영어음성학
Hz가 높을수록 고주파, 무지개 중 보라색 쪽, 빠른
빨간색, 느린,
검은색 부분이 많을수록 성분이 많다.

9.17 화
English consonants and vowels
(ex) gap에서 자음은? 그　철자와 소리의 구분
year 은 ear에 자음y를 붙인 것

둘이 pair 다
shy she - 혀 닿으면 안됨 --- 무성
vision - 혀가 닿으면 안된다 --- 유성

혀가 닿았다면 jive 와 유사
j(y) yearn youte? year 여 소리 발음기호 j라고 쓸 때도 발음기호로 여 소리
year과 ear이 같으면 안됨 이얼　여 는 자음이다

jive 소리에서 떨림을 뺀다고 생각하면됨
cheap chime 은 목이 안떨리지만
jive는 목이 떨린다

목이 떨고 안떨고
목이 떠는 것 - 유성음 voiced sound , 와 voiceless
모음은 모두 voiced, 모음은 모두 유성음이다.
자음 중 vocied - b d g m n n v　z vision l w r j　맨 밑 voiced
코로 나오는 소리 - m n n
p b　입술 두 개 쓰는 것
f, v 아랫입술과 윗니
혀와 윗니 - thigh thy

모음
단모음과 복모음

인지적인 과정이 phonology - 상위, 고정되어 있는
phonetics- 물리적이고, 늘 차이가 있는 physical
a study on speech　- speech는 사람이 하는 말에 대한 모든 연구
a study on sound system - phonology

연속적인 소리 - phonetics
성대의 떨림의 정도에 따라 - 소리의 높이
소리가 바뀌는 것 - 입모양(혀의 위치, 턱)
턱의 높낮이 때문에 소리가 바뀌는 것 - 주된 요인은 아닐 수 있다
한글은 음절의 반복, 영어는 stress를 기본으로해서 반복이 된다
한국어는 턱이 말할 때에 맞추어서 간다. 한국어는 턱을 많이 쓰는 language
턱과 반대로 가는 것이 혀, 반면에 stress를 rhythm으로 생각하면 혀를 더 자유롭게 쓸 때
1.지금까지 articulation
2.공기를 타고 가는 과정을 acoustic - 공기가 어떻게 움직여가는지
공기와 소리가 어떻게 되는가 - 물리, 사람이 수반되지 않은 물리
3.auditory - 공기를 타고 오는 소리를 귀로 듣는 것
귓바퀴는 소리를 증폭시키기 위해
고막이 움직이는 것까지 물리 - 청각세포가 신경으로 전달
<Articulation>
The vocal tract
upper structure은 고정
인강(목젖부터 후두까지 있는 긴 관), 후두
palate 입천장
alveolar, hard palate, soft palate(velum), uvula, pharynx wall, larynx 알아야함

lower - lip blade tip front center back root, epiglotis
tract는 관 - 식도와 기도 - epi(덮개) glotis - 기도로 가는 문을 막는 과정
음(소리) oral tract은 막히고, nasal tract은 열려있는 상태
아(소리) nasal tract을 안쓴다 - 닫아 놓았다 - velum은 lower or raised 두가지 밖에 없는데 올라가면 막히고 내려가면 열린다
velum이 raised됐습니다 nasal tract은 열렸을까요 닫혔을까요? - 막혔다 -
velum이 raised 되면 nasal tract은 막힌다.
소리의 예는 모든 모음, 자음 중에는 비음을 뺀 모든 자음들 음 은 응 빼고는 음은 응 은
velum이 lower되서 소리가 난다
숨을 쉴 때 velum이 raised 될까 lower가 될까 - lower가 된다, nasal tract은 열린다

