190905

praat
헤르츠: 1초를 44100개로 쪼개서(44100분의 1초)
숙제: 녹음하고 저장하기 WAV 파일

show pitch: 파란선 나옴, 목소리 높낮이, 여자가 높음
intensity: 세기, 노란선 나옴(폭이 넓으면 값이 높고 폭이 좁으면 값이 낮음)

wave 밑에 흑백: 소리의 spectrum(높은게 고주파, 낮은게 저주파)

190917

English consonants & vowels
철자와 소리는 다름-g/a/p

y/j/는 자음-> year라고 쓰면 자음으로 소리가 시작됨(모음으로 시작하는 ear에 자음 y가 붙은 것)

voiced/voiceless

모음: monophthongs(단모음) / diphthongs(복모음)

phonology: 소리가 어떻게 grouping되는지, 머릿속에서 일어나는 일, 머릿속의 인지 과정, study on sound 'system'
phonetics: 더 물리학적, 늘 차이가 생김, physical(가를 똑같이 열 번 얘기한다고 머릿속으로는 생각하지만, 실제로는 미묘한 차이가 생겨남), study on speech
speech: 사람이 하는 모든 말

'아'에서 '이'로 소리가 바뀌는 이유? 입모양 때문(입모양 속에 혀의 위치, 턱 등 포함-주된 요인은 아님)

한국어는 음절이 반복, 영어는 stress가 반복
한국어는 턱을 많이 쓰는 언어, 영어는 혀를 많이 쓰는 언어

articulatory: 말의 시작, 소리를 만들어내는 사람의 원리
acoustic: 공기와 소리가 어떻게 조화되는가, 사람과 관계없이 완전 물리적 특성
auditory: 어떻게 듣는가, 사람이 수반된 매커니즘, 고막이 움직이는 것 고려(물리학)

성대(larynx): 후두

인강(pharynx): 목젖부터 후두까지 긴 관
구개(palate): hard / soft(velum)
alveolar: 아주 중요한 부분
목젖(uvula)

upper structure는 그대로 있고 lower structure가 움직임(e.g. lip, tongue tip, tongue, epiglottis- 기도로 가는 길을 막음)
말할 때 식도는 이용하지 않고 기도를 이용

입말고 코로 가는 tract도 있음

비음을 뺀 모든 자음(nasal)은 velum이 raised된 상태에서 소리가 남(nasal은 lowered)
코로 숨을 쉴 때: velum은 lowered된 상태

larynx: 막히면 진동, 유성음/무성음 나눔

phonation process in larynx

articulation: 성대에서 유성/무성 구분, velum에서 lowered/raised되는 것

190919

유튜브에 남호성 교수님 강의 올라옴

조음, 음향(acoustic): 말하는게 어떻게 공기를 타고 가는가, 청각(auditory): 우리 뒤로 들어가는 과정

조음: 크게 세 가지 요소(ppt에 세 동그라미)
제일 중요한 부분: 혀를 중심으로

숨을 쉴 때: velum은 내려옴-> 코로 가는 길이 생김
velum이 lowered 되면 m. n, ng 등 소리가 남

lip, tongue tip, tounge body 모두 constrictor(협착을 만드는 주체)
constriction location: 앞뒤 조절
constriction degree: 상하 조절
constriction location과 constriction degree에 의해 자세히 조정됨

lip이 앞으로 가면 b, 뒤로 오면 v
tongue body가 앞으로 오면 j

tongue tip: 윗니(th), alveolar(d, t, n), ...
constriction location 관점에서 lip, tongue body는 2개, tongue tip은 네 개 정도로 미세 조정됨

constriction degree: stops, fricatives, approximants(영어에서 네 개- r, l, w, j), vowels

!!시험
velum raised, glottis(larynx의 틈) open, constrictor tongue tip, CL alveolar, CD stop 소리는? /t/
모든 모음은 constrictor로서 tongue body만 씀(tongue tip과 lips는 안 씀)
모음과 같은 constrictor를 쓰는 자음의 예시 중 velum lowered? /ng/- glottis closed(진동 일어남-유성음)

phoneme: 개별적인 소리
모음과 자음 specify하는거 시험

소리를 measure하는 방법
duration: 길이, 자음은 짧고 모음은 길
pitch: 파란 곡선, 높다가 낮아지는 그래프(ppt), 소리의 높낮이(Hz), 여성의 목소리인지 남성의 목소리인지에 따라 setting-range를 다르게 해줘야함
intensity: 소리가 큰지 작은지 (같은 도라도 약하거나 세게 말할 수 있음)(dB)
spectrum: 소리가 1차원적인 것이 아님, frequency 관점에서 분석

spectrogram: 프리즘으로 빛을 분산시키듯이 frequency 관점에서 분석하는 것
까만 띠가 보임- formant라고 함
formant: 흑백의 띠, 모음이 뭔지 결정, 모음을 구별하는 수치적인 지표- formant 값에 따라 이 모음은 뭐라고 말할 수 있음, 줄여서 f1, f2 이런 식으로 말함

190924

어떻게 자음과 모음을 발성하는가?
세 가지 중요한 과정- larynx, velum, 입
p: lips-> location- bilabial, degree- stop, velum raised, larynx open
b: p에서 larynx만 closed로 바뀌면 됨
d: tongue tip-> location- alveolar, degree- stop
z: tongue tip-> location- alveolar, degree- fricative
n: tongue tip-> location- alveolar, degree- stop, velum lowered, larynx closed

praat 시험에 나올 수 있음

Hz: measure의 단위, 1초 동안 크게 반복되는 것이 몇 번 나오는가 (the number of occurrences of a repeating event per second)
1초에 sine wave가 몇 번 나오느냐에 따라 주파수를 얘기할 수 있음
frequency: 1초에 몇 번 반복되는지
magnitude: sine wave의 크기
vocal fold의 vibration에 의해 repeat이 일어남

sine wave를 결정짓는 것은 frequency와 magnitude(얼마나 소리가 큰지)
이 세상에 존재하는 모든 signal(sound 포함)은 여러 다르게 생긴 sine wave의 결합으로 표현됨
모든 신호는 조금씩 다른 sine wave의 합

파란 그래프: frequency가 상대적으로 높음(첫 번째 빨간 것보다 3배 빠름)
frequency가 작음-> slow하다는 것, 저음
magnitude는 두 번째 초록색이 제일 작고, 첫 번째 빨간색이 제일 큼
여러 sine wave의 합은 sine wave가 아닌 복잡한 소리로 만들어짐
복잡한 소리, 혹은 신호는 다양한 sine wave의 합으로 표현됨
합(complex tone): 1초에 100번 반복됨(빨간 wave와 똑같은 반복되는 주기)
sine wave에서 x축: 시간, y축: 단순한 숫자값, value(voltage)
time-value의 그래프를 frequency-amplitude 그래프(spectrum)로 변환시킬 줄 알아야함

spectrum이 어떻게 이루어져있는가? 우리 주변에서 보는 소리는 복잡한 형태
spectrogram- spectrum을 time으로 visualize함(spectrum은 시간개념이 없음, 한 given point에서 어떤 frequency 성분이 많은가를 보는 것), spectrum을 시간축으로 늘려놓은 것

아 라고 녹음하고 spectrum slice를 확인했더니 130Hz 간격으로 반복됨- 등간격 (아까 pure tone을 만들었을 때는 440hz하나만 있었는데, 사람 목소리는 complex tone이기 때문에 여러 개가 생김)
맨 처음 나오는 130Hz는 목소리의 pitch와 일치- 제일 낮은 주파수의 것과 일치(ppt- 100Hz의 반복 패턴이 합성한 그래프에서 나타나는 것처럼)
-> 사람의 음의 높낮이는 어떤 frequency와 일치하는가? 제일 작은 pure tone(sine wave)의 진동수와 일치(위에서는 130Hz)
-> 아 라고 얘기를 할 때, 여러 다른 simplex tone의 합으로 녹음됨-> 제일 slow한 simplex tone의 frequency가 우리 목소리의 pitch(음높이)와 동일-> 성대에서 vocal folds가 1초에 몇 번 떨리는지와 일치

진동수- frequency, 1초에 몇 번 반복되는지, 단위는 Hz

source(성대에서 나는 소리)에서 filter(tube, 입모양 등)가 어떻게 바뀌는지에 따라 ㅏ 소리도, ㅣ 소리도 만들 수 있음

graph가 decreasing- 모든 사람의 source의 패턴
제일 처음에 나오는 F0(fundamental frequency, pitch, the number of vocal fold's vibration in a second)이 이후 곱하기되어서 나옴(115->230->...)- harmonics(배음- 배를 이룸)
speech의 source는 sine wave의 합처럼 저렇게 생김(ppt에서 gradually decreasing하는 그래프), 점점 amplitude가 작아짐
모든 사람의 source는 똑같은 패턴이지만, 거리는 다를 수 있음- 여자면 첫 시작이 남자보다 높을 것-> graph 모양이 더 듬성듬성인 모양일 것 (10000Hz에서 자른다고 생각했을 때, 남자가 갖는 배음의 숫자가 여자보다 많음- 시험)

head 소리의 graph- 배음의 구조가 안 깨짐(115의 배수대로 가는 건 그대로 유지)
amplitude가 깨짐- smoothly decreased가 아닌 제멋대로 왔다갔다

spectrogram은 wave와 쌍을 이룸
x축은 wave와 마찬가지로 time이지만 y축은 frequency
밑은 까맣고 위로 갈수록 옅어짐, 진한 부분이 amplitude가 큼
low frequency 쪽에서 energy가 크고, high frequency에서 에너지가 약해짐 (까맣게 생긴 것이 에너지가 큼)-> 옆에 analysis한 graph도 마찬가지

190926

모든 소리: simplex sound의 합
sine wave: 다양하게 패턴 바꿈
wave form: x축이 시간, y축은 그냥 value

spectrogram에서 x축은 시간, y축은 frequency(!시험)

sine wave에 F0가 정해지고, 그것의 배음의 합으로 목소리의 source가 정해짐
목소리의 pitch는 첫 번째 frequency와 일치- F0 (단위는 Hz)
저주파에선 에너지가 높고, 고주파로 가면 약해짐

입모양이 filter 역할
harmonics가 gradually decreased되지 않음
위의 source와 마찬가지로 harmonics가 일정 간격 유지

source: peak가 없음, gradually 낮아지기 때문-> 산맥 형성 안 됨
pure tone들이 high frequency로 갈수록 decrease되는 것이 voice source의 패턴

누가 /아/라고 말하든 산맥은 똑같은 패턴으로 나타남
산맥- formants

harmonics에서 제일 첫 번째- F0(pitch)
산맥에서 첫 번째, 두 번째.. 산맥을 f1, f2... 라고 말함(harmonics와는 다름)

fundamental frequency: 220Hz (guitar)
기타는 complex tone- 사람의 목소리와 똑같음 (220, 440... 의 합으로 되어있음)

100Hz부터 1000Hz까지 combined mono된 소리: 반복주기는 F0와 일치- 인지심리학적으로 100Hz(F0)의 높이와 비슷하다고 느껴짐

F0: 첫 번째 나오는 것의 frequency(Hz)
F1: 처음 나오는 peak
F2: 두 번째로 나오는 peak

영어와 한국어의 /아/: 영어가 더 back, low한 소리

191001

컴퓨터 언어는 여러 종류가 있음(java, python 등)
다 다르지만 공통점 있음- 모든 language는 단어(정보가 들어 있음, 정보를 담는 그릇)와 문법(단어를 어떻게 combine?)이 존재

변수(variable): 단어에 해당, 정보를 담는 그릇의 역할 (정보의 종류는 많지 않음- 숫자와 글자 두 가지)
기계의 문법은 생각만큼 어렵지 않음
컴퓨터 문법: 변수라고 하는 그릇에 정보를 담음, conditioning(if문법), 여러 번 반복(for-loop?), 함수를 배움(입력-출력)(제일 중요)

디렉토리: 컴퓨터에서 어디에 위치해있는지

=: 오른쪽에 있는 정보를 왼쪽의 variable로 assemble하는 것 의미 (1이라는 정보를 a라는 변수에 넣음)

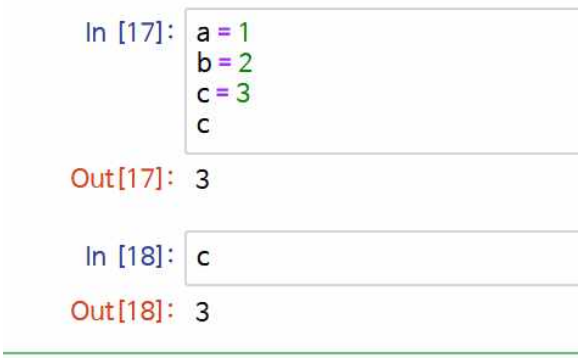
python의 모든 함수는 누가 만들어놨던걸 수도 있고, 내가 만들 수도 있고

