

Санкт-Петербургский Национальный Исследовательский  
Университет Информационных Технологий, Механики и Оптики

Факультет «Инфокоммуникационных Технологий»  
Направление подготовки «Программирование в  
инфокоммуникационных системах»

Тестирование программного обеспечения  
Лабораторная работа №1

Выполнила:  
Улитина Мария Сергеевна  
Группа №3322  
Проверил:  
Кочубеев Николай Сергеевич

Санкт-Петербург  
2024

**Цель работы:** научиться писать unit тесты.

**Задачи:**

1. Выбор репозитория с GitHub

Можно выбрать любой открытый проект на платформе GitHub. Это может быть и ваш пет проект.

2. Анализ тестируемых функциональностей

Важно выделить:

- Функциональные элементы (работа функций или методов).
- Критические части системы, которые должны быть протестированы.
- Важные случаи использования (use cases).

3. Написание тестов

- Создать модульные тесты для выбранных компонентов системы.
- Протестировать несколько сценариев работы, включая граничные случаи.
- Минимум 5 тестов должны быть написаны для разных функциональных частей приложения.
- Тесты должны быть написаны с использованием AAA (arrange, act, assert) и FIRST (fast, isolated, repeatable, self-validating, timely) principles

**Ход работы:**

**1-2. Выбор репозитория и анализ тестируемых функций.**

Принято решение реализовывать задачи на языке python. Был выбран репозиторий по ссылке: <https://github.com/BEPb/Python-100-days/blob/master/%D0%94%D0%B5%D0%BD%D1%8C%2016-20/%D0%94%D0%B5%D0%BD%D1%8C%2016/example02.py>

Этот репозиторий представляет из себя обучение python, поэтому содержит разные варианты кода, которые можно протестировать. Был выбран файл example02.py из папки «День 16».

На рисунках 1-3 код задания, который будет протестирован.

```

3 class Person(object):
4     def __init__(self, name, age):
5         self.name = name
6         self.age = age
7
8     def __gt__(self, other):
9         return self.name > other.name
10
11     def __str__(self):
12         return f'{self.name}: {self.age}'
13
14     def __repr__(self):
15         return self.__str__()
16
17
18 def select_sort(origin_items, comp=lambda x, y: x < y):
19     items = origin_items[:]
20     for i in range(len(items) - 1):
21         min_index = i
22         for j in range(i + 1, len(items)):
23             if comp(items[j], items[min_index]):
24                 min_index = j
25         items[i], items[min_index] = items[min_index], items[i]
26     return items
27
28
29 def bubble_sort(origin_items, *, comp=lambda x, y: x > y):
30     """Пузырьковая сортировка"""
31     items = origin_items[:]
32     for i in range(1, len(items)):
33         swapped = False
34         for j in range(i - 1, len(items) - 1):
35             if comp(items[j], items[j + 1]):
36                 items[j], items[j + 1] = items[j + 1], items[j]
37                 swapped = True
38         if swapped:
39             swapped = False
40             for j in range(len(items) - i - 1, i - 1, -1):
41                 if comp(items[j], items[j + 1]):
42                     items[j], items[j + 1] = items[j + 1], items[j]

```

Рисунок 1 – Код задания.

```

49 def merge_sort(items, comp=lambda x, y: x <= y):
50     """Слияние и сортировка"""
51     if len(items) < 2:
52         return items[:]
53     mid = len(items) // 2
54     left = merge_sort(items[:mid], comp)
55     right = merge_sort(items[mid:], comp)
56     return merge(left, right, comp)
57
58
59 def merge(items1, items2, comp=lambda x, y: x <= y):
60     """Объединить (объединить два упорядоченных списка в новый упорядоченный список)"""
61     items = []
62     index1, index2 = 0, 0
63     while index1 < len(items1) and index2 < len(items2):
64         if comp(items1[index1], items2[index2]):
65             items.append(items1[index1])
66             index1 += 1
67         else:
68             items.append(items2[index2])
69             index2 += 1
70     items += items1[index1:]
71     items += items2[index2:]
72     return items
73
74
75 def quick_sort(origin_items, comp=lambda x, y: x <= y):
76     """Быстрая сортировка"""
77     items = origin_items[:]
78     _quick_sort(items, 0, len(items) - 1, comp)
79     return items
80
81
82 def _quick_sort(items, start, end, comp):
83     """Рекурсивное деление и сортировка вызовов"""
84     if start < end:
85         pos = _partition(items, start, end, comp)
86         _quick_sort(items, start, pos - 1, comp)
87         _quick_sort(items, pos + 1, end, comp)

