Лабораторная работа № 3: Кортежи, множества, словари, функции, работа с файлами

1. Создайте файл `lab_03_01.py`:

```
'''
    Кортежи
'''
# создание кортежа
a1 = tuple()
a2 = 1, 2, 3, "abc"
a3 = (1, 2, 3, "abc")
print("Tuple a1 = ", a1)
print("Tuple a2 = ", a2)
print("Tuple a3 = ", a3)

# создание кортежа из других структур данных
l = [1, 2, 3, "abc"] # из списка
a4 = tuple(l)
print("Tuple a4 from list l = ", a4)
a5 = tuple("Hello, World!") # из строки
print("Tuple a5 from string = ", a5)

# вложенность кортежей
a6 = a2, a3
print("Tuple a6 formed by a2 and a3 = ", a6)

# объединение кортежей
a7 = a2 + a3
print("Tuple a7 by combining a2 and a3 = ", a7)

# доступ к элементам кортежей
print("a6[0]: ", a6[0])
print("a6[0][3]: ", a6[0][3])
# a6[0][3] = "cba"
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Tuple a1 =  ()
Tuple a2 =  (1, 2, 3, 'abc')
Tuple a3 =  (1, 2, 3, 'abc')
Tuple a4 from list l =  (1, 2, 3, 'abc')
Tuple a5 from string =  ('H', 'e', 'l', 'l', 'o', ',', ' ', 'W',
'o', 'r', 'l', 'd', '!')
Tuple a6 formed by a2 and a3 =  ((1, 2, 3, 'abc'), (1, 2, 3,
'abc'))
Tuple a7 by combining a2 and a3 =  (1, 2, 3, 'abc', 1, 2, 3, 'abc')
a6[0]:  (1, 2, 3, 'abc')
a6[0][3]:  abc
```

Кортежи

В языке программирования Python кортеж (Tuple) представляет собой неизменяемый список. Для него доступны все операции, что и для списков, за исключением влияющих на значения списка, например, добавление, изменение, удаление элементов и т.д. Кортежи записываются в круглых скобках и могут быть объявлены как:

```
a1 = tuple()
a2 = 1, 2, 3, "abc"
a3 = (1, 2, 3, "abc")
```

Если кортеж состоит из одного элемента, при объявлении кортежа без использования функции `tuple()` необходимо ставить запятую после значения элемента, например: `a=(5,)`. Функция `tuple()` также позволяет создать кортеж из существующего списка или строки.

Для объединения кортежей используется операция сложения: `a = b + c`. Для вложения кортежей в общий кортеж используется следующая форма записи: `a = b, c`. Доступ к элементам кортежа осуществляется так же, как и доступ к элементам списка.

2. Модифицируйте код программы `lab_03_01.py`. Раскомментируйте строку '`# a6[0][3] = "cba"`'. Объясните поведение программы.

3. Дополните код программы `lab_03_01.py`. Создайте кортеж `k1`, содержащий значения дня, месяца и года Вашего рождения, введенные с клавиатуры, и кортеж `k2` со значениями Ваших фамилии, имени и отчества. Объедините кортежи `k1` и `k2`, записав результат в `k3`. Осуществите вывод значения переменной `k3` на экран. Ознакомьтесь с результатом.

4. Дополните код программы `lab_03_01.py`. Создайте кортеж `k4`, в который будут вложены кортежи `k1` и `k2`. Осуществите вывод значения переменной `k4`, а также второго элемента второго вложенного кортежа на экран. Ознакомьтесь с результатом.

5. Создайте файл `lab_03_02.py`:

```
'''
    Множества
'''
# создание множества
b1 = set()
print("Set b1 = ", b1)
b2 = {"bear", "fox", "squirrel", "woodpecker", "woodpecker",
"wolf", "hedgehog"}
print("Set b2 = ", b2)
# создание множества из строки
b3 = set("abcdabcdefg")
print("Set b3 from string: ", set(b3))
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Set b1 =  set()
Set b2 =  {'fox', 'squirrel', 'bear', 'hedgehog', 'woodpecker',
'wolf'}
Set b3 from string:  {'g', 'a', 'b', 'c', 'd', 'f', 'e'}
```

