

ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ
СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Программирование на языках высокого уровня (Python)»

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Выполнил студент
Факультет: АИТ
Группа: ИВБ-211

Шефнер А.

Проверил:

Баталов Д.И.

Санкт-Петербург

2023

Оценочный лист результатов ЛР № 1

Ф.И.О. студента _____ Шефнер Альберт _____

Группа _____ ИВБ-211 _____

№ п/ п	Материалы необходимые для оценки знаний, умений и навыков	Показатель оценивания	Критерии Оценивания	Шкала оценивания	Оценка
1	Лабораторная работа №	Соответствие методике выполнения	Соответствует	7	
			Не соответствует	0	
		Срок выполнения	Выполнена в срок	2	
			Выполнена с опозданием на 2 недели	0	
		оформление	Соответствует требованиям	1 0	
			Не соответствует		
	ИТОГО количество баллов			10	

Доцент кафедры

«Информационные и вычислительные
системы»

_____ 2023 г.

Баталов Д.И. «__»

Тестирование

Для тестов используется модуль встроенной библиотеки unittest. Формат тестов везде одинаков: сначала вызов функции или процедуры, потом сверка её результата с ожидаемым с помощью assert. Если assert имеет два параметра, тестируемое значение всегда слева, ожидаемое всегда справа. Вот пример одного из тестов:

```
1 class TestTask20(TestCase):
2     def test_replace(self):
3         lst = [2, 5, 3, 8, 9]
4         lab.list_replace(lst, 2)
5         self.assertEqual(lst, [2, 5, 8, 3, 9])
6
7     def test_replace_throw(self):
8         lst = [2, 5, 3, 8, 9]
9         with self.assertRaises(Exception):
10            lab.list_replace(lst, -1)
11         with self.assertRaises(Exception):
12            lab.list_replace(lst, 4)
13         with self.assertRaises(Exception):
14            lab.list_replace(lst, 35)
```

Программа успешно проходит все приведённые ниже тесты.

```
albert@ryzenpc ~/D/u/3/P/Lab 1 (main)> python -m unittest test_lab.py
.....
-----
Ran 47 tests in 0.003s
```

Более подробные результаты:

```
albert@ryzenpc ~/D/u/3/P/Lab 1 (main)> python -m unittest test_lab.py -v
test_str_to_lower_upper (test_lab.TestTask11.test_str_to_lower_upper) ... ok
test_str_most_frequent_symbol (test_lab.TestTask12.test_str_most_frequent_symbol) ... ok
test_str_swap_case (test_lab.TestTask13.test_str_swap_case) ... ok
test_list_sum_1 (test_lab.TestTask14.test_list_sum_1) ... ok
test_list_sum_2 (test_lab.TestTask14.test_list_sum_2) ... ok
test_list_multiply (test_lab.TestTask15.test_list_multiply) ... ok
test_list_min_max (test_lab.TestTask16.test_list_min_max) ... ok
test_list_min_max (test_lab.TestTask17.test_list_min_max) ... ok
test_list_copy_1 (test_lab.TestTask18.test_list_copy_1) ... ok
test_list_copy_2 (test_lab.TestTask18.test_list_copy_2) ... ok
test_list_concat_1 (test_lab.TestTask19.test_list_concat_1) ... ok
test_list_concat_2 (test_lab.TestTask19.test_list_concat_2) ... ok
test_str_len_1 (test_lab.TestTask2.test_str_len_1) ... ok
test_str_len_2 (test_lab.TestTask2.test_str_len_2) ... ok
test_replace (test_lab.TestTask20.test_replace) ... ok
test_replace_throw (test_lab.TestTask20.test_replace_throw) ... ok
test_list_join (test_lab.TestTask21.test_list_join) ... ok
test_dict_concat (test_lab.TestTask24.test_dict_concat) ... ok
test_dict_contains_key (test_lab.TestTask25.test_dict_contains_key) ... ok
test_dict_remove_key (test_lab.TestTask26.test_dict_remove_key) ... ok
test_dict_min_max_1 (test_lab.TestTask27.test_dict_min_max_1) ... ok
test_tuple_add_element (test_lab.TestTask29.test_tuple_add_element) ... ok
test_new_str (test_lab.TestTask3.test_new_str) ... ok
test_tuple_to_dict (test_lab.TestTask30.test_tuple_to_dict) ... ok
test_tuple_to_dict (test_lab.TestTask31.test_tuple_to_dict) ... ok
test_count_tuples (test_lab.TestTask32.test_count_tuples) ... ok
test_set_add (test_lab.TestTask34.test_set_add) ... ok
test_set_remove (test_lab.TestTask35.test_set_remove) ... ok
test_set_union (test_lab.TestTask37.test_set_union) ... ok
test_set_count_1 (test_lab.TestTask38.test_set_count_1) ... ok
test_set_count_2 (test_lab.TestTask38.test_set_count_2) ... ok
test_set_contains (test_lab.TestTask39.test_set_contains) ... ok
test_replace_dollar (test_lab.TestTask4.test_replace_dollar) ... ok
test_file_write (test_lab.TestTask40.test_file_write) ... ok
test_file_read (test_lab.TestTask41.test_file_read) ... ok
test_file_get_n_last_strings (test_lab.TestTask43.test_file_get_n_last_strings) ... ok
test_file_newline_count (test_lab.TestTask44.test_file_newline_count) ... ok
test_file_most_frequent_word (test_lab.TestTask45.test_file_most_frequent_word) ... ok
test_file_copy (test_lab.TestTask46.test_file_copy) ... ok
test_dict_write_to_file_and_return_contents (test_lab.TestTask47.test_dict_write_to_file_and_return_contents) ... ok
test_dict_write_to_file_and_return_contents (test_lab.TestTask48.test_dict_write_to_file_and_return_contents) ... ok
test_file_serialize_and_return_content (test_lab.TestTask49.test_file_serialize_and_return_content) ... ok
test_str_reverse (test_lab.TestTask5.test_str_reverse) ... ok
test_count_symbols (test_lab.TestTask6.test_count_symbols) ... ok
test_str_even_odd (test_lab.TestTask7.test_str_even_odd) ... ok
test_str_remove (test_lab.TestTask8.test_str_remove) ... ok
test_str_to_lower_upper (test_lab.TestTask9.test_str_to_lower_upper) ... ok

-----
Ran 47 tests in 0.003s

OK
```

