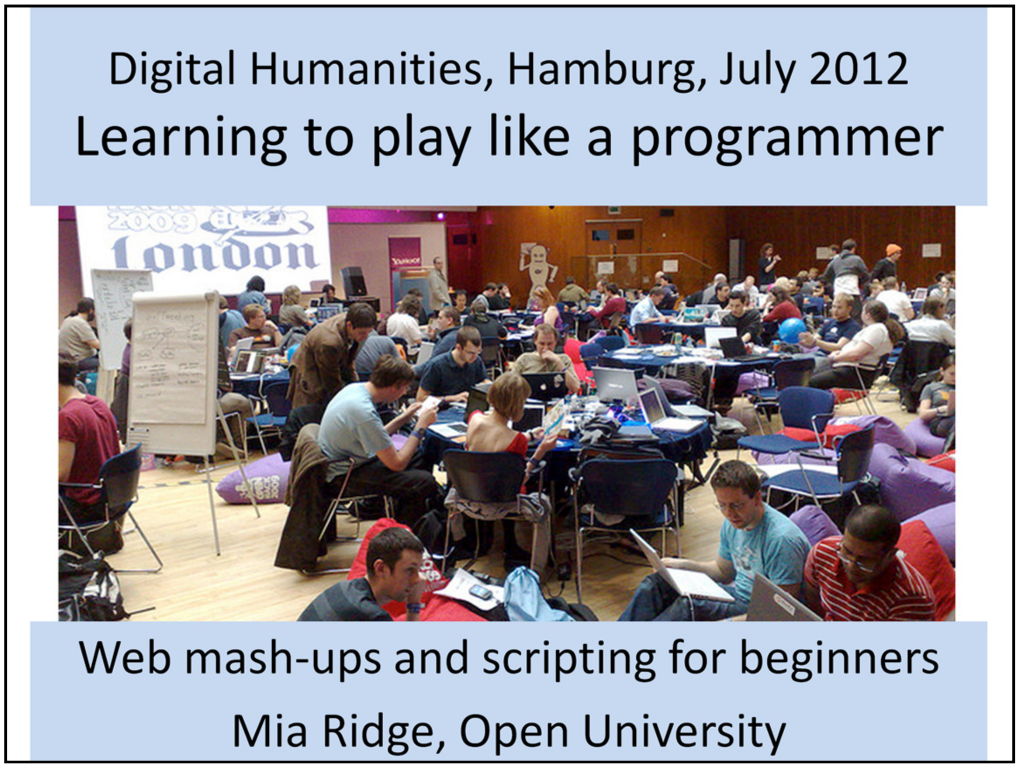


Digital Humanities, Hamburg, July 2012

Learning to play like a programmer

Web mash-ups and scripting for beginners

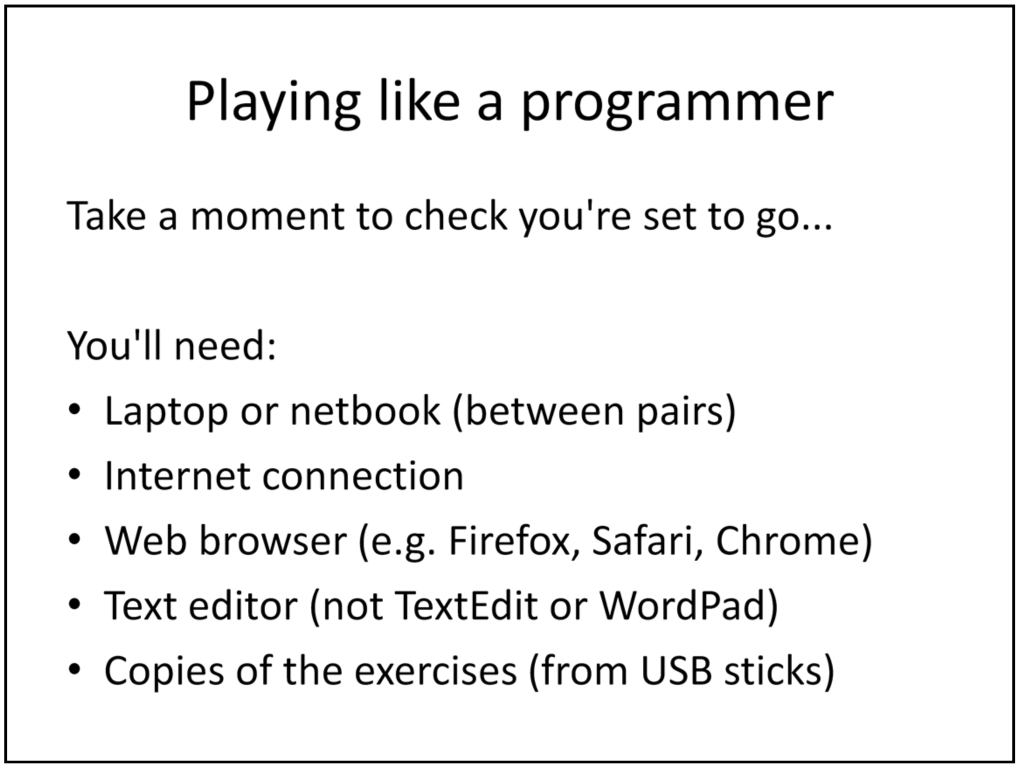
Mia Ridge, Open University

Learning to play like a programmer: web mash-ups and scripting for beginners, Digital Humanities conference, Hamburg July 2012 by Mia Ridge (twitter: @mia_out, blog http://openobjects.blogspot.com/, homepage http://www.miaridge.com/)

These slides should contain enough detail to remind you of what you've learnt in the workshop.

Welcome!

My goal is to get you excited about the possibilities of programming, to help you understand something of how programmers think, to give you some ways to explore data and start to work out how you might want to manipulate it. I've left lots out, but hopefully this small taste will get you interested in playing with more code and exploring computational thinking.



Playing like a programmer

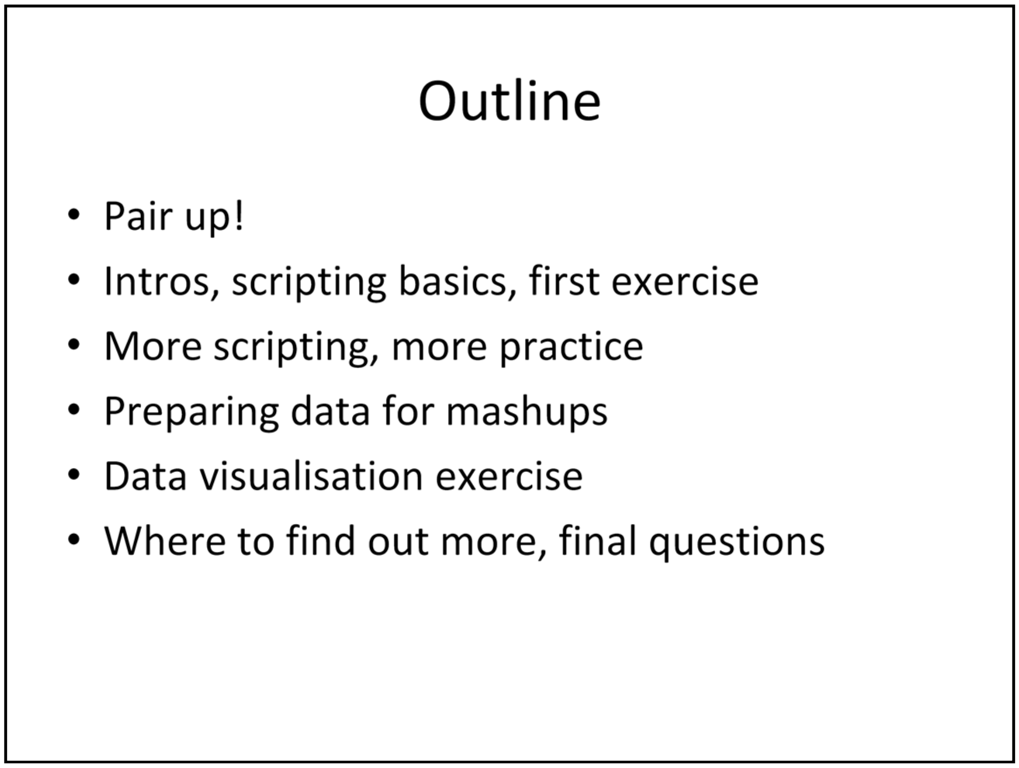
Take a moment to check you're set to go...

You'll need:

- Laptop or netbook (between pairs)
- Internet connection
- Web browser (e.g. Firefox, Safari, Chrome)
- Text editor (not TextEdit or WordPad)
- Copies of the exercises (from USB sticks)

[Keep on screen while people are settling in so they know they're in the right room and can start checking what they've got, and passing out memory sticks with the exercises on them.]

If you're on a Mac, TextEdit won't show you the raw code, you'll need to download another free tool like Smultron (http://sourceforge.net/projects/smultron/ for older Macs, http://www.peterborgapps.com/smultron/ for newer Macs) or TextWrangler (http://www.barebones.com/products/textwrangler/). Notepad will work on PCs, as will WordPad at a pinch but you can also try Notepad++ (http://notepad-plus-plus.org/)

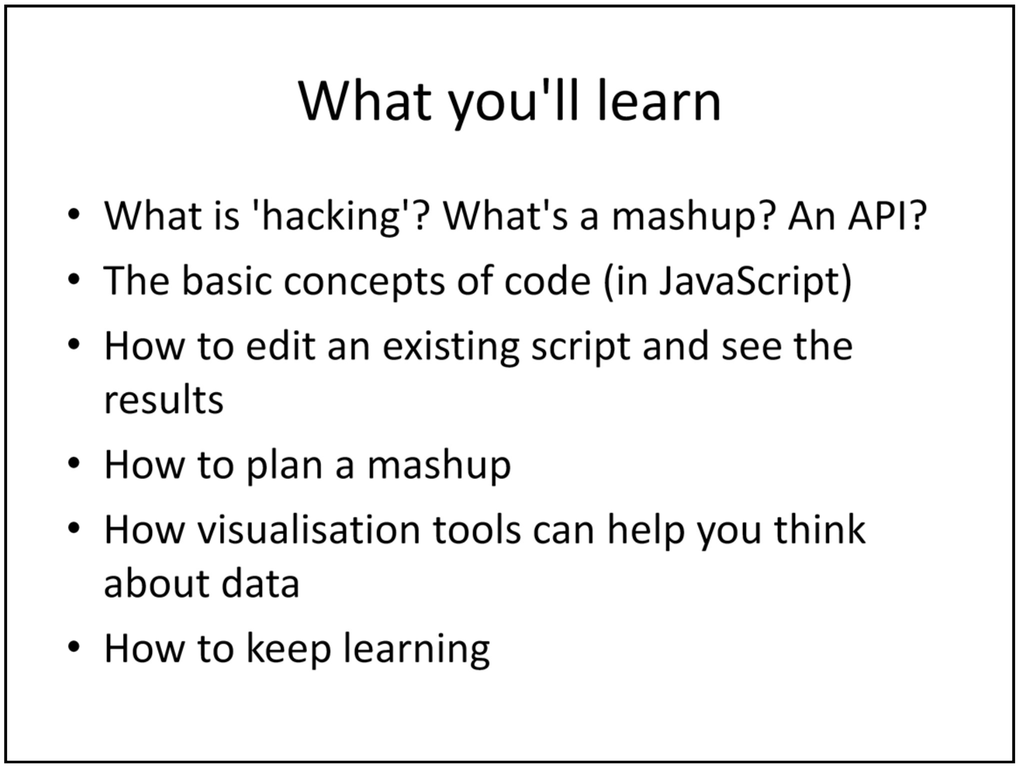


Outline

- Pair up!
- Intros, scripting basics, first exercise
- More scripting, more practice
- Preparing data for mashups
- Data visualisation exercise
- Where to find out more, final questions

Before I get into this, I want to pair you up. Hands up if you've got some previous programming experien

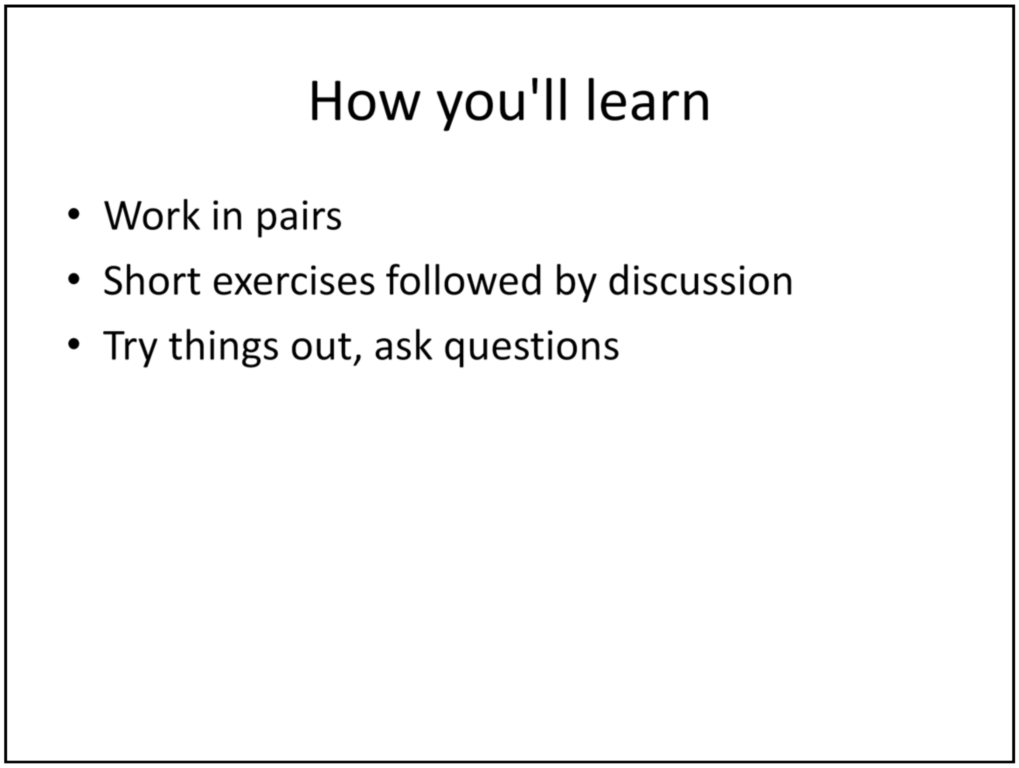
You'll work on exercises together, helping each other figure it out. I want to hear conversation during the



What you'll learn

- What is 'hacking'? What's a mashup? An API?
- The basic concepts of code (in JavaScript)
- How to edit an existing script and see the results
- How to plan a mashup
- How visualisation tools can help you think about data
- How to keep learning

There's a bit of talking to get out of the way first, so that we're all caught up on jargon...



