

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 №2.11**  
**дисциплины «Основы программной инженерии»**

Выполнил:  
Соколов Михаил Романович  
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:  
Богданов С.С., ассистент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

Тема: Замыкания в языке Python.

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

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и .gitignore файл для языка 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 (\*).*

**Repository template**

No template ▾

Start your repository with a template repository's contents.

**Owner \*** **Repository name \***

PoTtaTto ▾ / fundamentals\_of\_software

✔ fundamentals\_of\_software\_engineering\_lab2\_11 is available.

Great repository names are short and memorable. Need inspiration? How about [urban-waffle](#) ?

**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](#).

① You are creating a public repository in your personal account.

**Create repository**

Рисунок 1 – Создание репозитория с заданными настройками

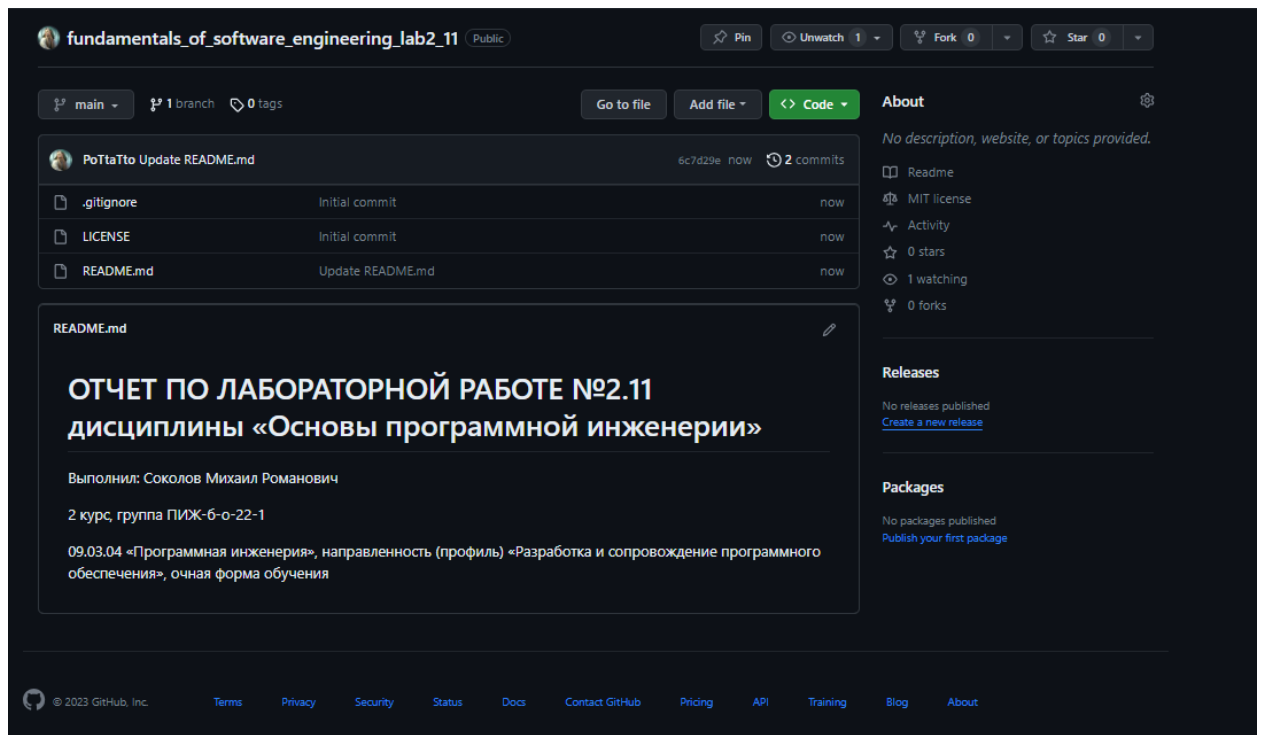


Рисунок 2 – Созданный репозиторий

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11> git clone https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_11
Cloning into 'fundamentals_of_software_engineering_lab2_11'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11> |
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11> |
```

Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза

```
.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/
```

Рисунок 5 – Часть .gitignore файла, созданного GitHub

2. Проработаем примеры лабораторной работы, фиксируя изменения.  
Создадим для каждого примера отдельный модуль языка Python:

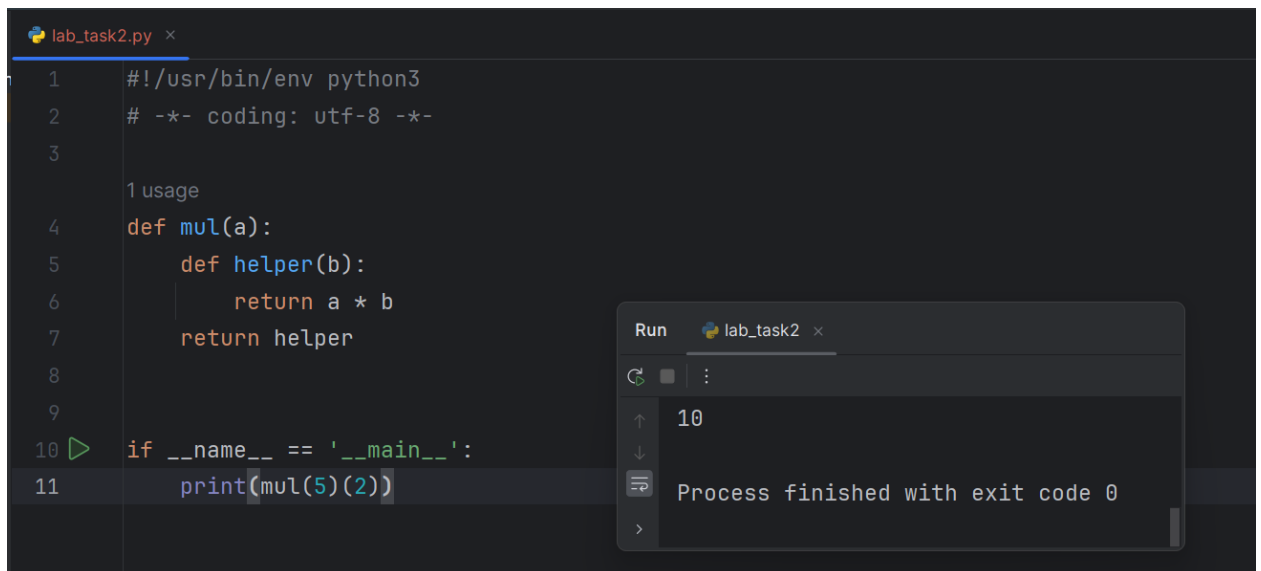
```
lab_task1.py x
2 # -*- coding: utf-8 -*-
3
4 1 usage
5 def fun1(a):
6     x = a * 3
7
8     def fun2(b):
9         nonlocal x
10        return b + x
11
12    return fun2
13
14 if __name__ == '__main__':
15     test_fun = fun1(4)
16     value = test_fun(7)
17     print(value)
```

Run lab\_task1 x

19

Process finished with exit code 0

Рисунок 6 – Пример кода замыкания и его выполнение (1)



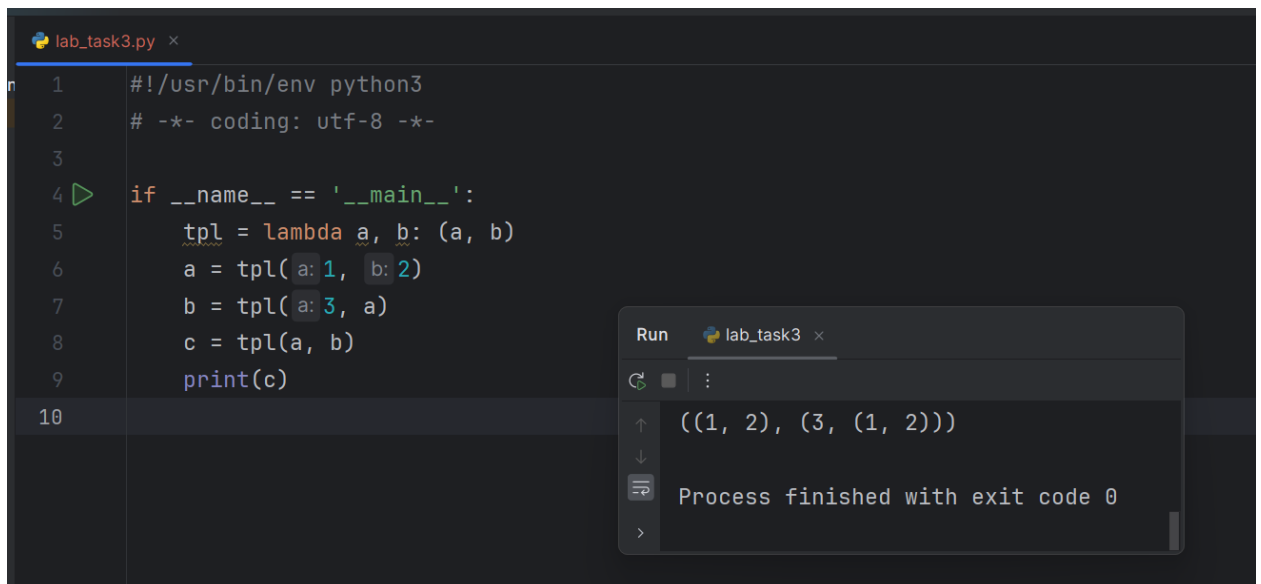
```
lab_task2.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  usage
5  def mul(a):
6      def helper(b):
7          return a * b
8      return helper
9
10 if __name__ == '__main__':
11     print(mul(5)(2))
```

Run lab\_task2 x

10

Process finished with exit code 0

Рисунок 7 – Пример кода замыкания и его выполнение (2)



