파이썬 프로그래밍

모듈의 활용과 패키지

한국기술교육대학교
온라인평생교육원

▣ 모듈의 다양한 import 방법

1. 모듈 import 하기

1) import 모듈명
•가장 기본적인 형태
- 이름 공간 mymath가 그대로 유지되므로 mymath.area() 형태로 자격 이름 사용

```
import mymath
print mymath.area(5)
```

```
78.5
```

▪import를 단순하게 사용하는 것에서 좀 더 복잡하게 사용하는 방법
▪모듈명. 안에 존재하는 함수(인자)

2) from 모듈명 import 이름들
•해당 모듈에 존재하는 지정 이름들을 현재 이름 공간으로 불러들인다.
•불러들인 각 이름들은 모듈 이름 없이 직접 사용 가능하다.
•import 하는 이름들이 기존에 미리 존재하고 있었다면 그 이름들에 의해 참조되던 기존 객체들은 상실된다.

```
from mymath import area, mypi
print area(5)
```

```
78.5
```

▪import 하는 다른 방법 = from 사용
▪from 모듈명 import 이름들 = 모듈명으로부터 이름들을 import
▪지정된 이름들 → area, mypl
▪현재이름공간 → test.py
▪mymath에 있는 area, mypl 함수가 현재 이름공간으로 import
▪함수 이름들을 import 하면 기존에 정의한 함수는 사라짐

▣ 모듈의 다양한 import 방법

1. 모듈 import 하기

3) from 모듈명 import *
•해당 모듈에 존재하는 '__'로 시작되는 이름들을 제외한 모든 이름들을
현재 이름 공간으로 불러들인다.

```
from mymath import *
print area(5)
```

```
78.5
```

▪import 뒤 * 사용하면 앞 이름공간의 모든 식별자를 import

4) import 모듈명 as 새로운 모듈 이름
•해당 모듈을 새로운 다른 이름으로 사용하고자 할 때 사용
•기존 모듈 이름이 너무 길거나 현재 사용중인 다름 이름들과 충돌이 일어날 때 유용

```
import string as chstr
print chstr
print
print chstr.punctuation
```

```
<module 'string' from '/Applications/Canopy.app/appdata/canopy-1.4.1.1975.
macosx-x86_64/Canopy.app/Contents/lib/python2.7/string.pyc'>
!"#$%&'()*+,-./:;<=>?@[₩]^_`{|}~
```

▪String에 원래 존재하고 있던 폴더 위치 + 실제 존재 파일 알려줌
→모듈을 print로 출력하면 해당 모듈이 원래 존재하고 있는 폴더 위치 및 파일명을 표기함

▪punctuation -> 문자열
→string 모듈내 punctuation 변수는 구두문자들을 포함하고 있음

▣ 모듈의 다양한 import 방법

1. 모듈 import 하기

5) from 모듈명 import 이름 as 새로운 이름[, 이름 as 새로운 이름]
•해당 모듈 내에 정의된 이름을 다른 새로운 이름으로 사용하고자 할 때 사용

```
from string import replace as substitute
print substitute
print substitute('ham chicken spam', 'chicken', 'egg')
```

```
<function replace at 0x1006a01b8>
ham egg spam
```

▪string 모듈에 있는 replace라는 함수를 현재모듈로 가져옴
▪replace를 그냥 가져오는 것이 아니라 as 뒤 이름으로 바꿔 가져옴

```
from string import replace as substitute, upper as up
print up
print up('abc')
```

```
<function upper at 0x10286b668>
ABC
```

▪string 모듈에 있는 replace 함수는 substitute로 가져옴
▪string 모듈에 있는 upper 함수는 up으로 가져옴

▣ 모듈의 다양한 import 방법

1. 모듈 import 하기

•import 문은 보통의 문(statement)이 작성될 수 있는 곳이면 어디에서나 사용 가능
- 예를 들면 함수 정의 def 문 안이나 if 문 안에서 사용할 수 있음

```
def str_test(s):
    import string
    t = string.split(s)
    return t
```

▪함수 정의 내에서 import 바로 사용

▣ 모듈의 다양한 import 방법

2. import에 의한 모듈 코드 수행

•import는 코드를 가져오기만 하는 것이 아니라 가져온 코드를 수행한다.

```
#FILE : mymath.py
mypi = 3.14

def add(a, b):
    return a + b

def area(r):
    return mypi * r * r

print area(4.0)
```

```
50.24
```

