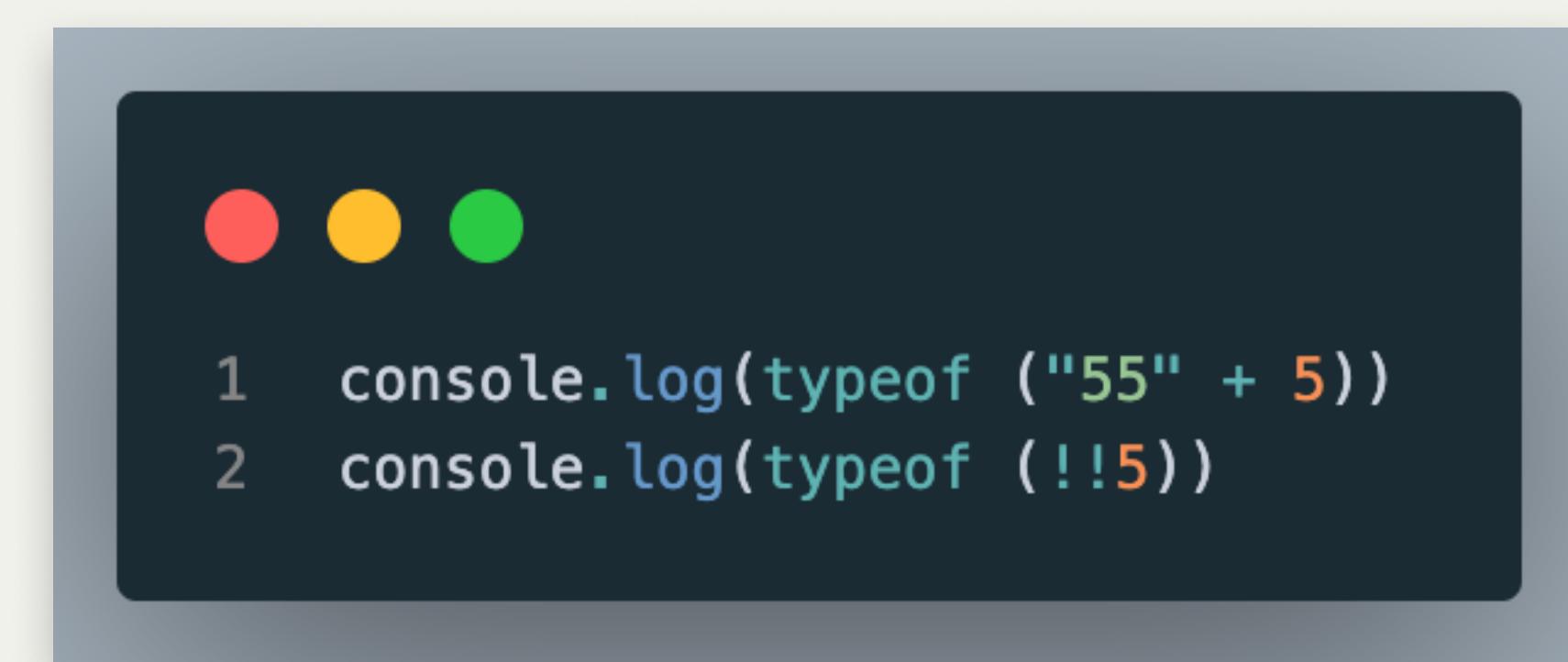


# ПРОБЛЕМЫ В СТАНДАРТНОМ JAVASCRIPT

**JS**

ЭТО ЯЗЫК С ДИНАМИЧЕСКОЙ ТИПИЗАЦИЕЙ,  
а это значит, вы можете выполнять любые операции с типами,  
например:



```

1 console.log(typeof ("55" + 5))
2 console.log(typeof (!!5))

