

**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 №2.8**

Тема: «Работа с функциями в языке Python»

Выполнил студент группы

ИВТ-б-о-21-1

Горшков В.И. «    » \_\_\_\_\_ 20\_\_ г.

Подпись студента \_\_\_\_\_

Работа защищена «    » \_\_\_\_\_ 20\_\_ г.

Проверил доцент

Кафедры инфокоммуникаций, старший  
преподаватель

Воронкин Р.А.

\_\_\_\_\_  
(подпись)

Ставрополь 2022

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

## Ход работы:

**1. Создал репозиторий в GitHub**, дополнил правила в .gitignore для работы с IDE PyCharm с ЯП Python, выбрал лицензию MIT, клонировал его на компьютер и организовал в соответствии с моделью ветвления git-flow.

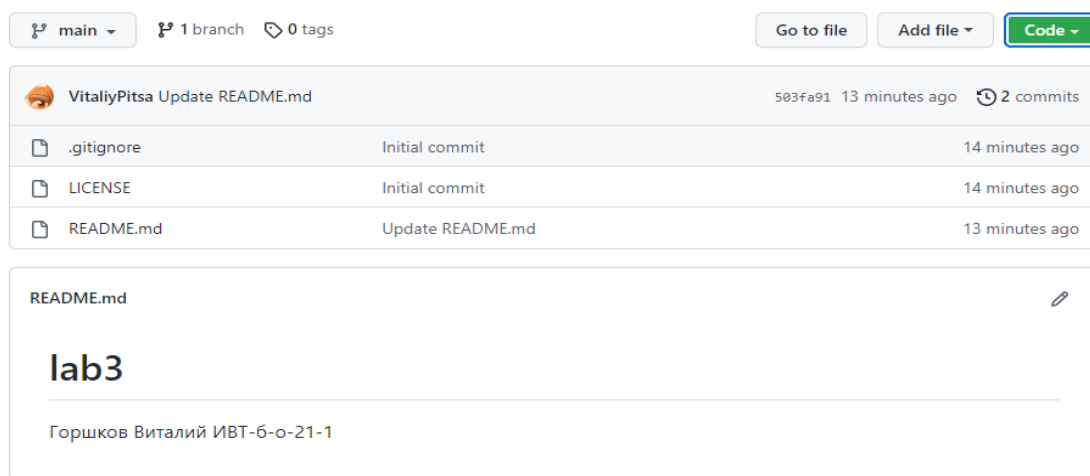


Рисунок 1.1 – Созданный репозиторий

```
.gitignore – Блокнот
Файл Правка Формат Вид Справка
.idea/
# Created by https://www.toptal.com/developers/gitignore/api/python,pycharm
# Edit at https://www.toptal.com/developers/gitignore?templates=python,pycharm

### PyCharm ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio, WebStorm and Rider
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839

# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf

# AWS User-specific
.idea/**/aws.xml

# Generated files
.idea/**/contentModel.xml

# Sensitive or high-churn files
.idea/**/dataSources/
.idea/**/dataSources.ids
.idea/**/dataSources.local.xml
```

