

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙ-
СКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Основы кроссплатформенного программирования

Отчет по лабораторной работе №5

Тема: «Функции с переменным числом параметров в Python»

Выполнил студент группы

ИВТ-б-о-21-1

Харченко Б.Р. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, старший
преподаватель

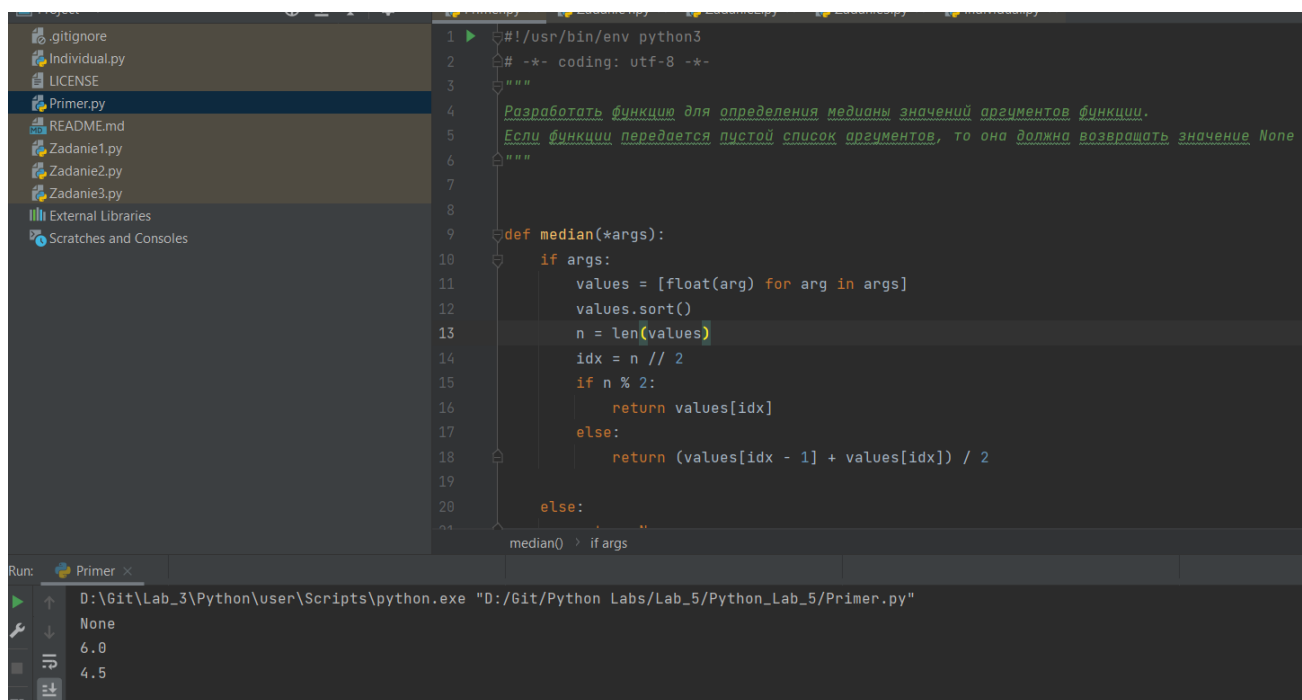
Воронкин Р.А.

(подпись)

