MC4204: Machine Learning in Practice In-Class Problems

#1: Basic Elements of Python

Instructor: Sang-Hyun Yoon

1. Write a function add:

- input parameter: two integers n1, n2
- return value: the sum of n1 and n2

```
def add(n1,n2):
    # ADD ADDITIONAL CODE HERE!

print(add(3,4))  # 7
print(add(3,5))  # 8
```

2. Write a function printAdd:

- input parameter: two integers n1, n2
- return value: 없음
- action: **print** out the sum of n1 and n2
  - return을 쓰면 안되고 대신 print를 사용
  - 함수에서 아무것도 return 하지 않으면 None이 자동으로 return 됨

```
def printAdd(n1,n2):
    # ADD ADDITIONAL CODE HERE!

printAdd(3,4)          # 7
printAdd(3,5)          # 8
print(printAdd(3,4))   # 7 None
```

3. What is the **fewest number** of Korean coins to make 730 Korean won? The answer is 6:

Write a function countCoins:

- input parameter: an integer `n` where $10 \le$ `n` ≤ 990 and `n` is a multiple of 10.
- return value: the fewest number of Korean coins ($10, 50, 100, 500$ won) to make `n` Korean won
  - e.g. `countCoins(730)` returns 6.
  - what is the meaning of `n//500` ?
  - what is the meaning of `n%500` ?

```
def countCoins(n):
    # ADD ADDITIONAL CODE HERE!


print(countCoins(730))    # 6
print(countCoins(790))    # 8
print(countCoins(260))    # 4
print(countCoins(70))     # 3
```

4. Write a function maximum:

- input parameter: two integers `n1, n2`
- return value: the maximum value among `n1` and `n2`

```
def maximum(n1,n2):
    # ADD ADDITIONAL CODE HERE!
    # use if-else statement

print(maximum(5,7))   # 7
print(maximum(7,5))   # 7
print(maximum(5,5))   # 5
```

5. Write a function better:

- input parameter: six integers which represent medal standings of two counties
  - `gold1, silver1, bronze1`: numbers of gold/silver/bronze medals of the first country
  - `gold2, silver2, bronze2`: numbers of gold/silver/bronze medals of the second country
- return value: a string $\begin{cases} \texttt{"First"} & \text{if the first country achieves the better result} \\ \texttt{"Second"} & \text{if the second country achieves the better result} \\ \texttt{"Tie"} & \text{if tied} \end{cases}$
  - according to the gold-silver-bronze order (not by the sum of total medals)
  - refer to the outputs for the sample inputs

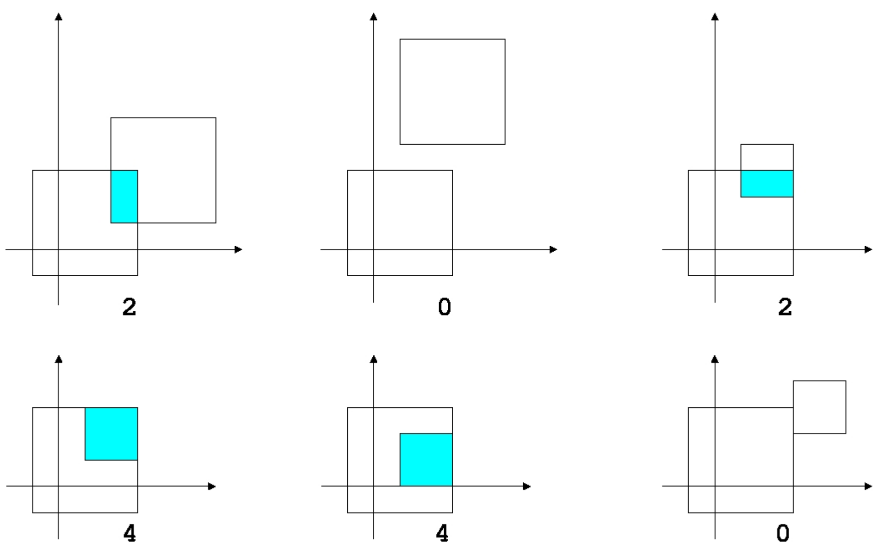
```
def better(gold1, silver1, bronze1, gold2, silver2, bronze2):
    if gold1 > gold2:
        return "First"
    if gold1 < gold2:
        return "Second"

    # ADD ADDITIONAL CODE HERE!


print(better(10,4,24, 1,35,25))    # First
print(better(1,35,25, 10,4,24))    # Second
print(better(10,18,0, 10,4,24))    # First
print(better(10,4,24, 10,18,0))    # Second
print(better(10,20,5, 10,20,4))    # First
print(better(10,20,4, 10,20,5))    # Second
print(better(10,20,5, 10,20,5))    # Tie
```

