<https://www.youtube.com/watch?v=M6kQTpIqpLs&t=284s> // 참고!

replit을 통해서 파이썬 간단하게 환경 만들기.

오른쪽 창을 이용해서 간단한 작업 가능.( 새로고침하면 사라지기때문에 간단한 임시적인 것을 실험하고싶다하면 사용하면 된다. )

저장하고 어떤 것들을 이용하기 위해서는 왼쪽 창을 이용해서.

x = 1

y = 2

print(x)

print(y)

->

1

2

print를 다른 줄에 쓰면 자동으로 개행문자 처리가 되는구나.

변수

**타입**

- 숫자

- 문자(열)

- boolean

**연산**

x + y

x - y

x \* y

x / y

x \*\* y (제곱)

x % y (mod 나머지연산)

**문자열**

x = "hello"

y = 'bye'

쌍따옴표랑 그냥 따옴표랑 같은 기능을 수행한다.

여러 줄을 처리하고 싶으면 """ 여러 줄 """ 로 처리해주면 된다

ex)

z = """

안녕하세요

워니입니다

"""

사칙 연산을 문자열끼리도 수행을 할 수 있다.

print("안녕" + "잘지내니") // 문자열끼리 더한 것이므로 옳은 표현

print("안녕" + '잘지내니') // 이런 식으로 따옴표랑 쌍따옴표랑 더하는 것은 가능하다.

print("너 혹시 몇살이니" + 4) // Error (문자열(str) + 숫자(int))

이런 경우에는 같은 타입으로 타입 **캐스팅**해줘야 한다.

print("너 혹시 몇살이니" + str(4))

x = 4 # 숫자 타입

y = "4" # 문자열 타입

에러가 안나게 연산을 수행하려면

이렇게!

print(str(x) + y)

print(x + int(y))

**주석**

C, C++처럼 //가 아니라 **#**로 한다.

**Boolean**

x = True

y = False

print(x) // True

print(y) // False

**조건문**

if 1 > 2 :

print(“Hello”) # 출력 X

if not 1 > 2 :

print(“Hello”) # 출력

if 1 > 0 and 2 > 1 :

print(“Hello”)

x = 3

if x > 5 :

print("Hello")

elif x == 3 :

print("Bye") # 출력

else :

print("Hi")

**함수**

반복되는 것을 그룹으로 묶어둔 것.

def 함수이름():

def chat():

print("함수 : 안녕? 넌 몇살이니?")

print("영희 : 나? 나는 20살")

chat()

chat()

chat()

chat()

이름을 자유롭게 바꾸고자 할때는.

def chat(name1, name2):

print("%s : 안녕? 넌 몇살이니" %name1) # C와 다른 점은 마지막에 ,가 안들어간 것.

print("%s : 나? 나는 20" %name2)

chat("알렉스", "윤하")

def chat(name1, name2, age):

print("%s : 안녕? 넌 몇살이니" %name1)

print("%s : 나? 나는 %d" %(name2, age) ) # 여기에 주목.

chat("알렉스", "윤하", 10)

chat("철수", "영희", 30)

def dsum(a, b):

result = a + b

return result # funtion에 리턴하는 값이 필요하면 return 만 넣어주면 된다.

d = dsum(1, 2)

print(d)

def sayHello(name, age):

if(age < 10) :

print("안녕" + name)

elif(age > 10 and age <= 20) :

print("안녕하세요" + name)

else :

print("안녕하십니까" + name)

sayHello("워니", 9)

sayHello("알렉스", 20)

sayHello("윤하", 40)

**반복문**

**# for, while이 있다.**

def chat():

print("철수 : 안녕 영희야 뭐해?")

print("영희 : 안녕 철수야, 그냥 있어")

# for

for i in range(10) : // i 는 0 ~ 9 까지 증가하면서 작동.

chat()

# while

i = 0

while i < 3:

chat()

i = i + 1

while이 편할때도 있고, for문이 편할때도 있지만,

보통 while 이 편할때는 무한루프를 사용하는 경우.

i = 0

while True:

print(i)

print("철수 : 안녕 영희야 뭐해?")

print("영희 : 안녕 철수야, 그냥 있어.")

i = i + 1

if i > 2:

break

break 는 반복문 밖으로 빠져나온다.

continue는 다음 i를 한다고 생각하면 된다.

