

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 №13
дисциплины «Основы программной инженерии»

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

(подпись)

Проверил: Воронкин Р. А.

(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.

Ход работы:


1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык 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 ().*

Owner *

 odik8


Repository name *

BSE13


 BSE13 is available.

Great repository names are short and memorable. Need inspiration? How about **fantastic-octo-umbrella** ?

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.](#)

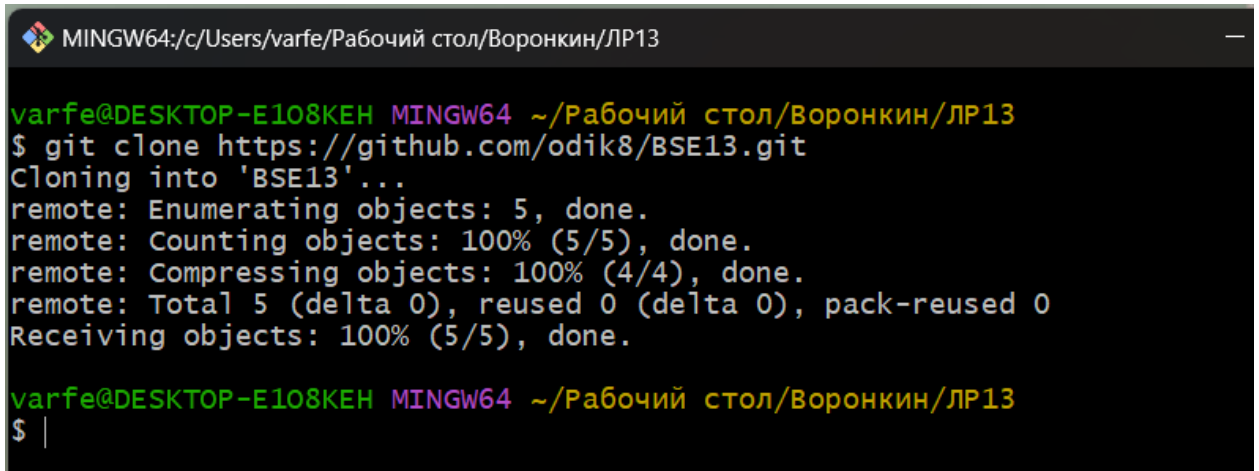
This will set  main as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

3. Выполнил клонирование созданного репозитория.



