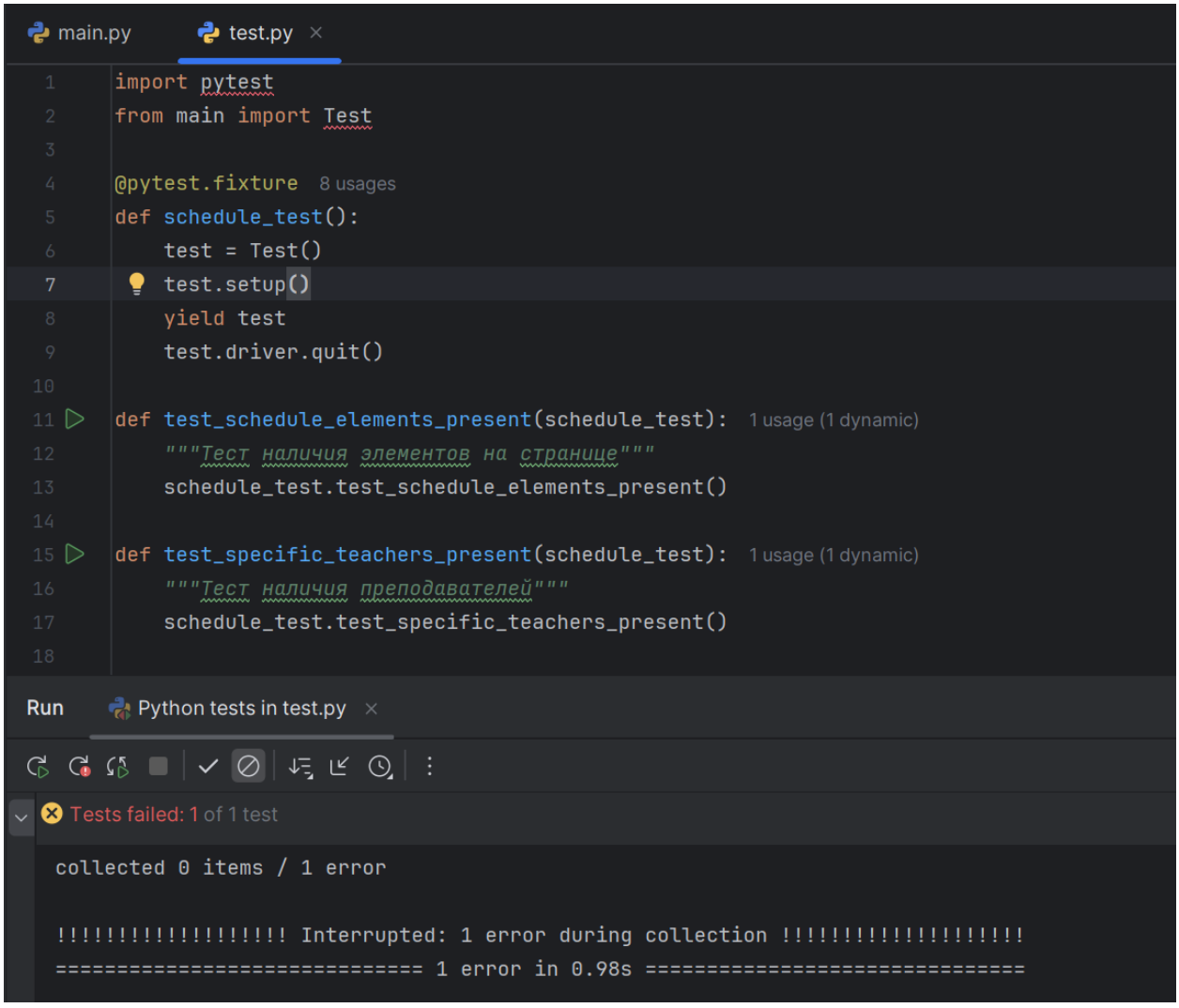
Практическая работа №4

Задание: получаем расписание на сайте https://api.nntu.ru/ и сверяем тестами pytest с заранее выбранным. Показываем что если заменяем свое расписание то тесты не проходят.