**자료구조**

**1) 리스트**

element 여러개를 그룹핑할때 사용한다

# 이렇게 두가지 형태로 사용한다.

x = list()

y = []

x = [1, 2, 3, 4]

y = ["hello", "world`"]

z = ["hello", 1, 2, 3]

print(x)

print(y)

print(z)

print(x + y) # [1, 2, 3, 4, “Hello”, “World”]

x = [1, 2, 3, 4]

print(x[3]) # 4

print(x[0]) # 1

print(x[4]) # error : 범위 초과.

x = [1, 2, 3, 4]

x[3] = 10

print(x) # [1, 2, 3, 10]

자주 사용하는 것

**1. len(x)**

ex)

x = [1, 2, 3, 4]

num\_elements = len(x)

print(num\_elements) # 4

**2. sort(x)**

ex)

x = [4, 2, 3, 1]

y = sorted(x)

print(y) # [1, 2, 3, 4]

**3. sum(x)**

ex)

x = [4, 2, 3, 1]

z = sum(x)

print(z) # 10

**4. 반복문 + 리스트**

x = [4, 2, 3, 1]

for i in x :

print(i)

**5. x.index() // 괄호 안에 있는 것이 어느 index에 있는지.**

x = [4, 2, 3, 1]

y = ["Hello", "there"]

print(y.index("Hello"))

**6. “Hello” in x : “Hello” 가 해당 리스트 안에 있는 지.**

x = [4, 2, 3, 1]

y = ["Hello", "there"]

print("Hello" in y)

# 조건문 버전

x = [4, 2, 3, 1]

y = ["Hello", "there"]

if "Hello" in y:

print("Hello 가 있다")

**2) 튜플 : 리스트는 [] (대괄호)나 list(), 튜플은 () (소괄호)나 tuple()**

**# 이런 식으로 리스트에서 사용했던 것들을 모두 사용할 수 있다.**

x = (1, 2, 3)

y = ('a', 'b', 'c')

z = (1, "hello", "there")

print(x+y)

print('a' in y)

print(z.index(1))

**# 단, 안되는 것은 assign(할당). 해당 값을 변경하는 것. 즉, 만들어 질때 바꿀 수 없는 const 값이라고 보면 됨.**

x[0] = 10 # error : list : mutable(가변), tuple : immutable(불변)

**3) 딕셔너리 :**

**리스트 : [] or list()**

**튜플 : () or tuple()**

**딕셔너리 : {} or dict()**

**딕셔너리는 키(Key)와 값(Value)로 이루어져 있다.**

x = {

"name" : "워니",

"age" : 20,

}

print(x["name"])

print(x["age"])

x = {

0: "Wonie",

1: "Hello",

"age": 20,

}

print(x[0])

print(x[1])

print(x["age"])|

**# 이 키 값에는 immutable 즉, 불변하는 값(문자열, 숫자)만 적을 수 있다.**

1. “key” in x : 해당 키가 x에 들어있는지를 검사할 수 있다.

print("age" in x) # true

2. x.keys() : 딕셔너리에 들어있는 모든 키를 보여달라.

3. x.values() : 딕셔너리에 들어있는 모든 값을 보여달라.

4. for문 사용해서 key 와 value 확인

for key in x:

print("key : " + str(key)) # key

print("value : " + str(x[key]) # value

5. 딕셔너리는 assign 가능함.

x = {

0 : "Wonie",

1 : "Hello",

"age" : 20,

}

x[0] = "워니"

print(x)

# 존재하지 않는 키를 이용해서 할당할 경우 : **새로운 key, value 값이 할당된다(뒤에 추가됨).**

x["school"] = "한빛"

print(x)

**실습**

fruit = ["사과", "사과", "바나나", "바나나", "딸기", "키위", "복숭아", "복숭아", "복숭아"]

d = {}

# d = {"사과" : 1}

# d = {"사과" : 2}

for f in fruit:

if f in d: # fruit의 f가 d에 들어있어?

d[f] = d[f] + 1

else:

d[f] = 1

print(d)

**클래스.**

**1)**

class Person:

def say\_hello(self):

print("안녕!")

