19. Python Functional Programming

19-1. 재귀함수 (Recursive Function)

- 재귀(Recursion) 알고리즘의 3 가지 조건
  1. 재귀함수는 종료조건 (exit condition, terminating condition, base case) 을 포함해야 한다.
  2. base case 를 향하여 자신의 status 를 변경한다.
  3. 한번이상 자기 자신을 호출한다.

재귀 연습 1 - Factorial 함수 작성

$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1$

1. for loop 사용 : 1 ~ n 까지를 순차적으로 곱한다.

2. recursive 방법 사용

In [1]:

```
# 1. for loop 사용

n = 10
nn = 1
for i in range(1, n+1):
    nn *= i
print(nn)
```

3628800

In [2]:

```
# 2. recursive method

def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)

factorial(10)
```

Out[2]:

3628800

재귀 연습 2 - list 의 flatten 화

list 내에 nested list 를 포함하고 있는 경우 단일한 list 로 flatten 시키는 함수를 작성

[[1, 2, [3, 4]], [5, 6], 7] ==> [1, 2, 3, 4, 5, 6, 7]

(recursive 순서)

1) flatten_list([[1, 2, [3, 4]], [5, 6], 7], flatten_result=None)

2) flatten_list([[1, 2, [3, 4]], [])
   flatten_result : [1, 2]

3) flatten_list([3, 4], [1, 2])
   flatten_result : [1, 2, 3, 4]

4) flatten_list([5, 6], [1, 2, 3, 4])
   flatten_result : [1, 2, 3, 4, 5, 6]

5) flatten_list(7, [1, 2, 3, 4, 5, 6])
   flatten_result : [1, 2, 3, 4, 5, 6, 7]

In [7]:

```
def flatten_list(alist, flatten_result=None):

    if flatten_result is None:       # 처음 시작인 경우 empty list 생성
        flatten_result = []

    for a in alist:
        if isinstance(a, list):
            flatten_list(a, flatten_result)
        else:
            flatten_result.append(a)
            print(flatten_result)

    return flatten_result
```

In [8]:

```
flatten_list([[1, 2, [3, 4]], [5, 6], 7])
```

```
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7]
```

Out[8]:

```
[1, 2, 3, 4, 5, 6, 7]
```

재귀 연습 3 - dictionary 의 flatten 화

nested dictionary 를 포함한 dictionary 를 단일 dictionary 로 flatten. 이 때 nested dictionary 의 key 는 . 로 표시

{'a': 1, 'b': {'x': 2, 'y': 3}, 'c': 4} ==> {'a': 1, 'b.x': 2, 'b.y': 3, 'c': 4}

(recursive 순서)

1) flatten_dict({'a': 1, 'b': {'x': 2, 'y': 3}, 'c': 4}, parent_key=None, flatten_result=None)
adict.items : [('a', 1), ('b', {'x': 2, 'y': 3}), ('c', 4)]
flatten_result : {'a': 1}

2) flatten_dict(({'x': 2, 'y': 3}, 'b', {'a': 1})
adict.items : [('x', 2), ('y', 3)]
flatten_result : {'a': 1, 'b.x': 2, 'b.y': 3}

3) flatten_result : {'a': 1, 'b.x': 2, 'b.y': 3, 'c': 4}

In [15]:

```
def flatten_dict(adict, parent_key=None, flatten_result=None):

    if flatten_result is None:      # 처음 시작인 경우 empty dictionary 생성
        flatten_result = {}

    #print("adict=", adict)

    for k, v in adict.items():
        if isinstance(v, dict):
            flatten_dict(v, k, flatten_result)
        else:
            if parent_key:
                flatten_result[parent_key + '.' + k] = v
            else:
                flatten_result[k]= v
                #print(flatten_result)

    return flatten_result
```

In [16]:

```
flatten_dict({'a': 1, 'b': {'x': 2, 'y': 3}, 'c': 4})
```

Out[16]:

```
{'a': 1, 'b.x': 2, 'b.y': 3, 'c': 4}
```

연습문제 - 문자열을 뒤에서 부터 출력

1. recursive 방법 사용
2. for loop 과 python 의 string 연산 사용
3. pythonic way (한줄 coding) 사용