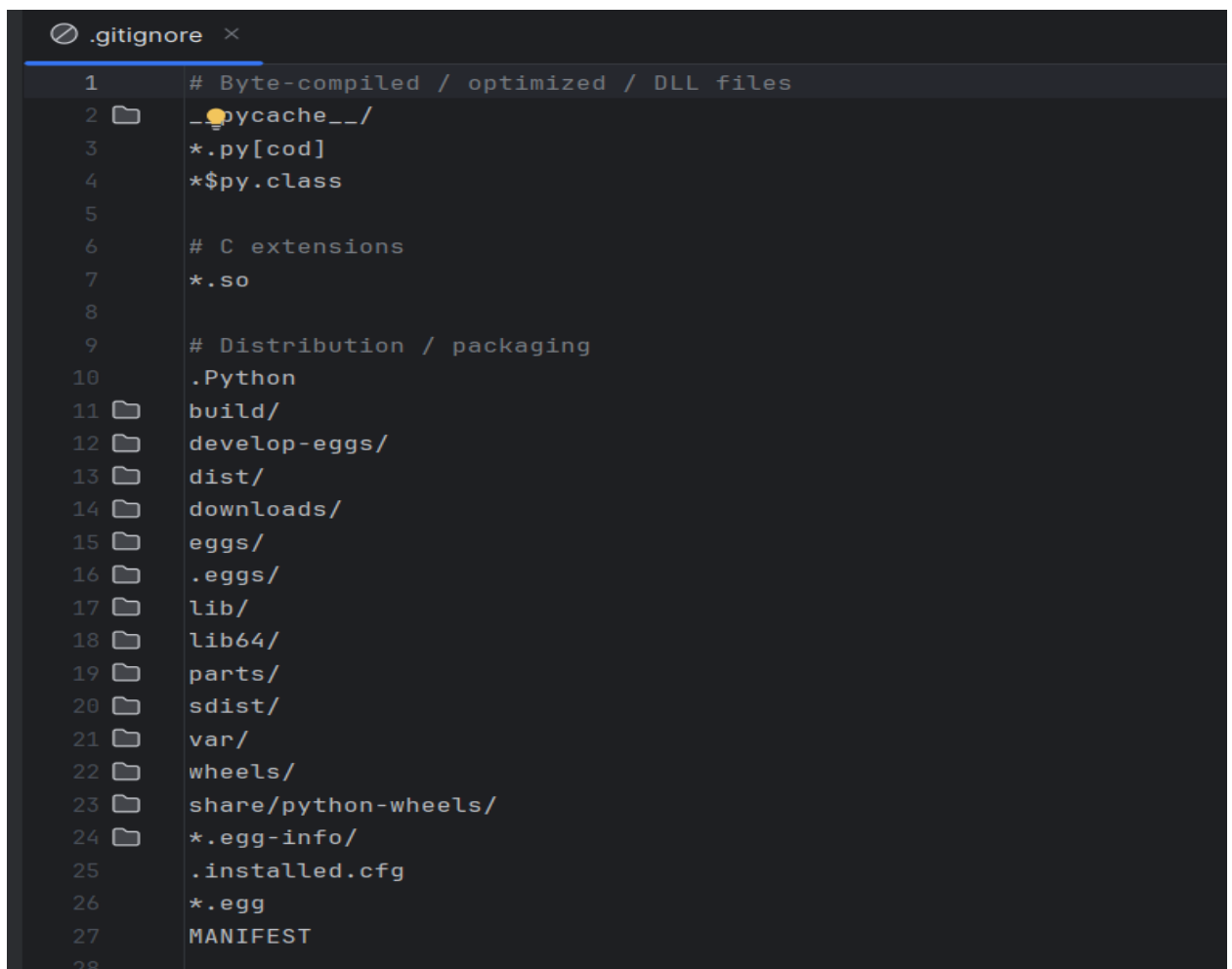
```
MINGW64:/c:/Users/varfe/Рабочий стол/Воронкин/ЛР13

varfe@DESKTOP-E108KEH MINGW64 ~/Рабочий стол/Воронкин/ЛР13
$ git clone https://github.com/odik8/BSE13.git
Cloning into 'BSE13'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

varfe@DESKTOP-E108KEH MINGW64 ~/Рабочий стол/Воронкин/ЛР13
$ |
```

Рисунок 2 – Клонирование репозитория

4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.



```
.gitignore x
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/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
```

Рисунок 2 – Файл .gitignore

5. Организовал свой репозиторий в соответствии с моделью ветвления git-flow.

```
varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/Воронкин/ЛР13
$ git flow init
Initialized empty Git repository in C:/Users/varfe/Рабочий стол/Воронкин/ЛР13/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [master]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/varfe/Рабочий стол/Воронкин/ЛР13/.git/hooks]

varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/Воронкин/ЛР13 (develop)
$ |
```

Рисунок 3 – Инициализация git-flow

6. Создал проект PyCharm в папке репозитория.

7. Решил поставленную задачу: написать функцию, вычисляющую среднее геометрическое своих аргументов

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage new *
5  def geometric_mean(*args):
6      if len(args) == 0:
7          return None
8
9      ans = 1
10     for i in args:
11         ans *= i
12     return ans ** (1 / len(args))
13
14  if __name__ == "__main__":
15     a = list(map(int, input("Введите значения через пробел: ").split()))
16     print(geometric_mean(*a))
17
```

Рисунок 4 – Код решения задачи

