Values and Refs

Primitives

- A primitive variable is one that stores a value
- Actually a value is an immutable object and the variable stores a reference to it
- Five primitive types in ES5: `number`, `string`, `boolean`, `undefined`, and `null`
- When you change a primitive variable's value you're changing the variable, not the object
  e.g.: `var x = 1; x += 1;`
- Primitive variables are said to be passed by copy

References

- A reference variable is one that stores a reference to/the memory address of an object

- Two reference types in ES5: `object` and `function` (but functions are objects too)

- When you change a reference variable's value you're changing the object, not the variable
  e.g.: `var obj = {x: 1}; obj.x += 1;`

- Reference variables are said to be passed by reference; actually, both primitive and reference variable are passed by copy (of reference)

Mutability

- Values are immutable objects

```
var x = 'Hello world';
x.replace('world', 'JavaScript');
// What is the value of x?
```

- Object literals, arrays, and functions are mutable

- `Object.defineProperty` may be used to configure props that are not modifiable/deletable, but creating fully immutable objects is not a trivial matter

Wrappers

- A wrapper is an object that wraps a value and has methods to enable the creation of new values from the existing one (transformation)
- Each of the primitive types number, string, and boolean has an associated wrapper - `Number`, `String`, and `Boolean`
- Wrapper functions may be used with/without (used to cast) the `new` keyword
- Coercion is the automatic conversion from value to wrapper object and back again, e.g. `'Hello'.toUpperCase();`

Equality Testing

- Two values are strictly equal (===) if they have the same value and are of the same type
- Equality testing of reference variables is a sameness test, i.e. do the variables reference the same object?
- Two values are loosely equal (==) if they have the same value after conversion to a common type, e.g. `1 == '1'` is effectively `1 == Number('1')`
- Everything in JS is truthy or falsey, i.e. everything can be converted to a boolean and will be when needed

Deleting and Dereferencing

- The `delete` operator may be used to delete configurable object props and array elements, e.g. `delete obj.x; delete arr[3];`

- Setting a variable to `null` makes the referenced object eligible for garbage collection provided it is not referenced by other variables

- The objects referenced by local variables are eligible for garbage collection when the variables go out of scope

undefined, null, NaN

- `undefined` is the value of a variable that has been declared but not initialised
  e.g. `var x; var y = noReturn();`

- `null` is the value of a variable that references nothing (actually null is an object!)

- `NaN` is a value meaning not a number and is a value that results from an unsuccessful attempt to convert to a number

Summary

- All variables store references; primitives store references to immutable objects; all variables are passed by copy
- number, string, and boolean objects are immutable and have associated wrappers - Number, String, and Boolean
- Strict equality (===) tests for equality of type and value; loose equality (==) tests for equality of value after conversion to a common type
- Object props and array elements may be deleted; objects may be dereferenced manually or automatically
- undefined means declared but not initialised; null means references nothing; NaN means not a number (failed conversion to number)