.strip(",.") for word in words if word.lower().startswith("i")]
    shortest_i_word = min(i_words, key=len, default=None)
    return shortest_i_word
```

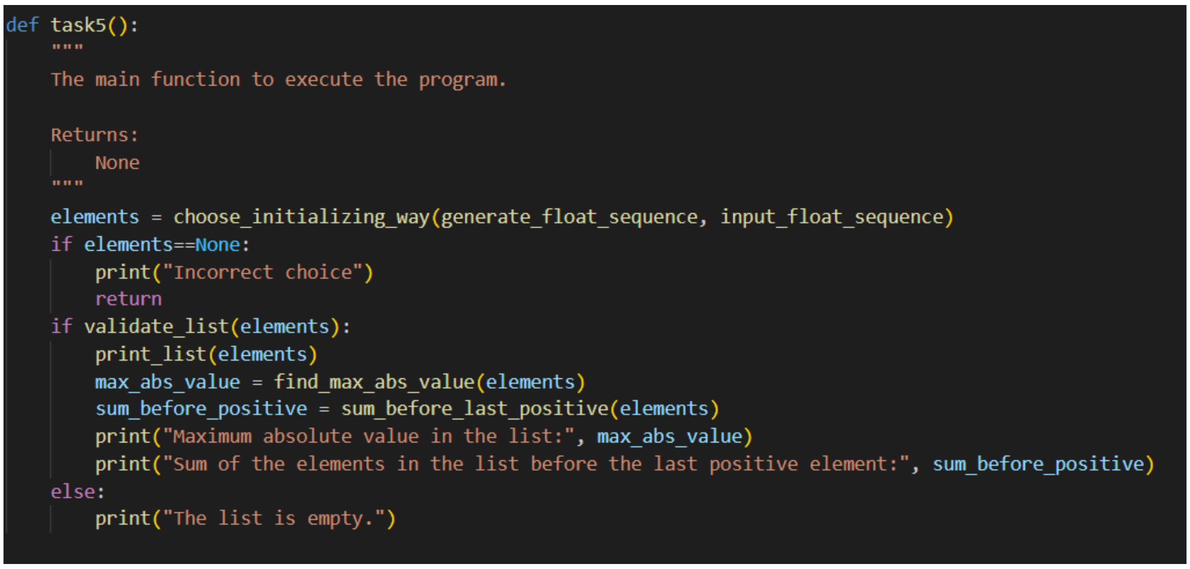


```
def find_repeated_words(words):
    """
    Finds the repeated words from the given list of words.

    Args:
        words (list): The list of words.

    Returns:
        list: The list of repeated words.
    """
    unique_words = set(words)
    repeated_words = [word for word in unique_words if words.count(word) > 1]
    return repeated_words


def analyze_string():
    """
    Analyzes a string by performing various operations and printing the results.

    The function performs the following operations:
    1. Counts the number of words in the string and prints all words with an odd number of letters.
    2. Finds the shortest word that starts with the letter 'i' and prints it.
    3. Prints the repeated words in the string.

    Args:
        None

    Returns:
        None
    """
    text = TEXT

    # Count the number of words in the string and print all words with an odd number of letters
    words, odd_letter_words = words_with_odd_number_of_letters(text)
    print(f"Number of words: {len(words)}")
    print("Words with an odd number of letters:", ", ".join(odd_letter_words))

    # Find the shortest word that starts with the letter 'i'
    shortest_i_word = shortest_word_starts_with_i(words)
    print("Shortest word starting with 'i':", shortest_i_word)

    # Print repeated words
    repeated_words = find_repeated_words(words)
    print("Repeated words:", ", ".join(repeated_words))
```

5. *Найти количество положительных четных элементов списка и сумму элементов списка, расположенных после последнего элемента, равного нулю*



```
from additional_options.initializer import generate_float_sequence, input_float_sequence
from additional_options.choose_options import choose_initializing_way
from additional_options.print import print_list
from additional_options.validation import validate_list

def find_max_abs_value(elements):
    """
    Find the maximum absolute value in the list.

    Args:
        elements (list): The list of elements.

    Returns:
        float: The maximum absolute value in the list.
    """
    max_abs_value = max(elements, key=abs)
    return abs(max_abs_value)


def sum_before_last_positive(elements):
    """
    Calculate the sum of the elements in the list before the last positive element.

    Args:
        elements (list): The list of elements.

    Returns:
        float: The sum of the elements in the list before the last positive element.
    """
    last_positive_index = -1
    for i, element in enumerate(elements):
        if element > 0:
            last_positive_index = i
    sum_before_last_positive = sum(elements[:last_positive_index])
    return sum_before_last_positive
```



```python
def task5():
    """
    The main function to execute the program.

    Returns:
        None
    """
    elements = choose_initializing_way(generate_float_sequence, input_float_sequence)
    if elements==None:
        print("Incorrect choice")
        return
    if validate_list(elements):
        print_list(elements)
        max_abs_value = find_max_abs_value(elements)
        sum_before_positive = sum_before_last_positive(elements)
        print("Maximum absolute value in the list:", max_abs_value)
        print("Sum of the elements in the list before the last positive element:", sum_before_positive)
    else:
        print("The list is empty.")
```