SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

JavaScript errors

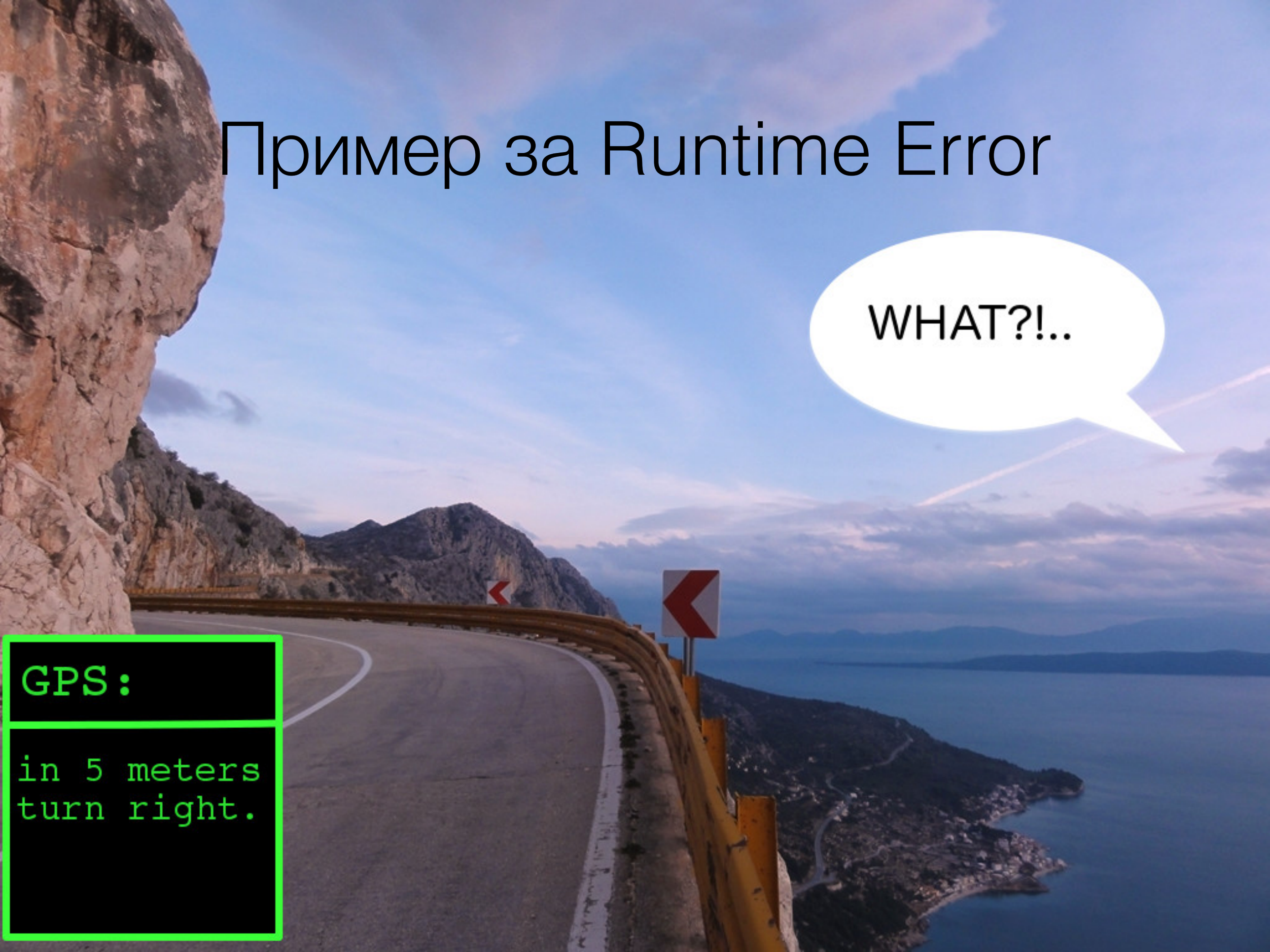
JavaScript грешки

От време на време кода, който пишем води до появяването на грешки в JavaScript конзолата.

Това са така наречените: **Runtime Errors**.

Наричат се така, защото се случват по време на изпълнението на програмата (скрипта) и като цяло не са хубаво нещо..

