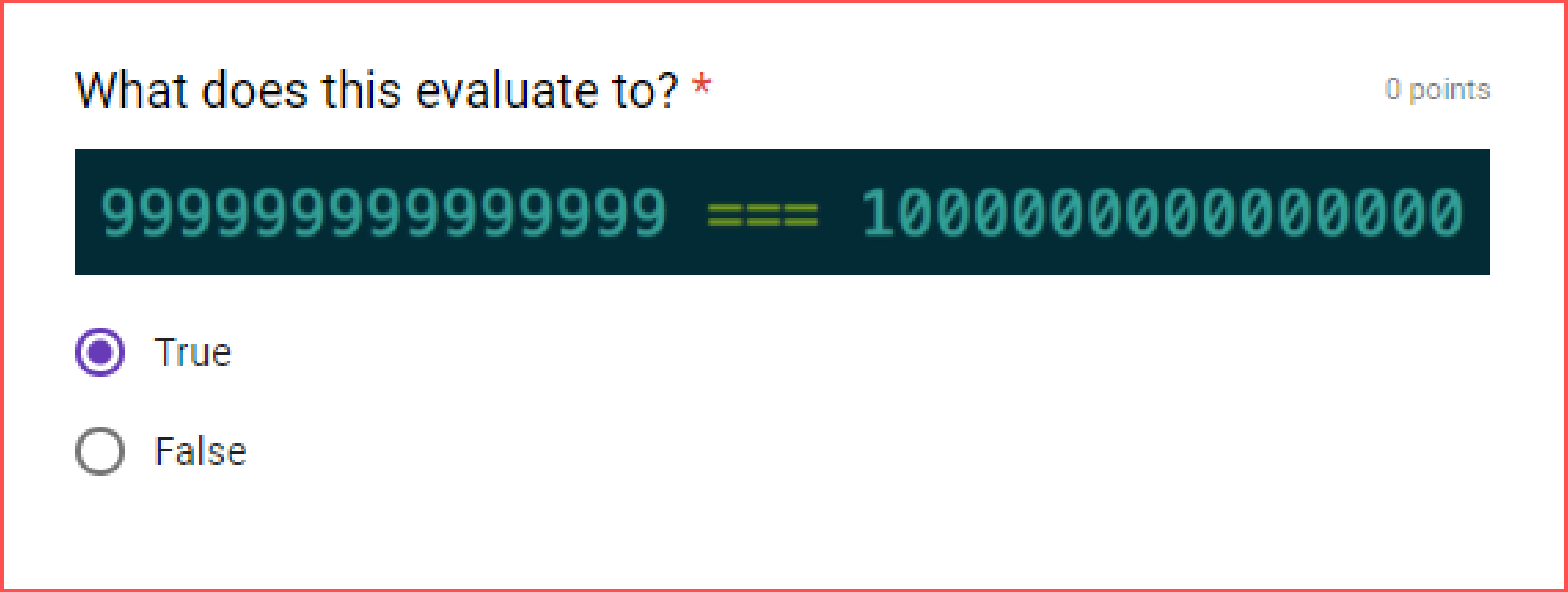
JAVASCRIPT QUIRKS

QUIZ #4

#1



What does this evaluate to? *

0 points

```
9999999999999999 === 10000000000000000
```

[✓ True]

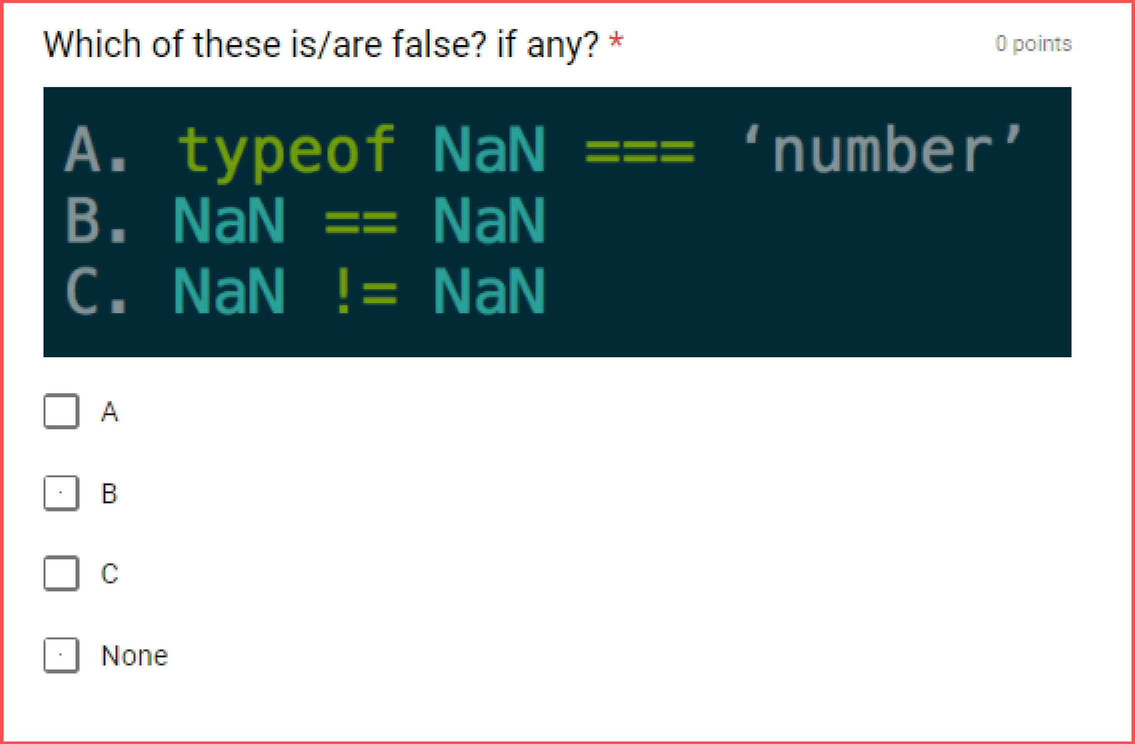
False

TRUE

- Javascript uses floating point arithmetic
- Floating point arithmetic is not always 100% accurate
- Take precautions when doing number comparisons in JS!

https://modernweb.com/what-every-javascript-developer-should-know-about-floating-points/

