



Web-технологии

Основы TypeScript.
Статический контроль с Flow

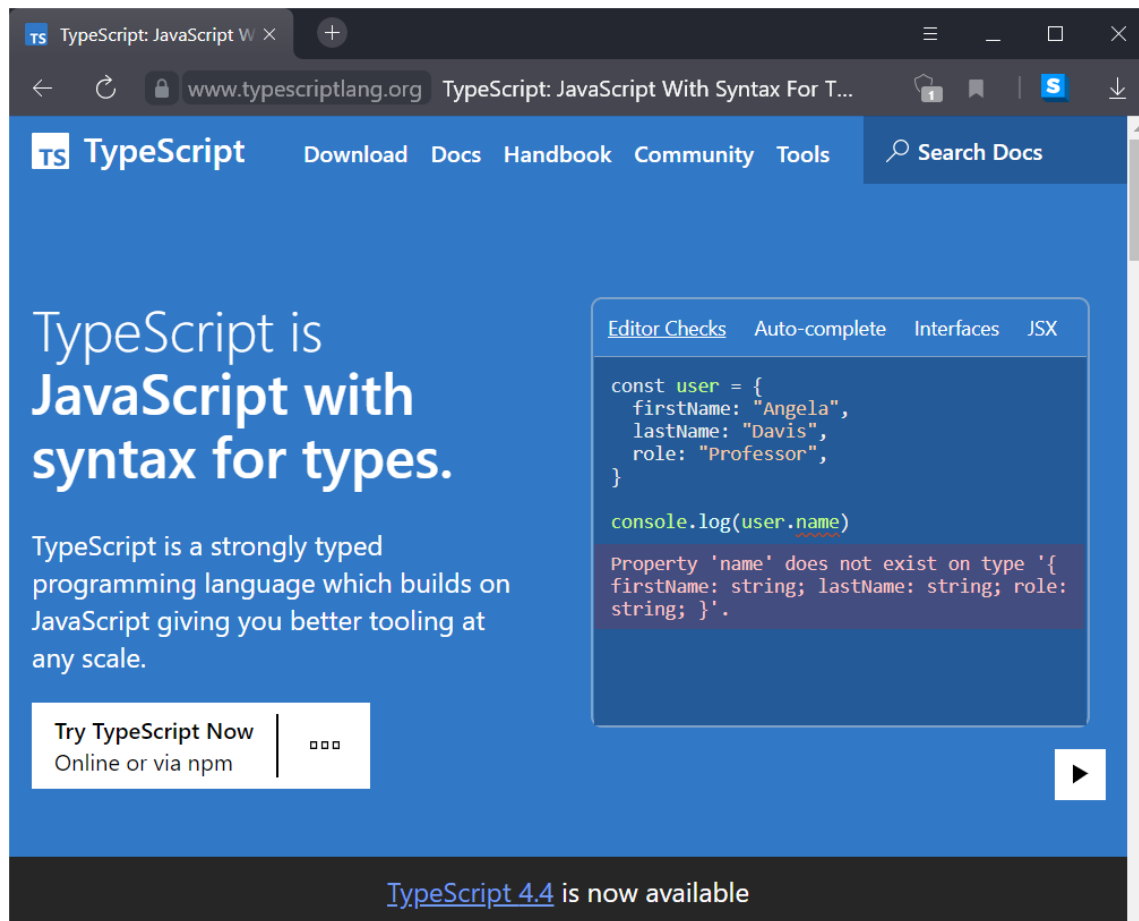
- Основы TypeScript
 - типы переменных, интерфейсы, классы, функции, шаблоны, перечисления, импорт-экспорт, пространства имён
 - конфигурации
 - JSX
- Статический контроль в JavaScript с использованием Flow
 - настройка среды исполнения, использование из командной строки
 - проверка типов переменных
 - проверка параметров функций
 - generics
 - перечислимые типы

<https://www.typescriptlang.org/>
<https://www.typescripttutorial.net/>
<https://metanit.com/web/typescript/>
<https://flow.org/>
<https://flow.org/try/>

TypeScript

3

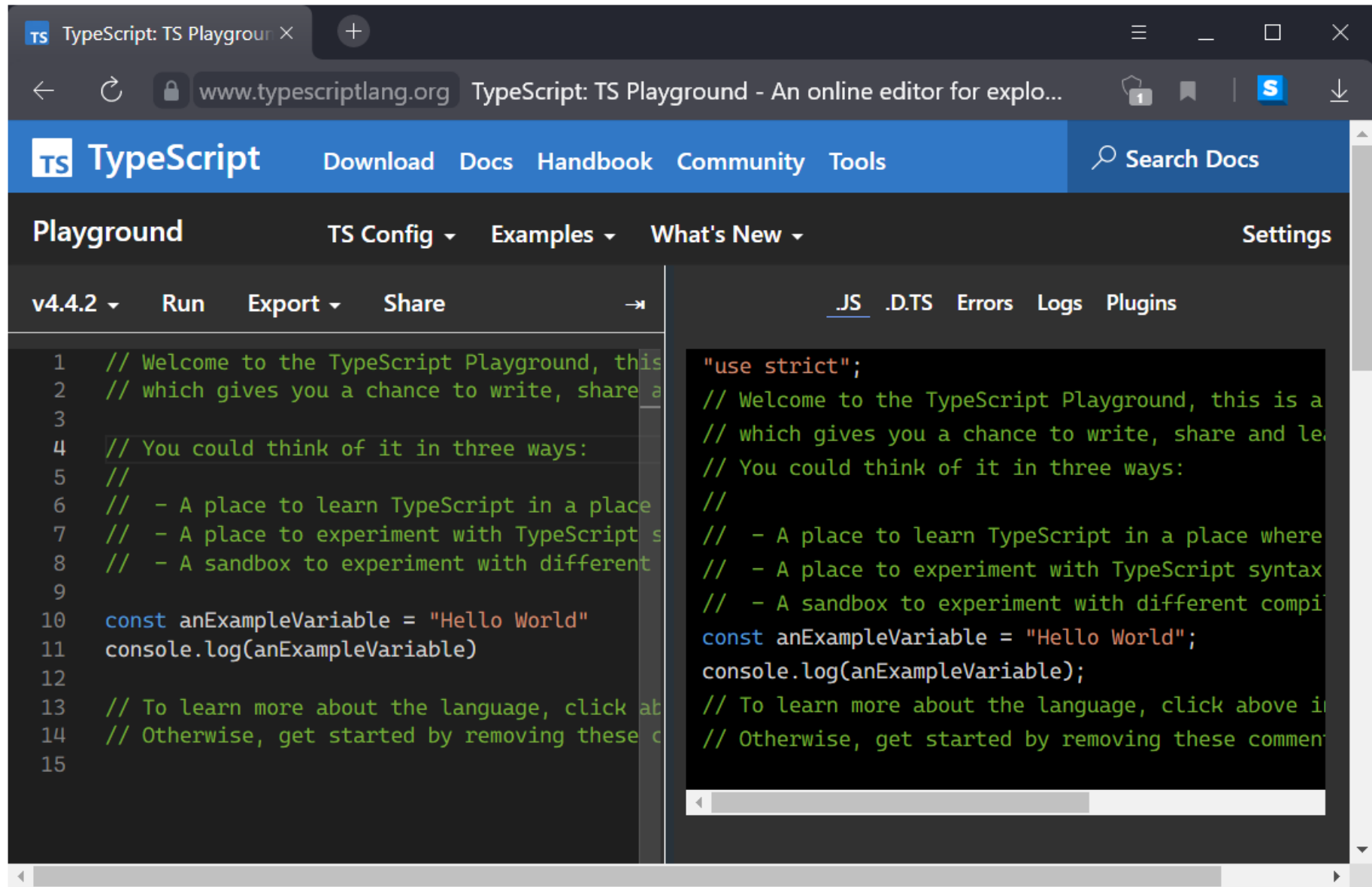
- Статическая типизация
- Лучшая поддержка в IDE
- Доступ к НОВЫМ ВОЗМОЖНОСТЯМ ECMAScript



What is TypeScript?

`npm install -g typescript`

<https://www.typescriptlang.org/>



The screenshot shows the TypeScript Playground interface in a web browser. The browser's address bar displays the URL www.typescriptlang.org. The page features a blue header with the TypeScript logo and navigation links: Download, Docs, Handbook, Community, and Tools. A search bar labeled "Search Docs" is also present. Below the header, the "Playground" tab is active, showing a code editor with a TypeScript file. The code includes a multi-line comment and a `const` declaration. The right-hand pane displays the output, which includes the `"use strict";` directive and the same comment and `console.log` statement as the input code. The interface includes standard browser controls and a sidebar for file management and settings.

TS TypeScript: TS Playgrou ×

← ↻ 🔒 www.typescriptlang.org TypeScript: TS Playground - An online editor for explo...

TS TypeScript Download Docs Handbook Community Tools Search Docs

Playground TS Config ▾ Examples ▾ What's New ▾ Settings

v4.4.2 ▾ Run Export ▾ Share →

```
1 // Welcome to the TypeScript Playground, this
2 // which gives you a chance to write, share a
3
4 // You could think of it in three ways:
5 //
6 // - A place to learn TypeScript in a place
7 // - A place to experiment with TypeScript s
8 // - A sandbox to experiment with different
9
10 const anExampleVariable = "Hello World"
11 console.log(anExampleVariable)
12
13 // To learn more about the language, click at
14 // Otherwise, get started by removing these c
15
```

.JS .D.TS Errors Logs Plugins

```
"use strict";
// Welcome to the TypeScript Playground, this is a
// which gives you a chance to write, share and le
// You could think of it in three ways:
//
// - A place to learn TypeScript in a place where
// - A place to experiment with TypeScript syntax
// - A sandbox to experiment with different compi
const anExampleVariable = "Hello World";
console.log(anExampleVariable);
// To learn more about the language, click above in
// Otherwise, get started by removing these comment
```

<https://www.typescriptlang.org/play>

Hello world (1) :string

5

type01.ts

```
function greeter(person: string) {  
    return "Hello, " + person;  
}  
  
let user = "Jane User";  
document.body.innerHTML = greeter(user);
```



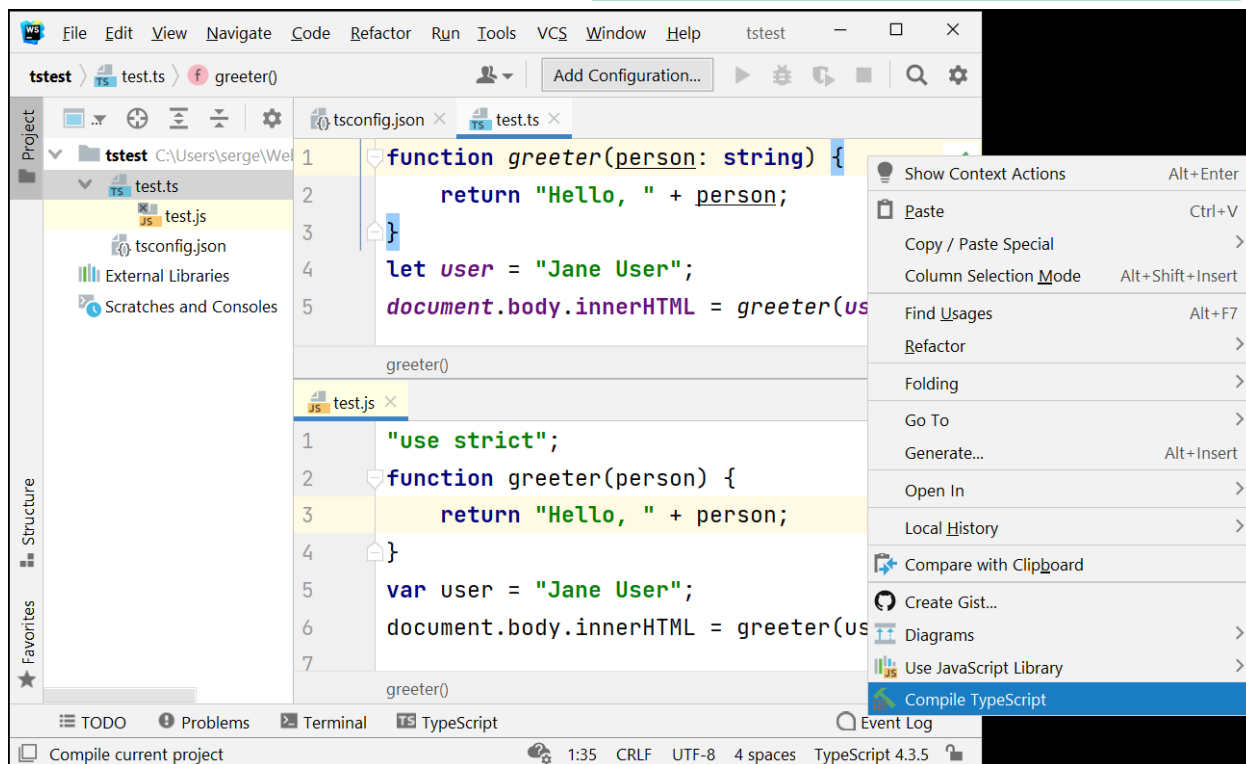
type01.js

```
"use strict";  
function greeter(person) {  
    return "Hello, " + person;  
}  
  
var user = "Jane User";  
document.body.innerHTML = greeter(user);
```

tsc type01.ts

В данном случае
в tsconfig.json
указано ES5,
поэтому JS
соответствует
ES5

tsc --init
tsconfig.json



Hello World (2) interface, :Person

type02.ts

```
interface Person {  
  firstName: string;  
  lastName: string;  
}  
  
function greeter(person: Person) {  
  return "Hello, " + person.firstName + " " + person.lastName;  
}  
  
let user = { firstName: "Jane", lastName: "User" };  
document.body.innerHTML = greeter(user);
```

type02.js

```
"use strict";  
  
function greeter(person) {  
  return "Hello, " + person.firstName + " " + person.lastName;  
}  
  
var user = { firstName: "Jane", lastName: "User" };  
document.body.innerHTML = greeter(user);
```

Hello World (3) Student, Person

7

type03.ts

```
class Student {
  fullName: string;
  constructor(public firstName: string, public middleInitial: string, public lastName: string) {
    this.fullName = firstName + " " + middleInitial + " " + lastName;
  }
}

interface Person {
  firstName: string;
  lastName: string;
}

function greeter(person : Person) {
  return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("Jane", "M.", "User");
document.body.innerHTML = greeter(user);
```

type03.js

```
"use strict";
var Student = /** @class */(function () {
  function Student(firstName, middleInitial, lastName) {
    this.firstName = firstName;
    this.middleInitial = middleInitial;
    this.lastName = lastName;
    this.fullName = firstName + " " + middleInitial + " " + lastName;
  }
  return Student;
})();

function greeter(person) {
  return "Hello, " + person.firstName + " " + person.lastName;
}

var user = new Student("Jane", "M.", "User");
document.body.innerHTML = greeter(user);
```

Типы переменных

8

type04.ts

// Boolean

let *isDone*: boolean = false;

// Number

let *decimal*: number = 6;

let *hex*: number = 0xf00d;

let *binary*: number = 0b1010;

let *octal*: number = 0o744;

// String

let *color*: string = "blue";

// Array

let *list1*: number[] = [1, 2, 3];

let *list2*: Array<number> = [1, 2, 3];

// Declare a tuple type

let *x*: [string, number];

x = ["hello", 10]; *// Initialize it*

// Enum

enum Color {*Red, Green, Blue*}

let *c*: Color = Color.*Green*;

// Any

let *notSure*: any = 4;

notSure = "maybe a string instead";

notSure = false;

let *list3*: any[] = [1, true, "free"];

// Not much else we can assign to these variables!

let *u*: undefined = undefined;

let *n*: null = null;

"use strict";

// Boolean

var isDone = false;

// Number

var decimal = 6;

var hex = 0xf00d;

var binary = 10;

var octal = 484;

// String

var color = "blue";

// Array

var list1 = [1, 2, 3];

var list2 = [1, 2, 3];

// Declare a tuple type

var x;

x = ["hello", 10]; *// Initialize it*

// Enum

var Color;

(function (Color) {

Color[Color["Red"] = 0] = "Red";

Color[Color["Green"] = 1] = "Green";

Color[Color["Blue"] = 2] = "Blue";

})(Color || (Color = {}));

var c = Color.Green;

// Any

var notSure = 4;

notSure = "maybe a string instead";

notSure = false;

var list3 = [1, true, "free"];

// Not much else we can assign to these variables!

var u = undefined;

var n = null;

Дополнительные типы

9

// void - функция не возвращает значение

```
function log(message: string): void {  
    console.log(message);  
}
```

let useless: void = undefined; *// бесполезная*

// useless = 1; // TS2322: Type 'number' is not assignable to type 'void'

// never - не содержит типа

```
function raiseError(message: string): never {  
    // return true // TS2322: Type 'boolean' is not assignable to type 'never'.  
    throw new Error(message);  
}  
function reject() {  
    return raiseError('Rejected');  
}
```

// Объединение типов

let result: number | string;

result = 42;

result = 'Hi world';

// result = false; // TS2322: Type 'boolean' is not assignable to type 'string | number'.

