Final Test

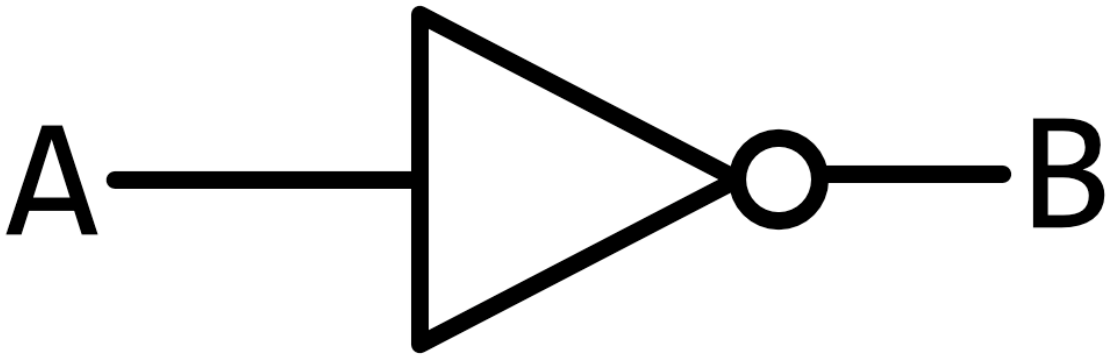
(25 points)

Logical Questions

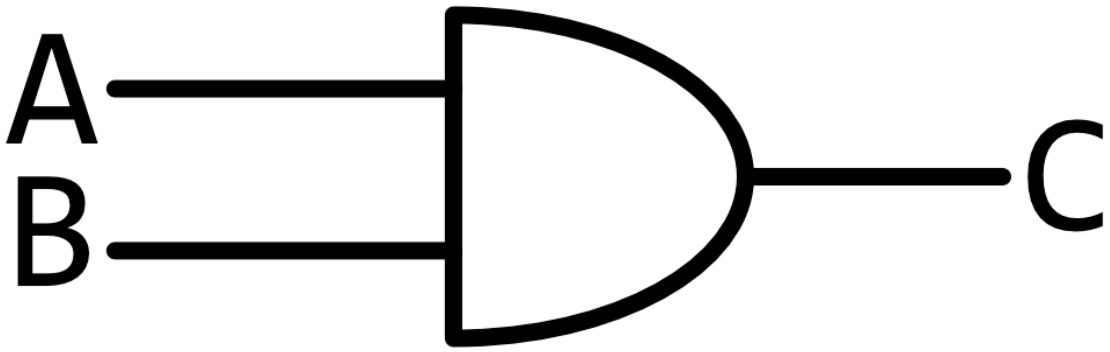
Even though their syntax differs, Scratch, C, Python, and SQL all have “logical operators” via which you can express “and”, “or”, and “not”, each of which evaluates to a Boolean value, “true” or “false”.

It turns out that these logical operators are among the fundamental building blocks of computer circuits, the wiring that makes everything work. Those circuits can be visualized via what are known as “circuit diagrams.” Those circuit diagrams are composed of “logic gates,” each of which corresponds to a logical operation. You can think of a logic gate as a tiny piece of hardware that has one or more wires as input and one wire as output. If no electricity is flowing on some wire, it represents false (aka 0); if electricity is flowing on some wire, it represents true (aka 1).

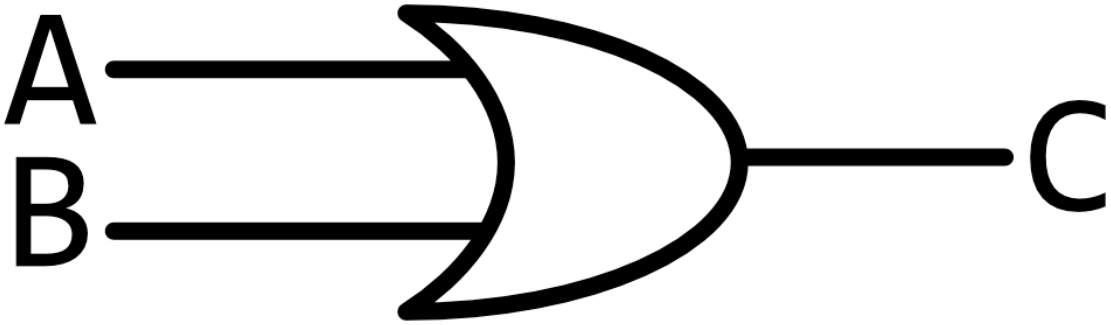
Below is a “NOT gate,” with one input, A, and one output, B. If the input is 1, then the output is 0. If the input is 0, then the output is 1.



