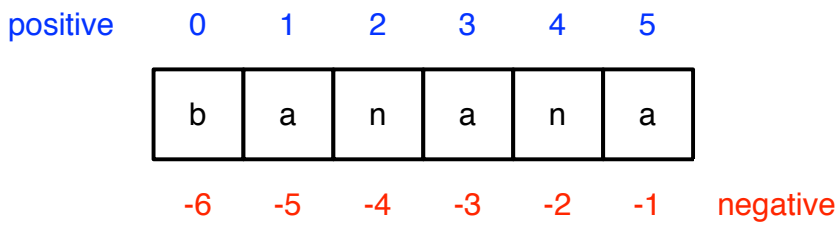
1 Len function

The `len` function takes one argument, a sequence such as a string or the return value of `range`, and returns its length. Examples:

```
>>> len("abc")
3
>>> s = "wxyz"
>>> len(s)
4
>>> len(range(13))
13
```

2 String indexing

A string is a sequence of characters. An **index** is a position within the string. **Positive indices** count from the left-hand side with the first character at index 0, the second at index 1, and so on. **Negative indices** count from the right-hand side with the last character at index -1, the second last at index -2, and so on. For the string `"banana"`, the indices are:



You can access individual characters using the index operator like this:

```
>>> s = "banana"
>>> s[0]
'b'
>>> s[-6]
'b'
```

Between the brackets, you can put any **expression** that evaluates to an `int`. This follows the general principle: anywhere you can put a data value, you can put an expression (since python will evaluate the expression and the result will be a data value).

```
>>> idx = 1
>>> s[idx]            # expression consists of a variable name
'a'
>>> s[2 ** 0 + 3]     # arithmetic expression
'n'
>>> s[min(4, 2 ** 0)] # expression consists of a function call
'a'
>>> s[len(s) - 1]     # expression has both function call and arithmetic
'a'
```

3 Strings are immutable

A string is an immutable object. If you try to modify one character of the string using the index operator and an assignment statement, you get an error:

```
>>> s = "banana"
>>> s[1] = 'A'
Traceback (most recent call last):
  File '<stdin>', line 1, in <module>
TypeError: 'str' object does not support item assignment
```

While we cannot change string objects, we can change the object to which the variable refers. This statement `s = s + '!'` does not change the string, but rather changes to which string `s` refers.

4 Looping over indexes

We have seen two kinds of **for** loops: loops over characters in strings and loops over numbers (using the `range` function). With string indexing, we can loop over numbers but use these numbers to index into the characters of a string. This capabitility comes in handy (see exercises below).

```
s = 'banana!'
for idx in range(len(s)):
        print s[idx],
```

This will print the letters in string `s` with spaces in between, as in `'b a n a n a !'`.

5 Search problems

As was discussed on the last handout, **search** problems are a common problem that arises naturally in many contexts. The basic idea is that we are given a sequence (say, a string of characters typed by the user) and we want find out whether a particular pattern occurs in that sequence.

1. Write a program that asks the user for a phrase and then reports the *index* where the letter "x" occurs in the phrase, or else report it was not found. If "x" occurs more than once, any index is an acceptable answer. Make the program case insensitive and search for either "x" or "X."
2. Same idea, but find the index of the *first* occurrence of "x". Again, be insensitive to case.
3. Now find the index of the *second* occurrence of "x" else report "x" occurs 1 or fewer times. Again, be insensitive to case.
4. Write a similar program but report the index of the two letter substring "ox" or else report that "ox" does not occur. For example, given input "the fox" it should report 5. Unlike the previous ones, make this program case sensitive.