SEC201.2 Web-Based Programming

JavaScript: Advanced Topics - I

Outline

- More on Loops
  - For Loops
  - Loop Control
- More on Arrays
  - Array Functions

More on Loops

- There are several types of **for** loops
  - for
  - for … in
  - for … of

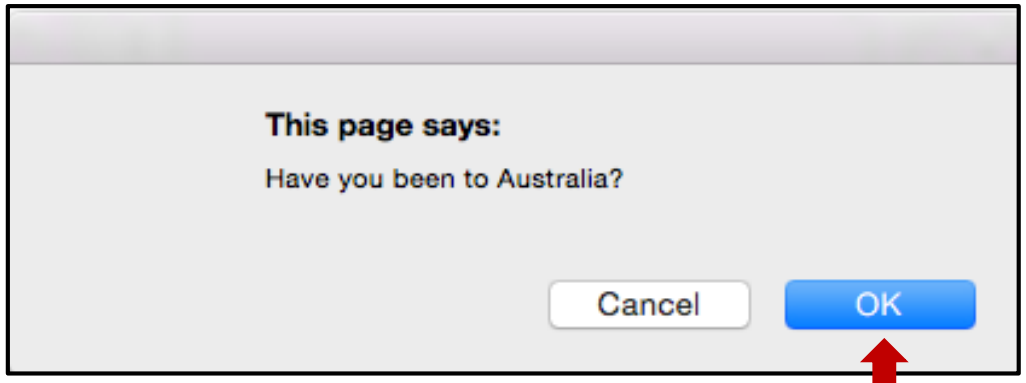
- **for** clearly shows the start and end values all in one line
- **for** is especially good for handling <u>a series of data</u>, for example in an array
  - What if you have a data structure, such as an array and you want to know how long, or how many items are in an array?

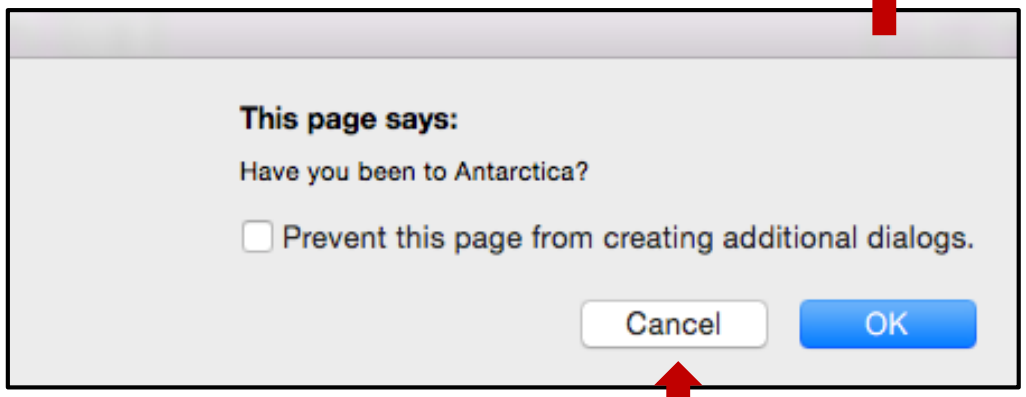
    data_structure.length → tells how many items **data_structure** contains

