Lecture 21: Object Oriented Programming
Creating objects using Class syntax

Comp 102

Forman Christian University

Recap

Object-Oriented Programming (OOP)

What is OOP?

A programming paradigm focused on **objects** - combining data and behavior to model real-world systems.

Object-Oriented Programming (OOP)

What is OOP?

A programming paradigm focused on **objects** - combining data and behavior to model real-world systems.

- **Encapsulation** - Combine data and methods
- **Abstraction** - Hide internal complexity
- **Inheritance** - Reuse and extend classes
- **Polymorphism** - One interface, many behaviors

Design principles for clean, reusable code

The Object Metaphor

Objects:

1. are like **black boxes**

The Object Metaphor

Objects:

1. are like **black boxes**
2. have **state** `(data)` and **behavior** `(functions)`
3. have unique **identities**

The Object Metaphor

Objects:

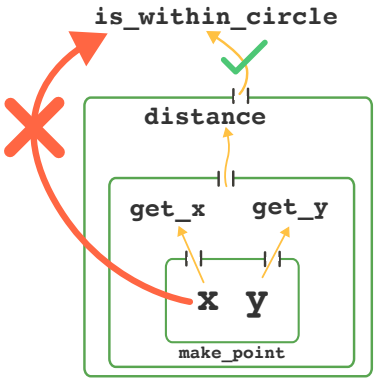
1. are like **black boxes**
2. have **state** (`data`) and **behavior** (`functions`)
3. have unique **identities**
4. can **interact** with each other by sending **messages**

The Object Metaphor

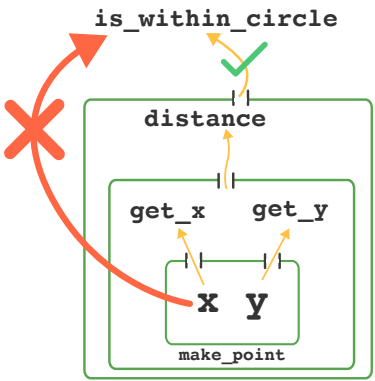
Objects:

1. are like **black boxes**
2. have **state** (`data`) and **behavior** (`functions`)
3. have unique **identities**
4. can **interact** with each other by sending **messages**

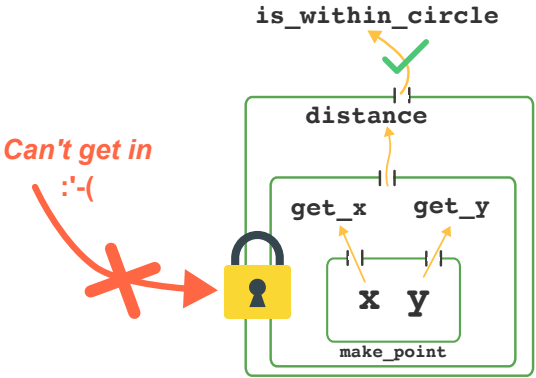
Abstraction



Abstraction



Encapsulation



Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9
10
11
12
13
14
15
16
17
```

Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15
16
17
```

Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15 p1 = create_point(1, 2)
16
17
```

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15 p1 = create_point(1, 2)
16
17
```

Creating Objects using Closure functions

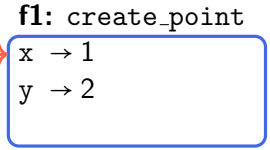
f1: create_point

x → 1

y → 2

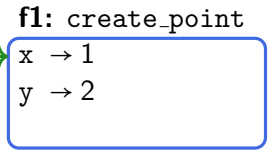
Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15 p1 = create_point(1, 2)
16 p1['print']()
17
```



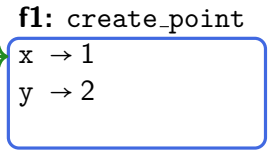
Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15 p1 = create_point(1, 2)
16 p1['print']()
17 p1['get_x']()
```



Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15 p1 = create_point(1, 2)
16 p1['print']()
17 p1['get_x']()
```



Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15 p1 = create_point(1, 2)
16 p1['print']()
17 p1['get_x']()
```

✗ Error

f1: create_point

x → 1

y → 2

Creating Objects using Closure functions

```
1  def create_point(x, y):
2      def get_x():
3          return x
4      def get_y():
5          return y
6      def print_point():
7          print(f'({x}, {y})')
8
9      return {
10         'get_x': get_x,
11         'get_y': get_y,
12         'print': print_point
13     }
14
15 p1 = create_point(1, 2)
16 p1['print']()
17 p1['get_x']()
```

✗ Error

f1: create_point

x → 1

y → 2

So far we've created objects following some of the core principles of OOP.

