# CS 105: Sample Midterm Exam #1

### Annotated Copy of Spring 2014's Midterm Exam #1
### Answers and Explanations

**1.** Which of the following lines of JavaScript code will change the HTML inside of an element with the attribute `id="myElement"` to `"CS 105"`?

A. `document.getElementById("myElement").innerHTML = "CS 105";`
B. `html["myElement"] = "CS 105";`
C. `html["myElement"].innerHTML = "CS 105";`
D. `document.html["myElement"].innerHTML = "CS 105";`
E. `html.myElement("CS 105");`

**Explanation**

As part of doing MP3, you learned that the JavaScript code:

`document.getElementById( ___<1>___ ).innerHTML = ____<2>____;`

..changes the element in the HTML document which has an attribute id="**____<1>____**" to the contents of <2>. The only option that follows this syntax is (A).

Additionally, you specifically programmed this line of code as part of completing MP3.

**2.** Which term is not synonymous with the term JavaScript?

A. ECMAScript
B. JS
C. Java
D. All of the terms above are synonymous with JavaScript.

**Explanation**

In Lecture 4.2, you learned that JavaScript, JS, and ECMAScript are all correct terms for JavaScript. However, Java is a completely different language than JavaScript. Saying "Java" when referring to "JavaScript" is incorrect.

**3.** Given the array, `var a = [20, 40, 60, 80, 100]`, which line of code accesses the value 40?

A. `a[0]`
B. `a[1]`
C. `a[2]`
D. `a[3]`
E. `a[i]`

**Explanation**

When accessing elements inside an array, we access them by their index. The first index in every array is zero (0). Therefore a[0] == 20, a[1] == 40, a[2] == 60, and so forth.

This question is looking for 40, which is contained in a[1].

For the next two questions, consider the following code:

```
1  var x = 20;
2  while (x < 10)
3  {
4     print("Hello");
5     x *= 2;
6  }
```

**4.** Assuming that the function `print()` has been defined that prints the value passed in as the parameter to it, how many times is "Hello" printed?

A. 0
B. 1
C. 2
D. 3
E. Infinity (or at least until you kill the program)

**Explanation**

Line 1 initially sets our variable x to 20.

Line 2 checks if x is less than 10.
…is 20 less than 10? **No.**

Since our conditional is false, we skip the code inside of the curly braces. Since there is nothing after Line 6, our program is finished. "Hello" is printed zero (0) times.

**5.** If the initial value of `x` (set on Line #1) is now 5 instead of 20, how many times is `"Hello"` printed?

A. 0
B. 1
C. 2
D. 3
E. Infinity (or at least until you kill the program)

**Explanation**

Line 1 initially sets our variable x to 5.

Line 2 checks if x is less than 10.
…is 5 less than 10? **Yes.**

Since our conditional is true, we run the code inside of the curly braces.
Line 4 prints "Hello"
Line 5 multiples x by 2. Since the old value was 5, the new value is now 5*2, so x = 10.

Line 6 reaches the end of our loop, so the program checks Line 2 again.
…is 10 less than 10? **No.**

Since our conditional is false, we skip the code inside of the braces. Since there is nothing after Line 6, our program is finished. "Hello" is printed 1 time.

For the next two questions, consider the following code:

```
 1  function foo(x)
 2  {
 3     var ct = 0;
 4     for (var i = 0; i < x.length; i++)
 5     {
 6        if (x[i] >= 70)
 7        {
 8           ct++;
 9        }
10     }
11     return ct;
12  }
```

**6.** Based on its usage in the function, what type of variable must **`x`** be?

A. A string
B. A number
C. An array of strings
D. An array of numbers
E. An array of arrays

**Explanation**

This question requires understanding of how the variable is used. Looking at how x is used:
Line #4: x.length
Line #6: x[i] >= 70

Since the variable x was asked for its length (by doing **.**length on Line #4), it must be either an array or a string as those are the only two things that have lengths associated with them.