---

nasal tract이 열리면 velum 은 lower
nasal tract이 닫히면 velum 은 raised

---

nasal sound와 그렇지 않은 sound를 구분하는 - oro-nasal process

(phonation process)
모든 영어의 소리는 유성음과 무성음으로

유성음은 모든 모음과 일부 자음

(phonation process in larynx)
phonation은 voiceless와 voiced 구분 짓는

articulation 유성 무성(성대) , velum에서 lower and raised, 혀 부분에

m,n,ng 이외에 nasal tract 이 막혀있음
숨을 쉴 때 nasal tract 열리고 velum이 내려감 (소리가 코로 가야하니까)

아파 아타 아카
velum larynx articulation (3개의 주 요인)
lips, tongue tip, tongue body = constrictors. 협착을 만들어내기 때문
degree=상하, location= 앞뒤, 이 요소들에 의해 더 자세히 specified됨

*d 는 lab 이자 stops 폐쇄음 (공기가 완전히 막힘)
ex. ㅍ ㅌㅋ, m,n,ng 도 stops임 oral tract에서 만큼은 닫혀있으니까
*s 는 fricatives 마찰음
ex. z, f ,th (뱉고나서도 음이 계속 유지되면 fricative. 유지 안 되고 막히면 stops)
*vowels 은 무조건 자음보다 막힘이 없음 degree가 작음
*approximants -r, l, w. j[여] (접근음)

1. specifying constictors
2. constrictor degree
3. constrictor location
4. velum 올리느냐 내리느냐
5. larynx를 여느냐 닫느냐

예시문제-velum raised, larynx의 그 틈을 glatis라 하는데 이게 완전히 오픈되어있고
const는 tongue tip, cd- lab , cl- stop = ㅌ

*모든 모음은 const 로 tongue body만 쓴다

예시문제 - 모음과 같은 const 쓰는 자음의 예를 드시오 =ㅋ
그 자음중에 velum 이 lower 되면 ㅇ

m 소리 larynx가 닫혀야함. 그래야 닫힌 채로 소리가 진동되니까.

가로로 긴 줄 4개가 formant. f1,f2 가 뭐냐에 따라서 모음이 뭔지를 결정함.
모음을 구별하는 수치임. 첫 번째 formant 800, 두 번째는 뭐시기. 결정됨

vowel acoustics
praat 가장 큰 파도 larynx
hz- 1초에 127번이 떨린 것
127.358 * 0.007852 =1

숙제 - youtube 들어가서 namz 채널 들어가서 구독 누르기
github 별표 누르고 자기 정보 이메일로, 학과 학번 이름 계정이름까지!
지금까지 했던거 요약해서 한페이지로 정리해서 깃허브 계정 사이트에 업로드하기 2019 fall (반드시 pdf 파일로!)
summary.pdf 에다가 계속 추가해서~

9/24

larynx, velum, 입에서의 과정. 세가지 combination으로 언어의 자음과 모음을 구분

constrictor
1.Lips
tongue tip
tongue body
2.velum
3.larynx

/p/라는 소리는 lips로 specified. cl- bilabiel. cd- stops.
velum- raise (lower 되는건 m,n,ng밖에 없음) larynx- voiceless니까 open
/b/로 바뀌면 larynx- closed
/d/ -> tt-> ALV , cd- stop
/z/ -> tt-> ALV, cd- fricative
/n/ -> tt-> ALV, cd- stop larynx- closed, velum-lower

sine wave- 1초가 몇 번 반복되느냐에 따라 결정됨,

가장 기본적인 signal 형태, pitch, frequency, magnitude 에 따라 sine wave가 정해짐. 이 세상에 존재하는 모든 signal은 여러 다르게 생긴 sine wave의 결합으로 표현된다. 19세기에 발견된 중요한 발견.

첫 번째~세 번째 (simplex tone) x축은 시간 y축은 value 혹은 voltage.
time value 그래프를 frequency amplitude 그래프로 변환할 수 있어야 함.
frequency amplitude 그래프는 다른 말로 SPECTRUM
음악 이퀄라이저도 complex tone in spectrum 임.

