

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.10
дисциплины «Основы программной инженерии»

Выполнил:
Соколов Михаил Романович
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Тема: Функции с переменным числом параметров в Python.

Цель работы: приобретение навыков по работе с функциями с переменным числом параметров при написании программ с помощью языка программирования Python версии 3.x.

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и .gitignore файл для языка программирования Python:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * PoTtaTto ▾ / **Repository name *** fundamentals_of_software

✔ fundamentals_of_software_engineering_lab2_10 is available.

Great repository names are short and memorable. Need inspiration? How about [super-duper-happiness](#) ?

Description (optional)

Public ☒ Anyone on the internet can see this repository. You choose who can commit.

Private ☐ You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

ⓘ You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория с заданными настройками

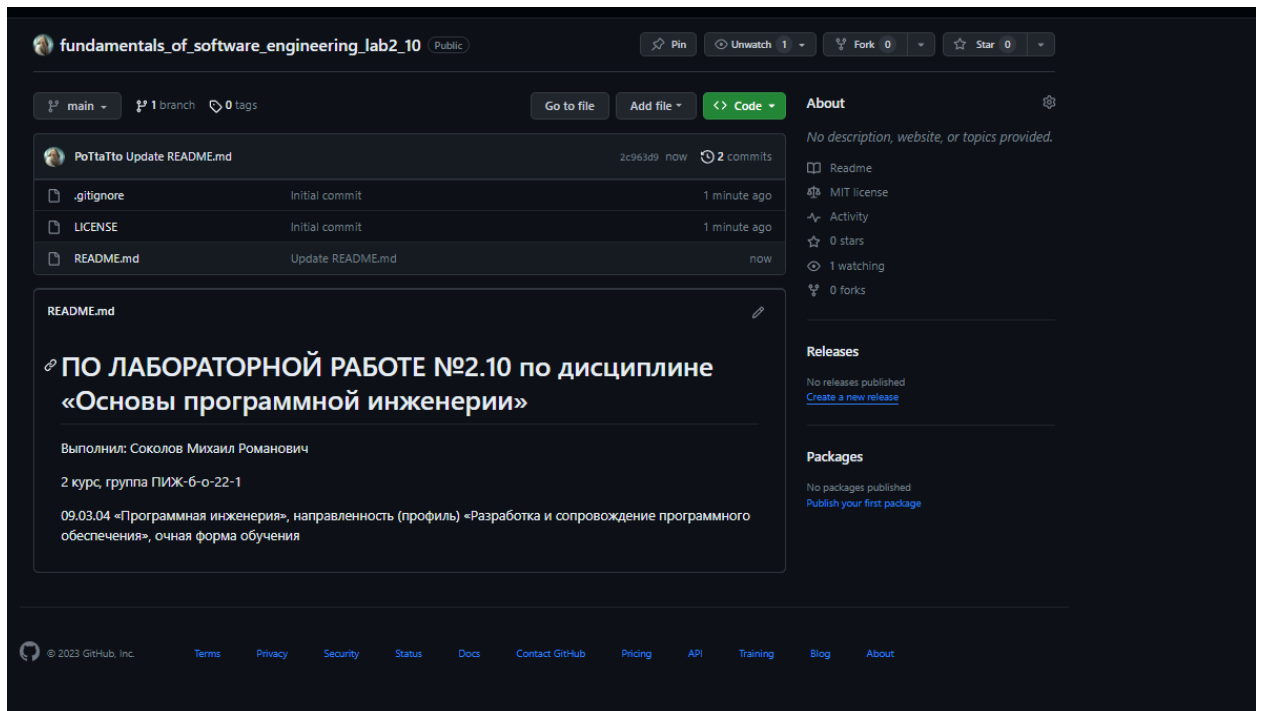


Рисунок 2 – Созданный репозиторий

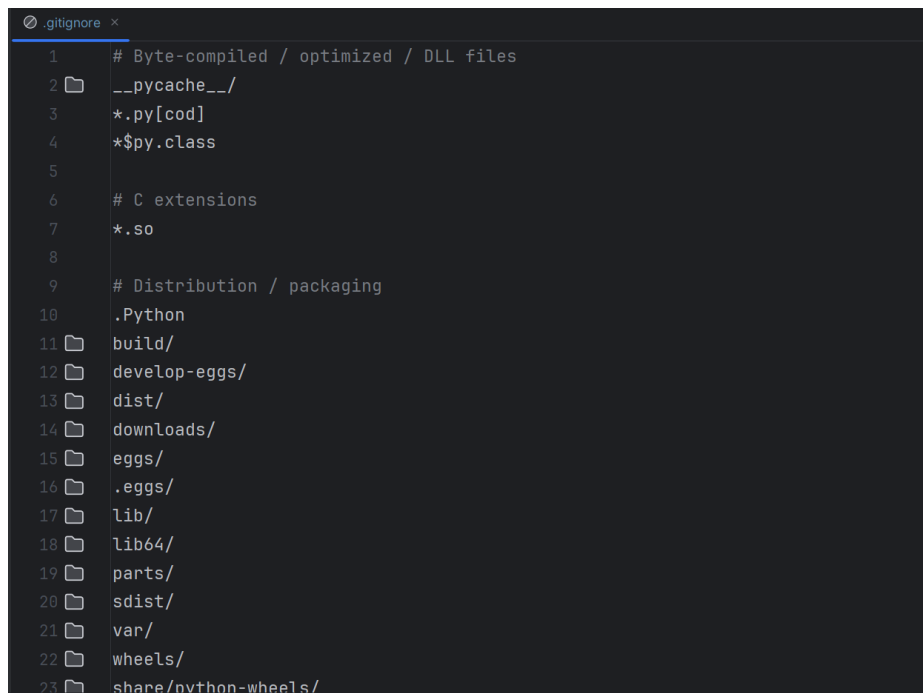
```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10> git clone https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_10
Cloning into 'fundamentals_of_software_engineering_lab2_10'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10> |
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10>
```

Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза

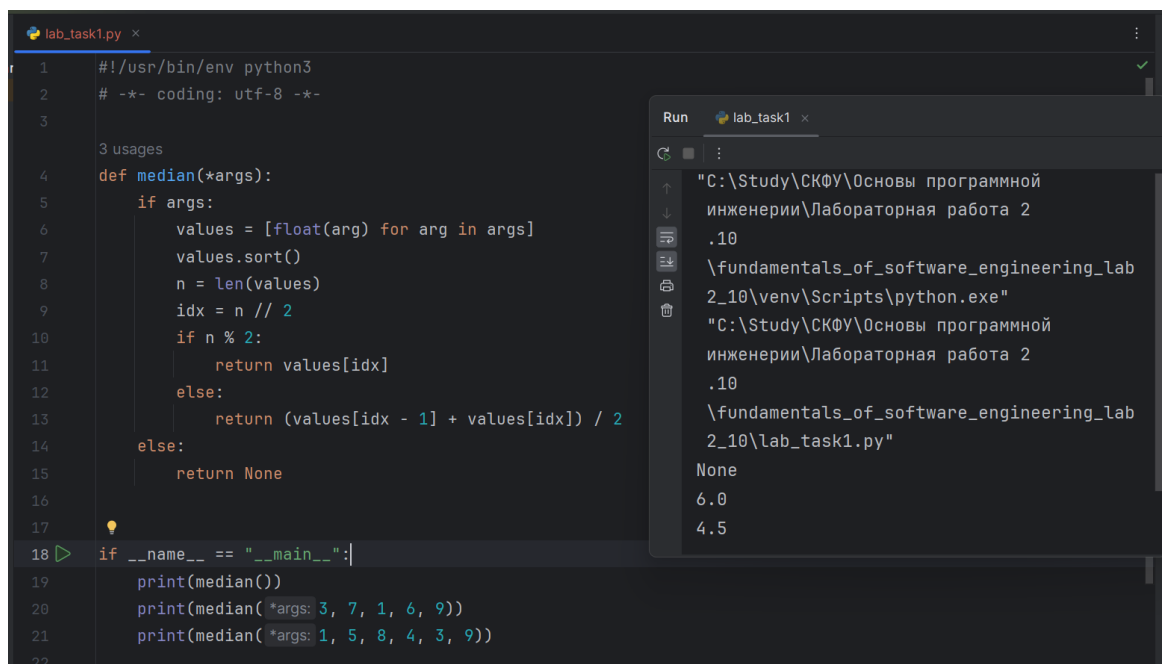


```
1 # Byte-compiled / optimized / DLL files
2 __pycache__/
3 *.py[cod]
4 *$py.class
5
6 # C extensions
7 *.so
8
9 # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
```

Рисунок 5 – Часть .gitignore файла, созданного GitHub

2. Проработаем пример лабораторной работы, фиксируя изменения. Создадим для примера отдельный модуль языка Python:

Задание: разработать функцию для определения медианы значений аргументов функции. Если функции передается пустой список аргументов, то она должна возвращать значение None:



