Python 독학 노트

**변수 선언:**

num = 5

s = "Hello"

**String 합치기:**

str = “Hello”

print str + “World!”

**if 문:**

if num == 5 or str != “H”:

print(“Hi!”)

elif num == 4:

print(“Hello!”)

else:

print(“No”)

**함수:**

def func\_name(par):

쏼라쏼라

func\_name(arg)

**print문에서 줄바꿈 없이 같은 줄에서 출력하기: print 끝에 , 붙이기:**

print(),

Import:

**Generic Import (일단 모듈 넣기):** import math -> 이럴 경우 math.sqrt() 이렇게 해야한다

**Function Import(모듈 안에 있는 특정 함수 넣기):** from math import sqrt -> 아럴 경우 그냥 sqrt()만 쓰면 된다

**Universal Import(math.sqrt()와 같이 . 없이 바로 sqrt만 사용하기):**

from math import \*

주의사항: 유니버설 임포트(Universal Imports)는 겉으로 보기엔 대단해 보이지만, 한 가지 정말 중요한 이유로 그리 좋은 아이디어가 아니라고 할 수 있습니다: 유니버설 임포트를 사용하면 어마어마한 양의 변수와 함수를 불러올 수 있지만, 이들이 여러분이 작성한 변수나 함수와 충돌을 일으킬 수 있기 때문입니다.

만약 여러분이 직접 만든 sqrt라는 이름의 함수가 있는 상태에서 math 모듈을 제네릭 임포트(generic import)를 통해 불러온다면, 여러분의 함수는 안전합니다: 여러분이 만든 함수는 sqrt가 될 것이고, 모듈로 부터 불러온 함수는 math.sqrt가 될 것이기 때문입니다. 하지만 만약 유니버설 임포트(universal import)를 통해 from math import \*라고 모듈을 불러오면, 문제가 생깁니다: 두 가지 다른 함수가 똑같이 sqrt라는 이름을 사용해서 충돌을 일으키기 때문이죠.

심지어 여러분이 작성한 정의가 모듈로 부터 불러들인 이름들과 직접적으로 충돌하지 않는다고 해도 import \*를 통해 여러 모듈들을 한꺼번에 불러오면 어떤 변수와 함수가 어떤 모듈로부터 왔는지 파악하기 어려울 겁니다. 이는 마치 제각기 다른 여러 박스로부터 수 없이 많은 도구들을 여러분의 방 안에 무작위로 쏟아부어 섞은 다음, 박스는 치워버려서 각각의 도구가 어느 박스로부터 왔는지 찾을 수 없게 만드는 것과 같습니다.

이러한 이유로, 필요에 따라 제네릭 임포트를 사용하거나 특정 변수 및 함수를 모듈로부터 불러오는 것이 가장 좋은 방법입니다.

Dictionary:

선언방법: my\_dic = {‘key1’: 2.3, ‘key2’: 543.2, ‘key3’: ‘HelloWorld’}

value가 없고 key만 있는 딕셔너리 선언 방법: my\_dic = {‘key1’:[]}

요소 추가: my\_dic[‘key4’] = 4.123 my\_dic[‘key5’] = [‘Hello’, ‘World’]

요소 제거(키key를 이용하여): del my\_dic[‘key1’] (del 이후 키값만 입력하면된다)

요소 제거(값value를 이용하여): my\_dic.remove(500) -> key2를 제거

요소 변경: my\_dic[‘key2’] = 500 (그냥 키 이름에 다른 값을 대입하면 된다)

딕셔너리에 리스트 값도 포함할 수 있다:

inventory = {'gold' : 500,

'pouch' : ['flint', 'twine', 'gemstone'], # Assigned a new list to 'pouch' key

'backpack' : ['xylophone','dagger', 'bedroll','bread loaf']}

딕셔너리의 리스트 값에서 일부분만 제거하는 방법:

inventory[‘backpack’].remove[‘dagger’]

딕셔너리의 모든 값 반환하기: .items()

예: print( my\_dic.items() ) 🡨 key와 value 모두 출력

my\_dict 선언방법 및 fish의 첫번째 요소 출력 방법:

my\_dict = {

"fish": ["c", "a", "r", "p"],

"cash": -4483,

"luck": "good"