Запуск pytest с ошибкой:



Запуск pytest без ошибки:

Мы запускаем программу и после того как программа заработала, на выход нам предлагают выбрать формат обучения и группу. Мы вручную вводим формат обучения и номер группы. После чего программа дает нам наше расписание.

Ход программы:

1.Выбор формата обучения и номера группы:

```
Браузер запущен
Страница расписания открыта

ВВОД ФОРМЫ ОБУЧЕНИЯ И ГРУППЫ:
Введите форму обучения (например: 'Заочная', 'Очная' и т.д.): Заочная 3 года 10 мес
Введите название группы (например: 'АЗИС 22-1', 'ПИН 21-2' и т.д.): АЗИС 22-1
```

2.Ожидание поиска и загрузки расписания:

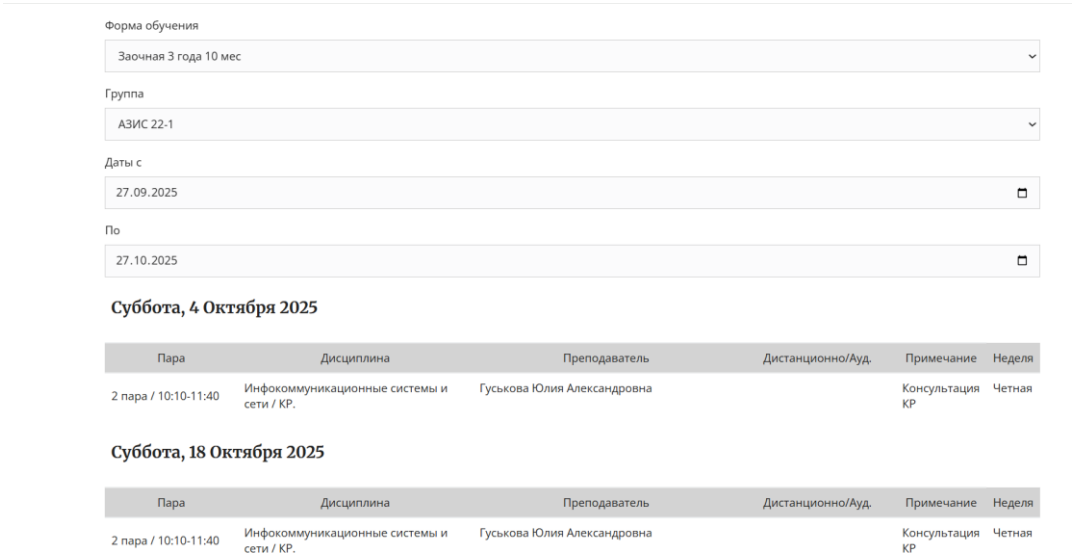
```
Ищем: Форма обучения - 'Заочная 3 года 10 мес', Группа - 'АЗИС 22-1'
Форма обучения выбрана
Группа выбрана

Загружаем расписание...
Кнопка нажата
```

3.Получаем полное расписание:

```
ПОЛНОЕ РАСПИСАНИЕ:
  1. Суббота, 4 Октября 2025
  2. Пара Дисциплина Преподаватель Дистанционно/Ауд. Примечание Неделя
  3. 2 пара / 10:10-11:40 Инфокоммуникационные системы и сети / КР. Гуськова Юлия Александровна Консультация КР Четная
  4. Суббота, 18 Октября 2025
  5. Пара Дисциплина Преподаватель Дистанционно/Ауд. Примечание Неделя
  6. 2 пара / 10:10-11:40 Инфокоммуникационные системы и сети / КР. Гуськова Юлия Александровна Консультация КР Четная

Всего строк: 6
Общий размер: 408 символов
```