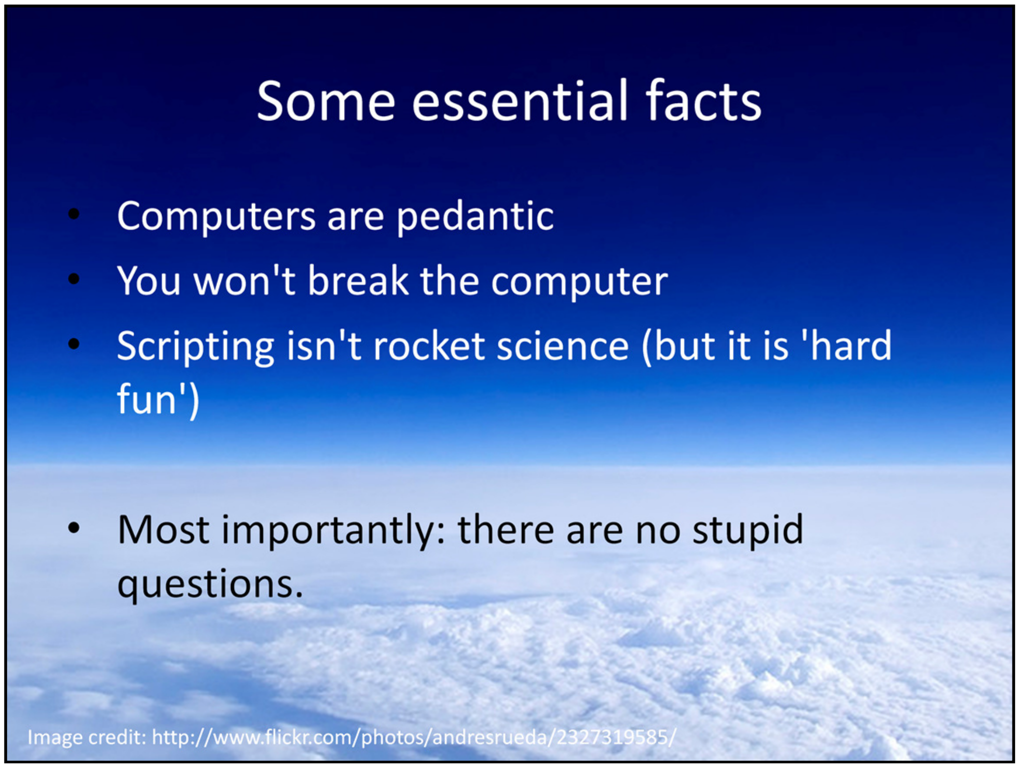
How you'll learn

- Work in pairs
- Short exercises followed by discussion
- Try things out, ask questions

You'll work on exercises together, helping each other figure it out. Ideally you're now paired up with someone with different experience levels to you...

The aim is to consolidate the skills you're learning, to help them stick in your brain. It's hard in the time we have here, but maybe you'll be inspired to find a mentor or partner and keep learning afterwards.

Any questions before we go on?



Some essential facts

- Computers are pedantic
- You won't break the computer
- Scripting isn't rocket science (but it is 'hard fun')

- Most importantly: there are no stupid questions.

Image credit: http://www.flickr.com/photos/andresrueda/2327319585/

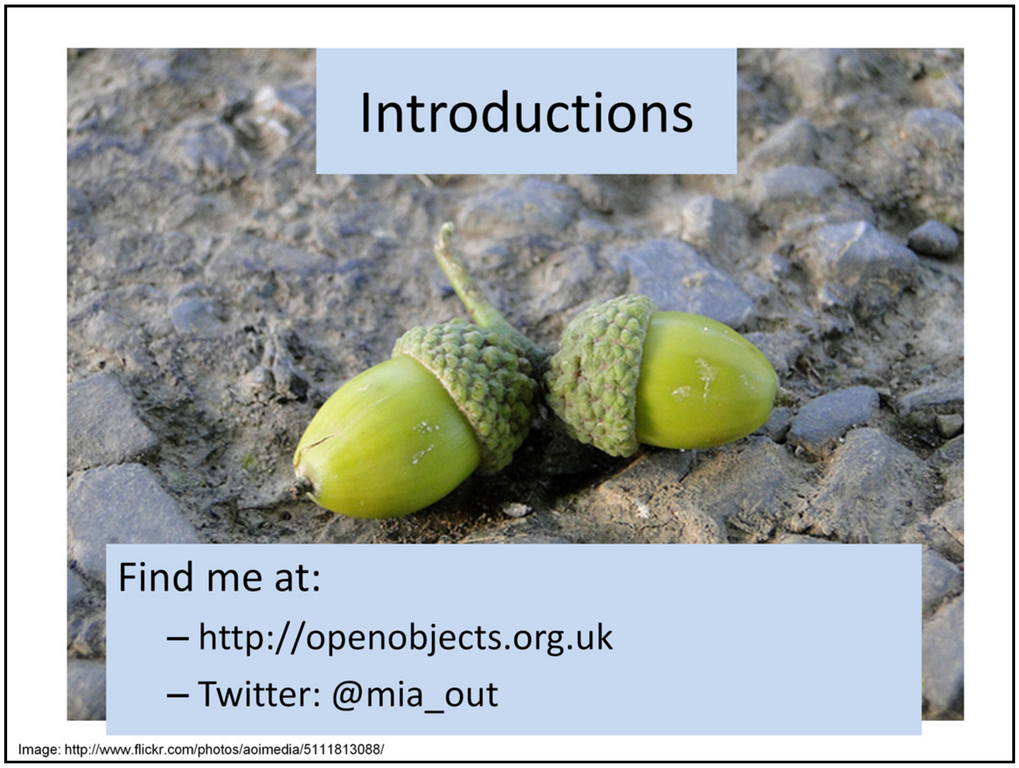
This is the “don't be scared” slide! Computers are really picky about spelling, white space, matching quote marks, how sentences end... Think of your most pedantic friend, and multiply that by 1000. It's like dealing with a grumpy six year old - it might be tricky to negotiate, but it's not going to kill either of you.

Thinking computationally is like cooking a few courses for a fancy dinner party – you learn what needs to be prepped in advance or just before serving, which steps must be done in a particular order and what can be done at any time.

Hard fun – phrase comes from gaming – when something is challenging it's even more rewarding when you finally crack it. A lot of my 'don't be scared' message is aimed at getting you over those first hurdles and into the rewarding stuff. Persistence (or stubbornness) is one of the key characteristics of a good programmer. The process of finding a path through something you're still figuring out is something programmers and researchers have in common.

Computer code is a lot like poetry: it's compact, dense and obscure at first, but you'll start to learn how to unpack it and make it readable.

On behalf of the programmers you work with, I should point out at the start that programming well is a skill and a craft where experience pays - you won't go back knowing how to do their job, but you will understand something of how they think.



Introductions

Find me at:
- http://openobjects.org.uk
- Twitter: @mia_out

Image: http://www.flickr.com/photos/aoimedia/5111813088/

I'm currently a PhD student in digital humanities in department of history at Open University, working on crowdsourcing and spatial history (i.e. participant digitisation and spatially indexed historical materials from Early Modern England). Before that, I worked as a programmer, user experience designer and/or systems analyst for the Science Museum, the Museum of London, Museum Victoria and in the commercial sector from the mid-90s.

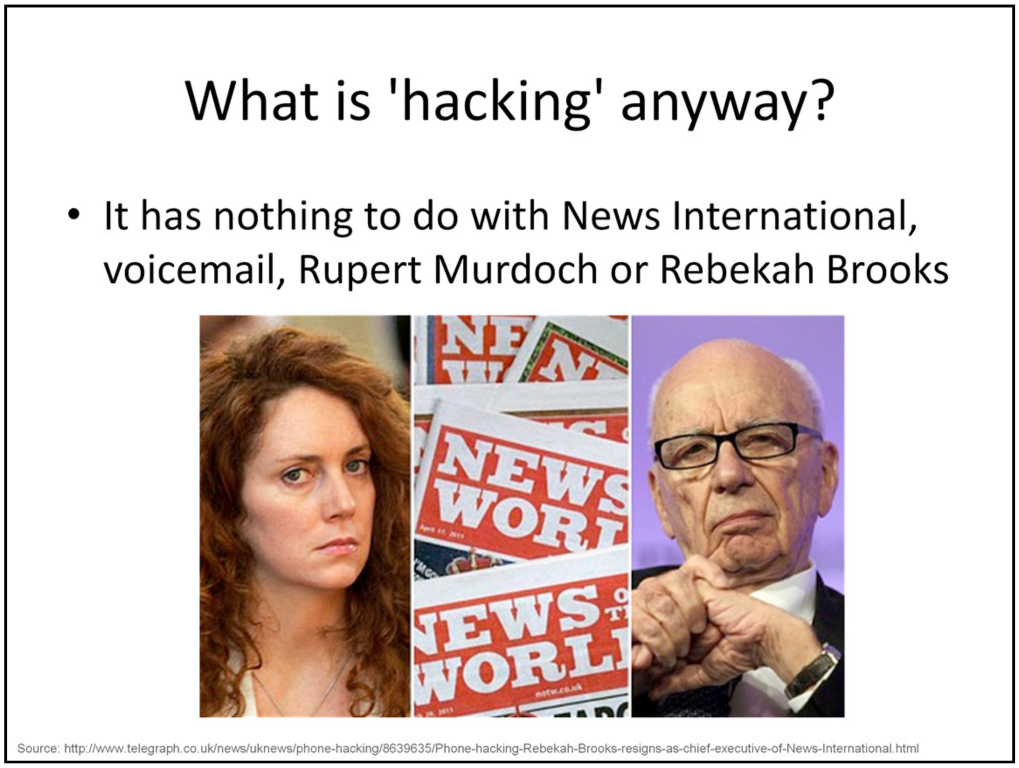
I taught myself HTML around 1994, then JavaScript around 1996... Somewhere along the way I went back to uni and did a postgrad computer science degree to learn how to program properly.

Why does 'self-taught' matter? Because from little things, big things grow... The web is still new enough that many developers of my generation were originally self-taught. The web made programming accessible (view source, etc) and then it became something you could do for a living.

The intros round

- What's your name? Any code experience?
- What motivated you to attend, what do you want to learn?

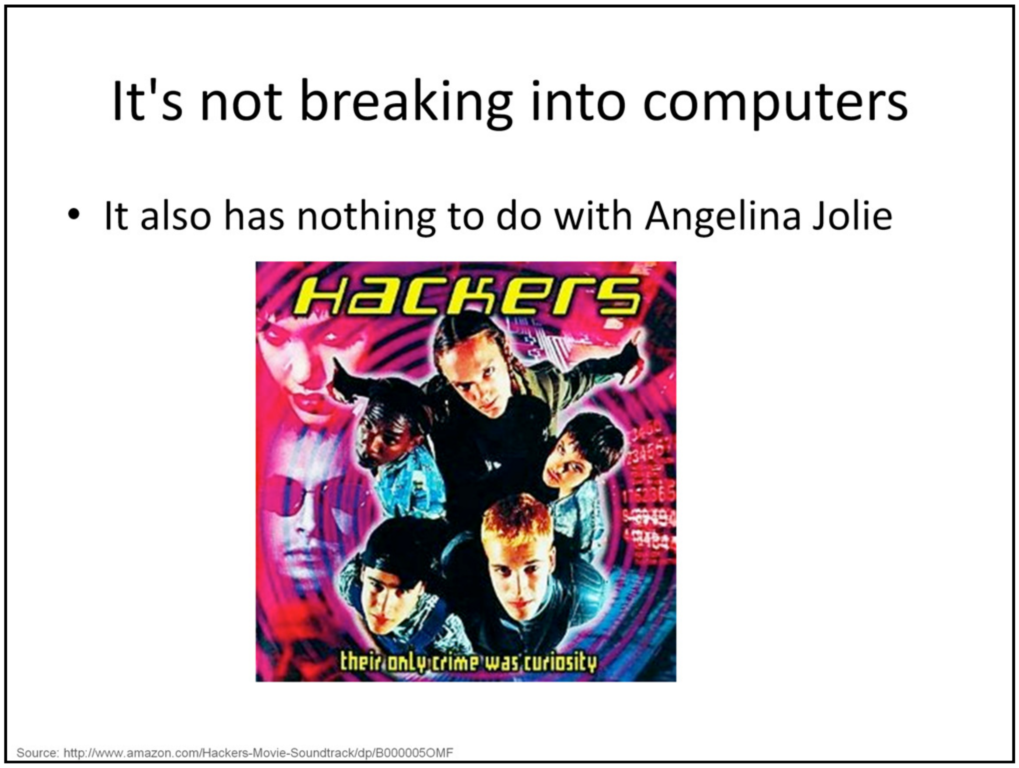
Try to keep it brief – one sentence intro on your interest in the digital humanities. If you've got any programming or HTML experience, mention it!



What is 'hacking' anyway?

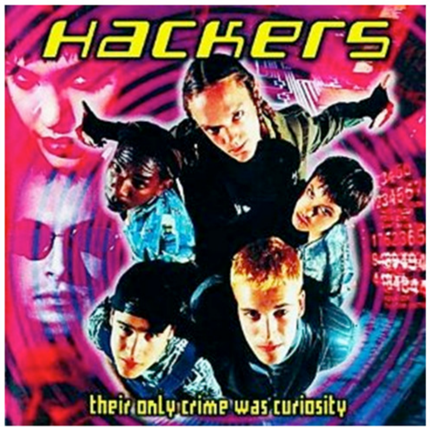
- It has nothing to do with News International, voicemail, Rupert Murdoch or Rebekah Brooks

Source: http://www.telegraph.co.uk/news/uknews/phone-hacking/8639635/Phone-hacking-Rebekah-Brooks-resigns-as-chief-executive-of-News-International.html



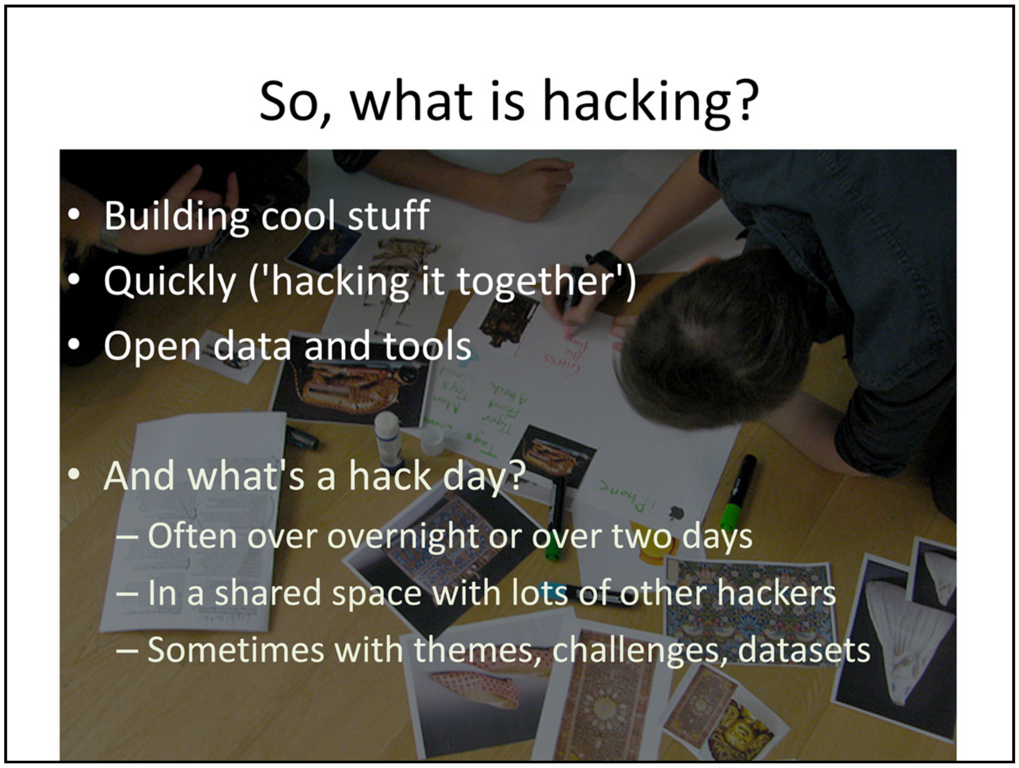
It's not breaking into computers

- It also has nothing to do with Angelina Jolie



Source: http://www.amazon.com/Hackers-Movie-Soundtrack/dp/B000005OMF

The sense I'm which I'm using the term has nothing to do with illegal access, whether phones or computers. It's tinkering, playing, about using your ingenuity and the tools available to you to get as much done as you can in a limited time.