}

print my\_dict["fish"][0]

for:

하나의 리스트에서 다른 리스트로 값 할당하기(for문 이용)

start\_list = [5, 3, 1, 2, 4]

square\_list = []

for num in start\_list:

square\_list.append(num \*\* 2) // square\_list안에 start\_list 값의 제곱 값 넣기

딕셔너리 안에서 특정 value 추출하기:

webster = {

"Aardvark" : "A star of a popular children's cartoon show.",

}

for definition in webster:

print webster[definition] -> 이러면 ‘A star..’를 출력함

주의: for문에서 in 뒤에는 하나의 딕셔너리를 포함했어도 다른 딕셔너리의 value를 사용할 수 있다. 예를들어

prices = {

"banana" : 4,

"apple" : 2,

"orange" : 1.5,

"pear" : 3

}

stock = {

"banana" : 6,

"apple" : 0,

"orange" : 32,

"pear" : 15

}

가 있고

for x in prices: 만 했어도

print x 🡨 “banana” 등 key를 표시

print "prices: " + str(prices[x])

print "stock: " + str(stock[x]) 🡨 이런 형식으로 stock에도 접근할 수 있다.

딕셔너리를 리스트 안에 넣은 후 그 리스트 안에 있는 모든 딕셔너리의 값 출력:

예: lloyd = {

"name": "Lloyd",

"homework": [90.0, 97.0, 75.0, 92.0],

"quizzes": [88.0, 40.0, 94.0],

"tests": [75.0, 90.0]

}

alice = {

"name": "Alice",

"homework": [100.0, 92.0, 98.0, 100.0],

"quizzes": [82.0, 83.0, 91.0],

"tests": [89.0, 97.0]

}

tyler = {

"name": "Tyler",

"homework": [0.0, 87.0, 75.0, 22.0],

"quizzes": [0.0, 75.0, 78.0],

"tests": [100.0, 100.0]

}

students = [lloyd, alice, tyler]

여기서 이름과 점수를 출력하고 싶은 경우:

for x in students:

print x["name"]

print x["homework"]

print x["quizzes"]

print x["tests"]

for문으로 문자열에서 문자 하나하나 출력하기:

word = “Hello”

for x in word:

print x

index와 value 모두 동시에 출력하기: 함수 enumerate 사용

choices = ['pizza', 'pasta', 'salad', 'nachos']

for index, item in enumerate(choices):

print index, item

두 개 이상의 리스트를 한 번에 반복하기: 함수 zip() 사용

list\_a = [3, 9, 17, 15, 19]

list\_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90]

for a, b in zip(list\_a, list\_b):

a는 list\_a의 값이고 b는 list\_b의 값이다

for문에서도 else 사용 가능

range:

range(2) -> 0부터 1까지 증가 [0, 1]

range(1, 4) -> 1부터 3까지 증가 [1, 2, 3]

range(0, 7, 2) -> 0부터 6까지 2씩 증가 [0, 2, 4, 6]

while:

while num < 10:

쏼라쏼라

while/else에서 while문이 break로 중단될 경우 else를 무시하고 바로 다음 코드 블록으로 건너뛴다

anonymous function: lambda

함수이기는 하나 def 어쩌고 할 필요 없이 한 줄로 선언이 가능한 함수

예: squares = [x \*\* 2 for x in range(1, 11)]

print ( filter(lambda x: x >= 30 and x <= 70, squares) )

Class:

class Classname(object): 🡨 관례상 클래스의 이름은 대문자로 시작

def \_\_init\_\_(self): 🡨 파이썬에서 클래스는 항상 \_\_init\_\_ 함수로 시작(부팅)

이를 포함한 클래스 안의 함수를 메소드라 부름

pass 🡨 함수 또는 클래스가 아무것도 하지 않게 만드는 명령어

zebra = Classname(“Jame”)

print(zebra)

상속

class Parents(child):

def 쏼라쏼라 (\_\_init\_\_ 없어도 된다)

덮어 쓰기(Override): 부모 클래스와 같은 이름의 메소드를 쓰지만 다른 방식으로 작동하도록 메소드를 새로 작성하는 방법

super(child, self).메소드: 덮어 쓰기를 시행하였지만 부모의 메소드를 또 사용하고 싶다면 super를 이용, 자세한건 아래 예 참조

