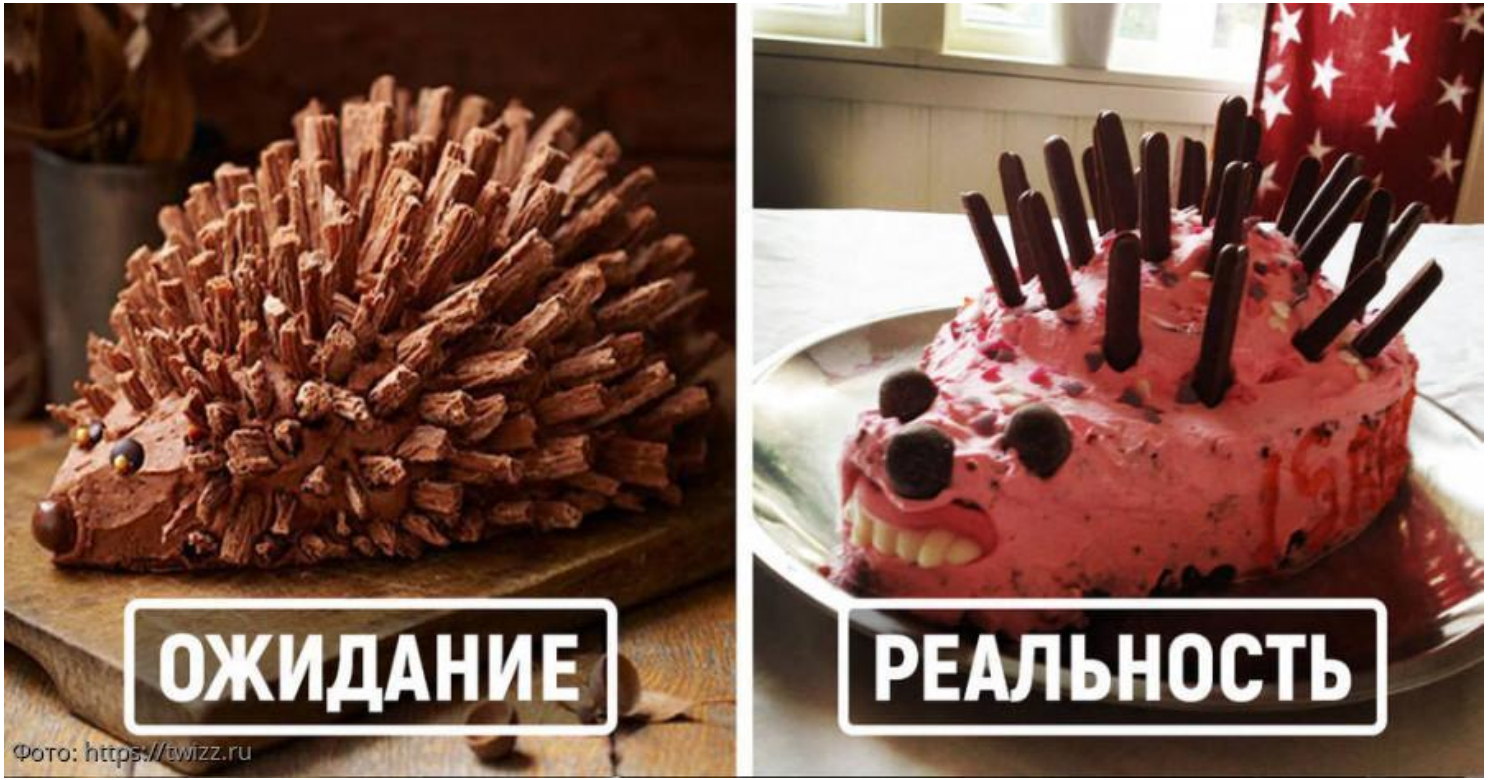
Тестирование и дебаг в JS

Айтишный АБФ

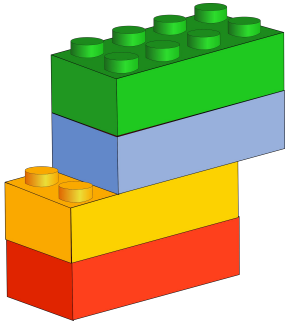
Ожидание и Реальность



Ожидание и Реальность

С чего начать?

