

세 가지: larynx, velum에서 비음, constrictor

constrictor: lips tongue tip, tongue body

specify되면 특정 소리 말할 수 있음

ex)/p/ lips cl: dilavelor cd: stop, velum: raised, larynx: open(voiceless)

/d/ tongue tip, cl: alveolar cd: stop, velum: raised, larynx: closed

/z/ tongue tip cl: alveolar cd: fricative, velum: raised, larynx: closed

/n/ tongue tip cl: alveolar cd: stop, velum: lower, larynx: closed

(velum: lower은 mn응밖에 없음)

Praat 사용법

10/1

코딩은 자동화라고 생각하자. 그런데 왜 자동화할까? 사인웨이브의 반복이라.

세상의 언어는 단어와 문법으로 이루어진다.

컴퓨터 언어도 이와 비슷하다.

컴퓨터 언어: 변수에 정보 지정. 조건if 컨디션, 반복for 루프, 함수 필요. 함수가 가장중요

정보의 종류는 숫자와, 문자

Jupyter-variable

a=1이라고 했을 때, 오른쪽에 있는게 정보이고 왼쪽에 있는 것이 변수.

함수() print(a) 일 때 ()가 입력이다.

셀 선택하고 b 누르면 below에 셀 생성, a누르면 above에 생성, x누르면 셀 제거

, 실행은 shift+enter

문자를 정보에 입력하려 할 때 무조건 "를 붙여줘야 함

변수명만 쳐도(print안해도) 결과값을 보여준다.

;입력하면 줄바꿈안해도 한줄에 쓸 수 있음.

한 변수에 여러개의 정보를 넣는게 list []대괄호로 표현,

type변수 입력하면 list(여러개), int(숫자), float(실수), str(문자) 등 출력

dict를 표현하려면 중괄호{} 안에 표제어:설명어 의 페어 형식을 가진다.

Q a=[1, 'love', [1, 'bye']] 일 때, a에는 세 개의 정보가 들어있고, type(a)를 입력하면 list를 출력한다. 이 안에는 int, str, list가 들어있고, 세 번째 list 안에는 int와 float이 들어있다.

()는 tuple

10/8

```
a=[1, 2]
```

```
b=[3, 4]
```

```
c=a[0]+b[0]
```

c를 하면 4가 출력됨. []는 0번째 값을 retrieve하는 역할을 함.

같은 원리로

```
a=[1,2]
```

```
b=[3, 4]
```

```
c=a[1]+b[1]
```

c를 하면 6이 출력됨.

partial value 값을 가져올 때는 대괄호 [] 사용.

float는 variable이 들어오면 float type으로 바뀌춤.

```
a의 type이 float인데 int로 강제하면 int로 출력 a = 1.2; a = int(a); print(type(a))
```

어떠한 내부정보로 들어갈 땐 []안에 인덱스를 적으면 내부정보를 가져온다.

```
a = '123'; a = list(a); print(type(a)); print(a); print(a[2]) 실행하면
```

```
<class 'list'>
```

```
['1', '2', '3']
```

a는 1, 2, 3이라는 문자를 가진 list가 된 거임. 리스트되고 a의 두 번째 값인 3이 문자로서 나온 것. 이 부분 이해.

```
a = [1,'2', [3, '4']]; print(type(a)); print(a[0]); print(a[1]); print(a[2])
```

```
<class 'list'>
```

```
1
```

```
2
```

```
[3, '4']
```

여기서 1은 숫자, 2는 문자로 출력된 것.

dict에서 access할 때는 pair에서 앞부분을 index로 씀.

list처럼 숫자를 인덱스로 쓰지 않음.

```
a = {"a": "apple", "b": "orange", "c": 2014} ;print(type(a)); print(a["a"])
```

```
<class 'dict'>
```

```
apple
```

여기서 print(a["a"])는 a 안의 "a"인덱스 값을 가져오는 것.

```
그래서 a = {"1": "apple", "b": "orange", "c": 2014}; print(type(a)); print(a["1"])
```

해도 같은 값 나옴.

```
a=[(1,2,3), (3,8,0)]; print(type(a)); a
```

```
<class 'list'>
```

```
[(1, 2, 3), (3, 8, 0)]
```

```
type(a[0])
```

```
tuple
```

string

```
s = 'abcdef'; print(s[0], s[5], s[-1], s[-6])
```

```
a f f a
```

인덱스가 음수일때도 확인.