Множества

Множество (Set) – это контейнер уникальных значений. Значения множества указываются в фигурных скобках. Множество создается с использованием функции `set()` или с указанием набора значений в фигурных скобках. Функция `set()` позволяет создать пустое множество, а также множество из строки, списка или другой итерируемой структуры данных. Если в наборе значений присутствуют повторяющиеся, все повторы будут удалены. Пустое множество не может быть создано с указанием пустых фигурных скобок. Для множеств не предусмотрено обращение к отдельному элементу. Например, для проверки принадлежности элемента `element` множеству `set1` и проходу по множеству, можно использовать:

```
if element in set1:
    <операторы>
```
и
```
for element in set1:
    <операторы>
```

6. Дополните код программы `lab_03_02.py`. Создайте строковую переменную `s` со значением *"Electricity is the set of physical phenomena associated with the presence of electric charge. Lightning is one of the most dramatic effects of electricity"*. Создайте множество `set1` из строки `s`. Осуществите вывод множества `set1` на экран.

7. Дополните код программы `lab_03_02.py`. Для множества `set1` осуществите проход по всем его элементам с выводом на экран гласных букв. Ознакомьтесь с результатом.

8. Создайте файл `lab_03_03.py`:

```
'''
    Операции над множествами
'''
print("Check 'bear' in b2 = ", "bear" in b2)
b4 = set("123456135")
b5 = set("12367")
print("Set b4: {0}, \nSet b5: {1}".format(b4,b5))
print("b4 - b5: ", b4 - b5) # присутствие в первом множестве, но
не во втором
print("b4 difference b5 (b4-b5): ", b4.difference(b5))
print("b4 | b5: ", b4 | b5) # присутствие хотя бы в одном
множестве
```

```
print("b4 union b5 (b4 | b5): ", b4.union(b5))
print("b4 & b5: ", b4 & b5) # присутствие в обоих множествах
print("b4 intersection b5 (b4&b5): ", b4.intersection(b5))
print("b4 ^ b5: ", b4 ^ b5) # присутствие только в одном из
множеств
# проверка на непересечение множеств
print("b4 and b5 are disjoint: ", b4.isdisjoint(b5))

b4.update(b5) # добавить элементы другого множества
print("add b5 to b4: ", b4)
b4.add("abc") # добавить элемент
print("add 'abc' to b4: ", b4)
b4.remove("5") # удалить элемент
print("remove element '5' from b4: ", b4)
b4.clear() # очистить множество
print("clear b4: ", b4)
print("\n ")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Check 'bear' in b2 =  True
Set b4: {'4', '6', '5', '2', '1', '3'},
Set b5: {'3', '2', '7', '6', '1'}
b4 - b5:  {'4', '5'}
b4 difference b5 (b4-b5):  {'4', '5'}
b4 | b5:  {'4', '7', '6', '5', '2', '1', '3'}
b4 union b5 (b4 | b5):  {'4', '7', '6', '5', '2', '1', '3'}
b4 & b5:  {'3', '1', '6', '2'}
b4 intersection b5 (b4&b5):  {'3', '1', '6', '2'}
b4 ^ b5:  {'4', '7', '5'}
b4 and b5 are disjoint:  False
add b5 to b4:  {'4', '7', '6', '5', '2', '1', '3'}
add 'abc' to b4:  {'4', '7', '6', '5', 'abc', '2', '1', '3'}
remove element '5' from b4:  {'4', '7', '6', 'abc', '2', '1', '3'}
clear b4:  set()
```

Операции над множествами

В языке программирования Python существуют следующие операции, выполняемые над множествами:

- Проверка наличия элемента `element` в множестве `set1`: `element in set1`
- Разность множеств (включаются элементы, присутствующие в первом множестве, но не во втором):
  - `set3 = set1 - set2`
  - `set3 = set1.difference(set2)`
- Объединение множеств (присутствие элемента хотя бы в одном множестве):
  - `set3 = set1 | set2`
  - `set3 = set1.union(set2)`

- Пересечение множеств (присутствие элемента в обоих множествах):
  - `set3 = set1 & set2`
  - `set3 = set1.intersection(set2)`
- Симметричная разность (присутствие элементов только в одном из множеств):
  - `set3 = set1 ^ set2`
  - `set3 = set1.symmetric_difference(set2)`
- Проверка на непересечение множеств: `set1.isdisjoint(set2)`