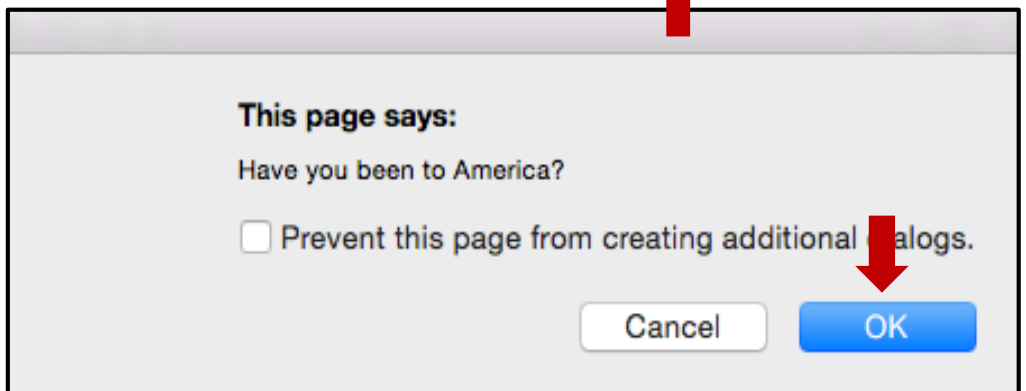
Example: FOR Loop

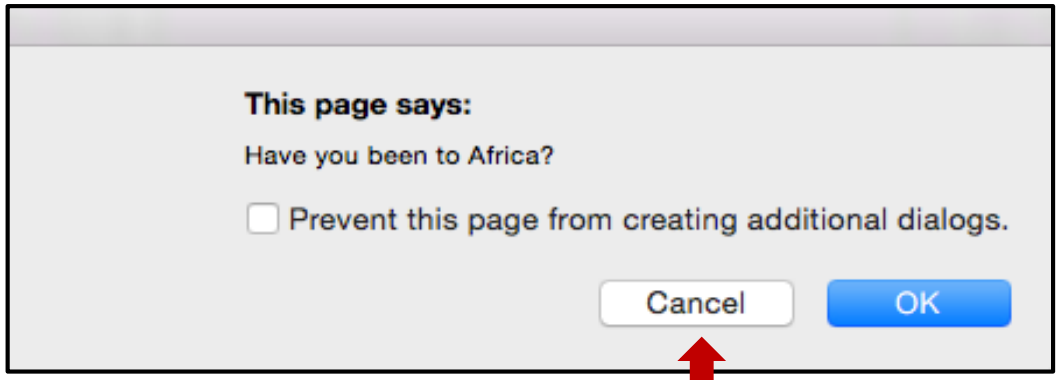
```
<!doctype html>
<html>
    <head>
        <script>
            var continents = ["Australia", "Africa", "Antarctica", "Eurasia", "America"];
            var response, count = 0;

            for (var index=0; index < continents.length; index++) {
                response = confirm("Have you been to " + continents[index] + "?");
                if (response) count++;
            }
            alert("You have been to " + count + " continents!");
        </script>
    </head>
</html>
```

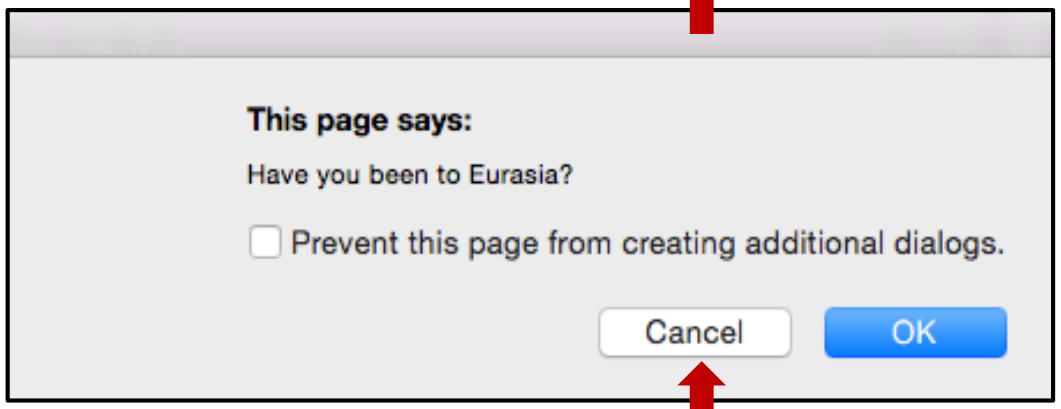
Example: FOR Loop - Output

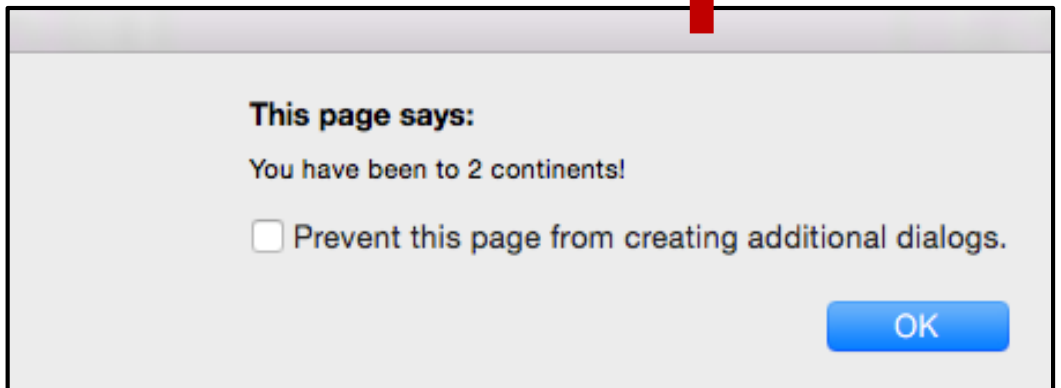












for … in Loops

- **for … in** gives you the index of each item
- Let's rewrite the previous example with **for … in** loop
  - look nicer, easier to type, no semicolons, easier to manage

