Javascript Cheatsheet

Inside the body goes <script></script>, as opposed to CSS in the head!

8 different data types: undefined, null, boolean, string, symbol, bigint, number, and object.

Variables allow dynamic data manipulation, declared like so:
var ourFirstName; *or more commonly* var ourFirstName = 25;

A method is a function which is the property of an object.

Algorithmic Scripting

Delete varx *similar to return (except the output or lack of!)*

Const names = [John, bob, Mary, Joe];

For (let I = 0; I < names.length; I++) {
console.log(names[i] + i); *this equals John bob Mary Joe*
If (I ===2) break;} *this equals John bob Mary*

For(name of names) Console.log(name);

Let I = 0;
While (I < 10) {
I++;
If (I == 5) continue; *omits 5!*
console.log(I);
}

string() = converts to string.
Object.keys() = converts keys of an object to string

Number.isInteger (x / y)

items.ForEach((item) => {
Console.log(price)})

Const hasInexpensiveItems = items.some((item) => {return item.price <= 100 }) *items.every as well*

Const total = items.reduce((currentTotal, item) => {
Return item.price + current Total}, 0) *beginning at 0, totals the array to a sum*
console.log(total)

Const includesTwo = items.includes(2)

Creating Objects

```
function Dog(name, color, numLegs)
{
   this.name = name;
   this.color = color;
   Dog.prototype.numLegs = 4; this is shared among the dogs
}

let terrier = new Dog('Ben', ‘Mauve')
```

New objects created in this way are called ‘instances’ of their ‘constructor’ (the function!)

You can compare instances to constructors: *duck* instanceof *bird***;**
———

```
let canary = new Bird("Tweety");
let ownProps = [];

for (let property in canary) {
  if(duck.hasOwnProperty(property)) {
    ownProps.push(property);
  }
}
```

Creates array of Tweety’s *own* properties: *those properties separately copied on the object*.
———

```
let prototypeProps = []
let ownProps = []

For (let property in duck) {
If (duck.hasOwnProperty(property)) {
ownProps.push(property);
}
else {
prototypeProps.push(property);
 }
}
```

Creates array of both *own* and *prototype* properties for a given object, such as *‘duck’*.

———

```
bird.prototype = {
Constructor: bird,
numLegs: 2,
Eat: function() {
console.log(’is eating’},
Describe: function() {
Console.log(this.name + ‘ is an avian animal.’)
}
};
```

Creates a prototype object with prototype properties most efficiently

— — —

```
Dog.prototype.isPrototypeOf(beagle)
```

Establishes the inherited prototype of an object with a true or false statement

— — —

```
Object.prototype.isPrototypeOf(Dog.prototype);
```
Object is the super prototype of JavaScript; it is the ultimate prototype of all objects.

— — —

```
Animal.prototype = {
eat: function() {
   console.log("nom nom nom");
  }
}

cat = Object.create(Animal.prototype);
cat.eat() prints 'nom nom nom';
Cat instanceof Animal; prints true
```

Creates a supertype and sets inheritance among objects of this prototype, simplifying code.

— — —

```
cat.prototype = Object.create(Animal.prototype)
cat.prototype.constructor = Cat;
```

Propagates inheritance from one constructor prototype to another, simplifying code.

— — —

```
Dog.prototype.bark = function() {
console.log ('Woof!') };
```

— — —

```
Animal.prototype = {
eat: function() {
   console.log("nom nom nom");
  }
}

cat = Object.create(Animal.prototype);
cat.prototype.constructor = Cat;
Cat.prototype.eat = function () {
Return 'Finds meals like a savage, wild beast.'
};
```

— — —

```
Let flyMixin = function(obj) {
obj.fly = function () {
console.log('Flying, woosh!') }
```

```
};


flyMixin(plane);
flyMixin(bird);
```

Allows for methodological approach to assigning properties ie both planes and birds fly!

— — —

```
Function Bird() {
let hatchedEgg = 10;  this is now a private variable ie cannot be changed outside the function

this.getHatchedEggCount = function ()
{return hatchedEgg};  a function can always access the context of its creation, this is 'closure'

Let ducky = new Bird();
Ducky.getHatchedEggCount(); returns 10
```

A method that can access a private property is a privileged method

— — —

```
function makeNest() {
  console.log("A cozy nest is ready");
}

makeNest();
```

But instead…

```
(function() {
  console.log("A cozy nest is ready");
})();
```

This transformation invokes an anonymous function immediately upon declaration

Functional Programming

```
function sortingorder(arr) {
return arr.sort (function (a, b) {
return
a - b sorts ascending order
b - a sorts descending order
a === b ? 0 : a < b ? -1 : 1 sorts alphabetically
a === b ? 0 : a < b ? 1 : -1 sorts reverse-alphabetically
})}
```