Last Lecture

Python’s building blocks:

- **Python statements**
  - Categories of Statements
    4. Operational statements
      - Mathematical/arithmetic operators
        - Augmented assignment operators
      - String manipulation operators
      - Function terminology
  - **Python strings**
Learning outcomes

At the end of this course, a student is expected to:

- Create (design), analyze, and explain the behaviour of simple algorithms:
  - Solve problems by designing simple algorithms, e.g., basic calculations, searching in strings and lists, etc…
- Create (design) small to medium size programs using Python:
  - Create programs that search or construct lists and strings
  - Create programs that modify lists in-place
Today’s Menu

• Introduce Python Lists
Another data type in Python: Lists

- First, let’s have a look at some examples and then, let’s come up with a definition:

```python
prices = [1.20, 0.75, 4.50]
names = ["Mike", "Xinghua", "Lise"]
somePrimes = [1, 3, 5, 7, 11, 13]
underTheBed = [3, "old socks"]
```
Lists

- **Definition:**
  - Sequence of elements
  - Versatile data type as it can contain elements of different data types
  - Dynamic length (lists are mutable)

1. How to create a list variable?
   **Answer:**

2. How to access a whole list?
   **Answer:**
3. How to access one list element at a time?

**Answer:** Use the index associated with the element as illustrated below:

```
positive indexing->  index:   0     1     2
```

**For example:** courses = ['phys', 'chem', 'math']

- So if we wish to access
  - The 1\text{st} element of the list, we use the index 0
  - The 2\text{nd} element of the list, we use the index 1
  - etc...
List indexing: positive indexing

- Examples:

When does the “IndexError: list index out of range” error occur?

Careful: Positive index starts at 0
List indexing: negative indexing

• There is another way we can use to access one list element at a time: negative indexing:

For example: courses = ['phys', 'chem', 'math']

negative indexing-\> index: -3 -2 -1

• So if we wish to access
  • The 1st element of the list, we use the index -3
  • The last element of the list, we use the index -1
  • etc...

Careful: Negative index starts at -1, not 0
List slicing (using positive indices)

4. How to access a section (slice) of a list at a time?
Answer: use indices to indicate the list slice

positive indexing→ index: 0 1 2

For example: courses = ['phys', 'chem', 'math']

• Index from 0..len-1 → like a string
• Syntax: <list>[start : stop : step]

• start
• stop
• end
List slicing (using positive indices)

- So if we wish to create the slice
  \[ ['\text{phys}', '\text{chem}'] \]
  from the list
  \[
  \text{courses} = ['\text{phys}', '\text{chem}', '\text{math}']
  \]
  we use courses[0:2]
  - We use index 0 to indicate the start of the list slice
    - \text{Inclusive} -> element at index 0 is included in the list slice
  - We use index 2 to indicate the end of the list slice
    - \text{Non-inclusive} -> element at index 2 is ***not*** included

- When we slice \textit{out of bound}, the result is an \textit{empty list}
Let’s try!

Remember: Just like String Slicing!!!
Lists are mutable!

• Definition of **mutable**: can change

• Can we modify a list?
  • For example, let’s try:
    ```python
courses[2] = 'stats'
```

• Here is what happens?
  ```python
>>> courses = ['phys', 'chem', 'math']
>>> courses
['phys', 'chem', 'math']
>>> courses[2] = 'stats'
>>> courses
['phys', 'chem', 'stats']
```
## Summary: List (sequence) manipulation

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Operator/ function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>concatenation</td>
<td>+</td>
<td>Combine lists together</td>
</tr>
<tr>
<td>repetition</td>
<td>*</td>
<td>Concatenate a list that is being repeated a number of times</td>
</tr>
<tr>
<td>indexing</td>
<td>[n]</td>
<td>Access an element of a list</td>
</tr>
<tr>
<td>slicing</td>
<td>[: : ]</td>
<td>Extract a part of a list</td>
</tr>
<tr>
<td>length</td>
<td>len(aList)</td>
<td>Determine the number of characters in a string aList</td>
</tr>
</tbody>
</table>
List built-in function

- `len()` is an example of a built-in function for list in Python
- Let's try it on `courses`:
  ```python
  >>> courses
  ['phys', 'chem', 'stats']
  >>> len(courses)
  3
  ```
- Notice that the valid range of the indices for the list `courses` is 0 to 2
- So ...
  - If the length of a list is $x$ then we know that the valid range of the indices for that list is 0 to $x-1$
  - If the valid range of the indices for a list is 0 to $y$ then we know that the length of that list is $y+1$
### Other useful list built-in functions

<table>
<thead>
<tr>
<th>Method name</th>
<th>Use</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum</td>
<td>\texttt{max(aList)}</td>
<td>Returns element from 	exttt{aList} with max value.</td>
</tr>
<tr>
<td>within a list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>minimum</td>
<td>\texttt{min(aList)}</td>
<td>Returns element from 	exttt{aList} with min value.</td>
</tr>
<tr>
<td>within a list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compare</td>
<td>\texttt{cmp(aList1, aList2)}</td>
<td>Compares elements of both lists.</td>
</tr>
<tr>
<td>create a list</td>
<td>\texttt{list()}</td>
<td>Creates an empty list (\texttt{length} = 0).</td>
</tr>
<tr>
<td></td>
<td>\texttt{list(&lt;sequence&gt;)}</td>
<td>Converts \texttt{&lt;sequence&gt;} into a list.</td>
</tr>
</tbody>
</table>

We can find more built-in functions at:

- [http://www.tutorialspoint.com/python/python_lists.htm](http://www.tutorialspoint.com/python/python_lists.htm)
- [https://docs.python.org/3.5/tutorial/datastructures.html](https://docs.python.org/3.5/tutorial/datastructures.html)
Methods

• Aside from **built-in functions**, Python also has **methods**, which are a specific type of functions that are related to classes
  • **Class** is a mechanism that allows us to create our own type of objects (variables)

• **Syntax for methods:**
  
  `<string> . <method> (<arguments>)`

  **dot notation**

  Arguments are not always needed. How do we know if they are? Answer: We must look at a description of the method.
List methods

- Methods are called using the **dot notation**, and are applied to a list that already exist.
- Let’s try this Python program:
  ```python
courses = ['phys', 'chem', 'math']
print('Courses: ', courses)
courses.append('stats')
print('Updates courses: ', courses)
```

The result is:
```python
>>> Courses:  ['phys', 'chem', 'math']
Updates courses:  ['phys', 'chem', 'math', 'stats']
```
List methods

• Here are some very useful list methods: count( ), insert( ), remove( ), etc.

• We can find more list methods at:
  http://www.tutorialspoint.com/python/python_lists.htm

• Even though we may not talk about all the Python list methods during our lectures, our task, as a Python software developer (i.e., CMPT 120 students), is to become familiar with Python list methods as they will be very useful to us in solving our problems/tasks throughout the semester.
Summary

• Lists
  • Indexing
  • Slicing
  • Mutable
  • Functions
  • Methods
Next Lecture

• Conditional Statements