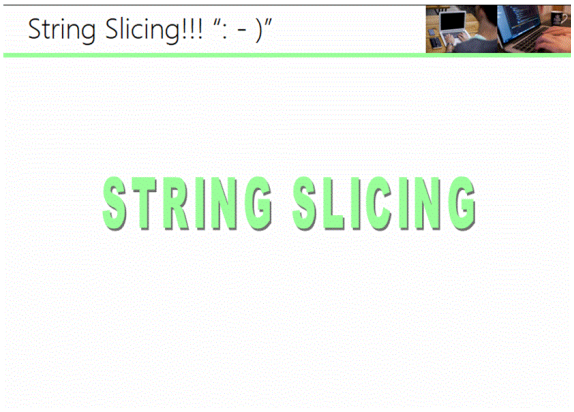
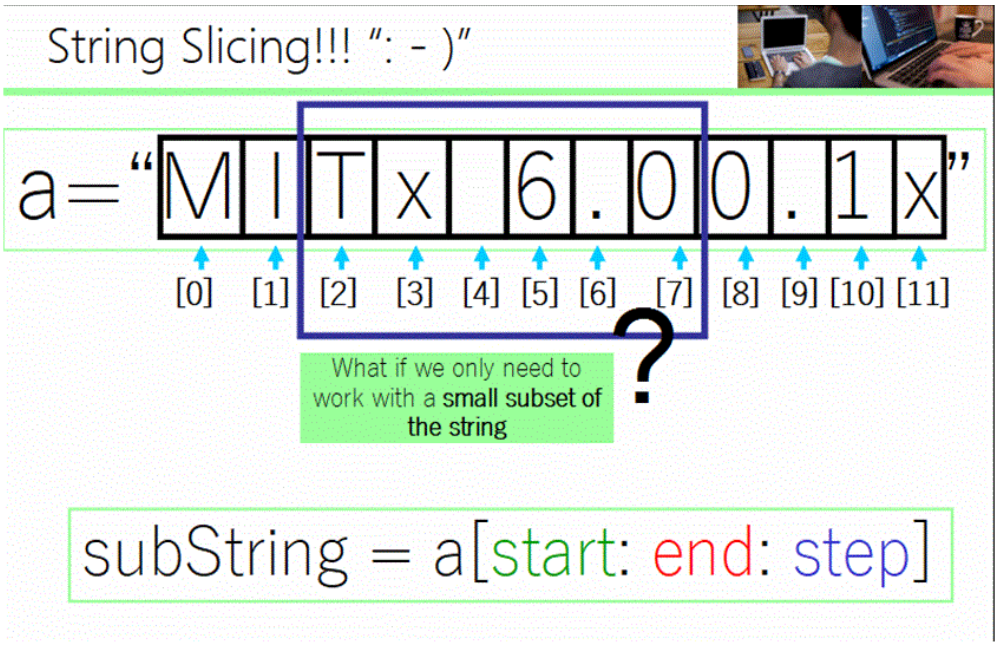


In the slide below you can see the syntax required to extract a string slice.

•First, we assign the slice to a variable (this is not necessary, but if you may want to use it subsequent steps in your code, you must store it in a variable)
•Then, we write the name of the variable that holds the original string. In this case, a.
•And we see a new syntax we haven't discussed. `[start: end: step]`

