SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

Още малко преговор

- Какво е `hoisting` ?
- Какво връщат функциите без `return`?
- Как работи тройния оператор `?`
- Кои типове данни са референтни ?
- `typeof NaN` ?
- Какво е `IIFE` ?

Още малко преговор

- Как сравняваме обекти?
- Какво ще изпише конзолата, ако изпълня следния код:

```
function add(list, item) {
  list.push(item);
  return list;
}
var months = ['May', 'June', 'July'];
var summerMonths = add(months, 'August');
summerMonths.splice(0,1);

console.log("months:", months);
console.log("summerMonths:", summerMonths);
```

Копиране на обекти

- Миналият път си говорихме как можем да сравняваме обекти с функцията `JSON.stringify` (защо?)
- Ако искаме физически да копираме един обект (за да не го предаваме по референция), правим следното:

```
var copy =
  JSON.parse(JSON.stringify(objToCopy));
```

- Това, което прави `JSON.parse`, е да обръща текстово представяне на JS обект, в нов JS обект (или примитив)

Примери

- `JSON.parse(JSON.stringify(4)); // => 4`
- `JSON.parse(JSON.stringify("Hello")); // => "Hello"`
- `JSON.parse(JSON.stringify(["a", "s", "d"]));`
  `// => ["a", "s", "d"]`

```
var obj = {
  firstName: "Douglas",
  lastName: "Crockford"
};

JSON.parse(JSON.stringify(obj));
  // => { firstName: "Douglas", lastName: "Crockford" };
```

Дебъгване на код

Six Stages of Debugging

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

http://www.sanettetanaka.com/thoughts/2015/10/1/debug-your-code-the-easy-way

Кога трябва да дебъгваме

- Бъг наричаме всичко, което не е част от спецификацията (изискванията) и нормалното поведение на дадено приложение/уебсайт
- Всеки път когато нещо не работи и не можем да разберем защо, трябва да дебъгваме кода си
- Дебъгването е процес на прследяване на начина, по който браузъра интерпретира кода ни, с цел откриването на причината за бъг-а
- **Важно**: Писането на добре структуриран код, намалява шансовете за поява на много бъгове и улеснява дебъгването

Как да дебъгваме

Въпреки, че превенцията е най-силното ни оръжие срещу бъговете (т.е. да пишем добре организиран и чист код), рано или късно ни се налага да дебъгваме кода си.

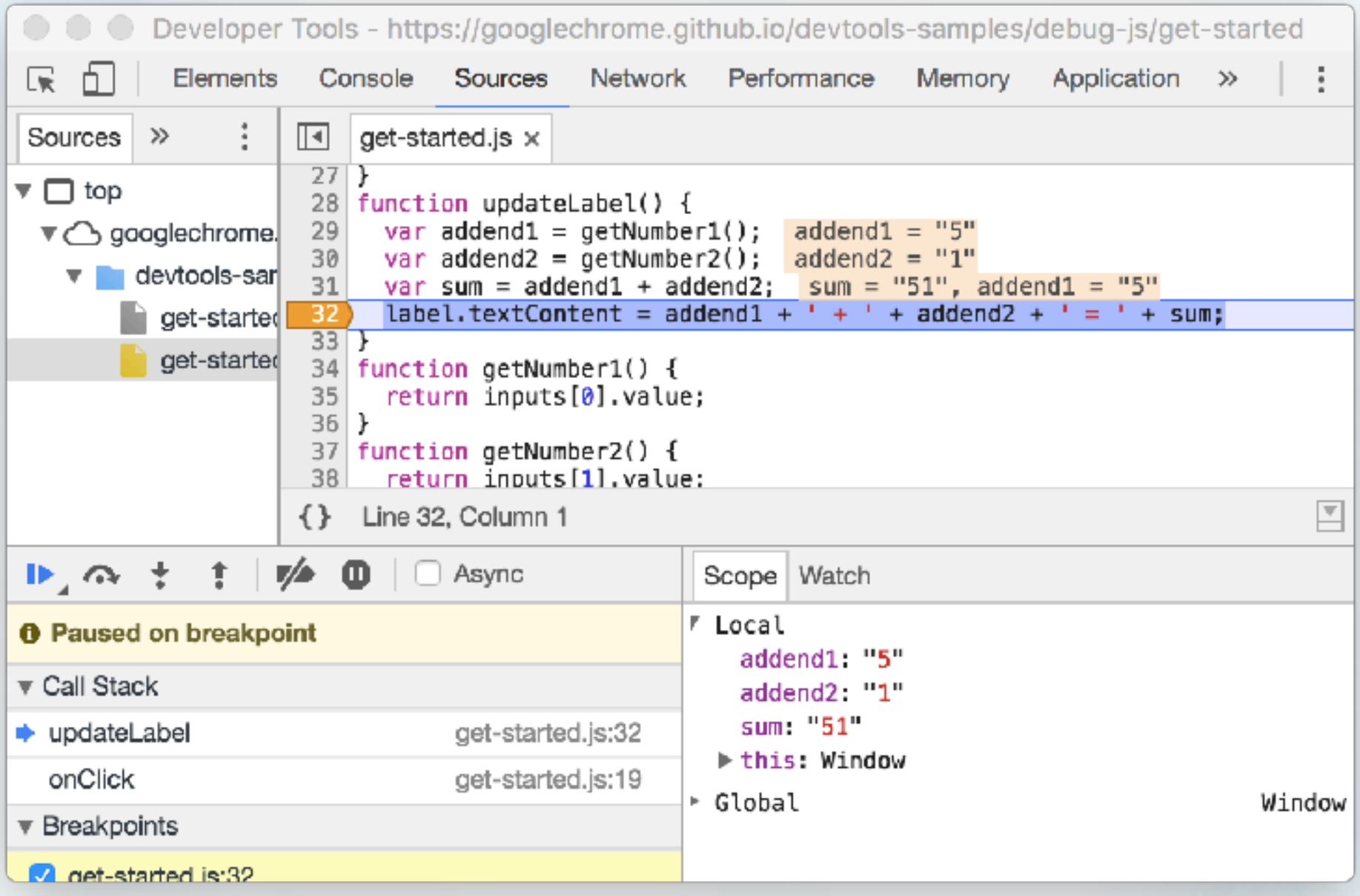
Ето няколко широко разпространени начина за дебъгване:

- Прегледайте кода си, като първо се уверете, че е подреден правилно (с правилна индентация)
- Проверете дали кода ви е валиден - редактора, който използвате показва error/warning, когато има грешка в синтаксиса (JSLint)
- Проверете дали има грешка в конзолата на браузъра

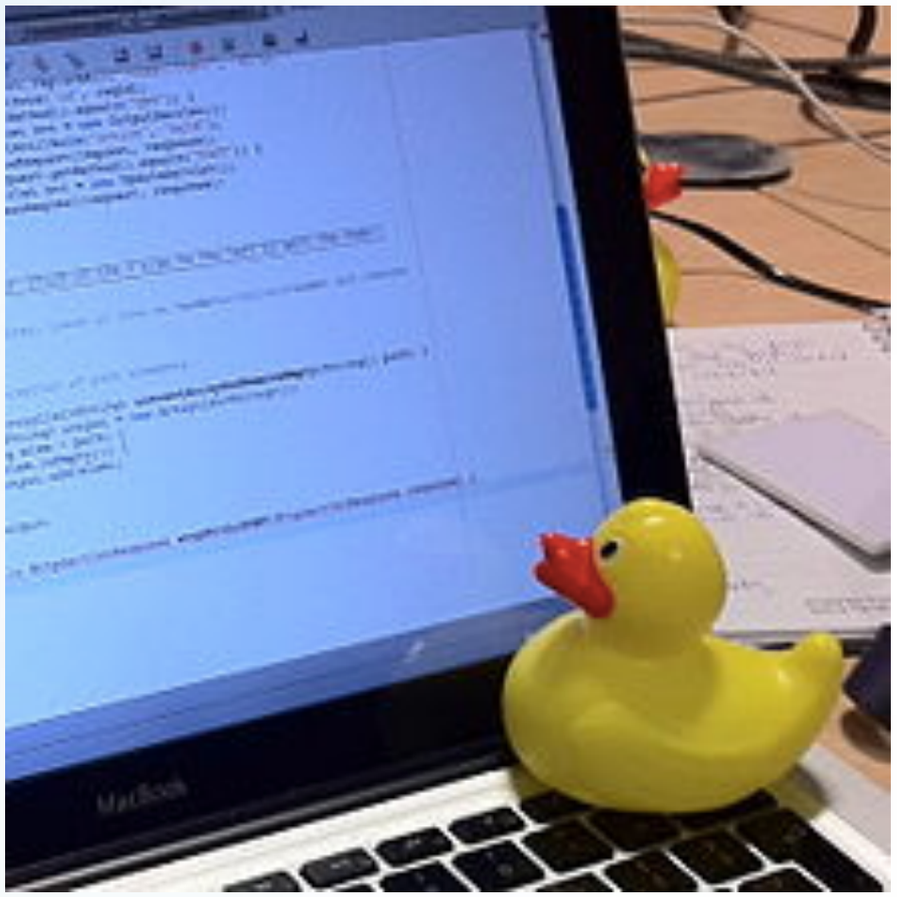
Как да дебъгваме

- Използвайте console.log за да проследите изпълнението на кода ви, както и текущите стойности на променливите
- Организирайте кода си по-добре, като го раздробите на по-малки функции и използвате повече междинни променливи
- Приложете метода на гуменото пате
- Формулирайте въпрос, с който да попитате Гугъл. Търсете отговорите в StackOverflow
- Използвайте дебъгера на браузъра

Използване на дебъгера



https://developers.google.com/web/tools/chrome-devtools/javascript/breakpoints



Въпроси?

Code smells

… & Refactoring

Code smell, in computer programming **code**, refers to any symptom in the source **code** of a program that possibly indicates a deeper problem.

— Wikipedia

Smelly code

- Код, който е написан така, че да е трудно-четим, трудно-разбираем или трудно-проследим
  - лошо именуване на променливите и методите (не са на английски, не са достатъчно описателни, не отговарят на естеството на обектите/методите)
  - лоша (неправилна) индентация
  - прекалено дълги функции (методи)
  - прекалено много вложени операции
- Код, който съдържа повторения - Wet code!

Чести code smells

- Използване на променливи преди да са декларирани (hoisting)
  - *решение: сложете всички декларации в началото на функцията*
- "Цапане" на глобалния контекст
  - *решение: изолирайте кода си във функция (IIFE) или използвайте Namespace*
- Работа без стриктен режим ("use strict")
  - *решение: сложете "use strict"; в началото на основната ви функция (IIFE)*

