Яндекс

```
lecture.theme
```

```
ReferenceError: lecture is not defined
```

Евгений Марков, разработчик интерфейсов

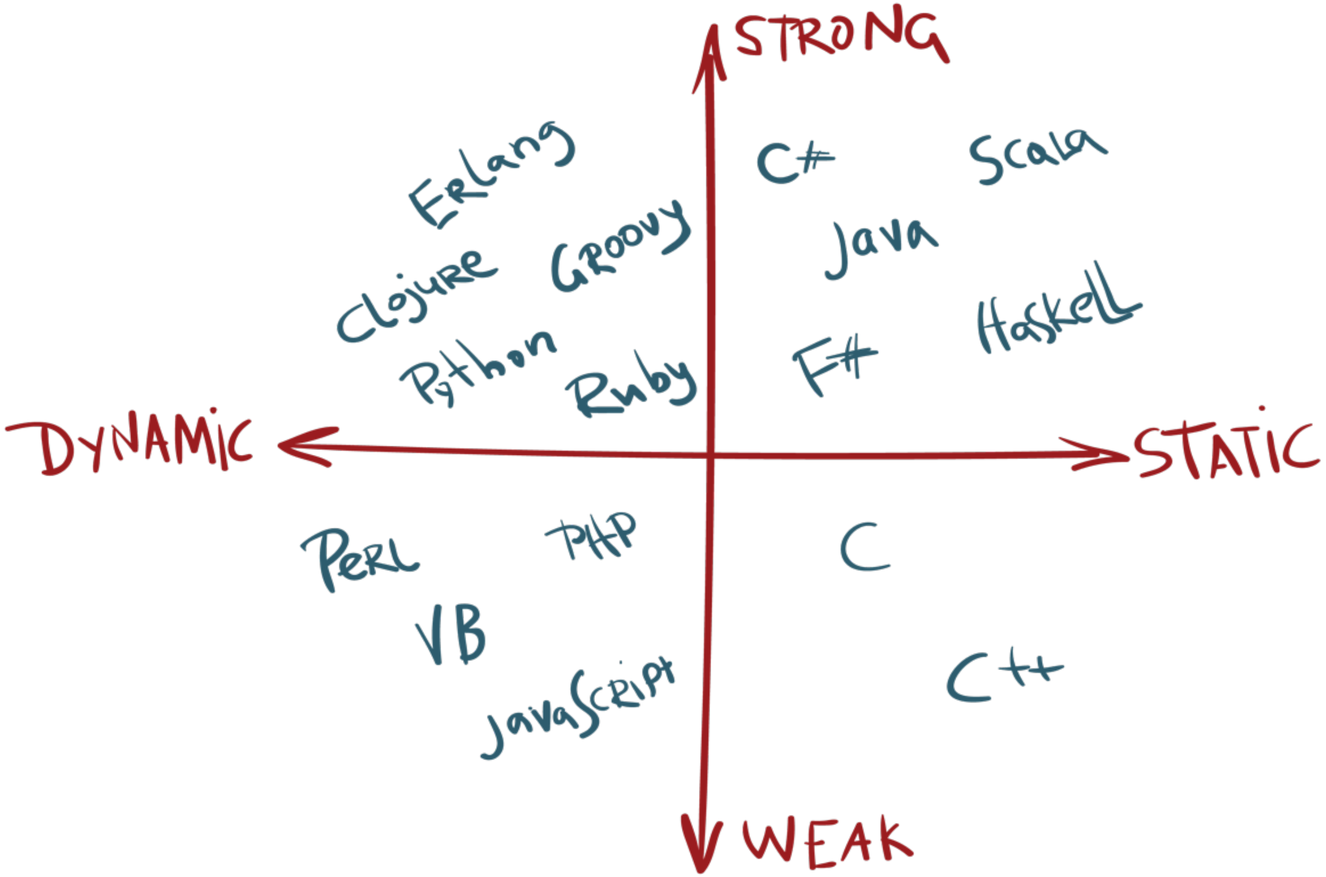
01

Проблемы JavaScript

- Типизация
- Уровень поддержки
- Developer Experience

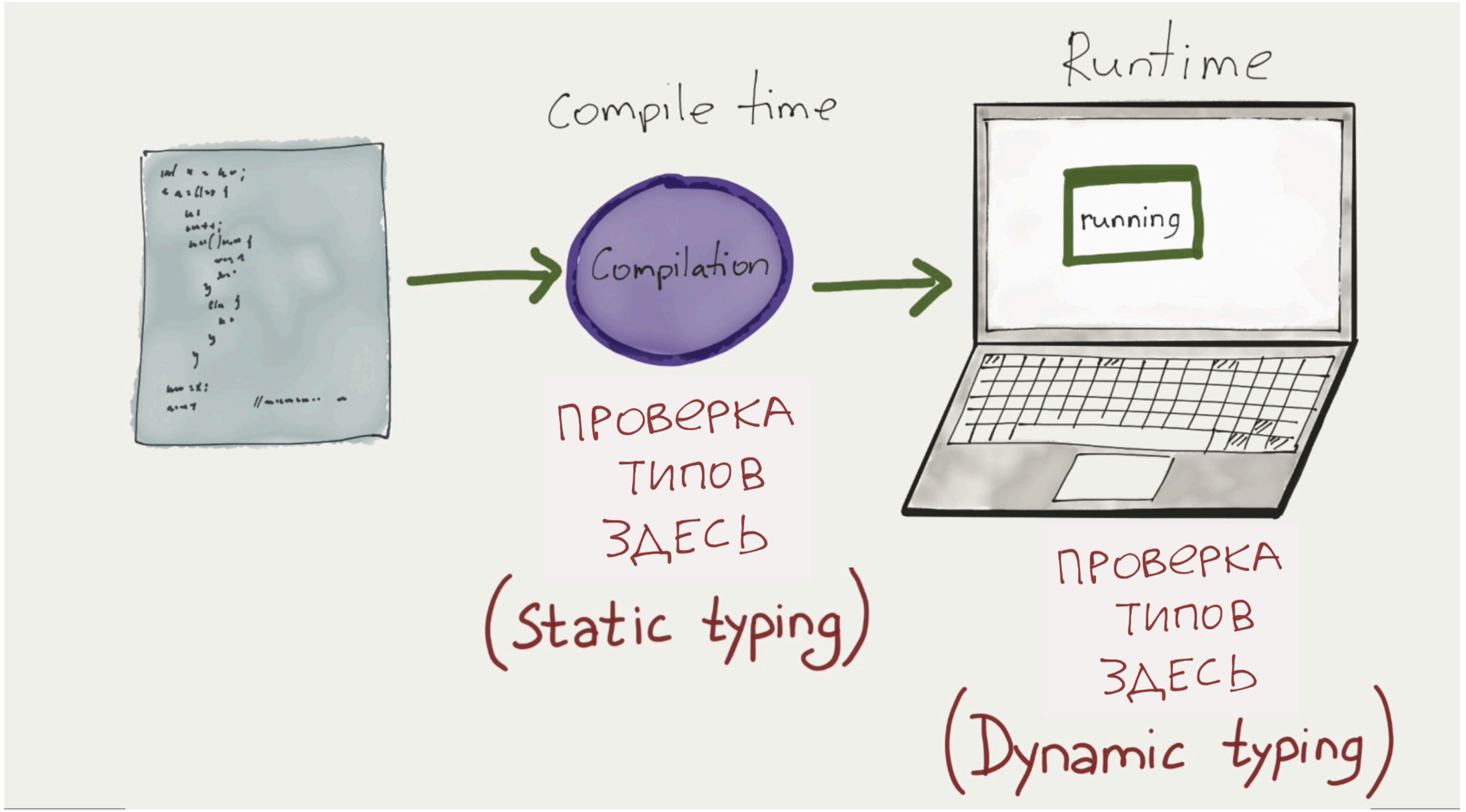
Типизация

В JavaScript слабая динамическая типизация



Типизация

В JavaScript динамическая типизация



Типизация

В JavaScript динамическая типизация

- Нет защиты от ошибок во время написания кода
- Много бесполезного кода с проверками на типы
- Меньше возможностей для оптимизации

Типизация

В JavaScript слабая типизация

```
4 + '7';        // '47'
4 * '7';        // 28
2 + true;       // 3
false - 3;      // -3
```

