

10. 29 Numpy

```
In [9]: a = [1, 3, 5]
        b = [2, 4, 6]
        a+b
```

Out[9]: [1, 3, 5, 2, 4, 6]

```
In [11]: import numpy as np
```

```
In [7]: A = numpy.array(a)
        B = numpy.array(b)
```

```
In [8]: A+B
```

Out[8]: array([3, 7, 11])

```
In [10]: type(A)
```

Out[10]: numpy.ndarray

```
In [13]: X = np.array([[1,2,3],[4,5,6]])
```

```
In [14]: X
```

Out[14]: array([[1, 2, 3],
 [4, 5, 6]])

```
In [15]: X.shape #차원 의미. 2 x 3
```

Out[15]: (2, 3)

10.31 Numpy

```
In [3]: import numpy as np
        #from numpy import A # numpy 안에 있는 A를 가져오자
        import matplotlib.pyplot as plt
        #A.B = A 패키지에 있는 B
        #시험문제에 바로 나올 수 있는 것
        #from matplotlib import pyplot as plt 라고도 쓸 수 있음
```

```
In [7]: np.empty([2,3], dtype='int')
```

```
Out[7]: array([[          0, 1072168960,          0],
               [1072168960,          0,          0]])
```

```
In [9]: np.zeros([2,3])
```

```
Out[9]: array([[0., 0., 0.],
               [0., 0., 0.]])
```

```
In [12]: np.array([[0,0,0],[0,0,0]])
```

```
Out[12]: array([[0, 0, 0],
                [0, 0, 0]])
```

```
In [15]: np.ones([2,3])
```

```
Out[15]: array([[1., 1., 1.],
                [1., 1., 1.]])
```

```
In [18]: np.ones([2,3], dtype='int')
```

```
Out[18]: array([[1, 1, 1],
                [1, 1, 1]])
```

```
In [19]: np.arange(5) #range()랑 비슷
```

```
Out[19]: array([0, 1, 2, 3, 4])
```

```
In [20]: np.arange(0,10,2, dtype='float64')
```

```
Out[20]: array([0., 2., 4., 6., 8.])
```

```
In [21]: np.linspace(0,10,6) #linear space. 처음과 끝을 포함하여 6개로 똑같이 나누어준다 (10까지 포함인 모습)  
        # np.linspace(시작, 종료, 개수) : 개수에 맞게끔 시작과 종료 사이에 균등하게 분배
```

```
Out[21]: array([ 0.,  2.,  4.,  6.,  8., 10.])
```

```
In [24]: X = np.array([[1,2,3],[4,5,6]]) #[] 대괄호 두개 = 이차원 ; 대괄호 3개 = 3차원  
        X
```

