

11.

```
a=[[1],[2,3],[4,5,6]]
```

```
n=0
```

```
for b in a:
```

```
    for d in b :
```

```
        n+=1
```

```
print(n)
```

6

12.13.

```
n=[1,2,3,4]
```

```
for l in range(len(n)):
```

```
    print(i)
```

6번의 함수 실행

18.

```
t=0
```

```
for p in range(3):
```

```
    if p <=1:          0,1
```

```
        for q in range(4):      0,1,2,3
```

```
            t+=q(t=t+q)
```

```
        for r in range(2):      0,1
```

```
            t +=r(t=t+r)
```

```
print(t*r)
```

14

10/31

숫자들의 나열은 벡터의 형태가 된다.

행렬은 벡터는 아니지만, 길게 만들면, 한줄한줄 벡터화할 수 있다.

(이미지는 행렬이다. 그걸 길게 올려놓은 것을 벡터화한다라고 말한다.)

모든 데이터는 벡터형태로 했을 때 다루기가 쉽다.

칼라 이미지는 3차원, 동영상은 4차원

소리도 벡터화 된다.

텍스트도 벡터화시킨다. 1첫째 단어 1000000000000 (0을 5만개)

numpy는 라이브러리

```
a=[1,3,5]
```

```
b=[2,4,6]
```

```
c=a+b
```

c # 곱하기, 더하기 안되는데, 이것을 해주는 것이 numpy이다.

```
→ [1,3,5,2,4,6]
```

```
import numpy # import numpy as np A=np.array(a), B=np.array(b)
```

```
A=numpy.array(a)
```

B=numpy.array(b) #array는 계산가능한 것을 바꾸어준다.

```
A+B
```

```
→ array([3, 7, 11])
```

```
type(A)
```

```
→ numpy.ndarray
```

```
import numpy as np
```

```
A=np.array(a) ; B=np.array(b)
```

```
X=np.array([[1,2,3],[4,5,6]])
```

```
X
```

```
→ array([[1,2,3],  
         [4,5,6]])
```

```
X.shape
```

```
→(2,3)
```

11.01

1. numpy.A.D.f

2. import numpy

numpy.A.D.f

3. from numpy import A

A.B.f

4. from numpy import A.D

#numpy 안에 A와 B가 부분집합으로, A안에 D

#import numpy numpy.A.D.f

#from numpy import A

A.D.f

#from numpy import A.D

```
import numpy as np
```

```
import matplotlib.pyplot as plt # from matplotlib import pyplot as plt
```

.은 포함관계이다.

```
np.empty([2,3], dtype='int')
→ array([[ 0, 1072168960,  0],
        [1072168960,  0,  0]])
```

```
np.empty([2,3])
→array([[ 4.64583300e-308,  4.66664552e-308, -2.57703408e+010],
        [-4.70231646e-064,  2.26262303e-319,  1.46914166e+195]])
```

numpy.empty(배열, dtype=자료형)을 사용하여 특정한 값으로 초기화하지 않는 배열을 생성할 수 있다.

```
np.zeros([2,3])
→ array([[0.,  0.,  0.],
        [0.,  0.,  0.]])
```

[[0,0,0],[0,0,0]] #list를 만든 것, 그런데 계산 안 됨. 계산되는 array로 만드는 방법이 밑에,

np.array([[0,0,0], [0,0,0]]) #np.zeros([2,3])과 똑같음

그냥 [[0,0,0], [0,0,0]] 은 벡터일 뿐이다.

np.ones([2,3]) #1이 6개 만들어진다. ones라는 함수는 데이터타입을 float로 하는구나 추측, 왜? 결과에 1. 이 돼서 = 따라서, np.ones([2,3], dtype='int')

#float64라고 쓰면 조금의 오차도 만들지 않으려고, 단점은 메모리를 차지한다. 정확도와 데이터 쓰는 양과 메모리가 반비례

np.arange(5) #for loop에서 range는 계산은 안 됨, 여기서는 계산이 됨.

```
→array([0,1,2,3,4])
```

```
np.arange(0,10,2)
```

```
→array([0,2,4,6,8])
```

```
np.arange(0,10,2, dtype='float64')
```

```
→ array([0., 2., 4., 6., 8.])
```

np.linspace(0,10,6) #0부터 10까지 6개의 숫자로 나눠라. linspace에서 linear은 늘 똑같다. 첫 번째에서 두 번째로 가는, 두 번째에서 세 번째로 가는 차이가 다 똑같다.

```
→ array([0., 2., 4., 6., 8., 10.])
```

```
X=np.array([4,5,6])
```

X

```
→ array([4,5,6]) #벡터가 만들어진다.
```

```
X=np.array([[1,2,3],[4,5,6]]) #2by3
```

```
X=np.array([[1,2],[4,5],[8,9]]) #3by2
```

X

```
→array([[1,2],
        [4,5],
        [8,9]])
```

X=np.array([[1,2], [4,5], [8,9]]) #그냥 벡터는 1차원, 직사각형 매트릭스 2차원, volume이 있으면 3차원, 3차원으로 표기도 가능하다. 대괄호가 두 개 나오면 2차원, 대괄호가 세 개 나오면 3차원이다.

```
X=np.array([[[1,2], [4,5], [8,9]], [[1,2], [4,5], [8,9]]]) #2차원 행렬이 두 개 있으면 3차원이네
```

X

```
→ array([[[1,2],
          [4,5],
          [8,9]],
        [[1,2],
          [4,5],
          [8,9]]])
```

X.ndim

→3

X.shape

→(2,3,2) # 3x2 가 2개 있다.

