1. We're assigning the value 0 to the variable x. So in line 2, x+1 is equivalent to 0+1, which is 1. The "gets" symbol is not equivalent to =. Wherein x=x+1 is for equivalence in mathematics.

2. If we do the following, we get an error

```
> 1<-0
Error in 1 <- 0 : invalid (do_set) left-hand side to
assignment.
```

   While the backtick has been correctly used (using no backtick results in an error, as shown above), the second line is not the correct way to get the value assigned to the variable name '1'. You have to type in `1` to get the return value of 0. When typing in the number 1, it will return the value of that object, which is 1, not 0. Also, in R, naming convention dictates that you can only assign names starting with '.' or a letter, which is why the backtick must be used.

3. The assignments to x and y create a vector of [True, False] and [0 0] respectively. When we use "&" with the two variables, it is a logical expression, which will return the Boolean, either True or False. In line 3, it first compares TRUE and 0, and returns FALSE, since the number 0 is coerced to being FALSE, and only TRUE and TRUE would evaluate to TRUE. Then the second elements are compared, FALSE and 0 (FALSE and FALSE), which also evaluates to FALSE. Hence, we get FALSE FALSE as a final result.
   While using "&" compares each element in a vector to another and returns a logical vector, True or False, the "&&" only compares the first element of the vectors and returns a single logical value. As discussed earlier, the first elements, when compared, return FALSE, and that is the answer for the comparison in line 5.

4. **Line 1**: `> x= y=1`
   x and y are assigned the value 1. The equal reads from right to left. First, y is assigned the value 1, then x is assigned the value of y, which is 1.

   Line 2: `> x <- y = 1`

```
Error in (x <- y) = 1 : could not find function "<-<-"
```

   The expression is read left to right so x gets y which has some value but R gets confused because it can't assign y's value to the value 1. After the assignment operator <-, R is expecting to see only a single value. R is confused with (x<-y)

as an object, to which we are trying to assign 1. To fix this error, you could put parenthesis around (y=1).

Line 3:
```
> print(x <- y <- 1)
      [1] 1
```

The gets function '<-' can be used regardless of what environment you're in, so it retains its functionality inside a function as a way to assign an object to a value. We are first assigning the value 1 y, and then the same value to x. Then we return that value.

Line 4:
```
> print(x = y = 1)
Error: unexpected '=' in "print(x = y ="
```

Within a function, the "=" symbol is only used to name an argument, not to assign a value. Here, we are trying to assign the value 1 to y and x inside a function with an "=" operator, which is why the error appears.

5. **Line 1:**
```
f <- function(a = 1, b) a + b
```

The function takes in arguments a and b, where the default value of a is 1. We can name this function as the object "f." So now we can call the function, "f."

Line 2:
```
> f(b = 3, 2)
   [1] 5
```

R runs through exact matching first, so the value 3 is matched with argument b. It then takes the actual argument 2 and performs positional matching. In this case, after the exact match is found, there is only one other argument left, so 2 is matched with a. It gives a+b = 5. There will be no error in line 2.

Line 3:
```
> f(3)
   Error in f(3) : argument "b" is missing, with no
default
```

When you call f(3), R runs through exact matching but doesn't have anything to exact match, then it goes to positional matching, so it assigns 3 to "a." Because there isn't a second argument, R doesn't know how to evaluate the function—b is not specified. We get an error.

Line 4:
```
> f(b = 3)
       [1] 4
```

R is able to do an exact match, since we specify that b should get the value 3. Even though we are not entering an argument for a, R knows to use the default value of 1, which is given in the function. And so the calculation becomes 1 + 3 =4.

6. (a)

(i) specifying the arguments by position.

```
> qnorm(0.90, 100, 10)
[1] 112.8155
```

(ii) Specifying the arguments by complete names.

```
> qnorm(p = 0.90, mean=100, sd=10)
[1] 112.8155
```

(b) Using pnorm

(i) Without use of optional argument lower.tail

```
> 1-pnorm(q = 90, mean = 100, sd = 10)
[1] 0.8413447
```

(ii) With the use of optional argument lower.tail

```
> pnorm(q = 90, mean = 100, sd = 10, lower.tail = FALSE)
[1] 0.8413447
```