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

```
In [33]: Y = np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]]) #2차원 두 개는 삼차원  
        Y
```

```
Out[33]: array([[[1, 2, 3],  
                [4, 5, 6]],  
               [[1, 2, 3],  
                [4, 5, 6]]])
```

```
In [28]: X.ndim #차원 알려주는
```

```
Out[28]: 2
```

```
In [29]: X.shape #2x3인 차원이다(?)
```

```
Out[29]: (2, 3)
```

```
In [34]: Y.shape #2x30이 2개인 차원
```

```
Out[34]: (2, 2, 3)
```

```
In [31]: X.dtype
```

```
Out[31]: dtype('int32')
```

```
In [35]: X.astype(float) #타입 변환
```

```
Out[35]: array([[1., 2., 3.],
                [4., 5., 6.]])
```

```
In [36]: np.zeros_like(X) # X*0해도 똑같은
# like(배열) 지정한 배열과 동일한 shape의 행열을 만들
# 종류 : np.zeros_like(), np.ones_like(), np.full_like(), np.empty_like()
```

```
Out[36]: array([[0, 0, 0],
                [0, 0, 0]])
```

```
In [38]: data = np.random.normal(0,1,100) # 정규분포normal distribution로 데이터 랜덤생성 .
# np.random.normal(정규분포 평균, 표준편차, (행, 열) or 개수) : 정규 분포 확률 밀도에서 표본 추출
print(data)
```

```
[ 0.17220936  3.63413319  1.79004811  0.43605779 -1.33729651  0.62791974
 -0.27576677  1.59299385  0.6866562  -0.39559477  0.14257864 -0.94367155
  0.61241866  0.5192135  0.34373035 -2.51310141  0.27534836 -0.92343738
  0.41763106 -0.58722763  1.44240769  2.1774158  -0.51703999  0.30987555
 -1.15293726 -0.05220753 -0.95385663  1.1801642  0.9309844  -0.25005757
 -1.07752434 -0.65908623 -0.24064692 -0.13865326 -0.24709929  0.02401597
  1.27916544  0.10482293  0.74158559  0.18170936 -1.03909671 -0.42212284
 -0.44528969  0.29721601  0.08980602 -0.52085625  0.92360159 -0.48917233
 -1.02450491  2.28863031  0.68496109  0.64009845 -1.41452731  1.26460298
  0.4795454  0.89346115  1.02752001 -1.92346047  1.05196862  0.33130818
  0.20861423  0.29873275  0.80336811  0.29719907 -1.12231016 -1.92610588
 -1.34559993 -0.10650772  0.12018836 -0.13937052 -0.29864508  0.63569818
  0.82943184  0.73361811 -0.03781109 -0.30225504 -0.86248564  0.09599242
 -0.56847991  1.39996332  1.24408033 -0.33508835  0.85697196 -0.61379087
  1.28343712  0.27461376 -0.42293543 -0.37213719 -0.11331183  1.90095436
 -0.35644987  0.58728444  0.38027114  0.1546923  -0.57848473 -1.60583919
 -1.15062939  1.9892623  -1.09017288  0.25643626]
```

```
In [39]: data.ndim
```

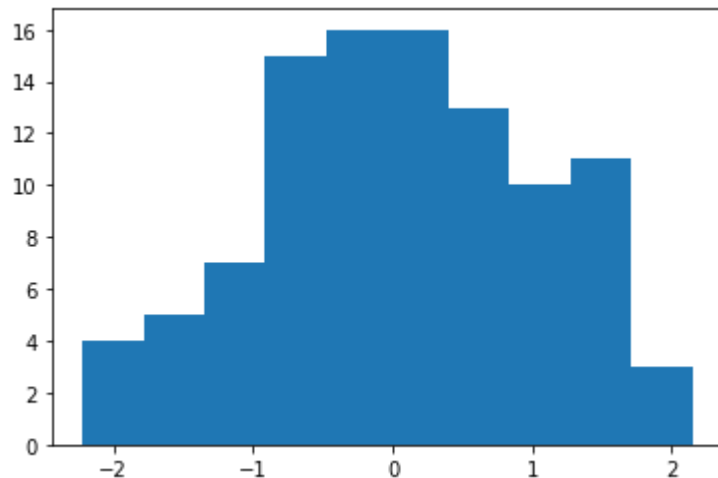
```
Out[39]: 1
```

```
In [40]: data.shape #그냥 벡터니까. 백개의 데이터가 나왔다는 뜻
```

```
Out[40]: (100,)
```

```
In [41]: data = np.random.normal(0,1,100)
print(data)
plt.hist(data,bins=10) #bins 바꾸니. 그래프의 막대 몇 개인지 정함.
plt.show()
```

```
[-0.3919344  0.85328007  1.31792715 -1.14214277  0.45993339  1.19237434
-1.02181608 -0.49631218  1.45856153 -0.30358796 -0.93243261  2.14510453
-2.14206908  0.50933277 -1.32785307 -0.19605045 -1.32569246 -1.42286844
 0.12187997 -0.39704067  0.89102163  0.44603138 -0.44873535  0.62541739
 0.63974312 -0.43001933  1.2323329 -1.16539204  1.28185058  0.81240661
-1.7697323 -0.05408883  1.394039 -0.34560812  1.40819354 -0.81996811
-0.65906678  1.5137701  1.88070512  0.58407675 -0.45690393  0.86449837
-0.31735365 -0.83041907 -2.22251954 -1.51265914 -0.18107539  0.8603055
-0.76774571 -0.6719022  0.53482455  1.30357288  0.02749976  0.09892231
 0.94722325 -0.090717  0.10708013  0.05055964  0.78872663 -0.62910701
-1.73355073  0.27333047  0.11956334  0.14437983  1.64404964 -1.04701272
 1.27026342 -0.03661248  0.01094321 -0.22536437 -1.58808479  0.7175769
 0.30770397 -0.53493031  0.92781105 -0.16238469  0.17518897  0.54869091
 0.0344384 -1.9409634 -2.11601669  1.70699606 -0.6335011 -0.46288451
 1.80403256  0.32196506  0.61978296  1.53970266 -0.06353558 -0.80258417
-0.78887344  0.26148957 -0.71720822 -0.56103709  0.85281175  0.82902079
-0.69590287  0.37296518  1.38004226 -0.6182597 ]
```



