이 세상이나 세상에 있는 것들을 사랑하지 말라 누구든지 세상을 사랑하면 아버지의 사랑이 그 안에 있지 아니하니 이는 세상에 있는 모든 것이 육신의 정욕과 안목의 정욕과 이생의 자랑이니 다 아버지께로부터 온 것이 아니요 세상으로부터 온 것이라 이 세상도, 그 정욕도 지나가되 오직 하나님의 뜻을 행하는 자는 영원히 거하느니라 (요일2:15-17)

NOTE: The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to `idebtor@gmail.com` Your assistances and comments will be appreciated.

List comprehension

학습 목표

- 리스트 컴프리핸션에 대해 이해한다.
- 리스트 컴프리핸션과 loop를 비교할 수 있다.
- 파이썬의 강점을 살린 코딩을 할 수 있다.

학습 내용

1. 리스트 컴프리핸션
2. 파이써닉 코딩

What is List Comprehension?

- List comprehensions provide us with a simple way to create a list based on some iterable. During the creation, elements from the iterable can be conditionally included in the new list and transformed as needed. An iterable is something you can loop over.

Three components of a comprehension

The components of a list comprehension are:

- Output Expression (Optional)
- Iterable
- Iterator variable which represents the members of the iterable

Syntax

- The syntax is:
    - [**expression** for **variable** in **iterable**]
    - [**expression** for **variable** in **iterable** if **condition**]

- It sounds like

```
- [output expression for item in iterable]
- [output expression for item in iterable if condition]
```

- The `if` **condition** part is optional, the statement and the condition can use variable.

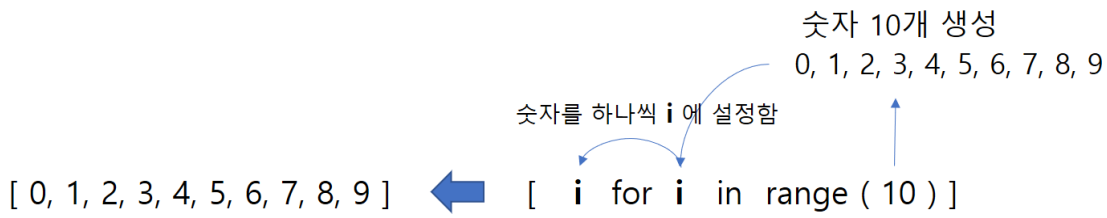

그림 1: 리스트 컴프리헨션

Three kinds of Comprehensions

There are three different kind of comprehensions in Python

- list comprehensions,
- set comprehensions and
- dictionary comprehensions

Loops and Comprehensions

The list comprehensions are more efficient both computationally and in terms of coding space and time than a for loop. Typically, they are written in a single line of code.

Example 1

Using hand-coding

Create a list from 0 to 9 by hand-coding.

In [ ]:
```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Using for loop

Creat a list from 0 to 9 using a for loop

In [ ]:
```
numbers = []
None

numbers
```

Using List comprehension

In [ ]:
```
numbers = None
numbers
```

Example 2

Generate numbers squared from 1 to 10 using a list comprehension as shown below

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Using a for loop

In [ ]:
```
squares = None
None
print(squares)
```

Using list comprehension

In [ ]:
```
numbers = None
numbers
```

Example 3

Generate odd numbers squared from 1 to 10 using a list comprehension as shown below

```
squares = [1, 9, 25, 49, 81]
```

Using a for loop

- With using a conditional statement `if`
- Without using a conditional statement `if` , but utilizing the step in range()

In [ ]:
```
# using for loop and a conditional statement `if`
squares = None
None
print(squares)
```

In [ ]:
```
# using for loop without a conditional statement `if`, but utilizing the step in range
squares = None
None
print(squares)
```

Using list comprehension

We can also create more advanced list comprehensions which include **a conditional statement** on the iterable.

In [ ]:
```
# list comprehension without if, but use the step in range()
squares = None
print(squares)
```

In [ ]:
```
# list comprehension with if
squares = None
print(squares)
```

Example 4

Convert a list of temperatures in Celsius into Fahrenheit using

- Using a for loop and
- Using a list comprehension,

You may convert a Celsisus into Fahrenheit using the formula:

```
f = 1.8 * c + 32
```

In [ ]:
```
clist = [-10, 0, 10, 50, 100, 200]
flist = None
None

print(flist)
```

In [ ]:
```
#use list comprehension
clist = [-10, 0, 10, 50, 100, 200]
flist = None
flist
```

Example 5:

Using nested `if`

Find numbers that are divisible by 2 and 3 from 1 to 50.

The output we are expecting is

```
[6, 12, 18, 24, 30, 36, 42, 48]
```

Using logical `and`

In [ ]:
```
num = [ None ]
num
```

Using nested `if`

In [ ]:
```
num = [ None ]
num
```

Example 6

Let's suppose we have lists in a list. For example, a matrix is a sort of lists in a list. Sometimes, we want to make lists in a list into a list. This operation is called `flatten` . For example, The following matrix may be flattened

```
matrix = [ [1, 2, 3, 4],
           [5, 6, 7, 8],
           [9, 10, 11, 12]]
```

into

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Flatten the following matrix by

- Using for loops
- Using list comprehension.

In [ ]:
```
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
]

flattened = []
None

flattened
```

It flattens the list of the list.

In [ ]:
```
flattened = None      # incorrect ->  [n for n in row for row in matrix]
flattened
```

List Comprehension vs loop

The list comprehensions is more efficient in terms of coding space than a `for` loop since they are typically written in a single line of code. Computationally, however, is it always faster than For-loops? Let's find out.

Step 1:

Let’s square for every item in a list from 1 to 5 such that it produces the following:

```
[1, 4, 9, 16, 25]
```

Let's implement this (when N = 5) when using a for loop.

In [ ]:
```
N = 5
result = []
for x in range(1, N + 1):
    result.append(x * x)

print(result)
```

Step 2:

Let us implement it as a function that returns a list.

In [ ]:
```
def squares(N):
```

```
    return None
```

Step 3:

Let us implement it as a list comprehension.

In [ ]:
```
def squares_x(N):
    return None
```

Step 4

Let us run two functions 10_000 times, respectively and time it.

In [ ]:
```
None
```

In [ ]:
```
None
```

A pythonic way of coding:

For-loop vs List comprehension

A sort of Palindrome(회문)

A string is said to be **palindrome** if the reverse of the string is the same as string. For example, “radar”or "civic" is a palindrome,

Emordnilaps are like **palindromes'** evil twins. Instead of being the same word, these rare words make a different real word when spelled backward! Reverse the word "stop" and it becomes "pots." Flip "drawer" and you get "reward." English contains a surprising variety of words that change meanings as soon as you read them right to left.

Emordnilap Examples:

- desserts and stressed,
- decaf and faced
- edit and tide
- deeps and speed
- stops and spots

In [ ]:
```
from urllib.request import urlopen
#book = urlopen('http://www.gutenberg.org/cache/epub/10/pg10.txt')
book = urlopen('http://composingprograms.com/shakespeare.txt')

# Make a list words out of the book read from url.
wlist = book.read().decode().split()
```

In [ ]:
```
## make a list that has a set of unique words.
```

```
ulist = None

# then, convert the set back to a list type
wlist = None
```

Finding Emordnilaps - a sort of Palindrome

- Using for-loop

In [ ]:
```
alist = []

print(alist)
```

a pythonic way?

- using list comprehension

In [ ]:
```
alist = None
print(alist)
```

Which one is more efficient computationally?

- You may use `%%time` instead of `%%timeit` since it takes long enough to run it once.

In [ ]:
```

```

In [ ]:
```

```

Tuple

A **tuple** is an **immutable** sequence type. One of the ways of creating tuple is by using the `tuple()` construct.

In [ ]:
```
# creating a tuple from a list
t1 = tuple([2, 8, 1, 9])
print('t1 =', t1)

# creating a tuple from a string
t2 = tuple('Python')
print('t2 =',t2)
```

In [ ]:
```
## Example 1: Make a list of tuples from two collections or lists.