```
lab_task3.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == '__main__':
5      tpl = lambda a, b: (a, b)
6      a = tpl(a: 1, b: 2)
7      b = tpl(a: 3, a)
8      c = tpl(a, b)
9      print(c)
10
```

Run lab\_task3 x

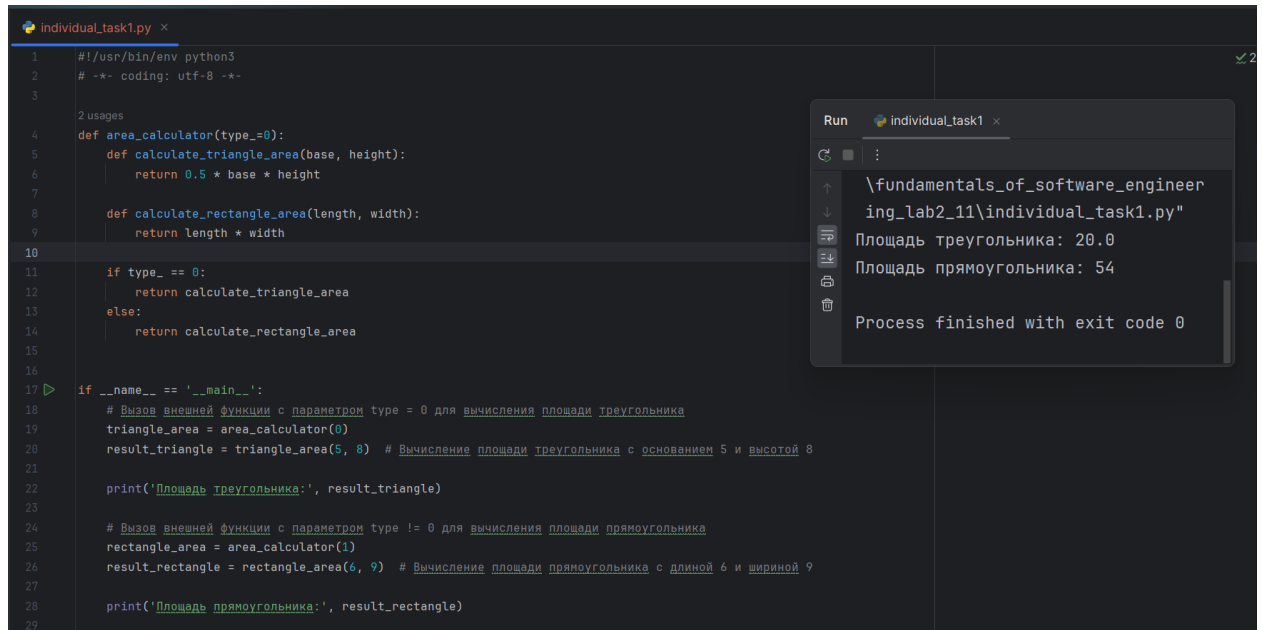
((1, 2), (3, (1, 2)))

Process finished with exit code 0

Рисунок 8 – Пример кода со свойством замыкания и его выполнение (2)

3. Решим индивидуальное задание (вариант №7). . Используя замыкания функций, объявите