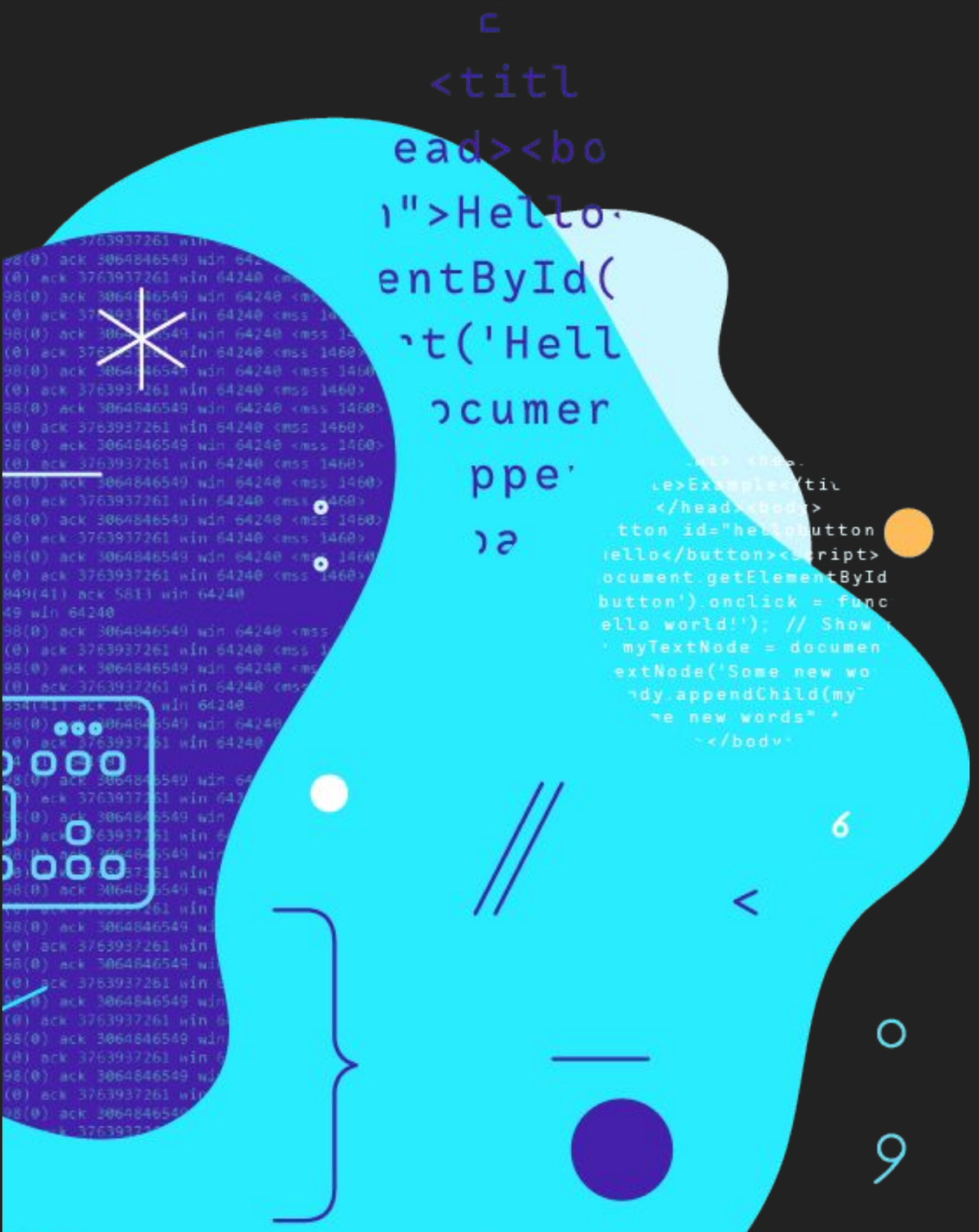
PHASE 1

WEEK 1

DAY 3



ОТЛАДКА

- Что делаешь?

