변수.

Python에선 변수를 선언 할 때 변수의 자료형을 지정해 줄 필요가 없다. 변수에 값을 대입하면 컴파일러가 자동으로 변수의 type을 설정해주는 듯 하다. 때문에 어떤 변수 a에 정수를 대입하여 a를 int로 이용하다가 문자열을 대입하여 a를 str로 이용하는 것이 가능하다. 또, python은 기본적으로 제공되는 내장 자료형이 class화 돼있으며 연산자와 함수들이 오버로딩이 잘 돼있어서 편리한 점이 확실히 있다.(예를들어 C++에서 배열을 int a={1,2,3}으로 선언하고 cout<<a를 하면 a[0]의 주소가 출력되지만 파이썬에서는 list=[1,2,3] 으로 선언하고 print(list) 를 하면 [1,2,3] 이 출력됨.)

함수

Python의 함수도 변수와 마찬가지로 return type을 지정해주지 않아도 된다. 사용자 정의 함수를 작성하기 위해선 함수 이름 앞에 'def' 를 붙여서 작성해야 한다. C++에서는 함수의 문장을 중괄호로 묶어서 작성하였지만 python에서는 들여쓰기로 작성한다. 만일 함수에 리턴값을 지정하지 않는다면 자동으로 NONE을 반환하며 이는 NONETYPE 클래스의 객체이다. C++의 void와 비슷한 역할로 생각하면 될 것 같다. 또한 python에선 함수에서 여러 개의 값을 리턴 할 수 있으며 그때 리턴되는 값들의 type이 달라도 가능하다.

```
def exam():
    return 1,'abc',[1,2,3]

a=exam()
print(a)
```

만일 위와 같이 코드를 짠다면 exam함수는 튜플 객체로 값을 반환하며, 출력 결과는 1,'abc',[1,2,3]이다. 튜플이라는 용어가 생소할텐데 이는 뒤에 설명하도록 하겠다

다음으로 람다 함수에 대해 소개하겠다. 람다함수는 python에서 처음 접할 것이다. 람다 함수는 간단한 함수를 만들 때 사용하며 문법은 'lambda 매개변수 : 표현식' 이다. 이 함수는

def lambda(매개변수):

return 표현식

과 비슷한 기작을 가진다. 다음 예문을 통해 이해 해보자.

```
def sum (x ,y ):
   return x +y
a =sum (3 ,4 )
b =lambda x ,y : x +y
print (a )
print (b (3 ,4 ))
```

.

<결과창>

7

7

클래스와 상속

python에서의 class의 개념과 그 활용은 C++에서와 몹시 유사하다. 앞으로 mathod와 instance라는 용어가 자주 등장할텐데 이는 각각 C++에서의 객체, 멤버 함수와 비슷한 개념으로 이해하면 좋을 것 같다.

```
class power :
   kick =0
   punch =0
   def __init__(self ,x ,y ):
     power .kick =x
     power .punch =y
   def swap (self ):
     tmp =power .kick
     power .kick =power .punch
     power .punch =tmp
def compare (a ):
   if a .kick >10 :
     print ('kick is more than ten')
   else :
     print ("kick is less than ten")
a =power (9 ,11 )
compare (a )
print (a .kick )
a .swap ()
compare (a )
print (a .kick )
```

<결과창>

kick is less than ten

9

kick is more than ten

11

위 코드에서 가장 생소할만한 것은 __init__ 과 self일 것이다. __init__은 C++에서의 생성자와 그 역할이 같다. self는 클래스의 인스턴스를 의미한다. 파이썬에서 매써드는 항상 첫 번째 파라미터로 인스턴스를 전달 받는다. 따라서 매써드를 선언할 때 첫 번째 파라미터는 self로 선언해주어야 한다. self에 대한 자세한 이해를 위하여 다음 예문을 살펴보자.

```
class Foo :
     def func1 ():
         print ("function 1")
     def func2 (self ):
         print (id (self ))
         print ("function 2")
```

```
>>> f = Foo()
```

```
>>>id (f)
```

<결과창>

```
43219856
```

여기서 id 함수는 파라미터의 메모리 주소를 출력해주는 내부함수이다. f라는 인스턴스의 메모리 주소가 43219856임을 알 수 있다. 다음으로 f인스턴스로 function2를 호출해보자.

```
f.func2()
```

<결과창>

```
43219856
function 2
```

id(f)에서와 같은 값이 출력 됨을 알 수 있다. 이로써 우리는 self가 인스턴스의 참조임을 알 수 있다. 즉, power 클래스 예문에서 a.swap() 함수에서 self로 받아간 값은 a라는 인스턴스의 참조라는 뜻이다. C++에서의 this와 연관지어 이해하면 더욱 이해가 쉬울 것이다. C++에서의 this는 객체의 주소였다는 점에서 self와 유사하나 self는 인서턴스의 참조이고 this는 주소라는 점에서 약간의 차이점이 있다.

