Front-end

JavaScript

JS.

- Рекурсия

JS. Рекурсия

Рекурсия в общем смысле - это повторение какого-либо объекта или процесса внутри самого этого объекта или процесса.

В программировании в том числе и в JavaScript под рекурсией чаще всего понимают метод, который предполагает создание функции, которая вызывает саму себя до тех пор пока программа не достигнет необходимого результата.

JS. Рекурсия

Рекурсия — это когда функция вызывает сама себя.

```
function factorial(x) {
  if (x < 0) return;
  if (x === 0) return 1;
  return x * factorial(x - 1);
}
    factorial(3); // => 6
    factorial(4); // => 24
```

JS. Рекурсия

Все рекурсивные функции должны иметь три ключевых особенности:

Условие завершения

Условие: if('случилось что-то плохое'){ СТОП }; Условие завершения— это способ предотвратить ошибки. Это как экстренный тормоз. Оно защищает функцию от возможности запуститься с неверными данными. В примере с факториалом, if (x < 0) return; наше условие окончания. Мы не можем (в не продвинутой математике) посчитать факториал отрицательного числа, поэтому мы не хотим запускать рекурсию, если в функцию передано такое число.

JS. Рекурсия

Все рекурсивные функции должны иметь три ключевых особенности:

Базовый сценарий

Simply put: if('это произошло') { УРА, Закончили!}; Базовый сценарий похож на условие завершения, он тоже завершает рекурсию. Но надо помнить, что условие завершения — это ловушка для плохих данных, а базовый сценарий— это цель нашей рекурсивной функции. Обычно базовые сценарии написаны с помощью оператора if. В примере с факториалом, if (x === 0) return 1; базовый сценарий. Мы знаем, что когда мы уменьшили x до нуля, мы закончили подсчитывать факториал!

Базовый случай — это условие, при выполнении которого рекурсия заканчивается и функция больше не вызывает саму себя.

Рекурсия

Функция вызывает сама себя с другим (обычно изменённым изначальным) параметром. В примере с факториалом, return x * factorial(x — 1); место, где происходит рекурсия. Мы возвращаем значение x умноженное на значение того, что подсчитает функция factorial(x-1).

JS. Рекурсия

```
function factorial(x) {

    if (x < 0) return; // => условие завершения
     if (x === 0) return 1; // => базовый сценарий
     return x * factorial(x - 1); // => рекурсия


}

    factorial(3); // => 6
```

JS. Рекурсия

Если не прописать **базовый случай** мы получим бесконечный вызов. Для большей надежности рекомендуют сначала прописать условие выхода, а затем описывать функционал.

Стек в случае с рекурсией - это специальная структура данных в которую записывается по порядку информация о каждом вызове функции самой себя, в том числе и первом вызове.

Отсюда исходит основная опасность работы с рекурсией - это переполнение стека. Глубина рекурсии равняется количеству контекстов, которые хранятся в стеке одновременно, а это число ограничено мощностью компьютера - обычно от 10 000 до 12 000. Для недопущения переполнения и существует базовый случай, тоесть условие, когда вызов функции самой себя должен прекратиться.

JS. Рекурсия

```
let days = 0;

  function howMuchToLearnJS() {
    days++;
    console.log(days);
    howMuchToLearnJS();
  }

  howMuchToLearnJS();
```

JS. Рекурсия

```
let days = 0;
  function howMuchToLearnJS() {
    if (days === 400) return; // условие выхода
    days++;
    console.log(days);
    howMuchToLearnJS();
  }
  howMuchToLearnJS();
```

JS. Рекурсия

Цикл и рекурсия

Любая задача, которую можно решить рекурсией, также решается и с помощью цикла

Рекурсия проигрывает циклу в следующем:

- Отлаживать рекурсию значительно сложнее, чем цикл, а если функция написана плохо — то и просто читать.
- Она может приводить к переполнению стека. Особенно это ощутимо в таких языках как JS, где переполнение стека может наступить раньше базового случая с высокой вероятностью.
- Её выполнение может (хотя необязательно) занимать больше памяти.

Цикл же проигрывает рекурсии в таких вещах:

- Его нельзя использовать в функциональном программировании, потому что он императивен.
- Циклом гораздо сложнее обходить вложенные структуры данных, например, каталоги файлов.
- При работе с общими ресурсами или асинхронными задачами чаще удобнее использовать рекурсивные функции из-за замыкания.

Поэтому на вопрос «Что использовать: рекурсию или цикл?» ответом будет «Зависит от задачи»

JS. Рекурсия

Итого

Рекурсия - это метод написания функции, когда она вызывает саму себя. С помощью рекурсии можно решить много задач, но большинство из них можно легко переписать используя цикл.

Структуры данных с глубокой вложенностью или неизвестной глубиной - отличный пример, когда рекурсия справляется с обходом лучше, чем цикл.

Базовый случай - это условие, когда рекурсия должна закончится. Такое условие важно предусмотреть, так как количество вызовов функции самой себя ограничено из-за переполнения стека.