Creating Objects using **class** Syntax

```
1
2
3  class Point:
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

Creating Objects using The 'Class' Syntax

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

keyword

class name

```
class Point:
```

Creating Objects using The 'Class' Syntax

Creating Objects using The 'Class' Syntax

```
1
2
3  class Point:
4
5
6
7
8
9
10
11
12
13
14
15 p1 = Point(1, 2)
16
17
```

```
1
2
3  class Point:
4
5
6
7
8
9
10
11
12
13
14
15 p1 = Point(1, 2)
16
17
```

Creating Objects using The 'Class' Syntax

<self>

```
1
2
3  class Point:
4      def __init__(self, x, y):
5          self.x = x
6          self.y = y
7
8
9
10
11
12
13
14
15 p1 = Point(1, 2)
16
17
```

constructor

Creating Objects using The 'Class' Syntax

<self>

```
1
2
3  class Point:
4    def __init__(self, x, y):
5      self.x = x
6      self.y = y
7
8
9
10
11
12
13
14
15 p1 = Point(1, 2)
16
17
```

p1

Creating Objects using The 'Class' Syntax

```
1
2
3  class Point:
4    def __init__(self, x, y):
5      self.x = x
6      self.y = y
7
8    def get_x(self):
9      return self.x
10   def get_y(self):
11     return self.y
12   def print(self):
13     print(f'({self.x}, {self.y})')
14
15 p1 = Point(1, 2)
16
17
```

public interface

Creating Objects using The 'Class' Syntax

<self> p1

x → 1

y → 2

```
1
2
3  class Point:
4    def __init__(self, x, y):
5      self.x = x
6      self.y = y
7
8    def get_x(self):
9      return self.x
10   def get_y(self):
11     return self.y
12   def print(self):
13     print(f'({self.x}, {self.y})')
14
15 p1 = Point(1, 2)
16
17
```

The first parameter is always **'self'**

Creating Objects using The 'Class' Syntax

<self> p1

x → 1

y → 2

Creating Objects using The 'Class' Syntax

```
1
2
3  class Point:
4    def __init__(self, x, y):
5      self.x = x
6      self.y = y
7
8    def get_x(self):
9      return self.x
10   def get_y(self):
11     return self.y
12   def print(self):
13     print(f'({self.x}, {self.y})')
14
15 p1 = Point(1, 2)
16
17
```

'self' is passed automatically by Python

<self> p1

x → 1

y → 2

```
1
2
3  class Point:
4    def __init__(self, x, y):
5      self.x = x
6      self.y = y
7
8    def get_x(self):
9      return self.x
10   def get_y(self):
11     return self.y
12   def print(self):
13     print(f'({self.x}, {self.y})')
14
15 p1 = Point(1, 2)
16 p1.print()
17
```

Accessing attributes using the **dot notation**

Creating Objects using The 'Class' Syntax

<self> p1

x → 1

y → 2

```
1
2
3  class Point:
4    def __init__(self, x, y):
5      self.x = x
6      self.y = y
7
8    def get_x(self):
9      return self.x
10   def get_y(self):
11     return self.y
12   def print(self):
13     print(f'({self.x}, {self.y})')
14
15 p1 = Point(1, 2)
16 p1.print()
17
```

Creating Objects using The 'Class' Syntax

<self> p1

x → 1

y → 2

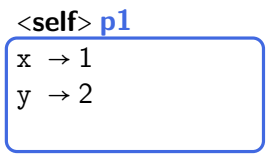
<obj>.<attribute>

note: *no parameters passes*

```
1
2
3  class Point:
4    def __init__(self, x, y):
5      self.x = x
6      self.y = y
7
8    def get_x(self):
9      return self.x
10   def get_y(self):
11     return self.y
12   def print(self):
13     print(f'({self.x}, {self.y})')
14
15 p1 = Point(1, 2)
16 p1.print()
17
```

p1

Creating Objects using The 'Class' Syntax



Big Idea

The `__init__()` method is invoked **automatically** when an object is created.

Big Idea

The `self` parameter is the object which invokes the method using the dot operator.

Creating Multiple Objects

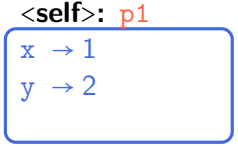
```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9
10
11
12
13
14
```

Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10
11
12
13
14
```

