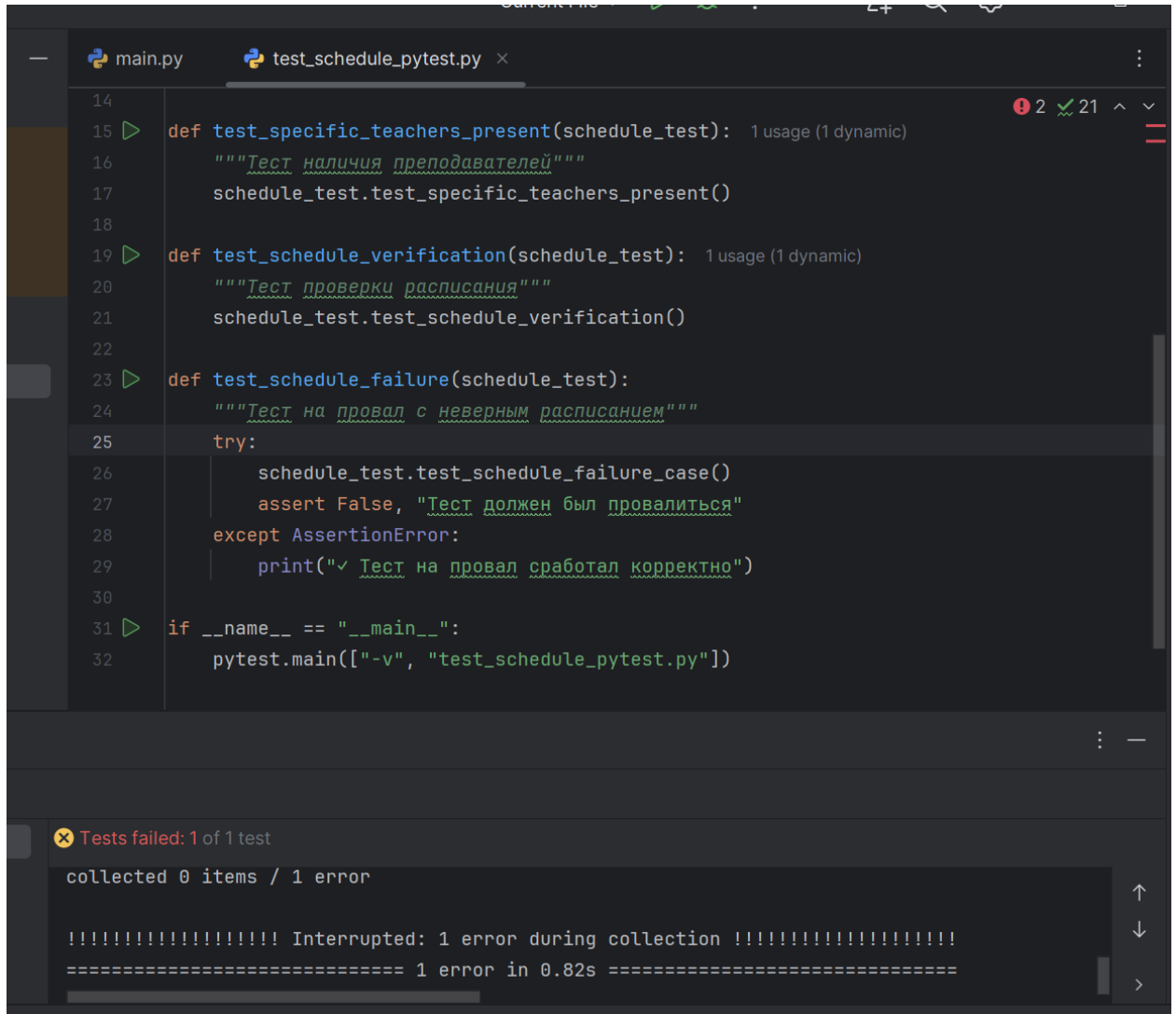


Практическая работа №4

Запуск pytest с ошибкой:



```
14
15 > def test_specific_teachers_present(schedule_test): 1 usage (1 dynamic)
16     """Тест наличия преподавателей"""
17     schedule_test.test_specific_teachers_present()
18
19 > def test_schedule_verification(schedule_test): 1 usage (1 dynamic)
20     """Тест проверки расписания"""
21     schedule_test.test_schedule_verification()
22
23 > def test_schedule_failure(schedule_test):
24     """Тест на провал с неверным расписанием"""
25     try:
26         schedule_test.test_schedule_failure_case()
27         assert False, "Тест должен был провалиться"
28     except AssertionError:
29         print("✓ Тест на провал сработал корректно")
30
31 > if __name__ == "__main__":
32     pytest.main(["-v", "test_schedule_pytest.py"])
```

