ES5

…umí všechny „moderní prohlížeče“…

…lze jej bez obav používat již dnes…

Striktní mód

- Aktivuje se pomocí řetězce 'use strict'; v hlavičce JS souboru.
- Způsob jak aktivovat striktnější mód vykonávání JS v prohlížeči.
- Brání některým chybám.
- Některé vlastnosti striktního módu:
  - Brání použití nedeklarovaných proměnných.
  - Zakazuje zanořené funkce.
  - „this“ uvnitř funkcí, které nejsou metody je „undefined“ a ne „window“

Accessors

- Podobné jako property v C#.
- Umožňují definovat get a set „kód“.

```
var obj = {
    __time: null,
    get time() {
        console.log('get');
        return this.__time;
    },
    set time(value) {
        console.log('set', value);
        this.__time = value;
    }
};
```

Práce s prototypy - Object.create()

- Vytvoří objekt na základě prototypu.
- Object.create(proto, [propertiesObject])

```
function Car() { }

Car.prototype.speed = 10;

var myCar = Object.create(Car.prototype, {
    maxSpeed: {
        value: 180,
        writable: false
    }
});
```

propertiesObject

- **configurable** – je možné měnit vlastnosti této vlastnosti (v podstatě aplikovat níže uvedená nastavení) a smazat ji.
- **enumerable** – je vlastnost viditelná při procházení cyklem for .. in?
- **value** - hodnota
- **writable** – lze změnit hodnotu (přiřadit do dané proměnné)?
- **get** – funkce, která vrací hodnotu dané vlastnosti (je volána při čtení).
- **set** – funkce, která nastavuje hodnotu dané vlastnsti (je volána při zápisu).

TODO

Vytvořte objekt „User“ s vlastností „income“, která bude nastavena na 30 000 a nepůjde měnit.

Úlohu zopakujte s tím rozdílem, že při pokusu o změnu příjmu se vypíše hláška do konzole (změna se ale nepovolí).

Práce s properties - Object.defineProperty()

- Umožňuje definovat vlastnost včetně jejího nastavení.
- Object.defineProperty(object, [propertyName], [attributes]);
- Object.defineProperties() – více property současně.

```
var obj = {};

Object.defineProperty(obj, 'speed', {
    value: 42, // hodnota
    writable: false, // nelze změnit hodnotu
    configurable: false, // nelze změnit vlastnosti
    enumerable: false // nelze procházet pomocí for ... in
});
```

Práce s prototypy - Object.getPrototypeOf()

- Vrátí prototyp ze kterého objekt vznikl.
- Lze volat „rekurzivně“ a tím projít celým řetězcem prototypů daného objektu.

Práce s properties – další metody

- Object.getOwnPropertyDescriptor(obj, propName);
  - Vrátí vlastnosti dané vlastnosti.
- Object.keys(obj);
  - Vrátí seznam názvů vlastností tak jak by jsme je dostali pomocí cyklu for … in
- Object.getOwnPropertyNames(obj)
  - Vrátí seznam názvů vlastností. Obsahuje i vlastnosti, které nejsou enumerable.

Zabezpečení objektu

- Object.preventExtensions(obj)
  - Brání rozšíření objektu o další vlastnosti (obdoba sealed v C#).
- Object.isExtensible(obj)
- Object.seal(obj)
  - Totéž co preventExtension, ale brání také odstranění vlastností a modifikaci atributů daných vlastností (writable, configurable, enumerable).
- Object.isSealed(obj)
- Object.freeze(obj)
  - Zamezí jakékoliv změně objektu a jeho hodnot.
- Object.isFrozen(obj)

Pole

- Array.isArray()
- Array.prototype.every()
- Array.prototype.filter()
- Array.prototype.forEach()
- Array.prototype.indexOf()
- Array.prototype.lastIndexOf()
- Array.prototype.map()
- Array.prototype.reduce()
- Array.prototype.some()

Text

- String.prototype.trim()

JSON

- JSON.parse(txt)
- JSON.stringify(obj)

Datum

- Date.now()
- Date.prototype.toISOString()

