CSc 120
Introduction to Computer Programming II

Adapted from slides
by Dr. Saumya Debray

11: List Comprehensions
List comprehensions

• A list comprehension is a simple and concise way to create lists

Example: compute a list of squares of numbers

```python
>>> def squares(n):
    outlist = []
    for i in range(n):
        outlist.append(i*i)
    return outlist

>>> squares(3)
[0, 1, 4]
>>> squares(6)
[0, 1, 4, 9, 16, 25]
```

```python
>>> def squares1(n):
    return [i*i for i in range(n)]

>>> squares1(3)
[0, 1, 4]
>>> squares1(6)
[0, 1, 4, 9, 16, 25]
```

list comprehension
Basic structure

\[
[ \text{expr for } item \text{ in } \text{some_list if } con] \]

|||
\[
\text{new_list} = []
\]

\[
\text{for } item \text{ in } \text{some_list:}
\]

\[
\text{if } cond:
\]

\[
\text{new_list.append(expr)}
\]
Example 1

```python
>>> def odds_and_evens(arglist):
    odds = []
    evens = []
    for i in range(len(arglist)):
        if i % 2 == 0:
            evens.append(arglist[i])
        else:
            odds.append(arglist[i])
    return (odds, evens)

>>> (o, e) = odds_and_evens([0, 1, 2, 3, 4, 5])
>>> o
[1, 3, 5]
>>> e
[0, 2, 4]
```
Example 1

```python
>>> def odds_and_evens(arglist):
    odds = []
    evens = []
    for i in range(len(arglist)):
        if i % 2 == 0:
            evens.append(arglist[i])
        else:
            odds.append(arglist[i])
    return (odds, evens)
```

Using list comprehensions:

```python
>>> def odds_and_evens(arglist):
    idxs = range(len(arglist))
    odds = [arglist[i] for i in idxs if i % 2 != 0]
    evens = [arglist[i] for i in idxs if i % 2 == 0]
    return (odds, evens)
```

```python
>>> (o, e) = odds_and_evens([0,1,2,3,4,5])
>>> o
[1, 3, 5]
>>> e
[0, 2, 4]
```
Example 2

```python
>>> import string

>>> # strip punctuation from around words
>>> def strip_punctuation(wordlist):
...     puncts = string.punctuation
...     return [wd.strip(puncts) for wd in wordlist]

>>> wordlist = "Look! Here's--> punctuation:!@#$%^&".split()
>>> strip_punctuation(wordlist)
["Look", "Here's", "punctuation"]
>>>
style considerations
Style considerations

• Use loops for:
  – code that has side effects, i.e., does I/O or modifies other objects

• Use list comprehensions for:
  – creating lists
    o using code that does not have side effects

• Avoid long or nested list comprehensions
  – these can be hard to read and understand

```python
def primes_upto(n):
    return [prime for prime in range(2, n) if prime not in 
        [notAPrime for i in range(2, i, (n**0.5)) for notAPrime in range(i * 2, n, i)]]
```