- `zip()` with `n` arguments returns an __iterator__ that generates `tuple`s of lengt
```

In [ ]:
```
s1 = ['apple', 'banana', 'kiwi', 'mango']
s2 = [2, 8, 1, 9]
```

```
store = list(zip(s1, s2))
print(store)
```

Example 2: Count the number of fruits

```
store = [('apple', 2), ('banana', 8), ('kiwi', 1), ('mango', 9)]
```

- using enumerate and for-loop
- using list comprehension

Hint: `sum([2, 8, 1, 9])` returns 20.

Solution 1:

In [ ]:
```
store = [('apple', 2), ('banana', 8), ('kiwi', 1), ('mango', 9)]
n = 0
for fruit, count in store:
    n += count

print(n)
```

In [ ]:
```
n = 0
for fruit, count in zip(s1, s2):
    n += count

print(n)
```

Solution 2:

In [ ]:
```
store = [('apple', 2), ('banana', 8), ('kiwi', 1), ('mango', 9)]
sum([ y for x, y in store])
```

Dictionary

Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair. Creating a dictionary is as simple as placing items inside curly braces `{ }` separated by commas.

An item has a key and a corresponding value that is expressed as a pair or `key: value` . You may use the `dict()` construct.

For example:

```
my_dict = {'apple': 5, 'orange': 10}
ur_dict = {'apple': 2, 'banana': 3, 'kiwi': 1, 'mango': 5}

store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

Example 1: Construct a dictionary from a list.

```
alist = [('apple', 2), ('banana', 8), ('kiwi', 1), ('mango', 9)]

store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
```

In [ ]:
```
alist = [('apple', 2), ('banana', 8), ('kiwi', 1), ('mango', 9)]
store = None
print(store)
```

Example 2: List all keys

```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
alist = ['apple', 'banana', 'kiwi', 'mango']
```

In [ ]:
```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
alist = None
print(alist)
```

Example 3: List all values and its sum

```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
alist = [2, 8, 1, 9]
20
```

In [ ]:
```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
alist = None
print(alist)
```

Example 4: List all `key:value` pairs

```
apple:2, banana:8, kiwi:1, mango:9,
```

Hint:

- Use for-loop and items()

In [ ]:
```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
None
```

Dict Comprehension

A dictionary is a collection of **key/value pairs**. Python has various methods to work in dictionaries.

Example 1. Make a dict from two lists.

```
fruit = ['apple', 'banana', 'kiwi', 'mango']
count = [2, 8, 1, 9]

store = {'apple': 2, 'banana': 8, 'kiwi': 1, {'mango': 9}
```

Hint:

- Use `zip()` to combine fruit and count as a pair.
- Use list comprehension to make a list of those pairs.
- Use `dict()` construct to convert the list to a dict data type

Solution 1: Using for-loop

In [ ]:
```
#converting two lists or sets into one dict type object
fruit = ['apple', 'banana', 'kiwi', 'mango']
count = [2, 8, 1, 9]

None

print(store)
```

Solution 2: Using list-comprehension

In [ ]:
```
fruit = ['apple', 'banana', 'kiwi', 'mango']
count = [2, 8, 1, 9]

None

print(store)
```

Homework

Find colors

Given a list of colors, create a list of colors of which the length is greater than 5.

```
- Use a for loop
- Use a list comprehension.
```

Using for-loop

In [ ]:
```
# Use a for loop
rainbow = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
colors = []


print(colors)
```

Using list comprehension

In [ ]:
```
# Use a list comprehension.

rainbow = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
colors = None
print(colors)
```

Grading

Given a list of grades, count how many are above the average.

1. Get an average
2. Get a list of grades above the average using a for loop
3. Get the number of grades above the average

The output expected:

```
17 grades are above the average 73.74074074074075
[88, 76, 78, 87, 100, 77, 89, 92, 87, 88, 95, 99, 76, 87, 98, 87, 89]
```

Using for loop:

In [ ]:
```
grades = [88,67,45,67,76,78,53,87,12,100,77,89,92,53,55,45,87,88,95,99,76,87,56,45,98,
avg = None
alist = []




print(len(alist), "grades are above the average", avg)
print(alist)
```

Using list comprehension

In [ ]:
```
avg = None
alist = None

print(len(alist), "grades are above the average", avg)
print(alist)
```

Given a three-digit number. Find the sum of its digits.

Print the sum of three-digit number which is an integer.

For example, print `10` if `316`, `13` if `256`, `18` if `567`.

Using built-in function `sum()`, `int()`, and list comprehension, we can just do it in one line of code.

- To use built-in `sum()` function.
- `Input()` returns a `str` type data.
- Use list comprehension and get each character.
- Use `sum()` to sum up the element in the list.

