Comprehensions
Rudiments of Functional Programming

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

E-mail: cazzola@dico.unimi.it

Comprehensions
Introduction

Comprehensions are a compact way to transform a set of data into another
- it applies to mostly all python's structured types, i.e., lists, sets, dictionaries
- it is in contrast to list all the elements

Some basic comprehensions applied to lists, sets and dictionaries respectively
- a list composed of the first ten integers
  ```python
  >>> [elem for elem in range(1,10)]
  [1, 2, 3, 4, 5, 6, 7, 8, 9]
  ```

- a set composed of the first ten even integers
  ```python
  >>> {elem*2 for elem in range(1,10)}
  {2, 4, 6, 8, 10, 12, 14, 16, 18}
  ```

- a dictionary composed of the first ten couples "n, n^2"
  ```python
  >>> {elem:elem**2 for elem in range(1,10)}
  {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
  ```

To Filter out Elements of a Dataset

Comprehensions can reduce the elements in the dataset after a constraint.

E.g., to select perfect squares out of the first 100 integers
```python
>>> [elem for elem in range(1,100) if (int(elem**.5))**2 == elem]
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```
- range(1,100) generates a list of the first 100 integers;
- the comprehension filters through the list selecting those elements whose square of the integral part of its square roots are equal to the element itself

E.g., to select the odd numbers out of a tuple
```python
>>> (x for x in (1, 22, 31, 23, 10, 11, 11, -1, 34, 76, 778, 10101, 5, 44) if x%2 != 0)
{1, 5, 11, 10101, 23, -1, 31}
```
- note that the second 11 is removed from the set;
- the set does not respect the tuple order (it is not ordered at all)
Comprehensions can select multiple values out of the dataset.

E.g., to swap keys and values in the dictionary

```python
>>> a_dict = {'a': 1, 'b': 2, 'c': 3}
>>> {value:key for key, value in a_dict.items()}
{1: 'a', 2: 'b', 3: 'c'}
```

Comprehensions can select values out of multiple datasets

E.g., to merge two sets in a set of couples

```python
>>> english = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
>>> greek = ['a','b','q','d','e','f','g','h','i','w','k','l','m','n','o','p','q','r','s','t','j','u','"','x','y','z']
>>> [(english[i],greek[i]) for i in range(0,len(english))]
[('a', 'a'), ('b', 'b'), ('c', 'q'), ('d', 'd'), ('e', 'e'), ('f', 'f'), ('g', 'g'), ('h', 'h'), ('i', 'i'), ('j', 'w'), ('k', 'k'), ('l', 'l'), ('m', 'm'), ('n', 'n'), ('o', 'o'), ('p', 'p'), ('q', 'q'), ('r', 'r'), ('s', 'c'), ('t', 't'), ('u', 'j'), ('v', 'u'), ('w', '"'), ('x', 'x'), ('y', 'y'), ('z', 'z')]
```

E.g., to calculate the Cartesian product

```python
>>> {(x,y) for x in range(3) for y in range(5)}
{(0,1),(1,2),(0,0),(2,2),(1,1),(1,4),(0,2),(2,0),(1,3),(2,3),(2,1),(0,4),(2,4),(0,3),(1,0)}
```

Comprehensions

Prime Numbers Calculation

Classic approach to the prime numbers calculation

```python
def is_prime(x):
    div = 2
    while div <= math.sqrt(x):
        if x%div == 0:
            return False
        else:
            div += 1
    return True

if __name__ == '__main__':
    primes = []
    for i in range(1, 50):
        if is_prime(i): primes.append(i)
    print(primes)
```

The algorithm again but using comprehensions

```python
def is_prime(x):
    div = [elem for elem in range(2,int(math.sqrt(x))+1) if x%elem ==0 ]
    return len(div) == 0

if __name__ == '__main__':
    print([elem for elem in range(1, 50) if is_prime(elem)])
```

```bash
[8:50]cazzola@ulik:~/esercizi-pa>python3 imp-sieve.py
[1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
[1:31]cazzola@ulik:~/esercizi-pa>python3 sieve.py
[1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Quicksort

```python
def quicksort(s):
    if len(s) == 0:
        return []
    else:
        return quicksort([x for x in s[1:] if x < s[0]]) + [s[0]] +
        quicksort([x for x in s[1:] if x >= s[0]])

if __name__ == '__main__':
    print(quicksort([]))
    print(quicksort([2, 4, 1, 3, 5, 8, 6, 7]))
    print(quicksort('pineapple'))
    print(''.join(quicksort('pineapple')))
```

```bash
[23:22]cazzola@ulik:~/esercizi-paw-python3 quicksort.py
[]
[1, 2, 3, 4, 5, 6, 7, 8]
['a', 'e', 'o', 'l', 'n', 'p', 'p', 'p', 'p']
```

Comprehensions

References