Ставрополь 2022

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

Ход работы



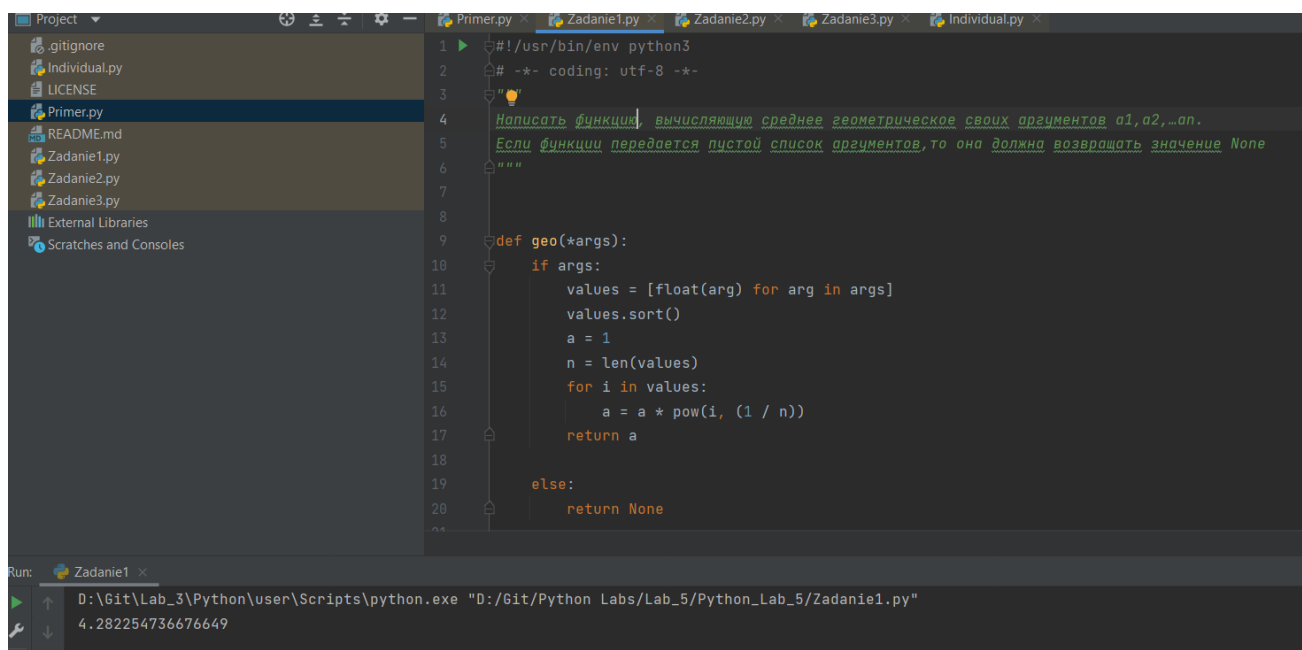
```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 Разработать функцию для определения медианы значений аргументов функции.
5 Если функции передается пустой список аргументов, то она должна возвращать значение None
6
7
8
9 def median(*args):
10     if args:
11         values = [float(arg) for arg in args]
12         values.sort()
13         n = len(values)
14         idx = n // 2
15         if n % 2:
16             return values[idx]
17         else:
18             return (values[idx - 1] + values[idx]) / 2
19
20     else:
21         return None
```

Run: Primer x

D:\Git\Lab_3\Python\user\Scripts\python.exe "D:\Git\Python Labs\Lab_5\Python_Lab_5\Primer.py"

None
6.0
4.5

Рисунок 1 – Результат выполнения примера



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 Написать функцию, вычисляющую среднее геометрическое своих аргументов a1,a2,...an.
5 Если функции передается пустой список аргументов, то она должна возвращать значение None
6
7
8
9 def geo(*args):
10     if args:
11         values = [float(arg) for arg in args]
12         values.sort()
13         a = 1
14         n = len(values)
15         for i in values:
16             a = a * pow(i, (1 / n))
17         return a
18
19     else:
20         return None
```

Run: Zadanie1 x

D:\Git\Lab_3\Python\user\Scripts\python.exe "D:\Git\Python Labs\Lab_5\Python_Lab_5\Zadanie1.py"

4.282254736676649

Рисунок 2 – Результат выполнения задания 1

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Решить поставленную задачу: написать функцию,
5 вычисляющую среднее гармоническое своих аргументов.
6 Если функции передается пустой список аргументов,
7 то она должна возвращать значение None.
8 """
9
10
11 def garmonic(*args):
12
13     if args:
14         values = [float(arg) for arg in args]
15         n = len(values)
16         a = 0
17         for i in values:
18             a = a + (1 / i)
19         return n / a
20
21
22 Run: Zadanie2
23 D:\Git\Lab_3\Python\user\Scripts\python.exe "D:/Git/Python Labs/Lab_5/Python_Lab_5/Zadanie2.py"
24 2.258064516129032
25
26 Process finished with exit code 0
```

Рисунок 3 – Результат 2 задания

```
1 def info(**args):
2     print("Data type of argument: ", type(args))
3
4     for key, value in args.items():
5         print("{} is {}".format(key, value))
6
7 if __name__ == '__main__':
8     info(Firstname="Nikita", Lastname="Nazarov", Age=19, Phone=9614624711)
9     info(Firstname="Bogdan", Lastname="Harchenko", Email="harchenkobo2003@gmail.com")
10
11
12 Run: Zadanie3
13 Firstname is Bogdan
14 Lastname is Harchenko
15 Email is harchenkobo2003@gmail.com
16 Country is Russian Federation
17 Age is 19
18 Phone is 9628535082
```

