REPORT

[과제 : NP 문제지(1)]

과 목 명	파이썬과학프로그래밍기초
교 수 명	김 병 정
학 번	20237107
작 성 자	하 태 영
제 출 일	2025.05.11

한림대학교

문제 NP11-0001

np1과 같이 0부터 9까지의 값으로 넘파이 1차원 배열을 채우고, 이 배열을 거꾸로 np2 를 만드는 프로그램을 작성하시오.

0	1	2	3	4	5	6	7	8	9	···	$np1$

9	8	7	6	5	4	3	2	1	0	···	$np2$

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

np1 = np.arange(0, 10) # 0 ~ 9 까지의 정수를 배열로 만들고, np1 에 저장
print(np1) # np1 출력

np2 = np.array(list(reversed(np1))) # np1 을 뒤집고, 리스트화하여 배열로
np2 에 저장
print(np2) # np2 출력
```

```
[0 1 2 3 4 5 6 7 8 9]
[9 8 7 6 5 4 3 2 1 0]
```

문제 NP11-0002

numpy 를 이용해서 np1 uniform 배열을 만들고, np2 와 같이 짝수만 추출하는 프로그램을 작성하시오.

- 조건

1	2	7	2	3	0	2	0	4	1	···	$np1$

2	2	0	2	0	4	···	$np2$

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 10 까지 10 개의 랜덤 정수타입으로 np1 에 저장
np1 = np.random.uniform(0, 10, 10).astype('int')
print(np1) # np1 출력

np2 = np1[np.where(np1%2==0)] # np1 의 배열에서 짝수만 골라서 np2 에 저장
print(np2) # np2 출력
```

```
[8 8 4 2 2 1 6 2 4 3]
[8 8 4 2 2 6 2 4]
```

문제 NP11-0003

다음 프로그램의 배열을 그려보자.

- 조건
  - np1 처럼 네모상자를 그리고 값을 채운다.
- 그림 예

0.31	0.48	0.23	0.31	0.73	0.94
0.94	0.50	0.11	0.45	0.56	0.97
0.26	0.58	0.89	0.79	0.24	0.99

$\cdots \quad np1$

```
import numpy as np
np1= np.arange(0,24).reshape(3,8).T
print(np1)
```

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# .T = 전치 연산 (행과 열을 바꿈)
# 0 ~ 23 까지의 정수로 3 행 8 열을 만들고, 행과 열을 전치하여 np1 에 저장
np1= np.arange(0,24).reshape(3,8).T
print(np.transpose(np1)) # np1 의 행과 열을 전치하여 3 행 8 열로 출력
```

```
[[ 0  1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14 15]
 [16 17 18 19 20 21 22 23]]
```

문제 NP11-0004

다음 그림과 같은 2차원 ndarray 데이터 배열을 만드시오.

- 조건
  - numpy 이용

0	8	16
1	9	17
2	10	18
3	11	19
4	12	20
5	13	21
6	14	22
7	15	23

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# .T = 전치 연산 (행과 열을 바꿈)
# 0 ~ 23 까지의 정수로 배열을 만들고, 3 행 8 열의 전치인 8 행 3 열로 만들어서
ndarray 에 저장
ndarray = np.arange(0, 24).reshape(3, 8).T
print(ndarray) # ndarray 출력
```

```
[[ 0  8 16]
 [ 1  9 17]
 [ 2 10 18]
 [ 3 11 19]
 [ 4 12 20]
 [ 5 13 21]
 [ 6 14 22]
 [ 7 15 23]]
```

문제 NP11-0005

np1 과 같은 크기의 랜덤 배열을 생성해보자.

- 조건
  - np.random.random() 함수 사용
  - 소숫점2째 자리까지 표현

0.33	0.10	0.74
0.64	0.89	0.81
0.38	0.28	0.56

$\cdots \quad np1$

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 1 사이의 난수를 3 행 3 열로 만들어주고, 소수점 2 자리까지의 값들을 np1 에 저장
np1 = np.round(np.random.random((3, 3)), 2)
print(np1) # np1 출력
```

```
[[0.74 0.96 0.75]
 [0.67 0.59 0.19]
 [0.43 0.81 0.76]]
```

문제 NP11-0006

다음 프로그램의 배열을 그려보자.

```
import numpy as np
aaa = np.arange(0,24).reshape(3,4,2)
print(aaa)
```

- 조건
  - 네모상자를 그리고 값을 채운다.
  - 왼쪽부터 1,2,3 면을 의미함
- 그림 예

0	1
2	3
4	5
6	7

8	9
10	11
12	13
14	15

16	17
18	19
20	21
22	23

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 23 의 정수를 3 개의 블록, 블록 1 개당 4 행 2 열로 만들고 aaa 에 저장
aaa = np.arange(0,24).reshape(3,4,2)
print(aaa) # aaa 출력
```

```
[[[ 0  1]
  [ 2  3]
  [ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]
  [12 13]
  [14 15]]

 [[16 17]
  [18 19]
  [20 21]
  [22 23]]]
```

문제 NP11-0007

다음 그림과 같은 3차원 ndarray 데이터 배열을 만드시오.