1. Что будет результатом следующих выражений?

```
"agility"[2:5] + "taxonomy"[3:6]  
[115,202,192,334,257][:4]  
len("crazy"[3:3+4])  
[9,8,7,6,5,4,3,2,1][-3:]  
type([False,True,False,True][2:3])  
"---".join( "this is important".split() )  
int( ''.join( "7/7/07".split('/') ) )  
"too soon to tell".replace('o','*').replace('* ','')
```

Решение:

```
iliono  
[115, 202, 192, 334]  
2  
[3, 2, 1]  
<class 'list'>  
this---is---important  
7707  
t sn t tell
```

2. Напишите скрипт, вычисляющий двумя способами длину строки.

Решение:

```
1 def str_len_1(s: str) → int:
2     return s.count("") - 1
3
4
5 def str_len_2(s: str) → int:
6     return len(s)
```

Тесты:

```
1 class TestTask2(TestCase):
2     def test_str_len_1(self):
3         len1 = lab.str_len_1("")
4         self.assertEqual(len1, 0)
5
6         len2 = lab.str_len_1("qe")
7         self.assertEqual(len2, 2)
8
9         len3 = lab.str_len_1("892738hd423d472d4j28379s8293yhfkhl34323hr23jrhk23jdn")
10        self.assertEqual(len3, 52)
11
12    def test_str_len_2(self):
13        len1 = lab.str_len_2("")
14        self.assertEqual(len1, 0)
15
16        len2 = lab.str_len_2("qe")
17        self.assertEqual(len2, 2)
18
19        len3 = lab.str_len_2("892738hd423d472d4j28379s8293yhfkhl34323hr23jrhk23jdn")
20        self.assertEqual(len3, 52)
```


3. Напишите скрипт, который позволяет из строки собрать другую по следующим правилам: новая строка должна состоять из двух первых и двух последних элементов исходной строки. Если длина исходной строки меньше двух, то результатом будет пустая строка.

Решение:

```
1 def new_str(s: str) → str:
2     if len(s) < 2:
3         return ""
4     return s[:2] + s[-2:]
```

Тесты:

```
1 class TestTask3(TestCase):
2     def test_new_str(self):
3         str1 = lab.new_str("")
4         self.assertEqual(str1, "")
5
6         str2 = lab.new_str("x")
7         self.assertEqual(str2, "")
8
9         str3 = lab.new_str("xd")
10        self.assertEqual(str3, "xdxd")
11
12        str4 = lab.new_str("Really big string.")
13        self.assertEqual(str4, "Reg.")
```

4. Напишите скрипт, который заменяет произвольный символ/букву в строке на «\$».

Решение:

```
1 def str_replace_dollar(s: str, index: int) → str:
2     if not 0 ≤ index < len(s):
3         return s
4     return s[:index] + "$" + s[index + 1 :]
```

Тесты:

```
1 class TestTask4(TestCase):
2     def test_replace_dollar(self):
3         str1 = lab.str_replace_dollar("01234567", 3)
4         self.assertEqual(str1, "012$4567")
5
6         str2 = lab.str_replace_dollar("", 3)
7         self.assertEqual(str2, "")
8
9         str3 = lab.str_replace_dollar("Bigger str", 23)
10        self.assertEqual(str3, "Bigger str")
11
12        str4 = lab.str_replace_dollar("str", -23)
13        self.assertEqual(str4, "str")
```


5. Напишите скрипт, который позволяет инвертировать последовательность элементов в строке.