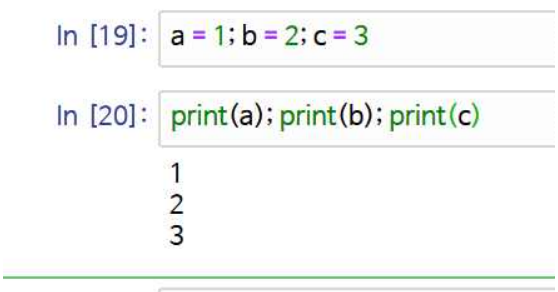
print라는 함수의 입력: a
입력을 표시해주는 것: 괄호()

셀을 초록색에서 파란색으로 바꾼 후 b라고 치면 below에 셀을 만들어줌
x를 치면 없어짐
print(a)라고 쳤을 때 1이 나오는 것은 새로운 셀이 아니라 결과값
a=2라고 치고 run을 눌러 실행하면 1은 없어지고 2가 됨-> print(a)를 했을 때 2가 나옴
print(a)=1인걸 실행하고 밑에 print(a)를 실행하면 2가 아니라 다시 1이 됨

문자를 입력하고 싶을 땐 반드시 quote를 넣어줘야 함(e.g. b = 'love')
run의 단축키: shift+enter

love = 2
b = love (b = 'love'였던 아까와는 다름)
라고 한 후 print(b)=> 2라는 결과값이 나옴

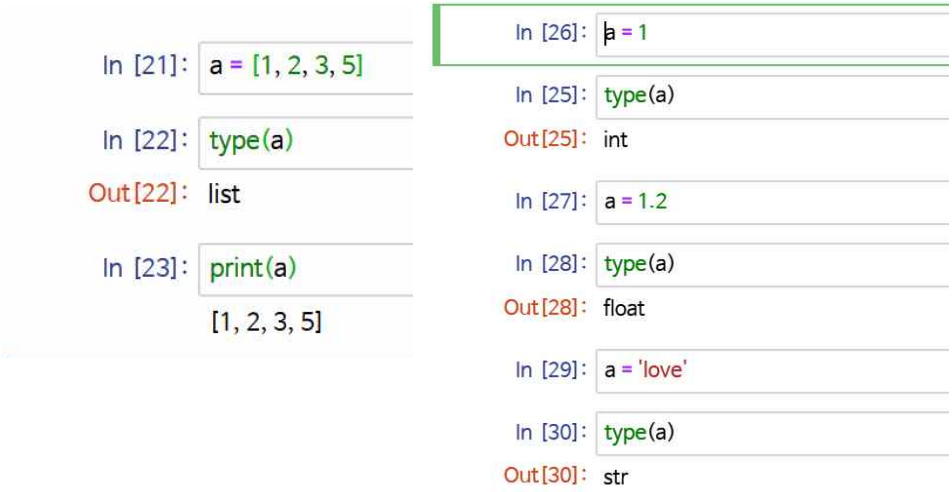




여러 번 입력한 것 중 변수를 한 번 더 써주면 print 함수 없이도 그 결과값을 보여줌, 다음

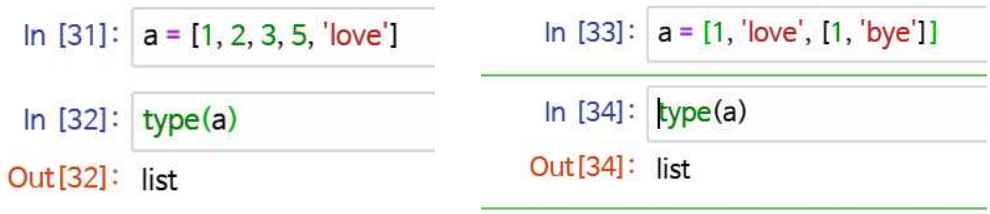
셀에 c를 그냥 쳐도 3을 보여줌

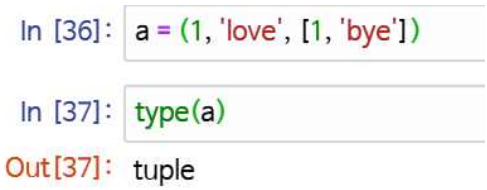
variable은 quote없이, 문자 정보는 반드시 quote(single, double 상관 없음)



[]: 여러 숫자를 한 번에 넣을 수 있음(e.g. a = [1, 2, 3, 5])

type의 종류: int, float, str, list, tuple, dict...
(type 종류 묻는 거 시험에 나올 수 있음)





list- 반드시 숫자일 필요는 없음(e.g. a = [1, 2, 3, 5, 'love'])
list와 tuple은 완전 똑같음- list는 대괄호 사용, tuple은 그냥 괄호 사용
tuple이 보안에 더 강함

a = {'a': 'apple', 'b': 'banana'} - dictionary에 2개를 넣은 것 (콤마로 인해 2개가 들어와 있음, 중괄호를 씀- 중괄호를 써야 dictionary, 몇 개를 넣을지는 콤마로 표현, 설명의 쌍은

콜론으로 표현, 콜론 안에 string만 들어갈 필요는 없음)

191008

중간- 오픈북, 암기 필요 없음, 이해해야하는 문제 위주로
순서: variables-> string-> syntax-> numpy (sound는 중간고사 이후)

```
In [6]: a = {"a":"apple", "b":"orange", "c":2014}
        print(type(a))
        print(a["a"])

        <class 'dict'>
        apple

In [9]: a = 1
        b = 1
        c = a+1
        print(c)

        2
```

제일 advanced된 것- dict
list는 단편적 정보를 넣지만, dict는 페어(콜론)로 넣음(두 개짜리 페어가 여러 개- a-apple, b-orange, c-2014)

```
In [13]: a= [1, 2]
         b = [3, 4]
         c = a[0]+b[0]
         c

Out[13]: 4

In [15]: a= [1, 2]
         b = [3, 4]
         c = a[1]+b[1]
         c

Out[15]: 6
```

a 리스트에서 0번째, b에서 0번째를 가지고 와서 더하면 4
각각 1번째를 가지고 와서 더하면 6

```
In [16]: a = 1.2; a = int(a); print(type(a))
         <class 'int'>
```

```
In [17]: a = 1; print(type(a))
         <class 'int'>

In [1]: a = 1; a = float(a); print(type(a))
        <class 'float'>

In [19]: a = 1.2; a = int(a); print(type(a))
         <class 'int'>
```

세미콜론: 두 줄 적을거 한 줄에 적고 싶을 때
float 함수: 어떤 variable이 들어오면 그것을 float로 바꿔줌(e.g. 원래 int였던 a=1이 float로 됨)
int 함수: int로 바꿔줌 (e.g. float를 int로 바꿔줌) a = 1.2; a = int(a); print(type(a))=> int

```
In [18]: a="123"; print(type(a)); print(a[1])
<class 'str'>
2
```

어떤 variable의 내부 정보에 들어갈 때는 반드시 대괄호를 씀
대괄호 안에 들어가는 것은 index
variable 이름을 a라고 적고-> 대괄호 열고 닫고-> 내부적 정보를 부분적으로 가져옴
-> 2라는 문자를 가지고 옴(숫자가 아님)

```
In [17]: a=123; print(type(a)); print(a[1])
<class 'int'>
---------------------------------------------------------------
TypeError                 Traceback (most recent call last)
<ipython-input-17-b0076a8d2193> in <module>
----> 1 a=123; print(type(a)); print(a[1])

TypeError: 'int' object is not subscriptable
```

a='123'이 아닌 a=123을 넣었을 때(quote를 뺐을 때) 에러 나는 이유: 0번째 것밖에 없기 때문

```
In [21]: a = '123'; a = list(a); print(type(a)); print(a); print(a[2])
<class 'list'>
['1', '2', '3']
3
```

str(문자)이었던 것을 list로 바꿈-> 123이 쪼개져서 각각 문자로 list화 됨

```
In [22]: a = [1,'2', [3, '4']]; print(type(a)); print(a[0]); print(a[1]); print(a[2])
<class 'list'>
1
2
[3, '4']
In [23]: a = (1,'2', [3, '4']); print(type(a)); print(a[0]); print(a[1]); print(a[2])
<class 'tuple'>
1
2
[3, '4']
```

1은 숫자로서, 2는 문자로서 print된 것임

```
In [24]: a = {"a": "apple", "b": "orange", "c": 2014}
         print(type(a))
         print(a["a"])

         <class 'dict'>
         apple

In [27]: a=[(1,2,3), (3,8,0)]
         print(type(a))
         a

         <class 'list'>
Out[27]: [(1, 2, 3), (3, 8, 0)]
```

```
In [23]: a=[(1,2,3), (3,8,0)]
         print(type(a))
         print(a[0])
         type(a[0])

         <class 'list'>
         (1, 2, 3)

Out[23]: tuple
```

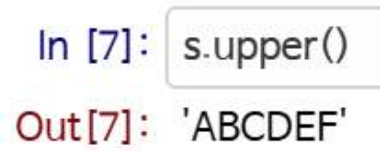
24번
a, b, c: index로서 기능 (페어에서 앞부분이 index)
사전과 같음- 표제어(a, b, c)와 내용
index의 type: 전부 다 string(quote로 되어 있음)
“a”를 1로 바꿔도 똑같음(quote 빼도 똑같음)

```
In [6]: s = 'abcdef'
        print(s[0], s[5], s[-1], s[-6])
        print(s[1:3], s[1:], s[:3], s[:])

        <class 'str'>
        a f f a
        bc bcdef abc abcdef

In [3]: n = [100, 200, 300]
        print(n[0], n[2], n[-1], n[-3])
        print(n[1:2], n[1:], n[:2], n[:])

        100 300 300 100
        [200] [200, 300] [100, 200] [100, 200, 300]

In [4]: len(s)

Out[4]: 6
```

제일 앞은 늘 0, 제일 마지막은 늘 -1, 끝에서 몇 번째를 구할때는 마이너스 기호를 붙임
[1:3] - 첫 번째부터 세 번째 직전(=두 번째)까지 가져온다는 뜻-> bc가 됨
[1:] - 첫 번째부터 끝까지
[:3] - 맨 처음부터 두 번째 까지
[:] - 전부 다

list(3번)와 string(6번)은 똑같은 방식으로 정보를 가져옴
print- 콤마를 찍으면 한 줄에서도 여러 개를 print out 할 수 있음

len: variable 내에 있는 정보의 길이, 개수
s의 길이: 6(abcdef)
n의 길이: 3



upper: 일종의 함수, s 속의 문자가 대문자로 바뀜, s를 upper 시킨 함수
파이썬과 같은 language에서는, 어떤 variable을 만들고 그 옆에 .을 붙여 함수같이 쓰면 실행됨

find: 찾아주는 함수 (string-7: s 속에 담겨있는 string 속에서 무언가를 찾는 것)
결과: 몇 번째로 나오는지 (e.g. 11번째에서 house가 시작됨- 제일 앞 띄어쓰기(space) 때문에 11번째가 됨 / this- 두 번 나오는 것 중 제일 처음 나오는 것을 찾아줌)
rindex: 여러 개 중 제일 마지막 index
strip: 잡스러운걸 지워주는 함수, 순수한 텍스트만 남겨줌

```
In [15]: tokens = s.split(' ')        # split s into a list wherever a t is found (whitespace by default)
         tokens
Out[15]: ['this', 'is', 'a', 'house', 'built', 'this', 'year.']

In [16]: s = ' '.join(tokens)         # combine the words of the text into a string using s as the glue
         s
Out[16]: 'this is a house built this year.'

In [17]: s = s.replace('this', 'that')  # replace instances of t with u inside s
         s
Out[17]: 'that is a house built that year.'
```

```
In [17]: s=','.join(token)
         s
Out[17]: 'this,is,a,house,built,this,year.'
```

split: . 앞에 오는 긴 string을(e.g. s) split 함수에 있는 입력을 이용해서 자르라는 뜻, 단어 수준에서 작업하고 싶을 때, 긴 string을 자름(여기서는 ‘’안이 space이므로, space를 기준으로 자름, 콤마가 사용되었으면 콤마일 수도 있음)
tokens: 임의로 정한 variable의 이름
join: split을 이용해 잘라 list로 자른 후 이것을 다시 문장으로 복구하고 싶을 때 사용
. 앞에 space 입력-> space(‘ ’)를 이용해 token에 들어있는 list를 붙이라는 뜻
replace: e.g. 이 string 속의 모든 this를 that으로 바꿔라

