반 : 금 / 화	학번 :	이름 :

- 반, 학번, 이름 기입을 잊지 맙시다.
- 뼈대 코드와 테스트 케이스가 담겨있는 exam2.py 파일을 다운받아 코드를 작성합니다.
- 뼈대 코드에는 코딩을 도와주는 힌트가 담겨있습니다. 하지만 꼭 뼈대코드에 맞추어 작성할 필요는 없습니다.
- 파일 이름을 exam2-nnnn.py로 수정하여 제출합니다. 여기서 nnnn은 자신의 학번의 뒤 4자리입니다.

문제	1	2	3	4	5	6	총계
배점	2	2	8	8	6	4	30
득점							

1. [예상 소요 시간 : 총 2문제 약 2분]

[2점] 컴퓨터과학이 여는 세계 / The Innovators

가. OX 문제 [다 맞아야 1점]

(1) 입력의 크기를 n이라고 할 때, 실행비용이 2^n에 비례하는 알고리즘은 쓸 수 없는 비현실적인 알고리즘으로 분류한다.

(2) 람다계산법의 관점에서 보면 컴퓨터 세계에서 언어와 논리는 동전의 양면일 뿐, 같은 것이라고 할 수 있다.

나.OX 문제 [다 맞아야 1점]

(1) 프로그램을 메모리에 저장한 뒤 실행하는 아키텍처를 von Neumann 아키텍처라고 하며, 지금 사용하고 있는 상용컴퓨터는 모두 이 아키텍처를 기반으로 하고 있다.

(2) Wikipedia는 오랜 역사와 전통을 자랑하는 백과사전인 Britannica의 웹 확장판으로 해당 출판사의 재정 지원으로 전 세계의 전문가들이 Crowdsourcing으로 만들었다.

(3) GUI 개념은 Xerox PARC 연구소에서 Alan Kay가 처음 실현했지만, 실제로 상용 컴퓨터에 도입한 사례는 Apple 사의 Macintosh가 최초이다.

2. [예상 소요 시간 : 총 1문제 약 7분]

[2점] 문자 하나로 구성된 문자열 char와 임의의 문자열 string을 인수로 받아서 char가 string에 나오는 횟수를 내주는 함수 occurred_in 함수를 작성하자. 첫 인수가 문자 하나로만 구성되어 있는지 검사할 필요는 없다. 다음 사례와 같이 작동하면 충분하다.

```
occurred_in('p', '') => 0
occurred_in('p', 'I love Python!') => 0
occurred_in('e', 'What happened to your college life?') => 5
```


©도경구(2016)

3. [예상 소요 시간 : 총 4문제 약 36분]

```
def numbers_art(n):
    for i in range(n):
        for j in range(n):
            print(j+1, end=' ')
        print()
```

numbers_art(5)를 실행하면 다음과 같이 실행창에 출력한다.

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

위 함수를 적절히 수정하여 다음과 같이 실행창에 출력하도록 코드를 작성하자.

(1) [2점] numbers_art1(5)를 호출하면 다음과 같이 출력하는 numbers_art1 함수를 작성하자.

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

(2) [2점] numbers_art2(5)를 호출하면 다음과 같이 출력하는 numbers_art2 함수를 작성하자.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

(3) [2점] numbers_art3(5)를 호출하면 다음과 같이 출력하는 numbers_art3 함수를 작성하자.

```
1 2 3 4 5
  2 3 4 5
    3 4 5
      4 5
        5
```

(4) [2점] numbers_art4(5)를 호출하면 다음과 같이 출력하는 numbers_art4 함수를 작성하자.

```
        5
      4 5
    3 4 5
  2 3 4 5
1 2 3 4 5
```


©도경구(2016)

4. [예상소요시간: 총 4문제 약 60분]

```
def is_prime(n):
    if n < 2:
        return False
    else:
        for i in range(2,n):
            if n % i == 0:
                return False
        return True
```