ES6

ECMAScript 2015

… umí jej aktuální verze prohlížečů…

Díky tržnímu podílu starších prohlížečů jej stále **není možné bezpečně používat „sám o sobě“.**

https://caniuse.com/

const a let

- Nové typy viditelnosti proměnných.
- const myVariable = value;
  - Znáte z jiných jazyků….
- let myVariable = value;
  - Blokově viditelná proměnná bez hoistingu.
  - Nelze ji „předeklarovat“.

```
let a = [1, 2, 3];
for (let i = 0; i < a.length; i++) {
        let x = a[i];
}
console.log(i, x); // chyba
```

```
let x = 10;

{
    let x = 5;
    console.log(x);
}

console.log(x); // 10
```

let

```
let elements = document.querySelectorAll('div');

for (let i = 0; i < elements.length; i++) {

    elements[i].addEventListener('click', function () {

        console.log(i); // i NEBUDE stejné pro všechny!

    });

}
```

Arrow funkce

- Podobné jako lambda výraz v C#.
- var fnX = x => x * x;
  fnX(10);

- var fnY = y => { return y * y; }
  fnY(10);

- var fnXY = (x, y) => x* y;
  fnXY(2, 3);

Arrow funkce a „this“

- Arrow funkce je „transparentní“ vůči „this“.
- „this“ uvnitř arrow funkce se chová jako kdyby nebylo uvnitř funkce.

```
function Car() { }

Car.prototype.a = () => {
        console.log('arrow function', this);
};

Car.prototype.b = function () {
        console.log('standard function', this);
};

var myCar = new Car();
myCar.a();
myCar.b();
```

Výchozí hodnoty argumentů

- Stejné jako v C#.

```
function myPow(x, y = 10) {
        return Math.pow(x, y);
}

myPow(2);
myPow(2, 3);
```

Rest argumenty

- „Zbývající“ argumenty funkce se předají jako pole.
- Rest parametr začína „…“

```
function sum(x, y, ...other) {
    let sum = x + y;
    for (let i = 0; i < other.length; i++) {
        sum += other[i];
    }
    return sum;
}

sum(1, 2, 3, 4, 5);
```

Spread operátor

- Rozbije pole na argumenty funkce, nebo prvky jiného pole.
- Značí se „…“.

```
let nums = [1, 2, 3];

function sum(label, x, y, z) {
        return label + ': ' + (x * y * z);
}

sum('obsah ', ...nums);
```

```
let nums = [1, 2, 3];
let moreNums = [-1, 0, ...nums, 4, 5, 6];
```

Template literals (Template strings)

- Nový typ textového řetězce.
- Text musí být v uvozovkách typu „` `“
- Umožňuje interpolaci pomocí „${…}“ - může obsahovat kód.
- Umožňuje odřádkování.
- Částečně znáte z C#.

```
let name = 'Honzo';
let txt = `
    Ahoj ${name}, jak se máš?
    Číslo je ${Math.random()}
`;
```

Tegged template

- Vlastní logika pro template string.

```
function tagFunction(strings, arg1) {
        return strings[0] + '<strong>' + arg1 + '<strong>' + strings[1];
}
var str = tagFunction `Ahoj ${name}, jak se máš.`;

console.log(str);
```

TODO

Vytvořte Tagged template, který přidá všem číselným proměnným sufix „Kč“ a všechny ne číselné proměnné převede na velká písmena.

Zkrácený zápis vlastností objektů

- Znáte z C#.

```
let name = 'Jan';
let x = 10;

let obj = {
    name,
    x
};
```

Výpočet názvů vlastností

```
let prefix = 'myPrefix_';

let obj = {
    [prefix + 'x']: 10,
    [prefix + 'y']: 50
};
```

Destruktivní přiřazení

```
let nums = [1, 2, 3];

let [a, , c] = nums;
```

Destruktivní přiřazení

```
let nums = [1, 2, 3];

let [a, , c] = nums;

let [a, , c, d = 10] = nums;
```

```
let customer = {
    name: 'test',
    age: 30,
    childCount: 3
};

let { name, age } = customer;

let { name, age, hasWife = false } = customer;

let { name: customName, age, hasWife = false } = customer;
```

Destruktivní přiřazení

```
let customer = {
    name: 'Jan',
    account: { balance: 5 }
};
let { name: customerName, account: { balance: customerBalance } } = customer;

console.log(customerName);
console.log(customerBalance);
```

```
let arr = [
    { name: 'Jan', age: 30 },
    { name: 'Michala', age: 31 }
];

let [{ age: firstAge }, { age: secondAge }] = arr;

console.log(firstAge);
console.log(secondAge);
```

Destruktivní přiřazení – volání funkcí

```
function print([a, , c]) {
        console.log(a, c);
}

let nums = [1, 2, 3];
print(nums);
```

```
function print({ name, age }) {
        console.log(name, age);
}

let customer = {
    name: 'test',
    age: 30,
    childCount: 3
};

print(customer);
```

TODO

Upravte / vyzkoušejte si nové funkce z ES6 na příkladu s geometrickými tvary.

