파이썬 - HW3

임베디드스쿨1기

Lv1과정

2020. 08. 11

강경수

1. RAM MEMORY

1) RAM 메모리와 포인터크기의 관계

기준 : 32bit cpu
포인터 변수 크기 : 4byte(32bit)
데이터 기본 크기 : 1byte(8bit)

정리1: 32bit cpu란, 레지스터가 한번에 처리하는 데이터 단위가 32bit이기 때문에. 32개의 신호선을 한번의 CLOCK에 처리한다고 생각 할 수 있음.

정리2: 32bit가 연산가능한 수의 범위. 2^32 = 약 42억 표현가능한 경우의 수 42억가지.

정리3: 컴퓨터가 42억개 처리가능한 수만 가지고 연산하면.. 공간이 부족함 따라서 이 42억개 각각을 하나의 주소로 생각하고 그 안에 기본처리단위로 1byte씩 할당함 왜 1byte? 16진수를 표현하기 위해

→ 4GB메모리 용량.
즉 방 개수(주소)가 1000개(42억개) 이고 그 방의 평수가 10평(1byte)이라고 생각할 수 있음
따라서 총면적은 1000*10평 = 10000평 (4GB)

2. mutuable vs immutable

1) mutable 객체를 생성한 후, 객체의 값 수정 가능
   변수가 의미하는 바는 수정된 같은 객체를 가르 킴.
   list,set,dict

2) immutable 객체를 생성한 후 객체의 값을 수정 불가능.
   변수는 해당 값을 가진 다른 객체를 가르 킴.
   int, float, complex, bool, string, tuple, frozen set

→ 알 수 있는 것. Python에서 변수란 없고 변수같은것이 곧 포인터.

c.f)
C) int a = 1; a라는 공간에 1값 할당
Python) a = 1; a는 1을 가르킴

2. mutuable vs immutable

3) 정수형의 경우 immutable이기 때문에 똑같은 id를 가짐.
4) 똑같은 값을 가진 list여도 a와 b는 다른 id 즉 unique하지 않음.

```
import copy

a=[1]
b=[1]
c=1
d=1

print(id(a))        61840072
print(id(b))        61840712
print(id(c))        2003937200
print(id(d))        2003937200
                    >>>
```

3. [:]

1)[1,2,3]과 같은 객체들은 고유의 id를 갖고 있다.
2)일반적인 변수 대입시 id는 같다. 즉 a,b,가 모두 동일한 id [1,2,3]을 가르킨다.

```
a = [1,2,3]
b = a
print(id(a))
print(id(b))
```

```
54907464
54907464
>>>
```

3)서로 다른 id를 가르키게 하는법
3-1) [:]사용
3-2) copy.deepcopy 사용

```
import copy

a=[1,2,3]
c= a[:]
b = copy.deepcopy(a)
print(id(a))
print(id(b))
print(id(c))
```

```
47950536
47951112
47951176
>>>
```

4. lambda

1) lambda (인자):(기능) 과 같이 사용함.
2) 필요한곳에 즉시 사용하고 버릴 수 있음.

```
#lambda 예시

b= lambda x,y:x+y
a=list(map(lambda x:x**2,range(5)))
#map은 list와 함수를 인자로 받아서 list로 출력함.

c=(b(1,2))

print(a)
print(c)

[0, 1, 4, 9, 16]
3
>>>
```

5. Function

1) def 함수이름(인자): #:빼먹지 말것!
   (tab) 본문
2) 인자가 없는 함수, return이 없는 함수, 인자,return 둘다 없는 함수, 둘 다 있는 함수.

```
def print_str(str1,str2):
    print(list_1)
    print(list_2)

list_1 = ['orange','apple']
list_2 = ['banana','mango']
print_str(list_1,list_2)
```

→ return 없음.

```
def plus(num_1,num_2):
    result = num_1+num_2
    return result

print(plus(3,5))
```

→ 인자,return 다 있음.

5. Function

3) C언어에서 함수는 값만을 바꿀 수 있었음. 이때 함수를 사용하여 변수 값을 직접 변경하려면 pointer 변수를 사용하여야 했음.
(명심! C언어에서 Call by reference란것은 존재 하지 않는다. 결국 다 메모리 값 Value 변경임!)

Python에서는 어떻게 될까?

```
def DataPlus(data):
    data = data+1
    return data

def DataPlus_list(data):
    data.append(2)
    return data

a=[1]
b=1

print(DataPlus_list(a))
print(DataPlus(b))
print(a)
print(b)
```

```
[1, 2]
2
[1, 2]
1
>>>
```

mutuable 객체는 값이 바뀌며

immutable 객체는 값이 변하지 않는다.

질문.
왜 이런 현상이 발생할까

5. Function

4) 함수 안에 자기 자신의 함수를 또 사용 가능하다. 즉 재귀함수.

```
def factorial(num):
    if num==1:
        return 1
    else:
        return num*factorial(num-1)

print(factorial(5))
```

5) 재귀함수를 이용한 유명한 알고리즘 → 하노이의 탑.
조금 더 시간을 들여 이해해 볼 것.

6. Pass

1) C언어의 continu와 같다.

```
#pass = continue

for i in range(1,5):
    if (i%2)==0:
        pass
    else:
        print(i)
```

```
1
3
>>>
```

7. Help

1) Help를 통해 함수에 대한 설명 사용법 확인 가능
2) 사용자가 직접 만든 함수에 관해서는 (함수명).__doc__ = “(내용)”을 통해 기록가능

```
#HELP!!!

def print_hi():
    print("hi")

print_hi.__doc__="이 함수는 하이를 출력합니다"

print_hi()

help(print_hi)
```

7. iter, next, yield

1) iter(): 반복가능한 데이터를 입력받아 반복자(인터레이터)를 반환하는 함수
2) next(): 인터레이터를 입력받아 다음 출력값을 반환하는 함수
3) yield(): 데이터를 입력받지 않아도 반복자를 만들어내는 함수
실행되면 스택 메모리를 그대로 보관하고 결과값을 호출한곳에 반환

```
#iter next yield

a=[1,2,3,4,5]

it=iter(a)

print(it)   #객체 가져옴
print(next(it))
print(next(it))
print(next(it))
```

```
<list_iterator object at 0x02FA3730>
1
2
3
>>>
```

7. iter, next, yield

```
#iter next yield

def reverse(data):
    for idx in range(len(data) -1,-1,-1): #range(start,stop,step)즉 거꾸로하나
        yield data[idx]  #data[idx]는 어떤 의미인지 다시 생각.

for char in reverse('glof'):
    print(char)
```

```
f
o
l
g
>>>
```

즉 반복가능한 개체에서 iter을 호출하고 이터레이터에서 next를 호출한다.

iter은 반복가능한 개체에서 이터레이터를 반환하고 next는 이터레이터에서 이값을 하나씩 꺼낸다.

함수안에서 yield사용시 함수는 제너레이터가 됨

yield에는 값(변수)를 지정함.

즉 위에서는 data라는 공간에 하나씩 거꾸로 값을 지정