Рисунок 1.2 – Дополнил правила в .gitignore

```

c:\Users\Admin\Desktop\git\Python3>git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

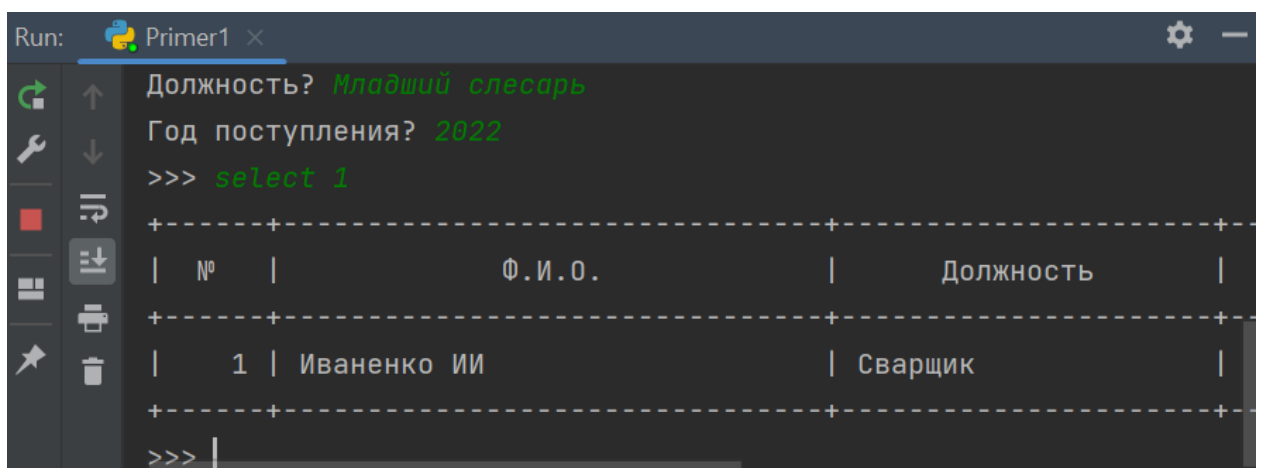
How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/Admin/Desktop/git/Python3/.git/hooks]

c:\Users\Admin\Desktop\git\Python3>

```

Рисунок 1.3 – Организация репозитория в соответствии с моделью ветвления git-flow

**2. Создал проект Pycharm в папке репозитория, проработал примеры ЛР.**



```

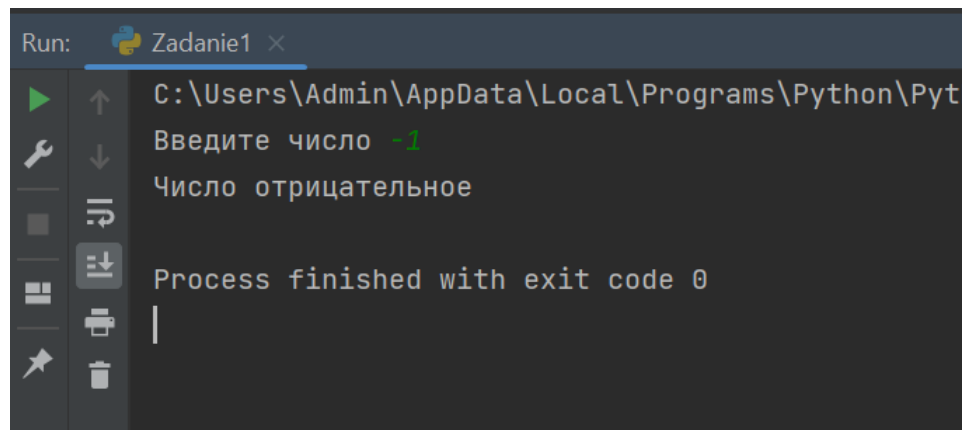
Run: Primer1 x
Должность? Младший слесарь
Год поступления? 2022
>>> select 1
+-----+-----+-----+-----+
| № |                Ф.И.О. |                Должность |
+-----+-----+-----+-----+
|  1 | Иваненко ИИ | Сварщик |
+-----+-----+-----+-----+
>>>

```

Рисунок 2.1 – Результат выполнения примера №1

**3. Задание №1. Решить следующую задачу:** основная ветка программы, не считая заголовков функций, состоит из двух строки кода. Это вы-

зов функции `test()` и инструкции `if __name__ == '__main__':`. В ней запрашивается на ввод целое число. Если оно положительное, то вызывается функция `positive()`, тело которой содержит команду вывода на экран слова "Положительное". Если число отрицательное, то вызывается функция `negative()`, ее тело содержит выражение вывода на экран слова "Отрицательное". Понятно, что вызов `test()` должен следовать после определения функций. Однако имеет ли значение порядок определения самих функций? То есть должны ли определения `positive()` и `negative()` предшествовать `test()` или могут следовать после него? Проверьте вашу гипотезу, поменяв объявления функций местами. Попробуйте объяснить результат.

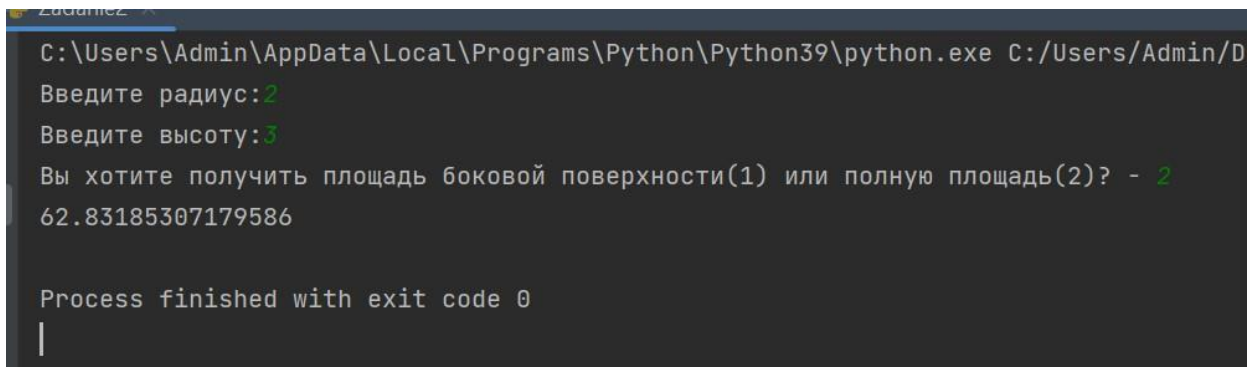


```
Run: Zadanie1 x
C:\Users\Admin\AppData\Local\Programs\Python\Python38\python.exe
Введите число -1
Число отрицательное
Process finished with exit code 0
```

Рисунок 3.1 – Результат выполнения программы

**4. Задание №2. Решите следующую задачу:** в основной ветке программы вызывается функция `cylinder()`, которая вычисляет площадь цилиндра. В теле `cylinder()` определена функция `circle()`, вычисляющая площадь круга по

формуле . В теле `cylinder()` у пользователя спрашивается, хочет ли он получить только площадь боковой поверхности цилиндра, которая вычисляется по формуле , или полную площадь цилиндра. В последнем случае к площади боковой поверхности цилиндра должен добавляться удвоенный результат вычислений функции `circle()`.

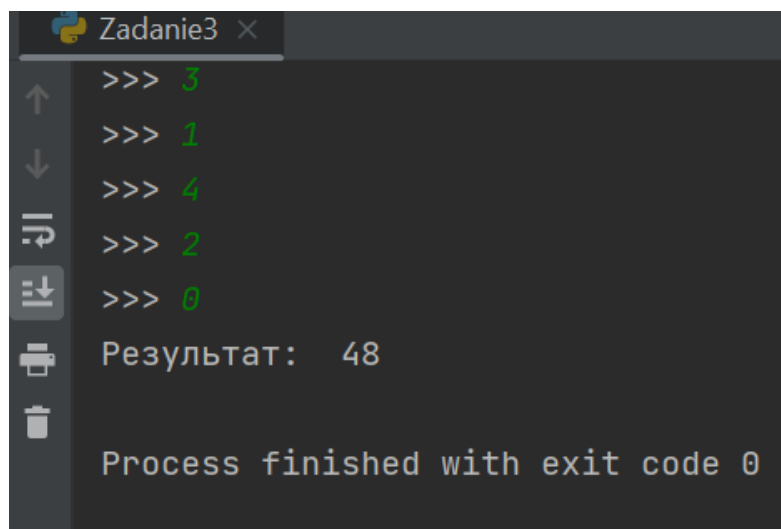


```
C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Admin/D
Введите радиус:2
Введите высоту:3
Вы хотите получить площадь боковой поверхности(1) или полную площадь(2)? - 2
62.83185307179586

Process finished with exit code 0
|
```

Рисунок 4.1 – Результат выполнения программы

**5. Задание №3. Решите следующую задачу:** напишите функцию, которая считывает с клавиатуры числа и перемножает их до тех пор, пока не будет введен 0. Функция должна возвращать полученное произведение. Вызовите функцию и выведите на экран результат ее работы.



```
>>> 3
>>> 1
>>> 4
>>> 2
>>> 0
Результат: 48
Process finished with exit code 0
```

Рисунок 5.1 – Результат выполнения программы

**6. Задание №4. Решите следующую задачу:** напишите программу, в которой определены следующие четыре функции:

1. Функция `get_input()` не имеет параметров, запрашивает ввод с клавиатуры и возвращает в основную программу полученную строку.