Пример за Runtime Error



JavaScript грешки

- Всяка грешка спира по-нататъшното изпълнение на нашия скрипт!!!
- Могат да бъдат 2 вида:
  - системни (когато кода ни се "чупи" някъде)
  - user defined (*собствени*) грешки: това са *нарочни* грешки, които се използват за **flow control**

  ***flow** наричаме хода на програмата (кое след кое се случва)*
- User defined грешките могат (и трябва!) да се опработват

Обработка на грешки
(Error handling)

“Errors Will Happen!”

- W3Schools

Неочаквани грешки

Наричаме ги още изключения (**Exceptions**) защото обикновенно не са част от нормалното изпълнение на скрипта

```
var coffeeMachine = {
  water: 1000,
  coffee: 200,
  cups: 15
};
cafeMachine.water; /* ReferenceError:
                      cafeMachine is
                      not defined */
```

Очаквани (предвидими) грешки

които могат да се избегнат

```
if (gps.isBroken) {
    drive_without_navigation();
}
else {
    gps.navigate();
}
```

Очаквани (предвидими) грешки

които НЕ могат да се избегнат

```
// gps has no property "isBroken"

try {
    gps.navigate();
}
catch(error) {
  ...
  // Обработка на грешката
}
```

Добри и лоши грешки

- Не всички грешки са лоши
- Често грешките ни помогат да контролираме нещата в програмата:

```
if (machine.water === 0) {
  throw new Error("There's no water in the tank");
}
```

- Обикновено лошите грешки са онези, които не очакваме
- Те са най-често грешки в нашата логика и изискват дебъгване на кода, за да ги открием

Catch

Прихващане на грешки

- ***Важно**: Използва се само за добрите грешки! Тоест:*
  - само когато знаем, че изпълняваме код, който може да "изгърми" (да предизвика грешка)
  - при това знаем че грешката не е предизвикана от сбъркана логика, а е очаквана и служи за flow control
- За да прихванем една грешка използваме конструкцията **try-catch**:

```
try {
  machine.makeCoffee;
}
catch (error) {
  console.log("can I offer you a Coke instead ?")
}
```

Пример:

```
try {
    gps.navigate();
}
catch(error) {
    if (error.name === “NavigationError”) {
      console.warn(“This GPS is broken!”);
      drive_without_navigation();
    }
    else {   /* any other error */
      console.error(“Unexpected error: ” + error.message);
      console.log(error.stack);
      throw error;   /* don't sweep under the carpet the
                        unexpected error */
    }
}
```

Кога се налага?

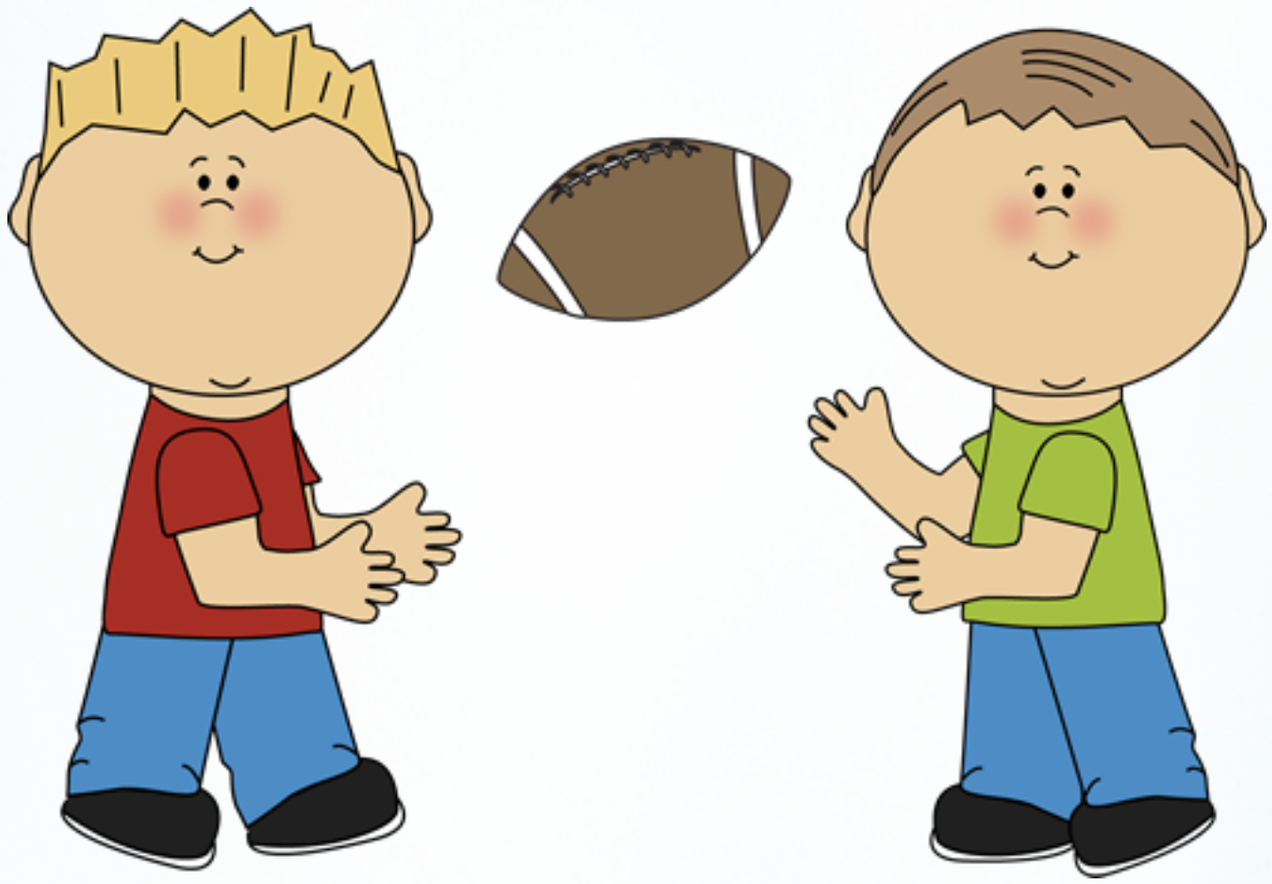
- Често функциите ни очакват да получат параметър от определен тип, например обект или Array, но ако вместо това подадем число, може взичко да се объркa и накрая да получим грешен резултат