191010

syntax 문법
loop: 여러 개를 반복할 때 씀
if: 조건, 컨디셔닝
function: 입력-출력

셀에 comment다는 방법: # 다음 글씨를 씀
markdown: 셀에 무엇을 써도 실행이 안 됨, 주석의 기능 (code라고 적힌 버튼을 누르고 markdown으로 바꾸면 됨)

for loop: 여러 개를 여러 번 할 때
for: 어떤 language든 똑같이 씀

```
In [2]: a=[1, 2, 3, 4]
        for i in a:
          print(i)

1
2
3
4
```

```
In [3]: a=[1, 2, 3, 4]
        for i in range(4):
          print(a[i])

1
2
3
4
```

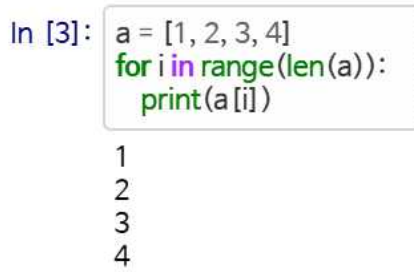
for i in a:
in 뒤에 있는 것(a)을 하나씩 돌려서 i가 그 하나하나씩을 받아 무언가를 하라를 콜론 밑에 적어주면 됨
a 속에 있는 것을 처음부터 하나씩 i에다가 넣어라(할당해라)
1번째 루프- i가 1을 받아 print하면 1이 나옴, 2번째 루프- 2가, 3번째는 3이, 그 다음에는 4

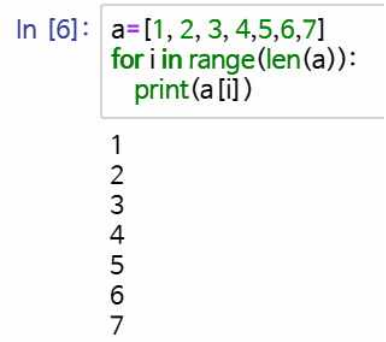
in 뒤에 list를 그대로 쓸 수도 있지만 range함수를 쓸 수도 있음
range 뒤에 어떤 숫자가 나오면 list를 만들어줌- e.g. 4를 넣으면 0부터 3까지 list를 만들어줌- 4개의 index를 만들어줌(0부터 3까지)(index는 0부터 시작)
-> 루프를 돌면서 a의 i번째가 print됨, 제일 처음에는 0번째 것(1)(i에 0이 들어와 있기 때문)-> a의 0번째 것이 print됨)이 print되고 쭉 이어지는 식

for loop 쓰는 두 가지 방법- 리스트를 그대로 쓰거나(각각의 element를 in 앞의 variable에 담아둠), range 함수를 씀(0부터 얼마까지 index를 만들어주고 그것을 i가 받음)

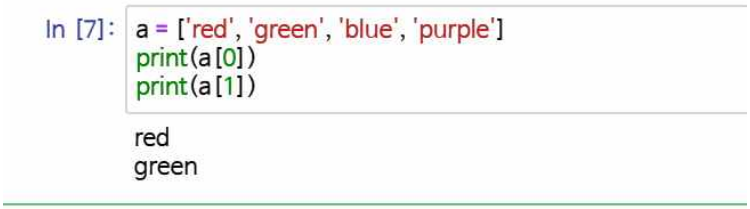
```
In [4]: a=[1, 2, 3, 4]
        for i in range(4):
          print(i)

0
1
2
3
```

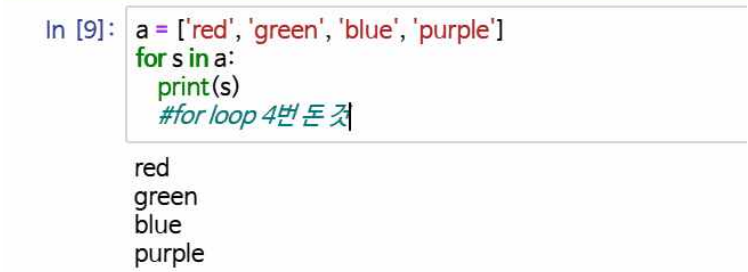
앞의 range 함수 예시에서 그냥 print(i)라고 하면 0,1,2,3이 나옴- range(4)는 0부터 3까지 4개의 index이기 때문





in[6]
range에 len(a)를 넣으면- 7개의 index(0부터 6)가 만들어짐. 그것이 for loop를 돌면서 i 받음
제일 첫 번째 루프에서 i는? 0 (range는 0에서 6)
-> a의 0번째, 1번째.. 이런 식으로 가게 됨





for loop를 쓰지 않고 결과를 내는 방법: print(a[0]), print(a[1]) 이런 식으로 노가다
for loop의 내용은 반드시 indent가 있어야 함(for loop를 치고 콜론 치고 엔터하면 자동으로 indent 생김)
in[9]
a list를 개수만큼 for loop를 돌려라- for loop를 4번 돈 것
s variable은 4번 바뀜

```
In [13]: a = ['red', 'green', 'blue', 'purple']
         for s in range(len(a)):
           print(s)

0
1
2
3
```

```
In [14]: a = ['red', 'green', 'blue', 'purple']
         for s in range(len(a)):
           print(a[s])

red
green
blue
purple
```

in[13, 14]

a의 길이만큼 range를 만들어서 loop를 돌리라는 뜻의 함수

13- 0, 1, 2, 3이 나옴-> range의 index 값이 4개가 만들어지기 때문(0부터 3)

14- a의 몇 번째를 print

```
In [15]: a = ['red', 'green', 'blue', 'purple']
         b = [0.2, 0.3, 0.1, 0.4]

         for i, s in enumerate(a):
           print(a[i])

red
green
blue
purple
```

```
In [16]: 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%
```

in[15]

a, b 각각 string과 number가 있는 길이가 같은 list

enumerate 함수: ( )안 list의 번호를 매김, output 값이 자기 자신도 되지만 그것에 대한 번호도 매겨줌 e.g. a의 list의 번호를 매김

variable이 두 개가 들어가 있음(i, s)- 번호까지 받아줘야 하기 때문(앞이 번호, 뒤에가 자기 자신인 element)

i는 번호가 enumerate되서 들어온 것-> a의 몇 번째 이런 식으로 4번 loop (s를 안 씀)

in[16]

첫 번째 variable i에는 index(번호값)를 받아오고, 두 번째 variable s에는 list의 값을 받음

-> 첫 번째 루프가 돌 때 i는 0, s는 red가 들어옴, 두 번째는 i는 1, s는 green 이런 식으로 들어옴
double quote를 하고 내가 원하는 variable의 개수만큼 중괄호를 써줌(format 안 2개가 for loop를 돌며 중괄호 속에 꽂힘)- format 안 콤마로 연결된 것의 개수랑 quote 속 중괄호 개수가 똑같음(2개)
format 함수 속에 두 개가 들어있음- 그 두 개가 for loop를 4번 돌면서 매번 두 개의 중괄호 안에 들어감
red와 20(b의 0번째인 0.2에 100을 곱한 값)이 double quote안의 중괄호 형태 속에 들어감
b[i]: b의 i번째 것

```
In [18]: 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))
```

```
red: 20.0%
green: 30.0%
blue: 10.0%
purple: 40.0%
```

a, b는 반드시 같은 길이의 list
zip: 두 개를 세트로 합침(독립적으로 존재하는 두 개의 list가 페어가 됨-> 루프를 돔)
첫 번째 루프에서 red와 0.2가 각각 s, i에 들어감

```
In [24]: a=0
         if a == 0:
           print("yay!")
           print("let's go")
```

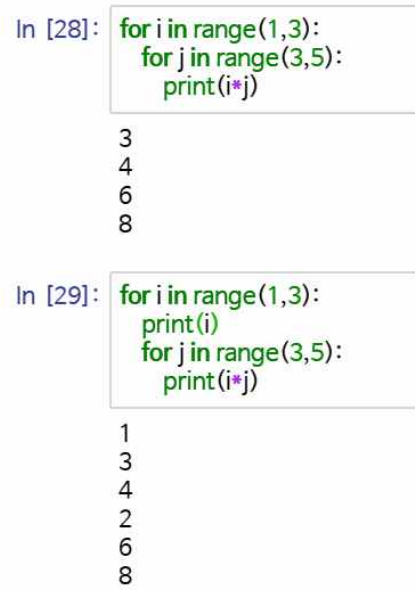
```
yay!
let's go
```

```
In [27]: a=0
         if a != 0:
           print("yay!")
           print("let's go")
         else:
           print("no")
```

```
no
```

만약 a가 0이면 yay!와 let's go를 print 해라
equal sign을 두 개 넣어줌(==)-> 진짜 equal sign이 됨 (if에서 equal sign을 많이 씀)
>=: 크거나 같음(부등호 순서 바뀌면 안 됨- equal sign 앞에, ==대신 넣어줌)
!=: 아닌 경우 (if a !=0: print("t") - a가 0이 아니면 t를 print 해라)
else: 맞으면 ~하고, 아니면 ~하고

range 다음 숫자 하나가 들어가면? (e.g. range(10)- 0부터 9까지 index가 만들어짐)
숫자 두 개가 들어가면?
e.g. for i in range (1,3):
print (i)
<결과>1, 2
: 1부터 3 직전까지 감-> 1, 2가 i에 받아지면서 loop가 돎



in[28]
시험- 두 번 for loop
제일 바깥 for loop- 2번 돎(range가 1, 2 이렇게 돎)-> 각각 1, 2 for loop 할 때도 2번씩 돎
-> 총 4번 실행됨
-> 1x3, 1x4, 2x3, 2x4인 결과가 나옴

in[29]
print(i): 첫 번째 for에서만 실행됨, 2번 실행됨
print(i*j): 4번 실행됨

```
In [30]: for i in range(1,3):
             for j in range(3,5):
                 if j>=4:
                     print(i*j)

4
8
```

```
In [31]: for i in range(1,3):
             if i >= 3:
             for j in range(3,5):
                 print(i*j)
           File "<ipython-input-31-cd2f4c498173>", line 3
             for j in range(3,5):
             ^
         IndentationError: expected an indented block
```

in[30]
일단 크게 for loop- i가 바뀌면서 1부터 2까지 2번 돎-> 각각의 i에 대해 j가 2번 돎
실행에 if를 걺- j가 4보다 커야한다는 조건을 통과해야함(j가 3일 때는 안 돌 것이고, 4일 때만 돌 것)
-> 4번이 아니라 2번만 돎

in[31]
error: if 밑에 나오는 for가 indent되어야 함
indent 되면? 아무런 결과가 안 나옴- i가 1, 2 이렇게 돌아 3보다 크거나 같을 수 없기 때문

191015

집중적으로 배웠던 부분: articulation
5 speech organs 그림 중요(velum, major articulation 3가지(lips, tongue tip, tongue body)- 제일 중요, larynx), 어떻게 소리 specify?
larynx: vocal cord가 vibration-> 유성음/무성음
velum(soft palate): lowered/raised (nasal sound를 나눔)
lip, tongue tip, tongue body

constriction degree: 혀를 어느 정도로 올리는지
constriction location: 혀가 앞으로 가는지/ 뒤로 가는지

constriction degree: approximants(r, l, j, w)
모든 영어 phoneme은 stop, fricative, approximant, vowel 중 하나

velum, larynx, lip/tongue tip/tongue body, constriction degree and location- 모든 가능성 중 거기 해당하는 영어 phoneme이 없을 수도 있음- 어떤 조건인지 생각해올 것
lips가 constriction location으로 velar 가질 수 있나? 영어에서의 gap인가 사람이 못 하는

건가? 사람이 할 수 없는 소리
lips가 alveolar를 constriction location으로 가지는 language가 이론상 가능한가? 가능하긴 함(accidental gap), velar는 physical하게 불가능

praat: pitch, intensity, formant가 뭔지
중요! source-filter theory 그림을 설명할 수 있어야 함
source 부분에서 pitch가 조절됨, pitch는 제일 처음에 있는 pure tone에 의해 결정됨(F0(pure tone의 frequency)의 크기가 pitch)
입모양이 filter 역할-> source spectrum이 filter에 의해 재구성됨(peak와 valley가 결정됨, source에선 peak 없음, 제일 첫 번째에 있는 peak- first formant(f1))
F0과 f1, f2는 다름- 그림도 다름
f1과 f2에 의해 모음이 결정됨
f1: 높낮이 결정
f2: front, back 결정, f2가 클수록 front(i>e)

variable- number(int/float)/str
하나 이상의 정보 저장- list(대괄호), tuple(괄호)
페어로 묶음- dict(표제어와 내용)