- Баг чиню

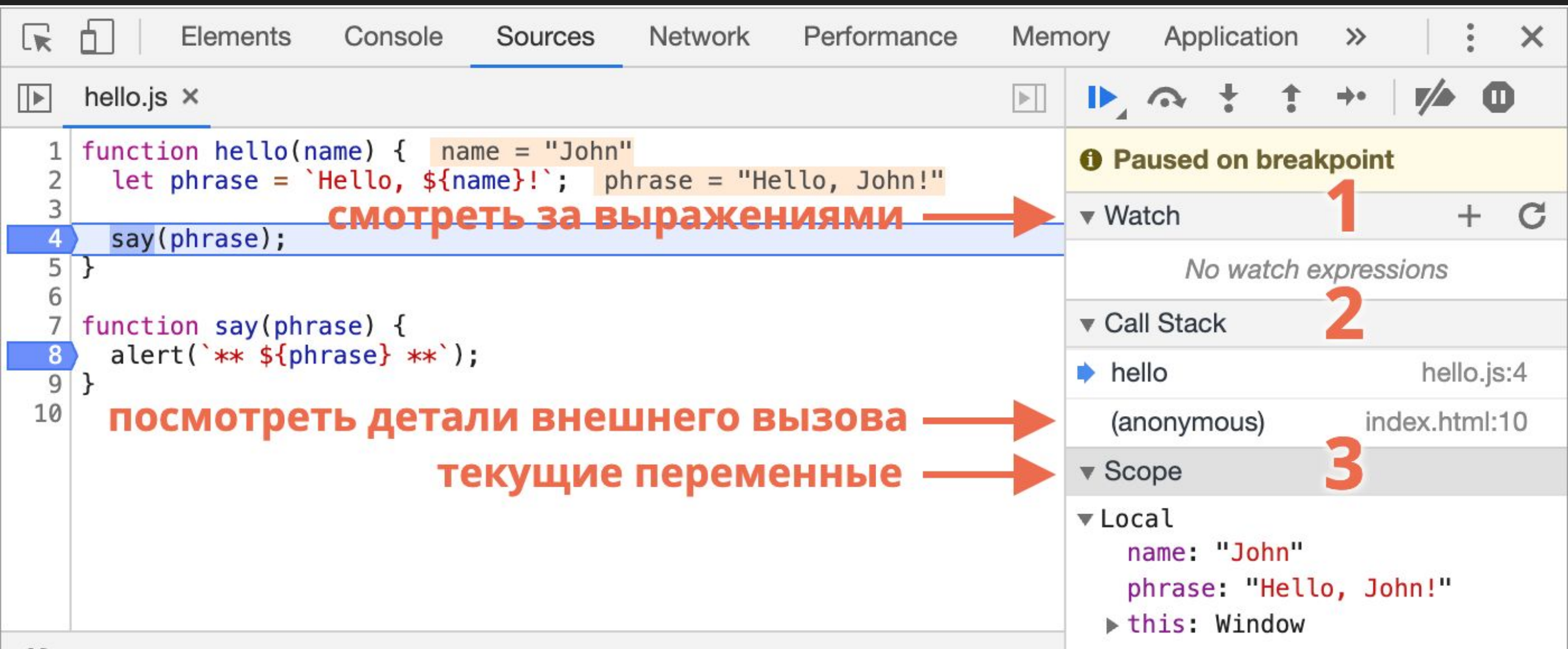
- Что это значит?

