1. [예상 소요 시간 : 10분] [3점]

삼각수(triangular number)는 1, 3, 6, 10, 14, 21, … 로서 다음과 같은 방식으로 구한다.

```
1
1 + 2 = 3
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
1 + 2 + 3 + 4 + 5 + 6 = 21
```

삼각수 구하는 함수를 **재귀함수**, **꼬리재귀함수**, **while문 함수의** 세 가지 형태로 주어진 틀에 맞추어 작성하시오. 0 이하의 인수에 대해서는 모두 0을 내주어야 한다.

2. [예상 소요 시간 : 10분] [3점]

팰린드롬(palindrome)은 거꾸로 읽어도 제대로 읽는 것과 같은 낱말이다. 문자열을 인수로 받아서 팰린드롬인지 확인하는 함수는 다음과 같이 작성하면 잘 작동한다.

```
def is_pal(s):
    if len(s) <= 1:
        return True
    elif s[0] != s[-1]:
        return False
    else:
        return is_pal(s[1:-1])
```

이 함수는 조건문을 사용하는 대신에 논리식 하나만으로 재작성 가능하다. 다음 틀에 맞추어 논리식 하나만으로 동일하게 작동하도록 함수를 재작성하시오.

```
def is_pal(s):
    return ____________________________
```

3. [예상 소요 시간 : 40분] [4+4 = 8점]

날짜는 해와 달과 날로 표현하는데, 서기로 해는 1 이상 정수이고, 달은 1부터 12까지 정수이고, 날은 달에 따라서 1부터 28~31일 까지 정수를 쓴다. 1, 3, 5, 7, 8, 10, 12월은 31일까지 있고, 4, 6, 9, 11월은 30일까지 있고, 2월은 평년에는 28일까지 윤년에는 29일까지 있다. 예를 들어, 2016년은 윤년이므로 2월 29일이 유효하지만, 2017년은 평년이므로 2월 29일은 유효하지 않다. 정수를 인수로 받아서 윤년인지 확인하는 함수 is_leap_year는 다음과 같다.

```
def is_leap_year(year):
    return year % 400 == 0 or year % 4 == 0 and year % 100 != 0
```

그리고 해, 달, 날을 각각 정수 인수로 받아서 유효한 날인지 확인하는 함수 is_valid_date는 다음과 같다.

```
def is_valid_date(year,month,day):
    return year > 0 and 1 <= month <= 12 and \
           (month in [1,3,5,7,8,10,12] and 1 <= day <= 31 or
            month in [4,6,9,11] and 1 <= day <= 30 or
            is_leap_year(year) and 1 <= day <= 29 or
            1 <= day <= 28)
```

A. [4점] 주민등록번호 앞 6자리 숫자문자열은 생년월일을 나타낸다. 첫 두자리는 태어난 해의 뒤 두자리를 나타내고, 다음 두자리는 태어난 달을 나타내고, 다음 두자리는 태어난 날을 나타낸다. 표준입력창에서 주민등록번호 앞 6자리를 입력받아서 유효한 생년월일인지 확인하고 해와 달과 날을 정수 튜플로 내주는 함수 get_valid_date_6를 주어진 틀에 맞추어 작성하시오.

- 첫 두자리 dd는 17이하인 경우는 20dd로 처리하고, 18이상인 경우는 19dd로 처리한다.
- 사용자 입력은 반드시 6자리의 숫자만 허용하도록 입력확인을 반드시 해야 한다.
- 위에 주어진 is_valid_date 함수를 사용한다.
- 유효하지 않은 입력에 대해서는 None을 내준다.

실행하면 다음과 같이 작동해야 한다.

```
$ python3 exam.py
Type 6 digits: 160425
(2016, 4, 25)
$ python3 exam.py
Type 6 digits: 160431
None
$ python3 exam.py
Type 6 digits: 160229
(2016, 2, 29)
$ python3 exam.py
Type 6 digits: 170229
None
```

B. [4점] 이번에는 날짜에 날을 더하는 함수 date_plus를 만들어보자. 즉, 다음과 같이 작동해야 한다.

```
date_plus(2017, 4, 20, 2)) => (2017, 4, 22)
date_plus(2017, 4, 20, 7)) => (2017, 4, 27)
date_plus(2017, 4, 20, 10)) => (2017, 4, 30)
date_plus(2017, 4, 20, 11)) => (2017, 5, 1)
date_plus(2017, 4, 20, 50)) => (2017, 6, 9)
date_plus(2017, 4, 20, 100)) => (2017, 7, 29)
date_plus(2017, 4, 20, 200)) => (2017, 11, 6)
date_plus(2017, 4, 20, 300)) => (2018, 2, 14)
date_plus(2017, 4, 20, 1000)) => (2020, 1, 15)
```

