



Урок 32. Реализация и автоматические тесты

Часть 1. Материалы ученика

Программирование на Python и C++ · 9 класс

Урок 32 из 34 · 45 минут · Модуль «Итоговый марафон»

Уровень: начинающий программист; Python 3.

Программа: <https://schoolotzyv.ru/blog/python-cpp-9-klass-programma>

Если программа прошла один красивый пример, это ещё не доказывает обработку пустоты, порога и неправильного файла. Проверим отдельно разбор и расчёт.

Результат урока: Собрать файловый конвейер и проверить его независимыми ожидаемыми результатами, включая ошибки.

01. Проблемная ситуация

0–5 минут. Ошибочная проверка `count=sum(x>20 for x in a)` проходит пример `[25,30]`. Придумайте один минимальный тест, который её разоблачит, и запишите ожидаемый ответ.

02. Понятия и разобранный пример

5–14 минут: 4 минуты на понятия, 3 на пример, 2 на объяснение способа партнёру.

Разделяем разбор и расчёт

Сохраняем контракт: UTF-8 data.txt, 0...100 ASCII-целых 0...1000, разделение пробелами или переводами строк; q=20 минут. Отчёт: n, сумма, количество ≥ 20 на первой строке; отсортированные значения на второй. Пустой файл разрешён. Неверный токен или диапазон вызывает отказ целого файла.

parse отвечает только за формат и диапазон; report получает проверенные числа. Так тест вычисления не зависит от файловой системы. Ожидаемые значения в проверке задаются отдельно вручную. Нельзя проверять `report(a,q)==report(a,q)`: одна и та же ошибка даст одинаковые результаты.

Код полностью приведён ниже. Создайте data.txt в рабочей папке и сохраните исходник как lesson.py. При ERROR новый result.txt не записывается. Для каждого ошибочного файлового теста используйте чистую тестовую папку. assert служит учебным тестом; запускайте без python -O, иначе проверки отключатся.

Отступы набирайте обычными пробелами: 4 на уровень. При копировании с веб-страницы замените неразрывные пробелы обычными; полный код с обычными пробелами есть в PDF.

Для `20 0 3` parse возвращает `[20,0,3]`, report — `(3,23,1,[0,3,20])`. Файловый результат: `3 23 1` и `0 3 20`. Для `20 -1` разбор вызывает ValueError до вычисления и записи.

```
def parse(text):
    tokens = text.split()
    if any(not (t.isascii() and t.isdecimal()) for t in tokens):
        raise ValueError("integer tokens required")
    a = list(map(int, tokens))
    if len(a) > 100 or any(x > 1000 for x in a):
        raise ValueError("range")
    return a

def report(a, q):
    b = sorted(a)
    count = sum(x >= q for x in a)
    return len(a), sum(a), count, b

from pathlib import Path

try:
    a = parse(Path("data.txt").read_text(encoding="utf-8"))
    n, total, count, b = report(a, 20)
    text = f"{n} {total} {count}\n" + " ".join(map(str, b)) + "\n"
    Path("result.txt").write_text(text, encoding="utf-8")
    print(text, end="")
except (OSError, ValueError) as error:
    print("ERROR:", error)
```



parse	Текст → проверенные числа
report	Числа → четыре результата
write	Две строки result.txt
Тест	Вход → ручной эталон → сравнение

Разделение ответственности делает ошибку локализуемой.

03. Практика 1. Проверяем материал

Запускаем приёмочные тесты

14–25 минут. Сначала сохраните готовый исходник и выполните обычный файловый запуск. Затем добавьте в конец файла приведённые проверки.

Текст входа	Ожидаемый результат ядра или ошибка
20 0 3	(3,23,1,[0,3,20])
Пустая строка	(0,0,0,[])
20 20	(2,40,2,[20,20])
2.5; -1; 1001	Каждый отдельно: ValueError

2 минуты — прогноз; 4 — запуск семи проверок; 3 — два файловых запуска с data.txt «20 0 3» и «2.5» в отдельных папках; 2 — журнал. Критерии: семь тестов пройдены, две строки верны, при плохом вводе ERROR и новый result.txt отсутствует.

```
assert report(parse("20 0 3"),20)==(3,23,1,[0,3,20])
assert report(parse(""),20)==(0,0,0,[])
assert report(parse("20 20"),20)==(2,40,2,[20,20])
for bad in ["2.5", "-1", "1001", "1"*101]:
    try:
        parse(bad)
    except ValueError:
        pass
    else:
        raise AssertionError(bad)
print("7 checks passed")
```

04. Практика 2. Перенос способа

Ловим ошибку порога

25–33 минуты. В копии report замените `>=` на `>`. За 2 минуты предскажите первый провал, за 3 запустите тесты, за 3 верните исправление и повторите весь набор. Результат: журнал «дефект / тест / ожидается / получено / после исправления».