Уровень поддержки

Наш код исполняется в различных средах

Google Chrome	7 916 145 845	39,84 %
Яндекс.Браузер	4 076 271 731	20,51 %
Safari	2 071 742 519	10,43 %
Opera	1 081 570 904	5,44 %
Firefox	955 857 494	4,81 %
Android Browser	594 230 694	2,99 %
Samsung Internet	518 875 151	2,61 %
Internet Explorer	403 732 235	2,03 %
Edge	314 114 728	1,58 %
MIUI browser	265 688 455	1,34 %
UC Browser	85 215 796	0,43 %
Amigo	59 642 733	0,30 %
Остальные	1 528 155 280	7,69 %

Данные из отчёта «Яндекс.Радара» на 10 апреля 2019

Таблица совместимости

https://kangax.github.io/compat-table

			Compilers/polyfills					Desktop browsers										Servers/runtimes						Mobile	
		78%	6%	56%	52%	55%	10%	0%	52%	58%	78%	78%	78%	100%	100%	83%	90%	2%	19%	77%	100%	8%	94%	83%	90%
Feature name	▶	Current browser	Traceur	Babel 6 + core-js	Closure 2018.10	Type-Script + core-js	es7-shim	IE 11	Edge 17	Edge 18	FF 60 ESR	FF 62	FF 63	CH 69, OP 56	CH 70, OP 57	SF 11.1	SF 12	PJS	Node >=6.5 <7[2]	Node >=8.10 <9[2]	Node >=10.13 <11[2]	DUK 2.2	GraalVM 1.0[3]	iOS 11.3	iOS 12
2016 features																									
exponentiation (**) operator	▶	3/3	2/3	3/3	3/3	2/3	0/3	0/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	0/3	3/3	3/3	2/3	3/3	3/3	3/3
Array.prototype.includes	▶	3/3	0/3	3/3	2/3	3/3	2/3	0/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	3/3	3/3	3/3	0/3	3/3	3/3	3/3
2016 misc																									
generator functions can't be used with "new" [7]		Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
generator throw() caught by inner generator [8]		Yes	No	No	Yes	Yes[9]	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
strict fn w/ non-strict non-simple params is error[10]		Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
nested rest destructuring, declarations [11]		Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
nested rest destructuring, parameters[12]		Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Proxy, "enumerate" handler removed [13]		Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Proxy internal calls, Array.prototype.includes		Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
2017 features																									
Object static methods	▶	4/4	0/4	4/4	3/4	4/4	3/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	4/4	0/4	4/4	4/4	4/4
String padding	▶	2/2	0/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	0/2	2/2	2/2	2/2
trailing commas in function syntax	▶	2/2	0/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	0/2	2/2	2/2	2/2
async functions	▶	15/15	3/15	3/15	9/15	8/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	0/15	15/15	15/15	0/15	13/15	15/15	15/15
shared memory and atomics	▶	0/17	0/17	0/17	0/17	0/17	0/17	0/17	0/17	0/17	0/17	0/17	0/17	17/17	17/17	0/17	0/17	0/17	0/17	17/17	17/17	0/17	17/17	0/17	0/17
2017 misc																									
Proxy "ownKeys" handler, duplicate keys for non-extensible targets (ES 2017 semantics) [22]		No	No	No	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RegExp "u" flag, case folding		Yes	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes
arguments.caller removed		Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes
2017 annex b																									
Object.prototype getter/setter methods	▶	16/16	0/16	16/16	0/16	16/16	0/16	8/16	14/16	14/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	12/16	10/16	16/16	16/16	16/16	16/16	16/16	16/16
Proxy internal calls, getter/setter methods	▶	4/4	0/4	0/4	0/4	0/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	4/4	0/4	4/4	4/4	4/4
assignments allowed in for-in head in non-strict mode		Yes	Yes	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Developer Experience

> **В IDE слабый автокомплит и рефакторинги**

```
'use strict';

function hello(name) {
  console.log(name);
}
```

```
(parameter) name: any
```