2. Manipulation

```
In [43]: X = np.ones([2,3,4]) # 2x3x4개 데이터  
X
```

```
Out[43]: 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.]])
```

```
In [44]: Y = X.reshape(-1,3,2) #같은 차원에서만 변경 가능. 니가 알아서 해라라고 할 때 첫번째에 -1이라고 함.(4,3,2)랑 같은결과  
Y
```

```
Out[44]: 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.]])
```

```
In [ ]: np.allclose(X.reshape(-1, 3, 2), Y)  
#어레이해서 두 개 비교 . assert는 몰라도 됨
```

```
In [49]: a = np.random.randint(0,10,[2,3])  
b = np.random.random([2,3])  
np.savez('test',a,b) # savez() 실제 파일로 저장해줌
```

```
In [53]: del a,b # print all interactive variables 메모리 전체 삭제
%who # show available variables now
```

```
X      Y      data      np      numpy      plt
```

```
In [57]: npzfiles = np.load("test.npz") # 불러오기
npzfiles.files
```

```
Out[57]: ['arr_0', 'arr_1']
```

```
In [58]: npzfiles['arr_0'] # 57행 결과. a 값. 'arr_1' 넣으면 b 값
```

```
Out[58]: array([[6, 6, 8],
               [3, 5, 7]])
```

```
In [4]: data = np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("X", "Y"), 'formats':('f', 'f')})
#파일 불러오기 : np.loadtxt("파일경로",파일에서 사용한 구분자, 데이터타입 지정), data 변수에 array로 넣어준다
data
```

```
Out[4]: array([( 3.3  , 1.7  ), ( 4.4  , 2.76 ), ( 5.5  , 2.09 ), ( 6.71 , 3.19 ),
               ( 6.93 , 1.694), ( 4.168, 1.573), ( 9.779, 3.366), ( 6.182, 2.596),
               ( 7.59 , 2.53 ), ( 2.167, 1.221), ( 7.042, 2.827), (10.791, 3.465),
               ( 5.313, 1.65 ), ( 7.997, 2.904), ( 5.654, 2.42 ), ( 9.27 , 2.94 ),
               ( 3.1  , 1.3  )], dtype=[('X', '<f4'), ('Y', '<f4')])
```

4.Inspecting

```
In [59]: arr = np.random.random([5,2,3])
```

```
In [62]: print(type(arr))
print(len(arr))
print(arr.shape)
print(arr.ndim)
print(arr.size) # 총 elements 개수
print(arr.dtype)
```

```
<class 'numpy.ndarray'>
5
(5, 2, 3)
3
30
float64
```

5.1 Arithmetic

```
In [66]: a = np.arange(1,5)
b = np.arange(9,5,-1)
print(a)
print(b)
```

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

```
In [65]: print(a-b)
print(a*b)
```

```
[-8 -6 -4 -2]
[ 9 16 21 24]
```

