English consonants

Voiced Sound / Voiceless sound [유성음, 무성음]

모음

1) 모두 Voiced Sound
2) Monophthongs & Diphthongs

자음

1) Voiced/Voiceless

수업의 핵심은 **Phonetics**

Characteristic : 매번 다르다. 같은 소리가 나지 않는다.

Articulation : 말을 할 때 시작되는 과정이자, 음성학의 대부분의 영역을 차지한다.

소리 발생에 영향을 끼치는 요소들

Vocal Tract (Upper, Lower) : 서로 맞물려가며 소리를 조성한다.

Tract 중에도 nasal tract와 oral tract가 존재한다. 소리를 낼 시, 어떤 것이 닫혀 있고 어떤 것이 열려 있느냐에 따라서 소리의 종류가 바뀐다(Oral은 Velum의 영향을 받는다). 또한, Larynx에서 성대가 닫히거나 열려 있는 상태 또한 소리의 종류를 다르게 한다. 이 부분에서 파생되는 것이 유/무성음(Voiced, Voiceless)이다.

Constrictor

Tongue tip / Lips / Tongue body – Which makes constrict

이 요소들의 위치에 따라서 소리가 바뀐다. 보통, 이 요소들에 의해서 소리가 '어떻게' 나는지에 대한 것이 결정된다.

Constriction Degree

T-완전히 막혀있다 ->stop

S-조금 바람이 샌다 -> fricative

모음은 자음보다 degree가 작고, Vowels 제외하고는 모두 자음이다

Approximants – r, l, j, w

Stops – p,t,k,b,d,g,m,n,ng(oral tracts이 막혔느냐가 기준. 따라서, m,n,ng도 stops)

Firicative – s,z,f,v,,th,th',long s,

Sin wave 가 1초 동안 몇 번 반복되는가/그리고 높낮이가 어떻게 되는가. = Hertz

Sound의 높이는 같지만 quality가 같지는 않다

성대에서 바로 녹음을 하면, mumble하는 소리가 난다. 결국 나오는 소리는 입모양에 따라서 바뀐다.

Sin wave는 [frequency(주기),magnitude(높낮이)] / 세상의 모든 signal은 여러 다르게 생긴 sin wave의 결합으로 표현된다. 복잡한 세상을 간단한 요소[sin wave]의 합으로 표현할 수 있다. [Idea : 복잡한 것도 쉬운 것의 총합]

세 가지의 sin wave가 합쳐진다면, 주기가 가장 큰 것의 주기를 따라서 간다.

Sin 그래프.

X축 : 시간

Y축 : Value

오른쪽 그래프에서 x축이 frequency가 되는 것, y축이 amplitude가 되는 것을 spectrum이라고 한다.

여러 요소들을 분석하는 것 : spectrum analysis

이 sin graph들을 모두 합하면 amplitude의 고점, 저점이 가장 크거나 작은 그것과 같다.

피츠 : 음의 높낮이를 말한다.

음의 높낮이 = 1초에 진동수가 높을 수록 높아지고, 낮을 수록 낮아지는데, 모든 기본 사인그래프들의 합의 주기는 가장 느린 것을 따라가기 때문에, 가장 느린 사인 그래프에 따라서 높낮이가 결정된다!

성대에서 계속 나는 소리를 source라고 한다. 이 source (압력을 통해서 낼 수 이따.)를 어떤 필터를 통하게 하느냐에 따라서 여러 소리들을 만들 수 있다.

F0(Fundamental Frequency) = Amplitude가 가장 크다.

Speech의 source는 점점 작아지는 것이 특징.

Harmonics : 배음

여성의 source는 더 높고, 듬성듬성 있다. 똑같은 범위만큼 잘라놓는다면, 남자의 배음간격 vs 여자의 배음간격 : 남자의 배음간격이 더 많다

색깔로 구분한 것이 스펙트로그램(?)

Frequency가 점점 작아진다.

우리가 만들 것은 소리. 그렇다면 우리는 사람의 소리와 비슷한 기본 소리를 만들고, 그 소리를 입모양대로 깎아 놓으면 된다! 우리가 코딩할 것은 이런 것들.

Coding : **반복**되는 과정의 **자동화**

모든 Language는 단어가 있다. 단어를 어떻게 combine할 지가 중요

기계의 문법

1. 변수라는 그릇에 정보 넣는 것
2. 자동화는 if 개념을 사용
3. 여러 번 반복 (for route)
4. 함수 : 어떤 입력을 하면 출력이 되는 것.

