Phonetics(음성학)과 Phonology(음운론)

1. Phonetics(음성학)
- 물리적, 구체적
- speech(사람의 말)에 대한 연구

2. Phonology(음운론)
- 인지적, 추상적
- sound system에 대한 연구

English consonants & vowels

1. Consonants(자음)
- Larynx 떨림
- voiced sound(유성음, 성대 막힌 상태)와 voiceless sound(무성음, 성대 열린 상태)으로 구분

2. Vowels(모음)
- 모든 모음은 voiced sound

Articulation

- 조음, 소리를 변형하는 과정
- 입모양(혀의 위치, 턱)을 달리하여 말소리 생성
- 입을 통해 조음된 음성은 물리적인 움직임을 통해 공기와 작용하고 귓바퀴를 통해 증폭된 소리로 전달됨
- vocal tract(성도)의 변형을 통해서도 다양한 말소리 생성
  (ㄱ) vocal → speech소리를 만들어내는 tract(관
  (ㄴ) upper과 lower로 나뉘며 이 두 개가 붙으면서 소리가 남
  (ㄷ) upper structure은 고정, lower structure(혀)를 움직이면서 입모양을 바꿔 소리가 달리 남
  (ㄹ) lower structure 중 epi(뚜껑)glottis(기도로 가는 쪽)은 삼킬 때 기도로 가는 문을 막아서 침/음식이 식도로 넘어가게끔 해주는 역할

5 speech organs = constrictors = articulators

- 스피치를 만드는 메커니즘

1. Phonation process in larynx
- Larynx의 사용에 따른 소리의 분류
  (ㄱ) voiced(유성음) → vibrate, 성대 막힌 상태
  (ㄴ) voiceless(무성음) → 성대 열린 상태

2. Oro-nasal process in velum
- lower → m, n, ng
- raised → 그 외의 음들
- 숨 쉴 때 nasal tract은 열리고 velum은 lower됨
  ↔ velum이 raised 되면 nasal tract은 닫힘(모음 해당)

3. Articulatory process in lips/ tongue tip/ tongue body
- lips: 주로 입술 위 아래가 움직여서 소리가 남
- tongue tip(혀 앞쪽) ex. [트] [드] : 혀 끝이 위천장을 치면서 소리가 남
- tongue body(혀 뒤쪽) ex. [그] [응] : lips와 tongue tip 움직이지 않고 혀의 뒷부분(body)부분이 위쪽을 치고 내려옴

Control of constrictors(articulators)

- 협착을 만드는 주체 constrictor(3): lips, tongue tip, tongue body는 constriction이 얼마나 일어나는지에 따라 Constriction degree(CD, 상하)와 Constriction location(CL, 앞뒤)로 더 자세하게 분류 가능함

1. CL의 관점에서 미세조정
- Lips(아파): 아랫입술이 조금 앞으로 가면(bilabial) b, p 뒤로 가면(Labiodental) f
- Tongue tip(아타): 윗니를 hit(Dental) th, 좀 뒤(Alveolar) d, 좀 더 뒤(Retroflex) sh, 말려서 더 뒤(Palato-Alveolar)로 r
- Tongue body(아카): [CL] 앞으로 가면(Palatal) y[여] 뒤로 가면(Velar) g[그]

2. CD의 관점에서 미세조정
- stops(폐쇄음): p, b, t, d, k, g
- fricatives(마찰음): f v, s, z, ð ʃ
- approximants(접근음): r, l, w, j/y
- vowels
  (ㄱ) 모음은 막힘이 없음
  (ㄴ) CD의 관점에서 모음은 자음보다 더 degree가 낮음
  (ㄷ) 모든 자음은 vowels 제외 나머지 세가지에 포함됨

Phonemes

- 철자가 아니라 소리나는 대로 읽는 것

Praat

- formant
  모든 모음을 구별하는 수치적인 지표로서 formant 사용됨
  formant 값에 따라 어떤 모음인지 알 수 있음
- Pitch
  (ㄱ) 1초에 성대가 떨리는 횟수
  (ㄴ) 분석하고 싶은 소리가 여성/남성 목소리인지에 따라 pitch setting 들어가서 pitch range 설정해줘야 함 → 남자: 65 – 200 여자: 145 – 275
- Digital signal processing(디지털 신호 처리)

Vowel acoustics

- HZ
  (ㄱ) 사인웨이브: 반복되는 신호
  (ㄴ) 1 초 동안 사인웨이브가 얼마나 나오는가 = 주파수(frequency)