Решение:

```
1 def str_reverse(s: str) → str:
2     return s[::-1]
```

Тесты:

```
1 class TestTask5(TestCase):
2     def test_str_reverse(self):
3         str1 = lab.str_reverse("")
4         self.assertEqual(str1, "")
5
6         str2 = lab.str_reverse("123")
7         self.assertEqual(str2, "321")
```

6. Напишите скрипт, который считает количество вхождений символа в строку. Например: «google.com» — {'o': 3, 'g': 2, '.': 1, 'e': 1, 'l': 1, 'm': 1, 'c': 1}

Решение:

```
1 | def count_symbols(s: str) → Dict[str, int]:
2 |     result = dict[str, int]()
3 |     for c in s:
4 |         if c in result:
5 |             result[c] += 1
6 |         else:
7 |             result[c] = 1
8 |
9 |     return dict(sorted(result.items(), key=lambda v: v[1], reverse=True))
```

Тесты:

```
1 | class TestTask6(TestCase):
2 |     def test_count_symbols(self):
3 |         symbols1 = lab.count_symbols("")
4 |         self.assertEqual(symbols1, {})
5 |
6 |         symbols2 = lab.count_symbols("google.com")
7 |         self.assertEqual(
8 |             symbols2, {"o": 3, "g": 2, "e": 1, "l": 1, ".": 1, "c": 1, "m": 1}
9 |         )
```

7. Напишите скрипт, позволяющий из исходной строки собрать две новые. Первая строка должна состоять только из элементов с нечетными индексами исходной строки, а вторая — с четными.

Решение:

```
2 | def str_even_odd(s: str) → Tuple[str, str]:  
3 |     return (s[::2], s[1::2])
```

Тесты:

```
1 | class TestTask7(TestCase):  
2 |     def test_str_even_odd(self):  
3 |         str1 = lab.str_even_odd("")  
4 |         self.assertEqual(str1, ("", ""))  
5 |  
6 |         str2 = lab.str_even_odd("0123456789")  
7 |         self.assertEqual(str2, ("02468", "13579"))
```

8. Напишите скрипт, который удаляет задаваемый произвольный символ в строке.

Решение:

```
1 def str_remove(s: str, index: int) → str:
2     if not 0 ≤ index < len(s):
3         return s
4     return s[:index] + s[index + 1 :]
```

Тесты:

```
1 class TestTask8(TestCase):
2     def test_str_remove(self):
3         str1 = lab.str_remove("012345", 3)
4         self.assertEqual(str1, "01245")
```

9. Напишите скрипт, который позволяет переводить все символы исходной строки в верхний и нижний регистры.

Решение:

```
1 | def str_to_lower_upper(s: str) → Tuple[str, str]:  
2 |     return (s.lower(), s.upper())
```

Тесты:

```
1 | class TestTask9(TestCase):  
2 |     def test_str_to_lower_upper(self):  
3 |         result = lab.str_to_lower_upper("Some String with MIXED case.")  
4 |         self.assertEqual(  
5 |             result, ("some string with mixed case.", "SOME STRING WITH MIXED CASE.")  
6 |         )
```


10. Напишите скрипт, выводящий все элементы строки с их номерами индексов.

Решение:

```
1  def print_symbols(s: str) → None:
2      for index, char in enumerate(s):
3          print(f"{index + 1}: {char}")
```

Тесты:

```
>>> lab.print_symbols("String to print")
1: S
2: t
3: r
4: i
5: n
6: g
7:
8: t
9: o
10:
11: p
12: r
13: i
14: n
15: t
```

11. Напишите скрипт, проверяющий, содержится ли произвольный символ (элемент) или слово в строке.

Решение:

```
1 def str_contains(s: str, c: str):  
2     return c in s
```

Тесты:

```
1 class TestTask11(TestCase):  
2     def test_str_to_lower_upper(self):  
3         self.assertTrue(lab.str_contains("Bigger string", "ing"))  
4         self.assertFalse(lab.str_contains("Bigger string", "big"))  
5
```

12. Выведите символ, который встречается в строке чаще, чем остальные.

Решение:

```
1 | def str_most_frequent_symbol(s: str) → str:
2 |     symbols = count_symbols(s)
3 |     return max(symbols, key=symbols.get) #
```

Тесты:

```
1 | class TestTask12(TestCase):
2 |     def test_str_most_frequent_symbol(self):
3 |         char1 = lab.str_most_frequent_symbol("string with some symbols")
4 |         self.assertEqual(char1, "s")
5 |
6 |         char2 = lab.str_most_frequent_symbol("gggggeee")
7 |         self.assertEqual(char2, "g")
```

13. Поменяйте регистр элементов строки.

Решение:

```
1 def str_swap_case(s: str) → str:  
2     return s.swapcase()
```

Тесты:

```
1 class TestTask13(TestCase):  
2     def test_str_swap_case(self):  
3         str1 = lab.str_swap_case("String. With some MIXED cAsE.")  
4         self.assertEqual(str1, "sTRING. wITH SOME mixed CaSe.")
```

14. Вычислите сумму элементов (чисел) в списке двумя разными способами.

