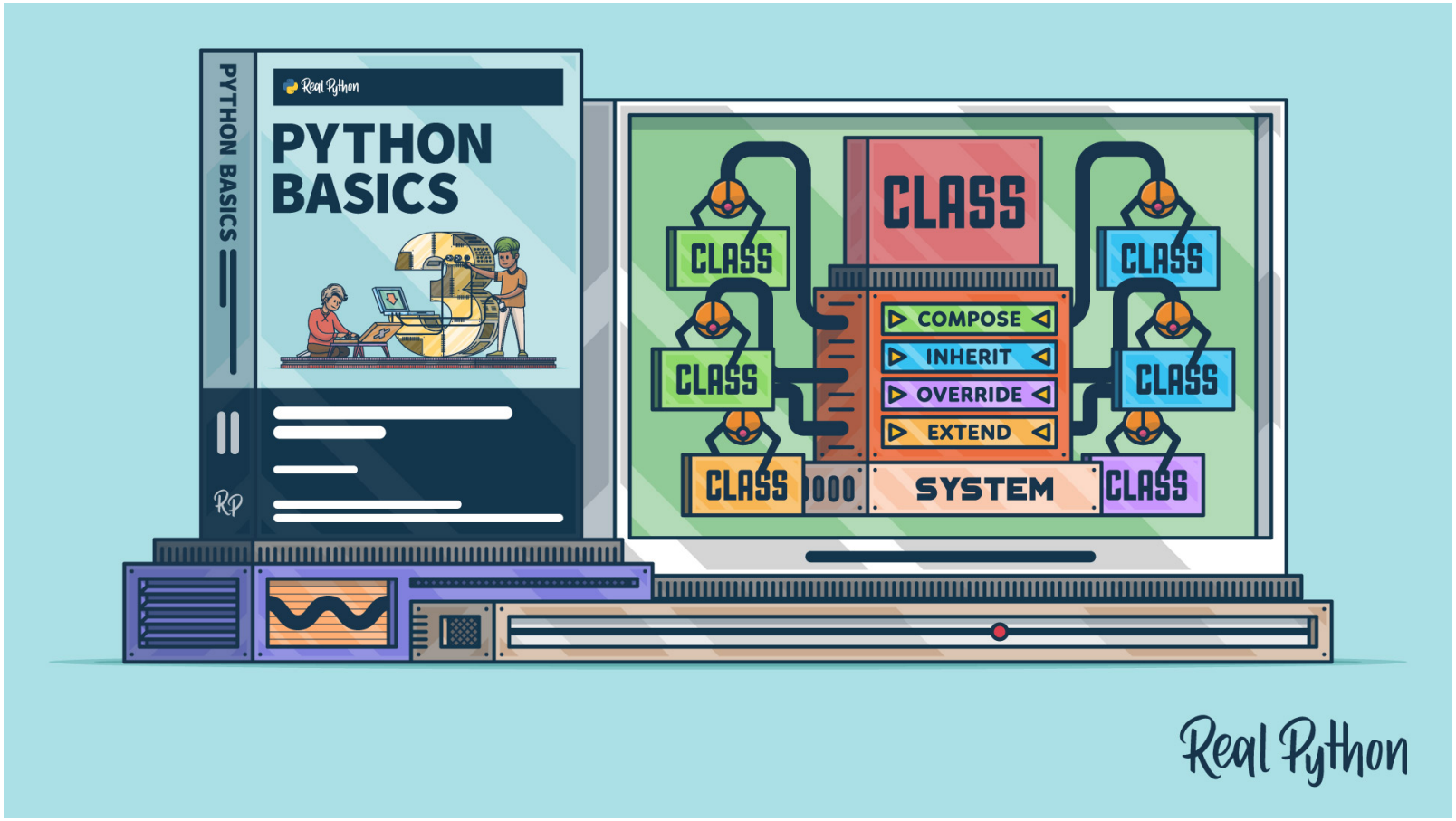
Python Basics: Building Systems With Classes



Python Basics: Building Systems With Classes

How do you use classes when coding?