```

Рисунок 2 – Код задания (1).

```

90 def _partition(items, start, end, comp):
91     """Разделение"""
92     pivot = items[end]
93     i = start - 1
94     for j in range(start, end):
95         if comp(items[j], pivot):
96             i += 1
97             items[i], items[j] = items[j], items[i]
98     items[i + 1], items[end] = items[end], items[i + 1]
99     return i + 1
100
101
102 def main():
103     """Основная функция"""
104     items = [35, 97, 12, 68, 55, 73, 81, 40]
105     # print(bubble_sort(items))
106     # print(select_sort(items))
107     # print(merge_sort(items))
108     print(quick_sort(items))
109     items2 = [
110         Person('Wang', 25), Person('Luo', 39),
111         Person('Zhang', 50), Person('He', 20)
112     ]
113     # print(bubble_sort(items2, comp=lambda p1, p2: p1.age > p2.age))
114     # print(select_sort(items2, comp=lambda p1, p2: p1.name < p2.name))
115     # print(merge_sort(items2, comp=lambda p1, p2: p1.age <= p2.age))
116     print(quick_sort(items2, comp=lambda p1, p2: p1.age <= p2.age))
117     items3 = ['apple', 'orange', 'watermelon', 'durian', 'pear']
118     # print(bubble_sort(items3))
119     # print(bubble_sort(items3, comp=lambda x, y: len(x) > len(y)))
120     # print(merge_sort(items3))
121     print(merge_sort(items3))
122
123
124 if __name__ == '__main__':
125     main()

```

Рисунок 3 – Код задания (2).

## Проанализируем функционал:

### Функциональные элементы

#### 1. Класс Person:

- Метод init: инициализация объекта.
- Метод gt: сравнение двух объектов по имени.
- Метод str: возвращает строковое представление объекта.
- Метод repr: возвращает представление для отладки.

#### 2. Функции сортировки:

- select\_sort: сортировка выбором.
- bubble\_sort: пузырьковая сортировка.
- merge\_sort: сортировка слиянием.
- quick\_sort: быстрая сортировка.
- Вспомогательные функции \_quick\_sort и \_partition.
- Функция merge: объединение двух отсортированных списков.

## **Критические части системы, которые должны быть протестированы**

### **1. Класс Person:**

- Конструктор `__init__`: убедиться, что объект создается с правильными параметрами.

- Методы `str`, и `repr`: проверить корректность работы методов и их взаимодействие.

### **2. Функции сортировки:**

- `select_sort`, `bubble_sort`, `merge_sort`, `quick_sort`: проверить корректность сортировки для различных входных данных.

- Вспомогательные функции `_quick_sort`, `_partition`, и `merge`: убедиться, что они работают правильно и не содержат ошибок.

## **Use cases**

### **1. Класс Person:**

- Создание объекта `Person` с корректными значениями имени и возраста.

- Сравнение объектов `Person` с различными значениями имен.

- Проверка строкового и репрезентативного представлений объектов `Person`.

### **2. Для функций сортировки:**

- Подача различных списков на вход сортировочным функциям (например, уже отсортированные, отсортированные в обратном порядке, случайные данные).

- Проверка работы с различными параметрами `comp`, включая стандартные и произвольно определенные функции сравнения.

- Проверка обработки пустых списков и списков с единственными элементами.

## **3. Написание тестов**

Далее были написаны тесты с учетом AAA и FIRST. Суть AAA заключается в трех пунктах:

1. Arrange (Подготовка)

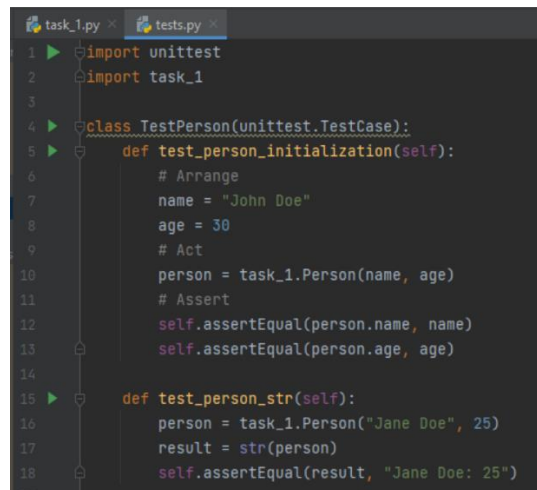
2. Act (Действие)

3. Assert (Проверка)

Суть FIRST заключается в 5 пунктах:

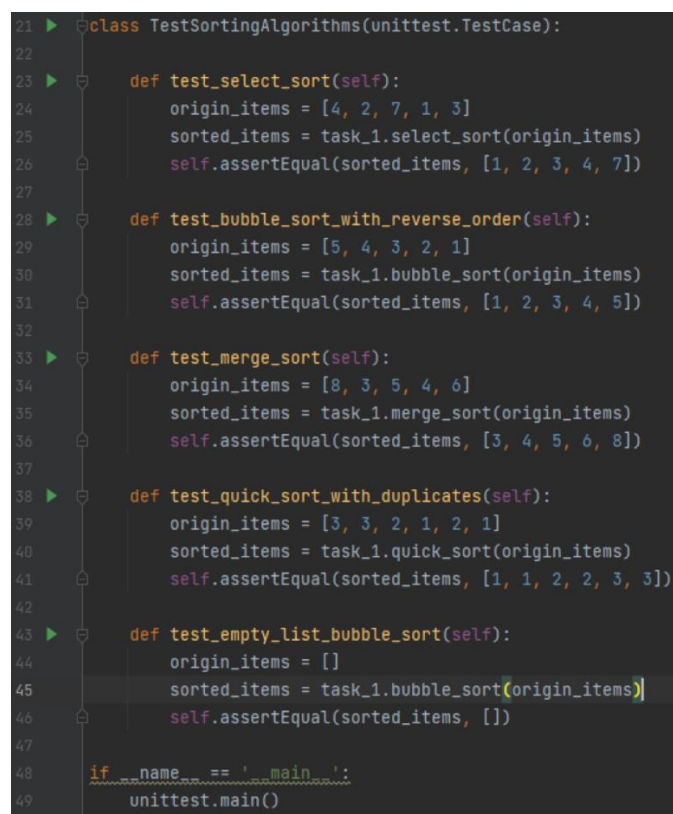
1. Fast (Быстрота)
2. Independent (Независимость)
3. Repeatable (Повторяемость)
4. Self-validating (Самопроверяемость)
5. Timely (Своевременность)

