

```
// 조각 #1.a

def gcd(a, b):
    if a < 0:
        a = -a
    if b < 0:
        b = -b

    if a + b > 0:
        while a > 0:
            a, b = b % a, a
        return b
    else:
        return 0

# main
for i in range(5, 11):
    print gcd(i, 36),
print
```



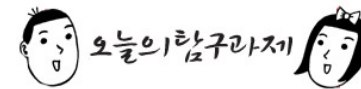
1. 들여쓰기의 모양은 문법적인 의미가 있을까?
2. 들여쓰기를 다르게 했을 때 동작의 차이가 있을까?
3. 이런 들여쓰기를 하면 어떤 장, 단점이 있을까?
4. def는 어떤 역할을 하는 것일까?
5. $a, b = b \% a, a$ 가 의미하는 것은 어떤 것일까?
6. range()의 각 인수의 의미는 무엇인지 인터랙티브 모드에서 여러 번 시도해 보아서 추론해 보자. (매뉴얼은 보지 말 것)
7. 실행 결과는 어떤 것이 나올까 예측해 보자.

```
// 조각 #1.b
```

```
# -----  
r = [1, 2, 7]  
r.append(19)  
print r, 8 in r, 2 in r  
  
r = ['egg', 'spam', 'sausage', 'spam', 'spam']  
print r.count('spam'), r.index('sausage')  
print r[1:3]  
del r[1:3]  
r.remove('egg')  
print r  
  
if 'egg' not in r or r[0] == 'spam':  
    print 'whoops~'  
  
# -----  
x = {  
    'egg': 'spam',  
    'spam': 'sausage',  
    'sausage': 'egg',  
    42: 'marvin',  
}  
print x['egg'], x[x[x['egg']]], x[42]  
x[6 * 7] = 'pinch'  
print 42 in x, 'pinch' in x  
del x[42]
```

(옆에서 계속)

```
# -----  
s = set()  
s1 = set((1, 2, 4, 8, 16))  
s2 = set((1, 3, 6, 8, 16))  
print s1 & s2, s2 - s1, list(s1 | s2)
```



1. # ----- 는 무슨 뜻일까?
2. 이 소스 조각에서 파이썬의 기본 자료형이 어떤 것이 있을지 발견해 보자.
3. 각 자료형에서 할 수 있는 일이 어떤 것이 있는지 소스에서 찾아보자.
4. 그 외에 자신이 파이썬을 디자인했다면 그 자료형에 어떤 것을 넣었을 지 더 생각해 보자.
5. 발견한 자료형 연산 방법과 원하던 연산을 정리해서 자신의 튜토리얼에 정리해 보자.

```
// 조각 #2.a

def log(msg, tm=None, actor='anonymous', level='INFO'):
    print (tm, actor, level), msg

log('Yo man', actor='nom')
log('Hey man', level='WARN', actor='nom')

x = ['spam egg spam spam', '12:00pm', 'boradori']
log(*x)
log(level='ERROR', *x)

y = {'level': 'IGNORE'}
log('apple egg apple apple', **y)

def nolog(msg, tm=None, actor='anonymous', level='INFO'):
    pass
log = nolog
log('how much is that doggie?')

powermod = lambda x, y, z: x ** y % z
print powermod(9, 10, 7), powermod(*range(1, 4))
```



1. 이 예제를 만든 사람은 뭘 보여주고 싶었던 것일까?
2. *와 **에 들어간 것이 어떤 차이가 있을까?
3. *와 **를 동시에 쓰면 어떻게 될까? 이럴 때 자신이 언어를 디자인한다면, 선택할 수 있는 방안을 여럿 떠올려 보고, 그 중 가장 좋다고 생각하는 것을 선택해서 그 이유를 적어 보자.
4. 여기서 nolog를 쓴 방식은, 파이썬 프로그래밍에서 실제로 드물지 않게 쓰인다. 어떤 경우에 유용할까? 어떤 문제가 있을까?
5. lambda가 def와 다른 점이 어떤 것이 있을까?

```
// 조각 #2.b

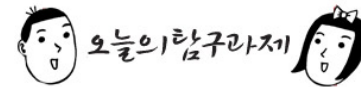
import os

print dir(os)

files = os.listdir('.')
for f in files:
    if os.path.isdir(f):
        continue

    fo = open(f)
    if 'spam' in fo.read():
        print 'Found spam here!', f

longest = ''
for line in open('/etc/profile'):
    if len(line) > len(longest):
        longest = line
print longest
```