- 조건
  - numpy 이용
  - 왼쪽부터 1,2,3 면을 의미함

0	1
2	3
4	5
6	7

8	9
10	11
12	13
14	15

16	17
18	19
20	21
22	23

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 23 의 정수를 3 개의 블록, 블록 1 개당 4 행 2 열로 만들고 ndarray 에 저장
ndarray = np.arange(0, 24).reshape(3, 4, 2)
print(ndarray) # ndarray 출력
```

```
[[[ 0  1]
  [ 2  3]
  [ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]
  [12 13]
  [14 15]]

 [[16 17]
  [18 19]
  [20 21]
  [22 23]]]
```

문제 NP11-0008

np1 배열에서 3의 배수이거나 5의 배수인 위치정보를 모두 찾아 아래와 같이 x,y 쌍으로 출력하시오.

- 조건
  - np1 배열을 만들면서 시작한다.
  - 방법1 : LC 이용
  - 방법2 : np.where() 이용

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17

$\cdots \quad np1$

- 출력 예

```
[(0, 0), (0, 3), (0, 5), (1, 0), (1, 3), (1, 4), (2, 0), (2, 3)]
```

```
# 방법 1 :  LC 이용
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 17 의 정수를 3 행, 6 열 형태의 배열로 생성 후 np1 에 저장
np1 = np.arange(0, 18).reshape(3, 6)

# 1. i 는 np1 의 행(3)만큼 반복, j 는 np1 의 열(6)만큼 반복
# 2. np1 의 원소가 3 의 배수이거나 5 의 배수이면
# 3. i 와 j 를 튜플로 result 에 저장
result = [(i, j) for i in range(np1.shape[0]) for j in
range(np1.shape[1])
        if np1[i, j] % 3 == 0 or np1[i, j] % 5 == 0]
print(result) # result 출력
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
[(0, 0), (0, 3), (0, 5), (1, 0), (1, 3), (1, 4), (2, 0), (2, 3)]
```

```python
# 방법 2 : np.where() 이용
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 17 의 정수를 3 행, 6 열 형태의 배열로 생성 후 np1 에 저장
np1 = np.arange(0, 18).reshape(3, 6)

# np1 의 3 의 배수이거나 5 의 배수인 경우 idx 배열에 저장
idx = np.where((np1 % 3 == 0) | (np1 % 5 == 0))

# idx 의 행과 열을 zip 으로 묶어서 반복하여 i, j 를 정수 형변환한 후 튜플의 형태로
result 에 저장
result = [(int(i), int(j)) for i, j in zip(idx[0], idx[1])]
print(result) # result 출력
```

```
[(0, 0), (0, 3), (0, 5), (1, 0), (1, 3), (1, 4), (2, 0), (2, 3)]
```

문제 NP11-0009

np1 배열에서 3의 배수이거나 5의 배수인 위치정보를 모두 찾아 아래와 같이 x,y 쌍으로 출력하시오.

- 조건
  - numpy 이용
  - 2가지 방법 모두 작성하시오
  - 방법1) 3중 반복문
  - 방법2) np.where() 사용

0	1	2	3
4	5	6	7
8	9	10	11

12	13	14	15
16	17	18	19
20	21	22	23

$\cdots \quad np1$

- 출력 예

```
[(0, 0, 0),
 (0, 0, 3),
 (0, 1, 1),
 (0, 1, 2),
 (0, 2, 1),
 (0, 2, 2),
 (1, 0, 0),
 (1, 0, 3),
 (1, 1, 2),
 (1, 2, 0),
 (1, 2, 1)]
```

```
# 방법 1) 3 중 반복문
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 2 개의 블록을 만들고, 블록 1 개당 3 행 4 열에 0 ~ 23 의 정수를 넣고 np1 에 저장
np1 = np.arange(0, 24).reshape(2, 3, 4)

# 1. i 는 np1 의 블록(2)만큼, j 는 np1 의 행(3)만큼, K 는 np1 의 열(4)만큼 반복
# 2. np1 의 원소가 3 의 배수이건 5 의 배수이면
# 3. i, j, k 를 튜플로 result 에 저장
result = [(i, j, k) for i in range(np1.shape[0])
                    for j in range(np1.shape[1])
                    for k in range(np1.shape[2])
          if np1[i, j, k] % 3 == 0 or np1[i, j, k] % 5 == 0]
print(result) # result 출력
```

```
[(0, 0, 0), (0, 0, 3), (0, 1, 1), (0, 1, 2), (0, 2, 1), (0, 2, 2), (1, 0, 0), (1, 0, 3), (1, 1, 2), (1, 2, 0), (1, 2, 1)]
```

```
# 방법 2 : np.where() 이용
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 2 개의 블록을 만들고, 블록 1 개당 3 행 4 열에 0 ~ 23 의 정수를 넣고 np1 에 저장
np1 = np.arange(0, 24).reshape(2, 3, 4)

# np1 의 3 의 배수이거나 5 의 배수인 경우 idx 배열에 저장
idx = np.where((np1 % 3 == 0) | (np1 % 5 == 0))

# idx 의 블록, 행, 열을 zip 으로 묶어서 반복하여 i, j, k 를 정수 형변환한 후 튜플의
형태로 result 에 저장
result = [(int(i), int(j), int(k)) for i, j, k in zip(idx[0], idx[1],
idx[2])]
print(result) # result 출력
```

```
[(0, 0, 0), (0, 0, 3), (0, 1, 1), (0, 1, 2), (0, 2, 1), (0, 2, 2), (1, 0, 0), (1, 0, 3), (1, 1, 2), (1, 2, 0), (1, 2, 1)]
```

문제 NP12-0001

다음과 3차원 배열 np1 을 만들고, axis 에 따른 합을 구하는 프로그램의 결과를 그리시오.