5.2 Comparison


```
In [67]: a = np.arange(1,10).reshape(3,3)
b = np.arange(9,0,-1).reshape(3,3)
print(a)
print(b)
```

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

```
In [68]: a == b
```

```
Out[68]: array([[False, False, False],
               [False,  True, False],
               [False, False, False]])
```

```
In [69]: a > b
```

```
Out[69]: array([[False, False, False],
               [False, False,  True],
               [ True,  True,  True]])
```

```
In [70]: a.sum(), np.sum(a)
```

```
Out[70]: (45, 45)
```

```
In [72]: a.sum(axis=0) , np.sum(a,axis=0) # axis = n : n+1번째 차원에서 sum할 것인지. 여기서는 1차원 합이니까 같은 열끼리 합
```

```
Out[72]: (array([12, 15, 18]), array([12, 15, 18]))
```

```
In [73]: a.sum(axis=1) , np.sum(a,axis=1) # 같은 행끼리 합
```

```
Out[73]: (array([ 6, 15, 24]), array([ 6, 15, 24]))
```

Broadcasting

```
In [75]: a = np.arange(1,25).reshape(4,6)
a
```

```
Out[75]: 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]])
```

```
In [76]: a + 100
```

```
Out[76]: 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]])
```

```
In [77]: b = np.arange(6)
b
```

```
Out[77]: array([0, 1, 2, 3, 4, 5])
```

```
In [78]: a + b #행마다 b 더하기
```

```
Out[78]: 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]])
```

Phasor

```
In [1]: 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
```

```
In [2]: # parameter setting
        amp = 1
        sr = 10000
        dur = 0.5
        freq = 100.0
```

```
In [ ]: t = 0.0001 0.0002 0.0003 ... 0.5000 # 우리가 만들고자 하는 타임. (duration. sr이 10000이고, 0.5까지. sr과 dur만 있으면 time 만들)
```

```
In [5]: t = np.arange(1, sr*dur+1)/sr
        t
```

```
Out[5]: array([1.000e-04, 2.000e-04, 3.000e-04, ..., 4.998e-01, 4.999e-01,
              5.000e-01])
```

```
In [ ]: # 매우 중요 generate time
        t = np.arange(1, sr * dur+1)/sr
```

```
In [ ]: # 매우 중요 generate phase
        theta = t * 2*np.pi * freq
```

In [1]:

```
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
```

Phasor

In [2]:

```
# 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
```

In [4]:

```
theta = np.arange(0, 2*np.pi)
theta
```

Out[4]:

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

In [161]:

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

In [162]:

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

In [8]:

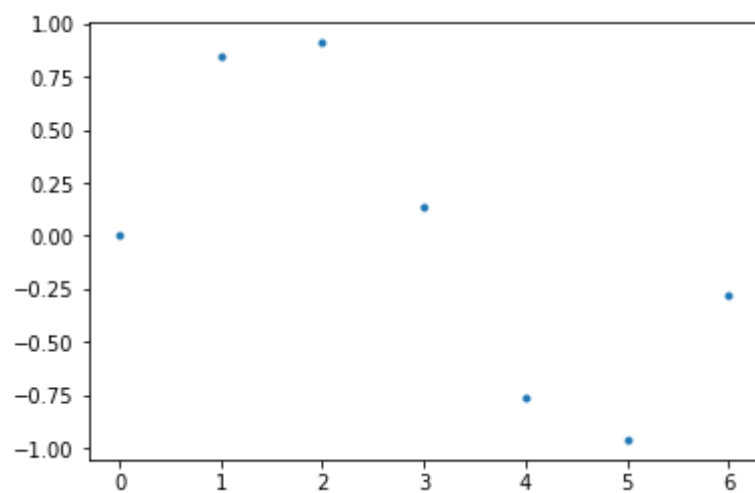
```
# generate signal by cosine-phasor
s = np.sin(theta)
```