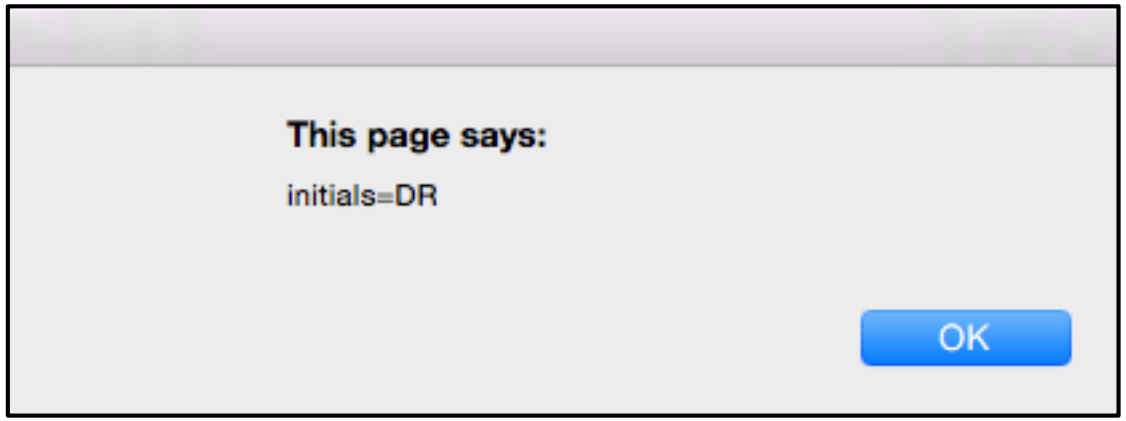
```
<!doctype html>
<html>
    <head>
        <script>
            var continents = ["Australia", "Africa", "Antarctica", "Eurasia", "America"];
            var response, count=0;

            for (var index in continents) {

                response = confirm("Have you been to " + continents[index] + "?");
                if (response) count++;
            }
            alert("You have been to " + count + " continents!");
        </script>
    </head>
</html>
```

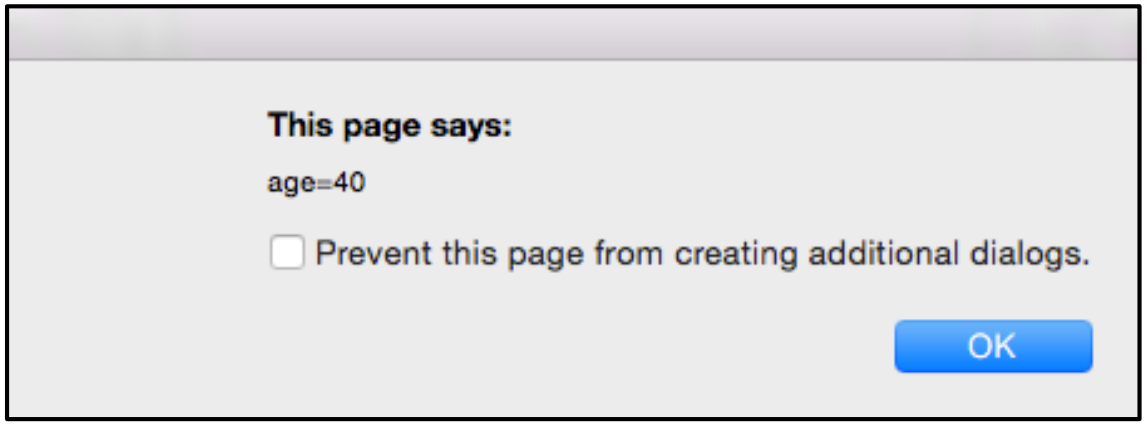
Example: FOR … IN Loop

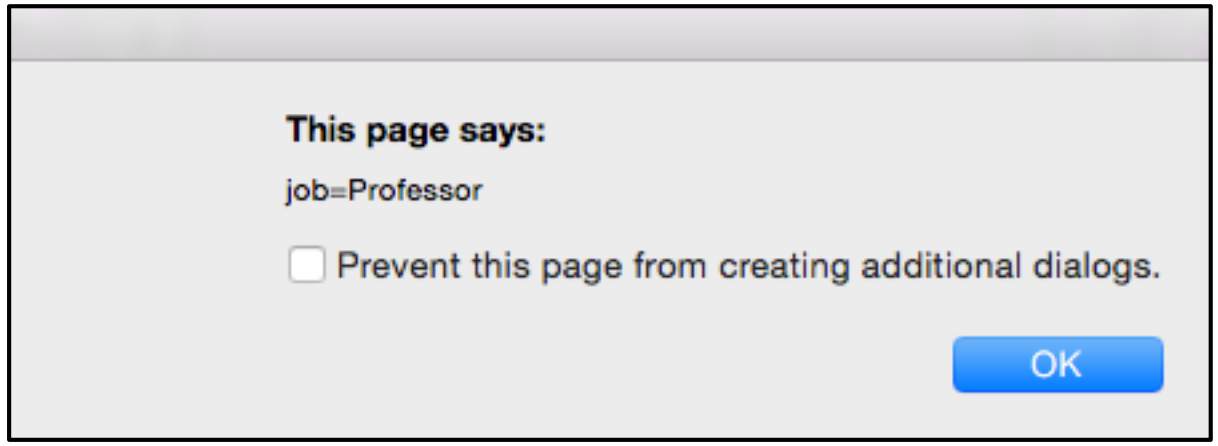
This example shows how **for … in** can be used to access the content of a data structure

```
<!doctype html>
<html>
    <head>
        <title>Example of for in</title>
        <script>
            var response, count = 0;
            var onePerson = { initials:"DR", age:40, job:"Professor" };

            for (var property in onePerson) {
                alert(property + "=" + onePerson[property]);
            }
        </script>
    </head>
</html>
```