오른쪽에 있는 것이 정보, 왼쪽에 있는 것이 variable

print같은 함수는 누군가가 만들어 놓은 것임.

Print : 어떤 변수를 넣으면 스크린에 표시해주는 함수. 입력을 말하는 함수는 (), 괄호 사이에 넣으면 된다.

구글링을 어떻게 할 것인가는 자신의 실력. 어떤 키워드를 넣어야 나올까.

처음에 a = 1을 해놓고, 그 후에 a = 2 를 넣으면 덮어쓰기가 된다.

파이썬에서 알파벳은 무조건 변수

그렇다면 단어를 넣고 싶다면? -> 'love' 이렇게.

단축키 : Shift + Enter = Run

Python의 명령어.

보통 python의 명령어는

변수 = 정보 의 형태로 이루어져 있다.

[Question : 변수를 숫자로 설정할 수 없는가?]

[Answer : 숫자는 literal 문자열이기 때문에, 숫자는 변수로 설정할 수 없다. 즉, 비유하자면 literal 문자열은 그 내용물이 다 채워져 있기 때문에 새로운 내용물을 채워 넣을 수 없다. In other words, 우리는 이 것들에 assign할 수 없다.]

가장 기본적인 것 : Directory / Information(정보) / Variable(변수) / Assign / Run(실행)

= means not equal, it means we assign information(right side) into variable(left side).

Variable = Information 형식을 사용할 경우, 가장 최근에 assign한 값이 표기됨. 예전 값은 사라진다.

1. **Assign**

a = 1 ; b = 3

c = d = 5

e = 'hello'

f = 1.3

g = [1, 3, 4, 5, [1, 3, 5, 7, 'love']

run

a + b -> 4

a + c -> 6

c + d -> 10

2. **명령어 [연산은 같은 type끼리만 가능하다]**

Print() : 괄호 안의 값을 화면에 나타내는 명령어

Print(a) -> 1

Print(a+b) -> 4

Print(e) -> hello

If : 여기서 a = 3 assign

Print(a) -> 3

Print(a + 3) -> 6

Type() : 각 변수의 정보의 type 을 나타내준다.

Type(a) = int(정수)

Type(f) = float(소수)

Type(g) = list(목록)

Type(e) = string(문자)

Variables

a = 1; print(type(a))

<class 'int'>

a = 1; a = float(a)

a-> class 'float' (float 로 바뀌는 순간, print 하면 1.0 으로 나온다.)

a = 1; a = int(a)

a -> class 'int' (int 로 바뀌는 순간, 소수점이 사라진다. 하지만 3.999..이런 식이면 올림, 이게 아니라면 버림.)

a = '123' ; print(type(a)); print(a[1])

class 'str'

2

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

<class 'list'>

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

3

*만약 a 가 list 인 상태에서, str 으로 바꿔버린다면?

a = str(a) ; print(type(a)) ; print(a[1])

"['1', '2', '3']"

,

*여기서 다시 list 로 바꾼다면?

a = list(a) ; print(a) ; print(a[2])

['[', "'", '1', "'", ',', ' ', "'", '2', "'", ',', ' ', "'", '3', "'", ']']

1

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

<class 'list'>

1

2

[3,'4']

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

<class 'tuple'>

1

2

[3,'4']

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

<class 'dict'>

apple

(숫자를 표제어로 쓸 수도 있습니다!)

print(type(a["a"]) ; print(type(a["c"])

str

int

만약

a = [1,2]

b = [3,4]

a + b = [1,2,3,4] (list 와 list 의 합이다)

a[0] + b[1] = 5 (int 와 int 의 합이다.)

*항상 기억해야 할 것은, 정보의 type/class 에 따라서 연산가능/불가능이 나뉜다.

또한, a 라는 그룹에서 여러가지 변수 중 하나를 콕 집고 싶다면,

a[index]로 하면 된다.

하지만, dictionary 정보에서는 앞의 index 부분을 써야 한다.(우리가 사전 찾을 때 생각)

***기억해야 할 부분 – 정보를 '수' 개념으로 assign 할 경우, 이 하나의 '수'는 하나의 값으로 인식이 된다. 따라서, 이 수를 나타내는 도구인 숫자들은 따로 하나의 값으로 지정되지 않는다. 하지만, type 이 str 이 될 경우, 이 숫자들은 고유의 의미를 가지며, 따라서 각자의 숫자를 집어낼 수 있다.**

이는 type 을 변환하는 것에서도 중요하게 다뤄지는 부분이기 때문에, 예시를 더 적도록 하겠다.