In [9]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(theta,s, '.')
```

Out[9]:

[<matplotlib.lines.Line2D at 0x19b9f8310b8>]



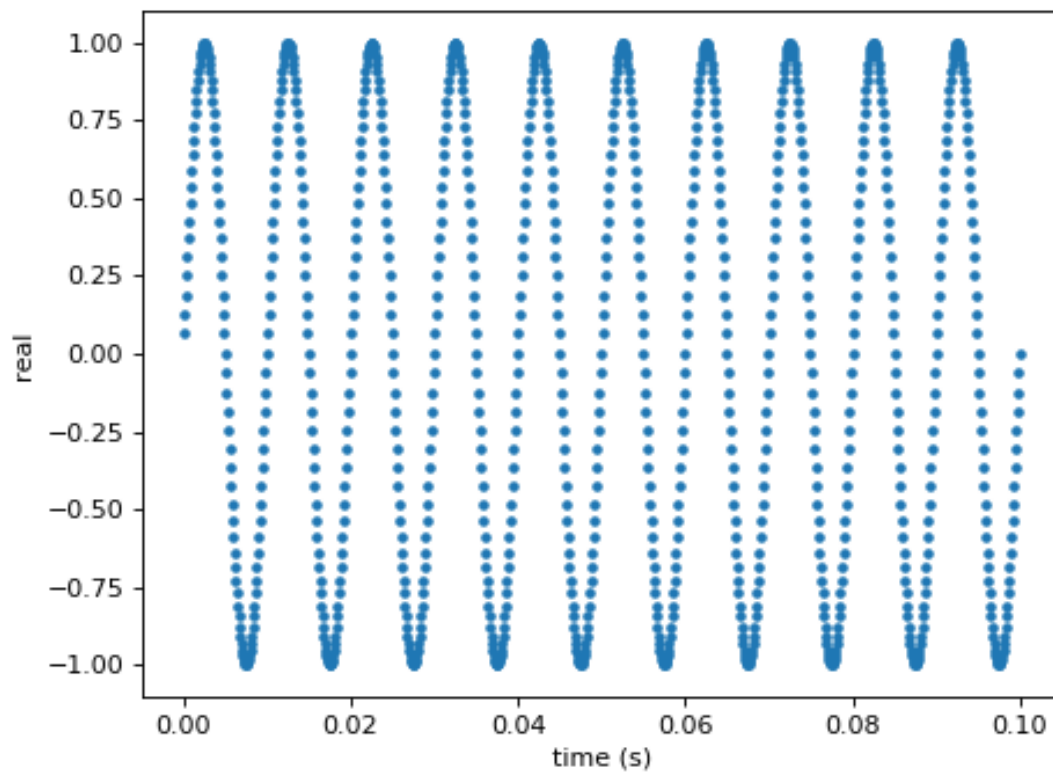
In [164]:

```
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')

<IPython.core.display.Javascript object>



Out[164]:

Text(0, 0.5, 'real')

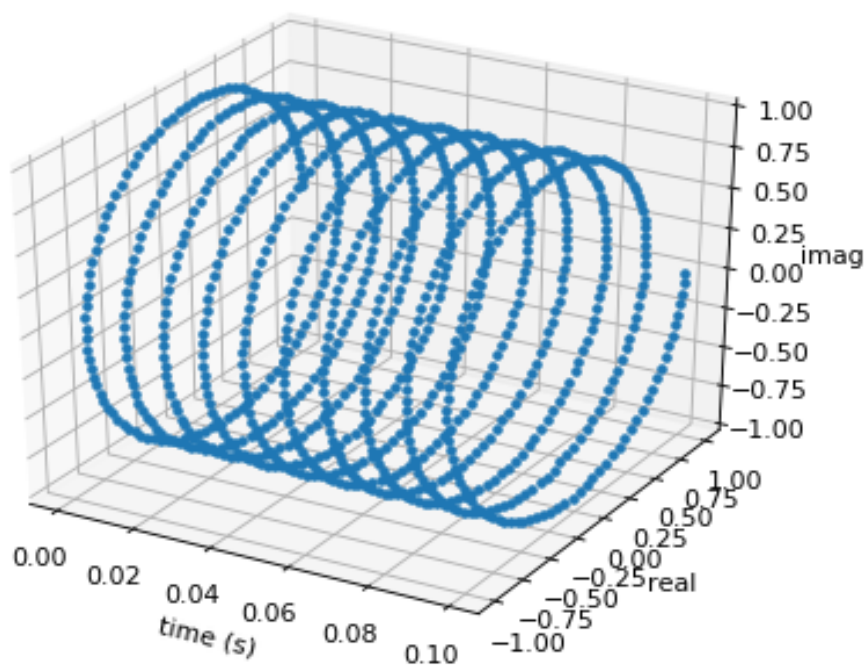
In [72]:

```
# generate signal by complex-phasor  
c = np.exp(theta*1j)
```