- Баг, это ошибка в коде, вот я её и исправляю

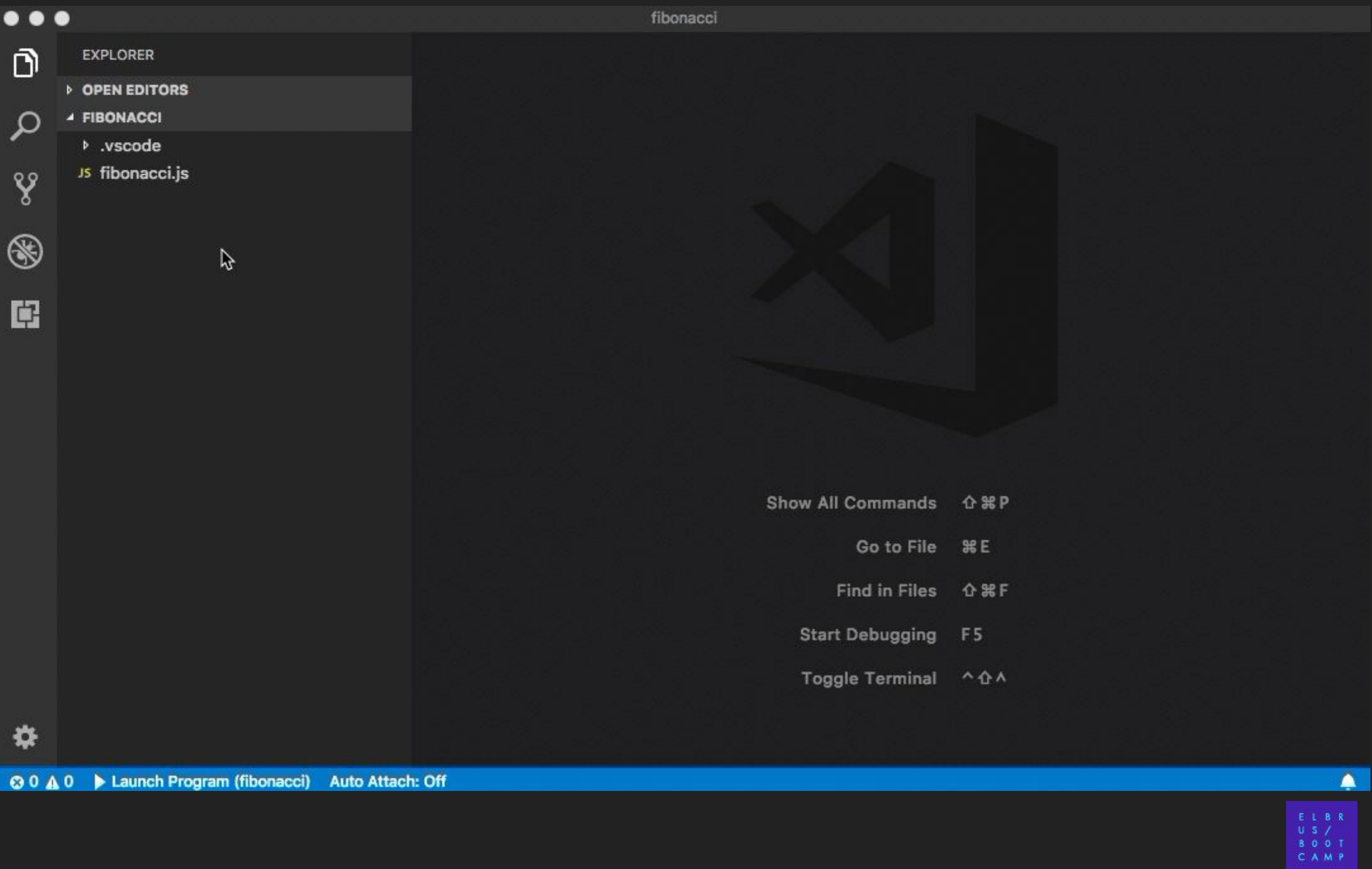
- А зачем вы ошибки в коде пишите? 17:11

Debug

Отладка – это процесс поиска и исправления ошибок в скрипте. Все современные браузеры и большинство других сред разработки поддерживают инструменты для отладки – специальный графический интерфейс, который сильно упрощает отладку. Он также позволяет по шагам отследить, что именно происходит в нашем коде.