So, what is hacking?

- Building cool stuff
- Quickly ('hacking it together')
- Open data and tools

- And what's a hack day?
  - Often over overnight or over two days
  - In a shared space with lots of other hackers
  - Sometimes with themes, challenges, datasets

Hacking means getting results by whatever means necessary in the limited time you have to play; or making stuff work even if it's not elegant.

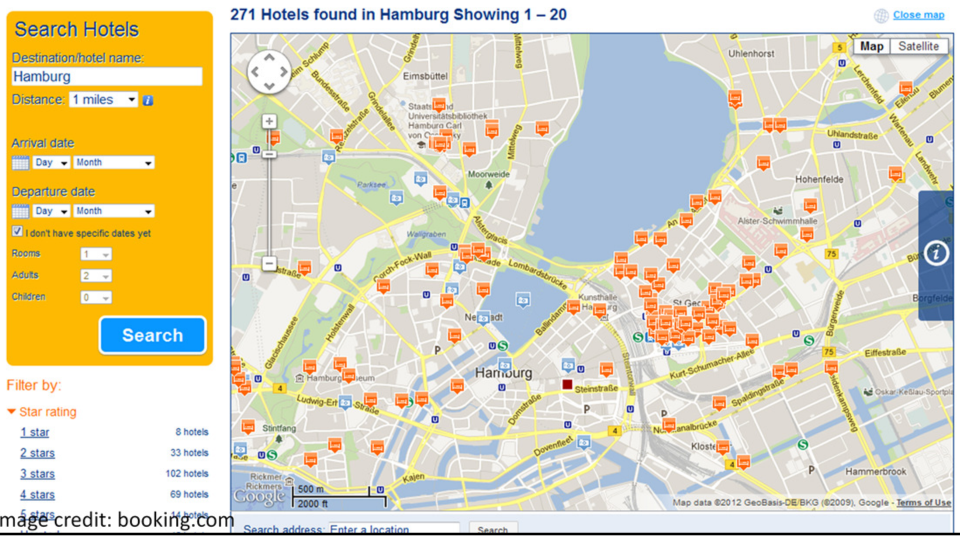
Hack days are time away from the normal concerns of the office in 'an environment free from political or monetary constraints' - and often free from documentation, testing, etc!

While at Yahoo, Chris Heilmann summed it up as: 'hack days are about innovating and playing with our own systems' and 'making something work regardless of the circumstances' http://www.slideshare.net/cheilmann/what-the-hack If a hack works, you can go back and tidy it up later.

Hack days don't have to be all about code. This image is from an event I ran at the V&A with Katy Beale. Participants were provided with markers, print-outs of objects, glue, scissors, paper, and asked to create 'their perfect museum experience' with paper prototypes.

What's a mashup? In the first of many dodgy food metaphors, a mashup is like fusion food... It could be French methods applied to Asian vegetables, or Indian spices in Japanese desserts...

Mashups combine data from one or more sources and/or data and visualisation tools such as maps or timelines.



Search Hotels

Destination/hotel name: Hamburg

Distance: 1 miles

Arrival date: Day Month

Departure date: Day Month

☑ I don't have specific dates yet

Rooms: 1

Adults: 2

Children: 0

Filter by:

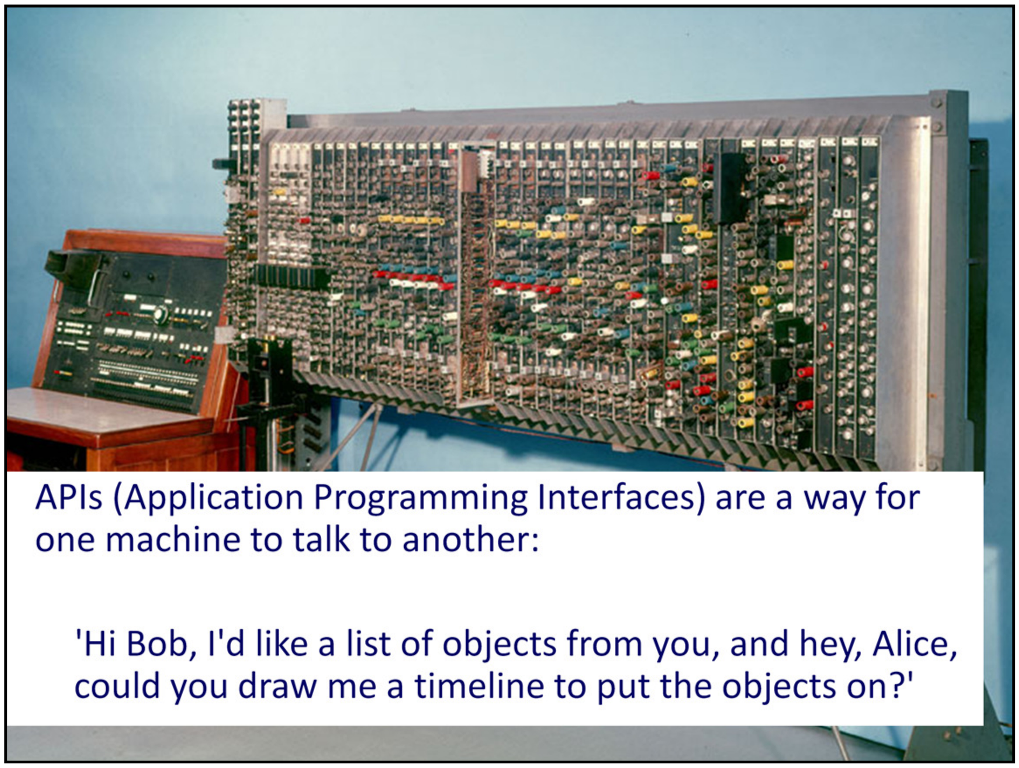
Star rating

1 star	8 hotels
2 stars	33 hotels
3 stars	102 hotels
4 stars	69 hotels

271 Hotels found in Hamburg Showing 1 – 20

Image credit: booking.com

Or, more formally, a mashup is a web page, website or app that combines data and services to create something new. We take them for granted now, but it wasn't that long ago you'd have to try and find the best hotel in a new city without being able to see all the options on a map. Mashups were made possible by APIs like the Google Map API... This example combines a tourist sites dataset with a list of hotels and displays them both on a map.



APIs (Application Programming Interfaces) are a way for one machine to talk to another:

'Hi Bob, I'd like a list of objects from you, and hey, Alice, could you draw me a timeline to put the objects on?'

What's an API? To continue the food analogy, an API is a menu of options. Just like in a fast-food restaurant, if it's not on the menu, you can't have it, so there's no point asking for a Whopper in McDonalds. But maybe you can ask them to hold the pickles...

Many APIs are a bit more sophisticated than that, and you can think of them as a menu of ingredients. The ingredients might be lists of objects, images, people or events that match search criteria, or a specific object, image, etc.

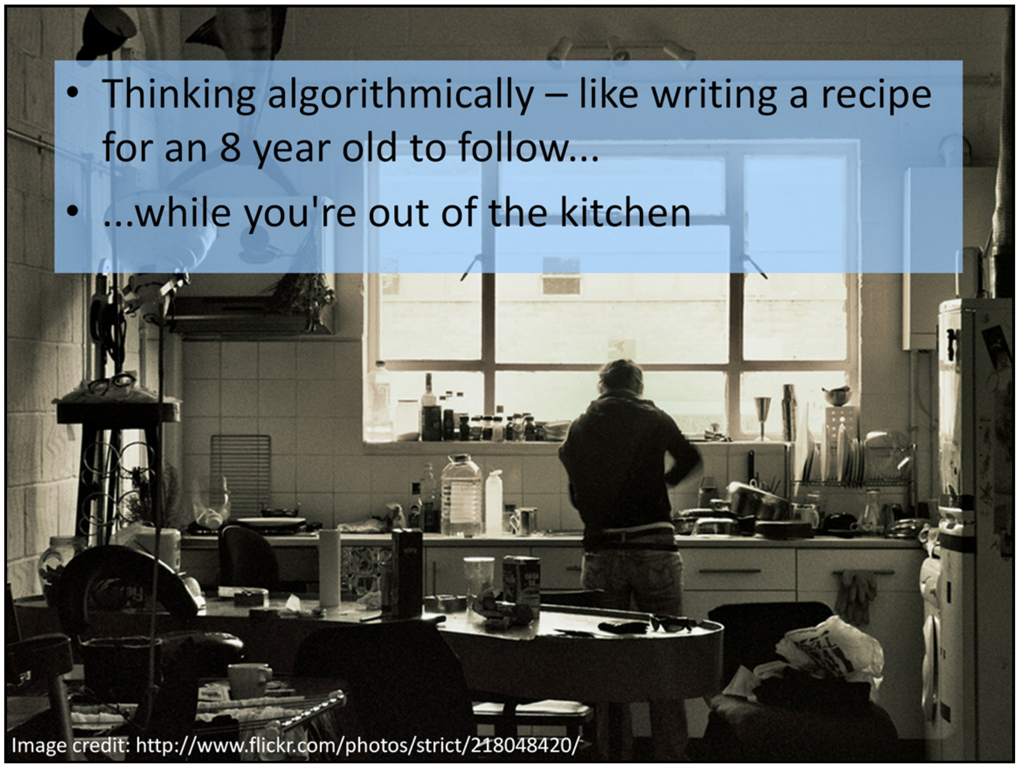
APIs let people make things that organisations haven't the resources to make themselves, or wouldn't have thought of making. They enable new questions to be asked by showing data in different ways or by combining it with other types of data – maps, timelines, other GLAM collections, external data sets...

There are other names for things like APIs – 'web services', linked open data, etc, but the principle is the same.

Thinking computationally

The basics of programming are not rocket science. Thinking computationally requires attention to detail,

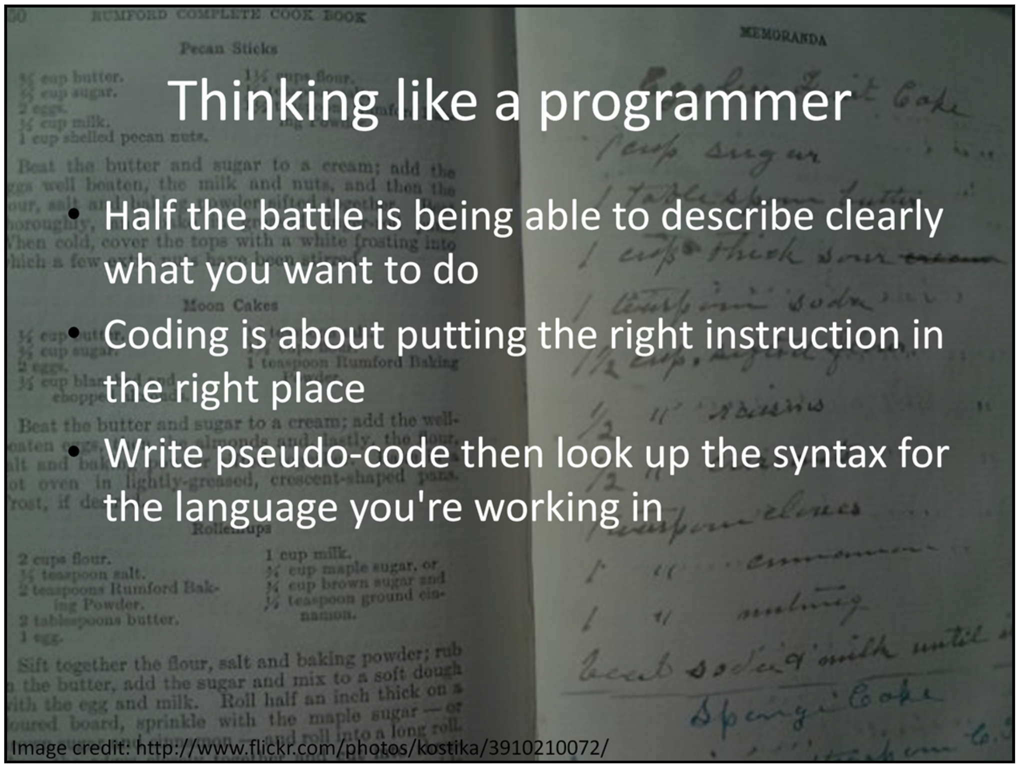
An algorithm is a step-by-step procedure for performing a function. As you'll see through the many food



- Thinking algorithmically – like writing a recipe for an 8 year old to follow...
- ...while you're out of the kitchen

Image credit: http://www.flickr.com/photos/strict/218048420/

Computers are smart, but they're also really stupid. It's a bit like that scene in Fantasia where the brooms k



Thinking like a programmer

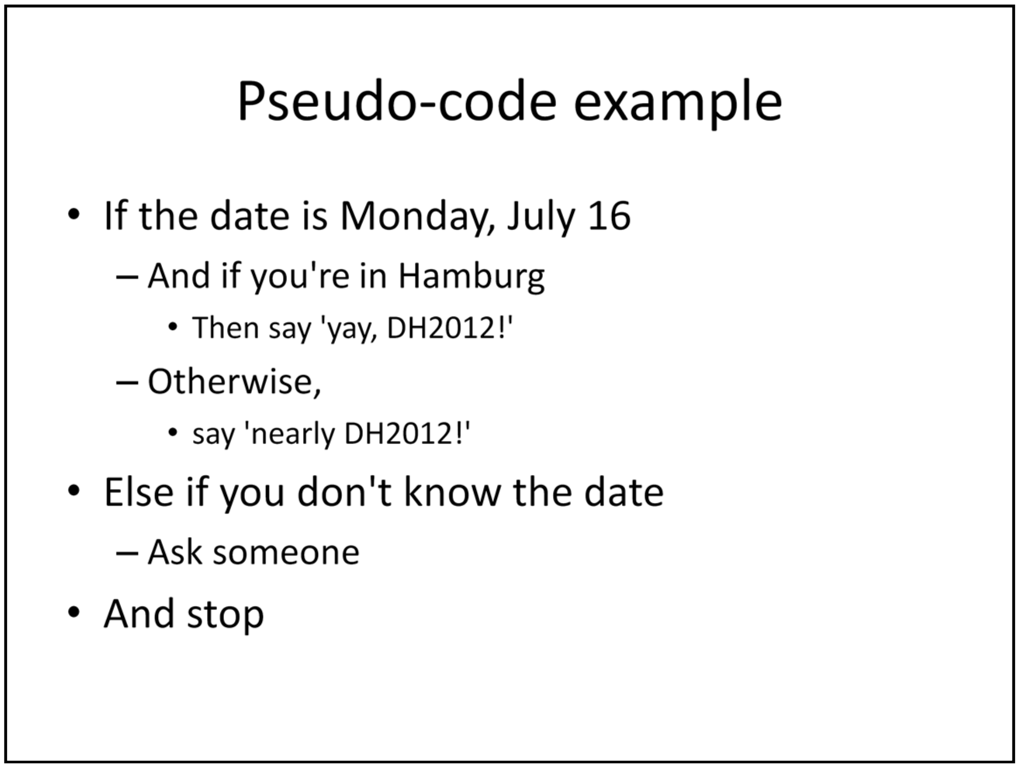
- Half the battle is being able to describe clearly what you want to do
- Coding is about putting the right instruction in the right place
- Write pseudo-code then look up the syntax for the language you're working in

Image credit: http://www.flickr.com/photos/kostika/3910210072/

Computers run one specific set of instructions after another. These instructions are called 'algorithms', which makes them sound wonderfully fancy.