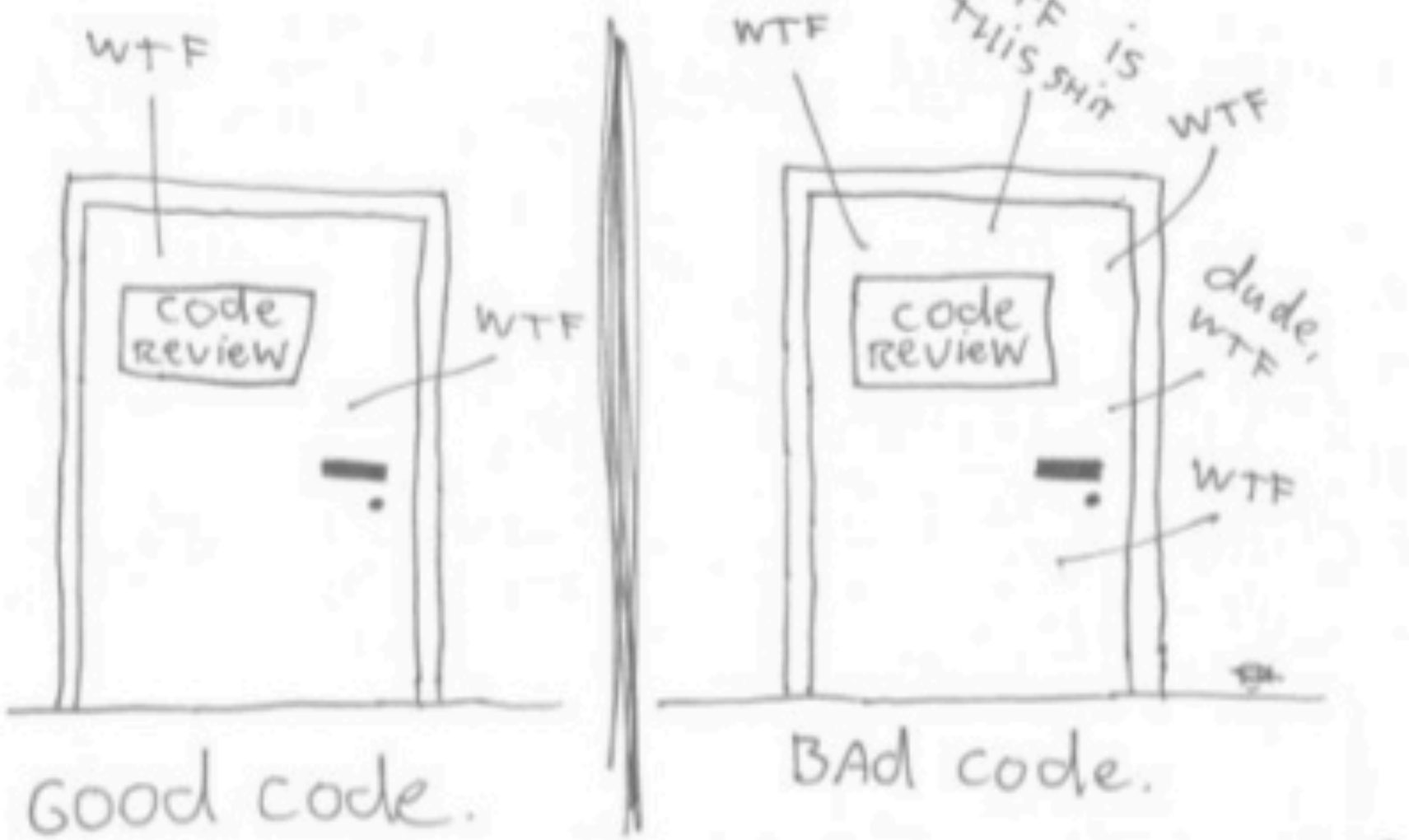
Чести code smells

- Твърде много код (прекалено дълги функции)
  - *решение: определете основните действия, които извършва кода и ги изнесете всяко в отделна функция*
- Използване на "==" вместо "==="
  - *решение: използвайте "===" и "!=="*
- Използване на променливи от външен контекст
  - *решение: предайте тези променливи като аргументи на функцията и така те ще станат част от локалния контекст*

Чести code smells

