

Introduction into Python

Python 3: Further Lists and Other Data Structures

Daniel Polani

Note: `help(list)` will give you a list of possible operations (called *methods*) on the `list` structure, e.g.

- `a.append(x)` appends an item to the list `a` (same as `a[len(a):] = [x]` (try it!))

```
Code
a = [1,2,3,4]
a.append(6)
print a
```

```
[1, 2, 3, 4, 6]
```

- `extend(L)` is equivalent to `a[len(a):] = L` (what exactly does it do?)

Notes on Using Methods

Note: Methods for any structures are invoked by a *dot* . between the name of the structure and that of the method and the *brackets*: `a.append(x)`

Note: like functions, methods can be assigned to variables, so if the structure is a list called `a`, the method `append` is called `append.a` and can be assigned as in

```
Code
my_append = append.a
```

More List Methods II

Further List Methods:

- `insert(i,x)` inserts `x` in front of the `i`-th member of the list

```
Code
a = [1,2,3,4]
a.insert(2, 'insertion')
print a
```

```
[1, 2, 'insertion', 3, 4]
```

Reminder: element 2 is the *third* element in the list, as counting starts from 0

- `a.remove(x)` removes the first occurrence of `x` from the list (what happens if `x` is not in list?)

More List Methods III

- `pop([i])` removes the item at position i and returns it
- `index(x)` returns the index of the first item with value x
- `count(x)`: how often does x appear in list?
- `sort(cmp=None, key=None, reverse=False)` sorts the elements of the list, in-place.
`cmp(x, y)` should return -1 , 0 or 1 , depending on whether $x < y$, $x = y$ or $x > y$. You can replace `cmp` by your own function (or lambda expression).
For standard types with standard comparison semantics, `cmp` returns the appropriate -1 , 0 or 1 , so it can be used in above lambda expression.
- `reverse()` reverses list, in-place

Daniel Polani: Introduction into Python – p.5??

Lists as Stacks

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

Daniel Polani: Introduction into Python – p.6??

Lists as Queues

```
>>> queue = ["Eric", "John", "Michael"]
>>> queue.append("Terry") # Terry arrives
>>> queue.append("Graham") # Graham arrives
>>> queue.pop(0)
'Eric'
>>> queue.pop(0)
'John'
>>> queue
['Michael', 'Terry', 'Graham']
```

Note: yes, it's Monty Python, not the snake after which the language's named!

Daniel Polani: Introduction into Python – p.7??

Filtering Tools

```
def even(x):
    return x % 2 == 0
>>> filter(even, range(0, 10))
[0, 2, 4, 6, 8]
def cube(x): return x * x * x
>>> map(cube, range(0, 10))
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Daniel Polani: Introduction into Python – p.8??

List Comprehensions

Allows: elegant construction of lists

```
Code
vec = range(10)
print vec
print [2*x for x in vec]
```

prints

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

List Comprehensions with Conditions

Conditions: use if

```
Code
print [[x, x * x] for x in vec]
print [[x, x * x] for x in vec if x % 2]
```

to obtain

```
[[0, 0], [1, 1], [2, 4], [3, 9], [4, 16],
 [5, 25],[6, 36], [7, 49], [8, 64], [9, 81]]

[[1, 1], [3, 9], [5, 25], [7, 49], [9, 81]]
```

Tuples

Note: almost like lists, but have fixed structure. Items separated by list

```
Input
>>> t = 12345, 54321, 'hello!'
>>> t[0]
```

```
12345
Input
>>> t
```

```
(12345, 54321, 'hello!')
```

```
Input
>>> # Tuples may be nested:
>>> u = t, (1, 2, 3, 4, 5)
>>> u
```

```
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
Input
>>> empty = ()
>>> singleton = 'hello', # <-- note trailing comma
>>> singleton
('hello',)
```

Note: this ugly syntax for singletons cannot be avoided

The del method

Operator: del removes a member from a list

```
Input
>>> a = [1,2,3,4,5]
>>> del a[0]
>>> a
```

[2,3,4,5]

```
Input
>>> del a[1:3]
>>> a
```

[2,5]

and can be used to delete a full variable: del a

Sets

Def.: a set is an unordered collection without duplicates.

```
Input
>>> basket = ['apple', 'orange', 'apple',
              'pear', 'orange', 'banana']
>>> fruits = set(basket) # create a set without duplicates
>>> fruits
```

```
set(['orange', 'pear', 'apple', 'banana'])
```

```
Input
>>> 'orange' in fruits # fast membership testing
```

```
True
```

```
Input
>>> 'crabgrass' in fruits
```

```
False
```

Dictionaries

Def.: dictionaries, also *maps*, *associative arrays* map (almost) arbitrary objects to other objects: like arrays with not-necessarily numeric indices

Note: in Python, indices can only be immutable objects: e.g. strings, numbers or immutable tuples (i.e. tuples that recursively contain only immutable objects) — no lists

Concept: unordered set of key:value pairs

Empty Dict: {}

Use: like an array/list

Further Functions: `keys()` returns a list of dict keys.
`has_key()` checks whether a key is in the dictionary.

Dict: Example

```
Input
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
```

```
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

```
Input
>>> tel['jack']
```

```
4098
```

```
Input
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
```

```
{'guido': 4127, 'irv': 4127, 'jack': 4098}
```

```
Input
>>> tel.keys()
```

```
['guido', 'irv', 'jack']
```

```
Input
>>> tel.has_key('guido')
```

```
True
```

Dict Constructor

```
Input
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```

```
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

```
Input
>>> dict([(x, x**2) for x in vec]) # use a list comprehension
```

```
{2: 4, 4: 16, 6: 36}
```

Set Operations (and Conversions)

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                     # unique letters in a
```

```
set(['a', 'r', 'b', 'c', 'd'])
```

```
>>> a - b                                     # letters in a but not in b
```

```
set(['r', 'd', 'b'])
```

```
>>> a | b                                     # letters in either a or b
```

```
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
```

```
>>> a & b                                     # letters in both a and b
```

```
set(['a', 'c'])
```

```
>>> a ^ b                                     # letters in a or b but not both
```

```
set(['r', 'd', 'b', 'm', 'z', 'l'])
```