Литеральные типы, алиасы

```
let click: 'click';
```

```
click = 'click';
```

```
// TS2322: Type '"double click"' is not assignable to type '"click"'
```

```
// click = 'double click';
```

```
let mouseEvent: 'click' | 'mouseup' | 'mousedown';
```

```
mouseEvent = 'click';
```

```
mouseEvent = 'mouseup';
```

```
mouseEvent = 'mousedown';
```

```
// TS2322: Type '"mouseover"' is not assignable to type '"click" | "mouseup" | "mousedown"'
```

```
// mouseEvent = 'mouseover';
```

```
// Алиас
```

```
type MyEvent = 'click' | 'mouseup' | 'mousedown';
```

```
let me: MyEvent;
```

```
me = 'click';
```

```
// TS2322: Type '"mouseover"' is not assignable to type 'MyEvent'.
```

```
// me = 'mouseover';
```

```
// Можно присвоить другой переменной
```

```
let another: MyEvent;
```

object vs. Object

- **object** представляет собой все значения, которые не являются примитивными типами
 - **number, bigint, string, boolean, null, undefined, symbol**

```
let student: object;  
student = {  
  fio: "Иван"  
}  
console.log(student) // { fio: 'Иван' }  
// student = "Иван"  
// TS2322: Type 'string' is not assignable to type 'object'.
```

- **Object** предоставляет функциональность всех объектов
 - **toString(), valueOf()...**
- Пустой тип **{}** не позволяет добавлять атрибуты

Работа с переменными

```
let input = [1, 2];
let [first, second] = input;
console.log(first); // outputs 1
console.log(second); // outputs 2
```

```
function f([first, second]: [number, number]) {
  console.log(first); // outputs 1
  console.log(second); // outputs 2
}
f([1, 2]);
```

```
let [one, ...rest] = [1, 2, 3, 4];
console.log(one); // outputs 1
console.log(rest); // outputs [ 2, 3, 4 ]
```

```
let o = {
  a: "foo",
  b: 12,
  c: "bar"
};

let { a, b } = o;
let { a: newName1, b: newName2 } = o;
let { a, b }: { a: string, b: number } = o;
function f({ a, b } = { a: "", b: 0 }): void {
  // ...
}

f(); // ok, default to { a: "", b: 0 }
```

var

let

const

Интерфейсы

13

// Необязательные атрибуты

```
interface SquareConfig {  
    color?: string;  
    width?: number;  
}
```

// Неизменяемые атрибуты

```
interface Point {  
    readonly x: number;  
    readonly y: number;  
}
```

// Функции

```
interface SearchFunc {  
    (source: string, subString: string): boolean;  
}
```

```
let mySearch: SearchFunc;
```

```
mySearch = function(source: string, subString: string) {  
    let result = source.search(subString);  
    return result > -1;  
}
```

// Индексируемые атрибуты

```
interface StringArray {  
    [index: number]: string;  
}
```

```
let myArray: StringArray;
```

```
myArray = ["Bob", "Fred"];
```

```
let myStr: string = myArray[0];
```

Реализация интерфейсов

14

```
// Реализация свойств
interface ClockInterface {
    currentTime: Date;
}
class Clock implements ClockInterface {
    currentTime: Date;
    constructor(h: number, m: number) { }
}
```

```
// Реализация методов
interface ClockInterface2 {
    currentTime: Date;
    setTime(d: Date);
}
class Clock2 implements ClockInterface2 {
    currentTime: Date;
    setTime(d: Date) {
        this.currentTime = d;
    }
    constructor(h: number, m: number) { }
}
```

```
// Определение конструктора
interface ClockConstructor3 {
    new (hour: number, minute: number);
}
// noinspection JSAnnotator
class Clock3 implements ClockConstructor3 {
    currentTime: Date;
    constructor(h: number, m: number) { }
}
```

```
// Расширение интерфейсов
interface Shape {
    color: string;
}
interface Square extends Shape {
    sideLength: number;
}
// Создание экземпляра класса
let square = <Square>{};
square.color = "blue";
square.sideLength = 10;
```

Типы / type, interface

```

type BirdType = { wings: 2; } // тип
interface BirdInterface { wings: 2; } // интерфейс
const bird1: BirdType = { wings: 2 };
const bird2: BirdInterface = { wings: 2 };
// Их можно "перемешивать"
const bird3: BirdInterface = bird1;
// Оба поддерживают расширение друг через друга
type Owl = { nocturnal: true } & BirdType;
type Robin = { nocturnal: false } & BirdInterface;
interface Peacock extends BirdType {
  colourful: true;
  flies: false;
}
interface Chicken extends BirdInterface {
  colourful: false;
  flies: false;
}
let owl: Owl = { wings: 2, nocturnal: true };
let chicken: Chicken = { wings: 2, colourful: false, flies: false };
/* Рекомендуются интерфейсы - они дают более внятные сообщения об ошибке */
/* Отличие интерфейсов от типов: интерфейсы «открыты» и могут быть
расширены путём повторного объявления */
interface Kitten { purrs: boolean; }
interface Kitten { colour: string; }

```

```

"use strict";
var bird1 = { wings: 2 };
var bird2 = { wings: 2 };
var bird3 = bird1;
var owl = { wings: 2,
  nocturnal: true };
var chicken = { wings: 2,
  colourful: false, flies: false };

```

Карты (map) на основе type

// Создаем ассоциативный тип

```
type User = {  
  id: number  
  username: string  
  email: string  
}
```

// Создаем объект `user`, соответствующий ассоциативному типу

```
const user: User = {  
  id: 42,  
  username: 'Superman',  
  email: 's@man.com',  
}
```

- Обычно используются для определения связи между ключами и значениями
- Нельзя создать не определив значения всех ключей

Объединение типов "&"

17

```
type Man = {  
  fio: string  
}  
type User = {  
  login: string,  
  email: string  
}  
type Student = Man & User & {  
  group: number  
}  
let student: Student = {  
  fio: "Иван",  
  login: "ivan",  
  email: "ivan@gmail.com",  
  group: 9383,  
  // pwd: ""  
  /* TS2322: Type '{ fio: string; login: string; email: string; group: number;  
  pwd: string; }' is not assignable to type 'Student' */  
}
```

Исключающие объединения “|”

```
type Member = {  
  type: 'member'  
  role: string  
}  
  
type Admin = {  
  type: 'admin'  
  rights: string[]  
}  
  
type User = Member | Admin  
function getFirstRole(u: User) {  
  if (u.type === 'member') {  
    // u.rights - TS2339: Property 'rights' does not exist on type 'Member'.  
    return u.role  
  }  
  // u.role - TS2339: Property 'role' does not exist on type 'Admin'.  
  return u.rights[0]  
}
```

Классы (1) extends, super, наследование

19

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

class Snake extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 5) {
        console.log("Slithering...");
        super.move(distanceInMeters);
    }
}

class Horse extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 45) {
        console.log("Galloping...");
        super.move(distanceInMeters);
    }
}

let sam = new Snake("Sammy the Python");
let tom: Animal = new Horse("Tommy the Palomino");

sam.move();
tom.move(34);
```

Классы (2) – спецификаторы доступа

```
class Animal {  
    private name: string;  
    public constructor(theName: string) { this.name = theName; }  
}
```

```
class Rhino extends Animal {  
    public constructor() { super("Rhino"); }  
}
```

```
class Employee {  
    private name: string;  
    public constructor(theName: string) { this.name = theName; }  
}
```

```
let animal = new Animal("Goat");  
let rhino = new Rhino();  
let employee = new Employee("Bob");
```

```
animal = rhino;
```

public

private

Классы (3) – получение и установка СВОЙСТВ

```
let passcode = "secret passcode";
class Employee {
    private _fullName: string;

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

    set fullName(newName: string) {
        if (passcode && passcode == "secret passcode") {
            this._fullName = newName;
        }
        else {
            console.log("Error: Unauthorized update of employee!");
        }
    }
}

let employee = new Employee();
employee.fullName = "Bob Smith";
if (employee.fullName) {
    console.log(employee.fullName);
}
```

set

get

Классы (4) static, статические СВОЙСТВА

```
class Grid {
  static origin = {x: 0, y: 0};
  calculateDistanceFromOrigin(point: {x: number; y: number;}) {
    let xDist = (point.x - Grid.origin.x);
    let yDist = (point.y - Grid.origin.y);
    return Math.sqrt(xDist * xDist + yDist * yDist) / this.scale;
  }
  constructor (public scale: number) { }
}

let grid1 = new Grid(1.0); // 1x scale
let grid2 = new Grid(5.0); // 5x scale

console.log(grid1.calculateDistanceFromOrigin({x: 10, y: 10}));
console.log(grid2.calculateDistanceFromOrigin({x: 10, y: 10}));
```

Классы (5) abstract, super, extends, абстрактные классы

```
abstract class Department {
    constructor(public name: string) {
    }
    printName(): void {
        console.log("Department name: " + this.name);
    }
    abstract printMeeting(): void; // must be implemented in derived classes
}
class AccountingDepartment extends Department {
    constructor() {
        super("Accounting and Auditing"); // constructors in derived classes must call super()
    }
    printMeeting(): void {
        console.log("The Accounting Department meets each Monday at 10am.");
    }
    generateReports(): void {
        console.log("Generating accounting reports...");
    }
}

let department = new AccountingDepartment(); // ok to create and assign a non-abstract subclass
department.printName();
department.printMeeting();
```

Атрибуты только для чтения / readonly

Вариант 1

```
class Person {  
  readonly birthDate: Date; // Только для чтения  
  constructor(birthDate: Date) {  
    this.birthDate = birthDate;  
  }  
}  
  
let person = new Person(new Date(1990, 12, 25));  
// TS2540: Cannot assign to 'birthDate' because it is a read-only property.  
// person.birthDate = new Date(1991, 12, 25);
```

Вариант 2

```
class Person {  
  constructor(readonly birthDate: Date) {  
    this.birthDate = birthDate;  
  }  
}  
  
let person = new Person(new Date(1990, 12, 25));  
// TS2540: Cannot assign to 'birthDate' because it is a read-only property.  
// person.birthDate = new Date(1991, 12, 25);
```


ФУНКЦИИ

25

```
// Не обязательные параметры
function buildName(firstName: string, lastName?: string) {
    if (lastName)
        return firstName + " " + lastName;
    else
        return firstName;
}
// Параметры по умолчанию
function buildName(firstName: string, lastName = "Smith") {
    return firstName + " " + lastName;
}
```

```
// Именованная функция
```

```
function add1(x, y) {
    return x + y;
}
```

```
// Анонимная функция
```

```
let myAdd1 = function(x, y) { return x + y; };
```

```
// Функции с определёнными типами
```

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

```
let myAdd2 = function(x: number, y: number): number { return x + y; };
```

```
// С использованием стрелочных функций
```

```
let myAdd3: (x: number, y: number) => number =
    function(x: number, y: number): number { return x + y; };
```

```
// Идентично предыдущей строке
```

```
let myAdd4: (baseValue: number, increment: number) => number =
    function(x: number, y: number): number { return x + y; };
```

Перегрузка функций

```
function add(a: number, b: number): number;  
function add(a: string, b: string): string;  
function add(a: any, b: any): any {  
    return a + b;  
}
```

```
console.log(add(2, 3)); // 5  
console.log(add("a", "&b")); // a&b
```

// TS2393: Duplicate function implementation.

```
function sub(a: number): number {  
    return a - 1  
}
```

// TS2393: Duplicate function implementation.

```
function sub(a: string): string {  
    return a + 1  
}
```

Можно реализовать
через объединение
типов **number | string**

Защита с использованием typeof

27

```
function add(a: number | string, b: number | string) {  
  if (typeof a === 'number' && typeof b === 'number') {  
    return a + b;  
  }  
  if (typeof a === 'string' && typeof b === 'string') {  
    return a.concat(b);  
  }  
}  
  
console.log(add(2,3)) // 5  
console.log(add("a", "&b")) // a&b  
console.log(add(2, "&b")) // undefined
```

Защита с использованием instanceof²⁸

```
class A {  
  a: string  
  constructor(x: string) {  
    this.a = x  
  }  
}  
  
class B {  
  b: string  
  constructor(x: string) {  
    this.b = x  
  }  
}  
  
function print(x: A | B) {  
  if(x instanceof A)  
    console.log(x.a)  
  if(x instanceof B)  
    console.log(x.b)  
}  
  
print(new A("aaa")) // aaa  
print(new B("bbb")) // bbb  
// print("")
```

/ TS2345: Argument of type 'string' is not assignable to parameter of type 'A | B'. */*

- **typeof** – примитивные типы и определённые с помощью type
- **instanceof** – классы

Проверка на наличие поля / in

```
class A {  
    a: string  
    constructor(x: string) {  
        this.a = x  
    }  
}  
  
class B extends A {  
    b: string  
    constructor(x: string) {  
        super("")  
        this.b = x  
    }  
}  
  
function print(x: A | B) {  
    if("b" in x) // Безопасная проверка  
        console.log(x.b)  
    else {  
        console.log(x.a)  
        // console.log(x.b)  
        // TS2339: Property 'b' does not exist on type 'A'.  
    }  
}  
  
print(new A("aaa")) // aaa  
print(new B("bbb")) // bbb
```

Приведение типа

```
type typeA = {  
  e: string  
}  
type typeB = {  
  e: string  
}  
let a: typeA = { e: "" };  
let b = a as typeB; // Приведение типа  
let c = <typeB> a; // Приведение типа
```

Универсальные шаблоны (Generics)

31

// Дженерики для функции

```
function identity<T>(arg: T): T {  
    return arg;  
}  
let output = identity<string
```

// Дженерики для класса

```
class GenericNumber<T> {  
    zeroValue: T;  
    add: (x: T, y: T) => T;  
}  
let myGenericNumber = new GenericNumber<numbermyGenericNumber.zeroValue = 0;  
myGenericNumber.add = function(x, y) { return x + y; };
```

Преимущества TypeScript generics:

- использование проверки типов во время компиляции
- исключение нарушение типов
- позволяют реализовывать универсальные алгоритмы

Ограничения generics / extends

32

```
function merge<U extends object, V extends object>(obj1: U, obj2: V) {  
    return {  
        ...obj1,  
        ...obj2  
    }  
}  
  
let person = merge(  
    { name: 'Serge' },  
    { age: 42 }  
)  
  
console.log(person) // { name: 'Serge', age: 42 }  
// person = merge(  
//     { name: 'Serge' },  
//     42  
// )  
// TS2322: Type '{ name: string; } & object' is not assignable to type ...  
// TS2345: Argument of type 'number' is not assignable to parameter of type 'object'.
```


Классы generics / extends

```
class A {  
    print() {  
        console.log("Generic rules")  
    }  
}  
  
class B extends A {}  
class C {}  
class D<T> extends A {  
    constructor(x:T) {  
        x.print()  
    }  
}  
  
let d1 = new D(new A()); // Generic rules  
let d2 = new D(new B()); // Generic rules  
// TS2345: Argument of type 'C' is not assignable to parameter of type 'A'...  
// let d3 = new D(new C());
```

Интерфейсы generics

```
interface Pair<K, V> {  
    key: K;  
    value: V;  
    get(k: K): V;  
}  
  
let month: Pair<number, string> = {  
    key: 1,  
    value: 'Jan',  
    get(k) {  
        return `[${this.key}]=${this.value}`  
    }  
};  
  
console.log(month); // { key: 1, value: 'Jan', get: [Function: get] }  
console.log(month.get(month.key)); // [1]=Jan
```

Индексированный тип generics

```
interface Options<T> {  
  [name: string]: T  
}  
  
let options: Options<number> = {  
  'disabled': 1,  
  'enabled': 2  
};  
  
console.log(options['disabled']) // 1
```

Перечисления (enum)

// Перечисления

```
enum Direction1 {  
    // Начало отсчёта (не обязательно)  
    Up = 1,  
    Down,  
    Left,  
    Right,  
}
```

// Строковое перечисление

```
enum Direction2 {  
    Up = "UP",  
    Down = "DOWN",  
    Left = "LEFT",  
    Right = "RIGHT",  
}
```

// Разнотипное перечисление

```
enum BooleanLikeHeterogeneousEnum {  
    No = 0,  
    Yes = "YES",  
}
```

Цикл FOR (for ... in, for ... of)

```
let list = [4, 5, 6];
```

```
for (let i in list) {  
    console.log(i); // "0", "1", "2",  
}
```

```
for (let i of list) {  
    console.log(i); // "4", "5", "6"  
}
```

```
let pets = new Set(["Cat", "Dog", "Hamster"]);  
pets["species"] = "mammals";
```

```
for (let pet in pets) {  
    console.log(pet); // "species"  
}
```

```
for (let pet of pets) {  
    console.log(pet); // "Cat", "Dog", "Hamster"  
}
```

Работа с циклами:

- for
- while
- do...while
- break
- continue

ZipCodeValidator.ts

```
export interface StringValidator {  
    isAcceptable(s: string): boolean;  
}  
  
export const numberRegexp = /^[0-9]+$/;  
  
export class ZipCodeValidator implements StringValidator {  
    isAcceptable(s: string) {  
        return s.length === 5 && numberRegexp.test(s);  
    }  
}
```

```
class SomeValidator implements StringValidator {}  
  
export { SomeValidator };  
  
export { SomeValidator as mainValidator };
```

1. Импорт

```
import { ZipCodeValidator } from "../ZipCodeValidator";  
let myValidator = new ZipCodeValidator();
```
2. Именованный

```
import { ZipCodeValidator as ZCV } from "../ZipCodeValidator";  
let myValidator = new ZCV();
```
3. Всего

```
import * as validator from "../ZipCodeValidator";  
let myValidator = new validator.ZipCodeValidator();
```

Пространства имён, namespace

39

```
namespace Validation {  
    export interface StringValidator {  
        isAcceptable(s: string): boolean;  
    }  
    const num = /^[0-9]+$/;  
    export class ZipCodeValidator implements StringValidator {  
        isAcceptable(s: string) {  
            return s.length === 5 && num.test(s);  
        }  
    }  
}  
  
// Примеры строк  
let strings = ["Hello", "98052", "101"];  
// Создание валидатора  
let zip: Validation.StringValidator = new Validation.ZipCodeValidator();  
// Проверка каждой строки  
for (let s of strings) {  
    console.log(`${ s } - ${ zip.isAcceptable(s) ? "matches" : "does not match" } ZIP`);  
}  
  
// "Hello" - does not match ZIP  
// "98052" - matches ZIP  
// "101" - does not match ZIP
```

Декораторы класса

```
function sealed(constructor: Function) {  
    console.log("Вызов sealed")  
    Object.seal(constructor); // Запрет расширения класса  
    Object.seal(constructor.prototype);  
}  
  
@sealed  
class User {  
    name: string;  
    constructor(name: string){  
        this.name = name;  
    }  
}  
  
// Вызов sealed
```

Бывают декораторы:

- класса
- метода
- атрибута
- параметра

Вызов нескольких декораторов

41

```
function first() {  
    console.log("first(): factory evaluated");  
    return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {  
        console.log("first(): called");  
    };  
}  
  
function second() {  
    console.log("second(): factory evaluated");  
    return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {  
        console.log("second(): called");  
    };  
}  
  
class ExampleClass {  
    @first()  
    @second()  
    method() {}  
}  
  
// first(): factory evaluated  
// second(): factory evaluated  
// second(): called  
// first(): called
```

tsconfig.json (по умолчанию)