▪import mymath → mymath 수행값이 있으면 가져오기도 함

```
import mymath
```

```
50.24
```

▣ 모듈의 다양한 import 방법

3. 컴파일과 적재시간

•import mymath를 수행할 때 발생하는 일
  1) 우선 mymath.pyc를 찾는다.
  2) mymath.pyc가 없다면 mymath.py를 찾아서 mymath.pyc를 생성한다.
  3) 생성된 mymath.pyc를 메모리로 읽어들여 수행한다.
•.pyc 파일
  - 바이트 코드 파일
    ◦기계나 플랫폼(OS)에 의존하지 않도록 만들어진 일종의 목적 코드 (Object Code)
    ◦파이썬은 컴파일 언어이면서 동시에 인터프리터 언어의 수행 방식을 취하고 있다.
•새로운 .pyc 생성에 대한 판단
  - .py 수정 시간이 .pyc 수정 시간보다 더 최근일 때
•.py가 없이도 .pyc 파일만 있어도 import 가능
  - 코드를 숨기는 간단한 기법으로 활용 가능

▪.pyc → 원래 mymath.py가 import 되는 순간 바이트 코드 같이 생성
▪.pyc → 바이너리 파일도 아니고, 텍스트 파일도 아닌 중간 역할
▪바이트 코드는 문자들로 이루어짐
▪pyc 파일은 처음 생성 후 다시 생성 가능
▪py 없이 pyc만 존재해도 모듈로서 역할 가능
▪pyc의 내용은 일반적으로 내용 확인 불가

▣ 모듈의 다양한 import 방법

4. 모듈 이름과 이미 사용하고 있던 이름이 같다면?

•이전의 이름이 참조하던 객체는 상실됨

```
string = "My first string"
import string
print string
```

```
<module 'string' from '/Applications/Canopy.app/appdata/canopy-1.4.1.1975.
macosx-x86_64/Canopy.app/Contents/lib/python2.7/string.pyc'>
```

▪기존에 정의한 string은 사라지고 모듈만 남음

▪표준 모듈 이름과 동일한 이름으로 변수를 정의하는 것은 비 추천

```
import string
string = "My first string" #여기서 string이란 이름은 문자열을 참조하게 된다.
print string
```

```
My first string
```

▪import string 후 string을 따로 정의하면 모듈 string 사라짐

▣ 모듈의 다양한 import 방법

4. 모듈 이름과 이미 사용하고 있던 이름이 같다면?

•한번 import 되었던 모듈은 메모리에 적재되어지고, 나중에 다시 동일한 모듈을 import하면 메모리에 적재되어 있던 모듈이 즉시 사용된다.

```
import string
string.a = 1
string = "My first string"
print string

import string
print string.a  # 여기서 string 모듈이 기존에 이미 등록되었던 것임을 알 수 있다.
```

```
My first string
1
```

▪다른 문자열로 바뀌었을 때, 기존 string 모듈이 남음

파이썬 프로그래밍

모듈의 활용과 패키지

한국기술교육대학교
온라인평생교육원

▣ 모듈의 실행과 테스트 코드

•__name__
- 현재의 모듈이 최상위 모듈로서 수행되는지, 아니면 다른 모듈에 의해 import 되어 수행되는지를 구별하기 위해 주로 활용

•prname.py를 직접 수행할 때의 출력 내용: __main__
- >>> ipython prname.py◦__main__
- prname.py가 최상위 모듈로서 수행됨을 의미

```
#FILE : prname.py
print __name__
```

```
__main__
```

▪__name__ 이 값이 디폴트로 존재
▪__main__ 값이 __name__에 할당되어 있음
▪모듈을 만드는 것 → .py를 만들어서 바로 수행하는 것
▪__name__은 바로 수행해서 사용하고 있으므로 최상위 모듈

•prname.py가 모듈로서 다른 이름 공간으로 import 되어질 때의 출력 내용: prname

```
import prname
print prname.__name__
```

```
prname
prname
```

▪test가 최상위 모듈 → 이 모듈 내 prname의 네임은 prname
▪prname 안 print name = test의 prname name
▪최상위 모듈 → 바로 prname을 정의해서 수행하는 것
▪다른 모듈에 의해 import 되어짐 → name 값은 모듈 이름 그대로 출력

▣ 모듈의 실행과 테스트 코드

```
import string
print string.__name__

import re
print re.__name__

import mimetools
print mimetools.__name__

import os
print os.__name__
```

```
string
re
mimetools
os
```