첫 번째 그래프. 100hz. 1초에 이런 움직임이 100번. frequency가 작음 (slow. 저음)

magnitude wide가 젤 큼
두 번째 그래프, 200hz, 1초에 200번. 첫 번째보다 2배 빠름.
세 번째 그래프, frequency가 높음, magnitude는 젤 작음
마지막 그래프(complex tone)- 위 세 그래프의 합을 나타낸 것. 여러 웨이브의 합은 sine wave가 아닌 복잡한 신호 혹은 소리가 됨. 단순한, 다양한 sine wave 들로 표현될 수 있다.
첫 번째 그래프처럼 1초에 100번 반복됨.

마지막 spectrum 해석법- 100hz짜리가 저정도 크기로 있구나. 200hz짜리가~, 300hz 짜리가~. 1+2+3 = 6 (synthecize, 합성), 6을 쪼개서 1,2,3찾는게 spectrum analysis
pure tone= simplex tone

praat- 440 hz 에 amplitude 1로. value의 점이 sine wave을 구성하고 있는데 최고값 최소값 1 , -1. 1초에 440개 들어감 sine wave 하나당 1/440 초.

spectrogram이라는 것은 spectrum을 time 으로 쭈루룩 visualize 한 것. 한 given point에서 어떤 frequency 가 많은지.

일정부분 선택해서 spectrum- view spectral slice. x축 frequency.
440짜리 하나 발견. 만약에 소리가 다양하면 다른 hz 의 그래프도 나올 것
pure tone이 아닌 아- 소리를 slice 로 분석하면 각 점의 간격이 완전 똑같음.
제일 첫 번째로 젤 높은 지점은 126.68hz. 두 번째 지점은 그것의 두배인 254hz.
각 sine wave의 간격이 똑같다는 것. 내 목소리도 126.68hz임.
반복되는 패턴의 제일 낮은 주파수와 내 목소리가 같음. 아까 봤던 세 그래프에서도 제일 낮은 simplex 그래프의 hz와 complex 그래프의 간격이 똑같았던 것처럼.
여러 다른 simplex톤의 합으로 이루어지는데, 제일 slow한 simplex톤의 frequency가 우리말의 frequency, 음높이와 동일하다. 성대가 1초에 몇 번 떨리느냐와도 일치하고.

아,이 는 입모양의 차이에서 발생하는 소리.
성대에서 떨리면서 소리가 난다 했는데 그걸 그대로 캡쳐햇을 때 그 소리를 source라 함.
말그대로 우리가 입모양을 가진, 머리뚜껑을 성대에 얹었다고 생각해보자
source에서 filter를 어떻게 바꾸느냐에 따라 '아'소리, '이'소리 만든다고 생각하면됨.
모든 사람의 source의 패턴은 모두 decrease 함. 첫 번째 나오는 것을 f0. fundamental frequency라 함. f0값이 나의 hz. harmonyx- f0에서 계속 곱하기 1, 2, 3 되는 것을 뜻함
115hz. 230hz, 345hz... 이 거리가 여성의 had 면 첫 시작이 남자보다 더 높음. 그래프 간격이 더 넓어서 더 듬성듬성한 그래프, 즉 spectrum이 생성됨. 따라서 남자께 여자것보다 한 소리에 들어있는 그래프 수가 더 많음.
사람의 pitch 는 f0값과 동일

wave와 쌍을 이루는 spectrogram. spectrogram의 x축은 time, y축은 frequency.
밑에는 까맣고 위쪽으로 갈수록 옅어짐. low frequency로 갈수록 색이 찐해짐(energy 큼)
spectrum에서도 frequency가 낮을수록 에너지 값이 큰 것을 확인할 수 있음

spectrum의 l l l l l l 가
spectrogram에서
-
-

이렇게 눕혀진 것.