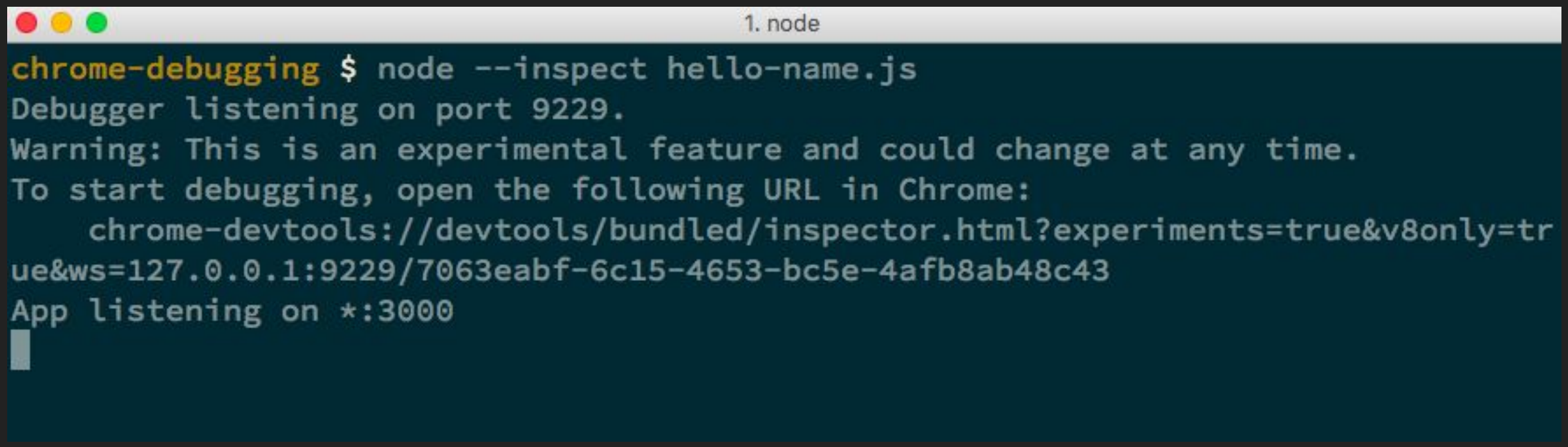


Debugger VSCode

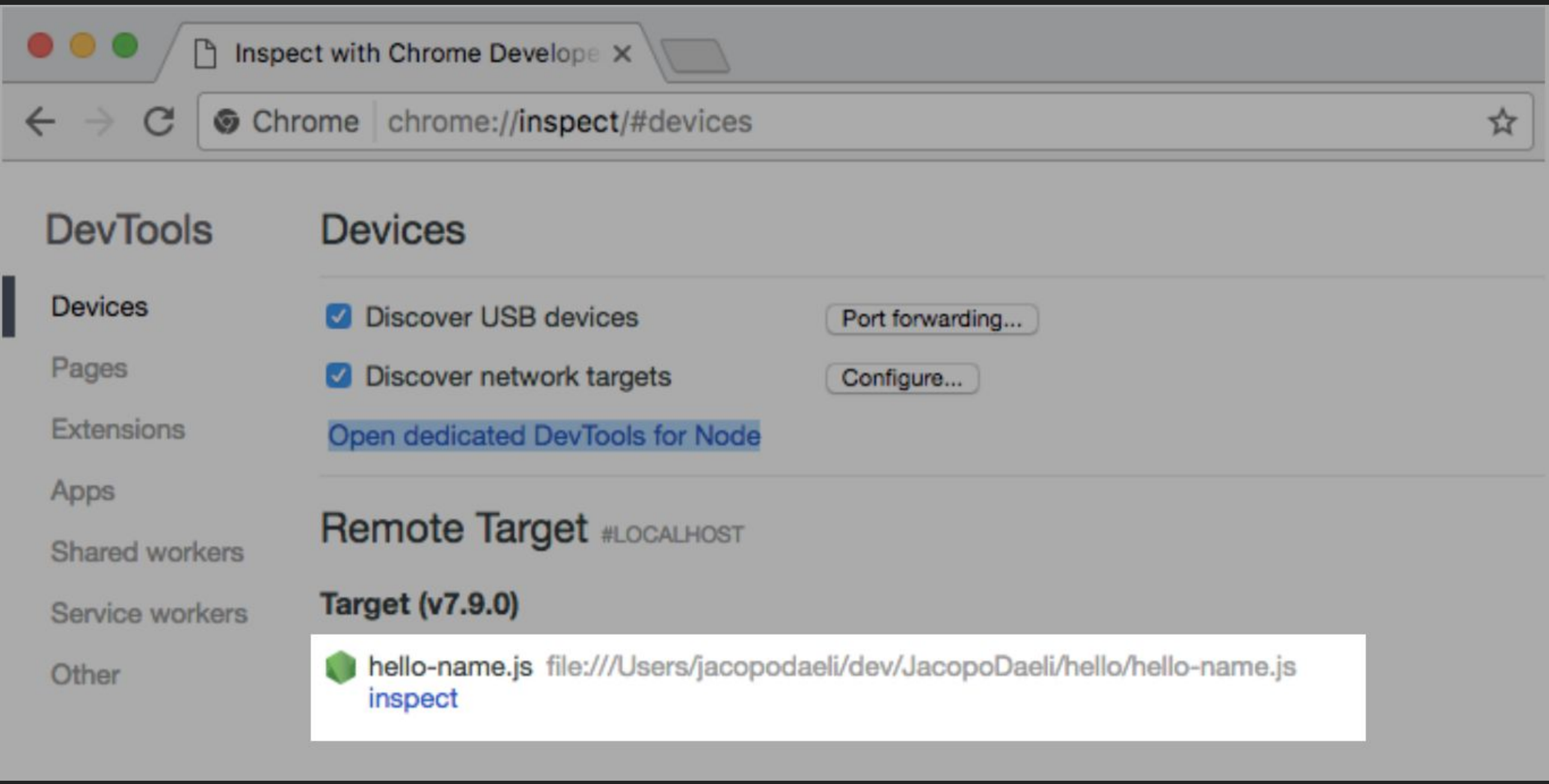


Google Chrome для отладки Node.js

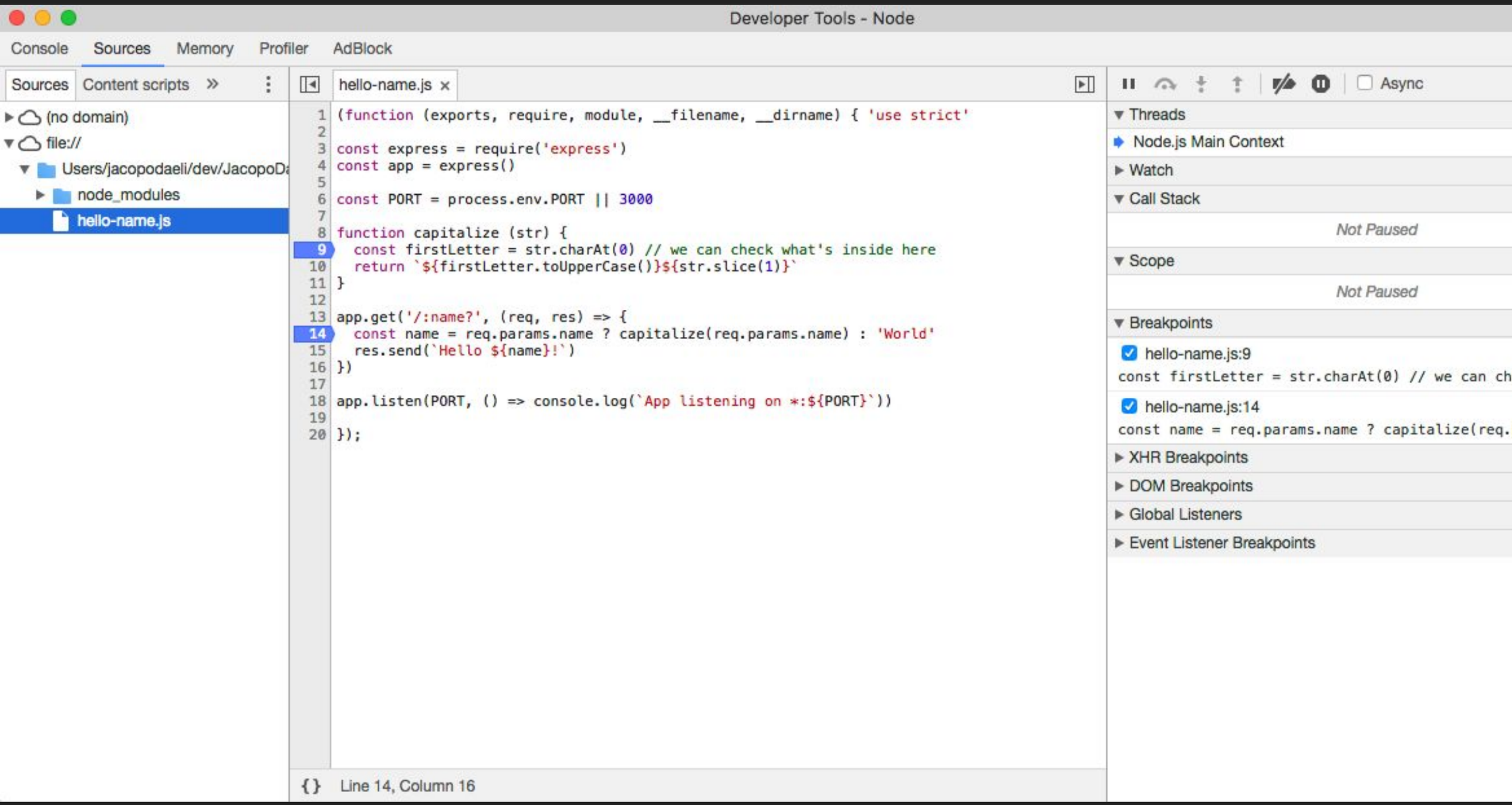
```
$ node --inspect <your_file>.js
```



Google Chrome для отладки Node.js



Google Chrome для отладки Node.js



ELBRUS / BOOTCAMP

РЕКУРСИЯ

РЕКУРСИЯ

Рекурсия - вызов функции из неё самой.

РЕКУРСИЯ

Рекурсия хороша, когда задача предполагает разделение на несколько аналогичных и простых действий.