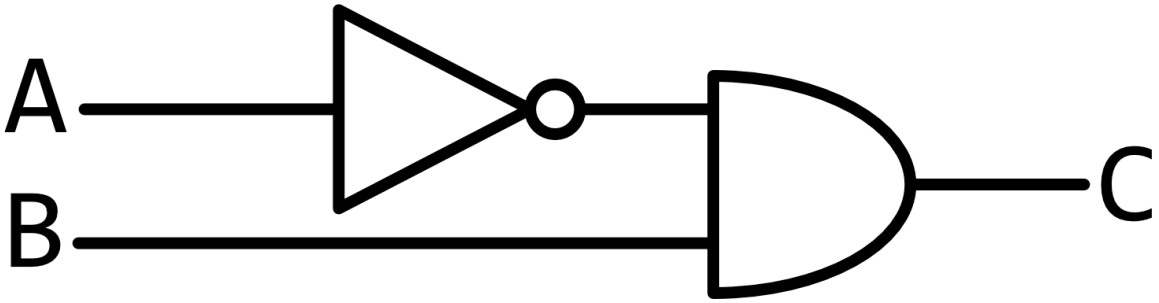
Below is an “AND gate,” with two inputs, A and B, and one output, C. If both inputs are 1, then the output is 1. If either or both inputs are 0, then the output is 0.



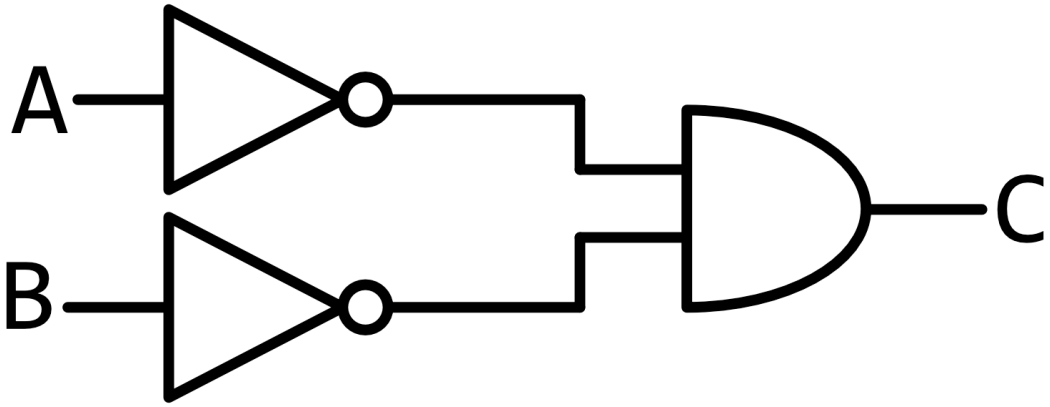
Below is an “OR gate,” with two inputs, A and B, and one output, C. If either or both inputs are 1, then the output is 1. If both inputs are 0, then the output is 0.