Изменение значений множеств можно проводить с помощью следующих операций:

- Обновление множества добавлением элементов другого множества: `set1.update(set2)`
- Добавление элемента `element` в множество `set1`: `set1.add(element)`
- Удаление элемента `element` из множества `set1`: `set1.remove(element)`
- Очистка множества `set1`: `set1.clear()`

Также существуют неизменяемые множества – `frozenset`, для которых недоступны операции, связанные с изменением значений. Неизменяемое множество может быть также получено из изменяемого множества или любой другой итерируемой структуры данных путем использования функции `frozenset()`, аргументом которой является структура данных:

```
fset = frozenset({"1","2","3"})
```

Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном стандартным типам данных:

```
https://docs.python.org/3/library/
stdtypes.html#set-types-set-frozenset
```

9. Дополните код программы `lab_03_03.py`. Создайте два множества `set1` и `set2` из строк *"qetuwrt"* и *"asfrewgq"* соответственно. Поочередно выполните операции разности, объединения, пересечения и симметричной разности, выводя значения на экран. Добавьте в множество `set1` элементы множества `set2` с использованием функции `update()`, в множество `set2` элементы "t" и "u" с использованием функции `add()`. Повторно выполните операции разности, объединения, пересечения и симметричной разности, выводя значения на экран. Сравните полученные результаты.

10. Дополните код программы `lab_03_03.py`. Создайте неизменяемое множество `set3` из множества `set1`. Удалите из множества `set3` элемент "q". Запустите программу. Ознакомьтесь с результатом и объясните, почему так произошло.

11. Создайте файл `lab_03_04.py`:

```
'''
    Словари
'''
d1 = {
    "day": 18,
    "month": 6,
    "year": 1983
```

```
}
d2 = dict(bananas=3,apples=5,oranges=2,bag="basket")
d3 = dict([("street","Kronverksky pr."), ("house", 49)])
d4 = dict.fromkeys(["1","2"], 3)
print("Dict d1 = ", d1)
print("Dict d2 by dict()= ", d2)
print("Dict d3 by dict([])= ", d3)
print("Dict d4 by fromkeys = ", d4)
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Dict d1 =  {'month': 6, 'year': 1983, 'day': 18}
Dict d2 by dict()=  {'apples': 5, 'bag': 'basket', 'bananas': 3,
'oranges': 2}
Dict d3 by dict([])=  {'house': 49, 'street': 'Kronverksky pr.'}
Dict d4 by fromkeys =  {'2': 3, '1': 3}
```

Словари

Словари в языке программирования Python представляют собой неупорядоченные коллекции объектов, доступ к которым осуществляется по ключу. Альтернативные названия – ассоциативные массивы, хэш-таблицы.

Существует три способа создания словаря:

- Указание пар "ключ-значение" в фигурных скобках: `d = {"a": 1, "b": 2}`
- С помощью функции `dict()`:
  - `d = dict(a = 1, b = 2)`
  - `d = dict([("a",1), ("b",2)])`
- Заполнение одинаковыми значениями по ключам: `d = dict.fromkeys(["a","b"], 1)`

12. Дополните код программы `lab_03_04.py`. Создайте словарь `startDict` с ключами `ready`, `set`, `go` и значениями 3, 2 и 1 соответственно тремя разными способами, добавляя индекс к имени переменной. Выведите получившиеся словари на экран.

13. Дополните код программы `lab_03_04.py`. Создайте словарь `dict1` с ключами `key1` и `key2`, заполнив их одинаковым значением, введенным с клавиатуры. Осуществите вывод словаря `dict1` на экран.

