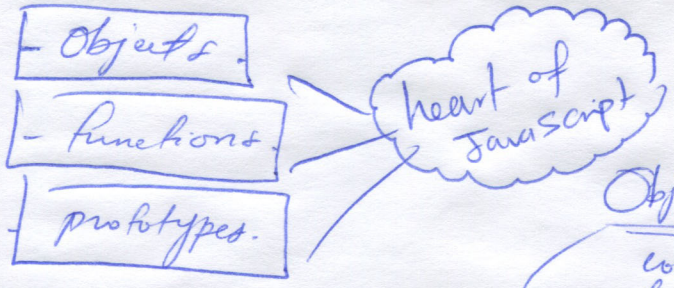
Functions, Objects, prototypes

- Important concepts in JavaScript.



Object : function.

constructor function

'Object' constructor is a function (funny name)

- Object:
  - map of property and values.
  - var x = {};
  - var x = new Object(); → same thing

Extra...

- x : object. x.name = 'Amin';
- x.name; // → 'Amin';
- x.hello = function(){};
  x.hello(); // → call function.

- Object.

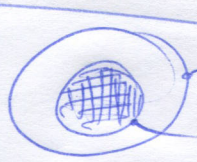
- Functions

  - function is an object with superpowers:
    + Call a function
    + piece of functionality
    + Object:
      - assign attributes
      - methods, ...

```
function Hello(){};
Hello.name = '___';
Hello.getName = function(){ ..... };
Hello();
Hello.name ... // -> '___'
Hello.getName(); // calling a method
```

} static properties and methods (not instance)

Object / function



objects with superpowers

+ function to create ~~obj~~ plain objects:

  . Constructor function.

  . var t1 = {};
    var t2 = {};
    var t3 = {};
    } a function to do this.

Constructor Function

a function object to create other objects.

+ Built-in constructors:
  - Object
  - Function
  - Number
  - String
  - Boolean

// Object.keys()
// Object.create()
etc

// → JavaScript does use behind the scenes.

Funny thing:

Function constructor that creates functions

Function.constructor
=
Function

we use static methods

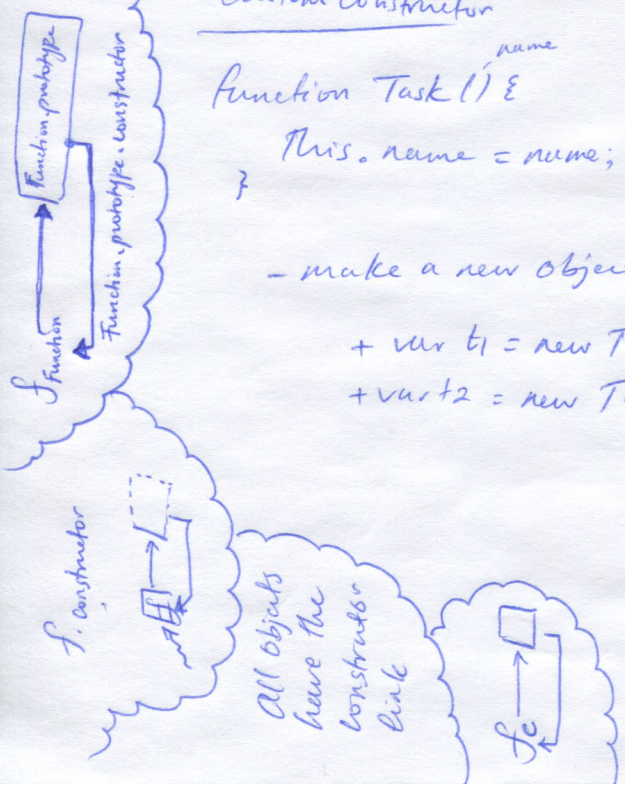
* Number.isNaN etc

+ Custom Constructor

```
Function Task(name) {
    this.name = name;
}
```

- make a new object (instance)

```
+ var t1 = new Task();
+ var t2 = new Task('clean house');
```



Prototypes

- prototypes are objects
- play 'class' role

Java, C#, C++, ... → class oriented

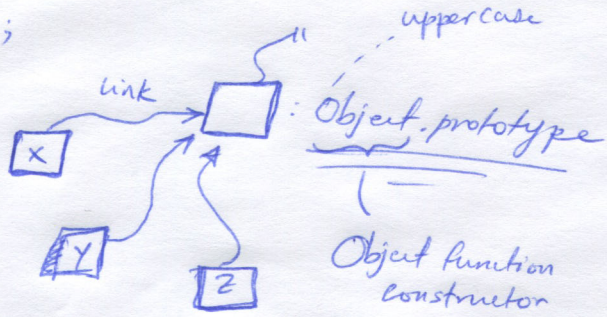
- Delegation Vs Class taxonomies

prototype is the analogous to classes to class-oriented languages BUT more powerful

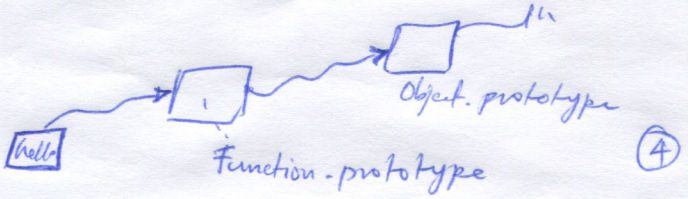
- two Rules:
  - + cloning object
  - + Delegation