6. Write a function <u>`area`</u>:

- input parameter: six positive integers `x1`, `y1`, `l1`, `x2`, `y2`, `l2` where
  - `x1`, `y1`, `l1` represent a square whose center is at (`x1`, `y1`) and side length `l1`.
  - `x2`, `y2`, `l2` represent another square whose center is at (`x2`, `y2`) and side length `l2`.
  - for simplicity, assume $\boxed{\texttt{l1} \geq \texttt{l2}}$ ($\boxed{\texttt{l1} \geq \texttt{l2}}$인 입력만 고려하면 됨)
- return value: the area of the intersection of the two squares



7. Write a function leapYear:

- input parameter: a positive integer year
- return value: a boolean value $\begin{cases} \texttt{True} & \text{if year is a leap year (윤년)} \\ \texttt{False} & \text{otherwise} \end{cases}$

Note:

- Basically, leap years occur in years divisible by 4.
  - 2009, 2010, and 2011 are not leap years, while 2008 and 2012 are leap years.
- The years ending with 00 are leap years only if they are divisible by 400.
  - 1700, 1800, 1900, 2100, and 2200 are not leap years, while 1600, 2000, and 2400 are leap years.

```
def leapYear(year):
    if year%4 != 0:
        return False

    # now, year is divisible by 4
    # ADD ADDITIONAL CODE HERE!

print(leapYear(2008), leapYear(2011), leapYear(2012))  # True False True
print(leapYear(2000), leapYear(2100), leapYear(2200))  # True False False
print(leapYear(2300), leapYear(2400), leapYear(3200))  # False True True
```

Write a function numDays by using leapYear implemented above:

- input parameter: two positive integers year and month
- return value: the number of days in the given year and month

```
def numDays (year, month):
    assert (1 <= month <= 12)

    if month == 1 or month == 3 or month == 5 or month == 7 or \
       month == 8 or month == 10 or month == 12:
        return 31

    # ADD ADDITIONAL CODE HERE!

print(numDays(2000,1), numDays(2001,4), numDays(2004,8))  # 31 30 31
print(numDays(2004,9), numDays(2005,3), numDays(2005,7))  # 30 31 31
print(numDays(2008,2), numDays(2011,2), numDays(2012,2))  # 29 28 29
print(numDays(2000,2), numDays(2100,2), numDays(2200,2))  # 29 28 28
print(numDays(2300,2), numDays(2400,2), numDays(3200,2))  # 28 29 29
```

8. Write functions printMultTable1 and printMultTable2:

- input parameter / return value: 없음
- action: print out parts of the multiplication table as in the outputs

† copy and slightly modify `printMultTable0`
  - the meaning of `for i in range(1, 10, 2): ?`
  - the meaning of `for i in range(1, 10, 2):`
    `for j in range(1, i+1): ?`

Output of `printMultTable1()`:

```
1 2 3 4 5 6 7 8 9
3 6 9 12 15 18 21 24 27
5 10 15 20 25 30 35 40 45
7 14 21 28 35 42 49 56 63
9 18 27 36 45 54 63 72 81
```

Output of `printMultTable2()`:

```
1
3 6 9
5 10 15 20 25
7 14 21 28 35 42 49
9 18 27 36 45 54 63 72 81
```

9. One way to calculate e^x is to use infinite series expansion

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^{100}}{100!}$$

(where 100 can be replaced by any larger integer for a greater precision.)

Write a function exp:

- input parameter: a `float` x
- return value: approximation of e^x computed by the above formula

```
# copy factorial() in Week06_P03.py here, and make use of it

def exp(x):
    sum = 1
    # ADD ADDITIONAL CODE HERE!

print(exp(1.0))  # 2.7182818284590455
print(exp(2.0))  # 7.389056098930649
print(exp(4.0))  # 54.598150033144265
```

10. Write a function dayOfWeek:

- input parameter: three integers `year`, `month`, and `day` where `year` ≥ 2000
- return value: the day of the week for the date (`year`, `month`, `day`)
  - return one of the strings `"Mon"`, `"Tue"`, `"Wed"`, `"Thu"`, `"Fri"`, `"Sat"`, `"Sun"`

Hint:

- Make use of the functions `leapYear` and `numDays` implemented in Problem 7.
- Use the fact that (2000, 1, 1) is Saturday.
- Count the number of days from 2000/1/1 to `year/month/day` by three steps:
  - for example, if `year/month/day` is 2015/4/13,
    (1) count the number of days from 2000 to 2014
    (2) count the number of days from 2015/Jan to 2015/Mar
    (3) count the number of days from 2015/Apr/1 to 2015/Apr/12

year	2000				2001				···	2014				2015			
month	1	2	···	12	1	2	···	12	···	1	2	···	12	1	2	3	4
														31	28	31	12

```
def dayOfWeek(year, month, day):
    counter = 0

    # step 1: count the number of days from 2000 to year-1

    # step 2: count the number of days from year/Jan to year/(month-1)

    # step 3: count the ... from year/month/1 to year/month/(day-1)

    n = counter%7
    if n==0:
        return "Sat"
    # step 4: complete the code for the other cases
    elif ...


print(dayOfWeek(2001,1,28))   # Sun
print(dayOfWeek(2002,11,21))  # Thu
print(dayOfWeek(2004,3,4))    # Thu
print(dayOfWeek(2008,7,1))    # Tue
print(dayOfWeek(2011,5,8))    # Sun
print(dayOfWeek(2013,3,23))   # Sat
```

11. Write a function sumSquares:

- input parameter: an integer list `a` (of length `n`)
- return value: the value of $a[0]^2 + a[1]^2 + a[2]^2 + \ldots + a[n-1]^2$
  - i.e. the sum of squares of elements in the list `a`
  - e.g. `sumSquares([4,3,12])` returns `169` $(= 4^2+3^2+12^2)$
  - recall that the Python code to compute x^2 is `x**2`

```
def sumSquares(a):
    n = len(a)
    sum = 0
    # ADD ADDITIONAL CODE HERE!
    for i in range(n):

print(sumSquares([3,5,4]))  # 50
print(sumSquares([2,5,4,0,1,-1,5,1]))  # 73
```

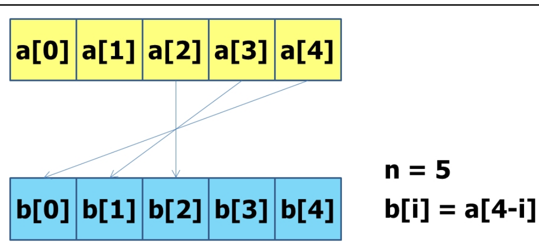
12. Write a function reverse:

- input parameter: a list of integers
- return value: a new list with the same length where the order is reversed
  - e.g. `reverse([1,5,3,7,6])` returns `[6,7,3,5,1]`

```
def reverse(a):
    n = len(a)
    b = [None] * n    # empty list of length n = len(a)

    for i in range(n):
        b[i] = a[??]  # ADD ADDITIONAL CODE HERE!
    return b

print(reverse([3,1,5,2,4]))        # [4,2,5,1,3]
print(reverse([7,6,3,1,5,8,2,4]))  # [4,2,8,5,1,3,6,7]
```



13. Write a function square:

- input parameter: a list of integers
- return value: a new list with the same length where each element is squared
  - e.g. `square([1,3,5,6,7])` returns `[1,9,25,36,49]`

† the overall structure of square is very similar to the above function reverse

```
def square(a):
    # ADD ADDITIONAL CODE HERE!

L = [7,6,3,1,5,8,2,4]
print(square(L))  # [49,36,9,1,25,64,4,16]
print(L)          # [7,6,3,1,5,8,2,4]
```

14. Write a function inversePermutation:

- input parameter: a list `a` that represents a permutation on the set $\{0, 1, \ldots, n-1\}$
  - i.e. `a[0]` , `a[1]` , ..., `a[`n`-1]` are distinct and
    each of `a[0]` , `a[1]` , ..., `a[`n`-1]` is one of $0, 1, \ldots, n-1$
- return value: the list that represents the inverse permutation of the permutation represented by `a`
  - e.g. `inversePermutation ([6, 5, 4, 9, 8, 7, 3, 2, 1, 0])`
    `== [9, 8, 7, 6, 2, 1, 0, 5, 4, 3]`

† Hint: If `b` is the solution, then it satisfies

`b[a[i]] == i` for all `i` .

What for the inverse (i.e. if `b[a[i]] == i` for all `i`, then `b` is the solution)?

