# Python Library
# Numpy

# Scalar, Vector, Matrix

- 스칼라 (Scalar):
  - 스칼라는 크기만을 가지고 방향을 갖지 않는 수
  - 실수, 정수, 복소수 등이 스칼라의 예시
  - 스칼라는 한 개의 값을 갖음
- 벡터 (Vector):
  - 벡터는 크기와 방향을 가지는 수
  - 여러 개의 값을 가지며, 순서가 중요
- 행렬 (Matrix):
  - 행렬은 2차원 배열로 구성되며, 행과 열의 요소를 가짐
  - 각 요소는 스칼라 값이며, 행과 열에 따라 위치가 결정
  - 행렬은 선형 변환, 선형 방정식, 데이터 표현 등에 널리 사용

# Python - Scalar, Vector, Matrix

- 스칼라 (Scalar):
  - 파이썬에서 스칼라는 단순히 숫자로 표현
  - 정수, 실수, 복소수 등의 숫자 데이터를 사용하여 스칼라를 선언
  - 예를 들어, x = 5 또는 y = 3.14와 같이 변수에 값을 할당하는 방식으로 스칼라를 선언
- 벡터 (Vector):
  - 벡터를 표현하기 위해 파이썬에서 주로 numpy 라이브러리를 사용
  - numpy 배열을 사용하여 벡터를 선언
  - 예를 들어, [1, 2, 3]과 같이 값들을 리스트 형태로 입력하여 벡터를 선언
  - numpy 배열을 사용하려면 import numpy as np와 같이 numpy를 먼저 import해야 함
  - 이후에는 x = np.array([1, 2, 3])과 같이 numpy 배열을 사용하여 벡터를 선언

# Python - Scalar, Vector, Matrix

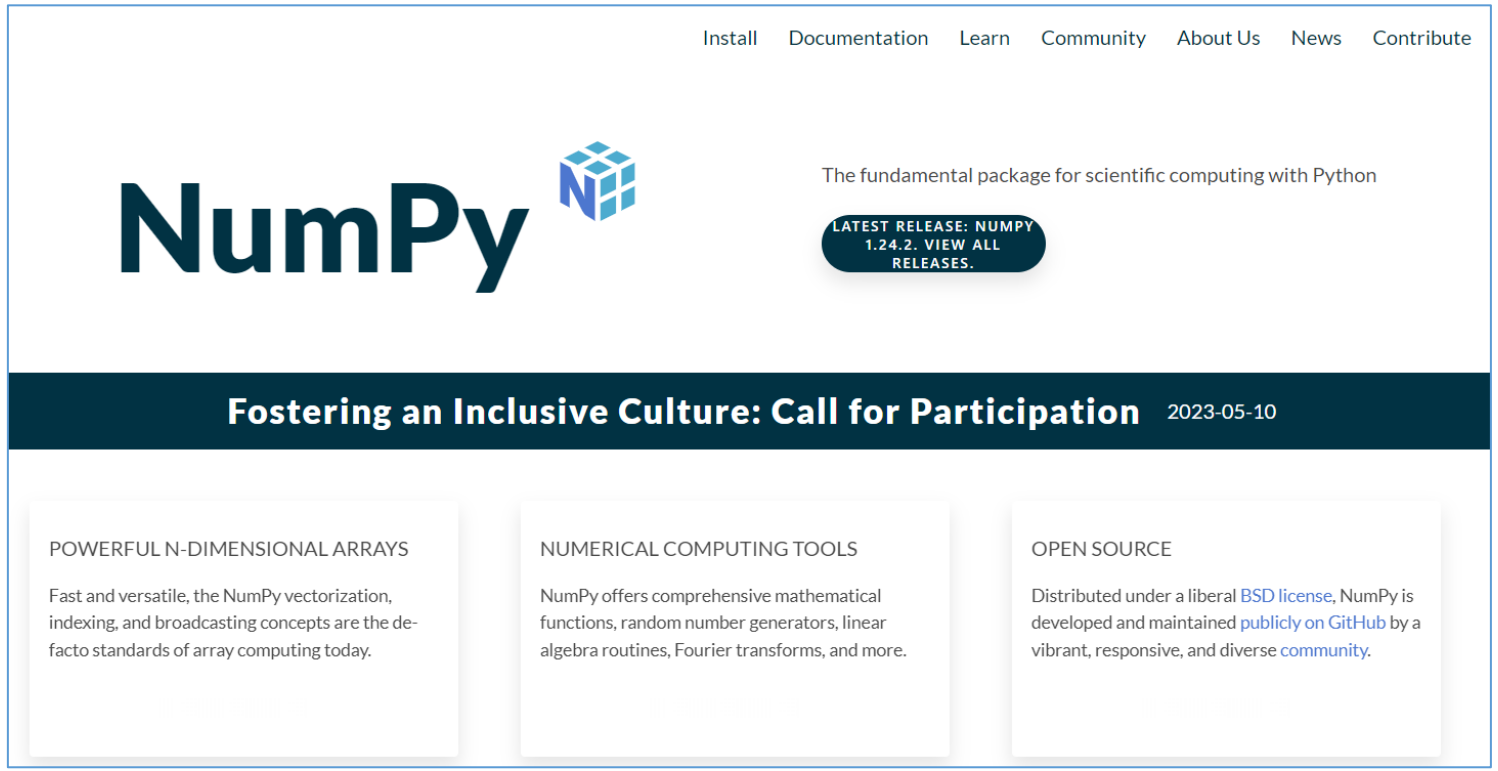
- 행렬 (Matrix):
  - 행렬을 표현하기 위해서도 numpy 라이브러리를 주로 사용
  - numpy의 array 함수를 사용하여 행렬을 선언
  - 예를 들어, [[1, 2], [3, 4]]와 같이 값들을 중첩된 리스트 형태로 입력하여 2x2 행렬을 선언
  - numpy 배열을 사용하기 위해 import numpy as np를 먼저 수행한 후, x = np.array([[1, 2], [3, 4]])와 같이 numpy 배열을 사용하여 행렬을 선언