2. Функция `test_input()` имеет один параметр. В теле она проверяет, можно ли переданное ей значение преобразовать к целому числу. Если можно, возвращает логическое `True`.

Если нельзя – `False`.

3. Функция `str_to_int()` имеет один параметр. В теле преобразовывает переданное значение к целочисленному типу. Возвращает полученное число.

4. Функция `print_int()` имеет один параметр. Она выводит переданное значение на экран и ничего не возвращает.

```
C:\Users\Admin\AppData\Local\Programs\Python\Python3
Введите строку: 3232
3232

Process finished with exit code 0
```

Рисунок 6.1 – Результат выполнения программы

### Индивидуальное задание

**7. Решить индивидуальное задание лабораторной работы 2.6, оформив каждую команду в виде отдельной функции.**

```
+-----+-----+-----+-----+
| No | Пункт назначения | Номер поезда | время отправления |
+-----+-----+-----+-----+
| 1 | moscow | 1 | 13:00:00 |
+-----+-----+-----+-----+

>>> select
Введите номер поезда:
1
+-----+-----+-----+-----+
| No | Пункт назначения | Номер поезда | время отправления |
+-----+-----+-----+-----+
| 1 | moscow | 1 | 13:00:00 |
+-----+-----+-----+-----+

>>> |
```

Рисунок 7.1 – Результат выполнения программы

**Вывод:** в результате выполнения лабораторной работы были приобретены навыки для работы с функциями при написании программ с помощью языка программирования Python версии 3.x.

## **Ответы на контрольные вопросы:**

### **1. Каково назначение функций в языке программирования Python?**

**Оптимизировать код, сократить его.**

Функция в программировании представляет собой обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции.

Функции можно сравнить с небольшими программками, которые сами по себе, т. е. автономно, не исполняются, а встраиваются в обычную программу. Нередко их так и называют – подпрограммы.

Других ключевых отличий функций от программ нет. Функции также при необходимости могут получать и возвращать данные. Только обычно они их получают не с ввода (клавиатуры, файла и др.), а из вызывающей программы. Сюда же они возвращают результат своей работы.

### **2. Каково назначение операторов `def` и `return`?**

**В языке программирования Python функции определяются с помощью оператора `def`.**

Рассмотрим код:

```
def countFood():  
  
    a = int(input())  
  
    b = int(input())  
  
    print("Всего", a+b, "шт.")
```

Это пример определения функции. Как и другие сложные инструкции вроде условного оператора и циклов функция состоит из заголовка и тела. Заголовок оканчивается двоеточием и переходом на новую строку. Тело имеет отступ.



**Ключевое слово `def` сообщает интерпретатору, что перед ним определение функции. За `def` следует имя функции. Оно может быть любым, также как и всякий идентификатор, например, переменная. В программировании весьма желательно давать всему осмысленные имена. Так в данном случае функция названа "посчитатьЕду" в переводе на русский.**

После имени функции ставятся скобки. В приведенном примере они пустые. Это значит, что функция не принимает никакие данные из вызывающей ее программы. Однако она могла бы их принимать, и тогда в скобках были бы указаны так называемые параметры.

После двоеточия следует тело, содержащее инструкции, которые выполняются при вызове функции. Следует различать определение функции и ее вызов. В программном коде они не рядом и не вместе. Можно определить функцию, но ни разу ее не вызвать. Нельзя вызвать функцию, которая не была определена. Определив функцию, но ни разу не вызвав ее, вы никогда не выполните ее тела.

Возврат значений из функции. Оператор `return`

Функции могут передавать какие-либо данные из своих тел в основную ветку программы.

Говорят, что функция возвращает значение. В большинстве языков программирования, в том числе Python, выход из функции и передача данных в то место, откуда она была вызвана, выполняется оператором `return`.