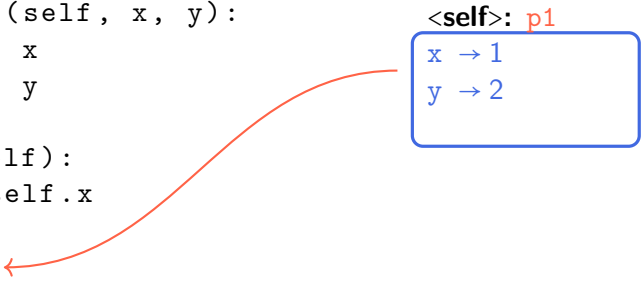
Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10
11
12
13
14
```



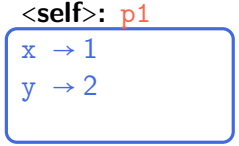
Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10
11
12
13
14
```



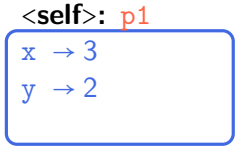
Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11
12
13
14
```



Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12
13
14
```



Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13
14
```

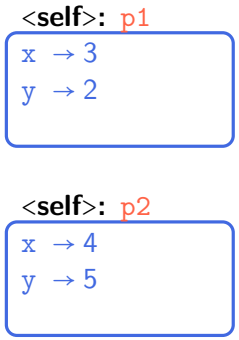
<self>: p1

x → 3

y → 2

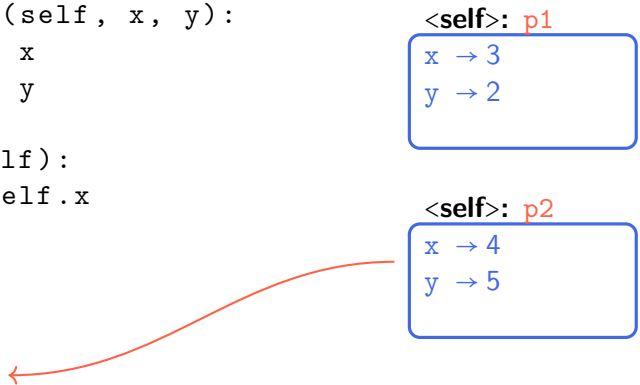
Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13
14
```



Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13
14
```



Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13 print(p2.get_x())
14
```

<self>: p1

x → 3
y → 2

<self>: p2

x → 4
y → 5

Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13 print(p2.get_x())
14
```

<self>: p1

x → 3
y → 2

<self>: p2

x → 4
y → 5

Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13 print(p2.get_x())
14
```

p2

<self>: p1

x → 3
y → 2

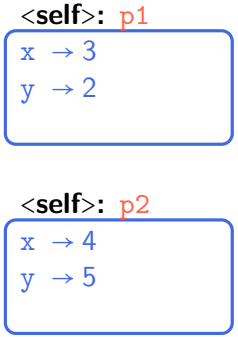
<self>: p2

x → 4
y → 5

Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13 print(p2.get_x())
14
```

p2



Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13 print(p2.get_x())
14 print(p1.get_x())
```

<self>: p1

x → 3

y → 2

<self>: p2

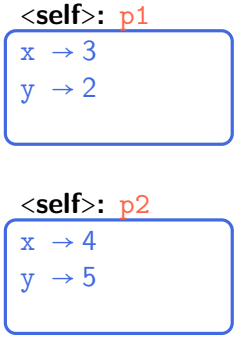
x → 4

y → 5

Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13 print(p2.get_x())
14 print(p1.get_x())
```

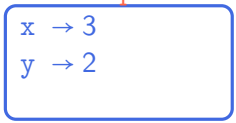
p1



Creating Multiple Objects

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def get_x(self):
7          return self.x
8
9  p1 = Point(1, 2)
10 print(p1.x)
11 p1.x = 3
12 p2 = Point(4, 5)
13 print(p2.get_x())
14 print(p1.get_x())
```

Observe:
Methods have **one** copy. They are shared between different objects. Hence the '**self**' parameter



Big Idea

The methods are created only **once**.
Different objects **share** the same methods by passing themselves as '**self**'.

You Try!

- Create a class `Rectangle` with attributes `length` and `width`.
- Add methods to calculate the area and perimeter of the rectangle.
- Create two rectangle objects and test the methods.

Summary

Questions?