

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.9
дисциплины
«Программирование на Python»

Выполнил:
Костукайло Кирилл Николаевич
2 курс, группа ИВТ-б-о-21-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Кафедры инфокоммуникаций, старший
преподаватель
Воронкин Р.А.

(подпись)

Отчёт защищён с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

Тема: Рекурсия в языке Python

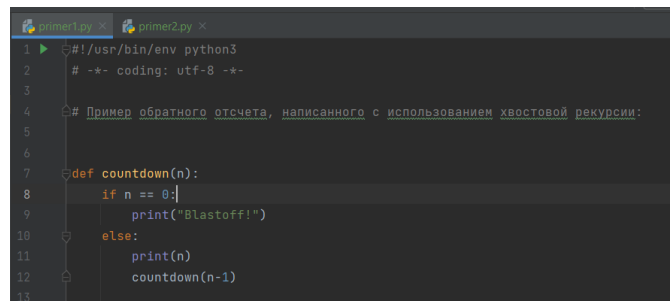
Цель работы: приобретение навыков по работе с рекурсивными функциями при написании

программ с помощью языка программирования Python версии 3.x.

Ход работы:

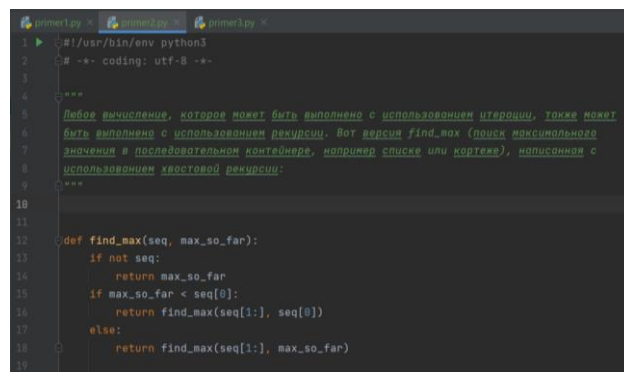
1. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python, клонировал созданного репозитория.

2. Создайте проект PyCharm в папке репозитория и проработал примеры лабораторной работы.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 # Пример обратного отсчета, написанного с использованием хвостовой рекурсии:
5
6
7 def countdown(n):
8     if n == 0:
9         print("Blastoff!")
10    else:
11        print(n)
12        countdown(n-1)
13
```

Рисунок 1.1. Пример 1



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """
5 Поиск вычисления, которое может быть выполнено с использованием итерации, также может
6 быть выполнено с использованием рекурсии. Вот версия find_max (поиск максимального
7 значения в последовательном контейнере, например списке или кортеже), написанная с
8 использованием хвостовой рекурсии:
9 """
10
11
12 def find_max(seq, max_so_far):
13     if not seq:
14         return max_so_far
15     if max_so_far < seq[0]:
16         return find_max(seq[1:], seq[0])
17     else:
18         return find_max(seq[1:], max_so_far)
19
```

Рисунок 1.2. Пример 2

```

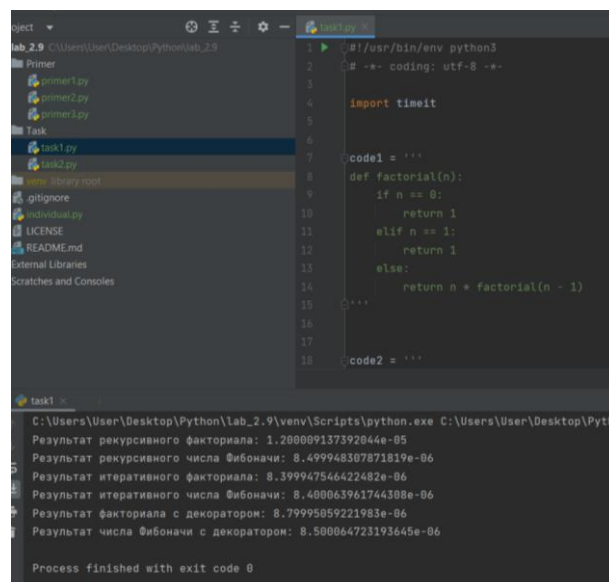
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 # Эта программа показывает работу декоратора, который производит оптимизацию
5 # хвостового вызова. Он делает это, вызывая исключение, если оно является его
6 # прародителем, и перехватывает исключение, чтобы вызвать стек.
7
8 import sys
9
10
11 class TailRecurseException(Exception):
12     def __init__(self, args, kwargs):
13         self.args = args
14         self.kwargs = kwargs
15
16
17 def tail_call_optimized(g):
18     """
19     Эта функция не работает, если функция декоратора не использует хвостовой вызов.
20     """
21
22     def func(*args, **kwargs):
23         f = sys._getframe()
24         if f.f_back and f.f_back.f_back and f.f_back.f_back.f_code == f.f_code:
25             raise TailRecurseException(args, kwargs)
26         else:
27             while True:
28                 try:
29                     return g(*args, **kwargs)
30                 except TailRecurseException as e:
31                     args = e.args
32                     kwargs = e.kwargs
33
34 if __name__ == '__main__':

