Pure and functional JavaScript

Developer Conference 2013

Jakob Westhoff (@jakobwesthoff)

November 7th, 2013

Qafoo

passion for software quality

I am

Jakob Westhoff

- Senior PHP professional
- Senior JavaScript professional
- Open source enthusiast
- Regular speaker at (inter)national conferences
- Consultant, Trainer and Author

We are

Qafoo

passion for software quality

We are

Qafoo

passion for software quality

Helping people to create high quality web applications.

`http://qafoo.com`

We are

passion for software quality

Helping people to create high quality web applications.

`http://qafoo.com`

- Trainings, Workshops and Consulting

We are

Qafoo

passion for software quality

Helping people to create high quality web applications.

`http://qafoo.com`

- Trainings, Workshops and Consulting
- Twitter: `@qafoo`
- Mail: `contact@qafoo.com`

Functional Programming

Functional programming is a programming paradigm [...], that [...] avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state - i.e. pure mathematical functions.

– Wikipedia

Functional Programming

Functional programming is a programming paradigm [...], that [...] avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state - i.e. pure mathematical functions.

– Wikipedia

Functional Programming

Functional programming is a programming paradigm [...], that [...] avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state - i.e. pure mathematical functions.

– Wikipedia

passion for software quality

Functional Programming

Functional programming is a programming paradigm [...], that [...] avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state - i.e. pure mathematical functions.

– Wikipedia

Functional Programming

Functional programming is a programming paradigm [...], that [...] avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state - i.e. pure mathematical functions.

– Wikipedia

JavaScript Functions

A simple function

```
function increment(a) {
  alert(a+1);
}
```

A simple function

```
function increment(a) {
    alert(a+1);
}
```

- Function Declaration

A simple function

```
function increment(a) {
    alert(a+1);
}
```

- Function Declaration
- Most common way to create a function in JavaScript

Functions

First-Level Citizens

Functions are special

```
var increment = function(a) {
   alert(a+1);
}
```

Functions are special

```
var increment = function(a) {
  alert(a+1);
}
```

- Function Expression

Functions are special

```
function one() {
    function two() {
        function three() {
            // ...
        }
    }
}
```

- Functions can be nested

Higher-Order Functions

```
window.setTimeout(function() {
    alert("A␣second␣has␣passed");
}, 1000);
```

Higher-Order Functions

```
window.setTimeout(function() {
    alert("A second has passed");
}, 1000);
```

- Functions can take other functions as arguments

Higher-Order Functions

```
var makeGreeter = function(name) {
  return function() {
    alert("Hello␣" + name + "!");
  };
};
```

Higher-Order Functions

```
var makeGreeter = function(name) {
    return function() {
        alert("Hello␣" + name + "!");
    };
};
```

- Functions may return other functions

Higher-Order Functions

- Functions which consume or return other functions are called

Higher-Order Functions

What comes next?

Sideeffects

Sideeffects

```
var count = function(items) {
    var length = 0;
    while(items.pop()) {
        length += 1;
    }

    return length;
}

var items = [1,2,3];
alert(count(items)); // 3
```

passion for software quality

Sideeffects

```
var count = function(items) {
    var length = 0;
    while(items.pop()) {
        length += 1;
    }

    return length;
}

var items = [1,2,3];
alert(count(items)); // 3
```

- Are there any problems with this implementation?

Sideeffects

```
var count = function(items) {
    var length = 0;
    while(items.pop()) {
        length += 1;
    }

    return length;
}

var items = [1,2,3];
alert(count(items)); // 3
```

- Modification of state

Sideeffects

```
var increment = function(a) {
  alert(a+1);
}
```

Sideeffects

```
var increment = function(a) {
    alert(a+1);
}
```

- Interaction with the outside world

Pure Functions

```
var increment = function(a) {
  return a+1;
}
```

Pure Functions

```
var increment = function(a) {
  return a+1;
}
```

- No sideeffects

Pure Functions

```
var increment = function(a) {
  return a+1;
}
```

- No sideeffects
- Easier to maintain, easier to test

Pure Functions

- Functions which are sideeffect free are called

Pure Functions

Pure Functions

- No program can consist of Pure Functions only

Pure Functions

- No program can consist of Pure Functions only
- It would simply do nothing

Pure Functions

- No program can consist of Pure Functions only
- It would simply do nothing
- Every program needs to have at least one impure function

What comes next?

Functional Built-Ins

passion for software quality

Functional Loops

- Loops (`for`, `while`, `...`) are quite common in every program

Functional Loops

- Loops (`for`, `while`, `...`) are quite common in every program
- Loops are mostly used to
  - Cause Sideeffects on every element of a set

Functional Loops

- Loops (`for`, `while`, `...`) are quite common in every program
- Loops are mostly used to
  - Cause Sideeffects on every element of a set
  - Transform or Extract data from a set

Functional Loops

- Loops (`for`, `while`, `...`) are quite common in every program
- Loops are mostly used to
  - Cause Sideeffects on every element of a set
  - Transform or Extract data from a set
  - Aggregate or Accumulate data from a set

Looping

```
var sessions = ...;

var i,len;
for(i=0, len=sessions.length; i<len; i++) {
  doSomething(sessions[i]);
}
```

passion for software quality

Looping

```
var sessions = ...;

var i,len;
for(i=0, len=sessions.length; i<len; i++) {
  doSomething(sessions[i]);
}
```

- Loop through data

Looping

```
var sessions = ...;

var i,len;
for(i=0, len=sessions.length; i<len; i++) {
  doSomething(sessions[i]);
}
```

- Loop through data
- Cause some sideeffect on each data entry

Array.prototype.forEach

```
var sessions = ...;

sessions.forEach(function(session) {
  doSomething(session);
});
```