- You can **compose** classes together
- You can **inherit** and **override** behaviors from other classes
- You can mix and match approaches

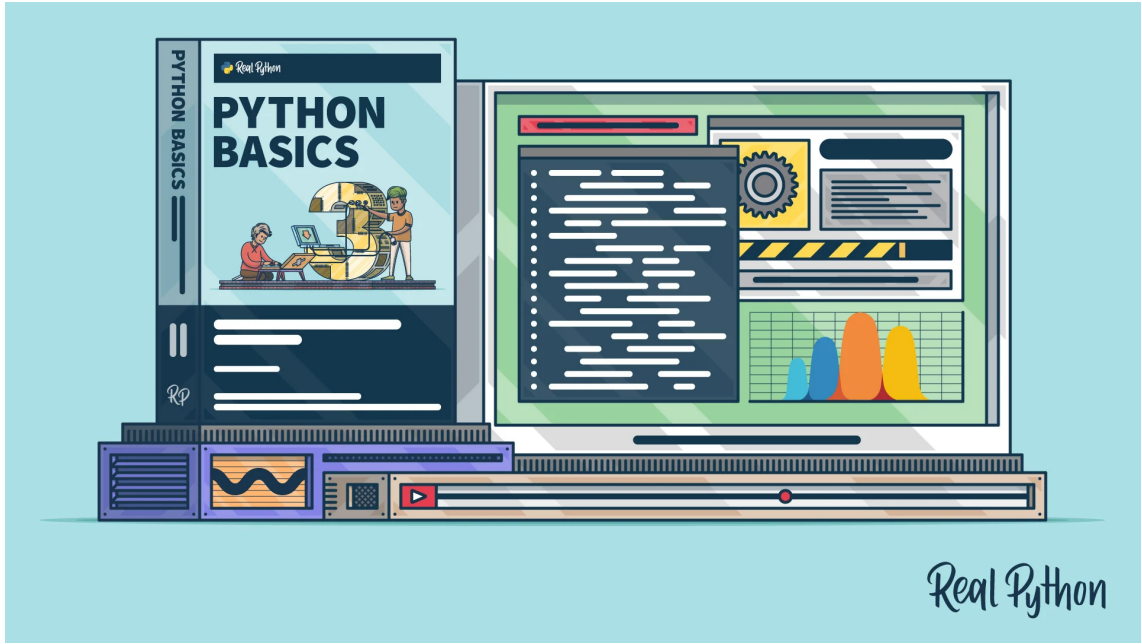
Python Basics: Building Systems With Classes

What you'll need:

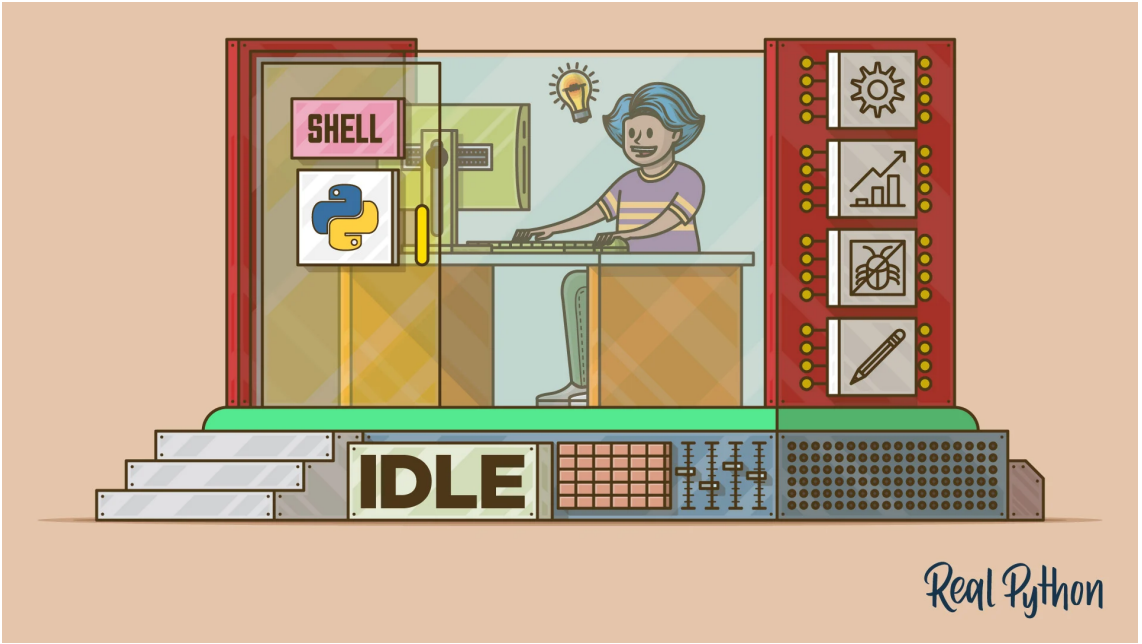
- **IDLE: I**ntegrated **D**evelopment and **L**earning **E**nvironment

Python Basics: Building Systems With Classes

What you'll need:



Python Basics: Setting Up Python



Starting With Python IDLE

Composing With Classes

Composing With Classes

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

Composing With Classes

```
class Shape:
    def __init__(self, points):
        self.points = points
```

Composing With Classes

```
>>> bottom_left = Point(0, 0)
>>> bottom_right = Point(10, 0)
>>> top_left = Point(0, 10)
>>> top_right = Point(10, 10)
```

Composing With Classes

```
>>> square = Shape([bottom_left, bottom_right, top_left, top_right])
```

Composing With Classes

```
>>> square = Shape([bottom_left, bottom_right, top_left, top_right])

>>> square
```

Composing With Classes

```
>>> square = Shape([bottom_left, bottom_right, top_left, top_right])

>>> square
<__main__.Shape at 0x2713d5e2a40>
```

Composing With Classes

```
>>> square = Shape([bottom_left, bottom_right, top_left, top_right])

>>> square
<__main__.Shape at 0x2713d5e2a40>

>>> square.points
```

Composing With Classes

```
>>> square = Shape([bottom_left, bottom_right, top_left, top_right])

>>> square
<__main__.Shape at 0x2713d5e2a40>

>>> square.points
[<__main__.Point at 0x2713d5e3f70>,
 <__main__.Point at 0x2713d4ef9a0>,
 <__main__.Point at 0x2713d5e3130>,
 <__main__.Point at 0x2713d5e2740>]
```

Inheriting From Other Classes

Inheriting From Other Classes

```
class Doggo:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

Inheriting From Other Classes

```
>>> miles = Doggo("Miles", 4)
>>> buddy = Doggo("Buddy", 9)
>>> jack = Doggo("Jack", 3)
>>> jim = Doggo("Jim", 5)
```

Inheriting From Other Classes

```
>>> buddy.speak("Yap")
'Buddy says Yap'
>>> jim.speak("Woof")
'Jim says Woof'
>>> jack.speak("Woof")
'Jack says Woof'
```

Inheriting From Other Classes

```
class JackRussellTerrier(Doggo):
    pass

class Dachshund(Doggo):
    pass

class Bulldog(Doggo):
    pass
```

Inheriting From Other Classes

```
>>> miles = JackRussellTerrier("Miles", 4)
>>> buddy = Dachshund("Buddy", 9)
>>> jack = Bulldog("Jack", 3)
>>> jim = Bulldog("Jim", 5)
```

Inheriting From Other Classes

```
>>> miles.species
'Canis familiaris'

>>> buddy.name
'Buddy'

>>> print(jack)
Jack is 3 years old

>>> jim.speak("Woof")
'Jim says Woof'
```

```
>>> type(miles)
<class '__main__.JackRussellTerrier'>

>>> isinstance(miles, Doggo)
True

>>> isinstance(miles, Bulldog)
False

>>> isinstance(jack, JackRusselTerrier)
True
```

Inheriting From Other Classes

- You can extend classes
  - You have a parent class
  - Then you can subclass it
- You may not need inheritance

Extending a Parent Class

Extending a parent class and overriding or augmenting its methods

Extending a Parent Class

```
class Doggo:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

Extending a Parent Class

```
class JackRussellTerrier(Doggo):
    def speak(self, sound="Arf"):
        return f"{self.name} says {sound}"
```

Extending a Parent Class

```
class JackRussellTerrier(Doggo):
    def speak(self, sound="Arf"):
        return f"{self.name} says {sound}"

>>> miles = JackRussellTerrier("Miles", 4)
```

Extending a Parent Class

```
class JackRussellTerrier(Doggo):
    def speak(self, sound="Arf"):
        return f"{self.name} says {sound}"

>>> miles = JackRussellTerrier("Miles", 4)

>>> miles.speak()
```

Extending a Parent Class

```
class JackRussellTerrier(Doggo):
    def speak(self, sound="Arf"):
        return f"{self.name} says {sound}"

>>> miles = JackRussellTerrier("Miles", 4)

>>> miles.speak()
'Miles says Arf'
```

Extending a Parent Class

```
class JackRussellTerrier(Doggo):
    def speak(self, sound="Arf"):
        return f"{self.name} says {sound}"

>>> miles = JackRussellTerrier("Miles", 4)

>>> miles.speak()
'Miles says Arf'

>>> miles.speak("Grrr")
```

Extending a Parent Class

```
class JackRussellTerrier(Doggo):
    def speak(self, sound="Arf"):
        return f"{self.name} says {sound}"

>>> miles = JackRussellTerrier("Miles", 4)

>>> miles.speak()
'Miles says Arf'

>>> miles.speak("Grrr")
'Miles says Grrr'
```

