

노트북: 첫 번째 노트북

만든 날짜: 2020-10-28 오후 11:48

URL: <https://continuous-development.tistory.com/116?category=736681>

Python

[Python] Numpy에 대한 기초 정리와 사용법 정리

2020. 10. 9. 16:26 수정 삭제 공개

Numpy 란?

NumPy는 행렬이나 일반적으로 대규모 다차원 배열을 쉽게 처리 할 수 있도록 지원하는 파이썬의 라이브러리 입니다. 또한 NumPy는 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공한다. 그 이유는 c 언어로 구성되어 있기 때문에 연산 속도가 빠르다. (컴퓨터 언어에 가깝다.)

Numpy의 특징

넘파이의 배열은 모든 원소가 같은 자료형이어야 한다.

resizing 이 안된다.

Numpy 에서는 1차원 배열을 vector 라고 하고 2차원 부터는 행렬 이라고 한다.

-Vector(1차원 - 배열) - pandas(Series)

-Martrix(2차원 - 행렬) - pandas(DataFrame)

또한 선형대수(행렬을 이용한 연산가능)가 가능하다.

Numpy 사용법

```
In [2]: import numpy as np
```

기본적으로 numpy 를 넣어주고 as 로 약어로 사용하게끔 한다.

```
In [3]: def aryinfo(ary):  
    print('type : {}'.format(type(ary)))  
    print('shape : {}'.format(ary.shape))  
    print('dimension : {}'.format(ary.ndim))  
    print('dtype : {}'.format(ary.dtype))  
    print('Array Data : \n',ary)
```

이 함수는 해당 배열의 형태를 보기위해 임의로 만들어졌다.

type = 해당 값의 타입

shape = 값의 형태

demension = 차원

dtype = 데이터 타입

마지막은 데이터를 보기위해 ary를 그대로 넣었다.

```
In [11]: oneAry = np.array([0,1,2,3,4,5,6,7,8,9])  
aryinfo(oneAry)
```

```
type : <class 'numpy.ndarray'>  
shape : (10,)  
dimension : 1  
dtype : int32  
Array Data :  
[0 1 2 3 4 5 6 7 8 9]
```

np.array 로 numpy 타입의 array를 만들어 놓고 해당 상태를 확인해 봤다.

- List vs Array 차이점

```
In [12]: data = [0,1,2,3,4,5,6,7,8,9]
         data * 2
Out[12]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

리스트에서는 해당 리스트를 곱하게 되면 위와 같이 리스트 자체가 2개로 된다.

```
In [13]: result = []
         for d in data:
             result.append(d*2)
         result
Out[13]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

안에 있는 값에 * 2를 하기 위해서는 위와 같은 방식으로 값을 하나씩 빼와서 넣는 형태나

```
In [14]: result2 = [d*2 for d in data]
         result2
Out[14]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

파이썬 만의 문법은 list comprehension 방식으로 이렇게 값을 넣어줘야 한다.

```
In [15]: result3 = oneAry * 2
         result3
Out[15]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

하지만 numpy를 이용한다면 위와 같이 간단하게 vector 연산을 진행 할 수 있다.

- 벡터 연산은 비교, 산술, 논리 연산을 포함하는 모든 수학 연산에 적용됨

```
In [16]: xAry = np.array([1,2,3])  
        yAry = np.array([10,20,30])
```

```
In [18]: xAry + yAry
```

```
Out [18]: array([11, 22, 33])
```

또한 numpy는 벡터 연산으로 비교, 산술, 논리 연산을 포함하는 모든 수학 연산에 적용된다.

```
In [19]: 2 * xAry + yAry
```

```
Out [19]: array([12, 24, 36])
```

곱셈 연산도 가능하다.

```
In [20]: xAry == 2
```

```
Out [20]: array([False,  True, False])
```

해당 값인 걸 찾는것도 가능하다.

```
In [21]: yAry > 20
```

```
Out [21]: array([False, False,  True])
```

20 이상인 값을 찾는다.

```
In [22]: (xAry == 2) & (yAry > 20)
```

```
Out [22]: array([False, False, False])
```

이런식의 연산도 가능하다.