15. Write a function findMin:

- input parameter: a list of integers
- return value: the minimum value in the list elements

```
def findMin(a):
    min = a[0]

    # ADD ADDITIONAL CODE HERE!
    for i in range(1,len(a)):
        if a[i] < min:

print(findMin([7,8,3,4,3,6]))        # 3
print(findMin([3,5,7,2,7,2,3,8,6]))  # 2
```

16. Write a function closestPair:

- input parameter: a list of points in the plane
  - where each point is represented by [x,y] as in Problem **??**
  - e.g. [[4,-4], [7,5], [2,1]] represents points $(4, -4)$, $(7, 5)$, $(2, 1)$
- return value: the distance of the closest pair of points (i.e. mininum distance)

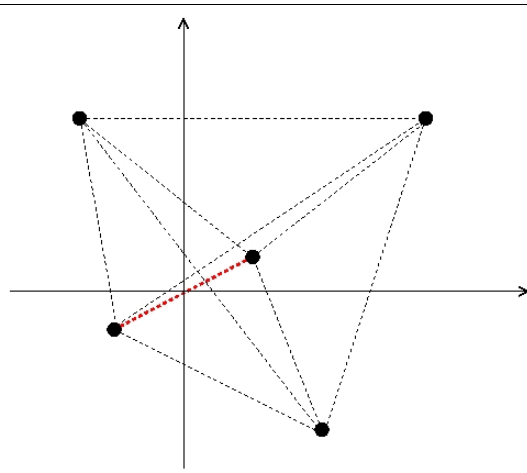
```
def distSquared(p1, p2):
    return (p2[0]-p1[0])**2 + (p2[1]-p1[1])**2

def closestPair(p):
    n = len(p)
    min = distSquared(p[0], p[1])

    # ADD ADDITIONAL CODE HERE!
    for i in range(n):
        for j in range(i+1,n):
            d = distSquared(p[i], p[j])
            if d < min:



points = [[4,-4],[7,5],[2,1],[-2,-1],[-3,5]]
print(closestPair(points))    # 4.47213595499958
                              # (distance bet'n [2,1] and [-2,-1])
```



17. Write a function countZero:

- input parameter: an integer list numbers
- return value: the number of occurrences of 0 in numbers

```
def countZero(numbers):
    # ADD ADDITIONAL CODE HERE!

print(countZero([0,4,0,-2,4,0]))          # 3
print(countZero([1,0,-2,4,0,0,-7,0,5]))  # 4
```

18. Write functions somePrime and allPrime :

- input parameter: a list of positive integers
- return value: a boolean
  - True: somePrime: if there is a prime in the list
    allPrime: if all the numbers in the list are primes
  - False: otherwise

```
# "for all" pattern
def isPrime(p):
    if p <= 1: return False
    for i in range(2, p//2+1):
        if p % i == 0:       # not (p % i != 0)
            return ??
    return ??

def somePrime(numbers):
    # ADD ADDITIONAL CODE HERE!
    for i in range(len(numbers)):
        if isPrime(numbers[i]):


def allPrime(numbers):
    # ADD ADDITIONAL CODE HERE!


num1 = [217, 287, 143, 163, 319]
num2 = [217, 287, 143, 169, 319]
num3 = [223, 281, 227, 151, 149]
print(somePrime(num1), allPrime(num1))   # True False
print(somePrime(num2), allPrime(num2))   # False False
print(somePrime(num3), allPrime(num3))   # True True
```

19. Write a function allDistinct:

- input parameter: an integer list `numbers`
- return value: a boolean
  - `True`: if all `numbers[0]`, `numbers[1]`, `numbers[2]`, $\cdots$ are distinct
  - `False`: otherwise

```
def allDistinct(numbers):
    # ADD ADDITIONAL CODE HERE!
    for i in range(len(numbers)):
        for j in range(i+1,len(numbers)):


print(allDistinct([1,3,2,5,2,1]))   # False
print(allDistinct([1,0,2,5,3,4]))   # True
```

Write a function allWithinRange:

- input parameter: an integer list `numbers`, and two integers `lower`, `upper`
- return value: a boolean
  - `True`: if `lower` $\leq$ `numbers[i]` $\leq$ `upper` for all `i = 0,1,`$\cdots$
  - `False`: otherwise

```
def allWithinRange(numbers, lower, upper):
    # ADD ADDITIONAL CODE HERE!

print(allWithinRange([1,0,2,6,3,4], 0,5))   # False
print(allWithinRange([1,0,2,5,3,4], 0,5))   # True
```

Write a function isPermutation using the functions allDistinct and allWithinRange implemented above: (very simple. you can implement it just in one line!)

