Modules and Packages

CS195 - Lecture 12

Instructor: Dr. V

Lecture 12

- import, from, as
- importing from your other .py files
- dir()
- docstrings
- if __name__ == "__main__":
- sys.argv
- directories as packages and subpackages
    - __init__.py
    - __all__
    - intra-package references (., .., ..., etc)
- sys.path

import, from, as

import modules

```python
# import random and time modules
import random
import time

# sleep for 1sec
time.sleep(1)
# print random integer between 0 and 9
print( random.randrange(10) )
```

import modules

```python
# import random and time modules
import random, time


# sleep for 1sec
time.sleep(1)
# print random integer between 0 and 9
print( random.randrange(10) )
```

import specific functions from modules

```python
# import randrange, randint, sleep from random and time modules
from random import randrange, randint
from time import sleep

# sleep for 1sec
sleep(1)
# print random integer between 0 and 9
print( randrange(10) )
# print random integer between 1 and 10
print( randint(1,10) )
```

import specific functions from modules

```python
# import everything from random, and one method (sleep) from time
from random import *
from time import sleep

# sleep for 1sec
sleep(1)
# print random integer between 0 and 9
print( randrange(10) )
# print random integer between 1 and 10
print( randint(1,10) )
```

using **as** to rename functions

```python
# import random.choice and rename it to randchoice
from random import choice as randchoice

GREETINGS = ['hello','good afternoon','yo','what up','sup']

# print random greeting
print( randchoice(GREETINGS) )
```

using **as** to rename modules

```python
# import random module and rename it to rand
import random as rand

GREETINGS = ['hello','good afternoon','yo','what up','sup']

# print random greeting
print( rand.choice(GREETINGS) )
```

your .py files are modules too

importing from your other .py files

```python
#fileA.py
import random as rand

GREETING = 'hello'

def foo() -> int:
    '''Returns a random integer
    between 1 and 100.'''
    return rand.randint(1,100)


def bar(x:int,y:int)->int:
    '''Returns a random integer
    between x and y.'''
    return rand.randint(x,y)
```

```python
#fileB.py
import fileA

print( fileA.GREETING )

print( fileA.foo() )

print( fileA.bar(1,10) )
```

importing from your other .py files

```python
#fileA.py
import random as rand

GREETING = 'hello'

def foo() -> int:
    '''Returns a random integer
    between 1 and 100.'''
    return rand.randint(1,100)


def bar(x:int,y:int)->int:
    '''Returns a random integer
    between x and y.'''
    return rand.randint(x,y)
```

```python
#fileB.py
from fileA import foo,bar

print( foo() )

print( bar(1,10) )
```

importing from your other .py files

```python
#fileA.py
import random as rand

GREETING = 'hello'

def foo() -> int:
    '''Returns a random integer
    between 1 and 100.'''
    return rand.randint(1,100)


def bar(x:int,y:int)->int:
    '''Returns a random integer
    between x and y.'''
    return rand.randint(x,y)
```

```python
#fileB.py
from fileA import *

print( GREETING )

print( foo() )

print( bar(1,10) )
```

importing from your other .py files

```python
# "file A.py" <- spaces in name
import random as rand

def foo() -> int:
    '''Returns a random integer
    between 1 and 100.'''
    return rand.randint(1,100)


def bar(x:int,y:int)->int:
    '''Returns a random integer
    between x and y.'''
    return rand.randint(x,y)
```

```python
#fileB.py
fileA = __import__('file A')

print( fileA.foo() )

print( fileA.bar(1,10) )
```

- it's better to name your python files using the same syntax as variable names
- if this is not possible, the workaround is to use `__import__('file name')`

dir()

```python
#fileA.py
import random as rand

GREETING = 'hello'

def foo() -> int:
    '''Returns a random integer
    between 1 and 100.'''
    return rand.randint(1,100)


def bar(x:int,y:int)->int:
    '''Returns a random integer
    between x and y.'''
    return rand.randint(x,y)
```

```python
#fileB.py
import fileA

#dir(X) will list all
# members of X

#in the case of modules,
# dir(module) will list all
# vars, functions, classes,
# and imported modules
# in the module
print( dir(fileA) )
```

module docstrings, __doc__, __name__

```python
#fileA.py
'''my stuff (version 1.1)

includes foo() and bar()...'''

import random as rand

def foo() -> int:
    '''Get rand int 1 to 100'''
    return rand.randint(1,100)


def bar(x:int,y:int)->int:
    '''Get rand int x to y'''
    return rand.randint(x,y)
```

```python
#fileB.py
import fileA as mymod



# print fileA name
print( mymod.__file__ )

# print fileA name
print( mymod.__name__ )

# print fileA docstring
print( mymod.__doc__ )
```

main script vs module

- run fileA.py as main script
  - in terminal:
    - python fileA.py
- import from fileA.py as a module
  - in some fileB.py:
    - import fileA
    - from fileA import …
- …but what if you wanted to be able to run fileA.py in a different way depending on whether it was the main script or imported as a module?