첫 번째 그래프- 저주파에서 에너지 높음 / 두 번째 그래프- 등간격으로 harmonics가 유지됨. 중간중간 에너지가 더 많은 부분이 존재
source의 특징)
simplex- sine wave, 그것들의 합 - complex tone
lowest pure tone이 Fundamental frequency(F0)
filter의 특징)
검은색- mountain, 검은색보다 연한 색- valley
어디에 산맥이 나타나느냐가 사람들의 입모양에 따라서 다름.

f1은 height 결정, f2는 backness를 결정.

10/01

computer language들의 공통점? 문법과 단어
모든 lang 은 단어가 있다. 단어들의 combine,
단어는 뭘까? 단어는 의미(정보)를 포함. 정보를 담는 그릇
그 그릇에 여러 가지가 왔다갔다 바뀔 수 있음.
단어들을 선택하고 어디에 위치시킬지, 어떤 순서로 어떻게 조합할지 생각해야 나의 완전한 의미 전달가능.
computer language 단어에 해당되는게 변수임. variable.
사람 말의 문법은 생각보다 많이 복잡함. 기계의 문법은 덜 어려움
1. 변수라는 빈 그릇에 정보를 넣는 것. variable assignment
2. 자동화, 기계화라고 생각할 때 직관적으로 떠오르는게. ~할땐 ~해주세요. 라는 조건이 붙는게 당연히 필요. conditioning 에 대한 문법 필요. If conditioning
3. 자동화의 가장 중요한 점이 여러번 반복하는 것. 여러번 반복하는건 for라는 문법을 씀. for loop
4. 함수를 배워야함. 문법에서 가장 중요한게 함수를 배우는 것. 입력- 마우스 클릭. 출력- 소리 play. 내부적으로 어떤 함수가 복잡하게 들어가 있을 것. 입력과 출력을 packaging하는게 함수. 두 개의 자연수를 주면, 7,10을 넣고 7에서 10까지의 자연수를 합해라.

varaible은 일종의 정보. 정보의 종류가 두가지 밖에 없음. 하나는 숫자, 하나는 문자.

‘= sign’ 은 오른쪽에 있는 정보를 왼쪽에 있는 variable로 assign 한다.
왼쪽은 무조건 variable, 오른쪽은 무조건 정보. 순서가 바뀌면 안됨.

print 라는 함수의 입력은 a, 입력이라고 표시해주는게 괄호.
in[] 누르고 b 누르면 below에 셀 만들어짐, a 누르면 above 에 만들어짐
지우고 싶을땐 x 누르기

a라는 변수에 처음 1 넣고 두 번째로 2 넣으면 2로 renewal 됨.
run 대신 shift+enter 눌러도 됨.

```
a = 1
b = 2
b
c = 3
c
```
하면 3 이 프린트됨

한줄로 다 쓰고 싶을땐 세미콜론 이용해서 이렇게 -> a = 1 ; b = 2; c = 3
프린트 할때도 print(a); print(b); print(c) 이렇게.

```
a = [1, 2, 3, 5]
```
여러개 숫자를 한 변수에 한꺼번에 넣는걸 list 라 함. 대괄호 써서 표현해주기.
list 에는 숫자, 문자 다 들어가도됨. 쉼표로 구분
list 로서의 1,2,3,5 가 a 에 들어있다.
type(a) 누르고 run 하면 list 가 나옴. 이 타입은 list 라는걸 말해주는 것
a = 1 하고 type(a)를 run 하면 int 라 나옴
a= ‘love’ 하고 type(a) 하면 string

list assignment 할 때 대괄호 대신에 괄호 써도 됨.
괄호로 쓰면 a = (1, 'love', [1, 'bye'])
type(a) 하면 tuple 이라 나오는데, tuple 은 list 와 완전 똑같음. 대괄호와 괄호의 차이일뿐.
tuple이 조금 더 보안에 강함.

```
a = {'a': 'apple', 'b' : 'banana'}
```
type(a) 하면 dict
표제어 설명어 표제어 설명어 순.
dictionary에 두 개 넣어놓은 것. list 에 몇 개 들어있는지는 콤마로 구분.
list 와 달리 컬리웅앵웅(중괄호) ‘{ }’를 씀
중괄호를 써야 dictionary, 늘 표제어와 설명어가 쌍으로 따라 다녀야함. pair로.