예를 들어, '수' 개념이 assign 되면 이는 list 화 시킬 수 없다. List 는 list 의 구성 요소들이 모두 하나의 값을 가지는 형식이기 때문에, string 형식을 가지고 있어야 변환이 가능하다. 다시 말해

'123'은 List 로 변환이 가능하지만,

123 은 list 로 변환시킬 수 없다.

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

<class 'list'>

print(type(a[0])

<class 'tuple'>

보통, list 안에 list 나 tuple 이 들어있게 되면, 그 그룹 하나를 하나의 원소로 인식한다.

String

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

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

<class 'list'>

100 300 300 100

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

* 범위로 index 를 출력할 때, 즉 x : x 이런 방식일 때, 음이 아닌 정수만 입력 가능하다. 굳이 입력하려면 가능은 하지만, 값이 출력되지 않는다.

* string 은 index 를 입력 시 string 의 형태로 출력된다.

*list 는 index 를 개별로 지정할 경우에는 원소 그대로(string), 범위로 지정해 줄 경우에는 list 의 형태로 출력된다.

*index 의 범위를 알고자 하는 대상의 길이보다 길게 지정할 수도 있다. 이 경우, 모든 원소가 출력된다.

*띄어쓰기 또한 정보 하나에 포함된다.

***[-1]같은 음수 index 가 존재하는 이유 : 뒤에서부터 정보를 찾기 위해서?**

len(s)

6

Len 은 **문자열**의 길이를 계산해준다.(띄어쓰기 포함)

당연히도, 수의 개념을 다루는 정보는 계산되지 않는다. 즉, 수 개념으로 a = 12312 라고 해서 len(a)를 넣어도 계산이 되지 않고 에러가 뜬다.

1) print(s[1]+s[3]+s[4:]*10)

bdefefefefefefefefefef

2) print([s[1]+s[3]+s[4:]]*10)

['bdef', 'bdef', 'bdef', 'bdef', 'bdef', 'bdef', 'bdef', 'bdef', 'bdef', 'bdef']

*1)에서는 s[4:]가 string 이기 때문에, s[4:]*10 은 string 정보로 판단되어, 그 정보를 10 번 반복하라는 의미가 된다. 따라서, 10 번 반복이 된 뒤 다른 string 정보들과 합치는 결과가 나온다. 결과는 string

*2)에서는 s[1], s[3], s[4:]를 더한 것(whole string)을 10 번 반복한다는 의미로 판단되어 저런 결과가 나온다. 결과는 list 로 나온다.

bdefbdefbdefbdef 이런 식으로 되게 하려면 어떻게 해야 할까?

s.upper()

'ABCDEF'

소문자를 대문자로 바꿔주는 함수

s = ' this is a house built this year. ₩n'

result = s.find('house')

result

11

*처음부터 시작해서 11 번째 칸에 house 가 시작된다.

result = s.rindex('this')

오른쪽에서부터 찾고 싶을 때 사용한다. 결과값은 앞에서부터 나온다.

- Index : 고유 번호라고 생각해도 될 듯. 보통 정수 차례대로 부여됨.(0 이 무조건 시작)

s = s.strip()

s 에 있던 string 에서 앞에 스페이스바, 혹은 /n 같은 것들을 제거하고 깔끔한 하나의 문장으로 만들어 준다.

tokens = s.split(' ')

s 라는 string 을 '와 ' 사이에 있는 것을 이용해서 잘라라.

s = ' '.join(tokens)

tokens 에 있는 list 의 것들을 ' 사이에 있는 것을 통해 붙여라

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

this 를 that 으로 replace 해라

Syntax

for loop : 여러 개를 여러 번 해야 할 때 사용.

A = [1,2,3,4]

for i in A:

print(i)

A 라는 집합 안에 있는 원소들을 i 로 받겠다 : 의 의미이다. Print(i)를 하게 되면, 그 원소들이 차례대로 프린트 된다.

a = [1,2,3,4]

for i in range(len(a)):

print(a[i])

a 라는 그룹 안에 len(a)의 범위만큼 프린트 해 달라.

print(a[i])로 하는 이유는, a 에서 i(괄호 안에 정해진 수)만큼 변수를 index 로 받으려고 하는 것이다.

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

각각의 것을 print 하고 싶을 때,

print(a[0])

print(a[1])

print(a[2])

print(a[3])

이렇게 하면 되지만, 수가 커질수록 효율성, 생산성의 차이가 크기 때문에, 루프를 사용한다.