внутреннюю функцию, которая на основе двух параметров вычисляет площадь фигуры. Какой именно фигуры: треугольника или прямоугольника, определяется параметром `type` внешней функции. Если `type` принимает значение 0, то вычисляется площадь треугольника, а иначе – прямоугольника. По умолчанию параметр `type` должен быть равен 0.

Вычисленное значение должно возвращаться внутренней функцией. Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы:

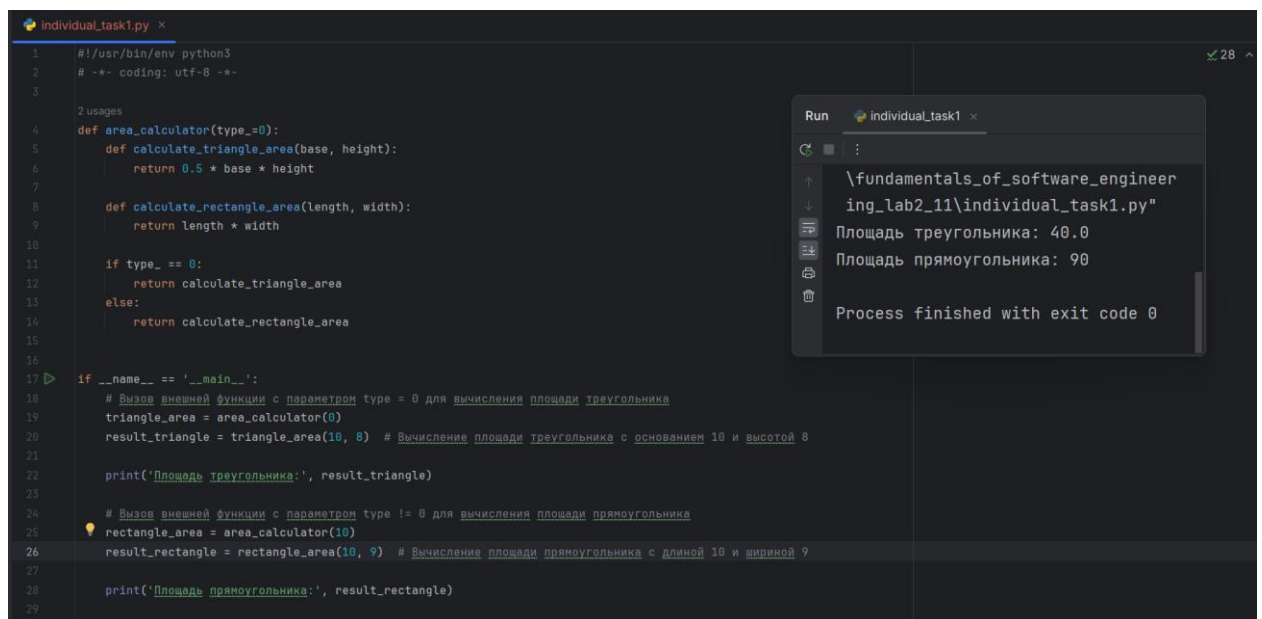