```

Рисунок 1.3. Пример 3

3. Выполнил задачи:

Задача 1: самостоятельно изучите работу со стандартным пакетом Python `timeit`. Оцените с помощью этого модуля скорость работы итеративной и рекурсивной версий функций `factorial` и `fib`. Во сколько раз измениться скорость работы рекурсивных версий функций `factorial` и `fib` при использовании декоратора `lru_cache`? Приведите в отчет и обоснуйте полученные результаты.



```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import timeit
5
6
7 code1 = '''
8 def factorial(n):
9     if n == 0:
10         return 1
11     elif n == 1:
12         return 1
13     else:
14         return n * factorial(n - 1)
15 '''
16
17 code2 = '''

```

task1

```

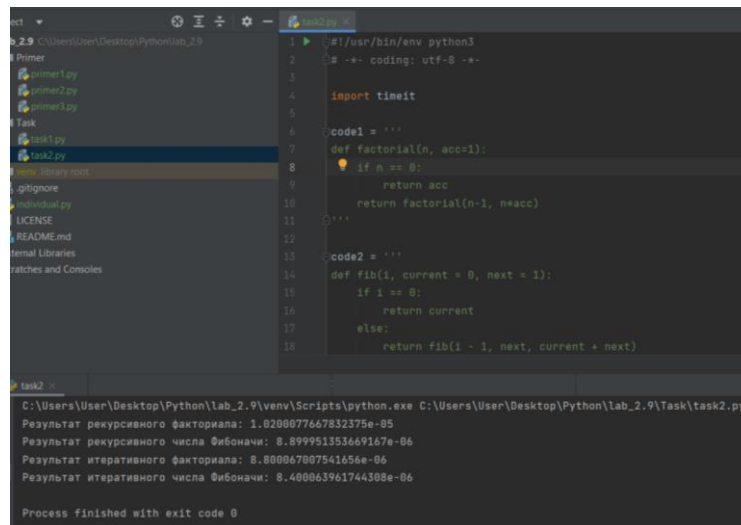
C:\Users\User\Desktop\Python\lab_2.9\env\Scripts\python.exe C:\Users\User\Desktop\Python\lab_2.9\env\Scripts\python.exe
Результат рекурсивного факториала: 1.208009137392044e-05
Результат рекурсивного числа Фибоначчи: 8.499948307871819e-06
Результат итеративного факториала: 8.399947546422482e-06
Результат итеративного числа Фибоначчи: 8.400063961744308e-06
Результат факториала с декоратором: 8.79995059221983e-06
Результат числа Фибоначчи с декоратором: 8.500064723193645e-06
Process finished with exit code 0

```

Рисунок 2. Задача 1

Задача 2: самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета `timeit` оцените скорость работы функций `factorial` и `fib` с использованием интроспекции стека и без

использования интроспекции стека. Приведите полученные результаты в отчет.



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import timeit

code1 = '''
def factorial(n, acc=1):
    if n == 0:
        return acc
    return factorial(n-1, n*acc)
'''

code2 = '''
def fib(i, current = 0, next = 1):
    if i == 0:
        return current
    else:
        return fib(i - 1, next, current + next)
'''