These algorithms should be well-defined instructions. They should be unambiguous – computers can't guess what you mean even when it's obvious to a person. Computers are like that gullible friend it's really easy to play tricks on: they'll just do whatever you set them to do, so it's best to make sure you're asking them to do sensible things.

Write pseudo-code then figure out the syntax for the language you're working in. Think about right place – right time (right order), operating on the right thing (without side-effects like not changing anything it's not meant to). Even if you don't end up coding things yourself, expressing your ideas in pseudo-code will help you figure exactly what you need it to do and what input data or actions it will work on.



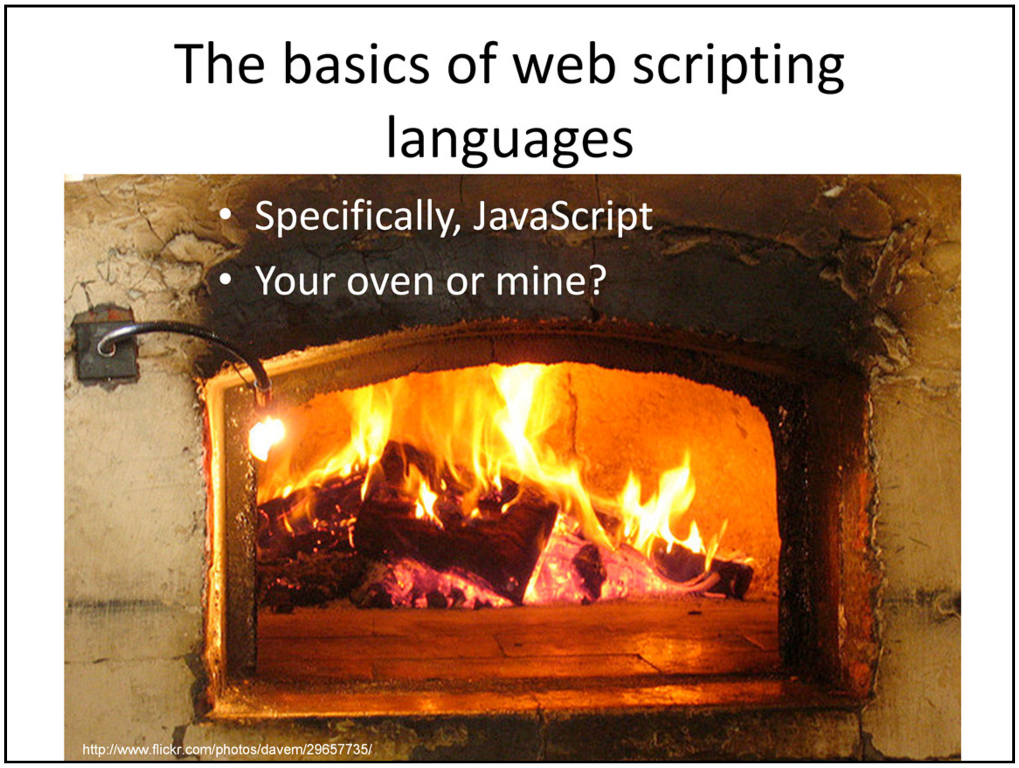
Pseudo-code example

- If the date is Monday, July 16
  - And if you're in Hamburg
    - Then say 'yay, DH2012!'
  - Otherwise,
    - say 'nearly DH2012!'
- Else if you don't know the date
  - Ask someone
- And stop

Writing pseudo-code is a bit like making notes about a recipe you know reasonably well – it's enough to remind you what the ingredients are, what order things have to be done in, what temperature the oven needs, but it's not enough detail to give it to a friend to try and make it. Writing it down means you have to think clearly about what you actually want.

Remember how I said computers were stupid? You often have to tell them when you're done, cos they can't figure it out for themselves and they'll just keep going and going and going.

Pseudo-code is also useful for figuring out what's going on in someone else's program – you can add comments to explain it to yourself while you're figuring it out.



The basics of web scripting languages

- Specifically, JavaScript
- Your oven or mine?

http://www.flickr.com/photos/davem/29657735/

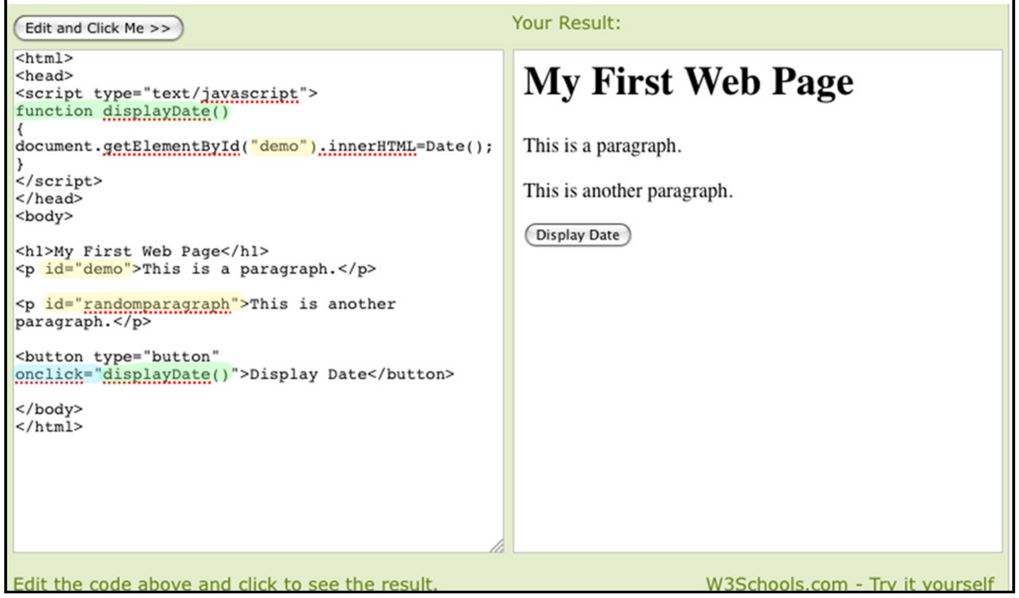
Different languages make different things easy or hard or offer more shortcuts. Just like human languages – some are good at talking about melancholy, others about love.

Scripts can run on a big shiny web server – a bit like ordering pizza, it's all cooked in an oven somewhere else and brought to your door – or locally in your browser – like cooking a pizza in your oven at home.

The advantage of JavaScript is that it can be run locally, from a text file. You don't need a web server. You do need a web server for languages like PHP, Python or Ruby. I wanted to keep things simple so you can concentrate on learning so I've gone for a scripting language you can run without a webserver, but what you learn here can help you with other languages.

I'm going to show you how JavaScript can be run without anything except a browser now....

Example 1


Edit and Click Me >>
Your Result:
Display Date
Edit the code above and click to see the result.
W3Schools.com - Try it yourself

```
<html>
<head>
<script type="text/javascript">
function displayDate()
{
document.getElementById("demo").innerHTML=Date();
}
</script>
</head>
<body>

<h1>My First Web Page</h1>
<p id="demo">This is a paragraph.</p>

<p id="randomparagraph">This is another
paragraph.</p>

<button type="button"
onclick="displayDate()">Display Date</button>

</body>
</html>
```

```
My First Web Page

This is a paragraph.

This is another paragraph.
```

Because we can run (or cook) things locally, our code is in a simple text file, with a '.html' extension to let the browser know it's a web page.

There's a lot going on here, so let's break it down a bit. You can open up the exercise file and follow along.

At the top and bottom there's some HTML that sets up the page, and also puts special labels (called **IDs**, shown in yellow bit) on some bits of text on the page. There are two paragraphs on this page, one has the ID 'demo', the other 'random paragraph'.

We hide the JavaScript (the bits in <script> tags to stop browsers that can't read JavaScript getting confused). We can hide it in the 'head' of the HTML page, or in HTML comments (<!-- -->).

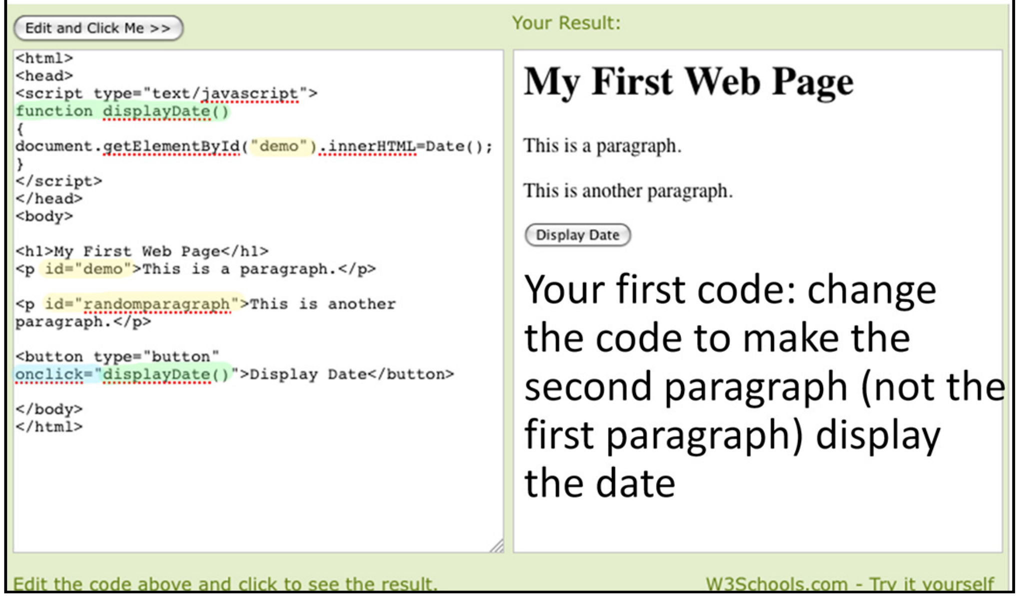
The bits in green are a **function**. A function is a pre-made set of instructions – like asking a friend to make the pastry while you make the pie filling. A function has two bits – the definition, where you tell your friend how you want the pastry to be made – and the function call, where you actually ask for the pastry when you need it.

The bit in blue is an '**event** handler' – it knows what to do when specific things (events) happen, like the timer going off, or someone clicking a button. They act a bit like the way our brains can hear our name in a crowd of noise – they're listening for something specific; when they hear it, they usually ask a function to run, using the function call.

Looking at the code, can you guess what's going to happen when I click the button?

Click the button.]

Exercise 1


Edit and Click Me >>
Your Result:
Display Date
Edit the code above and click to see the result.
W3Schools.com - Try it yourself

```
<html>
<head>
<script type="text/javascript">
function displayDate()
{
document.getElementById("demo").innerHTML=Date();
}
</script>
</head>
<body>

<h1>My First Web Page</h1>
<p id="demo">This is a paragraph.</p>

<p id="randomparagraph">This is another
paragraph.</p>

<button type="button"
onclick="displayDate()">Display Date</button>

</body>
</html>
```

My First Web Page

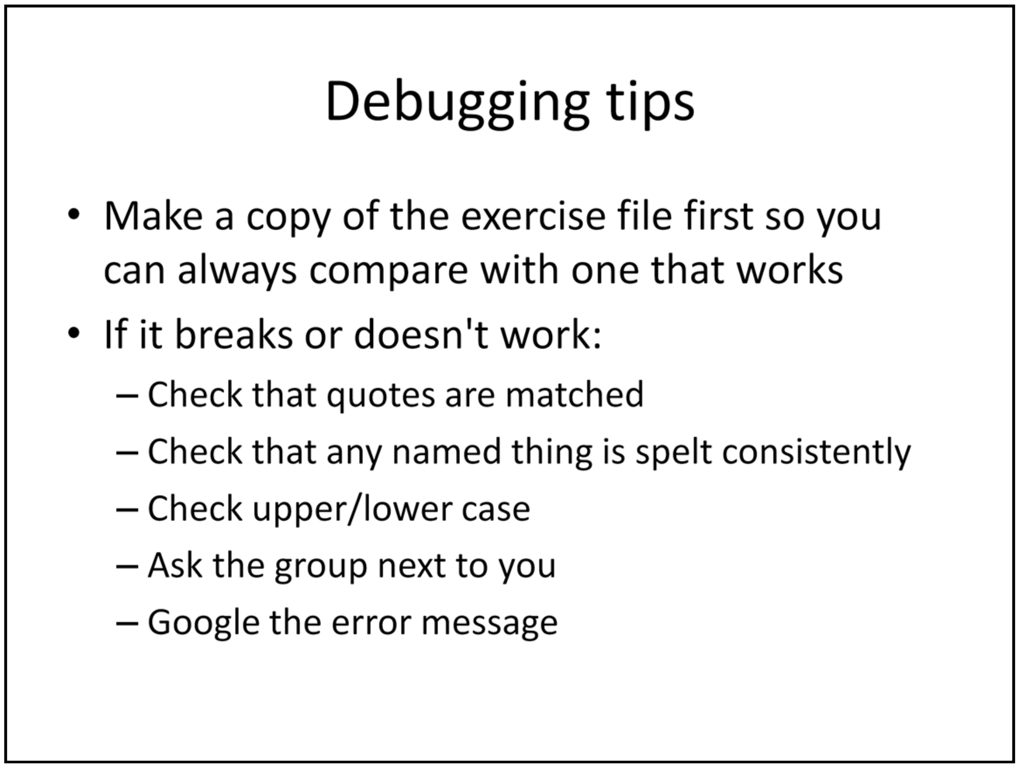
This is a paragraph.

This is another paragraph.

Your first code: change the code to make the second paragraph (not the first paragraph) display the date

Ok, enough watching me, it's time to get coding! What do you need to change in the code to make the second paragraph display the date, instead of the first paragraph?