따라서, for s in a:

print(s)

이렇게 함수를 사용하면 된다.(for loop)

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))

a 리스트 for loop 를 돌며 첫번째 variable 에 index 를 받아오고, 두 번째 s 에서는 그냥 list 의 값들을 받아온다.

(i , s,) = (index, variable)

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

b = [0.2, 0.3, 0.1, 0.4]

for a_, b_ in zip(a, b):

print("{}: {}%".format(a_, b_*100))

zip 이라는 함수를 통해서 (크기가 같은 리스트를) 합친다. 묶어서 한번에 print 해버리는 것.

If 함수

```
a = 0
if a == 0:
    print(a)
else:
    print(a+1)
```

0

기본적으로 알고리즘이랑 똑같음.

>=

<=

==

!=

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

3

4

6

8

range 함수는 (a, b)일 경우, a 이상 b 미만을 의미한다.

따라서 위 함수에서는 [1, 2] 와 [3, 4] 의 원소를 각각 곱한다.

총 4 번 실행된다. print 부분이 계산되면 완료.

또한, 맨 위의 for 에서는 2 번 돌지만, 그 아래 print 에서는 4 번이 돈다

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

여기서는 조건을 걸어주었다. j 가 4 보다 커야 프린트가 되기 때문에, 4,8 밖에 나오지 않는다.

Numpy

numpy 라는 그룹을 불러온다. 하지만, numpy 를 계속해서 치기는 힘들기 때문에 간단한 단어로 줄여준다.

Import numpy as np

Import matplotlib.pyplot as plt

여기서 matplotlib.pyplot 은 matplotlib 안의 pyplot 이라는 그룹을 import 하겠다는 것이다.

Np.emty([x,y], dtype =")

X,y 에 대응되는 숫자를 이용한 크기로 dtype 의 원소를 가진 행렬을 만든다.

Np.arange(0,10,2)

0 부터 10 전까지 2 의 간격으로 잘라 행렬을 만든다.

Np.zeros([2,3])

2x3 의 배열로 0 을 사용한 행렬을 만든다.

Np.ones([2,3])

2x3 의 배열로 1 을 사용한 행렬을 만든다.

Np.linspace(0,10,6)

0 부터 10 까지 6 칸으로 나눠 그 값을 나열한다.

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

X 를 뒤에 있는 두 리스트의 행렬로 만든다.

X.astype(np.float64)

X 의 원소의 type 을 바꾼다.

Np.zeros_like(X)

X 와 똑 같은 모양의 0 의 행렬을 만든다.

Data = np.random.normal(0,1,100)

랜덤으로 정규분포와 비슷한 형태의 그래프를 만든다.

Plt.hist(data,bins=10)

10 개의 막대로 그려진 히스토그램을 만든다.

Manipulation

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

2,3,4 는 각각

차원, 행, 열을 의미한다.

2 개의 행렬을 만들고, 이 각각의 행렬은 3x4 라는 의미이다.

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

여기서, X 는 3x4 행렬이었고, 2 개가 있었기 때문에 총 24 개의 원소가 있는데, 따라서 y 는 3x2 로 만들어진 행렬 4 개를 만들게 된다. -1 을 입력할 경우, 그 값을 알아서 찾으라는 의미로, 알아서 값이 들어가게 된다.

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

완전히 같은 행렬인가를 묻고 있다.

Inspecting

arr = np.random.random([5,2,3])

print(type(arr))

print(len(arr))

print(len.shape)

print(arr.ndim)

print(arr.size)

print(arr.dtype)

이것은 행렬을 보는 관점이다.

그리고, 차원에 관련된 개념 또한 제대로 알아두어야 한다.

행이 하나 있을 경우 1 차원,

행렬이 함께 있을 경우 2 차원,

그리고 행렬이 여러 개 있을 경우 3 차원이다.

계산법은 일반의 행렬 계산법과 같다. 명령어만 넣으면 된다.

비교하는 방법도 있는데, 이 또한 명령어만 넣으면 되어서 간단하다.

Sounds

1. 2 파이 동안 설정한 파라미터만큼 주기를 반복하는 Sin 그래프

2. (1) Radian 세타 및 sin 그래프 설정

세타 = np.arange(0,2*np.pi,1/주기)

s = np.sin(세타)

Phasor

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 [162]:

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

In [163]:

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

In [164]:

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

<IPython.core.display.Javascript object>

In [72]:

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

In [165]:

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