

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 №11
дисциплины «Основы программной инженерии»

Выполнил:
Магомедов Имран Борисович
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:
Воронкин Р.А., кандидат
технических наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

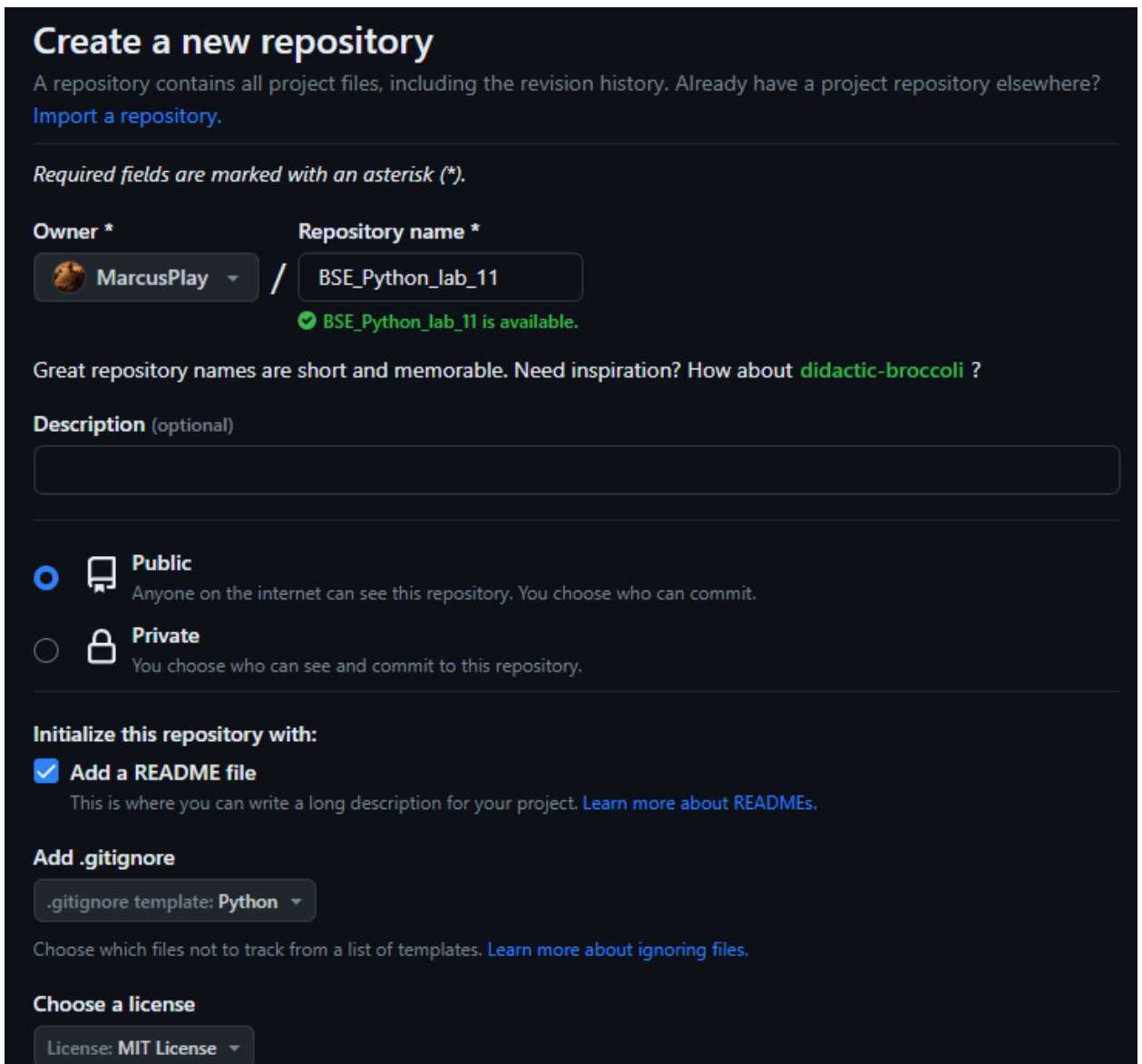
Ставрополь, 2023 г.