14. Создайте файл `lab_03_05.py`:

```
'''
    Операции со словарями
'''
d5 = d2.copy() # создание копии словаря
print("Dict d5 copying d2 = ", d5)
```

```
# получение значения по ключу
print("Get dict value by key d5['bag']: ", d5["bag"])
print("Get dict value by key d5.get('bag'): ", d5.get('bag'))
print("Get dict keys d5.keys(): ", d5.keys()) # список ключей
print("Get dict values d5.values(): ", d5.values()) # список значений
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Dict d5 copying d2 =  {'bananas': 3, 'bag': 'basket', 'apples': 5, 'oranges': 2}
Get dict value by key d5['bag']:  basket
Get dict value by key d5.get('bag'):  basket
Get dict keys d5.keys():  dict_keys(['bananas', 'bag', 'apples', 'oranges'])
Get dict values d5.values():  dict_values([3, 'basket', 5, 2])
```

Операции со словарями

В языке программирования Python существуют следующие операции, выполняемые со словарями:

- Копирование словаря `d1`: `d2 = d1.copy()`
- Получение значения словаря `d1` по ключу `'key'`:
  - `a = d1['key']`
  - `a = d1.get('key')`
- Удаление пары "ключ-значение" по ключу: `del d1['key']`
- Получить набор ключей словаря `d1`: `d1.keys()`
- Получить набор ключей словаря `d1`: `d1.values()`

Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном стандартным типам данных:

`https://docs.python.org/3/library/stdtypes.html#dict`

15. Дополните код программы `lab_03_05.py`. Создайте словарь `myInfo`, содержащий информацию о Ваших фамилии, имени, отчестве, дне, месяце, годе рождения и университете в полях `surname`, `name`, `middlename`, `day`, `month`, `year`, `university` соответственно. Получите и поочередно выведите на экран списки ключей и значений словаря `myInfo`. Ознакомьтесь с результатом.

16. Создайте файл `lab_03_06.py`:

```
'''
    Функции
'''
```

```
def dictUpdate(a):
    a.update([("x",5)])
    print("dict in function: ",a)
    return
def dictNoUpdate(a):
    a = a.copy()
    a.update([("y",3)])
    print("dict in function: ",a)
    return
def returnFunc(a):
    def f1(a):
        print("returned f1(a): ",a)
    return f1
d= {"v":7}
dictUpdate(d)
print("dict out of function: ",d)
dictNoUpdate(d)
print("dict out of function: ",d)
f = returnFunc(d)
print("f: ", f)
f(2)
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
dict in function:  {'v': 7, 'x': 5}
dict out of function:  {'v': 7, 'x': 5}
dict in function:  {'v': 7, 'y': 3, 'x': 5}
dict out of function:  {'v': 7, 'x': 5}
f:  <function returnFunc.<locals>.f1 at 0x7f6d7a778048>
returned f1(a):  2
```

Функции

Функции представляют собой объекты, принимающие на вход некоторые аргументы, и возвращающие определенные значения. В языке программирования Python объявление функции происходит с использованием ключевого слова `def`:

```
def <имя_функции>(<аргумент_1>, <аргумент_2>):
    <операторы>
    return <значение>
```

Аргументы функции (`<аргумент_1>, <аргумент_2>`) указываются в скобках после имени функции (`<имя_функции>`). Такие функции называются именными. Возвращаемое значение (`<значение>`) указывается после ключевого слова `return`. Аргументы в функции передаются по ссылкам, вследствие чего, во избежание перезаписи значений, в блоке операторов (`<операторы>`) необходимо аккуратно обрабатывать входные

аргументы. В качестве возвращаемого значения функции может быть также указана внутренняя функция, например:

```
def returnFunc(a):
    def f1(a):
        print("returned f1(a): ",a)
    return f1
```

Функция может не иметь возвращаемого значения. В этом случае, после слова `return` значение не указывается. При попытке вывода возвращаемой функции в терминал, будет выведена ссылка на объект функции.

17. Дополните код программы `lab_03_06.py`. Создайте функцию `returnMod`, возвращающую функцию, которая осуществляет расчет остатка от деления переданного аргумента на 15 и вывод этого значения на экран. Вызовите функцию `returnMod` и осуществите запись возвращенного значения в переменную `mod15`. С использованием переменной `mod15` добейтесь выполнения расчета остатка от деления и вывода значения на экран. Ознакомьтесь с результатом.