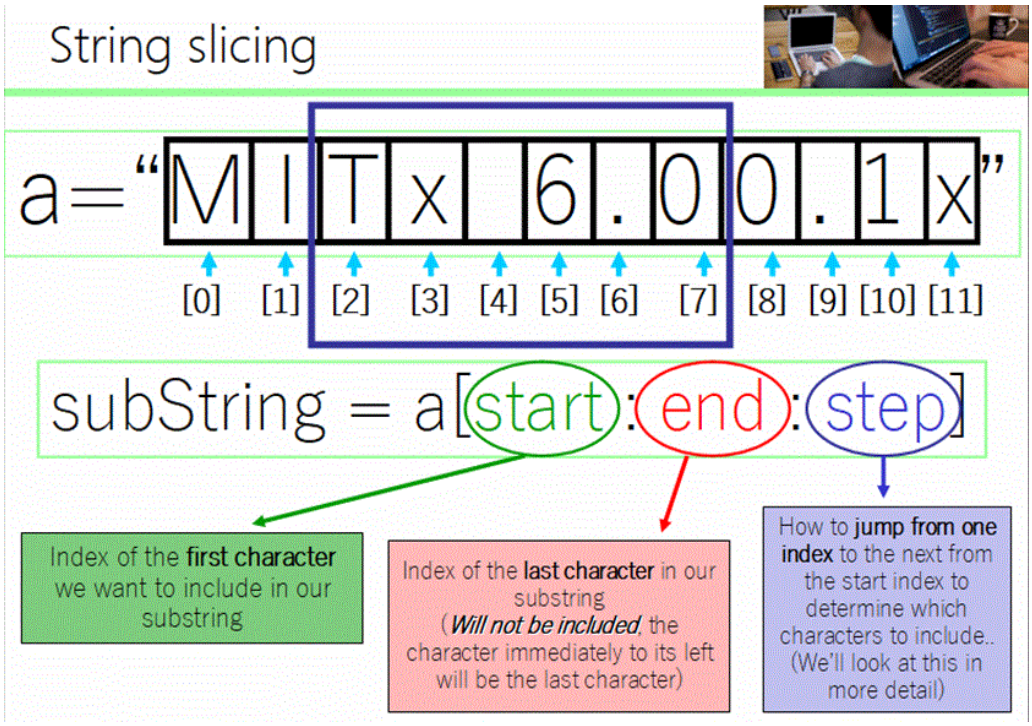


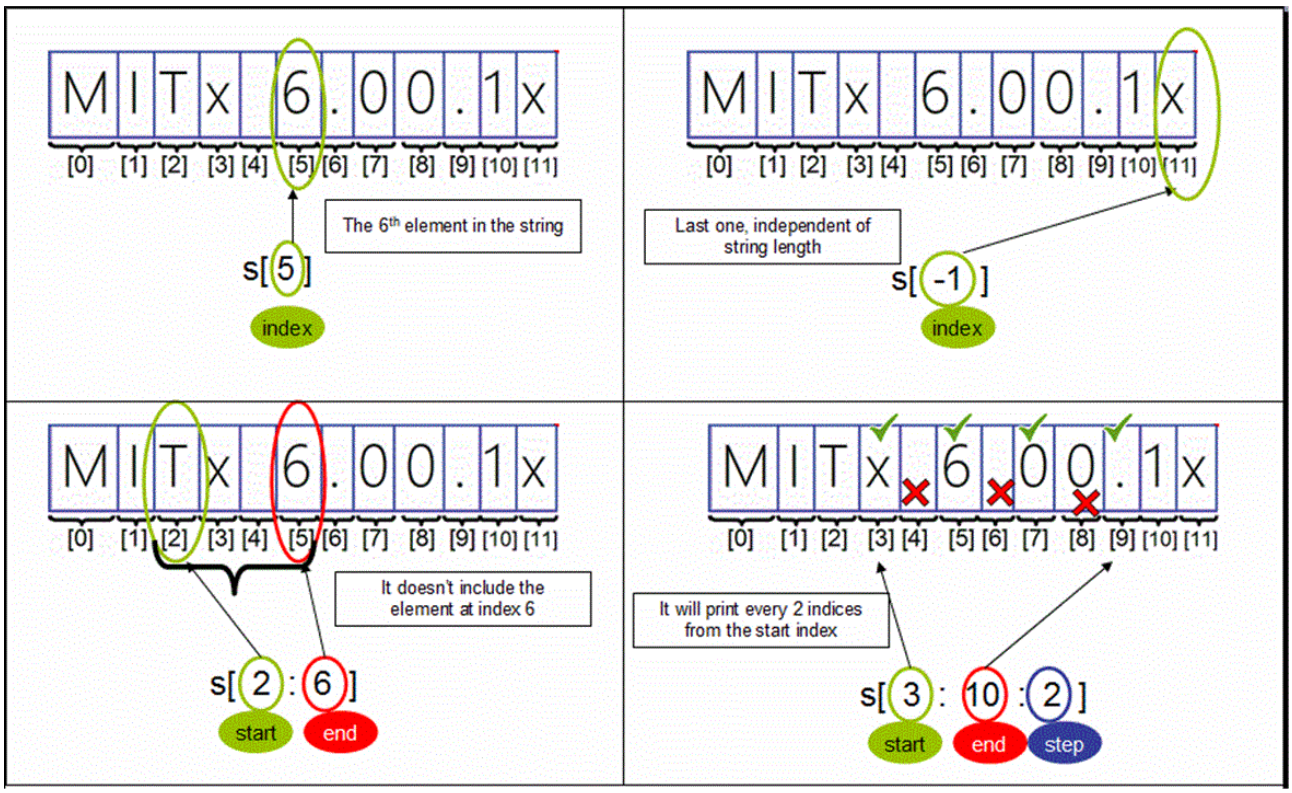
Let's look at different alternatives for the parameters inside [ ]:

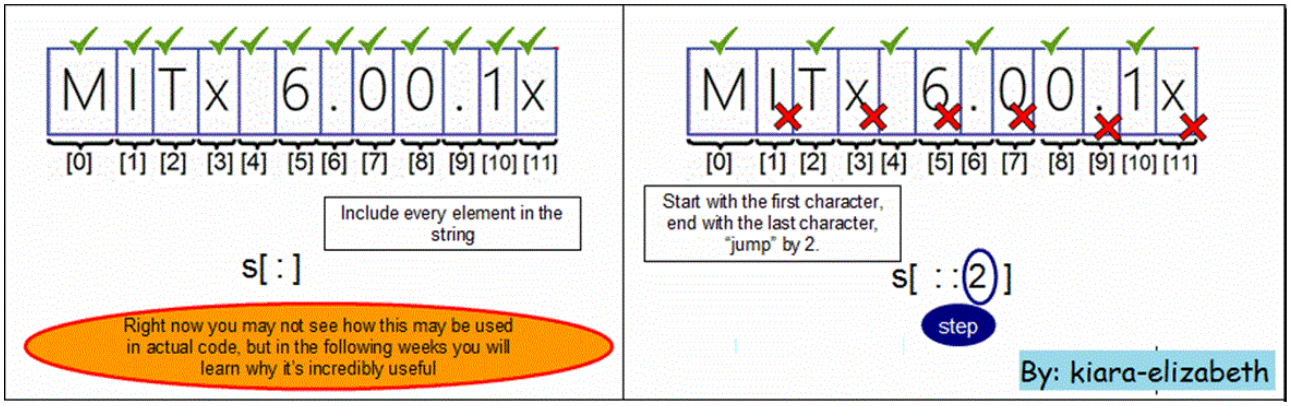
•If you just set one number inside the [ ] e.g 'hi'[1] that number will correspond to the index of a character in the grid, and you will get the single character located at that index.
•If you set a single number but you put a colon (:) before the number, like this: [: (number)] e.g [:5] you will get a slice of the string that starts from index 0 (this is the

default value for the starting index) up to but not including the character located at the index you set as the final index.
•If you set two numbers separated by a colon, like this [(number1) : (number2)] e.g [3:5] you will get a slice of the string that starts from 3 and contains all the characters up to but not including the character located at the end index.
•If you set three numbers separated by colons, like this [(number1) : (number2) : (number3)] e.g [1:8:2] you will get a slice that starts from the character located at index 1 up to but not including the character located at index 8. And the third parameter determines that characters every two indices will be included. It will jump from index 1 to index 3 and from index 3 to index 5 and so on, without including characters located in between.
•If you set -1 as the only parameter inside the [ ] , you will get the last character in the string, independent of its length.
•If you set a single number inside [ ] followed by a colon (:) like this: [(number1) :] e.g [2:] it will give you a slice of the string that starts at the index you set as the start, and will include every character up to the end of the string.
•If you include two colons (:) followed by a number, like this: [::(number1)] that number will be the step ("jump" from one index to the next) e.g [::2].
•If you only set a single colon (:) inside the [ ], you will get a "copy" of the string. In the following weeks, you will learn why this is an important feature and how it can help you in actual code.