X.dtype

→ dtype('int32')

X.astype(np.float64)

```
→ array([[[1., 2.],
          [4., 5.],
          [8., 9.]],
        [[1., 2.],
          [4., 5.],
          [8., 9.]])
```

np.zeros_like(X) # X*0 과 같다.

```
→ array([[[0, 0],
          [0, 0],
          [0, 0]]])
```

```
[0, 0]],
```

```
[[0, 0],  
 [0, 0],  
 [0, 0]])
```

```
data= np.random. normal(0,1,100) #정규분포 모양의 데이터를 만들어준다. 평균0, 표준편  
차1, 100(100개의 데이터를 만들어라.) random 하게 만들어봐라.  
print(data)
```

```
data.shape  
→(100,)
```

```
data.ndim  
→1
```

```
data=np.random.normal(0,1,100)  
print(data)  
plt.hist(data, bins=10) #matplotlib이라는 라이브러리 안에 있는 plt라는 subset안에 있는 hist  
바구니가 10개  
plt.show() # 그래프의 y축에는 0을 포함한 정수값만 나올 수 있다. 전체 값은 몇 개 인가?  
100개이다. (시험)
```

```
X=np.ones ([2,3,4]) #직사각형은 2차원, 3차원은 그게 여러 장 쌓인 것 [3,2,4] 이런 식으  
로 24개에 맞춰서 바꿀 수 있다. 그런데 reshape(-1,3,2) 는 -1은 니가 알아서 해라.
```

X

```
→ array([[[1., 1., 1., 1.],  
          [1., 1., 1., 1.],  
          [1., 1., 1., 1.]],
```

```
        [[1., 1., 1., 1.],  
         [1., 1., 1., 1.],  
         [1., 1., 1., 1.]])
```

2by3by4 에 들어있는 element의 개수 24개 - 바꾸더라도 24개가 되도록 바꿀 수 있다.
안되면 reshape은 error message가 뜬다.

```
Y=X.reshape(-1,3,2) #4인데, 4인지 뭔지 모르겠다. 그러면 -1넣을 수 있다. 4넣어도 똑같  
은 값이 나온다.
```

Y

```
→array([[[1., 1.],  
         [1., 1.],
```

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

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

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

```
[[1., 1.],  
 [1., 1.],  
 [1., 1.]])
```

```
np.allclose(X.reshape(-1,3,2),Y)  
true  
#allclose는 똑같은지 아닌지 판단하는 함수
```

```
a=np.random.randint(0,10,[2,3])  
b=np.random.random([2,3])  
np.savez("test",a,b) ---- 파일로 저장
```

! ls -al test* --- 실제 저장이 됐는지 안됐는지 보는 것이다.

```
del a,b #메모리 상에서 지우고 싶으면,  
%who #variable중에 뭐가 available한 지가 나옴.
```

```
npzfiles = np.load("test.npz") #저장한 후 파일을 불러오고 싶을 때 (앞에 저장했으니)  
npzfiles.files  
→['arr_0', 'arr_1']  
npzfiles['arr_0']  
→array([1,0,3],  
        [4,0,6]))
```

Comma Separated Values (CSV) (엑셀에서도 부를 수 있음)

```
data=np.loadtxt("regression.csv", delimiter=",",skiprows=1, dtype={'names': ("X",  
"Y"),'formats': ('f','f')}) #데이터를 파일로 주는 방법이다, delimiter 분리, skiprows는 첫  
번째 row(X, Y)는 빼겠다. f는 float라는 것, integer하고 싶으면, I쓰면 됨.
```

```
np.savetxt("regression_saved.csv", data, delimiter=",")  
!ls -al regression_saved.csv : 저장하는 것
```

```
arr= np.random.random([5,2,3])
```

```
print(type(arr))
```

```
print(len(arr))
```

```
print(arr.shape)
```

```
print(arr.ndim)
```

```
print(arr.size)
```

```
print(arr.dtype)
```

```
→ <class 'numpy.ndarray'>
```

```
5
```

```
(5, 2, 3)
```

```
3
```

```
30
```

```
float64
```

```
a=np.arange(1,5) #index만들어내라 1,2,3,4
```

```
b=np.arange(9,5,-1) #index 만들어내라 9,8,7,6
```

```
-- 계산 가능한 상태가 되었기 때문에,
```

```
print(a-b)
```

```
print(a *b) 계산 가능하다.
```

```
a,b 모두 numpy array이기 때문이다.
```

matrix는 다음에

```
a=np.arange(1,10).reshape(3,3) #reshape 해서 2차원으로 만들자
```

```
b=np.arange(9,0,-1).reshape(3,3)
```

```
→[[1 2 3]
```

```
   [4 5 6]
```

```
   [7 8 9]]
```

```
[[9 8 7]
```

```
   [6 5 4]
```

```
   [3 2 1]]
```

```
a==b
```

```
→array([[False, False, False],
```

```
        [False,  True, False],
```

```
        [False, False, False]])
```

```
a>b
```

```
→ array([[False, False, False],
```

```
        [False, False,  True],
```

```
[ True,  True,  True]])
```

#비교하려면 dimension과 shape이 같아야 한다.

a.sum() #a가 numpy로부터 만들어진 산물이다. a라는 자체가 numpy이다. numpy가 만들어냈다. 함수니까 괄호친다. 자기 자신을 sum하는 건데 ()뭐 안써도 된다.

np.sum(a) #위의 것과 같다. np속에 있는 sum을 쓸건데, 뭘 sum을 하나? a

a.sum(axis=0), # 3x3 의 2차원, 첫 번째 차원은 1,2,3, 4,5,6, 7,8,9 두 번째 차원은 1,4,7 2,5,8 3,6,9 // a.sum(axis=0)은 행의 관점에서 sum을 해라

→ (array([12, 15, 18]))

np.sum(a, axis=0) #행의 관점에서 합을 하는 것이다. 첫 번째 차원

a.sum(axis=1), np.sum(a, axis=1)

→ array([6, 15, 24]), array([6, 15, 24]))

#reshape이 브로드캐스팅이다. (broadcasting)

a=np.arange(1,25). reshape(4,6) #2차원으로 4X6

→ 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]])

a + 100

→

array([[101, 102, 103, 104, 105, 106],
 [107, 108, 109, 110, 111, 112],
 [113, 114, 115, 116, 117, 118],
 [119, 120, 121, 122, 123, 124]])

b = np.arange(6)

b

→ array([0, 1, 2, 3, 4, 5]) #a는 4by 6, b는 그냥 6

a + b

array([[1, 3, 5, 7, 9, 11],
 [7, 9, 11, 13, 15, 17],
 [13, 15, 17, 19, 21, 23],
 [19, 21, 23, 25, 27, 29]])

#sound가 continuous wave이지만, 컴퓨터에서는 하나하나 값들로 담김. 1초동안 간격은 같지만, 얼마나 듬성듬성, 뽕뽕이 담을 것인가?=sampling rate.