- input parameter: an integer list `numbers`
- return value: $\begin{cases} \texttt{True} & \text{if the list } \texttt{numbers} \text{ is a } \textbf{permutation} \\ \texttt{False} & \text{otherwise} \end{cases}$

† An integer list `numbers` with length `n` is called a **permutation** if

all `numbers[0]`, `numbers[1]`, $\cdots$, `numbers[n-1]` are distinct and

`0` $\leq$ `numbers[i]` $\leq$ `n-1` for all `i = 0,1,`$\cdots$`,n-1`

```
print(isPermutation([1,3,2,5,2,1]))   # False
print(isPermutation([1,0,2,5,3,4]))   # True
print(isPermutation([1,0,2,6,3,4]))   # False
```

20. Write a function gcd:

- input parameter: two positive integers a and b
- return value: the greatest common divisor (최대공약수) of a and b

The greatest common divisor (GCD) can be computed by the Euclidean algorithm:

- Given positive integers a and b ($a \geq b$), let $r = a \,\%\, b$ ($< b$).
  - i.e. r is the remainder when a is divided by b
- Then, the GCD of a and b is the same as the GCD of b and r.[1] Thus we can use the equation

$$\mathrm{gcd}(a, b) \;=\; \mathrm{gcd}(b, r)$$

- For example,

$$\mathrm{gcd}(36, 20) = \mathrm{gcd}(20, 16) = \mathrm{gcd}(16, 4) = \mathrm{gcd}(4, 0) = 4$$

  implies that the GCD of 36 and 20 is 4.
- For any two starting numbers, this repeated reduction eventually produces a pair where the second number is 0. Then the GCD is the other number.

```
def gcd(a,b):
    if a < b:  # swap so that a >= b
        a,b = b,a

    # ADD ADDITIONAL CODE HERE!
    while b != 0:
        r = a%b
        a = ??
        b = ??

    return ??

print(gcd(36, 20))            # 4
print(gcd(2408208, 2790876))  # 132
```

Hint: during the execution of `while` loop, the values of `a,b` must be changed to

`36,20` ⇒ `20,16` ⇒ `16,4` ⇒ `4,0`

[1] For a correctness proof, refer to
http://en.wikipedia.org/wiki/Euclidean_algorithm#Proof_of_validity

21. Write a function <u>`deleteThree`</u>:

- input parameter: a list L
- return value: the list obtained by removing all occurrences of 3
  - L에서 3을 모두 제거하여 얻은 list

새로운 list를 만들때는 `[]`로 초기화한 후, `M.append(·)` 로 하나씩 붙여나가면 된다.

```
M = []
for i in range(len(L)):
    if [some condition on] L[i]:
        M.append([??])
return M
```

이 문제의 경우 위의 $\boxed{\textit{some condition}}$과 $\boxed{??}$을 뭘로 채우면 될까?

```
def deleteThree(L):
    # ADD ADDITIONAL CODE HERE!

print(deleteThree([2,5,7,3,2,8,3,3]))  # [2,5,7,2,8]
print(deleteThree([2,3,7,3,2,8,3,3]))  # [2,7,2,8]
print(deleteThree([3,3,7,3,2,8,3,3]))  # [7,2,8]
```

22. Write a function <u>`makeSet`</u>:

- input parameter: a list L
- return value: the new sorted list which contains every elements of L exactly one
  - L의 원소들을 중복없이 정확히 하나씩만 포함한, 정렬된 list

- 21번 문제의 코드 형태와 유사. `L[i]`가 지금까지 만들어둔 M에 포함되지 않을때만 `M.append(·)` 로 붙이면 됨
- `L.sort()`는 L의 원소들이 증가하는 순서로 나열되도록 L을 변경한다. (리턴 값은 없음)
- `x` <u>`in`</u> L은 list L에 x가 포함되어 있으면 `True`. `x` <u>`not in`</u> L이나 <u>`not`</u> `(x` <u>`in`</u> `L)`은 반대

```
def makeSet(L):
    # ADD ADDITIONAL CODE HERE!

print(makeSet([1,1,3,5]))          # [1,3,5]
print(makeSet([2,1,2,8,8]))        # [1,2,8]
print(makeSet([3,4,5,6,7,3,4]))    # [3,4,5,6,7]
```

23. Write a function factorize:

- input parameter: a positive integer $n (\geq 2)$
- return value: the list of prime factors of n sorted in increasing order
  - e.g.: `factorize(504)` returns `[2,3,7]` since $504 = 2^3 \cdot 3^2 \cdot 7$

† `.append(·)`을 이용하여 작은 소인수부터 하나씩 차례대로 붙여나간다.