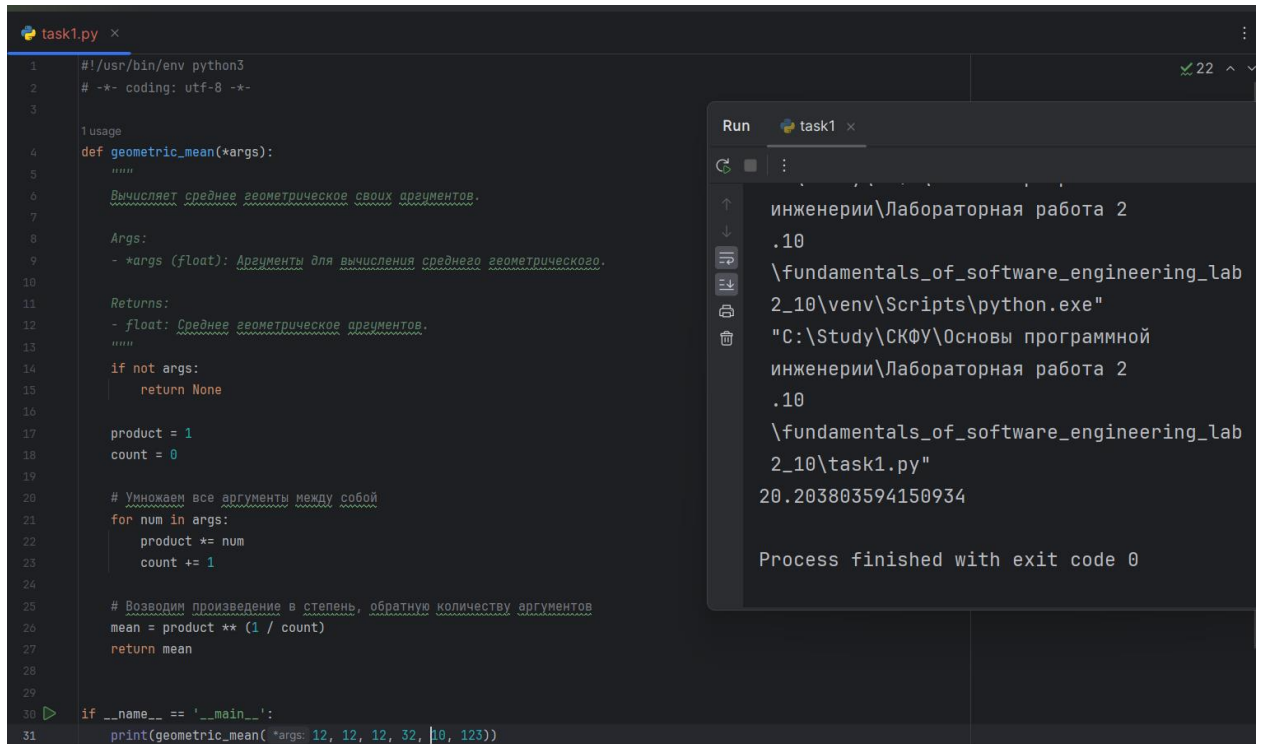
```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 3 usages
5 def median(*args):
6     if args:
7         values = [float(arg) for arg in args]
8         values.sort()
9         n = len(values)
10        idx = n // 2
11        if n % 2:
12            return values[idx]
13        else:
14            return (values[idx - 1] + values[idx]) / 2
15    else:
16        return None
17
18 if __name__ == "__main__":
19     print(median())
20     print(median(*args: 3, 7, 1, 6, 9))
21     print(median(*args: 1, 5, 8, 4, 3, 9))
22
```

Run lab_task1

```
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2
.10
\fundamentals_of_software_engineering_lab
2_10\venv\Scripts\python.exe"
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2
.10
\fundamentals_of_software_engineering_lab
2_10\lab_task1.py"
None
6.0
4.5
```

Рисунок 6 – Код программы и ее вывод в консоль

3. Решить поставленную задачу: написать функцию, вычисляющую среднее геометрическое своих аргументов a_1, a_2, \dots, a_n :



