

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра информационных систем и технологий

Отчет по лабораторной работе №11.

Дисциплина: «Основы программной инженерии»

Выполнил:

Студент группы ПИЖ-б-о-22-1,

направление подготовки: 09.03.04

«Программная инженерия»

ФИО: Джараян Арег Александрович

Проверил:

Воронкин Р. А.

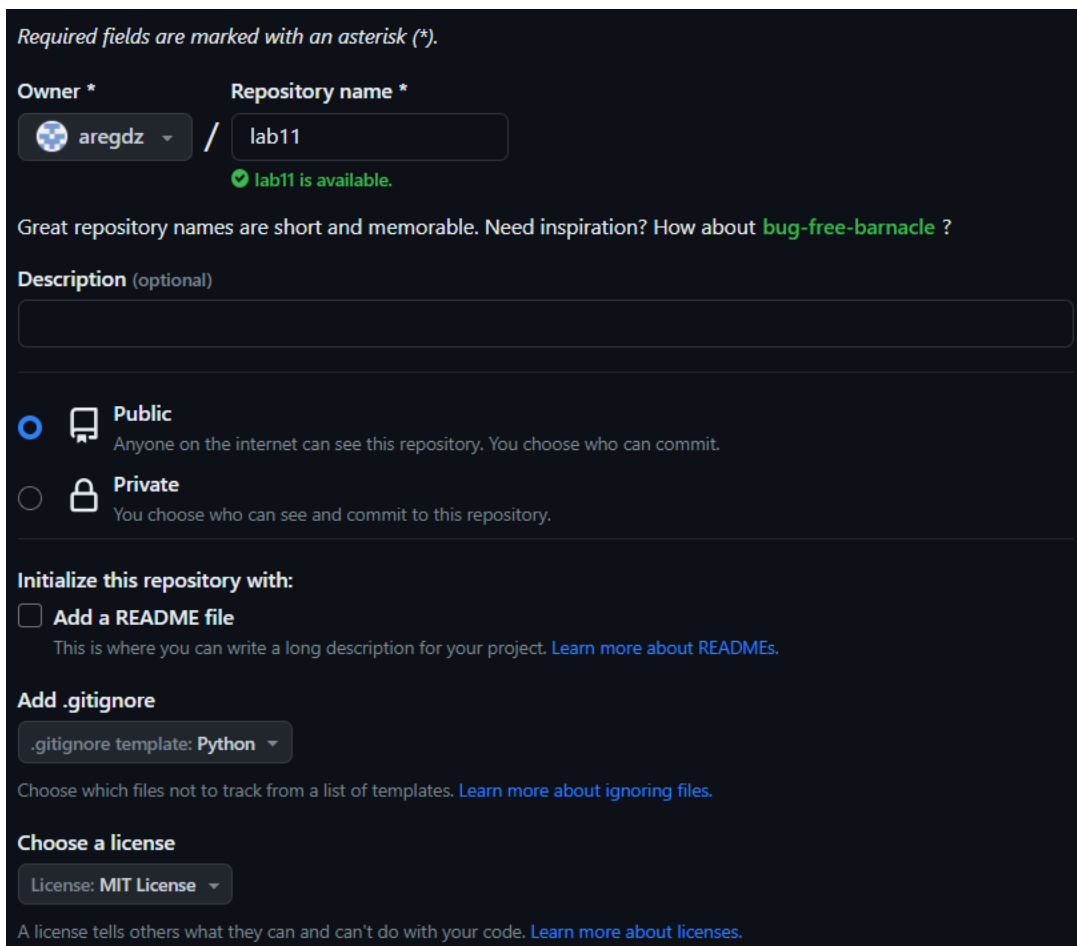
Ставрополь 2022

Тема: Лабораторная работа 2.8 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.

Выполнение работы:

1. Изучил теоретический материал работы.
2. Создал репозиторий на git.hub.



Required fields are marked with an asterisk (*).

Owner * Repository name *

aregdz / lab11

lab11 is available.

Great repository names are short and memorable. Need inspiration? How about [bug-free-barnacle](#) ?