__name__ == '__main__'

- use `if __name__=='__main__': <<codeBlock>>` to ensure that your <<*codeBlock*>> only runs if this python file runs as main script, but never when it's imported as a module
    - two underscores before and two after name
    - two underscores before and two after main

importing from your other .py files

```python
#fileA.py
import random as rand

def foo() -> int:
    '''Random int 1 to 100.'''
    return rand.randint(1,100)


print('print this either way')

if __name__=='__main__':
    print('hello from file A')
    print( foo() )
```

```python
#fileB.py
from fileA import *

print('hello from file B')
print( foo() )
```

- what is the output when we run the following in terminal:
  - python fileA.py
  - python fileB.py

importing from your other .py files

```python
#fileA.py
import random as rand

def foo() -> int:
    '''Random int 1 to 100.'''
    return rand.randint(1,100)


print('print this either way')

if __name__=='__main__':
    print('hello from file A')
    print( foo() )
```

```python
#fileB.py
from fileA import *

print('hello from file B')
print( foo() )
```

- `python fileA.py`:
  - print this either way
  - hello from file A…
- `python fileB.py`:
  - print this either way
  - hello from file B…

sys.argv

sys.argv

- python scripts can accept arguments from terminal:
  - python fileA.py **arg1 arg2 arg3**...
  - if you import the sys module, you'll be able to access **arg1**, **arg2**, **arg3** as sys.argv[1], sys.argv[2], and sys.argv[3], respectively
- sys.argv is a list of strings
- add this to hello.py:

```
import sys
print(sys.argv)
```

- then run in terminal: `python hello.py 11 12 abc`

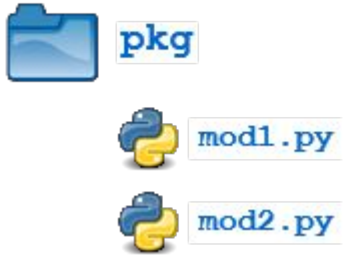
sys.argv

```python
# hello.py
import sys

if __name__=='__main__':
    if len(sys.argv)>1:
        print(f'hello {sys.argv[1]}!')
    else:
        print('hello!')

# what would this code output if in terminal you ran:
# > python hello.py

# what if you ran:
# > python hello.py joejoe

```

directories as packages

python packages

- if you want two or more modules to be in the same package, add them to a folder; e.g.,
  - add **mod1.py** and **mod2.py** to folder called **pkg**
  - how you can do things like this:
    - import both mod1 and mod2
      - from pkg import mod1, mod2
    - import everything from mod1
      - from pkg.mod1 import *
    - import mod2 module
      - from pkg import mod2
      - import pkg.mod2 as mod2
      - import pkg.mod2



importing modules from packages

```python
# pkg/mod2.py
'''module 2'''

import random as rand


GREETING = 'hello'

x = 10

def foo() -> int:
    '''Get rand int 1 to 100'''
    return rand.randint(1,100)
```

```python
# hello.py


import pkg.mod2


print( pkg.mod2.GREETING )
print( pkg.mod2.x )
print( pkg.mod2.foo() )
```

importing modules from packages

```python
# pkg/mod2.py
'''module 2'''

import random as rand


GREETING = 'hello'

x = 10

def foo() -> int:
    '''Get rand int 1 to 100'''
    return rand.randint(1,100)


```

```python
# hello.py


import pkg.mod2 as mod2


print( mod2.GREETING )
print( mod2.x )
print( mod2.foo() )






```

importing modules from packages

```python
# pkg/mod2.py
'''module 2'''

import random as rand


GREETING = 'hello'

x = 10

def foo() -> int:
    '''Get rand int 1 to 100'''
    return rand.randint(1,100)
```

```python
# hello.py


from pkg import mod2


print( mod2.GREETING )
print( mod2.x )
print( mod2.foo() )
```

importing from modules in packages

```python
# pkg/mod2.py
'''module 2'''

import random as rand


GREETING = 'hello'

x = 10

def foo() -> int:
    '''Get rand int 1 to 100'''
    return rand.randint(1,100)


```

```python
# hello.py


from pkg.mod2 import *


print( GREETING )
print( x )
print( foo() )
```

importing modules vs packages

- `in general, you can only import modules, not packages; e.g.,`
  - `import pkg.mod1 # imports mod1.py module from pkg`

