



Statistical Learning and Computational Finance Lab.  
Department of Industrial Engineering  
<http://slcf.snu.ac.kr>

# 빅데이터 분석 실습 1: 제어문 보충

<https://github.com/SLCFLAB/Data-Science-Python/tree/main/Day%201>

# 흐름 제어

# 조건문 if, elif, else

- 조건을 검사하여 그 조건이 True일 경우 if 블록 내의 코드 수행
- and 나 or 함께 사용하면 왼쪽 부터 오른쪽 순으로 조건 검사

```
In [92]: x = -3  
         if x < 0:  
             print('itW's negative')  
  
         it's negative
```

```
In [93]: if x < 0:  
         print('ItW's negative')  
         elif x == 0:  
             print('Equal to zero')  
         elif 0 < x < 5:  
             print('Positive but smaller than 5')  
         else:  
             print('Positive and larger than or eequal to 5')  
  
         It's negative
```

```
In [94]: a = 5; b = 7
```

```
In [95]: c = 8; d = 4
```

```
In [96]: if a < b or c < d:  
         print('Made it')  
  
         Made it
```

# For 반복문

- 리스트나 튜플 같은 컬렉션이나 이터레이터 순회

- 기본문법

```
for value in collection:  
    # value 를 사용하는 코드
```

- continue 예약어

```
In [97]: sequence = [1, 2, None, 4, None, 5]
```

```
In [98]: total = 0
```

```
In [99]: for value in sequence:  
         if value is None:  
             continue # 건너뛰고 다음 순회로 넘어감  
         total += value
```

```
In [100]: total
```

```
Out[100]: 12
```

# For 반복문

## ■ break 예약어

```
In [101]: sequence = [1, 2, 0, 4, 6, 5, 2, 1]
          total_until_5 = 0
          for value in sequence:
              if value == 5:
                  break # for 문을 빠져나감
              total_until_5 += value
```

```
In [102]: total_until_5
```

```
Out[102]: 13
```

- 컬렉션의 원소나 이터레이터가 튜플이나 리스트 같은 순차적인 자료라면 for 반복문 안에서 여러 개의 변수로 꺼낼 수 있음

```
for a, b, c in iterator:
    # 실행 내용
```

# 내장 순차 자료형 함수

## ■ enumerate

```
i = 0  
  
for value in collection:  
    # value를 가지고 무언가 한다  
    i += 1
```

**enumerate:** 순차 자료형에서 현재 아이템의 색인을 함께 처리하고자 할 때  
사용 (i, value) 튜플을 반환

```
for i, value in enumerate(collection):  
    # i, value를 가지고 무언가 한다
```

순차 자료형에서의 값과 그 위치를 dict에 넘겨주는 예제

```
In [182]: some_list = ['foo', 'bar', 'baz']
```

```
In [183]: mapping = dict( (v, i) for i, v in enumerate(some_list))
```

```
In [184]: mapping
```

```
Out[184]: {'foo': 0, 'bar': 1, 'baz': 2}
```

# in

- 리스트 내포(리스트 컴프리헨션)
  - 파이썬 언어에서 제공하는 기능 중 가장 사랑받는 기능
  - 간결한 표현으로 새로운 리스트 만들 수 있음

리스트 내포 기본 형태

[expr for val in collection if condtion]

반복문으로 구현하면 아래와 같음

```
result = []
for val in collection:
    if condition:
        result.append(expr)
```

문자열 길이가 2 이하인 문자열 제외하고 나머지를 대문자로 바꾸는 예제

```
strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
```

```
[x.upper() for x in strings if len(x) > 2]
```

```
['BAT', 'CAR', 'DOVE', 'PYTHON']
```

# in

- 사전과 세트에 대해서도 리스트 내포와 같은 방식 적용 가능

```
dict_comp = {key-expr : value-expr for value in collection if condition}
```

```
set_comp = {expr for val in collection if condition}
```



# in

<리스트 내의 문자열들의 길이가 저장된 세트 생성 예제>

```
In [240]: unique_lengths = {len(x) for x in strings}
```

```
In [241]: unique_lengths
```

```
Out[241]: {1, 2, 3, 4, 6}
```