#2



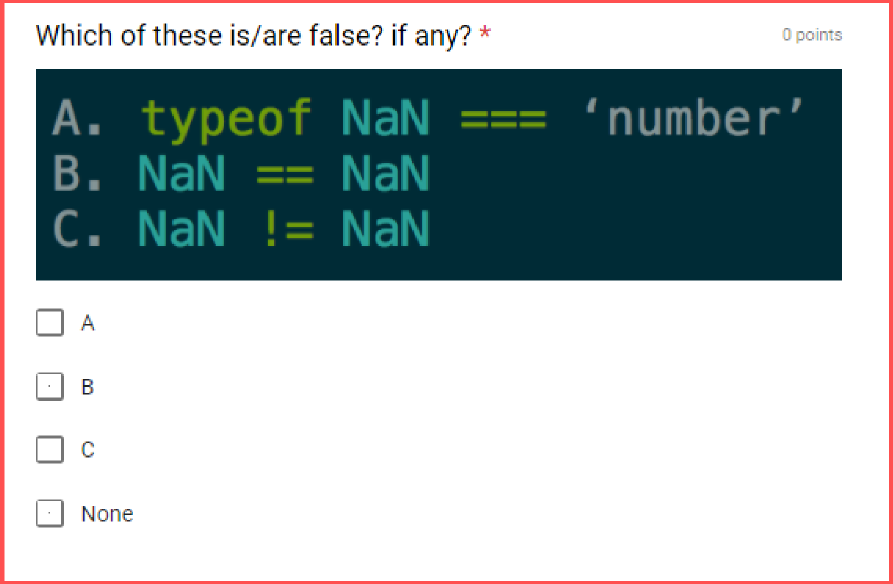
Which of these is/are false? if any? *

0 points

```
A. typeof NaN === 'number'
B. NaN == NaN
C. NaN != NaN
```

- ☐ A
- ☐ B
- ☐ C
- ☐ None

A)TRUE B)FALSE C)TRUE


Which of these is/are false? if any? * 0 points

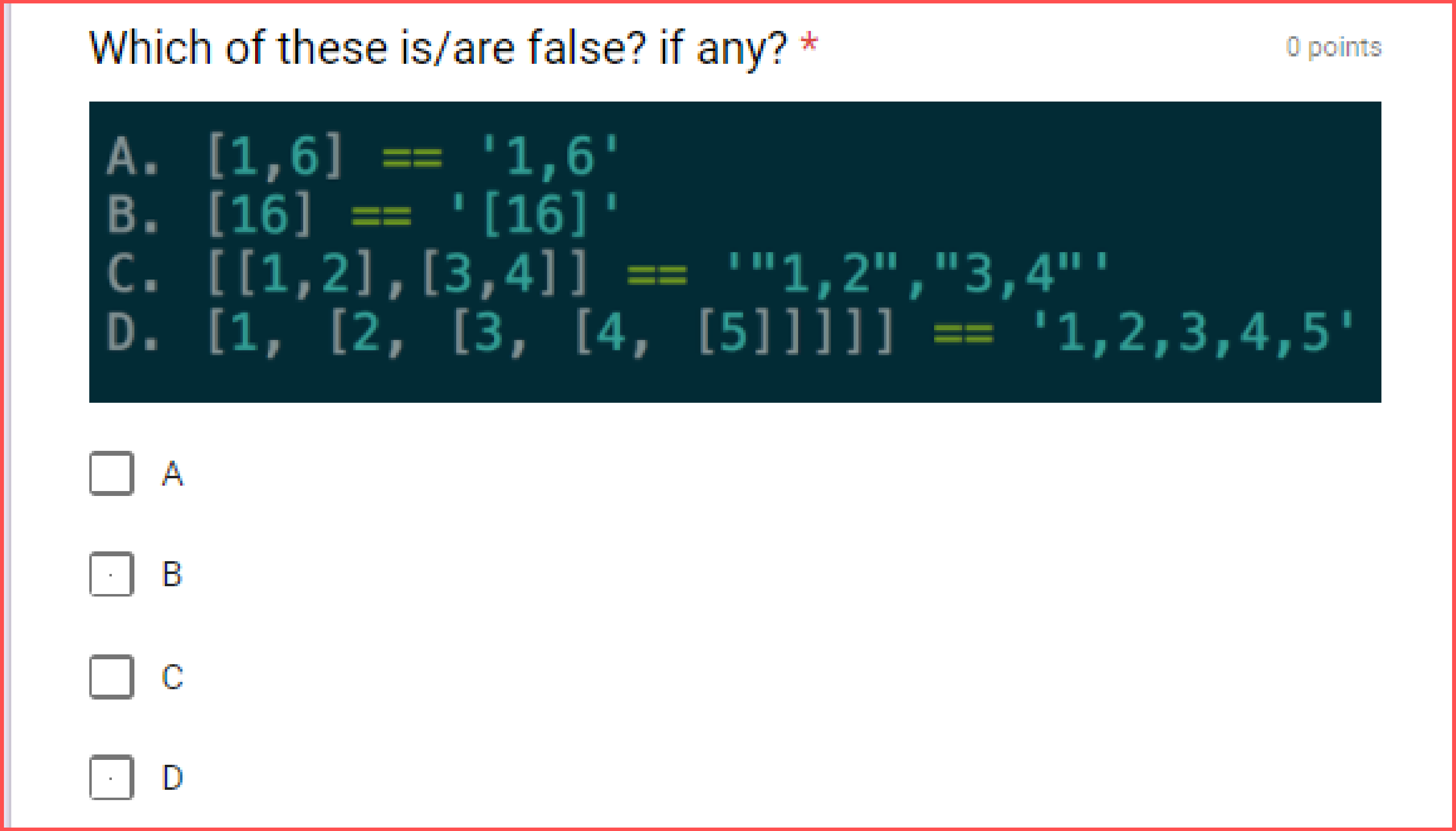
```
A. typeof NaN === ‘number’
B. NaN == NaN
C. NaN != NaN
```

