Comp 125 - Visual Information Processing

Spring Semester 2018 - week 2 - wednesday

Dr Nick Hayward

JS Basics - type conversion

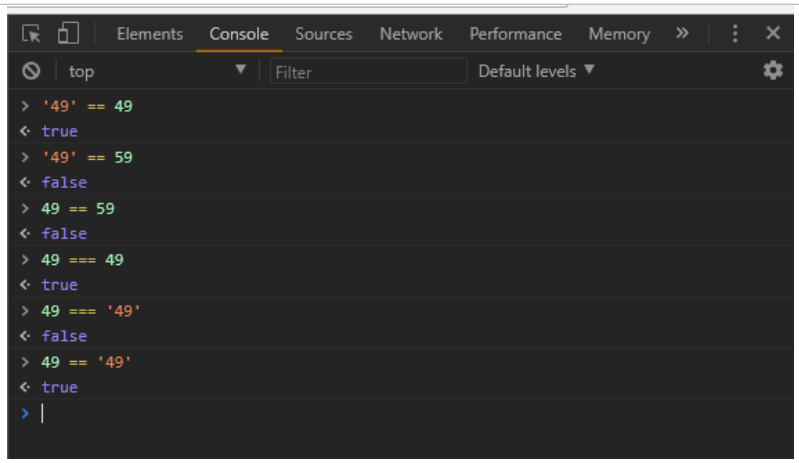
- option and ability to convert types in JS
  - *in effect,* ***coerce*** *our values and types from one type to another*
- convert a number, or coerce it, to a string
- built-in JS function, `Number()`, is an explicit coercion
  - *explicit coercion, convert any type to a number type*
- implicit coercion, JS will often perform as part of a comparison

```
"49" == 49
```

- JS implicitly coerces left string to a matching number
  - *then performs the comparison*
- often considered bad practice
  - *convert first, and then compare*
- implicit coercion still follows rules
  - *can be very useful*

JS Basics - examples of coercion - part I

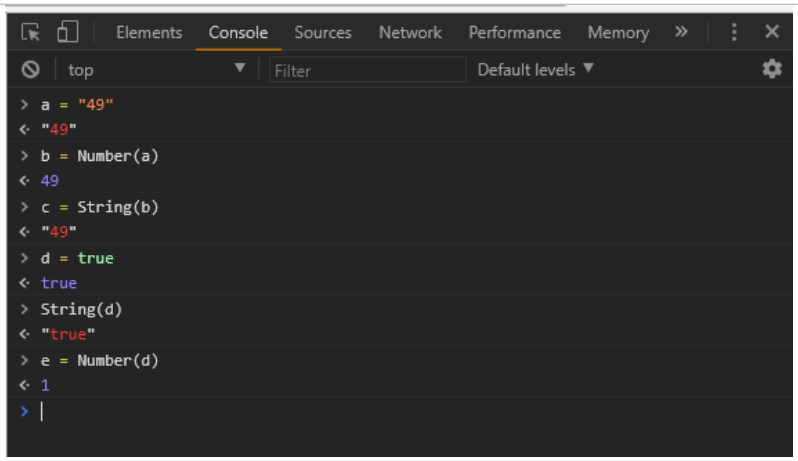
Coerce strings and numbers...



JavaScript - examples of type coercion

JS Basics - examples of coercion - part 2

Coerce strings, numbers, booleans...



JavaScript - examples of type coercion

JS Basics - variables - part I

- **symbolic** container for values and data
- applications use containers to keep track and update values
- use a **variable** as a container for such values and data
  - *allow values to vary over time*
- JS can emphasize types for values, does not enforce on the variable
  - ***weak typing*** *or* ***dynamic typing***
  - *JS permits a variable to hold a value of any type*
- often a benefit of the language
- a quick way to maintain flexibility in design and development

JS Basics - variables - part 2

- declare a variable using the keyword `var`
- declaration does not include **type** information

```
var a = 49;
//double var a value
var a = a * 2;
//coerce var a to string
var a = String(a);
//output string value to console
console.log(a);
```

- `var a` maintains a running total of the value of `a`
- keeps record of changes, effectively **state** of the value
- **state** is keeping track of changes to any values in the application

JS Basics - variables - part 3

- use variables in JS to enable central, common references to our values and data
- better known in most languages simply as **constants**
- JS is similar
  - *creates a read-only reference to a value*
  - *value itself is not immutable, e.g. an object...*
  - *it's simply the identifier that cannot be reassigned*
  - *JS constants are also bound by scoping rules*
- allow us to define and declare a variable with a value
  - *not intended to change throughout the application*
- **constants** are often declared together
  - *uppercase is standard practice - although not a rule...*
- form a store for values abstracted for use throughout an app
- JS normally defines constants using uppercase letters,

```
var NAME = "Philae";
```

- ECMAScript 6, ES6, introduces additional variable keywords
  - *e.g.* `const`

```
const TEMPLE_NAME = "Philae";
```

- benefits of abstraction, ensuring value is not accidentally changed
  - *change rejected for a running app*
  - *in* `strict` *mode, app will fail with an error for any change*

JS Basics - comments

- JS permits comments in the code
- two different implementations

single line

```
//single line comment
var a = 49;
```

multi-line

```
/* this comment has more to say...
we'll need a second line */
var b = "forty nine";
```

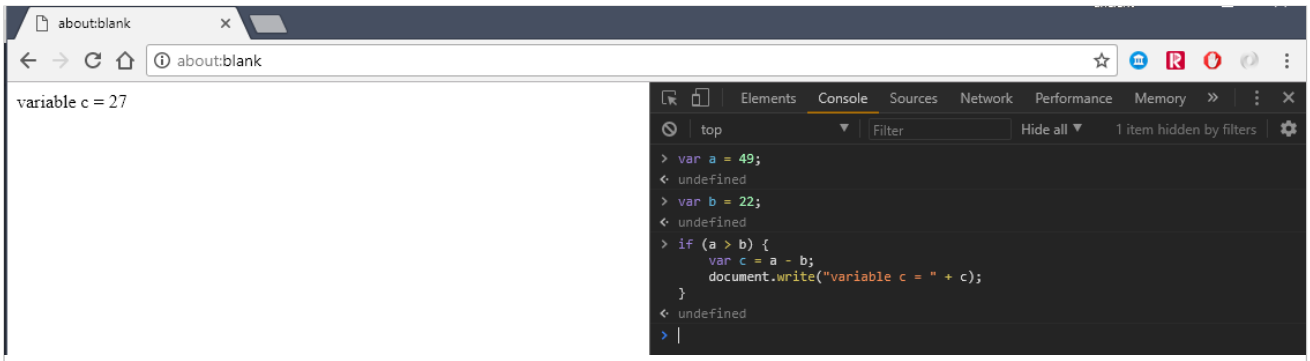
JS Basics - logic - blocks & conditionals - part I

- simple act of grouping contiguous and related code statements together
  - *known as **blocks***
- block defined by wrapping statements together
  - *within a pair of curly braces, { }*
- **blocks** commonly attached to other forms of control statement

```
if (a > b) {
...do something useful...
}
```

- conditional statements require a decision to be made
- JS includes many different ways we can express **conditionals**
- most common example is the `if` statement
  - *if this given condition is true, do the following...*
  - *`if` statement requires an expression between the parentheses*
  - *evaluates as either true or false*

JS Basics - logic - conditional statement



JavaScript - conditional statement with variables