Код программы:

```
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.action_chains import ActionChains


class UniversityScheduleTester:
    def __init__(self):
        self.setup_browser()

    def setup_browser(self):
        """Настройка браузера"""
        chrome_options = Options()
        chrome_options.add_argument("--start-maximized")
        chrome_options.add_argument("--disable-dev-shm-usage")
        chrome_options.add_argument("--disable-gpu")
        chrome_options.add_argument("--no-sandbox")
        chrome_options.add_argument("--disable-blink-
features=AutomationControlled")
        chrome_options.add_experimental_option("excludeSwitches", ["enable-
automation"])
        chrome_options.add_experimental_option('useAutomationExtension',
False)

        self.service = Service(ChromeDriverManager().install())
        self.driver = webdriver.Chrome(service=self.service,
options=chrome_options)
        self.driver.execute_script("Object.defineProperty(navigator, 'webdriver',
{get: () => undefined})")
        self.wait = WebDriverWait(self.driver, 20)
        self.actions = ActionChains(self.driver)
        print("Браузер запущен")
```

```
    def close_browser(self):
        """Закрытие браузера"""
        if hasattr(self, 'driver') and self.driver:
            self.driver.quit()
            print("Браузер закрыт")

    def open_schedule_page(self):
        """Открытие страницы расписания"""
        self.driver.get('https://api.nntu.ru/raspisanie')
        print("Страница расписания открыта")
        time.sleep(3)

    def scroll_to_element(self, element):
        """Прокрутка к элементу"""
        self.driver.execute_script("arguments[0].scrollIntoView({block: 'center',
behavior: 'smooth'});", element)
        time.sleep(0.5)

    def get_available_options(self, element_id):
        """Получение всех доступных опций из выпадающего списка"""
        try:
            select_element = self.wait.until(
                EC.presence_of_element_located((By.ID, element_id))
            )
            self.scroll_to_element(select_element)
            select = Select(select_element)
            options = []
            for option in select.options:
                if option.get_attribute("value") and option.get_attribute("value") !=
"null":
                    options.append({
                        'value': option.get_attribute("value"),
                        'text': option.text.strip(),
                        'visible': option.is_displayed()
                    })
            return options
        except Exception:
            return []

    def find_and_select_option(self, element_id, search_text, option_type):
        """Поиск и выбор опции по тексту"""
        try:
```

```
            options = self.get_available_options(element_id)
            if not options:
                print(f"Нет доступных {option_type}")
                return False

            # Ищем точное совпадение
            for option in options:
                if search_text.lower() in option['text'].lower():
                    return self.select_option_by_value(element_id, option['value'])

            # Если точное совпадение не найдено, ищем частичное
            for option in options:
                if any(word.lower() in option['text'].lower() for word in
search_text.split() if len(word) > 2):
                    print(f"Найдено приблизительное совпадение: {option['text']}")
                    return self.select_option_by_value(element_id, option['value'])

            print(f"{option_type} '{search_text}' не найдена")
            print(f"Доступные {option_type}:")
            for option in options[:10]:  # Показываем первые 10 вариантов
                print(f"  - {option['text']}")
            if len(options) > 10:
                print(f"  ... и еще {len(options) - 10} вариантов")
            return False

        except Exception as e:
            print(f"Ошибка при поиске {option_type}: {e}")
            return False

    def select_option_by_value(self, element_id, value):
        """Выбор опции по значению с обработкой исключений"""
        try:
            select_element = self.wait.until(
                EC.element_to_be_clickable((By.ID, element_id))
            )
            self.scroll_to_element(select_element)

            time.sleep(1)

            self.driver.execute_script(f"""
                var select = document.getElementById('{element_id}');
                if (select) {{
```

```
                select.value = '{value}';
                var event = new Event('change', {{ bubbles: true }});
                select.dispatchEvent(event);
            }}
            """)

            time.sleep(1)
            current_value = self.driver.execute_script(f"""
                return document.getElementById('{element_id}').value;
            """)

            if current_value == value:
                return True
            else:
                try:
                    select = Select(select_element)
                    select.select_by_value(value)
                    return True
                except:
                    return False

        except Exception:
            return False

    def manual_input_selection(self):
        """Ручной ввод формы обучения и группы"""