(ㄷ) 주파수 by ① 1 초에 사인웨이브가 몇 번 반복되는가 ② 사인웨이브의 크기

Source

- 우리가 어떤 소리를 내는지는 성대가 아니라 입에서 결정됨
- 사운드를 포함한 시그널은 다르게 생긴 **여러 사인웨이브의 결합**으로 이루어짐(이 세상에 존재하는 모든 신호는 조금씩 다른 사인 웨이브의 합으로 표현 가능, 복잡한 세계를 우리가 아는 가장 단순하고 간단한 것으로 쪼개서 이해하려는 관점)

Complex tone in spectrum

- pure tone = simplex tone
- tone frequency 와 amplitude 두 가지에 의해 결정
- 우리가 일반적으로 보는 equalize의 형태(=spectrum)은 x축이 쭉 채워져서 나타남
- 우리가 주변에서 보는 소리들은 complex tone이고 이걸 equalize에 뿌려보면 frequency/amplitude 그래프에 x값 많이 나타남
- 스펙트럼: 시간 개념 없음
  스펙트로그램: 스펙트럼을 시간축으로 계속 늘여 놓은 것
- 음의 높낮이는 pure tone 혹은 사인 웨이브의 진동수와 일치

Human voice source

- source 는 성대로만 낸 소리, tube 직전에서 즉, larynx 에서 나는 쇨
- source 에서 filter 를 어떻게 바꾸는지에 따라 모음 소리에 차이 생김
- pitch = F0 = fundamental frequency
- F0 이 정해지고 → 배음의 사인 웨이브의 합 → source 에서 나는 소리
- pitch 를 똑같이 한다고 가정했을 때 /아/와 /이/는 입모양에서 차이 만들어짐 만약 성대 윗부분 없다면 모두 같은 소리가 남
- F0 은 음의 높낮이에 따라 다르고(→ 여자는 시작이 높으므로 더 듬성듬성) 등간격으로 gradually decreasing
  amplitude 는 낮아지지만 사인 웨이브는 활발해짐
- low frequency → 에너지 높
  high frequency → 에너지 낮

Filtered by vocal tract

- source 에 입모양 O → 다양한 소리 낼 수 있음
- 배음의 구조는 그대로 유지되나 amplitude 의 패턴이 달라짐
- source → gradually/smoothly decreasing
  filtered → 제멋대로

source & filter

- source → harmonics gradually decrease, 산맥 형성 안됨
- filter → harmonics 왔다갔다함, 등간격으로 유지는 됨, 산맥 형성 O
- 산맥: 까만 부분이 높은 에너지이자 peak, 흰색/회색이 valley

filter

- vocal tract 의 shape 에 의해 소리 만들어짐 = filtered
- 첫번째 formant(F1) = 첫번째 나타나는 산맥
  F0 = 첫번째 harmonics
- 어디에 산맥이 나타나는지는 소리에 따라 결정됨, 특정한 입모양 O

Guitar plucking

- complex tone 이나 pure tone 이나 우리가 인식하는 높이는 동일함
- 기타 소리, 사람의 목에서 나는 소리는 harmonics 되어서 나는 소리임

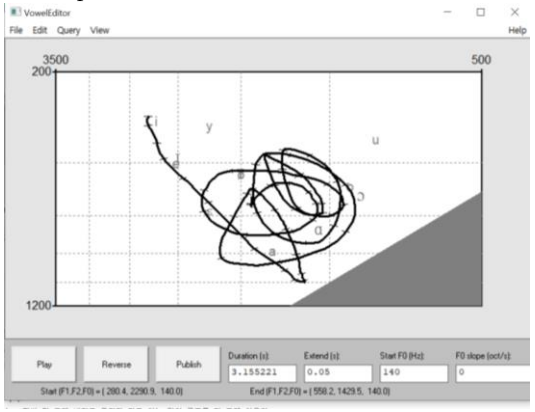
Source-filter theory

- 입 → 특정 소리 주파수 대 모양별로 다름
- 도장 역할을 하는 것이 filter(입)이고 도장 첫번째 무늬는 F1(첫번째 래그뭇)임

Formants

- F1 과 F2 로 모든 모음의 특징이 구별됨
- F1 과 F2 가 중복되는 모음은 없음 → F1 과 F2 의 모양에 따라서 모음 판별 가능함

Vowel space



- F1 과 F2 의 위치 = 입의 위치
- F1: 그 모음/혀의 높낮이를 결정(Height)
  F2: 그 모음/혀의 위치를 결정(Front/Back)
- 서로 다른 모음이 서로 다른 입모양 가짐