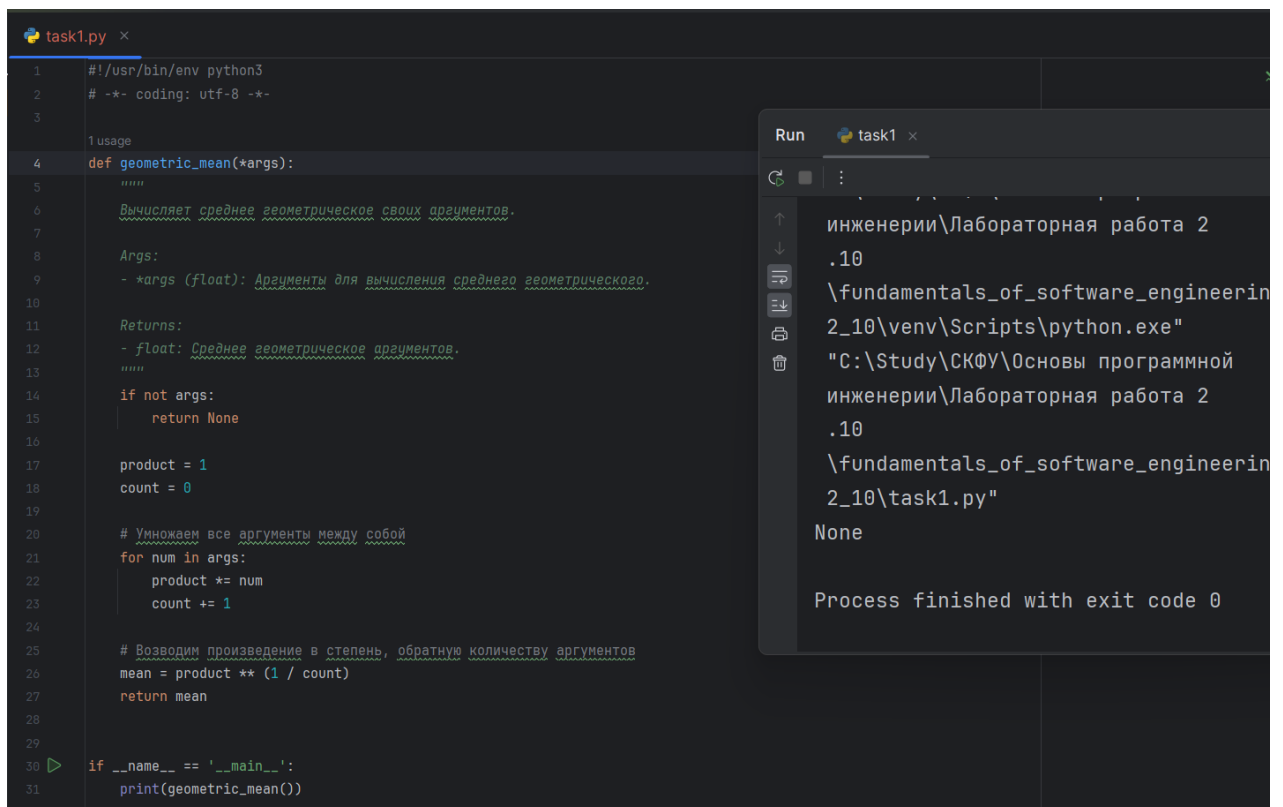
```
task1.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def geometric_mean(*args):
6      """
7      Вычисляет среднее геометрическое своих аргументов.
8
9      Args:
10         *args (float): Аргументы для вычисления среднего геометрического.
11
12     Returns:
13         - float: Среднее геометрическое аргументов.
14     """
15     if not args:
16         return None
17
18     product = 1
19     count = 0
20
21     # Умножаем все аргументы между собой
22     for num in args:
23         product *= num
24         count += 1
25
26     # Возводим произведение в степень, обратную количеству аргументов
27     mean = product ** (1 / count)
28     return mean
29
30 if __name__ == '__main__':
31     print(geometric_mean(*args=12, 12, 12, 32, 10, 123))
```

Run task1 x

инженерии\Лабораторная работа 2
.10
\\fundamentals_of_software_engineering_lab
2_10\venv\Scripts\python.exe"
"C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.10
\\fundamentals_of_software_engineering_lab
2_10\task1.py"
20.203803594150934

Process finished with exit code 0

Рисунок 7 – Код программы и ее вывод в консоль (1)



```
task1.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def geometric_mean(*args):
6      """
7      Вычисляет среднее геометрическое своих аргументов.
8
9      Args:
10         *args (float): Аргументы для вычисления среднего геометрического.
11
12     Returns:
13         - float: Среднее геометрическое аргументов.
14     """
15     if not args:
16         return None
17
18     product = 1
19     count = 0
20
21     # Умножаем все аргументы между собой
22     for num in args:
23         product *= num
24         count += 1
25
26     # Возводим произведение в степень, обратную количеству аргументов
27     mean = product ** (1 / count)
28     return mean
29
30 if __name__ == '__main__':
31     print(geometric_mean())
```