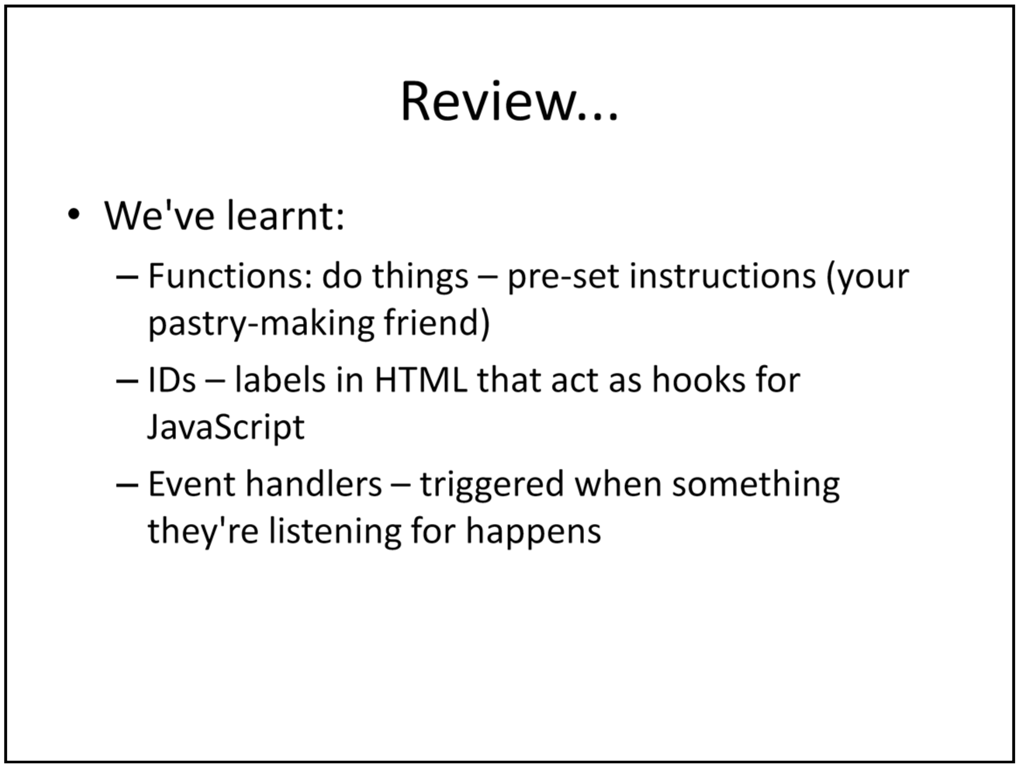
[First exercise - showing the text file is 'live' – we're going to change some text in the file and reload the page to show how it's changed. Edit the function to change the bit that it changes to 'randomparagraph' instead of 'demo' – save, reload the page, click the button, show that it's changed – check that everyone's managed this ok.]



Debugging tips

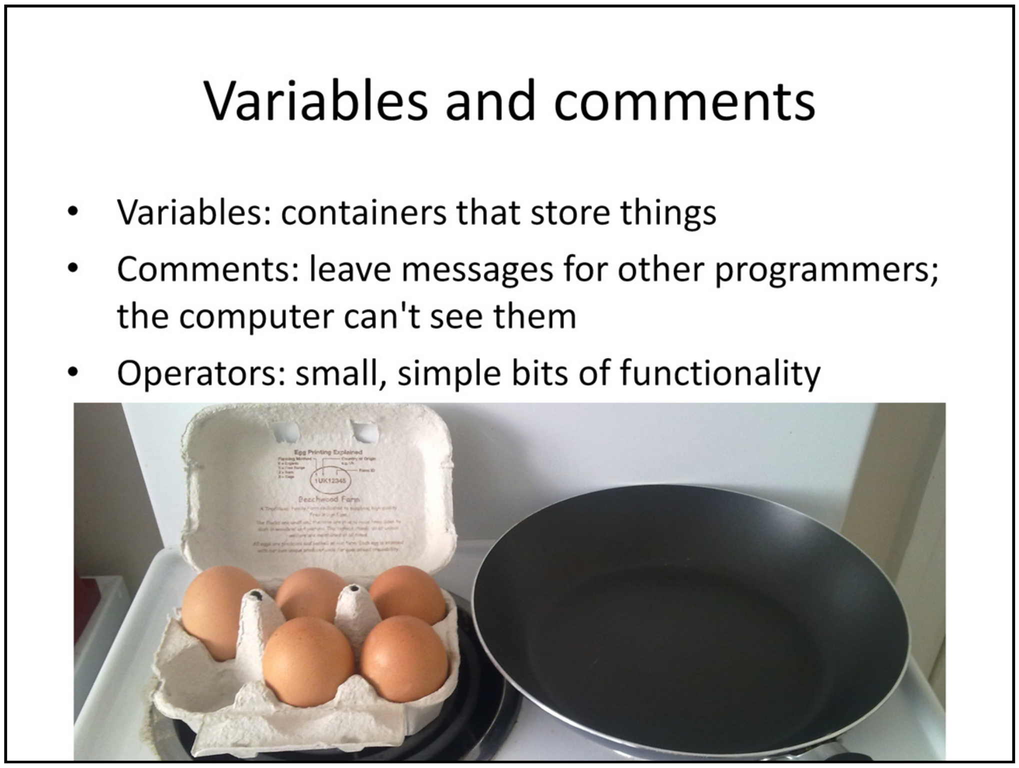
- Make a copy of the exercise file first so you can always compare with one that works
- If it breaks or doesn't work:
  - Check that quotes are matched
  - Check that any named thing is spelt consistently
  - Check upper/lower case
  - Ask the group next to you
  - Google the error message

[leave up during exercise]



Review...

- We've learnt:
  - Functions: do things – pre-set instructions (your pastry-making friend)
  - IDs – labels in HTML that act as hooks for JavaScript
  - Event handlers – triggered when something they're listening for happens



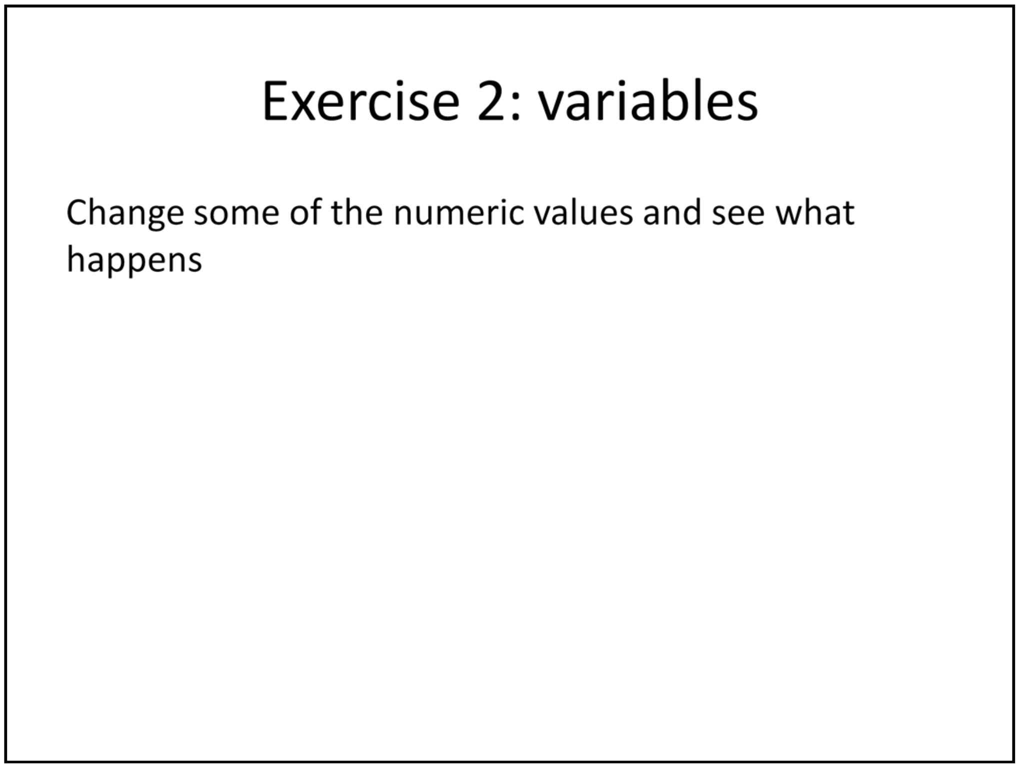
Variables and comments

- Variables: containers that store things
- Comments: leave messages for other programmers; the computer can't see them
- Operators: small, simple bits of functionality

Going with the cooking analogy, variables are containers – an egg carton, a milk jug, a bowl. First, you create the variable - like making room on the bench before you get out the bowl. Variable names can be anything you like as long as they're used consistently.

Then you tell the computer what you're going to put in it – 'assigning a value'. For simplicity's sake, we'll say most variables are numbers or 'strings'. A string is a bit of text that's treated literally – the computer passes it around without trying to figure out what it means, like a kid missing double entrendres in adult conversations.

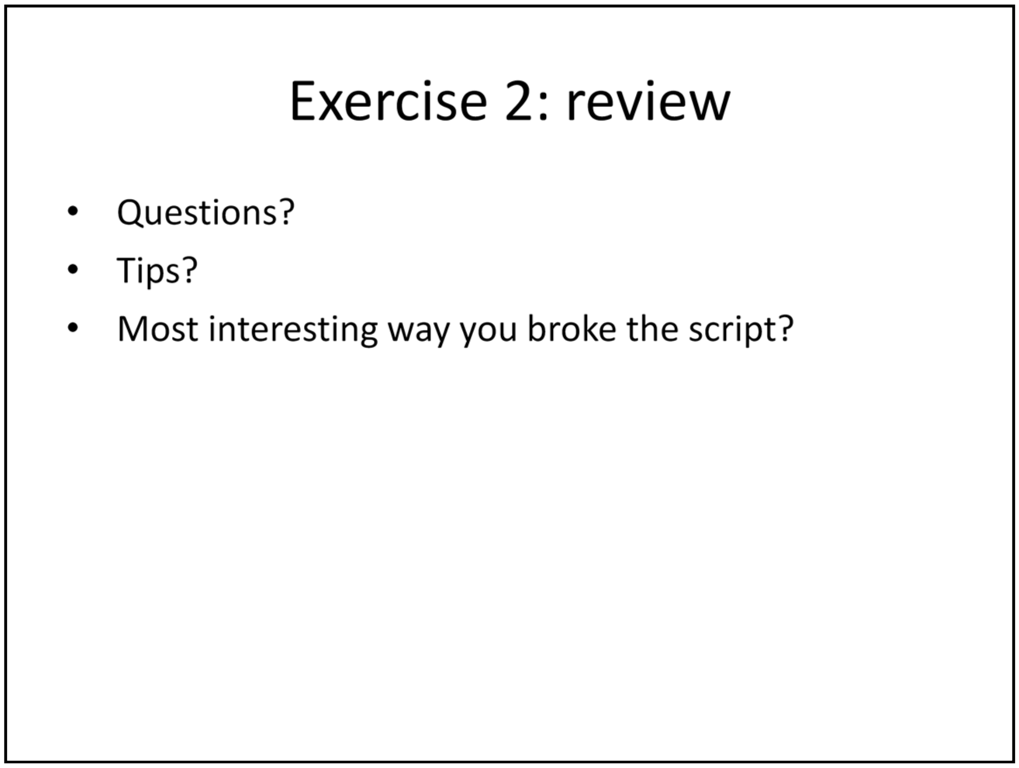
Operators are a bit like functions that do really simple things. A blender that smushes, a frying pan that heats, a fridge that cools things. The first operator you'll meet is the plus sign, which adds things together... But it does so in different ways depending on what the variable is. Sometimes, the plus sign adds together two numbers, but it can also join two strings into a longer one. Let's see it in action...



Exercise 2: variables

Change some of the numeric values and see what happens

See what happens when you change various values or try different operators...

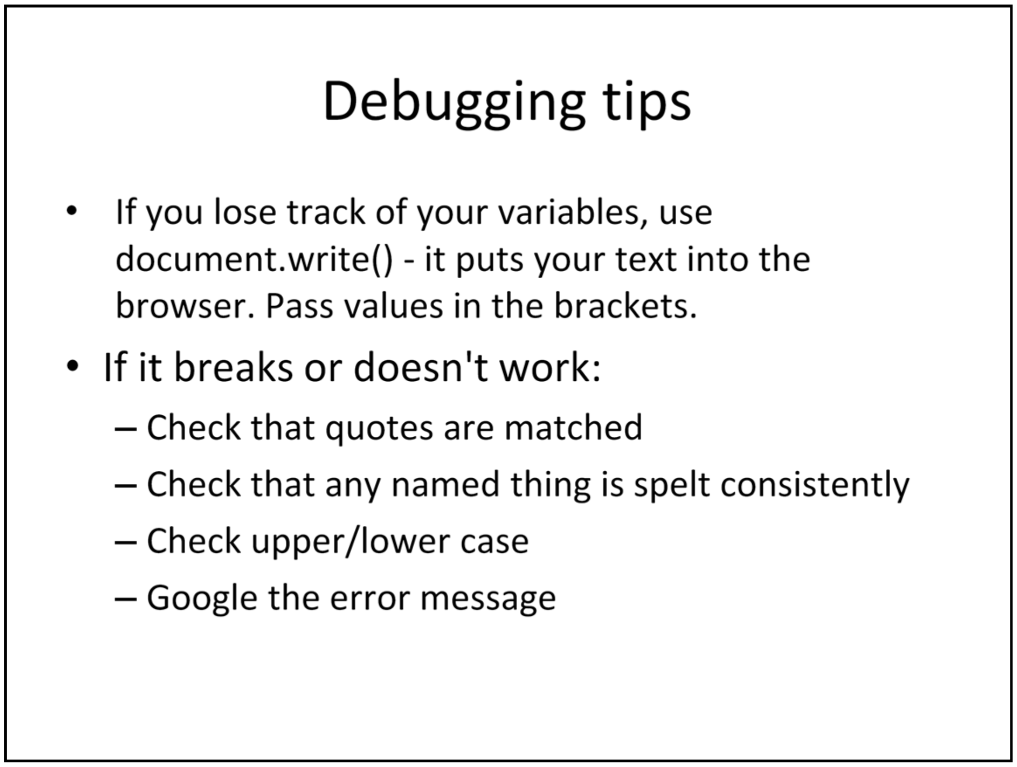


Exercise 2: review

- Questions?
- Tips?
- Most interesting way you broke the script?

What did you learn about how operators and variables work? About scripting in general?

Did anyone try to test the differences between =, ==?