Q. 첫 번째 인덱스는 항상 0이고 마지막 인덱스는 -1인 것.

```
s = 'abcdef'; print(s[1:3], s[1:], s[:3], s[:])
```

```
bc bcdef abc abcdef
```

인덱스가 1:3이면 1위치에서 3위치 전까지, 1:이면 1위치에서 끝까지.

:3이면 처음부터 2위치까지, :이면 전체

len()은 정보의 길이를 준다고 생각.

```
s = 'abcdef'; len(s)
```

```
6
```

```
s = 'abcdef'; s.upper()
```

```
'ABCDEF'
```

```
s = ' this is a house built this year.\n'; result = s.find('house'); result
```

```
11
```

띄어쓰기 포함 11번째에서 house가 시작된다는 의미.

```
result = s.find('this'); result; result = s.rindex('this'); result;
```

```
1
```

```
23
```

```
s = s.strip(); s
```

```
'this is a house built this year.'
```

space나 불필요한 기호를 제거해주는 함수

```
tokens = s.split(' '); tokens
```

```
['this', 'is', 'a', 'house', 'built', 'this', 'year.']
```

은 split 괄호 내부에 있는 것으로 자르라는 의미.

```
s = ' '.join(tokens); s
```

```
'this is a house built this year.'
```

은 다시 붙이는 것

```
s = s.replace('this', 'that'); s
```

```
'that is a house built that year.'
```

은 this를 that으로 바꿔주는 것.

10/10

variable assign은 =해서 하는 것. string하고 list하고 비슷한 성격을 가진다. variable 종류.

function에 대해 배웠다. 오늘은 for loop와 if조건에 대해 배울 것.

Jupyter에 노트남기는 법: #쓰고 쓰면 명령어 실행 안됨. 셀종류를 markdown으로 바꾸면 실행x.

syntax

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

```
for i in a:
```

```
print(i)
```

```
1
```

```
2
```

```
3
```

```
4
```

in 뒤에 있는 것을 하나씩 돌려서 한 번할 때마다 i로 받아서 뭔가를 하라

range뒤에 어떤 함수가 나오면 list를 만들어줌. 인덱스를 0부터 만들어주는것

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

```
for i in range(4):
```

```
print(a[i])
```

```
1
```

```
2
```

```
3
```

```
4
```

```
print (i)
```

```
0
```

```
1
```

```
2
```

```
3
```

range(4): 0부터 4개 (0 1 2 3)

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

```
for i in range(len(a)):
```

```
    print(a[i])
```

```
1
```

```
2
```

```
3
```

```
4
```

첫 번째 루프에서 i는 0

```
len(a)=1
```

```
len(a)=4
```

```
a = ['red', 'green', 'blue', 'purple']
```

```
for i in a:
```

```
    print(i)
```

랑

```
a = ['red', 'green', 'blue', 'purple']
```

```
print (a[0])
```

```
print (a[1])
```

```
print (a[3])
```

```
print (a[4])
```

의 결괏값은

red

green

blue

purple로 같다

```
a=[red, green, blue, purple]
```

```
for s in a:
```

```
    print (s)
```

하면 네 번 루프를 돌고 s는 매번 달라진다.

```
a = ['red', 'green', 'blue', 'purple']
```

```
for s in range(len(a)):
```

```
    print(s)
```

하면

0

1

2

3

```
a = ['red', 'green', 'blue', 'purple']
```

```
for s in range(len(a)):
```

```
    print(a[s])
```

enumerate는 번호를 매기는 것.

```
a = ["red", "green", "blue", "purple"]
```

```
b = [0.2, 0.3, 0.1, 0.4]
```

```
for i, s in enumerate(a):
```

첫 번째 루프에서 i에는 0, s에는 red가 들어감. for뒤에서 앞(i)에는 번호, 뒤(s)에는 element.

```
a = ["red", "green", "blue", "purple"]
```

```
b = [0.2, 0.3, 0.1, 0.4]
```

```
for i, s in enumerate(a):
```

```
    print("{}: {}".format(s, b[i]*100))
```

```
red: 20.0%
```

```
green: 30.0%
```

blue: 10.0%

purple: 40.0%

i=0 s=red이런식으로 네 번 돌아간다. format 함수 안 두 개가 {}에 각각 들어간다.

```
a = ["red", "green", "blue", "purple"]
```

```
b = [0.2, 0.3, 0.1, 0.4]
```

```
for s, i in zip(a, b):
```

```
    print("{}: {}%".format(s, i*100))
```