Here are some examples of how string slicing works behind the scenes.



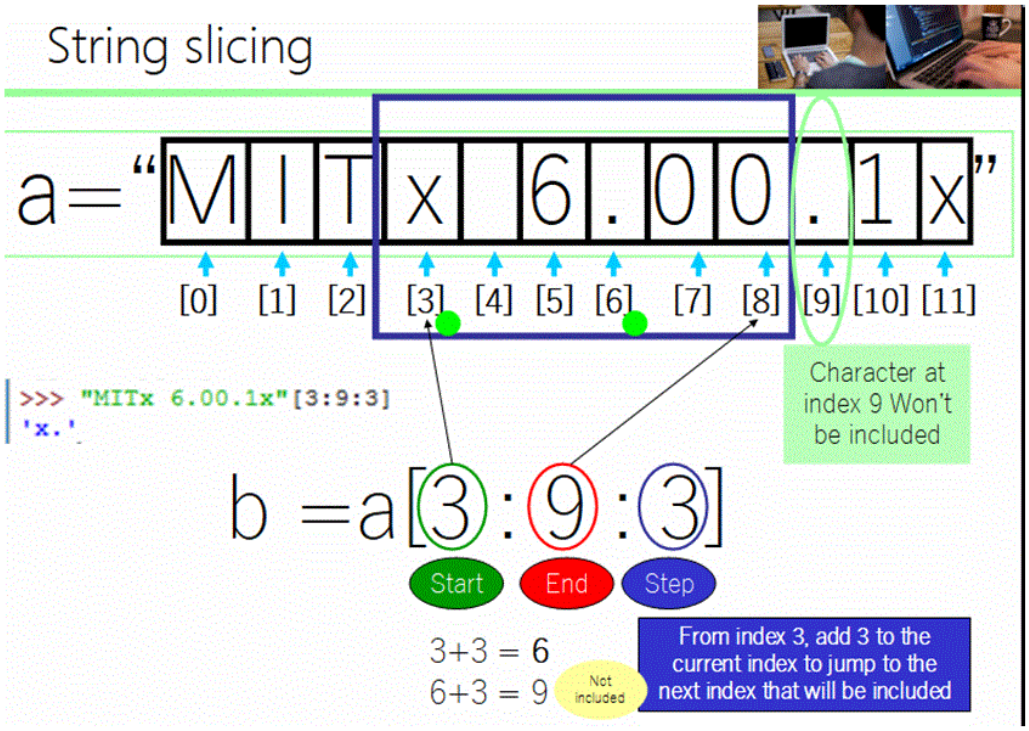




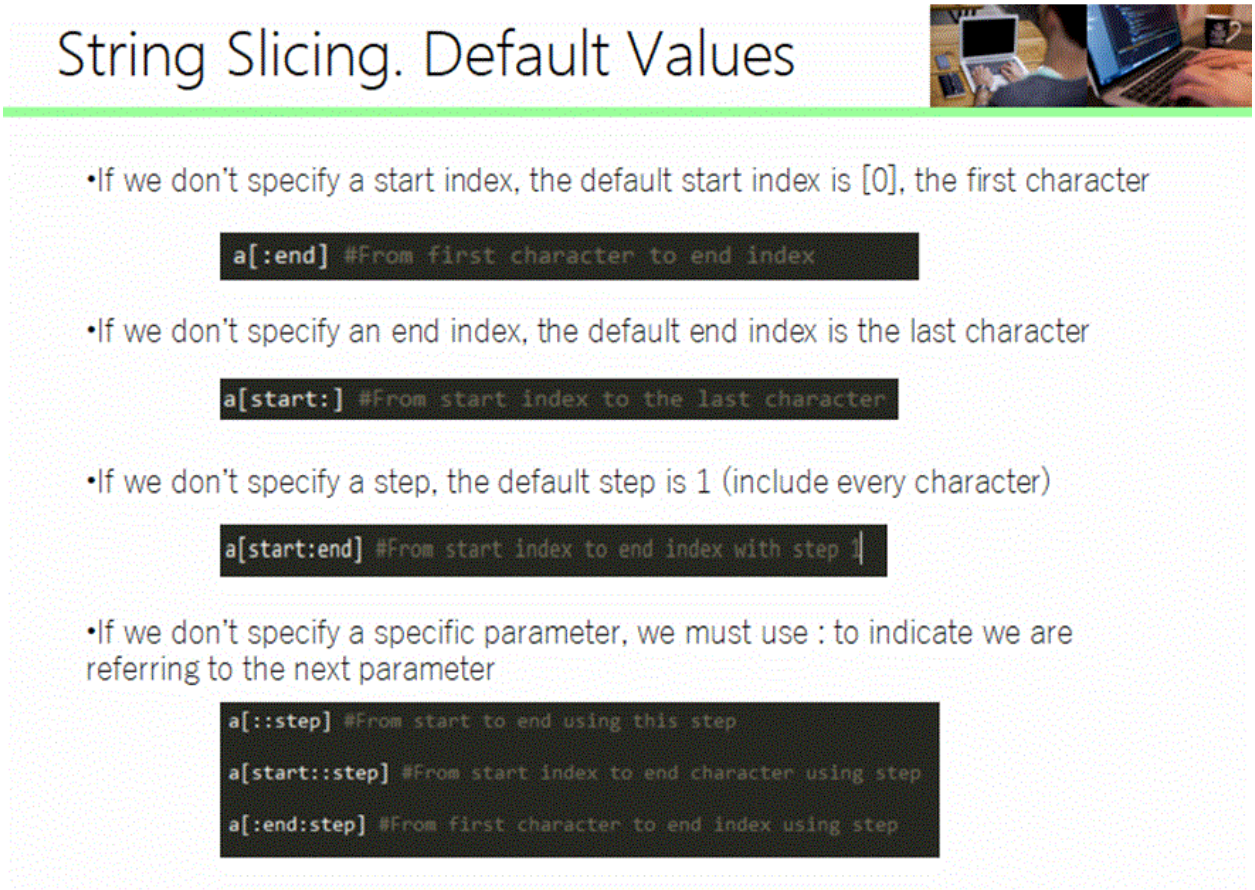
Here you can see the result of these examples executed in python's shell:

```
>>> s = 'MITx 6.00.1x'
>>> s[5]
'6'
>>> s[-1]
'x'
>>> s[2:6]
'Tx 6'
>>> s[3:10:2]
'x60.'
>>> s[::2]
'MT .01'
>>> s[:]
'MITx 6.00.1x'
```

Here we have an example of a substring:



Now that we've seen how String Slicing works, we have analyze alternative syntax for its parameters:



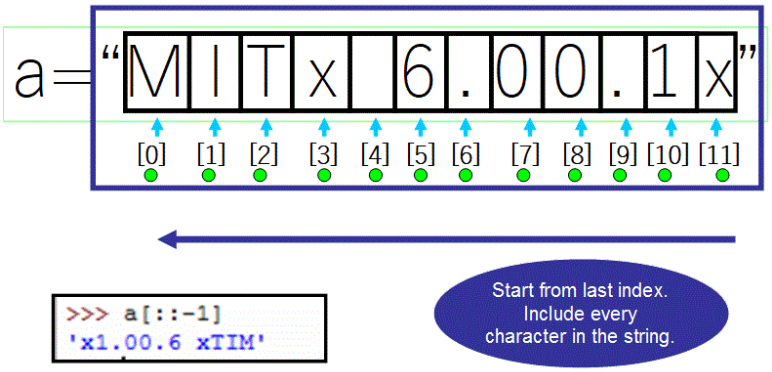
What happens when a step includes a negative number?