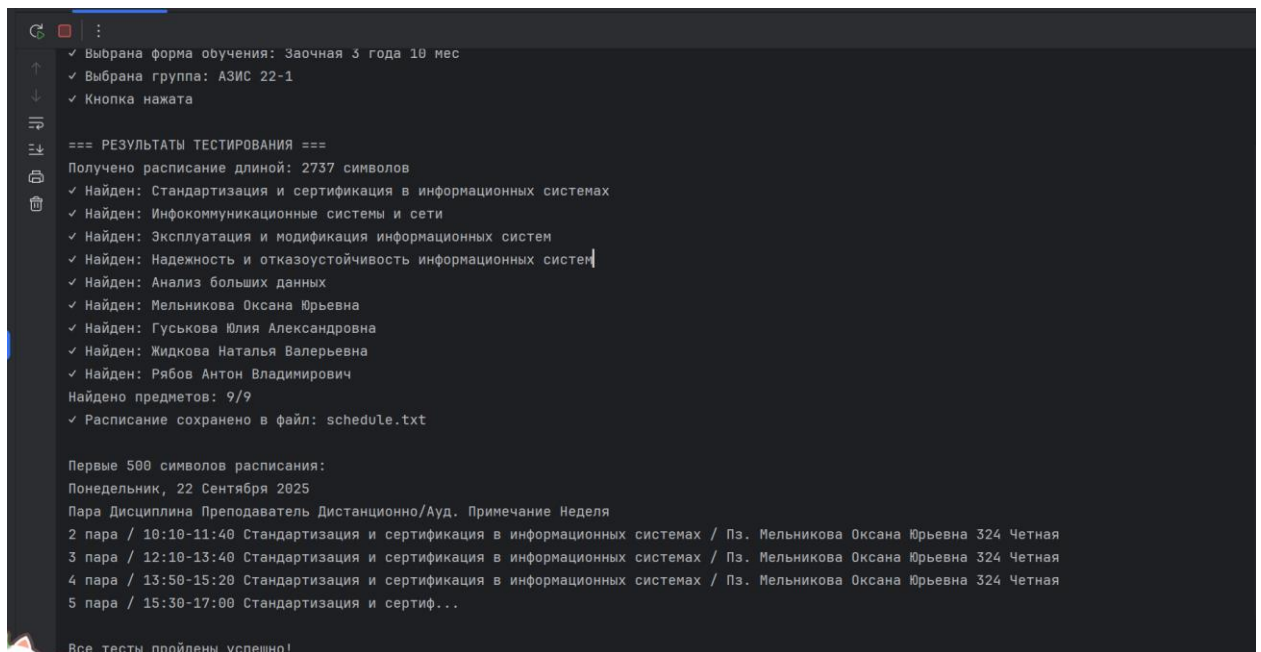
⊗ Tests failed: 1 of 1 test

collected 0 items / 1 error

!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!!!

===== 1 error in 0.82s =====

Запуск pytest без ошибки:



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
from datetime import datetime

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 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 automatic_selection(self):
    """Автоматический выбор заочной формы обучения и группы АЗИС 22-1"""
    print("\n=== АВТОМАТИЧЕСКИЙ ВЫБОР ПАРАМЕТРОВ ===")

    time.sleep(2)

    departments = self.get_available_options("studentAdvert__controls--department")

    if not departments:
        print("Нет доступных форм обучения")
        return None, None, None

    selected_dept = None
    for dept in departments:
        if "заочная" in dept['text'].lower() and "очно-заочная" not in dept['text'].lower():
            selected_dept = dept
            break

    if not selected_dept:
        selected_dept = departments[0]

    if self.select_option_by_value("studentAdvert__controls--department", selected_dept['value']):
        print(f"✓ Выбрана форма обучения: {selected_dept['text']}")
    else:
        print(f"Не удалось выбрать форму обучения: {selected_dept['text']}")
        return None, None, None

    time.sleep(3)

```

```

groups = self.get_available_options("studentAdvert__controls--groups")

if not groups:
    print("Нет доступных групп после выбора формы обучения")
    time.sleep(2)
    groups = self.get_available_options("studentAdvert__controls--groups")
    if not groups:
        return selected_dept, None, None