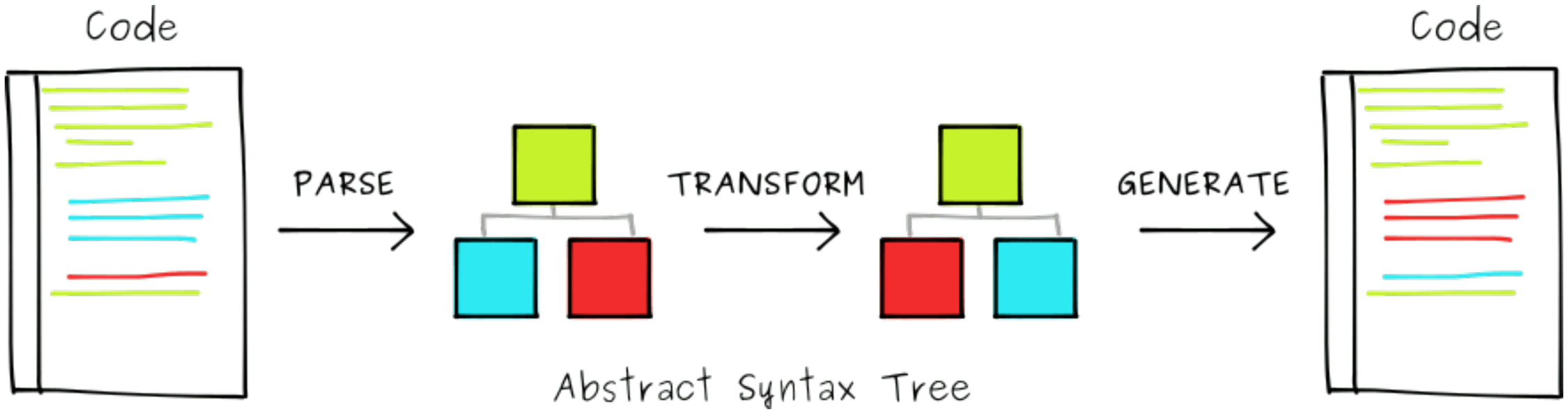
02

Транспиляция

- Общие принципы
- Kotlin
- Babel
- TypeScript

Транспиляция

Это процесс перевода исходного кода одного языка в другой



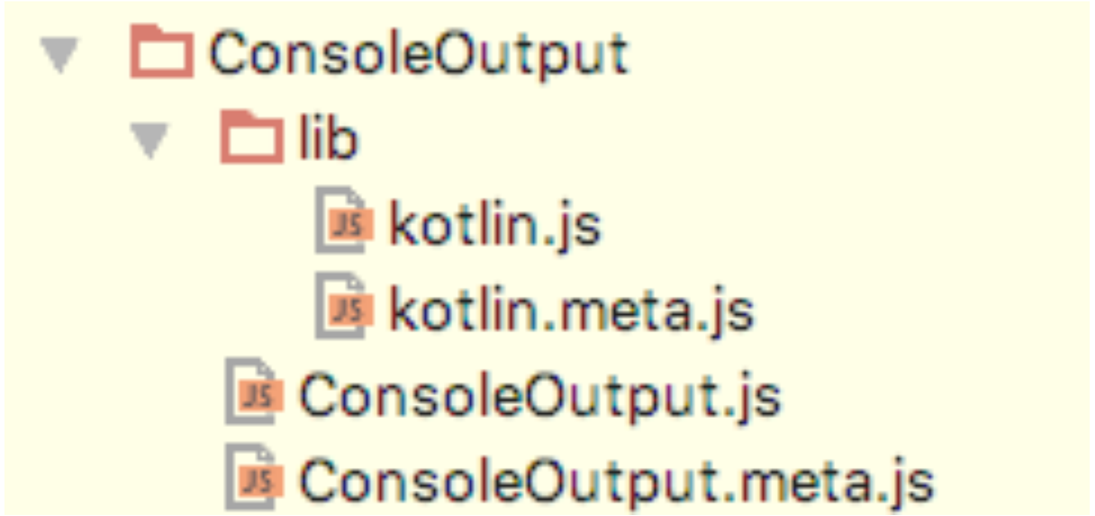
Транспиляция

Это процесс перевода исходного кода одного языка в другой

Kotlin/JS

src > ConsoleOutput.kt

```
1  fun main(args: Array<String>) {
2      println("Hello JavaScript!")
3  }
```




BABEL

Пример работы Babel

```
class Student {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  setSkills(skills = []) {
    this.skills = skills;
  }
}
```

Пример работы Babel

```
var Student = (function() {
  function Student(firstName, lastName) {
    _classCallCheck(this, Student);

    this.firstName = firstName;
    this.lastName = lastName;
  }

  _createClass(Student, [
    {
      key: 'setSkills',
      value: function setSkills() {
        var skills = arguments.length > 0 && arguments[0] !== undefined ? arguments[0] : [];

        this.skills = skills;
      }
    }
  ]);

  return Student;
})();
```

Система плагинов

Plugins

- for-of
- classes
- destructuring
- arrow-functions
- ...