```
{
  "compilerOptions": {
    /* Visit https://aka.ms/tsconfig.json to read more about this file */
    /* Language and Environment */
    "target": "es5", /* Set the JavaScript language version for emitted JavaScript
and include compatible library declarations. */
    /* Modules */
    "module": "commonjs", /* Specify what module code is generated. */
    /* Interop Constraints */
    "esModuleInterop": true, /* Emit additional JavaScript to ease support for
importing CommonJS modules. This enables `allowSyntheticDefaultImports` for
type compatibility. */
    "forceConsistentCasingInFileNames": true, /* Ensure that casing is correct
in imports. */
    /* Type Checking */
    "strict": true, /* Enable all strict type-checking options. */
    /* Completeness */
    "skipLibCheck": true /* Skip type checking all .d.ts files. */
  }
}
```

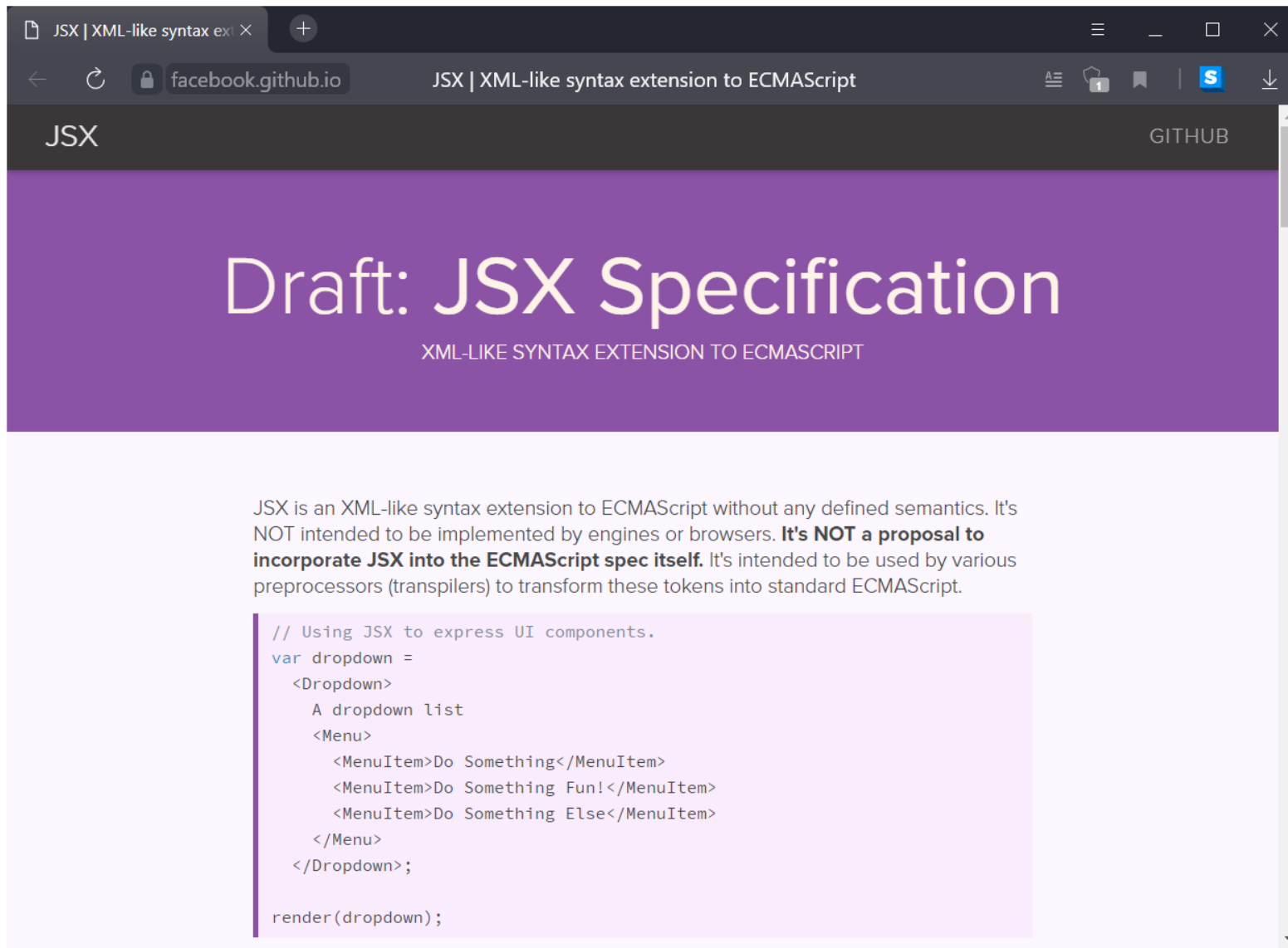
Здесь ***.d.ts** – декларативные (заголовочные) файлы
(объявляются через **declare**, используются через **reference**)

Конфигурация TypeScript (compilerOptions)

- **target**
 - ES3, ES5, ES6 (он же ES2015), ES2016, ES2017, ES2018, ES2019, ES2020, ESNext
- **module**
 - None, CommonJS, AMD, System, UMD, ES6, ES2015, ES2020, ESNext
- **outDir**
 - папка, куда будут помещаться собранные артефакты
- **esModuleInterop**
 - позволяет импортировать CommonJS пакеты как ES6
- **alwaysStrict**
 - добавление “use strict” в выходные файлы
- **downlevelIteration**
 - для ES5 for / of преобразовывается в обычный for
- **forceConsistentCasingInFileNames**
 - режим чувствительности к регистру (case-sensitive) для импорта файлов
- **allowJs**
 - обработка не только ts файлов, но и js
- **checkJs**
 - проверка ошибок не только в ts, но и в js-файлах

JSX Specification

XML-like syntax extension to EcmaScript



<https://facebook.github.io/jsx/>

Элементы

- JSXElement
- JSXSelfClosingElement
 - `<elem/>`
- JSXOpeningElement
 - `<elem>`
- JSXClosingElement
 - `</elem>`
- JSXFragment
 - `<>...</>`
- JSXElementName
- JSXIdentifier
- JSXNamespacedName
 - `id1:id2`
- JSXMemberExpression
 - `name.id`

Атрибуты

- JSXAttributes
- JSXSpreadAttribute
 - `{...expression}`
- JSXAttribute
- JSXAttributeName
- JSXAttributeInitializer
 - `=value`
- JSXAttributeValue
 - `""`, `"`, `{expression}`
- JSXDoubleStringCharacters
- JSXDoubleStringCharacter
- JSXSingleStringCharacters
- JSXSingleStringCharacter

Потомки

Стандартный пример JSX

// Использование JSX для отображения компонентов UI

```
var dropdown =  
  <Dropdown>  
    Выпадающее меню  
    <Menu>  
      <MenuItem>Делаем что-нибудь</MenuItem>  
      <MenuItem>Делаем что-нибудь забавное</MenuItem>  
      <MenuItem>Делаем что-нибудь ещё</MenuItem>  
    </Menu>  
  </Dropdown>;  
  
render(dropdown);
```

В примере наблюдаем

- JSXElement
- JSXOpeningElement
 - <elem>
- JSXClosingElement
 - </elem>
- JSXFragment
 - <>...</>
- JSXElementName


TSX → JSX

47

*.tsx

```
declare namespace JSX {  
  interface ElementClass {  
    render: any;  
  }  
}  
  
class MyComponent {  
  render() {}  
}  
  
function MyFactoryFunction() {  
  return { render: () => {} }  
}  
  
<MyComponent />; // ok  
<MyFactoryFunction />; // ok
```

*.jsx



```
var MyComponent = /** @class */ (function () {  
  function MyComponent() {  
  }  
  MyComponent.prototype.render = function () { };  
  return MyComponent;  
})();  
  
function MyFactoryFunction() {  
  return { render: function () { } };  
}  
  
<MyComponent />; // ok  
<MyFactoryFunction />; // ok
```

Команда:

tsc --jsx preserve test.tsx

Траспайлеры

- React JSX
- jsx-transform
- Babel

<https://www.typescriptlang.org/docs/handbook/jsx.html>

Типизация в языках программирования⁴⁸

Языки со статической типизацией

- C, Java, C#

Языки с динамической типизацией

- JavaScript, PHP, Python, Ruby

Строгая типизация (сильная)

- Java, Python, Haskell, Lisp

Нестрогая типизация (слабая)

- PHP, JavaScript, Visual Basic

Явная типизация

- C++, D, C#

Неявная типизация

- JavaScript, PHP, Lua

С чем мы можем столкнуться

49

- Неявное приведение типа
- Ошибки в коде
 - забытые обязательные параметры
 - несравнимые объекты
- Повторное присваивание значений константам
- Использование отсутствующих свойств и методов
- Появление непредусмотренных значений



The screenshot shows the Flow website homepage in a web browser. The browser's address bar displays 'flow.org' and the page title is 'Flow: A Static Type Checker for JavaScript'. The website's navigation bar includes the Flow logo, 'flow', and links for 'Getting Started', 'Docs', 'Try', and 'Blog'. Social media icons for Twitter, GitHub, and a search icon are also present. The main content area features a large, bold headline: 'FLOW IS A STATIC TYPE CHECKER FOR JAVASCRIPT.' Below this headline are three buttons: 'GET STARTED', 'INSTALL FLOW', and a GitHub Star button showing '21,373' stars. Under the 'GET STARTED' button, it says 'Current Version: [v0.159.0](#)'. At the bottom of the page, the text 'CODE FASTER.' is displayed, followed by the sentence 'Tired of having to run your code to find bugs? Flow'. To the right of this text is a small image of a code editor with syntax-highlighted code.

Flow: A Static Type Checker for JavaScript

flow Getting Started Docs Try Blog

FLOW IS A STATIC TYPE CHECKER FOR JAVASCRIPT.

GET STARTED **INSTALL FLOW** Star 21,373

Current Version: [v0.159.0](#)

CODE FASTER.

Tired of having to run your code to find bugs? Flow

Установка в варианте **flow-remove-types**

51

- При использовании **npm**
 - **npm install --save-dev flow-remove-types**
 - **npm install --save-dev flow-bin**
- При использовании **yarn**
 - **yarn add --dev flow-remove-types**
 - **yarn add --dev flow-bin**
- **flow init**

Настройка Flow в связке с babel

52

```
npm i --save-dev babel-cli babel-preset-flow
```

Настройка .babelrc

Настройка scripts в package.json

```
npm install --save-dev flow-bin
```

Донастройка scripts в package.json

```
npm run flow init
```

.babelrc ← **создать**

```
{  
  "presets":  
    [ "flow" ]  
}
```

package.json

```
"scripts": {  
  "build": "babel src/ -d lib/",  
  "prepublish": "npm run build"  
},
```

package.json

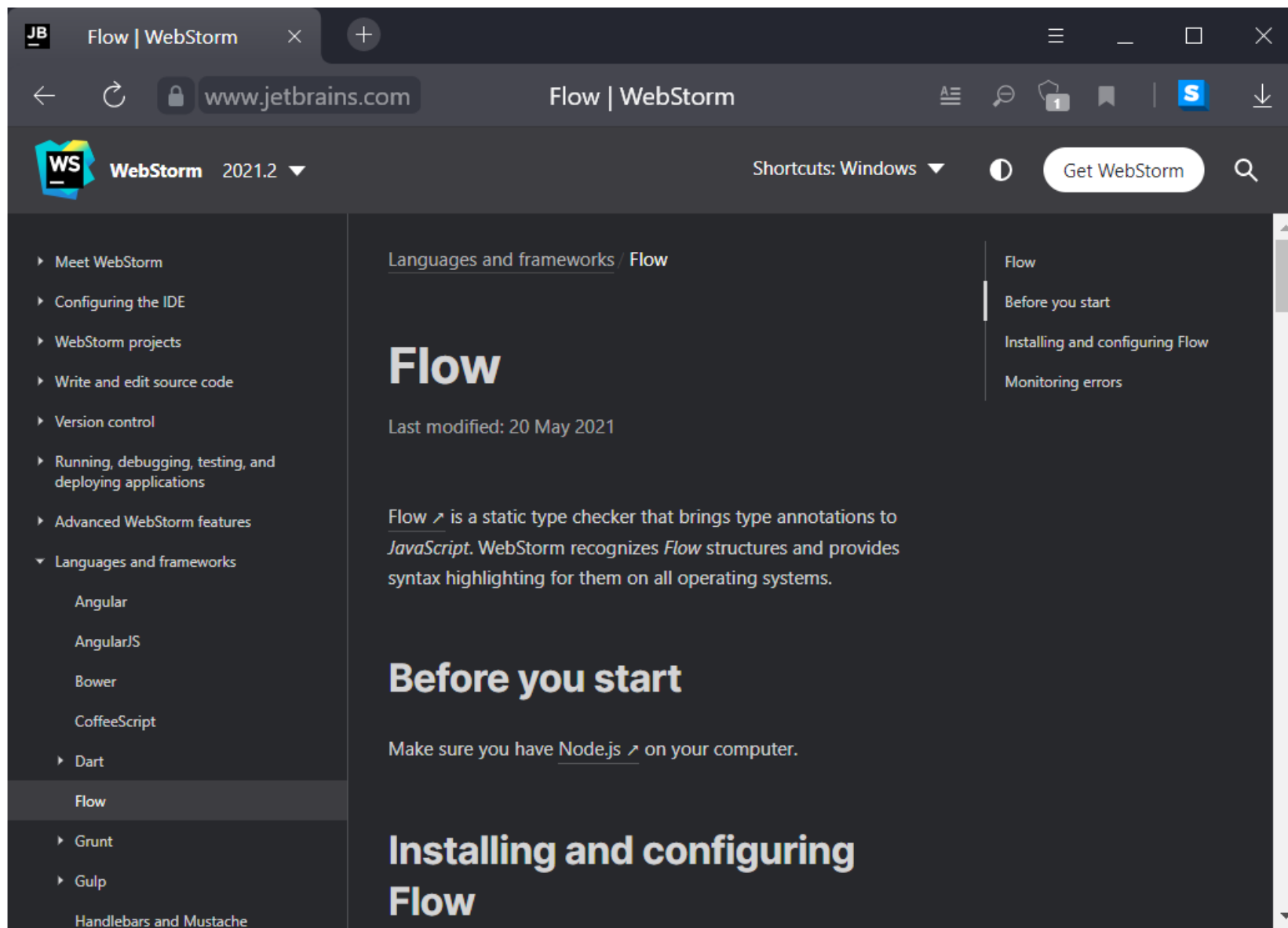
```
"scripts": {  
  "build": "babel src/ -d lib/",  
  "prepublish": "npm run build",  
  "flow": "flow"  
},
```

Использование:
npm run flow

<https://flow.org/en/docs/install/>
<https://flow.org/try/>

Настройка WebStorm (1)

53



The screenshot shows the JetBrains WebStorm documentation page for Flow. The browser address bar shows 'www.jetbrains.com'. The page title is 'Flow | WebStorm'. The left sidebar contains a navigation menu with categories like 'Meet WebStorm', 'Configuring the IDE', 'WebStorm projects', 'Write and edit source code', 'Version control', 'Running, debugging, testing, and deploying applications', 'Advanced WebStorm features', and 'Languages and frameworks'. The 'Languages and frameworks' section is expanded, showing 'Angular', 'AngularJS', 'Bower', 'CoffeeScript', 'Dart', 'Flow' (selected), 'Grunt', 'Gulp', and 'Handlebars and Mustache'. The main content area is titled 'Flow' and includes a subtitle 'Languages and frameworks / Flow'. It states 'Last modified: 20 May 2021'. The text describes Flow as a static type checker for JavaScript. Below this, there is a section titled 'Before you start' with the instruction 'Make sure you have Node.js on your computer.' and another section titled 'Installing and configuring Flow'.

Flow | WebStorm 2021.2

Shortcuts: Windows

Get WebStorm

Meet WebStorm

Configuring the IDE

WebStorm projects

Write and edit source code

Version control

Running, debugging, testing, and deploying applications

Advanced WebStorm features

Languages and frameworks

- Angular
- AngularJS
- Bower
- CoffeeScript
- Dart
- Flow**
- Grunt
- Gulp
- Handlebars and Mustache

Languages and frameworks / Flow

Flow

Last modified: 20 May 2021

Flow is a static type checker that brings type annotations to JavaScript. WebStorm recognizes Flow structures and provides syntax highlighting for them on all operating systems.

Before you start

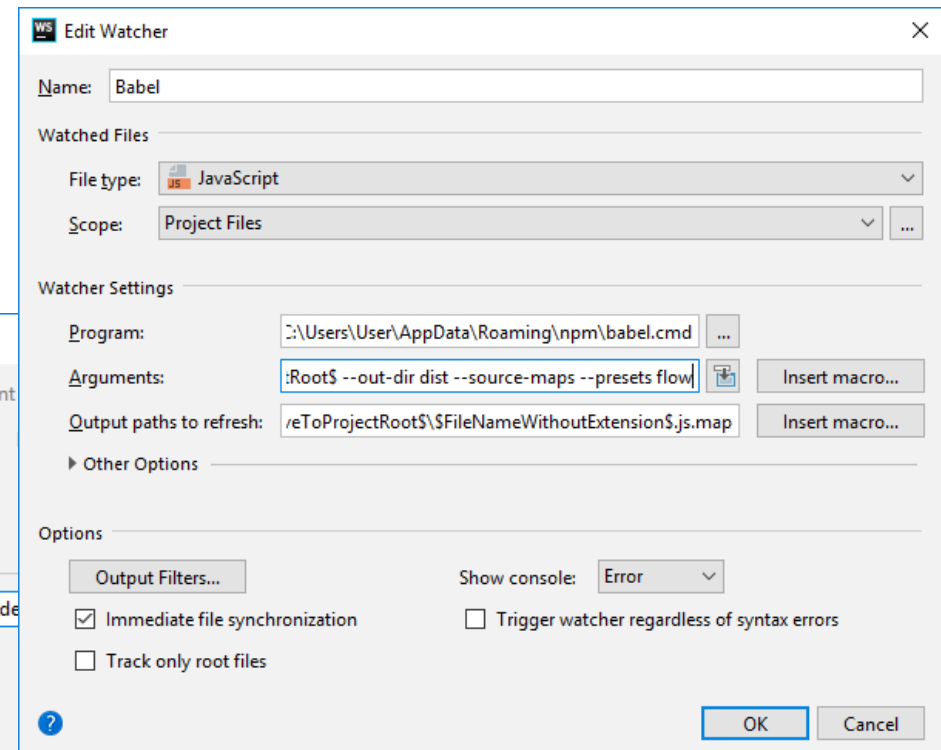
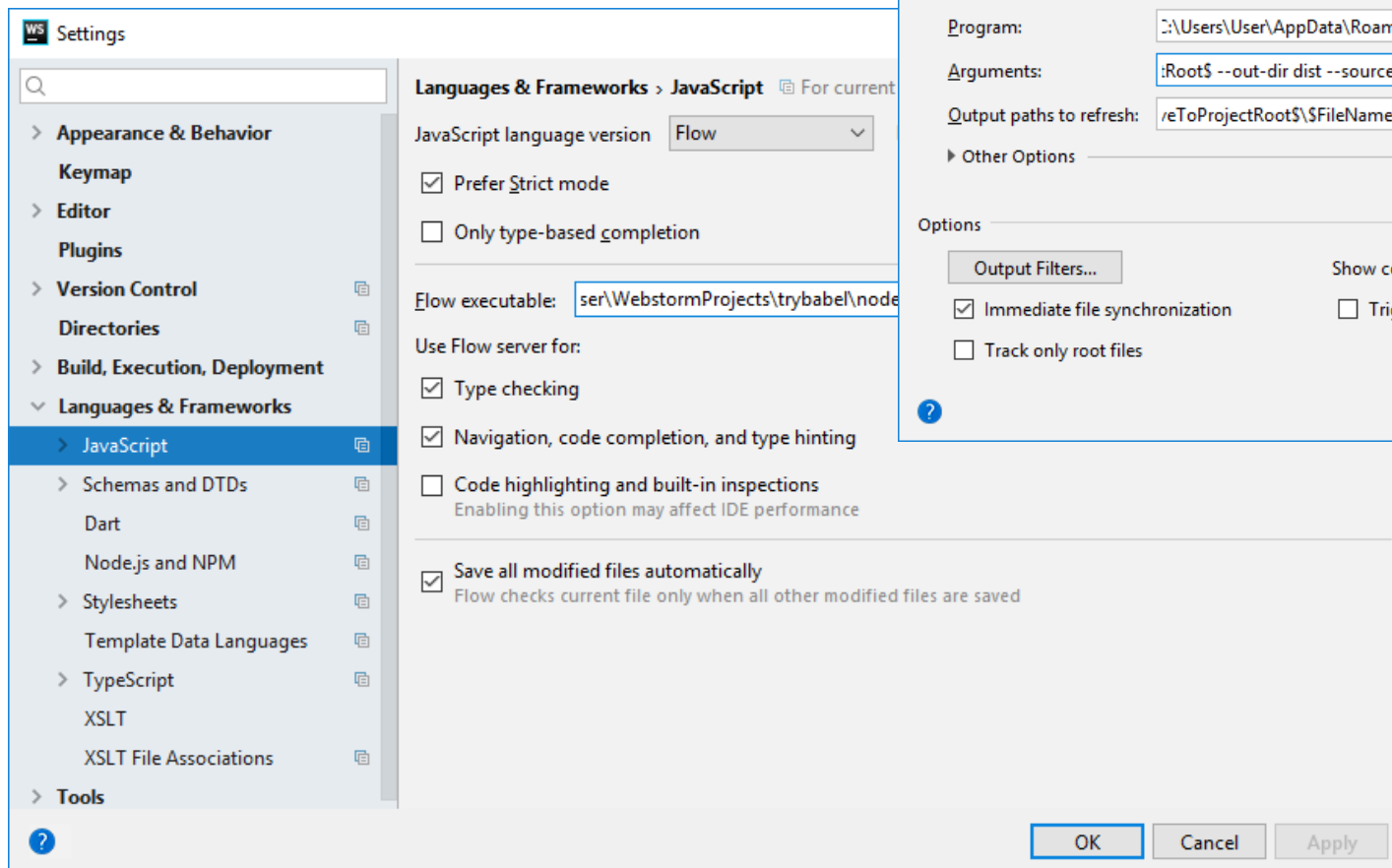
Make sure you have Node.js on your computer.

Installing and configuring Flow

<https://www.jetbrains.com/help/webstorm/using-the-flow-type-checker.html>

Настройка WebStorm (2)

54



Пример использования flow (1)

Указание типа

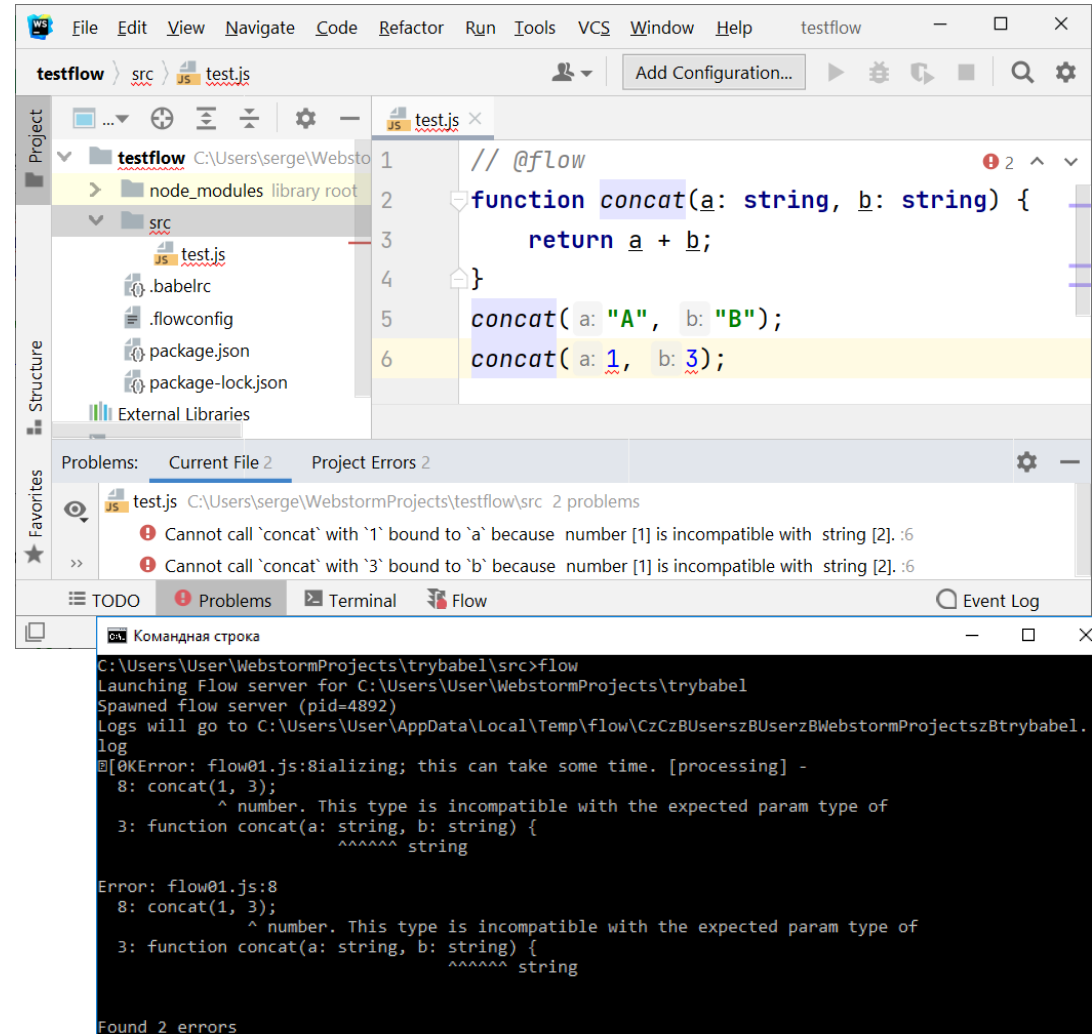
55

Без ошибок