**Если интерпретатор Питона, выполняя тело функции, встречает `return`, то он "забирает" значение, указанное после этой команды, и "уходит" из функции.**

```
import math

def cylinder():

    r = float(input())
```

```

h = float(input())

# площадь боковой поверхности цилиндра:

side = 2 * math.pi * r * h

# площадь одного основания цилиндра:

circle = math.pi * r**2

# полная площадь цилиндра:

full = side + 2 * circle

return full

square = cylinder()

print(square)

```

Результат: 3 7 188.4

В данной программе в основную ветку из функции возвращается значение локальной переменной `full`. Не сама переменная, а ее значение, в данном случае – какое-либо число, полученное в результате вычисления площади цилиндра.

В основной ветке программы это значение присваивается глобальной переменной `square`. То есть выражение `square = cylinder()` выполняется так:

1. Вызывается функция `cylinder()` .
2. Из нее возвращается значение.
3. Это значение присваивается переменной `square` .

### **3. Каково назначение локальных и глобальных переменных при написании функций в Python?**

В программировании особое внимание уделяется концепции о локальных и глобальных переменных, а также связанное с ними представление об областях видимости. Соответственно, локальные переменные видны только в

локальной области видимости, которой может выступать отдельно взятая функция. Глобальные переменные видны во всей программе. "Видны" – значит, известны, доступны. К ним можно обратиться по имени и получить связанное с ними значение.

К глобальной переменной можно обратиться из локальной области видимости. К локальной переменной нельзя обратиться из глобальной области видимости, потому что локальная переменная существует только в момент выполнения тела функции. При выходе из нее, локальные переменные исчезают. Компьютерная память, которая под них отводилась, освобождается. Когда функция будет снова вызвана, локальные переменные будут созданы заново.

#### **4. Как вернуть несколько значений из функции Python?**

##### **Возврат нескольких значений:**

В Питоне позволительно возвращать из функции несколько объектов, перечислив их через запятую после команды return пример: return side, full

#### **5. Какие существуют способы передачи значений в функцию?**

По ссылке и по значению

#### **6. Как задать значение аргументов функции по умолчанию?**

Аргументы по умолчанию в функциях Python – это те аргументы, которые принимают значения по умолчанию, если никакие явные значения не передаются этим аргументам из вызова функции. Давайте определим функцию с одним аргументом по умолчанию. **(В скобках после объявления функции)**

```
def find_square(integer1=2):  
  
    result = integer1 * integer1  
  
    return result
```

#### **7. Каково назначение lambda-выражений в языке Python?**

`lambda` – это выражение, а не инструкция. По этой причине ключевое слово `lambda` может появляться там, где синтаксис языка Python не позволяет использовать инструкцию `def`, – внутри литералов или в вызовах функций, например.

## **8. Как осуществляется документирование кода согласно PEP257?**

Если пояснение функции содержит одну строку, то достаточно двух кавычек с каждой стороны строки. Пример: ““Пояснение””. Если это многострочное пояснение, то необходимо три кавычки с каждой стороны. Пояснение находится в теле функции, сразу после её появления.

## **9. В чем особенность однострочных и многострочных форм строк документации?**

**Однострочные строки** используются для очевидных случаев и они должны действительно находиться на одной строке.

### **Пример:**

```
def kos_root():  
  
    """Вернёт путь к папке root KOS"""  
  
    global _kos_root  
  
    if _kos_root: return _kos_root  
  
    ...
```

Многострочные документации состоят из сводной строки (summary line) имеющей такую же структуру, как и однострочный docstring, после которой следует пустая линия, а затем более сложное описание. «Summary line» может быть использована средствами автоматического документирования; поэтому так важно располагать её на одной строке и после делать пропуск в одну линию. Сводная строка пишется сразу после открывающихся кавычек, но допускается сделать перенос и начать со следующей строки. [прим. после этого

предложения я был счастлив, ведь находились люди, которые упорно пытались мне доказать, что делать перенос ни в коем случае нельзя :- ) ] При этом, весь docstring должен иметь такой же отступ, как и открывающие кавычки первой строки (см. пример ниже).