시험! str과 list는 유사
list는 index 이용(정보에 부분적으로 access)
str도 정보 access 할 때 대괄호 사용할 수 있음- list와 똑같음

dict도 콜론의 앞부분을 index로 이용할 수 있음

제일 끝: -1, 그거에서 하나 앞: -2
range 이용하고 싶을 때: 콜론 사용

split이랑 join이 중요
split: s속에 담긴 str을 split 다음에 있는 캐릭터를 가지고 spilt하라

함수를 잘 알아야 함: print, range, len 등

indent 다음 내용이 for이 지배하는 부분
for in: in 뒤에거부터 먼저 읽음, in에 있는 것들을 하나하나씩 루프를 돌려서 in 앞에 있는 거에 넣음

enumerate, zip, format 어떻게 쓰는지 잘 보기

if 쓰는 방법: 콜론 꼭 써야함(else를 쓸 때도)

모음: 무조건 tongue body

int, float, join, split 이런 것도 함수임

파찰음은 안함

여자가 남자보다 F0가 높음
듬성듬성 있는 것이 여자 것(F0가 높기 때문)- 주어진 frequency range 속에서 남자의 pure tone이 더 많음
e.g. 남자가 105Hz면 0부터 10000까지 몇 개 들었는가? 105, 210, ... 세보면 됨 (filtered되어도 숫자값은 안 변함)

191029

영어의 nasal consonant는 velum lowered-> nasal tract으로 air flow가 있지만 oral tract는 막혀서 air flow가 없다: T
영어모음 /a/, nasal tract의 air flow 차단됨: T
발화시 air pressure와 가장 관계있는 것: 진폭과 관계가 있음-> intensity
자음 중 코를 막고 숨을 쉴 때 가장 유사한 articulation 상태인 음소? h
영어 /h/는 어떤 articulator(lip, tongue tip, body)에서 constriction? none
음소 중 tongue body에서 constriction, constriction degree는 frecative, location은 velar인 자음 개수: 없음
영어 자음 /v/의 articulator의 location을 bilabial, degree를 approximant로 바꾸면 어떤 음소? w
/s/, /l/은 major articulators 중 공통적으로 tongue tip에서 조음됨
어떤 모음의 pitch가 128Hz일 때 F1은 128Hz보다 반드시 크다: T
/i/ 모음 128Hz로 발화, 똑같은 pitch로 /a/ 발화, 각각의 모음에 대해 0Hz~5000Hz 사이 몇 개의 harmonics 존재하는지 구하면? 같은 개수

```
a=[[1], [2,3], [4,5,6]]
n=0
for b in a:
 for d in b:
    n+=1
print(n)
```

6(1번 돌고, 2번 돌고, 3번 도니까)

```
n=[1,2,3,4]
for i in range(len(n)):
   print(i)
```

몇 개의 unique한 함수가 있는가? range, len, print 3개
함수가 몇 번 실행되는가? len(1번), range(1번), print(4번 실행)- 6번

a=[1,2,[3,4]]
print(a[-1][-1])
4
이유: [3,4]-> 4

b= [1, 'a', {'a': [3,6], 'b':[9]}, 'b']
print(b[-2]['b'][-1])
9

c=[1, 'a', {'a': 'abc'
print(c[-1]['b'][-2])
e

print('join(d)[-3]
h

17번: e에 list 4개-> for loop 4번 돎
dict라고 나오는 것이 없음
k만 print out됨

18번: range 3: 0,1,2
14*1

19번: a의 length는 4

20번: tuple, element 3개
len(a): 3
int(10): 10

191029

for loop는 함수가 아님, 함수는 반드시 함수명과 괄호가 있어야함

행렬 알아야함- 직사각형 안에 숫자
vector화 해야함
이미지는 행렬임, 그것을 길게 늘어놓은 것을 vector화 한다고 함
모든 데이터는 vector의 형태로 해야 다루기 쉬움
컬러: 행렬이 3겹으로 있음(RGB)
흑백 이미지: 2차원, color image: 3차원, 시간까지 있으면 4차원
이미지도 벡터, 소리도 벡터화 됨, 텍스트도 벡터화 됨-> 모든 데이터는 벡터화

library: 쓰기 좋게 함수들을 누가 만들어놓는 것, 추가해서 우리가 쓸 때 library를 불러옴

numpy: list 중에서도 그 안에 숫자가 들어갈 때

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

In [4]: import numpy

In [5]: A = numpy.array(a)
        B = numpy.array(b)

In [6]: A + B
Out[6]: array([ 3,  7, 11])
```

```
In [7]: type(A)
Out[7]: numpy.ndarray

In [8]: import numpy as np

In [9]: A = np.array(a)
        B = np.array(b)

In [10]: X = np.array([[1,2,3], [4,5,6]])
         X
Out[10]: array([[1, 2, 3],
                [4, 5, 6]])
```

```
In [11]: X. shape
Out[11]: (2, 3)
```

in[3]: list 안에서는 수학적 계산이 일어나지 않음, 수학적 계산을 해주는 것이 numpy
import numpy를 해와야 numpy를 쓸 수 있음
array: list를 행렬 혹은 벡터로 만들어줌, 계산이 가능하게 바꿔줌(list에선 계산 불가)
numpy를 np로 많이 줄여씀

X.shape: 차원(2 by 3)- 2행 3열의 matrix(행렬)

191031

numpy를 import해야 함
numpy라는 library 속에 다른 package들이 있을 수 있음
from numpy로 쓸 수도 있음- from numpy import A(numpy 안에 있는 A를 불러오자), from numpy import A.C- numpy 안에 있는 A 안의 C 함수 불러옴

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
```

= from matplotlib import pyplot as plt
matplot: plotting과 관련됨
.은 library의 포함관계를 말함

numpy: list와 비슷
수학적으로 계산할 수 있기 때문에 사용-> 데이터 처리는 list가 아닌 numpy 처리를 해야 함

```
In [3]: np.empty([2,3], dtype='int')
Out[3]: array([[8, 0, 0],
               [0, 0, 0]])
```

empty: 함수(괄호로 input 받음)
입력 list로 들어감(2 by 3의 행렬(2행 3열)) / dtype: 데이터 타입(int)
2 by 3의 array로 결과값 나옴, 3개짜리 리스트가 두 개(두 줄) 나옴, 안에 있는 숫자는 랜덤이지만 int라고 적어뒀기 때문에 소수점은 아님, 여러 번 실행할 때마다 값이 달라짐

```
In [4]: np.zeros([2,3])
Out[4]: array([[0., 0., 0.],
               [0., 0., 0.]])
```

numpy 속의 zero라는 함수- 2 by 3의 matrix 만들어짐, 0으로 채워져서 나옴

```
In [5]: [[0, 0, 0], [0, 0, 0]]
Out[5]: [[0, 0, 0], [0, 0, 0]]

In [6]: np.array([[0, 0, 0], [0, 0, 0]])
Out[6]: array([[0, 0, 0],
               [0, 0, 0]])
```

in[5]
2 by 3(2행 3열)- list(아무 쓸모 없음- 계산이 안됨)
in[6]- 이 list를 array로 바꿔줌
2 by 3의 행렬처럼 생긴 list를 입력으로 하는 array 함수
in [4]와 [6]의 결과값 같음
[6]: list를 array로 convert해라

```
In [7]: np.ones([2,3])
Out[7]: array([[1., 1., 1.],
               [1., 1., 1.]])
```

in[7]
1이 만들어짐
.이 있음- int가 아니라 float(default로 data type을 float로 함)
뒤에 dtype = 'int' 라고 콤마 써서 덧붙이면 결과에서 점이 사라짐

arange, linspace- 중요한 함수

```
In [9]: np.arange(5)
Out[9]: array([0, 1, 2, 3, 4])

In [10]: np.arange(0,10)
Out[10]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [11]: np.arange(0,10, 2)
Out[11]: array([0, 2, 4, 6, 8])
```

```
In [5]: np.arange(0,10,2, dtype='float64')
Out[5]: array([0., 2., 4., 6., 8.])

In [6]: np.linspace(0,10,6, dtype=float)
Out[6]: array([ 0.,  2.,  4.,  6.,  8., 10.])
```

in[9]
계산이 될 수 있는 array를 만들어줌- 숫자 5개가 나옴(index 5개, 0부터 4까지)
in[10]
0부터 10 전까지 index 만들어줌
in[11]
0부터 2씩 뛰면서 만들어줌(10 미만까지)

linspace: arange와는 달리 10까지도 포함
in[6]: 처음(0)과 끝(10)을 포함하여 6개로 똑같이 나눠줌

```
In [12]: np.linspace(0,10, 7)
Out[12]: array([ 0.        ,  1.66666667,  3.33333333,  5.        ,  6.66666667,
                 8.33333333, 10.        ])
```

in[12]
linspace: 각 값의 space가 똑같음(각 값의 차이가 똑같음), 0부터 10까지 7개로 나눠줌

```
In [6]: X = np.array([[1,2],[4,5], [8,9]])
        X
Out[6]: array([[1, 2],
               [4, 5],
               [8, 9]])
```

```
In [14]: X = np.array([[[1,2],[4,5], [8,9]],[[1,2],[4,5], [8,9]]] )
         X
Out[14]: array([[[1, 2],
                 [4, 5],
                 [8, 9]],

                [[1, 2],
                 [4, 5],
                 [8, 9]]])
```

in[6]
3 by 2의 벡터
처음과 끝 대괄호 2개가 붙어있음-> 2차원(직사각형) (3개가 붙어있으면 3차원)

in[14]: 3차원(2차원 행렬이 2개)

```
In [16]: X.ndim
Out[16]: 3

In [17]: X.shape
Out[17]: (2, 3, 2)
```

X.ndim: 3차원 / X.shape: 숫자 3개-> 3차원, 3x2 직사각형이 2개 있음, 첫 번째 대괄호 안에 2개, 두 번째 대괄호 안에 3개, 세 번째 대괄호 안에 2개씩 있음

```
In [18]: X.dtype
Out[18]: dtype('int32')
```

```
In [19]: X.astype(np.float64)
Out[19]: array([[[1., 2.],
                 [4., 5.],
                 [8., 9.]],

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

X.dtype: 원래 type
astype: 괄호 안의 함수로 바꿔줌- int가 float가 됨

```
In [20]: np.zeros_like(X)
Out[20]: array([[[0, 0],
                 [0, 0],
                 [0, 0]],

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

in[20]: 모든 형태를 유지한 채 숫자만 다 0으로 바꿔줌
np.zeros_like(X) = X*0

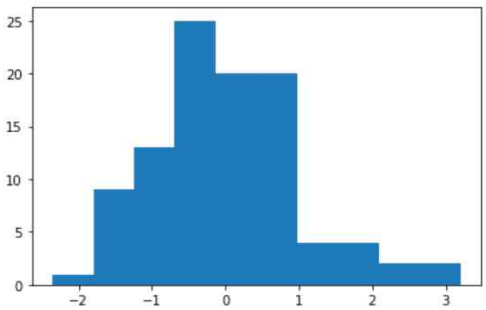
```
In [21]: data = np.random.normal(0,1, 100)
         print(data)
[-0.3090026  -0.32504424 -0.44614693  1.04910448  0.66478465  0.0
4274331
 0.21942682 -0.24659093  0.63294339  0.04867212  0.73260655 -0.35
262619
 -0.3024767  -1.35398983  0.74447787  1.73774169 -1.16687593  0.30
030576
 -0.33684799 -0.49332469  3.19782685 -2.35298113  1.37884563 -0.
33015695
 -0.50049738 -0.98808965 -0.14535265  0.04950957 -0.01851825 -1.
13624461
  0.89979606  0.61353755 -1.62206404  2.87083135  2.30170832  0.49
```

```
In [22]: data.ndim
Out[22]: 1

In [23]: data.shape
Out[23]: (100,)
```

np 안에 random이라는 subpackage, 그 속의 normal이라는 함수
normal: normal distribution을 만들어주는 함수
괄호 안에 차례로 mean(0), standard deviation(1), data의 개수(100)
in[21]

100개의 랜덤한 데이터가 나옴, normal distribution을 이루는 랜덤한 데이터
data ndim하면 1차원, shape은 100(총 100개의 숫자)

```
In [24]: plt.hist(data, bins=10)
         plt.show()
```



histogram
bins: 바구니, 바구니를 몇 개로 할지 정함(10개)
y값 절대 소수로 나올 수 없음, 전부 정수값(0 포함한 자연수)
y축 합하면 몇 개인지 시험에 나올 수 있음- 100개

```
In [25]: X=np.ones([2,3,4])
         X
Out[25]: 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 [26]: Y = X.reshape(-1, 3, 2)
         Y
Out[26]: 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[25]
2 by 3 by 4- 3차원의 행렬값(대괄호 3개)
안에 있는 element의 개수: 24

in[26]
reshape: shape을 바꿈 (중요한 함수)
shape이 바뀌어도 element 개수는 안 바뀜
-1: 원래 4를 적어야하는데, 모르겠을 때 알아서 하란 의미로 앞에 -1이라고 침
element는 여전히 24개