In [ ]:
```
# del sum
# Read an integer as a string:
s = None

# for example, print 6 if a is 123, 18 if a is 369.
```

```
total = None
print(total)
```

Create the following dict type data set.

```
data = {'a': 1, 'b': 2, 'c': 3, ..., 'x': 24, 'y': 25, 'z': 26}
```

- You may import the sequence of ASCII lowercase as shown below:

```
from string import ascii_lowercase as lowers
```

Solution 1: Using dict() function and zip().

In [ ]:
```
from string import ascii_lowercase as lowers

data = dict( None )
print(data)
```

Solution 2: Using dict comprehension

In [ ]:
```
data = { None }
print(data)
```

List all keys: value pairs

List all keys without extra comma or space at the end of the list.

```
apple:2, banana:8, kiwi:1, mango:9
```

Hint:

- Use `join()` and list-comprehension like syntax.

In [1]:
```
store = {'apple': 2, 'banana': 8, 'kiwi': 1, 'mango': 9}
print(None)
```

```
apple:2, banana:8, kiwi:1, mango:9
```

Pythagorean triplet

A Pythagorean triplet is a set of three positive integers a, b and c such that $a^2 + b^2 = c^2$. Given a limit, generate all Pythagorean Triples with values smaller than given limit n. Assume that a < b < c.

Pythagorean triplet 1 - Warming-up

피타고라스 정리(Pythagorean theorem)는 직각 삼각형의 빗변의 제곱이 두 직각변의 제곱의 합과 같다라는 것입니다. 삼각형 세 변의 길이가 각각 a, b, c 이며, a < b < c 라고 가정한다면, $a^2 + b^2 = c^2$ 를 만족합니다. 피타고라스 정리의 조건을 만족하는 n 보다 작은 정수 a, b, c를 요소로 하는 list를 출력하십시오.

If n is 10, the following output is expected:

```
3, 4, 5
6, 8, 10
```

If n is 20, the following output is expected:

```
3, 4, 5
5, 12, 13
6, 8, 10
8, 15, 17
9, 12, 15
12, 16, 20
```

Hints and Tips: You may use `sep` option duing print().

In [ ]:
```
n = 20
for a in range(1, n):
    for b in range(a, n):
        for c in range(b, n):
            if a * a + b * b == c * c:
                print(a, b, c, sep = ', ')
```

Pythagorean triplet 2

Write a function, `pythagorean(n)` , that returns a list of Pythagorean triplet lists.

If n is 20, the following output is expected:

```
[[3, 4, 5], [5, 12, 13], [6, 8, 10], [8, 15, 17], [9, 12, 15], [12,
16, 20]]
```

In [ ]:
```
def pythagorean(n):
    None
    for a in None
        for b in None
            for c in None
                if None
                    None
    return None

n = 20
pythagorean(n)
```

Pythagorean triplet 3

Write a function, `pythagorean(n)` , that returns a list of Pythagorean triplet tuples. The **tuple** data type in Python just works like a **list** but it is immutable.

If n is 20, the following output is expected:

```
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (8, 15, 17), (9, 12, 15), (12,
16, 20)]
```

In [ ]:
```
def pythagorean(n):
    None
    for a in None
        for b in None
            for c in None
                if None
                    None
    return None

n = 20
pythagorean(n)
```

Pythagorean triplet 4

Using a list comprehension, implement `pythagorean(n)`, that returns a list of Pythagorean triplet tuples.

In [ ]:
```
def pythagorean(n):
    return [None]

n = 20
pythagorean(n)
```

Reference for Pythagorean triplet

There are some useful discussions to make this function faster.

- https://stackoverflow.com/questions/575117/generating-unique-ordered-pythagorean-triplets
- https://www.geeksforgeeks.org/generate-pythagorean-triplets/

Key Points to Remember

- List comprehension is an elegant way to define and create lists based on existing lists.
- List comprehension is generally more compact and faster than normal functions and loops for creating list.
- However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.
- Remember, every list comprehension can be rewritten in for loop, but every for loop can't be rewritten in the form of list comprehension.

학습 정리

1. 리스트 컴프리핸션
2. 파이써닉 코딩

참고자료

- Python-List Comprehension

슬기로운 자는 재앙을 보면 숨어 피하여도 어리석은 자들은 나가다가 해를 받느니라. 잠언27:12