Два основных условия: базовый случай и шаг (рекурсивный случай)

РЕКУРСИЯ

```
function countdown(i) {

console.log(i)

  if (i <= 1){

    return;

  } else {

    countdown(i - 1)

  }

}

countdown(5);
```

Базовый случай

Рекурсивный случай

РЕКУРСИЯ

```
function recurSum(n) {
  if (n === 1) {
    return n
  }
  return n + recurSum(n - 1)
}
```

Базовый случай

Рекурсивный случай

```
const res = recurSum(10); // 55
```

recurSum(10) = 10 + recurSum(10-1)

9 + recurSum(9-1)

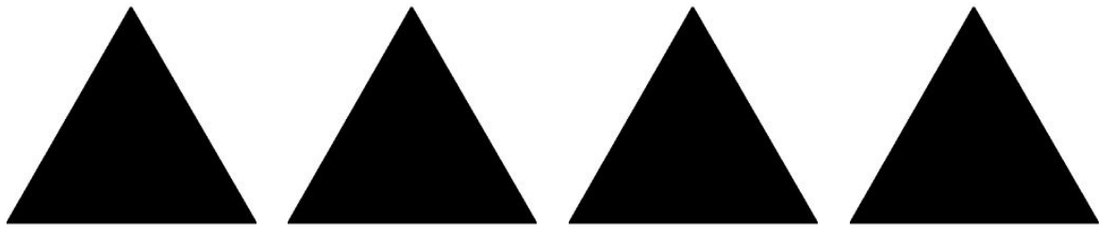
8+recurSum(8-1)

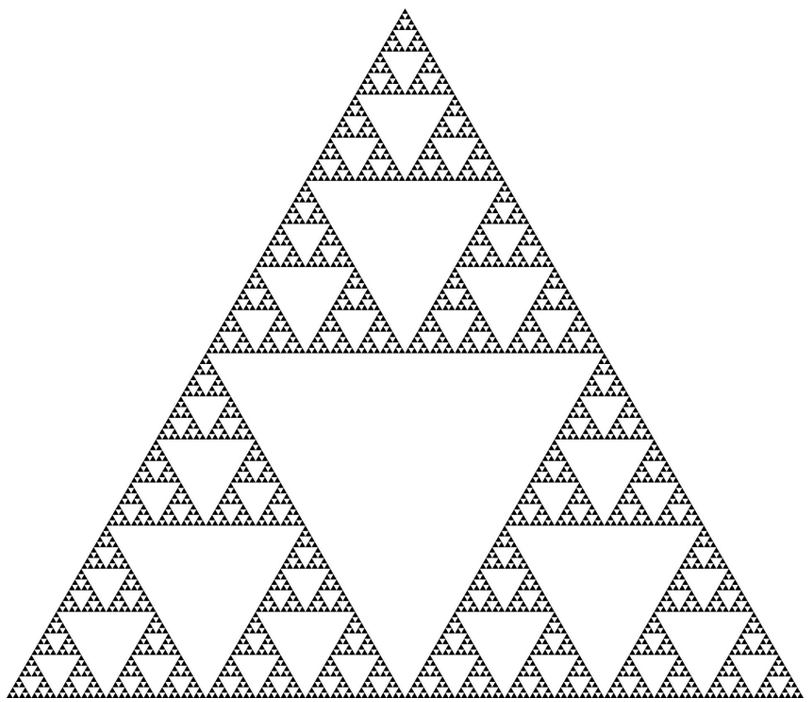
7+recurSum(7-1) ....

ИТЕРАЦИЯ

- Повторение, но не вызов самого себя
- Например, цикл for

ИТЕРАЦИЯ VS РЕКУРСИЯ





АЛГОРИТМЫ

АЛГОРИТМ ЭТО...

«Конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи.»

(бесполезное определение из Википедии)