24. Write a function countZero:

- input parameter: a list `a` that represents a 2-dimensional array
- return value: the number of 0's in the array

counter 패턴의 2중 `for` 루프. `height`는 행의 갯수이고 `width`는 열의 갯수. 슬라이드 그림/코드 참조

```
def countZero(a):
    height = len(a)
    width = len(a[0])
    # ADD ADDITIONAL CODE HERE!

print(countZero([[1,2,3],[0,0,5],[0,3,0],[0,0,0]]))  # 7
print(countZero([[0,2,3],[0,0,5],[0,3,0]]))          # 5
```

25. Write a function countZero:

- input parameter: a list `a` that represents a 3-dimensional array
- return value: the number of 0's in the array

슬라이드 그림/코드 참조

```
def countZero(a):
    depth = len(a)
    height = len(a[0])
    width = len(a[0][0])
    # ADD ADDITIONAL CODE HERE!

print(countZero([[[1,2],[0,0]],[[0,0],[0,0]]]))                  # 6
print(countZero([[[1,2],[0,0]],[[0,0],[0,0]],[[0,0],[0,0]]]))  # 10
```

26. Write a function <u>`sorted`</u>:

- input parameter: a list a that represents a 2-dimensional array
- return value: $\begin{cases} \texttt{True} & \text{if each row/column is in non-decreasing order} \\ \texttt{False} & \text{otherwise} \end{cases}$

2	3	7	9	11	12
5	6	8	10	12	15
7	7	8	10	12	15
8	9	10	10	13	17

- “for all” 패턴의 2중 `for` 루프를 2개 이용
- 루프 하나는 각 행에 대해 가로 방향으로 체크. 다음 루프는 각 열에 대해 세로 방행으로 체크
- `range`의 파라미터로 `height`, `width`가 그대로 들어가야 하는지, `-1`을 해서 들어가야 하는지 꼼꼼히 따져봐야 함

```
def sorted(a):
    # ADD ADDITIONAL CODE HERE!


test1 = [
    [2,3,7,9,11,12],
    [5,6,8,10,12,15],
    [7,7,8,10,12,15],
    [8,9,10,10,13,17]
]

test2 = [
    [2,3,7,9,11,12],
    [5,6,8,10,12,15],
    [7,7,8,10,12,18],
    [8,9,10,10,13,17]
]

print(sorted(test1))  # True
print(sorted(test2))  # False
```

27. Write a function <u>`countMines`</u>:

- input parameter: a list that represents a 2-dimensional minefield
- return value: the list that represents the 2-dimensional array of integers storing the count of bombs in each neighborhood
  - The neighborhood for a location includes the location itself and its eight adjacent locations

T	F	F	F	F	T
F	F	F	F	F	T
T	T	F	T	F	T
T	F	F	F	F	F
F	F	T	F	F	F
F	F	F	F	F	F

1	1	0	0	2	2
3	3	2	1	4	3
3	3	2	1	3	2
3	4	3	2	2	1
1	2	1	1	0	0
0	1	1	1	0	0

2차원 리스트를 만드는 법은 슬라이드 그림/코드 참조. 리스트의 각 자리는 None대신 0으로 초기화 해놓고 카운팅하면 됨

```
def withinBoundary(height, width, i, j):
    return i>=0 and i<height and j>=0 and j<width

def countMines(mineField):
    height = len(mineField)
    width = len(mineField[0])
    # ADD ADDITIONAL CODE HERE!


T = True
F = False
mineField = [
        [T, F, F, F, F, T],
        [F, F, F, F, F, T],
        [T, T, F, T, F, T],
        [T, F, F, F, F, F],
        [F, F, T, F, F, F],
        [F, F, F, F, F, F]
]
mines = countMines(mineField)
for i in range(len(mines)):   # [1, 1, 0, 0, 2, 2]
    print(mines[i])           # [3, 3, 2, 1, 4, 3]
                              # [3, 3, 2, 1, 3, 2]
                              # [3, 4, 3, 2, 2, 1]
                              # [1, 2, 1, 1, 0, 0]
                              # [0, 1, 1, 1, 0, 0]
```

28.

Let A and B be an $m \times p$ matrix and an $p \times n$ matrix, respectively:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mp} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p1} & b_{p2} & \dots & b_{pn} \end{pmatrix}.$$

The product of A with B, denoted $A \cdot B$, is defined to be an $m \times n$ matrix

$$\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix} \quad \text{where } c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj} \quad (1 \le i \le m,\ 1 \le j \le n).$$

Write a function `product`:

- input parameter: two matrices A and B that are represented by 2-dimensional lists
- return value: the 2-dimensional list that represents the matrix product $A \cdot B$
  - if the matrix product $A \cdot B$ is not well-defined (due to incompatible dimensions), then return `None` (행/열 갯수의 불일치로 matrix product가 정의되지 않는 경우는 `None`을 return)

† 위의 행렬 표현에서의 인덱스는 1부터 시작하지만 list에서 인덱스는 **0부터 시작**함에 유의

29. Write an object type `Circle` for circles and some functions that work on `Circle`.

(1) Write a function `__init__` (**modifier**) that creates an object of `Circle` class:

- input parameter: `self`, a `Point` object `c`, and an integer `r`
- action: create state variables `center` (for 원의 중심) and `radius` (for 반지름) and initialize them to `c` and `r`, respectively
- return value: 없음