```
individual_task1.py
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 2 usages
5 def area_calculator(type=0):
6     def calculate_triangle_area(base, height):
7         return 0.5 * base * height
8
9     def calculate_rectangle_area(length, width):
10        return length * width
11
12    if type == 0:
13        return calculate_triangle_area
14    else:
15        return calculate_rectangle_area
16
17 if __name__ == '__main__':
18     # Вызов внешней функции с параметром type = 0 для вычисления площади треугольника
19     triangle_area = area_calculator(0)
20     result_triangle = triangle_area(5, 8) # Вычисление площади треугольника с основанием 5 и высотой 8
21
22     print('Площадь треугольника:', result_triangle)
23
24     # Вызов внешней функции с параметром type != 0 для вычисления площади прямоугольника
25     rectangle_area = area_calculator(1)
26     result_rectangle = rectangle_area(6, 9) # Вычисление площади прямоугольника с длиной 6 и шириной 9
27
28     print('Площадь прямоугольника:', result_rectangle)
29
```

Run individual\_task1

```
\fundamentals_of_software_engineering_lab2_11\individual_task1.py"
Площадь треугольника: 20.0
Площадь прямоугольника: 54
Process finished with exit code 0
```

Рисунок 9 – Код задания и его выполнение (1)



```
individual_task1.py
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 2 usages
5 def area_calculator(type=0):
6     def calculate_triangle_area(base, height):
7         return 0.5 * base * height
8
9     def calculate_rectangle_area(length, width):
10        return length * width
11
12    if type == 0:
13        return calculate_triangle_area
14    else:
15        return calculate_rectangle_area
16
17 if __name__ == '__main__':
18     # Вызов внешней функции с параметром type = 0 для вычисления площади треугольника
19     triangle_area = area_calculator(0)
20     result_triangle = triangle_area(10, 8) # Вычисление площади треугольника с основанием 10 и высотой 8
21
22     print('Площадь треугольника:', result_triangle)
23
24     # Вызов внешней функции с параметром type != 0 для вычисления площади прямоугольника
25     rectangle_area = area_calculator(10)
26     result_rectangle = rectangle_area(10, 9) # Вычисление площади прямоугольника с длиной 10 и шириной 9
27
28     print('Площадь прямоугольника:', result_rectangle)
29
```

Run individual\_task1

```
\fundamentals_of_software_engineering_lab2_11\individual_task1.py"
Площадь треугольника: 40.0
Площадь прямоугольника: 90
Process finished with exit code 0
```

Рисунок 10 – Код задания и его выполнение (2)

4. Выполним merge веток main/develop и отправим изменения на удаленный репозиторий:

```

17e7f75 (HEAD -> develop) individual_task1.py is added
3a9ac2c lab_task3.py is added
446a517 lab_task2.py is added
85c0033 lab_task1.py is added
6c7d29e (origin/main, origin/HEAD, main) Update README.md
11d7482 Initial commit
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11\fundamentals_of_software_engineering_lab2_11>

```

Рисунок 11 – Коммиты проекта

```

Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11\fundamentals_of_software_engineering_lab2_11> git merge develop
Updating 6c7d29e..17e7f75
Fast-forward
 .gitignore      | 2 +-
 individual_task1.py | 28 ++++++
 lab_task1.py     | 17 ++++++
 lab_task2.py     | 11 ++++++
 lab_task3.py     | 9 ++++++
 5 files changed, 66 insertions(+), 1 deletion(-)
 create mode 100644 individual_task1.py
 create mode 100644 lab_task1.py
 create mode 100644 lab_task2.py
 create mode 100644 lab_task3.py
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11\fundamentals_of_software_engineering_lab2_11>

```

Рисунок 12 – Merge веток main/develop

```

PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11\fundamentals_of_software_engineering_lab2_11> git push origin main
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 12 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 1.88 KiB | 1.88 MiB/s, done.
Total 13 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/PoItaTo/fundamentals_of_software_engineering_lab2_11
 6c7d29e..17e7f75 main -> main
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.11\fundamentals_of_software_engineering_lab2_11>

```

Рисунок 13 – Отправка изменений на удаленный репозиторий

Ответы на контрольные вопросы:

1. Что такое замыкание?

Замыкание — это функция, которая сохраняет ссылку на некоторую переменную в окружающем контексте, где она была определена, и использует эту переменную при последующих вызовах.

2. Как реализованы замыкания в языке программирования Python?

В Python замыкания реализуются путем создания функции внутри другой функции. Внутренняя функция имеет доступ к переменным из внешней функции, сохраняя ссылку на них после завершения работы внешней функции.

3. Что подразумевает под собой область видимости Local?

Область видимости Local в Python относится к переменным, определенным внутри функции, и доступным только в пределах этой функции.

4. Что подразумевает под собой область видимости Enclosing?

Область видимости Enclosing (вложенная) в Python связана с переменными, определенными во внешней функции, и доступными внутри внутренней функции.

5. Что подразумевает под собой область видимости Global?

Область видимости Global в Python относится к переменным, определенным на верхнем уровне модуля, и доступным в любой части этого модуля.

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

Область видимости Built-in в Python связана со встроенными в язык функциями и объектами, доступными в любой части программы.

7. Как использовать замыкания в языке программирования Python?

В Python замыкания можно использовать, определяя внутренние функции внутри внешних функций и сохраняя состояние окружающего контекста.

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

Замыкания позволяют создавать функции, которые сохраняют состояние своего окружения, что полезно для создания иерархических структур данных,



таких как деревья или вложенные списки. Например, каждый вызов функции может создавать новый уровень вложенности данных.