Pass-by-value, Pass-by-reference

Overview

```
/*
  - Primitive vs. complex types

  - Variable assignment

  - Passing values into functions

  - Equality operators and PBV/PBR

  - Arrays and .slice
*/
```

Primitive/complex values

```
/* primitive values in JS:
      - string
      - number
      - boolean
      - undefined
      - null
      - ES6 symbols (not covered in this course)

   complex values in JS:
      - objects (including arrays)
      - functions
 */


```

Primitive/complex observed behavior

```
1   /* strings are primitive values */
2   let britishCity = 'York';
3   let americanCity = britishCity;
4
5   americanCity = 'New ' + americanCity;
6
7   console.log(britishCity);
8   console.log(americanCity);
9
10
11
12
13
14
```

```
York
New York
```

Primitive/complex observed behavior

```
[ New, York ]
[ New, York ]
```

```
1   /* arrays are complex values */
2   let britishCity = ['York'];
3   let americanCity = britishCity;
4
5   americanCity.unshift('New');
6
7   console.log(britishCity);
8   console.log(americanCity);
9
10
11
12
13
14
```

Primitive/complex observed behavior

```
1   /* why do primitive values and complex values behave differently? */
2
3   /* you'll be able to answer that question at the end of this lesson! */
4
5
6
7
8
9
10
11
12
13
14
```

```
30
31
```

Assignment: primitive values

```
1   /* when you assign a primitive value to a variable, the variable 'stores'
2      a copy of that value */
3
4   let first = 20; // first stores the value 20
5   first = 30; // first now stores a new value, 30
6
7   let second = 30; // second stores its own copy of 30
8   second++; // second now stores a new value, 31
9
10  console.log(first);
11  console.log(second);
12
13
14
```

```
20
21
```

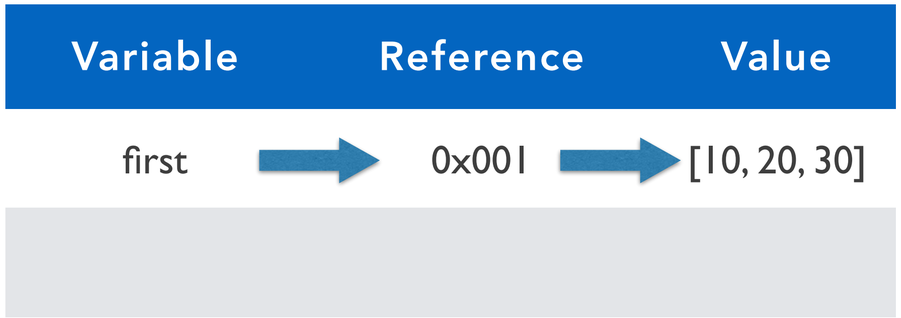
Assignment: primitive values

```
1   let first = 20; // first stores the value 20
2   second = first; // second stores its own copy of the value 20
3
4   second++; // has no effect on the value stored in first!
5
6   console.log(first);
7   console.log(second);
8
9
10
11
12
13
14
```

[10, 20, 30]

Assignment: complex values

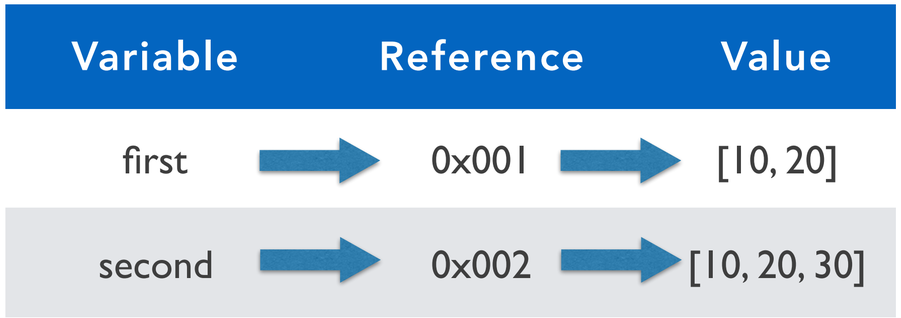
```
1  /* when you assign a complex value to a variable, the variable DOES NOT
2     store a copy of the value */
3
4  /* instead, the variable stores a reference in memory; the reference
5     points to the value */
6
7  let first = [10, 20]; // first stores a reference to [10, 20]
8  first.push(30); // first still stores the same reference
9
10 console.log(first);
11
12
13
14
```



```
[10, 20]
[10, 20, 30]
```