2차원 배열 생성

- ndarray(N-dimensional Array)
- 2차원, 3차원(다차원 배열 자료구조)

- list of list(2차원)
- list of list of list(3차원)

```
In [23]: # 2개의 행과 3개의 열을 가지는 배열을 만든다면
twoAry = np.array([[1,2,3],[4,5,6]])
```

2차원 행렬은 위와 같은 방식으로 만든다. `[[] , []` 리스트 안에 리스트를 넣어주면 된다.

```
In [24]: aryinfo(twoAry)

type : <class 'numpy.ndarray'>
shape : (2, 3)
dimension : 2
dtype : int32
Array Data :
[[1 2 3]
 [4 5 6]]
```

형태를 봐보자.

```
In [28]: # 행의 개수, 열의 개수
print(len(twoAry)) # 행의 개수
print(len(twoAry[0])) # 열의 개수
print(len(twoAry[1])) # 열의 개수

2
3
3
```

행의 개수와 열의 개수는 위와 같은 방법으로 확인하면 된다. 길이를 보고 행을 보고 해당 리스트의 인덱스로 한 행에 들어있는 값의 개수를 보아 열의 개수를 센다.

```
In [29]: # 2개의 행과 3개의 열을 가지는 배열을 만든다면
twoAry = np.array([[1,2,3],[4,5,6]], dtype=np.float64)
```

```
In [30]: aryinfo(twoAry)

type : <class 'numpy.ndarray'>
shape : (2, 3)
dimension : 2
dtype : float64
Array Data :
[[1. 2. 3.]
 [4. 5. 6.]]
```

`dtype`을 이용해서 float 형태로 np 를 만들 수도 있다.

3차원 배열생성

```
In [42]: # 2 3 4
threeAry = np.array([[[1,2,3,4],[5,6,7,8],[9,10,11,12]],
                    [[13,14,15,16],[17,18,19,20],[21,22,23,24]]])
aryinfo(threeAry)

type : <class 'numpy.ndarray'>
shape : (2, 3, 4)
dimension : 3
dtype : int32
Array Data :
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
```

리스트의 리스트의 리스트로 3차원 배열을 만들 수 있다.

```
In [41]: print('dept',len(threeAry)) # dept
print('row',len(threeAry[0])) # row
print('row[0]_col',len(threeAry[0][0])) # col

dept 2
row 3
row[0]_col 4
```

이 3차원의 dept 열 길이를 보기 위해서는 위와같은 방식으로 볼수 있다.

- 요소의 타입을 변경할 때는 **astype()** 을 사용한다.

```
In [44]: typeChange = threeAry.astype(np.float64)
aryinfo(typeChange)

type : <class 'numpy.ndarray'>
shape : (2, 3, 4)
dimension : 3
dtype : float64
Array Data :
[[[ 1.  2.  3.  4.]
  [ 5.  6.  7.  8.]
  [ 9. 10. 11. 12.]]

 [[13. 14. 15. 16.]
  [17. 18. 19. 20.]
  [21. 22. 23. 24.]]]
```

해당 np에 astype으로 타입을 바꿔 줄 수 있다.

```
In [45]: indexArray = np.array([1,2,3,4,5,6,7])
         aryinfo(indexArray)

         type : <class 'numpy.ndarray'>
         shape : (7,)
         dimension : 1
         dtype : int32
         Array Data :
         [1 2 3 4 5 6 7]
```

만약 여기서 3번째 값을 꺼내온다고 했을 때

```
In [46]: indexArray[2]
Out [46]: 3
```

이런식으로 파이썬에서 인덱스를 쓰던 방식과 같게 하면 된다.

```
In [48]: indexArray[-1]
Out [48]: 7
```

-1 번째 값은 마지막 값을 뜻한다.

'Python' 카테고리의 다른 글

[Python] Numpy에 있는 다양한 함수 사용법 - 2(전치행렬,zeors,ones, iterator,...

[Python] Numpy를 통한 배열 indexing(Boolen indexing, fancy indexing)

[Python] Numpy에 대한 기초 정리와 사용법 정리

[Python] python 에서 Seleium을 통한 동적 크롤링 - 4

[Python] python 에서 Seleium을 통한 동적 크롤링 - 3

[Python] python 에서 Seleium을 통한 동적 크롤링 - 2

numpy 사용법

numpy란

python numpy

python 넘파이

파이썬 numpy



나무늘보스

혼자 끄적끄적하는 블로그 입니다.