On Line 6, x was accessed with by the JavaScript code x[i] and then that value is compared to a number (**`x[i] >= 70`**). The access of x[i] implies x is an array and the comparison with a number implies that x is an array of numbers.

```
 1  function foo(x)
 2  {
 3     var ct = 0;
 4     for (var i = 0; i < x.length; i++)
 5     {
 6        if (x[i] >= 70)
 7        {
 8           ct++;
 9        }
10     }
11     return ct;
12  }
```

**7.** Which of the following statements are true about the code above?

A. The function will always return the value 0.
B. The function will never return the value 0.
C. The function will never return.
D. The function only returns if x is a number.
E. The function will never return a negative number.

**Explanation**

Besides simply finding the correct statement, we can also arrive at the correct statement by eliminating all of the incorrect statements. To do so, we will imagine sample inputs to the function and see what they will do.

For the first input, assume `x=[100]` (that is, an array with one element).
…when the function runs, the function returns the number 1.

Since 1 is returned, (A), (C), and (D) are all **false**. We are left with only (B) and (E).

For the second input, assume `x=[0]`.
…when the function runs, the function returns the number 0.

Since 0 is returned, (B) is also **false**. Therefore, (E) must be correct.

*[This question refers to Spring 2014's MP1, may not make sense outside of Spring 2014.]*

**8.** In MP1, you solved a maze using calls to `isWall(direction)` and `move(direction)`, where the direction parameter was either `"Forward"`, `"Left"`, `"Right"`, or `"Backward"`. Which of the following lines of code will never result in moving into a wall?

A.
```
if (isWall("Forward") && isWall("Left") && isWall("Right"))
{
   move("Backward");
}
```

B.
```
if (isWall("Left") || isWall("Right"))
{
   move("Left");
}
```

C.
```
if (isWall("Left") && isWall("Right"))
{
   move("Right");
}
```

D. Both (B) and (C) are correct.

E. All of (A), (B), and (C) are correct.

**Explanation**

This problem was based on a different MP1, though the underlying concept can still be observed. If you assume isWall() returns if a wall exists in given direction, which was the case, then we can run through this problem.

Answer (B) looks to see if there is a wall either to the **Left** or to the **Right**. If there is a wall either to the left or right, the computer is instructed to move **Left**. Since there might be a wall to the left (it's either to our left or our right), this does not guarantee that there is not a wall.

Answer (C) looks to see if there is a wall either to the **Left** and to the **Right**. If there is a wall either to the left and right, the computer is instructed to move **Right**. This movement would always run into a wall.

Answers (D) and (E) both include answers (B) and (C), so they also can be eliminated. The only instruction that always avoids a wall is (A).

For the next four problems, consider a payment such as a rent or a credit card payment. The variable **`daysLate`** stores the value of the number of days you are late on a payment.

**9.** If the penalty is $20 per day late, which of the following lines of code accurately calculates the late penalty?

A. `var penalty = daysLate + 20;`
B. `var penalty = daysLate * 20;`
C. `var penalty = daysLate ^ 20;`
D. `var penalty = daysLate && 20;`
E. `var penalty = daysLate;`

**Explanation**

Since the penalty is $20 per day, a payment is 4 days late will result in an $80 penalty. The math to do this is simply:

penalty = (days late) * $20

Translating this into JavaScript, (B) is the only answer that makes any sense at all.

**10.** For a different payment, the penalty is calculated with the following code:

```
1  var penalty;
2  if (daysLate < 3)
3  {
4     penalty = 0;
5  }
6  else
7  {
8     penalty = daysLate - 3;
9  }
```

What is an accurate English description of the code displayed above?

A. If the payment is **less than** three days late, the penalty is the number of days the payment was late. Otherwise, there is no penalty.
B. If the payment is **more than** three days late, the penalty is the number of days the payment was late. Otherwise, there is no penalty.
C. If the payment is **less than** three days late, the penalty is three dollars less than the number of days the payment was late. Otherwise, there is no penalty.
D. If the payment is **more than** three days late, the penalty is three dollars less than the number of days the payment was late. Otherwise, there is no penalty.
E. The penalty is always $20. That's simple enough, right?