Data extraction

```
var titles = [];

var i, len;
for(i=0, len=sessions.length; i<len; i++) {
  titles.push(sessions[i].title);
}
```

Qafoo
passion for software quality

Data extraction

```
var titles = [];

var i, len;
for(i=0, len=sessions.length; i<len; i++) {
  titles.push(sessions[i].title);
}
```

- Extract certain bits of data from a bigger structure

Array.prototype.map

```
var titles = sessions.map(function(session) {
    return session.title;
});
```

passion for software quality

Array.prototype.map

```
var titles = sessions.map(function(session) {
    return session.title;
});
```

- This kind of property extraction is needed quite often

Array.prototype.map

```
var titles = sessions.map(function(session) {
    return session.title;
});
```

- This kind of property extraction is needed quite often
- Let’s extract it to its own method

extract

```
var extract = function(property) {
    return function(object) {
        return object[property];
    };
};
```

extract

```
var extract = function(property) {
    return function(object) {
        return object[property];
    };
};
```

```
var titles = sessions.map(extract("title"));
```

passion for software quality

extract

```
var extract = function(property) {
    return function(object) {
        return object[property];
    };
};
```

```
var titles = sessions.map(extract("title"));
```

extract

```
var extract = function(property) {
    return function(object) {
        return object[property];
    };
};
```

```
var titles = sessions.map(extract("title"));
```

extract

```
var extract = function(property) {
    return function(object) {
        return object[property];
    };
};
```

```
var titles = sessions.map(extract("title"));
```

Data accumulation

```
var sessionList = "";

var i, len;
for(i=0, len=sessions.length; i<len; i++) {
  sessionList += "<li>" + sessions[i].title + "</
      li>";
}
```

Data accumulation

```
var sessionList = "";

var i, len;
for(i=0, len=sessions.length; i<len; i++) {
  sessionList += "<li>" + sessions[i].title + "</
      li>";
}
```

- Accumulate information from given data structure

Data accumulation

```
var sessionList = "";

var i, len;
for(i=0, len=sessions.length; i<len; i++) {
  sessionList += "<li>" + sessions[i].title + "</
      li>";
}
```

- Accumulate information from given data structure
- Two things are actually done here:
  1. Extract each sessions title
  2. Accumulate result into an HTML-List

passion for software quality

Array.prototype.reduce

```
var sessionList = sessions
  .map(function(session) {
    return session.title;
  })
  .reduce(function(accumulation, next) {
    return accumulation + "<li>" + next + "</li>"
  }, "");
```

Array.prototype.reduce

```
var sessionList = sessions
    .map(function(session) {
      return session.title;
    })
    .reduce(function(accumulation, next) {
      return accumulation + "<li>" + next + "</li>"
    }, "");
```

passion for software quality

Array.prototype.reduce

```
var sessionList = sessions
  .map(extract("title"))
  .reduce(function(accumulation, next) {
    return accumulation + "<li>" + next + "</li>"
  }, "");
```

passion for software quality

Array.prototype.reduce

```
var sessionList = sessions
  .map(extract("title"))
  .reduce(function(accumulation, next) {
    return accumulation + "<li>" + next + "</li>"
  }, "");
```

Array.prototype.reduce

```
var wrapIn = function(element) {
  return function(input) {
    return "<" + element + ">" + input + "</" +
        element + ">";
  };
};

var sessionList = sessions
  .map(extract("title"))
  .map(wrapIn("li"))
  .reduce(function(accumulation, next) {
    return accumulation + next
  }, "");
```

passion for software quality

Array.prototype.reduce

```
var wrapIn = function(element) {
  return function(input) {
    return "<" + element + ">" + input + "</" +
        element + ">";
  };
};

var sessionList = sessions
  .map(extract("title"))
  .map(wrapIn("li"))
  .reduce(function(accumulation, next) {
    return accumulation + next
  }, "");
```

Array.prototype.reduce

```
var wrapIn = function(element) {
  return function(input) {
    return "<" + element + ">" + input + "</" +
        element + ">";
  };
};

var sessionList = sessions
  .map(extract("title"))
  .map(wrapIn("li"))
  .reduce(function(accumulation, next) {
    return accumulation + next
  }, "");
```

passion for software quality

Array.prototype.reduce

```
var concatenate = function(accumulation, next) {
  return accumulation + next;
};

var sessionList = sessions
  .map(extract("title"))
  .map(wrapIn("li"))
  .reduce(concatenate, "");
```

passion for software quality

Array.prototype.reduce

```
var concatenate = function(accumulation, next) {
  return accumulation + next;
};

var sessionList = sessions
  .map(extract("title"))
  .map(wrapIn("li"))
  .reduce(concatenate, "");
```

Array.prototype.reduce

```
var sessionList = sessions
    .map(extract("title"))
    .map(wrapIn("li"))
    .reduce(concatenate, "");
```

Array.prototype.reduce

```
var sessionList = sessions
  .map(extract("title"))
  .map(wrapIn("li"))
  .reduce(concatenate, "");
```

vs.

```
var sessionList = "";

var i, len;
for(i=0, len=sessions.length; i<len; i++) {
  sessionList += "<li>" + sessions[i].title + "</
      li>";
}
```

passion for software quality

A little bit more real world please!

