[Python] Numpy를 통한 배열 indexing(Boolen indexing, fancy indexing) — 나무늘보의 개발 블로그

노트북: 첫 번째 노트북
만든 날짜: 2020-10-28 오후 11:48
URL: https://continuous-development.tistory.com/117?category=736681

Python

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

2020. 10. 9. 17:32 수정 삭제 공개

Numpy를 통한 배열 indexing

2차원 배열 indexing

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

np를 통해 2차원 배열을 만들었다.

```
In [55]: # twoAry indexing
         # 첫번째 행의 첫번째 열
         print(twoAry[0][0])
         print(twoAry[0,0])
         # 첫번째 행의 두번째 열
         print(twoAry[0][1])
         print(twoAry[0,1])
         # 마지막 행의 마지막 열
         print(twoAry[-1][-1])
         print(twoAry[-1,-1])

1.0
1.0
2.0
2.0
6.0
6.0
```

그 배열에서 인덱싱을 통해 값을 뽑아온다.
twoAry[0] 은 첫번째 행을 뜻하고
twoAry[0][0] 은 첫번째 행의 첫번째 열을 뜻한다.
twoAry[0,0] 또한 마찬가지의 의미이다.

```
In [60]: slicingAry = np.array([[1,2,3,4],
                                [5,6,7,8]])

         # 첫번째 행의 전체
         print(slicingAry[0])
         # 두번째 열의 전체
         print(slicingAry[:,1])
         # 두번째 행의 두번째 열부터 끝까지
         print(slicingAry[1,1:])

[1 2 3 4]
[2 6]
[6 7 8]
```

```
In [62]: m = np.array([[ 0,  1,  2,  3,  4],
                       [ 5,  6,  7,  8,  9],
                       [10, 11, 12, 13, 14]])
         # 이 행렬에서 값 7을 인덱싱한다.
         # 이 행렬에서 값 14를 인덱싱한다.
         # 이 행렬에서 배열 [6,7]을 슬라이싱 한다.
         # 이 행렬에서 배열 [7,12] 을 슬라이싱 한다.
         # 이 행렬에서 배열 [[3,4],[8,9]]을 슬라이싱한다.

In [70]: print(m[1,2])
         print(m[2,-1])
         print(m[1,1:3])
         print(m[1:,2])
         print(m[0:2,3:5])

7
14
[6 7]
[ 7 12]
[[3 4]
 [8 9]]
```

- boolean Indexing

값에 boolean(True,False)가 들어가면 boolean indexing 이다.

```
In [73]: arr = np.array([0,1,2,3,4,5,6,7,8,9])
         aryinfo(arr)

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

해당 2차원 배열을 만들었을 때 만약 짝수 값만 가져오는 것을 위해서

```
In [81]: arr % 2 ==0 # boolean Indexing

Out[81]: array([ True, False,  True, False,  True, False,  True, False,  True,
                False])
```

arr 를 연산으로 보고 이때 나온 배열이 boolean Indexing 구한다.

```
In [82]: arr[arr % 2 ==0]

Out[82]: array([0, 2, 4, 6, 8])
```

이걸 arr[] 안에 행으로 넣어서 사용하면 해당하는 값만 추출된다.

```
In [84]: cntIdx = np.array([0,2,4,6,8])
         print(arr[cntIdx])

         [0 2 4 6 8]
```

```
In [86]: x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                       11, 12, 13, 14, 15, 16, 17, 18, 19, 20])

         # 이 배열에서 3의 배수를 찾아라
         # 이 배열에서 4로 나누면 1이 남는 수를 찾아라
         # 이 배열에서 3으로 나누면 나누어지고 4로 나누면 1이 남는 수를 찾아라
```

이렇게 하나의 배열이 있었을 때

```
In [102]: print(x[x%3==0])
          print(x[x%4==1])
          print(x[(x%3==0)&(x%4==1)])

          [ 3  6  9 12 15 18]
          [ 1  5  9 13 17]
          [9]
```

이런식으로 연산을 안에 넣어서 뽑아 올수도 있다.

-fancy Indexing

정수 배열을 indexer로 사용해서 다차원 배열로 부터 Indexing하는 방법

```
In [104]: # 배열에 index 배열을 전달하여 배열 요소를 참조해 보자
          fancyAry = np.arange(0,12,1).reshape(3,4)
          aryinfo(fancyAry)

          type : <class 'numpy.ndarray'>
          shape : (3, 4)
          dimension : 2
          dtype : int32
          Array Data :
           [[ 0  1  2  3]
           [ 4  5  6  7]
           [ 8  9 10 11]]
```

이걸 해석해서 보면 np.arrange () 를 사용해서 0부터 12까지의 숫자를 1 간격으로 값을 가져온다.
그 다음 reshape를 통해 3행 4열로 만들어준다.

```
In [110]: # 10 값을 가져온다면?
          fancyAry[2,2]
Out[110]: 10
```

이 형태로 값을 가져오면 스칼라 타입(= 10)으로 가져와지고

```
In [112]: # 6의 값을 가져온다면?
          fancyAry[1:2,2]
Out[112]: array([6])
```

이 형태로 값을 가져오면 ([6]) 배열 타입으로 가져오게 된다. 이런 타입이 fancy indexing 이다.

```
In [113]: fancyAry[1:2,:]
Out[113]: array([[4, 5, 6, 7]])
```

이런 형태면 2차원의 배열로 가져오 게된다. 이런식으로 어떤 식의 코드로 값을 가져오냐 따라 리턴이 달라질 수 있다.

```
In [116]: aryinfo(fancyAry[1:2,1:2])

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

이렇게 가져온 값은 aryinfo로 형태를 봤을때 numpy의 형태로 가져오게 된다. 이런식으로 리턴값이 어떻게 들어오는지를 잘 확인해야한다.

```
In [130]: # 2와 10 값을 가져온다면?
          fancyAry[[0,2],2] # 사용하는 행은 0 행과 2번째 행이고 가지고 오는 값은 2번째 값이란 뜻이다.
Out[130]: array([ 2, 10])
```

이 같은 경우에는 index를 통해 값을 받고

```
In [134]: # 2와 10 값을 가져온다면?
          fancyAry[[0,2],2:3] # 사용하는 행은 0 행과 2번째 행이고 가지고 오는 값은 2번째 값이란 뜻이다.
Out[134]: array([[ 2],
                 [10]])
```

위와 같은 경우에는 sliceing 해서 값을 리턴 받는다. 이때는 차원이 하나 더 붙는다.

이렇게 가져오는 형태에 따라서 다른 리턴값을 가져오는 것을 볼 수 있다.

```
In [157]: rowidx = np.array([0,2])
          colidx = np.array([0,2])

          print(fancyAry[ [rowidx]] [:,colidx])
          [[ 0  2]
           [ 8 10]]
```


'Python' 카테고리의 다른 글

[Python] Numpy의 reshape 통한 차원 변경(재배열)

[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

boolean indexing fancy indexing numpy indexing

나무늘보스

혼자 끄적끄적하는 블로그 입니다.