#sampling rate를 만으로 한다고 정의한다면. 1초동안 만 개의 숫자를 담을 거다. 4만 4천 백이 CD, 그 이상은 사람의 귀로는 구분이 안되는데, 4만 4천 백이 인간이 구분할 수 있는 최대 음질.

1초를 표현하는 데 숫자의 개수가 4만 4천 백 개 있다. 아 - 소리 내는데 기록하는데 19만 ++ sine wave를 19만 2천개의 숫자로 표현도 되고, 3백 개도 되고, 이게 sampling rate이다. 1초에 얼마이다. Hz를 반드시 붙인다. 1초에 몇 번 떨리느냐 1초 당이라는 말이 들어가는 순간 모든 경우에서 Hz를 쓴다. 1초에 몇번 왔다갔다 하는가 pitch표현할 때도,

#sampling rate

11.5 화

list는 어떠한 계산도 안되니, numpy이 담아서 한다.

다차원의 array를 만든다.

차원에 대한 이야기

1차원이면 벡터, 2차원이면 정사각형 형태의 매트릭스, 3차원이면 volume 4차원이면 한 차원 더 늘어나서 되는 것이다.

#벡터는 숫자열이다. (1,0) (0,1) 이런 것 모두다 벡터이다.

#가로축이 theta - 오르락 내리락...

#pure tone들의 합이 복잡한 sound를 만들어 낸다. pure tone이 sine 코사인 wave, -- 그런것들을 - sinusoidal function - 이런 걸 만들어 내는 걸 phasor

#sine의 입력값으로 뭘 쓰는가? sin()

0~~~ 2X파이 = 6.1xxx 0은 0도, 2파이는 360도 -- 몇도는 degree 몇 도 이런것을 0, 2 π 로 표현하는 것이 radians 이다.

#sin() cos() 입력값은 degree가 아닌 radians가 들어가야 한다.

#degree 0 180 360

#radian 0 π 2 π

#sin 곡선

input은 각도값, 켄타

#frequency해당되는 개념이 들어가는가? (sin (켄타)라고 쓴다면 시간의 개념이 들어갈까?) - 시간의 개념이 안 들어가 있다. 몇 초에 몇 바퀴 돈든지는 안들어 있다. frequency는 초당 얼마, 몇번 왔다갔다 - 정확히 정의 안되었다.

#각도개념뿐만 아니라 초 개념도 넣어 줘야 진정한 소리가 나온다. 소리라는 실체는 시간의 개념이 들어있어야 함. sin켄타 라고만 하면 실제의 소리를 만들어 낼 수가 없다.

#parameter setting

amp=1 #range[0.0, 1.0]

sr=10000 #sampling rate, Hz

dur=0.5 #in seconds

freq=100.0 #sine frequency, Hz

#sampling rate에 단위도 Hz고, frequency도 Hz이다. sampling rate은 몇개의 숫자로 해

서, 음의 음질 상 얼마나 고 해상도로 하는가(음질의 해상도) 1초에 만개의 숫자를 가지고 표현하겠다. frequency는 1초에 태극문양이 얼마나 들어가는가

t 0.0001 0.0002 0.0003 0.5000 #이렇게 하면 몇 개가 ? 1초라면 만개가 들어가야 한다.

```
t=np.arange(1, sr) #numpy쓰는 이유는 계산하려고, // 1에서 9999 까지 만들어짐. sr+1
하면 만 개가 만들어진다. 1초라면 딱 맞는데, 0.5초라서 안맞는데, 맞추려면 sr*dur+1/sr
t # array([1.000e-04, 2.000e-04,..... 4.998e-01]) 이런식으로 나온다.
```

#썰타에 들어가는 각도값을 phase라고 한다. 타임과 연동시켜서 phase로 바꾸어줘야 한다.

theta=t*2*np.pi * freq #t * 2*np.pi (0초에서 1초까지 있는데 2π 곱하면 한바퀴 돈다. 몇 바퀴 만드느냐가 freq, freq=1이면 한바퀴돌면 끝)

#time과 theta는 벡터, time의 벡터의 사이즈 와 썰타의 벡터의 사이즈는 같다? 맞다.

```
#ipd. Audio(s, rate=sr)
```

11.7 목

#phasor , 오일러 phasor

#plotting하기 위한 function부르기 위해

```
# import matplotlib.pyplot (from matplotlib import pyplot)
```

#parameter setting #목적 : 무엇을 바꿀 때 여기만 바꿔주면 된다. 목적상은 parameter, 파이썬에서 코딩할 때는 variable이다.

#function들이 받아들이는 입력은 각도값이다. radian을 넣어야 함. time x degree x (sin cos 모두, 오일러 공식에서)

#time이 필요한 이유? 각도값만 넣어서는 실체의 소리를 만들 수 없다. 그게 왜 그런지 보여 주겠다.

```
from matplotlib import pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D
```

```
from mpl_toolkits.axes_grid1 import make_axes_locatable
```

```
import IPython.display as ipd
```

```
import numpy as np
```

```
%matplotlib notebook
```

```
from scipy.signal import lfilter
```

theta= np.arange(0, 2*np.pi) ##만약, theta=np.arange(0, 2*np.pi, 0.1)하면 훨씬 더 뽀뽀하게 만들 수 있다.(밑에서)

theta #radian으로 정리

```
# array([0., 1., 2., 3., 4., 5., 6.])
```

```
s=np.sin(theta)
```