In [165]:

```
fig = plt.figure()  
ax = fig.add_subplot(111, projection='3d')  
ax.plot(t[0:1000], c.real[0:1000], c.imag[0:1000], '.')  
ax.set_xlabel('time (s)')  
ax.set_ylabel('real')  
ax.set_zlabel('imag')
```

<IPython.core.display.Javascript object>



Out[165]:

Text(0.5, 0, 'imag')

sine wave만들기

1.시간 만들기

- 세타값만있어도 시간 만들수있음 모양은 만들지만 소리는 안나는... 페이지
- 페이지 인풋값은 각도값. 래디언.

amplitude 정의

sampling rate 얼마나 촘촘하게?

duration 얼마나 길게

frequency 1초 동안에 몇번 왔다갔다?

sr이랑 fr은 같은 유닛을 쓰지만 개념 구분 필요

-

Generate pulse train

1. 가장 최소 sine wave 설정(f0). frequency 잡고 그 뒤에는 배수로 늘려감....

f0 설정 > 싸인웨이브 > 배수로 늘려가기

매우중요

샘플링레이트가 100헤르츠라고 생각. 우리가 표현할 수 있는 숫자의 개수가 100개라는 뜻. 이 백개로 프리퀀시를 1헤르츠 프리퀀시를 표현할 수 있을까요 없을까요 답은 있다. 한번 왔다갔다하면 됨. 한번의 주기가 있으면 됨. 이헤르츠도 됨. 만 헤르츠 가능? 불가능. 우리가 가진 숫자가 더 작아서. 샘플링 레이트가 충분히 있어야 그만큼의 주파수를 표현할 수 있다.

sr= 10Hz, Fr = 100Hz. 최대 5번 사인웨이브 만들 수 있음. 주어진 숫자의 개수(sr)에 우리가 표현할 수 있는 최대 주파수는 그 반밖에 안됨. = Nyquist Frequency. sr의 반 무조건!

시디 음질은 44100Hz. 그러니까 nf는 22050. 사람이 들을 수 있는 주파수 max가 20000. 사람이 들을 수 있는 소리는 다 넣을 수 있는 정도.

=> 배수를 어디까지 올릴 수 있는가? sr의 반까지만!

Fend 제일 마지막

소리 만드는 순서

1. 점점 freq 낮아지도록
2. 산맥 만들기
3. 그 뒤로도 산맥 계속 만들어주기 반복...

