CIS192 Python Programming
Data Types

Joseph Cappadona
University of Pennsylvania
September 03, 2015
Outline

1. Data Types
   - Lists
     - Strings
     - Dictionaries
     - Sets

2. Comprehensions
   - Lists
   - Other Comprehensions

3. Extras
Creating a List

- `list()` and `[]` are both new empty lists
- Comma separated `[1, 2, 3]` and nested `[[1, 2], [3, 4]]`
- Construct from iterable `list(range(3))`
- Concatenating two lists with `+` creates a new list.
- Lists are mutable
- Implemented as a resizable array in CPython
Indexing and Slicing

- Index with square brackets
- Negative indexing gets elements from the end of list
  - \( \text{lst}[-1] \) is the last element
  - \( \text{lst}[-2] \) is the second to last element
- Can index multiple times with \( \text{lst}_\text{of}_\text{lst}[][] \)
Builtins

- `len(lst)`: gives the number of elements
- `sum(lst)`: adds up elements
- `a in lst`: checks presence
- `all(lst)/any(lst)`: return True is any/all in lst are True
- `max(lst)/min(lst)`: biggest/smallest element
- `reversed(lst)`: iterator of elements in reverse order
- `zip(lst1,lst2)`: list of tuples with one element from each list
- `sorted(lst)`: returns new sorted list
Iterate with `for x in lst:`
  ▶ Then use `x` in the loop

Never do `for i in range(len(lst)):`
  ▶ Then use `lst[i]` in the loop

Index and value with `for i, x in enumerate(lst):`
  ▶ Useful if you sometimes want `lst[2*i]` or `lst2[i]`
Modifying Lists

- `lst[i] = v`: Change an element or slice by assigning to it
- `lst.append(v)`: Add an element
- `lst.extend(vs)`: Add an iterable
- `lst.remove(v)`: remove a specific value
- `del lst[i]`: remove a specific index or range
- `lst.insert(i, v)`: insert before a certain index with
- `lst.pop(i)`: remove and return index
- `lst.sort()`: in place sort
Multiplying a list adds it to itself.
  - The component lists are not copies, they're the same object

Shallow copy a list with `lst[:]`

Use the `copy` module for deep copy
  - `copy.deepcopy(lst)`
Outline

1. Data Types
   - Lists
   - Strings
   - Dictionaries
   - Sets

2. Comprehensions
   - Lists
   - Other Comprehensions

3. Extras
Methods

- `s.split(sep)`: returns a list of substrings separated by `sep`
- `s.strip()`: strips whitespace from ends
  - Can specify non-whitespace chars to remove: `s.strip('abc')`
- `s.isspace()`: returns True if all chars in `s` are whitespace
- `s.lower()`: converts all characters to lowercase
Join

- `s.join(str_list)`: Concatenates the strings in `str_list` with `s` as a separator.
- When `s` is empty string: efficient way to concatenate strings
- Use space as `s` to join words with spaces
Find and Replace

- `s.find(sub)`
  - finds the starting index of the first occurrence of sub in s

- `s.replace(old,new)`
  - replaces all occurrences of old in s with new
s.format(arg1, arg2): replaces {} in s with args

{name!conversion:format} provides options on top of {}

Use {0}{1}... to refer to positional arguments

Use {name} and then s.format(name=arg) for named args

{!s} {!r} {!a} call str() repr() and ascii() before substitution

{:4}{:7} at least x number of chars

{:b}{:x}... formats number as binary, hex ...

Lots of other stuff in Format Specification Mini-Language
Outline

1 Data Types
   - Lists
   - Strings
   - Dictionaries
   - Sets

2 Comprehensions
   - Lists
   - Other Comprehensions

3 Extras
A dictionary is a hash map
- It hashes the keys to lookup values
- Keys must be immutable so that the hash doesn’t change

`dict()` and `{}` are empty dictionaries

`dict([(k1, v1), (k2, v2)])` or `{k1:v1, k2:v2}`

`dict(zip(key_lst, val_lst))`

`d[k]` accesses the value mapped to `k`

`d[k] = v` updates the value mapped to `k`
Methods

- `len()`, `in`, and `del` work like lists
- `d.keys()` and `d.values()` return views of the keys and values.
  - Views support iteration, `len()`, and `in`
  - Views change when the dictionary changes
- `d.items()` is a view of (k,v) pairs
- `d.get(k, x)` looks up the value of k. Returns x if k not in d
- `d.setdefault(k, x)` same as `d.get(k, x)`
  - Also sets `d[k] = x` if k not in d
- `d.pop(k, x)` Return and remove value at k. Returns x as default
Switch Statement