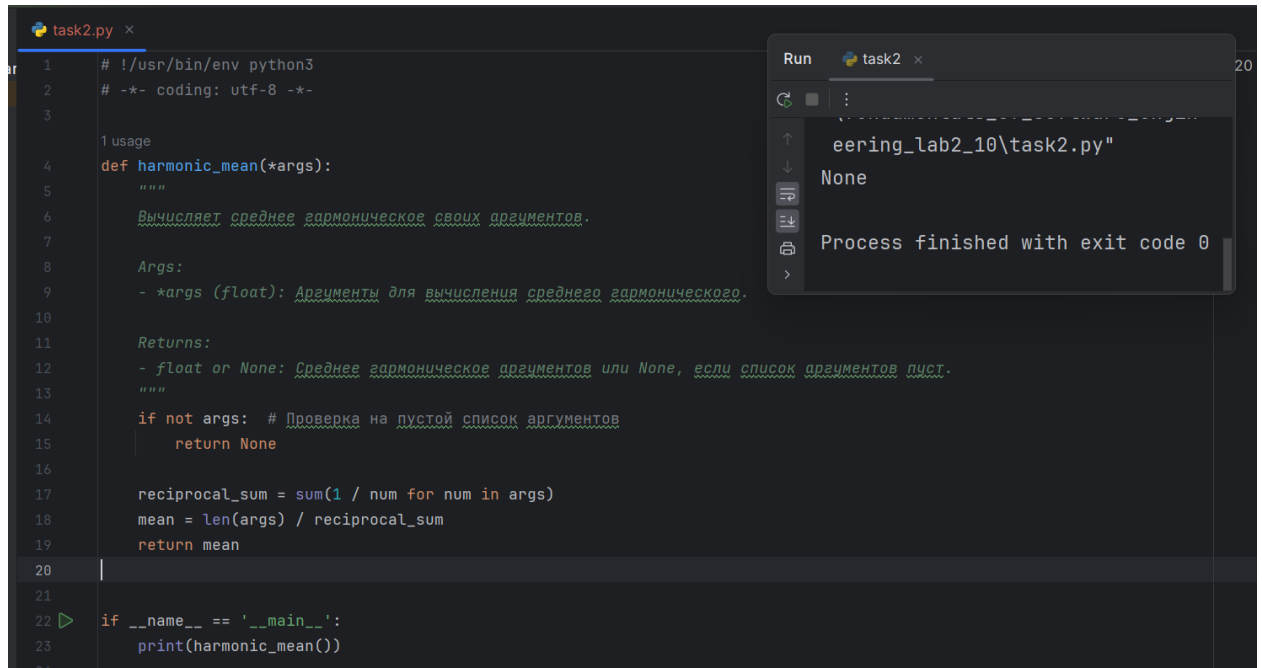
Run task1 x

инженерии\Лабораторная работа 2
.10
\\fundamentals_of_software_engineerin
2_10\venv\Scripts\python.exe"
"C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.10
\\fundamentals_of_software_engineerin
2_10\task1.py"
None

Process finished with exit code 0

Рисунок 8 – Код программы и ее вывод в консоль (2)

4. Решить поставленную задачу: написать функцию, вычисляющую среднее гармоническое своих аргументов a_1, a_2, \dots, a_n :



```
task2.py x
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 usage
5 def harmonic_mean(*args):
6     """
7     Вычисляет среднее гармоническое своих аргументов.
8
9     Args:
10     - *args (float): Аргументы для вычисления среднего гармонического.
11
12     Returns:
13     - float or None: Среднее гармоническое аргументов или None, если список аргументов пуст.
14     """
15     if not args: # Проверка на пустой список аргументов
16         return None
17
18     reciprocal_sum = sum(1 / num for num in args)
19     mean = len(args) / reciprocal_sum
20     return mean
21
22 if __name__ == '__main__':
23     print(harmonic_mean())
24
```