Explanation

This problem asks you to take JavaScript and translate the code into English. To do so, we need to understand the code and how it runs. Running with some sample input, we can look at the output to understand the function:

daysLate == 0 ➔ penalty == 0
daysLate == 1 ➔ penalty == 0
daysLate == 2 ➔ penalty == 0
daysLate == 3 ➔ penalty == 3 – daysLate = 3 - 3 == 0
daysLate == 4 ➔ penalty == 3 – daysLate = 4 - 3 == 1
daysLate == 5 ➔ penalty == 3 – daysLate = 5 - 3 == 2
daysLate == 6 ➔ penalty == 3 – daysLate = 6 - 3 == 3
…

From this, it becomes easy to examine each answer statement. Only (D) correctly corresponds to the output of the code.

**11.** For yet another payment, this payment offers a grade period of **`gracePeriod`** days where no penalty will be given if the payment is made within the number of grace period days. (This means that, if the grace period is three days, no penalty is given if the payment is made is made one, two, or three days late.) What is the conditional that is true if the payment is made within the grace period?

A. `if ( gracePeriod < daysLate )`
B. `if ( gracePeriod <= daysLate )`
C. `if ( gracePeriod > daysLate )`
D. `if ( gracePeriod >= daysLate )`
E. `if ( gracePeriod == daysLate )`

**Explanation**

If the grace period is 3, as the question's example states, then we are inside of our grace period if the days late is either 0, 1, 2, or 3.

The question asks for the conditional that is true if we are within our grace period, so we need a Boolean expression that is true when daysLate is equal to or less than gracePeriod.
…translating this from English to JavaScirpt, daysLate <= gracePeriod.

However, this is not an option. We can logically reverse it, giving us (daysLate >= gracePeriod), which yields the correct answer. However, even not being sure of the logical reversal, we can test each answer with input. We know that we must come with an expression that has the following properties

gracePeriod = 3, daysLate = 0 must be TRUE
gracePeriod = 3, daysLate = 1 must be TRUE
gracePeriod = 3, daysLate = 2 must be TRUE
gracePeriod = 3, daysLate = 3 must be TRUE
gracePeriod = 3, daysLate = 4 must be FALSE
gracePeriod = 3, daysLate = 5 must be FALSE
…

Checking (A),
gracePeriod = 3, daysLate = 0 ➔ (3 < 0) ➔ FALSE
…but this must be TRUE, so (A) is incorrect.

Checking (B):
gracePeriod = 3, daysLate = 0 ➔ (3 <= 0) ➔ FALSE
…but this must be TRUE, so (B) is incorrect.

Checking (C):
gracePeriod = 3, daysLate = 0 ➔ (3 > 0) ➔ TRUE …so far, so good.
gracePeriod = 3, daysLate = 1 ➔ (3 > 1) ➔ TRUE …so far, so good.
gracePeriod = 3, daysLate = 2 ➔ (3 > 2) ➔ TRUE …so far, so good.
gracePeriod = 3, daysLate = 3 ➔ (3 > 3) ➔ FALSE… yikes, this is incorrect. (C) is out.

Checking (E):
gracePeriod = 3, daysLate = 0 ➔ (3 == 0) ➔ FALSE
…but this must be TRUE, so (E) is incorrect.

The only answer that works for every case is (D).

**12.** What type of data is stored in the variable **`daysLate`**?

A. A number
B. A string
C. An array of numbers
D. An array of strings
E. An array of arrays

**Explanation**

Having worked with the variable daysLeft for the past three problems, it should be clear that this is a number.