Решение:

```
1 def list_sum_1(num_list: list) → Number:
2     return sum(num_list)
3
4
5 def list_sum_2(num_list: list) → Number:
6     return 0 if len(num_list) == 0 else reduce(lambda a, b: a + b, num_list)
```

Тесты:

```
1 class TestTask14(TestCase):
2     def test_list_sum_1(self):
3         sum1 = lab.list_sum_1([])
4         self.assertEqual(sum1, 0)
5
6         sum2 = lab.list_sum_1([1, 2, 3, 4, 5])
7         self.assertEqual(sum2, 15)
8
9     def test_list_sum_2(self):
10        sum1 = lab.list_sum_2([])
11        self.assertEqual(sum1, 0)
12
13        sum2 = lab.list_sum_2([1, 2, 3, 4, 5])
14        self.assertEqual(sum2, 15)
```


15. Умножьте каждый элемент списка на произвольное число.

Решение:

```
1 def list_multiply(num_list: NumberList, num: int) → NumberList:  
2     return list(map(lambda n: n * num, num_list))
```

Тесты:

```
1 class TestTask15(TestCase):  
2     def test_list_multiply(self):  
3         lst = [2, 4, 6, 9, 12]  
4         result = lab.list_multiply(lst, 3)  
5         self.assertEqual(result, [6, 12, 18, 27, 36])
```

16. Найдите максимальное и минимальное числа, хранящиеся в списке.

Решение:

```
1 | def list_min_max(num_list: NumberList) → Tuple[float, float]:  
2 |     return (min(num_list), max(num_list))
```

Тесты:

```
1 | class TestTask16(TestCase):  
2 |     def test_list_min_max(self):  
3 |         min_max_1 = lab.list_min_max([1, 6, 0, 3, -5, 34, 1])  
4 |         self.assertEqual(min_max_1, (-5, 34))
```

17. Напишите скрипт, удаляющий все повторяющиеся элементы из списка.

Решение:

```
1 def list_distinct(num_list: NumberList) → NumberList:  
2     return list(set(num_list))
```

Тесты:

```
1 class TestTask17(TestCase):  
2     def test_list_min_max(self):  
3         lst = lab.list_distinct([1, 6, 0, 3, 6, 34, 1])  
4         self.assertEqual(sorted(lst), [0, 1, 3, 6, 34])
```

18. Скопируйте список двумя различными способами.

Решение:

```
1  def list_copy_1(lst: List) → List:
2      return copy.copy(lst)
3
4
5  def list_copy_2(lst: List) → List:
6      return [e for e in lst]
```

Тесты:

```
1  class TestTask18(TestCase):
2      def test_list_copy_1(self):
3          lst1 = [5, "2", True, [74, "pk"]]
4          lst2 = lab.list_copy_1(lst1)
5          self.assertEqual(lst1, lst2)
6
7      def test_list_copy_2(self):
8          lst1 = [5, "2", True, [74, "pk"]]
9          lst2 = lab.list_copy_2(lst1)
10         self.assertEqual(lst1, lst2)
```

17. Напишите скрипт, удаляющий все повторяющиеся элементы из списка.

Решение:

```
1 def list_distinct(num_list: NumberList) → NumberList:  
2     return list(set(num_list))
```

Тесты:

```
1 class TestTask17(TestCase):  
2     def test_list_min_max(self):  
3         lst = lab.list_distinct([1, 6, 0, 3, 6, 34, 1])  
4         self.assertEqual(sorted(lst), [0, 1, 3, 6, 34])
```


19. Напишите скрипт для слияния (конкатенации) двух списков различными способами.

Решение:

```
1 | def list_concat_1(list_a: List, list_b: List) → List:
2 |     result = copy.copy(list_a)
3 |     result.extend(copy.copy(list_b))
4 |     return result
5 |
6 |
7 | def list_concat_2(list_a: List, list_b: List) → List:
8 |     result = []
9 |
10 |    for e in list_a:
11 |        result.append(e)
12 |
13 |    for e in list_b:
14 |        result.append((e))
15 |
16 |    return result
```

Тесты:

```
1 | class TestTask19(TestCase):
2 |     def test_list_concat_1(self):
3 |         lst1 = [4, 5, 6]
4 |         lst2 = ["x", "y", "z"]
5 |         lst3 = lab.list_concat_1(lst1, lst2)
6 |         self.assertEqual(lst3, [4, 5, 6, "x", "y", "z"])
7 |
8 |     def test_list_concat_2(self):
9 |         lst1 = [4, 5, 6]
10 |        lst2 = ["x", "y", "z"]
11 |        lst3 = lab.list_concat_2(lst1, lst2)
12 |        self.assertEqual(lst3, [4, 5, 6, "x", "y", "z"])
```

20. Напишите скрипт, меняющий позициями элементы списка с индексами n и $n + 1$.

