TypeScript

Basic Types

/* Екип 18 */

01

Primitive Data Types

```
// number  bigint
// string  symbol
// boolean
// undefined null
```

```
number {

    Stored as double-precision 64-bit number
    Special values - NaN


}

bigint {

    Needed configurations
    It's an int, duh…
    Arithmetics, comparison and bitwise operations

}
```

Линк към пример

```
string {

  String literals
  Methods

}    symbol {

    Characteristics

  }
```

Линк към пример

```
boolean {

    Declarations
    Falthy and Truthy values

}
```

Линк към пример

```
null  {

    Only null as a value
    Undefined only if strictNullChecks is disabled


}
   undefined{
    Only undefined as a value
    Null is a value if strictNullChecks is disabled

}
```

Линк към пример

02

Interfaces and type aliases

```
// Interfaces
// Type aliases
// Interfaces vs Type aliases
```

Линк към пример

Interface

```
interface Person{

    name: string;
    age: number;
    speak (text: string): void;
    isHungry: boolean;


}
```

Type aliases

```
type Person = {
    name: string;
    age: number;
    speak (text: string): void;
    isHungry: boolean;

}
```

Interface vs Type aliases

Feature	I	T	Example
Primitives	❌	✅	`type UUID = string`
Extend / Intersect	✅	✅	`Response & ErrorHandling`
Unions	❌	✅	`string | number`
Mapped object types	❌	✅	`['apple' | 'orange']: number`
Augment existing types	✅	❌	`declare global { interface Window { … } }`
Declare type with typeof	❌	✅	`Response = typeof ReturnType<fetch>`
Tuples	✅	✅	`[string, number]`
Functions	✅	✅	`(x: number, y: number): void`
Recursion	✅	✅	`Nested { children?: Nested[] }`

03

Optional properties and Optional chaining

```
// Optional properties
// '| undefined'
// '.' and '?.' - differences
// using '?.' with parameters,
functions, expressions, arrays
// combining '?.' with  the nullish
coalescing operator '??'
```

Interface with optional properties

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

Линк към пример

Combining ‘?’ and ‘| undefined’

```
interface Person{
gender?: "male" | "female";
}


interface Person{
gender?: "male" | "female" | undefined;
}
```

Линк към пример

Optional chaining

```
const adventurer = {
  name: 'Alice',
  cat: {
    name: 'Dinah'
  }
}
```

Линк към пример

Optional chaining - syntax

```
Obj.val?.prop
Obj.val?.[expr]
Obj.arr?.[index]
obj.func?.(args)
```

Линк към пример

Dealing with optional callbacks or event handlers

```
function doSomething(onContent, onError) {
  try {
    // ... do something with the data
  }
  catch (err) {
    if (onError) { // Testing if onError really exists
      onError(err.message);
    }
  }
}
```

```
function doSomething(onContent, onError) {
  try {
   // ... do something with the data
  }
  catch (err) {
    onError?.(err.message); // no exception if onError is undefined
  }
}
```

Combining with the nullish coalescing operator

```
let customer = {
  name: "Carl",
  details: { age: 82 }
};

const customerCity = customer?.city ?? "Unknown city";
console.log(customerCity); // Unknown city
```

04

Union types

```
function printId(id: number |
string) {
console.log("Your ID is: " +
id);
}
// OK
printId(101);
// OK
printId("202");
```

```
function printId(id: number | string) {
 if (typeof id === "string") {
   // In this branch, id is of type 'string'
   console.log(id.toUpperCase());
 } else {
   // Here, id is of type 'number'
   console.log(id);
 }
}
```

Линк към примери

Literal types and “as const”

```
let changingString = "Hello World";
changingString = "Olá Mundo";


const constantString = "Hello World";
```

```
const req = { url: "https://example.com", method: "GET" } as const;
handleRequest(req.url, req.method);
```

Линк към примери

“Null”, “undefined” and non-null assertion

strictNullChecks

Post-fix operator “!”

Линк към примери