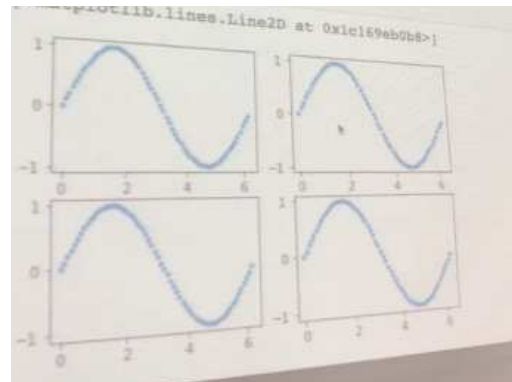
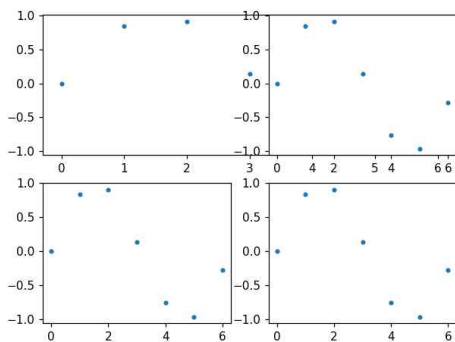
s #총 7개 벡터값이 나온다.

```
array([ 0.          ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
        -0.95892427, -0.2794155 ])
```

```

fig= plt.figure()
ax=fig.add_subplot(211)
ax.plot(theta, s, '.')
ax=fig.add_subplot(222)
ax.plot(theta, s, '.')
ax=fig.add_subplot(223)
ax.plot(theta, s, '.')
ax=fig.add_subplot(224)
ax.plot(theta,s, '.')
#theta는 0부터 2파이까지 만든거고, s는 sine이라는 것에 통과시켜서 /// 총 7개의 theta와
sine값 (corresponding)// (0,0) ... y축은sine 함수의 결과라고 보면됨

```

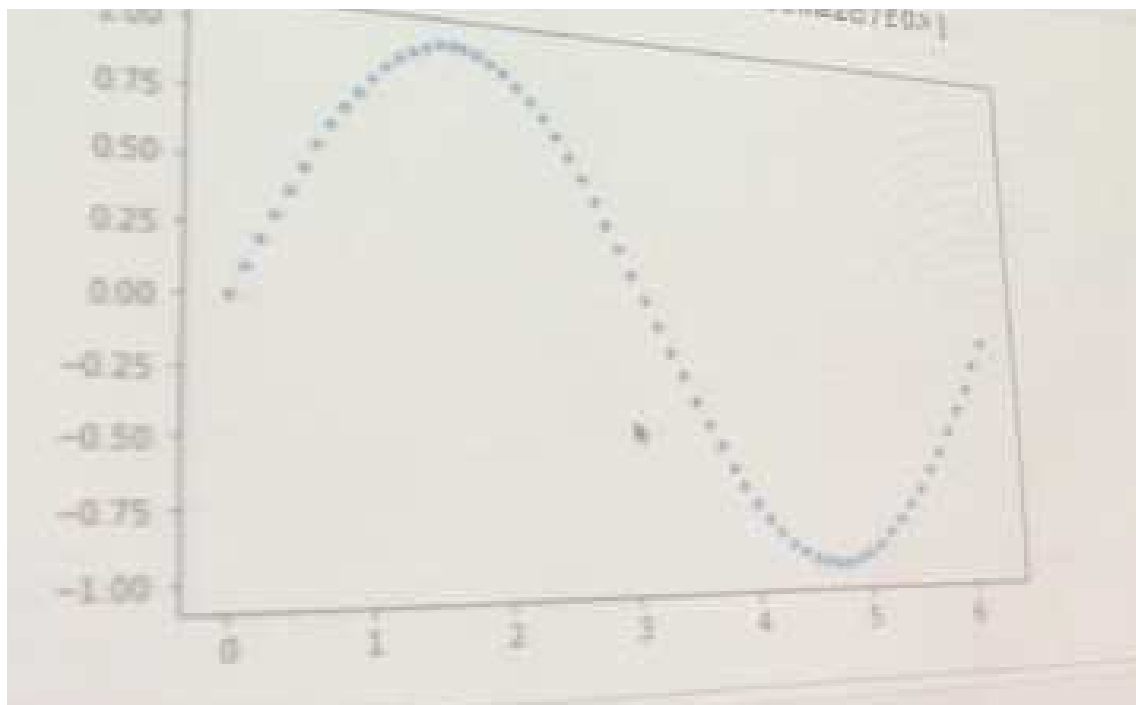
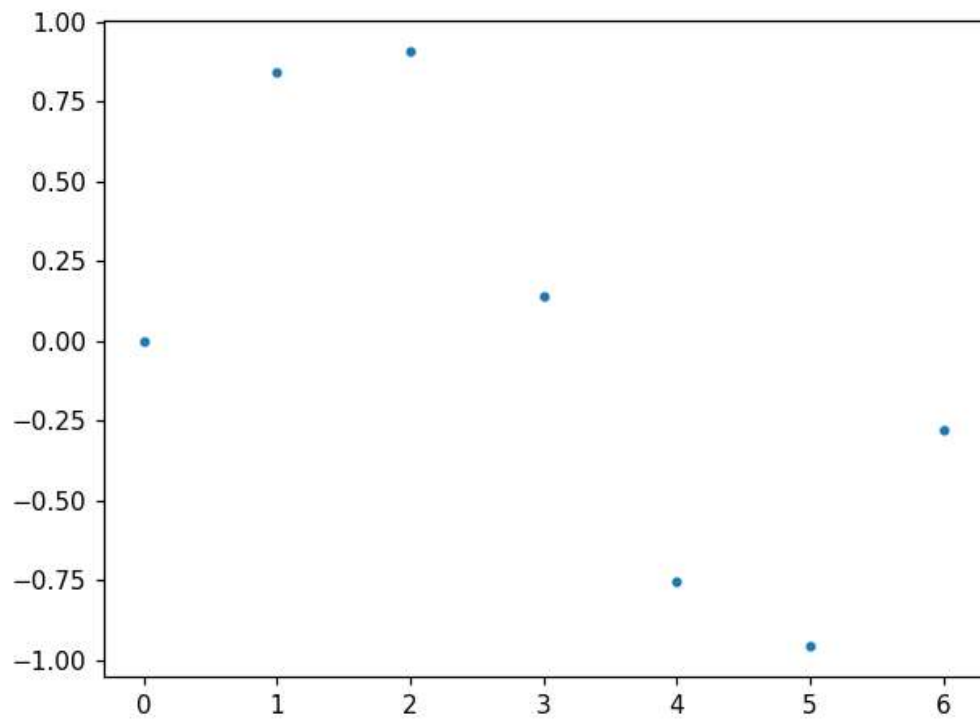


```

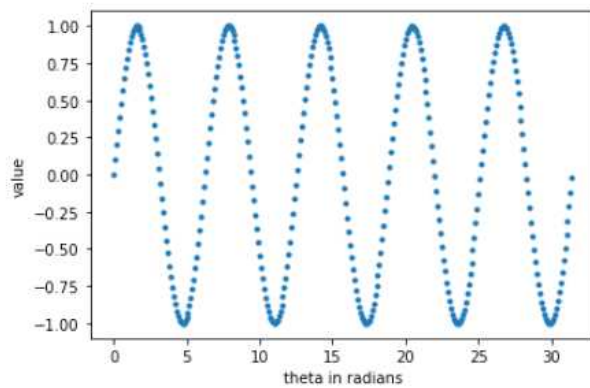
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(theta,s, '-') # '.'을 빼고 '-'로 하면 line으로 된다. 아무것도 없어도 line이 된다.
ax.set_xlabel('theta in radians')
ax.set_ylabel('value') #시험 : x축에서는 equidistance한데, y축에서는 equidistance안하다.
그럼 어떤 경우가 둘다 equidistance한가? linear(line같이 생기면 x의 변화가
equidistance하면, y의 변화도 equidistance하다 - nonlinear이면 y변화는 equidistance안
하다 linear한 것은 ax+B의 형태이다. 이것을 제외한 모든 함수는 nonlinear하다. y=logx ,
y=1/x 등등 다 nonlinear하다 곡선이 나타난다는 자체가 x의 equidistance가 y에는 나타나
지 않는다는 뜻 )

