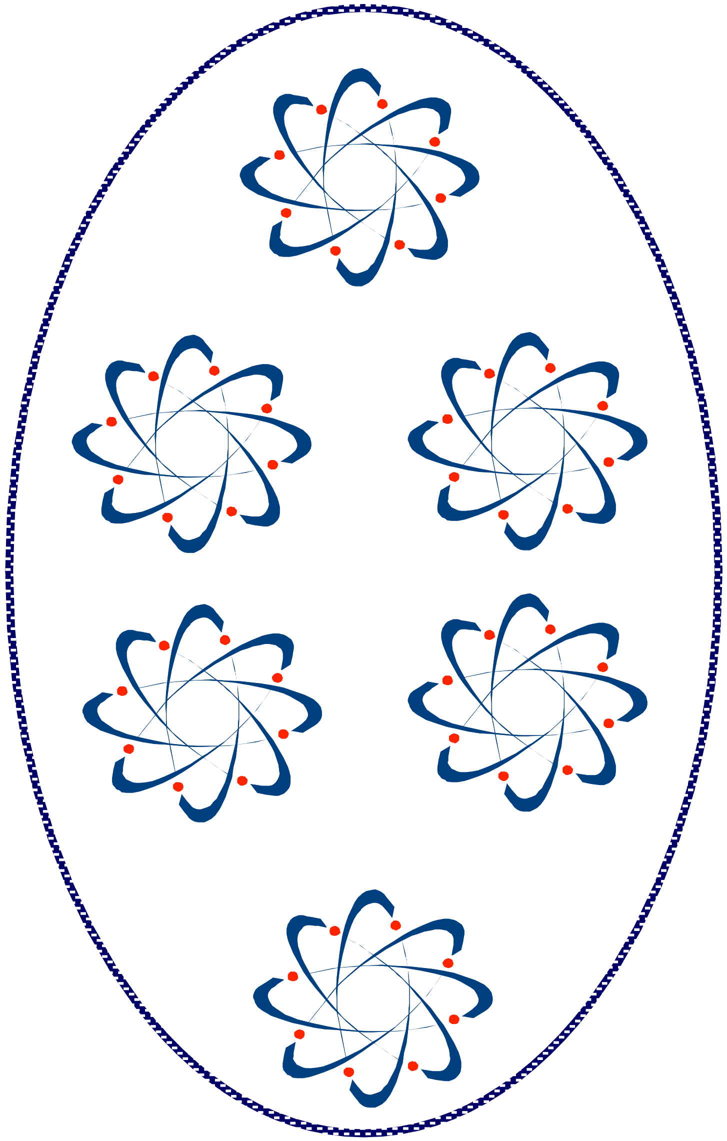
python™ 2 programming

Data Abstraction and Advanced Collections

Data Abstraction and Advanced Collections

- Contents
  - filter
  - List Comprehensions
  - Lazy Lists
  - Generators
  - Generator objects
  - Copying collections
  - Collection module
- Summary



Advanced list functions - filter and map

- `filter`(*function*, *sequence*)
  - Returns a list containing each item where function returns true
  - The function could be named, or a lambda

```
import glob
import os

pattern = 'C:/QA/Python/*'
for name in (filter(os.path.isdir,glob.iglob(pattern))):
    print(name)
```

Print a list of directories

- Also related is map
  - `map`(*function*, *sequence*)
  - Return a list after the function has been applied to each element of the old list

List comprehensions

- **A list comprehension returns a list**
- **It consists of:**
  - **An expression which identifies a list item**
  - **A loop - typically a `for` loop**

```
pattern = 'C:/QA/Python/*'
sizes = [os.path.getsize(fname)
         for fname in glob.iglob(pattern)]
```

Get a list of file sizes

  - **An optional condition to filter items**
    - *Pythonic* replacement of the `filter` built-in

```
dirs = [fname for fname in glob.iglob(pattern)
        if os.path.isdir(fname)]
```

Get a list of directories

dict comprehensions

```
{x: x**2 for x in (2, 4, 6)}
```

set comprehensions

```
a = {x for x in 'abracadabra' if x not in 'abc'}
```

Lazy lists

- Generating lists in memory can be an overhead
  - How big is a list?
  - What about sequences that have no end?
- Lazy lists only return a value when it is needed
  - One item at a time, as and when required
- Particularly suitable when iterators are used
  - An iterator function returns items one at a time
- Some Python functions return iterators rather than lists
  - xrange(), reversed(), and so on
  - A lot more in Python 3

Generators

- **A generator is a function which yields a lazy list**
  - **A lazy list item is returned at the `yield` statement**

```
def get_dir(path):
    pattern = path + '/*'
    for file in glob.iglob(pattern):
        if os.path.isdir(file):
            yield file
```

  - **Generators can often replace list comprehensions**
    - Can be used anywhere an iterator is expected

```
for dir in get_dir('C:/QA/Python'):
    print dir
```

Print a list of directories

```
dirs = list(get_dir('C:/QA/Python'))
```

Get a list of directories

Generator objects and next

- A generator function returns a generator object
  - Can be used when a 'for' loop is not appropriate

```
gen = get_dir('C:/QA/Python')
```

Using the generator function from the previous slide

- The next built-in gets the next item from a generator

```
while True:
    name = next(gen, False)
    if name: print name
    else: break
```

```
C:/QA/Python\Appendicies
C:/QA/Python\bak
```

- A loop does not have to be used

```
gen = get_dir('C:/QA/Python')
dir1 = next(gen, False)
dir2 = next(gen, False)
dir3 = next(gen, False)
```

