Pointers and Memory

Overview

```
1    /*
2      - Primitive vs. complex types
3
4      - Variable assignment
5
6      - Passing values into functions
7
8      - Equality operators and PBV/PBR
9
10     - Arrays and .slice
11   */
12
13
14
```

Primitive/complex values

```
1    /* primitive values in JS:
2            - string
3            - number
4            - boolean
5            - undefined
6            - null
7      - ES6 symbols (not covered in this course)
8
9    complex values in JS:
10           - objects (including arrays)
11           - functions
12    */
13
14
```

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
```

FULLSTACK

Primitive/complex observed behavior

```
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
```
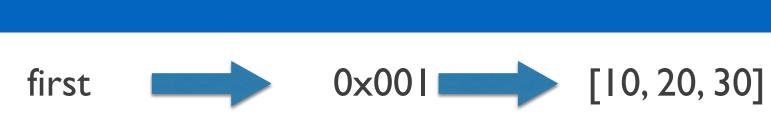
Variable	Value
first	30
second	31

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
```

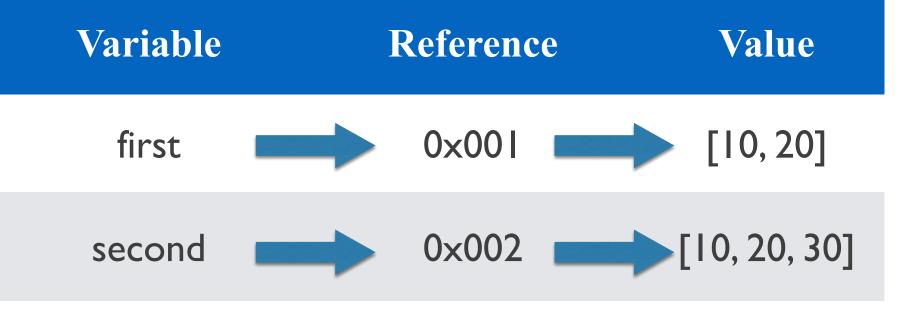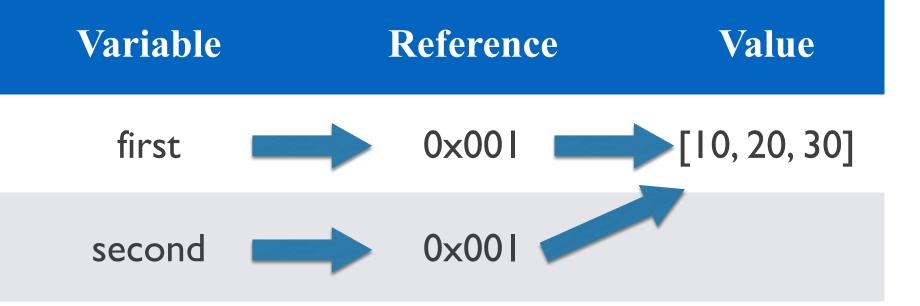| Variable	Value
first	20
second	21

20
21

Assignment: primitive values

```
1   let first = 20; // first stores the value 20
2   let 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]

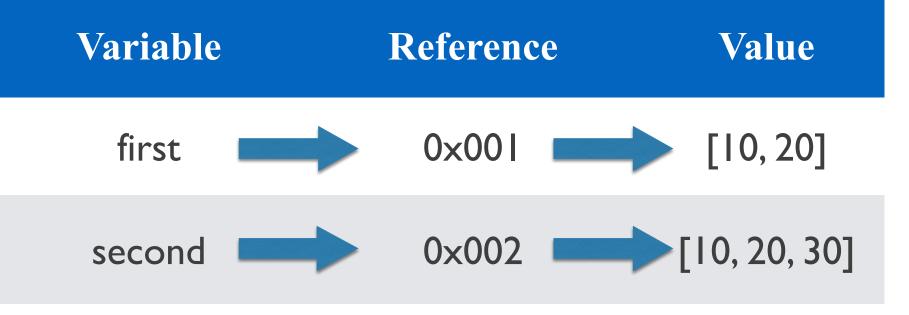# Assignment: complex values

```
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
```

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

[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
```

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

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

Assignment: complex values

```
1    let first = [10, 20];
2
3    let second = first; /* second now stores a copy of the REFERENCE that was
4    originally stored in first */
5
6    /* first and second share the same reference to the same array! */
7    second.push(30);
8
9    console.log(first);
10   console.log(second);
11
12
13
14
```

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

[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
```

Passing primitive values

```
1   let myNum = 10;
2
3   function adds20(num) {
4     num += 20;
5     return num;
6   }
7
8   let returnedNum = adds20(myNum);
9
10  console.log(myNum);
11  console.log(returnedNum);
12
13
14
```

FULLSTACK

BOOTCAMP PREP

Passing complex values

```
1   let myArray = [10];
2
3   function pushes20(array) {
4     array.push(20);
5     return array;
6   }
7
8   let returnedArray = pushes20(myArray);
9
10  console.log(myArray);
11  console.log(returnedArray);
12
13
14
```

FULLSTACK   BOOTCAMP PREP

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
```

FULLSTACK

BOOTCAMP PREP

Equality operator and PBV/PBR

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

FULLSTACK

BOOTCAMP PREP

Arrays and .slice

```
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
```

FULLSTACK

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

```
1    /*
2      - Primitive vs. complex types
3
4      - Variable assignment
5
6      - Passing values into functions
7
8      - Equality operators and PBV/PBR
9
10     - Arrays and .slice
11   */
12
13
14
```