Тема: Работа с функциями в языке Python

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

Методика и порядок выполнения работы


1. Изучить теоретический материал работы.
2. Создать 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 *  **MarcusPlay** / **Repository name ***

 **BSE_Python_lab_11** is available.

Great repository names are short and memorable. Need inspiration? How about **didactic-broccoli** ?

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

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

Choose a license

3. Выполните клонирование созданного репозитория.

```
Imran@Kaskad MINGW64 ~/Desktop/Work
$ git clone https://github.com/MarcusPlay/BSE_Python_lab_11.git
Cloning into 'BSE_Python_lab_11'...
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.
```

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

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

```
Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

7. Проработайте примеры лабораторной работы. Зафиксируйте изменения.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5  from datetime import date
6
7
8  def get_worker():
9      """
10     Запросить данные о работнике.
11     """
12     name = input("Фамилия и инициалы? ")
13     post = input("Должность? ")
14     year = int(input("Год поступления? "))
15
16     # Создать словарь.
17     return {
18         'name': name,
19         'post': post,
20         'year': year,
21     }
```

```
>>> add
Фамилия и инициалы? Магомедов И.Б.
Должность? Стажер
Год поступления? 2023
>>> list
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Магомедов И.Б. | Стажер | 2023 |
+-----+-----+-----+-----+
>>> exit
```

```
ImranāKaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (develop)
$ git commit -m "added folder examples"
[develop 877aee7] added folder examples
1 file changed, 133 insertions(+)
create mode 100644 examples/example_1.py
```

8. Решить следующую задачу: основная ветка программы, не считая заголовков функций, состоит из двух строки кода. Это вызов 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.

```
def test(num):  
    if num >= 0:  
        positive()  
    else:  
        negative()  
  
def positive():  
    print('Положительное')  
  
def negative():  
    print('Отрицательное')  
  
if __name__ == "__main__":  
    test(int(input("Введите число: ")))
```

```
def positive():  
    print('Положительное')  
  
def negative():  
    print('Отрицательное')  
  
def test(num):  
    if num >= 0:  
        positive()  
    else:  
        negative()  
  
if __name__ == "__main__":  
    test(int(input("Введите число: ")))
```

Мы можем объявить функции в любом порядке перед вызовом test(), и программа будет работать корректно. В Python интерпретатор сначала проходит по коду и ищет определения функций, а затем выполняет основной код.

9. Зафиксируйте изменения в репозитории.

```
Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (develop)
$ git commit -m "added task_1.1.py and task_1.2.py"
[develop 8d5d680] added task_1.1.py and task_1.2.py
2 files changed, 51 insertions(+)
create mode 100644 task_1.1.py
create mode 100644 task_1.2.py
```

10. 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 πr^2 . В теле *cylinder()* у пользователя спрашивается, хочет ли он получить только площадь боковой поверхности цилиндра, которая вычисляется по формуле $2\pi rh$, или полную площадь цилиндра. В последнем случае к площади боковой поверхности цилиндра должен добавляться удвоенный результат вычислений функции *circle()*.

```
from math import pi

def cylinder(r, h):
    question = int(input("Вы хотите получить только площадь боковой поверхности цилиндра или полную площадь цилиндра?\n"\
        "Только площадь боковой поверхности цилиндра - 0\n"\
        "Полная площадь цилиндра - 1\n>>> "))
    if question == 0:
        print("Площадь боковой поверхности цилиндра = ", 2 * pi * r * h)
    elif question == 1:
        print("Полная площадь цилиндра = ", (2 * pi * r * h) + 2 * circle(r))
    else:
        print("Введено неправильное значение!")

def circle(r):
    return pi*(r**2)

if __name__ == "__main__":
    r = int(input("Введите радиус цилиндра: "))
    h = int(input("Введите высоту цилиндра: "))
    cylinder(r, h)
```

Введите радиус цилиндра: 2
Введите высоту цилиндра: 12
Вы хотите получить только площадь боковой поверхности цилиндра или полную площадь цилиндра?
Только площадь боковой поверхности цилиндра - 0
Полная площадь цилиндра - 1
>>> 0
Площадь боковой поверхности цилиндра = 150.79644737231007

