Comp 125 - Visual Information Processing

Spring Semester 2018 - week 4 - monday

Dr Nick Hayward

JS Data Structures - arrays - practical abstraction & usage

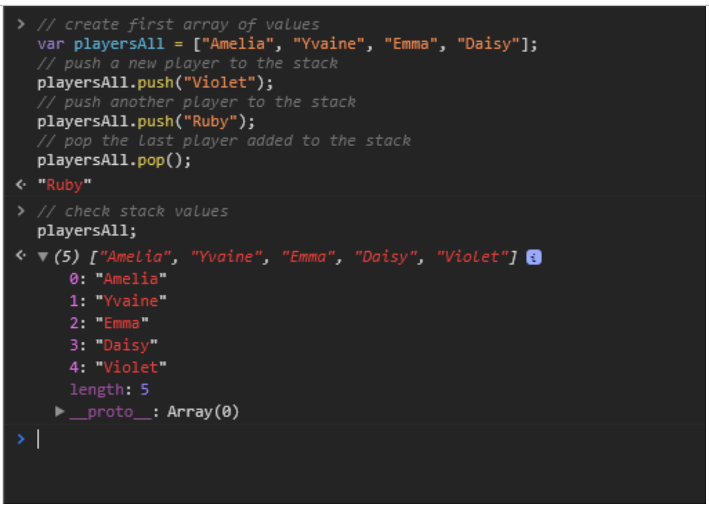
example 1 - create a stack

- many practical uses for an array data structure
- common use is a **stack** to store a sequence of data
- a **stack** stores data in a known, predictable pattern and order
  - *last data in the stack will be the first data out*
- use the following acronym,
  - ***LIFO** - Last In, First Out*
  - *use `push()` and `pop()` methods to create **LIFO**...*

JS Data Structures - arrays - practical abstraction & usage

example 1 - create a stack

use push() and pop() methods to create **LIFO**...



```
> // create first array of values
  var playersAll = ["Amelia", "Yvaine", "Emma", "Daisy"];
  // push a new player to the stack
  playersAll.push("Violet");
  // push another player to the stack
  playersAll.push("Ruby");
  // pop the last player added to the stack
  playersAll.pop();
<· "Ruby"
> // check stack values
  playersAll;
<· ▼(5) ["Amelia", "Yvaine", "Emma", "Daisy", "Violet"]
      0: "Amelia"
      1: "Yvaine"
      2: "Emma"
      3: "Daisy"
      4: "Violet"
      length: 5
    ▶ __proto__: Array(0)
> |
```

JavaScript - arrays - create a stack

JS Data Structures - arrays - practical abstraction & usage

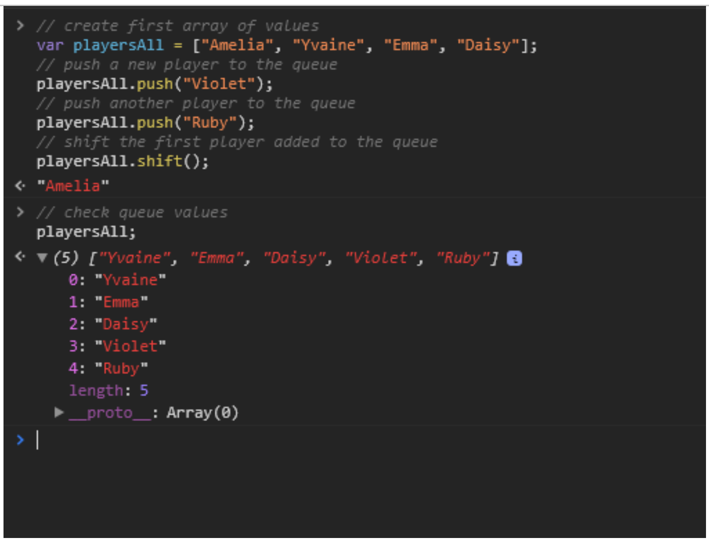
example 2 - create a queue

- also create the opposite of a stack with a **queue**
- like a stack, a **queue** uses a predictable pattern and order
- first data in the queue will be the first data out
  - *use the following acronym,*
  - ***FIFO** - First In, First Out*
- use `push()` and `shift()` methods to create **FIFO**...

JS Data Structures - arrays - practical abstraction & usage

example 2 - create a queue

use `push()` and `shift()` methods to create **FIFO**...



```
> // create first array of values
  var playersAll = ["Amelia", "Yvaine", "Emma", "Daisy"];
  // push a new player to the queue
  playersAll.push("Violet");
  // push another player to the queue
  playersAll.push("Ruby");
  // shift the first player added to the queue
  playersAll.shift();
<· "Amelia"
> // check queue values
  playersAll;
<· ▼(5) ["Yvaine", "Emma", "Daisy", "Violet", "Ruby"]
      0: "Yvaine"
      1: "Emma"
      2: "Daisy"
      3: "Violet"
      4: "Ruby"
      length: 5
    ▶ __proto__: Array(0)
> |
```

JavaScript - arrays - create a stack

Fun exercise - using arrays

- create a new array, named **cities**, with the following values
  - *Paris, Marseille, Nice*
- add the following values to the end of the array
  - *Toulouse, Lyon*
- remove the fourth value from the array
- add the following values to the start of the array
  - *Cannes, Avignon*
- move the third value in the array to the end of the array
- move the fourth value in the array to the start of the array

Output each answer to the document with a line break between each result.

JS Objects - intro

- **object** type includes a compound value
  - *use to set properties, or named locations*
  - *property is an association between **name (or key)** and its value*
  - *`name: value` or `key: value`*
- each of these properties holds its own value
  - *value can be defined as any type*

```
// declear variable - store object literal
var objectA = {
  a: 49,
  b: 59,
  c: "Philae"
};
```

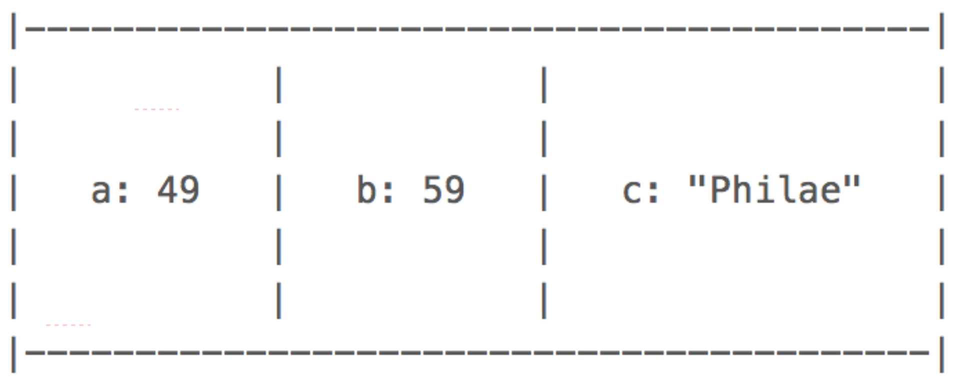
- object literal
  - *curly brackets and everything in between*
- object stores **name:value** (**key:value**) pair/s
  - *quotation marks around property names is optional*
  - *JS knows each name will be string...*
    - quotation marks only needed for multiple words, e.g.

```
var testObject = {
    "Temple Sites": {
        name: "Philae"
    }
}
```

- access these values using either **dot** or **bracket** notation

```
//dot notation
objectA.a;
//bracket notation
objectA["a"];
```

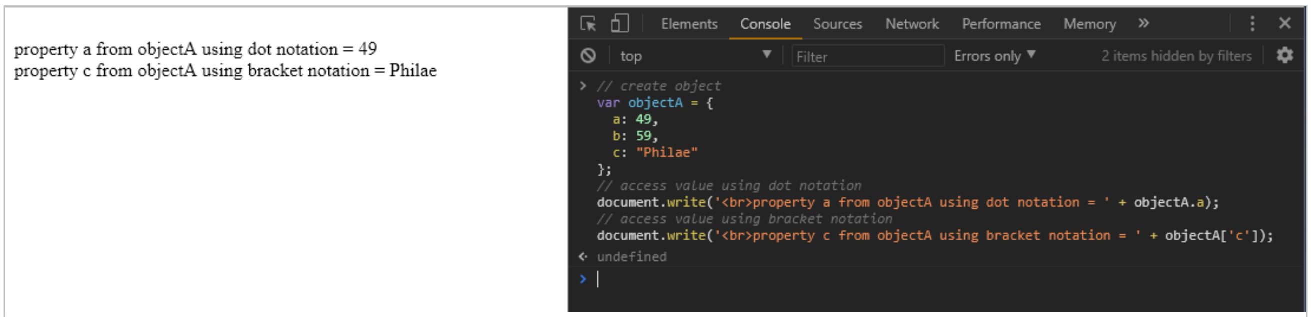
JS Objects - object structure



```
|----------------------------------|
|         |         |              |
|         |         |              |
|  a: 49  |  b: 59  |  c: "Philae" |
|         |         |              |
|         |         |              |
|----------------------------------|
```

JS Object structure

JS Objects - example output



```
property a from objectA using dot notation = 49
property c from objectA using bracket notation = Philae
```

```
> // create object
  var objectA = {
    a: 49,
    b: 59,
    c: "Philae"
  };
  // access value using dot notation
  document.write('<br>property a from objectA using dot notation = ' + objectA.a);
  // access value using bracket notation
  document.write('<br>property c from objectA using bracket notation = ' + objectA['c']);
< undefined
> |
```

JS Object - example output

JS Objects - example

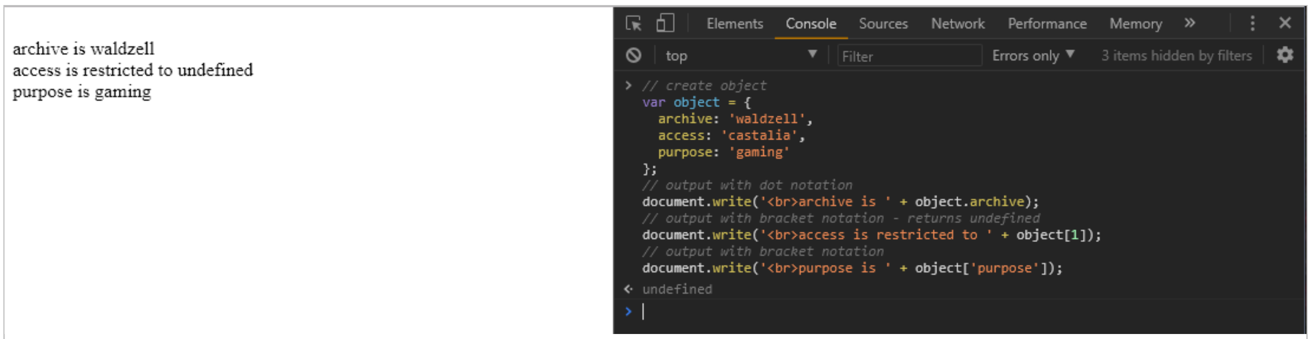
```
// create object
var object = {
  archive: 'waldzell',
  access: 'castalia',
  purpose: 'gaming'
};

// output with dot notation
document.write('<br>archive is ' + object.archive);

// output with bracket notation - returns undefined
document.write('<br>access is restricted to ' + object[1]);

// output with bracket notation
document.write('<br>purpose is ' + object['purpose']);
```

JS Objects - example output



```
archive is waldzell
access is restricted to undefined
purpose is gaming
```

```
> // create object
  var object = {
    archive: 'waldzell',
    access: 'castalia',
    purpose: 'gaming'
  };
  // output with dot notation
  document.write('<br>archive is ' + object.archive);
  // output with bracket notation - returns undefined
  document.write('<br>access is restricted to ' + object[1]);
  // output with bracket notation
  document.write('<br>purpose is ' + object['purpose']);
< undefined
> |
```

JS Object - example output

JS Objects - all keys

- access single values using a specific key
  - *dot or bracket notation...*
  - *JS provides method to access all keys in passed object*
  - *e.g. using `Object.keys()` method*

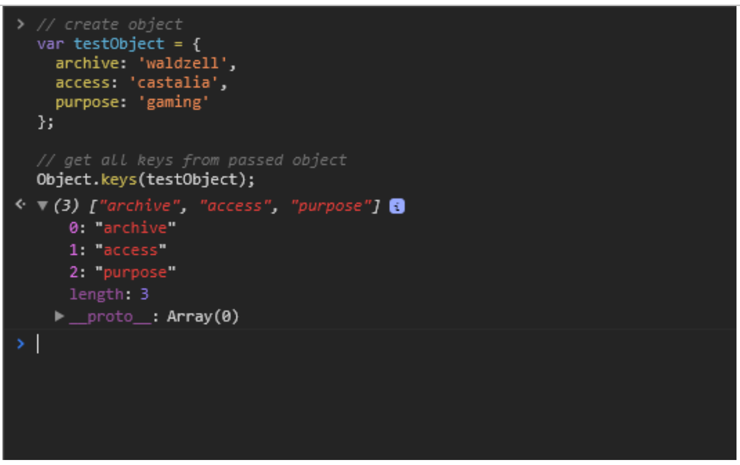
```
// create object
var testObject = {
  archive: 'waldzell',
  access: 'castalia',
  purpose: 'gaming'
};

// get all keys for passed object
Object.keys(testObject);
```

- `keys()` method returns an array of keys for `testObject`

JS Objects - all keys

get all keys from the passed object...



```
> // create object
  var testObject = {
    archive: 'waldzell',
    access: 'castalia',
    purpose: 'gaming'
  };

  // get all keys from passed object
  Object.keys(testObject);
<· ▼(3) ["archive", "access", "purpose"]
      0: "archive"
      1: "access"
      2: "purpose"
      length: 3
    ▶ __proto__: Array(0)
> |
```

JS Object - get all keys

JS Objects - add values

- to add values to an object, we might need to start with an empty object

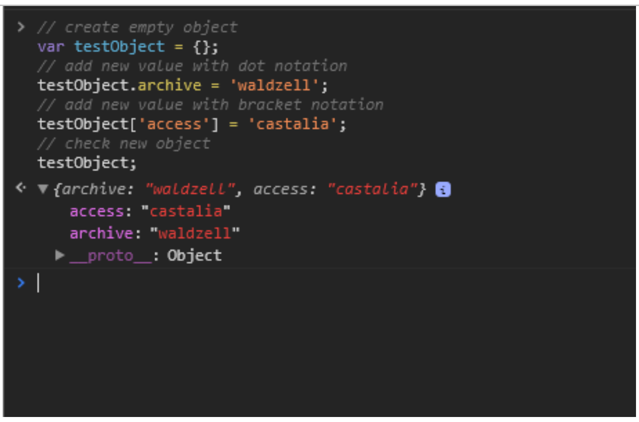
```
// create empty object
var testObject = {};
```

- uses same pattern as creating **array**
  - *{ } for object*
    - [ ] for array
  - *add single values to new object*

```
// create empty object
var testObject = {};
// add new value with dot notation
testObject.archive = 'waldzell';
// add new value with bracket notation
testObject['access'] = 'castalia';
```

JS Objects - add values

add some values to an empty object...



```
> // create empty object
  var testObject = {};
  // add new value with dot notation
  testObject.archive = 'waldzell';
  // add new value with bracket notation
  testObject['access'] = 'castalia';
  // check new object
  testObject;
< ▼{archive: "waldzell", access: "castalia"}
      access: "castalia"
      archive: "waldzell"
    ▶ __proto__: Object
> |
```

JS Object - add some values

JS Objects - get length of object

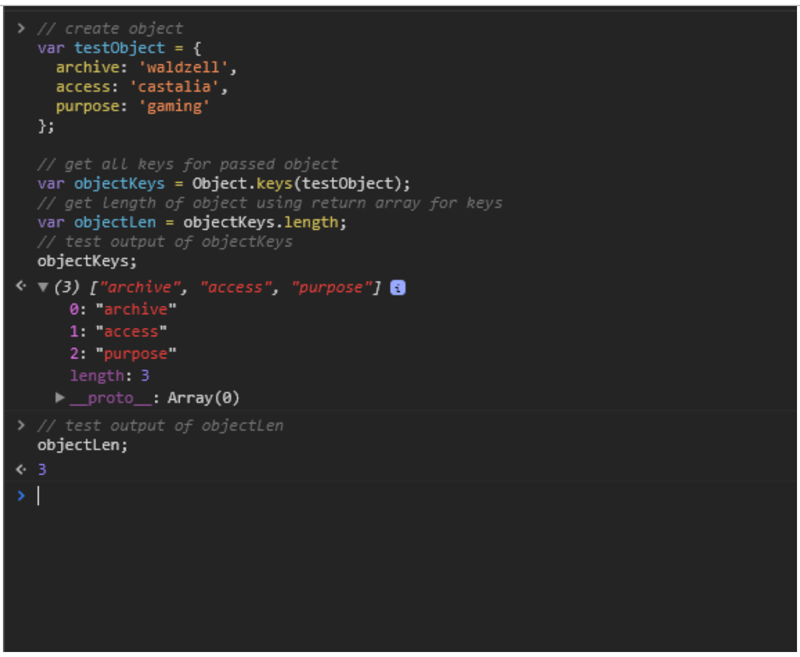
- an object does not include its own `length` property
  - *but array includes the `length` property*
  - *we can use `keys()` method to get array of keys*
  - *then get `length` from keys array for passed object*

```
// create object
var testObject = {
  archive: 'waldzell',
  access: 'castalia',
  purpose: 'gaming'
};

// get all keys for passed object
var objectKeys = Object.keys(testObject);
// get length of object using return array for keys
var objectLen = objectKeys.length;
```

JS Objects - get length of object - v.1

use `keys()` and array `length` property...return `keys` array and length of object



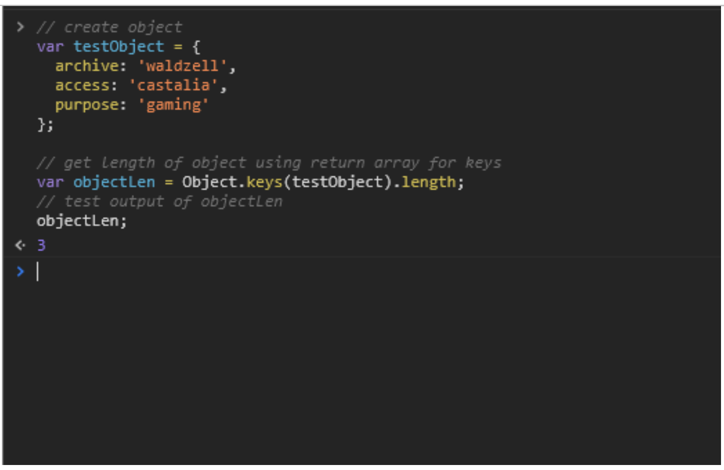
```
> // create object
  var testObject = {
    archive: 'waldzell',
    access: 'castalia',
    purpose: 'gaming'
  };

  // get all keys for passed object
  var objectKeys = Object.keys(testObject);
  // get length of object using return array for keys
  var objectLen = objectKeys.length;
  // test output of objectKeys
  objectKeys;
< ▼ (3) ["archive", "access", "purpose"]
      0: "archive"
      1: "access"
      2: "purpose"
      length: 3
    ▶ __proto__: Array(0)
> // test output of objectLen
  objectLen;
< 3
> |
```

JS Object - get object length

JS Objects - get length of object - v.2

use `keys()` and array `length` property...only return length of object



```
> // create object
  var testObject = {
    archive: 'waldzell',
    access: 'castalia',
    purpose: 'gaming'
  };

  // get length of object using return array for keys
  var objectLen = Object.keys(testObject).length;
  // test output of objectLen
  objectLen;
<· 3
> |
```

JS Object - get object length

JS Objects - arrays as objects

- JS array an object that contains values, of any type, in numerically indexed positions
  - *store a number, a string...*
  - *array will start at index position 0*
  - *increments by **1** for each new value*
- arrays can also have properties
  - *eg: automatically updated **length** property*

```
var arrayA = [
  49,
  59,
  "Philae"
];
arrayA.length; //returns 3
```

- each value can be retrieved from its applicable index position,

```
arrayA[2]; //returns the string "Philae"
```

JS Objects - array structure

```
|-----------------------------------------|
|           |           |                 |
|           |           |                 |
|   0: 49   |   1: 59   |   2: "Philae"   |
|           |           |                 |
|           |           |                 |
|-----------------------------------------|
```

JS Array

References

- W3Schools - Objects and Properties
  - *MDN - Working with Objects*