```

#theta는 0부터 2파이까지 만들었고, sin function으로 들어가서 s가 나옴.



theta= np.arange(0, 2*np.pi*5, 0.1) #밑에 그래프에서 보면 5번 반복된다.



-----앞 부분에는 시간이 전혀 개입되어 있지 않다.

```
# generate signal by complex-phasor
```

```
c = np.exp(theta*1j)
```

```
t = np.arange(1, sr) --- sampling rate만큼의 time tick을 만드는 것이다.
```

#time tick의 개수를 index로 먼저 만든다. 만약 1초라면 time tick을 총 몇 개 만드는 것인가? time tick의 개수는 sampling rate과 일치한다.

```
t= np.arange(1,sr*dur+1)/sr
```

#1가 아니라면, 그거보다 작아져야 한다.

#dur 이 1초면, 1 0.5초면 0.5가 돼서 sr의 개수만큼time tick이 만들어진다. dur이 비율로 반영된 것, timetick을 sr을 기반으로 해서 먼저 만든 것이다.

시간은 아니다. 그냥 time tick해서 index를 해준 거고, sr로 나누어주어야 시간이 된다.

#time과 연동되는 theta를 만드는 작업을 하는 것이다.

```
theta=t * 2*np.pi * freq
```

```
s= np.sin(theta)
```

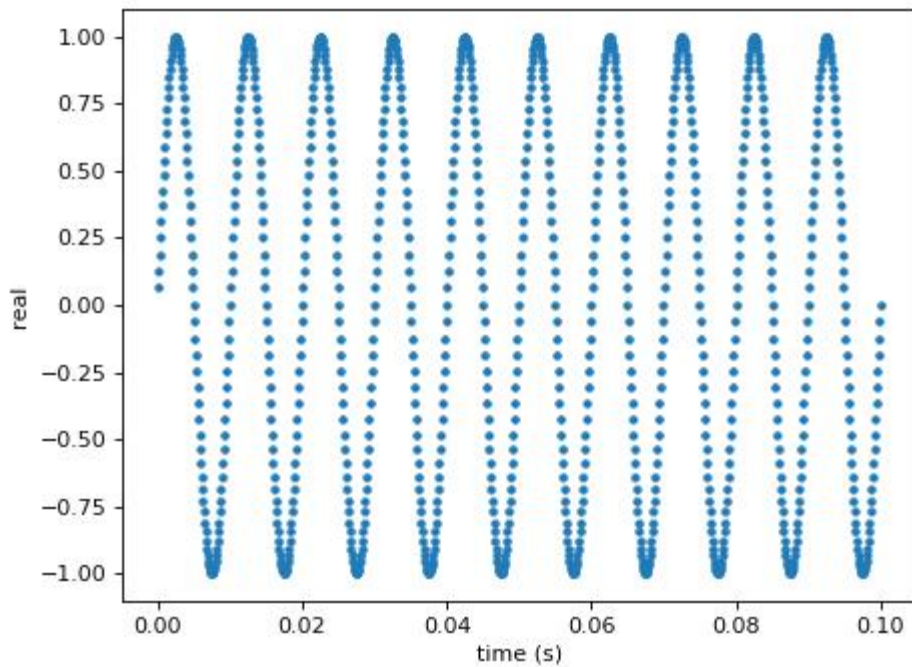
```
fig = plt.figure()
```

```
ax = fig.add_subplot(111)
```

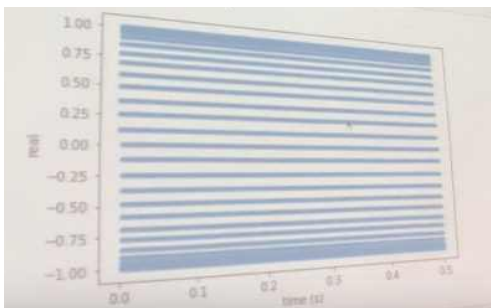
```
ax.plot(t[0:1000], s[0:1000], '.')
```

```
ax.set_xlabel('time (s)')
```

```
ax.set_ylabel('real')
```



#점들의 개수는? 1000개 , t와 s의 개수가 안맞으면 실행이 안된다. t[0:1000], s[0:1000] - 이렇게 되어 실행이 된다. 그냥 t,s로 하면 너무 뻑뻑해서 안보인다.



generate signal by complex-phasor

```
c = np.exp(theta*1j)
```

#exp도 함수다. np.exp가 큰 e라고 생각하면 됨. 그 안에 들어가는 것이 $\theta * 1j$ 이다.

