# Functions

```
// Javascript does not have passing by reference even though it looks like it's
passing by reference
// For objects you pass in a reference, a memory address of of the object, but
that reference itself is still a value
// pass IN a reference not BY a reference


const flight = 'LH234'; // Primitive type - On the stack
const kelsy = {         // Reference type - On the Heap
    name: 'Kelsy Watkins',
    passport: 8675309
}

// flightNum contains copy of flight value, not original value

// when passing primitive types to function the value is simply copied

// when passing reference type to function what is copied is the reference to the
object in the memory heap

// passenger takes copy of reference value to object

const checkIn = function (flightNum, passenger) {
    flightNum = 'LH999';
    passenger.name = 'Mrs. ' + passenger.name;

    if (passenger.passport === 8675309) {
        alert('Checked in')
    } else {
        alert('Wrong passport')
    }
}

checkIn(flight, kelsy);
console.log(flight); // still LH234 flightNum is on new memory address
console.log(kelsy); // name gets changed as memory address stay the same and
value does change on the heap
```

Flight = LH234

Passing in flight as parameter – points to memory address copies value

Sets new memory address to copied value

flightNum = LH999

---

kelsy = object

passing in passenger as parameter – points to memory address

STACK

| Line | Identifier | Address | Value |
|---|---|---|---|
| | flight | 0003 | LH234 |
| | flightNum | 0004 | LH234 |
| | | | ~~LH234~~ LH999 |
| | kelsy | 0005 | D30f |
| | passenger | | |

When we pass a reference type to a function what is copied the reference to the object in the memory HEAP

HEAP

| Line | Address | Value |
|---|---|---|
| Origonal object | D30F | `const kelsy = { name: 'Kelsy Watkins', passport: 8675309 }` |
| passenger.name = 'Mrs. ' + passenger.name; | | `const kelsy = { name: ' Mrs. Kelsy Watkins', passport: 8675309 }` |

Passing by Value - what JavaScript does

Passing by Reference – JavaScript does not do this. You have a reference to a memory value but that memory address itself is still a value

## First class functions

- JavaScript treats functions as **first-class citizens**
- This means that functions are **simply values**
- Functions are just another **type of object**

*Because functions are values you can:

```
const add = (a, b) => a + b; // store in variables, function expression
const counter = {
    value: 23,
    inc: function () {this.value++;} // store in object properties, object method
}
```

*Pass functions as arguments to OTHER functions

```
const greet = () => console.log('Hey Kelsy');
btnClose.addEventListener('click', greet) // pass greet function to
addEventListener function
```

*Return functions FROM functions

*Call methods on functions

```
counter.inc.bind(someOtherObject); // calling method on a function
```

## Higher Order functions

- A function that **receives** another function as an argument, that **returns** a new function, or **both**
- This is only possible because of first class functions

*function that receives another function

```
const greet = () => console.log('Hey Kelsy');
btnClose.addEventListener('click', greet)
```

Higher order function

Callback function

*function that return new function

```
function count() {        <-- Higher order function
    let counter = 0;
    return function () {  <-- returned function
        counter++;
    };
}
```