**13.** What is the decimal (base 10) value of the following binary number: 101

A. 3
B. 5
C. 6
D. 7
E. 9

**Explanation**

In Lecture 3.2, you saw translations of binary numbers to decimal numbers. By using the place value of each digit, we can do some easy translation between binary and decimal:

```
Place Value (raw):         2^2   2^1   2^0
Place Value (calculated): 4     2     1
                          *     *     *
Number to Translate:      1     0     1     (from the problem)

...doing multiplication
   down the columns     :  4     0     1

...and adding the results
   of our multiplication: 4 + 0 + 1 = 5
```

**14.** What is the decimal (base 10) value of the following binary number: 011

A. 3
B. 5
C. 6
D. 7
E. 9

**Explanation**

In Lecture 3.2, you saw translations of binary numbers to decimal numbers. By using the place value of each digit, we can do some easy translation between binary and decimal:

```
Place Value (raw):         2^2   2^1   2^0
Place Value (calculated): 4     2     1
                          *     *     *
Number to Translate:      0     1     1     (from the problem)

...doing multiplication
   down the columns     :  0     2     1

...and adding the results
   of our multiplication: 0 + 2 + 1 = 3
```

**15.** In investment banking, one property of a good loan is a healthy Loan-To-Value ratio (LTV). An LTV is calculated by taking the value of the loan and dividing it by the value of the property. An LTV of 0.8 or less is considered **safe**, an LTV of 1.0 or greater is considered **dangerous**, and all other LTVs are considered **risky**. Consider three functions that return the safety of a given property given its LTV:

**(i)**

```
function safety(ltv)
{
  if (ltv >= 1.0)
  {
    return "dangerous";
  }
  else if (ltv <= 0.8)
  {
    return "safe";
  }
  else
  {
    return "risky";
  }
}
```

**(ii)**

```
function safety(ltv)
{
  if (ltv <= 0.8)
  {
    return "safe";
  }
  else if (ltv > 1.0)
  {
    return "dangerous";
  }
  else
  {
    return "risky";
  }
}
```

**(iii)**

```
function safety(ltv)
{
  if (ltv <= 0.8)
  {
    return "safe";
  }

  if (ltv > 1.0)
  {
    return "dangerous";
  }

  return "risky";

}
```

Which one of the function(s) will always return the correct safety rating?

A. Only (i)
B. Only (i) and (ii)
C. Only (i) and (iii)
D. Only (ii)
E. (i), (ii), and (iii)

**Explanation**

In this entire exam, this problem requires the most attention to detail. Besides the code being rearranged and refactored, the key difference between (i) and (ii)/(iii) is that the conditional for "dangerous" is `ltv >= 1.0` vs. `ltv > 1.0`.

Since the problem states "an LTV of 1.0 or greater", we must include 1.0 inside our dangerous category. The only one to do this is (i).

**16.** What is ASCII code?

A. A special version of JavaScript that we are using in CS 105
B. A universally recognized translation between letters and numbers
C. An encryption technique that makes text hard to read
D. A programming language used to make Android and iPhone apps
E. The five primary colors of light used on a computer screen

**Explanation**

Simple application of a definition that was defined in lecture, lab, and on MP2.

For the next five questions, consider the following array:

```
var a = ["Red", "Orange", "Yellow", "Green", "Purple"];
```

**17.** What is value of `a.length`?

A. 5
B. 6
C. 7
D. 8

**Explanation**

The JavaScript code:

```
____.length
```

…will always return the number of elements in an array. By simply counting, we find there are five strings inside of the array a.

**18.** What is the value of `a[3]`?

A. "Red"
B. "Orange"
C. "Yellow"
D. "Green"
E. "Blue"

**Explanation**

When dealing with arrays, we always starting counting from index 0. Starting from 0,
a[0] == "Red"
a[1] == "Orange"
a[2] == "Yellow"
a[3] == "Green"
a[4] == "Purple"

The value of a[3] is "Green".

```
var a = ["Red", "Orange", "Yellow", "Green", "Purple"];
```

For the next three questions, consider the following code that continues to use the array `a` that is defined at the top of this page:

```
 1  var result = "";
 2  for (var i = 0; i < a.length; i++)
 3  {
 4     var s = a[i];
 5     var c = s.charAt(i);
 6     if (c == "e")
 7     {
 8        result += s;
 9     }
10  }
```

**19.** What is the value of `s` the **second** time the loop is run?

A. `"Red"`
B. `"Orange"`
C. `"Yellow"`
D. `"Green"`
E. `"Blue"`

**Explanation**

In answering this question, you have to execute the code as though you were a computer.

