CSC110 Lecture 3: Comprehensions and Introduction to Functions

Print this handout

Exercise 1: Practice with comprehension expressions

Here is a summary of the three types of comprehensions, for your reference:

Comprehension type	Syntax
set comprehension	`{ <expression> for <variable> in <collection> }`
list comprehension	`[ <expression> for <variable> in <collection> ]`
dict comprehension	`{ <key_expr>: <value_expr> for <variable> in <collection> }`

1. Suppose we assign the variable `numbers = [1, 2, 3]`.

   a. Fill in the table below.

Expression	Value
`numbers[0]`	1
`numbers[1]`	2
`numbers[2]`	3
`numbers[0] ** 3`	1
`numbers[1] ** 3`	8

Expression	Value
`numbers[2] ** 3`	27

b. Write a comprehension that evaluates to the *list* of every integer in `numbers` cubed (i.e., raised to the power of 3).

```
>>> [ ~~x~~ for x in numbers ]
      x**3
[1, 8, 27]    # Evaluating your expression should produce this
```

c. Write a comprehension that evaluates to a *dictionary* mapping every integer in `numbers` to three times that integer.

```
>>> { ~~3*x~~ for x in numbers }
      x: 3*x
{1: 3, 2: 6, 3: 9}    # Evaluating your expression should produce this
```

Hint: the *identity dictionary comprehension* has the following form:

```
>>> {x : x for x in numbers}
```

d. Write a comprehension that evaluates to the given output dictionary shown.

```
>>> {3*x : x   for x in numbers   }
{3: 1, 6: 2, 9: 3}    # Evaluating your expression should produce this
```

e. Write a comprehension that is a translation of the set builder expression $\{\frac{x}{x+1} \mid x \in \texttt{numbers}$

```
{ ~~x~~  for x in numbers  }
  x/(x+1)
```

Exercise 2: Comprehensions and `range`

1. Write down the integers that are contained in each of the following Python `range` expressions.

   a. `range(0, 5)`

   0, 1, 2, 3, 4

   b. `range(5, 10)`

   5, 6, 7, 8, 9

2. For each of the following descriptions, write a comprehension that evaluates to the described collection.

   a. The set of integers between 30 and 50, inclusive.

   { j for j in range(30, 51) }

   b. The list of integers between -30 and 30, inclusive (in ascending order).

   [k for k in range(-30, 31)]

   c. The set of the squares of the natural numbers less than 2000.

   { ~~t~~ t*t for t in range(0, 2000) }

   d. A mapping from a number to its square, for the natural numbers less than 2000.

   { y: y**2 for y in range(0, 2000) }

3. You are given a variable `s` that refers to a (very very long) string:

```
>>> s = 'nonsensenonsensenonsensenonsensenonsensenonsensenonsense'
```

Write a list comprehension expression that evaluates to a list containing the first 20 characters in th string, in the order they appear.

Hint: `s[19]` is the last character in `s` that should be included in the list.

[s[i] for i in range(0, 20)]

alternate for those who know about slicing (not req'd!)

[c for c in s[:20]]

Exercise 3: Practice with built-in functions

Suppose we have assigned the following variables in the Python console:

```
>>> n = -5
>>> numbers_list = [1, 10, n]
>>> numbers_set = {100, n, 200}
```

1. Complete the following table showing the value of each variable.

Variable	Value
n	-5
numbers_list	[1, 10, -5]
numbers_set	{100, -5, 200}

corrected: had dropped the - before 5

2. Write down what each of the following expressions evaluate to. *Do this by hand first! (Then check your work in the Python console.)*

You may find it helpful to consult Appendix A.1 Python Built-In Function Reference (https://www.teach.cs.toronto.edu/~csc110y/fall/notes/A-python-builtins/01-builtins.html).

```
>>> abs(n)
```

5

```
>>> sorted(numbers_list)
```

~~[1, 5, 10]~~ [-5, 1, 10]

corrected due to earlier mis

```
>>> sorted(numbers_set) + sorted(numbers_list)
```

~~[5, 100, 200, 1, 5, 10]~~ [-5, 100, 200, -5, 1, 10]

```
>>> len(numbers_set)
```

3

```
>>> len(numbers_list) == n
```

False

(100 + -5 + 200) - -5

```
>>> sum(numbers_set) - n
```

~~310~~ 300 Corrected

3. The variable `numbers` refers to a list that contains a mix of positive and negative integers (e.g., `[-1, 2, 3]`). Write a comprehension that evaluates to the set of the absolute values of every integer in `numbers`.

(Hint: the structure is the same as earlier problems on this worksheet. Use the `abs` function.)

{abs(x) for x in numbers}

note that the len of this set may be smaller than len(numbers). when both -y and y are in numbers for some value y.

Additional exercises

1. *Comprehension practice*. For each of the following mappings, write a Python dictionary expression that evaluates to the mapping.

   a. A mapping from a number to its square, for the first 50 natural numbers.

      { x: x **2 for x in range(0,50)}

   b. A mapping from input to output of the function $f(x) = \frac{x}{x-1}$, for integer inputs greater than 1 and less than 2000. { x: x/(x-1) for x in range(2, 2000)}

   c. A mapping from a number to a list that contains the same number of items, where every item i the string `'Hello'`, for the first 50 natural numbers. (e.g., `3` maps to the list `['Hello', 'Hello', 'Hello']`.) { x: ['Hello']*x for x in range(50)}

   d. A mapping from an integer to the set of integers between 0 and that integer inclusive, for integers 1 to 20, inclusive. { x: ~~[~~ { y for y in range(0, x+1) ~~]~~ } for x in range(1,21)}

      corrected!

2. *Comprehensions with multiple variables*. Suppose you have the lists: `nums1 = [1, 2, 3]` and `nums2 = [4, 5, 6]`.

   a. Using both `nums1` and `nums2`, write a comprehension that evaluates to: `[[1, 4], [1, 5], [1, 6], [2, 4], [2, 5], [2, 6], [3, 4], [3, 5], [3, 6]]`.

      [[a,b] for a in nums1 for b in nums2]

   b. Using both `nums1` and `nums2`, write a comprehension that evaluates to: `[[4, 1], [5, 1], [6, 1], [4, 2], [5, 2], [6, 2], [4, 3], [5, 3], [6, 3]]`.

      [[b,a] for a in nums1 for b in nums2]

   c. Using both `nums1` and `nums2`, write a comprehension that evaluates to: `{5, 6, 7, 8, 9}`.

      { a ~~*~~ + b for a in nums1 for b in nums2}

      corrected

3. *Function practice*. Using the same variables defined in Exercise 3, determine the value of each of the following Python expressions.

```
>>> type(n)
        int

>>> type(abs(n))
        Int

>>> type(numbers_list == n)
        bool

>>> type(numbers_list) == type(n)
        False

>>> max(numbers_list + [5])
        10

>>> max(numbers_list) + 5
        15

>>> max(sorted(numbers_list)) == max(numbers_list)
        True
```

4. *Interpreting errors*. Your friend is practicing in the Python console again, and is trying to add two numbers. They type in the following, and get an error:

```
>>> sum(3, 4)
Traceback (most recent call last):
  ... [some output omitted] ...
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Once again, explain this error to your friend, and how they can correctly add two numbers in Pythor
(*Hint*: treat "iterable" as another word for "collection".)

The sum builtin function expects to be given a collection data type and in this function call it was given 2 ints. To determine the desired result, put the literal values 3, 4 into a collection and give it to the sum function.