Tutorial 4: Objects, Constructors and Prototypes

We'll have a deeper look into what is meant by an Object in Javascript, and from there get a better understanding of Object-Oriented Programming as it should be done under the constraints of Javascript.

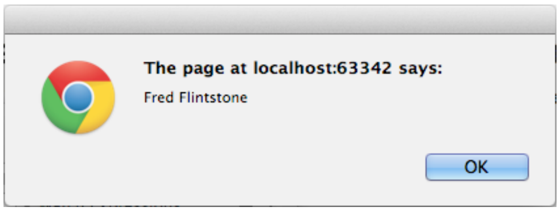
Objects and Classes

Unlike most other languages you will meet, Javascript does not contain the notion of a Class. In, e.g. Java, you define a Class so that you can set up the attributes and behaviour of a whole group of objects. The class says what data every object of a type should contain, and how every object of the type should behave.

Javascript is a Prototypal language, meaning that objects can be created either as unique definitions (one per object), or based on a specific existing object (a prototype). For example:

```
var person = {
    first: "Fred",
    last: "Flintstone"
}

var employee = {};
employee.__proto__ = person;
alert(employee.first + " " + employee.last);
```



The page at localhost:63342 says:

Fred Flintstone

Here we have made a person object and then made a new employee object inherit from the person's prototype (__proto__) – this means that the employee object has all of the properties previously defined for the person type (first and last names).

Exercises 1:

Start these exercises by creating a new HTML/Javascript project (e.g. in WebStorm), adding a HTML file and a JS file to it (e.g. OOP.html and OOP.js) and adding a <script> tag to the HTML file that references the JS file. If you don't have access to a laptop in the tutorial, you should attempt these exercises at home, but do your preparation in the tutorial class where you can ask questions.

Now copy the above code into the JS file and open the HTML file in a browser.

a) Add a property – dob (date of birth) to the ***person*** object, assign a value to it and amend the alert() statement so that it shows the value of the dob

b) Add properties jobTitle and salary to the ***employee*** object – amend the alert() statement to that it displays these values

c) Verify in a console window that the person object does not have jobTitle or salary fields

d) Use the dir() function in the console window to find out the composition of the person and employee objects (e.g. dir(person))

e) Change the first and last properties of the employee object by entering new values in a console window (e.g. employee.first = "Barney")

f)Again use dir() to find out the composition of the employee object

All Objects in Javascript have a prototype that can be shared with other objects. A major use of a prototype is to add functions that can be shared among whole groups of objects. However, if we want to create functions that are shared among objects, we first need to make sure in some way that objects are compatible with each other. Constructors are used to do this.

Constructors

Although the language makes it easy to build objects without defining classes (not possible in languages like Java, C++, Python etc.), the Javascript approach does make it more awkward to define objects that have a set template. For example, our employee object has first, last, jobTitle and salary members. If we were to make a mistake while assigning a new value to one of these (e.g. employee.slaary = 45000), we would end up with an object that did not behave as expected.

Constructors are functions that apply a recipe to creating an object. Instead of defining the object over a number of statements, a constructor function can be used to make creating an object the work of a single statement:

Note – capitalized. By convention this is the only type of identifier that should start with a capital letter

```
var Person = function(first, last, dobString) {
   this.first = first;
   this.last = last;
   this.dob = new Date(dobString);
};
```

"this" refers to whichever Person object this is called for.

e.g. in:
  p = new Person(…) ;
  alert(p.dob);

'this' would refer to the object 'p'

Now we can create a Person object using a single statement, like p = new Person("Fred", "Flintstone", "1980-06-15");. Note that here, we've taken advantage of the function so that we can pass a simple string giving the date of birth – this is converted into a Date object inside the constructor). Note also that ***convention*** dictates that the constructor function name should start with a capital letter (the only type of identifier in Javascript that should do this), and that to create an object we need to add the ***new*** keyword.

We can now define functions that apply to ALL objects created by this constructor:

```
Person.prototype.name = function () {
   return this.first + " " + this.last;
}
Person.prototype.age = function () {
   return new Date().getFullYear() - this.dob.getFullYear();
}
```

Exercises 2:

a) In a console window, create a new Person object and test the name() and age() functions. Note that to call these, you must include the parenthesis () after the function name – e.g. p.name()

b) Add a new prototype function to the Person type that puts a message (e.g. "Joe Bloggs is 25 years old") into an alert box. You can call it message().

c) The age() prototype function is not always accurate. If someone's birthday is before today in this year, his or her age will be reported as 1 year older than it is. See if you can re-write the age() function so that it is always correct (hint: for a Date object, .getMonth() indicates the month number and .getDate() gives the day in the month)

One useful aspect of using constructors is that we can create Classes (i.e. groups of objects that share a type) that inherit the capabilities of existing classes. A simple code pattern is used to do this:

```
var Employee = function(first, last, dobString, jobTitle, salary) {
   Person.apply(this, arguments);
   this.jobTitle = jobTitle;
   this.salary = salary;
   };
Employee.prototype = new Person();
```

Note that there are two stages to this type of inheritance – Person.apply() passes the constructor arguments (first, last etc.) to the Person constructor *so these are applied to the new Employee object*. The ***this*** parameter refers to the new Employee object. Since the Person constructor only knows about first, last and dobString, it will only assign the values of these to its internal members. This step makes sure that the Employee's *inner person object* is initialized with data, and note that the remainder of the Employee constructor goes on to assign the remaining initialization arguments (jobTitle, salary) to the Employee-specific fields.

The second step is to make the Employee type inherit all prototype members from the Person type. Generally, this will be any functions that have been assigned to the Person prototype

Exercises 3:

a) Add a new prototype function to the Employee type – pay() – which will return the employee's monthly pay rate (salary divided by 12)

b) Create a new Constructor – Student – that inherits from Person and adds member fields for course and year (a number – 1 to 4). Add prototype functions to the Student type that calculate the students' year of graduation (assume this is after the 4th year) and the students' age at graduation (their current age added to the number of years left to do)