Python classes

Computer & Information Sciences

W. H. Bell

Overview

- Mutability and copying objects.
- String representation of objects.
- Operator overloading.
- Summary.

Mutability, data flow and copying

- Objects are mutable, similar to lists or dictionaries.
- Assign reference, pass reference into function.
- Need to think about the data flow within an application.
- Can copy objects or deep copy objects.
- Deep copy copies dependencies too.
- With a careful data flow, it may not be necessary to copy objects.

Mutability

```
class MyClass:
    def __init__(self):
        self.name = "MyClass"

def updateName(object):
    object.name = "updated name"

if __name__ == "__main__":
    m = MyClass()
    updateName(m)
    print("name = \"" + str(m.name) + "\"")
```

Output

```
name = "updated name"
```

Copying objects

- Python does not have a copy constructor.
- Can create a class function to copy data.
- Can use the copy module to copy data.
- Copy local object data.
- Deep copy of object.

Copying objects: shallow copy

```
import copy

class MyClass:
    def __init__(self):
        self.name = "MyClass"

if __name__ == "__main__":
    m = MyClass()
    p = copy.copy(m)
    m.name = "Updated name"
    print("m.name = \"" + str(m.name) + "\"")
    print("p.name = \"" + str(p.name) + "\"")
```

Shallow copy affecting immutable data members.

Output

```
m.name = "Updated name"
p.name = "MyClass"
```

Copying objects: deep copy

```
import copy

class MyClass:
    def __init__(self):
        self.name = "My name"
        self.data = DataClass()

class DataClass:
    def __init__(self):
        self.ip = "8.8.8.8"

if __name__ == "__main__":
    m = MyClass()
    p = copy.copy(m)
    d = copy.deepcopy(m)
    m.data.ip = "8.8.4.4"
    print("m.data.ip = \"" + str(m.data.ip) + "\"")
    print("p.data.ip = \"" + str(p.data.ip) + "\"")
    print("d.data.ip = \"" + str(d.data.ip) + "\"")
```

Output

```
m.data.ip = "8.8.4.4"
p.data.ip = "8.8.4.4"
d.data.ip = "8.8.8.8"
```

Shallow copy.

Deep copy.

String methods

- Python classes support two standard string functions:
- `__str__(self)` - Called using str(object).
- `__repr__(self)` - Debugging feature. E.g. called when list or Dictionary of objects is printed.
- Generally useful to implement string methods.
- `__repr__` can call `__str__` function.

String methods

```
class MyClass:
    def __init__(self):
        self.name = "MyClass"

    def __str__(self):
        print("Called str")
        return "name=\"" + self.name + "\""

    def __repr__(self):
        print("Called repr")
        return "name=\"" + self.name + "\""

if __name__ == "__main__":
    l = [ MyClass() ]
    print(str(l[0]))
    print(str(l))
```

Calls __repr__

Calls __str__

Output

```
Called str
name="MyClass"
Called repr
[name="MyClass"]
```

Operator overloading

- Can define functions to allow object operations:
- Comparisons.
- Mathematical operations.
- Introduce two examples, others are beyond scope of course.
- Refer to Python reference material if you need other operators.

Object comparisons

```
class DataClass():
    def __init__(self, x):
        self.x = x

    def __eq__(self, other):
        return self.x == other.x

    def __ne__(self, other):
        return not self.__eq__(other)

if __name__ == "__main__":
    d = DataClass(10)
    p = DataClass(10)
    print("d == p : " + str(d == p))
    print("d != p : " + str(d != p))
```

Output

```
d == p : True
d != p : False
```

Using the operators.

Multiplication by a scalar

```
import copy

class DataClass():
    def __init__(self, x):
        self.x = x

    def __mul__(self, other):
        print("Called __mul__")
        result = copy.copy(self)
        result.x = self.x * other
        return result

    def __rmul__(self, other):
        print("Called __rmul__")
        return self.__mul__(other)

if __name__ == "__main__":
    d = DataClass(10)
    r = 3 * d
    r = d * 3
    print(r.x)
```

Output

```
Called __rmul__
Called __mul__
Called __mul__
30
```

Using the operators.

Summary

- Discussed mutability and copying objects.
- Introduced special Python class functions:
  - String.
  - Comparison.
  - Mathematical operations.
- More reading and practice is needed.