# Question 1

Suppose I have a function

```
1  function myFunction(x){
2    var y = 0;
3      if(y <= 0){
4        let x = y;
5        x = x + 1;
6      }
7      y = y + 1;
8      return y + 1;
9  }
10
11 var x = 1;
12
13 console.log("result = " + myfunction(x));
```

Which of the variables I declared in this function is block-scoped?

- line 11

  var x = 1;

- line 2

  var y = 0;

- line 4 ✔

  {let x = y …}

- line 7

  y = y + 1;

Attempt #1: 1/1 (Score: 1/1)

Correct feedback

Correct! When you use the let keyword to declare a variable, it makes it available only inside a function's block; after that block executes it, it no longer exists.

Teacher feedback

0 / 10000 Word Limit

## Question 2

In this code snippet, what is the value of b?

```
var a = 6;
var b = a;
a = 7;

console.log(b);
```

- (•) 6 ✔
- ( ) null
- ( ) 7
- ( ) undefined

Attempt #1: 2/2 (Score: 2/2)

Correct feedback

Correct! Primitive types in JavaScript cannot be directly manipulated, but they can be replaced.

There are six primitive data types: String, number, bigint, Boolean, undefined, and symbol

Reassigning a variable that holds a primitive type looks like this:

```
var a = 10;
var b = 11;

var b = 12;

console.log(b);
```

>> 12

However, variables that hold primitive types cannot be manipulated. An example of that looks like this:

```
var a = 10;
var b = a;

var a = 12;

console.log(b);
```

>> 10

Since I assigned b to equal a, even if I directly reassign a to a different value, b still points to the value of 10; because the type itself can't be directly manipulated. Changing the value of a with a reassignment, won't change the value of b.

Teacher feedback

0 / 10000 Word Limit

## Question 3

When I pass a variable into a function, what is different about the behaviors of a primitive type variable (string, Boolean, etc.) and a non-primitive type variable (arrays, objects, etc) within the function? Select all that apply.

- [x] Primitive type variables make a copy of themselves when they are passed into a function. ✔
- [ ] Primitive type variables are never available outside the function, whereas non-primitive type values are.
- [x] Non-primitive type variables are passed in by reference, and primitive type variables are passed in by value. ✔
- [ ] Primitive type variables are declared using the keyword let, and non-primitive type variables are declared using the keyword var.

Attempt #2: 1/1 (Score: 1/1)

Correct feedback

Correct! While it may not seem important now, in your programming journey, you will inevitably see the effects of thinking you manipulated a variable that points to a primitive type, which can help you debug errors.

Teacher feedback

0 / 10000 Word Limit

## Question 4

Why would you want to pass arrays and objects into functions by reference, and not by value? Select all that apply.

- [x] It cuts down on the code repeating itself
- [x] It significantly cuts down on the stack's execution time
- [x] Passing by value makes a copy of the variable. If I passed by value, I could potentially make copies of huge amounts of data all the time. But, this would slow me down and bog down the system.
- [x] Because using a pointer/reference saves on memory space

Attempt #3: 1/1 (Score: 1/1)