Staring with Line 1, we create a variable `result` and set it equal to an empty string.

Line number 2 is a for-loop, which first creates a new variable `i` and sets it equal to 0. The computer would then check the conditional to see if it should run the code inside of the loop. Checking `(i < a.length)`, we are checking (0 < 5), which is true.

Now we are inside the **first** iteration of the loop. Line 4 assigns the new variable s to the value contained at the i-th element of the array a. Since i is 0, a[i] is a[0], and a[0] is "Red". Therefore, s is set to "Red".

Line 5 assigns yet another new variable, this time variable c, to be equal to s.charAt(i). Since s is "Red" and i is 0, c = "Red".charAt(0). Asking "Red" for its $0^{th}$ index character, "Red" returns "R" and c is set to "R".

Line 6 checks if variable c is equal to "e". Since c is "R", this is false and we skip Line 8.

Reaching the end of our for-loop, the computer jumps back to Line 2 and runs the third part of the for-loop: i++. The variable i is changed from 0 to 1. Then, the conditional it checked again, `(i < a.length)`, which now evaluates as (1 < 5), which is still true.

Now inside the **second** iteration of our loop. Line 4 sets s = a[i] = a[1] = "Orange".

The question asks for the value of s in the second run, which we just found to be "Orange".

```
var a = ["Red", "Orange", "Yellow", "Green", "Purple"];
```

```
 1  var result = "";
 2  for (var i = 0; i < a.length; i++)
 3  {
 4     var s = a[i];
 5     var c = s.charAt(i);
 6     if (c == "e")
 7     {
 8        result += s;
 9     }
10  }
```

**20.** What is the value of `c` during the **third** time the loop is run?

A. `"p"`
B. `"u"`
C. `"r"`
D. `"l"`
E. `"e"`

**Explanation**

Continuing from the previous explanation, we can continue our process of tracing the code. Instead of the line-by-line detail, we will simply look at the result of each line. Staring back at the beginning of the program:

```
Line 1:  result = ""
Line 2:  i = 0,   (0 < 5) → TRUE
  /* First run of the loop */
  Line 4:  s = a[i] = a[0] = "Red"
  Line 5:  c = s.charAt(i) = "Red".charAt(0) = "R"
  Line 6: (c == "e") → ("R" == "e") → FALSE
Line 2:  i++ → i = 1,  (1 < 5) → TRUE
  /* Second run of the loop */
  Line 4:  s = a[i] = a[1] = "Orange"
  Line 5:  c = s.charAt(i) = "Orange".charAt(1) = "r"
  Line 6: (c == "e") → ("r" == "e") → FALSE
Line 2:  i++ → i = 2,  (2 < 5) → TRUE
  /* Third run of the loop */
  Line 4:  s = a[i] = a[2] = "Yellow"
  Line 5:  c = s.charAt(i) = "Orange".charAt(2) = "l"
  Line 6: (c == "e") → ("l" == "e") → FALSE
Line 2:  i++ → i = 2,  (3 < 5) → TRUE
  /* Fourth run of the loop */
  Line 4:  s = a[i] = a[3] = "Green"
  Line 5:  c = s.charAt(i) = "Green".charAt(3) = "e"
  Line 6: (c == "e") → ("e" == "e") → TRUE
    Line 8:  result → "Green"
Line 2:  i++ → i = 2,  (4 < 5) → TRUE
  /* Fifth run of the loop */
  Line 4:  s = a[i] = a[4] = "Purple"
  Line 5:  c = s.charAt(i) = "Orange".charAt(4) = "p"
  Line 6: (c == "e") → ("p" == "e") → FALSE
Line 5:  i++ → i = 2,  (5 < 5) → FALSE
/* Program complete! */
```

Looking at the value of c during the third run (highlighted), c is equal to "l".

```
var a = ["Red", "Orange", "Yellow", "Green", "Purple"];
```

```
 1  var result = "";
 2  for (var i = 0; i < a.length; i++)
 3  {
 4     var s = a[i];
 5     var c = s.charAt(i);
 6     if (c == "e")
 7     {
 8        result += s;
 9     }
10  }
```

**21.** What the value in `result` after the code has finished running?

A. `"Green"`
B. `"RedYellow"`
C. `"RedOrangeYellow"`
D. `"RedYellowGreenPurple"`
E. `"RedOrangeYellowGreenPurple"`

**Explanation**

In English, the code looks through the array and adds the word found at the $i^{th}$ index of the array to the result if and only if the $i^{th}$ index letter in the word is equal to an "e".

