

Working with Lists

Informatics 1 for Biomedical Engineers
Tutor Session 5

KTI, Knowledge Technologies Institute

13. November 2016

Today's Topics

- List comprehension
- Wrapping lists (eg.: `enumerate()`)
- Generators

Student Goals

- Feel comfortable with lists
- Know how to use the various comprehension features
- Understand how generators work

Rehearsal

- Lists
 - Ordered list of items of multiple data-types
- Sets
 - Unordered collection of unique items (must be hashable)
 - Can do intersections and unions
- Dictionaries
 - Key-Value based data structure
 - Keys must be hashable and values can be any data-type

List comprehension

- Perform an operation on an iterable in a single line
- Exists for dicts, lists, sets, tuples, ...
- Not in-place (returns new object)
 - Tuple creates generator (good for memory)
- Syntax

```
1  <new_iterable> = [<operation> for <item> in <iterable>]  
2  # also supports statements like 'if'  
3  # can be nested
```



- Improves readability and speed

List comprehension

Task: Square all odd numbers in a list

1. Using normal loops (`range(len(list))`)
2. Using list comprehension

Odd Squares “normal loop version”

```
1 fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] # Long enough...
2 odd_squared = []
3 for entry in fibonacci:
4     if entry % 2 == 1:
5         odd_squared.append(entry**2)
6     else:
7         odd_squared.append(entry)
8
9 print(odd_squared) # [0, 1, 1, 2, 9, 25, 8, 169, 441, 34]
```



Odd Squares “one-liner”

```
1 fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] # Long enough...
2 odd_squared = [x**2 if x % 2 == 1 else x for x in fibonacci ]
3 print(odd_squared)
```



Generators

- Fast and memory saving way for defining iterables
- use the *yield* statement
 - Returns function value but remembers state

enumerate()

```
1  fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] # Long enough...
2
3  for index, value in enumerate(fibonacci):
4      print(index, value)
5  # 0, 0
6  # 1, 1
7  # 2, 1
8  # 3, 2
9  # 4, 3
10 # 5, 8
11 # 6, 13
12 # 7, 21
13 # 8, 34
```



enumerate()

```
1  def my_enumerate(_list):  
2      index = 0  
3      for entry in _list:  
4          yield index, entry  
5          index += 1
```



Generators

Task: Cumulative sum of the Fibonacci series

$$(F_n = F_{n-1} + F_{n-2})$$

1. Using normal loops
2. Using list comprehension

Fibonacci sum “normal loop version”

```
1  fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] # Long enough...
2  fibonacci_sum = []
3  current_sum = 0
4
5  for entry in fibonacci:
6      current_sum += entry
7      fibonacci_sum.append(current_sum)
8
9  print(fibonacci_sum) # [0, 1, 2, 4, 7, 12, 20, 33, 54, 88]
```



Fibonacci sum “generator version”

```
1  fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] # Long enough...
2
3  # the generator
4  def accumulate(_list):
5      total = 0
6      for x in _list:
7          total += x
8          yield total
9
10 # the 1-liner
11 fibonacci_sum = list(accumulate(fibonacci))
12 print(fibonacci_sum) # [0, 1, 2, 4, 7, 12, 20, 33, 54, 88]
```



Complex example

Task: Calculate a retirement fund

- Estimate values
 - Current income
 - Remaining work years
 - Your burn rate in retirement
 - Assume you earn 3% extra every year
 - Interest at a set 1% per year
 - Ignore inflation
- How long do you last?
- “What percentage of your income should you save?”

Let's estimate the parameters

```
1  #set up the base values
2  current_income = 2500 # how much we make right now
3  remaining_years = 40 # how many more years do we work
4  ratio = 0.15 # what rate of our income do we save?
5  burn_rate = 2300 # how much money do we need once retired?
6
7  income_increase = 1.03
8  interest = 1.01
```



Define our generators for income and retirement

```
1  def salary(_base, _factor, _years):
2      current = _base
3      yield 0, current
4      for year in range(_years):
5          current = current * _factor
6          yield year, current
```



```
1  def retirement(_savings, _burn_rate, _interest):
2      while True:
3          _savings = (_savings - _burn_rate*12) * _interest
4          yield _savings
5          if _savings < 0:
6              break
```



Calculate your savings for the time you work

```
1 savings = [interest**(remaining_years - year) * ((income*ratio) * 12)
2           for year, income
3           in salary(current_income, income_increase, remaining_years)]
4
5 #print(savings)
```



Calculate how long we would last

```
1 _ = [print(years, remaining)
2       for years, remaining
3       in enumerate(retirement(total_savings, burn_rate, interest))]
```



Student Task

Task: 1-player scrabble

- Get all ASCII uppercase letters (import string)
 - transform to list to separate
- Assign values to letters (randomly 1-5 points)
- Present the user with 10 letters
 - at least 4 different characters (import random)
 - 4 totally random characters
 - add 2 vowels at the end
- Check and score user input (NO dictionary check)

Student Task - Solution

```
1  # import the packages we need
2  import string
3  import random
4
5  # set up our letters
6  letters = list(string.ascii_uppercase)
7  vowels = ["A", "E", "I", "O", "U"]
```



```
1 # assign value for each letter
2 dictionary = {letter: random.randint(1,5) for letter in letters}
```



```
1 # get the first 4 sampled letters
2 user_letters = random.sample(letters, 4)
3
4 # lets add 4 totally random ones and finally 2 vowels
5 user_letters +=
6     [letters[random.randint(0,len(letters) - 1)] for i in range(4)] +
7     [vowels[random.randint(0,len(vowels) - 1)] for i in range(2)]
8 # this is just one line of code!
```



Start game loop and ask for input

```
1  correct_word = False
2  while not correct_word:
3      print("These are your letters: " + str(user_letters))
4      user_input = input("Please enter your word: ").upper()
5      print("You entered: " + user_input)
6      input_letters = list(user_input)
7
8      # make a copy of the letters so we can remove them one by one
9      remaining_letters = user_letters
10     score = 0
11
12     # we start by assuming the input is correct
13     correct_word = True
```



```
1  # check each letter
2  for input_letter in input_letters:
3      # tell the user he is incorrect and jump back to the start
4      if input_letter not in remaining_letters:
5          print("Invalid letter: " + input_letter + "'")
6          correct_word = False
7          break
8
9      # remove the current letter from the allowed list and increase the score
10     remaining_letters.remove(input_letter)
11     score += dictionary[input_letter]
12
13 # if all letters were allowed the user was correct.
14 if correct_word:
15     print("You entered " + user_input + ".")
16     print("You get " + str(score) + " points!")
17     break
```