- [ ] A
- [ ] B
- [ ] C
- [ ] None

- NaN is a type of number
- What it MEANS: an unrepresentable value
- Since NaN is representable (as NaN), NaN != NaN

  https://stackoverflow.com/questions/10034149/why-is-nan-not-equal-to-nan

#3



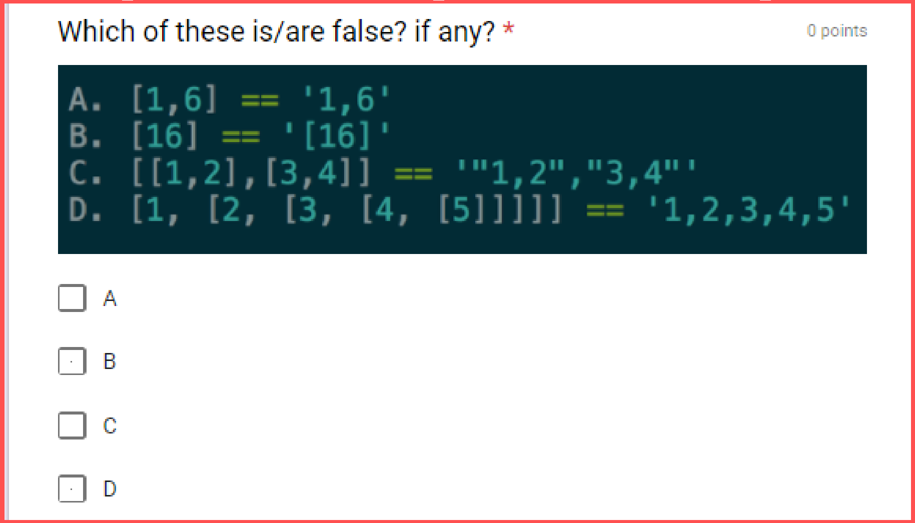
Which of these is/are false? if any? *

0 points

```
A. [1,6] == '1,6'
B. [16] == '[16]'
C. [[1,2],[3,4]] == '"1,2","3,4"'
D. [1, [2, [3, [4, [5]]]]] == '1,2,3,4,5'
```

- [ ] A
- [ ] B
- [ ] C
- [ ] D

A)TRUE | B)FALSE | C) FALSE | D) TRUE



Which of these is/are false? if any? *

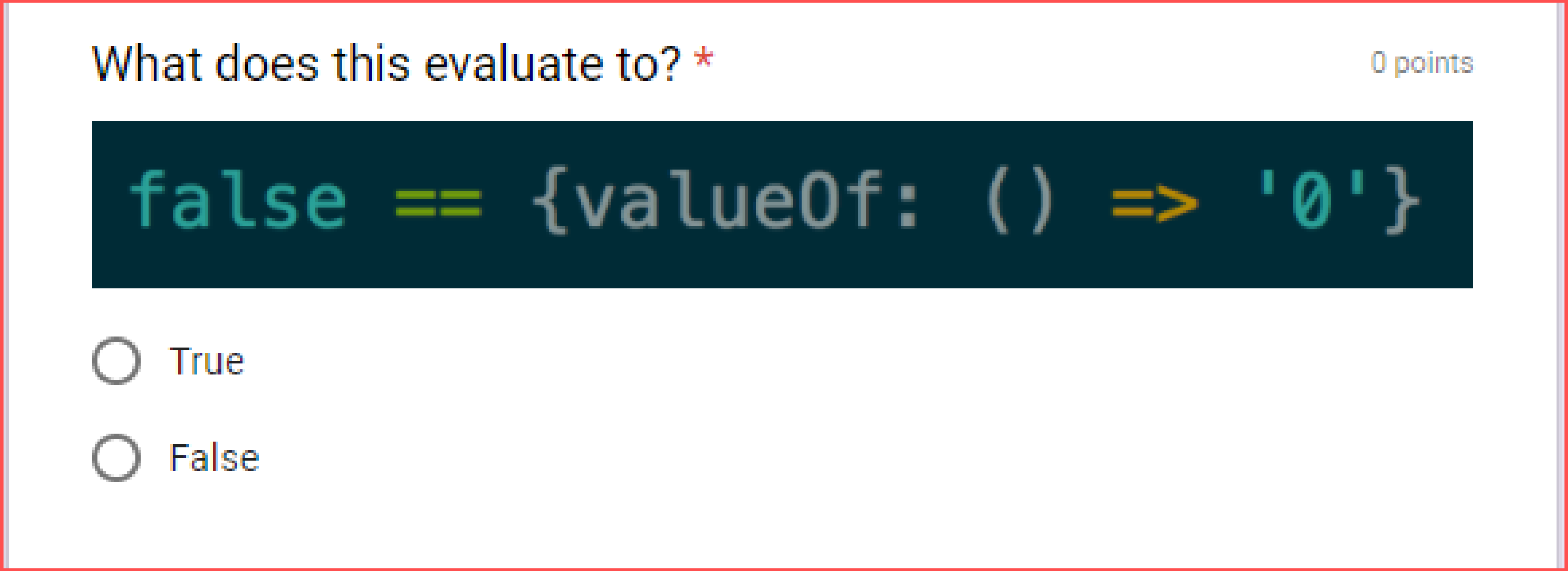
0 points

```
A. [1,6] == '1,6'
B. [16] == '[16]'
C. [[1,2],[3,4]] == '"1,2","3,4"'
D. [1, [2, [3, [4, [5]]]]] == '1,2,3,4,5'
```

- [ ] A
- [ ] B
- [ ] C
- [ ] D

- The == operator coerces values
- When Javascript coerces arrays into string, it stringifies all of the array values and separates them with commas
- Inner arrays are ignored!

