

# AJAX

Asynchronous JavaScript and XML

# Dagens lektion

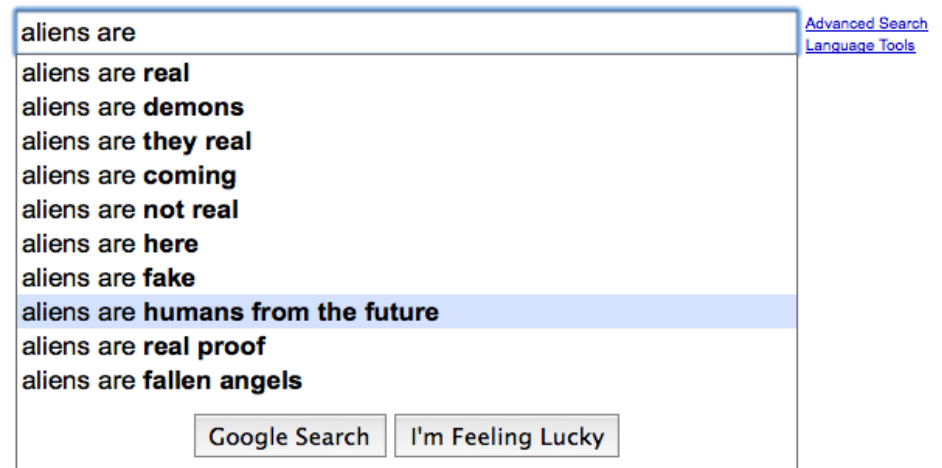
- Varför AJAX?
- Vad är AJAX?
- AJAX i praktiken (med kodexempel)
- JSON
  - Skapa AJAX-anrop med JavaScript
  - Generera JSON med PHP
  - Hantera JSON med JavaScript
- Ansluta till API med JavaScript
- Uppgift: Flickr API
- Hemläxa: <http://www.codecademy.com/tracks/youtube>

# Varför AJAX?

AJAX tillåter oss att uppdatera delar av en sida utan att ladda om hela sidan = snabba, dynamiska hemsidor som kräver mindre internettrafik. WIN.

Exempel: Googles sökförslag

Klassiska hemsidor (som inte använder AJAX) måste ladda om hela sidan för att en del av innehållet ska kunna bytas ut.

A screenshot of a Google search interface. The search bar contains the text "aliens are". Below the search bar, a list of suggestions is displayed: "aliens are real", "aliens are demons", "aliens are they real", "aliens are coming", "aliens are not real", "aliens are here", "aliens are fake", "aliens are humans from the future", "aliens are real proof", and "aliens are fallen angels". The suggestion "aliens are humans from the future" is highlighted with a light blue background. To the right of the suggestions list, there are two links: "Advanced Search" and "Language Tools". At the bottom of the search bar, there are two buttons: "Google Search" and "I'm Feeling Lucky".

aliens are

aliens are **real**

aliens are **demons**

aliens are **they real**

aliens are **coming**

aliens are **not real**

aliens are **here**

aliens are **fake**

aliens are **humans from the future**

aliens are **real proof**

aliens are **fallen angels**

[Advanced Search](#)  
[Language Tools](#)

Google Search I'm Feeling Lucky

# Vad är AJAX?

~~Det är ett fotbollslag från Amsterdam!~~

~~Det är en rengöringsprodukt!~~

~~Det är ett programmeringsspråk!~~

Det är en kommunikationsteknik som använder befintliga standarder och programmeringsspråk:

- XMLHttpRequest object
- JavaScript / DOM
- JSON / XML

# XMLHttpRequest object (XHR)

Används för att asynkront växla information (skicka och ta emot data) med en server med HTTP requests .

## **HTTP request**

En förfrågan från en klient till en server via HTTP.

Exempel: När man går in på [www.facebook.com](http://www.facebook.com) skickar man en förfrågan till Facebooks server för att få ladda Facebooks startsida. Servern svarar genom att skicka HTML, CSS, JS till användarens webbläsare.

All interaktion mellan klient och server på webben bygger på HTTP requests. XHR tillåter både POST och GET requests.

# XMLHttpRequest object (XHR)

## **Response**

Den data som en server skickar tillbaka till klienten vid en HTTP request.

Om du går in på (efterfrågar) en hemsida i din webbläsare skickar servern tillbaka HTML, CSS och JS-filer. Detta är en typ av respons.

När man använder AJAX efterfrågar man sällan hela sidor, eftersom att man ofta bara ska uppdatera en del av en sida.

Det man vill ha i retur är istället "ren data" som man sedan kan hantera i JavaScript. Denna data kan vara formaterad på olika sätt, de vanligaste formateringarna är JSON och XML.

# JavaScript / DOM

Används för att hantera information i klienten (webbläsaren).

Med JavaScript kan man skicka XHR object (dvs HTTP requests) till en server. För enkelhetens skull kallar vi dessa **AJAX-anrop**.

AJAX-anrop har en callback som avfyras när servern svarar genom att skicka tillbaka data (response). I denna callback kan man hanterar den mottagna datan och uppdatera innehållet på sidan.

# JavaScript / DOM

## **Asynkron kommunikation**

Under tiden servern genererar och skickar data är klienten fri att utföra andra operationer, dvs att ett AJAX-anropet körs i bakgrunden och inte påverkar klientens prestanda.

## **Synkron kommunikation**

Motsatsen till asynkron. All exekvering av kod stannar tills AJAX-anropet fått svar från servern. **Rekommenderas inte** eftersom att sidan “hänger sig” under tiden servern hanterar förfrågan.



# JSON / XML

Format för att skicka data mellan server och klient.

Främst förekommande idag är JSON, vilket vi kommer att fokusera på. De flesta moderna API returnerar data i form av JSON.

JSON följer samma struktur som JavaScript objects, dvs key-value pairs. Därav är det enkelt att använda i kombination med JavaScript.

Mer om JSON senare.

# AJAX

## Sammanfattning

All växling av information mellan server och klient på webbsidor sker genom HTTP requests; man laddar sidor, skickar formulär osv.

Den efterfrågade servern genererar en respons som skickas till klienten. Kort och gott: **förfrågan och svar**.

Traditionellt sett var man tvungen att ladda om en sida för att skicka en ny HTTP request. Men med hjälp av JavaScript och AJAX kan vi skicka HTTP requests till en server utan att ladda om sidan.

Datan som skickas tillbaka från servern kan sedan hanteras i JavaScript med hjälp av en callback-funktion.