Třídy

- **Syntactical sugrar…..**
- Ve výsledku jde stále o prototypování!
- Žádné modifikátory viditelnosti (zatím).
- Dědičnost přes „extends“.
- Volání přetížených metod přes „super“.
- Žádná deklarace instančních proměnných!
- Modifikátor static.
- Podpora accessorů (get, set).

```
class FirstClass {

    constructor() {
    }

    getName() {
        return 'Jan';
    }

}

let obj = new FirstClass();
```

```
class Car {
    constructor(speed) {
        this.speed = speed;
    }
    getSpeed() {
        return this.speed;
    }
}

class RedCar extends Car {
    constructor() {
        super(90);
    }

    getSpeed() {
        return super.getSpeed() * 2;
    }

    get color() {
        return 'red';
    }
}

let obj = new RedCar();
```

Moduly (ES6 modules)

- Modul ≈ soubor obsahující JS kód.
- Moduly automaticky používají striktní mód.
- Kód uvnitř modulu je „lokální“ = viditelný jen uvnitř modulu.
- Modul může exportovat část kódu = učinit jej viditelný pro jiné moduly.
- Moduly mohou importovat jiné moduly = používat exportovaný kód jiných modulů.
- Moduly se vkládají do stránky pomocí:
  <script **type="module"** src="...."></script>

Moduly (ES6 modules)

- Export kódu
  - Exportovat lze funkce třídy a proměnné typu var, let a const.
  - Kód musí být označen klíčovým slovem „export“.

```
export function myFn() { }

export var name = "Jan";



function myFn2() { }
var name2 = "Ondra";

export { myFn2, name2 };
```

Moduly (ES6 modules)

- Import modulů
  - Importovat lze jen exportovaný kód.
  - import …… from ……..;

```
import { myFn, name } from './my-module.js';

myFn();

console.log(name);
```

```
import * as myModule from './my-module.js';
console.log(myModule);
```

Moduly - default

```
export default function myFn() {
}
```

```
import someFunction from './my-module.js';
console.log(someFunction);
```

je alias pro…

```
import { default as someFunction } from './my-module.js';
console.log(someFunction);
```

TODO

Upravte úlohu s geometrickými tvary, tak aby používaly třídy a moduly.

Symboly

- 7. datový typ – Undefined, Null, Boolean, Number, String, Object, **Symbol**
- Symbol = hodnota jako žádná jiná ☺
- Symbol lze použít jako název vlastnosti. Tato vlastnost není bez znalosti symbolu přístupná (nejde ani enumerovat).
- Symbol("test") != Symbol("test");

Symboly

```
let mySymbol = Symbol();
let obj = {};

// tato hodnota je přístupná jen se "znalostí" symbolu
obj[mySymbol] = 5;

// 5
console.log(obj[mySymbol]);

// undefined
console.log(obj[Symbol()]);

// prázdný výpis
for (let i in obj) {
        console.log(obj[i]);
}
```

Symboly

- Existují 3 „typy“:
  - Symbol([desc])
    - Nový unikátní symbol.
  - Symbol.for(string)
    - Symbol pro daný „klíč“ (pro 1 klíč se vrací pokaždé ten samý symbol).
  - Předdefinované symboly:
    - Symbol.iterator - pro iterátory.
    - Symbol.toPrimitive – funkce pro převod objektu na primitivní typ.
    - Symbol.match – pro regulární výrazy
    - A spousta dalších: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol

TODO

Vytvořte třídu „Customer“ s jménem a příjmením.

Vytvořte třídu „Bank“ s metodou „init, které se předá zákazník a částka. V rámci metody „init“ dojde k nastavení stavu konta zákazníka (pomocí symbolu – na objektu customer).

