More on Sequence Types
Operations that work on strings and lists

1. \( x \text{ in } s, x \text{ not in } s \): Membership operations

2. \( s + t, s^n, n^s \): Concatenate operations.

3. \( s[i], s[i:j], s[i:j:k] \): Operations for accessing parts of strings and lists.
Examples: evaluate these expressions

1. "l"**2 in "hello"[:3]
2. "l"**2 in "hello"[2:]
3. ["How", "are", "you"][1][:1]
4. (range(1, 5, 3)*2)[2:3]
5. (range(1, 5, 3)*2)[2:3]*5
6. range(10) in range(20)
7. range(10) in [range(10), range(10)]
8. range(20)[3:12:2]
9. "w" in "Iowa" and (5!=4*3-7 or "k" not in "Hawk")
10. "easy" in ("yes we ease"**2)
Operator precedence including these new operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(...)</td>
<td>Function call</td>
</tr>
<tr>
<td>s[...]</td>
<td>Indexing into a sequence</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>-E</td>
<td>Change sign</td>
</tr>
<tr>
<td>*, /, %, //</td>
<td>Multiplication, division</td>
</tr>
<tr>
<td>+, -</td>
<td>Addition, subtraction</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=, !=, ==</td>
<td>Comparisons</td>
</tr>
<tr>
<td>in, not in</td>
<td>Membership</td>
</tr>
<tr>
<td>not E</td>
<td>Logical negation</td>
</tr>
<tr>
<td>and</td>
<td>Logical conjunction</td>
</tr>
<tr>
<td>or</td>
<td>Logical disjunction</td>
</tr>
</tbody>
</table>
Built-in Functions on lists and strings

1. \texttt{len(s)}: returns the length of sequence \( s \)
2. \texttt{min(s)}, \texttt{max(s)}: return the smallest (largest) element in \( s \).
3. \texttt{sum(s)}: returns the sum of the elements in \( s \).
4. \texttt{all(s)}: returns True if all elements in \( s \) are True; False otherwise.
5. \texttt{any(s)}: returns True if any element in \( s \) is True; False otherwise.
The min and max functions

- \( \min(s) \) (\( \max(s) \)) is the smallest (largest) element in \( s \)
  - If \( s \) is a list of numbers (integers, longs, and floats) these functions return the smallest (largest) number
  - If \( s \) is a list of strings, these functions return the lexicographically smallest (largest) string
  - If \( s \) is a string, these functions return the lexicographically smallest (largest) character in the string
  - If \( s \) is a list that contains a mixture of numeric and non-numeric objects, then the result is not specified by the language and you should not rely on such a result.
Examples

- \text{max}("hyperbole", "hyena", "hypotenuse")
  Strings are ordered in lexicographic or “telephone book” order.

- \text{min}("charming!")
  There is a standard encoding of characters used by computers called the \textit{American Standard Code for Information Interchange} (ASCII). Characters are ordered according to this encoding.
The “search” methods

- `s.index(e)` returns the index of the first occurrence of `e` in `s`
- `s.count(e)` returns the number of occurrences of `e` in `s`

```python
>>> L = [1, 3, 6] * 4
>>> L
[1, 3, 6, 1, 3, 6, 1, 3, 6, 1, 3, 6]
>>> L.index(3)
1
>>> L.count(3)
4
>>> L.index(0)
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
ValueError: 0 is not in list
>>> L.count(0)
0
```
Methods versus Functions

• Notice the new syntax. This reflects the fact that `index` and `count` are methods and not functions.
• There are some fundamental differences behind the scenes between methods and functions.
• The differences you should focus on for now are:
  ○ A method call (e.g., `L.index(3)`) is always applied on to an object (`L`, in this example).
  ○ The syntax of a method call is `object.methodName(argument list)`
  ○ The method has access to the object it is being applied on to and the arguments it is being sent.
Problem: Selection Sort

- **Sorting** is a fundamental algorithmic problem in computer science.
- The sorting problem asks that we rearrange elements in a list so that they are in ascending or descending order.
- There are many known algorithms for sorting: insertion sort, selection sort, bubble sort, quick sort, merge sort, heap sort, shell sort, radix sort, etc.
- Using the operations and functions we have just learned about, let us implement *selection sort*. 