```
np2 = np1.sum(axis=0)
np3 = np1.sum(axis=1)
np4 = np1.sum(axis=2)
```

0	1	2	3
4	5	6	7
8	9	10	11

12	13	14	15
16	17	18	19
20	21	22	23

$\cdots \quad np1$

```
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기
np1 = np.arange(0, 24).reshape(2, 3, 4)
np2 = np1.sum(axis=0) # 블록 기준의 합
np3 = np1.sum(axis=1) # 행 기준의 합
np4 = np1.sum(axis=2) # 열 기준의 합
print(np1) # np1 출력
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

문제 NP12-0002

임의값으로 3x6 배열을 만들고, 최대 최소값을 출력하는 프로그램을 작성하시오.

- 조건
  - np.random.random() 함수 사용
  - **소숫점2째 자리까지 표현**
  - **행별** 최소, 최대값을 출력한다. (아래 표 참조)
  - 반복문 사용x
  - 리스트 사용 x

0.31	0.48	0.23	0.31	0.73	0.94	
0.94	0.50	0.11	0.45	0.56	0.97	$\cdots$ $np1$
0.26	0.58	0.89	0.79	0.24	0.99	

0.94	0.97	0.99	$\cdots$ max

0.23	0.11	0.24	$\cdots$ min

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 1 사이 난수를 소수점 2 자리 값으로 np1 에 저장
np1 = np.round(np.random.random((3, 6)), 2)

max = np.max(np1, axis=1) # np1 행 기준으로 최대값을 max 에 저장
min = np.min(np1, axis=1) # np1 행 기준으로 최소값을 min 에 저장
print(max) # max 출력
print(min) # min 출력
```

```
[0.94 0.98 0.93]
[0.01 0.11 0.17]
```

문제 NP12-0003

임의값으로 3x6 배열을 만들고, 최대 최소값을 출력하는 프로그램을 작성하시오.

- 조건
  - np.random.random() 함수 사용
  - **소숫점2째 자리까지 표현**
  - **열별** 최소, 최대값을 출력한다. (아래 표 참조)
  - 반복문 사용x
  - 리스트 사용 x

0.40	0.23	0.67	0.36	0.36	0.33	
0.87	0.93	0.63	0.60	0.13	0.07	$\cdots$ $np1$
0.60	0.30	0.63	0.93	0.90	0.38	

0.87	0.93	0.67	0.93	0.90	0.38	$\cdots$ max

0.40	0.23	0.63	0.36	0.13	0.07	$\cdots$ min

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0 ~ 1 사이 난수를 소수점 2 자리 값으로 np1 에 저장
np1 = np.round(np.random.random((3, 6)), 2)

max = np.max(np1, axis=0) # np1 열 기준으로 최대값을 max 에 저장
min = np.min(np1, axis=0) # np1 열 기준으로 최소값을 min 에 저장
print(max) # max 출력
print(min) # min 출력
```

```
[0.52 0.91 0.91 0.61 0.75 0.49]
[0.21 0.07 0.09 0.12 0.19 0.24]
```

문제 NP12-0004

numpy 를 이용해서 배열의 시작과 끝에 가까운 6 사이의 값을 모두 찾아내시오. (6은 포함 x)

- 조건
  - 5줄 이내로 작성하시오.

0	5	2	6	8	6	3	6	3	9	···	$np1$

8	6	3	···	$np2$

```
import numpy as np

# np1 배열 생성
np1 = np.array([0, 5, 2, 6, 8, 6, 3, 6, 3, 9])

# 첫 번째 6과 마지막 6 사이의 값들 추출 (6은 포함되지 않음)
np2 = np1[np.where(np1 == 6)[0][0]+1:np.where(np1 == 6)[0][-1]]
np2 = np2[np.where(np2 != 6)]

# 결과 출력
print(np2)
```

```
[8 3]
```

문제 NP13-0001

다음과 같은 2차원 배열 np1 을 출력해보자.

- 조건
  - 테두리는 1로 채운다
  - np.zeros() 함수를 사용한다.
  - 슬라이스 연산자를 사용한다.
  - 반복문x
  - 리스트x

1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1

$\cdots \quad np1$

```python
import numpy as np # numpy 라이브러리를 np 이름으로 불러오기

# 정수형 8 행, 8 열 배열을 0 으로 채우고 np1 에 저장
np1 = np.zeros((8, 8), dtype=int)
np1[0, :] = 1  # np1 의 첫 번째 행의 값을 1 로 저장
np1[-1, :] = 1 # np1 의 마지막 행의 값을 1 로 저장
np1[:, 0] = 1  # np1 의 첫 번째 열의 값을 1 로 저장
np1[:, -1] = 1 # np1 의 마지막 열의 값을 1 로 저장
print(np1) # np1 출력
```

```
[[1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1]]
```

문제 NP13-0002

행, 열의 크기를 입력받아서 2차원 배열을 리턴하는 myfct() 함수를 작성하자.

- 조건
  - myfct(행크기, 열크기) 호출 예

```
np1 = myfct(3,8)
print(np2)
```

  - 테두리는 1로 채운다
  - 함수 호출로 만들어진 메모리 구조 예.