```
// @flow
function concat(a, b) {
  return a + b;
}
concat("A", "B");
concat(1, 3);
```

Ошибка в последней строке

```
// @flow
function concat(a: string, b: string) {
  return a + b;
}
concat("A", "B");
// Cannot call `concat` with `1` bound to `a` because number [1] is incompatible with string [2].
// Cannot call `concat` with `3` bound to `b` because number [1] is incompatible with string [2].
concat(1, 3);
```



```
// @flow
```

```
/* @flow */
```

Пример использования flow (2)

Указание типа

56

The screenshot shows the Flow Playground interface. The code editor contains the following code:

```
1 // @flow
2
3 function concat(a, b) {
4   return a + b;
5 }
6
7 concat("A", "B");
8 concat(1, 3);
9 |
```

The right sidebar shows the 'Errors' tab with the message 'No errors!'. The version dropdown is set to 'v0.59.0'.

This screenshot shows the same code as the previous one, but with type annotations added to the function signature and calls. The code is:

```
2
3 function concat(a: string, b: string) {
4   return a + b;
5 }
6
7 concat("A", "B");
8 concat(1, 3);
9
```

The right sidebar shows the 'Errors' tab with two error messages:

```
8: concat(1, 3);
      ^ number. This type is incompatible with the expected parameter type
3: function concat(a: string, b: string) {
      ^ string

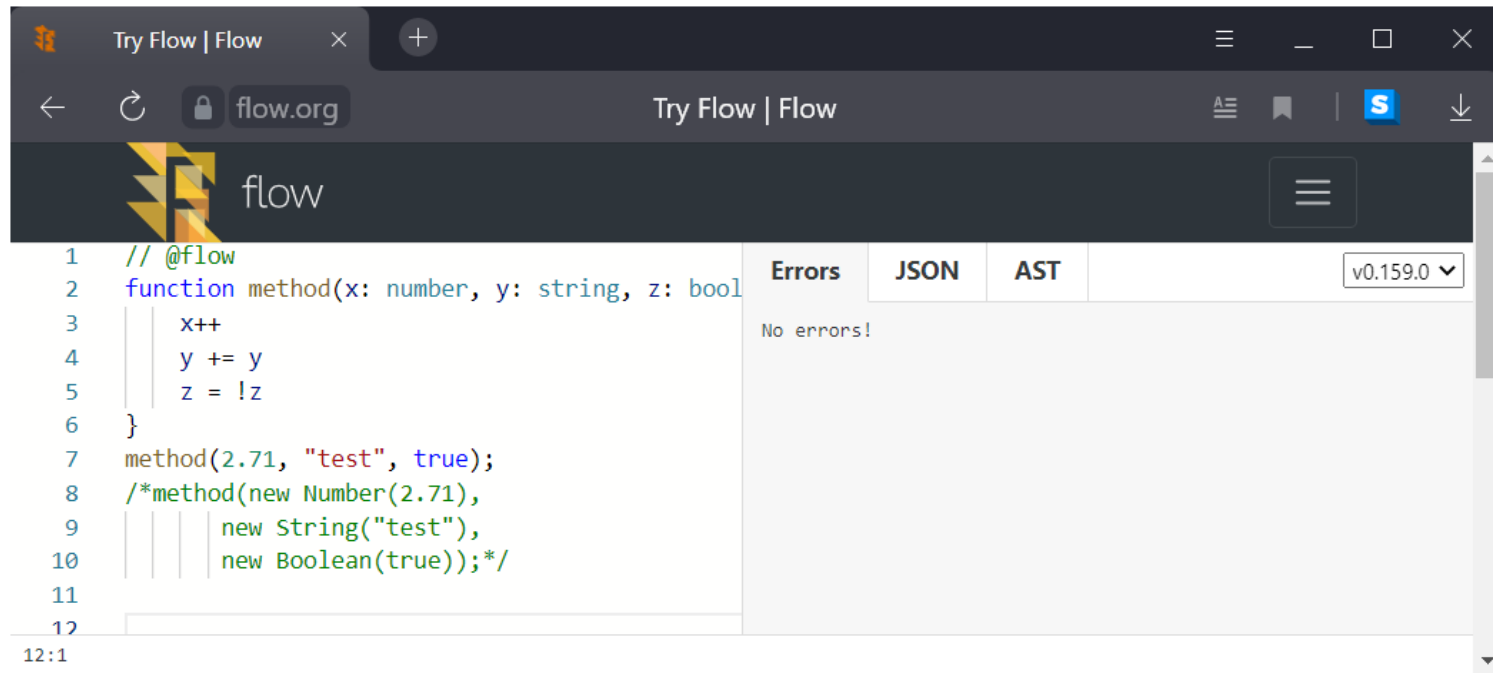
8: concat(1, 3);
      ^ number. This type is incompatible with the expected parameter type
3: function concat(a: string, b: string) {
      ^ string
```

The version dropdown is set to 'v0.59.0'.

Примитивные типы (1)

57

```
// @flow
function method(x: number, y: string, z: boolean) {
  x++
  y += y
  z = !z
}
method(2.71, "test", true);
/*method(new Number(2.71),
  new String("test"),
  new Boolean(true));*/
```



Примитивные типы (2)

58

```
// @flow
```

```
function method(x: number, y: string, z: boolean) {
```

```
  x++
```

```
  y += y
```

```
  z = !z
```

```
}
```

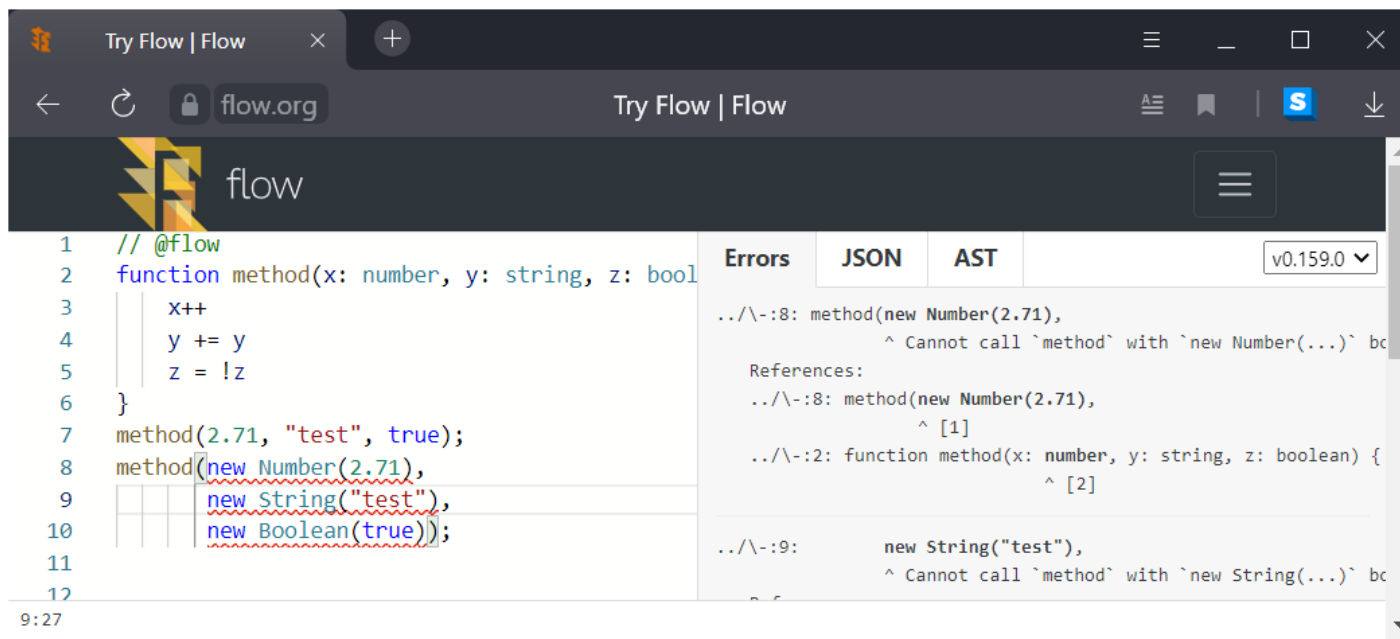
```
method(2.71, "test", true);
```

```
method(new Number(2.71),
```

```
  new String("test"),
```

```
  new Boolean(true));
```

Ожидался number, а
не Number



Try Flow | Flow

flow.org

flow

```
1 // @flow
2 function method(x: number, y: string, z: boolean) {
3   x++
4   y += y
5   z = !z
6 }
7 method(2.71, "test", true);
8 method(new Number(2.71),
9         new String("test"),
10        new Boolean(true));
11
12
```

Errors JSON AST v0.159.0

../\--:8: method(new Number(2.71),
^ Cannot call `method` with `new Number(...)` bc

References:
../\--:8: method(new Number(2.71),
^ [1]
../\--:2: function method(x: number, y: string, z: boolean) {
^ [2]

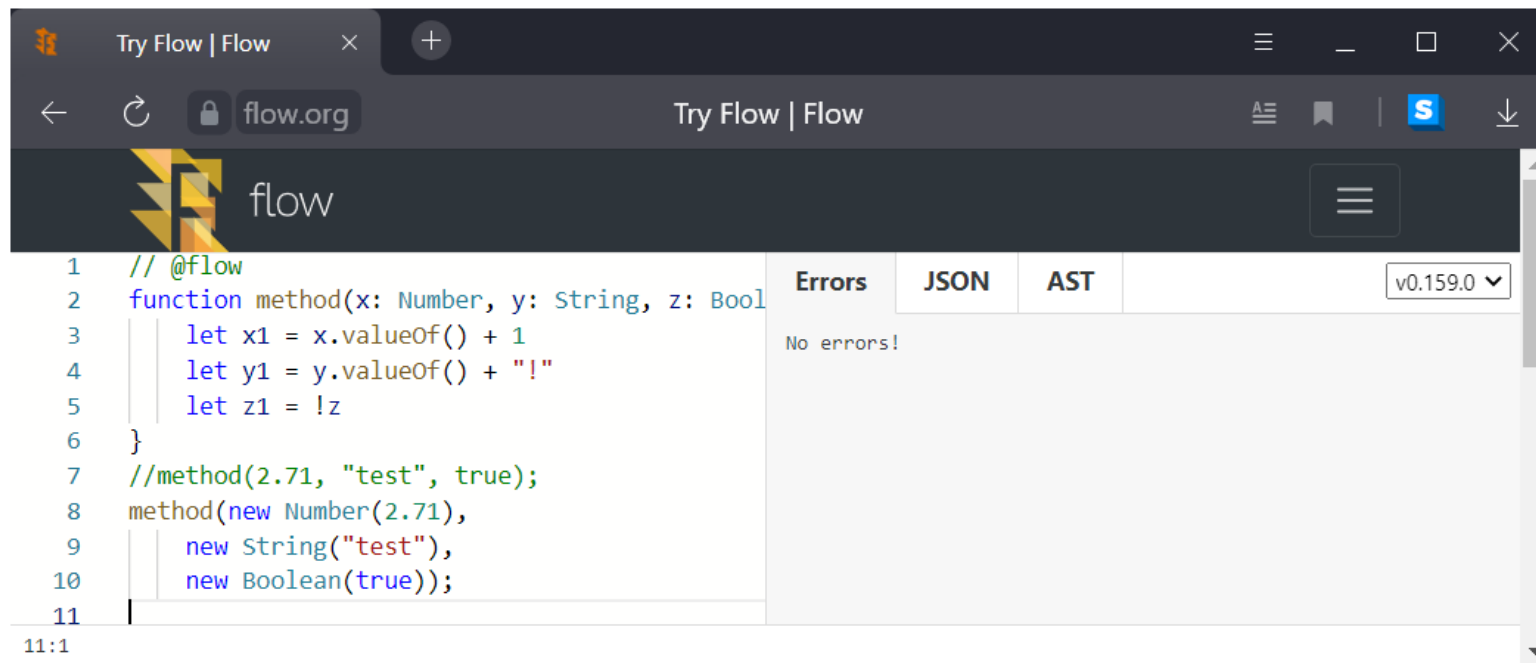
../\--:9: new String("test"),
^ Cannot call `method` with `new String(...)` bc

9:27

Объекты-обёртки (1)

59