예:

class Employee(object):

def \_\_init\_\_(self, employee\_name):

self.employee\_name = employee\_name

def calculate\_wage(self, hours):

self.hours = hours

return hours \* 20.00

class PartTimeEmployee(Employee):

def calculate\_wage(self, hours): 🡨 Override

self.hours = hours

return hours \* 12.00

def full\_time\_wage(self, hours):

return super(PartTimeEmployee, self).calculate\_wage(hours) 🡨 super()

milton = PartTimeEmployee("Jim")

print(milton.full\_time\_wage(10))

파일 입출력:

파일을 열 때, 해당 파일을 쓰기 전용 모드("w"), 읽기 전용 모드("r"), 읽고 쓰기 모드 ("r+"), 그리고 추가(append) 모드 ("a", 파일의 맨 끝에 작성한 새로운 데이터를 추가하는 모드)로 만들 수 있습니다.

open() 🡪 파일 오픈 my\_file = open("output.txt", "r+")

write() 🡪 텍스트 파일 등에 데이터 기입하기, write() 함수는 하나의 문자열 인자만 갖으므로, for문을 이용해서 데이터를 일일이 기입해야한다:

my\_list = [i\*\*2 for i in range(1,11)]

my\_file = open("output.txt", "r+")

for i in my\_list:

my\_file.write(str(i) + "\n")

my\_file.close()

read() 🡪 텍스트 등 파일의 데이터 모두 불러오기 (이때는 for문 없이 바로 f.read()만 사용 자동으로 한 줄씩 콘솔창으로 불러온다)

with… as… 🡪 close할 필요없이 파일을 자동으로 닫히게 해줌

with open("text.txt", "r+") as my\_file:

my\_file.write("Yes!!")

파이썬의 함수들:

var.upper() -> 문자열 대문자로

var.lower() -> 문자열 소문자로

length = len(string) -> string의 길이 반환

toStr = str(num) -> num이 숫자일 시 이를 문자열 형식으로 반환

num = max(num1, num2…) -> 최대값 (min(), abs()도 있음)

print type(23) -> 괄호 안에 있는 변수의 형식을 반환

var = raw\_input() -> 사용자로부터 입력을 받는다. 단, 문자열로 입력을 받으므로 형 변환을 하고 싶으면 다음과 같이 하면 된다:

var = int(raw\_input())

filter(조건, 리스트 등) 🡨 리스트 안에서 조건에 만족하는 요소들만 뽑음

bin(이진수나 숫자 등) 🡨 괄호 안 숫자를 문자열 형태의 이진수로 표현

int(“0b101”, 2) 🡨 int에 이렇게 두 개의 파라미터가 있을 경우, 왼쪽은 숫자를, 오른쪽엔 그 숫자의 진법을 표시하여 최종적으로 101 를 십진법으로 반환한다

그리고 주의할점!!! type으로 변수 형식 반환 시 string형식이 아니라 그냥 자료형 형식이다. 예를 들어, type(23)으로 int가 반환됐는지 확인할 경우, if type(23) == “int가 아니라 if type(23) == int 이다.

파이썬 list(딕셔너리)의 함수들:

my\_list.sort() -> 알파벳 또는 숫자 오름차순으로 정렬

주의: 딕셔너리에서는 개별 키를 정렬할 수 있다. 예:

inventory = {'gold' : 500, 'pouch' : ['flint', 'twine', 'gemstone']} 에서

inventory[‘pouch’].sort()로 pouch값만 정렬 가능

my\_list.append(something) -> my\_list에서 비어있는 부분으로 새 데이터 넣기

my\_list.remove(value) -> 해당 값을 제거(딕셔너리와 동일)

my\_list.pop(index) -> index번째 요소를 제거하고 그 value를 return

my\_list.keys() -> 리스트의 키 값만 반환

my\_list.values()

list에서 조건적인 value만 삽입하는 방법: list comprehension

my\_list = [x \*\* 3 for x in range(1, 11) if x % 3 == 0] 🡨 1부터10까지에서 3으로 나누어 떨어지는 값들의 세제곱 값을 list로 생성

list에서 시작과 끝, 그리고 간격 설정하기: list slicing

var\_list = my\_list[start:end:stride(간격)