In [ ]:

```
string = '.다니습있 수 할 을밍래그로프 nohtyP 는나'

# 1. 재귀적 방법
def recursive(s):
    if not s:
        return ''
    else:
        return s[?] + recursive(s[?])      # ? 부분 수정

recursive(string)
```

In [ ]:

```
# 2. for loop 과 string 연산 사용

sum = ''
for s in string:
    # CODE HRE

print(sum)
```

In [ ]:

```
# Pythonic way
string[?]          # ? 부분 수정
```

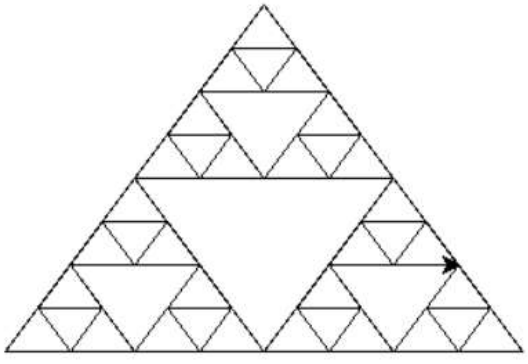
연습문제 : Sierpinksi Triangle

1. 정삼각형의 꼭지점과 recursive 할 횟수를 parameter 로 받는다. ex) ([[0, 100], [-150, -100], [150, -100]], 3)
2. 세 꼭지점을 연결하는 삼각형을 그린다.
3. recursive 수행 횟수 동안 다음을 recursive 하게 반복한다.

    - 꼭지점을 제외한 두 변의 중간점을 구한다.

```
def getMid(p1, p2):
    return ((p1[0] + p2[0])/2, (p1[1] + p2[1])/2)
```

(0, 100)
(-150, -100) (150, -100)



Turtle basic command

```
wn = turtle.Screen()  # canvas생성
t = turtle.Turtle()   # turtle object 생성

t.forward(150)        # 150 unit 전진
t.left(90)          # 90 도 좌회전
t.forward(75)         # 75 unit 전진

for i in range(4):
    t.forward(50)
    t.left(90)

t.penup()
t.left(150)
t.forward(150)
t.pendown()
```

In [18]:

```
import turtle

t = turtle.Turtle()

def drawTurtle(points):       # 삼각형의 세 꼭지점 연결 그리기
    t.penup()
    t.setpos(points[0][0], points[0][1])
    t.pendown()
    t.goto(points[1][0], points[1][1])
    t.goto(points[2][0], points[2][1])
    t.goto(points[0][0], points[0][1])

def getMid(p1, p2):
    return ((p1[0]+p2[0])/2, (p1[1]+p2[1])/2)

def Sierpinski(points, n):
    drawTurtle(points)
    if n > 0:
        Sierpinski([points[0],
                    getMid(points[0], points[1]),
                    getMid(points[0], points[2])], n-1)

        # CODE HERE

        Sierpinski([points[2],
                    getMid(points[2], points[1]),
                    getMid(points[2], points[0])], n-1)

Sierpinski([[0, 100], [-150, -100], [150, -100]], 3)
turtle.done()
```

19-2. lambda

- Python 의 함수 생성 방법은 def 와 lambda 두 가지가 있다.

- lambda 는 익명 (anonymous) 함수 이다.
- 한번 사용할 간단한 함수인 경우 사용

In [19]:

```
def sum(x, y):
    return x + y
```

In [20]:

```
sum(1, 2)
```

Out[20]:

3

In [21]:

```
(lambda x, y: x+y)(1,2)
```

Out[21]:

3

lambda 를 변수에 assign

In [22]:

```
g = lambda x: x**2

print(g(8))
```

64

In [23]:

```
f = lambda x, y: x + y

print(f(4, 5))
```

9

19-3. map, reduce, filter 함수

- map : 각각의 sequence 요소(element)에 대해 순서대로 한번씩 처리하여 새로운 list 를 반환
- filter : sequence 의 element 중 test 를 통과한 element 로 구성된 새로운 list 반환
- reduce : sequence 의 element 들을 왼쪽부터 두개씩 순차적으로 처리하며 누적된 결과가 최종적으로 하나가 되도록 한다.

map

In [24]:

```
def square(x):
    return x*x

list(map(square, [1, 2, 3, 4, 5]))
```

Out[24]:

[1, 4, 9, 16, 25]

In [25]:

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

Out[25]:

[1, 4, 9, 16, 25]

In [26]:

```
a = [1, 2, 3, 4]
b = [17, 12, 11, 10]

list(map(lambda x, y: x+y, a, b))
```

Out[26]:

[18, 14, 14, 14]

filter

filter 를 이용하면 if 문을 함수 안으로 숨길 수 있다.