Решение:

```
1 def list_replace(lst: List, n: int) → None:
2     if not 0 ≤ n < len(lst) - 1:
3         raise Exception("n should be bigger than 0 and less than len(lst) - 1")
4
5     temp = lst[n]
6     lst[n] = lst[n + 1]
7     lst[n + 1] = temp
```

Тесты:

```
1 class TestTask20(TestCase):
2     def test_replace(self):
3         lst = [2, 5, 3, 8, 9]
4         lab.list_replace(lst, 2)
5         self.assertEqual(lst, [2, 5, 8, 3, 9])
6
7     def test_replace_throw(self):
8         lst = [2, 5, 3, 8, 9]
9         with self.assertRaises(Exception):
10             lab.list_replace(lst, -1)
11         with self.assertRaises(Exception):
12             lab.list_replace(lst, 4)
13         with self.assertRaises(Exception):
14             lab.list_replace(lst, 35)
```

21. Напишите скрипт, переводящий список из различного количества числовых целочисленных элементов в одно число. Пример списка: [15, 23, 150], результат: 1523150

Решение:

```
1 | def list_join(num_list: List[int]) → int:
2 |     str_list = map(str, num_list)
3 |     s = "".join(str_list)
4 |     return int(s)
```

Тесты:

```
1 | class TestTask21(TestCase):
2 |     def test_list_join(self):
3 |         lst = [15, 23, 150]
4 |         result = lab.list_join(lst)
5 |         self.assertEqual(result, 1523150)
```

22. Объявите и инициализируйте словарь различными способами.

Решение:

```
1  def dictionary_init() → None:
2      dict_a = {"a": 1, "b": 2}
3
4      dict_b = dict[int, str]()
5      dict_b[3] = "c"
6      dict_b[4] = "d"
7
8      print(f"dict_a: {dict_a}")
9      print(f"dict_b: {dict_b}")
```

Тесты:

```
>>> lab.dictionary_init()
dict_a: {'a': 1, 'b': 2}
dict_b: {3: 'c', 4: 'd'}
```

23. Добавьте еще несколько пар «ключ: значение» в следующий словарь:
{0: 10, 1: 20}.

Решение:

```
1  def dict_add_script() → None:
2      starting_dict = {0: 10, 1: 20}
3
4      starting_dict[3] = 53
5      starting_dict[45] = 2
6      starting_dict[10] = 10
7
8      print(starting_dict)
```

Тесты:

```
>>> lab.dict_add_script()
{0: 10, 1: 20, 3: 53, 45: 2, 10: 10}
```


24. Напишите скрипт, который из трех словарей создаст один новый.

Решение:

```
1 def dict_concat(dict_a: Dict, dict_b: Dict, dict_c: Dict) → Dict:
2     result = copy.copy(dict_a)
3     result.update(dict_b)
4     result.update(dict_c)
5     return result
```

Тесты:

```
1 class TestTask24(TestCase):
2     def test_dict_concat(self):
3         dict1 = {"a": 6}
4         dict2 = {"j": 53, "b": 2, "r": 64}
5         dict3 = {"n": 5, "j": 23}
6         result_dict = lab.dict_concat(dict1, dict2, dict3)
7         self.assertEqual(
8             result_dict, {"a": 6, "j": 23, "b": 2, "n": 5, "r": 64, "n": 5}
9         )
```

25. Напишите скрипт, проверяющий, существует ли заданный ключ в словаре.

Решение:

```
1 | def dict_contains_key(dct: Dict, key: Any) → bool:
2 |     return key in dct
```

Тесты:

```
1 | class TestTask25(TestCase):
2 |     def test_dict_contains_key(self):
3 |         dct = {2: "two", 4: "xd", 7: "--="}
4 |         self.assertTrue(lab.dict_contains_key(dct, 2))
5 |         self.assertFalse(lab.dict_contains_key(dct, 0))
```

26. Напишите скрипт для удаления элемента словаря.

Решение:

```
1 | def dict_remove_key(dct: Dict, key: Any) → None:  
2 |     dct.pop(key)
```

Тесты:

```
1 | class TestTask26(TestCase):  
2 |     def test_dict_remove_key(self):  
3 |         dct = {1: "a", 2: "b", 3: "c"}  
4 |         lab.dict_remove_key(dct, 2)  
5 |         self.assertEqual(dct, {1: "a", 3: "c"})
```

27. Напишите скрипт, который выводит максимальное и минимальное числа среди значений словаря.

Решение:

```
1 | def dict_min_max(dct: Dict[Any, float]) → Tuple:  
2 |     min_res = min(dct, key=dct.get) # pyright: ignore  
3 |     max_res = max(dct, key=dct.get) # pyright: ignore  
4 |     return (min_res, max_res)
```

Тесты:

```
1 | class TestTask27(TestCase):  
2 |     def test_dict_min_max_1(self):  
3 |         dct = {"Jose": 15, "Mason": 49, "Pablo": 23, "Gibraltar": 98, "Ganesha": 44}  
4 |         mn, mx = lab.dict_min_max(dct) # pyright: ignore [reportGeneralTypeIssues]  
5 |         self.assertEqual((mn, mx), ("Jose", "Gibraltar"))
```

28. Объявите и инициализируйте кортеж различными способами, после чего распакуйте его.