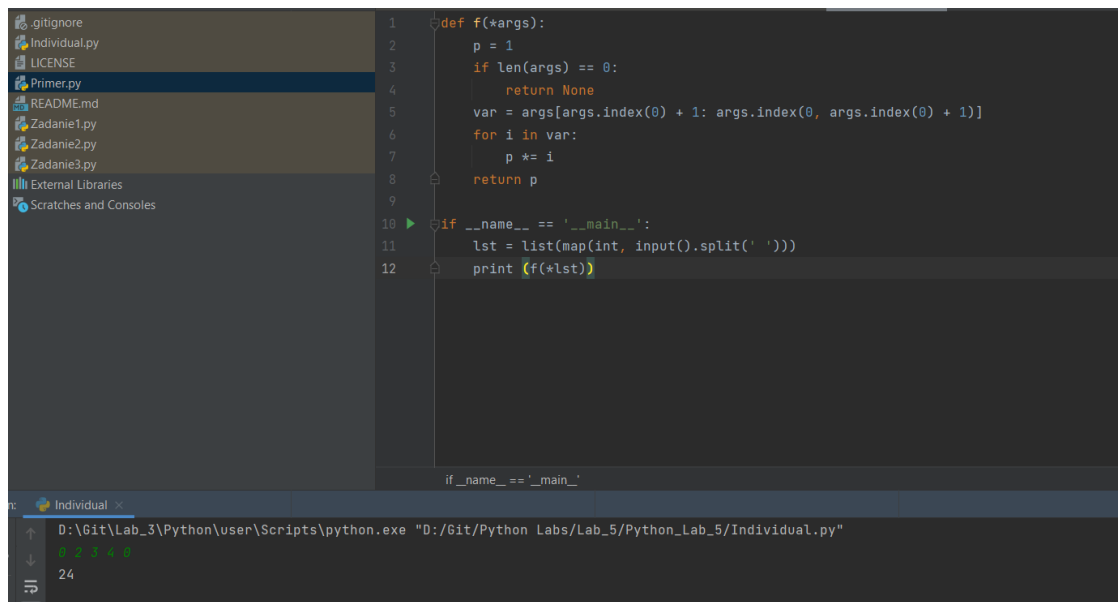
Рисунок 4 – Результат задания 3

1. Индивидуальное задание. Вариант – 24.

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.

Произведение аргументов, расположенных между первым и вторым нулевыми аргументами.



```
1 def f(*args):
2     p = 1
3     if len(args) == 0:
4         return None
5     var = args[args.index(0) + 1: args.index(0, args.index(0) + 1)]
6     for i in var:
7         p *= i
8     return p
9
10 if __name__ == '__main__':
11     lst = list(map(int, input().split(' ')))
12     print(f(*lst))
```

Individual x
D:\Git\Lab_3\Python\user\Scripts\python.exe "D:/Git/Python Labs/Lab_5/Python_Lab_5/Individual.py"
24

Рисунок 5 – Результат выполнения индивидуального задания

Вывод: в результате выполнения лабораторной работы были приобретены практические навыки и теоретические сведения по работе с функциями с переменным числом параметров при написании программ с помощью языка программирования Python версии 3.x.

Ответы на контрольные вопросы:

1. Какие аргументы называются позиционными в Python?

Аргументы, которые передаются без указания имен называются позиционными, потому что именно по позиции, расположению аргумента, функция понимает, какому параметру он соответствует.

2. Какие аргументы называются именованными в Python?

Аргументы, передаваемые с именами, называются именованными. При вызове функции можно использовать имена параметров из ее определения.

3. Для чего используется оператор `*`?

Оператор `*` чаще всего ассоциируется у людей с операцией умножения, но в Python он имеет и другой смысл. Этот оператор позволяет «распаковывать» объекты, внутри которых хранятся некие элементы. Вот пример: `a = [1, 2, 3]` `b = [*a, 4, 5, 6]` `print(b)` `# [1, 2, 3, 4, 5, 6]` Тут берётся содержимое списка `a`, распаковывается, и помещается в список `b`.

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

Итак, мы знаем о том, что оператор «звёздочка» в Python способен «вытаскивать» из объектов составляющие их элементы. Знаем мы и о том, что существует два вида параметров функций. А именно, `*args` — это сокращение от «arguments» (аргументы), а `**kwargs` — сокращение от «keyword arguments» (именованные аргументы). Каждая из этих конструкций используется для распаковки аргументов соответствующего типа, позволяя вызывать функции со списком аргументов переменной длины.