10/08

a = {"a": "apple", "b": "orange", "c": 2014}
print(type(a))
<class 'dict'>
print(a["a"]) -> 0,1,2 와 같은 인덱스로 접근하는게 아니라 사전처럼 단어로.
apple

---

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

단순히 표제어가 바뀌었을 뿐이므로 똑같이 apple로 나와야함.

하나의 변수에 새로운 값을 넣으면 rewrite, replace 되는 속성

a = [1 , 2]
b = [3 , 4]
c = a[0]+b[0]
c = 4

a = [1 , 2]
b = [3 , 4]
c = a[1]+b[1]
c = 6

두 줄 적을거 한줄에 적으려면 세미콜론 ; 으로 구분하기.

a= 1.2; a = int(a) ; print(type(a))
<class 'int'>
원래 float 인 a의 타입을 int 로 했더니 int 로 결과 나옴

어떤 정보와 내부정보를 들어갈 때에는 대괄호를 쓴다. 그것의 인덱스를 적으면 내부정보를 부분적으로 가져온다.

a = '123' ; print(a[2])
3

---

a = 123, 124 ; print(a[0])

123

---

a ='123'; a=list(a); print(a)
['1', '2', '3']

dict는 pair set에서 앞부분을 인덱스로 쓴다.
dict에 있는 정보 access 할때는 앞부분을 인덱스의 수단으로 쓴다.
말 그대로 사전과 똑같다고 보면됨.
a= [(1,2,3), (3,8,0)]
print(a[0])
(1, 2, 3)
type(a[0])
tuple

---

s = 'abcdef'
n = [100, 200, 300]
print(s[0], s[5], s[-1], s[-6])
a f f a

print(s[1:3]
첫 번째에서 세 번째 직전까지, 즉 첫 번째에서 두 번째꺼까지만 가져옴.
bc
print(s[:3] 세 번째 직전까지
abc
print(s[:]) 전부다
abcdef

n= [100, 200, 300]
print(n[1:2]
200
-> list 와 string은 똑같은 정보 전달 방식을 갖고 있음.

---

len(s)
6
len 은 length
len(n)
3

---

s[1]+s[3]+s[4:]*10
bdefefefefefefefef..(10번)

---

s.upper()
‘ABCDEF’
어떻게 이부분에선 . 만 넣어서 함수같이 쓰면 실행되는걸까?

---

s= ' this is a house built this year.\n'
result = s.find('house')
result
11
house가 11번째부터 시작함
result = s.find('this')
result
1
this가 문장에서 두갠데 1로 나오는 걸 보니 s.find 는 첫번째껏만 찾아주는 함수임

result = s.rindex('this')
result

s = s.strip()
s
'this is a house built this year.'

tokens = s.split(' ')
tokens
['this', 'is', 'a', 'house', 'built', 'this', 'year.']
tokens 는 걍 variable 이름. 다른 이름도 가능.

s = ' ' . join(tokens)
s
'this is a house built this year.'

s = ' , ' . join(tokens)
s
'this , is , a , house , built , this , year.'

s = s.replace('this', 'that')
s
'that , is , a , house , built , that , year.'

in 뒤에 있는 걸 하나씩 하나씩 돌려서 I 가 그 하나씩을 받아서 뭔가를 하라. 뭔가는 for 밑에 적어주면 됨.