- Python doesn’t have a switch(x)
- Dictionaries do the job
- Replace long `if x = a: elif x = b: elif...`
  - With a dictionary lookup
**Defaultdict**

- `from collections import defaultdict`
- `dd = defaultdict(f)`
- `if k not in dd then x = dd[k]
  - dd[k] = f()
  - x = dd[k]`
Outline

1. Data Types
   - Lists
   - Strings
   - Dictionaries
   - Sets

2. Comprehensions
   - Lists
   - Other Comprehensions

3. Extras
Basics

- No order, no duplicates
- Hash Set: elements must be immutable
- Empty set: `set()` not `{}` (empty dict)
- `{1, 'blah', 5, -1}`
- Can de-duplicate a list: `list(set(lst))`
Methods

- \texttt{s.add(v)}: adds a value to set
- \texttt{s.remove(v)}: removes v. will raise an error if v not in s
- \texttt{s.discard(v)}: removes v. will not raise error
- \texttt{s.difference(s2) \rightarrow s - s2}: elements in s but not s2
- \texttt{s.union(s2) \rightarrow s | s2}: elements in s or s2
- \texttt{s.intersection(s2) \rightarrow s & s2}: elements in s and s2
- \texttt{s.update(s2)}: s = s | s2
Frozen Sets

- \texttt{frozenset(\{x, y, z\})}
- Immutable version of set
- Can be used as dictionary keys and elements of other frozensets
- Same operations as sets except any that mutate (add, update)
Outline

1. Data Types
   - Lists
   - Strings
   - Dictionaries
   - Sets

2. Comprehensions
   - Lists
   - Other Comprehensions

3. Extras
List Comprehensions

- `[expr for v in iter]`
- `[expr for v1,v2 in iter]`
- `[expr for v in iter if cond]`

**Translation:**

```python
res = [v1 * v2 for v1, v2 in lst if v1 > v2]
res = []
for v1, v2 in lst:
    if v1 > v2:
        res.append(v1 * v2)
```
Nested List Comp

- \[ [x \text{ for } x \text{ in } \text{lst1}] \text{ for } y \text{ in } \text{lst2}] \]

Translation:

```python
res = []
for y in lst2:
    inter = []
    for x in lst1:
        inter.append(x)
    res.append(inter)
```
Extra ’for’s and ’if’s

[x for x in lst1 if x > 2 for y in lst2 for z in lst3 if x + y + z < 8]

Translation:

res = []
for x in lst1:
    if x > 2:
        for y in lst2:
            for z in lst3:
                if x + y + z > 8:
                    res.append(x)
Outline

1. Data Types
   - Lists
   - Strings
   - Dictionaries
   - Sets

2. Comprehensions
   - Lists
   - Other Comprehensions

3. Extras
Dictionary Comprehensions

- Like lists but swap `[]` for `{}`
- Starts with: `d = dict()`
- Appends with: `d[k] = v`
- `{k: v for k, v in lst}`
- Translation:

```python
    d = dict()
    for k, v in lst:
        d[k] = v
```
Set Comprehensions

- Like dictionaries but no :
- Starts with: \( s = \text{set()} \)
- Appends with: \( s.add(v) \)
- \{x for x in lst\}
- Translation:
  
  ```python
  s = set()
  for x in lst:
    s.add(x)
  ```
Tuple Comprehensions?

tup = (x for x in lst)

type(tup)

<class 'generator'>

We’ll cover generators later
- `x if cond else y`
- `x, y, z = 'a', 'b', 'c'`
- `x, y, z = 'abc'`
- `x, *rest, y, z = range(6)`
- `y <= x <= z`
- `(1,)`
• docs.python.org
• Library Reference
  ▶ Everything that’s builtin including modules (math, collections, ...)
• Language Reference
  ▶ What happens when I assign a variable, or import something
• Tutorial/HOWTOs/FAQs