Решение:

```
1  def tuple_script() → None:
2  |      tuple_1 = (3, 5)
3  |      tuple_2 = tuple([3, 5])
4  |
5  |      print(tuple_1)
6  |      print(tuple_2)
```

Тесты:

```
>>> lab.tuple_script()
(3, 5)
(3, 5)
```

29. Напишите скрипт для добавления элементов в кортеж.

Решение:

```
1 def tuple_add_element(tup: Tuple, elem: Any) → Tuple:  
2     return (*tup, elem)
```

Тесты:

```
1 class TestTask29(TestCase):  
2     def test_tuple_add_element(self):  
3         tup1 = (1, 2, "x")  
4         tup2 = lab.tuple_add_element(tup1, True)  
5         self.assertEqual(tup2, (1, 2, "x", True))
```

30. Напишите скрипт, конвертирующий список в кортеж.

Решение:

```
1 def list_to_tuple(lst: list) → Tuple:  
2     return tuple(lst)
```

Тесты:

```
1 class TestTask30(TestCase):  
2     def test_tuple_to_dict(self):  
3         lst = [1, 2, "BMW"]  
4         tup = lab.list_to_tuple(lst)  
5         self.assertEqual(tup, (1, 2, "BMW"))
```


31. Конвертируйте кортеж в словарь.

Решение:

```
1 def tuple_to_dict(tup: Tuple[Tuple[T, Any], ...]) → Dict[T, Any]:  
2     return dict(tup)
```

Тесты:

```
1 class TestTask31(TestCase):  
2     def test_tuple_to_dict(self):  
3         dct1 = lab.tuple_to_dict((("a", 1), ("b", 2)))  
4         self.assertEqual(dct1, {"a": 1, "b": 2})  
5  
6         dct2 = lab.tuple_to_dict(())  
7         self.assertEqual(dct2, {})
```

32. Напишите скрипт, подсчитывающий количество элементов типа кортеж в списке.

Решение:

```
1 def count_tuples(lst: List) → int:
2     return sum(isinstance(e, tuple) for e in lst)
```

Тесты:

```
1 class TestTask32(TestCase):
2     def test_count_tuples(self):
3         count1 = lab.count_tuples([(2, 4), 8, "xd", (3,), (), [2]])
4         self.assertEqual(count1, 3)
5
6         count2 = lab.count_tuples([])
7         self.assertEqual(count2, 0)
```

33. Объявите и инициализируйте множество различными способами.

Решение:

```
1  def set_script() → None:
2      set1 = {2, 4, 8, 9, 6}
3      print(type(set1))
4      print(set1)
5
6      set2 = set([8, 6, 3, 4, 0])
7      print(type(set2))
8      print(set2)
```

Тесты:

```
>>> lab.set_script()
<class 'set'>
{2, 4, 6, 8, 9}
<class 'set'>
{0, 3, 4, 6, 8}
```

34. Добавьте элемент в множество.

Решение:

```
1  def set_add(st: set, elem: Any) → None:  
2  |      st.add(elem)
```

Тесты:

```
1  class TestTask34(TestCase):  
2  |      def test_set_add(self):  
3  |          |      st1 = {3, 6}  
4  |          |      lab.set_add(st1, 7)  
5  |          |      self.assertEqual(st1, {3, 6, 7})
```

35. Удалите элемент из множества.

Решение:

```
1 def set_remove(st: set, elem: Any) → None:  
2     st.remove(elem)
```

Тесты:

```
1 class TestTask35(TestCase):  
2     def test_set_remove(self):  
3         st1 = {3, 6, 7}  
4         lab.set_remove(st1, 7)  
5         self.assertEqual(st1, {3, 6})
```

37. Напишите скрипт для объединения двух множеств.

Решение:

```
1  def set_union(set1: set, set2: set) → set:
2      result = set()
3      result.update(set1)
4      result.update(set2)
5      return result
```

Тесты:

```
1  class TestTask37(TestCase):
2      def test_set_union(self):
3          set1 = {2, 44, 3}
4          set2 = {7, 5, 3}
5          set3 = lab.set_union(set1, set2)
6          self.assertEqual(set3, {2, 3, 5, 7, 44})
```

38. Напишите скрипт, находящий длину множества двумя различными способами.

Решение:

```
1 def set_count_1(st: set) → int:
2     return len(st)
3
4
5 def set_count_2(st: set) → int:
6     return st.__len__()
```

Тесты:

```
1 class TestTask38(TestCase):
2     def test_set_count_1(self):
3         count1 = lab.set_count_1({2, 5, 6, 7})
4         self.assertEqual(count1, 4)
5
6         count2 = lab.set_count_1(set())
7         self.assertEqual(count2, 0)
8
9     def test_set_count_2(self):
10        count1 = lab.set_count_2({2, 5, 6, 7})
11        self.assertEqual(count1, 4)
12
13        count2 = lab.set_count_2(set())
14        self.assertEqual(count2, 0)
```


39. Напишите скрипт для проверки, входит ли элемент в множество.