- Използваме собствените грешки за да гарантираме че параметрите, които сме получили отговарят на това, което очакваме

- Така се подсигуряваме че функцията ни или ще работи, или ще изведе грешка, но няма да върне грешен резултат

- Често грешките се използват и за **flow control**

Throw



```
function Person(firstName, lastName, birthdateString) {
  if (firstName == undefined || lastName == undefined) {
    throw new Error("you must pass first name and last name");
  }
  if (new Date(birthdateString === "Invalid Date") {
      throw new Error(“wrong birthdate format!”);
  }
  this.birthdate = new Date(birthdateString);
  this.firstName = firstName;
  this.lastName = lastName;
}

try { new Person(); } catch (error) { console.log("Uh-oh!"); }
```

Собствени грешки

- Грешките могат да се ползват за контрол на хода на изпълнението на програмата. За целта обаче, трябва да можем да си ги произвеждаме

- За да си направим грешка, използваме конструктора на Error класа (или негов наследник) по следния начин:

```
var myError = new Error(ERROR_MESSAGE);
```

- Когато си направим грешката, можем да я хвърлим, с което ще предизвикаме аварийно излизане от текущият контекст и всички съдържащи го, до достигане на try-catch блок или до глобалния scope

```
throw myError;   // if not in try-catch block, will become a program error
```