05. Обсуждение и границы вывода

33–40 минут. Все семь тестов прошли. Можно ли объявить программу правильной для любых данных? За 2 минуты назовите границы контракта, за 3 предложите ещё два теста, за 2 отделите проверенное от недоказанного.

06. Индивидуальная проверка

40–45 минут. Напишите assert для текста «19 20 21», объясните независимость эталона и назовите ошибочный файл. Четыре балла.



Часть 2. Ответы и материалы учителя

Практика 1: разбор и критерии

Вывод тестов: 7 checks passed. Обычный отчёт: 3 23 1 и 0 3 20. В ошибочной папке результат не создаётся. Непойманный `AssertionError` означает провал теста, а не ожидаемую ошибку данных. Отсутствующий `data.txt` также даёт `ERROR` через `OSError`.

Практика 2: разбор и критерии

Первый тест ожидает `count=1` для `[20,0,3]`, ошибочный код получает 0. Тест `[20,20]` также различает 2 и 0. После восстановления `>=` все семь проходят. Тест пустоты не обнаруживает этот дефект, но проверяет другой риск. Критерии: локализован знак, предъявлен контрпример, повторён весь набор.

Обсуждение: возможный вывод

Пройдены только выбранные случаи. Дополнительные тесты: ровно 100 значений, токен 00020, табуляция и переводы строк, отсутствие файла. Вне контракта остаются вещественные и отрицательные числа. Тестирование повышает уверенность, но конечный набор не является доказательством для всех входов.

Индивидуальная проверка: ответ и критерии

`assert report(parse("19 20 21"),20)==(3,60,2,[19,20,21])`. Баллы: правильные `n` и сумма; `count` и порядок; эталон получен вручную, не той же функцией; пример неверного файла, например 19,20 с запятой.

Учителю: подготовка и план на 45 минут

Подготовьте компьютер, редактор, папку с правом записи, распечатку ученической части PDF, бумагу и ручку. Python 3.11+ и IDLE или редактор; запуск `python lesson.py` (либо `python3 lesson.py`). Внешние библиотеки не нужны. Установку и проверку запуска учитель выполняет заранее. Все программы, данные и ожидаемые ответы даны; регистрация детей и интернет на занятии не нужны.

Уровень — начинающий программист, 9 класс. Предпосылки: контракт проекта урока 31, функции, исключения, файлы. В паре автор прогнозирует и вводит код, проверяющий ведёт журнал, затем роли меняются. Без компьютера выполните трассировку полного кода на бумаге и сопоставьте с готовыми результатами; учитель демонстрирует подготовленный запуск. Бумажный вариант не считается фактическим запуском ученика. Перед практикой учитель выдаёт готовый файл разобранного примера, чтобы дети тратили время на изменение и проверку, а не длинное перепечатывание. Матрица программы предназначена учителю.

Минуты	Работа	Результат
0–5	Проблема	Вопрос и первое решение
5–14	Понятия и пример	Способ проверки
14–25	Практика 1	Готовый продукт и проверка
25–33	Практика 2	Перенос способа
33–40	Обсуждение	Аргумент и ограничение
40–45	Индивидуальная проверка	Самостоятельный ответ



07. Соответствие исходной программе

Пункт программы	Действие ученика	Проверяемый результат
Повторение и обобщение: файлы, функции, сортировка, поиск	Запускаем приёмочные тесты	Собрать файловый конвейер и проверить его независимыми ожидаемыми результатами, включая ошибки.
Самостоятельная проверка результата	Ловим ошибку порога	Обоснованный ответ и исправление ошибки

Источники и права на изображения

Исходная программа курса

<https://schoolotzyv.ru/blog/python-cpp-9-klass-programma>

Python: автоматические проверки результатов

<https://docs.python.org/3/library/unittest.html>

Тексты, учебные данные, реплики, программы и схемы подготовлены для уроков портала «Школы России». Числовые серии — учебные модели, не результаты проведённых классом опытов. Схемы авторские, сторонние фотографии не используются. Методические ссылки приведены для учителя; ученик получает все материалы здесь. Проверка первичных источников: 18.10.2026.