1. `import os`에서 `os`는 모듈이라고 부른다. 다른 언어에서 이에 해당하는 것이 어떤 것이 있을까?
2. 이 소스에서 사용된 `dir`과 `open`은 특별히 불러오는 것 없이도 사용할 수 있다. 이런 함수들을 뭐라고 부르면 좋을까?
3. `os.listdir`이 리턴해 준 값은 어떤 자료형일까?
4. 이 프로그램의 두 부분이 각각 하는 역할은 어떤 것일까? 의도를 살려 생각해 보자.
5. 소스 아랫부분은 일반적으로 파이썬에서 많이 쓰는 스타일로 된 것이 아니다. 창의적으로 생각해 볼 때, 더 간단하게 만들려면 문법적인 장치로 어떤 것이 필요할까?
6. 파이썬은 “배터리가 포함되어 있습니다.”라고 늘 광고한다. 파이썬 웹사이트 (<http://docs.python.org/modindex.html>) 에서 파이썬에 포함된 배터리들을 보고 유용해 보이는 것 하나를 선택해서 어떤 함수가 있는지 살펴보자.

```
// 조각 #3.a

x = 'abcdefg'
i = iter(x)
print i.next()
print i.next()
print ''.join(i)

from itertools import chain, tee

for i in chain('abcd', 'efgh', [1,2,3]):
    print i

print list(chain('abcd', 'efgh', [1, 2, 3]))

a, b = tee('hello world')
print a.next(), a.next(), b.next(), b.next(), b.next()
print b.next(), a.next(), b.next(), a.next()
print '/'.join(a), '.'.join(b).upper()
```



1. 이터레이터(iter)는 파이썬의 내장 자료형이다. 위 소스를 볼 때, 주로 어떤 목적으로 도입이 되었을까?
2. 이터레이터가 사용할 수 있는 자료형은 iterable하다고 부른다. 지금까지 발견했던 자료형 중 어떤 것들이 iterable할지 생각해 보자.
3. from..import 의 의미는 무엇일까? import와의 다른 점은 어떨까?
4. 이 프로그램에서 사용한 두 itertools 함수를 보아, 경험에 비춰 보아 유용할 것 같은 itertools 함수는 어떤 것이 있을지 떠올려 보자.

```
// 조각 #3.b
```

```
class Dog:  
    def bark(self):  
        print "bark bark!"
```

```
class Cow:  
    def bark(self):  
        print """\
```

```
< mooo~ >
```

```
-----  
    \ \  ^__^  
    \ \ (oo)\ \_____  
        ( _ )\ \      )\ \ / \  
            ||----w |  
            ||     || """"
```

```
class Chihuahua(Dog):  
    def run(self):  
        print "chongchong~~"  
        self.bark()
```

```
d = Dog()
```

```
m = Cow()
```

```
d.bark(); m.bark();
```

```
d.__class__ = Cow
```

```
d.bark()
```

```
d.__class__ = Chihuahua
```

```
d.run()
```



1. 파이썬의 객체지향프로그래밍 환경은 기본적으로 상당히 동적이다. 이 소스의 실행 결과를 소스의 의도를 살려서 예측해 보자.
2. `__class__`는 특별히 파이썬에서 미리 예약된 이름으로, 특별한 의미를 지닌다. 어떤 의미일까?
3. `__class__` 외에도 파이썬에서는 많은 예약 이름이 존재하는데, 클래스와 관련하여 어떤 것이 더 있으면 유용할지 언어를 디자인하는 사람의 입장에서 생각해 보자. (힌트: 멤버 이름 찾기, 초기화, 오버라이드 가능한 연산자 등..)
4. 이 프로그램과 같은 방법을 자주 실제로 쓴다면 생길 수 있는 어려움은 어떤 것일까?