selected_group = None
for group in groups:
    if "АЗИС 22-1" in group['text']:
        selected_group = group
        break

if not selected_group:
    selected_group = groups[0]
    print(f'Группа АЗИС 22-1 не найдена, выбрана первая доступная: {selected_group["text"]}')

if self.select_option_by_value("studentAdvert__controls--groups",
selected_group['value']):
    print(f'✓ Выбрана группа: {selected_group["text"]}')
else:
    print(f'Не удалось выбрать группу: {selected_group["text"]}')
    return selected_dept, None, None

time.sleep(3)

types = self.get_available_options("studentAdvert__controls--types")

selected_type = None
if types and len(types) > 0:
    for type_option in types:
        if "занятия" in type_option['text'].lower() or "расписание" in
type_option['text'].lower():
            selected_type = type_option
            break

if not selected_type and len(types) > 0:
    selected_type = types[0]

```

```

        if selected_type and self.select_option_by_value("studentAdvert__controls-
        -types", selected_type['value']):
            print(f"✓ Выбран тип расписания: {selected_type['text']}")
        else:
            selected_type = None
    else:
        print("✓ Тип расписания не требуется")

    time.sleep(2)
    return selected_dept, selected_group, selected_type

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

        time.sleep(2)

        printable = self.driver.find_element(By.ID, "printable")
        schedule_text = printable.text

        if len(schedule_text) < 100:
            schedule_text = self.driver.execute_script("""
                return document.getElementById('printable').innerText;
            """)

        return schedule_text

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

def test_schedule(self, expected_subjects):
    """Тестирование расписания"""
    schedule_text = self.get_schedule_text()

    if not schedule_text:
        print("Не удалось получить расписание")
        return False

    print("\n=== РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ===")

```



```

print(f'Получено расписание длиной: {len(schedule_text)} символов')

normalized_schedule = ' '.join(schedule_text.lower().split())

success = True
found_count = 0

for subject in expected_subjects:
    normalized_subject = ' '.join(subject.lower().split())

    if normalized_subject in normalized_schedule:
        print(f'✓ Найден: {subject}')
        found_count += 1
    else:
        found = False
        words = normalized_subject.split()
        if len(words) > 2:
            key_words = [word for word in words if len(word) > 4]
            if all(any(key_word in schedule_word for schedule_word in
normalized_schedule.split()) for key_word
in key_words):
                print(f'✓ Найден (по ключевым словам): {subject}')
                found_count += 1
                found = True

        if not found:
            print(f'Не найден: {subject}')
            success = False

print(f'Найдено предметов: {found_count}/{len(expected_subjects)}')
return success

def save_schedule_to_file(self, filename="schedule.txt"):
    """Сохранение расписания в файл"""
    schedule_text = self.get_schedule_text()
    try:
        with open(filename, 'w', encoding='utf-8') as f:
            f.write(f'Расписание получено: {datetime.now().strftime("%Y-%m-%d
%H:%M:%S")}\n')
            f.write("=" * 50 + "\n")
            f.write(schedule_text)

```

```

        print(f'✓ Расписание сохранено в файл: {filename}')
        return schedule_text
    except Exception:
        return schedule_text

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

    try:
        tester.open_schedule_page()

        dept, group, schedule_type = tester.automatic_selection()

        if not group:
            print("Не удалось выбрать группу. Завершение работы.")
            return

        if tester.show_schedule():
            expected_subjects = [
                "Стандартизация и сертификация в информационных системах",
                "Инфокоммуникационные системы и сети",
                "Эксплуатация и модификация информационных систем",
                "Надежность и отказоустойчивость информационных систем",
                "Анализ больших данных",
                "Мельникова Оксана Юрьевна",
                "Гуськова Юлия Александровна",
                "Жидкова Наталья Валерьевна",
                "Рябов Антон Владимирович"
            ]

            success = tester.test_schedule(expected_subjects)

            schedule_text = tester.save_schedule_to_file()

            if schedule_text:
                print(f'\nПервые 500 символов расписания:\n{schedule_text[:500]}...')

            if success:

```

```
        print("\nВсе тесты пройдены успешно!")
    else:
        print("\n Некоторые предметы не найдены в расписании")

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

except Exception as e:
    print(f'Произошла ошибка: {e}')
    import traceback
    traceback.print_exc()
finally:
    tester.close_browser()

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