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.](#)

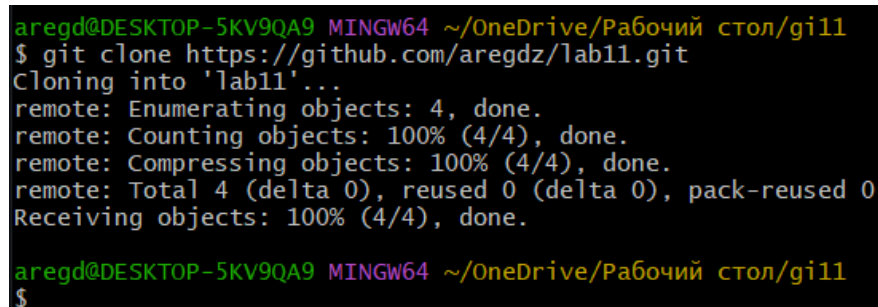
Choose a license

License: MIT License

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

Рисунок 1 – создание репозитория

3. Клонировал репозиторий.

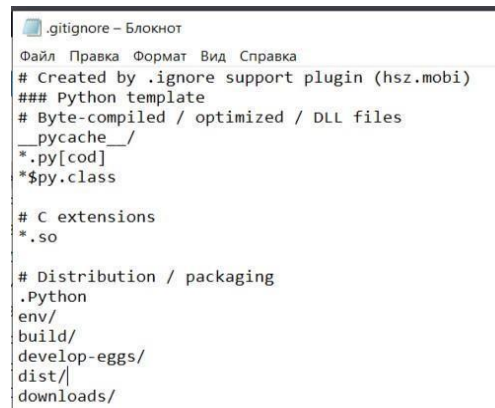


```
aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1
$ git clone https://github.com/aregdz/lab11.git
Cloning into 'lab11'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1
$
```

Рисунок 2 – клонирование репозитория 4.

4. Дополнить файл gitignore необходимыми правилами.

A screenshot of a text editor window titled ".gitignore - Блокнот". The editor shows a .gitignore file with the following content: "# Created by .ignore support plugin (hsz.mobi)", "### Python template", "# Byte-compiled / optimized / DLL files", "_pycache_/", "*.py[cod]", "*\$py.class", "# C extensions", "*.so", "# Distribution / packaging", ".Python", "env/", "build/", "develop-eggs/", "dist/", "downloads/".

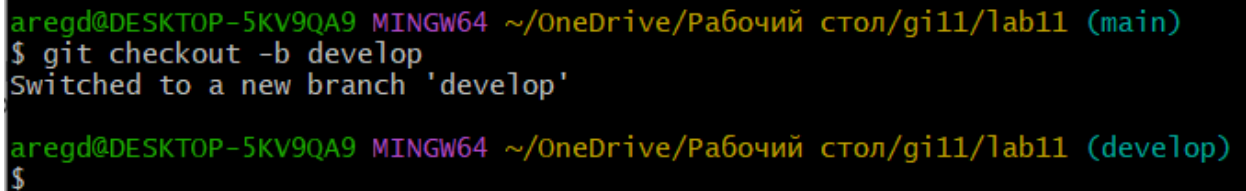
```
.gitignore - Блокнот
Файл  Правка  Формат  Вид  Справка
# Created by .ignore support plugin (hsz.mobi)
### Python template
# Byte-compiled / optimized / DLL files
_pycache_/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
env/
build/
develop-eggs/
dist/
downloads/
```

Рисунок 3 – .gitignore для IDE PyCharm

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

A screenshot of a terminal window with a black background and green text. The prompt is "aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1/lab11 (main)". The user enters the command "\$ git checkout -b develop". The output is "Switched to a new branch 'develop'". The prompt then changes to "aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1/lab11 (develop)".

```
aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1/lab11 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1/lab11 (develop)
$
```