a와 b의 길이는 같아야. a와b를 페어로 만들어줌

```
a = 0
```

```
if a == 0:
```

```
    print("yay")
```

```
yay
```

a가 0이면 yay를 print해라

```
a = 0
```

```
if a != 0:
```

```
    print("yay")
```

a가 0이 아니면 yay를 print해라. 그래서 프린트 안됨.

```
print("no")
```

```
a = 0
```

```
if a != 0:
```

```
    print("yay")
```

```
else:
```

```
    print("no")
```

```
no
```

요부분 유튜브 볼 것

```
for i in range(1,3):
```

```
    for j in range(3,5):
```

```
        print(i*j)
```


3

4

6

8

두 번씩, 1일 때 두 번째 줄 3,4 돌고 2일 때 3,4해서 총 네 번 돈다.

```
for i in range(1,3):
```

```
for j in range(3,5):
```

```
if j >=4:
```

```
print(i*j)
```

4

8

이때 indent가 없으면 다 에러뜨다!

Q. 특정 콤비네이션이 있을 때 없을수 있음. 그런데 뭘까? 존재한다 안한다로 답.

Q. lips 가 cl 로 velar를 가질 수 있을까?이거는 영어에서 못하는걸까 사람이 못하는걸까?

사람이 못하는 거임

Q. lips가 cl로 alveolar로 가지는 언어가 있을까?

이론상으로 가능. 인체 구조상 가능.

기말

숫자화되지 않은 데이터들이 어떻게 숫자화되는가. 숫자열로 표현되는가.

이미지: 모든데이터는 벡터의 상태일 때 다루기 쉽고 벡터화된다.

소리: 소리도 벡터화 된다.

텍스트: 5000번째 단어라고 했을 때 0000....1(5000개) 이런식으로 표현. 역시 벡터화

모든 벡터화된 데이터는 수학적 처리를 해야하는데, 그냥 더하기로 처리하면 안된다.

이걸 해주는게 numpy.

```
import numpy
```

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

```
B = numpy.array(b)
```

array는 list를 계산가능하게 만들어주는 것. import 반드시 해줘야.

```
A+B
```

```
array([ 3, 7, 11])
```

```
type(A)
```

numpy.ndarray

```
import numpy as np
```

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

```
B = np.array(b)
```

as np해주면 이렇게 바꿔 쓸 수 있다.

numpy라는 패키지 안에는 여러 패키지가 포함될 수 있다. import numpy하면 numpy안에 있는 모든 것들을 쓸 수 있다. A안에 D안에 f가 있다고 하자. 부르려면 numpy.A.D.f라고 하면 됨.

from numpy import A를 하면 그다음부터는 A.D.f로 불러올 수 있다.

Q. 불러올수 있는 다양한 방법 중 틀린 것은? import numpy as np

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

이렇게 하면 plt만 해도 불러올 수 있음.

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

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

```
array([[8, 0, 0],
```

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

숫자는 랜덤. [행, 열]이 만들어짐.

```
np.zeros([2,3])
```

```
array([[0., 0., 0.],
```

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

zeros, ones 함수는 기본적으로 dtype이 float인 array를 만든다.

```
np.array([[0,0,0],[0,0,0]])
```

```
array([[0., 0., 0.],
```

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

array는 리스트를 계산할 수 있게 만들어줌.

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

```
X
```

```
array([[1, 2, 3],
```

```
[4, 5, 6]])
```

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

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

dtype=뭐라고 하나에 따라 무슨 숫자가 나올지 정할 수 있다.

float다음 숫자는 몇 번째 소수점까지 표시할 것인가. 얼마나 정확하냐와 연관되어있음.

arange(0,10,2)는 0부터 10개 2씩 뛰면서

```
np.arange(5)
```

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

인덱스 0부터 다섯 개를 만든 것.

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

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

이렇게 하면 0부터 10전까지 2간격으로 만들

```
np.linspace(0,10,6, dtype=float)
```

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

0부터 10까지 6개로 나눠주는 것(끝 포함). 그 간격이 다 같음.