A little bit more real world

1. Extract `title`, `speaker` and description for display

A little bit more real world

1. Extract `title`, `speaker` and description for display
2. Preprocess the data according to certain rules
   - Uppercase names

A little bit more real world

1. Extract `title`, `speaker` and description for display
2. Preprocess the data according to certain rules
   - Uppercase names
   - Highlight certain buzzwords

A little bit more real world

1. Extract `title`, `speaker` and description for display
2. Preprocess the data according to certain rules
   - Uppercase names
   - Highlight certain buzzwords
   - Limit descriptions to a maximal length

A little bit more real world

1. Extract `title`, `speaker` and description for display
2. Preprocess the data according to certain rules
    - Uppercase names
    - Highlight certain buzzwords
    - Limit descriptions to a maximal length
3. Finally accumulate everything as HTML

A little bit more real world

```
var titles = sessions
  .map(extract("title"))
  .map(wrapIn("h2"));
```

passion for software quality

A little bit more real world

```
var titles = sessions
  .map(extract("title"))
  .map(wrapIn("h2"));
```

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

passion for software quality

A little bit more real world

```
var titles = sessions
  .map(extract("title"))
  .map(wrapIn("h2"));
```

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

```
var descriptions = sessions
  .map(extract("description"))
  .map(ellipsis(160))
  .map(highlight("JavaScript", "HTML5"))
  .map(wrapIn("p"));
```

Qafoo
passion for software quality

A little bit more real world

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

passion for software quality

A little bit more real world

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

Qafoo
passion for software quality

A little bit more real world

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

```
var uppercaseEveryFirst = function() {
  return function(input) {
    return input
      .split("␣")
      .map(uppercaseFirst())
      .join("␣");
  }
}
```

Qafoo
passion for software quality

A little bit more real world

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

```
var uppercaseEveryFirst = function() {
  return function(input) {
    return input
      .split("␣")
      .map(uppercaseFirst())
      .join("␣");
  }
}
```

A little bit more real world

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

```
var uppercaseEveryFirst = function() {
  return function(input) {
    return input
      .split("␣")
      .map(uppercaseFirst())
      .join("␣");
    }
}
```

```
var uppercaseFirst = function() {
  return function(input) {
    return input.charAt(0).toUpperCase()
           + input.substring(1);
    }
}
```

Qafoo
passion for software quality

A little bit more real world

```
var speakers = sessions
    .map(extract("speaker"))
    .map(uppercaseEveryFirst())
    .map(prefix("Speaker: "))
    .map(wrapIn("h3"));
```

A little bit more real world

```
var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));
```

```
var prefix = function(prefix) {
  return function(input) {
    return prefix + input;
  }
};
```

A little bit more real world

```
var descriptions = sessions
  .map(extract("description"))
  .map(ellipsis(160))
  .map(highlight("JavaScript", "HTML5"))
  .map(wrapIn("p"));
```

A little bit more real world

```
var descriptions = sessions
  .map(extract("description"))
  .map(ellipsis(160))
  .map(highlight("JavaScript", "HTML5"))
  .map(wrapIn("p"));
```

passion for software quality

A little bit more real world

```
var descriptions = sessions
  .map(extract("description"))
  .map(ellipsis(160))
  .map(highlight("JavaScript", "HTML5"))
  .map(wrapIn("p"));
```

```
var ellipsis = function(maxLength) {
  return function(input) {
  if (input.length <= maxLength) {
    return input;
  }

  return input.substring(0,maxLength-1) + "..."
  }
}
```

A little bit more real world

```
var descriptions = sessions
  .map(extract("description"))
  .map(ellipsis(160))
  .map(highlight("JavaScript", "HTML5"))
  .map(wrapIn("p"));
```

A little bit more real world

```
var descriptions = sessions
  .map(extract("description"))
  .map(ellipsis(160))
  .map(highlight("JavaScript", "HTML5"))
  .map(wrapIn("p"));
```

```
var highlight = function(/* args... */) {
  var args = Array.prototype.slice.call(arguments);
  return function(input) {
    args.forEach(function(replacement) {
      input = input.replace(
        new RegExp("\\b" + replacement + "\\b"),
        "<em>" + replacement + "</em>"
      );
    });

    return input;
  }
}
```

passion for software quality

A little bit more real world

```
var titles = ..., speakers = ..., descriptions = ...
```

A little bit more real world

```
var titles = ..., speakers = ..., descriptions = ...
```

```
var result = weave(titles, speakers, descriptions)
  .map(join())
  .map(wrapIn("div"))
  .reduce(concatenate());
```

A little bit more real world

```
var titles = ..., speakers = ..., descriptions = ...

var result = weave(titles, speakers, descriptions)
    .map(join())
    .map(wrapIn("div"))
    .reduce(concatenate());
```

A little bit more real world

```
var titles = ..., speakers = ..., descriptions = ...

var result = weave(titles, speakers, descriptions)
    .map(join())
    .map(wrapIn("div"))
    .reduce(concatenate());
```

```
[ 1,  2,  3,  4 ]
["a","b","c","d"]
["!","?","$","="]
------weave------
[
   [1,"a","!"],
   [2,"b","?"],
   [3,"c","$"],
   [4,"d","="]
]
```

Qafoo
passion for software quality

A little bit more real world

```
var titles = ..., speakers = ..., descriptions = ...
```

```
var result = weave(titles, speakers, descriptions)
    .map(join())
    .map(wrapIn("div"))
    .reduce(concatenate());
```

A little bit more real world

```
var titles = ..., speakers = ..., descriptions = ...
```

```
var result = weave(titles, speakers, descriptions)
  .map(join())
  .map(wrapIn("div"))
  .reduce(concatenate());
```

```
var join = function(delimiter) {
  return function(input) {
    return input.join(delimiter);
  }
}
```

passion for software quality

A little bit more real world

```
var titles = sessions
  .map(extract("title"))
  .map(wrapIn("h2"));

var speakers = sessions
  .map(extract("speaker"))
  .map(uppercaseEveryFirst())
  .map(prefix("Speaker:␣"))
  .map(wrapIn("h3"));

var descriptions = sessions
  .map(extract("description"))
  .map(ellipsis(160))
  .map(highlight("JavaScript", "HTML5"))
  .map(wrapIn("p"));

var result = weave(titles, speakers, descriptions)
  .map(join())
  .map(wrapIn("div"))
  .reduce(concatenate());
```

Games meet business

Speaker: Sven Tissot

Innovative Bedienoberflächen und kreative Anwendungen im professionellen Kontext – wie lässt sich dies unter einen Hut bringen? Vorgestellt wird die Realisierung einer modernen, mobilen App zur Visualisierung logistischer Kennzahlen auf Basis der Entwicklu…

Packing skills to be a DevOps and why that’s a good thing

Speaker: Ole Michaelis

Bei Jimdo wurde ich als Software Engineer eingestellt. Die Software ist in den letzten sechs Jahre gewachsen und sah genauso aus wie man es erwarten würde, wenn man hört dass ein System so alt ist. Keine wirklichen Abstraktionen, keine Services – kein moderne…

Pure and functional JavaScript

Speaker: Jakob Westhoff

JavaScript ist eine Sprache, die viele unterschiedliche Programmierparadigmen in sich vereint. Ob Sie funktional, objekt-orientiert, prozedural oder aspekt-orientiert Software entwickeln wollen, JavaScript bietet für alle Ansätze die nötige Flexibilität. Dies…

Produktiv- und Entwicklungsumgebung mit Puppet verwalten

Speaker: Hans-Christian Otto

Als ein wachsender Betreiber von Web-Applikationen müssen Sie mit einer wachsenden Menge an Servern umgehen. Einen neuen Datenbankserver aufzusetzen dauert Tage? Für das aufsetzen eines neuen Entwicklerarbeitsplatzes benötigen Sie einen Senior-Developer, der …

What comes next?

JavaScript is Asynchronous

Asynchronous functional handling

- JavaScript utilizes Evented-IO

Asynchronous functional handling

- JavaScript utilizes Evented-IO
- A lot of operations are asynchronous

Asynchronous functional handling

- JavaScript utilizes Evented-IO
- A lot of operations are asynchronous
- Especially those with the outside world

Fetching our `sessions` via XHR

- Our `sessions` data needs to be retrieved from the server

Fetching our `sessions` via XHR

- Our `sessions` data needs to be retrieved from the server
- It is devided into multiple chunks by day

Fetching our `sessions` via XHR

- Our `sessions` data needs to be retrieved from the server
- It is devided into multiple chunks by day

```
var urls = [
    "/conference/sessions/day1",
    "/conference/sessions/day2",
    "/conference/sessions/socialevent",
    "/conference/sessions/day3"
];
```

Fetching our `sessions` via XHR

```
var fetch = function(urls, done) {
    var receivedData = [];
    var i, len;

    for(i=0, len=urls.length; i<len; i++) {
        $.ajax({
            url: urls[i],
            success: function(data) {
                receivedData[i] = data;
                if(receivedData.length == urls.length) {
                    done(receivedData);
                }
            }
        })
    }
};
```

passion for software quality

Really?

It doesn’t even work!

Fetching our `sessions` via XHR

```
var fetch = function(urls, done) {
    var receivedData = [];
    var i, len;

    for(i=0, len=urls.length; i<len; i++) {
        $.ajax({
            url: urls[i],
            success: function(data) {
                receivedData[i] = data;
                if(receivedData.length == urls.length) {
                    done(receivedData);
                }
            }
        })
    }
};
```

Qafoo
passion for software quality

Fetching our `sessions` via XHR

```
var fetch = function(urls, done) {
    var receivedData = [];
    var i, len;

    for(i=0, len=urls.length; i<len; i++) {
        (function(i) {
            $.ajax({
                url: urls[i],
                success: function(data) {
                    receivedData[i] = data;
                    if(receivedData.length == urls.length) {
                        done(receivedData);
                    }
                }
            })
        })(i);
    }
};
```

passion for software quality

Btw: It still does not work!

Fetching our `sessions` via XHR

```
var fetch = function(urls, done) {
    var receivedData = [];
    var i, len;

    for(i=0, len=urls.length; i<len; i++) {
        (function(i) {
            $.ajax({
                url: urls[i],
                success: function(data) {
                    receivedData[i] = data;
                    if(receivedData.length == urls.length) {
                        done(receivedData);
                    }
                }
            })
        })(i);
    }
};
```

Qafoo
passion for software quality

Fetching our `sessions` via XHR

```
var fetch = function(urls, done) {
    var receivedData = [];
    var i, len;
    var receivedCount = 0;

    for(i=0, len=urls.length; i<len; i++) {
        (function(i) {
            $.ajax({
                url: urls[i],
                success: function(data) {
                    receivedData[i] = data;
                    receivedCount += 1;
                    if(receivedCount == urls.length) {
                        done(receivedData);
                    }
                }
            })
        })(i);
    }
};
```

Qafoo
passion for software quality

Fetching our sessions via XHR

```
var fetch = function(urls, done) {
    var receivedData = [];
    var i, len;
    var receivedCount = 0;

    for(i=0, len=urls.length; i<len; i++) {
        (function(i) {
            $.ajax({
                url: urls[i],
                success: function(data) {
                    receivedData[i] = data;
                    receivedCount += 1;
                    if(receivedCount == urls.length) {
                        done(receivedData);
                    }
                }
            })
        })(i);
    }
};
```

passion for software quality

Can’t we do better?

Asynchronous `map/reduce` to the rescue.