<리스트에서 문자열의 위치 저장하고 있는 사전 생성 예제>

```
In [242]: loc_mapping = {val : index for index, val in enumerate(strings)}
```

```
In [243]: loc_mapping
```

```
Out[243]: {'a': 0, 'as': 1, 'bat': 2, 'car': 3, 'dove': 4, 'python': 5}
```

```
In [244]: loc_mapping = dict((val, idx) for idx, val in enumerate(strings))
```

```
In [245]: loc_mapping
```

```
Out[245]: {'a': 0, 'as': 1, 'bat': 2, 'car': 3, 'dove': 4, 'python': 5}
```

# in

## 중첩된 리스트 내포

<e가 2개 이상 포함된 이름의 리스트 생성 예제>

```
In [246]: # 남녀의 이름 구분 저장
all_data = [['Tom', 'Billy', 'Jefferson', 'Andrew', 'Wesley', 'Steven'],
            ['Susie', 'Casey', 'Jill', 'Ana', 'Eva', 'Jennifer']]
```

```
In [247]: names_of_interest = []
for names in all_data:
    enough_es = [name for name in names if name.count('e') >= 2]
    names_of_interest.extend(enough_es)
```

```
In [248]: names_of_interest
```

```
Out[248]: ['Jefferson', 'Wesley', 'Steven', 'Jennifer', 'Stephanie']
```

```
In [249]: # 중첩된 리스트 내포 이용
result = [name for names in all_data for name in names
          if name.count('e') >= 2]
```

```
In [250]: result
```

```
Out[250]: ['Jefferson', 'Wesley', 'Steven', 'Jennifer', 'Stephanie']
```

# in

<숫자 튜플이 담긴 리스트를 단순한 리스트로 변환하는 예제>

```
In [251]: some_tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
In [252]: flattened = [x for tup in some_tuples for x in tup]
```

```
In [253]: flattened
```

```
Out[253]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [254]: flattened = []  
for tup in some_tuples:  
    for x in tup:  
        flattened.append(x)
```

```
In [255]: [[x for x in tup] for tup in some_tuples]
```

```
Out[255]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

# While 반복문

- 조건을 명시하고 해당 조건이 False가 되거나 break문을 사용해서 명시적으로 반복을 끝낼 때까지 블록 안의 코드 수행
- Pass: 아무것도 하지 않음을 나타냄

```
In [103]: x = 256
          total = 0
          while x > 0:
              if total > 500:
                  break
              total += x
              x = x // 2
```

```
In [104]: total
```

```
Out [104]: 504
```

```
In [105]: if x < 0:
          print('negative')
          elif x == 0:
              # todo: 나중에 뭔가 추가해야 함
              pass
          else:
              print('positive')

          positive
```

```
In [106]: def f(x, y, z):
          # todo: 이 함수는 구현해야 함
          pass
```

# range

- 연속된 정수의 리스트 생성: range(stop) 또는 range(start, stop[, step])

```
for i in range(10):  
    print(i)
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

```
for i in range(2, 20, 2):  
    print(i)
```

2  
4  
6  
8  
10  
12  
14  
16  
18

range는 주로 반복문에서 색인 리스트로 사용됨

```
seq = [4, 3, 9, 1]  
for i in range(len(seq)):  
    val = seq[i]
```

```
# 0 에서 9999까지 숫자 중 3이나 5의 배수 값의 합  
sum = 0  
for i in range(10000):  
    # % is the modulo operator  
    if i % 3 == 0 or i % 5 == 0:  
        sum += i  
  
sum
```

23331668

# 삼단표현

if-else 블록을 한줄로 표현할 수 있도록 함

```
In [ ]: value = true-expr if condition else false-expr
```

아래의 코드와 동일

```
In [ ]: if condition:
        value = true-expr
else:
        value = false-expr
```

```
In [112]: x = 5
          'Non-negative' if x >= 0 else 'Negative'
```

```
Out[112]: 'Non-negative'
```

함수

# 함수

## ■ 코드를 재사용하고 조직화하기 위한 가장 중요한 수단

- def 예약어로 정의
- return 예약어 사용해서 값을 반환
- return 문이 몇 개가 되든 상관없음
- 함수 블록이 끝날 때까지 return 문이 없다면 None이 반환

```
def 함수이름(매개변수):  
    수행할 문장1  
    수행할 문장2  
    ...
```

```
In [256]: def my_function(x, y, z=1.5):  
           if z > 1:  
               return z * (x + y)  
           else:  
               return z / (x + y)
```

함수호출

```
In [257]: my_function(5, 6, z = 0.7)
```

```
Out[257]: 0.06363636363636363
```

```
In [258]: my_function(3.14, 7, 3.5)
```

```
Out[258]: 35.49
```



# 입력값과 결괏값에 따른 함수의 형태

## ■ 일반적인 함수

```
▶ def add(a, b):  
    result = a + b  
    return result
```

```
▶ a = add(3, 4)  
print(a)
```

7



```
def 함수이름(매개변수):  
    수행할 문장1  
    수행할 문장2  
    ...  
    return 결괏값
```

## ■ 사용방법

결괏값을 받을 변수 = 함수이름(입력인수 1, 입력인수 2, ...)