- Пишите **модульный** код



- // Комментируйте все детали. Особенно ограничения на входные данные и ожидания от выходных

- На основе этой документации можно написать **тест кейсы**

Перед тестированием

- Убедитесь, что код вообще запускается. Не должно быть синтаксических ошибок
- Составьте таблицу из входных данных и ожидаемых ответов.

Вводная строка	Валидное слово?
adajglk	false
labas	true
ліхтар	true

Тестирование

- **Unit Tests** - программа разбивается на модули и каждый из них тестируется отдельно.
- **Регрессивное тестирование** - допустим нашелся баг, при каких условиях он возникает? После того как баг исправлен, тетстируем программу на тех же условиях, чтобы убедиться, что его нет.
- **Integration testing** - Если предыдущие тесты пройдены, тестируем "сразу всё".

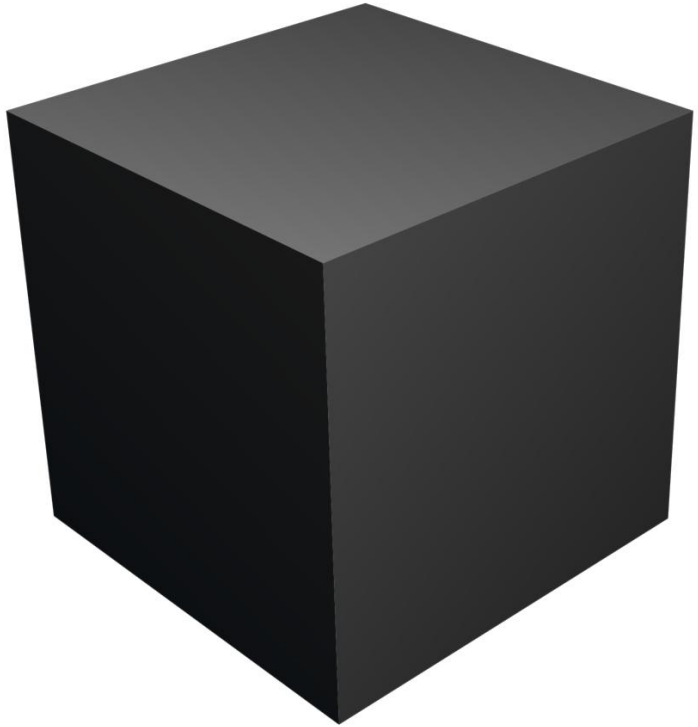
(и так пока все тесты не будут пройдены)

Подходы к тестированию

Черный ящик

- Реализация сокрыта
- Можем задать входные данные
- И проверить, что на выходе

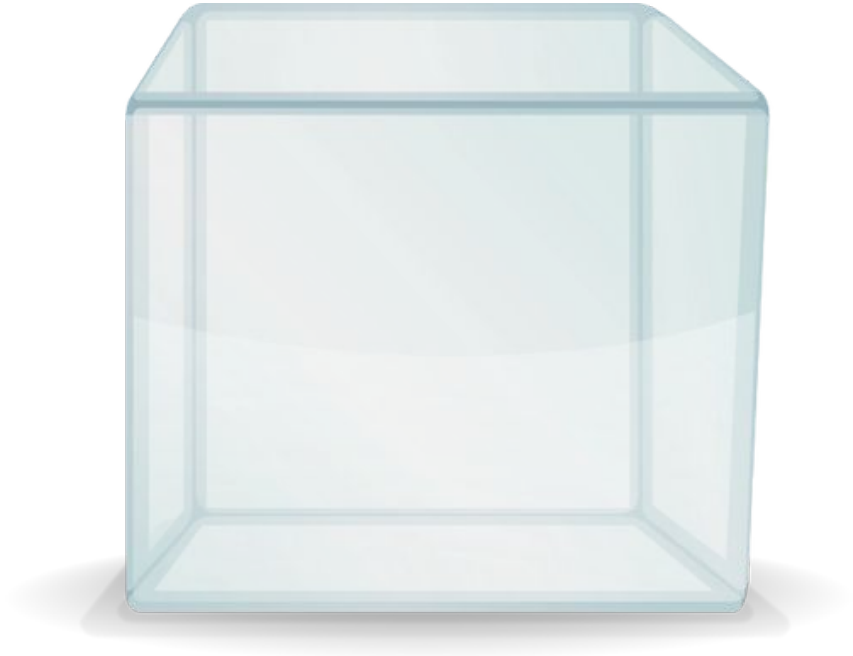
Обычно проверяем нормальное использование + граничные условия