Asynchronous `map/reduce`

- `https://github.com/caolan/async`

Asynchronous `map/reduce`

- `https://github.com/caolan/async`
- Functional style asynchronous handling

Asynchronous `map/reduce`

- `https://github.com/caolan/async`
- Functional style asynchronous handling
- … and more :)

Asynchronous `map/reduce`

```
var fetch = funtion(urls, done) {
    async.map(urls, ajax, done);
};
```

Asynchronous `map/reduce`

```
var fetch = funtion(urls, done) {
    async.map(urls, ajax, done);
};
```

Asynchronous map/reduce

```
var fetch = funtion(urls, done) {
    async.map(urls, ajax, done);
};
```

```
var ajax = function(url, done) {
    $.ajax({
        url: url,
        success: done
    });
};
```

Qafoo
passion for software quality

What comes next?

Performance

Performance considerations

- ”I am writing high performance JS. I can’t use `forEach`, `map`, `reduce` and others.”

Performance considerations

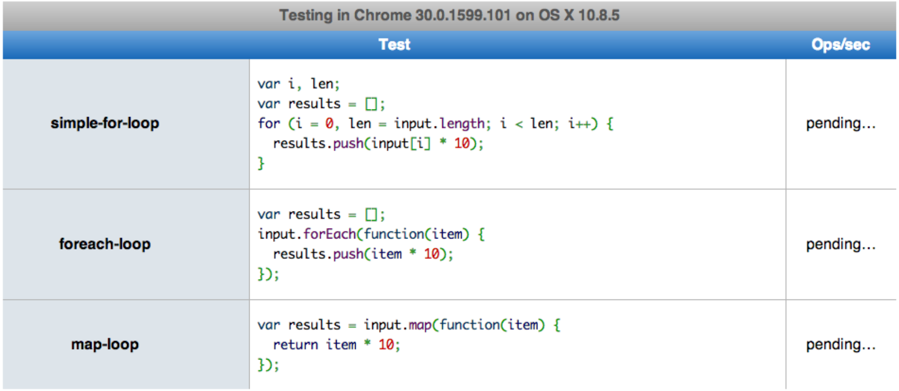
- ”I am writing high performance JS. I can’t use `forEach`, `map`, `reduce` and others.”
- Is this true?

Performance considerations

- ”I am writing high performance JS. I can’t use `forEach`, `map`, `reduce` and others.”
- Is this true?
- Yes. Due to the higher amount of function calls those are slower.

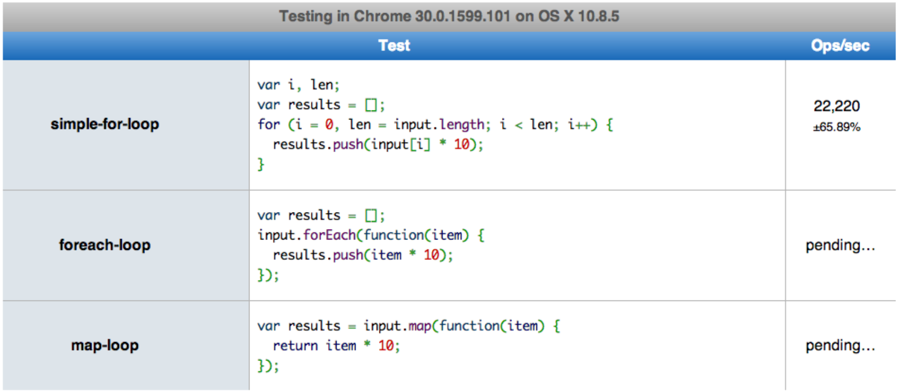
Performance considerations

- ”I am writing high performance JS. I can’t use `forEach`, `map`, `reduce` and others.”
- Is this true?
- Yes. Due to the higher amount of function calls those are slower.
- But is it a noticeable problem?



Testing in Chrome 30.0.1599.101 on OS X 10.8.5

	Test	Ops/sec
simple-for-loop	`var i, len;` `var results = [];` `for (i = 0, len = input.length; i < len; i++) {` `  results.push(input[i] * 10);` `}`	pending…
foreach-loop	`var results = [];` `input.forEach(function(item) {` `  results.push(item * 10);` `});`	pending…
map-loop	`var results = input.map(function(item) {` `  return item * 10;` `});`	pending…



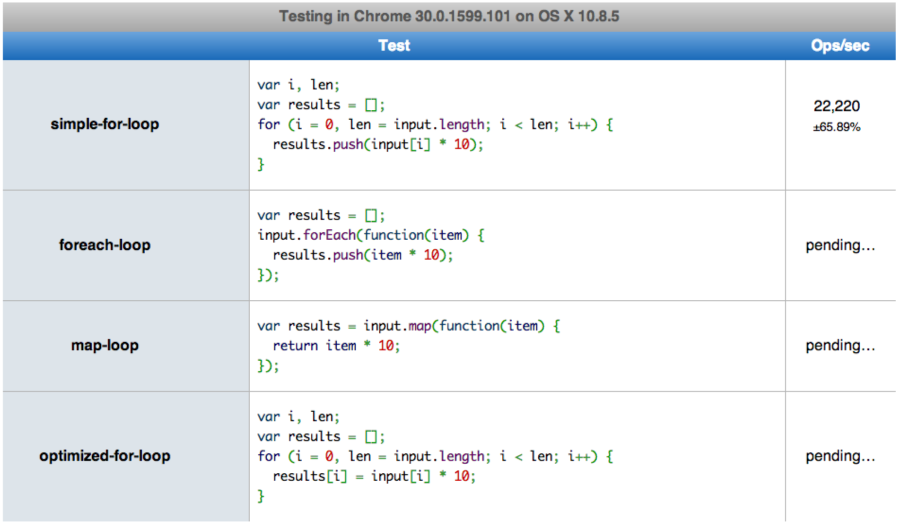
Testing in Chrome 30.0.1599.101 on OS X 10.8.5

	Test	Ops/sec

simple-for-loop	`var i, len;`
	`var results = [];`
	`for (i = 0, len = input.length; i < len; i++) {`
	`  results.push(input[i] * 10);`
	`}`
		22,220
		±65.89%

foreach-loop	`var results = [];`
	`input.forEach(function(item) {`
	`  results.push(item * 10);`
	`});`
		pending…

map-loop	`var results = input.map(function(item) {`
	`  return item * 10;`
	`});`
		pending…



Testing in Chrome 30.0.1599.101 on OS X 10.8.5

Test		Ops/sec
simple-for-loop	`var i, len; var results = []; for (i = 0, len = input.length; i < len; i++) { results.push(input[i] * 10); }`	22,220 ±65.89%
foreach-loop	`var results = []; input.forEach(function(item) { results.push(item * 10); });`	pending…
map-loop	`var results = input.map(function(item) { return item * 10; });`	pending…
optimized-for-loop	`var i, len; var results = []; for (i = 0, len = input.length; i < len; i++) { results[i] = input[i] * 10; }`	pending…

simple-for-loop

```
var i, len;
var results = [];
for (i = 0, len = input.length; i < len; i++) {
  results.push(input[i] * 10);
}
```

foreach-loop

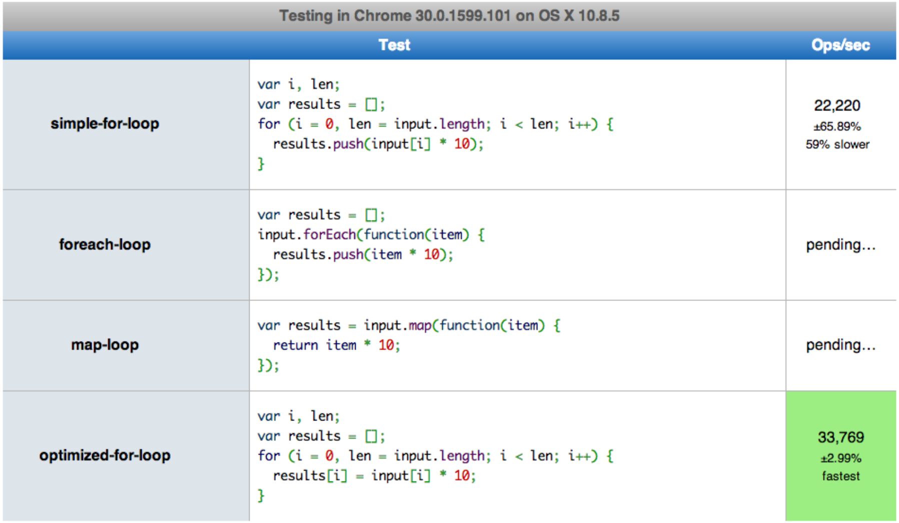
```
var results = [];
input.forEach(function(item) {
  results.push(item * 10);
});
```

map-loop

```
var results = input.map(function(item) {
  return item * 10;
});
```

optimized-for-loop

```
var i, len;
var results = [];
for (i = 0, len = input.length; i < len; i++) {
  results[i] = input[i] * 10;
}
```



Testing in Chrome 30.0.1599.101 on OS X 10.8.5

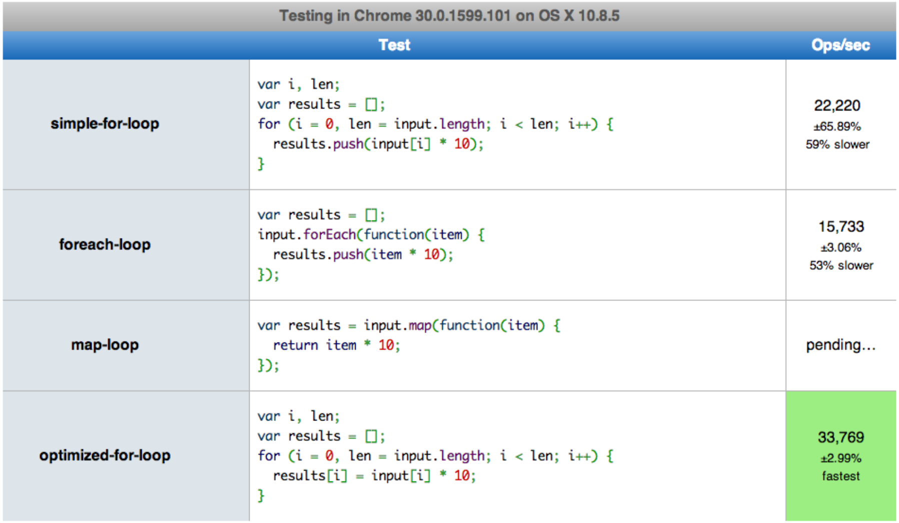
	Test	Ops/sec
simple-for-loop	`var i, len; var results = []; for (i = 0, len = input.length; i < len; i++) { results.push(input[i] * 10); }`	22,220 ±65.89% 59% slower
foreach-loop	`var results = []; input.forEach(function(item) { results.push(item * 10); });`	pending…
map-loop	`var results = input.map(function(item) { return item * 10; });`	pending…
optimized-for-loop	`var i, len; var results = []; for (i = 0, len = input.length; i < len; i++) { results[i] = input[i] * 10; }`	33,769 ±2.99% fastest

```
var i, len;
var results = [];
for (i = 0, len = input.length; i < len; i++) {
  results.push(input[i] * 10);
}
```

```
var results = [];
input.forEach(function(item) {
  results.push(item * 10);
});
```

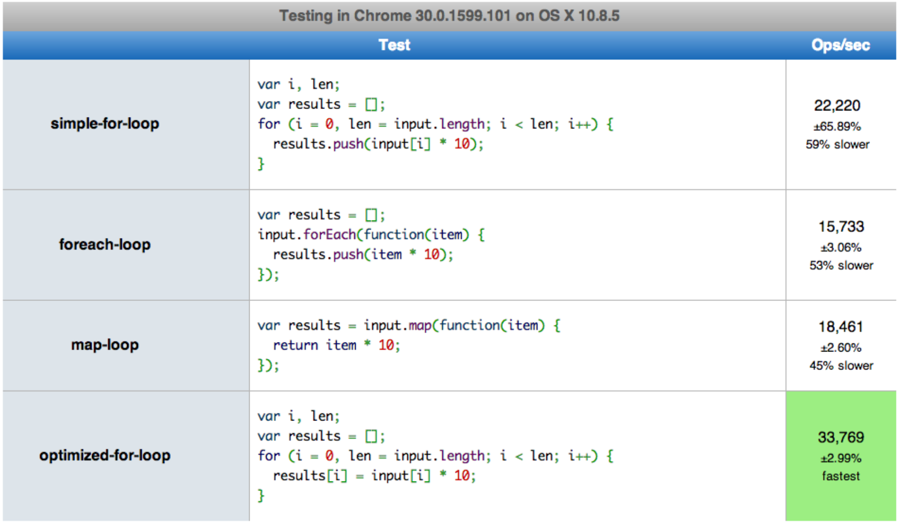
```
var results = input.map(function(item) {
  return item * 10;
});
```

```
var i, len;
var results = [];
for (i = 0, len = input.length; i < len; i++) {
  results[i] = input[i] * 10;
}
```



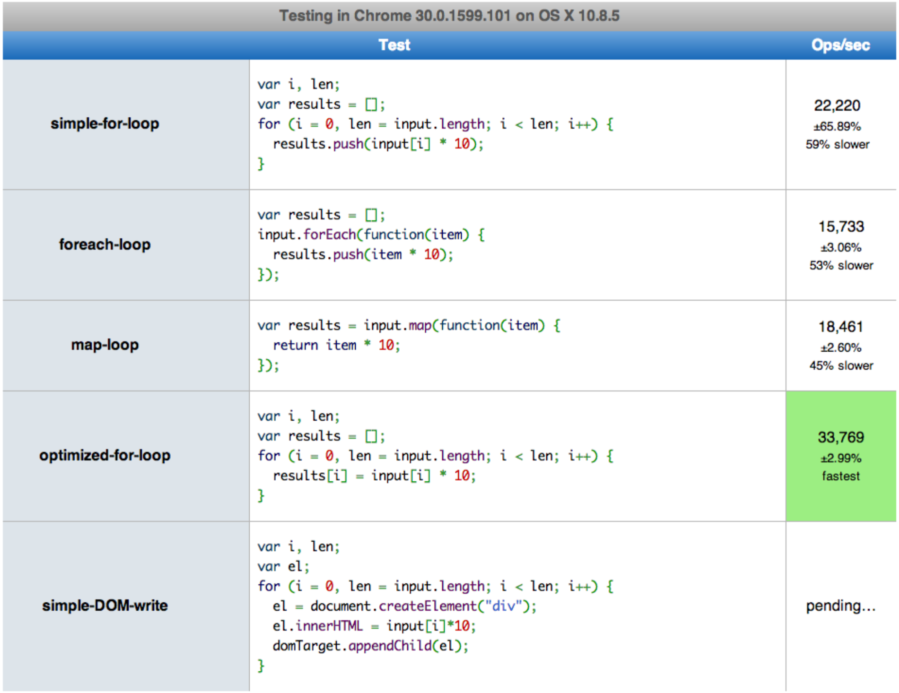
Testing in Chrome 30.0.1599.101 on OS X 10.8.5

	Test	Ops/sec
simple-for-loop	`var i, len;` `var results = [];` `for (i = 0, len = input.length; i < len; i++) {` `results.push(input[i] * 10);` `}`	22,220 ±65.89% 59% slower
foreach-loop	`var results = [];` `input.forEach(function(item) {` `results.push(item * 10);` `});`	15,733 ±3.06% 53% slower
map-loop	`var results = input.map(function(item) {` `return item * 10;` `});`	pending…
optimized-for-loop	`var i, len;` `var results = [];` `for (i = 0, len = input.length; i < len; i++) {` `results[i] = input[i] * 10;` `}`	33,769 ±2.99% fastest



Testing in Chrome 30.0.1599.101 on OS X 10.8.5

	Test	Ops/sec
simple-for-loop	`var i, len;` `var results = [];` `for (i = 0, len = input.length; i < len; i++) {` `results.push(input[i] * 10);` `}`	22,220 ±65.89% 59% slower
foreach-loop	`var results = [];` `input.forEach(function(item) {` `results.push(item * 10);` `});`	15,733 ±3.06% 53% slower
map-loop	`var results = input.map(function(item) {` `return item * 10;` `});`	18,461 ±2.60% 45% slower
optimized-for-loop	`var i, len;` `var results = [];` `for (i = 0, len = input.length; i < len; i++) {` `results[i] = input[i] * 10;` `}`	33,769 ±2.99% fastest