(1) [2점] 소수(prime number)는 1과 자신으로 밖에 나누어 떨어지지 않는 1보다 큰 자연수이다. 위 함수는 인수 n이 소수인지 아닌지 확인하는 함수이다. 실행 결과는 다음과 같다.

```
is_prime(0) => False
is_prime(1) => False
is_prime(2) => True
is_prime(3) => True
is_prime(4) => False
is_prime(5) => True
is_prime(6) => False
```

그런데 이 함수는 실행 효율면으로 개선의 여지가 있다. 예를 들면, 2를 제외한 짝수는 따져볼 필요도 없이 소수가 아닌데, 이 함수에서는 else 부분에 정수범위를 만든다. (비록 바로 2로 나누어짐을 확인하고 False를 내주긴 하지만 말이다.) 게다가 인수가 홀수인 경우, 홀수는 짝수로 나누어지는 경우는 없으므로 짝수로 나누는 시도는 할 필요조차 없는데, 여기서는 홀수와 짝수를 막론하고 모두 나누어 검사하여 실행시간을 낭비한다. 인수가 짝수인 경우에는 바로 False를 내주고(2만 예외), 인수가 홀수인 경우에는 홀수범위만 만들어 나누어지는지 검사하도록 위 함수를 수정하여 성능을 개선하자.

(2) [2점] 자연수 n미만의 소수를 2부터 시작하여 모두 오름차순으로 나열한 리스트로 만들어 내주는 함수 primes_less_than을 작성하자. 다음과 같이 작동해야 하며, 위의 is_prime을 호출하면 비교적 쉽게 작성할 수 있다.

```
primes_less_than(2) => []
primes_less_than(3) => [2]
primes_less_than(10) => [2, 3, 5, 7]
primes_less_than(30) => [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

(3) [2점] k개의 소수를 2부터 시작하여 오름차순으로 리스트로 만들어 내주는 함수 primes를 작성하자. 다음과 같이 작동해야 하며, 위의 is_prime을 호출하면 비교적 쉽게 작성할 수 있다.

```
primes(0) => []
primes(1) => [2]
primes(5) => [2, 3, 5, 7, 11]
primes(10) => [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

(4) [2점] 차이가 2인 소수의 쌍을 쌍둥이 소수라고 한다. 즉, (3, 5), (5, 7), (11, 13), (17, 19) 등이 쌍둥이 소수이다. 쌍둥이 소수를 오름차순으로 k쌍 찾아 리스트로 내주는 함수 twin_primes를 작성하자.

```
twin_primes(0) => []
twin_primes(1) => [(3, 5)]
twin_primes(5) => [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31)]
twin_primes(10) => [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61),
(71, 73), (101, 103), (107, 109)]
```


©도경구(2016)

5. **[예상소요시간: 총 3문제 약 45분]**

(1) [2점] 시퀀스에서 순서는 그대로 둔채 임의로 몇 개의 원소를 제거한 리스트를 부분시퀀스(subsequence)라고 한다. 예를 들어, 리스트 시퀀스 [1, 2, 3, 4, 5, 6]에서, 임의로 몇 개의 원소를 제거한 [1, 2, 3, 5, 6], [3, 4, 5], [2, 6]은 부분시퀀스라고 할 수 있다. 그리고 원소를 하나도 제거하지 않은 [1, 2, 3, 4, 5, 6]과 모두 제거해 버린 []도 부분시퀀스로 취급한다. 하지만 순서가 바뀐 [2, 5, 4]는 부분시퀀스라고 할 수 없다.

임의의 리스트 시퀀스를 받아서 가능한 부분시퀀스를 모두 리스트로 모아 다음 예제와 같이 내주는 함수 subsequences를 작성하자.

```
subsequences([]) => [[]]
subsequences([1]) => [[], [1]]
subsequences([1, 2]) => [[], [1], [2], [1, 2]]
subsequences([1, 2, 3]) => [[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
...
```

어떻게 만들지 알고리즘을 생각해보자. 예를 들어, subsequences([1, 2, 3])를 구하려면 맨 앞 원소부터 하나씩 차례로 다음과 같은 방법으로 부분시퀀스를 만들어 subs라는 이름의 변수에 모으면 된다.