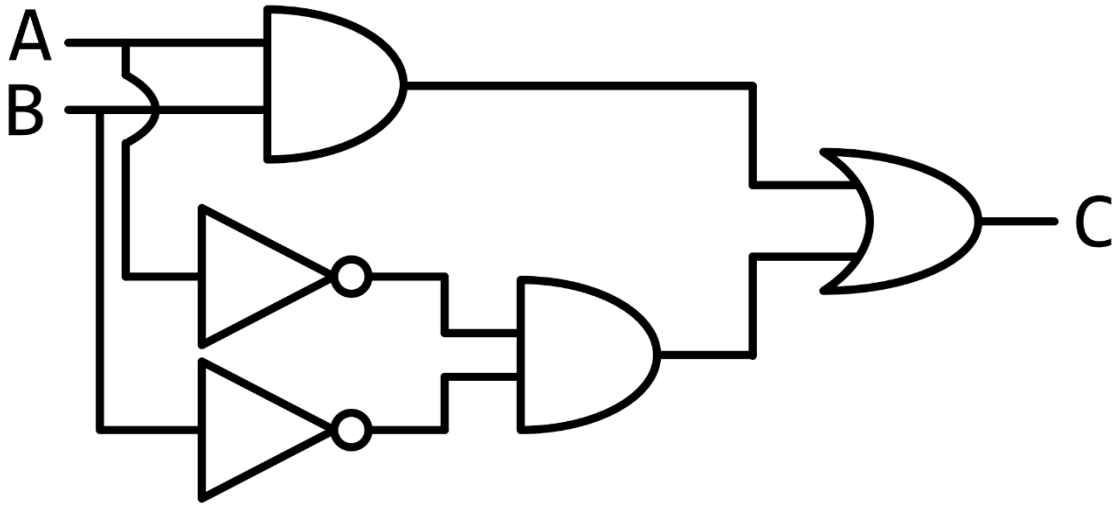
These logic gates can be combined to form more-complex circuits. For example, consider the circuit below, with two inputs, A and B. This circuit inverts A (but not B) via a NOT gate, passing its output and B into an AND gate, the output of which is C. As a result, C will be 1 if, and only if, A is 0 and B is 1; otherwise, C will be 0.



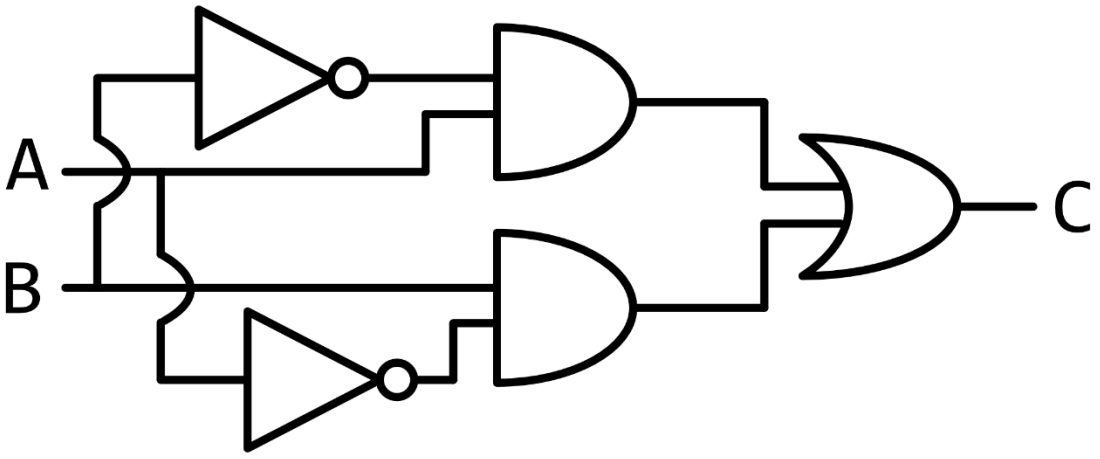
1. (0.5 points.) Consider the circuit below. In no more than three sentences, under what circumstances would the circuit output 1?



2. (0.75 points.) Consider the circuit below. In no more than three sentences, under what circumstances would the circuit output 1? Note that A splits into two wires: whatever its value is propagates to both wires. The same goes for B. Also note that the curved line in the circuit represents a wire that travels over or under the horizontal wire, without actually touching it.