1j는 I이다. 다 고정되어 있고, θ 만 바뀐다. sin과 마찬가지로이다.

c #표기법 : 컴퓨터가 쓰는 정보량이 똑같아 진다.(컴퓨터는 정보량을 정할 필요가 있다.) c에는 복소수(complex number)의 벡터가 꼭 들어있구나. 이렇게 생각하면 됨. 왜 복소수가 들어가있느냐? i를 썼으니 복소수 까지 확대 되는 것이다.

```
array([ 1.          +0.j          ,  0.99500417+0.09983342j,
        0.98006658+0.19866933j,  0.95533649+0.29552021j,
```

```

0.92106099+0.38941834j, 0.87758256+0.47942554j,
0.82533561+0.56464247j, 0.76484219+0.64421769j,
0.69670671+0.71735609j, 0.62160997+0.78332691j,
0.54030231+0.84147098j, 0.45359612+0.89120736j,
0.36235775+0.93203909j, 0.26749883+0.96355819j,
0.16996714+0.98544973j, 0.0707372 +0.99749499j,
-0.02919952+0.9995736j , -0.12884449+0.99166481j,
-0.22720209+0.97384763j, -0.32328957+0.94630009j,
-0.41614684+0.90929743j, -0.5048461 +0.86320937j,
-0.58850112+0.8084964j , -0.66627602+0.74570521j,
-0.73739372+0.67546318j, -0.80114362+0.59847214j,
-0.85688875+0.51550137j, -0.90407214+0.42737988j,
-0.94222234+0.33498815j, -0.97095817+0.23924933j,
-0.9899925 +0.14112001j, -0.99913515+0.04158066j,
-0.99829478-0.05837414j, -0.98747977-0.15774569j,
-0.96679819-0.2555411j , -0.93645669-0.35078323j,
-0.89675842-0.44252044j, -0.84810003-0.52983614j,
-0.79096771-0.61185789j, -0.7259323 -0.68776616j,
-0.65364362-0.7568025j , -0.57482395-0.81827711j,
-0.49026082-0.87157577j, -0.40079917-0.91616594j,
-0.30733287-0.95160207j, -0.2107958 -0.97753012j,
-0.11215253-0.993691j , -0.01238866-0.99992326j,
0.08749898-0.99616461j, 0.18651237-0.98245261j,
0.28366219-0.95892427j, 0.37797774-0.92581468j,
0.46851667-0.88345466j, 0.55437434-0.83226744j,
0.63469288-0.77276449j, 0.70866977-0.70554033j,
0.77556588-0.63126664j, 0.83471278-0.55068554j,
0.88551952-0.46460218j, 0.92747843-0.37387666j,
0.96017029-0.2794155j , 0.98326844-0.1821625j ,
0.9965421 -0.0830894j ])

```

```

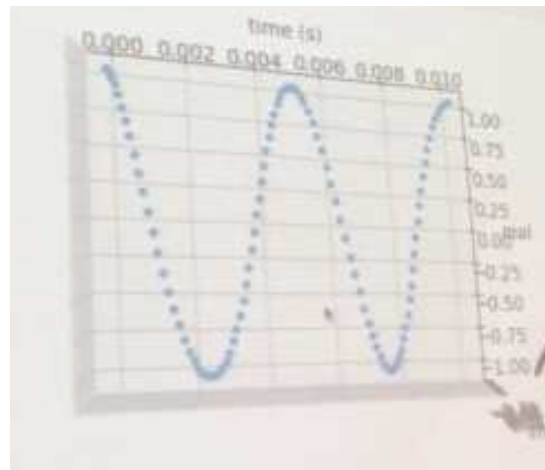
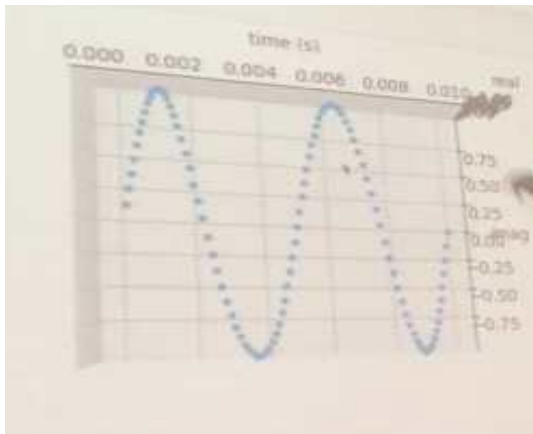
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(t[0:100], c.real[0:100], c.imag[0:100], '.') #t[0:100] s[0:100] -입력이 두 개,
// 여기서는 3차원 벡터 ..... // 3개의 값들은 서로서로 corresponding한다. 차례가 있는
데, 첫번째꺼 첫번째꺼 첫번째꺼 -- 한 점.. 총 1000개의 점이 찍힌다.
ax.set_xlabel('time (s)') #c.real이라고 하면, a+bi에서 a만 받아온다. c.imag라고 하면
b만 받아온다. real part와 imaginary part따로 받아와서
ax.set_ylabel('real')
ax.set_zlabel('imag') #c값들은 복소수형태로 나온다. 원래 값을 c라고 나오지만, 거기에는 2
개의 정보가 있다. sin 은 sin에 해당하는 하나만 나온다. exp function을 쓰면 2개의 값이
나온다.

```

#c.real이라고 하면 a만 쏙 받아온데, 0.9921147 만, c.imag라고 하면 1.25333234e-01j 만 받아온다.

#projection : 위에서 보면, real만 보는 것이다. 앞에서 보면 imaginary만 보는것(time은 항상 보고 있다.)