#4



What does this evaluate to? *

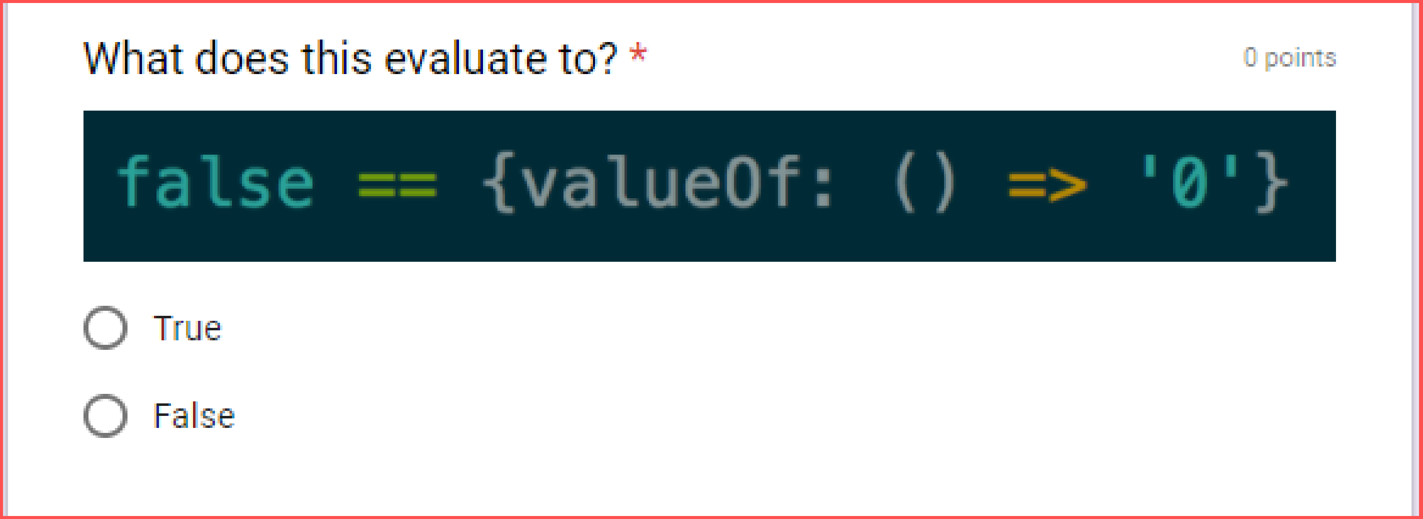
0 points

```
false == {valueOf: () => '0'}
```

True

False

TRUE



What does this evaluate to? *

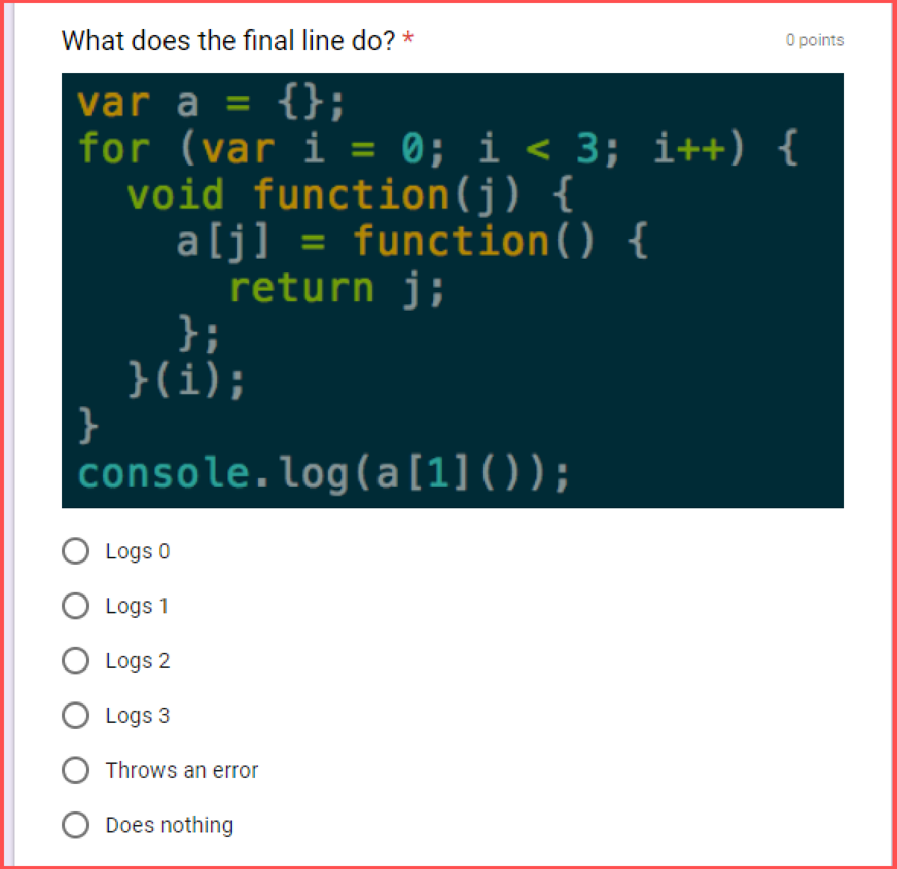
0 points

```
false == {valueOf: () => '0'}
```

- True
- False

- The **valueOf()** method returns the primitive value of the specified object.
- When comparing the object that contains a valueOf method, whatever gets returned from this method is used as the operand
- '0' == false!

#5



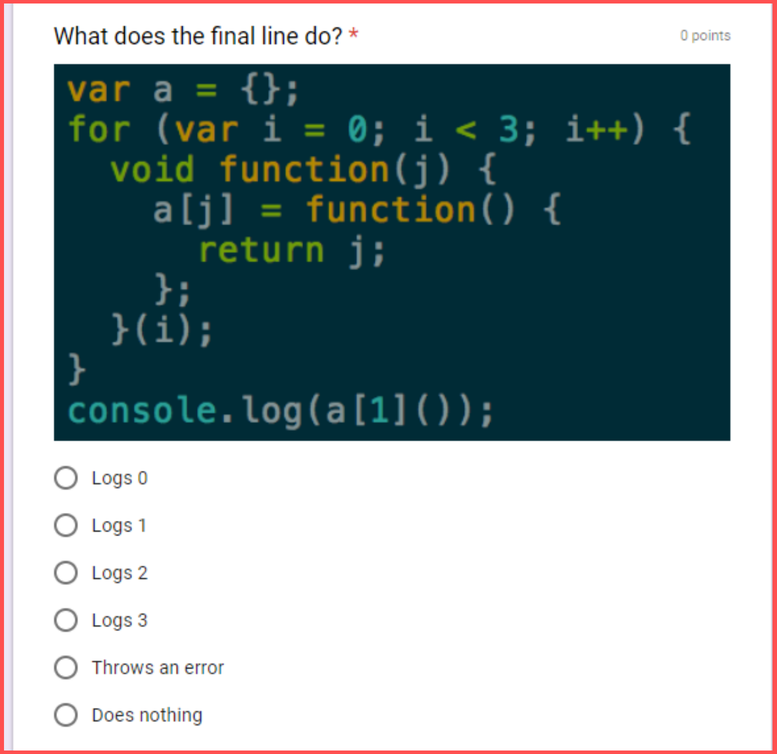
What does the final line do? *

0 points

```
var a = {};
for (var i = 0; i < 3; i++) {
  void function(j) {
    a[j] = function() {
      return j;
    };
  }(i);
}
console.log(a[1]());
```