- numpy는 파이썬에서 벡터와 행렬을 다루는 데 매우 유용한 도구
- numpy 배열을 사용하면 벡터와 행렬에 대한 다양한 연산과 변환을 효율적으로 수행
- numpy를 사용하여 선언한 벡터와 행렬은 다양한 수학 연산을 지원하며, 데이터 분석, 과학 계산, 기계 학습 등의 분야에서 널리 사용됨

# Numpy

- NumPy(넘파이)는 파이썬에서 수치 계산을 위한 핵심 라이브러리로 널리 사용되는 도구
- 다차원 배열 객체와 배열 처리에 대한 다양한 함수와 메서드를 제공하여 과학적이고 수치적인 계산을 효율적으로 수행

https://numpy.org/



# Python List vs Numpy Array

- Python List의 장점과 단점:

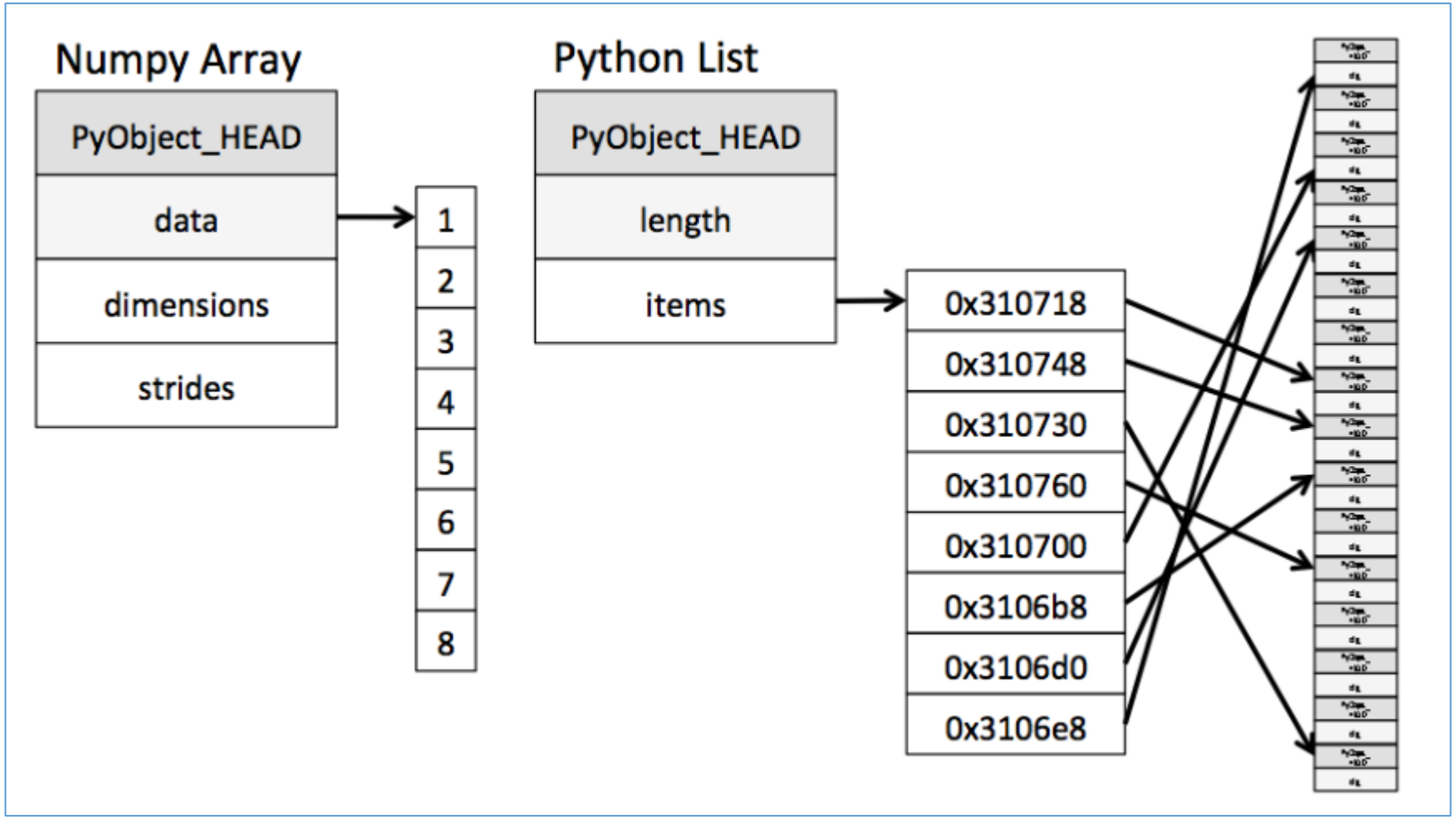
- 장점:
  - 유연성: 리스트는 다양한 데이터 유형을 포함할 수 있으며 크기가 동적으로 조정될 수 있음
  - 내장 함수: Python의 리스트에는 다양한 내장 함수와 메서드가 있어 데이터 조작과 분석을 쉽게 할 수 있음
- 단점:
  - 속도: 리스트의 연산은 반복문을 통해 요소별로 수행되기 때문에 큰 데이터셋에서는 속도가 느릴 수 있음
  - 메모리 사용: 리스트는 각 요소에 대한 정보를 저장하기 위해 추가적인 메모리 공간을 사용하므로 큰 데이터셋에 대해서는 메모리 사용이 비효율적일 수 있음

# Python List vs Numpy Array

- NumPy 배열의 장점과 단점:
- 장점:
  - 성능: NumPy 배열은 C로 구현되어 있어 반복문을 사용하지 않고도 벡터화된 연산을 수행할 수 있어 데이터 처리 속도가 빠르고 효율적
  - 메모리 사용: NumPy 배열은 연속된 메모리 공간에 데이터를 저장하므로 메모리 사용이 효율적