```
// @flow
function method(x: Number, y: String, z: Boolean) {
  let x1 = x.valueOf() + 1
  let y1 = y.valueOf() + "!"
  let z1 = !z
}
//method(2.71, "test", true);
method(new Number(2.71),
  new String("test"),
  new Boolean(true));
```



Объекты-обёртки (2)

60

```
// @flow
```

```
function method(x: Number, y: String, z: Boolean) {
```

```
  let x1 = x.valueOf() + 1
```

```
  let y1 = y.valueOf() + "!"
```

```
  let z1 = !z
```

```
}
```

```
method(2.71, "test", true);
```

```
method(new Number(2.71),
```

```
  new String("test"),
```

```
  new Boolean(true));
```

Ожидался Number

Try Flow | Flow

flow.org

Try Flow | Flow

flow

```
1 // @flow
2 function method(x: Number, y: String, z: Boolean) {
3   let x1 = x.valueOf() + 1
4   let y1 = y.valueOf() + "!"
5   let z1 = !z
6 }
7 method(2.71, "test", true);
8 method(new Number(2.71),
9   new String("test"),
10  new Boolean(true));
11
```

Errors JSON AST v0.159.0

../\:-:7: method(2.71, "test", true);
^ Cannot call `method` with `2.71` bound to `x`

References:

../\:-:7: method(2.71, "test", true);
^ [1]

../\:-:2: function method(x: Number, y: String, z: Boolean) {
^ [2]

../\:-:7: method(2.71, "test", true);

7:1 (x: Number, y: String, z: Boolean) => void

Не обязательные параметры и значения по умолчанию ⁶¹

```
// @flow
```

```
function method(x: number, y?: string = "def.value") {  
}  
method();  
method(2.71);  
method(2.71, "test");  
method(2.71, undefined);  
method(2.71, null);
```

Пустой и null нельзя

The screenshot shows the Flow IDE interface. The left pane contains the following code:

```
1 // @flow  
2 function method(x: number, y?: string = "def.value") {  
3 }  
4 method();  
5 method(2.71);  
6 method(2.71, "test");  
7 method(2.71, undefined);  
8 method(2.71, null);
```

The right pane shows the 'Errors' tab with the following messages:

```
../\:-:4: method();  
    ^ Cannot call `method` because function [1] requires ar  
References:  
../\:-:2: function method(x: number, y?: string = "def.value");  
    ^ [1]  
  
../\:-:8: method(2.71, null);  
                ^ Cannot call `method` with `null` bound t  
References:  
../\:-:8: method(2.71, null);  
                ^ [1]  
../\:-:2: function method(x: number, y?: string = "def.value");  
                ^ [2]
```

The bottom left corner of the IDE shows the time 8:20.

Перечисление возможных значений параметра

```
// @flow
function acceptsTwo(value: 2) {}
acceptsTwo(2);
acceptsTwo(3);
acceptsTwo("2");
function getColor(name: "success" | "warning") {
  switch (name) {
    case "success" : return "green";
    case "warning" : return "yellow";
  }
}
getColor("success");
getColor("error");
```

Нельзя строку «2»
Нельзя строку «error»

The screenshot shows the Try Flow | Flow web interface. The code editor on the left contains the following code:

```
1 // @flow
2 function acceptsTwo(value: 2) {
3 }
4 acceptsTwo(2);
5 acceptsTwo(3);
6 acceptsTwo("2");
7
8 function getColor(name: "success" | "warning") {
9   switch (name) {
10     case "success" : return "green";
11     case "warning" : return "yellow";
12   }
13 }
14 getColor("success");
15 getColor("error");
16
17
```

The right panel shows the 'Errors' tab with the following messages:

- ../\:-:5: acceptsTwo(3);**
 ^ Cannot call `acceptsTwo` with `3` bound to to
 References:
 ../\:-:5: acceptsTwo(3);
 ^ [1]
- ../\:-:2: function acceptsTwo(value: 2) {**
 ^ [2]
- ../\:-:6: acceptsTwo("2");**
 ^ Cannot call `acceptsTwo` with `"2"` bound
 References:
 ../\:-:6: acceptsTwo("2");
 ^ [1]
 ../\:-:2: function acceptsTwo(value: 2) {
 ^ [2]
- ../\:-:15: getColor("error");**
 ^ Cannot call `getColor` with `"error"` bound
 References:
 ../\:-:15: getColor("error");
 ^ [1]
 ../\:-:8: function getColor(name: "success" | "warning") {
 ^ [2]

The bottom status bar shows the cursor position: 17:1.

Смешанные типы (1)

63

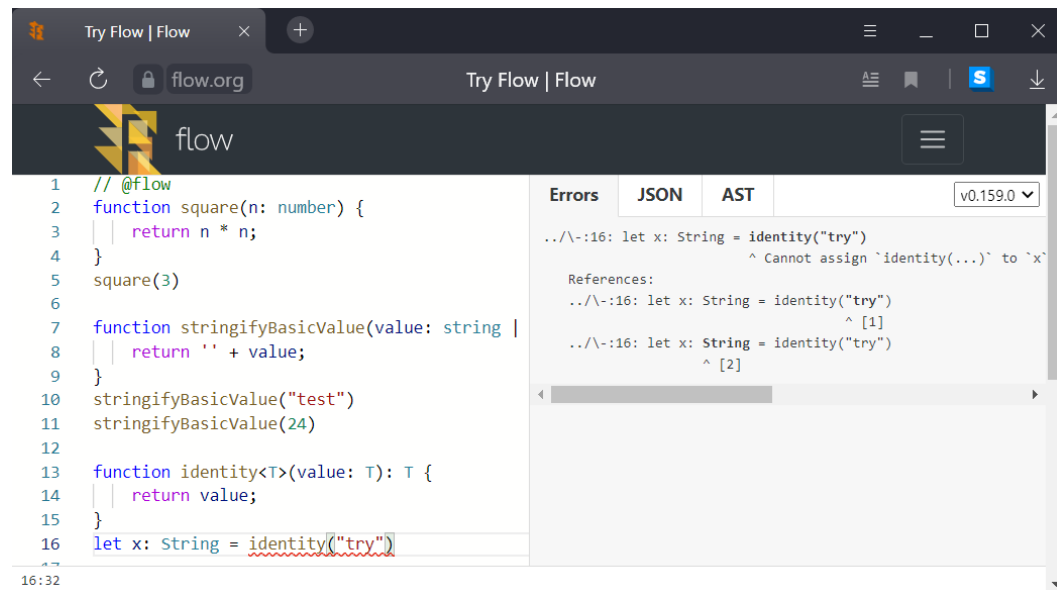
```
// @flow
```

```
function square(n: number) {  
  return n * n;  
}  
  
square(3)
```

```
function stringifyBasicValue(value: string | number) {  
  return '' + value;  
}  
  
stringifyBasicValue("test")  
stringifyBasicValue(24)
```

```
function identity<T>(value: T): T {  
  return value;  
}  
  
let x: String = identity("try")
```

Ожидался string, а не String



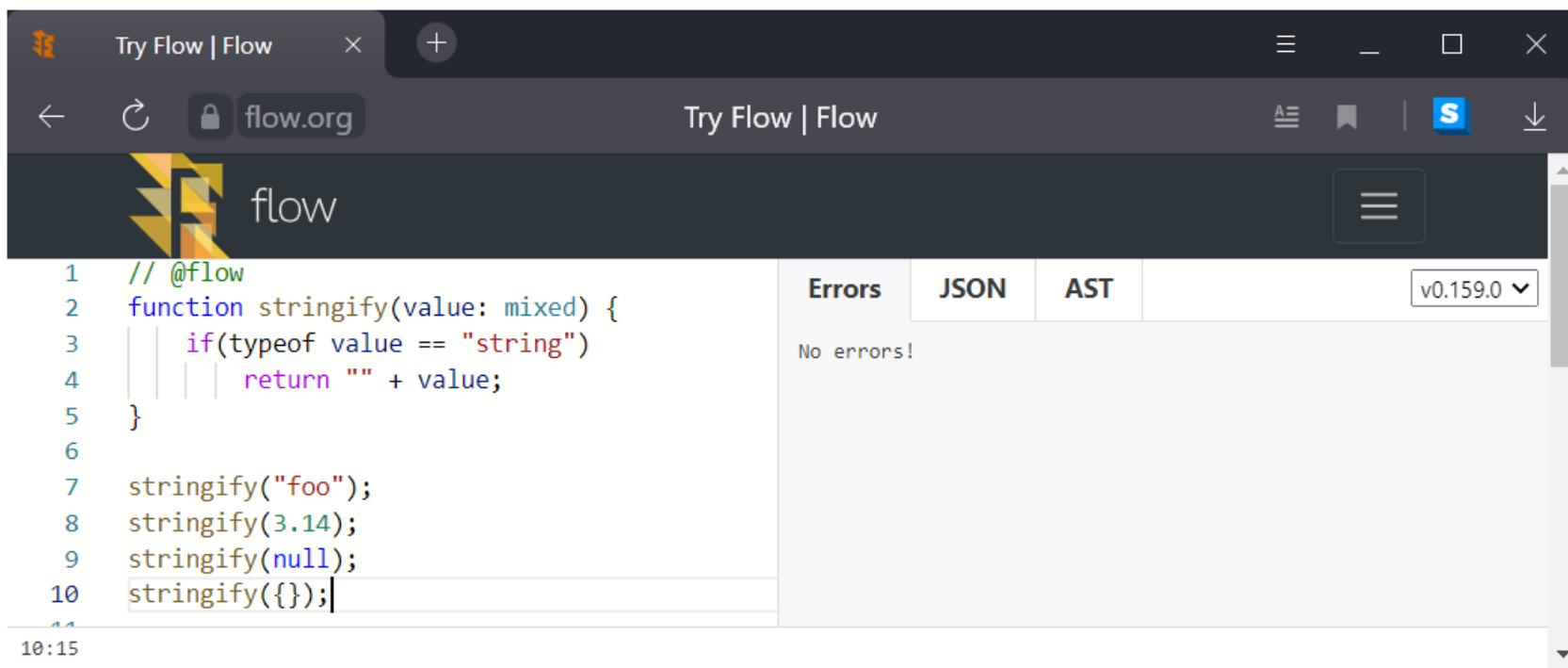
Смешанные типы (2)

64

```
// @flow
function stringify(value: mixed) {
  if (typeof value == "string")
    return "" + value;
}
```

```
stringify("foo");
stringify(3.14);
stringify(null);
stringify({});
```

Могут использоваться разные типы



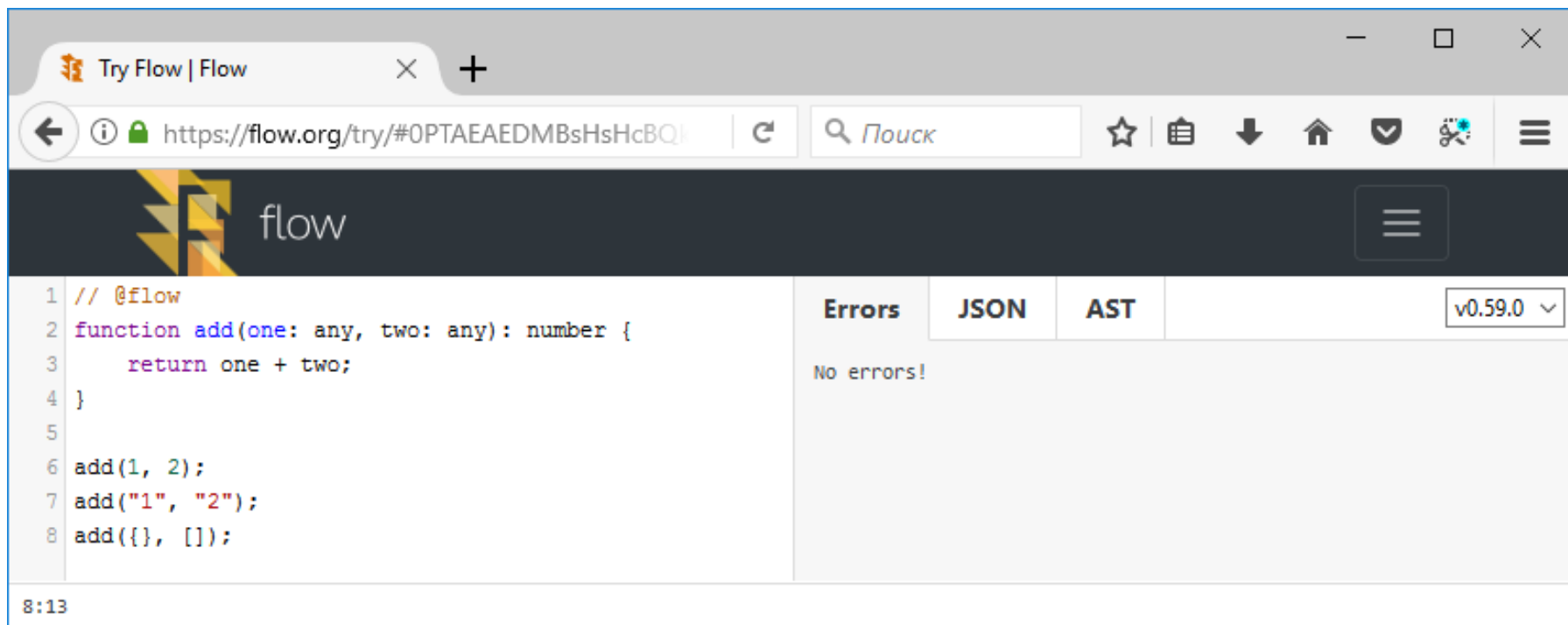
Произвольный тип

65

```
// @flow
function add(one: any, two: any): number {
  return one + two;
}
```

```
add(1, 2);
add("1", "2");
add({}, []);
```

Может быть произвольный тип



The screenshot shows a web browser window with the URL <https://flow.org/try/#OPTAEAEDMBsHsHcBQk>. The page features the Flow logo and a code editor with the following code:

```
1 // @flow
2 function add(one: any, two: any): number {
3   return one + two;
4 }
5
6 add(1, 2);
7 add("1", "2");
8 add({}, []);
```

On the right side of the editor, there are tabs for 'Errors', 'JSON', and 'AST'. The 'Errors' tab is active, displaying the message 'No errors!'. A version dropdown menu shows 'v0.59.0'. The bottom left corner of the interface displays the timestamp '8:13'.

ВОЗМОЖНЫЙ ТИП

66

```
// @flow
```

```
function acceptsMaybeNumber(value: ?number) {  
  if (typeof value === 'number') {  
    return value * 2;  
  }  
}
```

```
acceptsMaybeNumber(42);  
acceptsMaybeNumber();  
acceptsMaybeNumber(undefined);  
acceptsMaybeNumber(null);  
acceptsMaybeNumber("42");
```

Не может быть «42»

The screenshot shows the 'Try Flow | Flow' web interface. The code editor contains the following code:

```
1 // @flow  
2 function acceptsMaybeNumber(value: ?number) {  
3   if (typeof value === 'number') {  
4     return value * 2;  
5   }  
6 }  
7  
8 acceptsMaybeNumber(42);  
9 acceptsMaybeNumber();  
10 acceptsMaybeNumber(undefined);  
11 acceptsMaybeNumber(null);  
12 acceptsMaybeNumber("42");
```

The right-hand panel shows the 'Errors' tab with the following message:

```
../\:-:12: acceptsMaybeNumber("42");  
                                ^ Cannot call `acceptsMaybeNumber`  
  
References:  
../\:-:12: acceptsMaybeNumber("42");  
                                ^ [1]  
../\:-:2: function acceptsMaybeNumber(value: ?number) {  
                                ^ [2]
```

The status bar at the bottom left indicates '14:1'.

Присваивание значения переменным

67

```
// @flow
var varVariable = 1;
let letVariable = 1;
const constVariable = 1;
```

```
varVariable = 2;
letVariable = 2;
constVariable = 2;
```

Нельзя переопределить const

Try Flow | Flow

flow.org

Try Flow | Flow

flow

```
1 // @flow
2 var varVariable = 1;
3 let letVariable = 1;
4 const constVariable = 1;
5
6 varVariable = 2;
7 letVariable = 2;
8 constVariable = 2;
9
```

10:1

Errors JSON AST v0.159.0

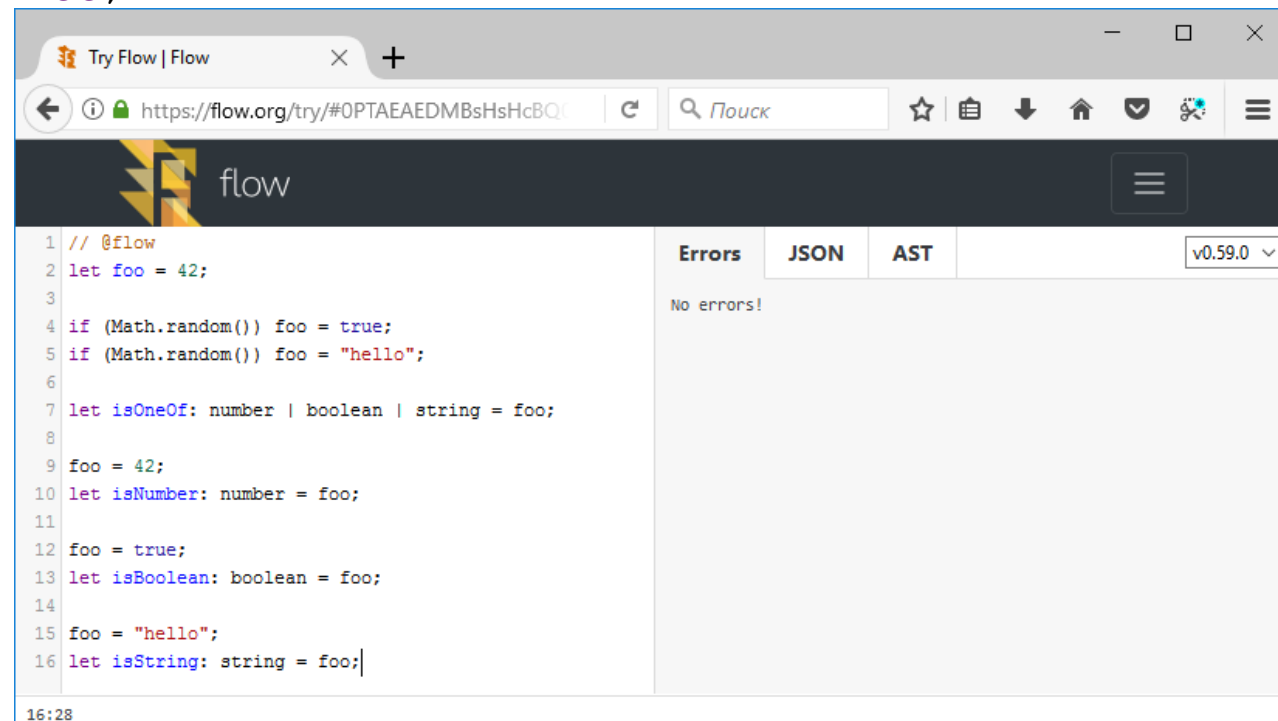
..\-\-:8: constVariable = 2;
^ Cannot reassign constant `constVariable` [1]. [reassi
References:
..\-\-:4: const constVariable = 1;
^ [1]

Присваивание значений разных типов (1)

68

