


# Python

담당 : 이도연

교육기간 : 2019.5.18~19.6.8

# > > lambda

- Python 의 함수 생성 방법은 def와 lambda 두가지 있음
- Lambda 는 익명(anonymous) 함수이다.
- 한번 사용할 간단한 함수인 경우 사용
- 람다 함수 내부의 로직은 한 줄로 밖에 작성할 수 없습니다.
- 형식>  
**lambda** 인자 : 표현식  
( lambda [매개변수]: 리턴값을 포함한 알고리즘 )( [매개변수 값] )
- 예)  
**def** hap(x, y):  
    **return** x + y  
hap(10, 20)  
  
(**lambda** x, y: x + y)(10, 20)

[https://www.w3schools.com/python/python\\_lambda.asp](https://www.w3schools.com/python/python_lambda.asp)

```
a = (lambda foo: foo * 2)(10)  
print(a) # 20
```

*# 매개변수가 없는 경우*

```
b = (lambda : 10)()  
print(b) # 10
```

## >> Closure(크로저)

- Closure는 first-class 함수를 변수
- 1급 객체(first-class citizen)의 조건  
변수나 데이터에 함수를 할당 할 수 있어야 한다. 함수의 인자로 넘길 수 있어야 한다.  
함수의 리턴 값으로 리턴 할 수 있어야 한다.
- 즉, Python은 함수 자체를 인자(argument)로써 다른 함수에 전달하거나 다른 함수의 결과값으로 리턴 할 수도 있고, 함수를 변수에 할당하거나 데이터 구조 안에 저장 할 수 있으므로 Python의 함수는 일급객체이다.
- Java나 C는 함수(method)의 인자로 함수를 넘길 수 없으므로 First-class citizen이 아니다. (이급객체)

```
1  def outerFunc():
2      msg = "Hello"
3
4      def innerFunc():
5          | print(msg)
6
7      return innerFunc
8
9  hi = outerFunc()
10 hi()
```

innerFunc()와 같은  
함수를 클로저(closure)  
함수라고 한다.

