[320] Object Oriented Programming

Department of Computer Sciences
University of Wisconsin-Madison

Creating New Types

Classes and Other Types

Objects

https://www.macys.com/shop/product/martha-stewart-collection-set-of-6-cookie-cutters-created-for-macys?ID=5467270

Classes

dicts can represent many kinds of things

classes (today) are often a better option when all your keys are the same

```
m1 = {...}
m2 = {...}

p1 = {}
p2 = {}
p3 = dict()

p1["Fname"] = "Joseph"
p2["fname"] = "Peyman"
p3["fname"] = "Shri Shruthi"

print(type(m1))
print(type(p1))
```

create some objects
of type **dict** for **movies**

create some objects
of type **dict** for **people**

set some keys/values

```
class Person:
   pass

p1 = Person()
p2 = Person()
p3 = Person()

p1.Fname = "Joseph"
p2.fname = "Peyman"
p3.fname = "Shri Shruthi"

print(type(p3))
```

create a Person type/class

create some objects of type Person

set some attributes

Objects created from classes are mutable.
Attribute names are not fixed at creation.

PythonTutor: Compare dicts to class types

Python 3.6
known limitations

```
1  p1 = {"x": 4, "y": 5}
2
3  class Coord:
4      pass
5
6  p2 = Coord()
7  p2.x = 4
8  p2.y = 5
```

Edit this code

line that just executed
next line to execute

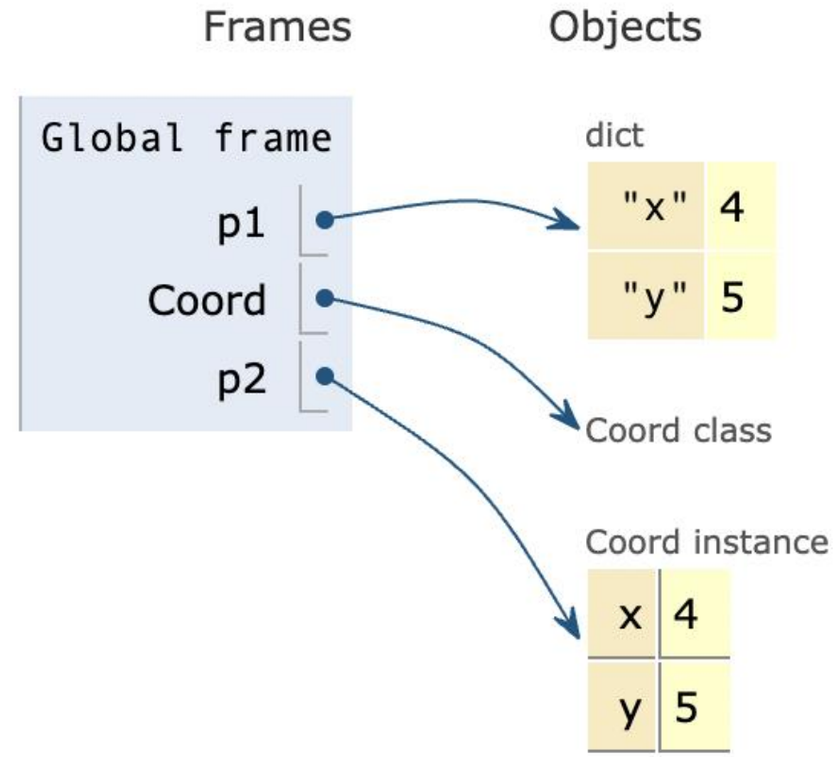
<< First | < Prev | Next > | Last >>

Done running (5 steps)

Visualized with pythontutor.com

NEW: subscribe to our YouTube

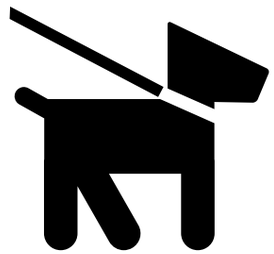
Move and hide objects

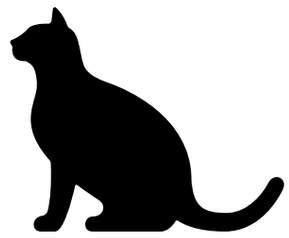


Coding Examples: Animal Classes

Principals

- objects and functions
- methods
- checking object type
- type-based dispatch
- receiver (self parameter)
- constructors





Classes

```
class Dog:
    def init(dog):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog()
```

which one is an attribute?

1. dog
2. name
3. mult
4. fido

Classes

```
class Dog:
    def init(dog):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog()
```

is this printed? do we crash?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)
```

is this printed? do we crash?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)
```

```
speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which call won't work?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)          # 1
fido.speak(5)           # 2
Dog.speak(fido, 5)      # 3
type(fido).speak(fido, 5)  # 4
```

which call won't work?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which one is NOT an example of type-based dispatch?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which one is NOT an example of type-based dispatch?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

speak(fido, 5)                  # 1
fido.speak(5)                   # 2
Dog.speak(fido, 5)              # 3
type(fido).speak(fido, 5)       # 4
```

which call style is preferred?

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)

fido.speak(5)
```

preferred style

Classes

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)


fido.speak(5)
```

what will be passed to the dog param?

Classes

what is a better name for the receiver parameter?

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)


fido.speak(5)
```

Classes

what is a better name for the receiver parameter?

answer: self

```
class Dog:
    def __init__(dog, name, age):
        print("created a dog")
        dog.name = name
        dog.age = age

    def speak(dog, mult):
        print(dog.name + ": " + "bark!"*mult)

fido = Dog("Fido", 9)


fido.speak(5)
```