In [27]:

```
def test(x):
    if x > 5:
        return x
    else:
        return None

list(filter(test, [1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

Out[27]:

[6, 7, 8, 9]

In [28]:

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

Out[28]:

[6, 7, 8, 9]

In [29]:

```
list(filter(lambda x: x.startswith('김'), ['김갑돌', '김성환', '오영제', '한영기']))
```

Out[29]:

['김갑돌', '김성환']

In [30]:

```
foo = [1, 18, 9, 22, 17, 24, 8, 27]

list(filter(lambda x: x%3 == 0, foo))
```

Out[30]:

[18, 9, 24, 27]

reduce

In [31]:

```
from  functools import reduce

def sum(x, y):
    return x+y

reduce(sum, [1, 2, 3, 4, 5])          # ((((1+2)+3)+4)+5)
```

Out[31]:

15

In [32]:

```
reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])      # ((((1+2)+3)+4)+5)
```

Out[32]:

15

In [33]:

```
reduce(lambda x, y: x+y, ['a', 'b', 'c', 'd', 'e'])
```

Out[33]:

'abcde'

19-4. OOP 와의 비교

In [34]:

```
class Sequence:
    def __init__(self, lst):
        self.lst = lst

    def filter(self, thresh):
        rlst = []
        for el in self.lst:
            if el > thresh:
                rlst.append(el)
        return list(rlst)
```

In [35]:

```
seq = Sequence([1, 2, 3, 4, 5, 6, 7, 8, 9])
seq.filter(5)
```

Out[35]:

[6, 7, 8, 9]

19-5. Closure (크로져)

- 퍼스트클래스(1급 객체) 함수를 지원하는 언어의 네임 바인딩 기술
- 어떤 함수를 함수 자신이 가지고 있는 환경과 함께 저장한 레코드
- 자신의 영역 밖에서 호출된 함수의 변수값 (외부변수, free variable) 과 레퍼런스를 복사하고 저장한 뒤, 이 캡처한 값들에 액세스 할 수 있게 함
- 1급 객체 (Frist-class citizen) 의 조건
  - 변수나 데이타에 함수를 할당 할 수 있어야 한다.
  - 함수의 인자로 넘길 수 있어야 한다.
  - 함수의 리턴값으로 리턴 할수 있어야 한다.
- 즉, Python 은 함수 자체를 인자 (argument) 로써 다른 함수에 전달하거나 다른 함수의 결과값으로 리턴 할수도 있고, 함수를 변수에 할당하거나 데이터 구조안에 저장할 수 있으므로 Python 의 함수는 일급객체이다.
- Java 나 C 는 함수(method)의 인자로 함수를 넘길 수 없으므로 Java 나 C 의 함수는 First-class citizen 이 아니다. (이급객체)

하나의 함수를 선언하고, 다른 이름의 변수로 closure 를 저장하면 여러개의 함수를 선언한 효과를 얻음

- ex) multiple 함수를 double, triple, five_times 변수로 저장

In [36]:

```
def multiple(a):                    # 외부함수

    def mult(number):
        return a * number           # 내부함수 mult 가 외부 변수(free variable) a 를 저장하고, numbe

    return mult                     # 내부함수 return
```

In [37]:

```
double = multiple(2)
```

In [38]:

```
double(4)
```

Out[38]:

8

In [39]:

```
triple = multiple(3)

triple(4)
```

Out[39]:

12

In [40]:

```
five_times = multiple(5)

five_times(3)
```

Out[40]:

15

lambda 를 closure 로 사용

In [41]:

```
def inc(n):
    return lambda x: x+n

add2 = inc(2)
print(add2(3))

add4 = inc(4)
print(add4(3))

print(add2(1) + add4(1))
```

```
5
7
8
```

(optional) - Python 이 외부 변수를 저장한 장소

In [42]:

```
1 dir(five_times)
```

Out[42]:

```
['__annotations__',
 '__call__',
 '__class__',
 '__closure__',
 '__code__',
 '__defaults__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__get__',
 '__getattribute__',
 '__globals__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__kwdefaults__',
 '__le__',
 '__lt__',
 '__module__',
 '__name__',
 '__ne__',
 '__new__',
 '__qualname__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

In [46]:

```
1 five_times.__name__
```

Out[46]:

```
'mult'
```

In [47]:

```
1 type(five_times.__closure__)
```

Out[47]:

```
tuple
```

In [48]:

```
1 dir(five_times.__closure__[0])
```

Out[48]:

```
['__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'cell_contents']
```

In [49]:

```
1 five_times.__closure__[0].cell_contents
```

Out[49]:

```
5
```

19-6. decorator

closure 는 외부 변수 (free variable) 을 내부 함수 (inner function)로 전달하여 기억하게 하는 것이고, decorator 는 함수를 내부 함수로 전달하여 기억하게 하는 것이다. 여기서 전달하는 함수를 original function 이라고 하고, 내부 함수를 wrapper function 이라고 한다.

따라서, decorator 역시 함수를 parameter 로 전달 받고 반환할 수 있는 First-class 객체 language 에서만 구현 가능하다.

- 목적 : 하나의 decorator 함수를 만들고 wrapper 함수에 변화를 줌으로서 parameter 로 받는 여러 함수들에 동작을 쉽게 추가

In [50]:

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

In [58]:

```
@decorator_function
def display(msg1):
    print("response complete")
```

In [59]:

```
display("여러 함수에 공통인 기능을 유지 관리하기 편합니다.")
```

display 함수가 실행되었습니다.
여러 함수에 공통인 기능을 유지 관리하기 편합니다.
response complete

In [60]:

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

In [61]:

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

display_info 함수가 실행되었습니다.
John
50
web server program (flask, django) 에서 많이 사용 합니다.

Generator

- yield 문을 사용하여 값 return
- memory 를 효율적으로 사용할 수 있으므로 large data 처리에 유용

In [43]:

```
# 일반적 함수 -> 한번에 결과 return

def fibs(n):
    result = []
    a = 1
    b = 1
    for i in range(n):
        result.append(a)
        a, b = b, a + b
    return result
```

In [44]:

```
print(fibs(30))
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6
765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832
040]
```

In [45]:

```
# generator 함수 ->

def fibs2(n):
    a = 1
    b = 1
    for i in range(n):
        yield a                # yield 문 안의 표현식을 반환하고, 실행 일시 중단
        a, b = b, a + b
```

In [46]:

```
fib = fibs2(30)            # generator object 반환
fib
```

Out[46]:

```
<generator object fibs2 at 0x0000029B15AC1048>
```

In [47]:

```
next(fib)
```

Out[47]:

```
1
```

In [48]:

```
for _ in range(3):
    print(next(fib))
```

```
1
2
3
```

In [49]:

```
print(list(fib))
```

[5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 1094
6, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040]

First Class Citizen (일급시민)

- 변수에 담을 수 있다
- 인자 (parameter) 로 전달할 수 잇다
- retrun 값으로 반환할 수 있다

변수에 할당

In [50]:

```
def first_class(a):
    print(a)

val = first_class
val(123)
```

123

리스트의 element

In [51]:

```
def plus(a, b):
    return a+b

def minus(a, b):
    return a - b

list = [plus, minus]

a = list[0](1, 2)
b = list[1](1, 2)

print(a + b)
```

2

다른 함수의 parameter

In [52]:

```
def love():
    return "I love"

def bye():
    return "Good bye"

def send(s, func1, func2):
    print(func1(), s, func2())

send('you', love, bye)
```

I love you Good bye

연습문제

- lambda 를 이용하여 test_list 의 각 문장이 몇개의 단어로 이루어져 있는지 한줄 coding

test_list = ['this is a book', 'good morning', 'apple', 'apple orange pear', 'hello python and functional programmiong']

In [63]:

```
test_list = ['this is a book', 'good morning', 'apple', 'apple orange pear', 'hello python and func

for s in test_list:
    print((lambda x: ?)(s.split(' ')))
```

4
2
1
3
5