```
C:\Users\varfe\AppData\Local\Programs\Python\Py
Введите значения через пробел: 1 3 4 1
1.8612097182041991
```

Рисунок 5 – Результат работы программы

9. Решил поставленную задачу: написать функцию, вычисляющую среднее гармоническое своих аргументов:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  usage new *
6  def harmonic_mean(*args):
7      if len(args) == 0:
8          return None
9
10     reciprocal_sum = sum(1 / x for x in args)
11     harmonic_mean_value = len(args) / reciprocal_sum
12
13     return harmonic_mean_value
14
15  if __name__ == "__main__":
16     numbers = list(map(int, input("Введите значения через пробел: ").split()))
17     print(f"Среднее гармоническое чисел {numbers}: {harmonic_mean(*numbers)}")
18
```

Рисунок 6 – Код решения задачи

```
C:\Users\varfe\AppData\Local\Programs\Python\Python312\python.exe "
Введите значения через пробел: 1 2 3 4 5
Среднее гармоническое чисел [1, 2, 3, 4, 5]: 2.18978102189781
Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

Индивидуальное задание: Напишите функцию, принимающую произвольное количество аргументов, и возвращающую требуемое значение. Если функции передается пустой список аргументов, то она должна

возвращать значение None . Номер варианта определяется по согласованию с преподавателем. В процессе решения не использовать преобразования конструкции *args в список или иную структуру данных. Найти сумму аргументов, расположенных после первого положительного аргумента.

```
1  ✓#!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  1usage new*
6  ✓def sum_after_positive(*args):
7      found_positive = False
8      result_sum = 0
9
10     for arg in args:
11         if found_positive:
12             result_sum += arg
13         elif arg > 0:
14             found_positive = True
15
16     if found_positive:
17         return result_sum
18     else:
19         return None
20
21  ▶ ✓if __name__ == "__main__":
22      user_input = list(map(float, input("Введите числа через пробел: ").split()))
23
24      result = sum_after_positive(*user_input)
25      print(result)
26
```

Рисунок 8 – Код решения задачи

```
C:\Users\varfe\AppData\Local\Programs\Python\Python
Введите числа через пробел: -2 -4 1 3 4 -3
4.0

Process finished with exit code 0
```

Рисунок 9 – Результат выполнения кода

10. Зафиксировал сделанные изменения в репозитории.

11. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.

12. Выполнил слияние ветки для разработки с веткой main.

13. Отправил сделанные изменения на сервер GitHub.

14. Отправил адрес репозитория GitHub на электронный адрес преподавателя.

Вопросы для защиты работы:

1. **Позиционные аргументы в Python** — это аргументы, передаваемые в функцию в порядке, определенном в ее сигнатуре. Значения этих аргументов связываются с параметрами функции в том порядке, в котором они переданы.

Пример:

```
def example_function(arg1, arg2):
```

«код функции»

В этом примере **arg1** и **arg2** — это позиционные аргументы.

2. **Именованные аргументы в Python** — это аргументы, передаваемые в функцию с явным указанием их имени (параметра, которому они присваиваются). Это позволяет передавать аргументы в любом порядке и указывать только те, которые вам нужны.

Пример:

```
def example_function(arg1, arg2):
```

```
    example_function(arg2=2, arg1=1)
```

В этом примере **arg1** и **arg2** могут быть переданы в любом порядке с использованием имен.

3. Оператор ***** в Python используется для распаковки последовательности (например, списка или кортежа) при передаче аргументов в функцию или при создании другой последовательности.

Примеры:

```
def example_function(arg1, arg2):  
  
    my_list = [1, 2] example_function(*my_list)
```

В этом примере значения из списка **my_list** распаковываются и передаются как аргументы функции.

4. *Конструкции **args** и **kwargs** в Python используются для работы с переменным числом аргументов в функциях.

- ***args** используется для передачи переменного числа позиционных аргументов в функцию. Может быть использовано любое имя вместо **args**, но общепринято использовать именно ***args**.

Пример:

```
def example_function(*args): # код функции
```

- ****kwargs** используется для передачи переменного числа именованных аргументов в функцию. Аналогично, может быть использовано любое имя вместо **kwargs**.

Пример:

```
def example_function(**kwargs):
```

«код функции»

Вместе они могут использоваться как ***args**, ****kwargs** для обработки любого количества позиционных и именованных аргументов в функции.