- Logs 0
- Logs 1
- Logs 2
- Logs 3
- Throws an error
- Does nothing

LOGS: 1



What does the final line do? *

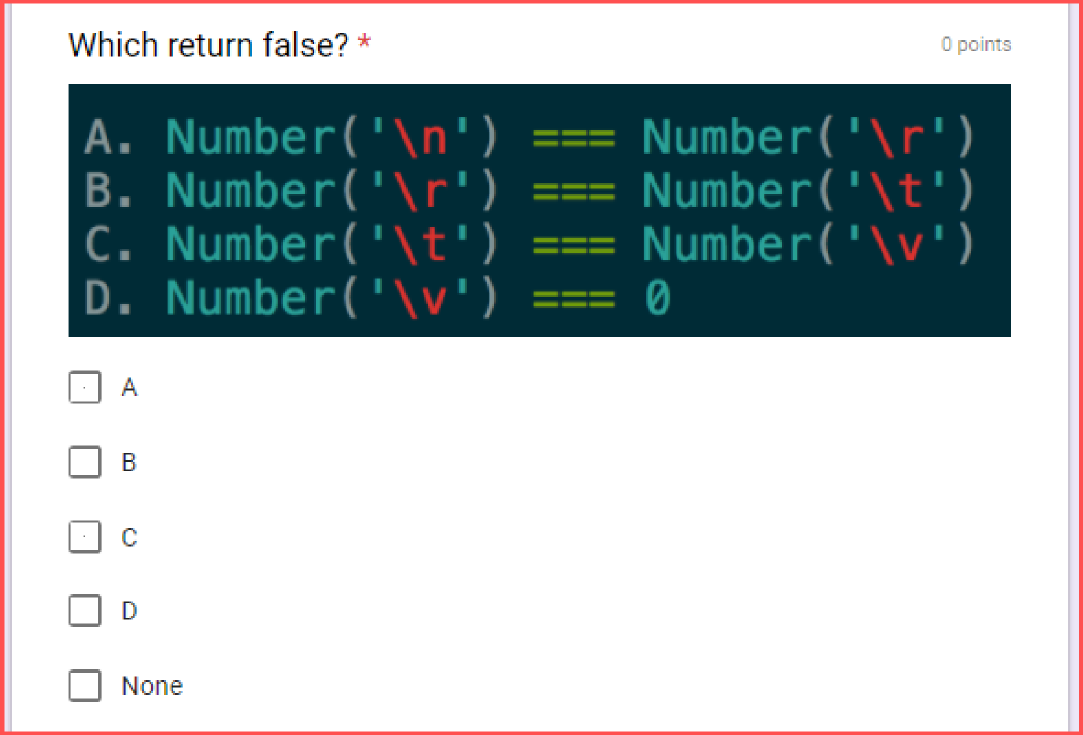
0 points

```
var a = {};
for (var i = 0; i < 3; i++) {
  void function(j) {
    a[j] = function() {
      return j;
    };
  }(i);
}
console.log(a[1]());
```

- Logs 0
- Logs 1
- Logs 2
- Logs 3
- Throws an error
- Does nothing

- Each iteration of the for loop adds a new entry to the object
- Void keyword evaluates the expression and return undefined, so it is not doing much here.

#6



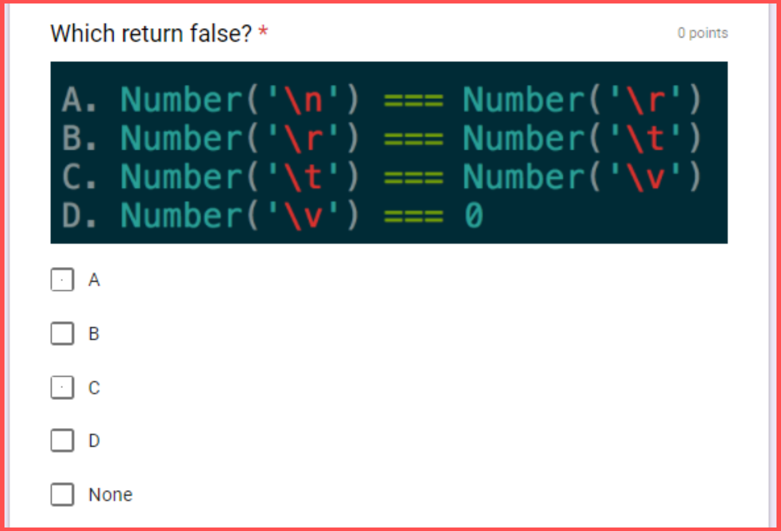
Which return false? *

0 points

```
A. Number('\n') === Number('\r')
B. Number('\r') === Number('\t')
C. Number('\t') === Number('\v')
D. Number('\v') === 0
```

- [ ] A
- [ ] B
- [ ] C
- [ ] D
- [ ] None

ALL TRUE



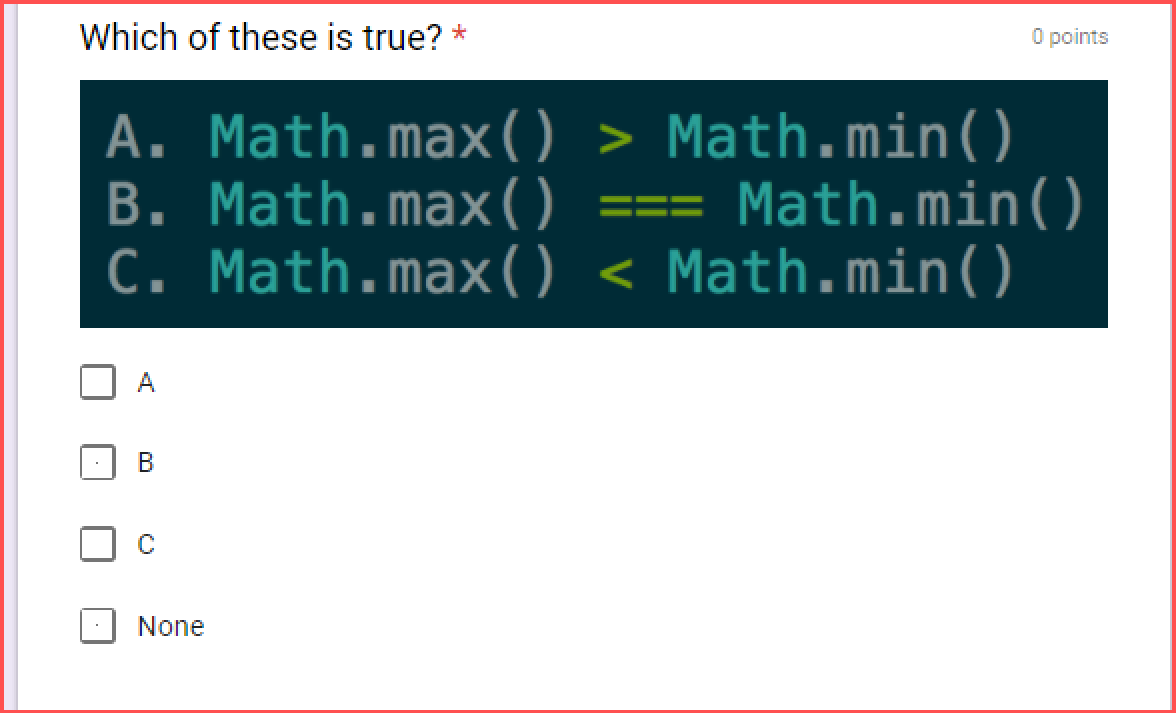
Which return false? * 0 points

```
A. Number('\n') === Number('\r')
B. Number('\r') === Number('\t')
C. Number('\t') === Number('\v')
D. Number('\v') === 0
```