TS

Автор языка

Андерс Хейлсберг
Известен также по C#, Turbo Pascal, Delphi

TypeScript это ...

- Статически типизированный язык
- Сильно типизированный язык
- Явно типизированный язык с возможностью вывода типов

```
// Транспилятор знает тип
const i: number = 10;

// Транспилятор не допускает неявных приведений
const result = 5 + []; // Error

// Транспилятор сам выводит типы данных
const stringObject1: String = new String();
const stringObject2 = new String();
```

Возможности

- Поддержка ES2015-ESNext
- Аннотации типов и их проверка
- Вывод типов
- Интерфейсы
- Обобщённые типы
- Кортежи
- Декораторы

Поддержка транспиляторами

03

Введение в TypeScript

- Объявление переменных
- Boolean, Number, String
- Null, Undefined
- Массивы и кортежи
- Перечисления (Enum)
- Void, Never
- Any, Unknown
- Object
- "Приведение" типов

Объявление переменных, примитивы

Используйте const и иногда let, никогда var

```
// Boolean
const isDone: boolean = false;

// Number
const decimal: number = 6;
const hex: number = 0xf00d;
const binary: number = 0b1010;
const octal: number = 0o744;

// String
let color: string = 'blue';
color = 'red';

const fullName: string = `Evgeny Markov`;
const sentence: string = `Hello, my name is ${fullName}`;
```

Объявление переменных, примитивы

Можно использовать объединения типов

```
let testNull: null = null;

let operation: string | null = 'fetch';
operation = null;

let state: string | undefined;
state = 'succeed';
```

Массивы

Предпочитайте запись ItemType[]

```
// Simple
let list1: number[] = [1, 2, 3];

// With generics
let list2: Array<number> = [1, 2, 3];
```

Кортежи

Кортежи основаны на массивах

```
// Объявление кортежа
let point: [number, number];

// Некорректная инициализация
point = [10, 'hello']; // Error

// Инициализация
point = [20, 10]; // OK

// Обращение
point[1]; // 10

// Деструктуризация
const [first, second] = point;
```

Перечисления

Это набор именованных числовых или строковых констант

```
enum Color {
  Red,
  Green,
  Blue
}

const c: Color = Color.Green;
```

Перечисления

Это набор именованных числовых или строковых констант

```
enum NumericColor {
  Red = 5,
  Green = 7,
  Blue = 9
}

enum StringColor {
  Red = '#F00',
  Green = '#0F0',
  Blue = '#00F'
}
```

Void и Never

Используйте void когда функция не имеет return

Используйте never когда функция никогда не завершит работу

```
function warnUser(): void {
  console.log('This is my warning message');
}

function error(): never {
  throw new Error();
}

function infiniteLoop(): never {
  while (true) {}
}
```

Any

Используйте any только для прототипирования

```ts
let notSure: any = 4;
notSure = 'maybe a string instead';
notSure = false;

let notSure: any = 4;
notSure.ifItExists(); // Ошибка в рантайме
notSure.toFixed(); // Всё ок, toFixed есть в рантайме

let prettySure: object = 4;
prettySure.toFixed(); // toFixed нет у object
```

Unknown

Используйте для работы с данными неизвестного формата

```
function workWithData(data: unknown) {
  if (typeof data === 'string' || typeof data === 'number') {
    data; // string | number
  }

  if (data instanceof Error) {
    data; // Error
  }

  // ...
}
```

Object

TypeScript позволяет указывать "форму" (shape) объекта

```ts
const colors: object = {
  red: '#F00',
  green: '#0F0',
  blue: '#00F'
};
```

```ts
const settings: {
  color: string;
  delay: number;
  retry: boolean;
} = {
  color: '#F00',
  delay: 2000,
  retry: false
};
```

Приведение типов

Пользуйтесь записью с "as" (из-за JSX синтаксиса)

```
const someValue: any = 'this is a string';

const strLength1: number = (<string>someValue).length;
const strLength2: number = (someValue as string).length;
```

04

Функции

- Объявление
- Вывод типов
- Опциональные параметры
- Значения по умолчанию
- Остаточные параметры

Объявление функций

```
function declarationAdd(x: number, y: number): number {
  return x + y;
}

const expressionAdd = function(x: number, y: number): number {
  return x + y;
};

const arrowAdd = (x: number, y: number): number ⟹ x + y;
```

Опциональные параметры

```
function buildName(firstName: string, lastName?: string): string {
  if (lastName) {
    return `${firstName} ${lastName}`;
  }

  return firstName;
}

const result1 = buildName('Evgeny'); // Работает правильно
const result3 = buildName('Evgeny', 'Markov'); // Тоже правильно
const result2 = buildName('Evgeny', 'Markov', 'Sr.'); // Error, слишком много параметров
```

> Порядок важен! Опциональные только в конце.