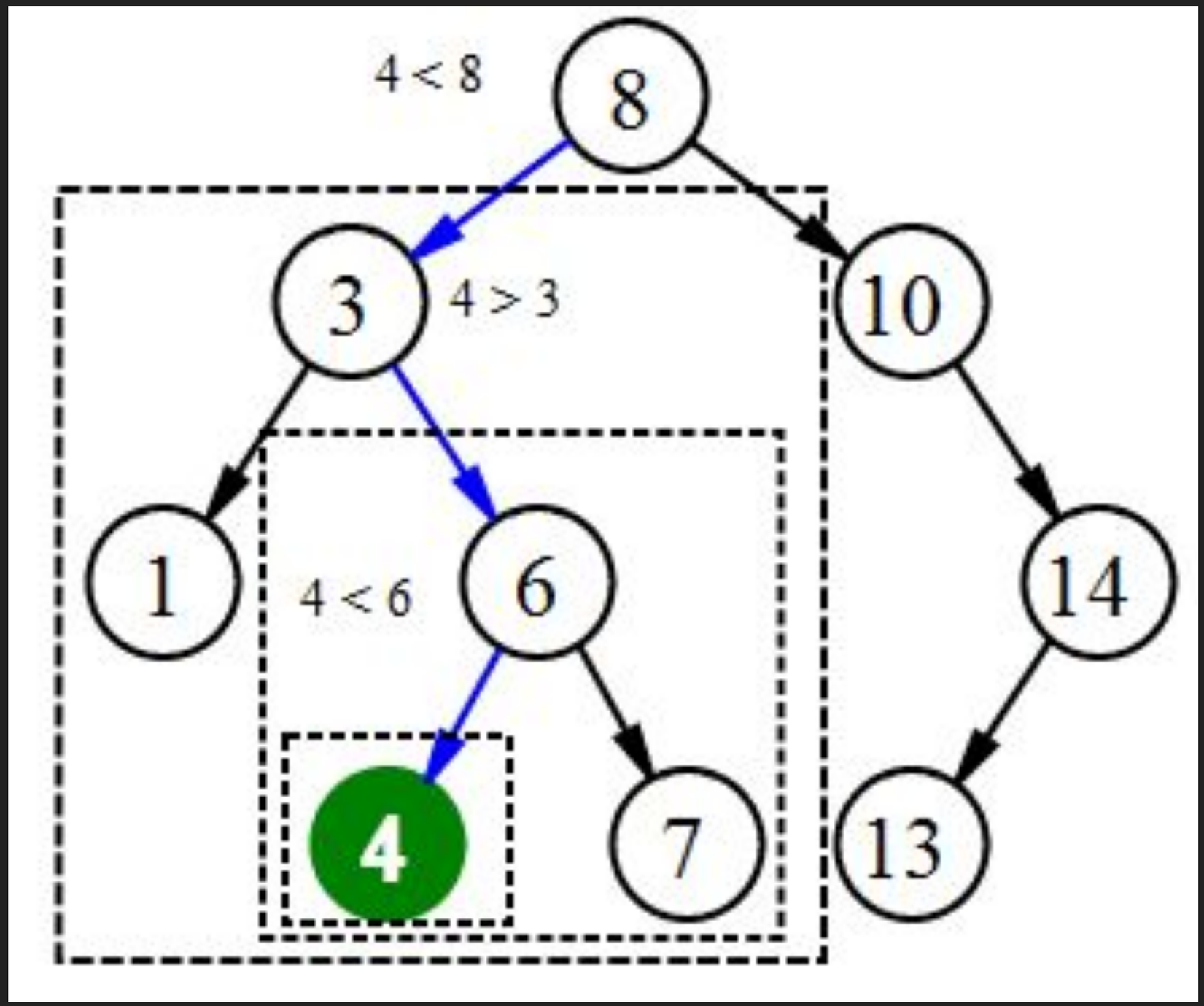
БИНАРНЫЙ ПОИСК

Выполняется по отсортированному массиву.

Бинарный поиск выполняется путем проверки того, является ли искомое значение больше, меньше или равно среднему значению в нашем массиве:

- Если оно меньше, мы можем удалить правую половину массива.
- Если оно больше, мы можем удалить левую половину массива.
- Если оно равно, мы возвращаем значение

БИНАРНЫЙ ПОИСК



ЛИНЕЙНЫЙ ПОИСК

Алгоритм линейного поиска (linear search) просто по очереди сравнивает элементы заданного списка с ключом поиска до тех пор, пока не будет найден элемент с указанным значением ключа (успешный поиск) или весь список будет проверен, но требуемый элемент не найден (неудачный поиск).

СОРТИРОВКА ПУЗЫРЬКОМ

Алгоритм состоит из повторяющихся проходов по сортируемому массиву.