1	1	1
1	0	1
1	1	1

$\cdots \quad myfct(3,3)$

1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1

$\cdots \quad myfct(3,8)$

```
import numpy as np # numpy 라이브러리를 np 이름으로 불러오기

def myfct(row, col): # 사용자 정의 함수 myfct(행, 열)
  # row, col 를 정수형 타입 배열을 만들고, 0 으로 채워 arr 에 저장
  arr = np.zeros((row, col), dtype=int)
  arr[0, :] = 1  # arr 의 첫 번째 행의 값을 1 로 저장
  arr[-1, :] = 1 # arr 의 마지막 행의 값을 1 로 저장
  arr[:, 0] = 1  # arr 의 첫 번째 열의 값을 1 로 저장
  arr[:, -1] = 1 # arr 의 마지막 열의 값을 1 로 저장
  return arr     # arr 반환

np1 = myfct(3, 8) # myfct(3, 8) 결과를 np1 에 저장
print(np1) # np1 출력
```

```
[[1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1]]
```

문제 NP14-0000

다음 프로그램의 결과를 작성하시오.

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

c = np.array([[1, 2, 3],[4, 5, 6]])
d = np.array([[7, 8, 9],[10, 11, 12]])
```

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

c = np.array([[1, 2, 3],[4, 5, 6]])
d = np.array([[7, 8, 9],[10, 11, 12]])

# np.hstack([]) = 수평(horizontal, 가로)으로 붙여서 하나의 배열로 만든다.
# a, b를 가로로 붙여서 1차원 배열을 만든다.
print("1-1. np.hstack([a,b])")
print(np.hstack([a,b]))
print()
# c, d를 가로로 붙여서 1차원 배열을 만든다.
print("1-2. np.hstack([c,d])")
print(np.hstack([c,d]))
print()
# a, d를 가로로 붙여서 1차원 배열을 만든다.
# 행 개수가 다르기 때문에 오류이다.
print("1-3. np.hstack([a,d]) \n","행 개수가 다르기 때문에 오류이다.")
print()

# a와 마지막 원소를 제외한 b를 가로로 붙여서 1차원 배열을 만든다.
print("1-4. np.hstack([a,b[:-1]]) \n", np.hstack([a,b[:-1]]))
print()

# c와 마지막 원소를 제외한 d를 가로로 붙여서 1차원 배열을 만든다.
# 행 개수가 다르기 때문에 오류이다.
print("1-5. np.hstack([c,d[:,1:]]) \n", "행 개수가 다르기 때문에 오류이다.")
print()
# np.r_[] = 1차원이면 가로로 붙이고, 2차원 이상이면 아래 행으로 붙여서 하나의
배열로 만든다.
# a, b를 가로로 붙여서 1차원 배열을 만든다.
print("1-6. np.r_[a, b] \n", np.r_[a, b])
print()
# a, b를 행 방향(아래)으로 붙여서 2차원 배열을 만든다.
```

```
print("1-7. np.r_[[a], [b]] \n", np.r_[[a], [b]])
print()
# a, c를 행 방향(아래)으로 붙여서 2차원 배열을 만든다.
print("1-8. np.r_[[a], c] \n", np.r_[[a], c])
print()
# a, c를 행 방향(아래)으로 붙여서 2차원 배열을 만든다.
print("1-9. np.r_[c, d] \n", np.r_[c, d])
print()
# np.c_[] = 열 방향(가로)으로 붙여서 1차원 배열로 만드는 함수
# 배열 c, b 열 방향(가로)으로 붙여서 1차원 배열로 만든다.
print("1-10. np.c_[c, d] \n", np.c_[c, d])
print()
# np.concatenate((), axis=0)
# 결과가 1차원 배열이면 가로로 붙이고, 2차원 배열이면 세로로 붙인다.
# a, b를 가로로 붙여서 1차원 배열을 만든다.
print("1-11. np.concatenate((a,b), axis=0) \n", np.concatenate((a,b),
axis=0))
print()
# 0 ~ 5까지 0.5씩 증가하여 가로로 붙인다.
print("1-12. np.r_[0:5:0.5] \n", np.r_[0:5:0.5])
print()
# 0 ~ 5까지 0.5씩 증가하여 세로로 붙인다.
print("1-13. np.c_[0:5:0.5] \n", np.c_[0:5:0.5])
print()
# np.vstack() = 수직으로 붙여서 하나의 배열로 만드는 함수
# a, b를 수직으로 붙여서 1차원 배열로 만든다.
print("2-1. np.vstack([a,b]) \n", np.vstack([a,b]))
print()

# c, d를 수직으로 붙여서 1차원 배열로 만든다.
print("2-2. np.vstack([c,d]) \n", np.vstack([c,d]))
print()

# a, c를 수직으로 붙여서 1차원 배열로 만든다.
print("2-3. np.vstack([a,c]) \n", np.vstack([a,c]))
print()

# a, b를 수직으로 붙여서 2차원 배열로 만든다.
print("2-4. np.r_[[a], [b]] \n", np.r_[[a], [b]])
print()

# a, c를 수직으로 붙여서 2차원 배열로 만든다.
print("2-5. np.r_[[a], c] \n", np.r_[[a], c])
print()