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ None

Javascript's Number constructor coerces white space characters into 0!

#7

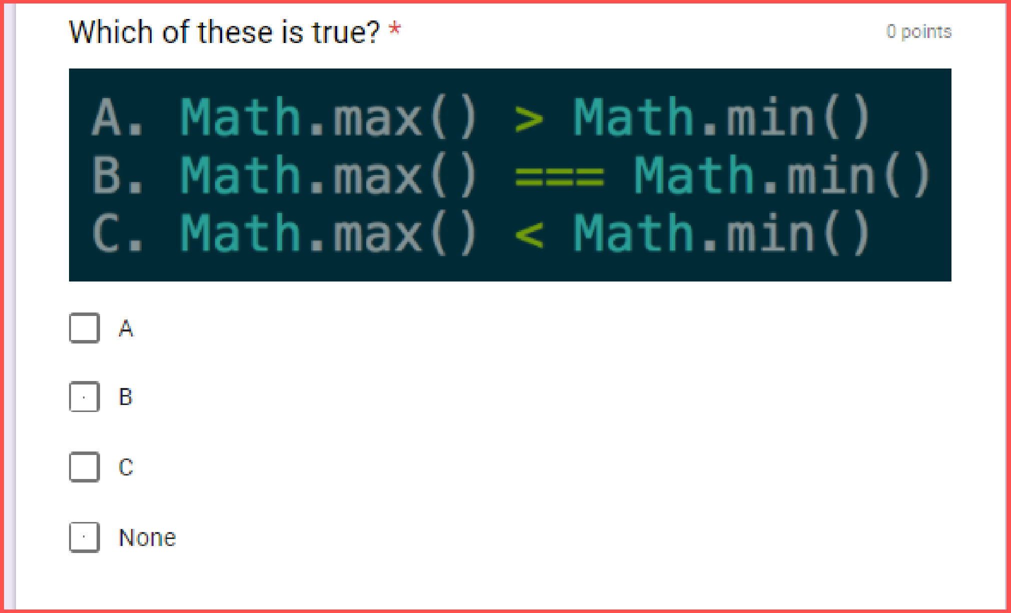


Which of these is true? * 0 points

```
A. Math.max() > Math.min()
B. Math.max() === Math.min()
C. Math.max() < Math.min()
```

- ☐ A
- ☐ B
- ☐ C
- ☐ None

C!



Which of these is true? *

0 points

```
A. Math.max() > Math.min()
B. Math.max() === Math.min()
C. Math.max() < Math.min()
```

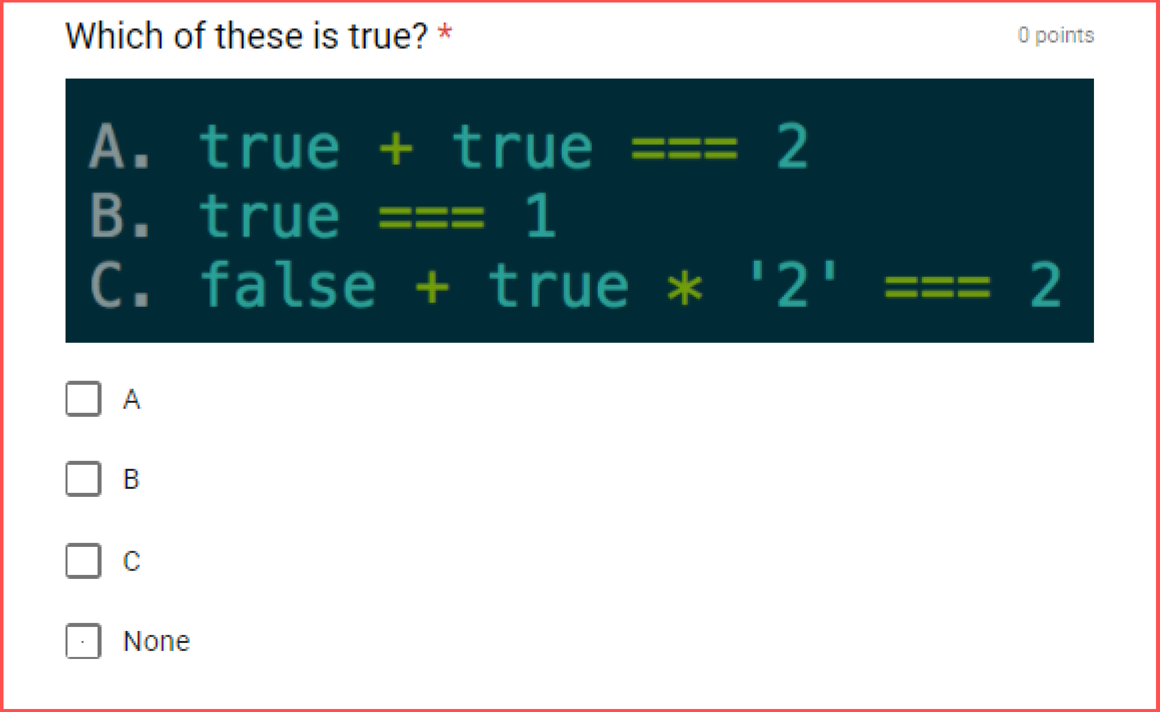
- [ ] A
- [ ] B
- [ ] C
- [ ] None

Math.max() === -Infinity, Math.min() === Infinity

What is the theoretical min/max of an empty set?