A. []는 모든 리스트 시퀀스의 부분시퀀스이므로 먼저 subs를 [[]]로 초기화 한다.
B. 리스트의 맨 앞 원소인 1을 subs의 모든 원소의 뒤에 append()를 사용하여 붙인 다음, subs에 추가한다. 그러면 subs는 [[], [1]]이 된다.
C. 다음 원소인 2를 subs의 모든 원소의 뒤에 append()를 사용하여 붙인 다음, subs에 추가한다. 그러면 subs는 [[], [1], [2], [1, 2]]가 된다.
D. 다음 원소인 3을 subs의 모든 원소의 뒤에 append()를 사용하여 붙인 다음, subs에 추가한다. 그러면 subs는 [[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]이 된다.

(2) [2점] 순수 오르막 시퀀스(strictly increasing sequence)는 자연수 리스트의 일종으로 이어지는 수가 최소한 1 이상 증가하는 리스트이다. 예를 들어, [2, 7, 9, 11, 12, 15]는 이어지는 수가 계속 1 이상 증가하므로 순수 오르막 시퀀스이다. 반면, [2, 7, 7, 11, 12, 15]는 셋째 수가 증가하지 않았으므로 순수 오르막 시퀀스가 아니고, [2, 7, 9, 12, 11, 15]는 다섯째 수가 감소하였으므로 순수 오르막 시퀀스가 아니다. 원소가 1개 이하인 경우는 따지지 않고 순수 오르막 시퀀스로 하기로 한다. 임의의 자연수 리스트를 받아서 순수 오르막 시퀀스인지 검사해주는 increasing 함수를 작성하자. 즉, 인수가 순수 오르막 시퀀스이면 True를 내주고, 그렇지 않으면 False를 내주면 된다. 예를 들면

```
increasing([]) => True
increasing([2]) => True
increasing([1,2]) => True
increasing([2,2]) => False
increasing([3,2]) => False
increasing([1,2,3]) => True
increasing([1,3,2]) => False
increasing([3,2,1]) => False
```


©도경구(2016)

(3) [2점] 임의의 자연수 리스트 시퀀스의 부분시퀀스 중에서 가장 긴 순수 오르막 부분시퀀스의 길이를 내주는 longest_increasing_subsequence 함수를 작성하자. (1)과 (2)에서 작성한 두 함수를 활용하여 작성하면 비교적 간단하게 작성할 수 있다. 실행 사례는 다음과 같다.

```
print(longest_increasing_subsequence([])) => 0
print(longest_increasing_subsequence([3])) => 1
print(longest_increasing_subsequence([5,4])) => 1
print(longest_increasing_subsequence([2,4])) => 2
print(longest_increasing_subsequence([4,3,2])) => 1
print(longest_increasing_subsequence([4,2,7,5,9])) => 3
print(longest_increasing_subsequence([4,2,7,5,4,7,6,8,9,6])) => 5
```

6. [예상소요시간: 총 2문제 약 30분]

십진수는 10을 기수(base)로 0, 1, 2, 3, 4, 5, 6, 7, 8, 9의 10종류의 숫자를 사용하여 수를 표현한다. 예를 들어, 2016은 다음과 같이 해석한다.

```
2 * 10**3 + 0 * 10**2 + 1 * 10**1 + 6 * 10**0
= 2 * 1000 + 0 * 100 + 1 * 10 + 6 * 1
```

이진수는 2를 기수로 0, 1의 2 종류의 숫자를 사용하여 수를 표현한다. 예를 들어, 10011은 다음과 같이 해석한다.

```
1 * 2**4 + 0 * 2**3 + 0 * 2**2 + 1 * 2**1 + 1 * 2**0
```