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

```
array([[1, 2],
```

```
[3, 4],
```

```
[5, 6]])
```

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

```
array([[[1, 2],
```

```
[3, 4],
```

```
[5, 6]],
```

```
[[1, 2],
```

```
[3, 4],
```

```
[5, 6]])
```

[]대괄호가 두 개 나오면 이차원 세 개 나오면 삼차원

X.ndim하면 차원확인 가능

```
X.shape
```

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

제일 큰 괄호 속 2개 두 번째 괄호 속 2개 제일 작은 괄호 속 2개

```
X.astype(np.float64)
```

```
array([[1., 2., 3.],
```

```
[4., 5., 6.]])
```

type바꿔주는 것.

```
np.zeros_like(X)
```

```
array([[0, 0, 0],
```

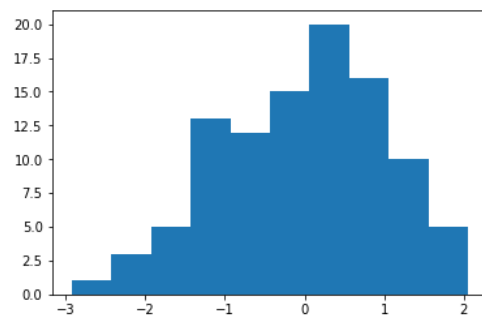
```
[0, 0, 0]])
```

다 0으로 바꿔줌. X*0해도 됨

```
data = np.random.normal(0,1, 100)
```

```
print(data)
```

```
plt.hist(data, bins=10)
```



```
plt.show()
```

100개의 랜덤한 데이터를 정규분포로 만들어라. 무조건 0을 포함한 자연수값.

Q. 요 블록 다 합하면 몇 개일까? 100개

여기서 data.ndim하면 1(1차원이라는 의미) data.shape하면 100 (100개의 벡터라는 의미)

```
X = np.ones([2, 3, 4])
```

```
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.]])
```

```
Y = X.reshape(-1, 3, 2)
```

```
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.]]])
```

x의 엘리먼트는 $2*3*4=24$ 개로 동일하다. 무슨값인지 모르면 -1적으면 되고 4적어도됨.

동일한 차원으로만 reshape가능

```
np.allclose(X.reshape(-1, 3, 2), Y)
```

```
True
```

X랑 Y랑 같냐

```
a = np.random.randint(0, 10, [2, 3])
```

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

```
np.savez("test", a, b)
```

0과 10사이에서 숫자를 픽해서 [2,3]모양을 만들어라.

파일로 저장해주는 것. test.npz 생겨났을 것.

```
ls -al test*
```

있는지 확인

```
del a, b
```

who

뭐가 지금 available한지. 근데 delete해서 없을 것.

```
npzfiles = np.load("test.npz")
```

```
npzfiles.files
```

```
['arr_0', 'arr_1']
```

```
npzfiles['arr_0']
```

```
array([[1, 5, 2],
```

```
[1, 7, 0]])
```

```
data = np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("X", "Y"),  
'formats':('f', 'f')})
```

```
data
```

csv는 콤마로 분리된 variable. csv로부터 data로 들어온 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, 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]]
```

```
a == b
```

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

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

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

```
a > b
```

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

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

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

```
a.sum(), np.sum(a)
```

```
(45, 45)
```

요부분 이해.

```
a.sum(axis=0), np.sum(a, axis=0)
```

```
(array([12, 15, 18]), array([12, 15, 18]))
```

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

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

몇 번째 차원에서 sum을 할건가. 첫 번째 차원에서 sum을 해라.

11/5

Phoasor

sinusoidal을 만들어내는게 phasor. sin의 입력값은 radians(2파이 등). degree(0도-360도)가 아니라.

쉽게 말해 sin, cos이 phasor.

오일러 공식: $e^{i\theta} = \cos(\theta) + i\sin(\theta)$

cos가 real, sin imaginary

그냥 e가 숫자값이구나..만 알면됨. $f(\theta) = e^{i\theta}$. e만 변하면 값이 나오는 것.

오일러 공식이 새로운 phasor.

θ	0,	$\pi/2$	π	$3\pi/2$	2π
$\cos\theta$	1	i	-1	-i	1
(a,b)	(1,0)	(0,1)	(-1,0)	(0,-1)	(1,0)

복소수를 plot하는 방법을 배울 것.

복소 평면. x축 a, y축 b이라고 하고 θ 값을 주욱 이어보면 원을 그리게 됨.

x축과 원점-(a,b) 연결한 선이 이루는 각이 θ

```
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
```

Q. 첫줄은 import matplotlib.pyplot와 같다.

1. 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
```

sr, freq 둘다 Hz가 단위. 1초에 몇 개이니까.