For "Red", the $0^{th}$ index in the array, it checks if "R" (the $0^{th}$ index letter in "Red") is an "e".
For "Orange", the $1^{th}$ index in the array, it checks if "r" (the $1^{th}$ index letter in "Orange") is an "e".
For "Yellow", it checks if "l" is an "e".
For "Green", it checks if "e" is an "e". Since it is, "Green" is added to the result.
For "Purple", it checks if "l" is an "e".

You can see this logic is also shown in the previous problem's explanation where we trace through the entire execution of the code.

Since "Green" is the only string to get added to result, the result only contains "Green". This also matches with the previous problem's explanation where result is only changed inside the fourth iteration of the loop.

**22.** Which of the following is **not** a primary color used by computers to represent every possible color on a traditional computer screen?

A. Blue
B. Green
C. Red
D. Yellow

**Explanation**

In Lecture 5.2, you learned that all images are made up of the **primary colors of light**: red, green and blue. These are the secondary colors of pigment you might be used to.

**23.** Which of the following array(s) can be searched using a **linear search**?

A. `["apple", "banana", "blackberry", "grape", "kiwi"]`
B. `["city", "crow", "country", "cow"]`
C. `["university", "of", "illinois", "at", "urbana", "champaign"]`
D. Both (A) and (B), but not (C)
E. (A), (B), and (C)

**Explanation**

In Lecture 5.1, you learned that data must be sorted in order to be used in a **binary search**. However, data does not need to be sorted in a **linear search**. Since the data does not need to be sorted, any type of data can be sorted and the answer is (E).

**24.** Which of the following array(s) can be searched using a **binary search**?

A. `["apple", "banana", "blackberry", "grape", "kiwi"]`
B. `["city", "crow", "country", "cow"]`
C. `["university", "of", "illinois", "at", "urbana", "champaign"]`
D. Both (A) and (B), but not (C)
E. (A), (B), and (C)

**Explanation**

In Lecture 5.1, you learned that data must be sorted in order to be used in a **binary search**. The arrays in (B) and (C) are not sorted (at least one word is not in alphabetical order in the array), so only (A) can be searched using a binary search.

**25.** After one pass (iteration) of a standard **selection sort** (as shown in lecture), what is always true about the array?

A. The first element in the array is in the correct location.
B. The middle element in the array is in the correct location.
C. The last element in the array is in the correct location.
D. Half of the array has been thrown out, since it is no longer needed.

**Explanation**

In Lecture 5.1, you learned about the **selection sort**. In this sort, the first pass of the selection sort found the smallest element and placed it at the beginning of the list. Therefore, the only true statement is (A).

**26.** Consider the following three code snippets:

| (i) | (ii) | (iii) |
|---|---|---|
| `var a = [2, 4, 6];`<br>`var b = a[0] + a[1];` | `var a = [1, 3, 5];`<br>`var b = a[1] + a[2];` | `var a = [3, 6, 9];`<br>`var b = a[2] – a[0];` |

Which code snippet(s) will have the variable `b` equal to 6?