Решение:

```
1 def set_contains(st: set, elem: Any) → bool:
2     return elem in st
```

Тесты:

```
1 class TestTask39(TestCase):
2     def test_set_contains(self):
3         self.assertTrue(lab.set_contains({2, 4}, 4))
4         self.assertFalse(lab.set_contains({2, 4}, 5))
```

40. Напишите скрипт для записи текста в файл.

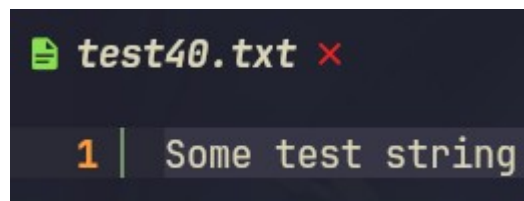
Решение:

```
1 def file_write(file_name: str, text: str) → None:
2     with open(file_name, "w") as file:
3         file.write(text)
```

Тесты:

```
1 class TestTask40(TestCase):
2     def test_file_write(self):
3         text = "Some test string"
4         file_name = "test40.txt"
5         lab.file_write(file_name, text)
6
7         with open(file_name, "r") as file:
8             same_text = file.read()
9             self.assertEqual(text, same_text)
```

Файлы, появляющиеся при тестах:



```
test40.txt x
1 | Some test string
```

41. Напишите скрипт для чтения текста из файла.


Решение:

```
1 def file_read(file_name: str) → str:
2     result: str
3     with open(file_name, "r") as file:
4         result = file.read()
5     return result
```

Тесты:

```
1 class TestTask41(TestCase):
2     def test_file_read(self):
3         text = "Some other text string"
4         file_name = "test41.txt"
5         with open(file_name, "w") as file:
6             file.write(text)
7
8         same_text = lab.file_read(file_name)
9         self.assertEqual(text, same_text)
```

Файлы, появляющиеся при тестах:

 test41.txt ×

1 | Some other text string

42. Напишите скрипт для добавления текста в файл и отображения содержимого файла.

Решение:

```
1 def file_append_script() → None:
2     file_name = input("Enter file name: ")
3     text = input("Enter text to append:\n")
4
5     with open(file_name, "a") as file:
6         print(text, end="\n", file=file)
7
8     with open(file_name, "r") as file:
9         all_content = file.read()
10        print("The contents of a file:")
11        print(all_content)
```


Тесты:

```
>>> lab.file_append_script()
Enter file name: myfile.txt
Enter text to append:
I have a string number 1
The contents of a file:
I have a string number 1

>>> lab.file_append_script()
Enter file name: myfile.txt
Enter text to append:
Lololol
The contents of a file:
I have a string number 1
Lololol

>>> lab.file_append_script()
Enter file name: myfile.txt
Enter text to append:
Last sentence.
The contents of a file:
I have a string number 1
Lololol
Last sentence.
```

Файлы, появляющиеся при тестах:

 myfile.txt ✖

0	I have a string number 1
1	Lololol
2	Last sentence.

43. Напишите скрипт для чтения последних n строк файла.

Решение:

```
1 def file_get_last_n_strings(file_name: str, n: int):
2     result = list()
3     with open(file_name, "r") as file:
4         result = file.readlines()[-n:]
5     return result
```

Тесты:

```
1 class TestTask43(TestCase):
2     def test_file_get_n_last_strings(self):
3         file_name = "test43.txt"
4         n = 3
5         lines = ["x1\n", "x2\n", "x3\n", "x4\n", "x5\n"]
6         with open(file_name, "w") as file:
7             file.writelines(lines)
8
9         result = lab.file_get_last_n_strings(file_name, n)
10        self.assertEqual(result, lines[-n:])
```

Файлы, появляющиеся при тестах:



	test43.txt
0	x1
1	x2
2	x3
3	x4
4	x5

44. Напишите скрипт, подсчитывающий количество строк в файле.

Решение:

```
1 def file_newline_count(file_name: str) → int:
2     result = None
3     with open(file_name, "r") as file:
4         text = file.read()
5         result = text.count("\n")
6     return result
```

Тесты:

```
1 class TestTask44(TestCase):
2     def test_file_newline_count(self):
3         file_name = "test44.txt"
4         text = "1\n2\n3\n87\n"
5         count = text.count("\n")
6
7         with open(file_name, "w") as file:
8             file.write(text)
9
10        same_count = lab.file_newline_count(file_name)
11        self.assertEqual(count, same_count)
```

Файлы, появляющиеся при тестах:



0	1
1	2
2	3
3	87

45. Напишите скрипт, позволяющий найти самое встречаемое слово в файле.



