String Manipulation - Indexing and Slicing
Goals

• Know how to **index** into a string

• Know how Python interprets **negative indices** into strings.

• Know how to use **slicing** to get **substrings**
Indexing into Strings

Strings are collections of individual characters. We can get access to an individual character by **index**.

```
outlook = "Winter is coming"
```

How is this stored in memory?

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Winter</td>
</tr>
<tr>
<td>1</td>
<td>i</td>
</tr>
<tr>
<td>2</td>
<td>n</td>
</tr>
<tr>
<td>3</td>
<td>t</td>
</tr>
<tr>
<td>4</td>
<td>e</td>
</tr>
<tr>
<td>5</td>
<td>is</td>
</tr>
<tr>
<td>6</td>
<td>coming</td>
</tr>
</tbody>
</table>

Syntax:

```
outlook[0] # => "W"
outlook[4] # => "e"
```

Indices in Python begin at 0.

Spaces are characters too!

```
outlook[6] # => " 
```

(just smaller strings!)
Indexing gives us other ways to loop through strings:

```python
for letter in a_string:
    print(letter, end=" ")
```

is equivalent to

```python
for i in range(len(a_string)):
    print(a_string[i], end=" ")
```

and also

```python
i = 0
while i < len(a_string):
    print(a_string[i], end=" ")
    i += 1
```
Nifty Python Feature: Negative Indices

Negative indices count backwards from len(s):

<table>
<thead>
<tr>
<th>Index:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Winter is coming</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Also Index:</td>
<td>-16</td>
<td>-15</td>
<td>-14</td>
<td>-13</td>
<td>-12</td>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Two possible ways to remember how this works:

-1 is always the last character, and indices count backwards from there.

```
a_string[-5] is equivalent to a_string[len(a_string)-5]
```
Slicing: indexing substrings

```ruby
alph = "abcdefg hij"
alph[0]  # => "a"
alph[4]  # => "e"
```

What if I want to "index" more than one character at a time?

```ruby
alph[???]  # => "cdef"
```
Slicing: indexing substrings

alph = "abcdefg hij"
alph[0] # => "a"
alph[4] # => "e"

index of first character 1 + index of last character

Slicing syntax: string[start:end]

alph[2:6] # => "cdef"
alph[0:10] # => "abcdefg hij"
alph[5:-2] # => "fgh"

just like the range function: the end index is not included

not like the range function: negative indices don't make empty substrings
Slicing: indexing substrings

alph = "abcdefghij"

<table>
<thead>
<tr>
<th>Ind</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
</tr>
</tbody>
</table>

index of first character \[1\] + index of last character \[9\]

**Slicing syntax:** \[\text{string}[\text{start}:	ext{end}]\]

If omitted, \text{start} defaults to 0
If omitted, \text{end} defaults to \text{len(string)}

alph[0:4] # => "abcd"
alph[5:] # => "fghij"
String Slicing: Demo
String Slicing: Demo

- \( s = "\text{fibonacci}" \)
- Positive indices: \( s[1:3] \)
- Negative indices!? \( s[-4:9] \)
- Leaving out start/endpoint: \( s[:6], s[4:] \)
- Indices past the end in a slice: \( s[1:21] \)
- Single indices past the end: \( s[9], s[21] \)
- Loop over a slice of a string
  ```python
  for c in s[2:6]:
      print(c, "!", sep=" ", end=" ")
  ```