Все приведенные пункты были учтены при написании тестов. Код тестов приведен на рисунках 4 – 5.



```
1  import unittest
2  import task_1
3
4  class TestPerson(unittest.TestCase):
5      def test_person_initialization(self):
6          # Arrange
7          name = "John Doe"
8          age = 30
9          # Act
10         person = task_1.Person(name, age)
11         # Assert
12         self.assertEqual(person.name, name)
13         self.assertEqual(person.age, age)
14
15     def test_person_str(self):
16         person = task_1.Person("Jane Doe", 25)
17         result = str(person)
18         self.assertEqual(result, "Jane Doe: 25")
```

Рисунок 4 – Unit тесты для класса Person.



```
21 class TestSortingAlgorithms(unittest.TestCase):
22
23     def test_select_sort(self):
24         origin_items = [4, 2, 7, 1, 3]
25         sorted_items = task_1.select_sort(origin_items)
26         self.assertEqual(sorted_items, [1, 2, 3, 4, 7])
27
28     def test_bubble_sort_with_reverse_order(self):
29         origin_items = [5, 4, 3, 2, 1]
30         sorted_items = task_1.bubble_sort(origin_items)
31         self.assertEqual(sorted_items, [1, 2, 3, 4, 5])
32
33     def test_merge_sort(self):
34         origin_items = [8, 3, 5, 4, 6]
35         sorted_items = task_1.merge_sort(origin_items)
36         self.assertEqual(sorted_items, [3, 4, 5, 6, 8])
37
38     def test_quick_sort_with_duplicates(self):
39         origin_items = [3, 3, 2, 1, 2, 1]
40         sorted_items = task_1.quick_sort(origin_items)
41         self.assertEqual(sorted_items, [1, 1, 2, 2, 3, 3])
42
43     def test_empty_list_bubble_sort(self):
44         origin_items = []
45         sorted_items = task_1.bubble_sort(origin_items)
46         self.assertEqual(sorted_items, [])
47
48     if __name__ == '__main__':
49         unittest.main()
```

Рисунок 5 – Unit тесты для сортировок.

На примере функции `def test_person_initialization ()` продемонстрировано использование принципа AAA (остальные тесты построены аналогично).

На рисунке 6 результат работы. Все тесты пройдены, время 0,007s. Все пункты FIRST также выполнены: код работает достаточно быстро, тесты независимы друг от друга, при удалении одного из них, остальные будут работать, при перезапуске результаты не изменяются, тесты не требуют ручной проверки, а самостоятельно проверяются и выдают результат.

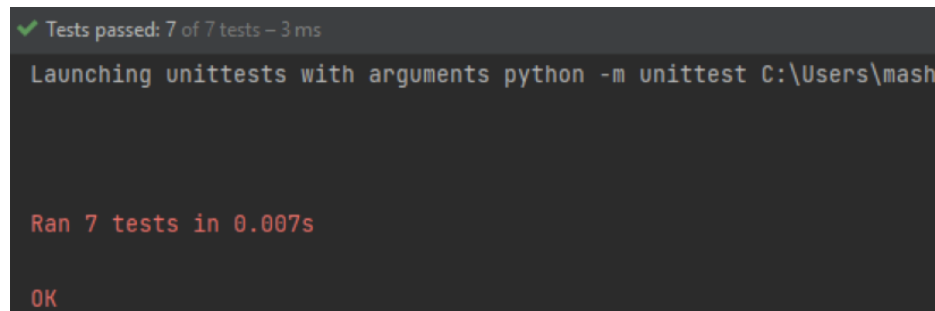
A screenshot of a terminal window with a dark background. At the top, a green checkmark is followed by the text "Tests passed: 7 of 7 tests - 3 ms". Below this, the command "Launching unittests with arguments python -m unittest C:\Users\mash..." is visible. Further down, the text "Ran 7 tests in 0.007s" is displayed in red. At the bottom left, the text "OK" is shown in red.

Рисунок 6 – Результат работы.

### **Вывод:**

Выполнены все задачи лабораторной работы, а именно:

- Выбран репозиторий с GitHub
- Проанализированы тестируемые функциональности
- Написаны тесты

Соответственно, достигнута цель работы – умение писать unit тесты.