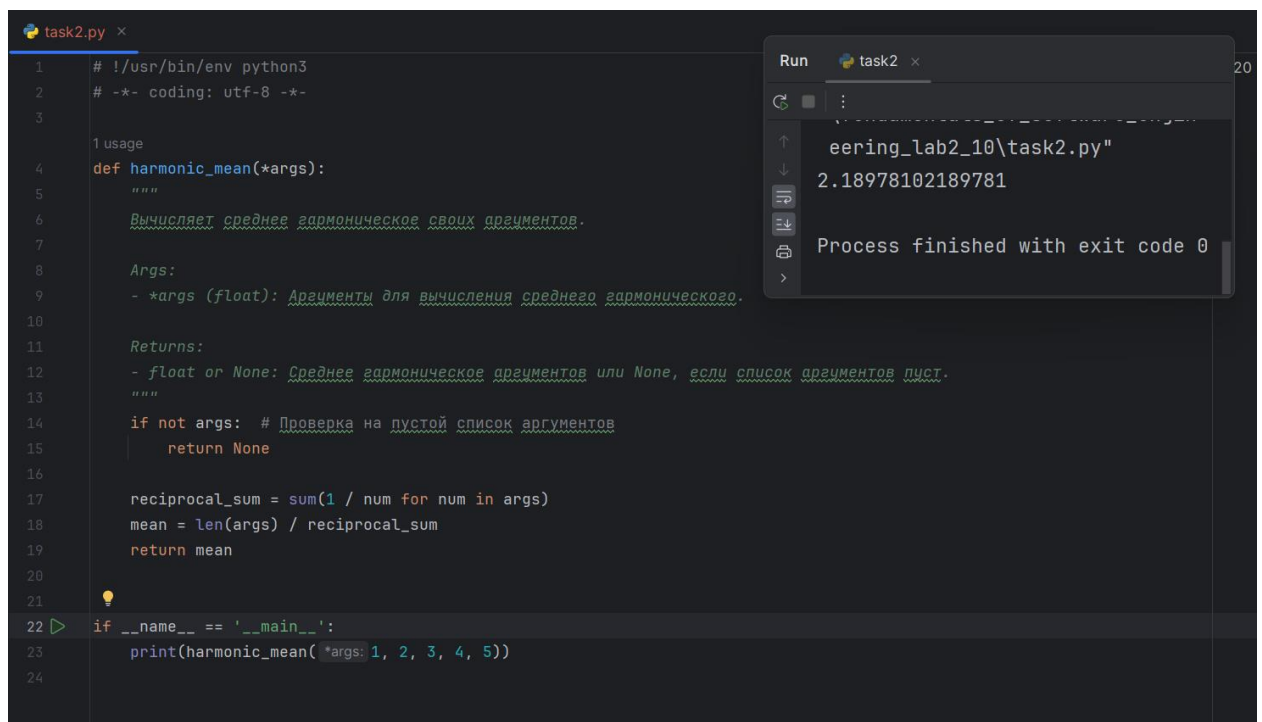
Run task2 x

engineering_lab2_10\task2.py"

None

Process finished with exit code 0

Рисунок 9 – Код программы и ее вывод в консоль (1)



```
task2.py x
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 usage
5 def harmonic_mean(*args):
6     """
7     Вычисляет среднее гармоническое своих аргументов.
8
9     Args:
10     - *args (float): Аргументы для вычисления среднего гармонического.
11
12     Returns:
13     - float or None: Среднее гармоническое аргументов или None, если список аргументов пуст.
14     """
15     if not args: # Проверка на пустой список аргументов
16         return None
17
18     reciprocal_sum = sum(1 / num for num in args)
19     mean = len(args) / reciprocal_sum
20     return mean
21
22 if __name__ == '__main__':
23     print(harmonic_mean(*args: 1, 2, 3, 4, 5))
24
```