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

for i in a:
    print(i)

->
1
2
3
4
```

range 뒤에 어떤 숫자가 나오면 리스트를 만들어줌. 예를들어 4가 들어가면 0부터 3까지 리스트를 만들어줌. 4개의 인덱스를 만들라는 뜻이므로 0부터 0,1,2,3을 만들어줌.
루프를 돌면서 I가 젤 첨에는 0을 받겠죠. a의 I 번째. a의 0번째꺼를 print out 해라.
range 는 0부터 4개의 리스트를 만들어줌. 그것을 이 I가 받아서,,

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

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

for i in range(4):
    print(i)
-> 0,1,2,3

a = [1, 2, 3, 4, 5, 6, 7]

for i in range(len(a)):
    print(a[i])
-> 1,2,3,4,5,6,7
```

len(a)는 0,1,2,3,4,5,6 이 들어가고 이게 I 에 그대로 들어감.
a의 0번째 - 1 , a의 1번째- 2, ... 의 식인것임

```
a = ['red', 'green', 'blue', 'purple']
print(a[0])
print(a[1])
```

```
print(a[2])
print(a[3])

->
red
green
blue
purple

a = ['red', 'green', 'blue', 'purple']
for s in a:
    print(s)
red
green
blue
purple

a = ['red', 'green', 'blue', 'purple']
for s in range(len(a)):
    print(a[s]) # a[s]- a의 몇 번째.
red
green
blue
purple

a = ['red', 'green', 'blue', 'purple']
b = [0.2, 0.3, 0.1, 0.4]

for i, s in enumerate-번호를 매긴다(a):
#variable 이 두 개임 I랑 s. enumerate가 a 인데 번호까지 있는게 쌍으로 있는데 이걸 i랑 s가 받는다.
I에는 0이 들어가고 s에는 red가 들어감
enumerate는 어떤 리스트를 아웃풋 값이 자기자신도 되지만, 그것에 대한 번호도 매겨준다.
그렇기에 변수도 2개 마련한 것임. I 와 s에.
앞에 있는게 번호고 뒤에 있는게 자기자신
```

```
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
```

```
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%
```

for loop가 4번 돎. I,s가 0하고 red를 물고 그 다음줄로 향함.
내가 이런 형태로 format을 하고 싶을때 더블 쿼트“ ”. 중괄호 { } 안에 들어갈거를 .format 하고 s, b[i]*100 라 적어주면 됨.

```
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))
```

zip 함수가 a와 b를 한데로 묶어줌

-- if 함수 --

```
a = 0
if a == 0:
    print("yay!")
->
yay!
```

a == 0, =를 두 번 써야 우리가 아는 부등호임. =는 variable에 값을 부여했던 것,

```
a = 0
if a != 0:
    print("yay!")
```

값 안나옴

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

->

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

```
a = 0
if a >= 0:
    print("yay!")
    print("let's go")
```

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

부등호는 항상 > 나 < 가 = 보다 먼저 나와야함. 이 순서가 틀릴시 작동 안됨

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

->

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

```
a = 0
```

```
if a != 0:
    print("yay!")
    print("let's go")
else:
    print("no")
->
no
```

100번의 루프가 돔. 각 루프에 대해서 50번의 루프가 또 돈다. 그러면 총 5000번의 무언가가 실행됨.
무조건 시험에 나오는 류

```
for i in range(1,3):
    for j in range(3,5):
        print(i*j)
```

for I in range(1,3) 이면 1부터 3이 아니라 1,2 까지 가는 것.
이 for loop는 두 번 돈다.
for j in range(3,5) 이면 3,4
이 for loop도 두 번 돈다.
따라서 총 4번이 돌겠지.

I가 1일 때 j루프가 돌아감. j는 첫 번째께 3.
1*3 이 실행됨.
I가 1일 때 j루프 돌아감 j는 두 번째께 4
1*4 가 실행됨
I가 2일 때 j는 3
2*3 = 6
I가 2일 때 j는 4
2*4 = 8

```
for i in range(1,3):
    for j in range(3,5):
        if j >=4:
            print(i*j)
->
4
8
```

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

```
    if i >=3:
        for j in range(3,5):
            print(i*j)
```

->

값 안나옴.

3이상의 I가 없으니까.