Raise your hand if...
JavaScript and Python developers
How it feels to learn JavaScript in 2016

No JavaScript frameworks were created during the writing of this article.

https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f
WHAT IS THIS?

THAT'S THE WHOLE JOKE: IT'S ABOUT JAVASCRIPT CONTEXT
Overview

• JavaScript history and versions

• Basics of the language

• JavaScript ecosystem

• How to make sense of it all?
Thinking about ideas for talks/tutorials at @djangocon. Wondering if Django + Angular would still be interesting? And whether that's better as a tutorial or a talk?

Hmm, what might be some other talk or tutorial ideas?

---

@ed_rivas

How about "JavaScript for Python developers"? Bet it would be valuable to those diving into frontend development or just JS in general.

11:52 PM - 19 Apr 2018
Overview

• JavaScript history and versions
  • Basics of the language
  • Different tools
  • How to make sense of it all?
<table>
<thead>
<tr>
<th>Edition</th>
<th>Date published</th>
<th>Changes from prior edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>June 1997</td>
<td>First edition</td>
</tr>
<tr>
<td>2</td>
<td>June 1998</td>
<td>Editorial changes to keep the specification fully aligned with ISO/IEC 16262 International standard</td>
</tr>
<tr>
<td>3</td>
<td>December 1999</td>
<td>Added <strong>regular expressions</strong>, better string handling, new control statements, try/catch exception handling, tighter definition of errors, formatting for numeric output and other enhancements</td>
</tr>
<tr>
<td>4</td>
<td>Abandoned</td>
<td>Fourth Edition was abandoned, due to political differences concerning language complexity. Many features proposed for the Fourth Edition have been completely dropped; some are proposed for ECMAScript Harmony.</td>
</tr>
<tr>
<td>5</td>
<td>December 2009</td>
<td>Adds &quot;strict mode,&quot; a subset intended to provide more thorough error checking and avoid error-prone constructs. Clarifies many ambiguities in the 3rd edition specification, and accommodates behaviour of real-world implementations that differed consistently from that specification. Adds some new features, such as getters and setters, library support for <strong>JSON</strong>, and more complete <strong>reflection</strong> on object properties.</td>
</tr>
<tr>
<td>5.1</td>
<td>June 2011</td>
<td>This edition 5.1 of the ECMAScript standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.</td>
</tr>
<tr>
<td>6</td>
<td>June 2015[^10]</td>
<td>The sixth edition, initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015 (ES2015)[^10], adds significant new syntax for writing complex applications, including classes and modules, but defines them semantically in the same terms as ECMAScript 5 strict mode. Other new features include iterators and <code>for...of</code> loops. <strong>Python</strong>-style generators and generator expressions, arrow functions, binary data, typed arrays, collections (maps, sets and weak maps), <strong>promises</strong>, number and math enhancements, <strong>reflection</strong>, and proxies (metaprogramming for virtual objects and wrappers). As the first &quot;ECMAScript Harmony&quot; specification, it is also known as &quot;ES6 Harmony.&quot;</td>
</tr>
<tr>
<td>8</td>
<td>June 2017[^8]</td>
<td>ECMAScript 2017 (ES2017), the eighth edition, includes features for concurrency and <strong>atomics</strong>, syntactic integration with promises (async/await).[^2][^8]</td>
</tr>
</tbody>
</table>
Overview

- JavaScript history and versions
- Basics of the language
- Different tools
- How to make sense of it all?
let myName = 'EuroPython 2018';
function sayHi(name) {
    console.log(`Hey there, ${name}`);
}
sayHi(myName); // 'Hey there, EuroPython 2018';

let someArray = [1, 2, 5, 10];
let newArray = [];

for (let el of someArray) {
    if (el > 2) {
        newArray.push(el);
    } else {
        console.log('Nope!');
    }
}
// 'Nope!'
// 'Nope!'
class Hero {
  constructor(name, superPower) {
    this.name = name;
    this.superPower = superPower;
  }

  superPower() {
    console.log('I can count really fast!');
    let count = 0;
    while (count < 1000) {
      count++;
    }
    return count;
  }
}

let superMan = new Hero('SuperMan');

superMan.superPower();
// 'I can count really fast!'
// 1001
let x = 1;  // x is a number
x = 'Hi!';  // x is now a string
x = () => { return 1; };  // x is now a function
## Syntax

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>1 + '2'</code></td>
<td>&quot;12&quot;</td>
</tr>
<tr>
<td>&quot;12&quot;</td>
<td>&quot;12&quot;</td>
</tr>
<tr>
<td><code>'1' + 2</code></td>
<td>10</td>
</tr>
<tr>
<td><code>'1' + 2 - 2</code></td>
<td>10</td>
</tr>
</tbody>
</table>

*Image: Welcome to JavaScript – Where the objects are made up and the types don’t matter.*
Variables

```javascript
var x = 1;
let name = 'John';
const someConstant = 45;
```
Variable hoisting

```javascript
var x = 1;

// Some other code

var name = 'John';
```

```javascript
var x;
var name;
x = 1;

// Some other code

name = 'John';
```
var txt = ["a","b","c"];

for (var i = 0; i < 3; ++i) {
    var msg = txt[i];
    setTimeout(function() { alert(msg); }, i*1000);
}