```

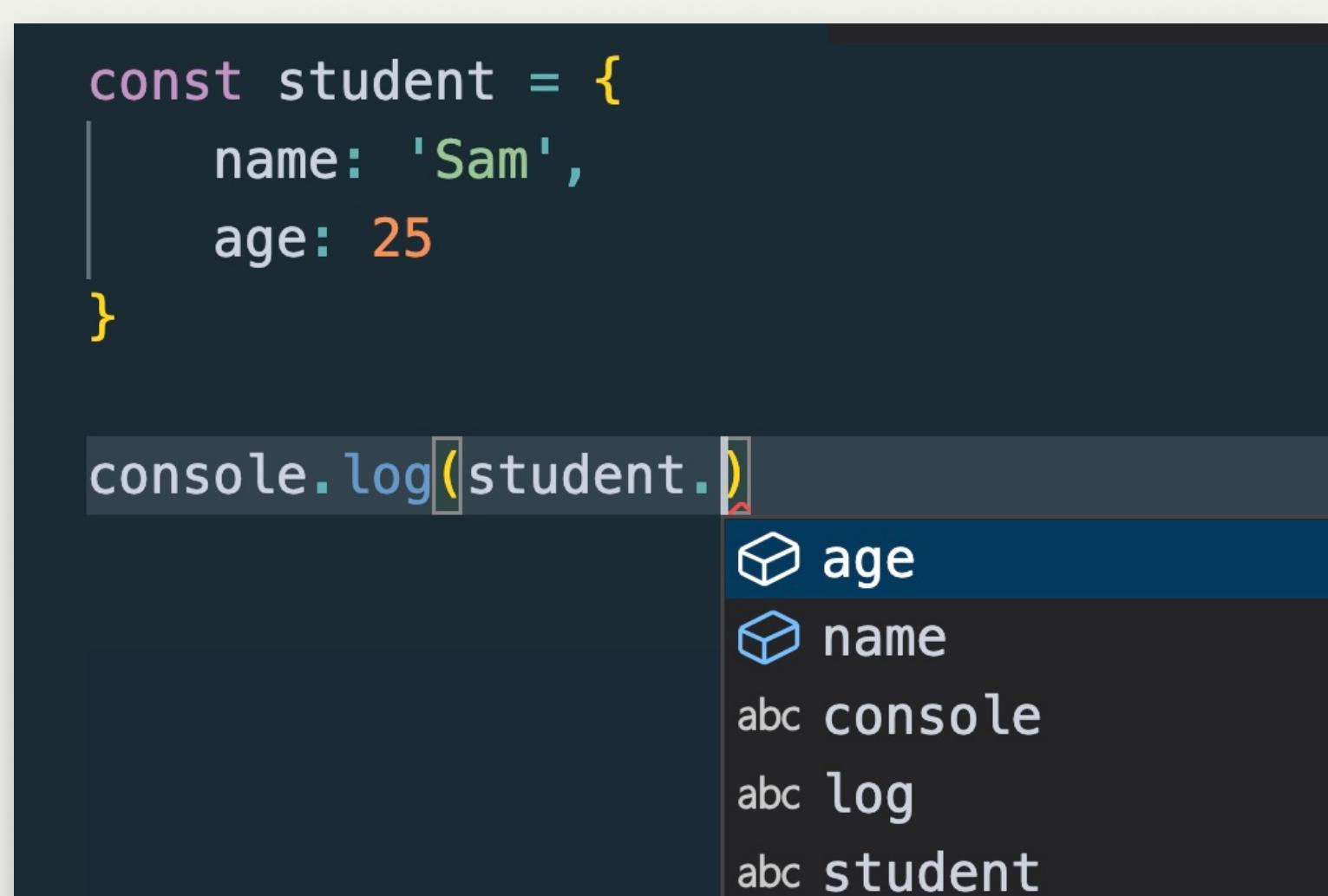
Иногда это удобно, но и вызывает трудности:

- 👉 Чем больше кода в проекте, чем больше разработчиков - тем сложнее уследить за всеми этими манипуляциями
- 👉 Сразу сложно понять структуру кода: что будет строкой, что числом, что обязательно, что нет. Мы тратим время на выяснение этих нюансов
- 👉 Сложности при исправлении ошибок в крупном проекте: получили ошибку, исправляем её, ждем сборку проекта, опять тестируем, опять исправляем... Мы не получаем уведомления об ошибках при разработке, только в runtime \*runtime ошибки - это те, которые мы видим только при запуске кода

И еще часть, где JS вам никак не помогает:

- 👉 Нет подсказок про несуществующие свойства и тп
- 👉 Нет автодополнения в коде

\*и других вещей, которые бы облегчили нам жизнь. Мы говорим конкретно про язык JS, а не дополнительные плагины и утилиты



```

const student = {
  name: 'Sam',
  age: 25
}

console.log(student.)

```

age  
name  
surname  
log  
student

Хотелось бы вот так



```

1 const student = {
2   name: 'Sam',
3   age: 25
4 }
5
6 console.log(student.surna)
7 console.log(student.ege)

```

Нет подсказок про опечатку

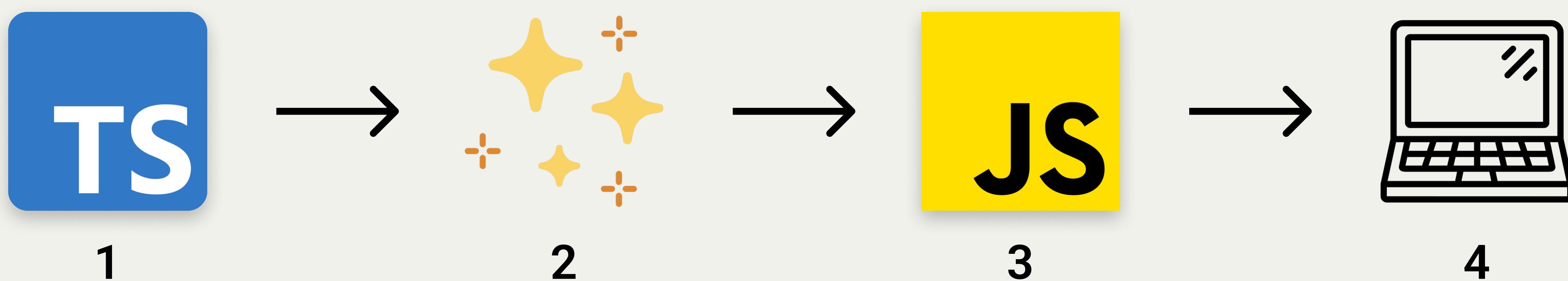
# РЕШЕНИЕ ПРОБЛЕМ С ПОМОЩЬЮ TYPESCRIPT

## Что дает нам язык TypeScript:

- 👉 Позволяет нам использовать “аннотации” типов. Простыми словами: говорить, что эта сущность всегда будет числом, эта - объектом, эта - массивом... Вы сразу понимаете с чем вы работаете.
- 👉 Накладывает ограничения на часть операций с разными типами. Вам проще отслеживать все действия
- 👉 За счет строгих типов код подвергается активному анализу во время разработки.  
Это в свою очередь дает нам:
  - 👉 Autocomplete кода, вам автоматически будут предлагаться только доступные к использованию варианты
  - 👉 Подсказки о прямых или возможных ошибках в коде еще на этапе разработки. Вы не тратите время на поиск их в runtime

\*и другое, но пока этого достаточно 😊

## КАК ЭТО РАБОТАЕТ ВНУТРИ



- 1 - тут мы пишем код на TypeScript (далее просто TS).
- 2 - код отправляется в компилятор для преобразования
- 3 - на выходе получаем все тот же JS код
- 4 - этот код идет в работу (сайты, приложения, сервера...)

Внутри TS **можно** писать и на стандартном JS, без всех возможностей TS. Это не приведет к ошибкам, так как в итоге мы получим все равно JS код

## ГДЕ МОЖНО ИСПОЛЬЗОВАТЬ TS



и другие...

Очень много инструментов поддерживают TypeScript, а в некоторых он обязателен для работы (Angular, Nestjs...)

## ПЛЮСЫ И МИНУСЫ TS

### Плюсы и зачем его желательно изучить:

- 👉 Много полезных вещей с технической стороны, это облегчает разработку
- 👉 Намного удобнее работать в команде, код документирует сам себя
- 👉 Ускоряет разработку, позволяет избегать ошибок и проблем
- 👉 Сильно облегчает масштабирование проектов
- 👉 Позволяет разработчикам понять, как устроены другие технологии и языки с использованием системы типов

### Из минусов:

- 👉 Добавляет лишнюю прослойку, которая может быть не нужна небольшим проектам