# c, d를 수직으로 붙여서 1차원 배열로 만든다.
print("2-6. np.r_[c, d] \n", np.r_[c, d])
print()
```

```
# a, b를 행 기준(axis=0)으로 붙여서 1차원 배열을 만든다.
print("2-7. np.concatenate((a,b), axis=0) \n", np.concatenate((a,b),
axis=0))
print()

print("2-8. np.concatenate((a,b), axis=1) \n", "1차원 배열은 열이 없기
때문에 열 기준(axis=1)으로 붙일 수 없다.")
print()

# a, b를 행 기준으로 붙이고 2차원 배열을 만든다.
print("2-9. np.concatenate(([a],[b]), axis=0) \n",
np.concatenate(([a],[b]), axis=0))
print()

# a, b를 열 기준으로 붙이고 2차원 배열을 만든다.
print("2-10. np.concatenate(([a],[b]), axis=1) \n",
np.concatenate(([a],[b]), axis=1))
print()

# np.column_stack() = 여러 배열을 열 방향(가로, axis=1)으로 붙여 2차원 배열을
만드는 함수
# a, b가 1차원 배열이면 각각 열 벡터로 변환하여 2차원 배열을 만든다.
print("3-1. np.column_stack([a,b]) \n", np.column_stack([a, b]))
print()

# c, d가 2차원 배열이면, 그대로 열 방향(가로)으로 붙여 2차원 배열을 만든다.
print("3-2. np.column_stack([c,d]) \n", np.column_stack([c, d]))
print()

# np.c_[] = 여러 배열을 열 방향(가로, axis=1)으로 붙여 2차원 배열을 만드는 도구
# a, b가 1차원 배열이면, 각각 열 벡터로 변환되어 2차원 배열이 된다.
print("3-3. np.c_[a, b] \n", np.c_[a, b])
print()

# c, d가 2차원 배열이면, 그대로 열 방향(가로)으로 붙여 2차원 배열을 만든다.
print("3-4. np.c_[c, d] \n", np.c_[c, d])
print()

# a_reshaped와 b_reshaped를 열 방향(가로, axis=1)으로 붙여서 하나의 2차원
배열로 만든다.
print("3-5. np.concatenate((a_reshaped, b_reshaped), axis=1 \n",
"a_reshaped, b_reshaped가 선언되지 않아 오류")
```

```
1-1. np.hstack([a,b])
[1 2 3 4 5 6]

1-2. np.hstack([c,d])
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]

1-3. np.hstack([a,d])
 행 개수가 다르기 때문에 오류이다.

1-4. np.hstack([a,b[:-1]])
 [1 2 3 4 5]

1-5. np.hstack([c,d[:,1:]])
 행 개수가 다르기 때문에 오류이다.

1-6. np.r_[a, b]
 [1 2 3 4 5 6]

1-7. np.r_[[a], [b]]
 [[1 2 3]
 [4 5 6]]

1-8. np.r_[[a], c]
 [[1 2 3]
 [1 2 3]
 [4 5 6]]

1-9. np.r_[c, d]
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

1-10. np.c_[c, d]
 [[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
```

```
1-11. np.concatenate((a,b), axis=0)
 [1 2 3 4 5 6]

1-12. np.r_[0:5:0.5]
 [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]

1-13. np.c_[0:5:0.5]
 [[0. ]
 [0.5]
 [1. ]
 [1.5]
 [2. ]
 [2.5]
 [3. ]
 [3.5]
 [4. ]
 [4.5]]

2-1. np.vstack([a,b])
 [[1 2 3]
 [4 5 6]]

2-2. np.vstack([c,d])
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

2-3. np.vstack([a,c])
 [[1 2 3]
 [1 2 3]
 [4 5 6]]

2-4. np.r_[[a], [b]]
 [[1 2 3]
 [4 5 6]]
```

```
2-5. np.r_[[a], c]
 [[1 2 3]
 [1 2 3]
 [4 5 6]]

2-6. np.r_[c, d]
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

2-7. np.concatenate((a,b), axis=0)
 [1 2 3 4 5 6]

2-8. np.concatenate((a,b), axis=1)
 1차원 배열은 열이 없기 때문에 열 기준(axis=1)으로 붙일 수 없다.

2-9. np.concatenate(([a],[b]), axis=0)
 [[1 2 3]
 [4 5 6]]

2-10. np.concatenate(([a],[b]), axis=1)
 [[1 2 3 4 5 6]]

3-1. np.column_stack([a,b])
 [[1 4]
 [2 5]
 [3 6]]

3-2.    np.column_stack([c,d])
 [[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]

3-3. np.c_[a, b]
 [[1 4]
 [2 5]
 [3 6]]
```

```
3-4. np.c_[c, d]
 [[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]

3-5. np.concatenate((a_reshaped, b_reshaped), axis=1
 a_reshaped, b_reshaped가 선언되지 않아 오류
```