Рисунок 4 – создание ветки develop

6. Проработать примеры из методички.

```

87     worker = get_worker()
88     # Добавить словарь в список.
89     workers.append(worker)
90     # Отсортировать список в случае необходимости.
91     if len(workers) > 1:
92         workers.sort(key=lambda item: item.get('name', ''))
93     elif command == 'list':
94         # Отобразить всех работников.
95         display_workers(workers)
96     elif command.startswith('select '):
97         # Разбить команду на части для выделения стажа.
98         parts = command.split( sep: ' ', maxsplit=1)
99         # Получить требуемый стаж.
100        period = int(parts[1])
101        # Выбрать работников с заданным стажем.
102        selected = select_workers(workers, period)
103        # Отобразить выбранных работников.
104        display_workers(selected)
105    elif command == 'help':
106        # Вывести справку о работе с программой.
107        print("Список команд:\n")
108        print("add - добавить работника;")
109        print("list - вывести список работников;")
110        print("select <стаж> - запросить работников со стажем;")
111        print("help - отобразить справку;")
112        print("exit - завершить работу с программой.")
113    else:
114        print(f"Неизвестная команда {command}", file=sys.stderr)
115
116 if __name__ == '__main__':
    select_workers()

```

Рисунок 5 – пример 1

```

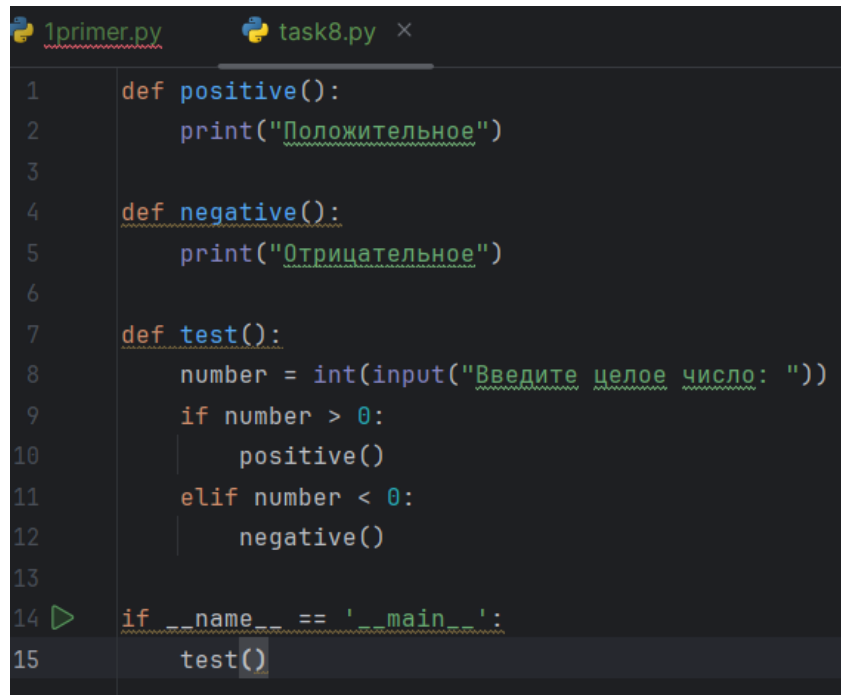
C:\Users\aregd\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\aregd\OneDrive\Рабочий стол\g11\lab11\1primer.py"
>>> add
Фамилия и инициалы? Dzharayan A.A.
Должность? Главный
Год поступления? 1989
>>> add
Фамилия и инициалы? Petrov G.G.
Должность? не главный
Год поступления? 2000
>>> select 6
+-----+-----+-----+-----+
| № |      Ф.И.О.      |      Должность      |      Год      |
+-----+-----+-----+-----+
| 1 | Dzharayan A.A.   |      Главный      |      1989     |
| 2 | Petrov G.G.      | не главный        |      2000     |
+-----+-----+-----+-----+
Список работников пуст.
>>> |

```

Рисунок 6 – пример выполнения примера 1

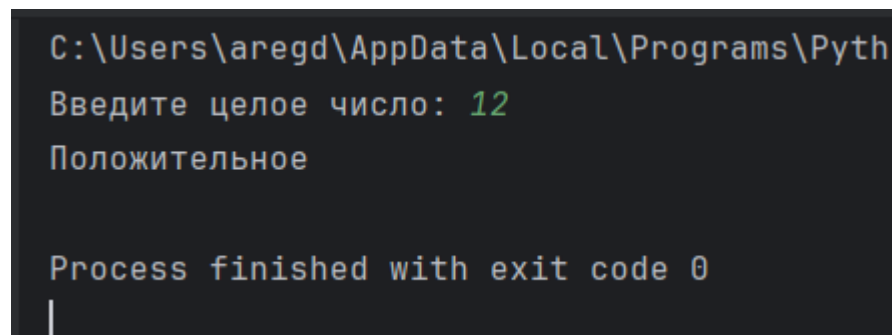
7. Основная ветка программы, не считая заголовков функций, состоит из двух строки кода. Это вызов функции `test()` и инструкции `if __name__ == '__main__':`. В ней запрашивается на ввод целое число. Если оно положительное, то вызывается функция `positive()`, тело которой содержит команду вывода на