```
In [27]: np.allclose(X.reshape(-1, 3, 2), Y)
Out[27]: True
```

allclose: 두 함수(X와 Y)가 똑같은 형태인지 비교

```
In [40]: a = np.random.randint(0, 10, [2, 3])
         b = np.random.random([2, 3])
         np.savez("test", a, b)

In [30]: del a, b
```

randint: 0부터 10 사이 숫자를 pick up해 2 by 3의 matrix로 만들어줌
random: random을 2 by 3로 만들어냄
savez: 실제 file로 저장해줌
data를 메모리에서 지우고 싶을 때: del하고 variable(a, b) 이름을 적음

```
In [32]: npzfiles = np.load("test.npz")
         npzfiles.files

Out[32]: ['arr_0', 'arr_1']
```

```
In [33]: npzfiles['arr_0']

Out[33]: array([[4, 8, 1],
                [6, 0, 9]])
```

in[32]: np.load 함수를 써서 만든 파일을 불러옴
in[33]: a라고 정의된(저장된 파일) variable을 불러옴

```
In [49]: data = np.loadtxt("regression.csv", delimiter=",", skiprows=
         1, dtype={'names':("X", "Y"), 'formats':('f', 'f')})
         data

Out[49]: 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')])
```

loadtxt: 저장된 데이터를 불러옴 (반대는 savetxt라고 적으면 됨)
csv: comma separated values(모든 것이 콤마로 분리 됨)(e.g. X, Y)
delimiter: 콤마, 분리시키는 것
skiprows=1: 처음 것 빼기
dtype: 처음은 X, 두 번째는 Y라고 하자고 적음
formats: 두 개 다 float

```
In [34]: 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
```

in[34]
random 함수를 써서 만들어지는 random data의 shape을 5 by 2 by 3로 만듦(3차원의 matrix)
type: numpy가 만들어낸 n dimensional array (numpy type)
length: 5
shape이랑 dimension이 중요- shape: 우리가 정한 대로 나옴 / dimension: 3(3차원)
size: 총 element의 개수(5*2*3)
dtype: default로 만들어진 것

```
In [55]: a = np.arange(1, 5)
         b = np.arange(9, 5, -1)

In [56]: print(a - b)
         print(a * b)

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

```
In [26]: a = np.arange(1, 5)
         print(a)

[1 2 3 4]
```

```
In [17]: a=np.arange(1, 5, +1)
         print(a)

[1 2 3 4]
```

```
In [19]: a=np.arange(1, 5, +2)
         print(a)

[1 3]
```

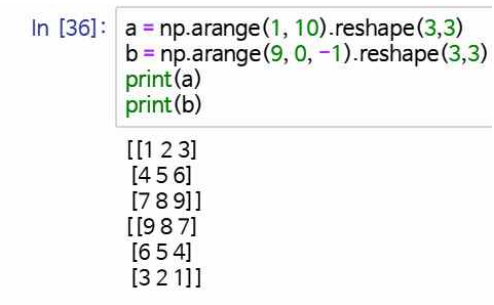
```
In [27]: b=np.arange(9, 5, -1)
         print(b)

[9 8 7 6]
```

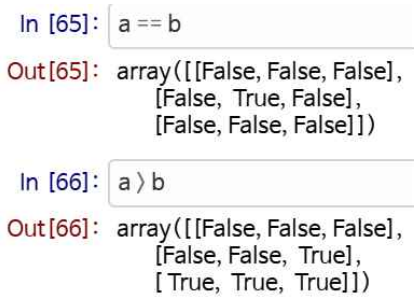
```
In [28]: b=np.arange(9, 5, -2)
         print(b)

[9 7]
```

```
In [29]: b=np.arange(9, 5, -3)
         print(b)

[9 6]
```

```
In [30]: b=np.arange(9, 5, -4)
         print(b)

[9]
```

a: 1부터 5 직전까지
b: 9부터 6까지 (제일 마지막은 포함하지 않음)
a, b 둘 다 numpy array-> 계산 가능한 상태(in[56])



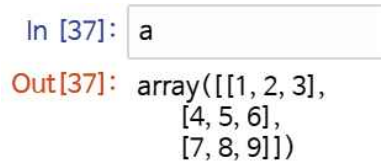


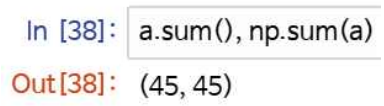
in[36]

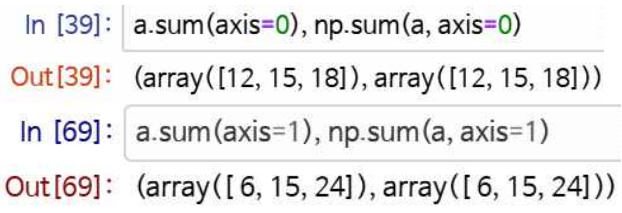
a: 1부터 9까지 1차원으로 나옴, reshape로 2차원으로 만듦(3*3)

b: 9부터 1까지, reshape로 3*3으로 나옴

in[65, 66]: 같은지, 큰지 작은지 비교, 비교하기 위해서는 두 개의 dimension과 shape이 같아야함







in[37]: a에 들어있는 것 확인

in[38]

np.sum과 a.sum은 같은 말- numpy 속에 들어있는 sum 함수를 씀, a를 sum하라

a 자체가 numpy의 산물이기 때문에 a.sum()이라고 써도 가능함(함수라서 반드시 괄호가 있어야 함, 자기 자신이기 때문에 괄호 안에 무언가를 적을 필요는 없음)

in[39]

axis=0: 0번째 차원(첫 번째)

a의 첫 번째 차원: [1, 2, 3], [4, 5, 6], [7, 8, 9]

a의 두 번째 차원: [1, 4, 7], [2, 5, 8], [3, 6, 9]

첫 번째 차원의 값들을 더함(위에서 아래로)-> 1+4+7, 2+5+8, 7+8+9

두 번째 차원의 값들을 더함(양 옆으로)-> 1+2+3, 4+5+6, 7+8+9

sampling rate가 10000이면? 1초에 10000개의 숫자를 담음

191105

np.array 한 다음 리스트를 넣어주면 numpy 형태의 data로 바뀜

다차원의 array를 만들 수 있음-> 계산 가능해짐

pure tone(사인, 코사인의 wave)의 합이 복잡한 sound 만듦

sinu soidal: 사인, 코사인처럼 생긴 wave, 곡선
phasor: sinu soidal을 만들어내는 것(사인 function, 코사인 function 둘 다 phasor)

sin( ), cos( ) 괄호 안에 들어가는 입력값(θ): degree(e.g. 0°, 180°)가 아닌 radian
π: 180도 2π: 360도
sin 0 = sin 2π
코사인, 사인 그래프 생긴 모양만 외우기
0부터 100π까지 사인 or 코사인 그래프를 그리면 총 몇 번의 반복? 50번 (2π가 반복되기 때문, sin이든 cos이든)

$\cos(\frac{3}{2}\pi) = 0$ $\sin(\frac{3}{2}\pi) = -1$

오일러 공식
e: 상수값(2.71...) i: imaginary의 약자, 허수
θ값이 변하면 결과가 달라지는 함수라고 생각하면 됨

모든 수를 포함하는 카테고리: 복소수(실수와 허수 모두 포함, a+bi)
복소수 평면: complex plane

벡터: 숫자열
a+bi에서 (a,b)도 벡터값

코사인 그래프는 1부터, 사인 그래프는 0부터 시작
a축(실수 부분): 코사인과 똑같음(1부터 시작해서 내려갔다 올라갔다)
b축(허수 부분): 사인과 똑같음(0부터 시작해서 올라갔다 내려갔다)

사인, 코사인, 오일러 공식의 input은 공통적으로 θ(radian, 각도값)-> phasor를 만듦
sin(θ)에 시간의 개념이 들어갈까(e.g. frequency(초당 몇 번 왔다갔다하는지))? 안 들어감(각도값이기 때문, 몇 초에 몇 바퀴를 도는지는 안 들어감)
소리라는 실체는 반드시 시간의 개념이 필요함, 단순한 sin(θ)는 실체의 소리가 될 수 없음

```
In [160]: # 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
```

amp: amplitude, 진폭(1에서 -1까지)
freq: frequency, 단위 Hz, wave가 1초에 몇 번 왔다갔다하는지, shape의 반복
dur: duration, 소리가 몇 초 동안 나는지
sr: sampling rate, 단위 Hz, 음질 상 얼마나 고해상도인지(1초에 10000개로 표현), 얼마나 정보를 촘촘하게 할지, 점들(숫자)이 1초에 얼마나 많이 나오는지- freq와의 차이점

```
In [161]: # generate time
          t = np.arange(1, sr * dur+1)/sr
```

t = 0.0001, 0.0002, ... 0.5000까지 가야함- duration 0.5, sampling rate이 10000이니까 1초라면 10000개까지 간다는 개념이 들어감 (amp와 freq의 개념은 아직 들어가지 않음)
in[161], [162]가 중요
sr*dur+1: sampling rate를 10000개 다 쓰는 것이 아니라 5000까지만 씀(+1 붙는 이유: 제일 마지막건 사용하지 않기 때문)
-> 1부터 5000까지 만들어지고 나누기 10000-> 10000분의 1초, 10000분의 2초, ... 10000분의 5000초까지 만들어짐
실제 time은 아님

```
In [11]: t
Out[11]: array([1.000e-04, 2.000e-04, 3.000e-04, ..., 4.998e-01, 4.999e-01,
                5.000e-01])
```

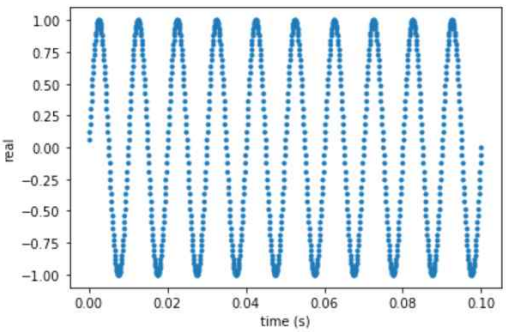
array로 1차원 벡터가 만들어짐
1.000e-04=10000분의 1초(0.0001) 4.998e-01=0.4998

```
In [162]: # generate phase
          theta = t * 2*np.pi * freq
```

```
In [7]: s = np.sin(theta)
```

time을 먼저 만들어내고 θ값을 통해 time과 연동시켜 phase로 바꿔줌- time을 먼저 만들고 거기에 phase를 연동시킴(time과 각도의 개념이 동시에 들어감)
2*np.pi=2π (np.pi는 그냥 π)
0부터 2π는 한 바퀴(t*2π는 한 바퀴), freq=100은 2π가 1초에 100개(100바퀴 반복)라는 뜻
in[161]과 in[162]에는 똑같은 개수의 벡터가 들어감(time-5000, theta-5000)

```
In [8]: 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')
Out[8]: Text(0, 0.5, 'real')
```



plot 함수는 두 개의 입력값을 받음(x와 y값)- time(t), sine의 결과값(s)
'.': 점으로 plotting하라는 뜻
5000개 다가 아니라 1000개만 plotting함

```
In [72]: # generate signal by complex-phasor
         c = np.exp(theta*1j)
```

exp: 오일러

191107

```
In [118]: 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
```

sound의 library

as: 줄임말

첫 번째 줄 큰 library의 이름: matplotlib (첫 번째 줄 = import matplotlib.pyplot)

```
In [160]: # 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 [161]: # generate time
          t = np.arange(1, sr * dur+1)/sr
```

in[160]: parameter setting- 변수를 미리 세팅해둠

in[161]: time을 제일 먼저 setting- 사인, 코사인, 오일러 공식이 직접적으로 받아들이는 입력값은 무조건 radian(각도값)이지만 각도값만으로는 실체의 소리를 만들 수 없음

```
In [31]: theta = np.arange(0, 2*np.pi)
         theta
Out[31]: array([0., 1., 2., 3., 4., 5., 6.])
```

0부터 2π까지 만든다고 할 때 제일 첫 번째 값은 0, 제일 마지막 값은 2*np*pi(6.28...) - 각도값이 정의됨(radian이 단위)

theta를 사인함수에 넣을 준비가 된 것

2*np*pi 뒤에 콤마하고 0.1 넣으면 그래프 모양이 더 빽빽해짐(array가 0, 0.1, 0.2, ... 6.2까지 나옴)

```
In [34]: s = np.sin(theta)
         s
Out[34]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
                -0.95892427, -0.2794155 ])
```

