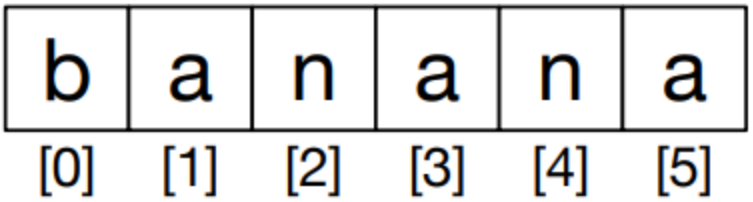
C6.1-2, C6.4-5, C6.8-9

6.1: A String is a sequence:

- A string is a sequence of characters. You can access the characters one at a time with the bracket operator:

```
>>> letter = fruit[0]
>>> print(letter)
b
```

- The second statement extracts the character at index position 1 from the fruit variable and assigns it to the ***letter*** variable. The expression in brackets is called an *index*. The index indicates which character in the sequence you want (hence the name).
- But in Python, the index is an offset from the beginning of the string, and the offset of the first letter is zero.



6.2: Getting the length of a string using len:

- len is a built-in function that returns the number of characters in a string

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

- To get the last letter of a string. To get the last character, you have to subtract 1 from length:

```
>>> last = fruit[length-1]
>>> print(last)
a
```

6.4 String slices:

- A segment of a string is called a *slice*. Selecting a slice is similar to selecting a character

- The operator [n:m] returns the part of the string from the “n-th” character to the “m-th” character, including the first but excluding the last.

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

- If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks:
- An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

6.5 Strings are immutable:

- The “object” in this case is the string and the “item” is the character you tried to assign. For now, an object is the same thing as a value, but we will refine that definition later. An item is one of the values in a sequence.

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print(new_greeting)
Jello, world!
```

6.8 String comparison:

- The comparison operators work on strings. To see if two strings are equal:

```
if word == 'banana':
    print('All right, bananas.')
```

^.9:

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
[... 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

- `>>> help(str.capitalize)`
- the dir function lists the methods,
- We call a method by appending the method name to the variable name using the period as a delimiter.

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print(new_word)
BANANA
```

-
- A method call is called an invocation; in this case, we would say that we are invoking upper on the word.
- For example, there is a string method named find that searches for the position of one string within another:

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print(index)
```

- `1`
- Line Strip

```
>>> line = '  Here we go  '
>>> line.strip()
'Here we go'
```

-
- Startswith returns boolean values

```
>>> line = 'Have a nice day'
>>> line.startswith('h')
False
>>> line.lower()
'have a nice day'
>>> line.lower().startswith('h')
True
```