экран слова "Положительное". Если число отрицательное, то вызывается функция `negative()`, ее тело содержит выражение вывода на экран слова "Отрицательное".



```
1 def positive():
2     print("Положительное")
3
4 def negative():
5     print("Отрицательное")
6
7 def test():
8     number = int(input("Введите целое число: "))
9     if number > 0:
10         positive()
11     elif number < 0:
12         negative()
13
14 if __name__ == '__main__':
15     test()
```

Рисунок 7 – выполнения 9 задания



```
C:\Users\aregd\AppData\Local\Programs\Python
Введите целое число: 12
Положительное

Process finished with exit code 0
```

Рисунок 8 – пример выполнения 9 задания

8. 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()`.

```
1primer.py task8.py task10.py x
1 import math
2
3 def cylinder():
4     def circle(r):
5         return math.pi * r**2
6
7     r = float(input("Введите радиус цилиндра: "))
8     h = float(input("Введите высоту цилиндра: "))
9     side_area = 2 * math.pi * r * h
10    full_area = side_area + 2 * circle(r)
11
12    choice = input("Хотите получить только площадь боковой поверхности цилиндра? (да/нет): ")
13    if choice.lower() == "да":
14        print(f"Площадь боковой поверхности цилиндра: {side_area}")
15    else:
16        print(f"Полная площадь цилиндра: {full_area}")
17
18 if __name__ == '__main__':
19     cylinder()
```

Рисунок 9 – выполнение задания 10

```
Введите радиус цилиндра: 12
Введите высоту цилиндра: 36
Хотите получить только площадь боковой поверхности цилиндра? (да/нет): да
Площадь боковой поверхности цилиндра: 2714.336052701581
```

Рисунок 10 – результат выполнения задания 10

9. 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.

```
1primer.py task8.py task10.py task12.py x
1 def proverka():
2     n = int(input())
3     s = n
4     while n != 0:
5         n = int(input())
6         s += n
7     print(s)
8
9 if __name__ == '__main__':
10     proverka()
```

Рисунок 11 – выполнение 12 задания

```
C:\Users\aregd\AppData\Local\Programs\Python\Pyt
2
3
5
7
8
34
8
9
0
76

Process finished with exit code 0
```

Рисунок 12 – результат выполнения 12 задания

10. Напишите программу, в которой определены четыре Функции.

```
1primer.py task8.py task10.py task12.py 14task.py
1 def get_input():
2     n = input("Введите значение: ")
3     return n
4 def test_input(value):
5     try:
6         int(value)
7         return True
8     except ValueError:
9         return False
10 def str_to_int(n):
11     s = int(n)
12     return s
13 def print_int(s):
14     print(s)
15 if __name__ == '__main__':
16     n = get_input()
17     if test_input(n):
18         s = str_to_int(n)
19         print_int(s)
20     else:
21         print("Введенное значение не является числом!!!")
```

Рисунок 13 – выполнение задания 14

```
C:\Users\aregd\AppData\Local\Programs\Pyth
Введите значение: 4578
4578

Process finished with exit code 0
```