// Alerts 'c', 'c', 'c'
Data Types

- Boolean
  
  ```
  let a = true;
  let b = false;
  ```

- String
  
  ```
  let name = 'John';
  name.length; // 4
  ```

- Number
  
  ```
  let num = -124.56;
  num = 10;
  ```

- Null
  
  ```
  let empty = null;
  ```

- Undefined
  
  ```
  let unknown = undefined;
  ```

- Object
  
  ```
  let something = {key: 'A value', anotherKey: name};
  let things = ['string', 2, (x, y) => { return x + y; }]
  ```
let bigObj = {
  key: 'Some string',
  add: function(x, y) { return x + y; },
  anotherObj: {
    name: 'I am a nested object'
  }
};
Objects are mutable

```python
define x as {a: 1}
define y as x

# Accessing the original value
x.a = 2

# Accessing the modified value
x.a
```

Result:
```
{x: 1, b: 2}
```
Operators

```java
if (!a && b) {
   // Some code
}
else if (a || b) {
   // Some code
}
```
Operators

== and !=

OR

=== and !==
Operators

"" == '0''" == false
"0 == ''" == true
"0 == '0''" == true

"false == 'false''" == false
"false == '0''" == true

"false == undefined" == false
"false == null" == false
"null == 'undefined''" == false
Functions

```javascript
let func = function(a, b) {
  return a + b;
};

let func = (a, b) => { return a + b; };
let func = (a, b) => a + b;
```
Functions

```javascript
function func(a = 1, b = 2) {
    return a + b;
}

func(5); // 7
```
Functions

```javascript
function func(a = 1, b = 2) {
  // Do some calculations
}

func(5); // undefined
```
var pets = {
  names: ['Baron', 'Chief', 'Axel'],
  owner: 'Jason',
  description: function(){
    return this.names.map(function(pet){
      return `${this.owner} knows an awesome dog named ${pet}.`;
    });
  }
};

pets.description()

["undefined knows an awesome dog named Baron.", "undefined knows an awesome dog named Chief.", "undefined knows an awesome dog named Axel." ]
let pets = {
  names: ['Baron', 'Chief', 'Axel'],
  owner: 'Jason',
  description: function () {
    let that = this;
    return this.names.map(function (pet) {
      return `${that.owner} knows an awesome dog named ${pet}.`;
    });
  }
};
pets.description();
let pets = {
    names: ['Baron', 'Chief', 'Axel'],
    owner: 'Jason',
    description: function () {
        return this.names.map((pet) => {
            return `${this.owner} knows an awesome dog named ${pet}.`;
        });
    }
};
pets.description();

"Jason knows an awesome dog named Baron.", "Jason knows an awesome dog named Chief.", "Jason knows an awesome dog named Axel."
Classes

definitions:

Python:
```python
class Animal:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hi {}'.format(self.name))

class Dog(Animal):
    pass
dog = Dog('Billy')
dog.say_hi()
```

JavaScript:
```javascript
class Animal {
    constructor(name) {
        this.name = name;
    }
    sayHi() {
        console.log('Hi ${this.name}');
    }
}
class Dog extends Animal {
}
dog = new Dog('Billy');
dog.sayHi();
```
Modules
Template literals

```javascript
var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);
// "Fifteen is 15 and not 20."
```
Template literals

```javascript
let a = 5;
let b = 10;
console.log('Fifteen is ' + (a + b) + ' and\nnot ' + (2 * a + b) + '.');
// "Fifteen is 15 and
// not 20."
```
// getPages returns a Promise
let loadPageContents = getPages(someUrl).then((result) => {
    return doSomething(result);
}).catch((error) => {
    handleError(error);
});

// loadPageContents is a Promise
loadPageContents().then(() => {
    changeElementOnPage();
});
Overview

• JavaScript history and versions
• Basics of the language
• Different tools
• How to make sense of it all?
Bad Parts

- Global variables
- `==`
- `+`
- `scope`
function greeter(person: string) {
    return "Hello, " + person;
}

let user = [0, 1, 2];

document.body.innerHTML = greeter(user);
Overview

- JavaScript history and versions
- Basics of the language
- Different tools
- How to make sense of it all?
Different tools

- npm, bower, yarn
- Babel
- Webpack
- gulp, grunt
Different tools

• npm, bower, yarn
• Babel
• Webpack
• gulp, grunt
Different tools

- npm, bower, yarn
- Babel
- Webpack
- gulp, grunt
Different tools

- npm, bower, yarn
- Babel
- Webpack
- gulp, grunt
Different tools

- npm, bower, yarn
- Babel
- Webpack
- gulp, grunt
Overview

• JavaScript history and versions
• Basics of the language
• Different tools
• How to make sense of it all?
How to get started

- Start somewhere
- Prepare your codebase
- No need to learn and use everything at once
Thank you!

Questions?