- Copy/paste
  - *решение: когато използвате копиран код, първо се уверете, че сте наясно как работи и какво точно прави*
- Повторения
  - *решение: в момента, в който забележите повторение, дори и да е само от един ред, си направете функция, която да замести повтарящия се код (DRY)*
- Писане на Javascript код директно в HTML-а
  - *решение: не го правете!*

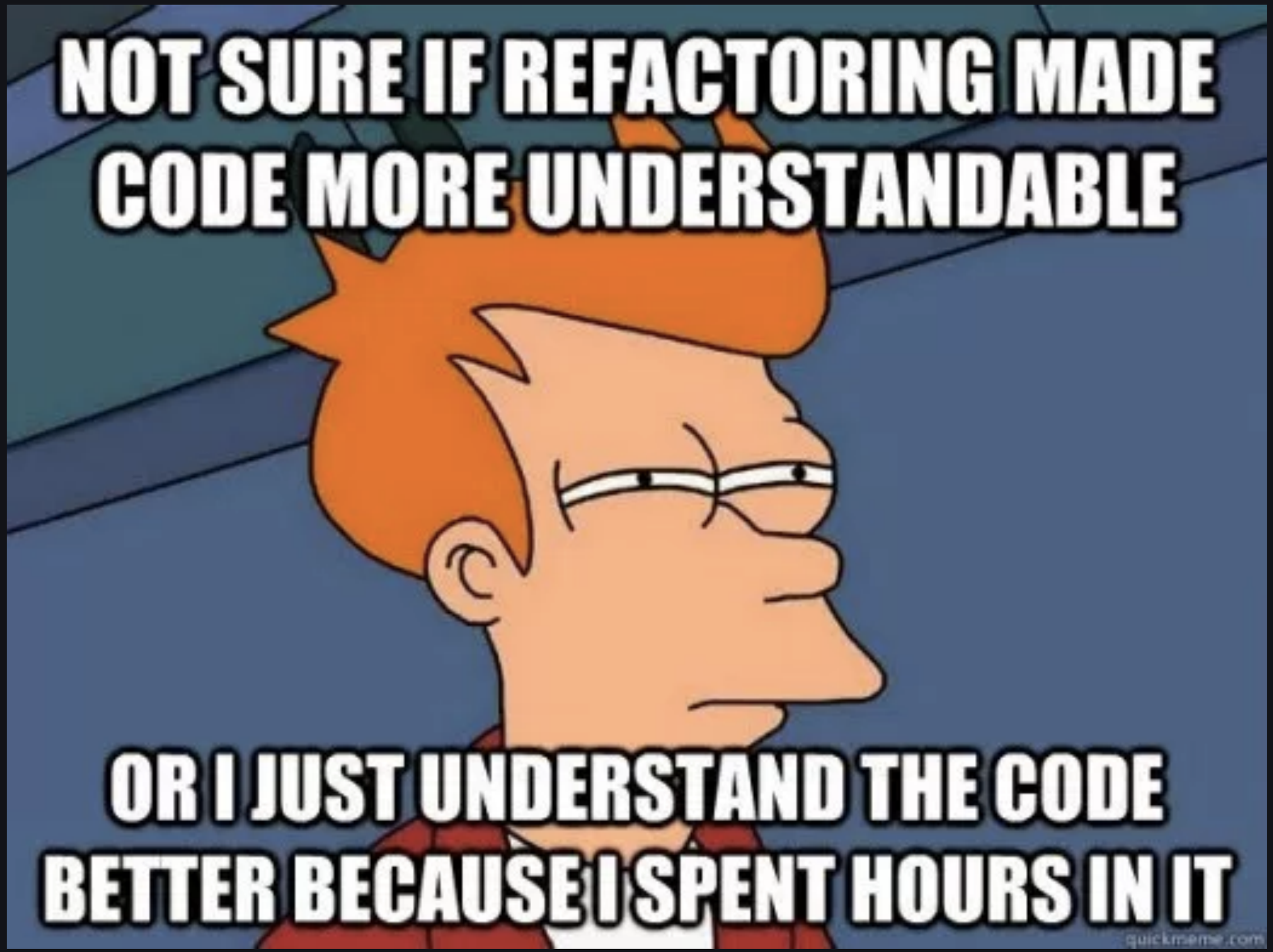
The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



Рефакторинг

Code **refactoring** is the process of restructuring existing computer code --changing the factoring -- without changing its external behavior. **Refactoring** improves nonfunctional attributes of the software — Wikipedia

- Това означава преработване на съществуващ работещ код, по начин, който не променя поведението на кода
- Обикновенно правим рефактокринг, за да изчистим появилите се Code Smells в кода
- Често рефакторинга се прави едновременно с тестване на кода (unit testing/manual), за да е сигурно, че не е променено поведението на програмата


NOT SURE IF REFACTORING MADE CODE MORE UNDERSTANDABLE
OR I JUST UNDERSTAND THE CODE BETTER BECAUSE I SPENT HOURS IN IT
quickmeme.com

Оптимизация

- Когато кода ни щади ресурсите на потребителя (процесор, памет, интернет), казваме че кода е оптимизиран
- Целта на оптимизацията е да направи сайта по-бърз
- Популярни техники за оптимизация:
  - Избягвайте вложени цикли (освен при малки списъци)
  - Изваждайте всичко, което може да е извън цикъла - отвън
  - Избягвайте рекурсия (самоизвикваща се функция)
  - Използвайте прототипа за instance variables & instance methods
  - Използвайте динамично зареждане на данни (асинхронни заявки)

Задачите от домашното

- Shopping cart
- Heroes
- Bikes
- Cars

KEEP

CALM

AND

REFACTOR

CODE

Допълнителни материали

- Списъци:
  http://swift-academy.zenlabs.pro/misc/Lists.pdf
- Обекти:
  http://swift-academy.zenlabs.pro/misc/oop-5-things.pdf
- Терминология в JS:
  http://swift-academy.zenlabs.pro/misc/JS%20terminology.pdf
- JS пунктуация:
  http://swift-academy.zenlabs.pro/misc/JS%20punctuation.pdf
- Задача за упражнение (и за домашно):
  http://swift-academy.zenlabs.pro/misc/CoffeeMaker.pdf

Полезни връзки

- Namespacing: https://javascriptweblog.wordpress.com/2010/12/07/namespacing-in-javascript/
- Optimization: https://developers.google.com/speed/articles/optimizing-javascript
- Работа с chrome debugger: https://developers.google.com/web/tools/chrome-devtools/javascript/breakpoints
- Code wars https://www.codewars.com/dashboard

Примери

http://swift-academy.zenlabs.pro/lessons/lesson17/examples/download.zip

Домашно

http://swift-academy.zenlabs.pro/lessons/lesson17/homework