(2) Write a function `__str__` (**pure function**) that returns a string for the `print` command:

- input parameter: `self`
- return value: the string in the following format: `"(center,radius)"`, e.g. `"((0,1) , 5)"`

(3) Write a function `area` (**pure function**):

- input parameter: `self`
- return value: the area of `self` (use `math.pi` for π)

(4) Write a function `getRadius` (**pure function**):

- input parameter: `self`
- return value: the radius of `self`

(5) Write a function `getCenter` (**pure function**):

- input parameter: `self`
- return value: the center of `self` (as `Point` object)

(6) Write a function `setRadius` (**modifier**):

- input parameter: `self` and an integer `r`
- action: change the radius of `self` to `r`
- return value: 없음

(7) Write a function `moveTo` (**modifier**):

- input parameter: `self` and two integers `x, y`
- action: move the center of `self` to `Point(x,y)`
- return value: 없음

(8) Write a function `move` (**modifier**):

- input parameter: `self` and two integers `dx, dy`
- action: move the center of `self` by the amount of `(dx, dy)`
  - 원중심의 x/y 좌표를 dx, dy 만큼 이동
- return value: 없음

```
class Point:
    # modifier
    def __init__(self, px, py):
        self.x = px
        self.y = py

    # pure function
    def __str__(self):
        return "(" + str(self.x) + ',' + str(self.y) + ")"

    # pure function
    def getX(self):
        return self.x

    # pure function
    def getY(self) :
        return self.y

    # modifier
    def setX(self, v):
        self.x = v

    # modifier
    def setY(self, v):
        self.y = v

    # pure function
    def distance(self, p):
        dx = self.x - p.x
        dy = self.y - p.y
        return (dx*dx + dy*dy)**0.5

    # pure function
    def add(self, p):
        x = self.x + p.x
        y = self.y + p.y
        return Point(x,y)

########################################
class Circle:
    # modifier
    def __init__(self, c, r):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!

    # pure function
    def __str__(self):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!

    # pure function
    def area(self):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!
```

```
    # pure function
    def getRadius(self):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!

    # pure function
    def getCenter(self):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!

    # modifier
    def setRadius(self, v):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!

    # modifier
    def moveTo(self, x, y):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!

    # modifier
    def move(self, dx, dy):
        pass # remove it after completing your code
        # ADD ADDITIONAL CODE HERE!

def test():
    p0 = Point (0,0)
    c1 = Circle(p0,3)
    print(c1)                # ((0,0) , 3)
    print(c1.area())         # 28.274333882308138
    print(c1.getRadius())    # 3
    print(c1.getCenter())    # (0,0)

    c1.setRadius(5)
    print(c1)                # ((0,0) , 5)
    print(c1.area())         # 78.53981633974483

    c1.moveTo(3,4)
    print(c1)                # ((3,4) , 5)

    c1.move(1,1)
    print(c1)                # ((4,5) , 5)

test()
```

30. A rational number is a number that can be represented as the ratio of two integers. For example, 2/3 is a rational number, where

- 2 is a **numerator** (분자) and
- 3 is a **denominator** (분모).
  - 7 is regarded as a rational number with an implicit 1 in the denominator.

For this problem, you are going to write an object type `Rational` for rational numbers and object methods that are overloaded to built-in operations/functions such as `+`, `-`, `<=`, `**`, `abs(·)`. Be aware that all these overloadable methods are supposed to be **pure functions** (but **not modifiers**).

(1) Write a function `__add__` (overloaded to `+` operation):
- input parameter: `self` and `r` (both are `Rational` objects)
- return value: a new `Rational` object that represents `self + r` (in the form of **irreducible** fraction)
  - Make sure that the result of the operation is reduced so that the numerator and denominator have **no common divisor** other than 1
  - This function should be a **pure function**; it should not modify the input objects `self` and `f`.

(2) Write a function `__sub__` (overloaded to `-` operation):
- input parameter: `self` and `r`
- return value: a new `Rational` object that represents `self - r` (in the form of **irreducible** fraction)

(3) Write a function `__mul__` (overloaded to `*` operation):
- input parameter: `self` and `r`
- return value: a new `Rational` object that represents `self * r` (in the form of **irreducible** fraction)

(4) Write a function `__div__` (overloaded to `/` operation):
- input parameter: `self` and `r`
- return value: a new `Rational` object that represents `self / r` (in the form of **irreducible** fraction)

(5) Write a function `__neg__` (overloaded to `unary -` operation):
- input parameter: `self`
- return value: a new `Rational` object that represents `- self`

(6) Write a function `__abs__` (overloaded to `abs(·)` function):
- input parameter: `self`

- return value: a new `Rational` object that represents the absolute value of `self`

(7) Write a function `__pow__` (overloaded to `**` operation):

- input parameter: `self` and a positive integer `p`
- return value: a new `Rational` object that represents $\boxed{\texttt{self}^{\texttt{p}}}$

(8) Write a function `__radd__` (overloaded to "reflected" + operation):

- input parameter: `self` and an integer `r`
- return value: a new `Rational` object that represents `r + self` (e.g. `0 + Rational(1,2)`)

† Refer to `http://www.rafekettler.com/magicmethods.html#numeric` for a detailed description of **reflected** arithmetic operations.

