Conditional statements

<table>
<thead>
<tr>
<th>Python</th>
<th>JavaScript</th>
</tr>
</thead>
</table>
| if x < y:   
    print('x is smaller than y')
| if (x < y) {
    console.log('x is smaller than y')
} |
| elif x == y:       
    print('x is equal to y')
| else if (x === y) {
    console.log('x is equal to y')
} |
| else:                      
    print('x is larger than y')
| else {          
    console.log('x is larger than y')
} |

In the JavaScript, note:
- the round parentheses
- the curly braces and
- the three equal signs!

Also, where Python uses and, or and not, JavaScript uses &&, || and !

Type coercion

Python is a strongly-typed language, whereas JavaScript is weakly-typed.

<table>
<thead>
<tr>
<th>Python (they all produce error messages)</th>
<th>JavaScript (they all do type coercions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 'abc' + 12</td>
<td>x = 'abc' + 12;</td>
</tr>
<tr>
<td>x = 'abc' - 12</td>
<td>x = 'abc' - 12;</td>
</tr>
</tbody>
</table>
| if 'abc' == 3:                           | if ('abc' == 3) {
    add something        |    # do something
    else:                 |    else {
    add something else    |        # do something else
}                                      |    }
| if '3' == 3:                            | if ('3' == 3) {
    add something        |    # do something
    else:                 |    else {
    add something else    |        # do something else
}                                      |    }
Type coercion
- JavaScript's type coercion is bizarre and causes many programming errors.
  - charlieharvey.org.uk/page/javascript_the_weird_parts
  - wtfjs
- Avoid JavaScript type coercion in equality tests by using identity (===), instead of equality (==)
  - Then in the JavaScript on the previous slide, both tests would be false

JavaScript objects
- At their simplest, objects in JavaScript are bundles of comma-separated properties, e.g.:
```javascript
let twinA = {
  firstname: 'John',
  surname: 'Grimes',
  age: 25
};
let twinB = {
  firstname: 'Edward',
  surname: 'Grimes',
  age: 25
};
```
- To refer to an object's properties, use the dot notation, e.g. twinA.firstname

Lists and arrays
The 'equivalent' of a Python list is a JavaScript array
<table>
<thead>
<tr>
<th>Python</th>
<th>JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>groceries = ['eggs', 'milk', 'tea']</td>
<td>let groceries = ['eggs', 'milk', 'tea'];</td>
</tr>
<tr>
<td>len(groceries)</td>
<td>groceries.length</td>
</tr>
<tr>
<td>groceries.append('bread')</td>
<td>groceries.push('bread')</td>
</tr>
</tbody>
</table>

(JavaScript arrays are very similar to Python lists, but not so similar to arrays in languages such as C or Java)

for loops
- Using a loop to 'visit' each item in a list or array:
```javascript
for (let item of groceries) {
  console.log(item);
}
```
- Using a loop to count, e.g. from 0 to 9 inclusive:
```javascript
for (let i = 0; i < 10; i += 1) {
  console.log(i);
}
```
A new version of particles.js

```javascript
let canvas;
let context;
let width;
let height;
let ps = [];
let gravity = 0.5;
document.addEventListener('DOMContentLoaded', init, false);

function init() {
  canvas = document.querySelector('canvas');
  context = canvas.getContext('2d');
  width = canvas.width;
  height = canvas.height;
  window.setInterval(draw, 33);
}

function draw() {
  for (let i = 0; i < 30; i++) {
    let p = {
      x: 250,
      y: 150,
      size: 10,
      xChange: getRandomNumber(-10, 10),
      yChange: getRandomNumber(-10, 10)
    };
    ps.push(p);
  }
  context.clearRect(0, 0, width, height);
  context.fillStyle = 'yellow';
  for (let p of ps) {
    context.fillRect(p.x, p.y, p.size, p.size);
    p.x = p.x + p.xChange;
    p.y = p.y + p.yChange;
    p.yChange = p.yChange + gravity;
  }
}

function getRandomNumber(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
```