if __name__ == '__main__':
    print('Результат рекурсивного факториала: ', timeit.timeit(code1, number=1000000))
    print('Результат рекурсивного числа Фибоначчи: ', timeit.timeit(code2, number=1000000))
    print('Результат итеративного факториала: ', timeit.timeit(code1, number=1000000))
    print('Результат итеративного числа Фибоначчи: ', timeit.timeit(code2, number=1000000))
```

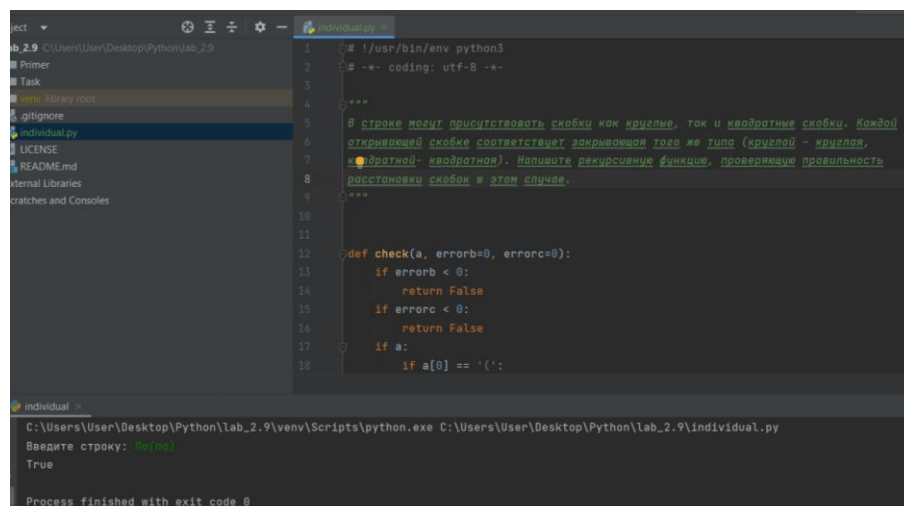
Process finished with exit code 0

Рисунок 3. Задача 2

4. Выполнил индивидуальную задачу:

Индивидуальное задача. В строке могут присутствовать скобки как круглые, так и квадратные скобки. Каждой открывающей скобке соответствует закрывающая того же типа (круглой – круглая, квадратной – квадратная). Напишите рекурсивную функцию, проверяющую правильность расстановки скобок в этом случае.

Пример неправильной расстановки: ([)].



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def check(a, errorb=0, errorc=0):
    if errorb < 0:
        return False
    if errorc < 0:
        return False
    if a:
        if a[0] == '(':
            errorb += 1
        elif a[0] == '[':
            errorb += 1
        elif a[0] == ')':
            errorb -= 1
        elif a[0] == ']':
            errorb -= 1
        return check(a[1:], errorb, errorc)
    return errorb == 0 and errorc == 0

if __name__ == '__main__':
    a = '( [ ) ]'
    print(check(a))
```

Process finished with exit code 0

Рисунок 6.2. Индивидуальное задание

Контрольные вопросы:

1. Для чего нужна рекурсия?

В программировании рекурсия — вызов функции (процедуры) из неё же

самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия). Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

2. Что называется базой рекурсии?

База рекурсии – это такие аргументы функции, которые делают задачу настолько простой, что решение не требует дальнейших вложенных вызовов.

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?

Стек — это структура данных, в которой элементы хранятся в порядке поступления.

Стек хранит последовательность данных. Связаны данные так: каждый элемент указывает на тот, который нужно использовать следующим. Это линейная связь — данные идут друг за другом и нужно брать их по очереди.

Из середины стека брать нельзя.

Главный принцип работы стека — данные, которые попали в стек недавно, используются первыми. Чем раньше попал — тем позже используется. После использования элемент стека исчезает, и верхним становится следующий элемент.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python?

Функция `sys.getrecursionlimit()` возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python. Этот предел предотвращает бесконечную рекурсию от переполнения стека языка C и сбоя Python. Это значение может быть установлено с помощью `sys`.

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python?

Существует предел глубины возможной рекурсии, который зависит от реализации Python. Когда предел достигнут, возникает исключение `RuntimeError`.

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью `sys.setrecursionlimit(число)`.

7. Каково назначение декоратора `lru_cache`?

Функция `lru_cache` предназначена для мемоизации (предотвращения повторных вычислений), т. е. кэширует результат в памяти.

Полезный инструмент, который уменьшает количество лишних вычислений.

8. Что такое хвостовая рекурсия? Как проводится оптимизация хвостовых вызовов?

Хвостовая рекурсия — частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции. Подобный вид рекурсии примечателен тем, что может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции. **Оптимизация хвостовой рекурсии путём преобразования её в плоскую итерацию** реализована во многих оптимизирующих компиляторах. В некоторых функциональных языках программирования спецификация гарантирует обязательную оптимизацию хвостовой рекурсии.

Типовой механизм реализации вызова функции основан на сохранении адреса возврата, параметров и локальных переменных функции в стеке и выглядит следующим образом:

1. В точке вызова в стек помещаются параметры, передаваемые функции, и адрес возврата.
2. Вызываемая функция в ходе работы размещает в стеке собственные локальные переменные.
3. По завершении вычислений функция очищает стек от своих локальных переменных, записывает результат (обычно — в один из регистров процессора).

4. Команда возврата из функции считывает из стека адрес возврата и выполняет переход по этому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров.