CSS AND JAVASCRIPT UPDATE

OVERVIEW

- A quick look back
- CSS: Flexbox
- JavaScript: ES6

A QUICK LOOK BACK

HTML AND CSS

From WEB1 you already know:

- Structuring content with HTML
- Document structure, lists, forms, tables, media
- Semantic markup
- CSS2 and CSS3 selectors
- Styling text and boxes
- Box model, positioning elements, floating boxes

If necessary consult:

- WEB1 slides and related material
- Learn to Code HTML & CSS

HTML AND CSS

You probably also know:

- Front-end frameworks like Foundation or Bootstrap
- CSS preprocessors like Less or Sass

HTML AND CSS

- CSS3 has a Flexible Box Layout Module, short: *Flexbox*
- Currently a *W3C Last Call Working Draft*
- This has not been addressed in WEB1

More on CSS3 *Flexbox* in a moment...

JAVASCRIPT

From WEB2 you know:

- JavaScript basics: statements, expressions, variables
- Values and types: numbers, strings, boolean, undefined, ...
- Functions, closures, this, scoping
- Objects, methods, constructors, prototypes
- Arrays, Math-object, regular expressions
- Strict mode

If necessary consult:

- WEB2 slides and related material
- Speaking JavaScript, Chapter 1: Basic JavaScript

JAVASCRIPT: FUNCTIONS

```
// Function declaration
function add (a, b) { return a+b; }

// Function expression
var add = function (a, b) { return a+b; };

// Named function expression
// Name fact is used here for recursive call
// It's scope is limited to the function
var factorial = function fact (n) {
    if (n <= 1) return 1;
    else return n * fact(n-1);
};
```

JAVASCRIPT: CLOSURES, CALLBACKS

```javascript
// Fade the background color of an element
var fade = function (node) {
    var level = 1;
    var step = function () {
        var hex = level.toString(16);
        node.style.backgroundColor = '#FFFF' + hex + hex;
        if (level < 15) {
            level += 1;
            setTimeout(step, 100);
        }
    };
    setTimeout(step, 100);
};

// Sample call
window.onload = function() {
    fade(document.querySelector(".newitem"));
};
```

JAVASCRIPT: OBJECTS AND CONSTRUCTORS

```
var Person = function (name) {
    this.name = name;
    this.greet = function () {
        return "Hello, my name is " + this.name;
    };
};

var eva = new Person("Eva");
console.log(eva.greet());   // Hello, my name is Eva
```

JAVASCRIPT: INHERITANCE

```
// User extends Person
var User = function (name, uid) {
    Person.call(this, name);
    this.uid = uid;
};

// Dummy Person object for the inheritance chain
User.prototype = new Person;
User.prototype.constructor = User;
```

JQUERY

```
// example from the exercise
readNotes : function () {
    $.getJSON( this.SERVER_URL + "notes", function(data) {

        //delete all notes
        $('#notes').empty();

        //now iterate over our notes (template preferred...)
        $.each(data.notes, function(i, note) {
          $('<li class="note" data-note-id="'+note.id+'">')
            .append('<span class="subject">'+note.subject+'</span>')
            .append('<span class="message">'+note.message+'</span>')
            .append('<a class="delete" href="#">delete</a>')
            .appendTo('#notes');
        });
    });
}
```

JQUERY

```
addHandler : function () {
    var url = this.SERVER_URL;

    $('#notes').on('click', '.delete', function (e) {
        var note = $(this).parents('.note'),
            noteid = note.data('note-id');

        $.getJSON(url + "deleteNote?id=" + noteid, function(data) {
          //check the server answer
          if (data.message == 'ok') {
            //if successful, reload the list
            NOTE_CLIENT.readNotes();
          }
          else {
            //if not successful, show an error message
            alert('Note could not be deleted');
          }
        });
        e.preventDefault();
    });
}
```

JAVASCRIPT: CURRENT DEVELOPMENT

- ECMAScript 6 (2015), more in a moment...
- Node.js, Io.js merged again
- jQuery 3 in development, status: alpha
- Web Components, in another lesson...

JAVASCRIPT APPLICATION FRAMEWORKS

- Backbone, Ember, Angular, Meteor, React
- These can prove useful for mobile applications, too
- We will have a look into React.js in another lesson
- React Native for building native apps using the React way

CSS FLEXBOX

CSS FLEXBOX

- Page layout is difficult with traditional CSS
- Hence the various grid frameworks like Bootstrap
- Flexbox aims to facilitate typical layout problems

CSS FLEXBOX

- Efficient way to lay out, align and distribute items in a container
- Option to re-order items helps making responsive designs
- Used in React Native for laying out native apps
- Will optionally be used by Bootstrap (V4, currently alpha)

CSS Flexible Box Layout Module Level 1

BROWSER SUPPORT

caniuse.com/#search=flexbox

- Overall good support of current browsers
- Problems with Internet Explorer, good support by Edge
- Safari up to version 8: Use -webkit prefix

The examples on the following slides are from http://flexbox.io/

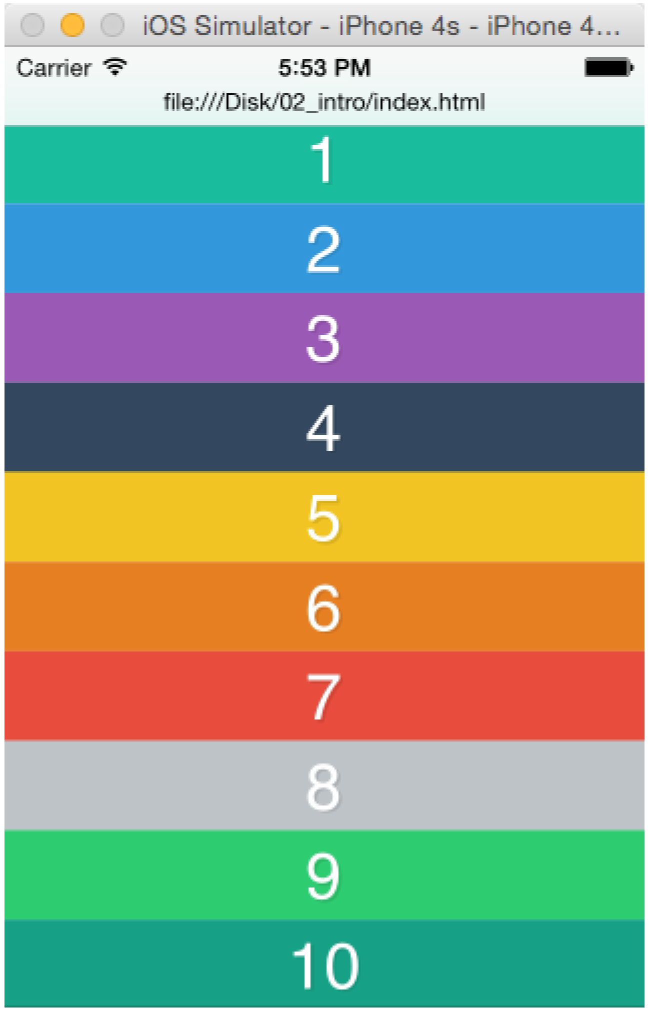
EXAMPLE

```
<!-- HTML -->
<div class="container">
    <div class="box box1">1</div>
    <div class="box box2">2</div>
    ...
    <div class="box box10">10</div>
</div>
```

```
/* CSS */
.box {
    padding:10px;
    text-align: center;
    color:white;
    text-shadow:4px 4px 0 rgba(0,0,0,0.1);
    font-size: 100px;
}
.box1 { background:#1abc9c;}
...
```

↓ preview ↓

EXAMPLE


iOS Simulator - iPhone 4s - iPhone 4…
Carrier
5:53 PM
file:///Disk/02_intro/index.html
1
2
3
4
5
6
7
8
9
10

EXAMPLE: FLEX CONTAINER

```
.container {
    display: flex;
    border: 10px solid goldenrod;
    height: 100vh;
}
```

- Flexbox consists of flex containers and flex items
- A flex container is declared with the *display* property
- Container is now a *flex container*
- Alternative: *display: inline-flex* results in an inline element

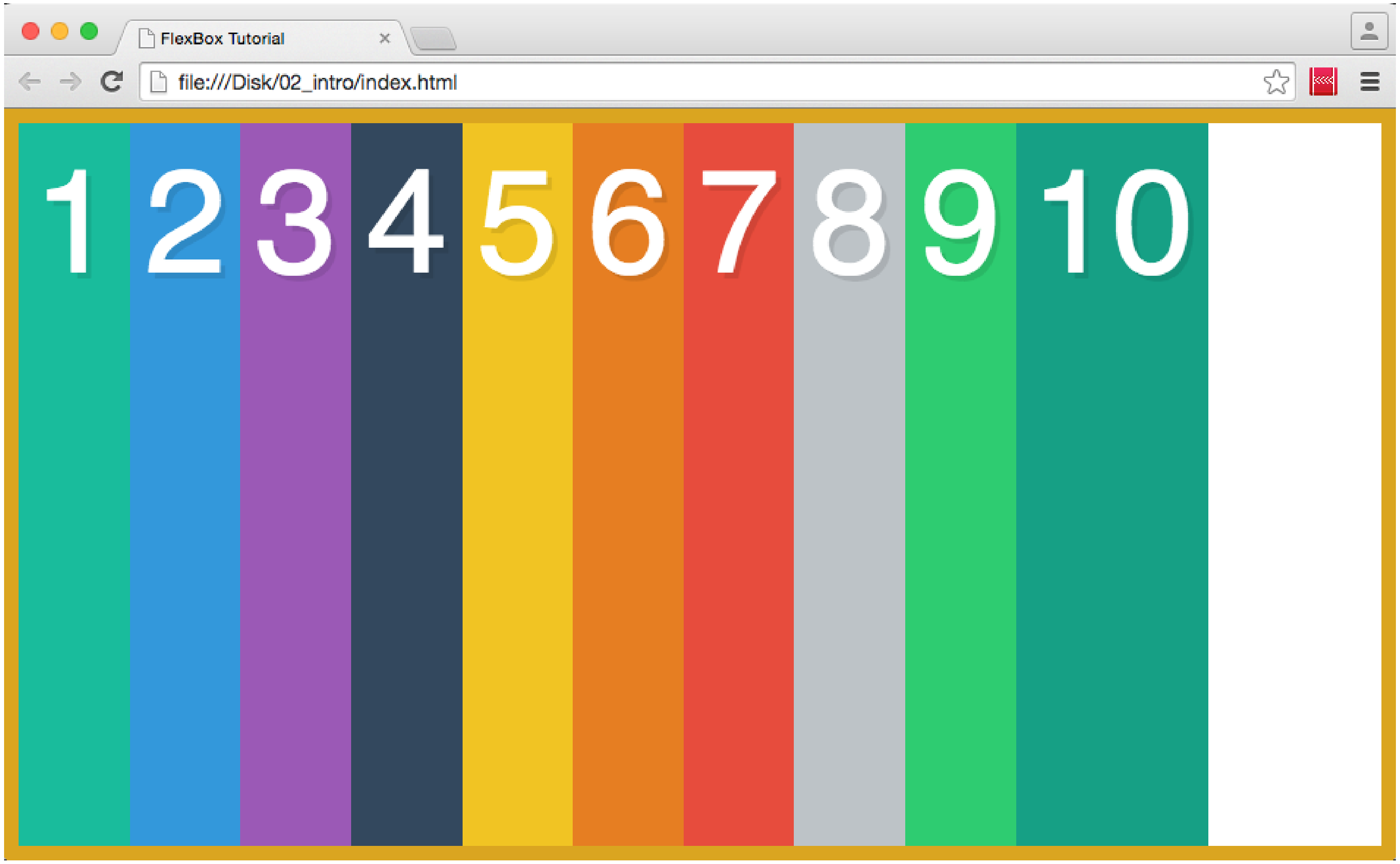
EXAMPLE: FLEX CONTAINER

```css
.container {
    display: flex;
    border: 10px solid goldenrod;
    height: 100vh;
}
```

- 100vh – what’s that ??
- It’s the viewport height

↓ preview ↓

EXAMPLE: FLEX CONTAINER



FLEX ITEMS

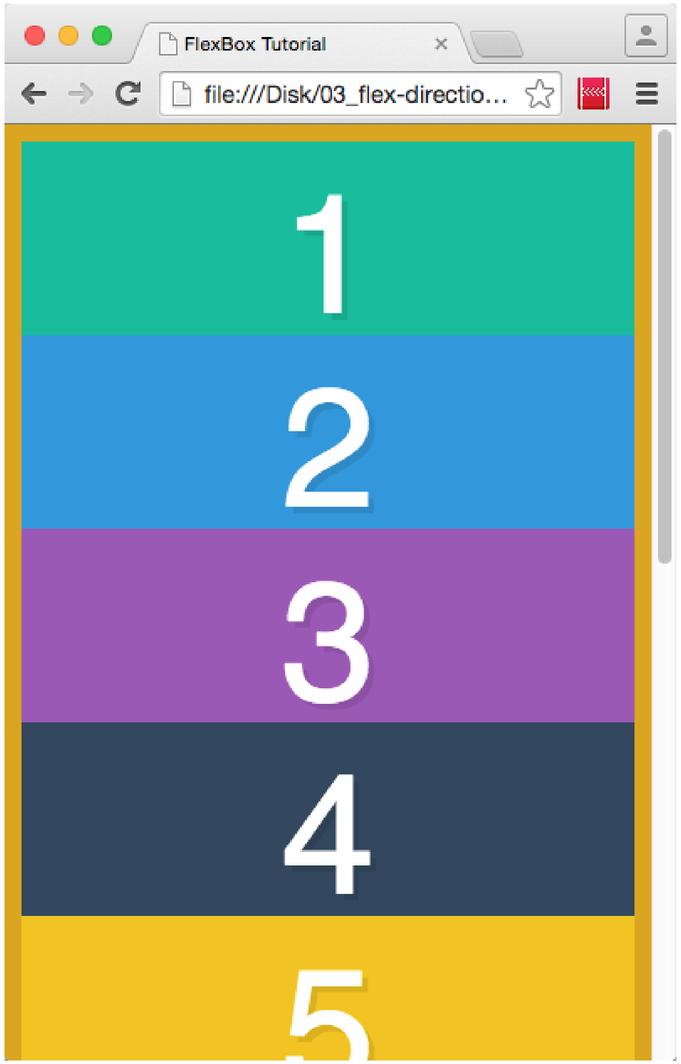
- Every child of a *flex container* is a *flex item*
- There can be any number of flex items
- Flexbox defines how flex items are laid out inside of flex containers

FLEX DIRECTION

```
.container {
    display:flex;
    border:10px solid goldenrod;
    min-height:100vh;
    flex-direction: column;
}
```

↓ preview ↓

FLEX DIRECTION



FLEX DIRECTION

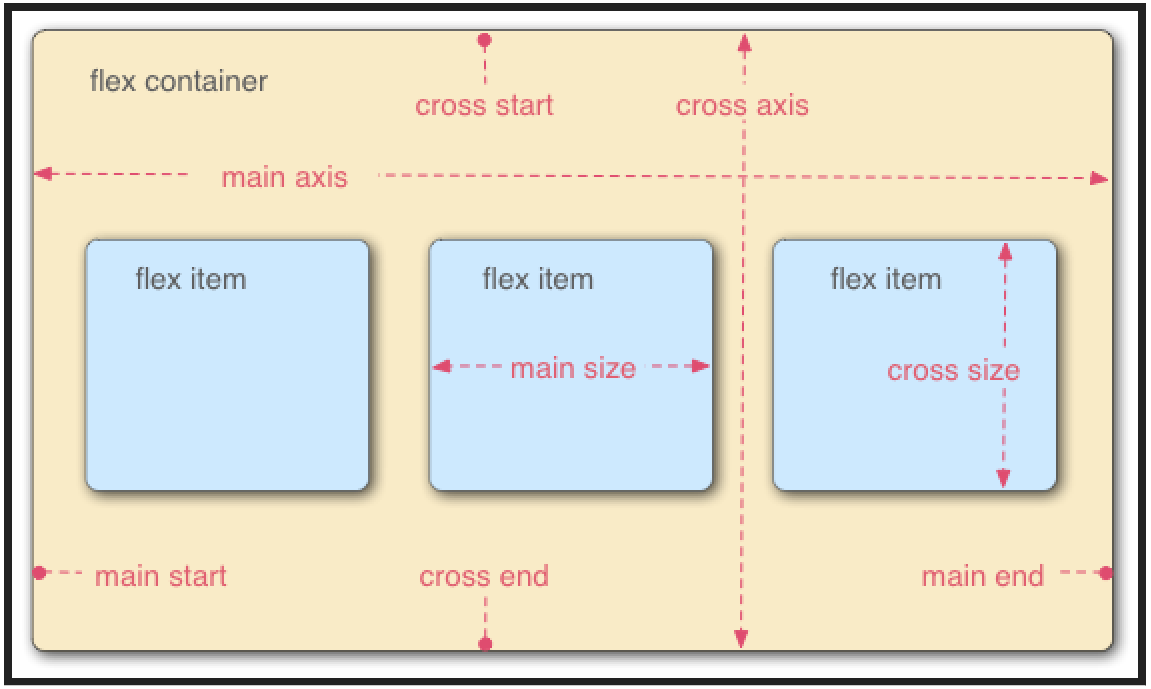
- CSS attribute: *flex-direction*
- Possible values: *row* | *row-reverse* | *column* | *column-reverse*
- Default: *row*
  (items are laid out in the direction of the writing-mode)
- Terminology:
  - Main direction (sometimes called the flow direction)
  - Main start and main end
  - Cross direction
  - Cross start and cross end

FLEX DIRECTION

```
body {
    direction: rtl;     /* main direction right to left */
}
.container {
  display:flex;
  border:10px solid goldenrod;
  min-height:100vh;
  flex-direction: row-reverse;  /* left to right again */
}
```

↓ preview ↓

FLEX DIRECTION



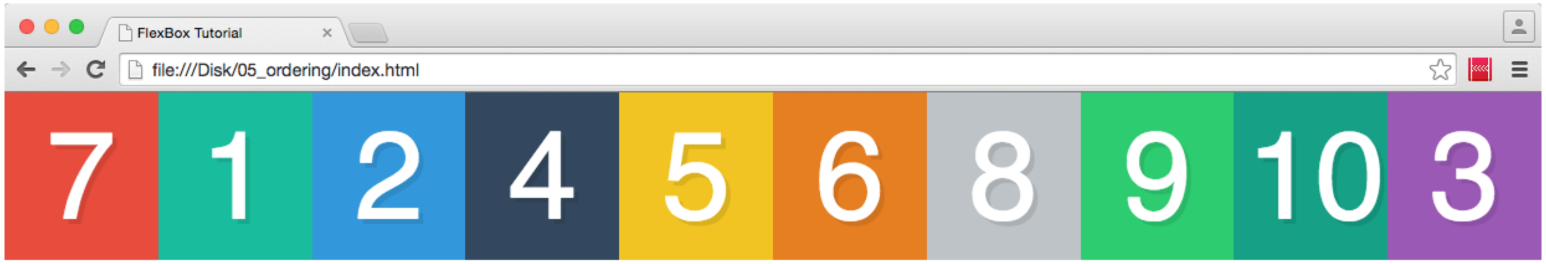
(Source: MDN)

FLEXBOX ORDERING

```
.container {
  display:flex;
}
.box {
  flex:1;
  order:1;
}
.box3 {
  order:3;
}
.box7 {
  order:-2;
}
```

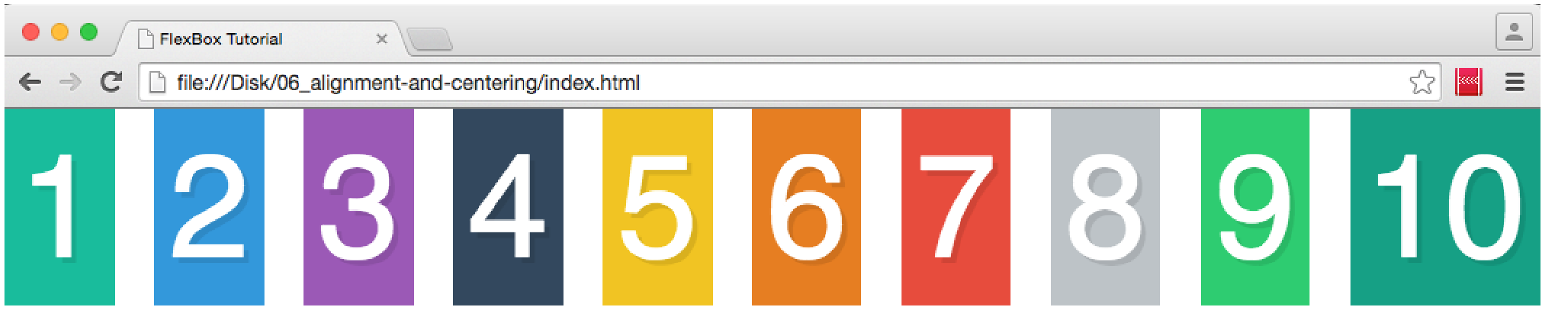
FLEXBOX ORDERING

- Flex items can be re-ordered with the `order` property
- Initial value: 0
- Problems likely when selecting text over multiple items



FLEXBOX ALIGNMENT

```
.container {
  display:flex;
  justify-content:space-between;
}
```



FLEXBOX ALIGNMENT

- Property: *justify-content*
- Defines the alignment along the *main* axis
- Remember: with *flex-direction:column* the main axis is from top to bottom
- Possible values: *flex-start* | *flex-end* | *center* | *space-between* | *space-around*
- Default: *flex-start*

FLEXBOX ALIGNMENT

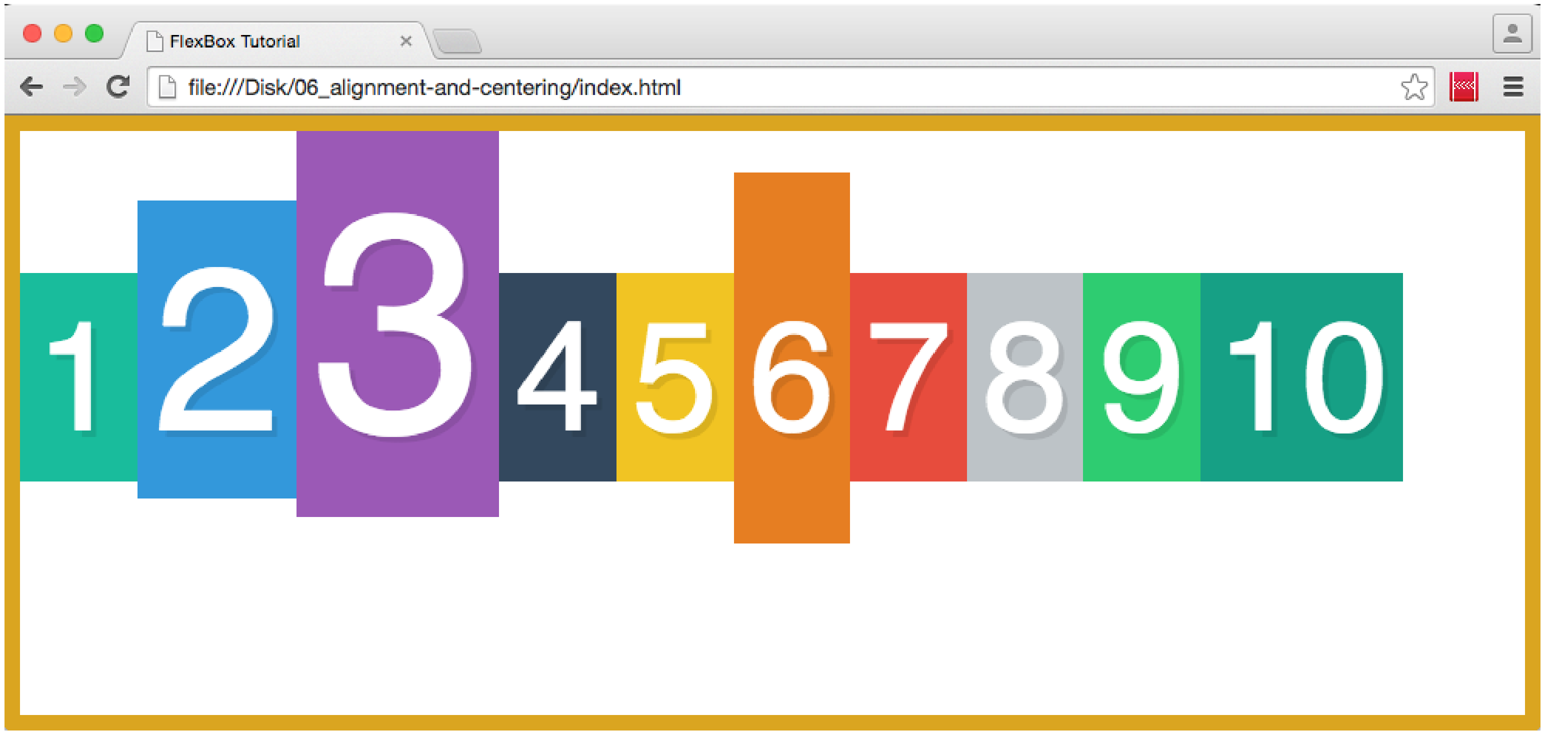
- Another property : *align-items*
- This one defines the alignment along the *cross* axis
- Possible values: *flex-start* | *flex-end* | *center* | *baseline* | *stretch*
- Default: *stretch*

FLEXBOX ALIGNMENT

```css
.container {
  display:flex;
  border:10px solid goldenrod;
  min-height:100vh;
  align-items:baseline;
}
.box2 {
  font-size: 150px;
}
.box3 {
  font-size: 200px;
}
.box6 {
  padding-bottom: 50px;
  padding-top: 75px;
}
```

↓ preview ↓

FLEXBOX ALIGNMENT



FLEXBOX ALIGNMENT

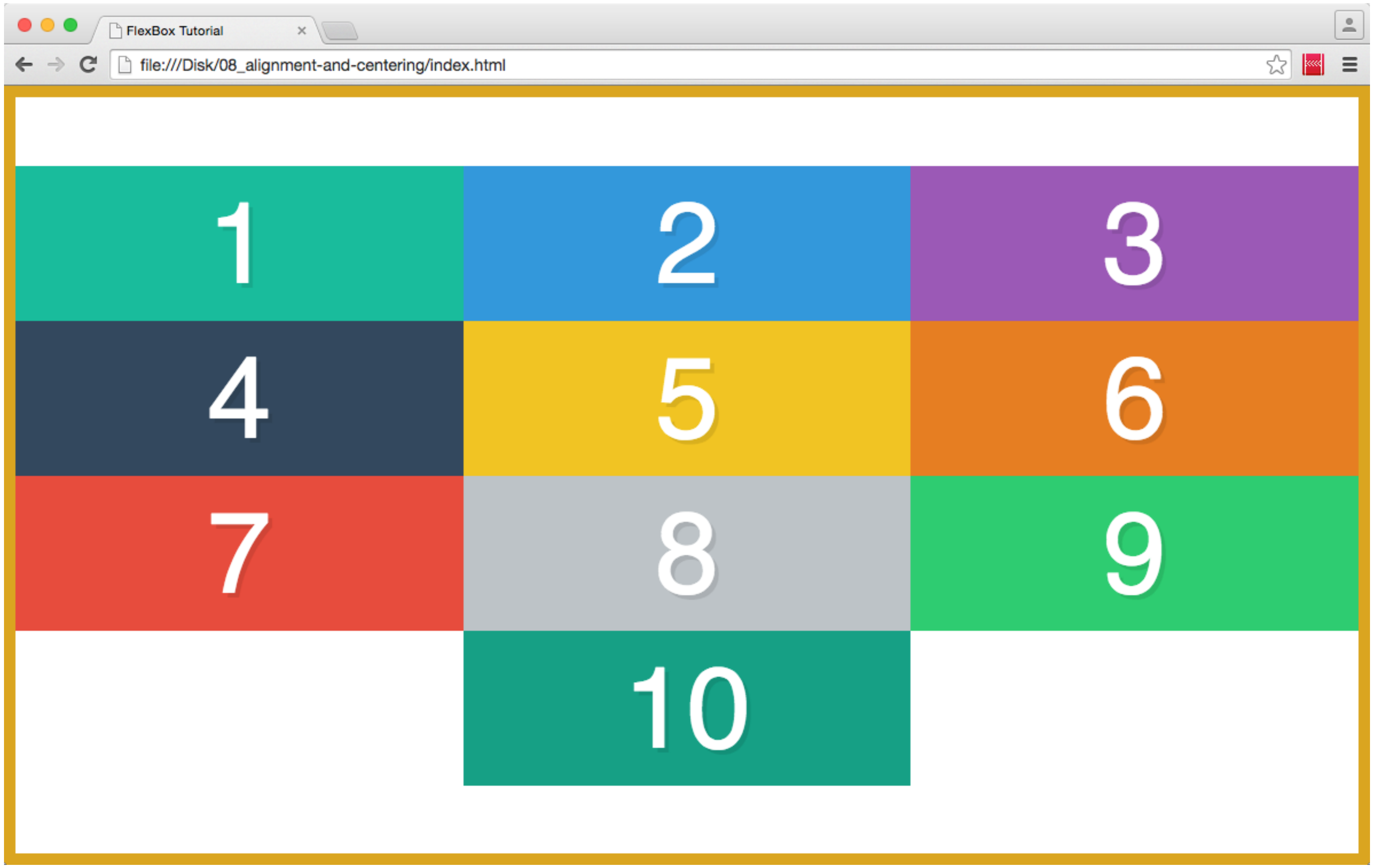
- One more property : *align-content*
- Alignment along the *cross* axis for the whole content
- Possible values: *flex-start* | *flex-end* | *center* | *space-between* | *space-around* | *stretch*
- Default: *stretch*

FLEXBOX ALIGNMENT

```
.container {
  display:flex;
  border:10px solid goldenrod;
  min-height:100vh;
  flex-wrap: wrap;
  justify-content: center;
  align-content: center;
}
.box {
  width: 33.33333%;
}
```

↓ preview ↓

FLEXBOX ALIGNMENT

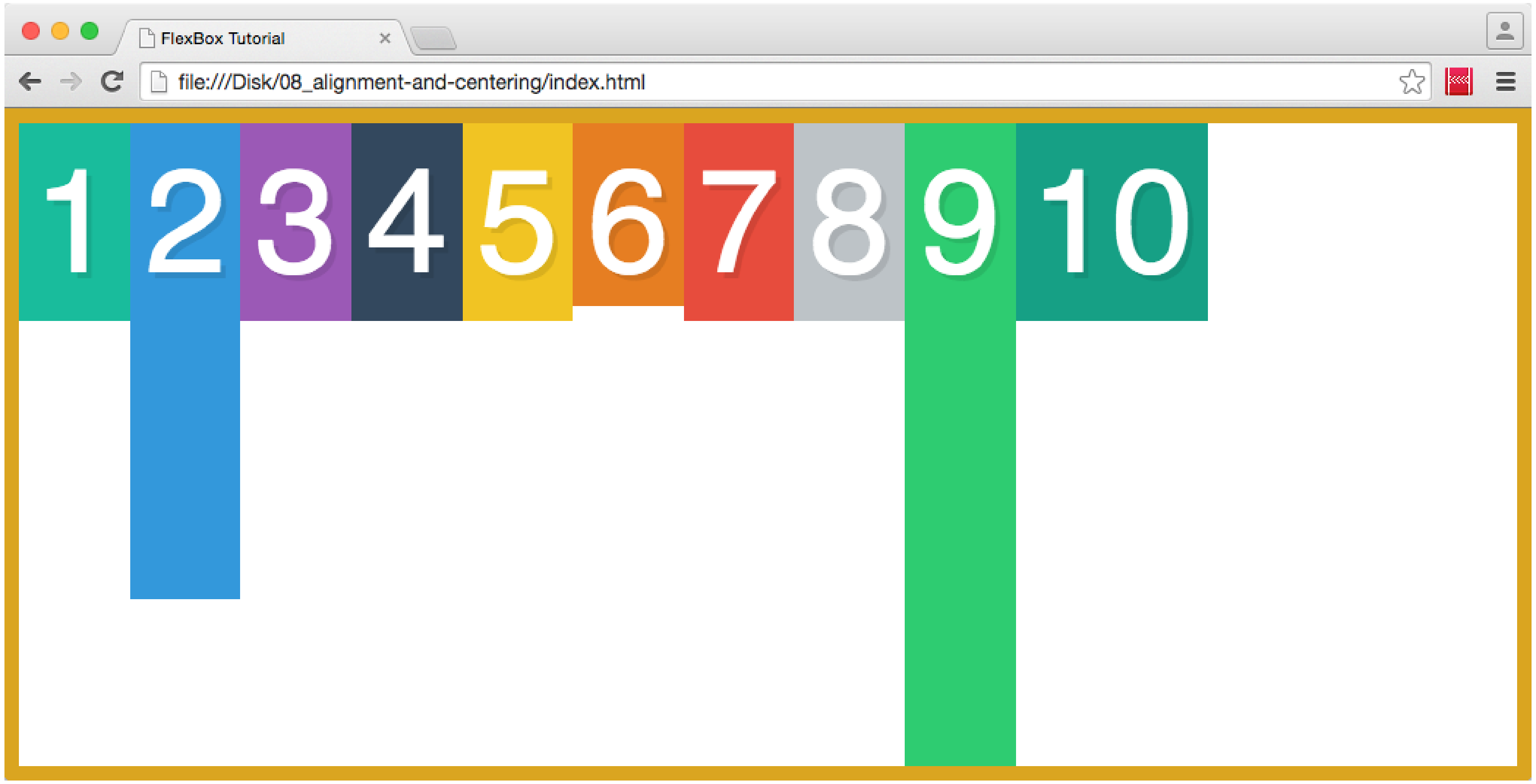


FLEXBOX SELF ALIGNMENT

```
.container {
  display:flex;
  border:10px solid goldenrod;
  min-height:100vh;
  align-items: flex-start;
}
.box2 {
  padding-bottom: 200px;
}
.box6 {
  padding-bottom: 0;
}
.box9 {
  padding-bottom: 50px;
  align-self: stretch;
}
```

↓ preview ↓

FLEXBOX SELF ALIGNMENT



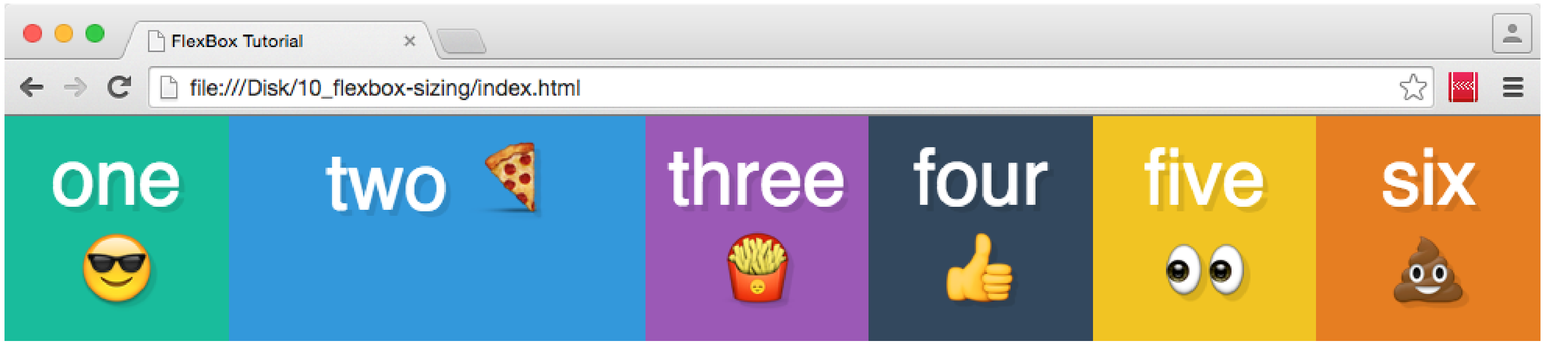
FLEXBOX SIZING

```
<!-- HTML -->
<div class="container">
  <div class="box box1">one   </div>
  <div class="box box2">two   </div>
  <div class="box box3">three   </div>
  ...
</div>
```

```
/* CSS */
.container {
  display:flex;
}
.box { /*...*/
  flex:1;
}
.box2 {
  flex:2;
}
```

↓ preview ↓

FLEXBOX SIZING



GROW AND SHRINK FLEX ITEMS

- `flex-grow`
  - Defines the ability for a flex item to grow if necessary
  - Unitless value that serves as a proportion
  - Default: 0
- `flex-shrink`
  - Defines the ability for a flex item to shrink if necessary
  - Default: 1
- `flex-basis`
  - Size of an element before the remaining space is distributed
  - Default: *auto*

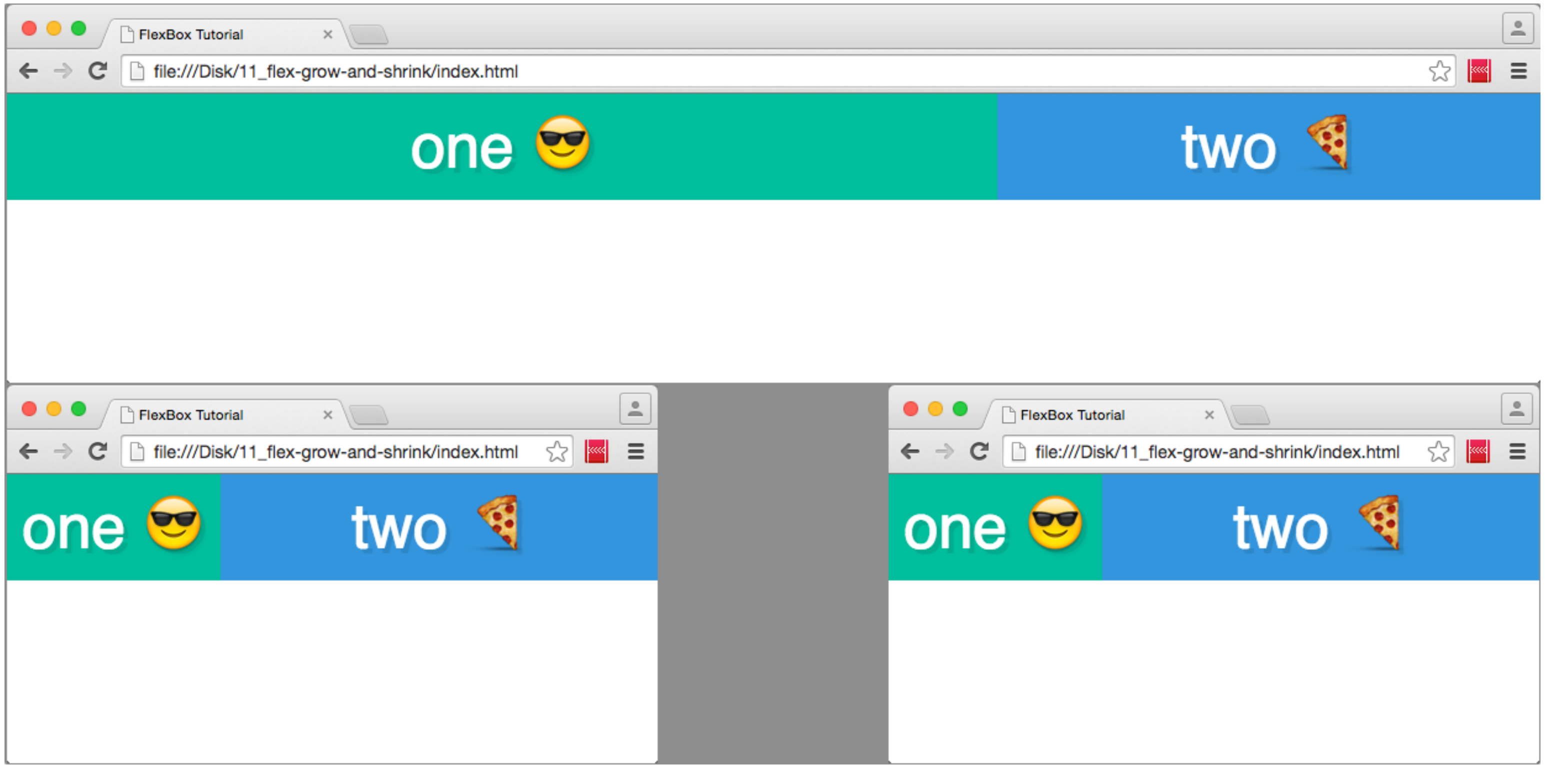
PROPERTY *FLEX*

- Shorthand for *flex-grow*, *flex-shrink* and *flex-basis* combined
- Second and third parameters are optional
- Default: 0 1 auto

```
.box1 {
    flex: 10 5 400px;
}
.box2 {
    flex: 1 1 400px;
}
```

↓ preview ↓

GROW AND SHRINK FLEX ITEMS


FlexBox Tutorial
file:///Disk/11_flex-grow-and-shrink/index.html
one 😎
two 🍕
FlexBox Tutorial
file:///Disk/11_flex-grow-and-shrink/index.html
one 😎
two 🍕
FlexBox Tutorial
file:///Disk/11_flex-grow-and-shrink/index.html
one 😎
two 🍕

RESIZING COMBINED WITH WRAPPING

- Resizing/wrapping combined allows for flexible layouts
- To demonstrate we use a container with six flex items

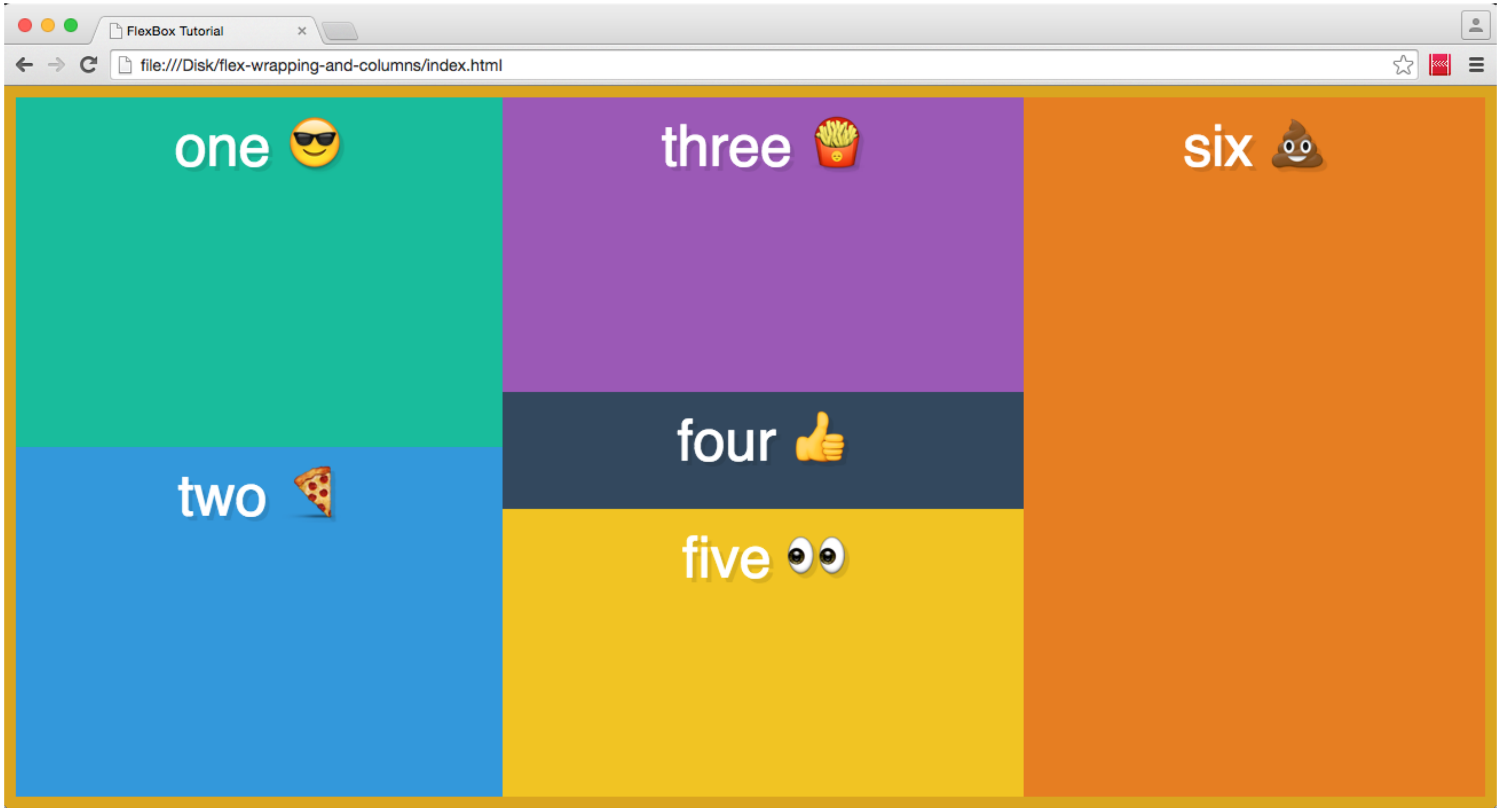
```
<div class="container">
    <div class="box box1">one   </div>
    ...
    <div class="box box6">six   </div>
</div>
```

RESIZING COMBINED WITH WRAPPING

```
.container {
    display:flex;
    flex-wrap:wrap;
    flex-direction:column;
    border:10px solid goldenrod;
    height:100vh;
}
.box {
    flex-basis:250px;
    flex-grow:1;
}
.box3 {
    flex-grow:5;
}
.box4 {
    flex-basis:100px;
}
```

↓ preview ↓

RESIZING COMBINED WITH WRAPPING



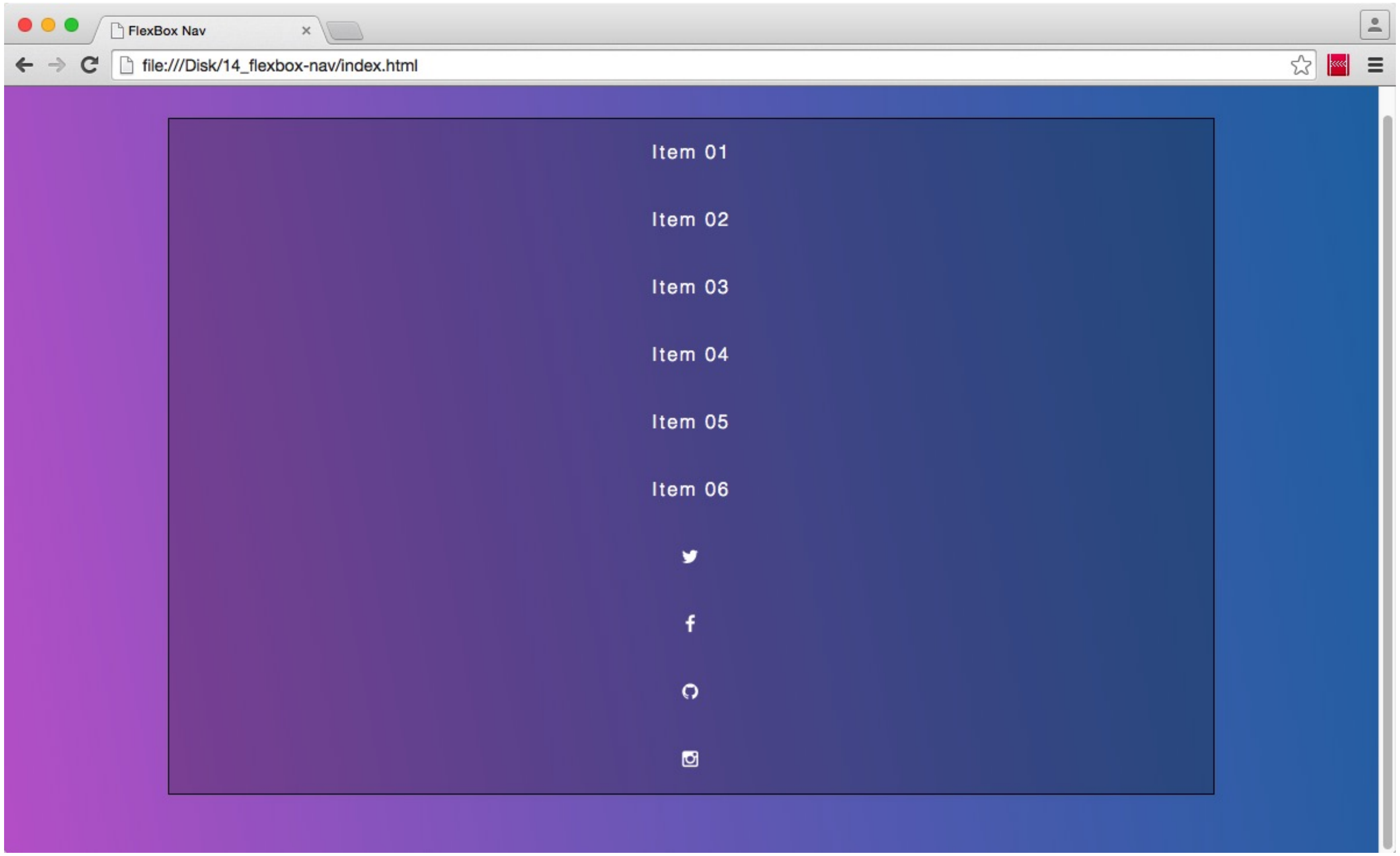
EXAMPLE: NAVIGATION

HTML

```
<nav class="flex-nav">
  <ul>
    <li><a href="#">Item 01</a></li>
    <li><a href="#">Item 02</a></li>
    ...
    <li class="social">
      <a href="http://twitter.com/wesbos"><i class="fa fa-twitter"></i></a>
    </li>
    ...
  </ul>
</nav>
```

↓ With a little bit of styling not shown here ↓

EXAMPLE: NAVIGATION

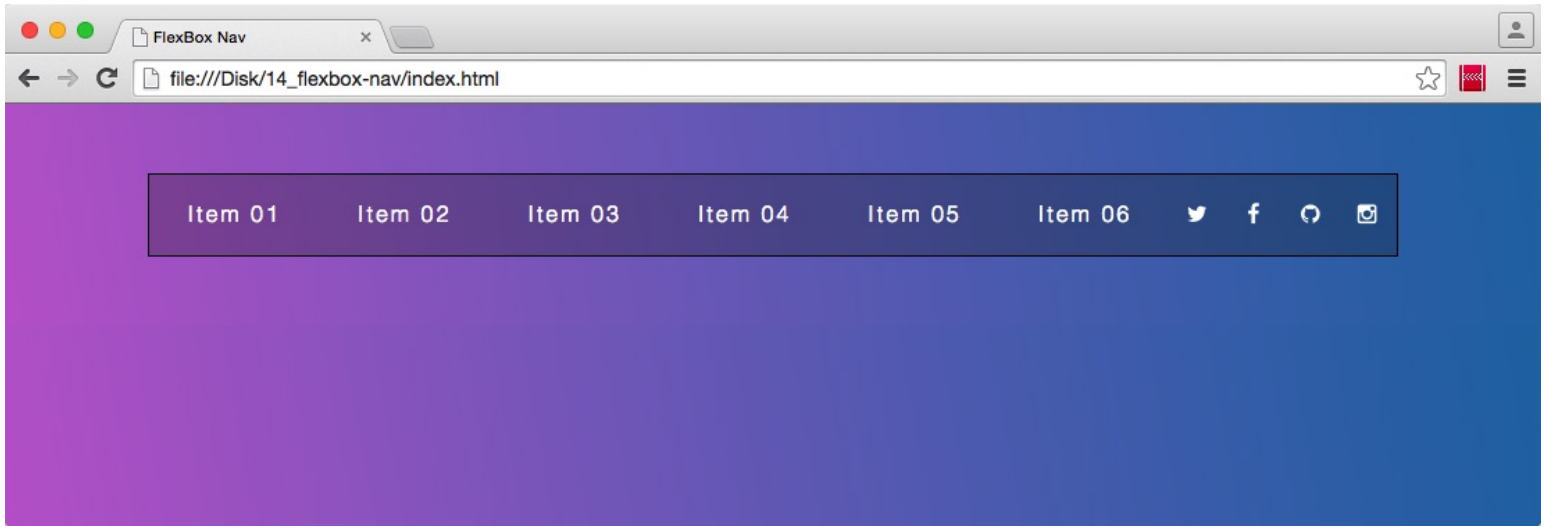


EXAMPLE: NAVIGATION

```css
.flex-nav ul {
    border:1px solid black;
    list-style: none;
    margin: 0;
    padding: 0;
    display: flex;
}
.flex-nav li {
    flex: 3;
}
.flex-nav .social {
    flex: 1;
}
```

↓ preview ↓

EXAMPLE: NAVIGATION

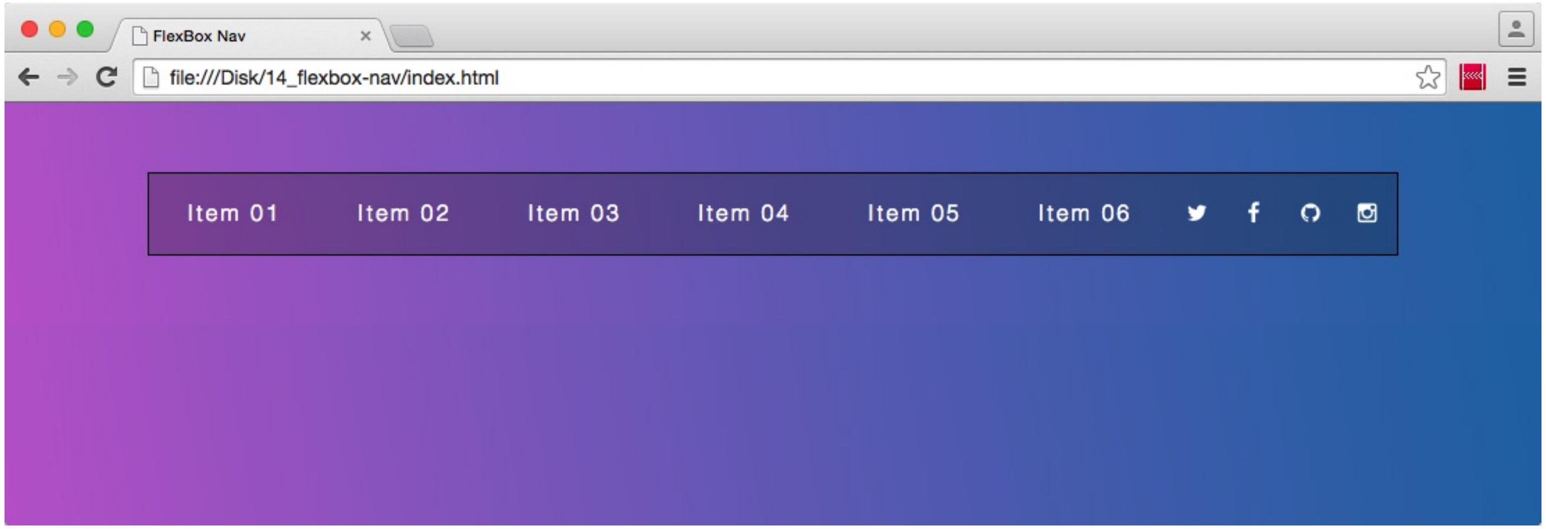


EXAMPLE: NAVIGATION

```css
@media all and (max-width:1000px) {
    .flex-nav ul {
        flex-wrap: wrap;
    }
    .flex-nav li {
        flex: 1 1 50%;
    }
    .flex-nav .social {
        flex: 1 1 25%;
    }
}
@media all and (max-width:500px) {
  .flex-nav li {
      flex-basis: 100%;
  }
}
```

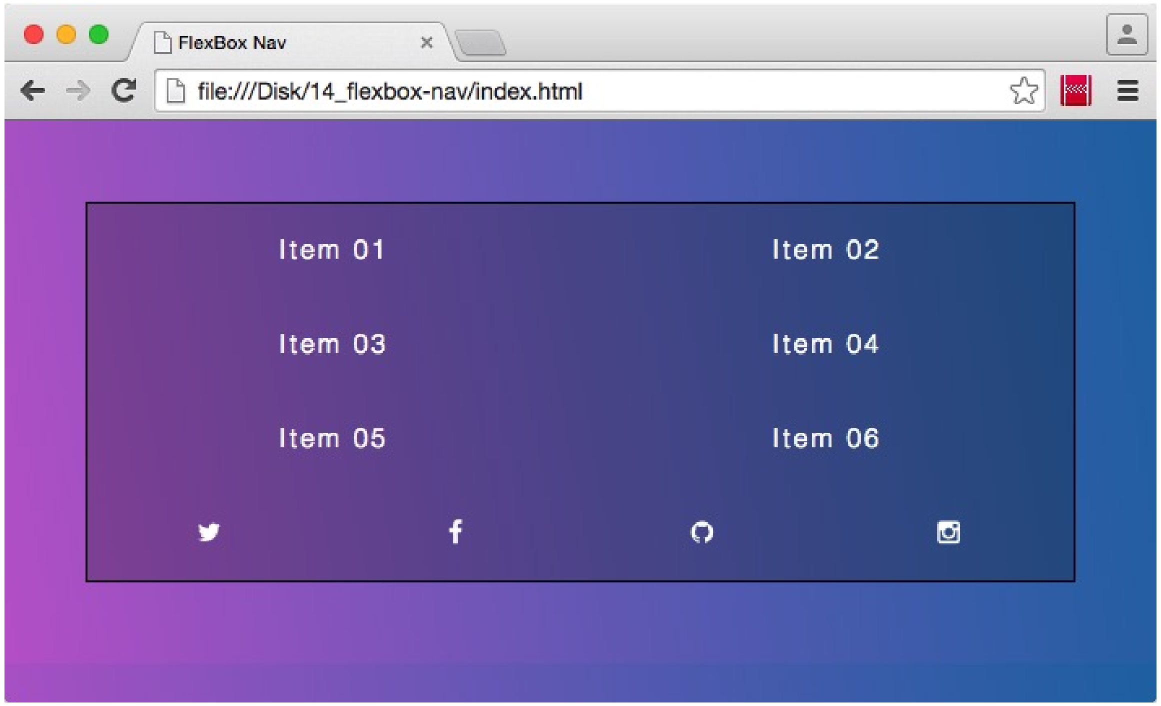
↓ preview ↓

EXAMPLE: NAVIGATION



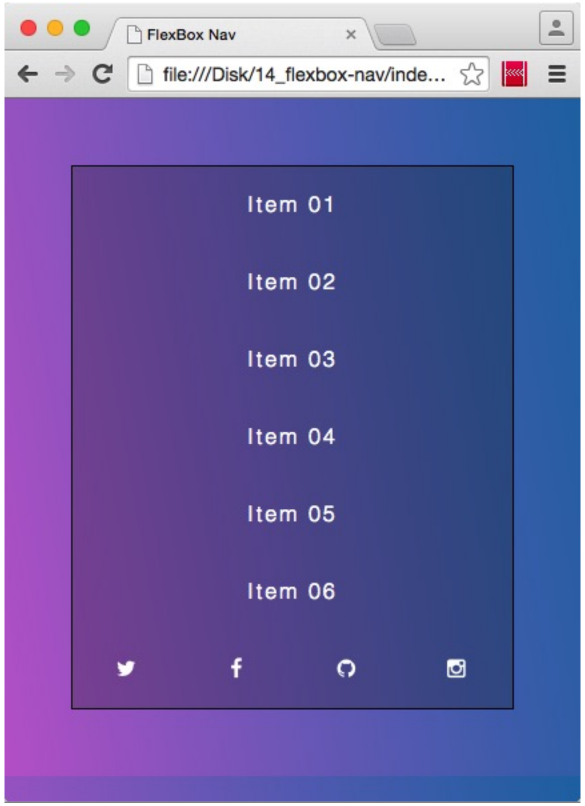
↓ more ↓

EXAMPLE: NAVIGATION

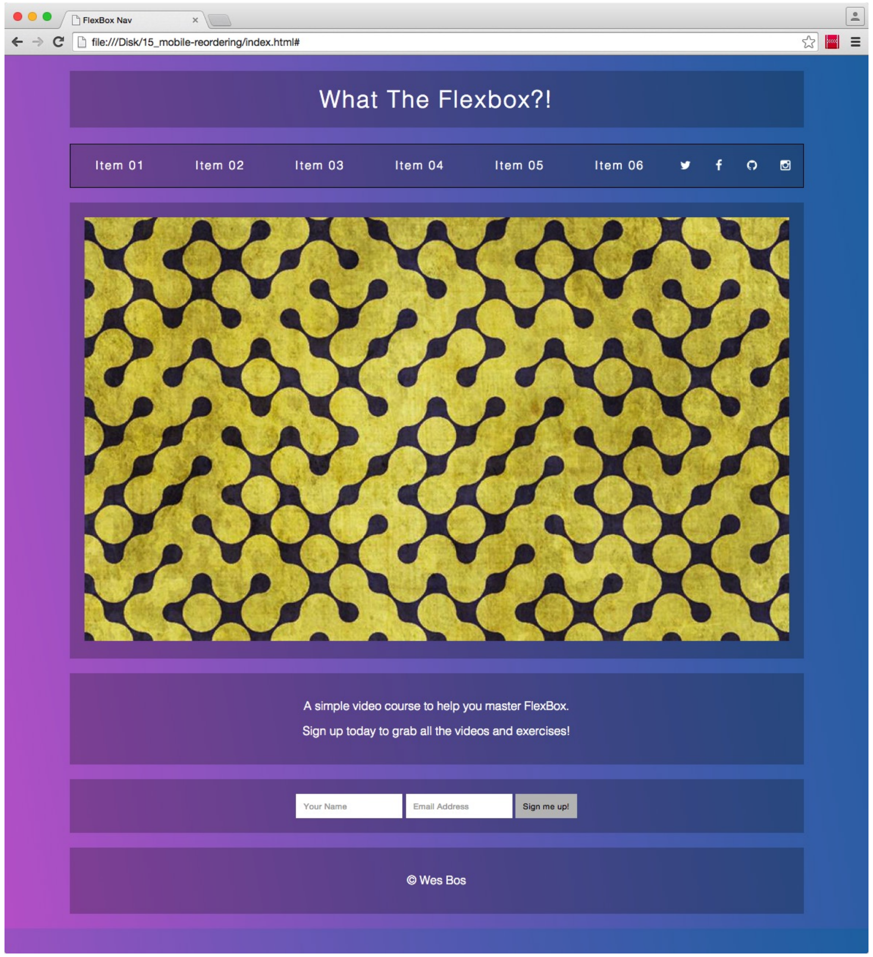


↓ more ↓

EXAMPLE: NAVIGATION



EXAMPLE: MOBILE REORDERING



What The Flexbox?!

Item 01 Item 02 Item 03 Item 04 Item 05 Item 06

A simple video course to help you master FlexBox.
Sign up today to grab all the videos and exercises!

© Wes Bos

EXAMPLE: MOBILE REORDERING

```
<body>
    <div class="wrapper">
        <header class="top">...</header>
        <nav class="flex-nav">...</nav>
        <section class="hero">...</section>
        <section class="details">...</section>
        <section class="signup">...</section>
        <footer>...</footer>
    </div>
  <script>...</script>
</body>
```

EXAMPLE: MOBILE REORDERING

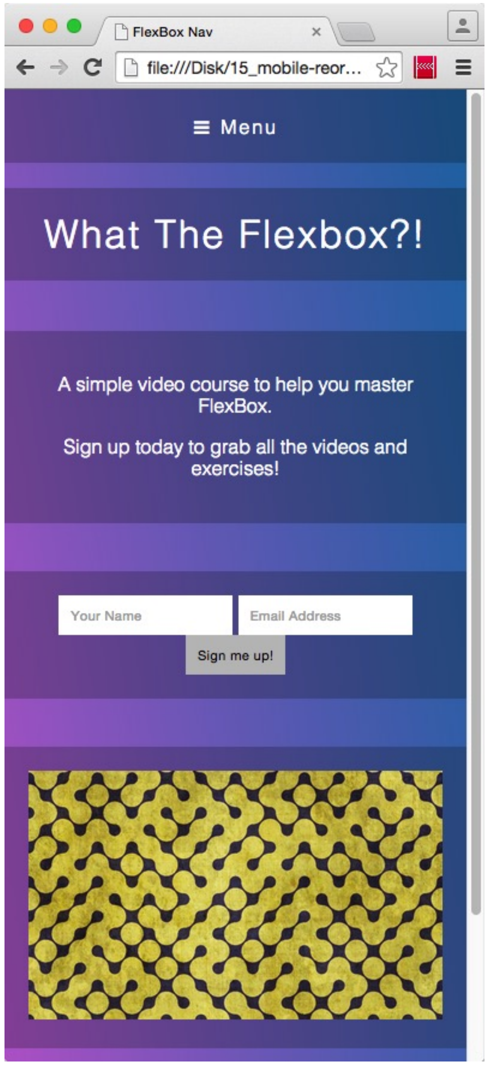
- On small mobile devices
  - the navigation should be on top of the screen
  - it should also be replaced by a toggle button
  - the details and signup areas should be moved up
- All this is easy with flexbox
- For re-ordering, the outer div must be *flex*

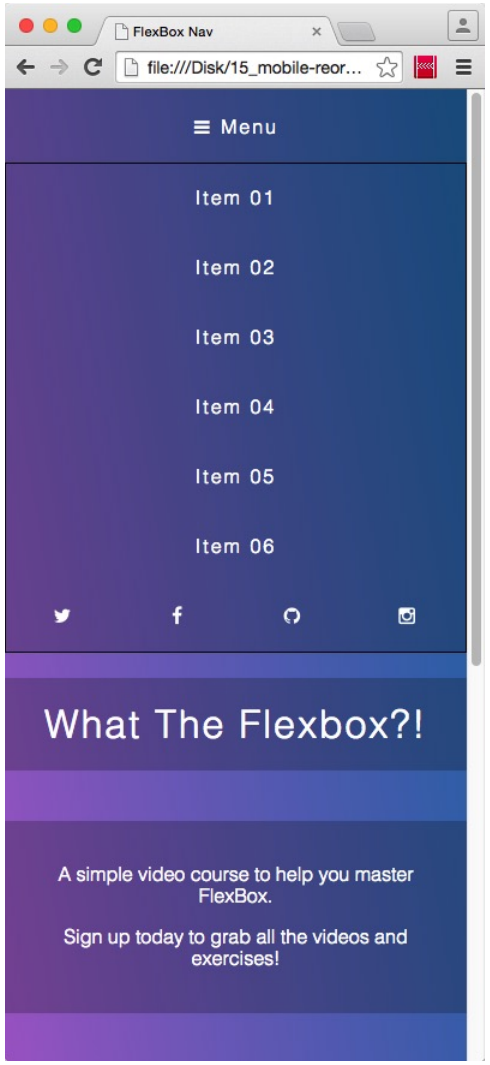
EXAMPLE: MOBILE REORDERING

```
@media all and (max-width:500px) {
  .wrapper {
    display: flex;
    flex-direction: column;
  }
  .wrapper > * {
    order: 9999;
  }
  .flex-nav { order: 1;
  }
  .toggleNav {
    display: block;
  }
  .flex-nav ul {
    display: none;
  }
  .flex-nav ul.open {
    display: flex;
  }
  .top { order: 2; }
  .details { order: 3; }
  .signup  { order: 4; }
}
```

↓ preview ↓

EXAMPLE: MOBILE REORDERING


FlexBox Nav
file:///Disk/15_mobile-reor…
≡ Menu
What The Flexbox?!
A simple video course to help you master FlexBox.
Sign up today to grab all the videos and exercises!
Your Name
Email Address
Sign me up!


FlexBox Nav
file:///Disk/15_mobile-reor…
≡ Menu
Item 01
Item 02
Item 03
Item 04
Item 05
Item 06
What The Flexbox?!
A simple video course to help you master FlexBox.
Sign up today to grab all the videos and exercises!

MOBILE APP LAYOUT

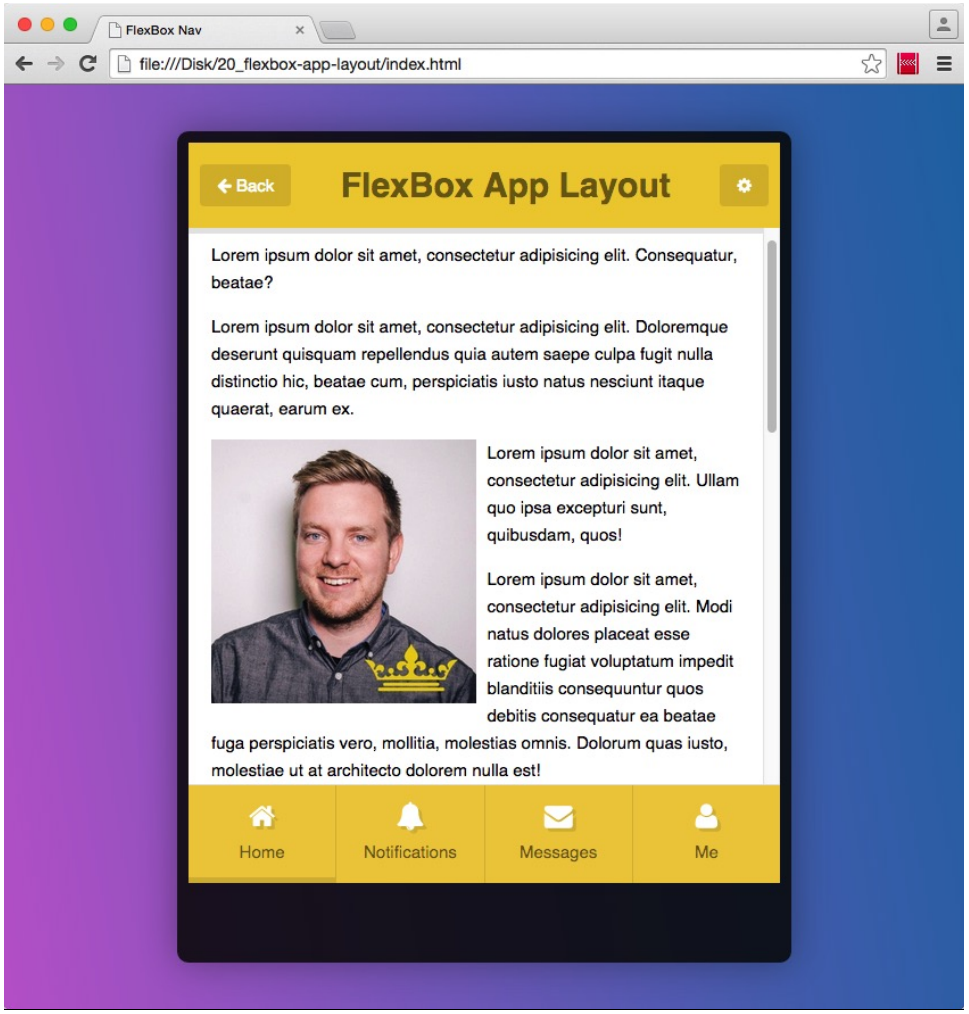
```html
<div class="app-wrap">
  <header class="app-header">
    <a class="button">...</a>
    <h1>FlexBox App Layout</h1>
    <a class="button">...</a>
  </header>

  <div class="content">
    <p>...</p><p>...</p><img><p>...</p>
  </div>

  <div class="icon-bar">
    <a><i class="fa fa-home"></i>Home</a>
    <a><i class="fa fa-bell"></i>Notifications</a>
    <a><i class="fa fa-envelope"></i>Messages</a>
    <a><i class="fa fa-user"></i>Me</a>
  </div>
</div>
```

MOBILE APP LAYOUT

```css
.app-wrap {
  display:flex;
  flex-direction:column;
}
.app-wrap > * {
  flex:1 1 auto;
}
.app-header {
  display:flex;
  align-items:center;
  justify-content:space-between;
}
.content {
  overflow-y:scroll;
  -webkit-overflow-scrolling:touch;
}
.icon-bar {
  display:flex;
}
.icon-bar a {
  flex:1;
}
```

↓ preview ↓

MOBILE APP LAYOUT



Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, beatae?

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Doloremque deserunt quisquam repellendus quia autem saepe culpa fugit nulla distinctio hic, beatae cum, perspiciatis iusto natus nesciunt itaque quaerat, earum ex.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ullam quo ipsa excepturi sunt, quibusdam, quos!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Modi natus dolores placeat esse ratione fugiat voluptatum impedit blanditiis consequuntur quos debitis consequatur ea beatae fuga perspiciatis vero, mollitia, molestias omnis. Dolorum quas iusto, molestiae ut at architecto dolorem nulla est!

FLEXBOX COMPATIBILITY

- Flexbox changed over time
    - CSS Tricks: Old Flexbox and New Flexbox
    - Mozilla Developer Network: flex-basis
- Some browsers (Safari < 9) need vendor prefixes
- Use an autoprefixer
    - Try: autoprefixer.github.io
- Make this part of your build procedure
    - Use Grunt, Gulp, CodeKit, or similar tool

JAVASCRIPT: ES6

ECMA-262 6TH EDITION

Also called:

- ECMAScript 6
- ES6
- ECMAScript 2015
- JavaScript 2015

ECMASCRIPT 2015

- Major improvement (?) over ES5
- Language spec has almost 600 pages (ES 5.1: 245)
- Much needed features such as modules and classes
- Useful features like Maps, Sets, Promises or Generators
- Work has started on ECMAScript 2016

GOALS FOR ECMASCRIPT 2015

Among others: make JavaScript better ...

- for complex applications
- for libraries (including the DOM)
- as a target of code generators

HOW TO UPGRADE A WEB LANGUAGE?

JavaScript engines

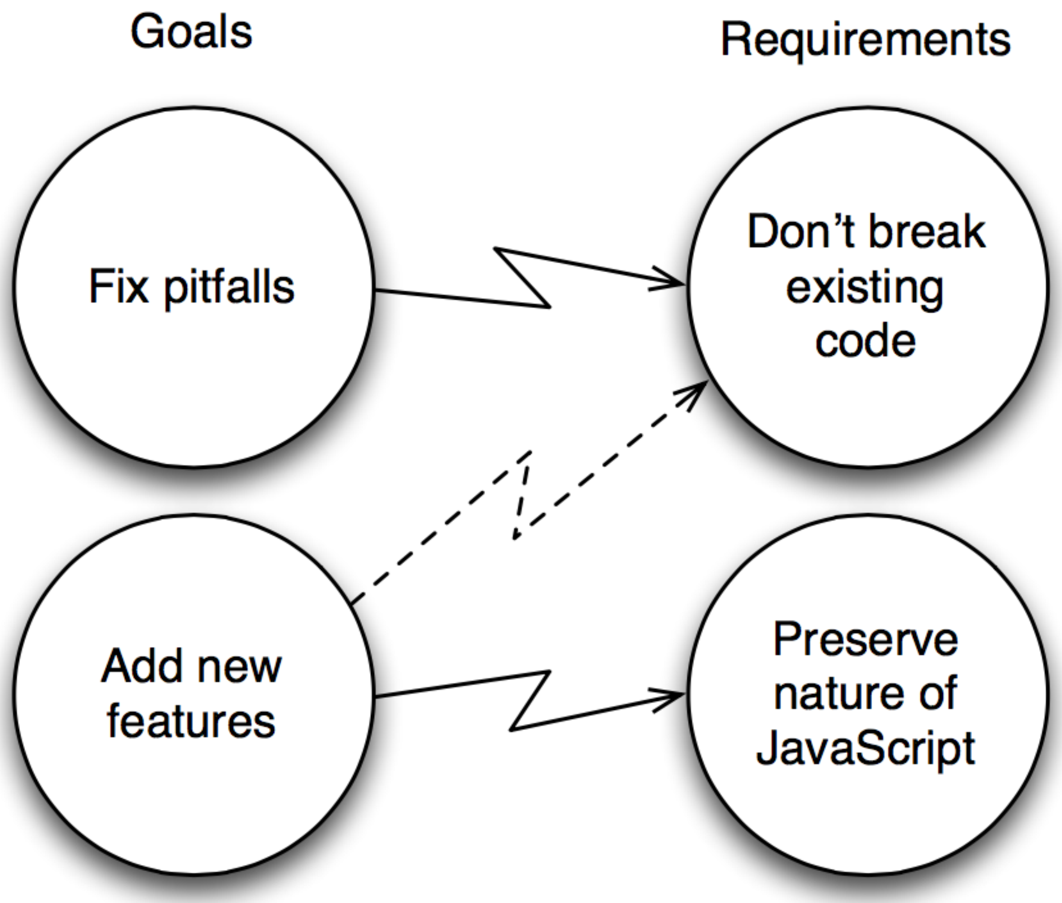
- New versions = forced upgrades
- Must run all existing code
- Consequence: ES6 only adds features

JavaScript code

- Must run on all engines that are in use
- Consequence: wait or compile ECMAScript 6 to ES5

↓ more ↓

GOALS AND REQUIREMENTS



ECMASCRIPT 2015

- Thes slides only cover a small selection of features
- Consult the link list at the end of the slides for more

BASICS

- Better unicode support
- New string methods
- Block bindings
- Destructuring assignment
- Numbers
- Other basics

BETTER UNICODE SUPPORT

- Various improvements
- For example escape sequences for characters with more than 16 Bits

Example:

```
console.log("\u0061");          // "a"
console.log("\u{20BB7}");       // "𠮷"
```

NEW STRING METHODS

- Since the early days of JavaScript: *indexOf*
- New in ES5: *trim*
- New in ES6: *includes*, *startsWith*, *endsWith*

Example:

```
var msg = "Hello world!";

console.log(msg.startsWith("Hello"));        // true
console.log(msg.endsWith("world!"));         // true
console.log(msg.includes("x"));              // false
```

LET DECLARATIONS

- Block level scope

Example:

```
for (var i=0; i < items.length; i++) {
    process(items[i]);
}
// i is still accessible here and is equal to items.length

for (let m=0; m < items.length; m++) {
    process(items[m]);
}
// m is not accessible here
```

CONSTANT DECLARATIONS

- Declaration of constants
- Value cannot be changed once set
- Every *const* variable must be initialized
- Constants are block-level declarations, similar to let

Example:

```
// Valid constant
const MAX_ITEMS = 30;
```

DESTRUCTURING ASSIGNMENT

```
var options = {
        repeat: true,
        save: false
    };

// later

var { repeat: localRepeat, save: localSave } = options;

console.log(localRepeat);          // true
console.log(localSave);            // false
```

- Object destructuring
- Array and mixed destructuring

NUMBERS

- *isFinite* and *isNaN* as methods of *Number*
- *parseInt* and *parseFloat* as methods of *Number*
- Several new *Math* methods
- Octal and Binary Literals

Example:

```
var value1 = 0o71;          // 57 in decimal
var value2 = 0b101;         // 5 in decimal
```

NUMBERS

- New methods *Number.isInteger*, *Number.isSafeInteger*
- Constants *Number.MAX_SAFE_INTEGER*, *Number.MIN_SAFE_INTEGER*

Example:

```
var inside = Number.MAX_SAFE_INTEGER,
    outside = inside + 1;

console.log(Number.isInteger(inside));          // true
console.log(Number.isSafeInteger(inside));      // true

console.log(Number.isInteger(outside));         // true
console.log(Number.isSafeInteger(outside));     // false
```

OTHER BASICS

- Repeat method for strings
- Regular expression enhancements
- Object.is() to fix rare problems with ===

↓ more ↓

NEW STRING METHODS

Repeat method for strings

```
// indent using a specified number of spaces
var indent = " ".repeat(size),
    indentLevel = 0;

// whenever you increase the indent
var newIndent = indent.repeat(++indentLevel);
```

REGULAR EXPRESSION CHANGES

- `u` flag for “Unicode”
- `y` (sticky) flag to save the index of the next character after the last match in `lastIndex` attribute
- `flags` property returns the string representation of flags
- Other improvements (cf. specs)

OBJECT.IS()

Only needed in some special cases

```
console.log(+0 == -0);                  // true
console.log(+0 === -0);                 // true
console.log(Object.is(+0, -0));         // false

console.log(NaN == NaN);                // false
console.log(NaN === NaN);               // false
console.log(Object.is(NaN, NaN));       // true
```

FUNCTIONS

- Default parameters
- Rest parameters
- Destructured parameters
- The spread operator
- Arrow functions
- Syntax

DEFAULT PARAMETERS

```
// ES5
function makeRequest(url, timeout, callback) {
    timeout = timeout || 2000;
    callback = callback || function() {};
    // the rest of the function
}

// ES6
function makeRequest(url, timeout = 2000, callback = function() {}) {
    // the rest of the function
}
```

REST PARAMETERS

- Indicated by three dots (...)
- Named parameter becomes an array containing the rest of the parameters
- No other named arguments can follow

Example:

```
function sum(first, ...numbers) {
    let result = first,
        i = 0,
        len = numbers.length;
    while (i < len) {
        result += numbers[i];
        i++;
    }
    return result;
}
```

DESTRUCTURED PARAMETERS

```
// pre ES6
function setCookie(name, value, options) {
    options = options || {};
    var secure = options.secure,
        path = options.path,
        domain = options.domain,
        expires = options.expires;
    // ...
}

// ES6
function setCookie(name, value, { secure, path, domain, expires }) {
    // ...
}
```

THE SPREAD OPERATOR

```
var values = [25, 50, 75, 100];

console.log(Math.max.apply(Math, values));  // pre ES6
console.log(Math.max(...values));           // ES6
```

ARROW FUNCTIONS

- Arrow functions are defined with a new syntax that uses an “arrow” (=>)
- The value of this inside of the function is determined by where the arrow function is defined not where it is used
- Cannot be used as constructors (with new)
- Can’t change *this*
- No arguments object

ARROW FUNCTIONS

```
// ES6
var sum = (num1, num2) => num1 + num2;

// pre ES6
var sum = function(num1, num2) {
    return num1 + num2;
};

// ES6
var doNothing = () => {};

// pre ES6
var doNothing = function() {};
```

ARROW FUNCTIONS

```
let arr = [1, 2, 3];
let squ;

squ = arr.map(function (a) {return a * a});
squ = arr.map(a => a * a);
```

OTHER FUNCTION ENHANCEMENTS

- All functions have an appropriate value for their name property
- Some changes when calling functions with new, new.target
- Functions in blocks are allowed and considered block-level

↓ more ↓

THE NAME PROPERTY

```
var doSomething = function doSomethingElse() { /* ... */ };
var doAnotherThing = function() { /* ... */ };
var person = {
    get firstName() {
        return "Nicholas"
    },
    sayName: function() {
        console.log(this.name);
    }
}
console.log(doSomething.name);       // "doSomethingElse"
console.log(doAnotherThing.name);    // "doAnotherThing"
console.log(person.sayName.name);    // "sayName"
console.log(person.firstName.name); // "get firstName"
```

BLOCK-LEVEL FUNCTIONS

- ES5 strict mode introduced an error when a function declaration was inside of a block
- In ES6, the function is considered a block-level declaration
- It can be accessed and called within the same block

SYMBOLS

- Creating symbols
- Enum-style values
- Property keys

SYMBOLS

A new kind of primitive value – unique IDs:

```
let sym = Symbol();
console.log(typeof sym)  // 'symbol'
```

SYMBOLS: ENUM-STYLE VALUES

```
const COLOR_RED = Symbol();
const COLOR_ORANGE = Symbol();
...

function getComplement(color) {
  switch (color) {
    case COLOR_RED:
      return COLOR_GREEN;
    case COLOR_ORANGE:
      return COLOR_BLUE; ...
    default:
      throw new Exception('Unknown color: '+color);
  }
}
```

SYMBOLS: PROPERTY KEYS

```
let specialMethod = Symbol();
obj[specialMethod] = function (arg) {
  ...
};
obj[specialMethod](123);
```

OBJECTS

- Object Literal Extensions
- Property value shorthands
- Computed property keys
- Other new features

OBJECTS: METHOD DEFINITIONS

```
let obj = {
  myMethod() {
    ...
  }
};

// instead of:
var obj = {
  myMethod: function () {
    ...
  }
};
```

OBJECTS: PROPERTY VALUE SHORTHANDS

```
let x = 4;
let y = 1;
let obj = { x, y };
// Same as { x: x, y: y }
```

OBJECTS: COMPUTED PROPERTY KEYS

```
let propKey = 'hello';
let obj = {
    ['fo'+'o']: 123,
    [propKey]() {
        return 'hi'; },
};
console.log(obj.hello()); // hi
```

OTHER OBJECT FEATURES

- Object Categories
- Object.assign() (similar to jQuery extend)
- Duplicate Object Literal Properties
- Changing Prototypes
- Super References
- Reflection Methods

CLASSES

- Class Declarations
- Subclassing

CLASS DECLARATIONS

```
class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    toString() {
        return '('+this.x+', '+this.y+')';
    }
}
```

↓ ES5 ↓

CLASSES IN ES5

```
function Point(x, y) {
    this.x = x;
    this.y = y;
}
Point.prototype.toString = function () {
    return '('+this.x+', '+this.y+')';
};
```

SUBCLASSING

```
class ColorPoint extends Point {
    constructor(x, y, color) {
        super(x, y);
        this.color = color;
    }
    toString() {
        return this.color+' '+super.toString();
    }
}
```

↓ ES5 ↓

SUBCLASSING IN ES5

```
function ColorPoint(x, y, color) {
    Point.call(this, x, y);
    this.color = color;
}
ColorPoint.prototype = Object.create(Point.prototype);
ColorPoint.prototype.constructor = ColorPoint;
ColorPoint.prototype.toString = function () {
    return this.color+' '+Point.prototype.toString.call(this);
};
```

OTHER CLASS FEATURES

- Class expressions
- Accessor properties
- Static members
- new.target

MODULES

- Basic Exporting and Importing

Other features:

- Exporting and Importing Defaults
- Re-exporting
- Importing Without Bindings

MODULES: BASICS

```
// lib/math.js
let notExported = 'abc';
export function square(x) {
    return x * x;
}
export const MY_CONSTANT = 123;

// main1.js
import {square} from 'lib/math';
console.log(square(3));

// main2.js
import * as math from 'lib/math';
console.log(math.square(3));
```

TEMPLATE STRINGS

- Basic Syntax
- Multiline Strings

Other features:

- Substitutions
- Tagged Templates

TEMPLATE STRINGS

```
// String interpolation
if (x > MAX) {
    throw new Error(
        `At most ${MAX} allowed: ${x}!`
        // 'At most '+MAX+' allowed: '+x+'!'
    );
}

// Multiple lines
let str = `this is
a text with
multiple lines`;
```

COLLECTIONS

- Maps
- Sets

COLLECTIONS: MAPS

```
let map = new Map();
let obj = {};

map.set(obj, 123);
console.log(map.get(obj)); // 123
console.log(map.has(obj)); // true

map.delete(obj);
console.log(map.has(obj)); // false

for (let [key,value] of map) {
    console.log(key, value);
}
```

COLLECTIONS: SETS

```
var items = new Set();
items.add(5);
items.add("5");

console.log(items.has(5));     // true
console.log(items.has(6));     // false

var coll = new Set([1, 2, 3, 4, 5]);

for (let num of coll) {
    console.log(num);
}
```

ITERATORS

- Objects with a certain interface
- Method called next() that returns a result object
- Result has two properties, `value` and `done`

ITERATORS

```
function createIterator(items) {
    var i = 0;
    return {
        next () {
            var done = (i >= items.length);
            var value = !done ? items[i++] : undefined;

            return {
                done: done,
                value: value
            };
        }
    };
}

var iterator = createIterator([1, 2, 3]);
console.log(iterator.next());            // "{ value: 1, done: false }"
console.log(iterator.next());            // "{ value: 2, done: false }"
```

GENERATORS

- Special kind of function that returns an iterator
- Indicated by inserting a `*` after the function keyword
- The yield keyword is used to specify the return values

GENERATORS

```
// generator
function *createIterator() {
    yield 1;
    yield 2;
    yield 3;
}

// generators are called like regular functions but return an iterator
let iterator = createIterator();

for (let i of iterator) {
    console.log(i);
}
```

PROMISES

```
import {readFile} from 'fs';

function readFilePromisified(filename) {
    return new Promise(
        function (resolve, reject) {
            readFile(filename, { encoding: 'utf8' },
                (error, data) => {
                    if (error) { reject(error); }
                    resolve(data);
                });
        });
}

readFilePromisified(process.argv[2])
.then(text => {
    console.log(text);
})
.catch(error => {
    console.log(error);
});
```

ECMASCRIPT 6 TODAY

Compile ES6 to ES5

- Babel (babeljs.io)
- Google Traceur (github.com/google/traceur-compiler)
- TypeScript (typescriptlang.org): ECMAScript 6 plus (optional) type annotations

READING MATERIAL, SOURCES

READING MATERIAL

Repetition of Web basics

- Learn to Code HTML & CSS, Shay Howe
  learn.shayhowe.com/html-css/
- Chapter 1 „Basic JavaScript“ of the book „Speaking JavaScript“, Axel Rauschmayer
  speakingjs.com/es5/ch01.html
- A guide to the basics of jQuery
  jqfundamentals.com

READING MATERIAL

- A Complete Guide to Flexbox, CSS-Tricks
  css-tricks.com/snippets/css/a-guide-to-flexbox/
- Learn ES2015, A detailed overview of ECMAScript 6 features
  babeljs.io/docs/learn-es2015/

SOURCES: FLEXBOX

- What The Flexbox?! Video Tutorial by Wes Bos
  flexbox.io/
- A Complete Guide to Flexbox, CSS-Tricks
  css-tricks.com/snippets/css/a-guide-to-flexbox/
- Old Flexbox and New Flexbox, CSS Tricks
  css-tricks.com/old-flexbox-and-new-flexbox/
- Mozilla Developer Network: Using CSS flexible boxes
  developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes
- Autoprefixer on GitHub
  autoprefixer.github.io
- Mozilla Developer Network: CSS Length Units
  developer.mozilla.org/en-US/docs/Web/CSS/length

SOURCES: ECMASCRIPT 6

- ECMAScript 2015 Language Specification
  ecma-international.org/ecma-262/6.0/index.html
- Understanding ECMAScript 6
  https://leanpub.com/understandinges6
- Using ECMAScript 6 today, Rolling Scopes Conference, 2015, Slides:
  speakerdeck.com/rauschma
- Exploring ES6: Upgrade to the next version of JavaScript, Dr. Axel Rauschmayer
  exploringjs.com
- ECMAScript compatibility table
  kangax.github.io/compat-table/es6/
- ECMAScript 6 Features
  github.com/lukehoban/es6features