Testing in Chrome 30.0.1599.101 on OS X 10.8.5

	Test	Ops/sec

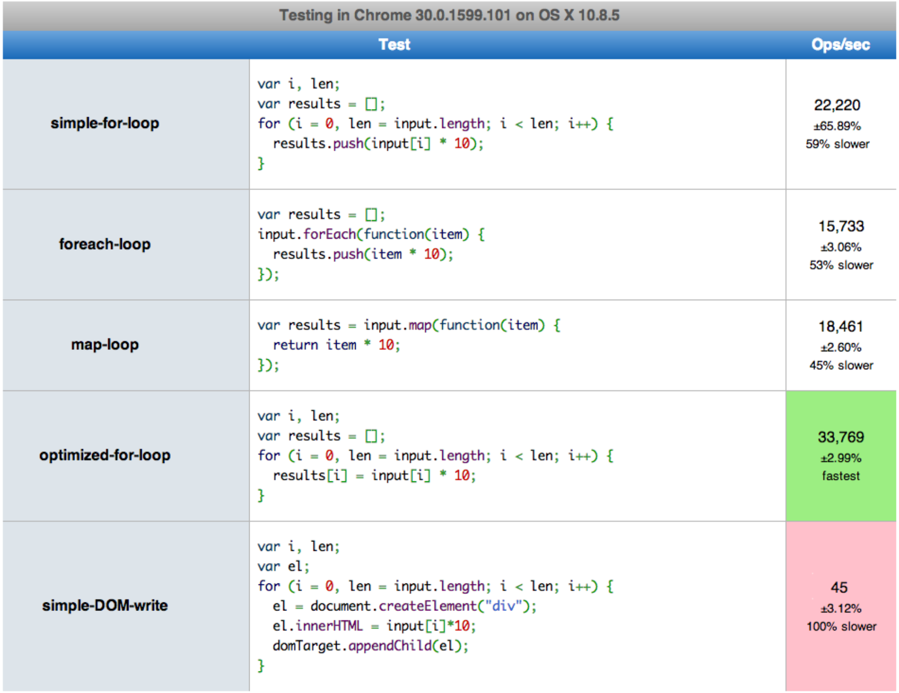
simple-for-loop	`var i, len;`
	`var results = [];`
	`for (i = 0, len = input.length; i < len; i++) {`
	`  results.push(input[i] * 10);`
	`}`
		22,220
		±65.89%
		59% slower

foreach-loop	`var results = [];`
	`input.forEach(function(item) {`
	`  results.push(item * 10);`
	`});`
		15,733
		±3.06%
		53% slower

map-loop	`var results = input.map(function(item) {`
	`  return item * 10;`
	`});`
		18,461
		±2.60%
		45% slower

optimized-for-loop	`var i, len;`
	`var results = [];`
	`for (i = 0, len = input.length; i < len; i++) {`
	`  results[i] = input[i] * 10;`
	`}`
		33,769
		±2.99%
		fastest

simple-DOM-write	`var i, len;`
	`var el;`
	`for (i = 0, len = input.length; i < len; i++) {`
	`  el = document.createElement("div");`
	`  el.innerHTML = input[i]*10;`
	`  domTarget.appendChild(el);`
	`}`
		pending…



Testing in Chrome 30.0.1599.101 on OS X 10.8.5

	Test	Ops/sec

simple-for-loop	`var i, len;`
	`var results = [];`
	`for (i = 0, len = input.length; i < len; i++) {`
	`  results.push(input[i] * 10);`
	`}`
		22,220
		±65.89%
		59% slower

foreach-loop	`var results = [];`
	`input.forEach(function(item) {`
	`  results.push(item * 10);`
	`});`
		15,733
		±3.06%
		53% slower

map-loop	`var results = input.map(function(item) {`
	`  return item * 10;`
	`});`
		18,461
		±2.60%
		45% slower

optimized-for-loop	`var i, len;`
	`var results = [];`
	`for (i = 0, len = input.length; i < len; i++) {`
	`  results[i] = input[i] * 10;`
	`}`
		33,769
		±2.99%
		fastest

simple-DOM-write	`var i, len;`
	`var el;`
	`for (i = 0, len = input.length; i < len; i++) {`
	`  el = document.createElement("div");`
	`  el.innerHTML = input[i]*10;`
	`  domTarget.appendChild(el);`
	`}`
		45
		±3.12%
		100% slower

What comes next?

Conclusion

Conclusion

- Functions are first-level citizens in JavaScript

Conclusion

- Functions are first-level citizens in JavaScript
- Try to create as many pure functions as possible

Conclusion

- Functions are first-level citizens in JavaScript
- Try to create as many pure functions as possible
  - Even if you are doing JavaScript OOP

Conclusion

- Functions are first-level citizens in JavaScript
- Try to create as many pure functions as possible
  - Even if you are doing JavaScript OOP
- Use JavaScripts functional built-ins

Conclusion

- Functions are first-level citizens in JavaScript
- Try to create as many pure functions as possible
  - Even if you are doing JavaScript OOP
- Use JavaScripts functional built-ins
- Create generic `map`/`reduce` functions

Conclusion

- Functions are first-level citizens in JavaScript
- Try to create as many pure functions as possible
  - Even if you are doing JavaScript OOP
- Use JavaScripts functional built-ins
- Create generic `map`/`reduce` functions
  - Btw: Closures ROCK!

Conclusion

- Functions are first-level citizens in JavaScript
- Try to create as many pure functions as possible
  - Even if you are doing JavaScript OOP
- Use JavaScripts functional built-ins
- Create generic `map`/`reduce` functions
  - Btw: Closures ROCK!
- Don’t let asynchronous scenarios stop you

Conclusion

- Functions are first-level citizens in JavaScript
- Try to create as many pure functions as possible
  - Even if you are doing JavaScript OOP
- Use JavaScripts functional built-ins
- Create generic `map`/`reduce` functions
  - Btw: Closures ROCK!
- Don't let asynchronous scenarios stop you
- **Write clean code first. Profile later.**

passion for software quality

THANK YOU

Rent a quality expert
qafoo.com