- When we use negative steps we are saying that we want out slice to start from the right and subsequently include item at indices further to the left by moving as many indices as we indicate on the step.

- This is better explained with a few examples.

Example 1:

In this first example, we use the `a[::-1]` syntax, which means "I want the whole string, but evaluated from right to left" (in other words, reversed). We start from the last index and include items that result from subtracting 1 to the last index.



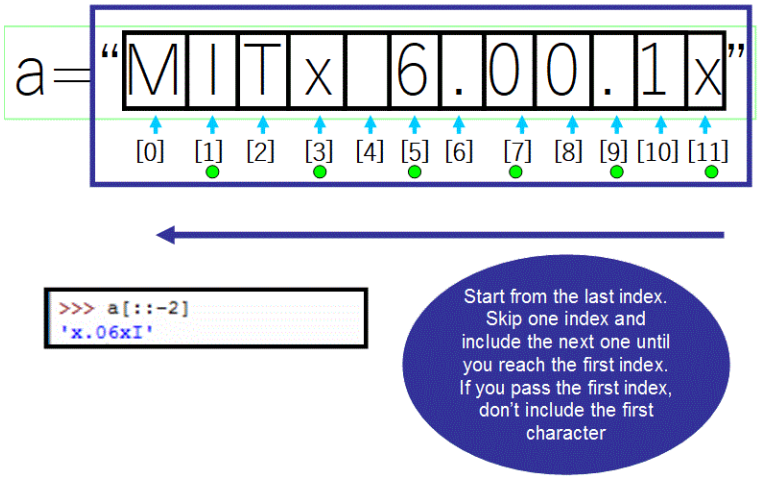
We first include the character at index 11, then the character at index 10 since we've subtracted 1, then the character at index 9 since we've subtracted one and so on, until we reach index 0 and we have included every character in the string.

Example 2:

In this example, we have a step of -2. We will take the entire string and start counting from the last index (in this case, 11).

We include the character at index 11, then we subtract 2 to reach index 9, we include the "."

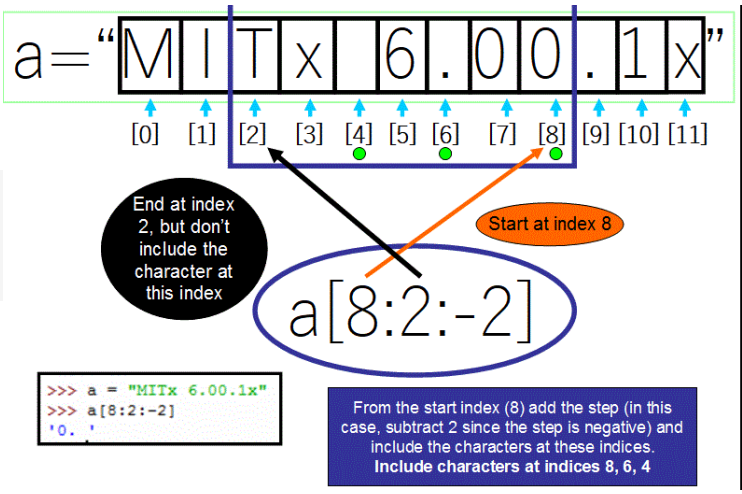
We subtract 2 again and reach index 7, we include the character "0" and so on until we reach index 0 or we pass index 0 by subtracting 2, in this case we don't include the first character (first index, index 0).



Example 3:

This example is a bit more elaborate, it illustrates how the different parameters interact to get a slice using a negative step.

When we use a negative parameter we must set a start index that is further to the right than the end index since we will be evaluating the string from right to left.



In this case we have `a[8:2:-2]`. We will start from index 8, include that character and then subtract 2 to index 8, which gives us index 6, include that character and so on until we reach index 2, but we will **NOT include the character at index 2** since the clause of "the character at the end index is NOT included" still remains true.

==========

QUESTION: To me, [::-1] translates to [start:end:-1] which should mean you start at position 0 and go backwards and there is nothing there, so it should be an error. This would comport with [8:2:-1], where you start on 8 and go backwards by -1.