=> 소리가 source에서 사람 목소리로 바뀜

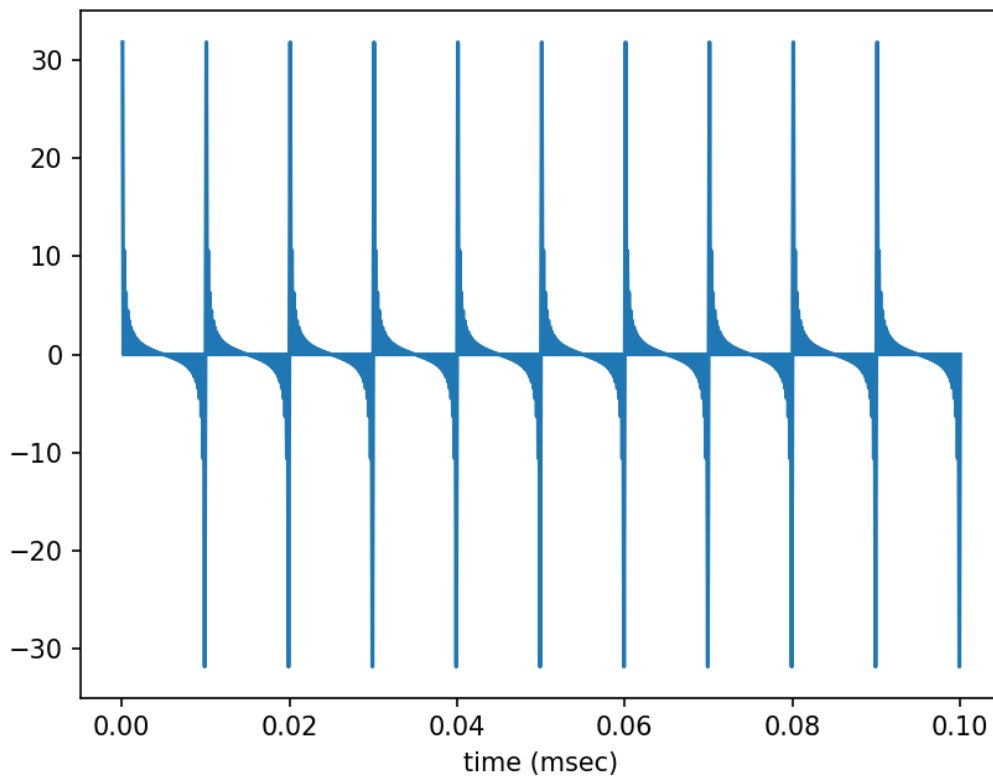
싸인의 라나 코싸인의 라나 똑같음. 소리가 변하지 않음. 싸인과 코싸인은 웨이프는 같지만 살짝 이동한 차이 (90° . $\pi/2$).

얼만큼 이동하든 상관 없음. 1도 2도 이렇게 움직여도 결국 소리는 똑같. 각도를 이야기할 때 페이즈라고 함. 페이즈에 대해 우리 귀는 sensitive하지 않다. 전혀 인식 못함.

In [30]:

```
# generate samples, note conversion to float32 array
F0 = 100; Fend = int(sr/2); # Fend= 배수의 끝. 이게 NF
s = np.zeros(len(t));
for freq in range(F0, Fend+1, F0): # Fend+1은 제일 마지막까지 포함시키기 위해서 더하기 일
    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)
```

<IPython.core.display.Javascript object>



Out[30]:

0:00 / 0:00

In [32]:

```
def hz2w(F, sr): # F = frequency, sr = sampling rate 입력
    NyFreq = sr/2;
    w = F/NyFreq *np.pi;
    return w # 제일 중요. 출력

def resonance (srate, F, BW):#BW = bandwidth
    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
```

In [34]:

```
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) # 산맥의 위치 = RG, 얼마나 산맥이 뾰족한지 완만한지. 뾰족하면 width가 작음
# 키가 똑같은 스펙트럼에서 gradually decrease 하는 모양 만들어야됨 그래서 0이 top이 되는 산맥 생성.
s = lfilter(b, a, s, axis=0)
ipd.Audio(s, rate=sr)
```

Out [34]:

0:00 / 0:00

In [35]:

```
RG = 500 # RG is the frequency of the Glottal Resonator. first formant 생성
BWG = 60 # BWG is the bandwidth of the Glottal Resonator. 앞에서 만든(BWG=100)보다 뾰족한 산맥
a, b=resonance(sr, RG, BWG)
s = lfilter(b, a, s, axis=0) #s = signal
ipd.Audio(s, rate=sr)
```

Out [35]:

0:00 / 0:00

In [36]:

```
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)
```

Out [36]:

0:00 / 0:00

In [37]:

```
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)
```

Out[37]:

0:00 / 0:00

In [38]:

```
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)
```

Out[38]:

0:00 / 0:00

In [39]:

```
s = lfilter(np.array([1, -1]), np.array([1]), s) #뭐하는지만 알면 됨. 입술이 없다면 윗라인에서 끝.
ipd.Audio(s, rate=sr)
```

Out[39]:

0:00 / 0:00

load wav

In [132]:

```
from scipy.io import wavfile
# sr, s = wavfile.read('a.wav')
nSamp = len(s)
dur = nSamp / sr
t = np.linspace(1/sr, dur, nSamp)
```

Fourier tranform