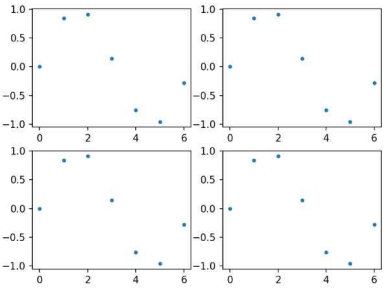
s(signal)로 받은 7개의 벡터값이 나옴-> plot하면 사인 곡선이 나옴

figure: function(괄호가 있기 때문)

add_subplot: figure에 담긴 함수

-> 전체 figure를 만들고, ax라는 변수로 받고 (figure는 전체 화면, subplot은 그 안에서 분리된 것)

```
In [36]: fig = plt.figure()
         ax = fig.add_subplot(221)
         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, '.')
```



subplot안 숫자(221): 총 2 by 2로 나누는데(2행 2열), 그 중 첫 번째(첫 번째 네모)
x축: theta(각도값, 0부터 2π까지)
y축: 사인함수의 결과(태극 문양처럼 이어짐, 0부터 시작, 7개의 theta값이 사인함수에 들어감 -> 7개의 s값이 나옴)
line처럼 생겼을 때 x축도, y축도 모두 equidistant, x와 y의 관계가 일차함수(캡쳐한 이미지는 non-linear, default는 line('.'을 없앰))
곡선이 나타난다는 말 자체가 x의 equidistant한 것이 y축에는 반영 안 됨
위의 그래프에는 시간의 개념이 들어가 있지 않음- theta 뿐만 아니라 time도 필요-> 시간부터 먼저 만듦(in[161])

```
In [37]: t=np.arange(1, sr)
```

```
In [19]: t = np.arange(1, sr*dur+1)/sr
```

```
In [162]: # generate phase
          theta = t * 2*np.pi * freq
```

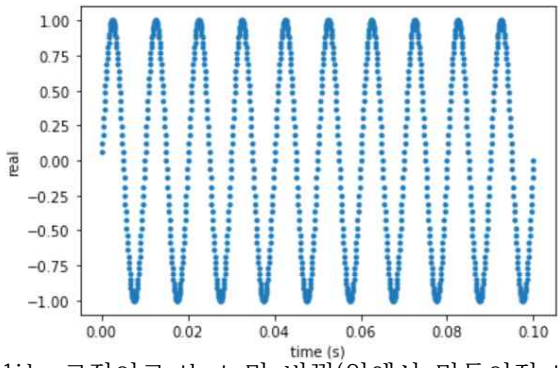
in[37]
time tick의 개수를 index로 먼저 만듦- 1초라면 time tick의 개수는 sampling rate와 일치, 1초가 아닌 경우에는 더 작아짐
in[19]
sr에 duration을 곱하면(sr*dur+1): sampling rate 개수만큼 time tick이 만들어짐- duration이 비율로 반영된 것(dur에 +1해야 제일 마지막 것까지 반영)
-> time이 만들어짐(t)-> 이후 theta 적용시킴(in[162])
freq: 몇 바퀴인지, 반복의 수

실제로 plot할 때는 theta가 아니라 time이 x축에 들어감(실체의 소리에 필요한 정보는 time, theta는 이미 사인의 형태로 반복되는 것을 우리가 알고 있기에 굳이 plot해주지 않음)
t벡터와 s벡터 대괄호 안에 들어가는 수는 같아야함(개수가 안 맞으면 실행이 안됨)
점은 2차원 벡터
점들의 개수: 1000개(시험에 나올 수 있음)

```
In [32]: c = np.exp(theta*1j)
         c
Out[32]: array([0.99802673+6.27905195e-02j, 0.9921147 +1.25333234e-01j,
                0.98228725+1.87381315e-01j, ..., 0.9921147 -1.25333234e-01j,
                0.99802673-6.27905195e-02j, 1.      +1.96438672e-15j])
```

np.exp: 오일러 공식에서의 e / 1j: 오일러 공식에서의 i

```
In [28]: 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')
Out[28]: Text(0, 0.5, 'real')
```



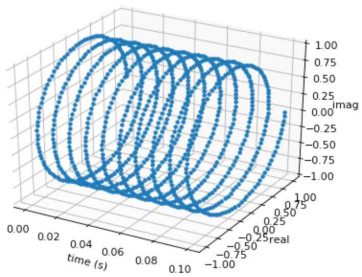
np.exp, 1j는 고정이고 theta만 바뀜(위에서 만들어진 theta가 입력으로 들어감)

c를 print했을 때 하나하나의 값이 엄청나게 긴 벡터, 전부 a+bi의 형태

-01: $\frac{1}{10}$, -02: $\frac{1}{10^2}$

array된 결과들의 표기방법이 같음- 쓰는 숫자, 정보의 양을 똑같이 함 (컴퓨터는 정보량을 정해야함)

```
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')
```



projection: 2d가 아니라 3d로 해줌

ax.plot에 입력값이 3개 들어감- 3차원이기 때문(3차원의 벡터로 된 점들, (x, y, z))

1000개의 점이 찍힘(세 입력값 모두 1000개로 똑같아야 함)

x축: time / y축: real / z축: imaginary

c.real: a+bi에서 a만 빼오는 역할(e.g. 0.99802673)(복소수를 real part와 imaginary part로 분할하여 받아와 그것을 plot)

오일러 공식에서 cos: real part와 관련, sin: imaginary part와 관련 (위에서 real만 보면 코사인 그래프처럼 보이고, 옆에서 imaginary만 보면 사인 그래프처럼 보임)

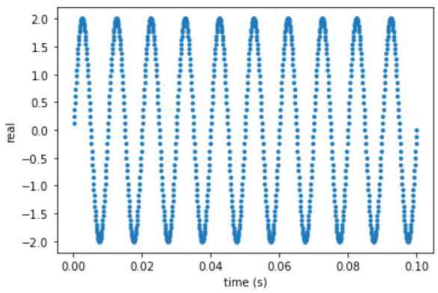
```
In [36]: ipd.Audio(c.real, rate=sr)
Out[36]:
```

ipd: 오디오를 재생하기 위해 import(입력값: 시그널(벡터, 여기서는 complex의 real)과 sr)

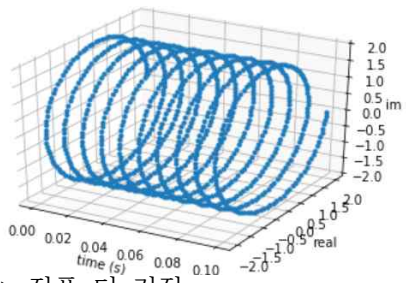
191112

사인, 코사인 wave는 시간 없이도 만들어짐(theta값만 있어도 커브는 만들어짐), 그것만으론 실제 소리를 만들 수는 없음(소리는 시간의 개념이 반드시 들어가야 함)
사인, 코사인(phasor)에 들어가는 입력값은 각도값, radian- 여기에 시간을 연동시켜야 소리가 됨

```
In [7]: s =amp*np.sin(theta)

In [8]: 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')
Out[8]: Text(0, 0.5, 'real')
```




```
In [9]: c = amp*np.exp(theta*1j)

In [10]: 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')
Out[10]: Text(0.5, 0, 'imag')
```



default amp 값은 1이지만 이를 2로 함-> 식에 곱해줌-> 진폭 더 커짐

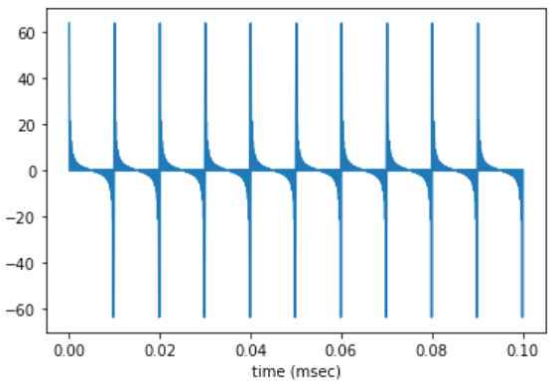
여러 다양한 harmonics를 만들어 우리 성대에서 나는 소리와 똑같은 소리 만들기
가장 최소의 harmonics를 먼저 정하고, 그것의 sine wave를 정함- 처음 frequency(f0)를 100Hz로 정하면, 그 다음 200, 300 이런 식으로 이어 나감

sampling rate와 frequency가 연결되는 부분이 있음
sr이 100Hz일 때-> 우리가 표현할 수 있는 숫자는 1초에 100개
이 100개의 숫자로 1Hz의 frequency를 표현할 수 있나? ㅇㅇ있다(한 번의 sine wave 주기가 있으면 표현 가능함)
2Hz는? 2번 왔다갔다하면 됨

주어진 100개의 숫자로 10000Hz 표현 가능(1초에 10000번 왔다갔다 하게 할 수 있는가)? 안됨, 숫자가 너무 적음-> sr이 충분히 있어야 그만큼의 주파수 표현 가능
우리가 표현할 수 있는 주파수는 주어진 숫자의 개수의 반까지 밖에 안 됨-> sr의 반까지 표현할 수 있음(sr이 10Hz면, 표현할 수 있는 frequency는 최대 5Hz)- nyquist frequency는 sr의 2분의 1
sampling rate의 half에 해당하는 nyquist frequency 까지가 숫자상으로 표현할 수 있는 frequency의 maximum
-> 우리가 표현하고자 하는 것보다 sr을 2배로, 넉넉하게

In [12]:
```
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)
```

Out[12]:



F0= 100Hz
Fend: 제일 마지막 frequency(sampling rate의 반, nyquist frequency)(int로 반올림 처리)
time은 이미 위에서 만들어짐
theta=time*2π*frequency -> 사인의 입력이 되는 theta 만듦
s(signal- sine wave)는 계속 더해지고 업데이트 됨(+tmp)
다음 for loop- range 부분이 핵심(F0부터 Fend까지, 100만큼(F0만큼))- 100부터 5000까지 frequency에 100, 200, 차례대로 들어와 50번 loop 돎
제일 처음(F0)에는 s가 정의되지 않기 때문에 error가 뜸-> 처음 s가 뭔지 위에서 정의해줘야 함(s=np.zeros(len(t))- 제일 처음에 해당되는 s는 0, 위에서 만들어진 time 벡터의 개수만큼 0을 만들어줌)
100Hz부터 5000Hz까지 다 더해지면 s를 plot함

그래프- pulse train
pulse train: sine wave같은 부드러운 부분이 없어짐- sr이 많으면 선-0-선-0의 반복이 될 것-> pulse train이라 부름
그래프: nyquist까지 더했을 때의 그림

191114

wave form- 여러 개 쌓여 성대에서 나는 소리가 됨-> 더해져서 pulse train 형태가 됨(x축 time, y축은 value)
신호를 frequency domain에서 볼 수도 있음- spectrogram(x축은 time, y축은 frequency)- 어떤 frequency 성분이 많은지 보여줌
spectrum은 한 순간을 보여주고, 이를 시간 순으로 나타낸 것이 spectrogram
spectrum에서 amp는 frequency가 높아짐에 따라 decrease
그 다음 spectrum에 산맥을 만들어주는 작업을 함

phasor: parameter setting-> time 만듦-> 각도 만듦-> 사인함수 만듦-> sine wave 만들어짐
sine wave를 complex phasor로 바꿔줌- 사인과 똑같지만 complex number 형태로 나옴(복소수)

사인과 코사인 때문에 소리가 바뀌지는 않음- 코사인에서 사인은 $\frac{\pi}{2}$만큼 차이남, 90°만큼 옮겨준 것이므로 소리가 달라지지는 않음(e.g. 같은 a, 라 소리)
우리 귀가 phase(각도)에는 sensitive하지 않음, frequency에는 sensitive

```
In [79]: 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
```

function의 내용은 몰라도 됨- function을 쓰는 방법만 알면 됨
in[79]: function을 만드는 방법
def라고 적고 function name을 적음, 괄호 열고 내가 쓰고 싶은 입력을 콤마를 이용하여 적음- 첫 번째 function은 입력 2개, 두 번째는 입력 3개
return: 출력
srate: sampling rate / F: frequency / BW: bandwidth

```
In [16]: 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)
         s = lfilter(b, a, s, axis=0)
         ipd.Audio(s, rate=sr)
```