이 함수를 작성하는데 다음 두 함수를 사용하면 편리하다.

```
def next_month(year, month):
    if month == 12:
        year += 1
        month = 1
    else:
        month += 1
    return (year, month)

def days_in_month(year, month):
    if month in [1,3,5,7,8,10,12]:
        return 31
    elif month in [4,6,9,11]:
        return 30
    elif month in [2]:
        if is_leap_year(year):
            return 29
        else:
            return 28
    else:
        return 0
```

4. [예상소요시간: 15분] [3점]

문자열에서 주어진 문자를 모두 지우는 remove 함수를 **재귀함수**, **꼬리재귀함수**, **while문 함수의** 세 가지 형태로 주어진 틀에 맞추어 작성하시오. 예를 들어, 호출 결과는 다음과 같아야 한다.

```
remove('a','abracadabra') => brcdbr
remove('z','abracadabra') => abracadabra
```

즉, 첫째 인수는 제거하고 싶은 문자이고, 둘째 인수는 제거 대상이 되는 문자열이다.

5. [예상소요시간: 20분] [4점]

자연수 n개 원소의 k 순열permutation은 서로 다른 n개의 원소 중에서 k개를 뽑아서 한줄로 세우는 방법으로, 그 가능한 방법의 개수는 $P(n,k)$로 표현하며 다음 식으로 구할 수 있다.

$P(1,1) = 1$

$P(2,1) = 2$

$P(2,2) = 2 * 1 = 2$

$P(3,1) = 3$

$P(3,2) = 3 * 2 = 6$

$P(3,3) = 3 * 2 * 1 = 6$

$P(4,1) = 4$

$P(4,2) = 4 * 3 = 12$

$P(4,3) = 4 * 3 * 2 = 24$

$P(4,4) = 4 * 3 * 2 * 1 = 24$

…

$P(n,k) = n\ (n\text{-}1)\ (n\text{-}2)\ \ldots\ (n - k + 1)$

이 성질을 이용하여 곱셈만으로 순열을 계산하는 함수 permutation을 강의시간에 배운 방식으로 **재귀함수**, **꼬리재귀함수**, **while 문 함수**, **for 문 함수** 차례로 각각 작성하자.

양수 인수만 제대로 처리하면 된다. 즉, 인수가 양수인지는 함수 호출하기 전에 이미 확인했다고 가정한다. 그리고 $n < k$ 인 경우에는 0을 내주도록 해야 한다.

6. **[예상소요시간: 20분] [1+3+1 = 5점]**

순서열의 일부를 덩어리로 만들어 내주는 공통 연산인 s[i:j]를 사용하면, 순서열 s의 위치번호 i부터 위치번호 j까지 연결된 순서열 일부(i 원소 포함, j 원소 제외)를 복사하여 만들어준다. 이 연산을 리스트에 한정지어 작동하도록 직접 함수로 구현해보자. 즉, 작성할 함수 sublist는 리스트와 위치번호 범위를 인수로 받아서, 범위에 해당하는 리스트를 만들어 내주면 된다. sublist(s,low,high)를 호출한 경우, 인수 s는 리스트, low는 범위의 시작 위치번호, high는 끝 위치번호를 나타낸다. 위치번호가 음수로 주어지는 경우 모두 0으로 처리하기로 한다. 예를 들어, 다음과 같이 작동하면 된다.

```
s = [1,2,3,4,5]
sublist(s,0,0) => []
sublist(s,0,1) => [1]
sublist(s,0,2) => [1, 2]
sublist(s,0,3) => [1, 2, 3]
sublist(s,0,4) => [1, 2, 3, 4]
sublist(s,0,5) => [1, 2, 3, 4, 5]
sublist(s,0,6) => [1, 2, 3, 4, 5]
sublist(s,1,1) => []
sublist(s,1,2) => [2]
sublist(s,1,3) => [2, 3]
sublist(s,1,4) => [2, 3, 4]
sublist(s,1,5) => [2, 3, 4, 5]
sublist(s,1,6) => [2, 3, 4, 5]
sublist(s,2,2) => []
sublist(s,2,3) => [3]
sublist(s,2,4) => [3, 4]
sublist(s,2,5) => [3, 4, 5]
sublist(s,2,6) => [3, 4, 5]
sublist(s,3,3) => []
sublist(s,3,4) => [4]
sublist(s,3,5) => [4, 5]
sublist(s,3,6) => [4, 5]
sublist(s,5,2) => []
sublist(s,-3,-2) => []
```

이 작업을 완수하기 위해서 앞 부분 떼어내는 작업과 뒷 부분 떼어내는 작업을 분리하여 따로 함수를 만든 다음, sublist 함수는 이 두 함수를 잇따라 호출하여 작업을 수행할 수 있도록 하자.