        print("\nВВОД ФОРМЫ ОБУЧЕНИЯ И ГРУППЫ:")

        # Запрашиваем форму обучения
        while True:
            department_input = input("Введите форму обучения (например:
'Заочная', 'Очная' и т.д.): ").strip()
            if department_input:
                break
            print("Форма обучения не может быть пустой")

        # Запрашиваем группу
        while True:
            group_input = input("Введите название группы (например: 'АЗИС
22-1', 'ПИН 21-2' и т.д.): ").strip()
            if group_input:
                break
```

```
            print("Название группы не может быть пустым")

        print(f"\nИщем: Форма обучения - '{department_input}', Группа -
'{group_input}'")

        # Ищем и выбираем форму обучения
        if self.find_and_select_option("studentAdvert__controls--department",
department_input, "формы обучения"):
            print(f"Форма обучения выбрана")
        else:
            print("Не удалось найти форму обучения")
            return False

        time.sleep(3)

        # Ищем и выбираем группу
        if self.find_and_select_option("studentAdvert__controls--groups",
group_input, "группы"):
            print(f"Группа выбрана")
        else:
            print("Не удалось найти группу")
            return False

        time.sleep(3)
        return True

    def show_schedule(self):
        """Показать расписание"""
        try:
            show_button = self.wait.until(
                EC.element_to_be_clickable((By.CSS_SELECTOR, "button.btn-
primary"))
            )

            self.scroll_to_element(show_button)
            self.actions.move_to_element(show_button).click().perform()
            time.sleep(1)
            show_button.click()

            print("Нажата кнопка 'Показать расписание'")
            time.sleep(5)
            return True
```

```
        except Exception:
            try:
                self.driver.execute_script("""
                    var buttons = document.querySelectorAll('button.btn-primary');
                    for (var i = 0; i < buttons.length; i++) {
                        if (buttons[i].textContent.includes('Показать')) {
                            buttons[i].click();
                            break;
                        }
                    }
                """)
                print("Кнопка нажата")
                time.sleep(5)
                return True
            except:
                return False

    def get_full_schedule(self):
        """Получение полного текста расписания"""
        try:
            self.wait.until(
                EC.presence_of_element_located((By.ID, "printable"))
            )

            time.sleep(3)

            # Получаем весь текст из блока printable
            printable = self.driver.find_element(By.ID, "printable")
            full_schedule = printable.text

            # Если текст короткий, пробуем получить через JavaScript
            if len(full_schedule) < 100:
                full_schedule = self.driver.execute_script("""
                    return document.getElementById('printable').innerText;
                """)

            return full_schedule

        except Exception:
            try:
                # Пробуем найти таблицу расписания
```

```
            tables = self.driver.find_elements(By.TAG_NAME, "table")
            if tables:
                return tables[0].text
        except:
            pass

        # Пробуем получить весь текст страницы
        try:
            return self.driver.find_element(By.TAG_NAME, "body").text
        except:
            return "Не удалось получить расписание"

    def display_schedule(self, schedule_text):
        """Отображение полного расписания"""
        if not schedule_text or len(schedule_text.strip()) < 50:
            print("Расписание не найдено или слишком короткое")
            return False

        print("\nПОЛНОЕ РАСПИСАНИЕ:")

        # Разделяем расписание на строки и выводим с нумерацией
        lines = schedule_text.split('\n')
        for i, line in enumerate(lines, 1):
            if line.strip():  # Пропускаем пустые строки
                print(f"{i:3d}. {line}")

        print(f"\nВсего строк: {len([l for l in lines if l.strip()])}")
        print(f"Общий размер: {len(schedule_text)} символов")

        return True


def main():
    """Основная функция"""
    tester = UniversityScheduleTester()

    try:
        # Открываем страницу
        tester.open_schedule_page()

        # Ручной ввод параметров
        if not tester.manual_input_selection():
```

```
            print("Не удалось выбрать параметры. Завершение работы.")
            return

        # Показываем расписание
        print("\nЗагружаем расписание...")
        if tester.show_schedule():
            # Получаем полное расписание
            schedule_text = tester.get_full_schedule()

            # Отображаем полное расписание
            tester.display_schedule(schedule_text)

        input("\nНажмите Enter для выхода...")

    except Exception as e:
        print(f"Произошла ошибка: {e}")
        import traceback
        traceback.print_exc()
    except KeyboardInterrupt:
        print("Программа прервана пользователем")
    finally:
        tester.close_browser()


if __name__ == "__main__":
    main()
```