클로저(closure)  
함수는 outerFunc()함수  
가 종료되더라도,  
지역변수를 기억하고  
있다.

```

import matplotlib.pyplot as plt
# %matplotlib inline

def shape(shape):
    color = 'yellow'

    def drawCircle(radius):
        plt.gca().add_patch(plt.Circle((0,0), radius=radius, fc=color))
        plt.axis('scaled')

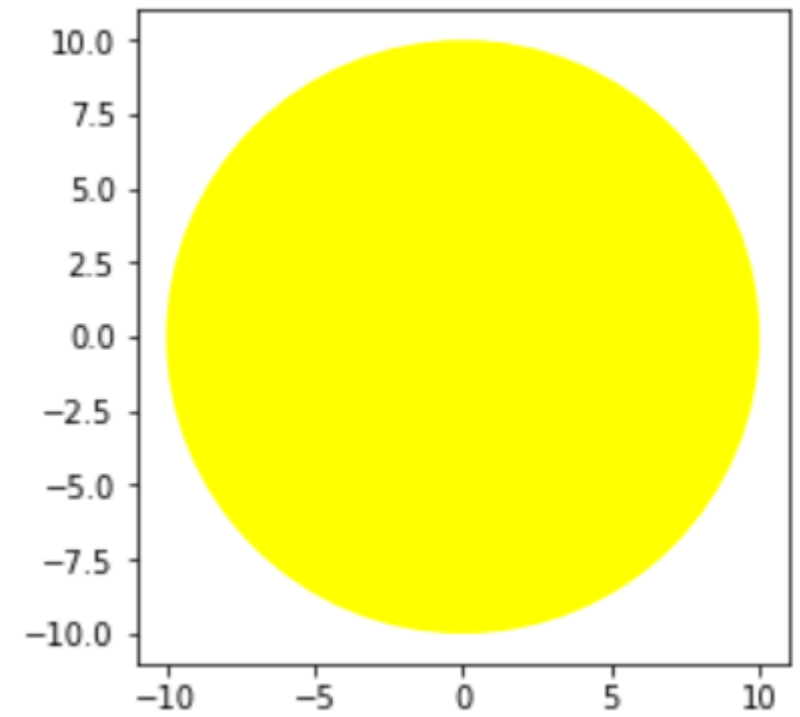
    def drawRect(width, height):
        plt.gca().add_patch(plt.Rectangle((0,0), width=width, height=height, fc=color))
        plt.axis('scaled')

    if shape == 'circle':
        return drawCircle
    elif shape == 'rect':
        return drawRect
    else:
        print("Invalid parameter")

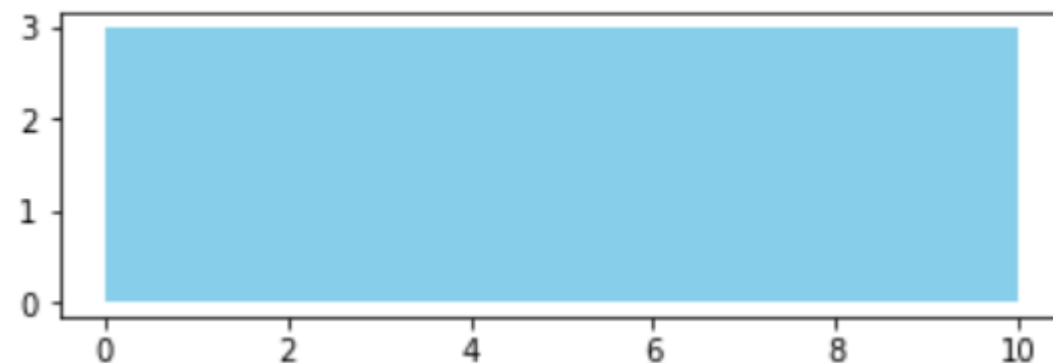
circle = shape('circle')
rect = shape('rect')

```

circle(10)



rect(10, 3)



## > > Decorator

- Closure는 외부 변수(free variable)를 내부함수(inner function)로 전달하여 기억하게 하는 것이고, decorator는 함수를 내부함수로 전달하여 기억하게 하는 것이다. 여기서 전달하는 함수를 original function 이라고 하고, 내부 함수를 wrapper function 이라고 한다.
- 따라서 decorator 역시 함수를 parameter로 전달 받고 반환할 수 있는 First-class 객체 language에서만 구현 가능하다.
- 목적 : 하나의 decorator함수를 만들고 wrapper 함수에 변화를 줌으로서 parameter로 받는 여러 함수들에 동작을 쉽게 추가

```
def decorator_function(original_function):  
    def wrapper_function(*args, **kwargs):  
        print("{} 함수가 실행되었습니다.".format(original_function.__name__))  
        for arg in args:  
            print(arg)  
        return original_function(*args, **kwargs)  
    return wrapper_function
```

```
@decorator_function  
def display():  
    pass
```

```
display()
```

display 함수가 실행되었습니다.

```
@decorator_function  
def display():  
    print("web programming 에서 많이 사용합니다.")
```

```
display()
```

display 함수가 실행되었습니다.  
web programming 에서 많이 사용합니다.

```
@decorator_function  
def display_info(name, age):  
    print("web server programming(flask, django) 에서 많이 사용합니다.")
```

```
display_info('John', 50)
```

display\_info 함수가 실행되었습니다.

John

50

web server programming(flask, django) 에서 많이 사용합니다.



