Software Explained Simply

Episode 4 - Loops

Outline

- Loops

Loops

- Allow us to do repetitive tasks

Loops

```
x = [1, 2, 3, 4]

puts x[0] => 1

puts x[1] => 2

puts x[2] => 3

puts x[3] => 4
```

Loops

```
x = [1, 2, 3, 4,...99, 100]

puts x[0] => 1

puts x[1] => 2

puts x[2] => 3

puts x[3] => 4

…

puts x[99]  => 100
```

Loops

```
x = [1, 2, 3, 4,...999_999, 1_000_000]

puts x[0] => 1

puts x[1] => 2

puts x[2] => 3

puts x[3] => 4

…

puts x[999_999]  => 1_000_000
```

Loops

```
x = [???]
```

Loops

- **<u>For each</u>** item in a collection, do *something*

Traditional Loops

```
i++
```

```
i += 1
```

```
i = i + 1
```

Traditional Loops

```
for (initial; while; change) {
   // do stuff in here
}
```

Traditional Loops

(in JavaScript)

```
var i = 1;
for (i; i < 4; i++) {
  console.log(i);
}
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 1
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 1

i < 4 => true
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 1

i < 4 => true

prints 1
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 1

i < 4 => true

prints 1

i++ => i = 2
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 1

i < 4 => true

prints 1

i++ => i = 2

*loop*
```

Traditional Loops

(in JavaScript)

```
var i = 1;                          i = 2

for (i; i < 4; i++) {

  console.log(i);

}
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 2

i < 4 => true
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 2

i < 4 => true

prints 2
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 2

i < 4 => true

prints 2

i++ => i = 3
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 2

i < 4 => true

prints 2

i++ => i = 3

*loop*
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 3

i < 4 => true

prints 3

i++ => i = 4

*loop*
```

Traditional Loops

(in JavaScript)

```
var i = 1;                          i = 4

for (i; i < 4; i++) {

  console.log(i);

}
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 4

i < 4 => false
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}
```

```
i = 4

i < 4 => false

*exit loop*
```

Traditional Loops

(in JavaScript)

```
var i = 1;

for (i; i < 4; i++) {

  console.log(i);

}

=> 1

=> 2

=> 3
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

x[0] = 1;

x[3] = 4;
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

x[0] = 1;

x[3] = 4;

_

var i = 0;

for (i; i < 4; i++) {

  console.log(x[i]);

}
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

var i = 0;

for (i; i < 4; i++) {
  console.log(x[i]);
}

=> 1

=> 2

=> 3

=> 4
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

var i = 0;

for (i; i < 4; i++) {
  console.log(x[i]);
}
```

```
=> 1

=> 2

=> 3

=> 4
```

```
for (i; i <= 3; i++)
```

Traditional Loops

(in JavaScript)

```
var x = [???];
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

x.length => 4
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

x.length => 4

for (i; i < 4; i++) {
  console.log(x[i]);
}
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

x.length => 4

for (i; i < x.length; i++) {
  console.log(x[i]);
}
```

Traditional Loops

(in JavaScript)

```
var x = [???];


for (i; i < x.length; i++) {
  console.log(x[i]);
}
```

Traditional Loops

(in JavaScript)

```
var x = [???];


for (i; i < x.length; i++) {
  console.log(x[i]);
}
```

```
for (i; i <= x.length - 1; i++)
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

var i = 0;

for (i; i < x.length; i++) {

  if (x[i] % 2 == 0) {

    console.log(x[i]);

  }

}
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

var i = 0;

for (i; i < x.length; i++) {

  if (x[i] % 2 == 0) {

    console.log(x[i]);

  }

}

=> 2

=> 4
```

Traditional Loops

(in JavaScript)

```
var i = 0;
for (i; i < x.length; i++)

_

for (var i = 0; i < x.length; i++)
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];

var y = [5, 6, 7, 8];


for (var i = 0; i < x.length; i++) {
  for (var j = 0; j < y.length; j++) {
    console.log("i = " + i + ", j = " + j);
  }
}
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];
var y = [5, 6, 7, 8];

for (var i = 0; i < x.length; i++) {
  for (var j = 0; j < y.length; j++) {
    console.log("i = " + i + ", j = " + j);
  }
}
```

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 0, j = 3
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 3, j = 0
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

Traditional Loops

(in JavaScript)

```
var x = [1, 2, 3, 4];
var y = [5, 6, 7, 8];

for (var i = 0; i < x.length; i++) {
  for (var j = 0; j < y.length; j++) {
    console.log("x = " + x[i] + ", y = " + y[j]);
  }
}
```

```
x = 1, y = 5
x = 1, y = 6
x = 1, y = 7
x = 1, y = 8
x = 2, y = 5
x = 2, y = 6
x = 2, y = 7
x = 2, y = 8
x = 3, y = 5
x = 3, y = 6
x = 3, y = 7
x = 3, y = 8
x = 4, y = 5
x = 4, y = 6
x = 4, y = 7
x = 4, y = 8
```

But there is a better way

Iterators

(back to Ruby)

```
x = [1, 2, 3, 4]
```

Iterators

(back to Ruby)

```
x = [1, 2, 3, 4]

x.each do |n|

  puts n

end
```

Iterators

(back to Ruby)

```
x = [1, 2, 3, 4]

x.each do |n|

  puts n

end

=> 1

=> 2

=> 3

=> 4
```

Iterators

(back to JavaScript)

```
var x = [1, 2, 3, 4];

x.forEach(function(n) {

  console.log(n);

});

=> 1

=> 2

=> 3

=> 4
```

While Loops

(back to JavaScript)

```
var i = 0;

while (i < 5) {
   console.log(i);

   i++;
}
```

While Loops

(back to JavaScript)

```
var i = 0;

while (i < 5) {
  console.log(i);

  i++;
}
```

```
var i = 0;

for (i; i < 5; i++) {
  console.log(i);
}
```

Episode Recap

- Loops allow us to do repetitive tasks easily
- Traditional loops use the `for (var x = 0; x < )`… structure
- Modern languages/frameworks have nice iterators built on top them (`.each` and `.forEach`)

Homework

- Practice iterating over arrays and printing their values
- Iterate over an array of numbers and do the following:
  - Print out the odd numbers
  - Add up all the values (hint: use another variable)
- Iterate every value in a nested array
  - Ex: `[[1, 2, 3], [4, 5, 6]]`
- Do these exercises with the traditional loops and the modern iterators

Thanks!

@johnmosesman