Run task2 x

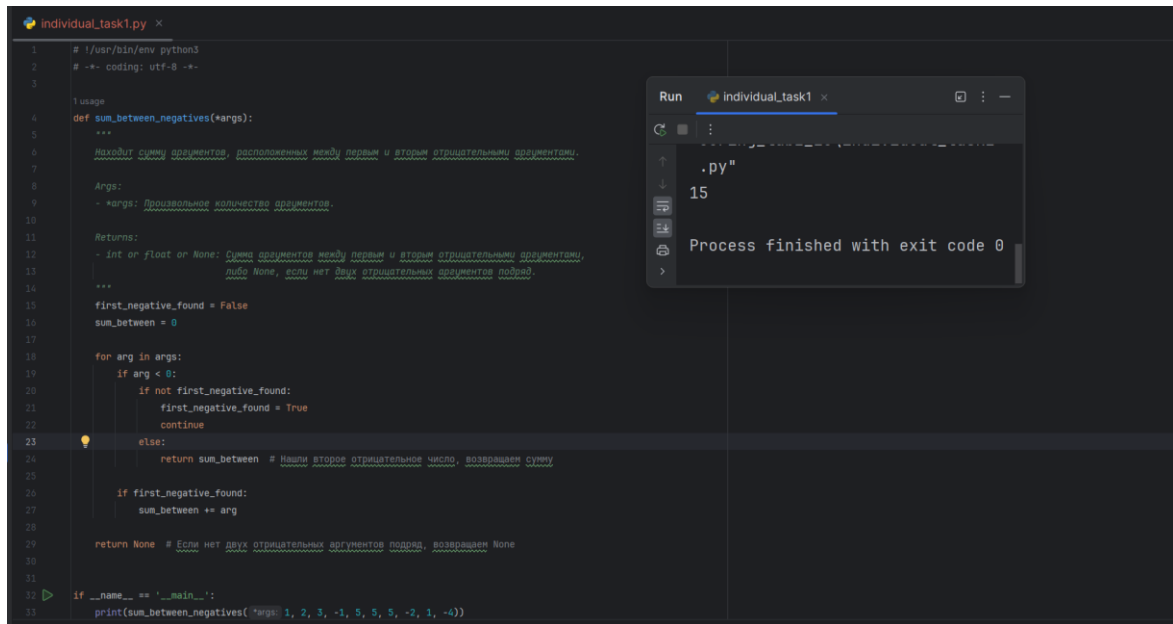
engineering_lab2_10\task2.py"

2.18978102189781

Process finished with exit code 0

Рисунок 10 – Код программы и ее вывод в консоль (2)

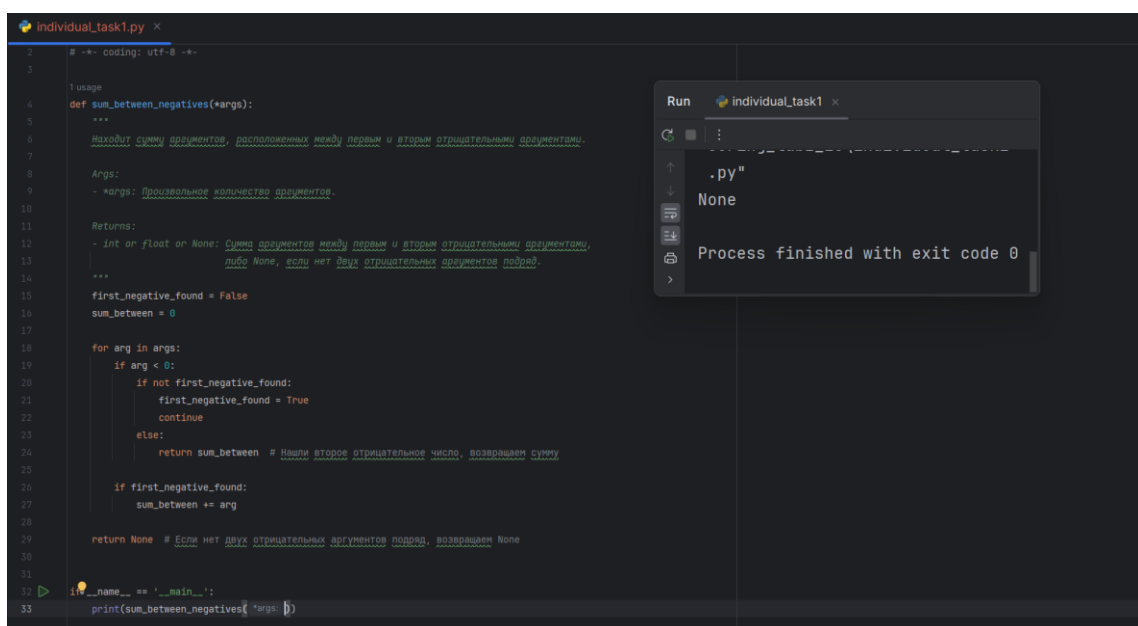
5. Решение индивидуального задания (вариант №7). Напишите функцию, принимающую произвольное количество аргументов, и возвращающую требуемое значение. Если функции передается пустой список аргументов, то она должна возвращать значение None. В процессе решения не использовать преобразования конструкции *args в список или иную структуру данных. Найти сумму аргументов, расположенных между первым и вторым отрицательными аргументами:



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 usage
5 def sum_between_negatives(*args):
6     """
7     Находит сумму аргументов, расположенных между первым и вторым отрицательными аргументами.
8
9     Args:
10         - *args: Произвольное количество аргументов.
11
12     Returns:
13         - int or float or None: Сумма аргументов между первым и вторым отрицательными аргументами,
14           либо None, если нет двух отрицательных аргументов подряд.
15     """
16     first_negative_found = False
17     sum_between = 0
18
19     for arg in args:
20         if arg < 0:
21             if not first_negative_found:
22                 first_negative_found = True
23                 continue
24             else:
25                 return sum_between # Нашли второе отрицательное число, возвращаем сумму
26
27             if first_negative_found:
28                 sum_between += arg
29
30     return None # Если нет двух отрицательных аргументов подряд, возвращаем None
31
32 if __name__ == '__main__':
33     print(sum_between_negatives(*args: 1, 2, 3, -1, 5, 5, -2, 1, -4))
```

Run individual_task1 .py" 15 Process finished with exit code 0

Рисунок 11 – Код программы и ее вывод в консоль (1)



```
2 # -*- coding: utf-8 -*-
3
4 usage
5 def sum_between_negatives(*args):
6     """
7     Находит сумму аргументов, расположенных между первым и вторым отрицательными аргументами.
8
9     Args:
10         - *args: Произвольное количество аргументов.
11
12     Returns:
13         - int or float or None: Сумма аргументов между первым и вторым отрицательными аргументами,
14           либо None, если нет двух отрицательных аргументов подряд.
15     """
16     first_negative_found = False
17     sum_between = 0
18
19     for arg in args:
20         if arg < 0:
21             if not first_negative_found:
22                 first_negative_found = True
23                 continue
24             else:
25                 return sum_between # Нашли второе отрицательное число, возвращаем сумму
26
27             if first_negative_found:
28                 sum_between += arg
29
30     return None # Если нет двух отрицательных аргументов подряд, возвращаем None
31
32 if __name__ == '__main__':
33     print(sum_between_negatives(*args: {}))
```

Run individual_task1 .py" None Process finished with exit code 0

Рисунок 12 – Код программы и ее вывод в консоль (1)

6. Сделаем merge веток main/develop и отправим изменения на удаленный репозиторий:

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10\fundamentals_of_software_engineering_lab2_10> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10\fundamentals_of_software_engineering_lab2_10> git merge develop
Updating 2c963d9..69bf7c2
Fast-forward
 .gitignore      | 2 +-
 individual_task1.py | 33 ++++++
 lab_task1.py     | 21 ++++++
 task1.py         | 31 ++++++
 task2.py         | 23 ++++++
 5 files changed, 109 insertions(+), 1 deletion(-)
 create mode 100644 individual_task1.py
 create mode 100644 lab_task1.py
 create mode 100644 task1.py
 create mode 100644 task2.py
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10\fundamentals_of_software_engineering_lab2_10>
```

Рисунок 13 – Merge веток main/develop

```
69bf7c2 (HEAD -> main, develop) individual_task1.py is added
d21707e task2.py is added
10bdd6f task1.py is added
ad019de lab_task1.py is added
2c963d9 (origin/main, origin/HEAD) Update README.md
30dbe26 Initial commit
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10\fundamentals_of_software_engineering_lab2_10>
```

Рисунок 14 – Коммиты проекта

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10\fundamentals_of_software_engineering_lab2_10> git push origin main
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 12 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 2.58 KiB | 2.58 MiB/s, done.
Total 13 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/PoItaTto/fundamentals_of_software_engineering_lab2_10
 2c963d9..69bf7c2  main -> main
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.10\fundamentals_of_software_engineering_lab2_10>
```

Рисунок 15 – Отправка изменений на удаленный репозиторий

Ответы на контрольные вопросы:

1. Какие аргументы называются позиционными в Python?

Позиционные аргументы – это аргументы функции, передаваемые в порядке их расположения, и связывающиеся с параметрами функции по позиции.

2. Какие аргументы называются именованными в Python?

Именованные аргументы – это аргументы, передаваемые в функцию с явным указанием их имени (ключевого слова), что позволяет привязывать значения к соответствующим параметрам функции независимо от порядка.

3. Для чего используется оператор *?

Оператор * используется для распаковки элементов из итерируемого объекта, такого как список или кортеж, в аргументы функции или в другой контейнер.

4. Каково назначение конструкций *args и **kwargs?

*args и **kwargs используются для работы с переменным числом аргументов в функциях. *args позволяет передавать произвольное количество позиционных аргументов в функцию, а **kwargs - произвольное количество именованных аргументов в виде словаря.