**1. numpy 패키지**

numpy는 과학 계산을 위한 라이브러리로서 다차원 배열을 처리하는데 필요한 여러 유용한 기능을 제공하고 있다.

numpy는 pip을 사용하여 아래와 같이 간단히 설치할 수 있다.

$ pip install numpy

**2. numpy 배열**

numpy에서 배열은 동일한 타입의 값들을 가지며, 배열의 차원을 rank 라 하고, 각 차원의 크기를 튜플로 표시하는 것을 shape 라 한다. 예를 들어, 행이 2이고 열이 3인 2차원 배열에서 rank는 2 이고, shape는 (2, 3) 이 된다.

numpy 배열을 생성하는 방법은 파이썬 리스트를 사용하는 방법과 numpy에서 제공하는 함수를 사용하는 방법이 있다. 아래 예제에서 list1은 4개의 요소를 갖는 리스트인데, 이를 array() 함수에 넣어 numpy 배열을 생성하는데, 이 배열의 rank는 1이 되고, shape는 (4, ) 가 된다. 튜플에 하나의 요소만 있으면 문법상 콤마를 뒤에 붙인다. 두번째 배열 b는 2x3 배열로서 shape는 (2, 3)이 되는데, 한가지 주의할 점은 array() 안에 하나의 리스트만 들어가므로 리스트의 리스트를 넣어야 한다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import numpy as np    list1 = [1, 2, 3, 4]  a = np.array(list1)  print(a.shape) # (4, )    b = np.array([[1,2,3],[4,5,6]])  print(b.shape) # (2, 3)  print(b[0,0])  # 1 |

numpy에서 제공하는 함수를 사용하여 numpy 배열을 만드는 방법을 살펴보자. 이러한 기능을 제공하는 함수로는 zeros(), ones(), full(), eye() 등이 있는데, zeros()는 해당 배열에 모두 0을 집어 넣고, ones()는 모두 1을 집어 넣는다. full()은 배열에 사용자가 지정한 값을 넣는데 사용하고, eye()는 대각선으로는 1이고 나머지는 0인 2차원 배열을 생성한다.  
아래 예제는 이들 함수들을 사용하여 numpy 배열을 생성한 예이다. 그리고 마지막 예는 0부터 n-1 까지의 숫자를 생성하는 range(n) 함수와 배열을 다차원으로 변형하는 reshape()를 통해 간단하게 샘플 배열을 생성해 본 것이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import numpy as np    a = np.zeros((2,2))  print(a)  # 출력:  # [[ 0.  0.]  #  [ 0.  0.]]    a = np.ones((2,3))  print(a)  # 출력:  # [[ 1.  1.  1.]  #  [ 1.  1.  1.]]    a = np.full((2,3), 5)  print(a)  # 출력:  # [[5 5 5]  #  [5 5 5]]    a = np.eye(3)  print(a)  # 출력:  # [[ 1.  0.  0.]  #  [ 0.  1.  0.]  #  [ 0.  0.  1.]]    a = np.array(range(20)).reshape((4,5))  print(a)  # 출력:  # [[ 0  1  2  3  4]  #  [ 5  6  7  8  9]  #  [10 11 12 13 14]  #  [15 16 17 18 19]] |

**3. numpy 슬라이싱**

numpy 배열은 파이썬 리스트와 마찬가지로 슬라이스(Slice)를 지원한다. numpy 배열을 슬라이싱하기 위해서는 각 차원별로 슬라이스 범위를 지정한다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import numpy as np    lst = [      [1, 2, 3],      [4, 5, 6],      [7, 8, 9]  ]  arr = np.array(lst)    # 슬라이스  a = arr[0:2, 0:2]  print(a)  # 출력:  # [[1 2]  #  [4 5]]    a = arr[1:, 1:]  print(a)  # 출력:  # [[5 6]  #  [8 9]] |

**4. numpy 정수 인덱싱 (integer indexing)**

numpy 슬라이싱이 각 배열 차원별 최소-최대의 범위를 정하여 부분 집합을 구하는 것이라면, 정수 인덱싱은 각 차원별로 선택되어지는 배열요소의 인덱스들을 일렬로 나열하여 부분집합을 구하는 방식이다. 즉, 임의의 numpy 배열 a 에 대해 a[[row1, row2], [col1, col2]] 와 같이 표현하는 것인데, 이는 a[row1, col1] 과 a[row2, col2] 라는 두 개의 배열요소의 집합을 의미한다.   
예를 들어, 아래 예제에서 a[[0, 2], [1, 3]] 은 정수 인덱싱으로서 이는 a[0, 1] 과 a[2, 3] 등 2개의 배열요소를 가리킨다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import numpy as np    lst = [      [1, 2, 3, 4],      [5, 6, 7, 8],      [9, 10, 11, 12]  ]  a = np.array(lst)    # 정수 인덱싱  s = a[[0, 2], [1, 3]]    print(s)  # 출력  # [2 12] |

**5. numpy 부울린 인덱싱 (boolean indexing)**

numpy 부울린 인덱싱은 배열 각 요소의 선택여부를 True, False로 표현하는 방식이다. 만약 배열 a 가 2 x 3 의 배열이이라면, 부울린 인덱싱을 정의하는 numpy 배열도 2 x 3 으로 만들고 선택할 배열요소에 True를 넣고 그렇지 않으면 False를 넣으면 된다.   
예를 들어, 아래 예제에서 3 x 3 배열 a 중 짝수만 뽑아내는 부울린 인덱싱 배열(numpy 배열)을 사용하여 짝수 배열 n 을 만드는 예이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import numpy as np    lst = [      [1, 2, 3],      [4, 5, 6],      [7, 8, 9]  ]  a = np.array(lst)    bool\_indexing\_array = np.array([      [False,  True, False],      [True, False,  True],      [False,  True, False]  ])    n = a[bool\_indexing\_array];  print(n) |

부울린 인덱싱 배열에 True/False 값을 일일이 지정하는 방법 이외에 표현식을 사용하여 부울린 인덱싱 배열을 생성하는 방법이 있다. 예를 들어, 배열 a 에 대해 짝수인 배열요소만 True로 만들고 싶다면, bool\_indexing = (a % 2 == 0) 와 같이 표현할 수 있다. 아래 예제는 이러한 표현을 사용하는 것을 예시한 것이고, 특히 마지막에 a[ a % 2 == 0 ] 와 같이 부울린 인덱싱 표현식을 배열 인덱스안에 넣어 간단하게 표현할 수도 있다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import numpy as np    lst = [      [1, 2, 3],      [4, 5, 6],      [7, 8, 9]  ]  a = np.array(lst)    # 배열 a 에 대해 짝수면 True, 홀수면 False  bool\_indexing = (a % 2 == 0)    print(bool\_indexing)  # 출력: 부울린 인덱싱 배열  # [[False  True False]  #  [ True False  True]  #  [False  True False]]    # 부울린 인덱스를 사용하여 True인 요소만 뽑아냄  print(a[bool\_indexing])  # 출력:  # [2 4 6 8]    # 더 간단한 표현  n = a[ a % 2 == 0 ]  print(n) |

**6. numpy 연산**

numpy를 사용하면 배열간 연산을 쉽게 실행할 수 있다. 연산은 +, -, \*, / 등의 연산자를 사용할 수도 있고, add(), substract(), multiply(), divide() 등의 함수를 사용할 수도 있다. 예를 들어, 아래 예제와 같이 배열 a 와 b 가 있을때, a + b를 하면 각 배열요소의 합을 구하는데 즉 a[0]+b[0], a[1]+b[1], ... 과 같은 방식으로 결과를 리턴하게 된다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import numpy as np    a = np.array([1,2,3])  b = np.array([4,5,6])    # 각 요소 더하기  c = a + b  # c = np.add(a, b)  print(c)  # [5 7 9]    # 각 요소 빼기  c = a - b  # c = np.subtract(a, b)  print(c)  # [-3 -3 -3]    # 각 요소 곱하기  # c = a \* b  c = np.multiply(a, b)  print(c)  # [4 10 18]    # 각 요소 나누기  # c = a / b  c = np.divide(a, b)  print(c)  # [0.25 0.4 0.5] |

numpy에서 vector와 matrix의 product를 구하기 위해서 dot() 함수를 사용한다. 아래 예제는 두 matrix의 product를 구한 예이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import numpy as np    lst1 = [      [1,2],      [3,4]  ]    lst2 = [      [5,6],      [7,8]  ]  a = np.array(lst1)  b = np.array(lst2)    c = np.dot(a, b)  print(c)  # 출력:  # [[19 22]  #  [43 50]] |

numpy은 배열간 연산을 위한 위해 많은 함수들을 제공하는데, 예를 들어, 각 배열 요소들을 더하는 sum() 함수, 각 배열 요소들을 곱하는 prod() 함수 등을 사용할 수 있다. 이들 함수에 선택옵션으로 axis 을 지정할 수 있는데, 예를 들어 sum()에서 axis가 1 이면 행끼리 더하는 것이고, axis가 0 이면 열끼리 더하는 것이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import numpy as np    a = np.array([[1,2],[3,4]])    s = np.sum(a)  print(s)   # 10    # axis=0 이면, 컬럼끼리 더함  # axis=1 이면, 행끼리 더함  s = np.sum(a, axis=0)  print(s)   # [4 6]    s = np.sum(a, axis=1)  print(s)   # [3 7]    s = np.prod(a)  print(s)   # 24 |