Out[16]: 0:00 / 0:00

아까 만든 두 번째 함수에서 첫 번째 함수도 불러옴- hz2w는 내부에 들어감
resonance 함수를 쓸 때 첫 번째 입력으로 sampling rate를 넣으면 됨(sr e.g. 10000)
RG: 산맥의 위치 적음- frequency 값
BWG: 산맥이 얼마나 뾰족한지- width가 작으면 산맥이 뾰족하고, width가 크면 산맥이 뭉툭함(뚱뚱)
RG=0 -> frequency가 0인 부분을 중심으로 하여 완만한 산맥을 만들어라(0이 제일 높고 양쪽으로 decreasing-> 그래프를 보면 0부터 decrease하는 것으로 보임(마이너스는 생각할 필요 없음))

```
In [127]: RG = 500 # RG is the frequency of the Glott
          BWG = 60 # BWG is the bandwidth of the G
          a, b=resonance(sr,RG,BWG)
          s = lfilter(b, a, s, axis=0)
          ipd.Audio(s, rate=sr)
```

```
In [128]: RG = 1500 # RG is the freque
          BWG = 200 # BWG is the ban
          a, b=resonance(sr,RG,BWG)
          s = lfilter(b, a, s, axis=0)
          ipd.Audio(s, rate=sr)
```

first formant: 500
BWG: 60(꽤 홀쭉한 것)
second formant: 1500 / third formant: 2500 / fourth formant: 3500

191119

행렬: 직사각형처럼 생겨 숫자가 들어감
벡터: 행렬 중에서도 가로로 한 줄 혹은 세로로 한 줄짜리로 되어있는 것

기계: 인공지능(함수)
앞부분과 뒷부분의 데이터의 형태는 벡터(숫자열)로 되어 있어야 함: 벡터-> 기계(행렬의 형태)-> 벡터
앞부분 text-> 기계-> 뒷부분 음성으로 나옴: 음성 합성
일본어 text-> 기계-> 한국어 text: 기계 번역
중간의 기계는 행렬의 형태를 지님
인공지능(기계, 함수): 행렬의 곱, 입력 벡터를 출력 벡터로 바꿔주는 함수의 역할(입력, 출력 벡터는 음성, 텍스트, 이미지 등 가능)
행렬의 곱셈을 하기 위해 데이터는 벡터의 형태가 되어야 함
선형대수(linear algebra): 행렬

matrices
행렬은 직사각형 형태로 생김
dimension: 차원, 행렬의 크기를 말할 때 사용됨
m행 n열(m*n 행렬)

벡터: 행렬의 일종
1 by n 행렬
벡터는 가로로 길게 있든 세로로 길게 있든 sequence of numbers
$\begin{bmatrix}3\\4\end{bmatrix}$처럼 길게 생긴 벡터: column vector / 그래프는 기하
$\begin{bmatrix}3\\-1\\2\end{bmatrix}$: 3차원, 차원이 늘어난다고 점이 늘어나진 않음(하나로 표현됨)

vector multiplication
$0.5\begin{bmatrix}6\\8\end{bmatrix}$=(3,4)
3차원 상에서도 가능 (3,-1,2)

vector addition
$\begin{bmatrix}3\\4\end{bmatrix}+\begin{bmatrix}2\\1\end{bmatrix}$ = (5,5) -> 그래프에서 확인 가능
3차원- 평행사변형 그려서 확인 가능

vector spaces

linear combinations(중요!): v와 w는 차원이 같은 두 벡터, c와 d는 scalar(단순한 숫자)
단순히 곱하고 더하기만 했을 때의 값이 linear combination
벡터 스페이스: 무수히 많은 여러 벡터들이 만들어내는 공간
무한히 많은 벡터를 만들었을 때, 그 벡터 스페이스가 차지하는 공간은 차원 전체
2차원에서 1사분면만 덮어진 것은 하나의 완성된 스페이스라고 할 수 없음- 모든 것을 커버하지 못함
벡터 스페이스는 1차원의 모든 공간, 2차원의 모든 공간, 3차원의 모든 공간...
n차원의 공간은 n개의 component(3차원의 공간은 3개의 component)가 모두 채워져 있는 공간

column space: column vector로 모든 공간을 채울 수 있음
(e.g. $\begin{bmatrix}2\\1\end{bmatrix}, \begin{bmatrix}-1\\3\end{bmatrix}$ 두 column vector를 찍어 생긴 두 점에 특정 값을 곱하고 더해서 나온 값의 점을 찍고 이를 무한히 하다보면 모든 공간 채워짐)
column space는 column vector보다 높은 차원일 수 없다(2차원)
원점과 두 칼럼 벡터(같은 선상에 있지 않은 independent한 벡터)를 연결하면 삼각형이 만들어짐- 평면 위에 있고 쭉 확장될 수 있음-> column space
whole space: 벡터 자체가 갖고 있는 dimension, 2차원 그대로
column space도 그것을 다 채움-> 2차원
같은 선상에 있으면 dependent-> 아무리 곱하고 더해도 같은 선상에만 있어 whole space를 다 쓰지 못함, column space는 1차원에 불과
whole space의 dimension은 몇 개의 row가 있는지 찾으면 됨
column space의 dimension은 몇 개의 independent한 column이 있는지 찾으면 됨

whole space는 3차원인데 column space는 2차원- P

직사각형(3*2의 행렬)도 얘기할 수 있음
column vector가 2개(3차원 속에서 찍힘)- whole space는 3차원, independent한 column은 2개라 column space는 2차원

transpose: 행렬을 뒤집음
2 by 3 행렬이 됨
whole space: column vector가 2차원
column vector가 2차원 상에서 3개가 있음- 벡터의 개수가 몇 개든지 간에 column vector가 들어가는 공간은 무조건 2차원
transpose하면 whole space는 달라지지만 column space는 여전히 2차원

four spaces in a matrix: column vector의 관점에서 whole space, column space, row vector의 관점에서 whole space, row space

m by n matrix가 있다면 whole space는 2개

(1,2,3)과 (2,4,6)의 column space는 1차원- dependent하기 때문
-> whole space는 3차원인데 column space는 1차원, 나머지 빈 공간(2차원)은 null space (의미 없는 공간)
null space: whole space에서 column space를 정의하고 난 나머지, 어떤 행렬이 있을 때 무엇을 곱하든지 간에 반드시 0이 되는 모든 가능성이 되는 공간
null space는 column space와 row space 모두에 존재할 수 있음

column space: column의 whole space보다 같거나 작음(r)

column vector가 얼마나 dependent하고 independent한지로 파악- column space

Linear transformation- 차원도 바꾸고, 숫자도 바꿈
행렬은 대문자로, 벡터는 소문자로 씀 (x가 입력 벡터, b는 출력 벡터, A라는 행렬과 곱해져서 출력됨)
입력 벡터와 출력 벡터의 차원은 항상 같을 필요는 없음, 달라질 수도 있음- 그 역할을 해주는 것이 행렬
A는 x를 b로 바꿔주는 transformation matrix

inverse matrix
example 3: 함수에 넣어 선상으로 합쳐진 벡터를 역함수하면 점이 원래 자리로 돌아갈 수 없음(not invertible)

eigenvector(중요!!)

191121

행렬 곱하기에서는 인접한 두 숫자가 같아야 곱해짐(e.g. $x\times2$, $2\times y$, 둘을 곱하면 $x\times y$의 형태가 됨)
transpose- 결과값도 transpose됨($Ax=b$, $x^TA^T=b^T$)

columnize- 세 점을 찍으면 plane(column space) 위의 삼각형이 됨, 이를 무한대로 밀면 결국 전체를 커버하게 됨-> spanning, 이 때 column space(2차원)는 whole space(3차원)와 다름
남은 1차원은 null space(선의 모양)

spanning: linear combination으로 표현가능한 모든 것

(노트)row vector의 whole space: 숫자를 2개 쓰니까 2
row vector가 만드는 space는 2차원을 넘어가지 않음- whole space가 2차원이라 2차원보단 작거나 같아야 함

rank: independent한 column의 개수
row vector는 3개
independent한 것은 나머지 점들의 linear combination으로 만들어질 수 없는 것이어야 함
column으로 보든, row로 보든, independent한 벡터의 숫자는 같음- rank(노트에서 2개)
column space의 개수=row space=independent
남는 것은 null space, null space는 column과 row 각각 다를 수 있음

linear combination: 어떤 두 벡터를 앞에 scaling 해서 더하는 것
linear combination 하면 여전히 spanning된 space 안에 있음

column space: 어떤 matrix의 column vector들이 spanning해서 만들어내는 space, column vector들의 모든 linear combination으로 만들 수 있는 공간
subspace: whole space에 포함되어있다는 말
row space와 column space의 차원은 항상 같아야 함(e.g. 6x7의 matrix에서 column space가 3차원이면 row space도 3차원)
-> independent한 column vector가 3개, row vector도 3개-> rank 3, span된 column space도 3

right, left null space는 x가 A의 오른쪽에 붙는지 왼쪽에 붙는지에 따라 결정됨

detransformation: 역행렬, 역함수로 표현됨

eigenvector
어떤 행렬의 eigenvector는? 어떤 vector를 transform했을 때, transform 전의 점과 원점과 일직선상에 놓일 때의 vector

191126

(노트)A의 row 부분이 x의 whole space가 됨- rowize(3차원), row vector 2개가 만들어냄 ($[1\,5\,3], [2\,6\,-1]$)
이 때 row vector가 만들어내는(rowize) whole space는 3차원-> x도 3차원이 됨
두 점이 spanning해서 만들어지는 row space는 2차원(independent), 3차원은 될 수 없음- 점이 두 개이기 때문

null space의 정의: Ax = 0, given A에 대해 결과로 0을 만드는 모든 x를 찾아라 (e.g. null space가 1차원이면 x는 그 선상에 존재함)
null space는 어떤 입력이 들어오든지 간에 output, 출력에 영향을 끼치지 않음, 효과가 0
task를 위해 중요한 공간은 아니지만, 그 공간을 확대시킴, output에는 영향 없음
특정 방향으로의 변환은 출력에 영향을 미치기도, 미치지 않기도 함- null space와 그렇지

않은 space의 차이(null space는 입력이 조금 바뀌어도 전혀 영향을 주지 않는 공간)
null space: Ax=b가 있다고 할 때, x를 조금 변환시켜도 b에 전혀 영향을 끼치지 않음- x와 null space가 평행하도록 하면 됨

(1,0), (0,1)에 해당하는 matrix- 어떤 입력이 들어가도 그대로 나옴

아이겐 벡터는 입력, 출력과 원점이 평행하는 순간이 생김- 벡터는 값이라기보다는 어떤 plane에서의 방향(물리학적 개념), 반드시 같은 선상을 따라갈 필요는 없음
e.g. 주어진 행렬(2*2)의 eigenvector는? 어떤 특정 방향 전체(선상), 어떤 값 하나만 존재한다고 해도 되긴 하지만 여러 가능한 eigenvector의 집합이 있음- eigenvector space
2*2에서 eigenvector는 두 개가 있음(선이 2개), 각각의 eigenvector에 대해 eigen value가 있음- eigenvector 위에서 확대하고 축소해도 점 간 길이의 비율이 똑같음, eigenvector가 두 개면 value도 두 개

null space의 목적: b라는 task를 하려는데 존재하는 장애물을 피해가기 위한 공간, 입력이 많이 변해도 같은 결과를 내기 위해 필요한 공간, 유니크하고 고유한 것으로 바꿔줌(주어진 matrix는 여러 개가 섞여있지만 eigen analysis를 통해 그것을 쪼개고 고유한 것으로 바꿈)

상관관계(correlation): 같이 가는 느낌(e.g. 영어 점수 오르면 국어 점수도 오름), r이라는 수치값으로 표현할 수 있음($-1 \leq r \leq 1$, 0일 때 상관관계가 제일 작음(그래프가 완전한 동그라미))
음의 상관관계: e.g. 커피와 수면량
1과 -1이 정확히 나오는 순간: 완전한 선상에 있을 때

inner product(dot product): 각각 다 곱하고 안으로 해서 다 더함
기하학적 의미- 한 벡터에서 다른 벡터로 수직으로 내리고 길이를 곱함
correlation이 많으면 inner product 값이 높게 나옴

191128

eigen vector: A라는 행렬이 있고, 그것을 곱해주는 값인 v가 있다고 하고, 결과값인 Av가 원점과 일직선이 되는 모든 벡터
eigen analysis 식: $Av=\lambda v$ (transform된 결과(Av)가 transform 되지 않은 벡터(v)의 상수배)
원점, v, Av가 전부 일직선상