Python

- 코딩, 자동화
- 어떤 정보를 담느냐에 따라 단어 달라짐
  ex. '단어'라는 그릇 하나에 정보로서 '사과'를 담아놓으면 그 단어는 '사과'가 됨 사과를 빼고 물을 담으면 물이 됨 → 그 그릇에 여러가지 왔다갔다 바뀔 수가 있음
- 똑 같은 그릇인데 여러 개 가지고 있음 → 어떤 단어를 선택해서 combine을 할 것인가(순서, 위치 고려)
- 컴퓨터 언어에서 단어에 해당하는 부분이 **변수**(variable)
- 변수에 정보를 담고 기계와 소통하기 위한 **문법** 필요
  ① 변수에 정보를 넣는 것(**할당**, assignment)
  ② ~할 때 ,,, 실행 → 조건화(if **컨디셔닝**)
  ③ 여러 번 반복 → for **루프**
  ④ **함수**: 어떤 입력을 넣었을 때 내가 원하는 출력이 나오는 것
  함수 속에도 함수 들어갈 수 있음, 재사용/반복 사용 가능
- 숫자와 문자 assign
- 문자 반드시 인용 부호 ''나 ""필요함
- variable 정보, 정보를 주는 게 (사람의 말에 대한) 단어, 정보의 종류는 숫자와 글자 두 가지
- =은 같다는 뜻이 아니라 왼쪽에 있는 정보를 오른쪽에 있는 variable변수로 assign 한다는 의미, = 오른쪽에 있는게 항상 정보
- 한 칸은 cell, 파란색으로 선택된 상태에서 b 치면 아래에 새로운 셀 생김 a를 치면 above에 셀 생김 지우고 싶으면 x, 셀 추가 b
- 실행은 cell by cell로
- print 함수의 역할: 어떤 변수를 논항으로 넣으면 변수 안에 있는게 뭔지를 결과로서 보여줌
- variable이름은 unique하기 때문에 처음 a에 1이 들어갔는데 2를 새로 넣으면 기존 값 사라지고 2의 값만 할당됨
- 그냥 영어만 쓰면 무조건 변수이고 숫자만 쓰면 숫자, 문자는 반드시 인용부호!

```
In [18]: b = love
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-0f483ce471a7> in <module>
----> 1 b = love

NameError: name 'love' is not defined
```

- ;세미콜론 이용해서 한 줄에 나타낼 수 있음
- a에 들어간 데이터의 유형을 알고 싶을 땐 type함수 이용 논항으로 변수 들어감
- 리스트의 원소 str, int, float 등 모두 가능함 이 정보를 집합적으로 갖고 있는 게 list/ ,의 개수 리스트 원소 개수
- list assignment 할 때는 [] 대괄호 대신 ()쓰면? 튜플
  튜플과 리스트는 같음 다만 괄호 차이/ 튜플이 더 보안에 강함, 원소 바꾸기 더 힘듦

```
In [51]: a = (1, 'love', [1, 'bye'])

In [52]: type(a)

Out[52]: tuple
```

- 사전의 원리: 앞에 표제어 있고 그 뒤에 설명, 여기선 a와 b가 표제어 apple과 banana가 설명
- dict: 중괄호{}, 몇개 들어갈 것인가에 대한 구분은 ,로 대응어 표시는 :로, str말고도 여러가지 OK

```
In [54]: a = {'a': 'apple', 'b': 'banana'}

In [55]: type(a)

Out[55]: dict
```

* 변수

In[1]

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

        <class 'dict'>
        apple
```

- 여려개의 정보는 ,로 구분 각각의 정보는 :로
- 어떻게 정보를 가지고 오느냐

In[3]

```
In [3]: a = 1
        b = 1
        c = a + b
        c

Out[3]: 2
```

- a = 1 → a에 1이라는 정보가 들어 있는 것, a에 1이라는 정보를 assign
- a와 b에 저장되어 있는 정보 가져와서 c에 담김

In[4]

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

Out[4]: 4
```

- 왜 a[0]이 1이고 b[0]이 3? 괄호의 역할
- 어떤 정보에 access할 때: 정보를 통째로 갖고 오는 것은 변수명 그 자체로, 어떤 부분(partial information)을 가져올 땐 대괄호[]