```
// @flow
let foo = 42;
if (Math.random()) foo = true;
if (Math.random()) foo = "hello";
let isOneOf: number | boolean | string = foo;
foo = 42;
let isNumber: number = foo;
foo = true;
let isBoolean: boolean = foo;
foo = "hello";
let isString: string = foo;
```

Нет нарушений присваивания



Присваивание значений разных типов (2)

69

```
// @flow
let foo = 42;
function mutate() {
  foo = true;
  foo = "hello";
}
mutate();
let isString: string = foo;
```

**string не совместима с
number и boolean**

The screenshot shows the Flow IDE interface. The editor displays the same code as the previous block. The variable `foo` is initially assigned the value `42` (number). Inside the `mutate` function, `foo` is first assigned `true` (boolean) and then `"hello"` (string). After the function call, the code attempts to assign `foo` to a variable of type `string`. The IDE highlights the assignment `let isString: string = foo;` with a red squiggly line under `foo`, indicating a type error. The right-hand pane shows the 'Errors' tab with two messages. Both messages state: `../\ -:8: let isString: string = foo; ^ Cannot assign `foo` to `isString``. The first message includes references to the initial assignment of `foo` to `42` (line 2) and the subsequent assignment to `"hello"` (line 5). The second message includes references to the assignment of `foo` to `true` (line 4) and the subsequent assignment to `"hello"` (line 5). The Flow version is shown as v0.159.0 in the top right corner of the IDE.

Параметры функции – callback (1)

70

```
// @flow
```

```
function method1(callback:
  (str: string, bool?: boolean, ...nums: Array<number>) => void) {
}

function method2(callback:
  (error: Error | null, value: string | null) => void) {
}
```

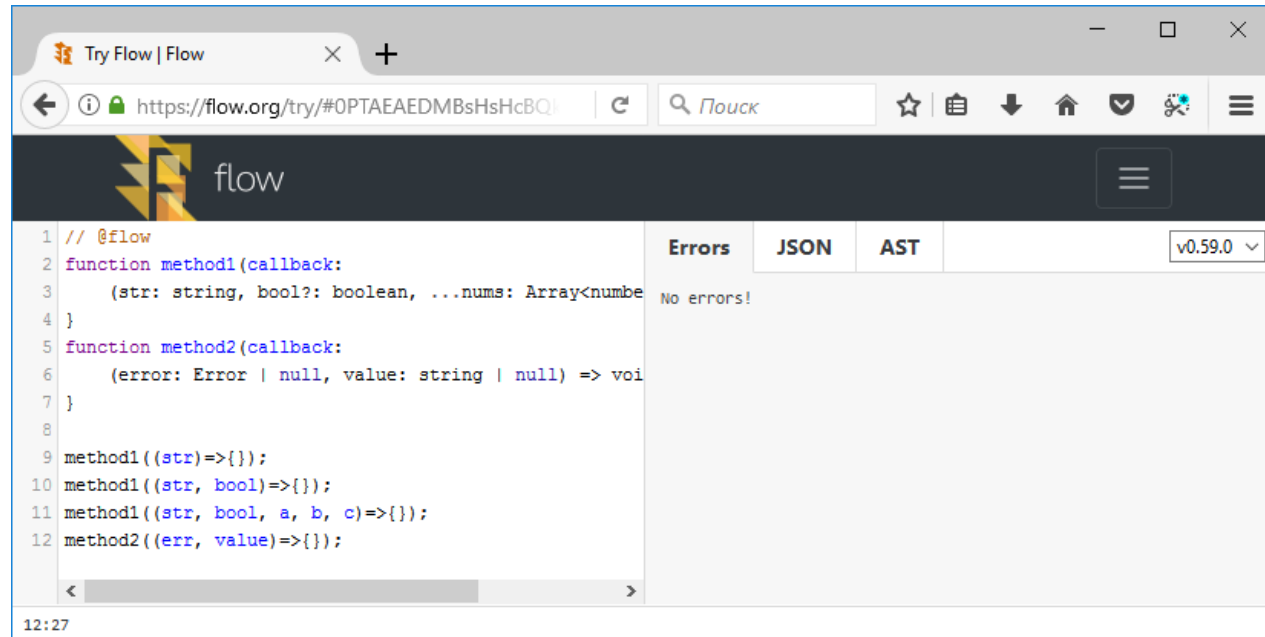
```
method1((str)=>{});
```

```
method1((str, bool)=>{});
```

```
method1((str, bool, a, b, c)=>{});
```

Возможен разный набор параметров

```
method2((err, value)=>{});
```



Параметры функции – callback (2)

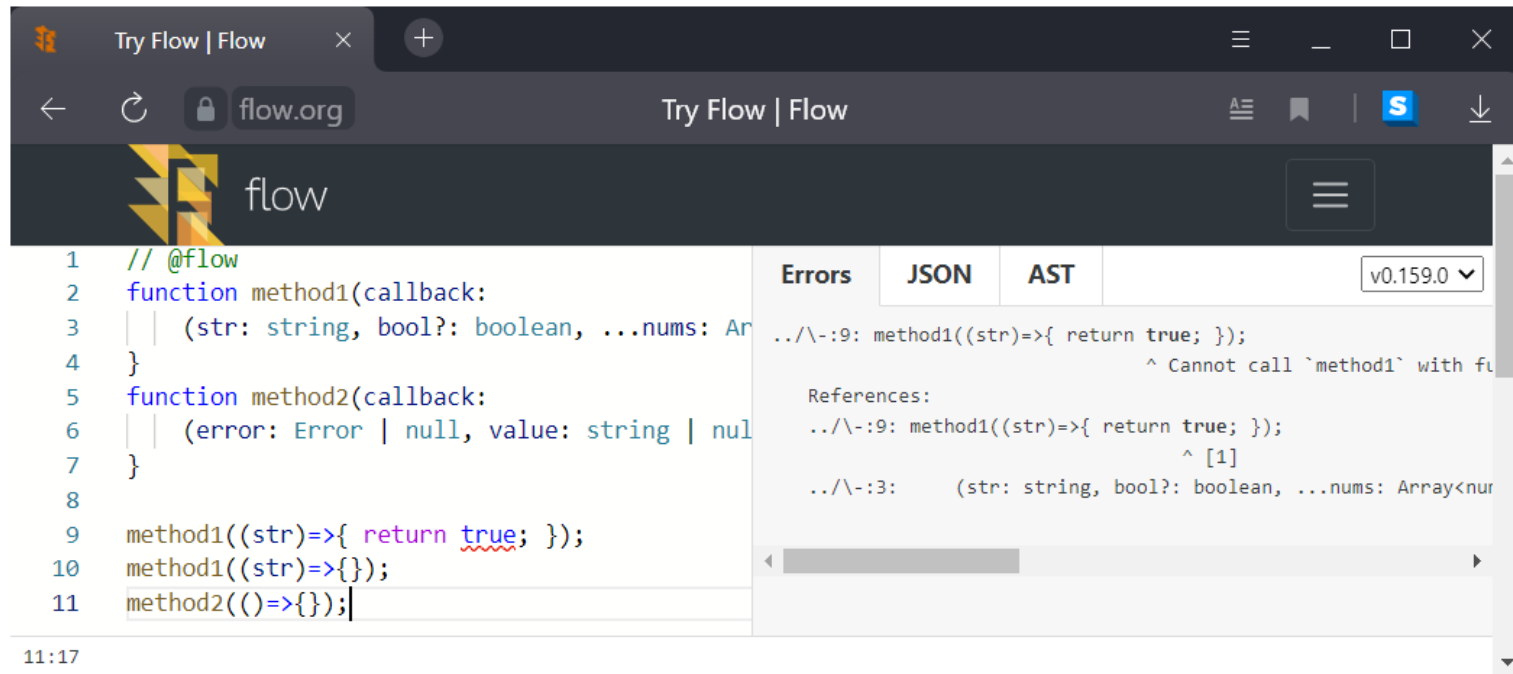
71

```
// @flow
```

```
function method1(callback:  
  (str: string, bool?: boolean, ...nums: Array<number>) => void) {  
}  
function method2(callback:  
  (error: Error | null, value: string | null) => void) {  
}
```

```
method1((str)=>{ return true; });  
method1((str)=>{});  
method2(()=>{});
```

Нельзя возвращать boolean



Try Flow | Flow

flow.org

Try Flow | Flow

flow

```
1 // @flow
2 function method1(callback:
3   (str: string, bool?: boolean, ...nums: Array<number>) => void) {
4 }
5 function method2(callback:
6   (error: Error | null, value: string | null) => void) {
7 }
8
9 method1((str)=>{ return true; });
10 method1((str)=>{});
11 method2(()=>{});
```

Errors JSON AST v0.159.0

../\:-:9: method1((str)=>{ return true; });
^ Cannot call `method1` with function argument of type `() => boolean` because its return type is not compatible with the return type of the function signature, which is `void`.

References:
../\:-:9: method1((str)=>{ return true; });
^ [1]
../\:-:3: (str: string, bool?: boolean, ...nums: Array<number>) => void

11:17

Работа с обiectами

72

```
// @flow
```

```
let obj1: { foo: boolean } = { foo: true };
```

```
let obj2 = { foo: "bar" }; // sealed  
obj2.bar = "";
```

```
let obj3 = {} //unsealed  
if (Math.random()) obj3.prop = true;  
else obj3.prop = "hello";  
var val3: boolean | string = obj3.prop;
```

**Нелъзя изменять
структуру обiectа**

The screenshot shows the Flow IDE interface. The code editor on the left contains the following code:

```
1 // @flow
2 let obj1: { foo: boolean } = { foo: true };
3
4 let obj2 = { foo: "bar" }; // sealed
5 obj2.bar = "";
6
7 let obj3 = {} //unsealed
8 if (Math.random()) obj3.prop = true;
9 else obj3.prop = "hello";
10 var val3: boolean | string = obj3.prop;
```

The right sidebar shows the 'Errors' panel with the following message:

```
../\:-:5: obj2.bar = "";
      ^ Cannot assign empty string to `obj2.bar` because
References:
../\:-:4: let obj2 = { foo: "bar" }; // sealed
      ^ [1]
```

The bottom status bar shows the line and column number: 12:1.

Использование объектов в качестве map ⁷³

```
// @flow
```

```
var o: { [string]: number } = {};  
o["foo"] = 0;  
o["bar"] = 1;  
var foo: number = o["foo"];
```

```
// optional user_id
```

```
var obj: { [user_id: number]: string } = {};  
obj[1] = "Julia";  
obj[2] = "Camille";  
obj[3] = "Justin";  
obj["foo"] = 0;
```

**В качестве индекса
ождается number**

The screenshot shows the Flow IDE interface. The left pane displays the code from the previous blocks. The right pane shows the 'Errors' tab with a TypeScript error message. The error message states: 'Object literal may only specify known properties, but 'foo' does not exist in type 'obj'. Did you mean to write 'obj[1]'?' This is a reference error, not a type error, as the code is not strictly typed. The error message also shows references to the variable 'obj' and the property 'foo'.

```
1 // @flow  
2 var o: { [string]: number } = {};  
3 o["foo"] = 0;  
4 o["bar"] = 1;  
5 var foo: number = o["foo"];  
6  
7 // optional user_id  
8 var obj: { [user_id: number]: string } = {};  
9 obj[1] = "Julia";  
10 obj[2] = "Camille";  
11 obj[3] = "Justin";  
12 obj["foo"] = 0;  
13  
14
```

Errors JSON AST v0.159.0

../\:-:12: obj["foo"] = 0;
^ string `foo` [1] is incompatible with number [2]. [i
References:
../\:-:12: obj["foo"] = 0;
^ [1]
../\:-:8: var obj: { [user_id: number]: string } = {};
^ [2]

../\:-:12: obj["foo"] = 0;
^ Cannot assign `0` to `obj["foo"]` becau
References:
../\:-:12: obj["foo"] = 0;
^ [1]
../\:-:8: var obj: { [user_id: number]: string } = {};
^ [2]

14:1

Проверка массивов

```
// @flow
```

```
let arr1: Array<number> = [1, 2, 3];
```

```
let arr2: number[] = [0, 1, 2, 3];
```

```
let arr3: Array<boolean> = [true, false, true];
```

```
let arr4: Array<string> = ["A", "B", "C"];
```

```
let arr5: Array<mixed> = [1, true, "three"]
```

```
arr4[1] = 23;
```

```
let arr6: ?number[] = null;
```

```
let arr7: ?number[] = [1, 2];
```

```
if(arr7) { // без проверки, без "?"
```

```
    let value: ?number = arr7[111];
```

```
}
```

number не
совместимо со string

Try Flow | Flow

flow.org

flow

```

1 // @flow
2 let arr1: Array<number> = [1, 2, 3];
3 let arr2: number[] = [0, 1, 2, 3];
4
5 let arr3: Array<boolean> = [true, false, true];
6 let arr4: Array<string> = ["A", "B", "C"];
7 let arr5: Array<mixed> = [1, true, "three"];
8 arr4[1] = 23;
9
10 let arr6: ?number[] = null;
11 let arr7: ?number[] = [1, 2];
12 if(arr7) { // без проверки, без "?"
13     let value: ?number = arr7[111];
14 }
  
```

Errors JSON AST v0.159.0

../\:-:8: arr4[1] = 23;
^ Cannot assign `23` to `arr4[1]` because number is not assignable to string

References:

```

../\:-:8: arr4[1] = 23;
          ^ [1]
../\:-:6: let arr4: Array<string> = ["A", "B", "C"];
          ^ [2]
  
```

16:1

Кортежи

75

```
// @flow
```

```
let tuple: [number, boolean, string] = [1, true, "three"];
```

```
let num : number = tuple[0];
```

```
let bool : boolean = tuple[1];
```

```
let str : string = tuple[2];
```

```
function getItem(n: number) {  
    let val: number | boolean | string = tuple[n];  
}
```

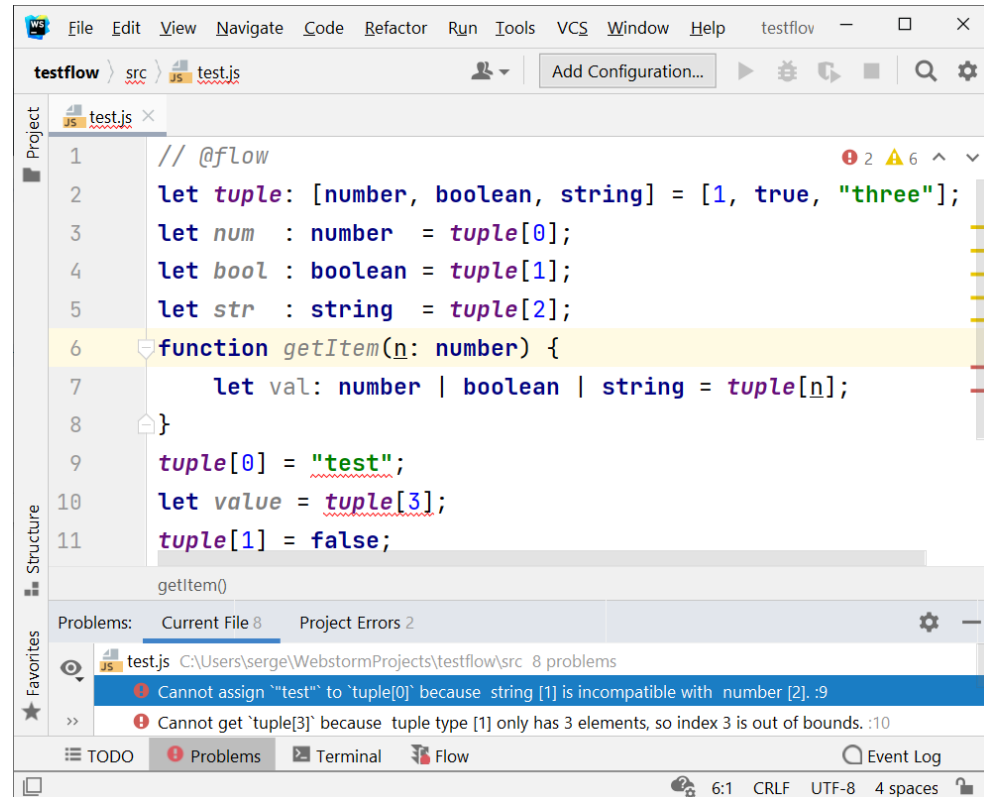
```
tuple[0] = "test";
```

```
let value = tuple[3];
```

```
tuple[1] = false;
```

1. Нулевой параметр типа `number`, нельзя присвоить `string`
2. В `tuple` 3 параметра, нельзя взять 4

Метод `push()` у кортежей отсутствует



Классы (1)

76

```
// @flow
class MyClass1 {
  method(value: string): number { return 0; }
}
class MyClass2 {
  method() {
    this.prop = 42;
  }
}
class MyClass {
  prop: number;
  method() {
    this.prop = 42;
  }
}
```

Свойство prop не описано

The screenshot shows the Try Flow | Flow web IDE interface. The editor displays the same code as the previous block. An error message is shown in the right-hand pane:

```
Errors JSON AST v0.159.0
../\:-:7:      this.prop = 42;
                ^ Cannot assign `42` to `this.prop` because prop
References:
../\:-:5: class MyClass2 {
          ^ [1]
```

The error indicates that the property `prop` is not defined in the class `MyClass2`, which is why the assignment `this.prop = 42` is invalid.

Классы (2)

77

```
// @flow
class MyClass1<A, B> {
  property: A;
  method(val: B): void {}
}

class MyClass2<A, B> {
  constructor(arg1: A, arg2: B) {}
}

var val1: MyClass1<number, string> = new MyClass1()
val1.property = "test";
var val2: MyClass2<number, boolean> = new MyClass2(1, true)
```

string не совместимо
с number

The screenshot shows the Try Flow web IDE interface. The code editor contains the following code:

```
1 // @flow
2 class MyClass1<A, B> {
3   property: A;
4   method(val: B): void {}
5 }
6 class MyClass2<A, B> {
7   constructor(arg1: A, arg2: B) {}
8 }
9
10 var val1: MyClass1<number, string> =
11   new MyClass1()
12   val1.property = "test";
13 var val2: MyClass2<number, boolean> =
14   new MyClass2(1, true)
```

The right-hand side of the IDE shows the 'Errors' panel with the following message:

```
../\-:12: val1.property = "test";
                        ^ Cannot assign `\"test\"` to `val1.property`

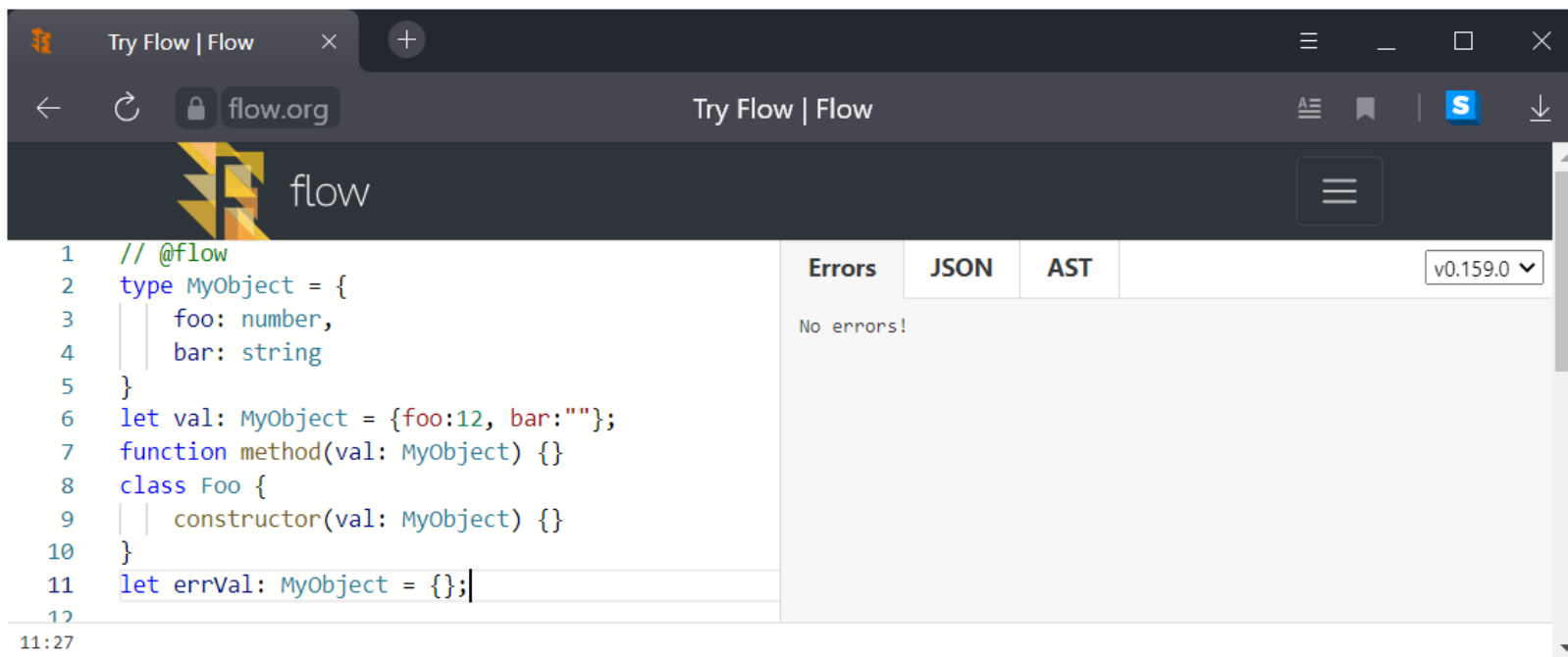
References:
../\-:12: val1.property = "test";
                        ^ [1]
../\-:10: var val1: MyClass1<number, string> =
                        ^ [2]
```

The status bar at the bottom left indicates the time 14:26.

Алиасы для типов

```
// @flow
type MyObject = {
  foo: number,
  bar: string
}
let val: MyObject = {foo:12, bar:""};
function method(val: MyObject) {}
class Foo {
  constructor(val: MyObject) {}
}
let errVal: MyObject = {};
```

Можно использовать
«короткую» запись type

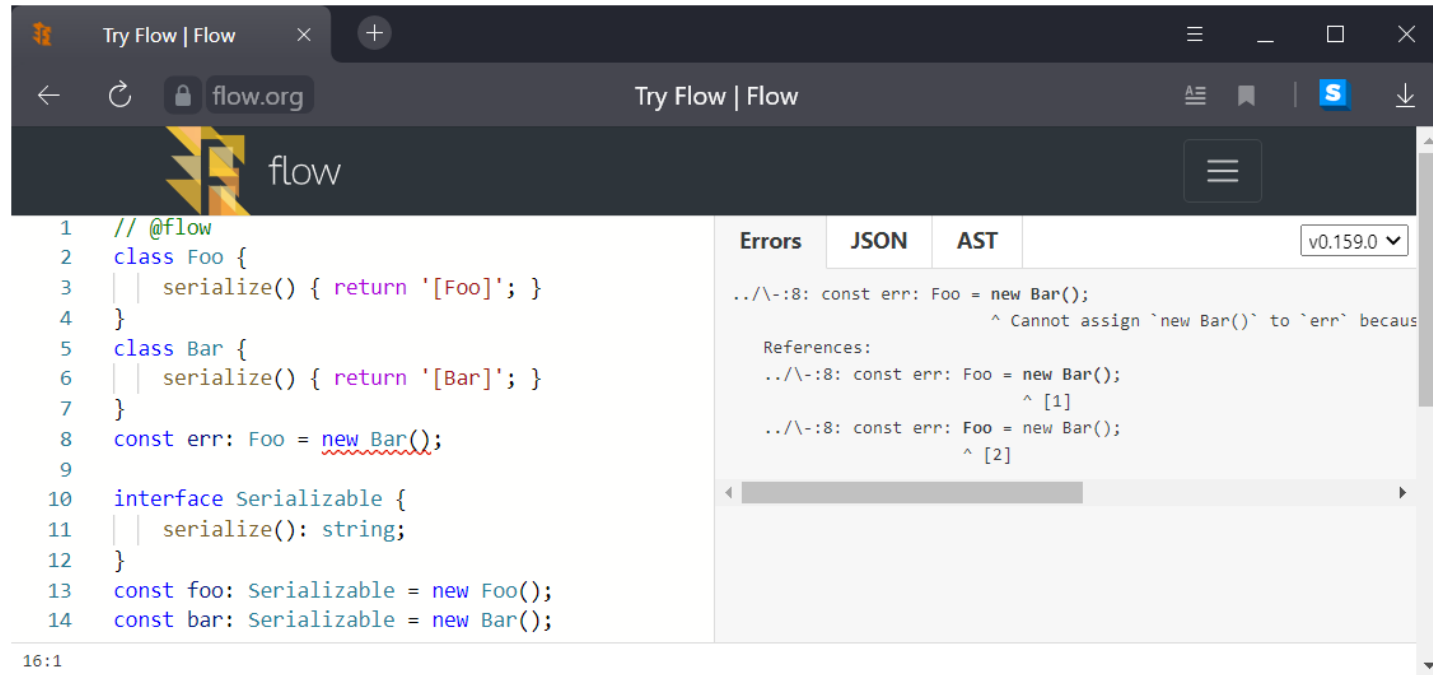


Интерфейсы (1)

79

```
// @flow
class Foo {
  serialize() { return '[Foo]'; }
}
class Bar {
  serialize() { return '[Bar]'; }
}
const err: Foo = new Bar();
interface Serializable {
  serialize(): string;
}
const foo: Serializable = new Foo();
const bar: Serializable = new Bar();
```

Тип Bar не совместим с Foo



Try Flow | Flow

flow.org

Try Flow | Flow

flow

```
1 // @flow
2 class Foo {
3   | serialize() { return '[Foo]'; }
4 }
5 class Bar {
6   | serialize() { return '[Bar]'; }
7 }
8 const err: Foo = new Bar();
9
10 interface Serializable {
11   | serialize(): string;
12 }
13 const foo: Serializable = new Foo();
14 const bar: Serializable = new Bar();
```

Errors JSON AST v0.159.0

../\--:8: const err: Foo = new Bar();
^ Cannot assign `new Bar()` to `err` because

References:

../\--:8: const err: Foo = new Bar();
^ [1]

../\--:8: const err: Foo = new Bar();
^ [2]

16:1

Интерфейсы (2.1)

80

```
// @flow
```

```
interface Invariant {  property?: number | string }
```

```
interface Covariant { +readOnly: number | string }
```

```
class I implements Invariant {  
  property: number | string;  
  constructor(prop: number | string) {  
    this.property = prop;  
  }  
}
```

```
var value1: I = new I(12);
```

```
var value2: Invariant = {}
```

```
value2.property = 12;
```

```
var value3: Covariant = { readOnly: 42 };
```

```
function method1(value: Invariant) {  
  value.property;  
  value.property = 3.14;  
}
```

```
function method2(value: Covariant) {  
  value.readOnly;  
  value.readOnly = 3.14;  
}
```

1. property не совместимо с undefined
2. Нельзя присвоить значение readonly

Интерфейсы (2.2)

81

Try Flow | Flow

flow.org

Try Flow | Flow

flow

```
1 // @flow
2 interface Invariant { property?: number | string
3 interface Covariant { +readOnly: number | string
4
5 class I implements Invariant {
6   property: number | string;
7   constructor(prop: number | string) {
8     this.property = prop;
9   }
10 }
11 var value1: I = new I(12);
12
13 var value2: Invariant = {};
14 value2.property = 12;
15 var value3: Covariant = { readOnly: 42 };
16
17 function method1(value: Invariant) {
18   value.property;
19   value.property = 3.14;
20 }
21 function method2(value: Covariant) {
22   value.readOnly;
23   value.readOnly = 3.14;
24 }
```

Errors

JSON

AST

v0.159.0

..\-:5: class I implements Invariant {
 ^ Cannot implement `Invariant` [1] with `I` because in proper
References:
..\-:5: class I implements Invariant {
 ^ [1]
..\-:2: interface Invariant { property?: number | string }
 ^ [2]
..\-:6: property: number | string;
 ^ [3]
..\-:6: property: number | string;
 ^ [4]

..\-:23: value.readOnly = 3.14;
 ^ Cannot assign `3.14` to `value.readOnly` because

1. property не совместимо с undefined

2. Нельзя присвоить значение readonly

26:1

«Продвинутые» дженерики

```
// @flow
```

```
function logFoo1<T>(obj: T): T {
  console.log(obj.foo);
  return obj;
}
```

```
function logFoo2<T>(obj: T): T {
  if (obj && obj.foo)
    console.log(obj.foo);
  return obj;
}
```

```
logFoo2({ foo: 'foo', bar: 'bar' });
```

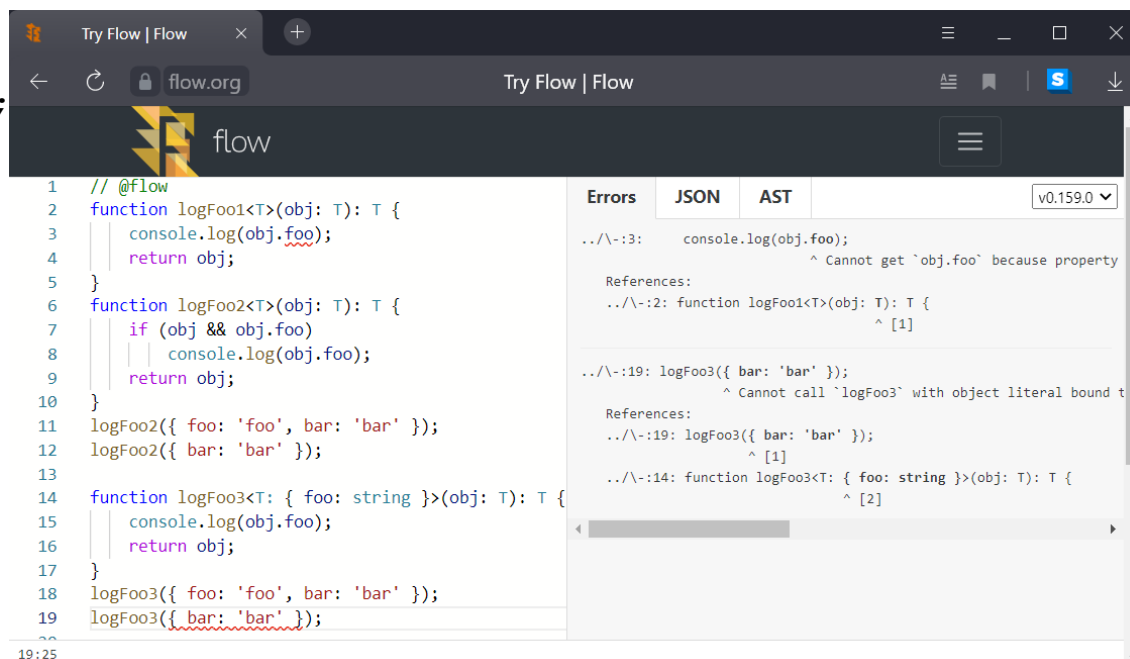
```
logFoo2({ bar: 'bar' });
```

```
function logFoo3<T: { foo: string }>(obj: T): T {
  console.log(obj.foo);
  return obj;
}
```

```
logFoo3({ foo: 'foo', bar: 'bar' });
```

```
logFoo3({ bar: 'bar' });
```

1. Нельзя использовать `foo` – его нет в `T`
2. Нельзя вызвать `logFoo3`, т.к. отсутствует `foo`

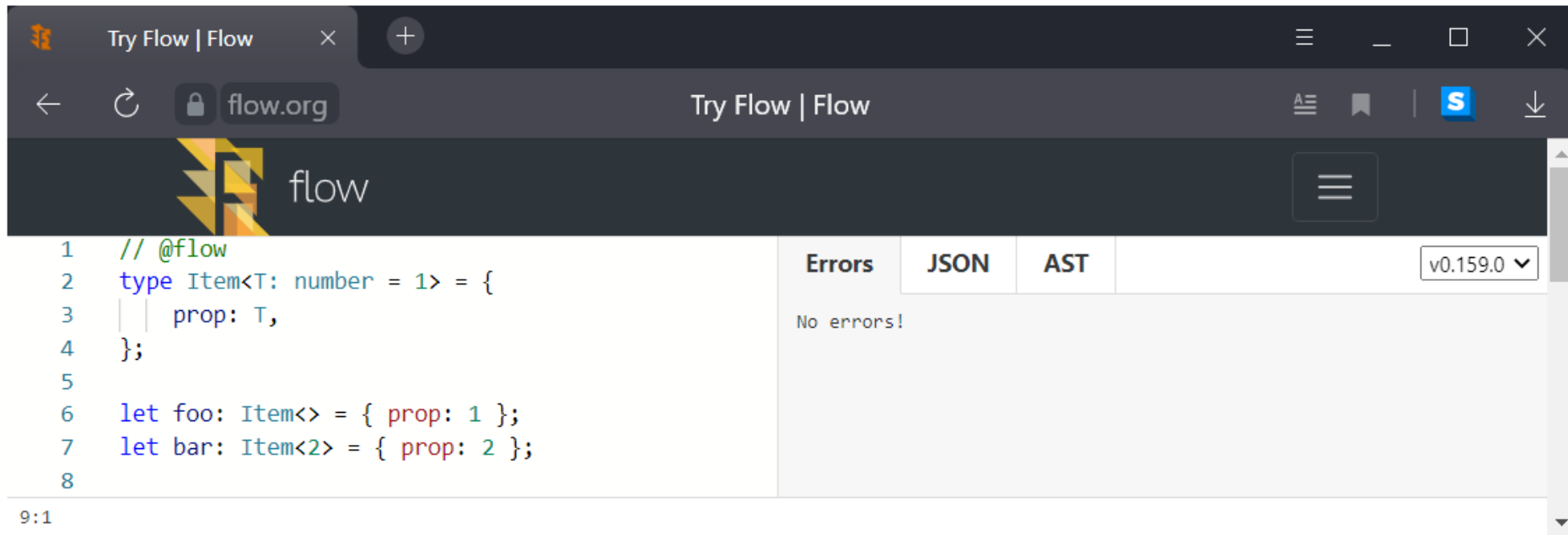


Значения по умолчанию в «дженерике»

83

```
// @flow
type Item<T: number = 1> = {
  prop: T,
};

let foo: Item<> = { prop: 1 };
let bar: Item<2> = { prop: 2 };
```



The screenshot shows the Flow Playground interface. The top bar includes the Flow logo and the text 'flow'. The main editor area displays the code from the previous block, with line numbers 1 through 8 on the left. The right sidebar has tabs for 'Errors', 'JSON', and 'AST', with 'Errors' selected. Below the tabs, it says 'No errors!'. A version dropdown menu in the top right corner shows 'v0.159.0'. The bottom left corner of the interface shows the cursor position '9:1'.