Подходы к тестированию

Белый ящик

- Реализация открыта
- Уделяем внимание ветвлению и control flow.
- "Полнопроходное" тестирование



Подходы к тестированию

Белый ящик

```
function abs(num) {
    if (num < -1) {
        return -num;
    } else {
        return num;
    }
}

abs(-2); // 2
abs(2); // 2
```

Полнопроходное тестирование + граничные условия

Задача

Функция len() принимает на входе массив, а на выходе возвращает количество элементов в нем.

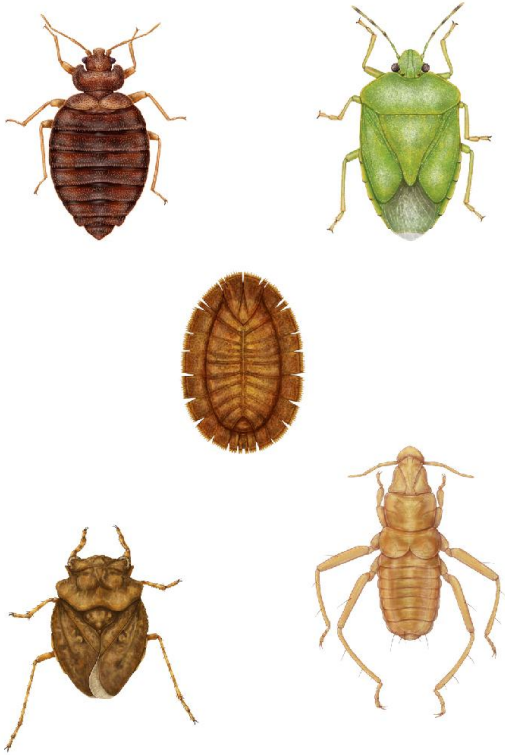
- Какие граничные условия подобрать для тестирования?
- Перечислите входные данные и ожидаемые ответы.

Задача

```
function max(a, b, c) {
    let maximum = a;
    if (b > maximum) {
        maximum = b;
    }
    if (c > maximum) {
        maximum = c;
    }
    return maximum;
}
```

- Как за минимум проходов протестировать функцию max()?

Типы багов



Типы багов






- Явные (крэши) и скрытые
- Повторяющиеся и редко воспроизводимые

Если баг появляется редко, то задача тестировщика - отловить условия, при которых он возникает

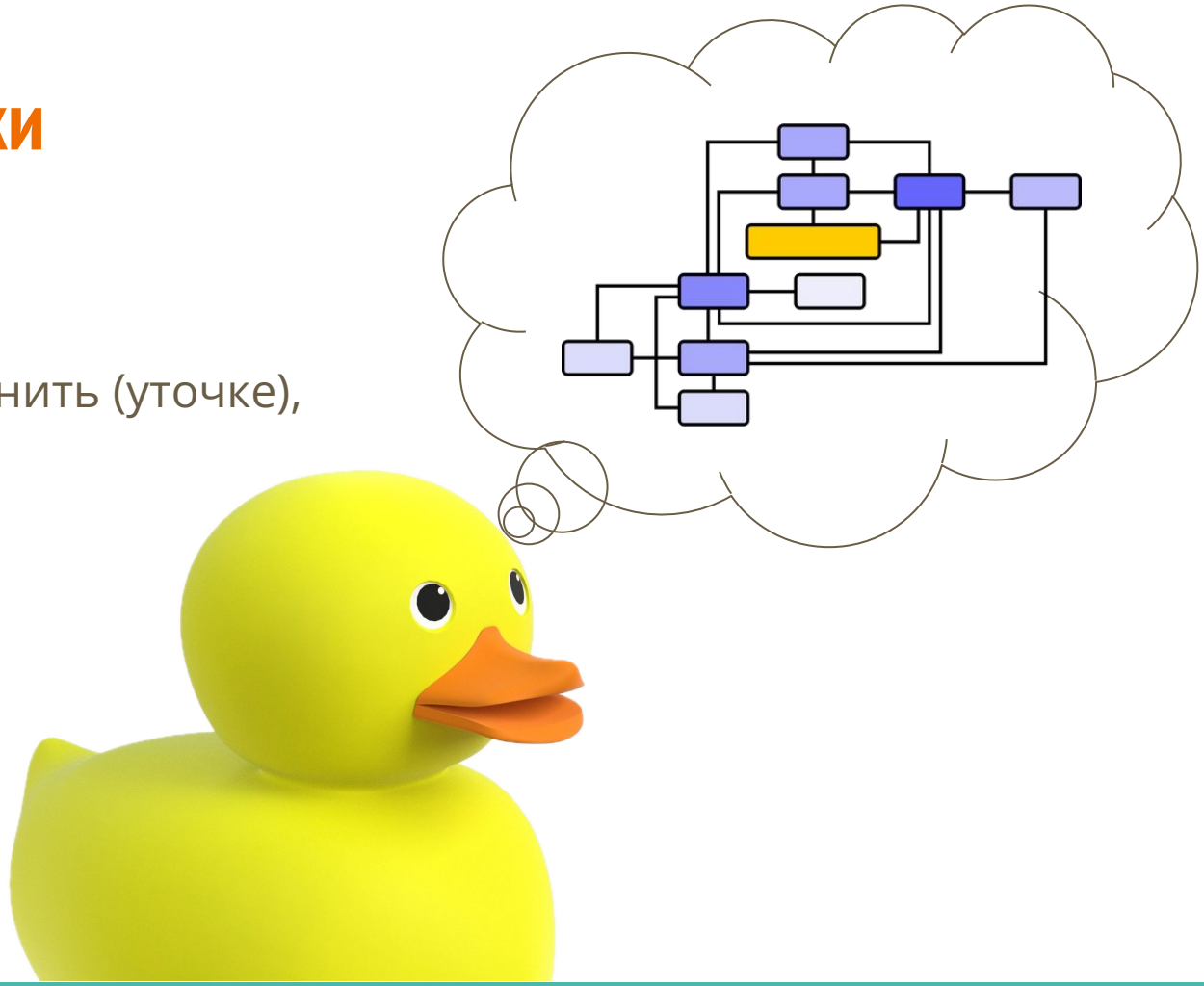
Защитное программирование

- Явные (крэши) и скрытые
- Повторяющиеся и редко воспроизводимые

Стараемся делать все баги **явными**

Логические ошибки

- Их сложно найти
- Нарисуйте схему
- Постарайтесь объяснить (уточке), что делает код



Как думать?

В процессе тестирования:

- Подумайте, как вы дошли до такого
- Применяйте научный метод:
  - гипотеза
  - тест кейс, который ее проверит
  - исправьте код
  - тестируйте снова!

Задача

```
function rem(x, a) {
    /**
     * x: a non-negative integer argument
     * a: a positive integer argument
     *
     * returns: integer, the remainder
     * when x is divided by a.
     */
    if (x === a) {
        return 0;
    } else if (x < a) {
        return x;
    } else {
        rem(x - a, a);
    }
}
```

rem(2, 5) - возвращает 2

rem(5, 5) - возвращает 0

но rem(7, 5) - возвращает 0!

- В какой строке ошибка?

Задача

```
function factorial(n) {
    /**
     * n: integer, n >= 0.
     */
    if (n === 0) {
      return n;
    } else {
      return n * f(n-1);
    }
}
```

factorial(3) - ожидаем 6, возвращает 0

factorial(1) - ожидаем 1, возвращает 0

factorial(0) - ожидаем 1, возвращает 0

- В какой строке ошибка?