  - 다차원 배열 지원: NumPy는 다차원 배열을 지원하므로 행렬 연산 및 다차원 데이터 처리가 용이
  - 다양한 수학 함수: NumPy는 다양한 수학 함수와 연산을 제공
- 단점:
  - 유연성: NumPy 배열은 한 가지 데이터 유형만 포함할 수 있으며 크기를 동적으로 조정할 수 없음 배열의 크기를 변경하려면 새로운 배열을 생성해야 함
  - 추가 기능의 부족: Python의 리스트에는 다양한 내장 함수와 메서드가 있지만, NumPy 배열은 기본적인 수학 함수와 배열 조작 함수에 중점을 두고 있어 몇 가지 추가 기능이 제한적일 수 있음

# Numpy Array vs Python List

- http://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/



# Python List vs Numpy Array

```python
# Python 리스트 예제
height = [1.65, 1.70, 1.75, 1.80]  # 키 (단위: 미터)
weight = [60.50, 65.75, 70.33, 75.50]  # 몸무게 (단위: kg)

bmi_list= weight / height ** 2

print(bmi_list)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-db5d8d85abe4> in <module>
      3 weight = [60.50, 65.75, 70.33, 75.50]  # 몸무게 (단위: kg)
      4
----> 5 bmi_list= weight / height ** 2
      6
      7 print(bmi_list)

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

# Python List vs Numpy Array

```python
import numpy as np

# Python 리스트 예제
height = [1.65, 1.70, 1.75, 1.80]  # 키 (단위: 미터)
weight = [60.50, 65.75, 70.33, 75.50]  # 몸무게 (단위: kg)

np_height = np.array(height)
np_weight = np.array(weight)

bmi_list= np_weight / np_height ** 2

print(bmi_list)
```

```
[22.22222222 22.75086505 22.96489796 23.30246914]
```

# Numpy 주요 함수

- numpy.array(): 배열을 생성하는 함수이며, 파이썬의 리스트나 튜플과 같은 객체를 입력으로 받아 NumPy 배열로 변환
- numpy.zeros(): 모든 요소가 0인 배열을 생성하는 함수이며, 지정된 크기의 0으로 채워진 배열을 생성
- numpy.ones(): 모든 요소가 1인 배열을 생성하는 함수이며, 지정된 크기의 1로 채워진 배열을 생성
- numpy.arange(): 주어진 범위 내에서 일정한 간격으로 값이 증가하는 1차원 배열을 생성하는 함수
- numpy.linspace(): 주어진 범위 내에서 지정된 개수의 일정한 간격으로 값이 증가하는 1차원 배열을 생성하는 함수
- numpy.reshape(): 배열의 형태를 변경하는 함수이며, 배열의 차원과 크기를 재조정할 수 있음
- numpy.transpose(): 배열의 Transpose를 수행하는 함수이며, 배열의 차원을 변경하거나 축의 순서를 변경할 때 사용
- numpy.concatenate(): 여러 배열을 연결하여 하나의 배열로 만드는 함수이며, 주어진 축을 따

# Numpy 주요 함수

- numpy.sum(): 배열의 모든 요소의 합을 계산하는 함수이며, 선택적으로 축을 지정하여 특정 축을 따라 합을 계산
- numpy.mean(): 배열의 평균을 계산하는 함수이며, 선택적으로 축을 지정하여 특정 축을 따라 평균을 계산
- numpy.max(): 배열의 최댓값을 찾는 함수이며, 선택적으로 축을 지정하여 특정 축을 따라 최댓값 찾음
- numpy.min(): 배열의 최솟값을 찾는 함수이며, 선택적으로 축을 지정하여 특정 축을 따라 최솟값 찾음
- numpy.matmul(): 다차원 배열에 대한 행렬 곱셈을 수행
- numpy.sort(): 배열을 정렬하는 함수이며, 선택적으로 축을 지정하여 특정 축을 따라 정렬
- numpy.argmax(): 배열에서 최댓값의 인덱스를 반환하는 함수이며, 선택적으로 축을 지정하여 특정 축을 따라 최댓값의 인덱스를 찾음
- numpy.argmin(): 배열에서 최솟값의 인덱스를 반환하는 함수이며, 선택적으로 축을 지정하여 특정 축을 따라 최솟값의 인덱스를 찾음

# Numpy array 생성과 초기화 함수

- NumPy는 다차원 배열을 생성하고 초기화하는 기능을 제공
- np.array 함수를 사용하여 파이썬 리스트를 배열로 변환 가능
- np.zeros, np.ones, np.random 등의 함수를 사용하여 배열을 생성하고 초기화

# Example

```python
import numpy as np

# 0으로 초기화된 3x3 배열 생성
zeros_arr = np.zeros((3, 3))
print("zeros_arr:")
print(zeros_arr)

# 1로 초기화된 2x4 배열 생성
ones_arr = np.ones((2, 4))
print("₩nones_arr:")
print(ones_arr)

# 0부터 9까지의 랜덤한 값을 가지는 2x3 배열 생성
random_arr = np.random.randint(10, size=(2, 3))
print("₩nrandom_arr:")
print(random_arr)
```

```
zeros_arr:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

ones_arr:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]

random_arr:
[[9 0 8]
 [6 6 5]]
```

# Example

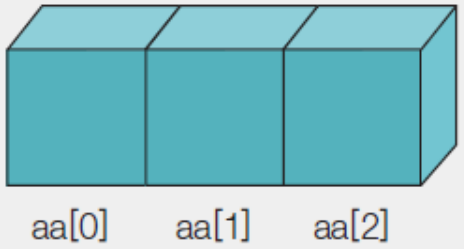
```python
import numpy as np

# 배열 연산
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# 요소별 덧셈
result = arr1 + arr2
print("Element-wise addition:")
print(result)

# 행렬 곱셈
result = np.dot(arr1, arr2)
print("₩nDot product:")
print(result)

# 전치 행렬
arr3 = np.array([[1, 2], [3, 4]])
transposed_arr = arr3.T
print("₩nTransposed matrix:")
print(transposed_arr)
```

```
Element-wise addition:
[5 7 9]

Dot product:
32

Transposed matrix:
[[1 3]
 [2 4]]
```

# 2차원 리스트 , 배열

- 2차원 리스트, 배열의 기본 개념
  - 1차원 리스트(배열)를 여러 개 연결한 것, 두 개의 첨자를 사용하는 리스트

```
aa = [10, 20, 30]
```




1차원

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


2차원

# Example

```python
import numpy as np

# 2차원 배열 생성
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12]])

# 인덱싱
print("1행 2열의 요소:", arr[1, 2])  # 출력: 7

# 슬라이싱
print("0행 전체:", arr[0, :])  # 출력: [1 2 3 4]
print("3열 전체:", arr[:, 3])  # 출력: [4 8 12]
print("1행부터 2행까지, 1열부터 3열까지의 부분 배열:")
print(arr[1:3, 1:4])
# 출력:
# [[ 6  7  8]
#  [10 11 12]]
```

```
1행 2열의 요소: 7
0행 전체: [1 2 3 4]
3열 전체: [ 4  8 12]
1행부터 2행까지, 1열부터 3열까지의 부분 배열:
[[ 6  7  8]
 [10 11 12]]
```

# Example

```python
import numpy as np

birth_rates = np.array([1.24, 1.30, 1.19, 1.21, 1.24, 1.17, 1.05, 0.98, 0.92, 0.84, 0.81, 0.78])

mean = np.mean(birth_rates)
median = np.median(birth_rates)
max_value = np.max(birth_rates)
min_value = np.min(birth_rates)
std_dev = np.std(birth_rates)
variance = np.var(birth_rates)

print("한국 출산율 데이터 (2011년부터 2022년까지):")
print(birth_rates)
print("평균:", mean)
print("중앙값:", median)
print("최댓값:", max_value)
print("최솟값:", min_value)
print("표준편차:", std_dev)
print("분산:", variance)
```

```
한국 출산율 데이터 (2011년부터 2022년까지):
[1.24 1.3  1.19 1.21 1.24 1.17 1.05 0.98 0.92 0.84 0.81 0.78]
평균: 1.0608333333333333
중앙값: 1.1099999999999999
최댓값: 1.3
최솟값: 0.78
표준편차: 0.18011377577026755
분산: 0.03244097222222222
```

# Example(Stack_array)

```python
import numpy as np

class Stack:
    def __init__(self, size):
        self.size = size
        self.stack = np.empty(size, dtype=np.object)
        self.top = -1

    def is_empty(self):
        return self.top == -1

    def is_full(self):
        return self.top == self.size - 1

    def push(self, element):
        if self.is_full():
            print("Stack is full. Cannot push element.")
        else:
            self.top += 1
            self.stack[self.top] = element
```

# Example(Stack_array)

```
    def pop(self):
        if self.is_empty():
            print("Stack is empty. Cannot pop element.")
        else:
            element = self.stack[self.top]
            self.stack[self.top] = None
            self.top -= 1
            return element

    def peek(self):
        if self.is_empty():
            print("Stack is empty.")
        else:
            return self.stack[self.top]
```

# Example(Stack_array)

```
# 스택 객체 생성
stack = Stack(5)

# 요소 추가
stack.push(10)
stack.push(20)
stack.push(30)
stack.push(40)

# 요소 제거
popped_element = stack.pop()
print("Popped element:", popped_element)

# 스택 탑 확인
top_element = stack.peek()
print("Top element:", top_element)
```

```
Popped element: 40
Top element: 30
```

# Example(Stack_array)

- __init__(self, size): 스택의 크기를 입력받아 초기화
- 스택은 np.empty 함수를 사용하여 size 크기의 비어있는 배열로 생성
- top 변수는 스택의 가장 위의 요소의 인덱스를 나타내며, 초기값은 -1로 설정
- is_empty(self): 스택이 비어있는지 여부를 확인하며, 스택이 비어있으면 True를 반환하고, 그렇지 않으면 False를 반환
- is_full(self): 스택이 가득 찼는지 여부를 확인하며, 스택이 가득 차있으면 True를 반환하고, 그렇지 않으면 False를 반환
- push(self, element): 스택에 요소를 추가하며, 스택이 가득 차 있으면 "Stack is full. Cannot push element." 메시지를 출력하고, 그렇지 않으면 top 인덱스를 증가시키고 해당 인덱스에 요소를 저장

# Example(Stack_array)

- pop(self): 스택에서 요소를 제거하고 반환하며, 스택이 비어있으면 "Stack is empty. Cannot pop element." 메시지를 출력하고, 그렇지 않으면 top 인덱스에 해당하는 요소를 반환하고 해당 요소를 None으로 설정한 후 top 인덱스를 감소시킴

- peek(self): 스택의 가장 위에 있는 요소를 반환하며, 스택이 비어있으면 "Stack is empty." 메시지를 출력하고, 그렇지 않으면 top 인덱스에 해당하는 요소를 반환

- Stack 클래스를 사용하여 스택 객체를 생성하고, push() 메서드를 사용하여 요소를 추가하고, pop() 메서드를 사용하여 요소를 제거하고 반환하며, peek() 메서드를 사용하여 스택의 가장 위에 있는 요소를 확인

# Example(Bubble Sort_array)

```python
import numpy as np

def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# 정렬할 배열 생성
arr = np.array([64, 34, 25, 12, 22, 11, 90])

# 버블 정렬 수행
bubble_sort(arr)

# 정렬 결과 출력
print("정렬된 배열:", arr)
```

```
정렬된 배열: [11 12 22 25 34 64 90]
```

# Example(matrix sum)

**– 행렬의 덧셈**

두 행렬 $A, B$가 같을 꼴일 때, $A$와 $B$의 대응하는 성분의 합을 성분으로 하는 행렬을 '$A$와 $B$에 합'이라 하며, 기호로 $A+B$로 나타낸다.

$$A=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \text{이면} \quad A+B=\begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} \\ a_{21}+b_{21} & a_{22}+b_{22} \end{pmatrix}$$

$$A=\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix},\quad B=\begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix} \text{이면},\quad A+B=\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}+\begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}=\begin{pmatrix} 1+2 & 2+3 \\ 3+4 & 4+5 \end{pmatrix}=\begin{pmatrix} 3 & 5 \\ 7 & 9 \end{pmatrix}$$

# Example(matrix sum)

```python
import numpy as np

# 두 개의 3x3 배열 생성
array1 = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

array2 = np.array([[10, 11, 12],
                   [13, 14, 15],
                   [16, 17, 18]])

# 배열 더하기
array_sum = array1 + array2

# 결과 출력
print("배열1:₩n", array1)
print("배열2:₩n", array2)
print("배열 더하기 결과:₩n", array_sum)
```

```
배열1:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
배열2:
 [[10 11 12]
 [13 14 15]
 [16 17 18]]
배열 더하기 결과:
 [[11 13 15]
 [17 19 21]
 [23 25 27]]
```

# Example(matrix multiplication)

- 행렬의 곱셈(matrix multiplication)은 행렬의 곱셈에서 앞 행렬의 열의 수와 뒤 행렬의 행의 수가 같아야 함

두 행렬 $A, B$가 각각 $m \times n, n \times r$ 행렬일 때,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nr} \end{pmatrix}$$

이라고 하면 행렬의 곱 $AB$는 $m \times r$ 행렬이며,

$$AB = \begin{pmatrix} \sum_k a_{1k}b_{k1} & \sum_k a_{1k}b_{k2} & \cdots & \sum_k a_{1k}b_{kr} \\ \sum_k a_{2k}b_{k1} & \sum_k a_{2k}b_{k2} & \cdots & \sum_k a_{2k}b_{kr} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_k a_{mk}b_{k1} & \sum_k a_{mk}b_{k2} & \cdots & \sum_k a_{mk}b_{kr} \end{pmatrix}$$

이다. (단, $k = 1, 2, ..., n$)

mxn행렬과 nxr행렬 곱을 수행하면 답으로 나오는 행렬은 mxr

# Example(matrix multiplication)

- A 학교에는 우등생을 따로 모아서 교육시키는 특별반인 '우수반'과 '수학반'이 있으며, 이들 반에서 학생을 선발하기 위해서 국어, 수학, 영어 과목에 각각 가중치를 둔다.

| | 국어 | 수학 | 영어 |
|---|---|---|---|
| 김 | 80 | 90 | 60 |
| 이 | 75 | 80 | 90 |
| 박 | 90 | 95 | 65 |
| 최 | 99 | 70 | 70 |

<표 A: 각 학생의 과목별 성적>

| | 우수반 | 수학반 |
|---|---|---|
| 국어 | 3 | 1 |
| 수학 | 3 | 8 |
| 영어 | 4 | 1 |

<표 B: 각 반의 과목별 가중치>

위와 같은 <표 A>와 <표 B>를 이용하여 김, 이, 박, 최 4명의 학생이 이들 각 반에 들어가려고 할 때의 가중치를 이용한 세 과목의 총점을 구하려고 한다.
이때 각 과목별로 (점수 × 가중치)의 총합을 계산

$$A = \begin{pmatrix} 80 & 90 & 60 \\ 75 & 80 & 90 \\ 90 & 95 & 65 \\ 99 & 70 & 70 \end{pmatrix}, B = \begin{pmatrix} 3 & 1 \\ 3 & 8 \\ 4 & 1 \end{pmatrix}$$

일 때,

$$AB = \begin{pmatrix} 80 \cdot 3 + 90 \cdot 3 + 60 \cdot 4 & 80 \cdot 1 + 90 \cdot 8 + 60 \cdot 1 \\ 75 \cdot 3 + 80 \cdot 3 + 90 \cdot 4 & 75 \cdot 1 + 80 \cdot 8 + 90 \cdot 1 \\ 90 \cdot 3 + 95 \cdot 3 + 65 \cdot 4 & 90 \cdot 1 + 95 \cdot 8 + 65 \cdot 1 \\ 99 \cdot 3 + 70 \cdot 3 + 70 \cdot 4 & 99 \cdot 1 + 70 \cdot 8 + 70 \cdot 1 \end{pmatrix} = \begin{pmatrix} 750 & 860 \\ 825 & 805 \\ 815 & 915 \\ 787 & 729 \end{pmatrix}$$

| | 우수반 | 수학반 |
|---|---|---|
| 김 | 750 | 860 |
| 이 | 825 | 805 |
| 박 | 815 | 915 |
| 최 | 787 | 729 |

# Example(matrix multiplication)

```python
import numpy as np

# 학생들의 점수 행렬 생성
scores = np.array([[80, 90, 60],
                   [75, 80, 90],
                   [90, 95, 65],
                   [99, 70, 70]])

# 가중치 행렬 생성
weightings = np.array([[3, 1],
                       [3, 8],
                       [4, 1]])

# 행렬 곱셈으로 계산
result = np.matmul(scores, weightings)

# 각 학생의 결과 출력
for i, student_result in enumerate(result):
    print(f"학생 {i+1}의 계산 결과: {student_result}")
```

```
학생 1의 계산 결과: [750 860]
학생 2의 계산 결과: [825 805]
학생 3의 계산 결과: [815 915]
학생 4의 계산 결과: [787 729]
```

# Example(matrix multiplication)

- 학생들의 점수 행렬 생성:
  - scores는 4x3 크기의 행렬로, 각 행은 한 명의 학생의 국어, 수학, 영어 점수
- 가중치 행렬 생성:
  - weightings는 3x2 크기의 행렬로, 각 행은 국어, 수학, 영어 과목에 대한 가중치
- 행렬 곱셈으로 계산:
  - np.matmul(scores, weightings)를 통해 scores 행렬과 weightings 행렬의 곱을 계산
  - 행렬 곱을 위해서는 앞 행렬의 열의 수와 뒤 행렬의 행의 수가 같아야 함
  - scores의 shape는 (4, 3)이고, weightings의 shape는 (3, 2)이므로 행렬 곱셈이 가능
- 결과로 얻어지는 행렬은 (4, 2) 크기로, 각 행은 한 명의 학생의 국어, 수학, 영어 점수에 가중치가 적용된 결과를 나타냄
- enumerate() 함수를 사용하여 각 학생의 인덱스를 가져옴
- student_result 변수에는 각 학생의 계산 결과인 행렬의 행이 저장되어 있음