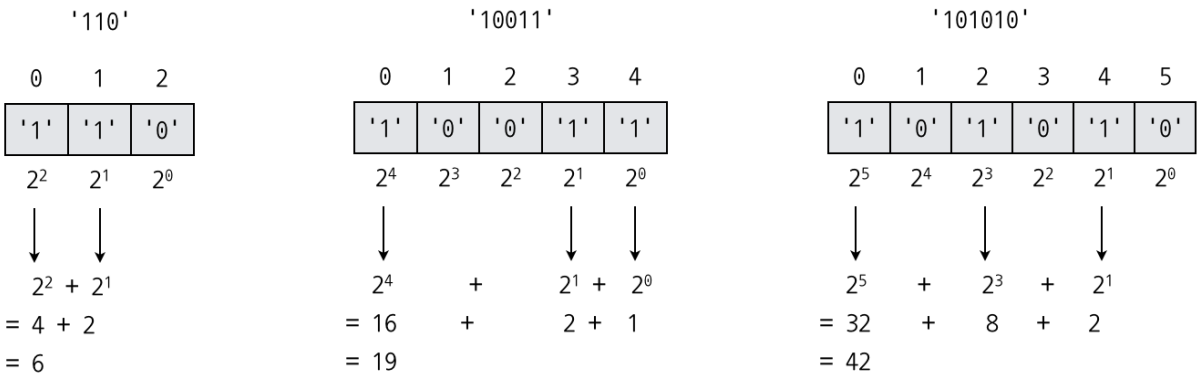
이를 십진수로 바꾸면 다음과 같다.

```
1 * 2**4 + 0 * 2**3 + 0 * 2**2 + 1 * 2**1 + 1 * 2**0
= 1 * 16 + 0 * 8 + 0 * 4 + 1 * 2 + 1 * 1
= 16 + 0 + 0 + 2 + 1
= 19
```

(1) [2점] 이진수 문자열을 인수로 받아서 이를 십진수로 변환해주는 함수 bin2dec을 작성하자. 이진수 인수는 문자열 타입으로 받으며, 변환한 십진수는 정수 타입이다. 실행 사례는 다음과 같다.

```
bin2dec('0') => 0
bin2dec('1') => 1
bin2dec('110') => 6
bin2dec('10011') => 19
bin2dec('101010') => 42
```

다음의 변환 사례를 공부하여 변환 알고리즘을 먼저 이해하고 이 알고리즘 대로 함수를 작성한다.

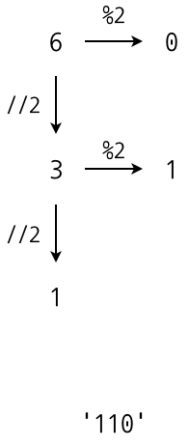


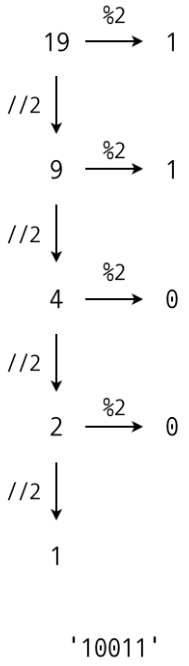
©도경구(2016)

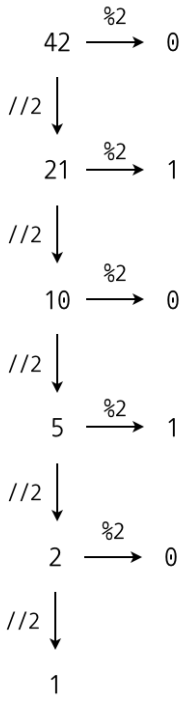
(2) [2점] 십진수를 인수로 받아서 이를 이진수 문자열로 변환해주는 함수 dec2bin을 작성하자. 십진수 인수는 정수 타입으로 받으며, 변환한 이진수는 문자열 타입이다. 실행 사례는 다음과 같다.

```
dec2bin(0) => '0'
dec2bin(1) => '1'
dec2bin(6) => '110'
dec2bin(19) => '10011'
dec2bin(42) => '101010'
```

다음의 변환 사례를 공부하여 변환 알고리즘을 먼저 이해하고 이 알고리즘 대로 함수를 작성한다.







©도경구(2016)