- string, re, mimetools, os → 표준 모듈
- __name__을 쓰면 전부 다 모듈의 이름과 동일 내용 출력

▣ 모듈의 실행과 테스트 코드

•아래 코드는 최상위 모듈로서 수행될 때에만 test() 함수 호출이 일어난다.
•보통 파이썬 모듈을 개발할 때에는 마지막 부분에 if __name__ == "__main__": 과 같은 코드를 추가하여 테스트 코드를 삽입한다.

```
#file: module_test.py
def add(a, b):
    return a + b

def f():
    print "Python is becoming popular."

if __name__ == "__main__":
    print add(1, 10)
    f()
```

```
11
Python is becoming popular.
```

▪새로운 .py = 새로운 모듈
▪모듈이 최상위 모듈로 활용된다면 name에 main값이 들어가 있음
▪다른 모듈에서 import 하는 순간 name 값은 모듈 이름 그대로 출력
▪모듈 개발 시 if절 삽입 → 현재 개발 중인 모듈의 test 가능

▣ 모듈의 실행과 테스트 코드

•정의된 모듈이 다른 모듈로 import되어질 때에는 __name__은 모듈 이름 자체이므로 위 코드에서 if 문이 수행되지 않는다.
- 즉, test() 함수 호출이 곧바로 되지 않는다.

```
import module_test
```

▪모듈 개발 시 if절이 많이 활용됨

파이썬 프로그래밍

모듈의 활용과 패키지

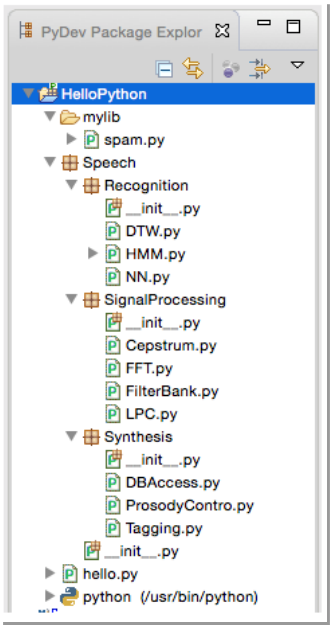
한국기술교육대학교
온라인평생교육원

▣ 패키지

1. 패키지의 이해

•패키지 (Package)
- 여러 모듈들을 한데 묶어서 정리해 놓은 구조
- 물리적으로 여러 모듈 파일을 모아 놓은 디렉토리에 해당
  ◦최상위 디렉토리 이름이 패키지 이름이 된다.
  ◦최상위 디렉토리 하위에 여러 서브 디렉토리는 해당 최상위 패키지의 하위 패키지가 된다.

•예제: Speech 패키지
- 각 폴더마다 __init__.py 파일 존재에 유의



▪모듈 = 파일, 패키지 = 디렉토리
▪최상위 패키지 이름 = 패키지 이름
▪speech → 최상위 디렉토리, 패키지 이름
▪동일한 기능, 동일한 역할을 하는 모듈끼리 하위 패키지 내 묶음
▪패키지 = 모듈(라이브러리)이 많을 때 분류하여(디렉토리) 묶음
▪__init__.py → 디렉토리를 패키지로 인식시키는 역할
▪서브 패키지에도 패키지와 마찬가지로 필요 (없으면 그냥 폴더)

▣ 패키지

1. 패키지의 이해

```
import Speech
```

•Speech 디렉토리의 위치
- sys.path(또는 PYTHONPATH 환경 변수)에 열거된 폴더 중 하나에 위치해야 한다.

▪이클립스의 경우 Python interpreter 창의 리스트 내 위치도 가능

•Speech/Recognition/HMM.py 코드 내용

```
def train():
    print "Train"
    pass

def loadModel():
    print "LoadModel"
    pass

def saveModel():
    print "SaveModel"
    pass
```

▣ 패키지

2. __init__.py의 역할

- __init__.py의 역할

3. import 하기

```
import Speech.Recognition.HMM
Speech.Recognition.HMM.train()
```

Train

▪패키지 내 특정 모듈, 모듈 내 함수 활용하는 방법
▪import를 통해서 패키지 명 옆에 마침표(.)찍어 특정 모듈, 함수 활용

```
from Speech.Recognition import
HMM HMM.train()
```

Train

▪import 옆에 마침표 찍는 것이 불편 → from 구문 사용
▪HMN이란 모듈 이름은 test.py 최상위 모듈에 import 됨
▪HMN 이름 곧바로 써서 활용 가능

▣ 패키지

3. import 하기

```
from Speech.Recognition.HMM import
train train()
```

```
Train
```

3. import * 하기

```
from Speech.Recognition.HMM import *
train() loadModel() saveModel()
```

```
Train
LoadModel
SaveModel
```

▪import * 모듈 내 있는 변수 전부 다 가져옴