In[5]

```
In [6]:  a = 1; a = float(a); print(type(a)); print(a)

         <class 'float'>
         1.0
```

- 함수(논항) → 함수 실행
- float(a) → a라는 변수를 float으로 바꿔줌
- print(a)의 결과? 1

In[8]

```
In [8]:  a = '123'; a = list(a); print(type(a)); print(a); print(a[2])

         <class 'list'>
         ['1', '2', '3']
         3
```

- variable 이름[인덱스] → 세부적인 정보 가져옴
- 2라는 문자 가져온 것

In[11]

```
In [11]:  a ={"a": "apple", "b": "orange", "c": 2014}
          print(type(a))
          print(a["a"])

          <class 'dict'>
          apple
```

- dict도 list처럼 여러 정보 담고 있는데 pair로 담고 있음
- dict에 있는 정보에 access할 때는 pair에서 앞부분을 인덱스로 씀
- "a"에 대응/해당하는 정보를 가져와라
- 인덱싱을 할 때 왼쪽에 있는 정보를 통해서 접근을 할 수 있음
- 표제어 부분 전부다 string → int같은 것도 표제어가 될 수 있을까? 똑 같은 결과 나옴(∵단순히 표제어가 바뀌었을 뿐임)

```
In [12]:  a ={1: "apple", "b": "orange", "c": 2014}
          print(type(a))
          print(a[1])

          <class 'dict'>
          apple
```

In[18]

```
In [18]:  a=[(1,2,3), (3,8,0)]
          print(type(a)); print(a[0]); type(a[0])

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

Out[18]: tuple
```

- 튜플

* string

In[21]

```
In [21]:  s = 'abcdef'
          n = [100, 200, 300]
          print(s[0], s[5], s[-1], s[-6])
          print(s[1:3], s[1:], s[:3], s[:])
          print(n[0], n[2], n[-1], n[-3])
          print(n[1:2], n[1:], n[:2], n[:])

          a f f a
          bc bcdef abc abcdef
          100 300 300 100
          [200] [200, 300] [100, 200] [100, 200, 300]
```

- string과 list 정보 접근 방법 비슷, 정보를 갖고 올 때 똑 같은 방식 취함
- 왼쪽에서 오른쪽으로 인덱싱 0 1 2 …
  오른쪽에서 왼쪽으로 인덱싱 −1, −2, −3 …
- 제일 첫번째 [0]
  제일 마지막 [−1]
- 범위 설정해서 여러 개 갖고 오려면 [:]
  ex. [1:3]은 첫번째부터 세번째 직전까지, 다시 말해 두번째까지 가져오는 것 1≤ <3
  ex. [1:]은 첫번째부터 끝까지
  ex. [:3]은 0번째부터 두번째까지
  ex. [:] 리스트 원소 전체

In[24]

```
In [24]:  s = 'abcdef'
          n = [100, 200, 300]
          len(s), len(n)

Out[24]: (6, 3)
```

- len()함수
  ① variable 내에 있는 정보의 개수, 길이(length)를 알려줌

In[25]

```
In [25]: s = 'abcdef'
         s.upper()

Out[25]: 'ABCDEF'
```

- s.upper(): 대문자로 바꿔줌

* string의 특징

In[26-34]

```
In [26]: s = ' this is a house built this year.₩n'
         s
Out[26]: ' this is a house built this year.₩n'

In [29]: result = s.find('house')
         result
Out[29]: 11

In [30]: result = s.find('this')
         result
Out[30]: 1

In [31]: result = s.rindex('this')
         result
Out[31]: 23

In [32]: s = s.strip()
         s
Out[32]: 'this is a house built this year.'

In [34]: tokens = s.split(' ')
         tokens
Out[34]: ['this', 'is', 'a', 'house', 'built', 'this', 'year.']
```

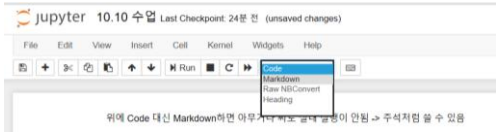
- 문장 단위로 string에 담아 보기
- 변수.함수() → 실행 됨
- s.find(논항필요): s 속에서 논항을 찾아라, 제일 첫번째 나오는 index를 찾는 것이지 전부 다 찾지 않음
  ex. s.find('house')의 값 11 → 제일 첫번째 house가 11번째에 있음
- s.rindex(논항필요): 오른쪽에서부터 일치하는 첫번째 index 값 찾는 것 right index
- s.strip(): whitespace 제거
- s.split(' '): 문자열을 리스트로 단어별로 끊어줌 s라는 긴 string을 split함수에 있는 입력을 이용해서 잘라라
  cf. 만약 단어들이 ' '로 연결이 된 게 아니라 ,로 연결되었다면 ','를 입력해서 ,기준으로 쪼갤 수 있음