```
1 // @flow
2 type Item<T: number = 1> = {
3   | prop: T,
4 };
5
6 let foo: Item<> = { prop: 1 };
7 let bar: Item<2> = { prop: 2 };
8
```

Errors JSON AST v0.159.0

No errors!

9:1

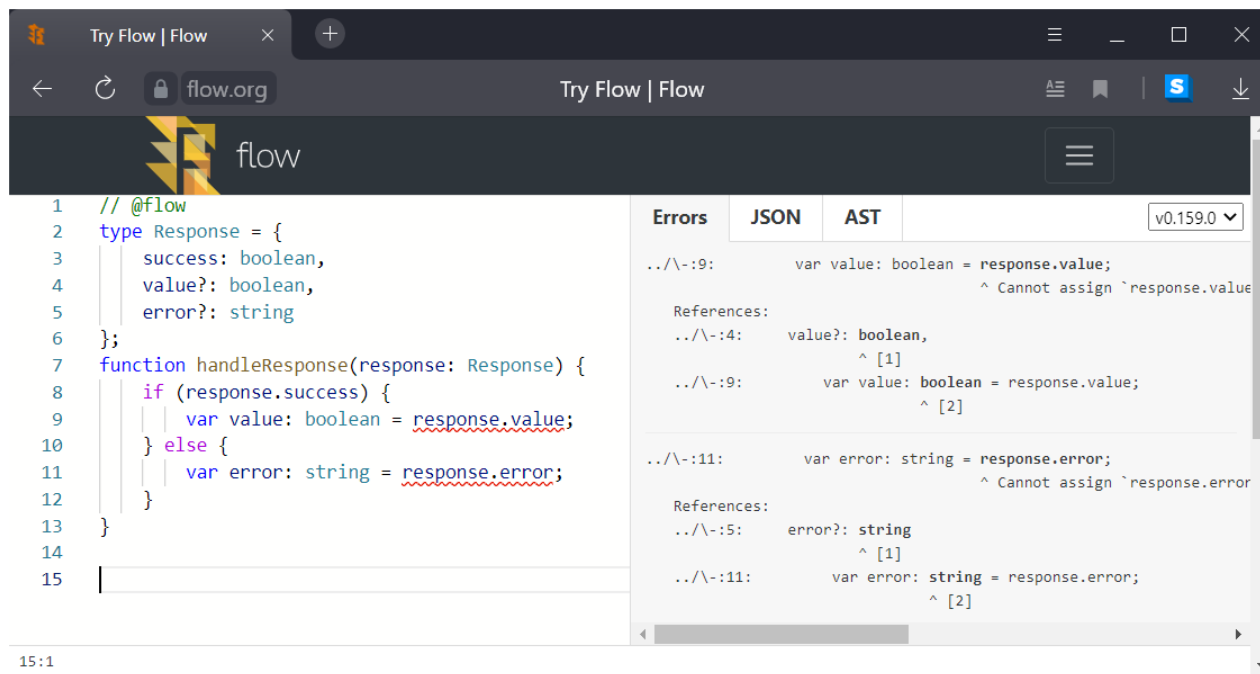
Объединение типов (1)

84

```
// @flow
type Response = {
  success: boolean,
  value?: boolean,
  error?: string
};

function handleResponse(response: Response) {
  if (response.success) {
    var value: boolean = response.value;
  } else {
    var error: string = response.error;
  }
}
```

1. **boolean не совместим с undefined**
2. **string не совместим с undefined**



Объединение типов (2)

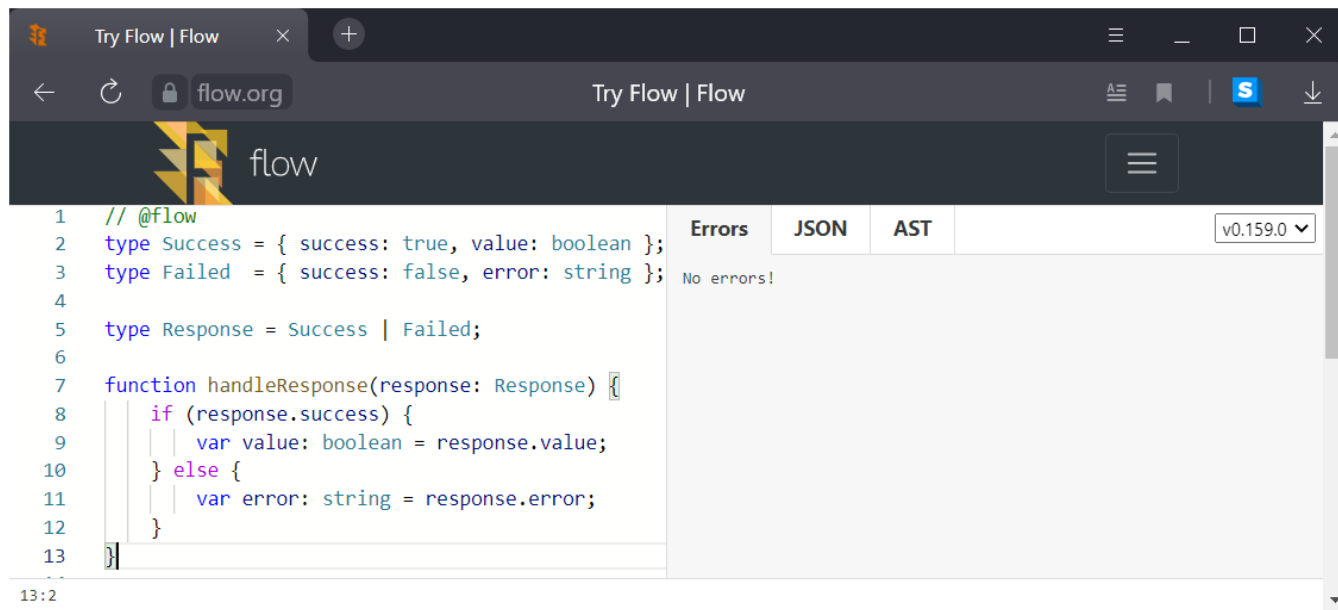
85

```
// @flow
type Success = { success: true, value: boolean };
type Failed   = { success: false, error: string };

type Response = Success | Failed;

function handleResponse(response: Response) {
  if (response.success) {
    var value: boolean = response.value;
  } else {
    var error: string = response.error;
  }
}
```

**Контролируется наличие
полей объединённых
типов**



ИСПОЛЬЗОВАНИЕ НЕСКОЛЬКИХ ТИПОВ

86

```
// @flow
```

```
type A = { a: number };
```

```
type B = { b: boolean };
```

```
type C = { c: string };
```

```
function method(value: A & B & C) {
```

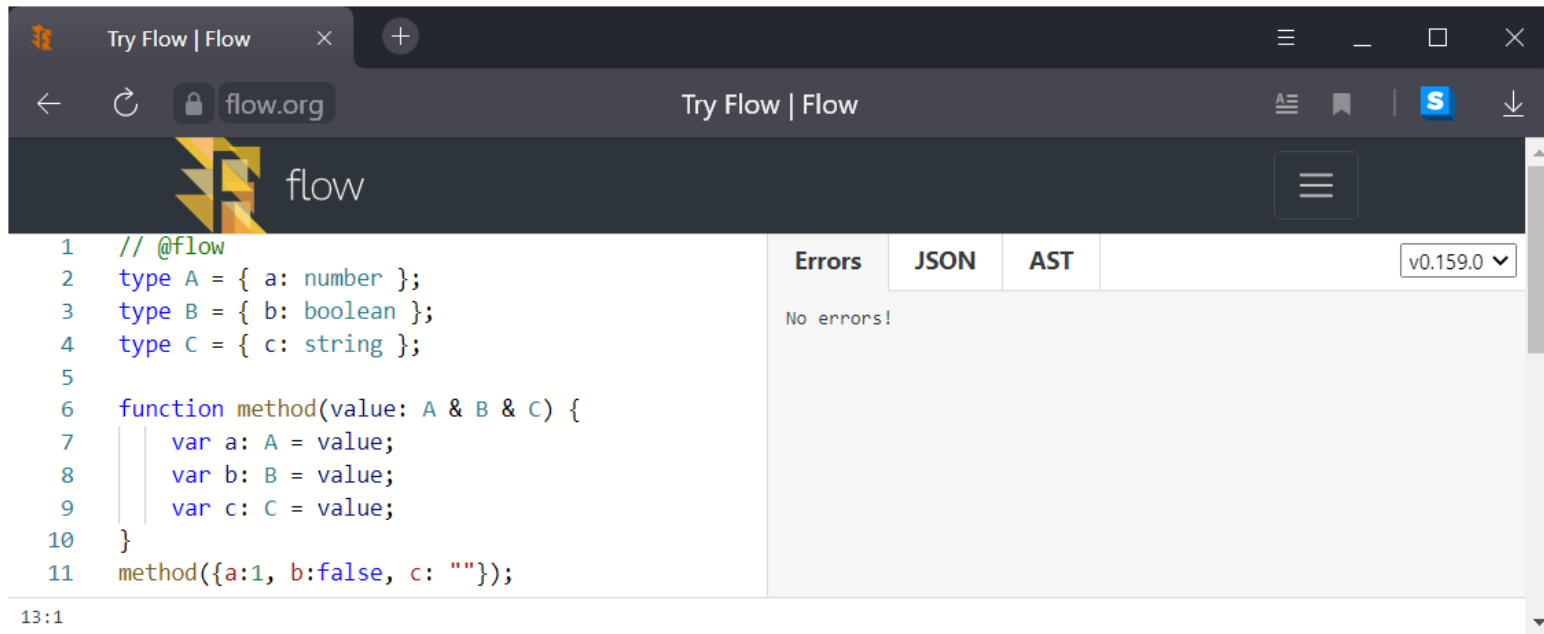
```
  var a: A = value;
```

```
  var b: B = value;
```

```
  var c: C = value;
```

```
}
```

```
method({a:1, b:false, c: ""});
```



Использование typeof

87

```
// @flow
let num1 = 42;
let num2: typeof num1 = 3.14;
let num3: typeof num1 = 'world';

let obj1 =
  { foo: 1, bar: true, baz: 'three' };
let obj2: typeof obj1 =
  { foo: 42, bar: false, baz: 'hello' };
```

**string не совместим с
number**

Против классических в JS:
typeof { foo: true } === 'object'
typeof { bar: true } === 'object'
typeof [true, false] === 'object'

The screenshot shows the Flow IDE interface. The editor displays the same code as the previous block. On line 4, the string 'world' is underlined with a red squiggly line, indicating a type error. The right-hand pane shows the 'Errors' tab with the following message:

```
../\ -:4: let num3: typeof num1 = 'world';
                                   ^ Cannot assign ``world`` to `num3`

References:
../\ -:4: let num3: typeof num1 = 'world';
                                   ^ [1]
../\ -:4: let num3: typeof num1 = 'world';
                                   ^ [2]
```

At the bottom left of the IDE, the text '11:1' is visible.

Приведение типов

```
// @flow
let value = 42;
(value: 42 | 43);
(value: number);
(value: string);
let newValue = ((value: any): string);

(newValue: string);
(newValue: number);
```

1. **number не совместимо с string**
2. **string не совместим с number**

The screenshot shows the Flow IDE interface with the following code in the editor:

```
1 // @flow
2 let value = 42;
3 (value: 42 | 43);
4 (value: number);
5 (value: string);
6 let newValue = ((value: any): string);
7
8 (newValue: string);
9 (newValue: number);
10
11
```

The right-hand pane displays the 'Errors' tab, showing two type errors:

Error 1: Cannot cast `value` to string because number [1] is incompatible. References: `../\:-:2: let value = 42;` and `../\:-:5: (value: string);`.

Error 2: Cannot cast `newValue` to number because string [1] is incompatible. References: `../\:-:6: let newValue = ((value: any): string);` and `../\:-:9: (newValue: number);`.

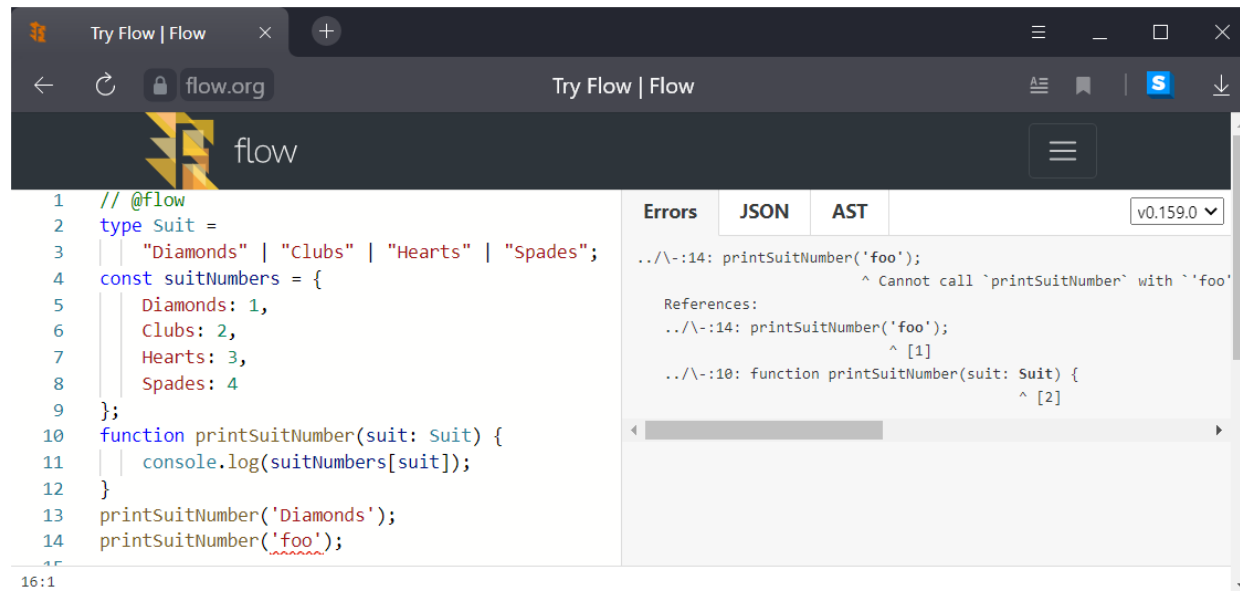
The status bar at the bottom indicates line 11, column 1.

Перечисления с использованием Flow

89

```
// @flow
type Suit =
  "Diamonds" | "Clubs" | "Hearts" | "Spades";
const suitNumbers = {
  Diamonds: 1,
  Clubs: 2,
  Hearts: 3,
  Spades: 4
};
function printSuitNumber(suit: Suit) {
  console.log(suitNumbers[suit]);
}
printSuitNumber('Diamonds');
printSuitNumber('foo');
```

**string не
совместим с
union-литералом**



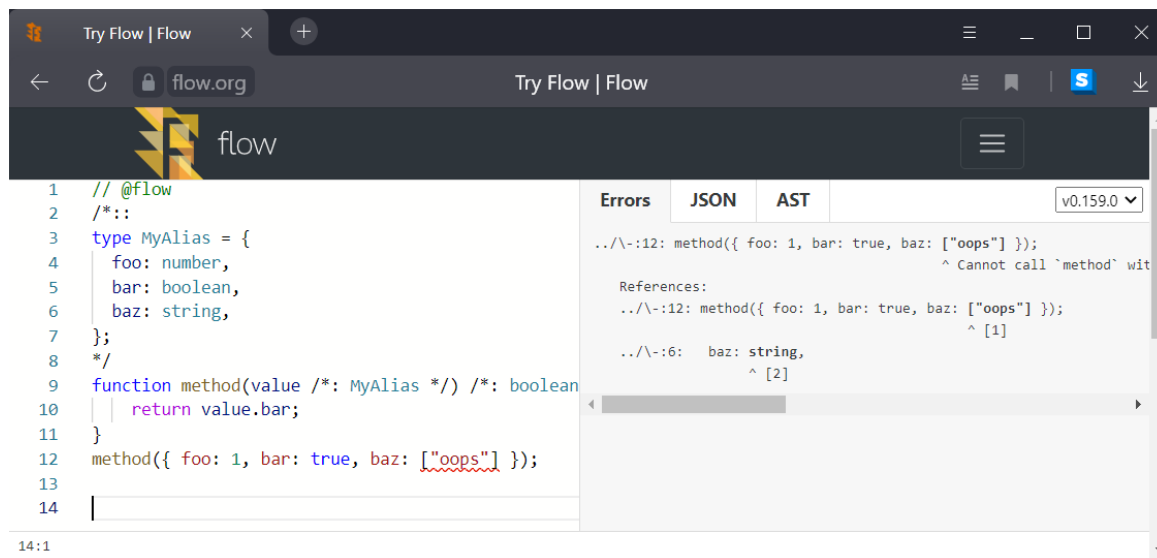
Использование обрабатываемых комментариев

```
// @flow
/*::
type MyAlias = {
  foo: number,
  bar: boolean,
  baz: string,
};
*/

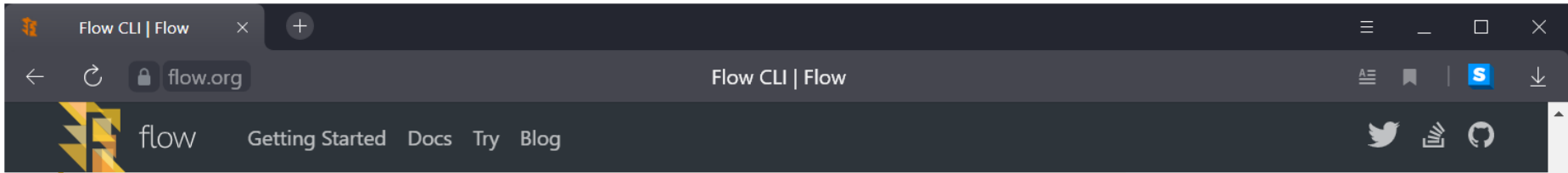
function method(value /*: MyAlias */) /*: boolean */ {
  return value.bar;
}

method({ foo: 1, bar: true, baz: ["oops"] });
```

**Массив string не
совместим со string**



Использование Flow в командной строке⁹¹



This will give you information about everything that flow can do. Running this command should print something like this:

```
Usage: flow [COMMAND] [PROJECT_ROOT]
```

Valid values for COMMAND:

ast	Print the AST
autocomplete	Queries autocompletion information
batch-coverage	Shows aggregate coverage information for a group of files or directories
check	Does a full Flow check and prints the results
check-contents	Run typechecker on contents from stdin
config	Read or write the .flowconfig file
coverage	Shows coverage information for a given file
cycle	Output .dot file for cycle containing the given file
find-module	Resolves a module reference to a file
find-refs	Gets the reference locations of a variable or property
force-recheck	Forces the server to recheck a given list of files
get-def	Gets the definition location of a variable or property
get-imports	Get names of all modules imported by one or more given modules
graph	Outputs dependency graphs of flow repositories
init	Initializes a directory to be used as a flow root directory
ls	Lists files visible to Flow
lsp	Acts as a server for the Language Server Protocol over stdin/stdout [experimental]
print-signature	Prints the type signature of a file as extracted in types-first mode

<https://flow.org/en/docs/cli/>

Преимущества использования flow 92

- Контроль типов массивов
- Возможность использовать кортежи
- Возможность создания объектов с фиксированной структурой
- Использование сокращений записи (алиасы для типов)
- Контроль перечня полей классов
- Возможность использования generics в JavaScript

Вопросы для самопроверки

- Кто разрабатывает и поддерживает TypeScript?
- В чем ключевое отличие TS от JS?
- Какие типы появились в TS в отличие от JS?
- Какие варианты объединения `type` вы знаете? В чём их отличие?
- В чём отличие атрибутов и методов классов TS от JS?
- Как в TS реализуется перегрузка функций?
- Как можно организовать «защиту» используемого типа?
- Что такое `generics`? Какие они бывают в TS?
- Что такое пространство имён?
- Что такое декораторы? Для чего они могут применяться?
- Что такое JSX? Кто его разрабатывает и поддерживает?
- Для чего нужен Flow?
- Зачем ему для работы нужен Babel?
- Без какой команды Flow не будет работать (оба варианта)?
- Какие новые возможности Flow добавляет в JavaScript?
- Что такое обрабатываемые комментарии?