Pro uložení stavu konta a čísla kreditní karty využijte symbol.

Vytvořte metody „deposit“ (vložení částky), „withdraw“ (výběr částky) a „getBalance“ (získání stavu konta).

for... of

- Nový typ cyklu umožňující procházení kolekcí s iterátorem.

```
let nums = [1, 2, 3];

for (let val of nums) {
        console.log(val);
}
```

Iterátory

- Umožňují procházení objektů pomocí for … of
- Objekt musí mít vlastnost s názvem Symbol.iterator.
- Iterátor musí vracet objekt s metodou „next“ (generátor).
- Metoda „next“ musí vracet objekt s vlastnostmi „done“ a „value“.

```
class Range {
    constructor(from = 0, to = 10) {
        this.from = from;
        this.to = to;
    }

    [Symbol.iterator]() {
        let from = this.from;
        let to = this.to;
        return {
            next() {
                from++;
                return {
                    value: from - 1,
                    done: from - 1 > to
                };
            }
        };
    }
}

var range = new Range(5, 20);
for (let n of range) {
        console.log(n);
}
```

Generátory

- Funkce před jejímž názvem je * a v rámci níž je použit yield.
- Znáte z C#

```
function* firstGenerator() {
    for (let i = 0; i < 10; i++) {
        yield i;
    }
}

for (let val of firstGenerator()) {
        console.log(val);
}
```

Generátory

```
class Range {
    constructor(from = 0, to = 10) {
        this.from = from;
        this.to = to;
    }

    *[Symbol.iterator]() {
        for (let i = this.from; i <= this.to; i++) {
                yield i;
        }
    }
}


var range = new Range(5, 20);

for (let n of range) {
        console.log(n);
}
```

Generátory

```
function* range(from, to) {
    for (let i = from; i <= to; i++) {
        yield i;
    }
}


let data = range(30, 40);

while (true) {
    let item = data.next();
    if (item.done) {
        break;
    }
    console.log(item.value);
}
```

TODO

Upravte kolekci geometrických tvarů, tak aby používala iterátory.

Promise

- Příslib ..... dat, nebo toho, že něco nastane…
- Asynchronní programování.
- Podobnost s Task v C#.
- Nemá nic společného s paralelním programováním.
- „Alternativa“ k callbacku.

Promise

- Promise je objekt, který má hodnotu a stav (interně).
- Stavy:
  - Pending – zatím nebyla dokončena.
  - Fulfilled – úspěšně dokončena.
  - Rejected – dokončena s chybou.
- Mezi stavy Fulfilled a Rejected nelze přecházet.
- Ze stavů Fulfilled a Rejected nelze opětovně přejít do stavu Pending.
- Promise má hodnotu pouze pokud je ve stavu Fulfilled, nebo Rejected.

Promise - vytvoření

- Způsoby vytvoření:
  - new Promise(function(resolve, reject) { …. });
    - Předaná funkce se ihned spustí.
    - resolve – callback, který má být zavolán v případě úspěchu.
    - reject – callback, který má být zavolán v případě neúspěchu.
    - V rámci volání metod resolve a reject lze předat data (jako parametr volání).
  - Promise.resolve([value]);
  - Promise.reject([value]);

Promise - zpracování

- Pomocí metod:
  - .then(function(data){ … });
  - .catch(function(error){ … });

- Metoda „then“ je volána v případě zavolání „resolve“. Parametr „data“ obsahuje data předaná metodě „resolve“.
- Metoda „catch“ je volána v případě neúspěchu (volání reject), nebo chybě (jakékoliv neodchycené chybě). Parametr „error“ obsahuje informace o chybě (data předaná metodě „reject“).

Promise

```
let firstPromise = new Promise(function (resolve, reject) {

    setTimeout(function () {
        if (Math.random() < 0.5) {
                resolve();
        } else {
                reject();
        }
    }, 2000);

});

firstPromise
    .then(function () {
        console.log('resolved');
    })
    .catch(function () {
        console.log('rejected');
    });
```

Promise - then(...)

