20171664 소프트웨어학부 이범석

HW3 p57 ~ p90 In and Out Practice

실습 결과

1. enumerate를 이용하여 list num의 값들에 2를 곱한 리스트를 출력했습니다.

```
In [1]: num = [2,4,6,8,10]
        for i, n in enumerate(num):
            num[i]= n*2
        print(num)
```

```
[4, 8, 12, 16, 20]
```

2. list 형이 벡터로 처리되는지 확인했습니다.

   벡터로 처리되는 것이 아닌 str형 + 연산자로 처리되어 연결됩니다.

```
In [2]: [1,2] + [3,4]
```

```
Out[2]: [1, 2, 3, 4]
```

3. 넘파이 라이브러리를 import 했습니다.

```
In [3]: import numpy as np
```

4. list 형으로 정의된 np.array를 x에 저장하고 x를 출력했습니다.

```
In [4]: x=np.array([1,2,3])
        x
```

```
Out[4]: array([1, 2, 3])
```

5. print(x) 를 통해 깔끔하게 x를 출력했습니다.

```
In [5]: print(x)
```

```
[1 2 3]
```

6. np.array로 처리한 변수들은 벡터로 처리되어 요소들이 더해집니다.

```
In [6]: y = np.array([4,5,6])
        print(x+y)
```

```
[5 7 9]
```

7. type(x)를 통해 x의 type을 확인했습니다.

```
In [7]: type(x)
```

```
Out[7]: numpy.ndarray
```

8. 요소 참조는 파이썬의 list와 동일했습니다.

```
In [8]: x[0]
```

```
Out[8]: 1
```

9. 요소 수정을 해 주었습니다.

In [9]:
```
x[0] =100
print(x)
```

```
[100   2   3]
```

10. 연속된 정수 벡터를 생성해 주었습니다.

In [10]:
```
print(np.arange(10))
```

```
[0 1 2 3 4 5 6 7 8 9]
```

11. arange 안에 범위를 지정하고 지정한 범위의 배열을 만들었습니다.

In [11]:
```
print(np.arange(5,10))
```

```
[5 6 7 8 9]
```

12. ndarray 함수에서 요소를 수정했습니다.

    copy를 하지 않고 b의 요소를 수정한 결과 a[0]과 b[0] 이 같은 값으로 변했습니다.

In [12]:
```
a = np.array([1,1])
b =a
print ('a ='+str(a))
print ('b ='+str(b))
b[0] = 100
print ('b ='+str(b))
print ('a ='+str(a))
```

```
a =[1 1]
b =[1 1]
b =[100   1]
a =[100   1]
```

13. copy를 사용하여 b를 a와 다른 독립된 변수로 취급한 후 b[0]의 값만 바꿔 주었습니다.

In [13]:
```
a = np.array([1,1])
b = a.copy()
print('a =' + str(a))
print('b =' + str(b))
b[0] = 100
print ('b ='+str(b))
print ('a ='+str(a))
```

```
a =[1 1]
b =[1 1]
b =[100   1]
a =[1 1]
```

14. ndarray의 2차원 배열을 정의했습니다.

In [14]:
```
x = np.array([[1, 2, 3], [4, 5, 6]])
print(x)
```

```
[[1 2 3]
 [4 5 6]]
```

15. shape 명령으로 행렬의 크기를 출력했습니다.

In [15]:
```
x = np.array([[1,2,3],[4,5,6]])
x.shape
```

Out[15]:
```
(2, 3)
```

16. w, h 변수에 행렬의 각 크기를 저장해 주었습니다.

In [16]:
```
w,h = x.shape
print(w)
print(h)
```

```
2
3
```

17. 각 차원을 ',' 로 구분하여 요소를 참조했습니다.

In [17]:
```
x = np.array([[1,2,3],[4,5,6]])
x[1,2]
```

Out[17]: 6

18. 요소를 참조하고 해당 요소를 수정했습니다.

In [18]:
```
x = np.array([[1,2,3],[4,5,6]])
x[1,2] = 100
print(x)
```

```
[[  1   2   3]
 [  4   5 100]]
```

19. zeros 함수를 이용하여 길이가 10고 모든 요소가 0인 벡터를 생성했습니다.

In [19]:
```
print(np.zeros(10))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

20. 2 x 10 크기의 모든 요소가 0인 행렬을 생성했습니다.

In [20]:
```
print(np.zeros((2,10)))
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

21. 2 x 10 크기의 모든 요소가 1인 행렬을 생성했습니다.

In [21]:
```
print(np.ones((2,10)))
```

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

22. 임의의 요소로 이루어진 행렬을 생성했습니다.

In [22]:
```
np.random.rand(2,3)
```

Out[22]:
```
array([[0.5393216 , 0.33536616, 0.33127026],
       [0.48411791, 0.86653324, 0.3018064 ]])
```

23. 연속된 정수 벡터를 생성해 주었습니다.

In [23]:
```
a = np.arange(10)
print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

24. reshape를 사용하여 벡터를 행렬로 바꾸었습니다.

In [24]:
```
a.reshape(2,5)
```

Out[24]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

25. 요소끼리 사칙연산이 되는지 확인했습니다.

In [25]:
```
x = np.array([[4,4,4],[8,8,8]])
y = np.array([[1,1,1],[2,2,2]])
print(x + y)
```

```
[[ 5  5  5]
 [10 10 10]]
```

26. 스칼라를 행렬에 곱했습니다.

In [26]:
```
x = np.array([[4,4,4],[8,8,9]])
print(10*x)
```

```
[[40 40 40]
 [80 80 90]]
```

27. exp 함수를 모든 요소에 적용시켰습니다.

```
In [27]: x = np.array([[4,4,4],[8,8,8]])
         print(np.exp(x))
```

```
[[  54.59815003   54.59815003   54.59815003]
 [2980.95798704 2980.95798704 2980.95798704]]
```

28. dot을 이용하여 요소별로 계산하지 않는 행렬을 곱해줬습니다.

```
In [28]: v = np.array([[1,2,3],[4,5,6]])
         w = np.array([[1,1],[2,2],[3,3]])
         print(v.dot(w))
```

```
[[14 14]
 [32 32]]
```

29. 해당 숫자까지 벡터를 슬라이싱했습니다.

```
In [29]: x = np.arange(10)
         print(x)
         print(x[:5])
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4]
```

30. 해당 숫자에서 마지막 요소까지 슬라이싱해서 출력했습니다.

```
In [30]: print(x[5:])
```

```
[5 6 7 8 9]
```

31. 3번 인덱스에서 7번 인덱스까지 출력했습니다.

```
In [31]: print(x[3:8])
```

```
[3 4 5 6 7]
```

32. 3번 인덱스에서 7번 인덱스까지 2의 간격으로 출력했습니다.

```
In [31]: print(x[3:8])
```

```
[3 4 5 6 7]
```

33. 배열의 순서를 반대로 바꿨습니다.

```
In [33]: print(x[::-1])
```

```
[9 8 7 6 5 4 3 2 1 0]
```

34. 1차원 이상의 배열도 슬라이싱해 보았습니다.

```
In [34]: y = np.array([[1,2,3],[4,5,6],[7,8,9]])
         print(y)
         print(y[:2, 1:2])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[2]
 [5]]
```

35. bool 형식의 배열을 반환했습니다.

```
In [35]: x = np.array([1,1,2,3,5,8,13])
         x >3
Out[35]: array([False, False, False, False,  True,  True,  True])
```

36. bool 배열의 요소를 참조하여 true인 요소만 출력했습니다.

```
In [36]: x[x >3]
Out[36]: array([ 5,  8, 13])
```

37. 조건을 충족하는 요소만 바꾸었습니다.

```
In [37]: x[x > 3] = 999
         print(x)
[ 1  1  2  3 999 999 999]
```

38. help를 사용하여 함수의 사용법을 출력해 보았습니다.

In [38]:
```
help(np.random.randint)
```

```
Help on built-in function randint:

randint(...) method of numpy.random.mtrand.RandomState instance
    randint(low, high=None, size=None, dtype=int)

    Return random integers from `low` (inclusive) to `high` (exclusive).

    Return random integers from the "discrete uniform" distribution of
    the specified dtype in the "half-open" interval [`low`, `high`). If
    `high` is None (the default), then results are from [0, `low`).

    .. note::
        New code should use the ``integers`` method of a ``default_rng()``
        instance instead; see `random-quick-start`.

    Parameters
    ----------
    low : int or array-like of ints
        Lowest (signed) integers to be drawn from the distribution (unless
        ``high=None``, in which case this parameter is one above the
        *highest* such integer).
    high : int or array-like of ints, optional
        If provided, one above the largest (signed) integer to be drawn
        from the distribution (see above for behavior if ``high=None``).
        If array-like, must contain integer values
    size : int or tuple of ints, optional
        Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
        ``m * n * k`` samples are drawn.  Default is None, in which case a
        single value is returned.
    dtype : dtype, optional
        Desired dtype of the result. Byteorder must be native.
        The default value is int.

        .. versionadded:: 1.11.0

    Returns
    -------
    out : int or ndarray of ints
        `size`-shaped array of random integers from the appropriate
        distribution, or a single such random int if `size` not provided.

    See Also
    --------
    random_integers : similar to `randint`, only for the closed
        interval [`low`, `high`], and 1 is the lowest value if `high` is
        omitted.
    Generator.integers: which should be used for new code.

    Examples
    --------
    >>> np.random.randint(2, size=10)
    array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0]) # random
    >>> np.random.randint(1, size=10)
    array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

    Generate a 2 x 4 array of ints between 0 and 4, inclusive:

    >>> np.random.randint(5, size=(2, 4))
    array([[4, 0, 2, 1], # random
           [3, 2, 2, 0]])

    Generate a 1 x 3 array with 3 different upper bounds

    >>> np.random.randint(1, [3, 5, 10])
    array([2, 2, 9]) # random

    Generate a 1 by 3 array with 3 different lower bounds

    >>> np.random.randint([1, 5, 7], 10)
    array([9, 8, 7]) # random

    Generate a 2 by 4 array using broadcasting with dtype of uint8

    >>> np.random.randint([1, 3, 5, 7], [[10], [20]], dtype=np.uint8)
    array([[ 8,  6,  9,  7], # random
           [ 1, 16,  9, 12]], dtype=uint8)
```

39. Hi를 출력하는 함수를 만들고 실행했습니다.

In [39]:
```
def my_func1():
    print('Hi!')
my_func1()
```

```
Hi!
```

40. a, b를 받아 더한 값을 c에 저장하고 return해주는 함수를 만들고, 실행했습니다.

In [40]:
```
def my_func2(a,b):
    c = a+b
    return c

my_func2(1,2)
```

Out[40]: 3

41. D를 입력받아 1차원ndarray 형으로 함수에 전달하고, 평균값과 표준편차를 출력하는 함수를 만들었습니다.

In [41]:
```
def my_func3(D):
    m = np.mean(D)
    s = np.std(D)
    return m, s
```

42. 변수에 각각 평균값과 표준편차를 넣어 주고 형식에 맞게 출력해 주었습니다.

In [42]:
```
data = np.random.randn(100)
data_mean, data_std = my_func3(data)
print('mean:{0:3.2f}, std:{1:3.2f}'.format(data_mean,data_std))
```

```
mean:-0.26, std:1.10
```

43. 하나의 변수에 두 개의 반환값을 tuple 형으로 받고 출력했습니다.

In [43]:
```
output = my_func3(data)
print(output)
print(type(output))
print('mean:{0:3.2f}, std:{1:3.2f}'.format(output[0],output[1]))
```

```
(-0.2631937162492418, 1.098345570952248)
<class 'tuple'>
mean:-0.26, std:1.10
```

44. 하나의 ndarray 형을 save를 이용하여 파일에 저장하고, load 를 이용하여 읽어왔습니다.

In [44]:
```
data = np.random.randn(5)
print(data)
np.save('datafile.npy',data)
data=[]
print(data)
data = np.load('datafile.npy')
print(data)
```

```
[-0.64670266  0.29690281 -0.5308344   0.12563413 -2.02865636]
[]
[-0.64670266  0.29690281 -0.5308344   0.12563413 -2.02865636]
```

45. savez를 이용하여 여러 ndarray 형을 저장했습니다.

In [45]:
```
data1 = np.array([1,2,3])
data2 = np.array([10,20,30])
np.savez('datafile2.npz', data1 = data1, data2 = data2)
data1=[]
data2=[]
outfile = np.load('datafile2.npz')
print(outfile.files)
data1 = outfile['data1']
data2= outfile['data2']
print(data1)
print(data2)
```

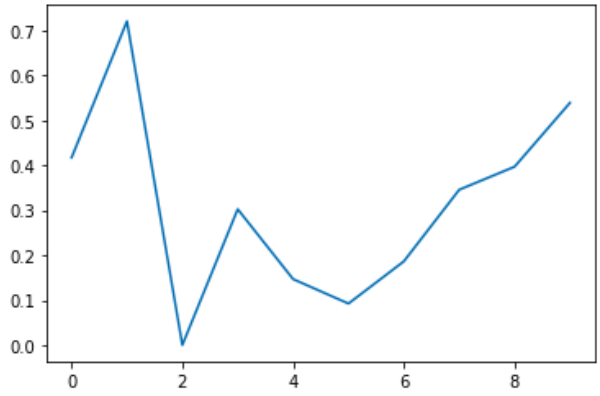
```
['data1', 'data2']
[1 2 3]
[10 20 30]
```

46. 임의의 그래프를 출력했습니다.

In [46]:
```
#리스트 1-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#data 작성
np.random.seed(1)
x = np.arange(10)
y = np.random.rand(10)

plt.plot(x,y)
plt.show()
```



47. 정한 리스트 번호의 규칙 이력을 삭제했습니다.

In [66]:
```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

48. 함수 f(x)를 정의했습니다.

In [48]:
```
#리스트 2-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def f(x):
    return (x - 2) * x * (x+2)
```

49. x에 숫자를 넣고, 대응하는 값을 출력했습니다.

In [49]:
```
#리스트 2-(2)
print(f(1))
```

```
-3
```

50. ndarray 배열 각각에 대응한 f를 ndarray로 반환받았습니다.

In [50]:
```
#리스트 2-(3)
print(f(np.array([1,2,3])))
```

```
[-3  0 15]
```

51. x의 범위와 간격을 정의했습니다.

In [51]:
```
#리스트 2-(4)
x = np.arange(-3, 3.5, 0.5)
print(x)
```

```
[-3.  -2.5 -2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2.   2.5  3. ]
```

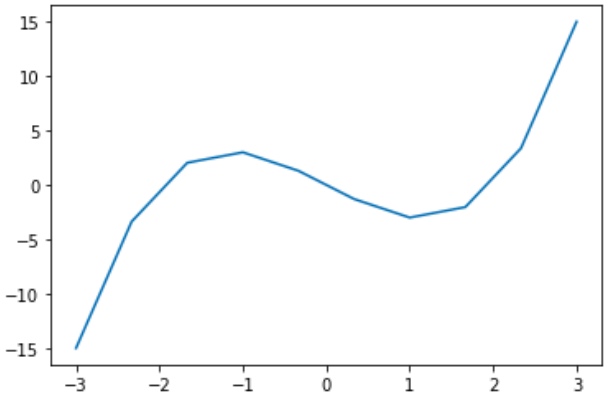
52. x의 범위를 linspace를 이용해 일정 간격으로 구간을 나눈 값을 출력했습니다.

In [52]:
```
#리스트 2-(5)
x = np.linspace(-3,3,10)
print(np.round(x,2))
```

```
[-3.   -2.33 -1.67 -1.   -0.33  0.33  1.    1.67  2.33  3.  ]
```

53. x의 값으로 그래프를 출력했습니다.

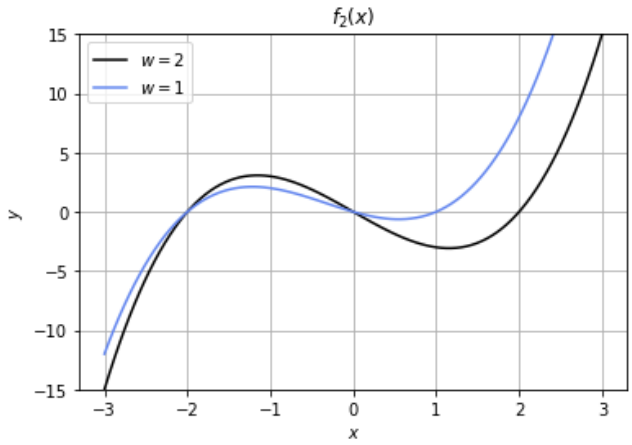
In [53]:
```
#리스트 2-(6)
plt.plot(x, f(x))
plt.show()
```



54. 구간을 세밀하게 나눠 매끄러운 그래프를 출력하고, 그리드, 라벨, 제목, 범례를 같이 출력했습니다.

In [54]:
```
#리스트 2-(7)

def f2(x, w):
    return(x-w)*x*(x+2)
x = np.linspace(-3,3,100)
plt.plot(x, f2(x,2), color='black', label='$w=2$')
plt.plot(x, f2(x,1), color='cornflowerblue', label='$w=1$')
plt.legend(loc="upper left")
plt.ylim(-15,15)
plt.title('$f_2(x)$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.grid(True)
plt.show()
```



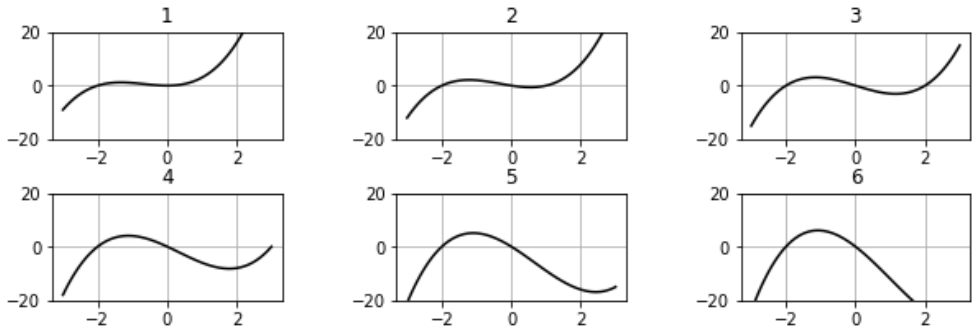
55. 사용할 수 있는 색상들을 출력했습니다.

In [55]:
```
#리스트 2-8)
import matplotlib
matplotlib.colors.cnames
```

Out[55]:
```
{'aliceblue': '#F0F8FF',
 'antiquewhite': '#FAEBD7',
 'aqua': '#00FFFF',
 'aquamarine': '#7FFFD4',
 'azure': '#F0FFFF',
 'beige': '#F5F5DC',
 'bisque': '#FFE4C4',
 'black': '#000000',
 'blanchedalmond': '#FFEBCD',
 'blue': '#0000FF',
 'blueviolet': '#8A2BE2',
 'brown': '#A52A2A',
 'burlywood': '#DEB887',
 'cadetblue': '#5F9EA0',
 'chartreuse': '#7FFF00',
 'chocolate': '#D2691E',
 'coral': '#FF7F50',
 'cornflowerblue': '#6495ED',
 'cornsilk': '#FFF8DC',
```

56. subplot을 이용하여 여러 그래프를 출력했습니다.

In [56]:
```
#리스트 2-9)
plt.figure(figsize=(10,3))
plt.subplots_adjust(wspace=0.5, hspace=0.5)
for i in range(6):
    plt.subplot(2,3,i+1)
    plt.title(i+1)
    plt.plot(x, f2(x, i), 'k')
    plt.ylim(-20,20)
    plt.grid(True)
plt.show()
```



57. f3을 정의하고 x0, x1값에 대한 f3의 값을 계산했습니다.

In [57]:
```
#리스트 3-1)
import numpy as np
import matplotlib.pyplot as plt

def f3(x0, x1):
    r = 2*x0**2 + x1**2
    ans = r* np.exp(-r)
    return ans
xn = 9
x0 = np.linspace(-2,2,xn)
x1 = np.linspace(-2,2,xn)
y = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        y[i1, i0]= f3(x0[i0],x1[i1])
```

58. xn이 9이므로 x0은 9개의 요소를 가지고 있습니다.

In [58]:
```
#리스트 3-(2)
print(x0)
```

```
[-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
```

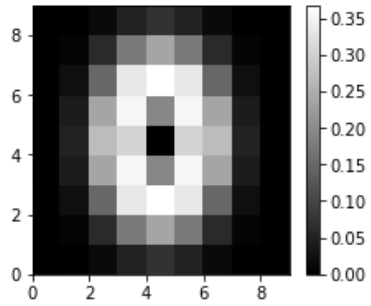
59. 해당 자리수의 소수값까지 round를 이용하여 반올림해주었습니다.

In [59]:
```
#리스트 3-(3)
print(np.round(y,1))
```

```
[[0.  0.  0.  0.  0.1 0.  0.  0.  0. ]
 [0.  0.  0.1 0.2 0.2 0.2 0.1 0.  0. ]
 [0.  0.  0.1 0.3 0.4 0.3 0.1 0.  0. ]
 [0.  0.  0.2 0.4 0.2 0.4 0.2 0.  0. ]
 [0.  0.  0.3 0.3 0.  0.3 0.3 0.  0. ]
 [0.  0.  0.2 0.4 0.2 0.4 0.2 0.  0. ]
 [0.  0.  0.1 0.3 0.4 0.3 0.1 0.  0. ]
 [0.  0.  0.1 0.2 0.2 0.2 0.1 0.  0. ]
 [0.  0.  0.  0.  0.1 0.  0.  0.  0. ]]
```

60. 2차원 행렬의 요소를 색상으로 표현했습니다.

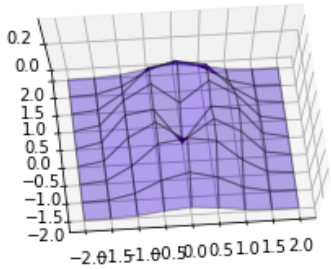
In [60]:
```
#리스트 3-(4)
plt.figure(figsize=(3.5,3))
plt.gray()
plt.pcolor(y)
plt.colorbar()
plt.show()
```



61. 3차원의 입체 그래프로 출력했습니다.

In [61]:
```
#리스트 3-(5)
from mpl_toolkits.mplot3d import Axes3D
xx0, xx1 = np.meshgrid(x0,x1)

plt.figure(figsize=(5,3.5))
ax = plt.subplot(1,1,1, projection = '3d')
ax.plot_surface(xx0,xx1,y,rstride=1,cstride=1,alpha=0.3,
                color='blue',edgecolor='black')
ax.set_zticks((0,0.2))
ax.view_init(75, -95)
plt.show()
```



62. x0, x1의 내용을 확인했습니다.

In [62]:
```
#리스트 3-(6)
print(x0)
print(x1)
```

```
[-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
[-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
```

63. meshgrid 명령으로 2차원 배열을 생성했습니다.

In [63]:
```
#리스트 3-(7)
print(xx0)
```

```
[[-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
 [-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]]
```

64. x1도 2차원 배열이 되었습니다.

In [64]:
```
#리스트 3-(8)
print(xx1)
```

```
[[-2.  -2.  -2.  -2.  -2.  -2.  -2.  -2.  -2. ]
 [-1.5 -1.5 -1.5 -1.5 -1.5 -1.5 -1.5 -1.5 -1.5]
 [-1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1. ]
 [-0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5]
 [ 1.   1.   1.   1.   1.   1.   1.   1.   1. ]
 [ 1.5  1.5  1.5  1.5  1.5  1.5  1.5  1.5  1.5]
 [ 2.   2.   2.   2.   2.   2.   2.   2.   2. ]]
```

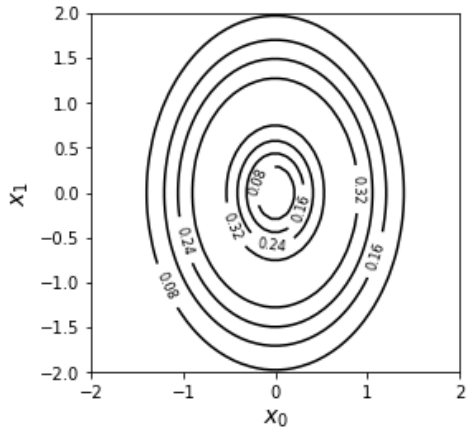
65. 함수의 높이를 등고선 플롯으로 출력했습니다.

In [65]:
```
#리스트 3-(9)
xn = 50
x0 = np.linspace(-2,2,xn)
x1 = np.linspace(-2,2,xn)

y = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        y[i1, i0] = f3(x0[i0], x1[i1])

xx0, xx1 = np.meshgrid(x0,x1)

plt.figure(1, figsize=(4,4))
cont = plt.contour(xx0, xx1, y, 5, colors='black')
cont.clabel(fmt='%3.2f', fontsize=8)
plt.xlabel('$x_0$', fontsize=14)
plt.ylabel('$x_1$', fontsize=14)
plt.show()
```



배열을 이용하여 그래프를 효과적으로 그리는 방법을 알게 되었습니다.

직관적이고 명확한 그래프가 나오는 것 같습니다.

이범석

국민대학교 소프트웨어학부, 20171664

Mobile 010-6401-6042

qpwoeiru6486@gmail.com

ijkoo16@kookmin.ac.kr