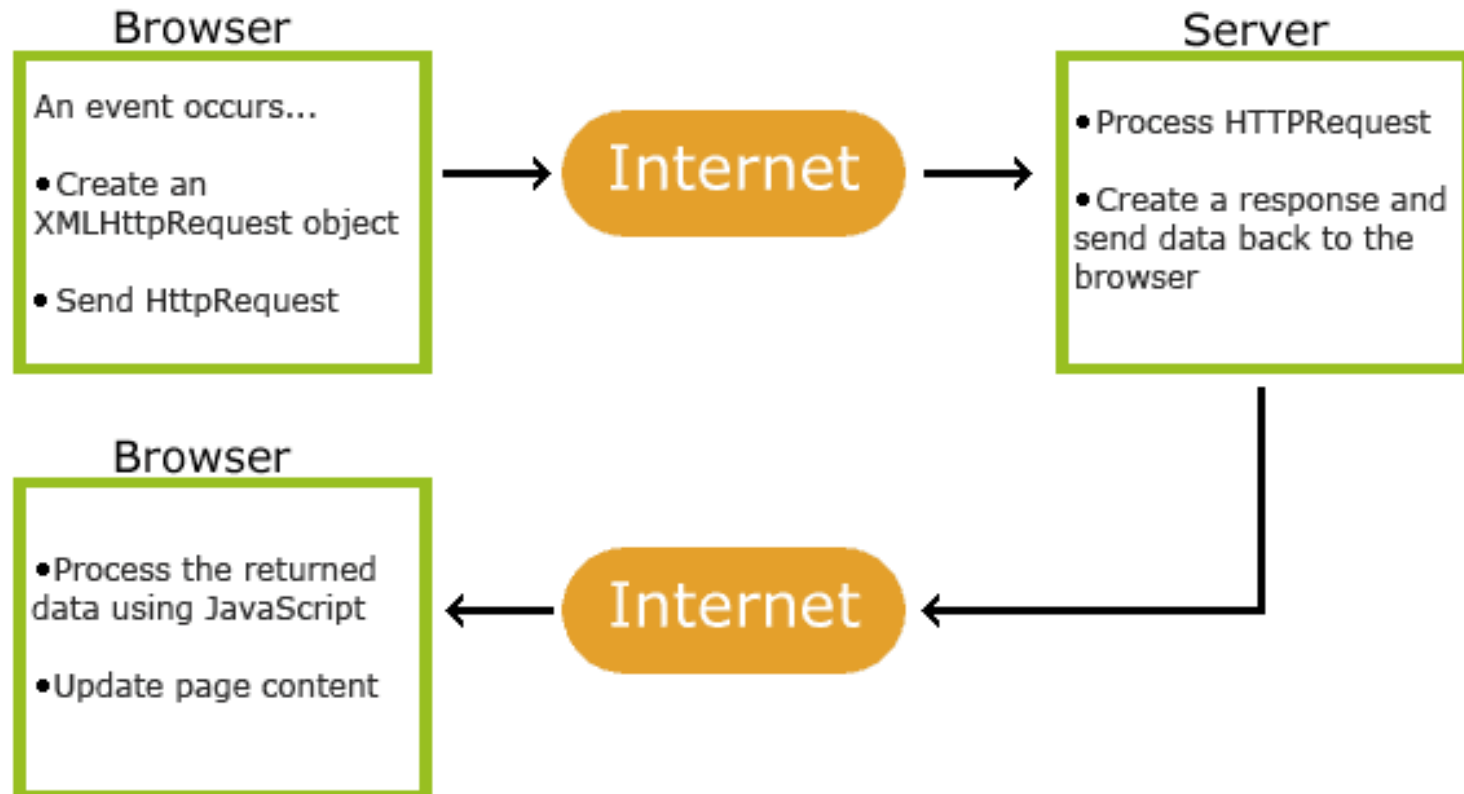
# Sammanfattning

AJAX används främst för att hantera små mängder data och man använder ofta JSON för att formatera datan.

Eftersom att JSON bygger på JavaScript objects kan JavaScript enkelt hantera informationen och applicera den på sidan.

Hela sidor (HTML, CSS eller andra typer av filer) kan också tas emot via AJAX och appliceras på den befintliga sidan, men detta förekommer inte lika ofta.

# Sammanfattning



# AJAX i praktiken (feat. jQuery)

Att skapa ett AJAX-anrop i jQuery är enklare än att göra det direkt i JavaScript. Man använder metoden `$.ajax()`

Det finns ett flertal shorthand-metoder som bygger på denna metod, men vi kommer att fokusera på originalet.

Metoden tar ett JavaScript object (key-value pairs) innehållande egenskaper som argument och följer därför denna struktur:

```
$.ajax({  
  key: value,  
  key: value,  
  ...  
});
```

# AJAX i praktiken (feat. jQuery)

Ett AJAX-anrop kräver egentligen bara en URL, men man bör använda alla dessa egenskaper:

- **type** - typ av anrop; GET eller POST
- **url** - URL dit anropet ska skickas
- **data** - de parametrar som ska skickas med anropet (tänk inputs i ett formulär)
- **dataType** - förväntat format på datan som skickas tillbaka från servern (text/json/xml)
- **success** - callback-funktion som körs om anropet lyckas
- **error** - callback-funktion som körs om anropet misslyckas
- **[async]** - true för asynkront, false för synkront (default value är true)

# AJAX i praktiken (feat. jQuery)

Ett enkelt anrop skulle kunna se ut så här:

```
$.ajax({  
  type: 'GET',  
  url: 'myurl.php',  
  data: {  
    param1: 'value1',  
    param2: 'value2'  
  },  
  dataType: 'text',  
  success: function(data) {  
    // do something with data  
  },  
  error: function() {  
    alert('There was a problem!');  
  }  
});
```

Notera *dataType: 'text'*.

Detta innebär att responsen från servern förväntas vara ren text och inte formaterad på något vis.

# AJAX i praktiken (feat. jQuery)

Praktiskt exempel: [https://github.com/godiagonal/ajax\\_demo/tree/master/text](https://github.com/godiagonal/ajax_demo/tree/master/text)

## **Beskrivning**

När man klickar på knappen skickar webbläsaren ett AJAX-anrop till en PHP-sida på servern som då skickar tillbaka en text. Texten placeras sedan i en div med hjälp av JavaScript.

Kopiera och testkör gärna själva!



# AJAX med JSON

# JSON

Föregående exempel hanterade data i form av ren text. Detta är inget bra sätt att skicka information eftersom att text inte har någon bestämt struktur!

För att skicka strukturerad information använder man istället ett format kallat JSON (kort för *JavaScript Object Notation*).

