# Python Class 2: Object-Oriented Programming

Betül Demirkaya

August 4, 2015

## Namespace and Scope

- Namescape: “mapping from names to objects”

## Namespace and Scope

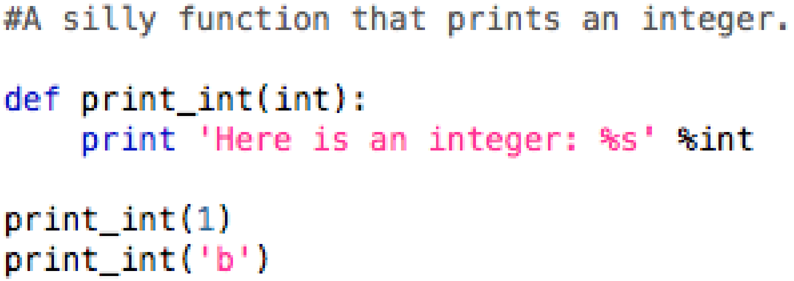
- Namescape: “mapping from names to objects”
- Scope: level at which “a namespace is directly accessible”

## Namespace and Scope

- Namescape: “mapping from names to objects”
- Scope: level at which “a namespace is directly accessible”
- Python follows the hierarchy:
  - Local names in functions.
  - Global names in module.
  - Built-in names such as `int()`, `sum()`.

Source: https://docs.python.org/2/tutorial/classes.html

## Namescape: How does it work?



```
#A silly function that prints an integer.

def print_int(int):
    print 'Here is an integer: %s' %int

print_int(1)
print_int('b')
```

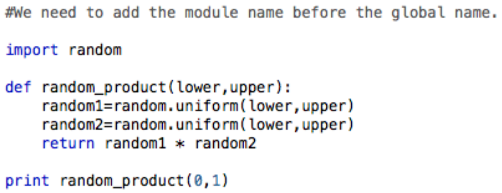
## Namescape: How does it work?

```python
#Function that returns the product of random draws from a uniform distribution.
def random_product(lower,upper):
    random1
    random2
    return random1 * random2

print random_product(0,1)

#NameError: global name 'random1' is not defined
```

## Namescape: How does it work?

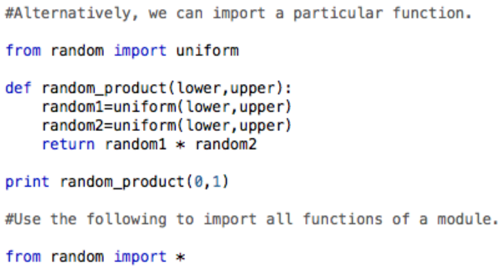
```
#We need to define numbers random1 and random2.
#We need to import the module random.

import random

def random_product(lower,upper):
    random1=uniform(lower,upper)
    random2=uniform(lower,upper)
    return random1 * random2

print random_product(0,1)

#NameError: global name 'uniform' is not defined
```

## Namescape: How does it work?



```
#We need to add the module name before the global name.

import random

def random_product(lower,upper):
    random1=random.uniform(lower,upper)
    random2=random.uniform(lower,upper)
    return random1 * random2

print random_product(0,1)
```

## Namescape: How does it work?

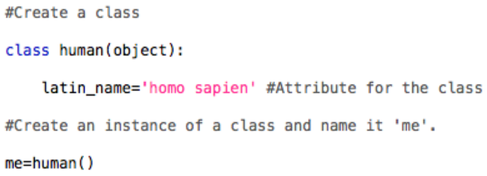


```
#Alternatively, we can import a particular function.

from random import uniform

def random_product(lower,upper):
    random1=uniform(lower,upper)
    random2=uniform(lower,upper)
    return random1 * random2

print random_product(0,1)

#Use the following to import all functions of a module.

from random import *
```

## Class and Instance

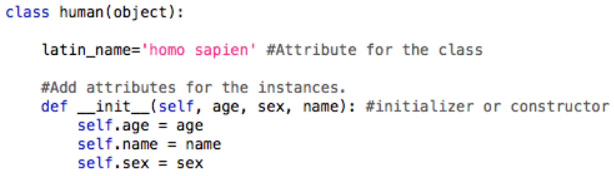
- Classes helps you create objects with
  - certain attributes
  - ability to perform certain functions.
- An instance is a particular realization of a class.

## Class and Instance: How to do it?



```
#Create a class

class human(object):

    latin_name='homo sapien' #Attribute for the class

#Create an instance of a class and name it 'me'.

me=human()
```

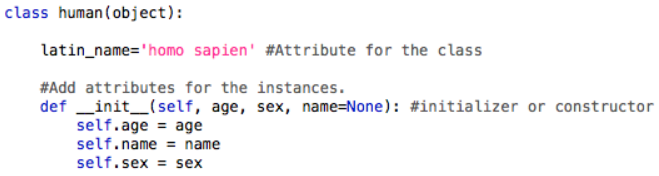
## Class and Instance: How to do it?



```
class human(object):

    latin_name='homo sapien' #Attribute for the class

    #Add attributes for the instances.
    def __init__(self, age, sex, name): #initializer or constructor
        self.age = age
        self.name = name
        self.sex = sex
```

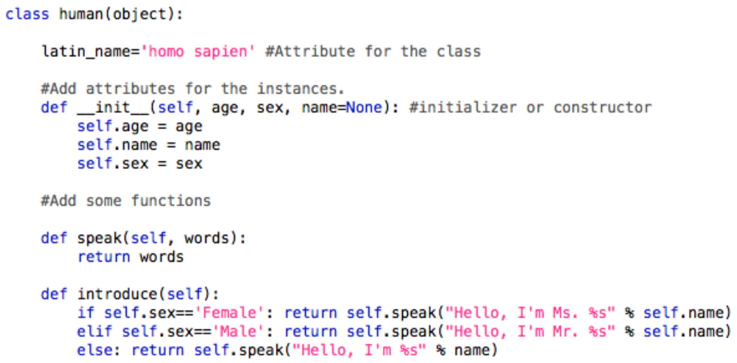
## Class and Instance: How to do it?

- You can set default values for attributes.
- Make sure you list non-default arguments first.



```
class human(object):

    latin_name='homo sapien' #Attribute for the class

    #Add attributes for the instances.
    def __init__(self, age, sex, name=None): #initializer or constructor
        self.age = age
        self.name = name
        self.sex = sex
```

## Class and Instance: How to do it?



```
class human(object):

    latin_name='homo sapien' #Attribute for the class

    #Add attributes for the instances.
    def __init__(self, age, sex, name=None): #initializer or constructor
        self.age = age
        self.name = name
        self.sex = sex

    #Add some functions

    def speak(self, words):
        return words

    def introduce(self):
        if self.sex=='Female': return self.speak("Hello, I'm Ms. %s" % self.name)
        elif self.sex=='Male': return self.speak("Hello, I'm Mr. %s" % self.name)
        else: return self.speak("Hello, I'm %s" % name)
```

`dir(human)` lists all the methods of the class.

## Inheritance and Polymorphism

- Inheritance enables you to create sub-classes that inherit the methods of another class.
- Polymorphism adapts a given method of a class to its sub-classes.