18. Создайте файл `lab_03_07.py`:

```
'''
    Аргументы функции
'''
def sum(x, y, z=1):
    return x + y + z

print("sum(1,2,3): ",sum(1,2,3))
print("sum(1,2): ",sum(1,2))
print("sum(x=1,y=3): ",sum(x=1,y=3))

# переменное количество аргументов
def printArgs(*args):
    print("args of printArgs(): ",args)
    return
# переменное количество аргументов и аргументов-ключевых слов
def printArgsnKwargs(m,*args,**kwargs):
    print("main argument of printArgsnKwargs(): ",m)
    print("args of printArgsnKwargs(): ",args)
    print("args of printArgsnKwargs(): ",kwargs)
    return

printArgs("Hello World!", 1, 3, 5)
printArgsnKwargs("Earth", 7.125, radius=6371, pos=3)
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
sum(1,2,3):  6
sum(1,2):  4
sum(x=1,y=3):  5
args of printArgs():  ('Hello World!', 1, 3, 5)
main argument of printArgsnKwargs():  Earth
args of printArgsnKwargs():  (7.125,)
args of printArgsnKwargs():  {'pos': 3, 'radius': 6371}
```

Аргументы функции

Количество аргументов функции может быть переменным. Для указания базового значения аргумента, который в некоторых случаях может отсутствовать, его значение указывается после имени аргумента и знака равно, например:

```
def sum(x, y, z=0):
    return x + y + z
```

В случае, если количество аргументов функции неизвестно, можно использовать следующую запись, позволяющую получить список аргументов позже в теле функции:

```
def func(*args):
    <операторы>
    return <значение>
```

Часть аргументов может быть представлена парой "ключ-значение", аналогично записи пропускаемого аргумента, описанной ранее. Число таких аргументов, называемых аргументами-ключевыми словами (keyword arguments, или kwargs), может быть неизвестном, аналогично обычным аргументам. В этом случае, необходимо использовать следующую форму записи:

```
def printArgsnKwargs(*args,**kwargs):
    <операторы>
    return <значение>
```

19. Дополните код программы `lab_03_07.py`. Создайте функцию `checkArgs()` с неизвестным количеством аргументов и аргументов-ключевых слов. Функция `checkArgs` проверяет количество переданных аргументов и аргументов-ключевых слов и:
    - Если количество аргументов меньше либо равно трем и количество аргументов-ключевых слов строго меньше трех – выводит их на экран.
    - Иначе – выводит предупреждение о превышении количества передаваемых аргументов.

20. Создайте файл `lab_03_08.py`:

```
'''
    Анонимные функции, lambda-выражения
'''
lfunc = lambda x, y, z = 1: x + y + z
print("lfunc(1,2,3): ",lfunc(1,2,3))
print("lfunc(1,2): ",lfunc(1,2))
```

```
print("lfunc(x=1,y=3): ",lfunc(x=1,y=3))
print("lambda result: ", \
    (lambda a,b,sep=", ": sep.join((a,b)))("Hello","World!"))
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
lfunc(1,2,3):  6
lfunc(1,2):  4
lfunc(x=1,y=3):  5
lambda result:  Hello, World!
```

Анонимные функции, lambda-выражения

Анонимные функции могут содержать лишь одно выражение, однако выполняются они быстрее, чем именные функции. Анонимные функции представляют собой так называемые `lambda`-выражения, определяемые ключевым словом `lambda`, например:

```
f = lambda <аргумент_1>, <аргумент_2>,…: <оператор>
```

Анонимные функции так же, как и именные могут содержать переменное количество аргументов. Для анонимных функций результат выполнения оператора (`<оператор>`) является возвращаемым значением. Этот тип функций также может быть использован и без записи в переменную с помощью вызова при объявлении. В этом случае, сначала в скобках пишется `lambda`-выражение, после чего в скобках перечисляются аргументы функции:

```
(lambda a,b,sep=", ": sep.join((a,b)))("Hello","World!")
```

21. Дополните код программы `lab_03_08.py`. Создайте `lambda`-выражение, присвоив его переменной `lam`, осуществляющее проверку равенства остатка от деления введенного с клавиатуры числа, передаваемого в качестве аргумента, на 3. При равенстве остатка от деления на три нулю функция выводит значение на экран.