음질의 정도가 sr. 1초동안 들어갈 수 있는 time tics. $t=0.001\ 0.002\ 0.003\ \dots$ 일 때 $sr=1000$

freq는 1초에 태극문양이 몇 개 들어가느냐.

나중에 변수값 바뀔 때 애네만 바꿔주면 돼서 편리.

2. generate time

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

+1은 제일 마지막까 더해주는 것. $sr \cdot dur$ 은 time tic을 dur 까지 켜.
여기서는 $1/10000$ 부터 $5000/10000$ 까지인 것.

왜 시간을 만들어주어야 할까. phasor 함수들은 받아들이는 입력이 radian만이기 때문.

이것만으로는 소리의 실체를 만들 수 없음.

3. generate phase

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

들어가는 각도값이 phase. 총 100바퀴를 돌아라.
t 연동시키는 것

4. generate signal by cosine-phasor

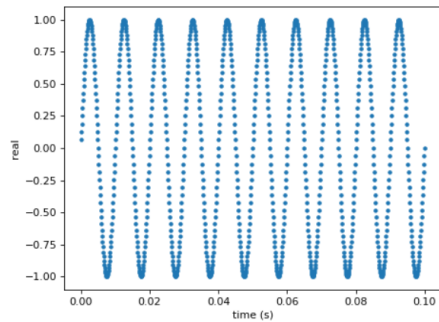
```
s = np.sin(theta)
```

Q. time과 theta의 벡터사이즈는 같다.

5. 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')
```



실제 소리를 만들 때는 t 가 필요해서 x축으로. θ 는 그냥 한바퀴 반복, 뻥함.

Q. 점들의 개수는 몇 개인가 1000개.

t 랑 s 랑 같아야 함. 다르면 왜 실행이 안되는지 알아야.

시간이 없다고 가정하면,

$\theta = \text{np.arange}(0, 2*\text{np.pi},)$ 이때 θ 의 단위는 radian

$s = \text{np.sin}(\theta)$

총 7개의 값이 모두 corresponding하다.

$\theta = \text{np.arange}(0, 2*\text{np.pi}, 0.1)$ 하면 열배로 촘촘하게 값을 그리게 됨.

시간이 없음.

subplot은 화면 분리해서 여러개 들어갈 수 있는 것.

111은 1x1의 첫 번째 221은 2x2의 첫 번째 222, 223, 224도 가능.

non-linear하다

41분 유튜브 확인할것

6. generate signal by complex-phasor

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

c 의 표기법이 다 같은 이유는 형식($a+bi$)을 통일시키는 것. -01 은 소수점 하나를 왼쪽으로 옮긴다는 의미

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

```
ax = fig.add_subplot(111, projection='3d')
```

```
ax.plot(t[0:1000], c.real[0:1000], c.imag[0:1000], '.')
```

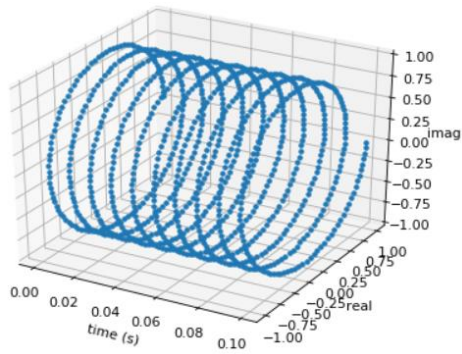
```
ax.set_xlabel('time (s)')
```

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

c라는 하나의 복소수 값에는 a,b값도 포함되어있다.

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

s대신 c.imag(=s), c.real을 넣어도 소리 나옴.



11/12

```
!pip install sounddevice
```

```
import sounddevice as sd
```

```
sd.play(c.real, sr)
```

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

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

지금껏 amp이 1이 곱해져있던 것.

Generate pulse train

sr이 100Hz일 때, 우리가 표현할 수 있는 게 1초에 100개라는 의미. 이걸로 1Hz freq를 표현할 수 있을 가? 있다. 한번의 사인웨이브로 표현가능하다. 10000Hz freq는 불가. sr이 1초에 충분히 있어야 그만큼의 주파수를 표현할 수 있다. $sr=10Hz / freq=100Hz$ 가능? 불가. 10Hz로는 5개밖에 못만들. sr의 절반이하만 freq로 가능. 이를 **Nyquist freq**라고 함. 표현할 수 있는 freq의 맥시멈. sr의 무조건 반. cd의 sr은 44100Hz이고 nyquist freq = 22050Hz. 그러면 왜 22050? 인간의 가청주파수가 20000이 최대. 들을 수 있는 소리는 다 표현 가능. 그래서 CD음질을 저 숫자로 고정. 사람의 말소리는 4000 내에서 다 표현 가능해서 유선전화기의 sr= 8000Hz 요즘은 16000Hz.

