Introduction to Python: Day 2 - Classes

Ryden Butler

Definitions

- Namespace: "mapping from names to objects"
- Scope: level at which "a namespace is directly accessible"
- Python follows the hierarchy:
  - Innermost scope: local names
  - Scope of enclosing functions, innermost first
  - Next-to-last scope: global names in current module
  - Outermost scope: built-in names such as `int()`, `sum()`
- Source: https://docs.python.org/3/tutorial/classes.html

Examples

```
# A silly function that prints an integer

def print_int(int):
        int = 5
        print('Here is an integer: %s' % int)

print_int(1)
  5
```

- What's wrong with this?
- It works because the function searches within itself (local scope) then the global scope
- Best not to rely on scopes when re-naming is sufficient

Examples - continued

```
# function for the product of random uniform draws

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

print(random_product(0, 1))

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

Examples - continued

```
# must define values for objects random1, random2
# must load module for random sampling

import random

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

print(random_product(0, 1))

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

Examples - continued

```
# must add module name before 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))
  0.554891709253324
```

Examples - continued

```
# or we can import functions

from random import uniform # specific
from random import * # general

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

Definitions

- Classes help you create your own objects with
  - specific attributes
  - ability to perform specific functions
- An instance is a particular realization of a class
- You use attributes and methods of classes all the time in `R`

Examples

```
# create a class

class Human:
        # attribute for the class
        latin_name = 'homo sapien'

# create an instance of a class and name it 'me'

me = Human(0)
```

Examples

```
# create a class

class Human:
        # attribute for the class
        latin_name = 'homo sapien'

        # add attributes for the instance
        # this is an initializer ()or constructor)
        def __init__(self, age, sex, name):
                self.age = age
                self.name = name
                self.sex = sex
```

Examples

```
# create a class

class Human:
        # attribute for the class
        latin_name = 'homo sapien'

        # add attributes for the instance
        # can set defaults for attributes
        # list non-default arguments first
        def __init__(self, age, sex, name = None):
                self.age = age
                self.name = name
                self.sex = sex
```

Examples

```
class Human:
        # attribute for the class
        latin_name = 'homo sapien'

        # add attributes for the instance
        def __init__(self, age):
                self.age = age

        # add a function for the class
        def speak(self, words):
                return words

# see all methods of a class
dir(Human)
```

Definitions

- Inheritance enables the creation of sub-classes that inherit methods from another class
- Polymorphism adapts a given method of a class to its sub-classes
- Helps to keep code DRY

Other standard operators