A. Only (i)
B. Only (i) and (ii)
C. Only (i) and (iii)
D. Only (ii)
E. (i), (ii), and (iii)

**Explanation**

By simply running the code:

(i):
a = [2, 4, 6];
b = a[0] + a[1] == 2 + 4 == 6

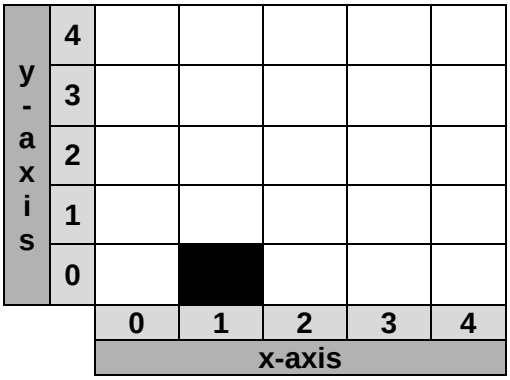
(ii):
a = [1, 3, 5];
b = a[1] + a[2] == 3 + 5 == 8

(iii):
a = [3, 6, 9]
b = a[2] – a[0] == 9 – 3 == 6

From above, (i) and (iii) both result in the variable b becoming 6, so the answer is (C).

**Stenography** is the practice of concealing data within another form of data. You saw this in lecture with images, but now you will practice stenography right here on your CS 105 midterm exam.

In the JavaScript below, you will notice that there is a call to the function `fill(x, y)`. When this function is called by the code, you should fill in the cell at the location specified. For example, if the variable `A` has the value of 1 and `B` has the value of 0, a call to `fill(A, B)` should fill the cell at (1, 0). *We have filled this cell for you already as an example.*




Consider the following code for the next three questions:

```
1  var a = [1, 2, 2, 2, 1];
2  for (var i = 0; i < a.length; i++)
3  {
4     if (a[i] == 1)
5     {
6        fill(1, i);
7        fill(3, i);
8     }
9     fill(2, i);
10 }
```

**27.** When the for-loop runs for the first time, how many times is the `fill()` function called?

A. One time
B. Two times
C. Three times
D. Four times
E. Five times

**Explanation**

When the for-loop runs for the first time, the variable i is 0 and a[0] == 1.

Since a[0] is 1, Line 4 is **true** and Line 6 and 7 are ran during the first run of the loop. Since fill() is called on Lines 6, 7, and 9, fill() is called a total of three times.

**28.** When the for-loop runs for the first time, which of the following is happening?

A. Several cells within a single **row** are filled.
B. Several cells within a single **column** are filled.

**Explanation**

From the previous problem, we know that Lines 6, 7 and 9 are all ran. Filling in the value of I for those lines:

fill(1, 0);
fill(3, 0);
fill(2, 0);

…these commands all fill up the cells in the bottom **row**, meaning rows are filled.

Consider the following code for the next three questions:

```
 1  var a = [1, 2, 2, 2, 1];
 2  for (var i = 0; i < a.length; i++)
 3  {
 4     if (a[i] == 1)
 5     {
 6        fill(1, i);
 7        fill(3, i);
 8     }
 9     fill(2, i);
10  }
```

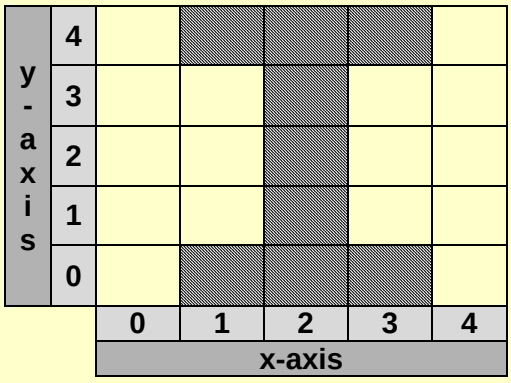
**29.** After the program has completed, what is the state of the cell at (2, 2)?

A. It is filled.
B. It is **not** filled.

**Explanation**

Running the code to completion, we find the following fill() functions are called:

fill(1, 0)
fill(3, 0)
fill(2, 0)
fill(2, 1)
fill(2, 2)
fill(2, 3)
fill(1, 4)
fill(3, 4)
fill(2, 4)




These functions were ran to fill in the chart on the right (➔), which makes an Illini "`I`". Checking cell (2, 2), we find it is indeed filled.

**30.** When the program has completed, what letter is made on the grid?

A. `I`
B. `O`
C. `A`
D. `H`
E. `K`

**Explanation**

(See previous explanation)

***This page was intentionally left blank.***