Assignment: complex values

```
1  let first = [10, 20];
2
3  let second = [10, 20]; // second stores a new reference to a new array
4
5  /* since first and second reference different arrays, pushing a value into
6     second will not affect the array referenced by first */
7  second.push(30);
8
9  console.log(first);
10 console.log(second);
11
12
13
14
```



```
[10, 20, 30]
[10, 20, 30]
```

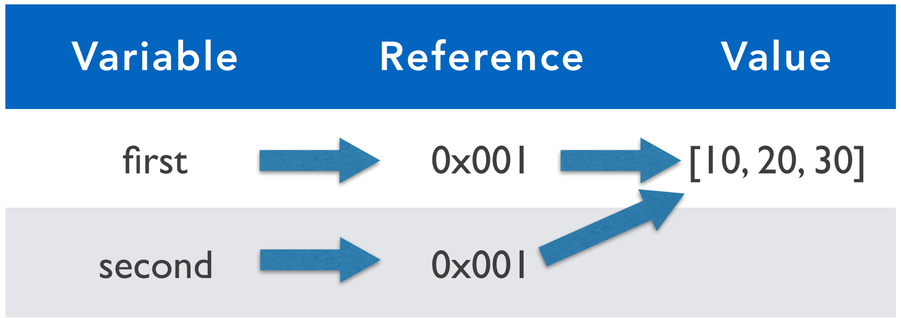
Assignment: complex values

```
let first = [10, 20];

let second = first; /* second now stores a copy of the REFERENCE that was
originally stored in first */

/* first and second share the same reference to the same array! */
second.push(30);

console.log(first);
console.log(second);
```



```
[10, 20]
[10, 20, 30]
```

Assignment: complex values

```
1   let first = [10, 20];
2
3   let second = first.slice(); // slice creates a new array!
4
5   second.push(30); // first and second reference different arrays
6
7   console.log(first);
8   console.log(second);
9
10
11
12
13
14
```

Variable	Reference	Value
first	0x001	[10, 20]
second	0x002	[10, 20, 30]

```
10
30
```

Passing primitive values

```
let myNum = 10;

function adds20(num) {
  num += 20;
  return num;
}

let returnedNum = adds20(myNum);

console.log(myNum);
console.log(returnedNum);
```

Passing complex values

```
[ 10, 20 ]
[ 10, 20 ]
```

```
let myArray = [10];

function pushes20(array) {
  array.push(20);
  return array;
}

let returnedArray = pushes20(myArray);

console.log(myArray);
console.log(returnedArray);
```

false

Equality operator and PBV/PBR

```
1   // === will compare complex values by reference, not by value!
2
3   let array1 = [1, 2, 3];
4   let array2 = [1, 2, 3];
5
6   console.log(array1 === array2);
7
8
9
10
11
12
13
14
```

Equality operator and PBV/PBR

```
1   // === will compare complex values by reference, not by value!
2
3   let array1 = [1, 2, 3];
4   let array2 = array1; // array1 and array2 share the same reference
5
6   console.log(array1 === array2);
7
8
9
10
11
12
13
14
```

```
true
```

Arrays and .slice

```
[1, [2, 3]]
false
```

```
1   /* we showed earlier that .slice creates a copy of an array */
2
3   let array1 = [1, [2, 3]];
4   let array2 = array1.slice();
5
6   console.log(array2);
7   console.log(array1 === array2);
8
9
10
11
12
13
14
```

```
[2, 3, 4]
true
```

Arrays and .slice

```
1  /* if an array has a complex value as an element, only the reference
2     to that complex value is copied into the new array */
3
4  /* that's why we say slice makes a "shallow" copy of an array; it doesn't
5     make new copies of any complex values stored inside the array */
6
7  let array1 = [1, [2, 3]];
8  let array2 = array1.slice();
9
10 array1[1].push(4);
11 console.log(array2[1]);
12
13 console.log(array1[1] === array2[1]);
14
```

Recap

```
/*
   - Primitive vs. complex types

   - Variable assignment

   - Passing values into functions

   - Equality operators and PBV/PBR

   - Arrays and .slice
*/
```