- Volání then vždy vrací novou Promise.
- Volání then lze zřetězit.

```
Promise.resolve(2)
    .then(function (data) {
        return data * 2;
    })
    .then(function (data) {
        return new Promise(function (resolve) {
            setTimeout(function () {
                resolve(data * 10);
            }, 1000);
        });
    })
    .then(function (data) {
        console.log(data);
    });
```

Promise - then

- Vícenásobné volání „then“ pouze „znovupoužívá“ hodnotu z promisy.

```
let promise = new Promise(function (resolve) {
    // tato metoda se zavolá jen 1x
    setTimeout(function () {
        resolve(10);
    }, 1000);
});

promise.then(function (data) {
        console.log(data); // 10
});
promise.then(function (data) {
        console.log(data); // 10
});
setTimeout(function () {
    promise.then(function (data) {
        console.log(data); // 10
    });
}, 2000);
```

```
let promise = new Promise(function (resolve) {
    setTimeout(function () {
        reject();
    }, 1000);
});

// zpracování
promise
    .then(function (data) {
        …
    })
    .catch(function () {
        …
    });

// tento zápis není ekvivalentní k předchozímu
promise.then(function (data) {
        …
});
promise.catch(function () {
        …
});
```

Spojení více Promise - čekání

- Promise.all([promise1, promise2, ….]);

```
let promises = [
    Promise.resolve(10),
    Promise.resolve('Jan'),
    Promise.resolve({ name: 'Pepa' })
];

Promise.all(promises)
    .then(function (data) {
        console.log(data[0]); // 10
        console.log(data[1]); // Jan
        console.log(data[2]); // { name: 'Pepa' }
    });
```

TODO

Implementujte vlastní Promise.

Proxy

- Objekt pro definici vlastní logiky pro základní operace s objektem.
- Proxy se na venek tváří, jako by byla objekt pro nějž je definována.
- let myProxy = new Proxy(target, handler);
  - target – objekt pro který se proxy vytváří
  - handler – objekt na které je definováno chování dané proxy (základní operace s objektem).

```
function User() { }
User.prototype.name = 'Jan';
User.prototype.age = 30;

let obj = new User();

let handler = {
     get: function (target, prop, receiver) {
          if (prop === 'name') {
                    return target.name.toUpperCase();
          }
          return target[prop];
     },
     set: function (target, prop, value, receiver) {
          if (prop === 'age') {
                    target[prop] = value * 2;
          } else {
                    target[prop] = value;
          }
     }
};

let proxy = new Proxy(obj, handler);

console.log(proxy.name);
proxy.age = 40;

console.log(proxy instanceof User);
```

Proxy – metody pro handler

- getPrototypeOf(target)
- setPrototypeOf(target, prototype)
- isExtensible(target)
- preventExtensions(target)
- getOwnPropertyDescriptor(target, property)
- defineProperty(target, property, descriptor)
- has(target, property) – in operátor – if("key" in obj) { }
- get(target, property, receiver) – receiver je proxy samotné, nebo objekt, který z něj dědí
- set(target, property, value, receiver) – receiver je proxy samotné, nebo objekt, který z něj dědí
- deleteProperty(target, prototype)
- ownKeys(target)
- apply(target, thisArg, argumentsList) – volání funkce
- construct(target, argumentsList, newTarget) – operátor new

TODO

Vyzkoušejte si logování interakce s objektem – logujte do konzole kdykoliv je volána nějaká funkce, přečtena, nebo nastavena nějaká vlastnost, atd..

Map/Set & WeakMap/WeakSet

- Set ≈ HashSet v C#.
- Map ≈ Dictionary v C#.
- WeakSet a WeakMap
  - Mají oproti Set a Map omezenou sadu instrukcí.
  - Mohou obsahovat jen objekty.
  - Drží hodnoty pouze po dobu existence daných objektů.

Set & Map

```
let mySet = new Set();
mySet.add("b");
mySet.add("a");
mySet.add("c");
mySet.add("a");

console.log(mySet);
```

```
let myMap = new Map();
myMap.set("b", 10);
myMap.set("a", 20);
myMap.set("c", 30);
myMap.set("a", 50);

console.log(myMap);
console.log(myMap.get('a'));
```

Set

- Vlastnosti a metody
  - size
  - has(value)
  - add(value)
  - delete(value)
  - [Symbol.iterator]()
  - forEach(f)
  - clear()
  - keys(), values(), entries()

Map

- Vlastnosti a metody
  - size
  - has(key)
  - get(key)
  - get(key, value)
  - delete(key)
  - [Symbol.iterator]()
  - forEach(f)
  - clear()
  - keys(), values(), entries()