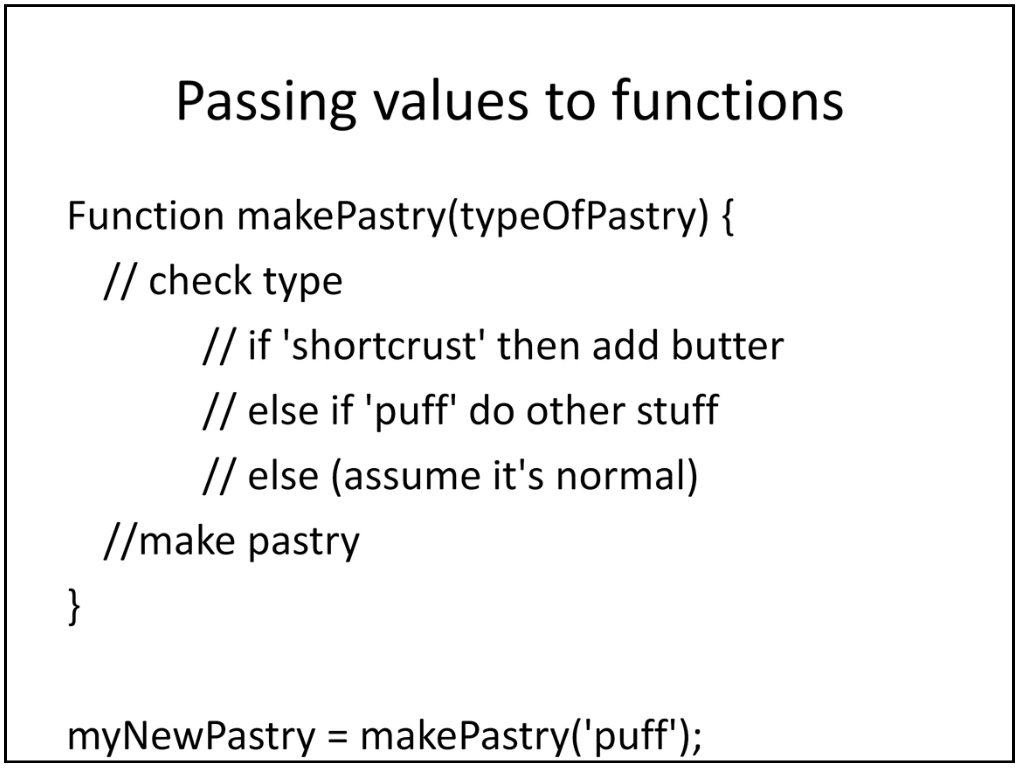
Debugging tips

- If you lose track of your variables, use document.write() - it puts your text into the browser. Pass values in the brackets.
- If it breaks or doesn't work:
  - Check that quotes are matched
  - Check that any named thing is spelt consistently
  - Check upper/lower case
  - Google the error message

I'll explain more about passing values next, but for now, just remember that there's a way to help you keep track of things. Document.write() is a very handy function when you're beginning.

'Stubs' are useful for debugging. Writing out the values of your variables at different points (e.g. before and after a function) can help you figure out what's going on.

Access to-browser debugging consoles: http://www.alistapart.com/articles/modern-debugging-tips-and-tricks 'Launching the JavaScript Console' section



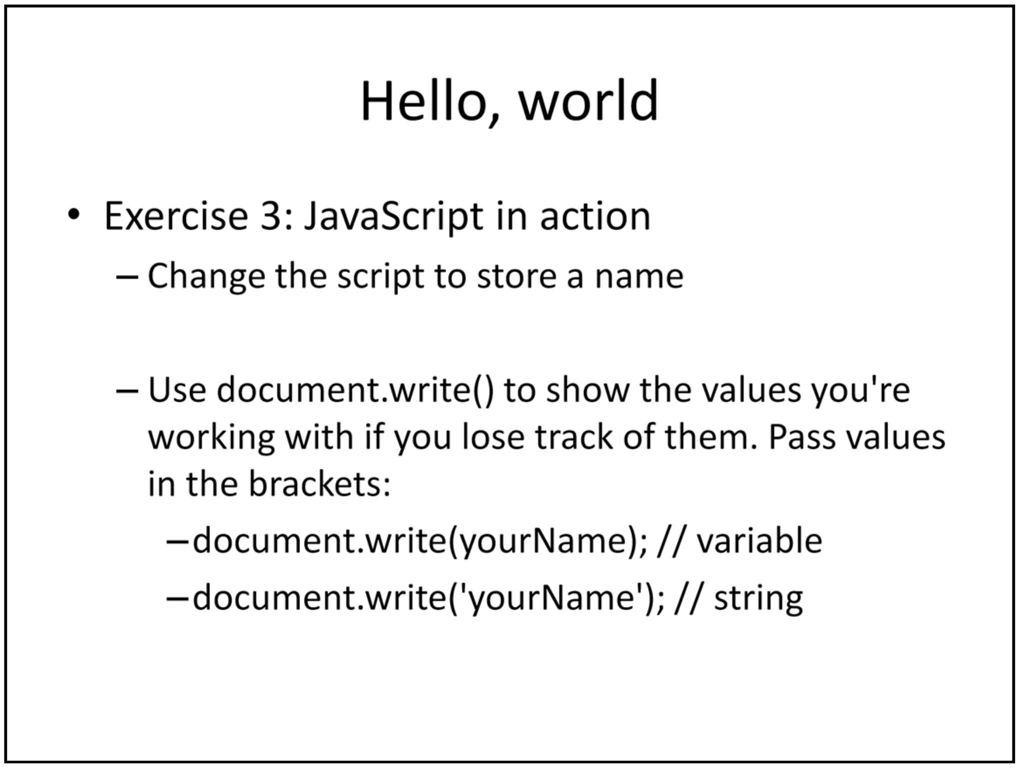
Passing values to functions

```
Function makePastry(typeOfPastry) {
   // check type
         // if 'shortcrust' then add butter
         // else if 'puff' do other stuff
         // else (assume it's normal)
   //make pastry
}

myNewPastry = makePastry('puff');
```

Remember how I said a function was a bit like your friend making pastry while you made pie filling? Well, if it turns out your friend is really good at pastry, you can ask them to make lots of different sorts of pastry, depending on what type of pie you're making. But you don't want a bunch of different functions when most parts of them are the same, so you make one function and add some detail depending on what's needed for different types of pastry. You tell the function knows what kind of pastry you want by 'passing an argument' when you call it.

This pseudo-code also shows some control structures – what happens depends on the input to the function.



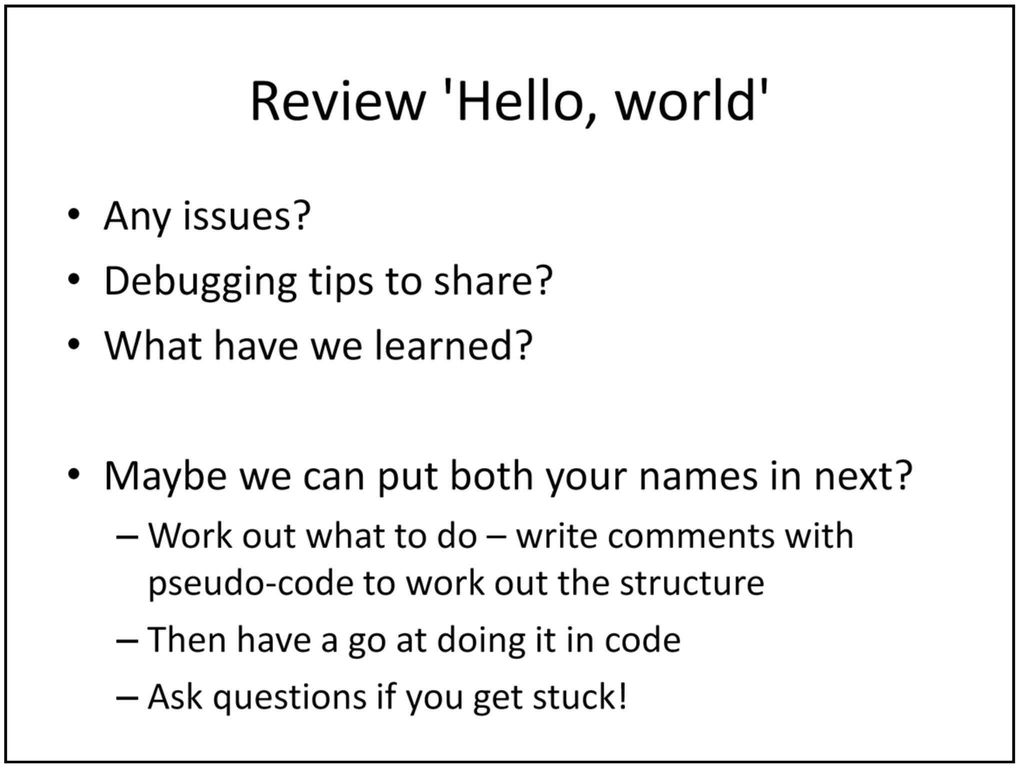
Hello, world

- Exercise 3: JavaScript in action
  - Change the script to store a name
  - Use document.write() to show the values you're working with if you lose track of them. Pass values in the brackets:
    - document.write(yourName); // variable
    - document.write('yourName'); // string

So, back to our document.write() function – we're going to pass it the text we want printed, as an argument (ie in brackets).

Do it on the supplied text files or use the console on http://www.w3schools.com/js/tryit.asp?filename=tryjs_events

Check the error console on your browser if it breaks, or have a think about whether you want to use a string or a variable in different places. Discuss it in your pairs to figure out what's going on, and I'll be around to check that each of you are getting it.

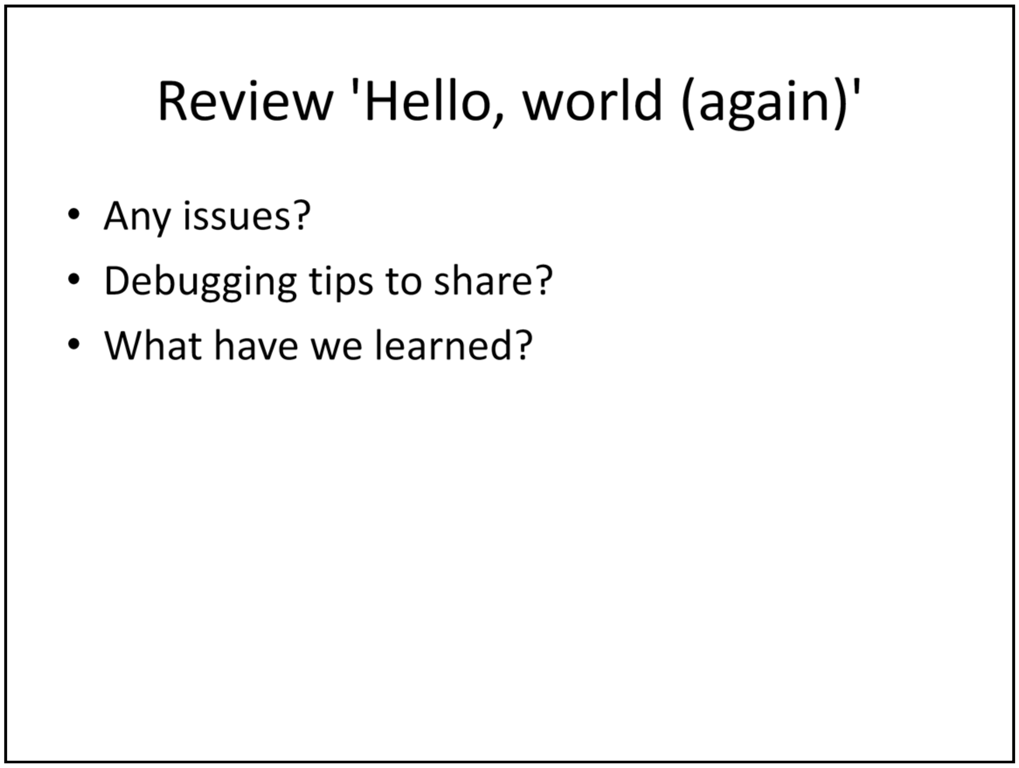


Review 'Hello, world'

- Any issues?
- Debugging tips to share?
- What have we learned?

- Maybe we can put both your names in next?
  - Work out what to do – write comments with pseudo-code to work out the structure
  - Then have a go at doing it in code
  - Ask questions if you get stuck!

Exercise 4 builds on the file from Exercise 3.



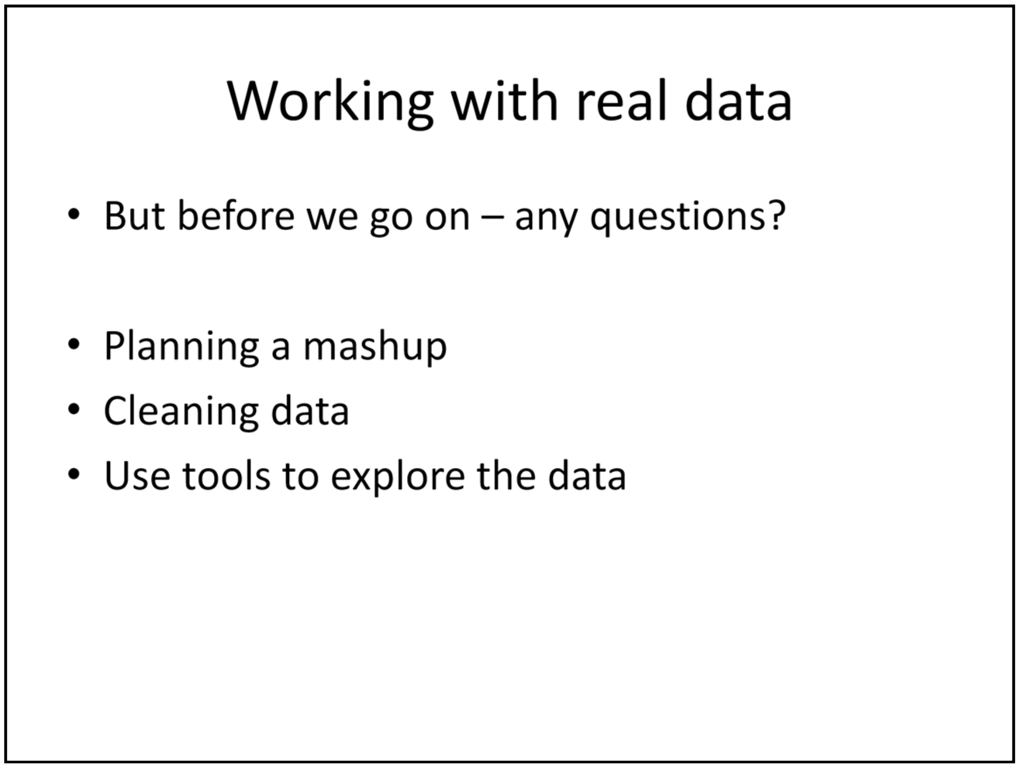
Review 'Hello, world (again)'

- Any issues?
- Debugging tips to share?
- What have we learned?

You might find you want an ' and ' – including the white space. There are a few ways to do it – creating another variable that stores whitespace, adding whitespace to someone's name, including a string...