What should we compare a single input to?

#8



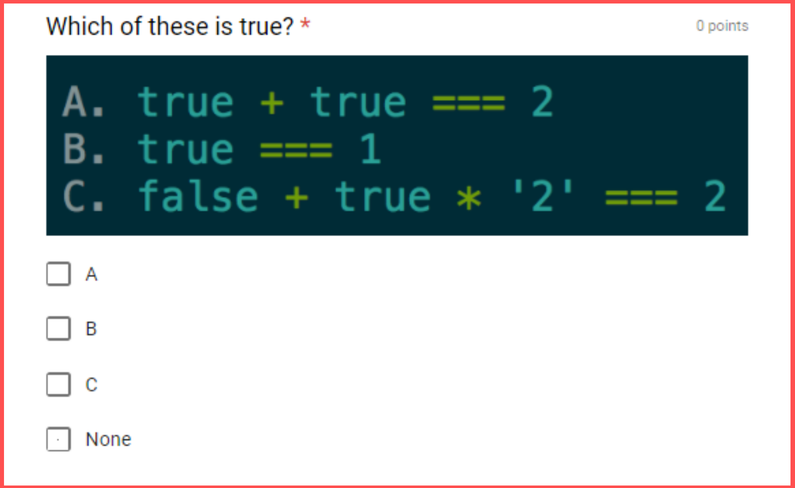
Which of these is true? *

0 points

```
A. true + true === 2
B. true === 1
C. false + true * '2' === 2
```

- [ ] A
- [ ] B
- [ ] C
- [ ] None

A & C!



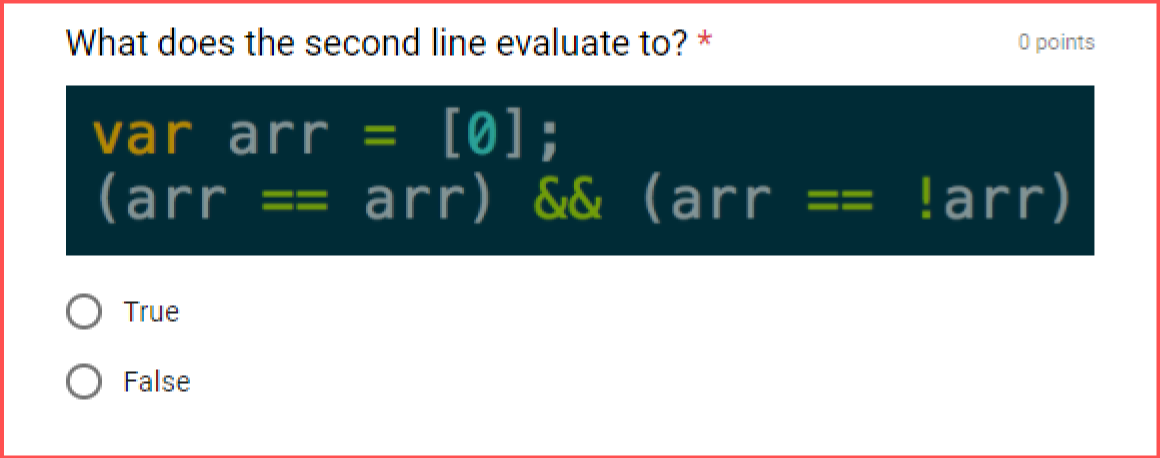
Which of these is true? * 0 points

```
A. true + true === 2
B. true === 1
C. false + true * '2' === 2
```

- ☐ A
- ☐ B
- ☐ C
- ☐ None

- Mathematical operations coerce values
- true == 1, false == 0, '2' == 2
- B) is false because === does not coerce operands!

#9



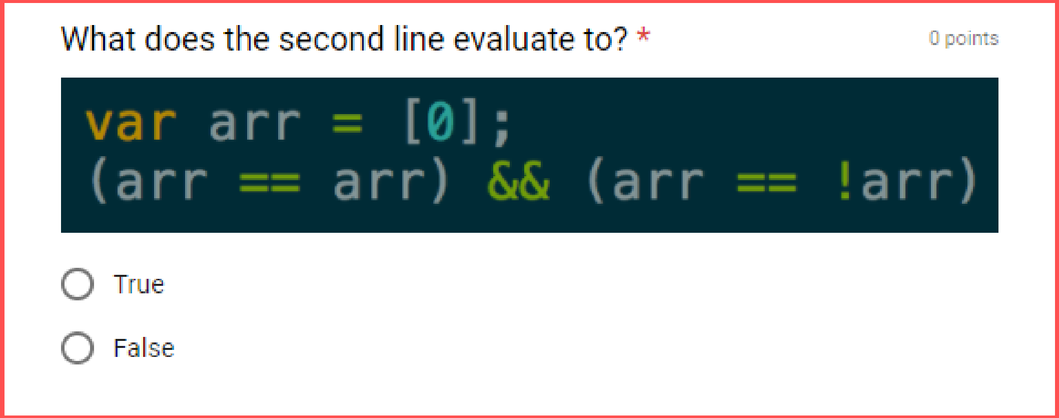
What does the second line evaluate to? * 0 points

```
var arr = [0];
(arr == arr) && (arr == !arr)
```