11. Зафиксируйте изменения в репозитории.

```
Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (develop)
$ git commit -m "added task_2.py"
[develop f158700] added task_2.py
1 file changed, 31 insertions(+)
create mode 100644 task_2.py
```

12. 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.

```
def multiply_until_zero():
    product = 1
    while True:
        num = int(input("Введите число (для завершения введите 0): "))
        if num == 0:
            break
        product *= num
    return product

if __name__ == "__main__":
    result = multiply_until_zero()
    print(f"Результат перемножения: {result}")
```

```
Введите число (для завершения введите 0): 1
Введите число (для завершения введите 0): 2
Введите число (для завершения введите 0): 3
Введите число (для завершения введите 0): 5
Введите число (для завершения введите 0): 7
Введите число (для завершения введите 0): 9
Введите число (для завершения введите 0): 4
Введите число (для завершения введите 0): 4
Введите число (для завершения введите 0): 6
Введите число (для завершения введите 0): 8
Введите число (для завершения введите 0): 12
Введите число (для завершения введите 0): 0
Результат перемножения: 17418240
```

13. Зафиксируйте изменения в репозитории.

```
Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (develop)
$ git commit -m "added task_3.py"
[develop 04857d8] added task_3.py
1 file changed, 19 insertions(+)
create mode 100644 task_3.py
```

14. 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.

В основной ветке программы вызовите первую функцию. То, что она вернула, передайте во вторую функцию. Если вторая функция вернула *True*, то те же данные (из первой функции) передайте в третью функцию, а возвращенное третьей функцией значение – в четвертую.


```

def get_input():
    return input("Ввод: ")

def test_input(value):
    try:
        int(value)
        return True
    except ValueError:
        return False

def str_to_int(value):
    return int(value)

def print_int(value):
    print("Вывод:", value)

if __name__ == "__main__":
    a = get_input()
    if test_input(a):
        print_int(str_to_int(a))
    else:
        print_int("Error 404!!!")

```

Ввод: 45

Вывод: 45

Ввод: asdsf45

Вывод: Error 404!!!

15. Зафиксируйте изменения в репозитории.

```

Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (develop)
$ git commit -m "added task_4.py"
[develop 3cfc595] added task_4.py
1 file changed, 40 insertions(+)
create mode 100644 task_4.py

```

17. Приведите в отчете скриншоты работы программ решения индивидуального задания.

```

def add_user(users):
    name = input('Введите Фамилию и Имя: ')
    phone_number = input('Введите Номер телефона: ')
    year = list(map(int, input('Введите дату рождения (пример: 05 07 2004): ').split()))

    user = {
        'name': name,
        'phone_number': phone_number,
        'year': year,
    }

    users.append(user)

    if len(users) > 1:
        users.sort(key=lambda item: item.get('name', ''))

def list_users(users):
    print('Введите число месяца (1 - 12): ')
    while True:
        num = int(input())
        if num < 1 or num > 12:
            print("Значение введено неправильно! Попробуйте еще раз.")
        else:
            break

    line = '+-()-+--()-+--()-+--()-+'.format('-' * 4, '-' * 20, '-' * 18, '-' * 10)
    print(line)
    print('| {:^4} | {:^20} | {:^18} | {:^10} |'.format(
        "№",
        "Название",
        "Номер телефона",
        "Дата"
    ))
    print(line)

    for idx, user in enumerate(users, 1):
        if user['year'][1] == num:
            print(
                '| {:^4} | {:^20} | {:^18} | {:^10} |'.format(
                    idx,
                    user['name'],
                    user['phone_number'],
                    ''.join(map(str, user['year']))
                )
            )
            print(line)

if __name__ == "__main__":
    users = []

    while True:
        command = input("<| ").lower()

        if command == 'exit':
            break

        elif command == 'add':
            add_user(users)

        elif command == 'list':
            list_users(users)

```

```

~<| add
Введите Фамилию и Имя: Магомедов Иман
Введите Номер телефона: 89289795120
Введите дату рождения (пример: 05 07 2004): 05 07 2004
~<| list
Введите число месяца (1 - 12):
07
+-----+-----+-----+-----+
| № |      Название      |  Номер телефона  |   Дата   |
+-----+-----+-----+-----+
|  2 | Магомедов Иман    |  89289795120    |  5 7 2004 |
+-----+-----+-----+-----+

```

18. Зафиксируйте сделанные изменения в репозитории.

```

Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (develop)
$ git commit -m "added individual_task.py"
[develop a10af06] added individual_task.py
1 file changed, 74 insertions(+)
create mode 100644 individual_task.py

```

19. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.