p = Person()

p.say\_hello()

2)

class Person:

name = "워니"

def say\_hello(self):

print("안녕! 나는" + self.name)

p = Person()

p.say\_hello()

**3)**

class Person:

def \_\_init\_\_(self, name): # initializer.

self.name = name

def say\_hello(self):

print("안녕! 나는" + self.name)

wonie = Person("워니")

michael = Person("마이클")

jenny = Person("제니")

**self는 this와 같은 역할을 하는듯하다.**

class Person:

def \_\_init\_\_(self, name):

self.name = name

def say\_hello(self, to\_name):

print("안녕!" + to\_name + "나는" + self.name)

wonie = Person("워니")

michael = Person("마이클")

jenny = Person("제니")

wonie.say\_hello("철수")

michael.say\_hello("영희")

jenny.say\_hello("미지")

**6)**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def say\_hello(self, to\_name):

print("안녕!" + to\_name + "나는" + self.name)

def introduce(self):

print("내 이름은" + self.name + "그리고 나는" + str(self.age) + "살이야")

wonie = Person("워니", 20)

wonie.introduce()

**상속**

**: 공통된 클래스가 있고 그 밑에 세부적인 클래스를 만들고자 할때.**

# 이런식으로 괄호안에 Person을 만들면 Police 가 Person을 상속하는 것.

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def say\_hello(self, to\_name):

print("안녕!" + to\_name + "나는" + self.name)

def introduce(self):

print("내 이름은" + self.name + "그리고 나는" + str(self.age) + "살이야")

class Police(Person):

def arrest(self, to\_arrest):

print("넌 체포됐다,", to\_arrest )

class Programmer(Person):

def program(self, to\_program):

print("다음엔 뭘 만들지? 아 이걸 만들어야 겠다: " + to\_program)

wonie = Person("워니", 20)

jenny = Police("제니", 21)

michael = Programmer("마이클", 22)

jenny.introduce()

jenny.arrest("워니")

wonie.introduce()

michael.program("자바")

michael.introduce()

**이건 단순하게 말하자면.**

**Person의 모든 코드를 Police나, Programmer 에도 마찬가지로 복사 -> 붙여넣기 되어 있는 것과 같은 것이다.**

**Person을 상속함으로써 Person의 모든 코드를 사용할 수 있는 것.**

**패키지, 모듈**

라이브러리 = 패키지(python)

패키지 : 모듈들을 합

모듈 : 코드를 잘 모아서 기능 하나를 구현해놓은 파일.

폴더의 이름 : 패키지의 이름(animal)

폴더 내에 dog.py, cat.py를 만들어서 기능을 넣는다.

animal 을 패키지라고 하려면 \_\_init\_\_.py 이 필요하다.

\_\_init\_\_.py에서 animal package가 어떤 모듈들의 합인지를 나타내줘야 한다.

**animal 폴더**

animal 폴더 내부의 파일

1) dog.py

class Dog:

def hi(self):

print("bark!") # 개가 안녕하는 소리

2) cat.py

class Cat:

def hi(self):

print("meow") # 고양이가 안녕하는 소리.

3) \_\_init\_\_.py

from .cat import Cat # . <-- "이 폴더에 있는" cat.py이라는 파일에서 Cat이라는 클래스를 가져와달라.

from .dog import Dog # 위와 마찬가지.

**최종 main.py의 기능.**

from animal import dog # animal package에서 dog 라는 모듈을 가져와달라

from animal import cat # animal package에서 cat이라는 모듈을 가져와달라

#from animal import \* # animal package에서 갖고있는 모듈을 다 불러와라

**# 여기서 차이를 나타냄.**

d = dog.Dog()

c = cat.Cat()

d.hi()

c.hi()

from animal import dog # animal package에서 dog 라는 모듈을 가져와달라

from animal import cat # animal package에서 cat이라는 모듈을 가져와달라

from animal import \* # animal package에서 갖고있는 모듈을 다 불러와라

from animal import dog는

#from animal.dog import dog 과 같음

**# 여기서 차이를 나타냄.**

d = Dog()

c = Cat()

d.hi()

c.hi()

**이런 식으로 패키지를 만든것과 같이 외부에서 패키지를 다운 받아서 사용할 수 있다.**