Значения по умолчанию

```
function buildName(firstName = 'Evgeny', lastName: string) {
  return `${firstName} ${lastName}`;
}

const result1 = buildName('Evgeny'); // Error, слишком мало аргументов
const result2 = buildName('Evgeny', 'Markov', 'Sr.'); // Error, слишком много аргументов
const result3 = buildName('Evgeny', 'Markov'); // Ok, вернёт 'Evgeny Markov'
const result4 = buildName(undefined, 'Markov'); // Ok, вернёт 'Evgeny Markov'
```

> Порядок не важен.

Остаточные параметры

```
function buildName(firstName: string, ...restOfName: string[]) {
  return `${firstName} ${restOfName.join(' ')}`;
}

// 'Joseph Samuel Lucas MacKinzie'
const employeeName = buildName('Joseph', 'Samuel', 'Lucas', 'MacKinzie');
```

05

Классы

- Объявление
- Наследование
- Модификаторы доступа
- Модификатор Readonly
- Parameter Properties
- get и set
- Статические поля и методы
- Абстрактные классы

Объявление

```
class Greeter {
  greeting: string;

  constructor(message: string) {
    this.greeting = message;
  }

  greet() {
    return `Hello, ${this.greeting}`;
  }
}

const greeter = new Greeter('world');
```

Наследование

Для наследования используется ключевое слово extends

```
class Animal {
  move(distanceInMeters: number = 0) {
    console.log(`Animal moved ${distanceInMeters}m.`);
  }
}

class Dog extends Animal {
  bark() {
    console.log('Woof! Woof!');
  }
}

const dog = new Dog();
dog.bark();
dog.move(10);
dog.bark();
```

Наследование

> **Если объявили конструктор – обязательно вызовите super()**

```
class Animal {
  name: string;

  constructor(theName: string) {
    this.name = theName;
  }

  move(distanceInMeters: number = 0) {
    console.log(`${this.name} ...`);
  }
}
```

```
class Snake extends Animal {
  constructor(name: string) {
    super(name);
  }

  move(distanceInMeters = 5) {
    console.log('Slithering...');
    super.move(distanceInMeters);
  }
}
```

Модификаторы доступа

Их три: public, private, protected. По умолчанию public.

```
class Animal {
  public name: string;

  public constructor(theName: string) {
    this.name = theName;
  }

  public move(distanceInMeters: number) {
    console.log(`${this.name} moved ${distanceInMeters}m.`);
  }
}
```

Модификаторы доступа

private разрешает доступ только внутри класса

```
class Animal {
  private name: string;

  constructor(theName: string) {
    this.name = theName;
  }
}

new Animal('Cat').name; // Error: 'name' is private
```

Модификаторы доступа

> **protected разрешает доступ внутри класса и в наследниках**

```
class Person {
  protected name: string;

  constructor(name: string) {
    this.name = name;
  }
}
```

```
class Employee extends Person {
  private department: string;

  constructor(name: string, department: string) {
    super(name);
    this.department = department;
  }

  public greet() {
    console.log(`Hello, I'm ${this.name}`);
    console.log(`I'm working at ${this.department}`);
  }
}
```

```
const howard = new Employee('Howard', 'Sales');
console.log(howard.greet());
console.log(howard.name); // Error, 'name' is protected
```

Модификатор Readonly

Разрешает менять поля только в пределах конструктора

```
class Octopus {
  readonly name: string;
  readonly numberOfLegs: number = 8;

  constructor(theName: string) {
    this.name = theName;
  }
}

const cat = new Octopus('Cat with the 8 strong legs');
cat.name = 'OctoCat'; // Error, 'name' is readonly
```