3. (0.75 points.) Consider the circuit below. In no more than three sentences, under what circumstances would the circuit output 1? Note that A splits into two wires: whatever its value is propagates to both wires. The same goes for B. Also note that the curved lines in the circuit represent wires that travel over or under the horizontal wires, without actually touching them.



Programming in B

Suppose that Brian has designed his own programming language called B. The language is “loosely typed,” which means you needn’t specify a variable’s type: values that look like integers (e.g., `-1`, `0`, `1`, et al.) will be treated as integers, values that look like strings will be treated as strings, and `true` and `false` will be treated as Boolean values. As in C, comments in B begin with `//`. But the language comprises only these building blocks, wherein `{}` represents one or more lines of code:

`function name(parameters) {}`

which defines a function called `name` that accepts as input a comma-separated list of zero or more parameters

`return(value)`

which returns a single `value` from a function

`$variable`

which represents a variable (of any type), whose name must start with a dollar sign, called `$variable`

`x <- y`

which stores `y` in `x`, where `x` represents a variable

`if (x) {}`

which executes any code within `{}` if `x` is `true` (or any value other than `0`)

`print(x)`

which prints `x` (which can be of any type)

`add(x, y)`

which returns `x` plus `y`, which are assumed to be integers

`subtract(x, y)`

which returns `x` minus `y`, which are assumed to be integers

`multiply(x, y)`

which returns `x` times `y`, which are assumed to be integers

`divide(x, y)`

which returns `x` divided by `y`, which are assumed to be integers, using integer division (and truncating) like C

`greater(x, y)`

which returns `true` if `x` is greater than `y` or `false` if `x` is not greater than `y`

As in C, function calls can be nested: one function's output can be passed to another function as input. But no need for semicolons!

B has no other features. In particular, it does not support binary operators like `+`, `-`, `*`, `/`, `%`, `&&`, `||`, `!`, `>`, `>=`, `==`, or `<=`, though it does support `-` as a prefix for negative integers. But you can still program in it!

For instance, whereas in C you might write:

```
int c = a - b;
```

in B you would write:

```
$c <- subtract($a, $b)
```

1. (1 points.) In creating B, it seems Brian forgot to add support for the boolean "or" operator (which is `||` in C). Complete the implementation of `or`, below, which accepts two Boolean values, `$a` and `$b`, as input and returns `true` if either or both of `$a` and `$b` are `true`, otherwise returning `false`, using only those building blocks that Brian did implement.

   ```
   function or($a, $b)
   {
     // TODO
   }
   ```

2. (1 points.) In creating B, it seems Brian forgot to add support for the boolean "and" operator (which is `&&` in C). Complete the implementation of `and`, below, which accepts two Boolean values, `$a` and `$b`, as input and returns `true` if both `$a` and `$b` are `true`, otherwise returning `false`, using only those building blocks that Brian did implement.

   ```
   function and($a, $b)
   {
     // TODO
   }
   ```

3. (1 points.) In creating B, it seems Brian forgot to add support for the remainder operator (which is `%` in C), which computes the remainder when one integer is divided by another. (For example, `8 % 3` would be `2`). Complete the implementation of `remainder`, below, which accepts two integers, `$a` and `$b`, as

input and returns the remainder when dividing `$a` by `$b`, using only those building blocks that Brian did implement. Assume that both `$a` and `$b` will be positive integers.

```
function remainder($a, $b)
{
  // TODO
}
```

4. (1.25 points.) In creating B, it seems Brian forgot to add support for loops. Nonetheless, complete the implementation of `meow`, below, which should print `"meow"` a total of `$n` times, using only those building blocks that Brian did implement. No need to include any `\n`. Assume that `$n` will be a non-negative integer.

```
function meow($n)
{
  // TODO
}
```

Key Questions

Some social networks, including Twitter and Instagram, allow one user (a “follower”) to follow another user (a “followee”). Let’s consider how such networks might represent data about their users. A social network might, for instance, have a SQL database with tables like the below.

```
CREATE TABLE users (
    username TEXT UNIQUE,
    name TEXT,
    PRIMARY KEY(username)
);

CREATE TABLE followers (
    follower TEXT,
    followee TEXT,
    FOREIGN KEY (follower) REFERENCES users(username),
    FOREIGN KEY (followee) REFERENCES users(username)
);
```

Assume that Ileana is registered for this social network with a username of `ileana`, that Reese is registered with a username of `reese`, and that Max is registered with a username of `max`.