JSON bygger, som namnet antyder, på samma struktur som JavaScript objects vilket gör det enkelt att arbeta med i kombination med JavaScript och AJAX.

Hädanefter kommer jag relatera till JavaScript objects som "JSO", för enkelhetens skull.

# JSON

Exempel på data i JSON-format:

```
[  
  { "firstname": "Samuel", "lastname": "Johansson" },  
  { "firstname": "Tommy", "lastname": "Testson" },  
  { "firstname": "Anita", "lastname": "Testson" }  
]
```

Oftast ser det mer grötigt ut. Jag har lagt till radbrytningar och mellanslag.

# JSON

Vi ska nu skapa en JavaScript-funktion som anropar en PHP-sida, som i sin tur skickar tillbaka en lista med användare i JSON-format. Därefter ska vi presentera denna lista på vår sida med hjälp av JavaScript.

För enkelhetens skull är ingen databas inblandad.















Ansluta till API med JavaScript

# Viktigt om API

Alla API fungerar på olika sätt och man måste därför följa **API:ets dokumentationen** noga när man utvecklar sin applikation!

Det är inte alltid lätt att veta hur datan man får tillbaka från ett API är strukturerad, trots att man har läst dokumentationen. Dokumentationen kan vara bristfällig.

Här är `console.log();` er bästa vän! Använd det för att lägga in den returnerade datan i webbläsarens konsol så att ni kan analysera den.

# API-nycklar

Av säkerhetsskäl kräver de flesta API en API-nyckel för att man ska kunna ansluta till dem.

Dessa nycklar får man genom att registrera sin applikation på respektive APIs hemsida. Den nyckel man mottar får endast användas av den registrerade applikationen.

Vissa API kräver att man registrerar den domän som applikationen ska ansluta från.

# API med JavaScript-bibliotek

Många API har JavaScript-bibliotek som underlättar requests med AJAX. T ex Google och SoundCloud.

Det är alltid föredraget att använda dessa bibliotek om de finns.

# API med AJAX

För API som inte har bibliotek måste man använda en speciell metod för att kunna komma åt dessa med AJAX.

Detta beror på att AJAX-anrop endast kan komma åt servrar på samma domän som de skickas från på grund av säkerhetsskäl. Detta kallas *same-origin policy*.

Om man kör sin server lokalt kan man alltså bara komma åt sidor på domänen *localhost*.

# API med AJAX

Dataformatet **JSONP** togs fram för att kringgå problemet med AJAX cross-domain requests och används i jQuery tillsammans med egenskapen `crossDomain`.

I övrigt är JSONP identisk med JSON, datastrukturen är densamma.

För att ansluta till ett API måste man lägga till följande i sitt anrop:

```
dataType: "jsonp"  
crossDomain: true
```

Ni kan läsa mer om dessa här: <http://api.jquery.com/jQuery.ajax/>



# Exempel

## Anslut till Flickr's API med AJAX

```
var flickerAPI = "http://api.flickr.com/services/feeds/...."
```

```
$.ajax({  
  type: "GET",  
  url: flickerAPI,  
  data: {  
    tags: "mount rainier",  
    tagmode: "any",  
    format: "json"  
  },  
  dataType: "jsonp",  
  crossDomain: true,  
  success: function (data) {  
    // do something with data  
  },  
  error: function () {  
    alert("Kunde inte ansluta till Flickr!");  
  }  
});
```

# Uppgift: Flickr API

Ni ska skapa en sida där man kan söka bilder på Flickr med hjälp av bildtaggar. Sökresultaten ska visas på sidan.

Extrauppgift: Max 5 bilder får visas.

Till er hjälp har ni en halvfärdig HTML-fil med ett påbörjat AJAX-anrop: [https://github.com/godiagonal/ajax\\_demo/tree/master/flickr](https://github.com/godiagonal/ajax_demo/tree/master/flickr)

Samt Flickr API:ets dokumentation:

[http://www.flickr.com/services/feeds/docs/photos\\_public/](http://www.flickr.com/services/feeds/docs/photos_public/)

**Tips:** Använd `console.log(data);` för att analysera datan ni får från API:et. Tänk på att ni letar efter bildlänkar och att det kommer att finnas flera bilder i resultatet... En array måntro?

Diskutera gärna med varandra!

# Heml äxa: YouTube API

<http://www.codecademy.com/tracks/youtube>