# 입력값과 결괏값에 따른 함수의 형태

## ■ 입력값이 없는 함수

```
▶ def say():  
    return('Hi')
```

```
▶ a = say()  
print(a)
```

Hi



```
def 함수이름():  
    수행할 문장1  
    수행할 문장2  
    ...  
    return 결괏값
```

## ■ 사용방법

결괏값을 받을 변수 = 함수이름()

# 입력값과 결괏값에 따른 함수의 형태

## ■ 결괏값이 없는 함수

```
▶ def add(a, b):  
    print("%d , %d의 합은 %d입니다." %(a, b, a+b))
```

```
▶ add(3, 4)
```

3 , 4의 합은 7입니다.

```
▶ a = add(3, 4)
```

3 , 4의 합은 7입니다.

```
▶ print(a)
```

None



```
def 함수이름(매개변수):  
    수행할 문장1  
    수행할 문장2  
    ...
```

## ■ 사용방법:    함수이름(입력인수 1, 입력인수 2, ...)

# 입력값과 결괏값에 따른 함수의 형태

## ■ 입력값도 결괏값도 없는 함수

```
▶ def say():  
    print("Hi")
```

```
▶ say()
```

Hi



```
def 함수이름():  
    수행할 문장1  
    수행할 문장2  
    ...
```

## ■ 사용방법:    함수이름()

# 함수 안에서 선언한 변수의 효력 범위

## ■ 네임스페이스, 스코프, 지역 함수

- 함수는 **전역**과 **지역**, 두가지 스코프(scope)에서 변수 참조
- 함수 내부에서 선언된 변수는 기본적으로 모두 지역 네임스페이스(함수 안)에 속함
- 지역 네임스페이스는 함수가 호출될 때 생성되며 함수의 인자를 통해 즉시 생성됨
- 함수의 실행이 끝나면 지역 네임스페이스는 사라짐

```
In [345]: def func():  
          a = []  
          for i in range(5):  
              a.append(i)
```

```
In [346]: a = []  
          func()
```

```
In [347]: a
```

```
Out[347]: []
```

```
In [354]: a = []  
          def func():  
              for i in range(5):  
                  a.append(i)
```

```
In [355]: func()
```

```
In [356]: a
```

```
Out[356]: [0, 1, 2, 3, 4]
```

# 함수 안에서 선언한 변수의 효력 범위

- global 예약어

: 함수 안에서 전역 변수에 값을 대입하려면 그 변수는 global 예약어를 통해 전역 변수로 선언

```
In [265]: a = None
```

```
In [266]: def bind_a_variable():  
            global a  
            a = []  
  
            bind_a_variable()
```

```
In [267]: a
```

```
Out[267]: []
```

# 여러 값 반환

## ■ 여러 값 반환하기

```
In [269]: def f():  
           a = 5  
           b = 6  
           c = 7  
           return a, b, c  
  
a, b, c = f()
```

```
In [270]: a, b, c
```

```
Out[270]: (5, 6, 7)
```

```
In [271]: return_value = f()
```

```
In [272]: return_value
```

```
Out[272]: (5, 6, 7)
```

```
In [273]: def f():  
           a = 5  
           b = 6  
           c = 7  
           return {'a' : a, 'b' : b, 'c' : c}
```

```
In [274]: return_value = f()
```

```
In [275]: return_value
```

```
Out[275]: {'a': 5, 'b': 6, 'c': 7}
```

# Lambda 함수

## ■ lambda 함수

- lambda는 함수를 생성할 때 사용하는 예약어
- def와 동일한 역할을 함
- 보통 한줄로 간결하게 함수를 만들 때 사용
- 사용법

lambda 매개변수1, 매개변수2, ...: 매개변수를 사용한 표현식

```
▶ add = lambda a, b: a+b  
result = add(3,4)  
print(result)
```

7



```
▶ def add(a, b):  
    return a+b  
  
result = add(3,4)  
print(result)
```

7