다음으로 compare 함수의 선언부를 살펴보면 C++과는 다르게 파라미터의 변수 타입을 적지 않았음을 알 수 있다. 이 역시 컴파일러가 자동으로 compare함수의 파라미터 a의 타입을 power클래스의 인스턴스로 정해주기 때문이다.

소멸자는 __del__함수로 정의되어 있으며 소멸자 역시 매써드이기 때문에 self를 반드시 파라미터로 받아야 한다. 소멸자는 C++에서와 마찬가지로 self 이외의 파라미터를 받을 수 없는 것 같다. (확실치 않아서 조교님들께 여쭤보고 싶습니다.)

다음으로 상속, 오버로딩, 오버라이딩에 대해 살펴보자. 기본적으로 C++에서와 원리가 거의 같다. 다음 예문을 살펴보자

```
class Person :
   def greeting (self ):
     print ('super')

class Student (Person ):
   def greeting (self ):
     super ().greeting ()    # 기반 클래스의 메서드 호출하여 중복을 줄임
     print ('sub')
a =Student ()
a .greeting ()
```

여기서 super() 함수는 superclass의 매써드를 불러오는 함수이다. 따라서 결과가 다음과 같다.

<결과창>

super

sub

반복 자료형

list

list는 가장 빈번하게 사용되는 반복 자료형이다. Python의 list는 원소들의 type이 동일하지 않아도 된다. python에서는 배열의 인덱스를 두가지 형식으로 표현할 수 있다. 하나는 C++에서와 마찬가지로 첫 원소의 인덱스가 0 이고 다음 원소로 넘어갈 때 마다 1씩 증가하는 표현법이다. 또 다른 하나는 마지막 원소의 인덱스가 -1이고 이전 원소로 넘어 갈 때마다 1씩 감소하는 표현 법이다. 즉 ls=[1,2,3,'a','b'] 라는 list가 있다면 원소 1의 인덱스는 0 또는 -5이다.

split함수는 문자열을 분리해주는 함수이다. parameter로 문자열을 받으며 그 문자열을 구별자로 문자열을 분리하여 list객체를 리턴한다. parameter로 받은 구별자는 list의 원소에 포함시키지 않는다. 따라서 split하는 문자열에 구별자가 n개 있다면 split함수로 return된 list객체의 원소는 n+1개가 된다.

Join함수는 split과 반대로 list를 parameter로 받아서 하나의 문자열로 변환해준다. 이때 원소들 사이는 .join() 앞의 문자열로 이어준다.

tuple

list는 원소를 대괄호[] 로 둘러싸지만 tuple은 소괄호()로 둘러싼다. 기본적으로 튜플은 리스트와 거의 비슷하며 다른 점은 리스트는 원소를 수정, 변경 할 수 있지만 튜플은 불가능 하다는 것이다.

dictionary

dictionary는 중괄호{}로 원소를 둘러싸며, 특이하게도 순서를 가지지 않는다. dictionary는 키와 값을 가지며 {키:값} 형태로 선언한다. 다음 예문을 살펴보자.

```
dic ={'C':'C++','p':'python'}
print (dic [0 ])
```

앞서 설명했다시피 dictionary는 순서가 없기 때문에 인덱스로 값에 접근하는 것이 불가능하다. 때문에 위 코드는 컴파일 에러가 난다. 그렇다면 dictionary의 특정 값에 접근하려면 어떻게 해야할까? 바로 대괄호 안에 키 값을 넣어 불러오면 된다. 다음 예문을 보자.

```
dic ={'C':'C++','p':'python'}
print (dic ['C'])
```

<결과창>

C++

제어문

제어문의 경우 개념 자체는 C++에서와 같으나 문법이 다른 부분이 많고 enumerate()와 같은 새로 접하는 함수들이 많기 때문에 교과서를 참고하여 그 문법을 익히는 것이 좋을 것 같다.