If anyone has an apostrophe in their name, they might run into an issue

http://www.w3schools.com/js/js_special_characters.asp



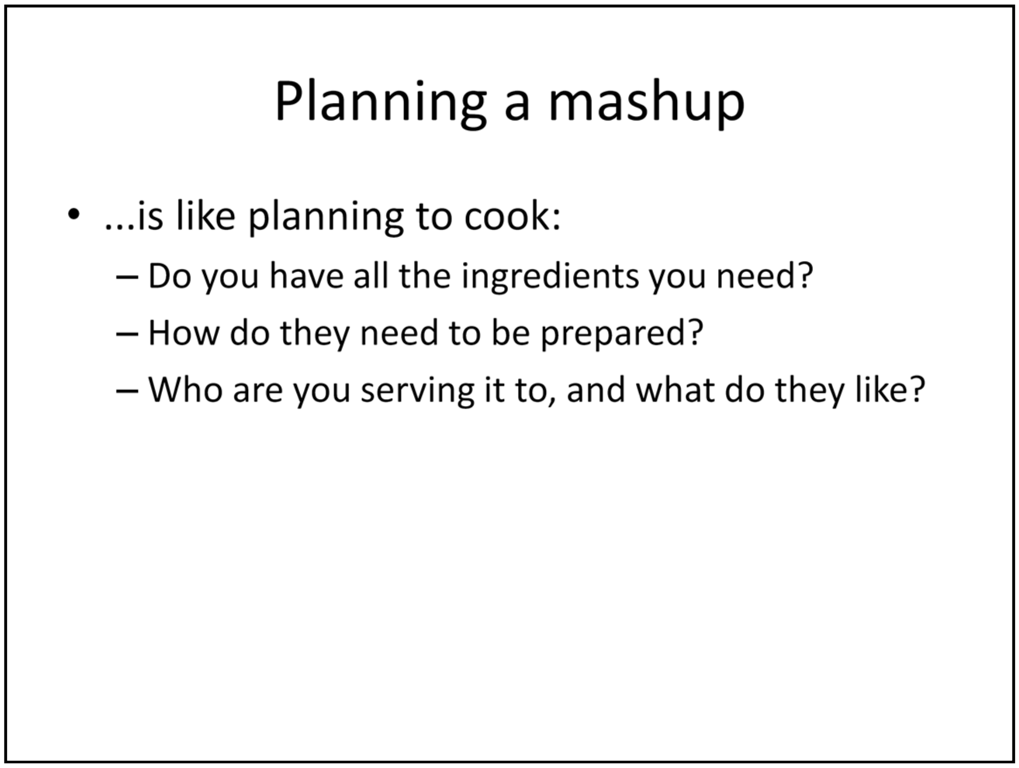
Working with real data

- But before we go on – any questions?

- Planning a mashup
- Cleaning data
- Use tools to explore the data

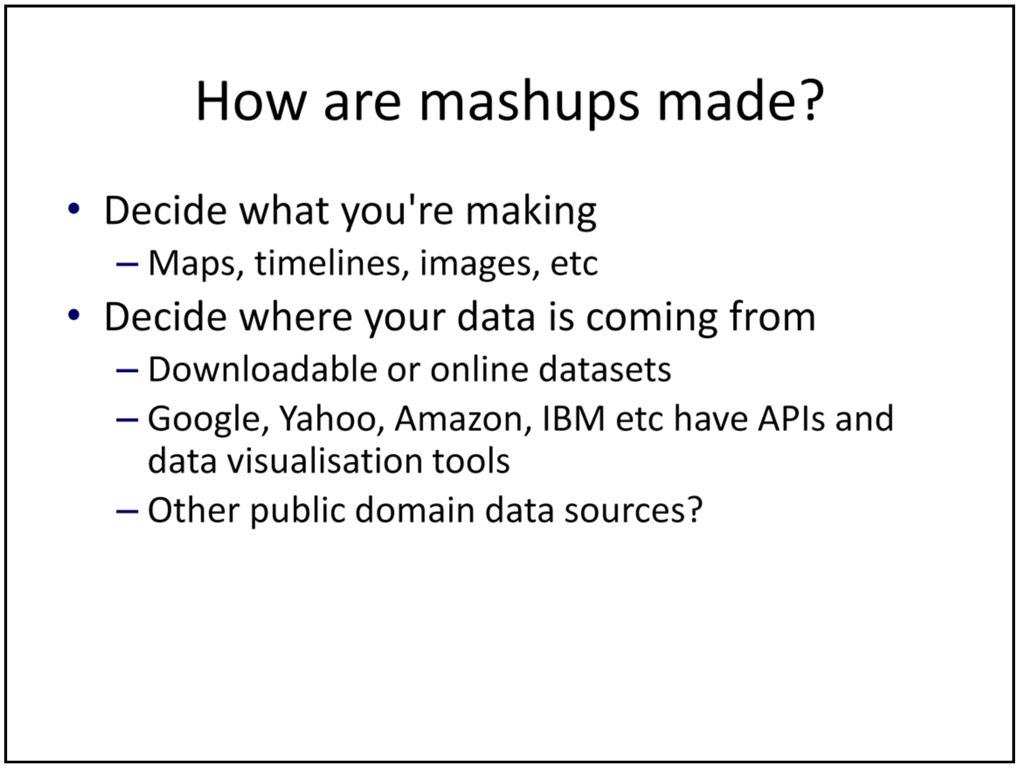
In the next section we'll look at visualisation tools. Visualisations can be products in their own right, or you can use them to start to the explore shape of data. Visualisations include timelines, maps, graphs, charts, word clouds.

I've included some object data released by the Science Museum Group (the parent organisation of the UK's Science Museum, National Media Museum and National Railway Museum) and from the Cooper-Hewitt Smithsonian Design Museum in these examples. The files are in the folders on the USB stick.



Planning a mashup

- ...is like planning to cook:
  - Do you have all the ingredients you need?
  - How do they need to be prepared?
  - Who are you serving it to, and what do they like?

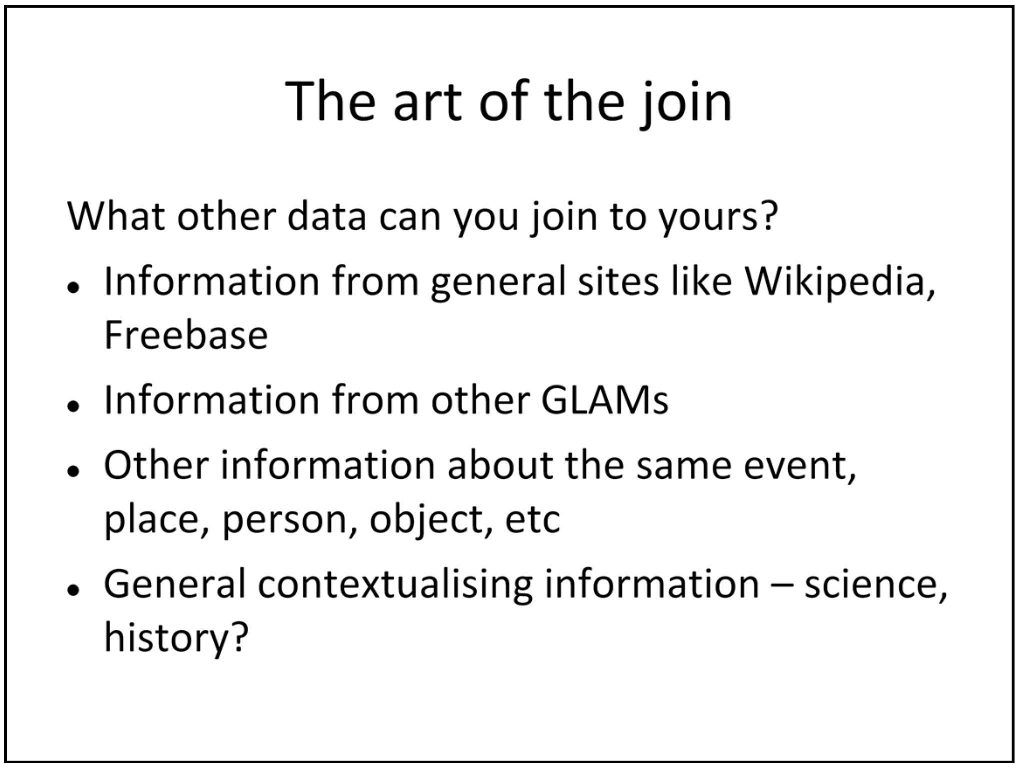


How are mashups made?

- Decide what you're making
  - Maps, timelines, images, etc
- Decide where your data is coming from
  - Downloadable or online datasets
  - Google, Yahoo, Amazon, IBM etc have APIs and data visualisation tools
  - Other public domain data sources?

How are mashups made?

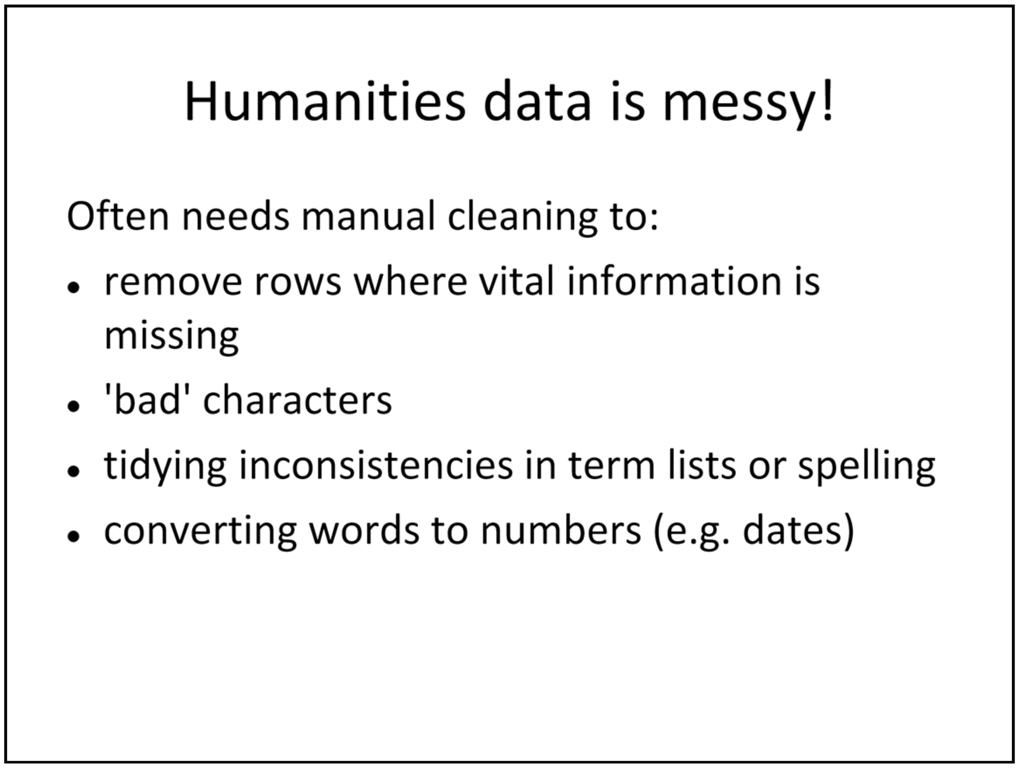
- Think through technical issues
  - Do you have the programming, user experience, design expertise you need? What framework will you use? Where will you host it?
- Can you describe what you want in pseudo-code?
- Start designing, programming, testing – start small and build incrementally



The art of the join

What other data can you join to yours?

- Information from general sites like Wikipedia, Freebase
- Information from other GLAMs
- Other information about the same event, place, person, object, etc
- General contextualising information – science, history?



Humanities data is messy!

Often needs manual cleaning to:

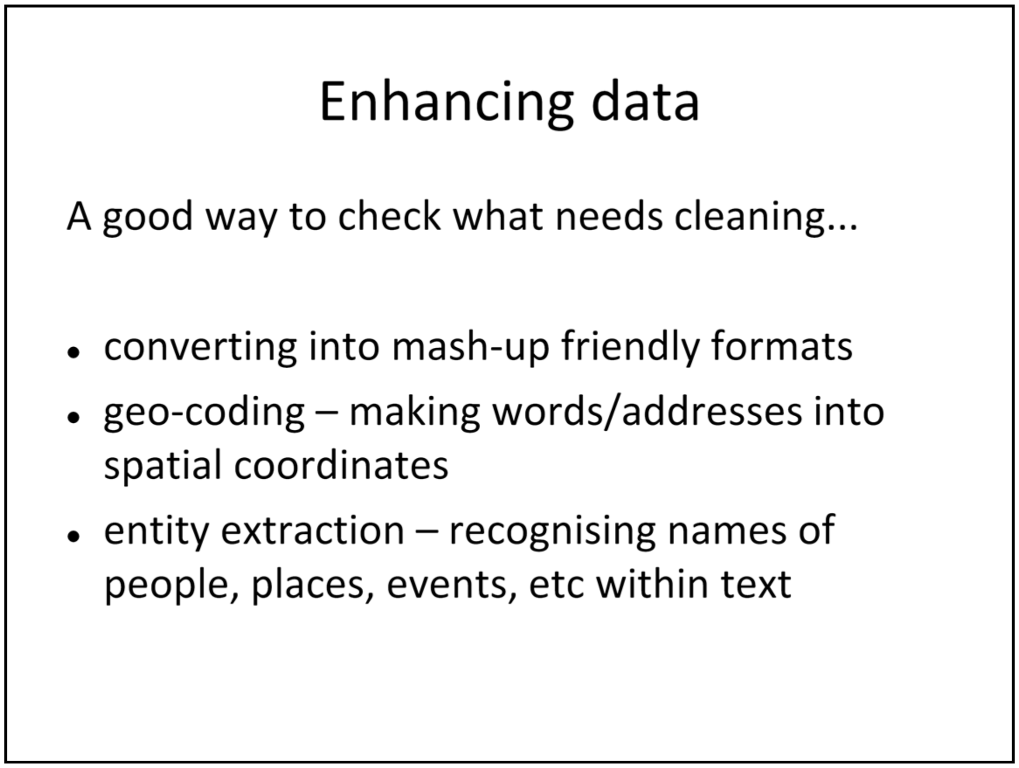
- remove rows where vital information is missing
- 'bad' characters
- tidying inconsistencies in term lists or spelling
- converting words to numbers (e.g. dates)

Humanities data is only as good as the hundreds of people who've created, transcribed and recorded it over the centuries or decades... Office tools like Excel and Access are useful for quickly finding empty rows; or download Google Refine or use online spreadsheets.

Raw collections data often needs a bit of cleaning before it can be used. Heritage datasets, particularly museum data, is often incomplete or contains fuzzy data that's fine for people but hard for computers to cope with. We can agree among ourselves that 'c1905 means 'within ten years of 1905' and 'c1900' means 'within fifty years of 1900' but how would a computer know that? And what does it do when it encounters '1900s'? (For that matter, would an ordinary person?) Specialised services that understand heritage data can help but there aren't many of those. Data might also contain odd characters or unexpected empty fields that will break scripts.

Sometimes humanities data is more vague about dates and places than a computer would like, but that can't be helped. Converting words to numbers should be considered carefully where it creates apparent precision from fuzziness – better to work to date ranges?

The necessity to leave out data that isn't clean enough is one reason visualisations should be taken with a pinch of salt...