Рисунок 14 – результат выполнения задания 14

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

```
1primer.py task8.py task10.py task12.py 14task.py individ.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import sys
4  def help1():
5      """
6      Функция для вывода списка команд
7      """
8      # Вывести справку о работе с программой.
9      print("Список команд:\n")
10     print("add - добавить рейс;")
11     print("list - вывести список рейсов;")
12     print("select <тип> - вывод на экран пунктов назначения и номеров рейсов для данного типа самолёта")
13     print("help - отобразить справку;")
14     print("exit - завершить работу с программой.")
15
16 def add1():
17     """
18     Функция для добавления информации о новых рейсах
19     """
20     # Запросить данные о работнике.
21     name = input("Название пункта назначения рейса? ")
22     number = int(input("Номер рейса? "))
23     tip = input("Тип самолета? ")
24     # Создать словарь.
25     i = {
26         'name': name,
27         'number': number,
28         'tip': tip,
29     }
30     return i
31 def error1():
32     """
33     Функция для неопознанных команд
34     """
35     print(f"Неизвестная команда {command}")
36
37 def list1(aircrafts):
38     """
39     Функция для вывода списка добавленных рейсов
40     """
41     # Заголовок таблицы.
42     line = '+-{}-+-{}-+-{}-+-{}-+'.format(
43         *args: '-' * 4,
44         '-' * 30,
45         '-' * 20,
46         '-' * 8
47     )
48     print(line)
49     # Вывести данные о всех сотрудниках.
50     for idx, i in enumerate(aircrafts, 1):
51         print(
52             '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
53                 *args: idx,
54                 i.get('name', ''),
55                 i.get('number', ''),
56                 i.get('tip', '')
57             )
58         )
59     print(line)
```



```

61 def select(command, aircrafts):
62     """
63     Функция для получения номера рейса и пункта назначения по заданному типу самолёта.
64     """
65     # Разбить команду на части для выделения номера года.
66     parts = command.split(' ', maxsplit=1)
67     # Проверить сведения работников из списка.
68     count = 0
69     for i in aircrafts:
70         for k, v in i.items():
71             if v == parts[1]:
72                 print("Пункт назначения - ", i["name"])
73                 print("Номер рейса - ", i["number"])
74                 count += 1
75     # Если счетчик равен 0, то работники не найдены.
76
77     if count == 0:
78         print("Рейс с заданным типом самолёта не найден.")
79
80
81 def main():
82     """
83     Главная функция программы.
84     """
85     print("Список команд:\n")
86     print("add - добавить рейс;")
87     print("list - вывести список рейсов;")
88     print("select <тип> - вывод на экран пунктов назначения и номеров рейсов для данного типа самолёта")
89     print("help - отобразить справку;")
90     print("exit - завершить работу с программой.")
91     # Список работников.
92     aircrafts = []
93     # Организовать бесконечный цикл запроса команд.
94     while True:
95         # Запросить команду из терминала.
96         command = input(">>> ").lower()
97         # Выполнить действие в соответствие с командой.
98
99         if command == 'exit':
100             break
101
102         elif command == 'add':
103             # Добавить словарь в список.
104             i = add1()
105             aircrafts.append(i)
106             # Отсортировать список в случае необходимости.
107             if len(aircrafts) > 1:
108                 aircrafts.sort(key=lambda item: item.get('name', ''))
109
110         elif command == 'list':
111             list(aircrafts)
112
113         elif command.startswith('select '):
114             select(command, aircrafts)
115
116         elif command == 'help':
117             help1()
118
119         else:
120             error1()
121
122
123 if __name__ == '__main__':
124     main()

```

Рисунок 15 – выполнение индивидуального задания

```

Список команд:

add - добавить рейс;
list - вывести список рейсов;
select <тип> - вывод на экран пунктов назначения и номеров рейсов для данного типа самолёта
help - отобразить справку;
exit - завершить работу с программой.

>>> add
Название пункта назначения рейса? madrid
Номер рейса? 12
Тип самолета? boeing
>>> add
Название пункта назначения рейса? paris
Номер рейса? 34
Тип самолета? airbus
>>> add
Название пункта назначения рейса? Athens
Номер рейса? 234
Тип самолета? embraer
>>> list
+-----+-----+-----+
| 1 | Athens | 234 | embraer |
| 2 | madrid | 12 | boeing |
| 3 | paris | 34 | airbus |
+-----+-----+-----+
>>> select boeing
Пункт назначения - madrid
Номер рейса - 12
>>> |