문제 NP14-0001

np1 배열과 같이 체스보드 판을 그리는 프로그램을 작성하시오.

- 조건
  - 방법1 : zeros() 이용 (0 배열을 만들고, 1로 값을 치환하는 방법)
  - 방법2 : 리스트를 이용 (LC 2차원 행렬을 만드는 방법)
  - 방법3 : vstack 을 이용
  - 방법4 : hstack 을 이용
  - 방법5: concatenate 이용

1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0

$\cdots \quad np1$

```
# 방법 1 : zeros() 이용 ( 0 배열을 만들고, 1 로 값을 치환하는 방법)
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기
# 정수형 3 행, 8 열 배열을 0 으로 채우고 np1 에 저장
np1 = np.zeros((3, 8), dtype=int)
np1[0, :] = 1  # np1 의 첫 번째 행의 값을 1 로 저장
np1[-1, :] = 1 # np1 의 마지막 행의 값을 1 로 저장
np1[:, 0] = 1  # np1 의 첫 번째 열의 값을 1 로 저장
np1[:, -1] = 1 # np1 의 마지막 열의 값을 1 로 저장
print(np1)
```

```
[[1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1]]
```

```
# 방법 2 : 리스트를 이용 (LC 2 차원 행렬을 만드는 방법)
rows, cols = 3, 8 # 행은 3, 열은 8
np1 = [
    # 한 행(row)을 만듭니다. 각 열(column)마다
    # 행 인덱스가 0 이거나 마지막이거나 열 인덱스가 0 이거나 마지막이면 1, 그렇지
않으면 0 값을 결정한다.
    [1 if r== 0 or r == rows-1 or c == 0 or c == cols-1 else 0 for c in
range(cols)]
    # 위 행을 전체 행(rows)만큼 반복해서 2 차원 리스트를 만듭니다.
    for r in range(rows)
]

# np1 리스트의 각 행(row)을 출력합니다.
for row in np1:
  print(row) # 한 행씩 출력
```

```
[1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 0, 0, 0, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1]
```

```
# 방법 3 : vstack 을 이용
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기
rows, cols = 3, 8 # 행은 3, 열은 8
# 정수형 1 행, 8 열 배열을 1 로 채우고 top_bottom 저장
top_bottom = np.ones((1, cols), dtype=int)

# 정수형 1 행, 8 열 배열을 0 으로 채우고 middle 저장
middle = np.zeros((1, cols), dtype=int)
middle[0, 0] = 1 # 1 행 1 열은 1 로 저장
middle[0, -1] = 1 # 1 행 마지막열은 1 로 저장

# top_bottom, middle, top_bottom 을 수직으로 붙여서 np1 에 저장
np1 = np.vstack([top_bottom, middle, top_bottom])

print(np1) # np1 출력
```

```
[[1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1]]
```

```
# 방법 4 : hstack 을 이용
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

rows, cols = 3, 8 # 행은 3, 열은 8

# 정수형 3 행, 1 열 배열을 1 로 채우고 side 저장
side = np.ones((rows, 1), dtype=int)

# 정수형 3 행 6 열 배열을 0 으로 채우고 middle 저장
middle = np.zeros((rows, cols-2), dtype=int)

# side, middle, side 배열을 수평으로 붙여서 np1 에 저장
np1 = np.hstack([side, middle, side])
np1[0, :] = 1 # np1 의 1 행은 전부 1 로 저장
np1[-1, :] = 1 # np1 의 마지막행은 전부 1 로 저장
print(np1) # np1 출력
```

```
[[1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1]]
```

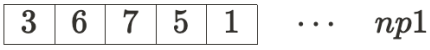
```
# 방법 5: concatenate 이용
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

rows, cols = 3, 8 # 행은 3, 열은 8

# 정수형 3 행, 1 열 배열을 1 로 채우고 side 저장
side = np.ones((rows, 1), dtype=int)

# 정수형 3 행 6 열 배열을 0 으로 채우고 middle 저장
middle = np.zeros((rows, cols-2), dtype=int)

# side, middle, side 배열을 행 기준으로 붙여서 np1 에 저장
np1 = np.concatenate([side, middle, side], axis=1)
np1[0, :] = 1 # np1 의 1 행은 전부 1 로 저장
np1[-1, :] = 1 # np1 의 마지막행은 전부 1 로 저장
print(np1) # np1 출력
```

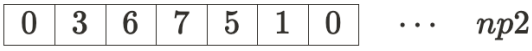
```
[[1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 0 1]
 [1 1 1 1 1 1 1 1]]
```

문제 NP14-0002

다음과 같은 random 배열을 만들고, 배열의 처음과 끝에 배열에 존재하지 않는 임의의 원소를 채우는 프로그램을 작성하시오.

- 조건
  - np.pad() 함수 사용
  - randint() 사용x






```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0~9 까지 숫자 중에서 5 개를 임의로 뽑아서 np1 에 저장
np1 = np.random.choice(np.arange(10), size=5, replace=True)

# np.pad 로 앞뒤에 0 추가
np2 = np.pad(np1, (1, 1), mode='constant', constant_values=0)

print(np1)  # np1 출력
print(np2)  # np2 출력
```

```
[7 7 3 5 9]
[0 7 7 3 5 9 0]
```

문제 NP14-0003

다음과 같은 random 배열을 만들고, 배열의 처음과 끝에 배열에 존재하지 않는 임의의 원소를 채우는 프로그램을 작성하시오.

- 조건
  - np.pad() 함수 사용
  - randint() 사용x

7	0	8	7
8	1	6	5
3	2	7	0

$\cdots \quad np1$

4	4	4	4	4	4
4	7	0	8	7	4
4	8	1	6	5	4
4	3	2	7	0	4
4	4	4	4	4	4

$\cdots \quad np2$

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0~9 까지 숫자 중에서 무작위로 12 개(3x4)를 뽑아 배열 생성
# replace=True: 중복 허용
np1 = np.random.choice(np.arange(10), size = (3, 4), replace=True)

# np1 배열 주변을 4 로 패딩
# 상하좌우 1 줄씩 추가 (패딩 후 크기: 5x6)
# mode='constant': 고정값 채우기
# constant_values=4: 채울 값 지정
np2 = np.pad(np1, ((1, 1), (1, 1)), mode='constant', constant_values=4)

print(np1) # np1 출력
print(np2) # np2 출력
```

```
[[6 2 2 7]
 [2 0 6 8]
 [7 3 0 3]]
[[4 4 4 4 4 4]
 [4 6 2 2 7 4]
 [4 2 0 6 8 4]
 [4 7 3 0 3 4]
 [4 4 4 4 4 4]]
```

문제 NP14-0004

다음과 같은 5x6 배열 np1을 만들고, 3 값을 3의 위치에 있는 주변값들의 평균값으로 치환한 배열 np2 를 만드는 프로그램을 작성하시오.

- 조건
  - 연산 순서는 좌상 부터 우하로 진행한다.
  - 연산에 필요한 테두리값은 가장 가까운 값을 사용하도록 한다.
  - 주변값(Mask)은 자신값을 포함한 9개를 사용한다.
  - np.pad() 함수 사용
  - randint() 사용x

4	10	11	0	13	9
13	13	13	6	12	6
3	0	14	3	1	0
4	12	6	12	2	4
8	3	12	11	7	14

$\cdots \quad np1$

4.00	10.00	11.00	0.00	13.00	9.00
13.00	13.00	13.00	6.00	12.00	6.00
7.22	0.00	14.00	7.67	1.00	0.00
4.00	12.00	6.00	12.00	2.00	4.00
8.00	7.56	12.00	11.00	7.00	14.00

$\cdots \quad result$

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0~9 까지 숫자 중에서 무작위로 30 개(5x6)를 뽑아 배열 생성
# replace=True: 중복 허용
np1 = np.random.choice(np.arange(15), size = (5, 6), replace=True)

# 가장 가까운 값으로 상하좌우 1 줄씩 패딩
pad_np1 = np.pad(np1, ((1,1),(1,1)), mode='edge')

# np1 과 같은 크기의 0.0(실수형) 배열을 np2 에 생성
np2 = np.zeros_like(np1, dtype=float)
for i in range(5): # i 는 0~4 까지 반복
  for j in range(6): # j 는 0~5 까지 반복
    # (i, j)를 좌상단으로 하는 3x3 영역(부분 배열)을 mask 에 저장
    mask = pad_np1[i:i+3, j:j+3]
    # mask 배열들의 평균값을 np2 에 저장
    np2[i, j] = np.mean(mask)

# 소수점 2 자리로 출력 (과학적 표기법 억제)
np.set_printoptions(precision=2, suppress=True)
print(np1) # np1 출력

# np.array2string : 배열을 문자열로 변환
# formatter 를 사용해 모든 실수를 항상 소수점 둘째자리까지(빈자리는 0 으로) 문자열로 변환
print(np.array2string(np2, formatter={'float_kind': lambda x: "%.2f" % x}))
```

```
[[ 8  6  7 10  6  5]
 [ 2  6  3  4 13 10]
 [ 6  7  8  6 12  7]
 [ 5 13  8  8  5  0]
 [ 0  0  7  0 12  8]]
[[6.00 5.89 6.56 7.33 7.67 7.22]
 [5.67 5.89 6.33 7.67 8.11 8.33]
 [5.78 6.44 7.00 7.44 7.22 7.11]
 [4.67 6.00 6.33 7.33 6.44 6.56]
 [2.56 4.44 4.78 6.56 5.89 6.78]]
```

문제 NP15-0001

그림과 같이 2차원 배열을 변형하시오.

- 조건
  - np1 : 2차원 배열의 리스트 값 생성 ***(Nested List Comprehension 사용)***
  - np2 : 중심부의 값을 0 로 만드시오. (슬라이싱)

1	2	3	4
5	6	7	8
9	10	11	12

$\cdots \quad np1$

1	2	3	4
5	0	0	8
9	10	11	12

$\cdots \quad np2$

```python
import copy # copy 라이브러리 불러오기

# np1: 2 차원 배열의 리스트 값 생성 (Nested List Comprehension 사용)
# i(0~2)가 행을, j(0~3)가 열을 생성하며, 각 요소는 i*4 + j +1 로 계산
np1 = [[i * 4 + j + 1 for j in range(4)] for i in range(3)]

# np2: np1 의 깊은 복사 후 중심부(2 행 2-3 열) 값을 0 으로 변경
np2 = copy.deepcopy(np1)
np2[1][1:3] = [0, 0] # 2 행(인덱스 1)의 2-3 열(인덱스 1:3)을 0 으로 할당

# 결과 출력
for k in np1:
  print(k)
print()
for k in np2:
  print(k)
```

```
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]

[1, 2, 3, 4]
[5, 0, 0, 8]
[9, 10, 11, 12]
```

문제 NP15-0002

그림과 같이 2차원 배열을 변형하시오.

- 조건
  - np1 : 2차원 배열의 리스트 값 생성 ***(Nested List Comprehension 사용)***
  - np2 : 상하좌우 테두리에 행과 열을 추가하는 리스트를 만드시오. (numpy 사용)

1	2	3	4
5	6	7	8
9	10	11	12

$\cdots \quad np1$

1	1	2	3	4	4
1	1	2	3	4	4
5	5	6	7	8	8
9	9	10	11	12	12
9	9	10	11	12	12

$\cdots \quad np2$

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# np1: 2 차원 배열의 리스트 값 생성 (Nested List Comprehension 사용)
# i(0~2) 행을 j(0~3) 열을 생성하며, 각 요소는 i*4+j+1 로 계산하여 np1 에 저장
np1 = [[i * 4 + j + 1 for j in range(4)] for i in range(3)]

# np1 의 상하좌우 행과 열의 1 줄씩 가장자리와 같은 값으로 추가하여 np2 에 저장
np2 = np.pad(np1, pad_width=((1, 1), (1, 1)), mode='edge')

# 결과 출력
for k in np1:
  print(k)
print(np2)
```

```
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[[ 1  1  2  3  4  4]
 [ 1  1  2  3  4  4]
 [ 5  5  6  7  8  8]
 [ 9  9 10 11 12 12]
 [ 9  9 10 11 12 12]]
```

문제 NP15-0003

그림과 같이 2차원 배열을 만들고, 1차원 리스트 정보를 출력해보자.

- 조건
  - np1 : 2차원 배열의 리스트 값 생성 ***(Nested List Comprehension 사용)***
  - 왼쪽 상단위치에서 시작해서 시계방향으로의 값을 갖는 리스트를 만드시오.
  - 슬라이싱 이용
  - 리스트의 총 합을 출력하시오.

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline \end{array} \quad \cdots \quad np1$$

- 입출력 예

```
[1, 2, 3, 4, 8, 12, 11, 10, 9, 5]
sum 65
```

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# i(0~2) 행을 j(0~3) 열을 생성하며, 각 요소는 i*4+j+1 로 계산하여 np1 에 저장
np1 = [[i * 4 + j + 1 for j in range(4)] for i in range(3)]

result = [] # 시계방향 테두리 값을 저장할 리스트
sum = 0 # 테두리 값의 합계 계산용 변수

# 1. 맨 위 행(0 행) 전체 추가 (왼→오)
result += np1[0]

# 2. 오른쪽 열(3 열) 추가 (위→아래, 1 행~2 행)
for i in range(1, 3):
  result.append(np1[i][3])

# 3. 맨 아래 행(2 행) 역순 추가 (오→왼, 2 열~0 열)
result += np1[2][2::-1]

# 4. 왼쪽 열(0 열) 추가 (아래→위, 1 행만 추가)
for i in range(1, 0, -1): # i=1 만 반복 (0 행은 이미 추가됨)
  result.append(np1[i][0])

# np1 배열 출력
for row in np1:
    print(row)

# result 출력
print(result)

# 합계 계산 및 출력
for num in result:
    sum += num
print(sum)
```

```
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[1, 2, 3, 4, 8, 12, 11, 10, 9, 5]
65
```

문제 NP15-0004

np1 배열을 만들고, 행의합과 열의합을 순서대로 결합해서 새로운 행렬(np2) 을 출력해보자.

- 조건
  - 방법1 : numpy 배열이용 (반복문x)
  - ~~방법2 : pandas 이용~~

0	1	2	3
4	5	6	7
8	9	10	11

$\cdots \quad np1$

0	1	2	3	6
4	5	6	7	22
8	9	10	11	38
12	15	18	21	66

$\cdots \quad np3$

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0~12 의 값을 3 행 4 열로 만들어 np1 에 저장
np1 = np.arange(12).reshape(3, 4)

row_sums = np1.sum(axis=1) # np1 의 행 기준의 합
col_sums = np1.sum(axis=0) # np1 의 열 기준의 합

np2 = np.vstack([np1, col_sums]) # np1 의 열의 합을 np1 에 수직으로 붙여서
np2 에 저장

total_sum = np1.sum() # np1 의 전체 합을 total_sum 에 저장

# row_sums 에 total_sum 을 추가하고, 수직행렬로 만듬
row_sums = np.append(row_sums, total_sum).reshape(-1, 1)

# np2 에 row_sums 를 수평으로 붙여서 np2 에 저장
np2 = np.hstack([np2, row_sums])

# 결과 출력
print(np1)
print(np2)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[[ 0  1  2  3  6]
 [ 4  5  6  7 22]
 [ 8  9 10 11 38]
 [12 15 18 21 66]]
```

문제 NP16-0001

다음 그림과 같이 3차원 np1 배열을 만들고, 각 평면에 2와 4를 더한 새로운 3차원 배열을 만드는 프로그램을 작성하시오.

- 조건
  - 브로드케스팅 이용

0	1	2	3
4	5	6	7
8	9	10	11

12	13	14	15
16	17	18	19
20	21	22	23

$\cdots \quad np1$

2	3	4	5
6	7	8	9
10	11	12	13

16	17	18	19
20	21	22	23
24	25	26	27

$\cdots \quad np2$

```python
import numpy as np # numpy 라이브러리를 np 라는 이름으로 불러오기

# 0~23 까지의 수를 2 블록 3 행 4 열 3 차원 배열로 생성
# ▶️ reshape(2,3,4): (블록, 행, 열)
np1 = np.arange(24).reshape(2, 3, 4)
print(np1)

# 블록별 더할 값 [2, 4]를 3 차원 형태(2,1,1)로 변환
# ▶️ -1: 차원 자동 계산 (2 개 블록)
# ▶️ 1x1: 브로드캐스팅을 위해 행/열 차원 확장
add_values = np.array([2, 4]).reshape(-1, 1, 1)

# 브로드캐스팅으로 각 블록에 다른 값 더하기
# ▶️ 첫 번째 블록: 모든 요소 +2
# ▶️ 두 번째 블록: 모든 요소 +4
np2 = np1 + add_values

print(np2)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
[[[ 2  3  4  5]
  [ 6  7  8  9]
  [10 11 12 13]]

 [[16 17 18 19]
  [20 21 22 23]
  [24 25 26 27]]]
```