WeakMap/WeakSet

- Můžou obsahovat jen objekty.
- **Nebrání GC odstranění objektu z paměti!**

- Metody WeakMap:
  - .has(), .get(), .set() a .delete()

- Metody WeakSet:
  - .has(), .add() a .delete()

Typová pole

- Pro práci s binárními daty.
- ArrayBuffer – základní struktura pro typová pole – pole bytů.
- Pohledy:
  - Int8Array
  - Uint8Array
  - Uint8ClampedArray
  - Int16Array
  - Uint16Array
  - Int32Array
  - Uint32Array
  - Float32Array
  - Float64Array

Typová pole

```
let data = new ArrayBuffer(8);
let ints = new Int32Array(data);

ints[0] = 20;
ints[1] = 58466;

let bytes = new Int8Array(data);

console.log(bytes);
```

Typová pole

- Vypíšeme 2 int-y jako pole bytů.

```
let data = new ArrayBuffer(8);
let ints = new Int32Array(data);

ints[0] = 20;
ints[1] = 58466;

let bytes = new Int8Array(data);


// [20, 0, 0, 0, 98, -28, 0, 0]
console.log(bytes);
```

- Object.assign
- Array.prototype.find
- Array.prototoype.findIndex
- String.prototype.repeat
- String.prototype.startsWith
- String.prototype.endsWith
- String.prototype.includes

- Number.isNaN
- Number.isFinite
- Number. isSafeInteger
- Number.EPSILON
- Math.trunc
- Math.sign

ES2016

ES2016 přidává jen 2 novinky

- Exponencionální operátor **
  - 2**3 == Math.pow(2, 3)

- Array.prototype.includes()
  - [1, 2, 6, 10].includes(6)

ES2017

ES2017 - novinky

- Object.values()
  - Seznam hodnot vlastností, tak jak by jimi procházel cyklus for … in.
- Object.entries()
  - Seznam párů [„klíč“, „hodnota“], tak jak by jimi procházel cyklus for … in.
- Object.getOwnPropertyDescriptor()
- String.prototype.padEnd()
- String.prototype.padStart()
- async/await

async / await

- JavaScript umí asynchronní programování již od jeho vzniku.
- Async/await přibyl pouze pro jeho zjednodušení.
- Jsou pouze „syntactical sugar“.
- Fungují obdobně jako v C#.

async

- Asynchronní funkce musí být označena jako „async“.
- Nemusí nic vracet a nemusí obsahuvat jiná asynchronní volání jako v C#.
- Volání funkce vrací Promise.

```
async function FirstAsyncFn() {
        // ....
}
```

await

- Zahájí čekání na dokončení asynchronního kódu.
- Aplikuje se na libovolnou Promise.
- Může být použit jen uvnitř funkce označené jako async.

```
var myPromise = new Promise((resolve) => {
        resolve(10);
});

async function FirstAsyncFn() {
        return 20;
}

async function SecondAsyncFn() {
    var a = await FirstAsyncFn();
    var b = await myPromise;
    return a + b;
}
```

TODO

Vytvořte seznam filmů na základě dat z:

http://janjanousek.cz/autocont/movies.json
http://janjanousek.cz/autocont/directors.json

Pro stažení dat použijte metodu „fetch“, která vrací Promise:

fetch('...url...')

Využijte prvků moderního JS (Promise, async/await, iteátory, třídy, moduly, nové metody polí, atd...).

Polyfills

Polyfills?

- Starší prohlížeče nepodporují prvky moderního JavaScriptu.
- Polyfills umožňují používat moderní kód i ve starších prohlížečích.
- Polyfill je kus JS, který implementuje určitou standardní funkcionalitu z moderního JS, pro použití ve starších prohlížečích.
- Příklady:
  - Promise
  - Map, Set
  - Nové metody pro práci s poli, objekty, atd..
- Ne vše jde implementovat!
  - Třídy, await/async, for… of, iterátory, atd..

Polyfills

- core-js: https://github.com/zloirock/core-js
- es5-shim: https://github.com/es-shims/es5-shim
- es5-shim: https://github.com/paulmillr/es6-shim/
- es6-promise: https://github.com/stefanpenner/es6-promise

- HTML5 promises: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills