CST 366 - Internet Programming

Spring 2020 - Week 2.1

Intro to JavaScript

Due Date

- Lab 2 due on Sunday 2/9 @ 11:59 PM PST
- Assignment 1 due on Sunday 2/9 @ 11:59PM PST

Primitive Datatypes

- **Numbers**: 4, 9.3, -10
- **Strings**: "Hello World”, "43"
- **Booleans**: true, false
- **null**
- **undefined**

Primitive Datatypes

- **Numbers**: 4, 9.3, -10
- **Strings**: "Hello World”, "43"
- **Booleans**: true, false
- **null**
- **undefined**

We can do some math with numbers:

- 4 + 10 // 14
- 1/5 // 0.2
- 10 % 3 // 1

Primitive Datatypes

- **Numbers**: 4, 9.3, -10
- **Strings**: "Hello World”, "43"
- **Booleans**: true, false
- **null**
- **undefined**

- String with single quote or double quotes
  ‘Hello world’, “Hello World”
- Concatenation
  “CST” + “ 336”
- Escape Characters start with “\”
  “This is a backslash: \\”
- String have a length property
  “hello”.length //5
- Access individual characters
  “hello”[0] //h
  “hello”[3] //l

Variables & Constants

let constant_name = Value
var variable_name = Value

- **var** department = “CST”;
  var course_number = “336”;
  var course = department + “ ” + course_number; // CST 336

- var num = 37;
  num+3+10; // 50

- let constant = 50;
  constant = 100 // Error Warning

Boolean Logics

Assuming x = 5

Operator	Name	Example	Result
>	Greater than	x > 10	false
>=	Greater than or equal to	x >= 5	true
<	Less than	x < -50	false
<=	Less than or equal to	x <= 100	true
==	Equal to	x == "5"	true
!=	Not equal to	x != "b"	true
===	Equal value and type	x === "5"	false
!==	Not equal value or equal type	x !== "5"	true

Boolean Logics

== vs. ===

```
var x = 99;

x == "99"  //true

x === "99" //false

var y = null;

y == undefined //true

y === undefined //false
```

Boolean Logics

== vs. ===

```
var x = 99;
x == "99"  //true
x === "99" //false

var y = null;
y == undefined //true
y === undefined //false
```

A few interesting cases

```
true == "1"          //true
0 == false           //true
null == undefined    //true
NaN == NaN           //false
```

Boolean Logics

AND, OR, and NOT

Operator	Name	Example	Result
&&	AND	x < 10 && x !== 5	false
||	OR	y > 9 || x === 5	true
!	NOT	!(x === y)	true

Assuming x = 5 and y = 9

Boolean Logics

AND, OR, and NOT

Operator	Name	Example	Result
&&	AND	x < 10 && x !== 5	false
||	OR	y > 9 || x === 5	true
!	NOT	!(x === y)	true

Assuming x = 5 and y = 9

Falsy Values:

- false
- 0
- ""
- null
- undefined
- NaN

Everything Else Is Truthy

Boolean Logics

Exercise 1

```
var x = 10;
var y = "a"

y === "b" || x >= 10
```

Exercise 2

```
var x = 3;
var y = 8;

!(x == "3" || x === y) && !(y != 8 && x <= y)
```

Exercise 3

```
var str = ""
var msg = "haha!"
var isFunny = "false"

!(( str || msg ) && isFunny)
```

Boolean Logics

Exercise 1

```
var x = 10;
var y = "a"

y === "b" || x >= 10
```

true

Exercise 2

```
var x = 3;
var y = 8;

!(x == "3" || x === y) && !(y != 8 && x <= y)
```

false

Exercise 3

```
var str = ""
var msg = "haha!"
var isFunny = "false"

!(( str || msg ) && isFunny)
```

false

Loops

Printing numbers from 1-5 with a for loop

```
for(var count = 0; count < 6; count++) {
  console.log(count);
}
```

Printing numbers from 1-5 with a while loop

```
var count = 1;

while(count < 6) {
 console.log("count is: " + count);
 count++;
}
```

Loops

Write code to create an ASCII art triangle like the one pictured. Use for loops.

Loops

Write code to create an ASCII art triangle like the one pictured. Use for loops.


```
var str = “”;
for(var r=1; r<=5; r++){
  for(var c=1; c<=r; c++){
      str +="$";
  }
  str += "\n"
}
console.log(str);
```

Functions

Functions let us wrap bits of code up into REUSABLE packages. They are one of the building blocks of JS.

Declare a function first:

```
function doSomething() {
  console.log("HELLO WORLD");
}
```

Then call it:

```
doSomething();
```

Functions

Often we want to write functions that take inputs.

```
function square(num) {
  console.log(num * num);
}
```

Now when we call *square* we need to pass in a value

```
square(10);  //prints 100
```

Functions

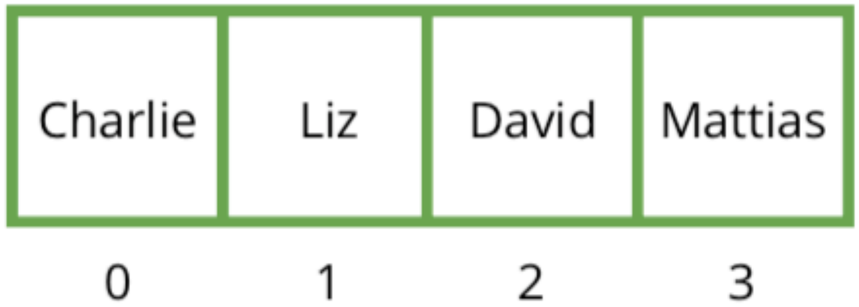
We use the *return* keyword to output a value from a function

```
//this function capitalizes the first char in a string:

function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}

var city = "paris";               //"paris"
var capital = capitalize(city);   //"Paris"
```

Arrays

Arrays let us group data together in lists

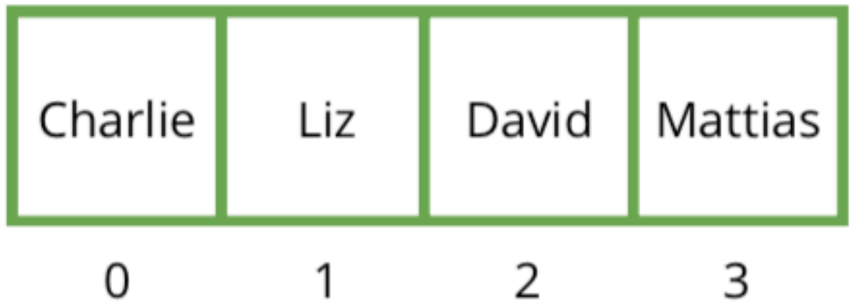
```
var friends = ["Charlie", "Liz", "David", "Mattias"];
```



Array are indexed starting at 0. Every slot has a corresponding number

Arrays

We can use those indices to retrieve data

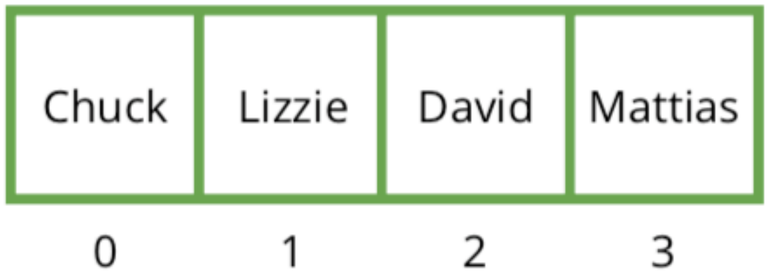



```
var friends = ["Charlie", "Liz", "David", "Mattias"];

console.log(friends[0])   //"Charlie"

friends[1] + " <3 " + friends[2]  //"Liz <3 David"
```

Arrays

We can also update arrays


```
var friends = ["Charlie", "Liz", "David", "Mattias"];

friends[0] = "Chuck";
friends[1] = "Lizzie";
```



Arrays - Methods

Arrays come with a few
built-in methods:

- push/pop
- shift/unshift
- indexOf
- slice
- splice

Arrays - Methods

Arrays come with a few built-in methods:

- ***push/pop***
- shift/unshift
- indexOf
- slice
- splice

Use push to add to the end of an array:

```
var colors = ["red", "orange", "yellow"];
colors.push("green");
//["red", "orange", "yellow", "green"]
```

Use pop to remove the last item in an array

```
var colors = ["red", "orange", "yellow"];
colors.pop();
//["red", "orange"]

//pop() returns the removed element
var col = colors.pop();  //orange
```

Arrays - Methods

Arrays come with a few built-in methods:

- push/pop
- ***shift/unshift***
- indexOf
- slice
- splice

Use unshift to add to the front of an array:

```
var colors = ["red", "orange", "yellow"];
colors.unshift("infrared")
//["infrared", "red", "orange", "yellow"]
```

Use shift to remove the first item in an array

```
var colors = ["red", "orange", "yellow"];
colors.shift();
//["orange", "yellow"]

//shift() also returns the removed element
var col = colors.shift();  //orange
```

Arrays - Methods

Arrays come with a few built-in methods:

- push/pop
- shift/unshift
- ***indexOf***
- slice
- splice

Use indexOf() to find the index of an item in an array

```
var friends = ["Charlie", "Liz", "David", "Mattias", "Liz"];

//returns the first index at which a given element can be found
friends.indexOf("David"); //2
friends.indexOf("Liz"); //1, not 4

//returns -1 if the element is not present.
friends.indexOf("Hagrid"); //-1
```

Arrays - Methods

Arrays come with a few built-in methods:

- push/pop
- shift/unshift
- indexOf
- ***slice***
- splice

Use slice() to copy parts of an array

```
var fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];
//use slice to copy the 2nd and 3d fruits
//specify index where the new array starts(1) and ends(3)
var citrus = fruits.slice(1, 3);

//this does not alter the original fruits array
//citrus contains ['Orange','Lemon']
//fruits contains ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];
```

Arrays - Methods

Arrays come with a few built-in methods:

- push/pop
- shift/unshift
- indexOf
- slice
- ***splice***

Use splice to remove

```
var fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];
//use splice to remove 'Orange' from the array
//specify index of the element to be removed and
//how many elements should be removed from that index
fruits.splice(1, 1);
// returns: ["Orange"]
console.log(fruits);
// prints: ["Banana", "Lemon", "Apple", "Mango"]
```

Objects

Store data in key-value pairs

```
var person = {
  name: "Travis",
  age: 21,
  city: "LA"
};


//bracket notation, similar to arrays:
console.log(person["name"]);
//dot notation:
console.log(person.name);
```

Objects

Just like an array: access a property and reassign it

```
var person = {
  name: "Travis",
  age: 21,
  city: "LA"
};

//to update age
person["age"] += 1;
//to update city
person.city = "London";
```

Objects

Creating Objects

Like arrays, there are a few methods of initializing objects

```
//make an empty object and then add to it
var person = {}
person.name = "Travis";
person.age = 21;
person.city = "LA";

//all at once
var person = {
  name: "Travis",
  age: 21,
  city: "LA"
};

//another way of initializing an Object
var person = new Object();
person.name = "Travis";
person.age = 21;
person.city = "LA";
```

Objects

Objects can hold all sorts of data

```
var junkObject = {
  age: 57,
  color: "purple",
  isHungry: true,
  friends: ["Horatio", "Hamlet"],
  pet: {
   name: "Rusty",
   species: "Dog",
   age: 2
  }
};
```

Objects - Methods

Object.keys() creates an array containing the **keys** of an object

```
// Initialize an object
const employees = {
    boss: 'Michael',
    secretary: 'Pam',
    sales: 'Jim',
    accountant: 'Oscar'
};

// Get the keys of the object
const keys = Object.keys(employees);

console.log(keys);
```

```
Output
["boss", "secretary", "sales", "accountant"]
```

Objects - Methods

Object.values() creates an array containing the **values** of an object

```
// Initialize an object
const session = {
    id: 1,
    time: `26-July-2018`,
    device: 'mobile',
    browser: 'Chrome'
};

// Get all values of the object
const values = Object.values(session);

console.log(values);
```

```
Output
[1, "26-July-2018", "mobile", "Chrome"]
```

Objects - Methods

Object.entries() creates a nested array of the **key/value** pairs of an object

```
// Initialize an object
const operatingSystem = {
    name: 'Ubuntu',
    version: 18.04,
    license: 'Open Source'
};

// Get the object key/value pairs
const entries = Object.entries(operatingSystem);

console.log(entries);
```

```
Output
[
    ["name", "Ubuntu"]
    ["version", 18.04]
    ["license", "Open Source"]
]
```

Objects - Methods

Write a function prettyPrint*()* that accepts an object as an argument and prints out a "pretty" string version of the object.

```
prettyPrint({name: "Rusty", species: "dog", breed: "mutt"});

//the above code should print the following 3 lines:
//name: Rusty
//species: dog
//breed: mutt
```

Objects - Methods

Write a function prettyPrint() that accepts an object as an argument and prints out a "pretty" string version of the object.

```
prettyPrint({name: "Rusty", species: "dog", breed: "mutt"});

//the above code should print the following 3 lines:
//name: Rusty
//species: dog
//breed: mutt
```

```
function prettyPrint(obj){
  var keys = Object.keys(obj);
  for(var i=0; i<keys.length; i++){
     var key = keys[i];
     var val = obj[key];
     console.log(key + ":" + val);
  }
}
```

Objects - Methods

Write a function prettyPrint() that accepts an object as an argument and prints out a "pretty" string version of the object.

```
prettyPrint({name: "Rusty", species: "dog", breed: "mutt"});

//the above code should print the following 3 lines:
//name: Rusty
//species: dog
//breed: mutt
```

```
function prettyPrint(obj){
  var keys = Object.keys(obj);
  for(var i=0; i<keys.length; i++){
     var key = keys[i];
     var val = obj[key];
     console.log(key + ":" + val);
  }
}
```

```
function prettyPrint(obj){
   Object.keys(obj).forEach(function(k){
        console.log(k + ":" + obj[k]);
   });
}
```