In[37-38]

```
In [37]: s = ' '.join(tokens)
         s
Out[37]: 'this is a house built this year.'

In [38]: s = s.replace('this', 'that')
         s
Out[38]: 'that is a house built that year.'
```

- ' '.join(tokens): list를 다시 문자열로 만들되 띄어쓰기 해서!
- s.replace(논항 두개 무엇을 무엇으로 바꿀 것인지)
- variable 속에 들어가있는 정보는 []로 담는 게 리스트
- 스트링과 리스트 성질 비슷
- num(float, int), string, list, tuple, dict
- for 루프와 fuction(입력이 들어가고 출력이 나오는 것, 기존에 정의된 게 있고 새로 정의할 수도 있음)
- Markdown



for 루프

- 반복할 때
- 문법

for i in a:

① in 뒤에 있는 것을 하나씩 돌려서 돌린 것을 i가 하나씩 받아서 뭔가를 하라(뭔가를 하라는 것은 for 밑에 적음)
② in 다음 (리스트가 할당된) 변수명 → 리스트를 그대로 줌
③ in 다음 range 함수 → range는 인덱스를 만들어줌
ex. range(4) : 0부터 4미만까지

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

for i in range(4):
    print(a[i])
```

```
1
2
3
4
```

④ enumerate 함수 → 번호를 추가로 매겨줌, output: 자기 자신에 대한 값 + 번호
첫번째 i에는 번호가 매겨져서 0 s에는 red가
두번째 i에는 번호가 매겨져서 1 s에는 green이 …

```
In [14]:
a = ['red', 'green', 'blue', 'purple']
b = [0.2, 0.3, 0.1, 0.4]
for i, s in enumerate(a):
    print(i, s)
```

```
0 red
1 green
2 blue
3 purple
```

⑤ 첫번째 variable 인덱스 값 두번째 variable에는 list 각각의 값

```
In [15]:
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%
```

format 논항으로 들어간 s 와 b[i]*100 이 for루프 돌면서 중괄호 안에 값 들어가서 나옴
“ ” 안에는 어떤 식으로 결과 도출하고 싶은지
for 루프 네 번 돌음
첫번 째 i와 s 각각 0과 red 물고 그 다음 줄로 들어감
“ ” 속 중괄호 개수는 format의 논항 개수와 똑같음

⑥ in 다음 zip 함수 →

```
In [17]:
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%
```

a와 b는 반드시 원소 개수가 같은 리스트여야 함
zip을 함으로써 dict 비슷하게 페어로 4개짜리로 존재함

- for 속 for

```
In [24]:
for i in range(1,3):
    for j in range(3,5):
        print(i*j)
```

```
3
4
6
8
```

i = 1 → j = 3, 4 → i * j = 3, 4
I = 2 → j = 3, 4 → i * j = 6, 8

```
In [25]:
for i in range(1,3):
    print(i)
    for j in range(3,5):
        print(i*j)
```

```
1
3
4
2
6
8
```

if

- if 다음 조건문:
  조건문에 부합할 경우 수행할 명령
- = 은 variable assignment이고 == 는 왼쪽과 오른쪽 항이 ‘같다’는 뜻

```
In [18]:
a = 0
if a == 0:
    print("yay")
```

```
yay
```

```
In [20]:
a = 0
if a != 0:
    print("yay")
```

해당하지 않을 경우 출력 결과 나오지 않음

```
In [21]:
a = 2
if a != 0:
    print("yay")
    print("let's go")
```

```
yay
let's go
```

print 여러 개 할 수 있음

- ~하면 „„하고 그렇지 않으면 . . .을 한다 → else:

In [23]:
```
a = 0
if a != 0:
    print("yay")
    print("let's go")
else:
    print("no")
```

```
no
```

for과 if를 모두 포함하는 경우

- if 조건이 충족할 때만 print 수행

In [26]:
```
for i in range(1, 3):
    for j in range(3, 5):
        if j >= 4:
            print(i*j)
```

```
4
8
```