Python Name Spaces, Modules, and Objects

CSE 415: Introduction to Artificial Intelligence
University of Washington
Spring, 2017

Name Spaces

A name space is mapping from names to objects, typically implemented as a dictionary.

The following have their own name spaces:

- the main (default) module during an interactive session,
- each imported module,
- each class
- most objects
- each function invocation

A Sample Module

A module is a collection of code, typically residing in a single file.

A module typically establishes a set of names, such as variables, function names, and classes.

A module may also execute code when it is loaded, with or without establishing any names.

A Custom Module

```
'''solfege.py
These are for translating from piano keys to solfege
and vice-versa. '''

NOTES = 'CDEFGABc'
SYLLABLES = 'DO RE MI FA SO LA TI do'

def keyToSyllable(key):
  i = NOTES.index(key)
  return SYLLABLES[3*i:3*i+2]

def syllableToKey(syllable):
  i = SYLLABLES.index(syllable)
  return NOTES[i // 3]
```


Using the Custom Module

```
import solfege

for k in 'CDECDEFGABccc':
    print(solfege.keyToSyllable(k), end=' ')
```

DO RE MI DO RE MI FA SO LA TI do do do

More on Module Importing

```
import solfege
f = solfege.keyToSyllable  # abbreviating the FQN

ODE_TO_JOY = 'EEFGGFEDCCDEEDD'


for k in ODE_TO_JOY :
    print(f(k), end=' ')
```

MI MI FA SO SO FA MI RE DO DO RE MI MI RE RE

What's in a Module?

```
dir(solfege)
['NOTES', 'SYLLABLES', '__builtins__', '__cached__', '__doc__', '__file__',
    '__name__', '__package__', 'keyToSyllable', 'syllableToKey']
```

This is a snapshot of the namespace of the solfege module.

How is a Name Space Typically Represented?

```
solfege.__dict__.keys() # __dict__ is a 'hidden'
 # member of the solfege module object.
dict_keys(['__builtins__', '__file__', 'syllableToKey', '__package__',
    'keyToSyllable', 'NOTES', '__cached__', '__name__', 'SYLLABLES',
    '__doc__'])
```


Alternative Styles of Importing

```
import solfege                    # module itself
solfege.keyToSyllable('A')
'LA'

import solfege as solf            # module, aliased
solf.keyToSyllable('A')
'LA'

from solfege import keyToSyllable # specific name only
keyToSyllable('A')
'LA'

from solfege import *             # all nonprivate names
SyllableToKey('LA')
'A'
```


Object-Oriented Python

Python supports object-oriented programming.

All python data consists of objects.

But programmers are not forced to use object-oriented style for everything.

Classes

```
class greeting:
  '''A simple example of a class
      definition in Python.'''
   g = 'Good '

   def __init__(self, d='day'):
     self.when = d

   def greet(self):
     print(greeting.g+self.when)

hi = greeting()
hi.greet()
Good day
eve = greeting('evening')
eve.greet()
Good evening
```


Class and Object Names Spaces

A class supports two operations:
instantiation of objects;
e.g., **x = greeting()**
retrieval of attributes.
e.g., **greeting.g** (from the class)
x.greet (from an instance)

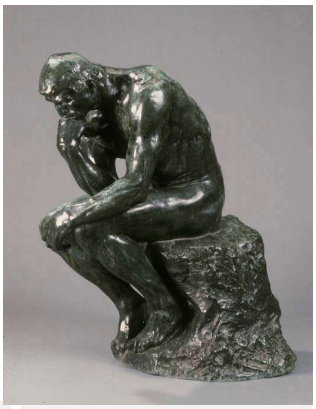
Each class has its own name space.

Each instance has its own name space.

```
x = greeting('night') # x.when could have a unique value
y = greeting('dawn')  # y.when is different.
x.hour = '11 PM'      # hour possibly unique attribute
```

A More Artsy Example

```
class artWork:
  def __init__(self, artist_name, title, medium):
    self.artist = artist_name
    self.title = title
    self.medium = medium

  def dimension(self):
     if self.medium == 'moving sculpture':
       return 4
     if (self.medium == 'sculpture') or (self.medium == 'film'):
       return 3
     else: return 2

  def report(self):
    return '"'+self.title+'"\n ('+self.medium+" by "+\
      self.artist+" having dimension "+str(self.dimension())+")"
```

Auguste Rodin:
The Thinker

A Subclass

```
class sculpture(artWork):
  def __init__(self, artist_name, title, material="bronze"):
    artWork.__init__(self, artist_name, title, 'sculpture')
    self.material = material

  def report(self):
    return artWork.report(self)+" made in "+self.material
```

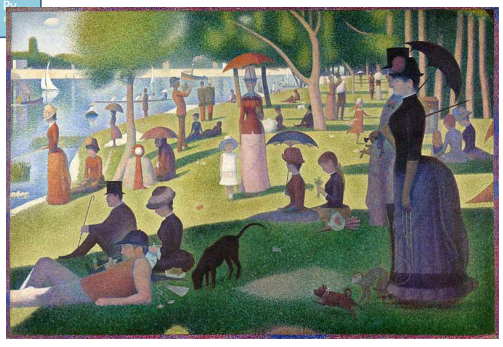

A Sibling Subclass

```
class painting(artWork):

  def dimension(self): return 2

  def setBrushStyle(self, style):
    self.brushStyle = style

  def report(self):
   return artWork.report(self)+" with the "+\
     self.brushStyle+" brush style"
```


Using the Classes

```
Thinker = sculpture("Rodin", "The Thinker")
print(Thinker.report())

GrandeJatte = painting(\
 "Seurat",
 "Un dimanche après-midi à l'Île de la Grande Jatte",
 "oil")
GrandeJatte.setBrushStyle("dots")
print(GrandeJatte.report())
```

```
"The Thinker"
 (sculpture by Rodin having dimension 3) made in bronze
"Un dimanche après-midi à l'Île de la Grande Jatte"
 (oil by Seurat having dimension 2) with the dots brush style
```

Un dimanche après-midi à l'Île de la Grande Jatte

Name Resolution

Retrieving, at runtime, the value associated with an attribute of an instance:

myObject.attribute

How attributes of instances are retrieved...

1. First look in the instance's namespace.
2. Then the class' namespace.
3. Then search the superclasses' namespaces.

Once the attribute is retrieved, it is processed either...

as an instance method (passing the instance as the first arg.)
as a class method (passing the class as the first arg)
as a static method (not passing the class).

The Name Space of An Instance

```
Thinker = sculpture("Rodin", "The Thinker")
Thinker.location = 'Musee Rodin, Paris'
Thinker.__dict__
```

```
{'material': 'bronze', 'medium': 'sculpture', 'title': 'The Thinker', 'location': 'Musee
    Rodin, Paris', 'artist': 'Rodin'}
```

and its class:

```
sculpture.__dict__
```

```
dict_proxy({'report': <function report at 0x02C91588>, '__module__': '__main__',
    '__doc__': None, '__init__': <function __init__ at 0x02C91540>})
```


The Name Space(s) of a Function Invocation

```
def fact(n):
  if n in [0, 1]: return 1
  if (n % 2)==1 :
    parity = '%d is odd' % n
  else:
    parity = '%d is even' % n
  print(parity)
  ans = n * fact(n-1)
  print('Again, '+parity)
  return ans

print(fact(7))
```

```
7 is odd
6 is even
5 is odd
4 is even
3 is odd
2 is even
Again, 2 is even
Again, 3 is odd
Again, 4 is even
Again, 5 is odd
Again, 6 is even
Again, 7 is odd
5040
```