1. (0.75 points.) Suppose that Max starts following Ileana. What (single) SQL statement should be executed?

2. (1 points.) With what (single) SQL query could you select the usernames of every user that follows Reese and whom Reese also follows back?

3. (1 points.) Suppose that Reese is looking for additional users to follow. One suggestion that a social network might provide is to suggest users who are “two degrees of separation” away: users who are followed by users whom Reese already follows. With what (single) SQL query could you select the usernames of users who are followed by users whom Reese follows?

Social networks like Twitter and Instagram support asymmetric relationships: Alice might follow Bob, but that does not mean that Bob also follows Alice. Other social networks, like Facebook, support symmetric relationships as well: in order for Alice and Bob to be “friends,” one of them must send a “friend request” to the other, which the other must then accept.

4. (1 points.) With what SQL statement could you create a `friendships` table that allows for the representation of friendships and friend requests? Assume that `users` exists as above.

5. (0.75 points.) Suppose that Max sends a friend request to Ileana. What (single) SQL statement should be executed?

6. (0.75 points.) Suppose that Ileana accepts Max's friend request. What (single) SQL query should be executed?

Random Questions

Consider the implementation of `sort` , below, which takes a `list` as input and returns a sorted version thereof. It calls `issorted`, also below, to determine whether those numbers are sorted. It also imports a module, `random`, and calls `random.shuffle` therein. We've included the bottommost line so that you can try out the code in CS50 Sandbox or CS50 IDE, but assume that `numbers` could be any `list` of size *n*.

```
def issorted(numbers):
    if sorted(numbers) == numbers:
        return True
    else:
        return False
def sort(numbers):
    import random
    while not issorted(numbers):
        random.shuffle(numbers)
    return numbers
print(sort([6, 3, 8, 5, 2, 7, 4, 1]))
```

1. (0.5 points.) According to Python's documentation, what's the running time (on average) of `sorted` and, in turn, `issorted`? Why?

2. (0.5 points.) Complete the reimplementation of `issorted` below in such a way that its running time is in *O*(*n*). If its input, a `list` of numbers, each of which you may assume is a unique `int`, is sorted from smallest to largest, your implementation should return `True`; if its input is not sorted so, your implementation should return `False`.

   ```
   def issorted(numbers):
       # TODO
   ```

3. (0.5 points.) Why can the running time of `issorted`, however implemented, not be in *O*(1)?

4. (0.5 points.) In the worst case, what might the running time of `sort` itself be? Why?

Consider the re-implementation of `sort`, below, which imports a module, `itertools`, and calls `itertools.permutations` therein. As before, assume that `numbers` could be any `list` of size *n*. And assume that this implementation of `sort` calls an implementation of `issorted` in *O*(*n*).

```
def sort(numbers):
    import itertools
    for permutation in itertools.permutations(numbers):
        if issorted(permutation):
```

```
            return permutation
```

5. (0.5 points.) What's a lower bound on the running time of this implementation of `sort`? Why?
6. (0.5 points.) What's an upper bound on the running time of this implementation of `sort`? Why?

Cracking Passwords

Suppose that a hacker has stolen a company's SQLite database, inside of which is a table with hashes of users' passwords, including your own. And suppose that each password has been hashed with `generate_password_hash` and can be checked against a non-hashed password with `check_password_hash`, as in Problem Set 9.

1. (1.5 points.) Suppose that your password (foolishly!) happens to be an English word and that the hacker knows that. Assuming the hacker has a dictionary of English words, as from Problem Set 5, the hacker could try to "crack" (i.e., figure out) your password by writing a program that iterates over that dictionary, comparing each word therein against the stolen hash of your password, using `check_password_hash` until it returns `True`.
    - Write a program that implements this algorithm.
    - Assuming there are *n* English words in the dictionary, what's the running time of the adversary's algorithm, using big-O notation? In no more than three sentences, why?