#real은 cosine과 같다. imaginery만 보면 sin이 나온다. (complex phasor(오일러 페이지)는 sin 과 cosine 둘다 가지고 있다.)



```
ipd.Audio(s, rate=sr) # ipd를 import했다. audio play하기 위해서
```

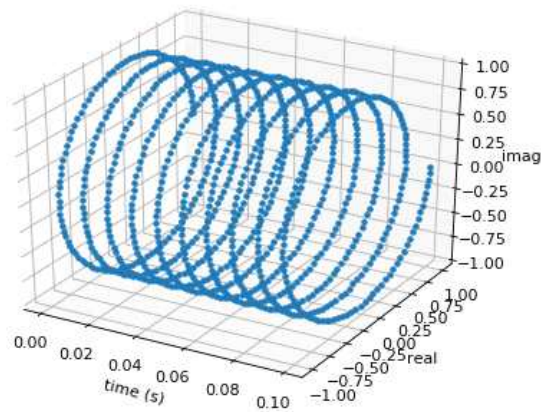
#s는 signal이고, 벡터와 원하는 sampling rate을 쓰면 됨.. s는 sin으로 만든 것이었는데, 그걸 안쓰고, c.imag로 해도 된다. c.real도 된다.

```
ipd.Audio(c.imag, rate=sr) #이 두개는 같다.
```

11.14

sine wave여러 개 쌓여서 성대에서 나는 소리

100Hz, 200Hz 이렇게 sine wave를 차곡차곡 더하면, sampling rate의 half까지 더하면,



pulse train이 만들어진다. pulse train의 가로축은 time이고, 세로축은 value(energy) 이다. 가로축을 time으로, 세로축은 frequency로 한 게 spectrogram이다. (frequency domain에서 보기)

어떤 frequency 성분대가 많은지 보여준다.

시간 축으로 쭉 나열해서 보는 것이다.

가로축은 frequency로, 세로축은 amplitude로 해서 보는 것이 spectrum이다. 한순간, 한 점의 시간에 어떤 성분이 있는지를 분석하는 것이다. 하나하나가 sine wave이다.(100Hz, 200Hz) 여기서 formant는 어디에 산맥이 있느냐 산맥의 특성에 따라 아~ 이~ 달라진다.

1. gradually decreasing하게 만드는 작업
 2. 산맥하나하나 만들기
- 이 두 작업을 할 것이다.

```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.axes_grid1 import make_axes_locatable
import IPython.display as ipd
import numpy as np
%matplotlib notebook
from scipy.signal import lfilter
```

```
# parameter setting
amp = 1          # range [0.0, 1.0]
sr = 10000       # sampling rate, Hz
dur = 0.5        # in seconds
freq = 440.0     # sine frequency, Hz
```

```
# generate time
t = np.arange(1, sr * dur+1)/sr
```

```
# generate phase
theta = t * 2*np.pi * freq
```

```
# generate phase
theta = t * 2*np.pi * freq
```

```
#generate signal by cosine-phasor
```

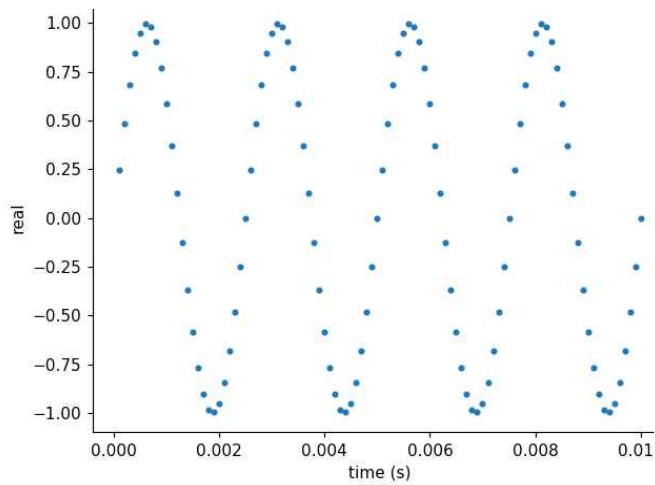
```
a=amp*np.sin(theta)    #cosine으로 바꾸어도 소리는 같다. 시작점이 달라질 뿐이다.
sin그래프를 cos로 바꾸려면 90도 차이가 있다. 2분의 파이만큼 차이, 8분의 파이만 이동을
했다. 그러면 소리가 달라지는 가? 아니다. 얼마큼 이동하든 상관없다. 1도2도3도4도 이동해
도 소리가 다 똑같다. 각도를 이야기할 때 phase라고 한다. phase에는 우리가 sensitive하지
않아 인식하지 못한다. phase shift에 대한 어떠한 sensitivity를 갖고 있지 않다. frequency
에 대해 sensitivity가 가지고 있다.
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(t[0:100], s[0:100], '.')
ax.set_xlabel('time (s)')
ax.set_ylabel('real')
```

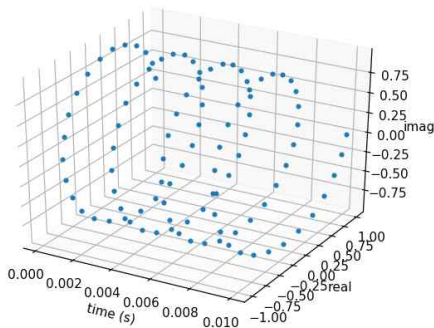
```
ipd.Audio(s, rate=sr)  #옥타브를 떨어뜨리면 배수로 하면 된다.
55, 110 ----- 같은 음이 나온다. '라'로 통일된다.
```

```
#generate signal by complex phasor
```

```
c=amp*np.exp(theta*1j) #theta들어간 것만 빼면, sine들어간것과 똑같다. #complex
number를 plotting못한다. 그래서 a에 real값, b에 imaginary... 2차원에서 ./ sin wave는
1차원상에서 찍는 것이고, 이거는 이차원 상에서 찍는 방법으로 구현한다.
```



```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(t[0:100], c.real[0:100], c.imag[0:100], '.')
ax.set_xlabel('time (s)')
ax.set_ylabel('real')
ax.set_zlabel('imag')
```

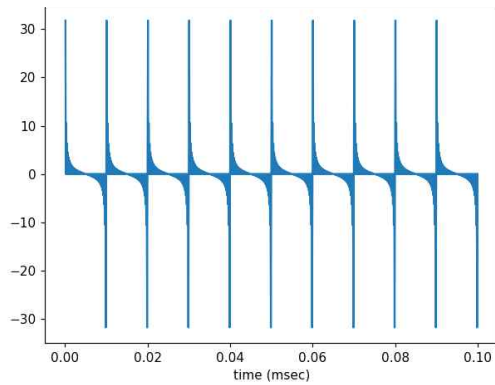


#import2번 실행하면, 3차원으로 돌려볼 수 있다.

```
ipd.Audio(c.real, rate=sr)
#real이니까 cosine이다. sin을 썼다면 c.imag하고 똑같은 소리가 나온다. cos과 sin소리가 다르지 않다.
```

```
# generate samples, note conversion to float32 array
F0 = 100; Fend = int(sr/2); s = np.zeros(len(t));
for freq in range(F0, Fend+1, F0):
    theta = t * 2*np.pi * freq
    tmp = amp * np.sin(theta)
    s = s + tmp
fig = plt.figure()
```

```
ax = fig.add_subplot(111)
ax.plot(t[0:1000], s[0:1000]);
ax.set_xlabel('time (msec)')
ipd.Audio(s, rate=sr)
```



-- 이 그림은 waveform이고, 스펙트럼이 아니다. 하나 time point에서의 spectrum, 어떤 주파수 성분이 많은지, 100 200 300 400 amplitude가 똑같으니 decreasing하게 만들고, 하나하나 산맥을 만들자.