- Fundamentally:
  - + Every object has a link
    - link: pointer: reference to the prototype

- var x = { };
  var y = { };
  var z = { };



- function hello() { }

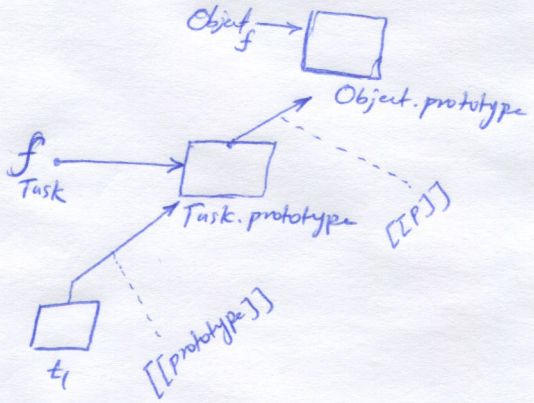


Custom Creation with Constructor Functions

var t_1 = new Task('clean');

4 things happen:

1) a new object gets created
2) this new object gets linked to the function's prototype
3) 'this' gets bound to the new object
4) 'this' is returned.



function Task(){}

var t_1 = new Task();

Object.getPrototypeOf(t_1) === Task.prototype.

(returns the pointer to by [[prototype]] : the internal property of an object.

Using Object.create

```
var p = {
    hello: function() {
        return 'hello';
    }
};

var t1 = Object.create(p);
```

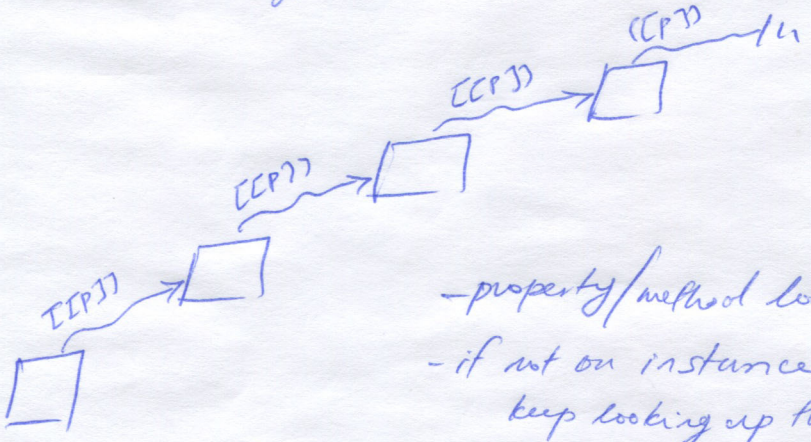
'p' will be used as the prototype object for instance 't1';

```
t1.hello(); // -> 'hello'

Object.getPrototypeOf(t1) === p; // -> true
```

Prototype look up (Chain traversal)

- [[P]]: [[prototype]] : internal property of every object that points to the prototype object.



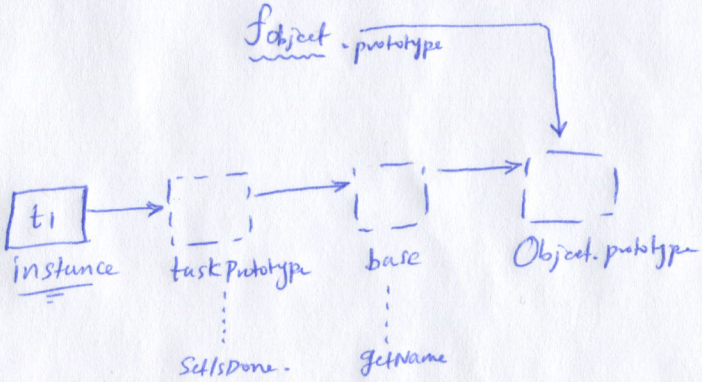
- property/method look up
- if not on instance keep looking up the links (chain) until find it or not
  ↳ return undefined.

ex:
x.constructor

Prototype Chaining

- Object.create:

```
var base = {
    getName() { return this.name; }
};
var taskPrototype = Object.create(base);
Object.assign(taskPrototype, {
    setIsDone() { this.isDone = true; }
});
var t1 = Object.create(taskPrototype);
t1.name = 'clean room';
t1.isDone = false;
⋮
t1.setIsDone(); // → calls method using taskPrototype
t1.getName(); // → 'clean room'
```



Resources (no particular Order)

① - You Don't Know Javascript by Kyle Simpson

② - Speaking Javascript: Dr Axel ?

③ - JavaScript good parts: Dough Crockford

④ - Eric Elliot: pillars of Javascript progrmng

etc ...

Performance: quick note

Object.create VS new.

- in practice, for most apps, not noticeable even if you are drawing (creating) a thousand objects per second.
- For extremely performance sensitive applications ~~new~~ using new and the constructor function can help the compiler do some optimization