Example: FOR … IN Loop - Output







for … of Loops

- **for … of** gives you each item
- Again, let's rewrite the previous example with **for … of** loop
  - look nicer, easier to type, no semicolons, easier to manage

```
<!doctype html>
<html>
    <head>
        <title>Example of for of</title>
        <script>
            var continents = ["Australia", "Africa", "Antarctica", "Eurasia", "America"];
            var response, count = 0;

            for (var continent of continents) {
                response = confirm("Have you been to " + continent + "?");
                if (response) count++;
            }
            alert("You have been to " + count + " continents!");
        </script>
    </head>
</html>
```

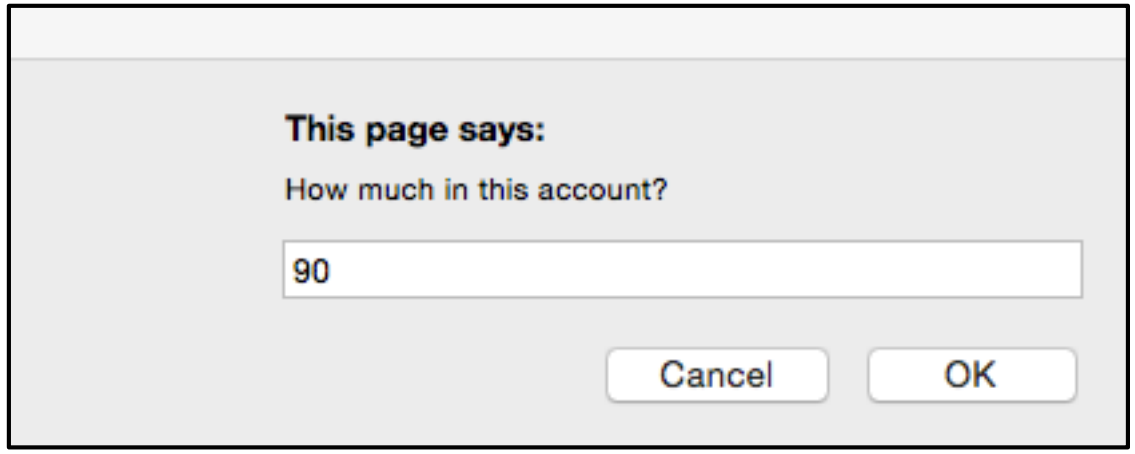
Loop Control

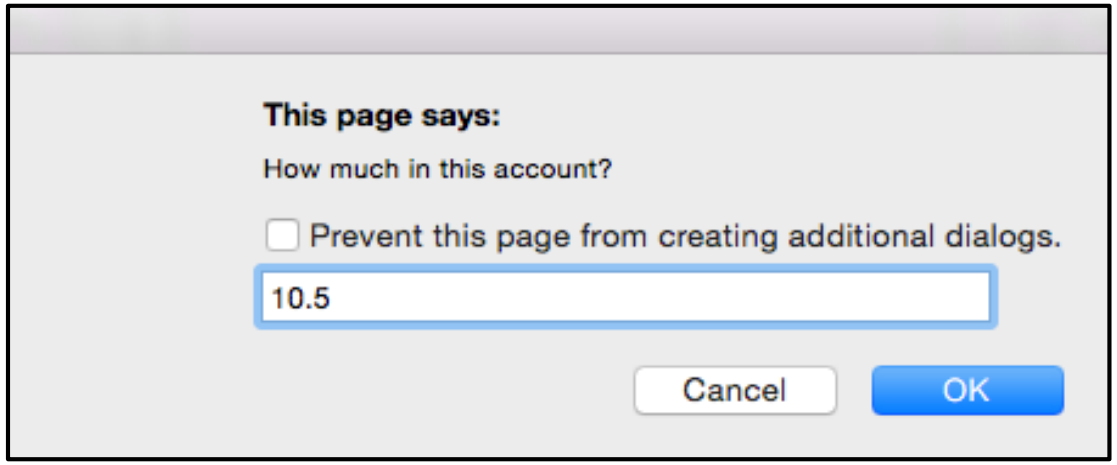
- Two ways to control loops in JavaScript
  - **break** totally stops the loop
  - **continue** stops the current iteration of the loop
    - It doesn't permanently stop the loop
    - It just tells the browser to go to the next loop

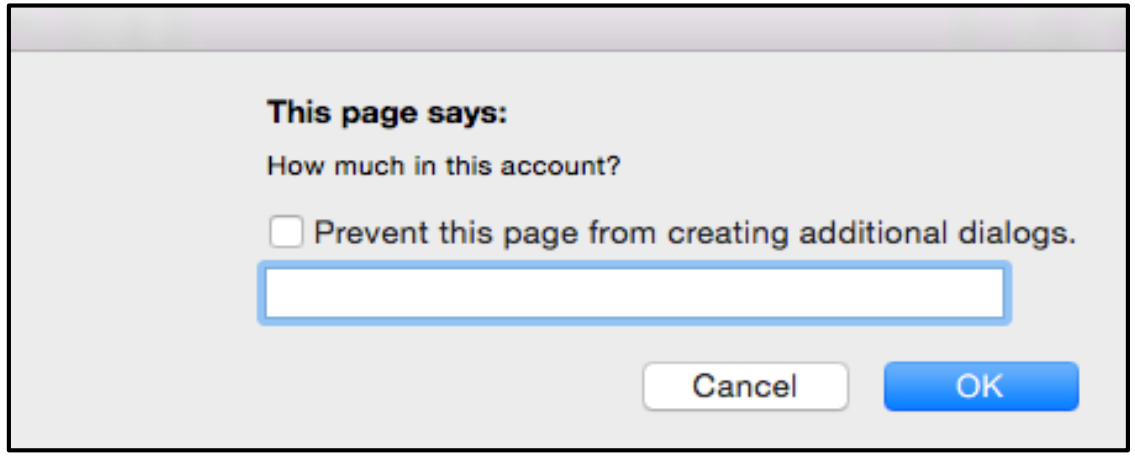
Example: BREAK

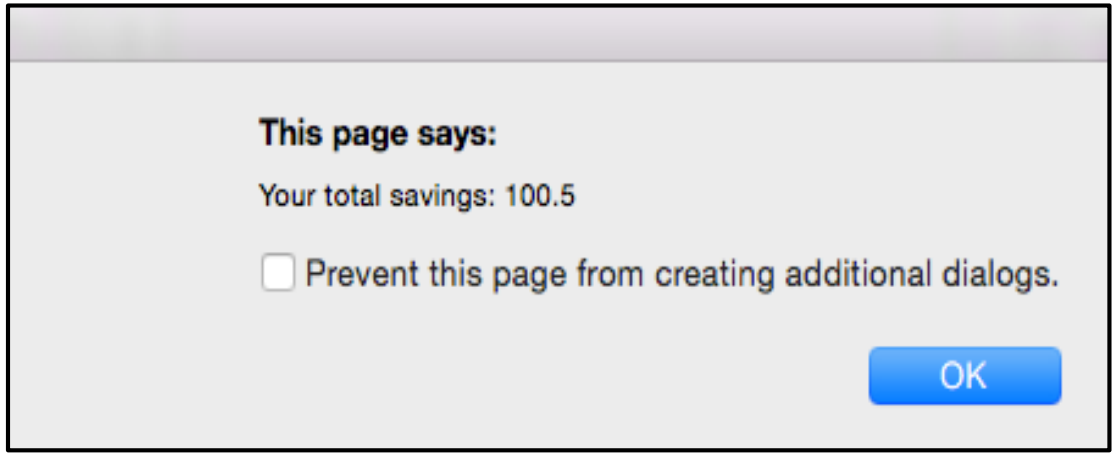
```
<!doctype html>
<html>
    <head>
        <script>
            var total_amount = 0;

            while (true) {
                this_amount = prompt("How much in this account?");
                this_amount = parseFloat(this_amount);
                if (this_amount > 0)
                    total_amount += this_amount;
                else
                    break;
            }
            alert("Your total savings: " + total_amount);
        </script>
    </head>
</html>
```

Example: BREAK - Output







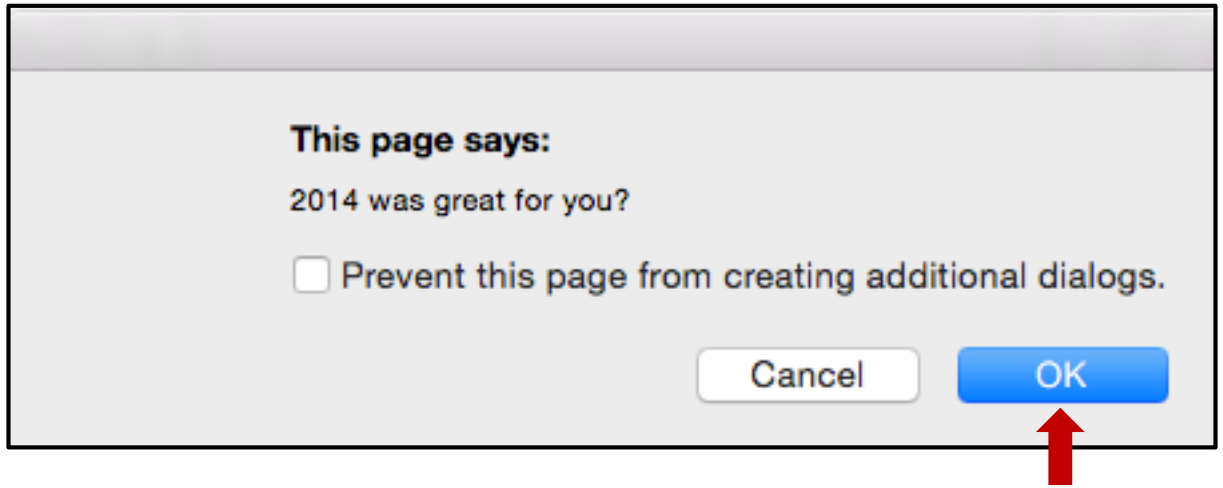


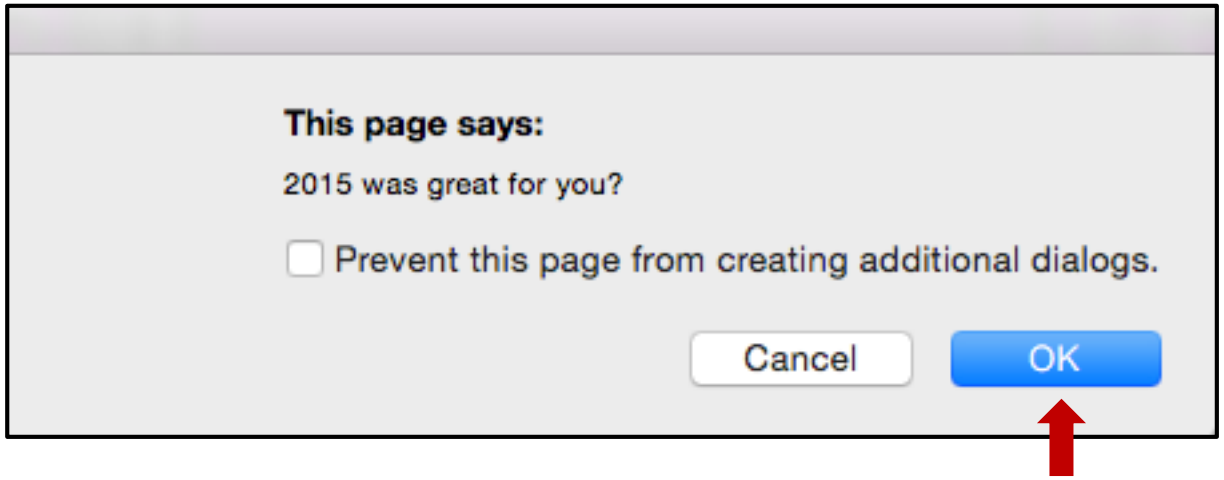
Example: CONTINUE

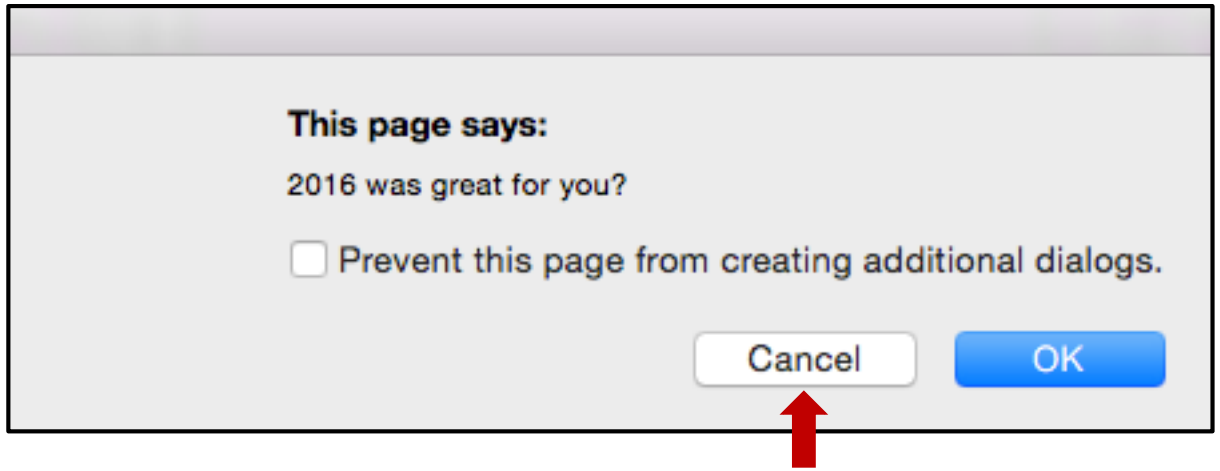
Remember: **array.push()** adds an item to the end of array

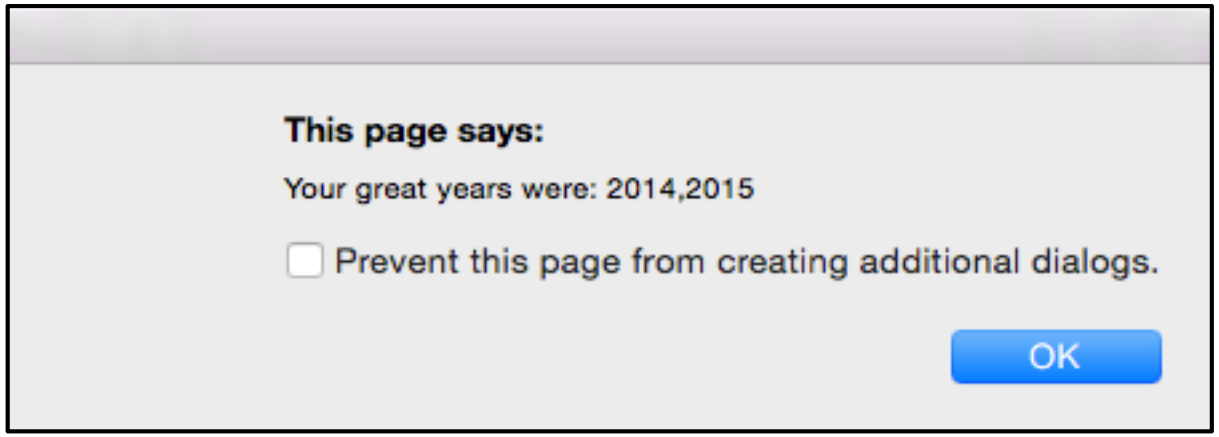
```
<!doctype html>
<html>
    <head>
        <script>
            var year, great_years = [];
            for (year = 2014; year <= 2016; year ++) {
                correct = confirm(year + " was great for you?")
                if (!correct) continue;
                great_years.push(year);
            }

            alert("Your great years were: " + great_years);
        </script>
    </head>
</html>
```

Example: CONTINUE - Output









More on Arrays

- Advanced array functions
  - sort()
  - reverse()
  - indexOf()
  - lastIndexOf()
  - slice()
  - splice()

Sorting and Reverse

- ***array.sort()*** sorts the elements in array

```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
pets.sort();
// Now pets is ["Cat", "Dog", "Hamster", "Rabbit"]
```

- ***array.reverse()*** reverses array
  - The first element becomes the last element;
  - The last element becomes the first element

```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
pets.reverse();
// Now pets is ["Hamster", "Rabbit", "Cat", "Dog", ]
```

- By combining **sort()** and **reverse()**, you can sort things in ***descending order***

```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
pets.sort().reverse();
// Now pets is ["Rabbit", "Hamster", "Dog", "Cat", ]
```

Finding An Element

- ***array.indexOf(target)*** to find the <u>index of</u> the **first** occurrence of ***target*** in array
- If ***target*** is <u>not</u> in array → `indexOf()` will return -1

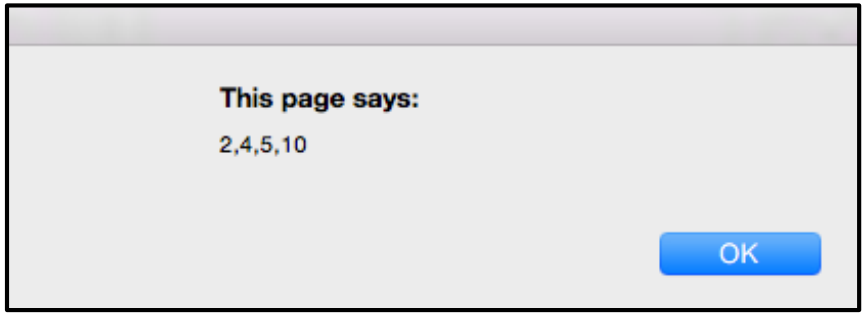
```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
alert(pets.indexOf("Rabbit")); // This will show 2
alert(pets.indexOf("Bird"));   // This will show -1
```

- Pass a second value to `indexOf()` to control where to start the search

  array.indexOf(target, startPosition)

Example: indexOf()

```
<!doctype html>
<html>
    <body>
        <script>
            var pets = ["Dog", "Cats", "Rabbit", "Hamster",
                        "Rabbit", "Rabbit", "Dog", "Cat",
                        "Hamster", "Hamster", "Rabbit"];

            var rabbitPositions = [], startSearchAt = 0;

            do {
                foundAt = pets.indexOf("Rabbit", startSearchAt);
                if(foundAt != -1) {
                  rabbitPositions.push(foundAt);
                  startSearchAt = foundAt + 1;
                }
            } while(foundAt != -1);

            alert(rabbitPositions);
        </script>
    </body>
</html>
```



Finding Element Backwards

- ***array.lastIndexOf(target)*** to find ***target*** in array, starting from the last element in array

```
var pets = ["Rabbit", "Dog", "Cat", "Rabbit", "Hamster"];
alert(pets.lastIndexOf("Rabbit")); // This will show 3
```

SLICE()

- The slice() method returns the selected elements in an array, as a new array object
- Extract part of an array by ***array.slice(startPosition)***

```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
var result = pets.slice(1);
// result is ["Cat", "Rabbit", "Hamster"]
```

- You can also set where to stop, by ***array.slice(startPosition, endPosition)***

```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
var result = pets.slice(1, 3);
// result is ["Cat", "Rabbit"]
```

- The slice() method selects the elements starting at the given *start* argument, and ends at, *but does not include*, the given *end* argument

Remove Something Anywhere in an Array

- **splice()** is used when you want to **remove element(s)** anywhere from an array
- To remove element(s) anywhere from an array, use ***array.splice(position, quantity)***

```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
var result = pets.splice(1, 1);
// Now pets is ["Dog", "Rabbit", "Hamster"]
// and result is ["Cat"]
```

- **splice()** returns the removed element(s)

Add Something Anywhere in an Array

- **splice()** can also be used when you want to **add element(s)** anywhere to an array
- To add an element anywhere to an array, use ***array.splice(position, 0, element)***

```
var pets = ["Dog", "Cat", "Hamster"];
var result = pets.splice(2, 0, "Rabbit");
// Now pets is ["Dog", "Cat", "Rabbit", "Hamster"]
// and result is []
```

- If ***0*** is set, then means no items will be removed
- Because nothing is removed from pets, result is []

Replace Something in an Array

- To **replace element(s)** anywhere in an array, use

  array.splice(position, quantity, element(s))

```
var pets = ["Dog", "Cat", "Hamster"];
var result = pets.splice(1, 1, "Rabbit", "Fish");
// Now pets is ["Dog", "Rabbit", "Fish", "Hamster"]
// and result is ["Cat"]
```

https://www.w3schools.com/jsref/jsref_splice.asp

Array Functions

- **FOREACH()**
  - You can go through every element using loop (for / while)

```
var pets = ["Dog", "Cat", "Hamster"];
for (var i = 0; i < pets.length; i++) {
    alert(pets[i]);
}
```

  - You can also use ***array.forEach(function)***

```
var pets = ["Dog", "Cat", "Hamster"];
pets.forEach(alert);
// This will show 3 separate alerts
```

More on FOREACH()

- The **forEach()** method calls a provided function (a callback function) once for each array element in an array, in order
- You can think of **forEach()** in this way

```
function forEach(theArray, fn) {
    for (var i = 0; i <theArray.length; i++) {
        fn(theArray[i], i, theArray);
    }
}
```

- `fn(theArray[i], i, theArray);` A function to be run for each element in the array (Required)

Function arguments:

Argument	Description
currentValue	Required. The value of the current element
index	Optional. The array index of the current element
arr	Optional. The array object the current element belongs to

https://www.w3schools.com/jsref/jsref_foreach.asp

Example: FOREACH()

```
<!doctype html>
<html>
    <body>
        <script>
            var numbers = [1, 2, 3, 4, 5];

            numbers.forEach( function(elem, idx, arr) {
                                arr[idx] = elem * elem;
                            });

            alert(numbers); // This shows [1,4,9,16,25];
        </script>
    </body>
</html>
```

Array Functions

- **MAP()**
  - ***array.map(function)*** stores the result of each execution of function into a new array it returns
  - The **map()** method creates a new array with the results of calling a function for every array element
  - The **map()** method calls the provided function once for each element in an array, in order
  - You can think of ***map(function)*** as

```
function map(theArray, fn) {
    var results = [];
    for (var i = 0; i <theArray.length; i++) {
        results.push(fn(theArray[i], i, theArray));
    }
}
```

  - Same as in FOREACH()

```
fn(theArray[i], i, theArray);
```

A function to be run for each element in the array (Required)

Example: MAP()

```
<!doctype html>
<html>
    <body>
        <script>
            var square = function(el) { return el * el; }
            var numbers = [1, 2, 3, 4, 5];
            var results = numbers.map(square);

            alert(results); // This shows [1,4,9,16,25];
        </script>
    </body>
</html>
```

Note: *A function expression can be stored in a variable*

Note: *After a function expression has been stored in a variable, the variable can be used as a function (next slide)*

Notes on JS Functions

- Functions can be used as values

```
var x = function (a, b) {return a * b};
```

- After a function expression has been stored in a variable, the variable can be used as a function

```
var z = x(4, 3);
```

- The function above is an ***anonymous function*** (a function without a name)
- Functions stored in variables do not need function names
- They are always invoked (called) using the variable name