Callbacks Quiz Recall

```
let foo = function(n, cb) {
  console.log("vroom");
  for (let i = 0; i < n; i++) {
    cb();
  }
  console.log("skrrt");
};

foo(2, function() {
  console.log("swoosh");
});
```

In what order will the code above print out?

- vroom, swoosh, skrrt, swoosh, skrrt
- swoosh, vroom, skrrt
- vroom, swoosh, swoosh, swoosh, skrrt
- [✓ vroom, swoosh, swoosh, skrrt]

EXPLANATION

Since the loop iterates twice, 'swoosh' will print twice between 'vroom' and 'skrrt'.

```
let foo = function() {
  console.log("Everglades");
  console.log("Sequoia");
};

console.log("Zion");
```

```
  foo();
  console.log("Acadia");
```

In what order will the code above print out?

- [✓ Zion, Everglades, Sequoia, Acadia]
- Zion, Everglades, Acadia, Sequoia
- Everglades, Sequoia, Zion, Acadia
- Everglades, Zion, Acadia, Sequoia

EXPLANATION

The prints that belong to `foo` will be executed only when it is called after 'Zion', but before 'Acadia'.

Are functions considered first class objects in JavaScript?

- no
- [✓ yes]

EXPLANATION

Functions are first class objects in JavaScript, because they can be assigned, passed as an argument, and returned.

```
let foo = function() {
  console.log("hello");
  return 42;
};

foo;
```

When executed in node, what will the code snippet above print out?

- ○ `[Function: foo]`
- ○ `42`
- [✓ It will print nothing]
- ○ `hello`

EXPLANATION

Nothing will be printed because the only `console.log` is within the `foo` function, but `foo()` is never called.

Which of the following is not required to be a first class object?

- ○ ability to be assigned to a variable
- [✓ ability to be mutated]
- ○ ability to be a return value of a function
- ○ ability to be an argument to a function

EXPLANATION

A first class object does not need to mutable. For example, strings are immutable but still first class because they can be assigned, passed as an argument, and returned.

```
let foo = function() {
  console.log("hello");
  return 42;
};

console.log(foo);
```

When executed in node, what will the code snippet above print out?

- ○ `hello`
- ○ It will print nothing
- ○ `42`
- ✓ `[Function: foo]`

EXPLANATION

The `foo()` is not called, instead the `foo` function object itself is printed out.

```
let bar = function(s) {
  return s.toLowerCase() + "...";
};

let foo = function(message, cb1, cb2) {
  console.log(cb1(message));
  console.log(cb2(message));
};

foo("Hey Programmers", bar, function(s) {
  return s.toUpperCase() + "!";
});
```

When executed in node, what will the snippet above print out?

- ○ `[Function]`, `[Function]`
- ✓ `hey programmers...`, `HEY PROGRAMMERS!`

`HEY PROGRAMMERS!`, `hey programmers...`

EXPLANATION

Since arguments are passed positionally, `cb1` is `bar` and `cb2` is the anonymous function. Both `cb1` and `cb2` are called and their return values are printed out.

```
let bar = function() {
  console.log("Ramen");
};

let foo = function(cb) {
  console.log("Gazpacho");
  cb();
  console.log("Egusi");
};

console.log("Bisque");
foo(bar);
console.log("Pho");
```

In what order will the code above print out?

Bisque, Gazpacho, Egusi, Ramen, Pho

Bisque, Pho, Gazpacho, Egusi, Ramen

Ramen, Gazpacho, Egusi, Bisque, Pho

[✓ Bisque, Gazpacho, Ramen, Egusi, Pho]

EXPLANATION

The `bar` function is passed as a callback to `foo`, so the name `cb` refers to `bar` inside of `foo`

```
let bar = function() {
  console.log("Arches");
};

let foo = function() {
  console.log("Everglades");
  bar();
  console.log("Sequoia");
};

console.log("Zion");
foo();
console.log("Acadia");
```

In what order will the code above print out?

- Arches, Everglades, Sequoia, Zion, Acadia
- [✓ Zion, Everglades, Arches, Sequoia, Acadia]
- Zion, Everglades, Sequoia, Arches, Acadia
- Zion, Arches, Everglades, Sequoia, Acadia

EXPLANATION

The code inside of functions only execute once the function is called. When a function returns, execution jumps back to the line after where it was called.

```
let bar = function(mystery) {
  mystery("sneaky");
};

let foo = function(secret) {
  console.log(secret);
};
```

```
bar(foo);
```

In the snippet above, which function is acting as a "callback"?

- `console.log`
- `bar`
- [✓ `foo`]

EXPLANATION

A callback is a function that is passed as an argument to another function. In this example, `foo`is passed as an argument to `bar`, making `foo`the callback.

```
function foo() {
  console.log("fizz");
}

function bar() {
  console.log("buzz");
}

function boom(cb1, cb2) {
  console.log("zip");
  cb1();
  console.log("zap");
  cb2();
  console.log("zoop");
}

boom(bar, foo);
```

In what order will the code above print out?

- zip, zap, zoop, buzz, fizz
- [✓ zip, buzz, zap, fizz, zoop]
- zip, fizz, zap, buzz, zoop
- fizz, buzz, zip, zap, zoop

EXPLANATION

`bar` and `foo` are passed in as arguments for `cb1` and `cb2` respectively.