CSC326H Programming Languages
(Week 8, Monday)

Yilan Gu
guyilan@ecf.toronto.edu
http://www.cs.toronto.edu/~yilan/326f09

Python ABC

• Running Python programs
• Variables
• Basic data types:
  – Numbers
  – Strings
  – Booleans
• Basic Input/Output
• If statement and while loop

Today
(Container: list, tuple, sequence, dictionary, set)
Containers and sequences

- A **container** is a collection of objects.
  - In Python, an object can be practically anything.
  - Properties of a container include:
    - **Size**
    - Whether or not a particular object is contained
    - Whether or not objects can be added or removed
- A **sequence** is a container with some kind of ordering.
  - A sequence can contain a particular object more than once.
  - Sequences can be indexed to obtain the N-th object in the sequence.
- In Python:
  - Sequence types are `string`, `list`, and `tuple`.
  - Non-sequence container types are `dictionary` and `set`.
- Reference: Sebesta, Chapter 6

Lists

- **More examples:**

  ```python
  >>> ls = [1, "CSC", 2, [], "321"]
  >>> i = 0
  >>> while i < len(ls):
  ...     ls[i] = i
  ...     i += 1
  ...  >>> ls
  [0, 1, 2, 3, 4]
  ```

Nested Lists

- **Nested lists:**

  ```python
  nest = [[1, 2], [3, 4], [5, 6]]
  nest[0]       # [1, 2]
  nest[0][1]    # 2
  y = nest[0]    # y = [1, 2]
  y[1] = 4
  nest[0][1]    # 4 (and not 2!)
  y = [99, 99]   # y = [99, 99]
  nest[0][1]    # 4 (and not 99!)
  ```

- Observe that values associated with Python variables are actually references! This should remind you of pointers in C or references in Java.
  - In the example above:
    - `nest` is a reference to a 3-element list
    - `nest[0]` is a reference to the first element of `nest`
    - `y=nest[0]` sets `y` to be a reference to the object that `nest[0]` refers to.
    - `y[1]=4` modifies the object that `y` refers to.
**Updating Lists**

- Use the `append` method to add to a list.
  ```python
  ls = [32]
  ls.append(4)
  ls         # [32,4]
  # Do not do the following:
  ls = ls.append(5)  # ls is now None
  ls2 = ['cat', 'dog']
  ls2.append(['rabbit'])
  ls2 # ['cat', 'dog', ['rabbit']]
  ```
- Use the `extend` method to add to a list
  ```python
  ls.extend(['1', '2'])
  ```
- Use the `insert` method
  ```python
  ls.insert(index, element)  # if the index is out of range then
  # it will insert to the tail if index >= len(ls)
  # it will insert to the head if index <= -len(ls)
  ls.insert(0, element)
  ls.insert(len(ls), element)  # the same as ls.append(element)
  ```

**Strings**

- Strings are essentially lists of characters except:
  - Strings are immutable. You cannot change an existing string object. Instead, you have to create a new object.
  - There isn't a "character" type in Python.
  - But, intuitively, you can think of strings as unchangeable lists where each element is a single character.
  - As we saw last time, this is exactly what happens when we use a string as a list.
    ```python
    s = 'bear'
    s[2]       # 'a'
    t = ['teddy']
    t += s
    t         # ['teddy', 'b', 'e', 'a', 'r']
    ```

Remember that lists store references, so when we copy a list we get a copy of the stored references.
Strings

- To create or assign a string -- single or double quotes, str()
- To access values in a string
  - Python does not support a character type
  - Essentially a list
- To update – reassign,
- To clear or remove a string (assign " or del)
- Operators
  - Standard type operators such as comparison <, ==, !=, ...
  - Slices ([index], [from : to])
  - Membership (in, not in)
- +, *
- Build-in functions
  - cmp()
  - len()
  - max() and min()
  - ...