A. [1점] 앞 부분을 떼어내는 함수를 먼저 만들자. 리스트 s와 위치번호 index를 인수로 받아서 index 전에 있는 원소를 제외한 나머지 리스트를 내주는 함수 drop_before는 다음과 같이 작성할 수 있다.

```
def drop_before(s,index):
    if s != [] and index > 0:
        return drop_before(s[1:],index-1)
    else:
        return s
```

이 함수를 다음 실행사례를 보면서 이해한 다음, **<u>while 반복문</u>**을 사용하여 재작성하자.

```
s = [1,2,3,4,5]
drop_before(s,0) => [1, 2, 3, 4, 5]
drop_before(s,1) => [2, 3, 4, 5]
drop_before(s,2) => [3, 4, 5]
drop_before(s,3) => [4, 5]
drop_before(s,4) => [5]
drop_before(s,5) => []
drop_before(s,6) => []
drop_before(s,-3) => [1,2,3,4,5]
drop_before([],4) => []
```

B. [3점] 이번에는 뒷부분을 떼어내는 함수 만들 차례이다. 리스트 s와 위치번호 index를 인수로 받아서 index 뒤에 있는 원소를 제외한 (index 원소도 제외에 포함) 나머지 리스트를 내주는 함수 take_before를 강의시간에 배운 방식으로 **재귀함수**, **꼬리재귀함수**, **while 문 함수** 차례로 각각 작성하자. 이 함수의 실행 결과는 다음과 같아야 한다.

```
s = [1,2,3,4,5]
take_before(s,0) => []
take_before(s,1) => [1]
take_before(s,2) => [1, 2]
take_before(s,3) => [1, 2, 3]
take_before(s,4) => [1, 2, 3, 4]
take_before(s,5) => [1, 2, 3, 4, 5]
take_before(s,6) => [1, 2, 3, 4, 5]
take_before([],4) => []
take_before(s,-3) => []
```

C. [1점] 위에서 작성한 drop_before와 take_before를 사용하여 다음 형식에 맞추어 sublist 함수를 완성하자.

```
def sublist(s,low,high):
    if low < 0: low = 0
    if high < 0: high = 0
    if low <= high:
        return ________________________
    else:
        return []
```

7. [예상소요시간: 20분] [2 + 2 = 4점]

[4점] 아래 longest_streak 함수는 임의의 숫자열(숫자문자열)을 인수로 받아서 연속 이어지는 숫자(영어로 streak이라고 함)가 가장 긴 streak을 찾아서, 그 숫자와 길이를 내주는 함수이다. 예를 들어, 다음과 같이 작동한다.

```
longest_streak0("0647901995590720004118500878052838481126567811671") => ('0', 3)
longest_streak0("4971511425086345555901320722839515498488256083467 4") => ('5', 4)
longest_streak0("7908378726215981563883404228248519527083693748809 7") => ('8', 2)
longest_streak0("36888653851748777011129000999371447120618209984726") => ('8', 3)
```

가장 긴 streak의 숫자와 연속된 횟수의 튜플을 내준다. 여러 개가 동률인 경우 가장 먼저 나온 streak을 선택한다. 빈 숫자열("")은 함수 호출 전에 이미 확인했다고 가정하고 처리하지 않기로 한다. 이 함수는 다음과 같이 구현하였다. 위와 같이 작동되는지 주어진 테스트 사례를 가지고 직접 실행하여 확인하자.

```
def longest_streak0(s):
    contender = leader = s[0]
    streak_length = streak_record = 1
    for n in s[1:]:
        if n == contender:
            streak_length += 1
        else:
            contender = n
            streak_length = 1
        if streak_length > streak_record:
            leader = contender
            streak_record = streak_length
    return leader, streak_record
```

A. 가장 긴 streak 숫자와 연속 횟수 뿐 아니라, 찾은 streak의 시작 위치번호도 함께 내주도록 위 코드를 수정한 longest_streak1을 작성하자. 같은 예제로 실행한 결과는 다음과 같아야 한다.

```
longest_streak1("0647901995590720004118500878052838481126567811671") => ('0', 3, 15)
longest_streak1("4971511425086345555901320722839515498488256083467 4") => ('5', 4, 15)
longest_streak1("7908378726215981563883404228248519527083693748809 7") => ('8', 2, 19)
longest_streak1("36888653851748777011129000999371447120618209984726") => ('8', 3, 2)
```

결과 튜플의 마지막 원소는 해당 streak의 시작 위치번호이다.

B. 위에서는 가장 긴 streak 중에서 가장 먼저 나오는 streak 만 찾아주었다. 가장 긴 streak 모두를 찾아서 리스트로 모아서 내주도록 수정한 longest_streak2 함수를 작성하자. 위와 같은 예제로 실행한 결과는 다음과 같아야 한다

```
longest_streak2("0647901995590720004118500878052838481126567811671")
=> [('0', 3, 15), ('1', 3, 44)]
longest_streak2("4971511425086345555901320722839515498488256083467 4")
=> [('5', 4, 15)]
longest_streak2("7908378726215981563883404228248519527083693748809 7")
=> [('8', 2, 19), ('2', 2, 25), ('8', 2, 45)]
longest_streak2("36888653851748777011129000999371447120618209984726")
=> [('8', 3, 2), ('7', 3, 14), ('1', 3, 18), ('0', 3, 23), ('9', 3, 26)]
```