## >> List Comprehension(리스트 컴프리헨션)

- List\_variable =[x for x in iterable]과 같이 list 내에서 for loop 표현식을 사용하여 쉽게 새로운 list를 생성할 수 있다
- List comprehension을 사용하면 for loop 과 map, filter함수를 대체할 수 있다.
- List comprehension 내에서 if~else 를 사용 할 수 있다

```
nums = [1,2,3,4]
squares=[ n ** 2 for n in nums]
print(squares)
```

[1, 4, 9, 16]

```
strs =['hello', 'and', 'goodbye']
shouting =[s.upper() + '!!!' for s in strs]
print(shouting)
```

['HELLO!!!', 'AND!!!', 'GOODBYE!!!']

```
nums = [2, 8, 1, 6]
small = [n for n in nums if n <= 2]
small
```

[2, 1]

```
a = [x for x in range(1, 21)]
a
```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

```
# a가 포함된 과일명을 대문자로 바꾸기
fruits = ['apple', 'banana', 'cherry', 'lemon']
[fruit.upper() for fruit in fruits if 'a' in fruit]
```

```
['APPLE', 'BANANA']
```

```
a = range(1, 21)
list(a)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

```
# 1~50 까지의 홀수 list 작성
b = [x for x in range(1, 51) if x % 2 == 1]
print(b)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49]
```

```
%timeit [x for x in range(1, 51) if x % 2 == 1]
```

```
2.64 µs ± 9.5 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
# map + lambda
list(map(lambda x: x*x, [1,2,3,4,5]))
```

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

```
[x*x for x in [1,2,3,4,5]]
```

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

```
# filter + lambda
```

```
list(filter(lambda x: x > 5, [1,2,3,4,5,6,7,8,9]))
```

```
[6, 7, 8, 9]
```

```
[x for x in [1,2,3,4,5,6,7,8,9] if x > 5]
```

```
[6, 7, 8, 9]
```

```
[x if x > 5 else x * 2 for x in [1,2,3,4,5,6,7,8,9]]
```

```
[2, 4, 6, 8, 10, 6, 7, 8, 9]
```

> > Algorithm – stack, queue, recursive

# > Factorial

```
1 n=0
2 m=1
3 for i in range(1, n+1):
4     m *= i
5 print(m)
6
7 print("\n2. recursive method =====")
8 def factorial(n):
9     if n == 1:
10         return 1
11     return n * factorial(n-1)
12
13 print(factorial(10))
```

```
15 print("\n3. =====")
16 def flatten_list(alist, result=None):
17     if result is None:
18         result = []
19     for a in alist:
20         if isinstance(a, list):
21             flatten_list(a, result)
22         else:
23             result.append(a)
24     return result
25
26 print(flatten_list([[1,2,[3,4]], [5, 6], 7]))
```

## > 정렬(Sorting)

1. Bubble Sort  
list를 처음부터 반복처리하며 인접 item과 비교하여 위치 교환. Iteration 마다 가장 큰 숫자가 마지막 element로 위치  
<https://thrillfighter.tistory.com/209>
2. Selection Sort  
bubble sort algorithm 을 개선. 뒤에서부터 index 위치를 거꾸로 내려가며 maxIndex를 찾아서 한번만 swap한다.  
<https://thrillfighter.tistory.com/210>
3. Merge Sort  
Divide and Conquer (분할정복) 재귀 알고리즘  
<https://thrillfighter.tistory.com/315>

```
1  def bubble_sort(arr):
2      for i in range(len(arr)):
3          for j in range(len(arr)-1-i):
4              if arr[j] > arr[j+1]:
5                  arr[j], arr[j+1] = arr[j+1], arr[j]
6      return arr
7
8  print(bubble_sort([100, 4, 1, 3, 200, 5,3,4,1,2]))
9
10 def selection_sort(arr):
11     for i in range(len(arr)-1, 0, -1):
12         maxIndex = 0
13         for j in range(i+1):
14             if arr[j] > arr[maxIndex]:
15                 maxIndex = j
16         arr[i], arr[maxIndex] = arr[maxIndex], arr[i]
17     return arr
18
19 print(selection_sort([100, 4, 1, 3, 200, 5,3,4,1,2]))
```