За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

ВИЗУАЛИЗАЦИЯ

В виде танцев:
https://www.youtube.com/watch?v=lyZQPjUT5B4

СОРТИРОВКА ПУЗЫРЬКОМ


```
const arr = [5, 2, 1, 3, 9, 0, 4, 6, 8, 7];

for (let i = 0; i < arr.length; i += 1) {
  for (let j = 0; j < arr.length - i; j += 1) {
    if (arr[ j ] > arr[ j + 1]) {
      const temp = arr[ j ];
      arr[ j ] = arr[ j + 1 ];
      arr[ j + 1 ] = temp;
    }
  }
}
```

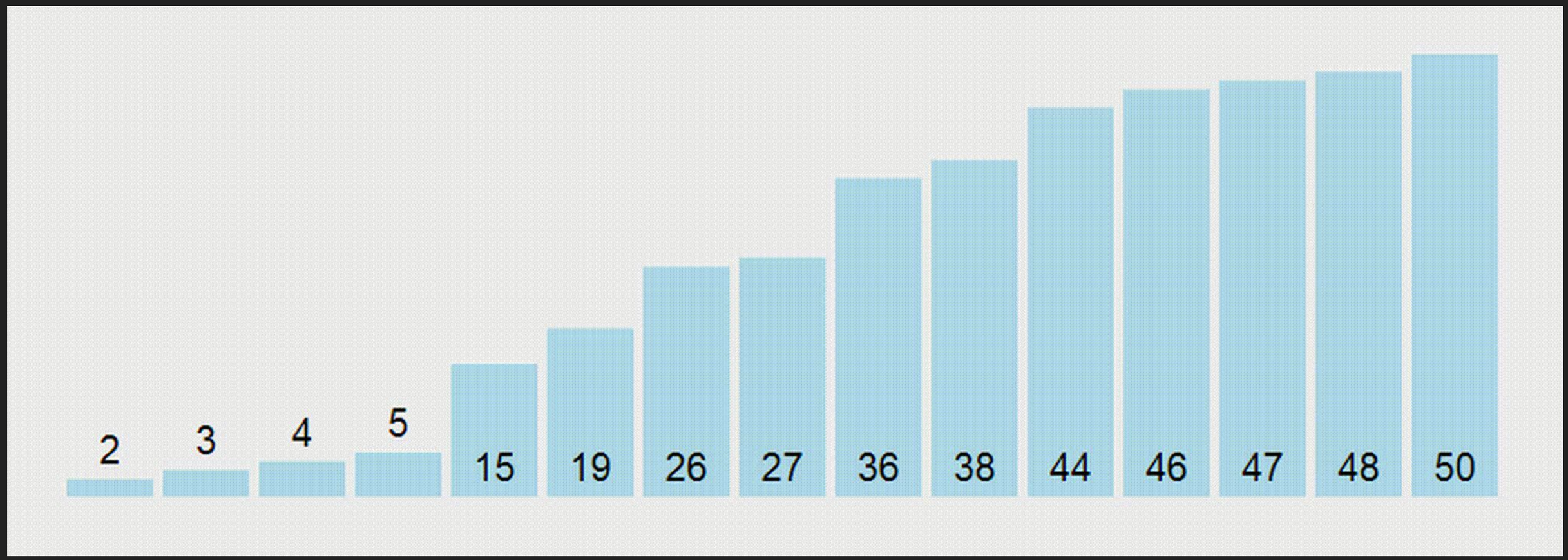
QUICKSORT

В начале выбирается “опорный” элемент массива. Это может быть любое число, но от выбора этого элемента сильно зависит эффективность алгоритма. Если нам известна медиана, то лучше выбирать элемент, который как можно ближе к медиане. В нашей реализации алгоритма, мы будем брать самый левый элемент, который в результате займет свое место.

Элементы в массиве делятся на две части: слева те кто меньше опорного элемента, справа те кто больше. Таким образом опорный элемент занимает свое место и больше никуда не двигается.

Для левого и правого массива действия повторяются рекурсивно.

ВИЗУАЛИЗАЦИЯ



В виде танцев:
https://www.youtube.com/watch?v=ywWBy6J5gz8

QUICKSORT

```
const arr = [15, 4, 10, 100, 2, 34, 6, 8];
function quickSort(items, left, right) {
  let index;
  if (items.length > 1) {
    index = partition(items, left, right);
    if (left < index - 1) {
      quickSort(items, left, index - 1);
    }
    if (index < right) {
      quickSort(items, index, right);
    }
  }
  return items;
}
quickSort(arr, 0, arr.length - 1);
```

РАЗБИЕНИЕ МАССИВА НА 2 ЧАСТИ

```
function partition(items, left, right) {
  let pivot   = items[Math.floor((right + left) / 2)],
      i       = left,
      j       = right;
  while (i <= j) {
    while (items[i] < pivot) {
      i++;
    }
    while (items[j] > pivot) {
      j--;
    }
    if (i <= j) {
      const temp = items[i];
      items[i] = items[j];
      items[j] = temp;
      i++;
      j--;
    }
  }
  return i;
}
```