Co-routines and send() method

- **Data can be returned to the generator using send**

```
import glob, os

def get_dir(path):

    while True:

        pattern = path + '/*'

        for file in glob.iglob(pattern):
            if os.path.isdir(file):
                path = yield file
                if path: break

        if not path: break

gen = get_dir('C:/QA/Python')

print next(gen)
print next(gen)
print gen.send('C:/MinGW')
print next(gen)
```

Both `next()` and *gen*`.send()` get the next yielded value

```
C:/QA/Python\AdvancedPython
C:/QA/Python\Appendicies
C:/MinGW\bin
C:/MinGW\dist
```

List comprehensions as generators

- A list comprehension may be used instead of `yield`
  - Sometimes - this does not support sending values
  - Enclose the comprehension in `()` instead of `[]`
  - Original example:

```
def get_dir(path):
    pattern = path + '/*'
    for file in glob.iglob(pattern):
        if os.path.isdir(file):
            yield file
```

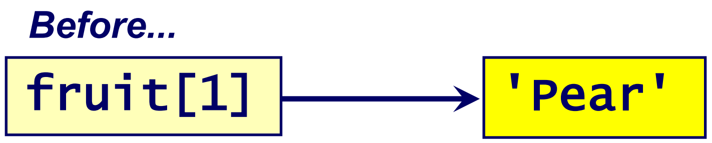
  - Rewritten as a list comprehension:
    - Function returns a generator object, as before

```
def get_dir(path):
    pattern = path + '/*'
    return (file
            for file in glob.iglob(pattern)
                if os.path.isdir(file))
```

Copying collections - problem

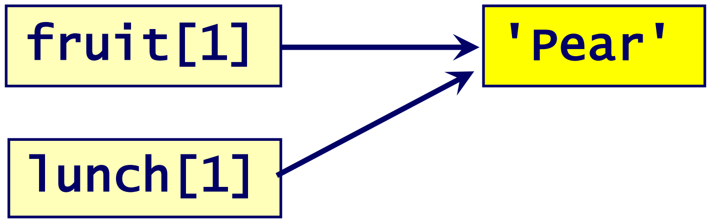
- **Any problems with assignments?**
  - **Remember that Python objects are references**

```
fruit = ['Apple','Pear','Orange']
```



Shallow copy...

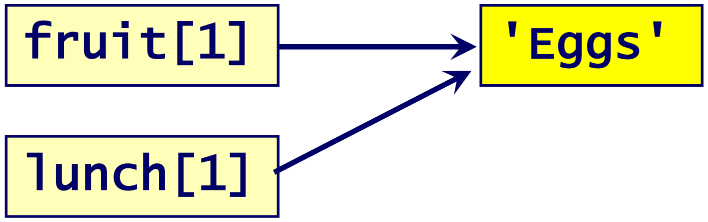
```
lunch = fruit
```



Oops...

```
lunch[1] = 'Eggs'
print fruit
```

```
['Apple', 'Eggs', 'Orange']
```



Copying collections - slice solution?

- **For a sequence, take a slice**

```
fruit  = ['Apple', 'Pear', 'Orange']
lunch = fruit[:]
lunch[1] = 'Eggs'
print 'fruit:',fruit,'\nlunch:',lunch
```

```
fruit: ['Apple', 'Pear', 'Orange']
lunch: ['Apple', 'Eggs', 'Orange']
```

- **We need a better solution for more complex structures**
  - **A slice is still a shallow copy**

```
fruit = ['knife','plate',['Apple', 'Pear', 'Orange']]
lunch = fruit[:]
lunch[2][1] = 'Eggs'
print 'fruit:',fruit,'\nlunch:',lunch
```

```
fruit: ['knife', 'plate', ['Apple', 'Eggs', 'Orange']]
lunch: ['knife', 'plate', ['Apple', 'Eggs', 'Orange']]
```

Copying collections - deepcopy solution

- **A better solution for more complex structures**
- **The copy module, distributed with Python**
  - **Can do a shallow copy or a deep-copy**

```
import copy

fruit = ['knife','plate',['Apple', 'Pear', 'Orange']]
lunch = copy.deepcopy(fruit)
lunch[2][1] = 'Eggs'
print 'fruit:',fruit,'\nlunch:',lunch
```

```
fruit: ['knife', 'plate', ['Apple', 'Pear', 'Orange']]
lunch: ['knife', 'plate', ['Apple', 'Eggs', 'Orange']]
```

- **Beware! "copy" usually means a shallow copy**

Collection module

- **High-performance container datatypes**
- **Implements specialized container datatypes**
- **Providing alternatives to Python’s general purpose built-in containers**

`namedtuple()`	factory function for creating tuple subclasses with named fields *New in version 2.6.*
`deque`	list-like container with fast appends and pops on either end *New in version 2.4.*
`Counter`	dict subclass for counting hashable objects *New in version 2.7.*
`OrderedDict`	dict subclass that remembers the order entries were added *New in version 2.7.*
`defaultdict`	dict subclass that calls a factory function to supply missing values *New in version 2.5.*

defaultdict Example

```
>>> s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
>>> d = defaultdict(list)
>>> for k, v in s:
...     d[k].append(v)
...
>>> d.items()
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

OrderedDict Example

```
>>> # regular unsorted dictionary
>>> d = {'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2}


>>> # dictionary sorted by key
>>> OrderedDict(sorted(d.items(), key=lambda t: t[0]))
OrderedDict([('apple', 4), ('banana', 3), ('orange', 2), ('pear', 1)])
```

Summary

- filter() returns items that are true
  - Maybe with the help of a lambda
- List comprehensions replace filter() and map()
  - Possibly with the help of a lambda
- Generators yield values as they are needed
- Generators can replace list comprehensions
- Copying collections might not be a simple assignment
  - A deep copy might be required
- Collections module gives more power to the built-in types