```
21     def mergeSort(arr):
22         if len(arr) == 1:
23             return arr
24         a = arr[:int(len(arr)/2)]
25         b = arr[int(len(arr)/2):]
26
27         a = mergeSort(a)
28         b = mergeSort(b)
29         c = []
30         i = 0
31         j = 0
32
33         while i < len(a) and j < len(b):
34             if a[i] < b[j]:
35                 c.append(a[i])
36                 i += 1
37             else:
38                 c.append(b[j])
39                 j += 1
40
41         c += a[i:]
42         c += b[j:]
43         return c
44
45     print(mergeSort([100, 4, 1, 3, 200, 5,3,4,1,2]))
```

# > split, sort

```
1 lorem = """
2 대법원장은 국회의 동의를 얻어 대통령이 임명한다. 헌법재판소는 법률에 저촉되지 아니하는 범위안에서 심판에 관한 절차, 내부규율과 사무처리에 관한 규칙을 제
3
4 국군은 국가의 안전보장과 국토방위의 신성한 의무를 수행함을 사명으로 하며, 그 정치적 중립성은 준수된다. 헌법재판소는 법관의 자격을 가진 9인의 재판관으로
5 """
6 word_list = lorem.split()
7 word_list.sort()
8
9 def findword(word):
10     start = 0
11     end = len(word_list)
12     words = word_list
13     search_count = 0
14
15     while(start < end):
16         search_count += 1
17         middle = len(words) // 2
18         if words[middle] == word:
19             return search_count
20         elif words[middle] > word:
21             start, end = 0, middle
22         else:
23             start, end = middle + 1, len(word_list)
24         words = words[start:end]
25
26 print(findword('헌법재판소는'))
```

```
28     max_count = 0
29     for word in word_list:
30         count = findword(word)
31         if count > max_count:
32             max_count = count
33
34     print("max_count = ", max_count)
35
36     total_search_count = 0
37     for word in word_list:
38         count = findword(word)
39         total_search_count += count
40     average_cnt = total_search_count / len(word_list)
41
42     print('average_cnt = {:.2f}'.format(average_cnt))
43
44     import math
45     print('{:.2f}'.format(math.log(len(word_list), 2)))
```

## > Fibonacci

```
1  def fib(n):
2      if n in (1, 2):
3          return 1
4      return fib(n-1) + fib(n-2)
5
6  print(fib(10))
7
8  memo = {}
9
10 def fib2(n):
11     if n in memo:
12         return memo[n]
13
14     if n in (1, 2):
15         memo[n] = 1
16         return 1
17     result = fib2(n-1) + fib2(n-2)
18
19     memo[n] = result
20
21     return result
22
23 print(fib2(64))
```

# >> Graph Data Structure

- 그래프는 노드(node)와 그 노드를 연결하는 간선(edge)을 하나로 모아 놓은 자료 구조를 말한다.
- 노드(Node)  
정점(vertex)라고도 부르며 위치를 나타낸다
- 간선(Edge)  
위치간의 연결선을 나타낸다. 일방향 혹은 양방향일 수 있다
- 가중치(weight)  
두 node 사이를 이동하는 비용(cost)를 의미한다. 예를 들어 두 개의 도시(node)를 연결하는 길의 가중치는 두 도시 사이의 거리이다.
- 그래프(Graph)  
그래프는  $G = (V, E)$ 로 표시할 수 있고, 각 edge는 연결되는 node와 weight의 tuple로 표시한다.

```
1  class Graph:
2      def __init__(self):
3          self.adjacentList = {}
4
5      def __iter__(self):
6          return iter(self.adjacentList.items())
7
8      def add_vertex(self, vertex):
9          if not vertex in self.adjacentList:
10             self.adjacentList[vertex] = []
11
12      def add_edge(self, v1, v2, weight):
13          self.adjacentList[v1].append(v2)
14          self.adjacentList[v2].append(v1)
15
16  g = Graph()
17  g.add_vertex('a')
18  g.add_vertex('b')
19  g.add_vertex('c')
20  g.add_vertex('d')
21  g.add_vertex('e')
22  g.add_vertex('f')
```