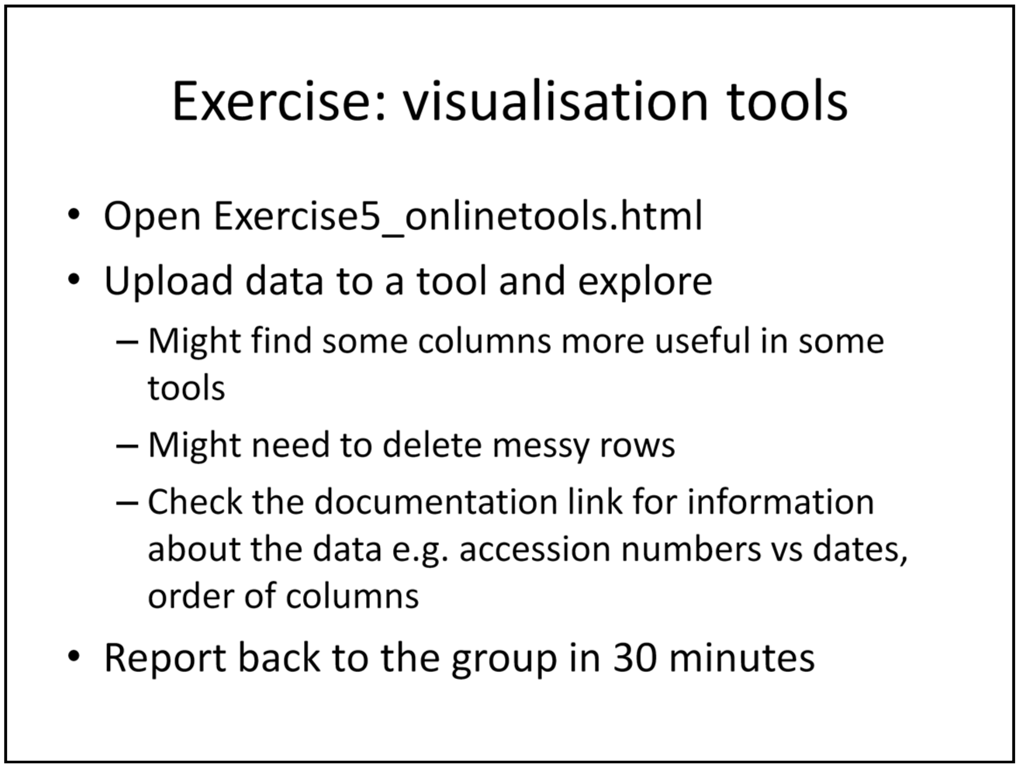


Enhancing data

A good way to check what needs cleaning...

- converting into mash-up friendly formats
- geo-coding – making words/addresses into spatial coordinates
- entity extraction – recognising names of people, places, events, etc within text

Data prep can also include making into a format like XML, JSON, CSV, TSV... Don't be put off by the acronyms, you can break it all down into smaller tasks and do them one after the other.



Exercise: visualisation tools

- Open Exercise5_onlinetools.html
- Upload data to a tool and explore
  - Might find some columns more useful in some tools
  - Might need to delete messy rows
  - Check the documentation link for information about the data e.g. accession numbers vs dates, order of columns
- Report back to the group in 30 minutes

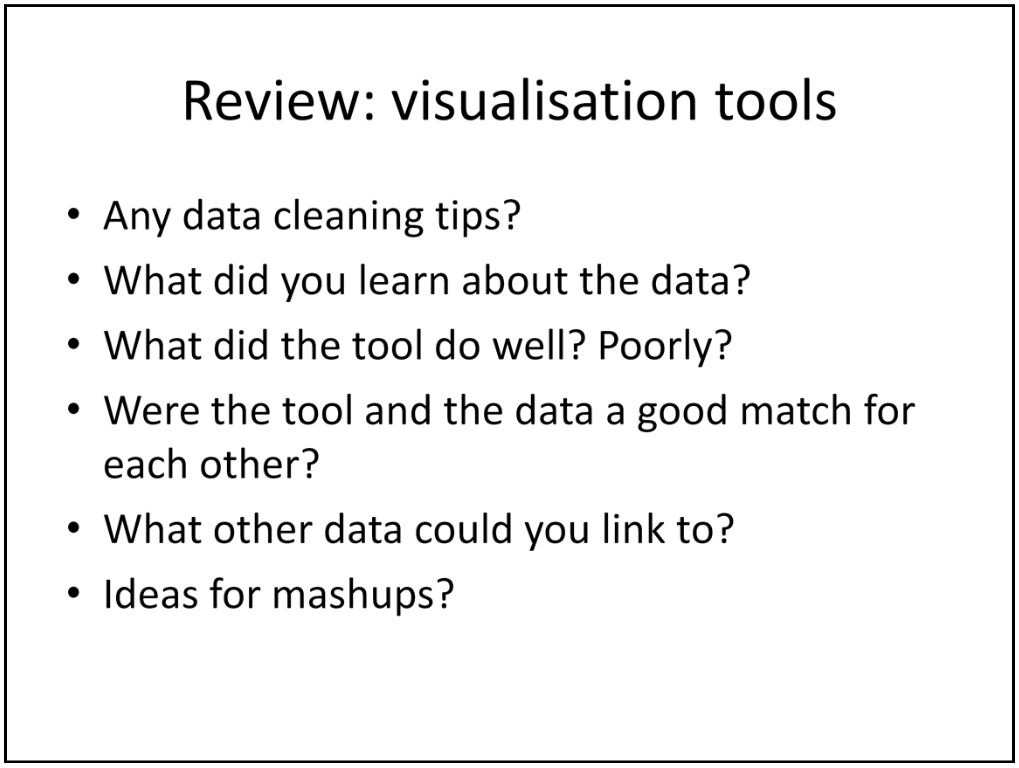
These visualisation tools can help you understand which datasets are worth zooming in on, and which reveal more from a 30,000 feet view. There's a sense of the deluge of data in the digital humanities – visualisations can help us make more of the increasing sets of new digital data, help us use this data to tell stories about the past that are better because of the context provided by all this new data.

You might need to try out these visualisations and clean up the data to improve your results in quick cycles.

NB: not all Google accounts have immediate access to Fusion tables, and the instructions aren't clear about uploading a table vs creating a new one.

Thanks to https://github.com/patrickmj/ http://hackingthehumanities.org/ Patrick Murray-John for sharing his tidied Cooper-Hewitt data; and for the Science Museum data, thank you, Ant Beck! The cleaned data source was:

http://dl.dropbox.com/u/393477/Temp/chn11/NMSI_Object_Cleaned_With_Grouping.txt

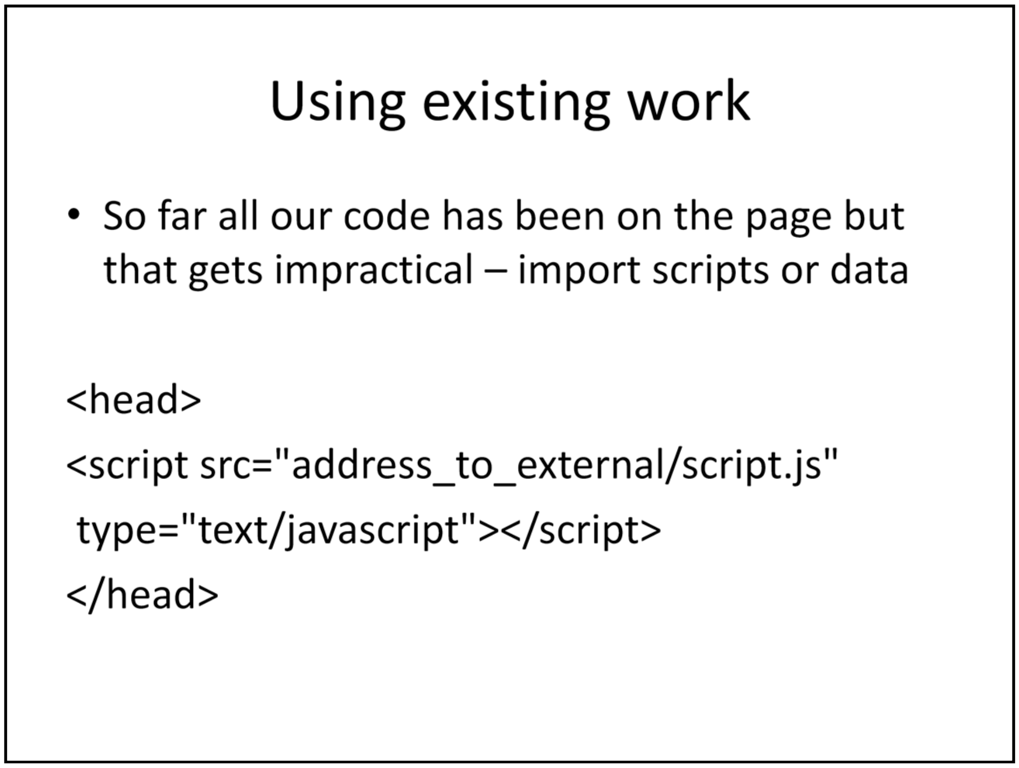


Review: visualisation tools

- Any data cleaning tips?
- What did you learn about the data?
- What did the tool do well? Poorly?
- Were the tool and the data a good match for each other?
- What other data could you link to?
- Ideas for mashups?

Things you might have discovered:

Lots of cleaning is required, especially for dates. What decisions did you make about keeping or deleting
Some visualisations are better for numbers, others for text
Work out what the tool is good at to focus your cleaning efforts, or select which columns of data to use
You might need to try different things to work out what the visualisation tool does



Using existing work

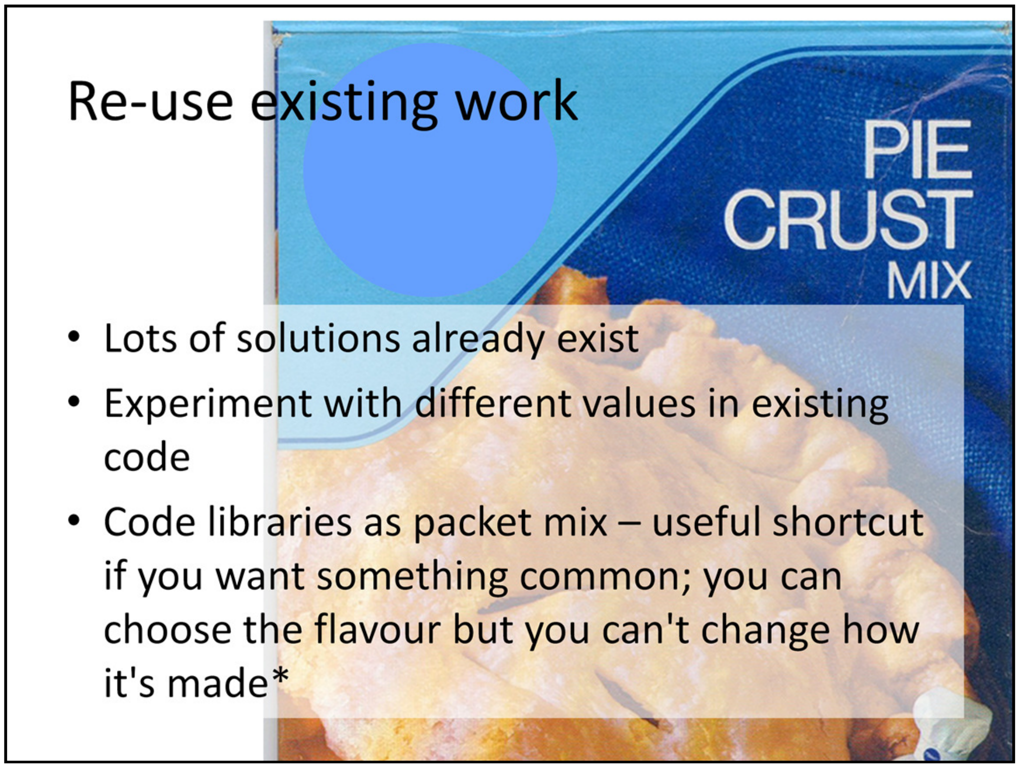
- So far all our code has been on the page but that gets impractical – import scripts or data

```
<head>
<script src="address_to_external/script.js"
 type="text/javascript"></script>
</head>
```

To insert a JavaScript into an HTML page, use the <script> tag.
Inside the <script> tag use the type attribute to define the scripting language.
The <script> and </script> tells where the JavaScript starts and ends (like before), but now it has a source

The script tag can go in the page or in a special bit at the top of the page.

You can use this to pull in libraries from several different sites.

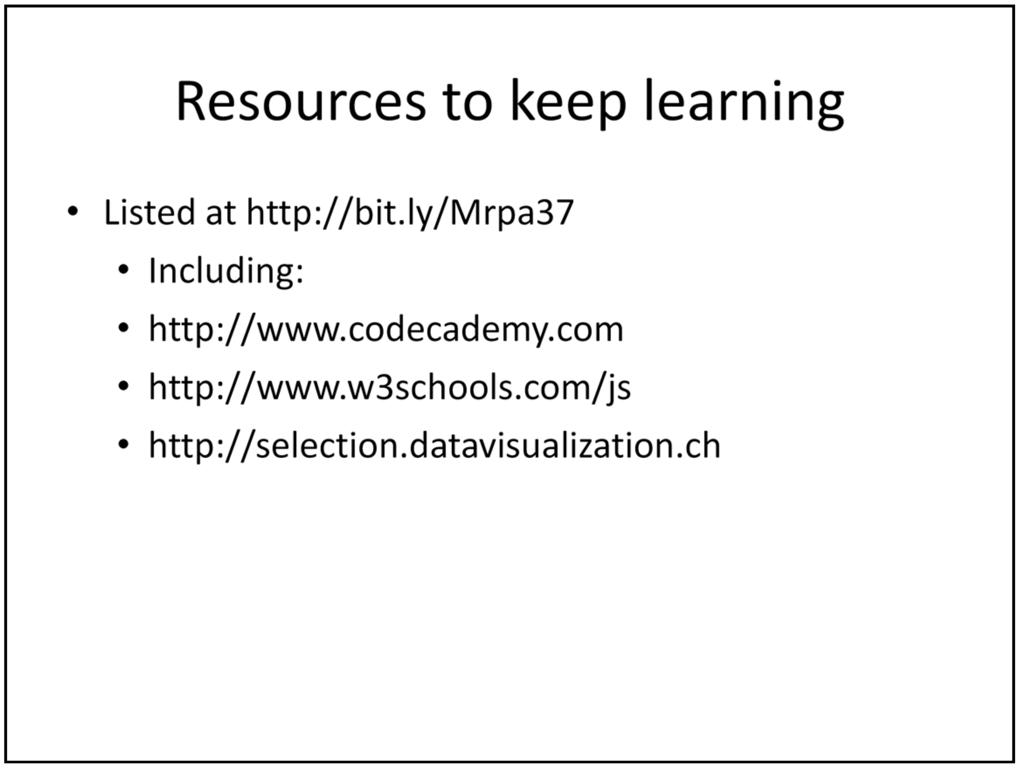

PIE
CRUST
MIX

Re-use existing work

- Lots of solutions already exist
- Experiment with different values in existing code
- Code libraries as packet mix – useful shortcut if you want something common; you can choose the flavour but you can't change how it's made*

* ok, you can, but let's not complicate things just yet

Being able to use libraries gives you access to cool tools like jQuery and MooTools as well as the visualisa



Resources to keep learning

- Listed at http://bit.ly/Mrpa37
  - Including:
  - http://www.codecademy.com
  - http://www.w3schools.com/js
  - http://selection.datavisualization.ch

And well done – you've written your first JavaScript, learnt something of how programmers think, and hopefully have some ideas for cool things you can make with humanities data... Give yourselves a round of applause!