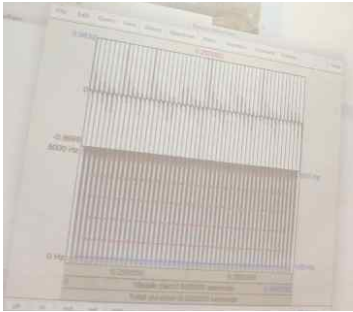
```
def hz2w(F, sr):
    NyFreq = sr/2;
    w = F/NyFreq *np.pi;
    return w

def resonance (srate, F, BW):
    a2 = np.exp(-hz2w(BW,srate))
    omega = F*2*np.pi/srate
    a1 = -2*np.sqrt(a2)*np.cos(omega)
    a = np.array([1, a1, a2])
    b = np.array([sum(a)])
    return a, b
```

#function 을 누가 만들었는데, 만든 실체, 만든 방법이다. def라고 적고 function name을 적는다. function이 두 개 있는 것이다. 괄호 열고, 내가 쓰고 싶은 입력들을 쓴다. 위의 것은 입력은 2개 밑의 것은 입력은 3개 return은 출력 return이 없으면 출력 없다.

#hn2w만들었는데, resonance 이 함수 속에서 hn2w를 또 불러온다. 우리는 resonance만 쓰면 됨. hn2w몰라도 됨.

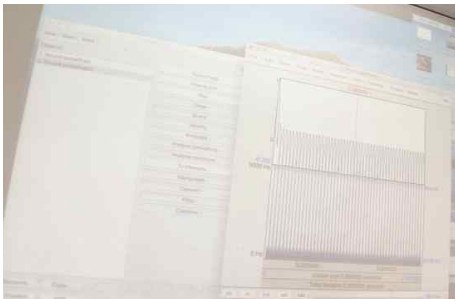
```
RG = 0 # RG is the frequency of the Glottal Resonator
BWG = 100 # BWG is the bandwidth of the Glottal Resonator
```



a, b=resonance(sr, RG, BWG) #산맥이 뾰족하면 width가 작고, 뚱뚱하게 생겼으면 width가 크다. 100정도면 꽤나 뚱뚱한 것이다. #0위치에다가 뚱뚱한 산을 만들어라.

s = lfilter(b, a, s, axis=0)

ipd.Audio(s, rate=sr)



아까는 북 찢어지는 소리였는데, 이제는 좀 더 부드러운 소리

RG = 500 # RG is the frequency of the Glottal Resonator

BWG = 60 # BWG is the bandwidth of the Glottal Resonator

a, b=resonance(sr, RG, BWG)

s = lfilter(b, a, s, axis=0)

ipd.Audio(s, rate=sr)

#formant를 만들기 시작, 500에다가 산맥을 하나 만들어 보자. 60이면 꽤나 훌쩍하다. 소리는 점점 사람이 되가고 있다. resonance함수를 써서 어디에다가 얼마나 뚱뚱하게 만들어서, a,b구하고 그걸 lfilter에 넣어주고, s는 이전단계의 signal이다. s는 위에서 받아서 또 쓰고 또 쓰고,

RG = 1500 # RG is the frequency of the Glottal Resonator

BWG = 200 # BWG is the bandwidth of the Glottal Resonator

a, b=resonance(sr, RG, BWG)

s = lfilter(b, a, s, axis=0)

ipd.Audio(s, rate=sr)

RG = 2500 # RG is the frequency of the Glottal Resonator

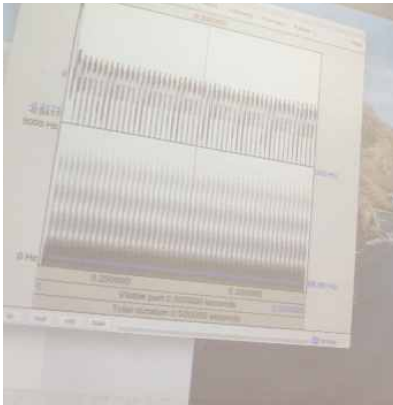
BWG = 200 # BWG is the bandwidth of the Glottal Resonator

a, b=resonance(sr, RG, BWG)

s = lfilter(b, a, s, axis=0)

```
ipd.Audio(s, rate=sr)
```

```
RG = 3500 # RG is the frequency of the Glottal Resonator  
BWG = 200 # BWG is the bandwidth of the Glottal Resonator  
a, b=resonance(sr, RG, BWG)  
s = lfilter(b, a, s, axis=0)  
ipd.Audio(s, rate=sr)
```



```
s = lfilter(np.array([1, -1]), np.array([1]), s)
```

ipd.Audio(s, rate=sr) #입술이 만약 없다면, 위의 소리에서 끝이다. 입술이 있기 때문에, ex) 나팔에 있는 입술 - 소리가 공명되서 크게 나가는 것이다. 입술이 없는 상태에서 - 입술을 만들면 - 소리가 더 커진다. 이게 여러분들의 어어 소리와 동일. 미묘하게 빠진게 있지만. ex)praat에서 500 1500되는 거 찾으면,.. // 제일 중간이다 사실. 소리도 그런소리가 난다. praat에서 만든 것과 위에서 만든 소리 별반 다르지 않다. 왜냐? 500 1500 2500으로 만들었고, praat에서 들은 소리도 500 1500으로 만들었기 때문이다.