22. Создайте файлы со следующим содержимым:

Имя файла	*Содержимое*
`file1.txt`	Hello, World!
`file2.txt`	Yesterday all my troubles seemed so far away. Now it looks as though they're here to stay. Oh, I believe in yesterday.
`file3.txt`	

23. Создайте файл `lab_03_09.py`:

```
'''
    Работа с файлами
```

```
'''
file1 = open("file1.txt", "r") # открыть на чтение
st = file1.read(1) # считать 1 символ
st += file1.read() # считать до конца файла
print("File1: ", st)
file1.close()

# считать построчно
file2 = open("file2.txt", "r")
print("File2: ")
i = 0
for line in file2: # итерация по строкам файла
    print("Line {}: {}".format(i,line.replace("\n","")))
    i += 1
file2.close()

# запись в файл
file3 = open("file3.txt","w")
for s in "Strings can be easily written to file":
    file3.write(s)
    if s == " ":
        file3.write("\n")
print("File 3 was written successfully")
file3.close()
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
File1:  Hello, World!
File2:
Line 0: Yesterday all my troubles seemed so far away.
Line 1: Now it looks as though they're here to stay.
Line 2: Oh, I believe in yesterday.
File 3 was written successfully
```

Работа с файлами

Для открытия файла используется функция `open()`, аргументами которой является полное имя файла, а также спецификатор, указывающий режим открытия файла:

```
f = open(<имя_файла>, <спецификатор>)
```

Спецификатор	*Описание*
`'r'`	Открытие файла на чтение
`'w'`	Открытие файла на запись с перезаписью размещенных ранее данных. В случае отсутствия файла – файл создается автоматически
`'x'`	Открытие файла на запись. В случае отсутствия файла – выдается ошибка
`'a'`	Открытие файла на дополнение
`'b'`	Открытие файла в двоичном виде

't'	Открытие файла в виде текста (по умолчанию)
'+'	Открытие файла на чтение и запись

Режимы могут комбинироваться между собой, например при открытии файла на чтение в двоичном режиме используется запись 'rb'.

Для чтения данных из файла используется функция read(), вызываемая от объекта, созданного при открытии файла, которая может принимать на вход количество символов, которое необходимо считать. При отсутствии аргумента файл считывается до конца:

```
s = f.read(4)
s = f.read()
```

Для записи значения s в файл используется функция write(), вызываемая от объекта, созданного при открытии файла:

```
f.write(s)
```

Считывание файла построчно может проходить в цикле:

```
for line in f:
    <операторы>
```

Закрытие файла после завершения работы с ним происходит с помощью функции close():

```
f.close()
```

24. Создайте файл lab_03_10.py. Создайте словарь textDict для содержимого файла text1.txt, который приведен далее, по следующему принципу: ключи словаря – слова текста, значения словаря – частота появления слова в отрывке. Запишите словарь textDict в файл с именем textDict.txt.

Файл text1.txt:

```
Around the new position a circle, somewhat larger than in the former instance, was now described, and we again set to work with the spades. I was dreadfully weary, but, scarcely understanding what had occasioned the change in my thoughts, I felt no longer any great aversion from the labor imposed. I had become most unaccountably interested --nay, even excited.
```

25. Создайте файл lab_03_11.py. Создайте функции encodeHuffman(fileIn, fileOut) и decodeHuffman(fileIn, fileOut), осуществляющие кодирование и декодирование текста с использованием метода Хаффмана соответственно. fileIn – полное имя файла с исходным текстом, fileOut – полное имя файла, куда необходимо записать результат. Функции возвращают True, если ошибок не возникло, и False в ином случае. Осуществите проверку работы функций на произвольном тексте.

26. Дополните код программы lab_03_11.py. Создайте функции encodeLZ(fileIn, fileOut) и decodeLZ(fileIn, fileOut), осуществляющие кодирование и декодирование текста с использованием метода Лемпеля-Зива соответственно. fileIn – полное имя файла с исходным текстом, fileOut –полное имя

файла, куда необходимо записать результат. Функции возвращают `True`, если ошибок не возникло, и `False` в ином случае. Осуществите проверку работы функций на произвольном тексте.

27. Подберите два текста, для одного из которых лучше подходит метод Хаффмана, а для второго – метод Лемпеля-Зива. Рассчитайте коэффициенты сжатия информации. Выведите тексты и коэффициенты их сжатия разными методами на экран.