2. (1.75 points.) Suppose that your password happens to be an English word to which you've concatenated an additional ASCII character, so that your password is no longer an English word, and that the hacker knows that but still has access to that same dictionary.
    - Write a program for this problem.
    - In no more than three sentences and/or pseudocode, how would the hacker's algorithm need to change, and what would be its new running time, using big-O notation? In no more than three sentences, why?

3. (1.75 points.) Suppose that your password happens to be two English words (separated by a space) and that the hacker knows that.
    - Write a program for this problem.
    - In no more than three sentences and/or pseudocode, how would the hacker's algorithm need to change, and what would be its new running time, using big-O notation? In no more than three sentences, why?

4. (1.75 points.) Suppose that your password happens to be a random sequence of *n* decimal digits and that the hacker knows that.
    - Write a program for this problem.
    - Using big-O notation, what would be the running time of an algorithm that tries to crack that password of yours via "brute force," by trying all possible sequences? In no more than three sentences, why?

Teetering on the Edge

Consider the teeter-totters below, otherwise known as seesaws.

Let's assume that four staff members (Analysia, Bernardo, Christie, and Diana) want to teeter (i.e., sit) on these totters, and let's further assume that Analysia weighs 30kg, Bernardo weighs 35kg, Christie weighs 25kg, and Diana weighs 30kg. Is it possible to arrange the staff in some way on the teeter-totter such that it would be balanced? The answer is yes: with Analysia and Diana on one side, and Bernardo and Christie on the other, each side of the teeter-totter would have 60kg of weight on it.

Now let's imagine we add a fifth staff member: Elijah, who weighs 50kg. Can the teeter-totter be balanced now? Again, yes. With Bernardo and Elijah on one side, and with Analysia, Christie, and Diana on the other, each side of the teeter-totter has 85kg of weight on it, even though the two sides don't have the same number of staff.

Equivalently, we can consider a list "balanceable" if it is possible to *permute* its elements and put an imaginary fulcrum between two elements such that the sum of elements on the left side of the fulcrum is equal to the sum of the elements on the right half of the fulcrum.

For example,

```
a = [9, 26, 2, 19]
```

is balanceable, because we could rearrange the elements of `a` to be `[9, 19, 2, 26]` and place the fulcrum between `a[1]` and `a[2]` and the sum of the elements on each side of the fulcrum would be `28`. (Note that other arrangements of the elements of `a` are possible as well.) Similarly,

```
b = [1, 2, 3, 9, 3]
```

is also balanceable, because we could, for instance, rearrange the elements of `b` to be `[9, 3, 3, 1, 2]` and place the fulcrum between `b[0]` and `b[1]`, in which case the sum of the elements on each side of the fulcrum would be `9`. Note that the list is balanceable even though the number of elements on each side of the fulcrum is different, with one element on the left and four elements on the right.

By contrast, note that

```
c = [19, 8, 12]
```

is not balanceable. There is no way to permute the elements of `c` and place a fulcrum between any two of them and have the list be balanced.

For each of the below questions, you may assume that all elements of input lists will be nonnegative integers.

1. (0.25 point.) Is the list `[5, 5]` balanceable?

2. (0.25 point.) Is the list `[1, 0, 1, 0, 1, 0]` balanceable?

3. (0.25 point.) Is the list `[3, 4, 5, 6, 8]` balanceable?

4. (0.75 points.) Identify at least two potential features or properties of a list of items that would allow you to relatively quickly determine that the list is **not** balanceable.

5. (2.25 points.) Implement a function called `balanceable` that accepts one parameter: a `list` of integers called `numbers`. The function should return `True` if `numbers` is balanceable and `False` otherwise. You may assume that `len(numbers)` is positive and greater than or equal to 2. You may write additional helper functions if you wish, but your `balanceable` function must be prototyped as described above.