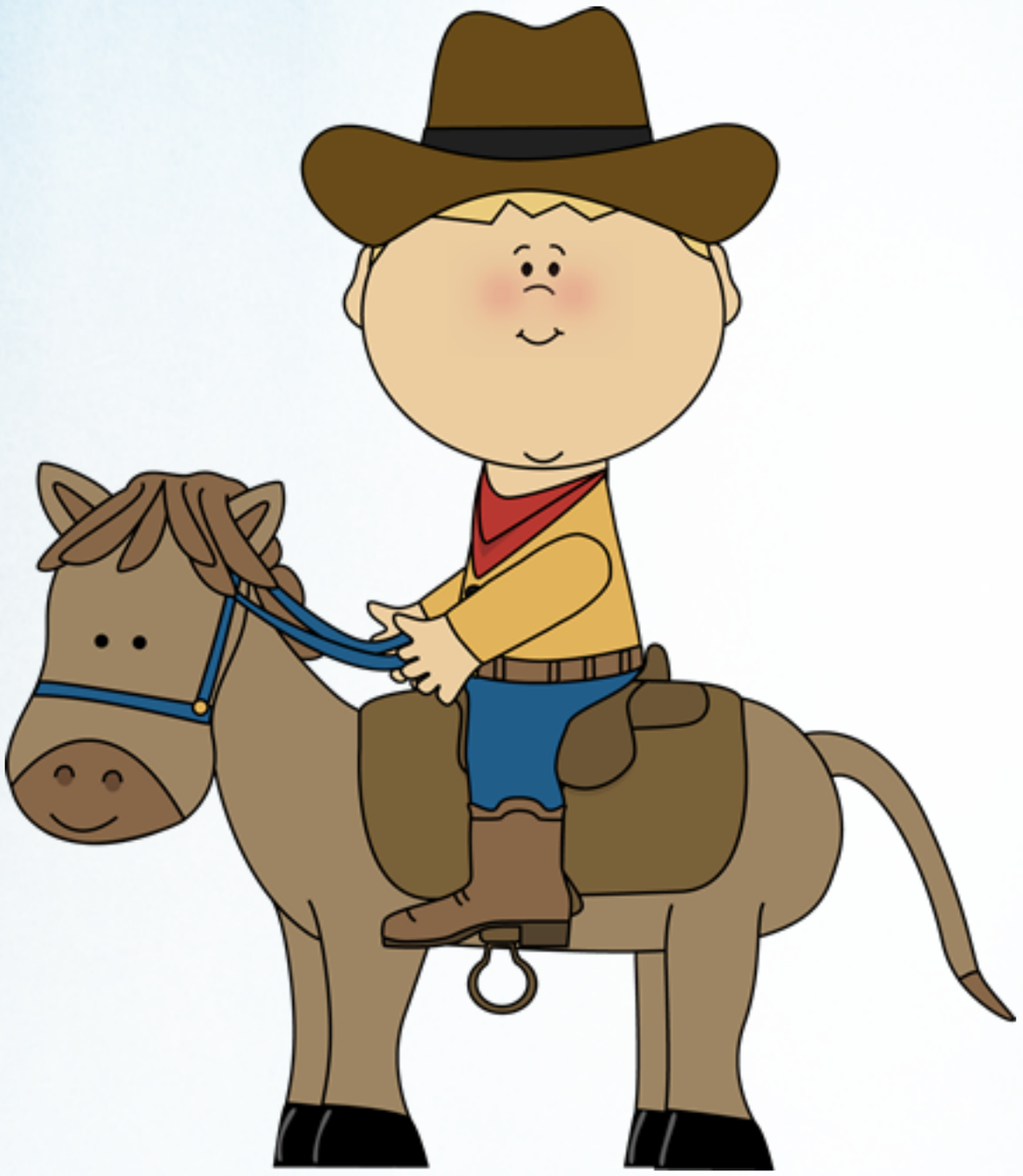
класът Error

- полета: **name**, **message**
- методи: **stack**()
- типове грешки
  - вградени (които са част от езика): SyntaxError, ReferenceError, RangeError и т.н.
  - собствени (такива, които сме си направили сами): DateOfBirthError, PersonNamesError, NoCoffeeError
- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Error#Custom_Error_Types

```
// Create a new object, that prototypically inherits from the Error
function MyError(message) {
  this.name = 'MyError';
  this.message = message || 'Default Message';
  this.stack = (new Error()).stack;
}
MyError.prototype = Object.create(Error.prototype);
MyError.prototype.constructor = MyError;

try {
  throw new MyError();
} catch (e) {
  console.log(e.name);     // 'MyError'
  console.log(e.message);  // 'Default Message'
}

try {
  throw new MyError('custom message');
} catch (e) {
  console.log(e.name);     // 'MyError'
  console.log(e.message);  // 'custom message'
}
```



Finally

Finally

- try {
  // do some unstable coding with eventual bangs
  }
  catch (e) {
  // handle the bangs (if any)
  }
  finally {
  // no matter if there were bangs or not - do some final stuff here
  }

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#The_finally_clause

Error wrapping

Понякога искаме да опаковаме една грешка в друга грешка - това се нарича error wrapping

```
try {
  person = new Person("12-03-1987");
} catch (err) {
  if (err.name === "DateOfBirthError") {
    throw new ParamsError(err.message);
  }
  throw err;
}
```

Въпроси?

Примери

http://swift-academy.zenlabs.pro/lessons/lesson20/examples/download.zip

Домашно

http://swift-academy.zenlabs.pro/lessons/lesson20/homework