Parameter Properties

Позволяют объявлять и инициализировать поля в одном месте

```
class Octopus {
  // private readonly name: string;
  // private readonly numberOfLegs: number = 8;

  constructor(
    private readonly name: string,
    private readonly numberOfLegs: number = 8,
  ) {}

  greet() {
    console.log(`Hi, I'm ${this.name}`);
    console.log(`I have ${this.numberOfLegs} legs`);
  }
}
```

get и set

Дают больший контроль за доступом к полям

```
const FULL_NAME_MAX_LENGTH = 10;

class Employee {
  constructor(private _fullName: string) {}

  get fullName(): string {
    return this._fullName;
  }

  set fullName(newName: string) {
    if (newName && newName.length > FULL_NAME_MAX_LENGTH) {
      throw new Error('fullName has a max length of ' + FULL_NAME_MAX_LENGTH);
    }

    this._fullName = newName;
  }
}
```

Статические поля и методы

```
class Grid {
  static origin = { x: 0, y: 0 };

  constructor(public scale: number) {}

  calculateDistanceFromOrigin(point: { x: number; y: number }) {
    const xDist = point.x - Grid.origin.x;
    const yDist = point.y - Grid.origin.y;

    return Math.sqrt(xDist * xDist + yDist * yDist) / this.scale;
  }
}
```

Абстрактные поля и классы

```
abstract class Department {
  constructor(public name: string) {}

  printName(): void {
    console.log('Department name: ' + this.name);
  }

  abstract printMeeting(): void; // Должен быть реализован в дочерних классах
}

class AccountingDepartment extends Department {
  constructor() {
    super('Accounting and Auditing');
  }

  printMeeting(): void {
    console.log('The Accounting Department meets each Monday at 10am.');
  }
}
```

06 LIVE

Интерфейсы

- Объявление
- Расширение
- Реализация интерфейсов
- Опциональные свойства
- Readonly свойства
- Интерфейсы для объектов

07

Новые возможности

- Optional Chaining
- Nullish Coalescing

Optional Chaining

Позволяет безопасно обращаться к свойствам

```
let foo:
  | {
      bar: { baz: () ⇒ void };
    }
  | undefined;

const x = foo?.bar.baz(); // void | undefined

// Эквивалентная запись
let x = foo === null || foo === undefined ? undefined : foo.bar.baz();
```

Optional Chaining

Заменяет длинные логические выражения

```
// До
if (foo && foo.bar && foo.bar.baz) {
  // ...
}

// После
if (foo?.bar?.baz) {
  // ...
}
```

Optional Chaining

Позволяет получать элементы по индексу если массив объявлен

```
function tryGetFirstNumber(numbers?: number[]) {
  return numbers?.[0];
  // Эквивалентная запись:
  //   return (numbers === null || numbers === undefined) ?
  //       undefined :
  //       numbers[0];
}
```

Optional Chaining

> **Позволяет вызывать функцию если она определена**

```
async function logEvent(event: string, log?: (message: string) ⇒ void) {
  log?.(`Event ${event} occurred at ${new Date().toISOString()}`);
  // Эквивалентная запись:
  //   if (log =/= null && log =/= undefined) {
  //     log(`Event ${event} occurred at ${new Date().toISOString()}`);
  //   }
}
```

Nullish Coalescing

Более безопасная замена foo || bar

```
let foo;
const bar = () ⇒ {
  console.log('Hello');
};

const x = foo ?? bar();

let x = foo ≠≠ null && foo ≠≠ undefined ? foo : bar();
```

Nullish Coalescing

> **Учитывает только null и undefined, но не "", NaN, 0**

```
// localStorage.volume === 0
const wrongVolume = localStorage.volume || 0.5;
const rightVolume = localStorage.volume ?? 0.5;
```

08 LIVE

Конфигурация проекта

- Компилятор
- Конфигурационный файл
- Результаты компиляции

Что почитать?

- 🇺🇸 TypeScript Handbook
- 🇺🇸 TypeScript Deep Dive

Что посмотреть?

- 🇷🇺 Андрей Морозов (Яндекс) — Типизация
- 🇷🇺 Андрей Старовойт (JetBrains) — Эволюция TypeScript
- 🇷🇺 Константин Кичинский (Microsoft) — Разработка на TypeScript

Яндекс

Спасибо

Евгений Марков

Разработчик интерфейсов

 evgenymarkov@yandex-team.ru

 @evgenymarkov