λ(람다): eigen vector를 expand 해주는 ratio(eigen value), 그냥 상수

inner product(dot product): 중요! 기하로 먼저 이해해야
어떤 두 벡터가 있다고 할 때, 원점과 삼각형을 이룸(원래 벡터의 차원과 상관없이 세 점이 2차원의 평면을 이룸)
값을 작게 만듦(차원이 줄어듦), 무조건 1x1이 나옴(한 개의 값)
cf) outer product는 값을 크게 만듦

cosθ 아는 법: 노트
3차원 상에서 원점으로부터의 거리: e.g. 루트(1^2+2^2+3^2)
cos90=0: correlation이 없음, inner product로 생각해도 마찬가지- projection 시켜도 0, 0이 될 수밖에 없음

두 개의 벡터를 주고 cosine similarity 구하는거 시험에 나옴(cosine similarity- a와 b가 얼마나 비슷한지 수치적으로 얘기해줌)
cosine similarity: cosθ, correlation r과 거의 유사

100Hz, 200Hz, 차례로 올려나감- sine wave 만듦(frequency 값들은 phasor)
100Hz가 spectrogram에서 제일 밑에 있음, frequency가 높을수록 위에 있음

a와 b: 완전 correlate, 따라서 a×b의 값은 큼
a와 c: a가 올라갈 때 c가 떨어질 수 있음, 불일치 존재-> a×c의 값은 a×b보다 작을 수밖에 없음
복잡한 wave에 같은 성분이 inner product되면 반응이 크고, 똑같은 성분이 없거나 적으면 inner product 값이 낮게 나옴

차이가 90° 날 때 inner product: 0
frequency는 똑같아도 90°를 이동(phase shift)했기 때문에 0이 됨

inner product를 써서 어떤 주파수가 많은지 알아내는게 목표
phase에 민감하게 작용하지 않도록 해야함
-> sine, cosine이 아닌 complex phasor를 써서 inner product를 함(사인, 코사인은 민감도가 너무 높음)

191203

1×8 과 8×1의 벡터가 곱해져서 1×1의 scalar가 나옴 -> 1×n과 n×1을 곱하면 1×1이 나옴(하는 이유: 서로 얼마나 비슷한지 보기 위해)

n차원의 공간에 a라는 벡터와 b라는 벡터가 있을 때, a와 b벡터의 원점으로부터의 길이가 고

정되어있으면 angle(θ)에 의해 dot product의 크기가 결정됨- angle이 0에 가까울수록 값이 큼, 90°에 가까워질수록 0이 됨

wave도 벡터, wave 속에 어떤 sine 성분이 많은가 아는 것이 중요- 어떤 한 시점만 분석하면 spectrum, 그것이 시간적으로 이어지면 spectrogram
spectrum analysis: 어떤 frequency 성분이 많은가- 느린 frequency vs 빠른 frequency, 무수한 frequency의 합에서 그 중에도 어떤 성분이 얼마나 많은지 analysis
inner product의 기법을 씀- 여러 종류의 sine wave를 frequency별로 만들어서 복잡한 complex signal에 inner product를 함, 그 값을 plot하면 spectrogram

wave가 90° 이동하면 0이 될 수 있음(target wave에 대해 probe를 하는데 이 target이 shift되는 것에 따라 probe되는 결과값들, 즉 inner product가 변할 수 있음- phasor sensitivity가 너무 큼-> 따라서 쓸 수 없음)
-> complex phasor를 씀(phasor에 대해 sensitive하지 않음)

complex phasor로 만든 값들은 허수를 포함한 complex number가 나옴- 벡터이긴 하지만 complex number가 됨-> 곱하기는 할 수 있지만 inner product로 나오는 값이 complex 값으로 나옴
inner product할 때는 두 개의 dimension이 똑같아야 함(e.g. 1×n, n×1)
inner product된 결과로 complex number가 나오면 plotting이 불가능-> a+bi에 절댓값을 씌우면 실수값이 됨
즉 기존에 있는 wave(실수 벡터)와 complex phasor를 inner product-> 그 결과 complex number가 나오고, 여기서 a+bi에 절댓값을 씌우면 실수값이 됨
a+bi의 절댓값: complex plane의 원점에서 (a,b)의 거리
-> 이를 알면 각 frequency 별의 에너지가 어떻게 되는지 알 수 있음

s라는 소리가 만들어졌다고 가정-> 이후 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)
```

scipy.io에서 wavfile을 쓸 수 있도록 import(외부에서 wav 파일을 가져오고 싶을 때 사용, 저장된 a.wav라는 파일이 있을 시에- 우리는 위에서 만들어진 s를 쓰기에 사용하지 않음)
s: 500, 1500, 2500의 formant가 있는 소리
s 벡터의 사이즈: nSamp (샘플 개수 알아냄)

```
In [133]: nFFT = nSamp
          amp = [];
          for n in range(0,nFFT):
            omega = 2*np.pi*n/nFFT # angular velocity
            z = np.exp(omega*1j)**(np.arange(0,nSamp))
            amp.append(np.abs(np.dot(s,z)))
```

샘플 개수가 중요

분석하고자 하는 s가 target wave- target wave의 사이즈, 즉 샘플 개수 파악, nFFT로 이름 바꿈

이후 샘플 개수만큼 for loop를 돎

z로 시작하는 부분: complex wave를 만드는 부분(complex phasor 이용)

omega: ω / 1j: i

: 지수, 뒷부분이 앞부분의 지수로 올라감(e.g. a3 = a^3)

for loop 식에서 omega, z 부분 잘 알아둬야 함

z: complex wave를 loop의 개수만큼 만듦

z는 0번째부터 nSamp-1번째까지 있음, 이 여러개의 z를 s와 각각 inner product-> 그것에 대한 값이 나옴, for loop할 때마다 값이 다름(e.g. s와 첫 번째 만들어진 z를 dot product, 두 번째 loop에서 만들어진 z를 dot product...)

abs: 절댓값(absolute), dot product에서 complex 값이 나오더라도 abs를 취하면 원점에서부터 a+bi까지의 길이로 나옴(크기값을 알 수 있음)-> 이후 amp.append

amp.append: amp라는 variable에다가 값을 하나씩 append함, 다 하면 루프의 개수만큼 amp의 크기가 나옴- for loop이 끝나고 난 후 len(amp)의 답을 낼 수 있음(시험에 나올 수 있음)

amp에는 허수가 들어갈 수 없음- abs했기 때문- 시험에 나올 수 있음

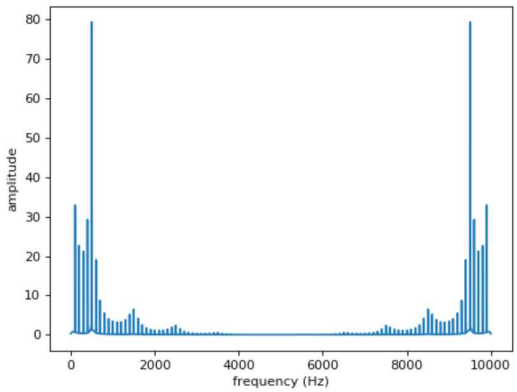
```
In [134]: fig = plt.figure()
          ax = fig.add_subplot(111)
          freq = np.arange(1,nFFT+1)*sr/nFFT;
          ax.plot(freq, amp)
          ax.set_xlabel('frequency (Hz)')
          ax.set_ylabel('amplitude')
```

freq: x값

amp: y값(amp값은 위에서 이미 정해짐), 각 frequency 성분에 대한 에너지 값

y값은 알지만 x에 해당되는 것은 모름- 알아내야 함

nFFT에 +1해야 마지막까지 포함됨-> [1...100]×10000/100



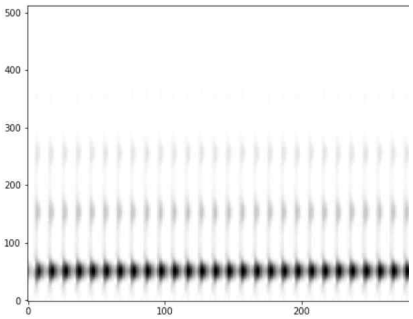
spectrum
x축: 0부터 10000까지(sr=10000, 10000까지가 range)
bar: inner product를 abs한 값
총 bar의 개수는 sample의 개수(100)와 같음- 시험에 나올 수 있음
그래프의 왼쪽 오른쪽이 같음, 대칭- sampling rate의 half만 의미가 있는 것과 일맥상통, 10000까지라 치면 5000까지만 의미가 있음, 그 이후는 중복되어 의미가 없음(5000부터 10000까지는 필요 없는 정보)
그래프의 half(0부터 5000까지)를 spectrum이라 부름- spectrum은 각 frequency 성분들이 inner product해서 받아오는 에너지 값, inner product의 absolute 값
500, 1500, 2500에 산맥 나타나는 것 확인 가능- formant
시간축이 아닌 주파수축- 고정 time에서 어떤 성분이 많은 것인지 확인, 바뀌는 시간도 확인할 수 있는 것은 spectrogram

191205

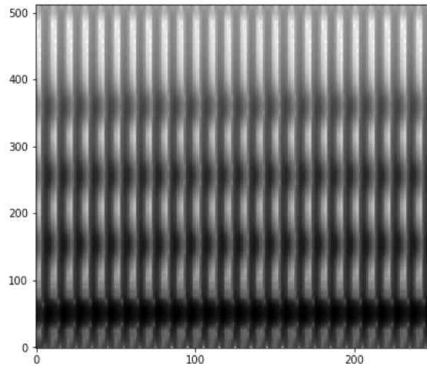
dot product: a와 b 점, 원점을 연결했을 때 나오는 각도, 그것을 코사인에 넣으면 나오는 r(correlation)
90도: orthogonal, 코사인 그래프에서 0이 됨(cosθ = r -> cos90 = 0)
r=1일 때: a벡터와 b벡터가 일직선상에 놓인 경우(cf- 기울기가 반대로 되면 r이 -1이 됨)
10차원에 들어가는 영어점수, 국어점수(각각 점 하나씩): 한 축이 subject가 됨, 10차원의 한 벡터(점)은 10개의 숫자를 가짐-> 10차원이 됨, 영어점수와 국어점수라는 두 벡터
이 때 영어벡터와 국어벡터의 각도가 0이 되면-> r=1, 일차함수 그래프와 비교하는 것 시험문제
cosθ- cosine similarity

phase shift시킨 그래프- 180도 만큼 차이가 남, 한 그래프가 올라가는 모양이면 다른 그래프는 내려가는 모양, 한 그래프는 진폭이 크고 다른 그래프는 작아도 r은 여전히 -1(축이 subject인 그래프에서 길이=진폭)
cos180 = -1

```
In [101]: max_freq = None # cutoff freq
          win_size = 0.008 # sec
          win_step = 0.001 # sec
          win_type = 'hanning' # options: 'rect', 'hamming', 'hanning', 'kaiser', 'blackman'
          nfft = 1024

          # Emphasize signal
          s = preemphasis(s)
          # Frame signal
          frames = frame_signal(s, sr, win_size, win_step)
          # Apply window function
          frames *= get_window(win_size, sr, win_type)
          print('frames:', frames.shape)
In [102]: magspec = np.abs(np.fft.rfft(frames, n=nfft)) # frames x (nfft//2
          plot_spectrogram(magspec)
Out[102]: (<Figure size 1008x432 with 2 Axes>,
           <matplotlib.axes._subplots.AxesSubplot at 0x1c2140e358>)
```




```
In [103]: powspec = 1/nfft * (magspec**2)
          plot_spectrogram(powspec);
In [104]: logspec = 10 * np.log10(magspec) # dB scale
          plot_spectrogram(logspec);
```



sampling rate중 표현할 수 있는 것은 반밖에 안됨-> 5000Hz까지 밖에 안 됨(3000=7000, 대칭의 형태)

spectrogram에서 색깔이 진한 부분이 산맥이 올라가는 부분

y축- 아래쪽은 저주파, 위쪽은 고주파

win_size, win_step: 여러 spectrum을 만들고 그것을 다 합쳐 spectrogram을 만들 때, 얼마만큼의 벡터를 가지고 dot product를 할지 사이즈를 정함

저주파는 강하고, 고주파는 약함- 첫 번째 나오는 spectrogram에 반영됨(위로 올라갈수록 연해짐, complex number로 dot product 되는데 이에 절댓값을 취해서 나온 값들)

-> 위에도 강해질 수 있게 제곱을 시켜 처리해줌(in[103])(**로 제곱해줌- 로그를 취하기 위해(in[104]), 아주 작은 수나 큰 수가 로그를 취하면 적당한 수로 떨어짐)

-> 골고루 분포된 spectrogram이 나옴, 위로 올라갈수록 연해지긴 하지만 산맥이 저주파에서 고주파로 갈 때 급하게 낮아지는 것이 좀 덜해짐