Важно помнить, что «args» — это всего лишь набор символов, которым принято обозначать аргументы. Самое главное тут — это оператор `*`. А то, что именно идёт после него, особой роли не играет. Благодаря использованию `*` мы создали список позиционных аргументов на основе того, что было передано функции при вызове. После того, как мы разобрались с `*args`, с пониманием `**kwargs` проблем быть уже не должно. Имя, опять же, значения не имеет. Главное — это два символа `**`. Благодаря им создаётся словарь, в котором содержатся именованные аргументы, переданные функции при её вызове.

при написании программ с помощью языка программирования Python версии 3.x..

Ответы на контрольные вопросы:

1. Что такое замыкание?

Замыкание (closure) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами.

2. Как реализованы замыкания в языке программирования Python?

Замыкания в Python реализованы посредством манипулирования областью видимости функций.

3. Что подразумевает под собой область видимости Local?

Эту область видимости имеют переменные, которые создаются и используются внутри функций.

```
>>> def add_two(a):
```

```
    x = 2
```

```
    return a + x
```

```
>>> add_two(3)
```

```
5
```

```
>>> print(x)
```

```
Traceback (most recent call last):
```

```
File "<pyshell#5>", line 1, in <module> print(x)
```

```
NameError: name 'x' is not defined
```


Пример. В данной программе объявлена функция `add_two()`, которая прибавляет двойку к переданному ей числу и возвращает полученный результат. Внутри этой функции используется переменная `x`, доступ к которой снаружи невозможен. К тому же, эта переменная удаляется из памяти каждый раз (во всяком случае, должна удаляться), когда завершается `add_two()`.

4. Что подразумевает под собой область видимости Enclosing?

Суть данной области видимости в том, что внутри функции могут быть вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в `enclosing` области видимости.

```
>>> def add_four(a):  
  
    x = 2  
  
    def add_some():  
  
        print("x = " + str(x))  
  
        return a + x  
  
    return add_some()  
  
>>> add_four(5)  
  
x = 2  
  
7
```

5. Что подразумевает под собой область видимости Global?

Переменные области видимости `global` – это глобальные переменные уровня модуля (модуль – это файл с расширением `.py`).

```
>>> x = 4  
  
>>> def fun():  
  
    print(x+3)  
  
>>> fun()
```

6. Что подразумевает под собой область видимости Build-in?

Уровень Python интерпретатора. В рамках этой области видимости находятся функции `open`, `len` и т. п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. Built-in – это максимально широкая область видимости.

7. Как использовать замыкания в языке программирования Python?

Для создания замыкания в Python, должны быть выполнены следующие пункты:

- У нас должна быть вложенная функция (функция внутри функции);
- вложенная функция должна ссылаться на значение, определенное в объемлющей функции;
- объемлющая функция должна возвращать вложенную функцию.

8. Как замыкания могут быть использованы для построения иерархических данных?

Начнем разбор данного термина с математической точки зрения, а точнее с алгебраической. Предметом алгебры является изучение алгебраических структур – множеств с определенными на них операциями. Под множеством обычно понимается совокупность определенных объектов. Наиболее простым примером числового множества, является множество натуральных чисел. Оно содержит следующие числа: 1, 2, 3, ... и т.д. до бесконечности. Иногда, к этому множеству относят число ноль, но мы не будем этого делать. Над элементами этого множества можно производить различные операции, например, сложение.

Какие бы натуральные числа мы не складывали, всегда будем получать натуральное число. С умножением точно также. Но с вычитанием и делением это условие не выполняется.

Среди натуральных чисел нет числа -3 , для того, чтобы можно было использовать вычитание без ограничений, нам необходимо расширить множество натуральных чисел до множества целых чисел:

Таким образом, можно сказать, что множество натуральных чисел замкнуто относительно операции сложения – какие бы натуральные числа мы не складывали, получим натуральное число, но это множество не замкнуто относительно операции вычитания.

Теперь перейдем с уровня математики на уровень функционального программирования. Вот как определяется “свойство замыкания” в книге “Структура и интерпретация компьютерных программ” Айбельсона Х., Сассмана Д. Д.: **“В общем случае, операция комбинирования объектов данных обладает свойством замыкания в том случае, если результаты соединения объектов с помощью этой операции сами могут соединяться этой же операцией”**.