```
24     g.add_edge('a','b',7)
25     g.add_edge('a','c',9)
26     g.add_edge('a','f',14)
27     g.add_edge('b','c',10)
28     g.add_edge('b','d',15)
29     g.add_edge('c','d',11)
30     g.add_edge('c','f',2)
31     g.add_edge('d','e',6)
32     g.add_edge('e','f',9)
33
34     for node in g:
35         print(node)
```

```
1  """
2  >>도전문제
3  피보나치 수열(Fibonacci Sequence)을 계산하는 프로그램도 파이썬으로 간단히 작성할 수 있다.
4  피보나치 수열은 0 과 1 로 시작하고 다음의 숫자는 이전 숫자 두개를 더한 숫자들로 이루어 진다.
5  0, 1, 1, 2, 3, 5, 8, 13 .....
6  n 개의 숫자로 이루어진 피보나치 수열을 출력하는 함수를 작성하라.
7  """
8  def fibonacci_func(n):
9      old, new = 0, 1
10     for i in range(n):
11         old, new = new, old + new
12     return old
13
14     listx = []
15     for i in range(10):
16         listx.append(fibonacci_func(i))
17     print(listx)
```



5) 임의의 범위의 숫자를 모두 곱하는 함수를 작성하라.

ex) multiply(2,4) ==> 2 \* 3 \* 4 = 24

"""

```
def multiply(x, y):
```

```
    result = 1
```

```
    for i in range(x, y+1):
```

```
        result = result * i
```

```
    return result
```

```
print(multiply(2, 4))
```

```
1 """
```

6) 숫자로 이루어진 list 의 평균을 구하는 함수를 작성하라.

단, built-in 함수를 이용하지 않고 for loop 을 이용하여 새로운 함수 작성.

```
4 """
```

```
5 def average(listx):
```

```
6     sum = 0
```

```
7     for n in listx:
```

```
8         sum += n
```

```
9     return sum / len(listx)
```

```
11 print(average([2,3,4,5,6]))
```

## > > Inheritance (상속) 문제

7) class 구현하기

```
class Emp:
```

```
    이름, 부서, 직급, 연락처, 사원번호
```

```
class Regular(Emp):
```

```
    급여
```

```
class Sales(Regular):
```

```
    커미션
```

```
class Employee:
    result =0
    def __init__(self, empNo='0', name='noname', depart='', position='사원', phone=''):
        self.name = name
        self.depart = depart
        self.position = position
        self.phone = phone
        self.empNo = empNo

    def userInput(self):
        self.empNo = input("empno : ")
        self.name = input("name : ")
        self.depart = input("depart : ")
        self.position = input("position : ")
        self.phone = input("phone : ")

    def output(self, empNo='0', name='noname', depart='', position='사원', phone=''):
        # print("output result : ")
        print("{} {} {} {} {}".format(self.empNo, self.name, self.depart, self.position, self.phone), end=' ')
```

```
class Regular(Employee):

    def pays(self):
        pay = int(input('pay : '))
        return pay

    def output(self):
        p = Regular.pays(self)
        print('\n >>> 출력 결과 : ')
        super(Regular, self).output()
        print('{:,.}'.format(p))
```

```
class Salse(Regular):
    def commision(self):
        self.commi = float(input('commi : '))
        return self.commi

    def calc(self):
        pay = super(Salse, self).pays()
        com = Salse.commision(self)
        re = pay + (pay * com)
        return re

    def output(self):
        cal = Salse.calc(self)
        print('\n >>> 출력 결과 : ')
        super(Regular, self).output()
        print('{:.2f}'.format(cal))
```

```
salse = Salse()
salse.userInput()
# salse.calc()
salse.output()

# regular = Regular()
# regular.userInput()
# regular.output()
```

```
...

emp = Employee()
emp.output()
print("\n=====")

emp1 = Employee('doyeon', 'insa', '대표', '010-9872-0202', '1')
emp1.userInput()
emp1.output()
print("\n=====")

emp2 = Employee()
emp2.userInput()
emp2.output()
...
```