WEEKLY SEMINAR

(ABARA)

Jan. 08. 2023

Index

- 👉 Introduction
- 👉 Advanced Python
- 👉 Numpy
- 👉 BootCamp Register
- 👉 Jan. 2th Week Summary

Introduction

- Jan. 2. : Advanced Python Lec.01~02
- Jan. 3. : Advanced Python Lec.03~05
- Jan. 4. : Advanced Python Lec.06~07, Numpy Lec.01
- Jan. 5. : BootCamp Register (2023 KRAFTON AI Fellowship Program)
- Jan. 6. : BoostCamp Register (Naver BoostCamp AI Tech 5th)

Advanced Python

👉 Feature of Arithmetic & Bit Operator

- In-Place 연산
  - `a += 1`은 In-Place 연산이다.
  - 기존 객체를 수정시도하고, 불가능할 시 새로운 객체를 생성한다.
- Out-Place 연산
  - `a = a+1`은 Out-Place 연산이다.
  - 명시적으로 새로운 객체를 생성한다.
- `a++` 연산은 파이썬에 존재하지 않는다.

```
# In-Place 연산
a = 5
a = a+1
a

# 6

a+=1
a

#7
```

```
# Out-Place 연산
a = 7
a = a^4
a

# 3

a^=4
a

# 7
```

Advanced Python

👉 List Comprehension

- 기존 list 사용하여 간단히 다른 list를 만드는 기법
- 포괄적인 list, 포함되는 리스트라는 의미로 사용됨
- for + append보다 속도가 빠르다.

```
case_1 = ["A", "B", "C"]
case_2 = ["D", "E", "A"]
res = [i+j for i in case_1 for j in case_2]
res

# ['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
```

```
res = [i+j for i in case_1 for j in case_2 if not(i==j)]
# Filter : i와 j가 같다면 list에 추가하지 않음
res.sort()
res

# ['AD', 'AE', 'BA', 'BD', 'BE', 'CA', 'CD', 'CE']
```

```
words = "The quick brown fox jumps over the lazy dog".split()
words

# ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
```

Advanced Python

👉 Enumerate

List의 element를 추출할 때 번호를 붙여서 추출함.

```
# Enumerate
for i, v in enumerate(['tic', 'tac', 'toe']):
    print(i, v)

# 0 tic
# 1 tac
# 2 toe
```

```
mylist = ["a", "b", "c", "d"]
list(enumerate(mylist))

# [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

```
{i:j for i, j in enumerate('Gachon University is an academic institute located in South

# Result ▼
{0: 'Gachon',
 1: 'University',
 2: 'is',
 3: 'an',
 4: 'academic',
 5: 'institute',
 6: 'located',
 7: 'in',
 8: 'South',
 9: 'Korea.'}
```

Advanced Python

👉 Zip()

2개의 list의 값을 병렬적으로 추출함.

```
a,b,c = zip((1,2,3), (10,20,30), (100,200,300))
print(a, b, c)

# (1, 10, 100) (2, 20, 200) (3, 30, 300)
```

```
[sum(x) for x in zip((1,2,3), (10,20,30), (100,200,300))]

# [111, 222, 333]
```

```
alist = ['a1', 'a2', 'a3']
blist = ['b1', 'b2', 'b3']

for i, (a, b) in enumerate(zip(alist, blist)):
    print(i, a, b)

# 0 a1 b1
# 1 a2 b2
# 2 a3 b3
```

Advanced Python

👉 Lambda

- 함수 이름없이 사용할 수 있는 익명함수
- python3부터 권장하지 않음.

```
# General function
def f(x, y):
    return x+y

print(f(1,2))

# 3
```

```
# Lambda function
f = lambda x ,y:x+y
print(f(1,3))

# 4
```

```
def set_RecentAcademic():
    academic = list(db['academic'].find({}, {"_id":0, "fileName":0, "fileLink":0}).limit(10))
    academic = sorted(academic, key=lambda x: datetime.strptime(x["dateCreated"].lstrip(), "%Y-%m-%d"), reverse=True)
    print(academic)
    return academic
```

Advanced Python

👉 Map()

- Sequence 자료형 각 element에 동일한 function을 적용함

```
ex = [1,2,3,4,5]
f = lambda x:x**2
print(list(map(f, ex)))

# [1, 4, 9, 16, 25]

ex = [1,2,3,4,5]
t = lambda x, y:x+y
print(list(map(t, ex, ex)))

# [2, 4, 6, 8, 10]

print(list(map(lambda x: x**2 if x%2==0 else x, ex)))
# [1, 4, 3, 16, 5]
```

👉 Reduce()

- map function과 달리 list에 똑같은 함수를 적용하여 통합함

```
from functools import reduce
print(reduce(lambda x, y: x+y, [1,2,3,4,5]))

# 15
```

```
def factorial(n):
    return reduce(lambda x, y: x*y, range(1, n+1))

factorial(5)

# 120
```

Advanced Python

👉 Asterisk

- 흔히 알고있는 `*`를 말함
- 곱셈 및 거듭제곱 연산으로 사용할 때
- 리스트형 컨테이너 타입의 데이터를 반복 확장하고자 할 때
- 가변인자 (Variadic Arguments)를 사용하고자 할 때
- 컨테이너 타입의 데이터를 Unpacking 할 때

```python
# *args (= Asterisk arguments)
# args 가변인자로 사용할 경우 명시된 params을 제외하고 packing되어 tuple로 반환
def aa(a, *args):
  print(a, args)
  print(type(args))

aa(1,2,3,4,5,6)

# 1 (2,3,4,5,6)
# <class 'tuple'>
```

```python
# **kargs (= Double Asterisk Keywords arguments)
# kargs 가변인자로 사용할 경우 명시된 params를 제외하고도 키워드와 함께 호출해야함.
# 키워드와 함께 호출하지 않을경우 Type Error가 발생함
# 명시된 params를 제외하고 packing되어 dict로 반
def bb(a, **kargs):
  print(a, kargs)
  print(type(kargs))

bb(1, b=2, c=3, d=4, e=5, f=6)
#1 {'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
#<class 'dict'>
```

Advanced Python

```python
numbers = [1, 2, 3, 4, 5, 6]

# unpacking의 좌변은 리스트 또는 튜플의 형태를 가져야하므로
# 단일 unpacking의 경우 *a가 아닌 *a,를 사용

*a, = numbers
# a = [1, 2, 3, 4, 5, 6]

*a, b = numbers
# a = [1, 2, 3, 4, 5]
# b = 6

a, *b, = numbers
# a = 1
# b = [2, 3, 4, 5, 6]

a, *b, c = numbers
# a = 1
# b = [2, 3, 4, 5]
# c = 6
```

```python
def ast(a,b,c,d):
    print(a,b,c,d)

data = {"b":1, "c":2, "d":3}
ast(10, **data)

# 10 1 2 3
```

```python
for data in zip(*([1,2], [3,4], [5,6])):
    print(data)

# (1, 3, 5)
# (2, 4, 6)
```

Advanced Python

Collections Library

> Collections Library

- Collections
- Data Structure
- Deque
- Counter
- orderedDict
- defaultdict
- namedtuple

```
# Module Importing
from collections import deque, Counter, OrderedDict, defaultdict, namedtuple
```

Advanced Python

👉 Deque

```
# Deque
deque_list = deque()
for i in range(5):
  deque_list.append(i)

print(deque_list)

# deque([0,1,2,3,4])
```

```
deque_list.appendleft(10)
deque_list

# deque([10,0,1,2,3,4])
```

```
deque_list.rotate(2)
deque_list

# deque([3, 4, 10, 0, 1, 2])

deque_list.rotate(2)
deque_list

# deque([1, 2, 3, 4, 10, 0])

deque_list.extend([5,6,7])
deque_list

# deque([1, 2, 3, 4, 10, 0, 5, 6, 7])

deque_list.extendleft([8,9])
deque_list

# deque([9, 8, 1, 2, 3, 4, 10, 0, 5, 6, 7])
```

Advanced Python

👉 DefaultDict

```
d = defaultdict(lambda: 0)
print(d['test'])

# 0
```

```
letters = "dingadingading"
letters_dict = defaultdict(int)
for k in letters:
  letters_dict[k] += 1

letters_dict

# defaultdict(int, {'d': 3, 'i': 3, 'n': 3, 'g': 3, 'a': 2})
```

👉 Counter

```
c = Counter()
c = Counter('gallahad')
print(c)

# Counter({'a': 3, 'l': 2, 'g': 1, 'h': 1, 'd': 1})
```

```
m = Counter({'red':4, 'blue':3})
print(m)
print(list(m.elements()))

# Counter({'red': 4, 'blue': 3})
# ['red', 'red', 'red', 'red', 'blue', 'blue', 'blue']
```

```
n = Counter(dogs=4, cats=2)
print(n)
print(list(n.elements()))

# Counter({'dogs': 4, 'cats': 2})
# ['dogs', 'dogs', 'dogs', 'dogs', 'cats', 'cats']
```

Advanced Python

▼ *Itertools Library*

Itertools Library

- count()
- cycle()
- repeat()
- compress()
- accumulate()
- filterfalse()
- product()
- permutations()
- combinations()
- combinations_with_replacement()

```
# Importing Library
import itertools as it
```

Advanced Python

👉 Compress()

```
# Compress()
# Compress(data, selectors)
# return iterator
# 특정 요소를 인덱스로 필터링
data = 'ABCDEFG'
list(it.compress(data, [1,1,1,0,0,0]))

# ['A', 'B', 'C']
```

👉 Accumulate()

```
# accumulate()
# accumulate(p, [,func])
# return iterator
# 각 항목에 대한 누적값
data = [1,2,3,4,5]
list(it.accumulate(data))

# [1, 3, 6, 10, 15]
```

👉 filterfalse()

```
# filterfalse()
# filterfalse(pred, seq)
# pred(elem)이 거짓인 seq의 요소들을 반환
list(it.filterfalse(lambda x: x%2, range(10)))

# [0, 2, 4, 6, 8]
```

Advanced Python

👉 Product()

- Cartesian Product와 같은 의미이고 '데카르트 곱' 혹은 '중복순열'이라고 한다.

$$_n\prod_r = n^r$$

```
# Product()
# Product(p,q,[repeat=1])
# 데카르트 곱(cartesian product), 중첩된 for 루프와 동일함!
target = 'ABC'
t = list(it.product(target, repeat=2))
print(t)
print(len(t))
print('='*100)
Target = ['ABC', '123']
T = list(it.product(*Target))
print(T)
print(len(T))

# [('A', 'A'), ('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'B'), ('B', 'C'), ('C', 'A'),
# 9
# ====================================================================================
# [('A', '1'), ('A', '2'), ('A', '3'), ('B', '1'), ('B', '2'), ('B', '3'), ('C', '1'),
# 9
```

Advanced Python

👉 Permutations()

- 순열은 순서를 고려 O, 중복 허락 X

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

```
# Permutations()
# permutations(p, [,r])
# 순열
data = "ABCD"
t = list(it.permutations(data, 2))
print(t)
print(len(t))

# [('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'A'), ('B', 'C'), ('B', 'D'), ('C', 'A'),
# 12
```

Advanced Python

👉 Combinations_with_replacement()

- 중복조합은 순서를 고려 X, 중복허락 O

$$_nH_r = {}_{n+r-1}C_r$$

```
# combinations_with_replacement()
# combinations_with_replacement(p, r)
# 중복조합
data = "ABCD"
t = list(it.combinations_with_replacement(data, 2))
print(t)
print(len(t))

# [('A', 'A'), ('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'B'), ('B', 'C'), ('B', 'D'),
# 10
```

Advanced Python

👉 Combinations()

- 조합은 순서를 고려 O, 중복허락 X

$$ {}_nC_r = \frac{n!}{r!(n-r)!} $$

```
# combinations()
# combinations(p, r)
# 조합
data = 'ABCD'
t = list(it.combinations(data, 2))
print(t)
print(len(t))

# [('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'C'), ('B', 'D'), ('C', 'D')]
# 6
```

Numpy

Data Science Library with ML

Numpy Tutorials

- Nupy is the core library for scientific computing in Python.
- Ref link : https://cs231n.github.io/python-numpy-tutorial/

Numpy

Arrays

Numpy array is a grid of values. all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array. The shape of an array is a tuple of integers giving the size of the array along each dimension.

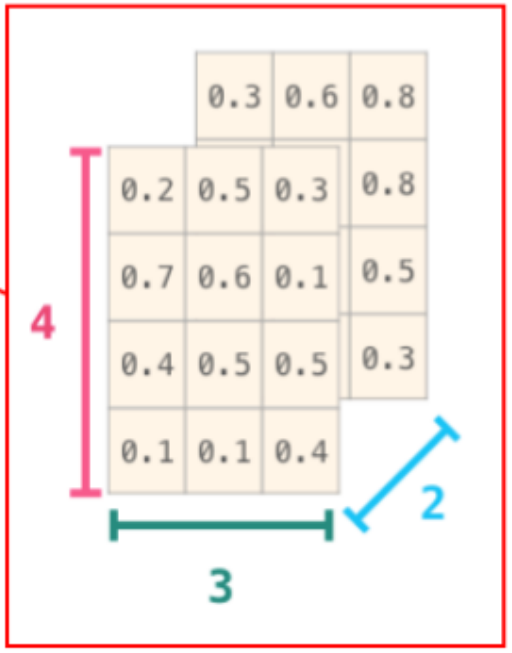
```
1  c = np.ones((2,4,3))
2  print(c, "\n")
3  print("size :", c.size)
4  print("dim :", c.ndim)
5  print("shape :", c.shape)
```

```
[[[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]]

size : 24
dim : 3
shape : (2, 4, 3)
```

Why not write two 4x3 matrices like (4, 3, 2)



Numpy

Numpy Terminology

- `Tensor` : **array**나 **matrix**와 매우 유사한 특수한 자료구조
  - 특수한 가공처리를 위해 개발된 data type이다.
  - matrix와 2dim 이상의 array와 동일시해도 무방하다.
- `axis` : 각 차원의 축
  - `axis = 0` : Matrix에서 row를 가리키는 축
  - `axis = 1` : Matrix에서 column을 가리키는 축
- `dim` : Tensor가 존재하는 축의 개수 (= 보통 col의 개수를 의미한다.)
  - 단, dim 용어의 경우 **축**과 **차원** 2개 표현을 공용하여 사용하기도 한다. 이는 vector나 matrix의 **차원**과는 다른 의미이니 헷갈리지 말자!
- `rank` : tensor가 몇 차원인지 나타내는 값
- `shape` : 각 축(차원)에 존재하는 원소의 수
  - div ≥ 3 일 경우 가장 바깥쪽 차원부터 작성한다.
  - 이를 표헌하기에 (배치사이즈 x 행의 개수 x 열의 개수)로 말한다.
  - 4x3 2 dim matrix 2개를 만든다면 shape(2,4,3)
  - 4x3x2 3 dim matrix 3개를 만든다면 shape(3, 2, 4, 3)
- `size` : 총 element의 개수

Numpy

Create Arrays

- `np.array(list type)`
  - np.array([1,2,3])
- `np.array(list type, dtype = data type)`
  - np.array([(1,2), (3,4), dtype = float]

```
a = np.array([1,2,3])       # Create a rank 1 array
print(f"type : {type(a)}")  # Prints "<class 'numpy.ndarray'>"
print(f"shape : {a.shape}") # Prints "(3, )"
print(a[0], a[1], a[2])     # Prints "1 2 3"

a[0] = 5                    # Change an element of the array
print(a)                    # Prints "[5,2,3]"
print()

b = np.array([[1,2,3], [4,5,6]])    # Create a rank 2 array
print(f"shape : {b.shape}")         # Prints "(2, 3)"
print(b[0, 0], b[0,1], b[1,0])      # Prints "1 2 4"
```

```
a = np.array([(1.5,2,3), (4,5,6)], dtype=float)
a

# array([[1.5, 2. , 3. ],
#        [4. , 5. , 6. ]])
```

Numpy

👉 Inspecting Your Array

- numpy.array.shape
- numpy.array.ndim
- numpy.array.size
- numpy.array.dtype
- numpy.array.dtype.name
- numpy.array.astype(_ another data type _)

```
t = np.array([(1,2,3), (4,5,6)])
print(f"t.shape : {t.shape}") # t.shape : (2, 3)
print(f"t.ndim : {t.ndim}") # t.ndim : 2
print(f"t.size : {t.size}") # t.size : 6
print(f"t.dtype : {t.dtype}") # t.dtype : int32
print(f"t.dtype.name : {t.dtype.name}") # t.dtype.name : int32
print(f"t.astype(str) : {t.astype(str)}")
# t.astype(str) : [['1' '2' '3']
#  ['4' '5' '6']]
```

Numpy

👉 Initial Placeholder

- `np.zeros()` : 0으로 채워진 배열을 만들기 위해 사용, 메모리를 할당받아 0으로 초기화한 뒤 반환
- `np.ones()` : 1로 채워진 배열을 만들기 위해 사용
- `np.arange()` : 반열린구간에서 step의 크기만큼 일정하게 떨어져 있는 숫자들을 array형태로 반환
  - `numpy.arange([start, ], stop, [step, ], dtype=None)`
- `np.linspace` : start부터 stop까지 num개만큼 array형태로 반환
  - `numpy.linspace(start, stop, num)`
- `np.full()` : shape에 맞게 value값을 채워 array형태로 반환
  - `numpy.full(shape, value)`
- `np.eye()` : 특정 rank의 단위 행렬을 만든다. 정사각행렬에 주대선의 원소들이 1인 행렬
- `np.random.rand()` : 주어진 형태의 난수 array를 생성
- `np.empty()` : 빈 배열을 만들기 위해 사용, 메모리를 할당만 받고 초기화 없이 반환
  - `numpy.empty(shape, dtype)`

BootCamp Register

모집 일정

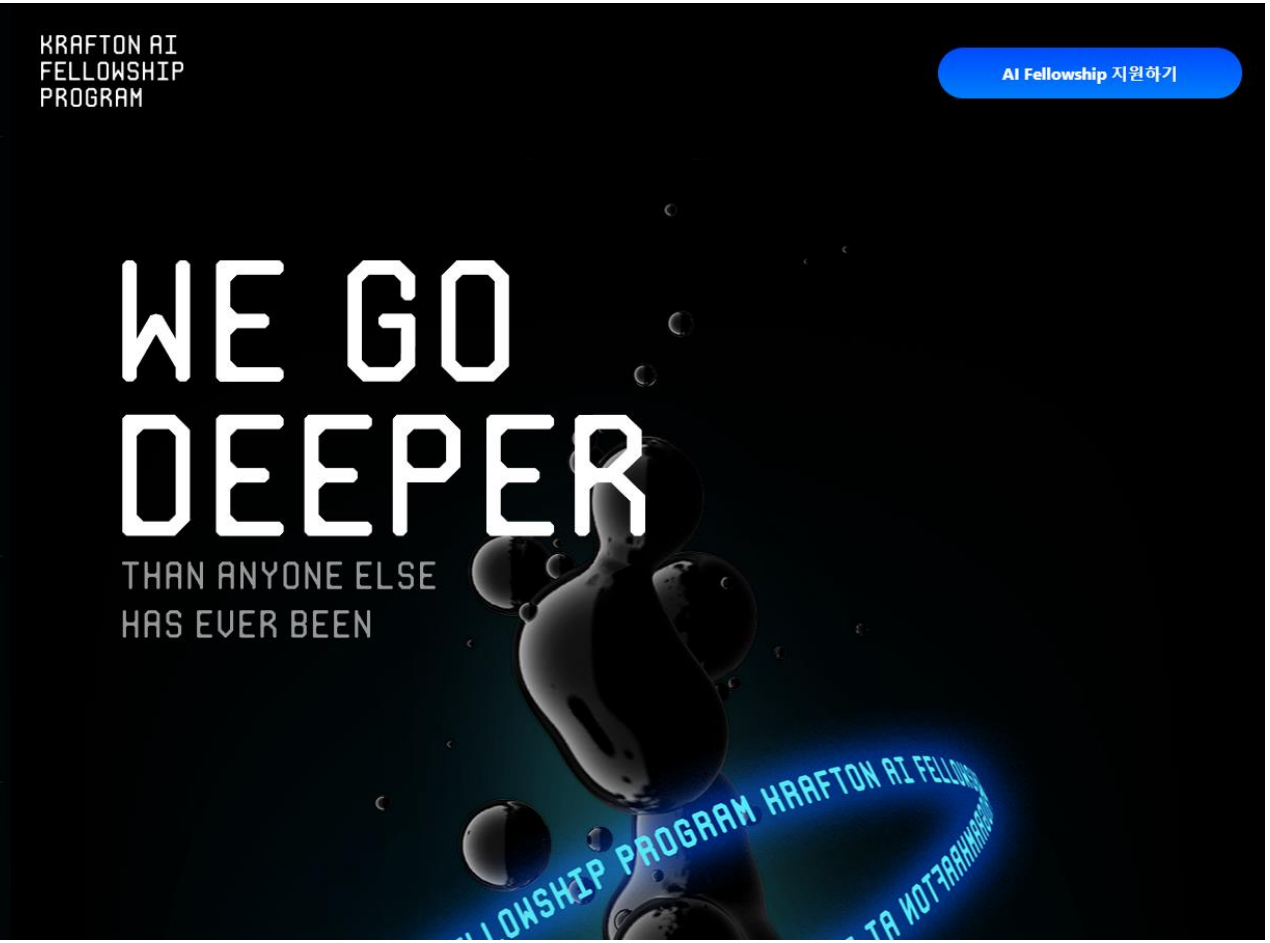
- 서류 접수: 2022년 12월 12일 (월) ~ 2022년 12월 31일 (토)
- 1차 온라인 시험: 2023년 1월 7일 (토)
- 2차 오프라인 시험: 2023년 1월 14일 (토) (장소: 크래프톤 역삼 본사)
- 3차 구술 대면 면접: 2023년 1월 16일 (월) ~ 2023년 2월 3일 (금) (장소: 크래프톤 역삼 본사)
- 최종 발표: 2023년 2월 20일 (월) ~ 2023년 2월 24일 (금)
- 각 전형별 일정은 진행 현황에 따라 유동적으로 조정될 수 있습니다.
- 일정에 대한 상세한 부분은 개별적으로 안내 드릴 예정입니다.

지원 요건

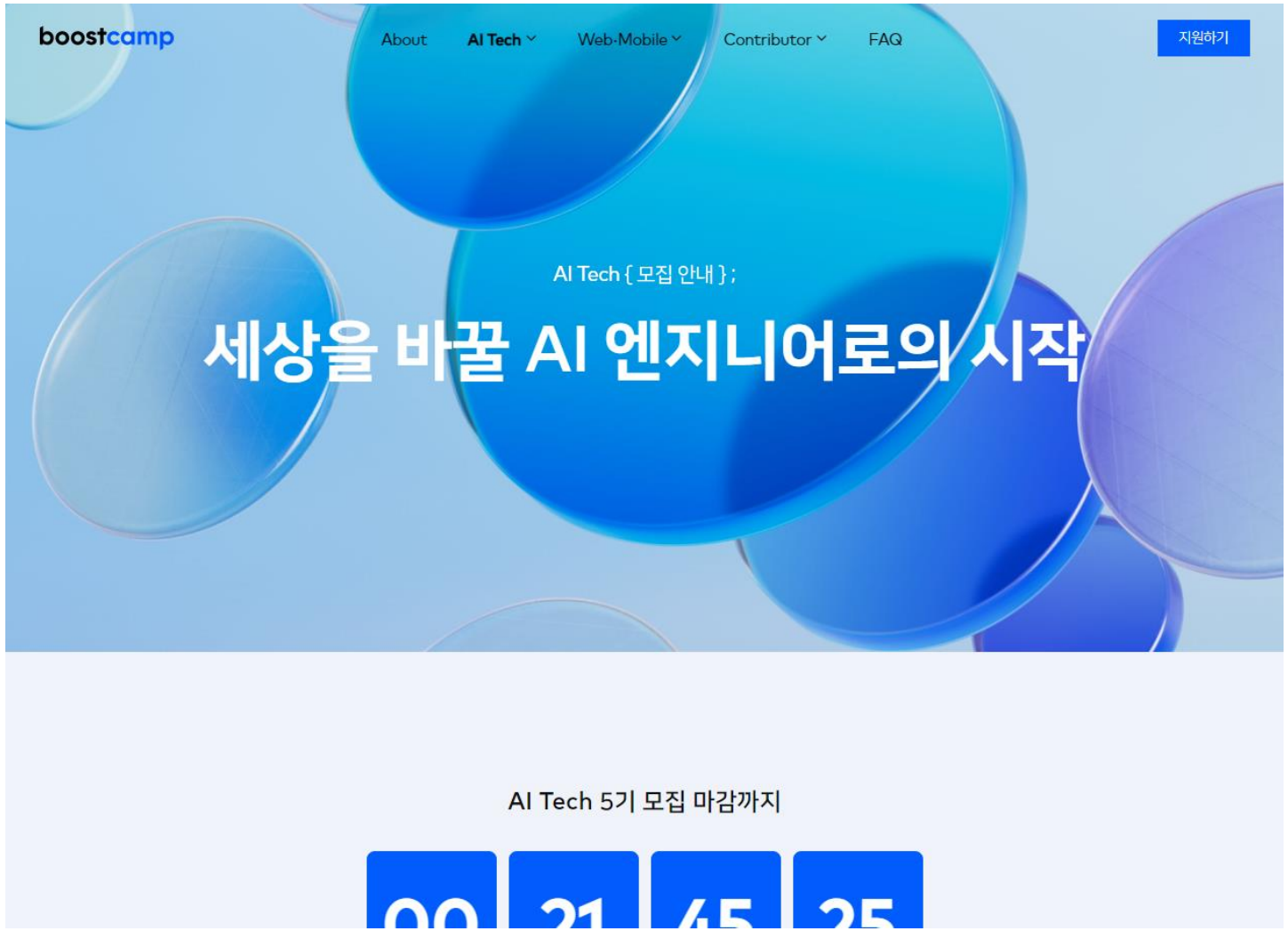
- 대학교 1학년 이상 모든 학부생 중 2023년 6월 ~ 2023년 8월 AI Research Internship 참여가 가능한 자
- 석사 재학생 이상 지원 불가

제출 서류

- 필수: 이력서
- 선택: AI 관련 프로젝트 포트폴리오 (필수 제출 서류는 아니며, 선발 점수에 전혀 영향을 주지 않는 요소로 구술 대면 면접에서 참고 자료로만 활용합니다)



BootCamp Register



Jan. 2th week Summary

- Numpy Lecture
- CS 기본 정리
- Linear Algebra
- Machine Learning Lecture
- 2 Day 1 Algorithm Problem