```

Рисунок 16 – результат выполнения индивидуального задания

12.Зафиксировал все изменения в github в ветке develop.

```

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gill/lab11 (develop)
$ git add .

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gill/lab11 (develop)
$ git commit -m"f"
[develop 7c64c5c] f
 6 files changed, 304 insertions(+)
 create mode 100644 PyCharm/14task.py
 create mode 100644 PyCharm/1primer.py
 create mode 100644 PyCharm/individ.py
 create mode 100644 PyCharm/task10.py
 create mode 100644 PyCharm/task12.py
 create mode 100644 PyCharm/task8.py

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gill/lab11 (develop)
$ git push origin develop
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads

```

Рисунок 17 – фиксация изменений в ветку develop

13.Слил ветки.

```
aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1/lab11 (develop)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

aregd@DESKTOP-5KV9QA9 MINGW64 ~/OneDrive/Рабочий стол/gil1/lab11 (main)
$ git merge develop
Updating e8afd11..7c64c5c
Fast-forward
 PyCharm/14task.py | 21 ++++++
 PyCharm/1primer.py | 116 ++++++
 PyCharm/individ.py | 124 ++++++
 PyCharm/task10.py | 19 ++++++
 PyCharm/task12.py | 10 +++++
 PyCharm/task8.py | 14 +++++
 6 files changed, 304 insertions(+)
 create mode 100644 PyCharm/14task.py
 create mode 100644 PyCharm/1primer.py
 create mode 100644 PyCharm/individ.py
 create mode 100644 PyCharm/task10.py
 create mode 100644 PyCharm/task12.py
 create mode 100644 PyCharm/task8.py
```

Рисунок 18 – сливание ветки develop в ветку main

Вывод: приобрел навыки по работе с функциями при написании программ с помощью языка программирования Python версии 3.x.

Контрольные вопросы:

1. Каково назначение функций в языке программирования Python?

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. Внедрение функций позволяет решить проблему дублирования кода в разных местах программы. Благодаря им можно исполнять один и тот же участок кода не сразу, а только тогда, когда он понадобится.

2. Каково назначение операторов def и return?

Оператор def в Python используется для определения функции. Он начинает заголовок функции и может принимать ноль или более аргументов, которые могут использоваться в теле функции. Оператор return используется для возврата результата выполнения функции. Он может быть необязательным, так как функция может ничего не возвращать. Оператор return не только возвращает значение, но и производит выход из функции. Поэтому он должен определяться после остальных инструкций. Если функция не возвращает никакого значения, после оператора return не ставится никакого возвращаемого значения.

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

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?

В Питоне позволительно возвращать из функции несколько объектов, перечислив их через запятую после команды return.

5. Какие существуют способы передачи значений в функцию?

```
figure4 = cylinder(r=2, h=10)
```

```
def cylinder(h, r=1):
```

```
figure4 = cylinder(i, h)(передаются значения глобальных переменных)
```

6. Как задать значение аргументов функции по умолчанию?

```
def cylinder(h, r=1):
```

7. Каково назначение lambda-выражений в языке Python?

Lambda-выражения в Python, также известные как “анонимные функции”, используются для создания небольших функций без необходимости использования ключевого слова def. Они представляют собой компактный способ определения функции.

8. Как осуществляется документирование кода согласно PEP257?

Все модули должны, как правило, иметь строки документации, и все функции и классы, экспортируемые модулем также должны иметь строки документации. Публичные методы (в том числе `__init__`) также должны иметь строки документации. Пакет модулей может быть документирован в `__init__.py`. Для согласованности, всегда используйте `"""triple double quotes"""` для строк документации. Многострочные строки документации состоят из однострочной строки документации с последующей пустой строкой, а затем более подробным описанием. Первая строка может быть использована автоматическими средствами индексации, поэтому важно, чтобы она находилась на одной строке и была отделена от остальной документации пустой строкой. Первая строка может быть на той же строке, где и открывающие кавычки, или на следующей строке. Вся документация должна иметь такой же отступ, как кавычки на первой строке

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

Однострочные строки документации используются для краткого описания функции, метода, класса или модуля, что делает и какие аргументы принимает. Они заключаются в тройные кавычки и пишутся в императивной форме. Многострочные строки документации используются для более подробного

описания функции, метода, класса или модуля, включая их параметры, типы, возвращаемые значения, исключения, примеры и другие детали. Они также заключаются в тройные кавычки, но имеют определенный формат и стиль, в зависимости от выбранной конвенции.