But I suppose [::] just means "the whole thing" and doesn't fill in start and end in the same way?

So, if you were to do this with a string with a len of 11:

[0:10:-1]

it would not be the same as

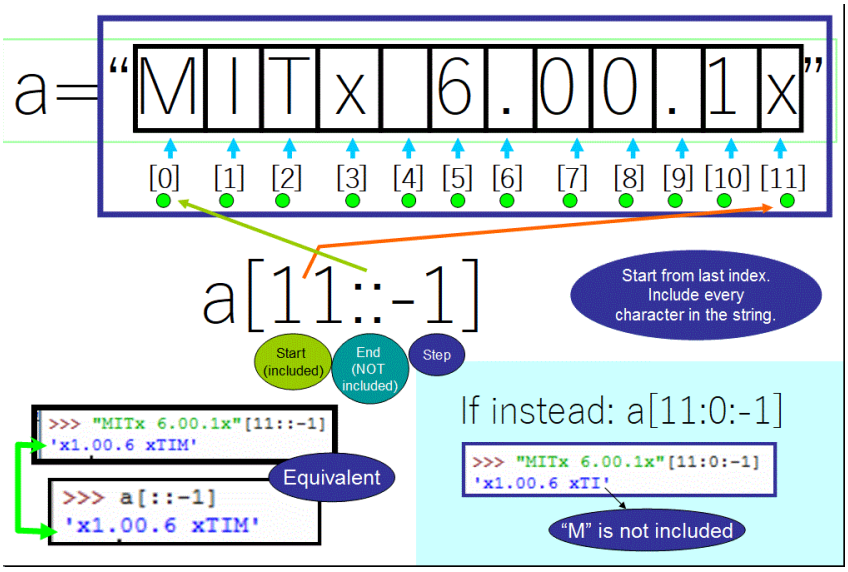
[::-1]

which would in effect be doing:

[10:0:-1]

Is that correct?

Answer: Let's find an equivalent way to write a[::-1] using start and end parameters!



Let's try a[11:0:-1] This means that we start from the last index (because in this case, the string's last index is 11) and we stop at index 0. We use a step of -1, so we will include characters from right to left.

But something very curious happens when using this syntax, the fact that the character at the end index is not included still holds, therefore, defining the end index to be 0 means that we will not include the character at index 0. This discards this option since it is not equivalent to [::-1] which returns the entire string.

•Now let's see how we can include the character at index 0. If we set 0 to be the end index, it will not be included, therefore we must use the default index by leaving the space blank.

```
a[11::-1]
```

Using this syntax we get a string that is equivalent to a[::-1] because we are starting to include characters from the last index and we move towards the left one by one until we reach the character at index 0, that in this case, since we haven't defined an end index, is included.

You can check in the diagram above that both expressions are equivalent, they produce the same string.

These are the examples you proposed executed in Python's shell. I've adapter the string so its length is 11 as you proposed.
•First example: you can see how starting from 0 with a negative step returns an empty string. (You can check the fourth example, which returns an empty string as well) (Experimenting with this syntax I've noticed that whenever you slice a string with a negative step and the start index is further to the left than the end index, you get an empty string).

•Second example: The string reversed.
•Third example: we have the same problem as the example above, the character at the first index is not included.

```
>>> "MITx 6.00.1"[0:10:-1]
''
>>> "MITx 6.00.1"[::-1]
'1.00.6 xTIM'
>>> "MITx 6.00.1"[10:0:-1]
'1.00.6 xTI'
```

```
>>> "MITx 6.00.1x"[5:8:-1]
''
```

This example illustrates that it is possible to define a start index of 0 with a negative step and still have a character included. But in practice, you shouldn't need to use slicing, you would just access the character by its index.
•Since we are starting from 0 and we haven't defined an end index, but we're using a step of -1, we include only the first character in the string. This is because in this case, not setting an end index gives us the default start index, which is the start of the string.

```
>>> "HI"[0::-1]
'H'
```