True

False

TRUE



What does the second line evaluate to? *

0 points

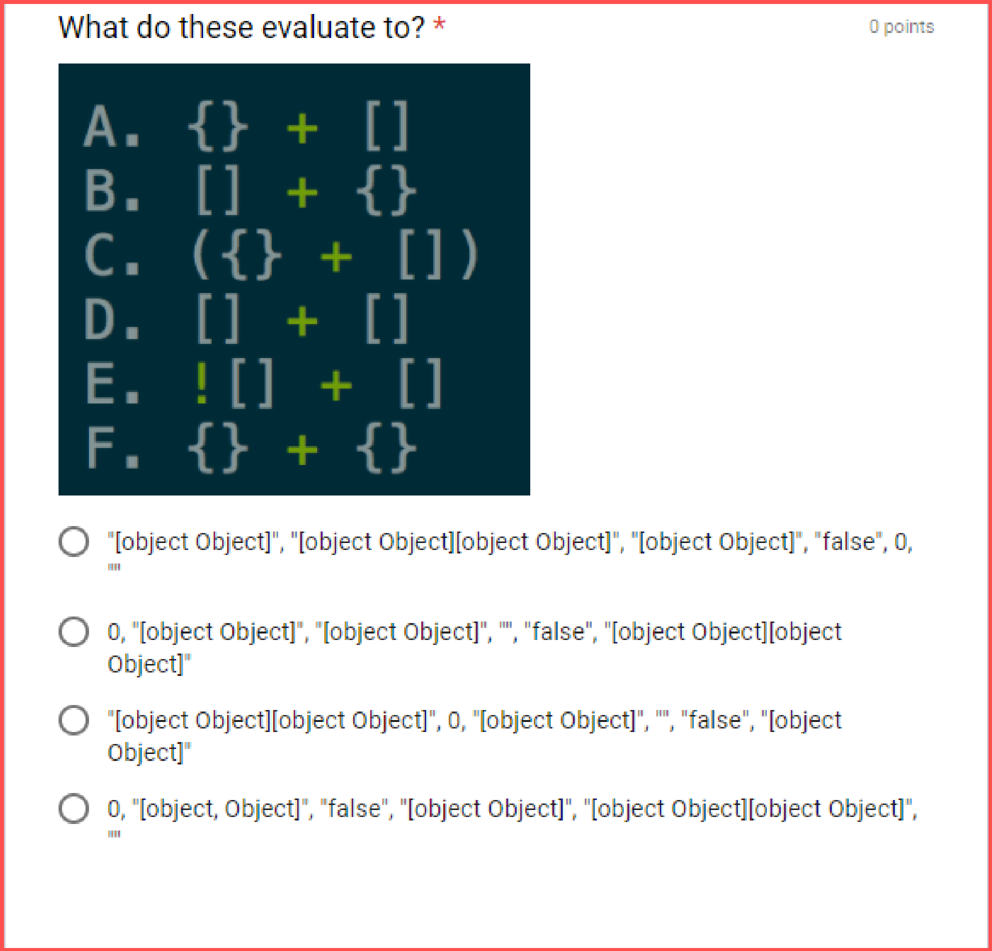
```
var arr = [0];
(arr == arr) && (arr == !arr)
```

True

False

- (arr == arr) just checks equality -> true
- (arr == !arr)
    - !arr checks if array does not exists -> false
    - arr == false coerces arr to boolean
    - arr stringified -> '0'
    - '0' coerced into boolean -> false
    - false === false = true!

#10



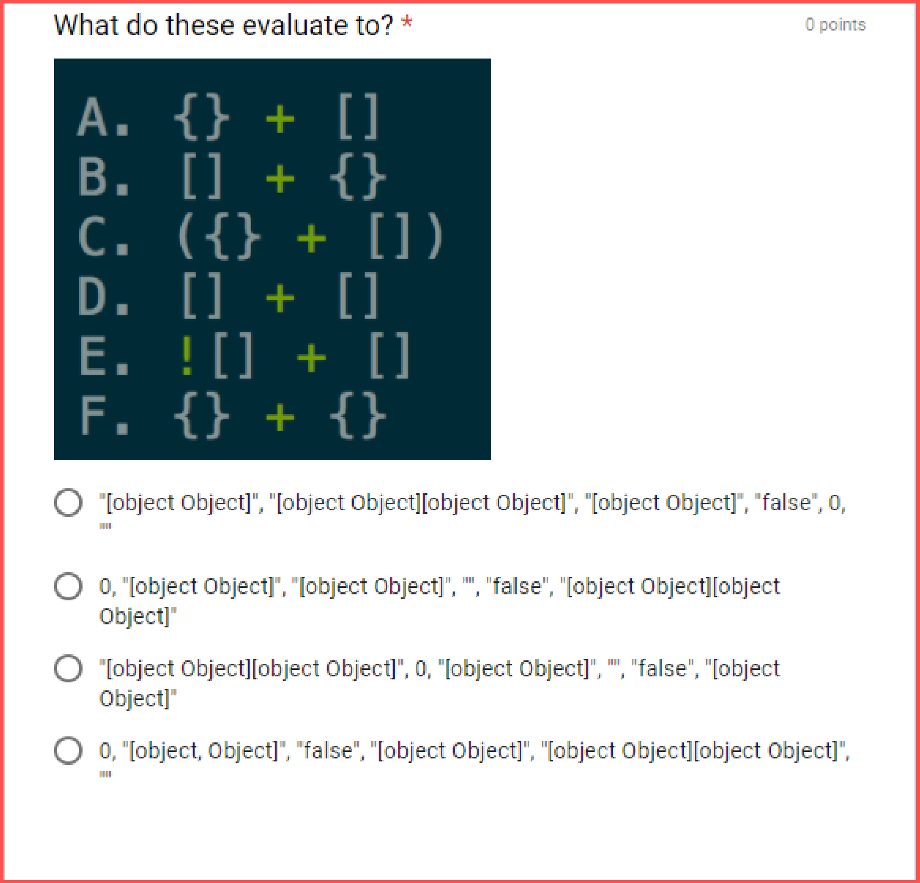
What do these evaluate to? *

0 points

```
A. {} + []
B. [] + {}
C. ({} + [])
D. [] + []
E. ![] + []
F. {} + {}
```

- "[object Object]", "[object Object][object Object]", "[object Object]", "false", 0, ""
- 0, "[object Object]", "[object Object]", "", "false", "[object Object][object Object]"
- "[object Object][object Object]", 0, "[object Object]", "", "false", "[object Object]"
- 0, "[object, Object]", "false", "[object Object]", "[object Object][object Object]", ""

B!



What do these evaluate to? * 0 points

```
A. {} + []
B. [] + {}
C. ({} + [])
D. [] + []
E. ![] + []
F. {} + {}
```

- ○ "[object Object]", "[object Object][object Object]", "[object Object]", "false", 0, ""
- ○ 0, "[object Object]", "[object Object]", "", "false", "[object Object][object Object]"
- ○ "[object Object][object Object]", 0, "[object Object]", "", "false", "[object Object]"
- ○ 0, "[object, Object]", "false", "[object Object]", "[object Object][object Object]", ""