20. Выполните слияние ветки для разработки с веткой *master/main*.

```

Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (main)
$ git merge develop
Updating 799387b..a10af06
Fast-forward
 examples/example_1.py | 133 +++++++++++++++++++++++++++++++++++++
 individual_task.py     |  74 +++++
 task_1.1.py           |  26 +++++
 task_1.2.py           |  25 +++++
 task_2.py             |  31 +++++
 task_3.py             |  19 +++++
 task_4.py             |  40 +++++
 7 files changed, 348 insertions(+)
 create mode 100644 examples/example_1.py
 create mode 100644 individual_task.py
 create mode 100644 task_1.1.py
 create mode 100644 task_1.2.py
 create mode 100644 task_2.py
 create mode 100644 task_3.py
 create mode 100644 task_4.py

```

21. Отправьте сделанные изменения на сервер GitHub.

```
Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_11 (main)
$ git push
Enumerating objects: 21, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 12 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (20/20), 6.42 KiB | 1.28 MiB/s, done.
Total 20 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), done.
To https://github.com/MarcusPlay/BSE_Python_lab_11.git
799387b..a10af06 main -> main
```

22. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Вопросы для защиты работы

1. Назначение функций в языке программирования Python:

Функции в Python предназначены для группировки повторяющихся блоков кода, чтобы обеспечить его повторное использование, улучшить читаемость кода и облегчить его тестирование. Функции также позволяют разделить программу на более мелкие и логические части, что упрощает разработку и поддержку кода.

2. Назначение операторов `def` и `return`:

- `def`: Оператор `def` используется для определения функций в Python. Он позволяет создавать пользовательские функции с определенным именем и набором инструкций.
- `return`: Оператор `return` используется внутри функций для возврата значения из функции. Он завершает выполнение функции и возвращает указанное значение.

3. Назначение локальных и глобальных переменных:

- Локальные переменные создаются внутри функции и существуют только в пределах этой функции. Они не видны за её пределами.
- Глобальные переменные создаются вне функций и могут быть использованы в любом месте программы, включая функции. Их видимость охватывает весь код.

4. Как вернуть несколько значений из функции Python:

Функция в Python может возвращать несколько значений с помощью кортежа, списка или других структур данных. Например:

```
def multiple_values():  
    return 1, 'hello', 3.14  
  
result = multiple_values()  
  
print(result)  # Вывод: (1, 'hello', 3.14)
```

5. Способы передачи значений в функцию:

- По значению (по умолчанию в Python).
- По ссылке (с использованием списков и словарей).
- С использованием именованных аргументов.
- С использованием аргументов по умолчанию.

6. Как задать значение аргументов функции по умолчанию:

Значения аргументов по умолчанию могут быть заданы при определении функции, например:

```
def example_function(arg1, arg2=10, arg3='default'):  
    # тело функции
```

7. Назначение lambda-выражений в Python:

Lambda-выражения (или анонимные функции) представляют собой компактный способ определения маленьких функций. Они обычно используются там, где требуется краткость, например, при передаче функции в качестве аргумента другой функции.

8. Документирование кода согласно PEP257:

PEP257 описывает стандарты документирования в Python. Для документирования кода используется тройная строка (docstring) в начале модуля, функции, класса или метода. Эта строка предоставляет описание и документацию к соответствующему элементу.

9. Особенности однострочных и многострочных форм строк документации:

- Однострочные строки документации используются для кратких описаний.
- Многострочные строки документации предоставляют более подробное описание и обычно охватывают несколько строк. В них могут быть указаны детали, параметры, возвращаемые значения и примеры использования.