DAY-13 List Comprehension

List Comprehension

List comprehension in Python is a compact way of creating a list from a sequence. It is a short way to create a new list. List comprehension is considerably faster than processing a list using the *for* loop.

```
# syntax
[i for i in iterable if expression]
```

Example:1

For instance if you want to change a string to a list of characters. You can use a couple of methods. Let's see some of them:

```
# One way
language = 'Python'
lst = list(language) # changing the string to list
print(type(lst))     # list
print(lst)           # ['P', 'y', 't', 'h', 'o', 'n']

# Second way: list comprehension
lst = [i for i in language]
print(type(lst)) # list
print(lst)       # ['P', 'y', 't', 'h', 'o', 'n']
```

Example:2

For instance if you want to generate a list of numbers

```
# Generating numbers
numbers = [i for i in range(11)]  # to generate numbers from 0
to 10
print(numbers)                    # [0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10]

# It is possible to do mathematical operations during
iteration
squares = [i * i for i in range(11)]
print(squares)                    # [0, 1, 4, 9, 16, 25, 36,
49, 64, 81, 100]

# It is also possible to make a list of tuples
numbers = [(i, i * i) for i in range(11)]
print(numbers)                             # [(0, 0), (1, 1),
(2, 4), (3, 9), (4, 16), (5, 25)]
```

Example:2

List comprehension can be combined with if expression

```
# Generating even numbers
even_numbers = [i for i in range(21) if i % 2 == 0]  # to
generate even numbers list in range 0 to 21
print(even_numbers)                    # [0, 2, 4, 6, 8, 10,
12, 14, 16, 18, 20]

# Generating odd numbers
odd_numbers = [i for i in range(21) if i % 2 != 0]  # to
generate odd numbers in range 0 to 21
print(odd_numbers)                      # [1, 3, 5, 7, 9, 11,
13, 15, 17, 19]
# Filter numbers: let's filter out positive even numbers from
the list below
numbers = [-8, -7, -3, -1, 0, 1, 3, 4, 5, 7, 6, 8, 10]
positive_even_numbers = [i for i in numbers if i % 2 == 0 and
i > 0]
print(positive_even_numbers)                    # [2, 4, 6, 8,
10, 12, 14, 16, 18, 20]

# Flattening a three dimensional array
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened_list = [ number for row in list_of_lists for number
in row]
print(flattened_list)    # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lambda Function

Lambda function is a small anonymous function without a name. It can take any number of arguments, but can only have one expression. Lambda function is similar to anonymous functions in JavaScript. We need it when we want to write an anonymous function inside another function.

Creating a Lambda Function

To create a lambda function we use *lambda* keyword followed by a parameter(s), followed by an expression. See the syntax and the example below. Lambda function does not use return but it explicitly returns the expression.

```
# syntax
x = lambda param1, param2, param3: param1 + param2 + param2
print(x(arg1, arg2, arg3))
```

Example:

```
# Named function
def add_two_nums(a, b):
    return a + b

print(add_two_nums(2, 3))     # 5
# Lets change the above function to a lambda function
add_two_nums = lambda a, b: a + b
print(add_two_nums(2,3))    # 5

# Self invoking lambda function
(lambda a, b: a + b)(2,3) # 5 - need to encapsulate it in
print() to see the result in the console

square = lambda x : x ** 2
print(square(3))    # 9
cube = lambda x : x ** 3
print(cube(3))    # 27

# Multiple variables
multiple_variable = lambda a, b, c: a ** 2 - 3 * b + 4 * c
print(multiple_variable(5, 5, 3)) # 22
```

Lambda Function Inside Another Function

Using a lambda function inside another function.

```
def power(x):
    return lambda n : x ** n

cube = power(2)(3)   # function power now need 2 arguments to
run, in separate rounded brackets
print(cube)          # 8
two_power_of_five = power(2)(5)
print(two_power_of_five)  # 32
```

🌕 Keep up the good work. Keep the momentum going, the sky is the limit! You have just completed day 13 challenges and you are 13 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 13

1. Filter only negative and zero in the list using list comprehension

```
numbers = [-4, -3, -2, -1, 0, 2, 4, 6]
```

2. Flatten the following list of lists of lists to a one dimensional list :

```
list_of_lists =[[[1, 2, 3]], [[4, 5, 6]], [[7, 8, 9]]]

output
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3. Using list comprehension create the following list of tuples:

```
[(0, 1, 0, 0, 0, 0, 0),
(1, 1, 1, 1, 1, 1, 1),
(2, 1, 2, 4, 8, 16, 32),
(3, 1, 3, 9, 27, 81, 243),
(4, 1, 4, 16, 64, 256, 1024),
(5, 1, 5, 25, 125, 625, 3125),
(6, 1, 6, 36, 216, 1296, 7776),
(7, 1, 7, 49, 343, 2401, 16807),
(8, 1, 8, 64, 512, 4096, 32768),
(9, 1, 9, 81, 729, 6561, 59049),
(10, 1, 10, 100, 1000, 10000, 100000)]
```

4. Flatten the following list to a new list:

```
countries = [[('Finland', 'Helsinki')], [('Sweden',
'Stockholm')], [('Norway', 'Oslo')]]
output:
[['FINLAND','FIN', 'HELSINKI'], ['SWEDEN', 'SWE',
'STOCKHOLM'], ['NORWAY', 'NOR', 'OSLO']]
```

5. Change the following list to a list of dictionaries:

```
countries = [[('Finland', 'Helsinki')], [('Sweden',
'Stockholm')], [('Norway', 'Oslo')]]
output:
[{'country': 'FINLAND', 'city': 'HELSINKI'},
{'country': 'SWEDEN', 'city': 'STOCKHOLM'},
{'country': 'NORWAY', 'city': 'OSLO'}]
```

6. Change the following list of lists to a list of concatenated strings:

```
names = [[('Asabeneh', 'Yetayeh')], [('David', 'Smith')],
[('Donald', 'Trump')], [('Bill', 'Gates')]]
output
['Asabeneh Yetaeyeh', 'David Smith', 'Donald Trump', 'Bill
Gates']
```

7. Write a lambda function which can solve a slope or y-intercept of linear functions.

🎉 CONGRATULATIONS ! 🎉