(9) Write a function `__rmul__` (overloaded to "reflected" * operation):

- input parameter: `self` and an integer `r`
- return value: a new `Rational` object that represents `r * self` (e.g. `1 * Rational(1,2)`)

(10) Write functions `__eq__`, `__ne__`, `__le__`, `__ge__`, `__lt__`, `__gt__` (overloaded to `==, !=, <=, >=, <, >` relations):

- input parameter: `self` and `r` (both are `Rational` objects)
- return value: a boolean
  - for example, `__le__(self, r)` returns $\begin{cases} \texttt{True} & \text{if } \texttt{self} \leq \texttt{r} \\ \texttt{False} & \text{if } \texttt{self} > \texttt{r} \end{cases}$

31. Write object methods of the `Matrix` class which represents matrices of `int`/`Rational`. All these methods should be **pure functions** (but **not modifiers**). Make sure that your program also works well with matrices over `Rational` objects.

(1) Write a function `__add__` (overloaded to `+` operation):
- input parameter: `self` and `M` (both are `Matrix` objects)
- return value: a new `Matrix` object that represents `self + M`
  - This function should be a **pure function**; it should not modify the input objects `self` and `M`.

(2) Write a function `__sub__` (overloaded to `-` operation):
- input parameter: `self` and `M`
- return value: a new `Matrix` object that represents `self - M`

(3) Write a function `__mul__` (overloaded to `*` operation):
- input parameter: `self` and `M` (which have compatible dimensions)
- return value: a new `Matrix` object that represents the multiplication $\texttt{self} \cdot \texttt{M}$

(4) Write a function `__pow__` (overloaded to `**` operation):
- input parameter: `self` and a positive integer `p`
- return value: a new `Matrix` object that represents the matrix exponentiation $\texttt{self}^{\texttt{p}}$

† Use the fast matrix exponentiation based on **recursion** (refer to the lecture slides "#3: Recursion I"). Otherwise, your program will not finish in reasonable time; (we test with very large `p`).

(5) Write a function `__rmul__` (overloaded to "reflected" `*` operation):
- input parameter: `self` and an integer `factor`
- return value: a new `Matrix` object that represents the **scalar** multiplication $\texttt{factor} \cdot \texttt{self}$

(6) Write a function `transpose`:
- input parameter: `self`
- return value: a new `Matrix` object that represents the transpose $\texttt{self}^{\mathrm{T}}$

(7) Write a function $\boxed{\texttt{submatrix}}$:

- input parameter: `self` and two integers `i, j`
- return value: a new `Matrix` object that represents the submatrix of `self` formed by deleting the `(i+1)`$^{\text{th}}$ row and `(j+1)`$^{\text{th}}$ column

† Use the `.pop(·)` method for lists

(8) Write a function $\boxed{\texttt{determinant}}$:

- input parameter: `self` (square matrix)
- return value: the determinant of the matrix `self`

† Implement the Laplace expansion by using recursion:

Let $A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$ be an $n \times n$ matrix and A_{ij} be the submatrix of A formed by deleting the i-th row and j-th column. Then,

$$\det(A) = \sum_{j=1}^{n} \Big((-1)^{1+j} \cdot a_{1j} \cdot \det(A_{1j}) \Big)$$

(9) Write a function $\boxed{\texttt{inverse}}$:

- input parameter: `self` (square matrix)
- return value: the inverse of the matrix `self`

† Use the simple (computationally inefficient) formula based on the cofactors.

$$A^{-1} = \frac{1}{\det(A)} \cdot \begin{pmatrix} C_{11} & C_{12} & \dots & C_{1n} \\ C_{21} & C_{22} & \dots & C_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n1} & C_{n2} & \dots & C_{nn} \end{pmatrix}^{\mathrm{T}} \quad \text{where } C_{ij} = (-1)^{i+j} \cdot \det(A_{ij})$$

32. numpy 슬라이드의 모든 코드를 입력하여 수행해서 결과를 이해하도록 한다.