```
# 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)
```

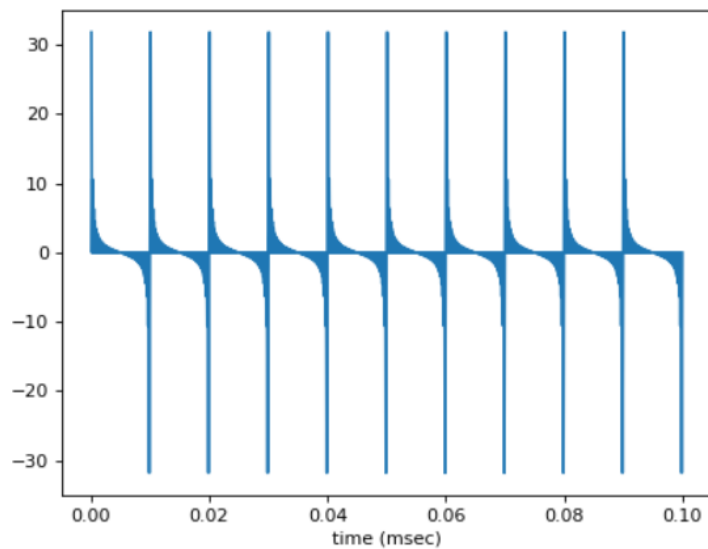
```
    s = s + tmp
```

```
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)
```

Fend = int(sr/2) freq의 마지막을 Fend라고 하고, sr/2=Nyquist freq, 소수나오면 지저분해서 int

s = np.zeros(len(t)) 처음의 s 값.

for freq in range(F0, Fend+1, F0): f0부터 Fend+1(마지막까지 포함)까지 F0의 간격으로.

11/14

freq=440으로 설정해보자

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

```
s = amp*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')
```

소리나게 해주는거 `ipd.Audio(s, rate=sr)`

여기서 소리의 음계는 A(라)이다. `freq=880, 220`으로 하면 다 라이다. 옥타브를 뛰려면 배수로.

```
s = amp*np.cos(theta)
```

`sin`을 `cos`로 바꾸면 시작점이 달라진다. `sin`과 `cos`사이에는 $\pi/2$ 차이 크기가 있구나. 하지만 소리가 변하진 않는다. `phase(각도)`를 얼마나 옮기느냐에 대해서는 인간이 감지하지 못한다. `freq`만 인지가능.

`c`자체로는 plotting이 안됨. 이를 plotting하기 위해서는 `a,b`를 가져와야. `real, imag`값을 파트를 가져와서 2차원에서 찍음. 찍는 방법으로 plotting을 한다.

`ipd.Audio(c.real, rate=sr)`로 재생. 위에서의 `sin`값은 `c.imag`랑 같다.

지금부터 **carving**. pulse train은 wave form이지 spectrum이 아니니까 혼동하지 말 것.

```
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
```

여기는 알면 좋지만 몰라도 됨. 함수가 만들어지는 실체임.

`F,sr`이 입력이고 `return`이 출력. `srate, F, BW`가 나오고 `a,b`가 나옴

어떻게 쓰는지만 알면 됨.

`resonance`가 위에 `hz2w`를 불러와서 쓴다.

```
RG=0, BWG=100
```

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

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

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

RG는 x축 산맥의 위치 BWG는 산맥이 얼마나 뚱뚱한지 뽕족한지.

0위치에 100만큼 뚜꺼운 산맥을 만들어라.

고주파쪽만 남고 나머지는 다 carve된걸 praat에서 확인가능

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)
```

RG를 키우면 조금 더 사람 목소리에 가까워짐

.

.

.

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)
```

다 실행하면 500,1500,2500,3500에 산맥 만들어진거 Praat에서 확인 가능

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

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

요거는 입술 통해서 공명한 소리.