- `however, your package folder should have a file called` **`__init__.py`**`, which is a namesake module for the package, and you can import it (and from it)`
  - `import pkg     # import __init__.py module from pkg`
- **`WARNING`**`: __init__.py file will run regardless of which modules are imported from the package`

importing from packages

```python
# pkg/__init__.py
'''pkg (v1.0)'''



print('loading __init__.py...')

a,b = 1,2

def foobar():
    print(a+b)
```

```python
# hello.py


import pkg

pkg.foobar()

print( pkg.mod2.GREETING )
```

- what is the output when we run the following in terminal:
    - python hello.py

importing from packages

```python
# pkg/__init__.py
'''pkg (v1.0)'''



print('loading __init__.py...')

a,b = 1,2

def foobar():
    print(a+b)
```

```python
# hello.py


import pkg
# this will print:
   loading __init__.py...

pkg.foobar()
# this will print:
   3

print( pkg.mod2.GREETING )
# this will throw an error
#   because there's no mod2
#   in __init__.py
```

importing from packages

```python
# pkg/__init__.py
'''pkg (v1.0)'''



print('loading __init__.py...')

a,b = 1,2

def foobar():
    print(a+b)
```

```python
# hello.py


import pkg, pkg.mod2

pkg.foobar()

print( pkg.mod2.GREETING )
```

- what is the output when we run the following in terminal:
  - python hello.py

packages and subpackages

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wav.py
              mp3.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              karaoke.py
              ...
```

Let's say you want to build a package for processing sound...

packages and subpackages

```python
# main.py
# what is the expected output?

from sound.formats import mp3

print( mp3.FILE_EXTENSION )

```

```python
# sound/formats/__init__.py

print('initiating \
 sound/formats/__init__.py''')



```

```python
# sound/__init__.py

print('initiating \
 sound/__init__.py''')



```

```python
# sound/formats/mp3.py

FILE_EXTENSION = '.mp3'

print('hello from mp3.py')


```

packages and subpackages

```python
# main.py

# what if you wanted to import
#  all the formats?
from sound.formats import *
```

```python
# sound/formats/wav.py

FILE_EXTENSION = '.wav'

print('hello from wav.py')
```

```python
# sound/formats/__init__.py

# you would expects all format
#   modules to be listed here
```

```python
# sound/formats/mp3.py

FILE_EXTENSION = '.mp3'

print('hello from mp3.py')
```

packages and subpackages

```python
# main.py

# what if you wanted to import
#  all the formats?
from sound.formats import *


```

```python
# sound/formats/wav.py

FILE_EXTENSION = '.wav'

print('hello from wav.py')


```

```python
# sound/formats/__init__.py

# this would import all modules
#  from package, but you
#  SHOULD NOT do this. why?
from . import wav,mp3,aiff,au

```

```python
# sound/formats/mp3.py

FILE_EXTENSION = '.mp3'

print('hello from mp3.py')


```

packages and subpackages

```python
# main.py

# what if you wanted to import
#  just one module from
#  formats?
from sound.formats import mp3

```

```python
# sound/formats/wav.py

FILE_EXTENSION = '.wav'

print('hello from wav.py')


```

```python
# sound/formats/__init__.py

# all of these would have to
#  get processed anyway!
from . import wav,mp3,aiff,au


```

```python
# sound/formats/mp3.py

FILE_EXTENSION = '.mp3'

print('hello from mp3.py')


```

packages and subpackages

```python
# main.py

# what if you wanted to import
#  all the formats?
from sound.formats import *
```

```python
# sound/formats/wav.py

FILE_EXTENSION = '.wav'

print('hello from wav.py')
```

```python
# sound/formats/__init__.py

# do this to list all
#  modules importable via *:
__all__ =  ['wav', 'mp3', ...]
```

```python
# sound/formats/mp3.py

FILE_EXTENSION = '.mp3'

print('hello from mp3.py')
```

Intra-package references

- if you want to reference what's inside the wav.py module from mp3.py, you can import wav.py from mp3.py:
  - `from sound.format import wav # absolute reference`
  - `from . import wav              # relative reference`
    - the . indicates current package
  - `from ..format import wav      # relative reference`
    - the double-dots (..) indicates parent package
    - add more double-dots to go higher in the hierarchy; e.g., four dots (....) indicates grandparent package

sys.path

- where does python search for packages and modules?
  - when your script says **import random** or **import sound** or **import mypkg**, where are those modules/packages located on your computer?
- all importable packages must be located in one of the folders listed in **sys.path**
- sys.path is initialized with
  - current folder
  - installation-dependent defaults
  - PYTHONPATH environment variable
- you can alter sys.path during runtime