Tuple

- Tuples are immutable lists.
- Normally enclosed in round brackets, but this isn't always necessary.

```python
    t1 = (2, 3, 4)  # 4
t1[2]
t2 = 2, 3, 4
    t1 == t2  # True
t3 = (1)  # t3 is *not* a tuple, it's the int 1
t4 = (1,)
    t4 == t5  # True
```

- Observe that tuples of size one need a trailing comma to differentiate them from a single object.

Slicing

- Slicing allows us to extract a portion of a sequence (rather than just a single element).

```python
    ls = [5, 6, 7, 8]
    ls[1:4]  # [6, 7, 8]
    ls[1:3]  # [6, 7]
    ls[2:]  # [7, 8]
    ls[:2]  # [5, 6]
    s = "hello"
    s[1:4]  # "ell"
    t = (5, 6, 7, 8)
    t[:]  # (5, 6, 7, 8)
```

- A slice is of the form:

  `<sequence>[:<start-index>:<end-index + 1>]`
- When the start index is omitted, the slice starts at the beginning of the sequence.
- When the end index is omitted, the slice ends at the end of the sequence.
**Negative and out-of-range indices**

- Negative indices count backward from the end of a sequence. The last item in a sequence has index -1.
  
  ```python
  ls = ['c', 'e', 'f', 'd', 'g']
  ls[-1]  # 'g'
  ls[-2]  # 'd' (think of it as ls[len(ls)-2])
  ls[1:-1]  # ['e', 'f', 'd']
  ls[1:-2]  # ['e', 'f'] (think of it as ls[1:len(ls)-2])
  ```

- Out-of-range indices cause errors in single-element references but not in slices.
  ```python
  ls = ['c', 'e', 'f', 'd', 'g']
  ls[99]  # IndexError
  ls[3:99]  # ['d', 'g']
  ls[-10:-8]  # []
  ```

**Range Function**

- The `range()` function makes a list of numbers.
  ```python
  range(3) # [0, 1, 2]
  range(2, 5) # [2, 3, 4]
  range(2, 10, 3) # [2, 5, 8]
  range(3, 1) # []
  range(3, 1, -1) # [3, 2]
  ```

  ```python
  s = 'python'
  for i in range(len(s)):
    print i, s[i],
  #output is: 0 p 1 y 2 t 3 h 4 o 5 n
  ```

**Dictionaries**

- A dictionary is a container but it is not a sequence. It is a mutable set of key-value pairs.
  - Like an instance of the HasTable class in Java

  ```python
  d = {'Newton': 1642, 'Darwin': 1809}
  d['Newton']  # 1642
  d['Darwin'] = 1809
  d['Turing']  # KeyError
  if 'Turing' in d:
    # False
    print d['Turing']
  else:
    d['Turing'] = 1912
  # Getting a list of keys:
  d.keys() # ['Darwin', 'Turing', 'Newton']
  ```

**Dictionaries**

- Example:

  ```python
  bdays = ['May', 'Jun', 'Jun', 'Apr', 'May', 'Jun']
  freq = {}
  for v in bdays:
    if v in freq:
      freq[v] += 1
    else:
      freq[v] = 1
  print freq
  # {'Apr': 1, 'Jun': 3, 'May': 2}
  # We can replace the if/else with:
  freq[v] = freq.get(v, 0) + 1
  ```

- Observe that `freq.get(v, 0)` returns `freq[v]` if this exists, and 0 otherwise.
Sets

• Set – An unsorted collection with no duplicate elements

• Examples:
  >>> basket = ['apple', 'grape', 'pear', 'apple', 'pear']
  >>> fruit = set(basket)
  >>> fruit
  set(['grape', 'pear', 'apple'])

• Set operations
  >>> a = set('abracadabra')
  >>> b = set('alacazam')
  >>> a - b       # letters in a but not in b
  >>> a | b       # letters in a or in b
  >>> a & b       # letters in both a and b
  >>> a ^ b       # letters in a or b but not in both