MIT 코스웨어

C6 | 재귀와 딕셔너리

재귀는 자신과 비슷한 형식을 반복하는 것을 뜻한다.
알고리즘 적으로는 어떤 복잡한 문제를 작은 문제로 나누어 문제를 해결하는 divide-and-conquer 형식!
기능적으로는 함수 자기 자신을 call하는 프로그래밍적 방법
▶ 재귀가 성립하려면 1) base call 2) recurs. fun.

재귀 함수가 실행되면 각각의 scope/envi.가 생성된다.
실제로 안에서는 별일이 없다가 n이 base case가 도달하면 그때부터 역으로 흘러가 연산을 처리하는 방식!
▶ 재귀가 보기에는 좋지만, 항상 효율적이지는 X

재귀 함수를 통해 해결할 수 있는 문제는:
1) 팩토리얼 : n=1, return n * factorial(n-1)
2) 하노이의 탑: n-1개를 spare로 옮기고 하나를 옮기기!

```
def printMove(fr, to):
    print('move from ' + str(fr) + ' to ' + str(to))

def Towers(n, fr, to, spare):
    if n == 1:
        printMove(fr, to)
    else:
        Towers(n-1, fr, spare, to)
        Towers(1, fr, to, spare)
        Towers(n-1, spare, to, fr)
```

3) 피보나치 수열:
base case는 f(0)=1, f(1)=1
recursive f.는 f(n) = f(n-1)+f(n-2)

```
def fib(x):
    """assumes x an int >= 0
       returns Fibonacci of x"""
    if x == 0 or x == 1:
        return 1
    else:
        return fib(x-1) + fib(x-2)
```

4) 회문 판별 여부 (palindrome)
base case는 단어가 1개 이하인 경우
아니면 양끝과 n-1개로 이루어진 단어가 회문인 경우

```
def isPalindrome(s):
    def toChars(s):
        s = s.lower()
        ans = ''
        for c in s:
            if c in 'abcdefghijklmnopqrstuvwxyz':
                ans = ans + c
        return ans

    def isPal(s):
        if len(s) <= 1:
            return True
        else:
            return s[0] == s[-1] and isPal(s[1:-1])

    return isPal(toChars(s))
```

딕셔너리는 특정 key, value로 이루어진 자료형으로,
하나의 데이터 구조로 indexing이 가능해 다양하게 사용

기본적인 딕셔너리는 mydict = {} 형식으로 생성,
mydict[key] = Value로 index를 할 수 있다.
mydict[new_key] = new_value로 add도 가능하다.
del(mydict[key])로 삭제도 가능하다.
mydict.keys/value로 전체 data 일괄 call도 가능!
(단, 순서는 보장되지 않으며 keys는 중복 0, val는 X)

가사에서 특정 단어의 빈도 찾는 code:

```
def most_common_words(freqs):
    values = freqs.values()
    best = max(values)
    words = []
    for k in freqs:
        if freqs[k] == best:
            words.append(k)
    return (words, best)
```

this is an iterable, so can apply built-in function
can iterate over keys in dictionary

딕셔너리 통해 재귀 함수에서 중복값 저장이 가능하다.
이 경우 중복 계산을 줄여 코드의 속도가 눈에 띄게 증가!

```
def fib_efficient(n, d):
    if n in d:
        return d[n]
    else:
        ans = fib_efficient(n-1, d) + fib_efficient(n-2, d)
        d[n] = ans
        return ans

d = {1:1, 2:2}
print(fib_efficient(6, d))
```

Method sometimes called " memoization"
Initialize dictionary with base cases