Решение:

```
1 def file_most_frequent_word(file_name: str) → str:
2     text = None
3     with open(file_name, "r") as file:
4         text = file.read()
5
6     filter_str = ",.!?;:-'\\"
7     for c in filter_str:
8         text = text.replace(c, "")
9     text = text.lower()
10    words = filter(lambda s: len(s) > 2, text.split(" "))
11    word_counts = {}
12    for word in words:
13        if word in word_counts:
14            word_counts[word] += 1
15        else:
16            word_counts[word] = 0
17
18    return dict_min_max(word_counts)[1]
```

Тесты:

```
1 class TestTask45(TestCase):
2     def test_file_most_frequent_word(self):
3         file_name = "test45.txt"
4         most_frequent = lab.file_most_frequent_word(file_name)
5         self.assertEqual(most_frequent, "the")
```

Файлы, появляющиеся при тестах:

 test45.txt 

```
0 | Python 3.12 is the latest stable release of the Python
   | programming language, with a mix of changes to the language and
   | the standard library. The library changes focus on cleaning up
   | deprecated APIs, usability, and correctness. Of note, the
   | distutils package has been removed from the standard library.
   | Filesystem support in os and pathlib has seen a number of
   | improvements, and several modules have better performance.
1 |
2 | The language changes focus on usability, as f-strings have had
   | many limitations removed and 'Did you mean ...' suggestions
   | continue to improve. The new type parameter syntax and type
   | statement improve ergonomics for using generic types and type
   | aliases with static type checkers.
3 |
4 | This article doesn't attempt to provide a complete
   | specification of all new features, but instead gives a
   | convenient overview. For full details, you should refer to the
   | documentation, such as the Library Reference and Language
   | Reference. If you want to understand the complete
   | implementation and design rationale for a change, refer to the
   | PEP for a particular new feature; but note that PEPs usually
   | are not kept up-to-date once a feature has been fully
   | implemented.
```

46. Напишите скрипт, копирующий содержимое одного файла в другой.

Решение:

```
1 def file_copy(src: str, dest: str) → None:
2     content = None
3     with open(src, "rb") as file:
4         content = file.read()
5
6     with open(dest, "wb") as file:
7         file.write(content)
```

Тесты:

```
1 class TestTask46(TestCase):
2     def test_file_copy(self):
3         src = "test46_src.txt"
4         dest = "test46_dest.txt"
5         text = "Not really important text just to test file copy in binary mode.\n"
6         same_text = None
7
8         with open(src, "w") as file:
9             file.write(text)
10
11         lab.file_copy(src, dest)
12
13         with open(dest, "r") as file:
14             same_text = file.read()
15
16         self.assertEqual(text, same_text)
```

Файлы, появляющиеся при тестах:

```
test46_src.txt ×
0 | Not really important text just to test file copy in binary mode.
```

```
test46_dest.txt ×
0 | Not really important text just to test file copy in binary mode.
```

47. Запишите словарь в файл посредством модуля pickle и прочитайте его.

Решение:

```

1 def object_write_to_file_and_return_contents(obj: object, file_name: str) → object:
2     with open(file_name, "wb") as file:
3         pickle.dump(obj, file)
4
5     content = None
6     with open(file_name, "rb") as file:
7         content = pickle.load(file)
8
9     return content

```

Тесты:

```
1 class TestTask47(TestCase):
2     def test_dict_write_to_file_and_return_contents(self):
3         dct = {"one": "Home", "two": "End", "three": "Ins"}
4         file_name = "test47.txt"
5
6         content = lab.object_write_to_file_and_return_contents(dct, file_name)
7         self.assertEqual(dct, content)
```


48. Запишите список в файл посредством модуля pickle и прочитайте его.

Решение:

См. задание 47

Тесты:

```
1 class TestTask48(TestCase):
2     def test_dict_write_to_file_and_return_contents(self):
3         lst = ["one", "Home", "two", "End", "three", "Ins"]
4         file_name = "test48.txt"
5
6         content = lab.object_write_to_file_and_return_contents(lst, file_name)
7         self.assertEqual(lst, content)
```

Файлы, появляющиеся при тестах:

```
<80>^D<95>,^@^@^@^@^@^@^@<94><(8c>^Cone<94><8c>^DHome<94><8c>^Ctwo<94><8c>^CEnd<94><8c>^Ethree<94><8c>^CIns<94>e.
```

49. Запишите словарь в файл посредством модуля json и прочитайте его.

Решение:

```
1 def file_serialize_and_return_content(obj: object, file_name: str) → str:
2     with open(file_name, "w") as file:
3         json.dump(obj, file)
4
5     content = None
6     with open(file_name, "r") as file:
7         content = file.read()
8
9     return content
```

Тесты:

```
1 class TestTask49(TestCase):
2     def test_file_serialize_and_return_content(self):
3         obj = {"a": [5, 6, 2], "b": 5, "c": True}
4         file_name = "test49.txt"
5         content = lab.file_serialize_and_return_content(obj, file_name)
6         content = content.replace(" ", "").replace("\n", "").replace("\t", "")
7         self.assertEqual(content, r'{"a":[5,6,2],"b":5,"c":true}')
```

Файлы, появляющиеся при тестах:

```
test49.txt ×
0 | {"a": [5, 6, 2], "b": 5, "c": true}
```