Extending a Parent Class

- You can override methods from a parent class
- Just subclass and redefine the method
- All the other methods will work as before, including the constructor

Using Composition and Inheritance

Using Composition and Inheritance

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

Using Composition and Inheritance

```
class Shape:
    def __init__(self, points):
        self.points = points
```

Using Composition and Inheritance

```
class WrongNumberOfPoints(Exception):
    pass

class Square(Shape):
    def __init__(self, points):
        if (len(points) != 4):
            raise WrongNumberOfPoints
        self.points = points
```

Using Composition and Inheritance

```
class Triangle(Shape):
    def __init__(self, points):
        if (len(points) != 3):
            raise WrongNumberOfPoints
        self.points = points
```

Introducing the super() Function

- Sometimes you need to extend a certain method
- You can call the parent class's method with `super()`
- Very common to see with the `.__init__()` constructor

Introducing the super() Function

```
class WrongNumberOfPoints(Exception):
    pass

class Point: ...

class Shape:
    def __init__(self, points):



        self.points = points

class Square(Shape):
    def __init__(self, points):
        if (len(points) != 4):
            raise WrongNumberOfPoints
        self.points = points
```

Introducing the super() Function

```
class WrongNumberOfPoints(Exception):
    pass

class Point: ...

class Shape:
    def __init__(self, points):
        for point in points:


        self.points = points

class Square(Shape):
    def __init__(self, points):
        if (len(points) != 4):
            raise WrongNumberOfPoints
        self.points = points
```

Introducing the super() Function

```
class WrongNumberOfPoints(Exception):
    pass

class Point: ...

class Shape:
    def __init__(self, points):
        for point in points:
            if not isinstance(point, Point):

        self.points = points

class Square(Shape):
    def __init__(self, points):
        if (len(points) != 4):
            raise WrongNumberOfPoints
        self.points = points
```

Introducing the super() Function

```
class WrongNumberOfPoints(Exception):
    pass

class Point: ...

class Shape:
    def __init__(self, points):
        for point in points:
            if not isinstance(point, Point):
                raise TypeError("All points should be members of Point class")
        self.points = points

class Square(Shape):
    def __init__(self, points):
        if (len(points) != 4):
            raise WrongNumberOfPoints
        self.points = points
```

Introducing the super() Function

```
class WrongNumberOfPoints(Exception):
    pass

class Point: ...

class Shape:
    def __init__(self, points):
        for point in points:
            if not isinstance(point, Point):
                raise TypeError("All points should be members of Point class")
        self.points = points

class Square(Shape):
    def __init__(self, points):
        if (len(points) != 4):
            raise WrongNumberOfPoints
        super().__init__(points)
```

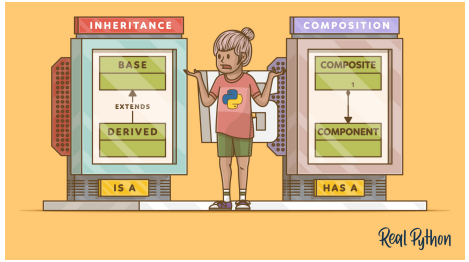
Challenge: Model a Farm

- Model four classes
  - One `Animal` class
  - Three classes that inherit from `Animal`
- Each class should have a few attributes and at least one method
- Model two more classes:
  - One `Field`
  - One `Barn`
- Make the places have an attribute where they can store `Animal` objects.

Python Basics: Building Systems With Classes

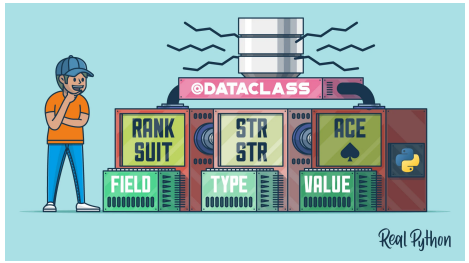
- You can **compose** classes together
- You can **inherit** and **override** behaviors from other classes
- You can mix and match approaches

Additional Resources



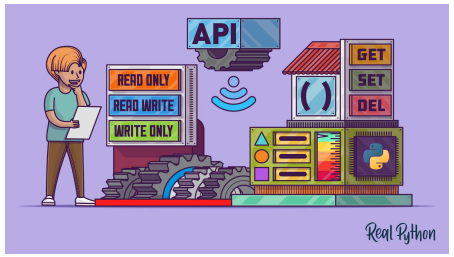
Inheritance and Composition: A Python OOP Guide

Supercharge Your Classes With Python super()

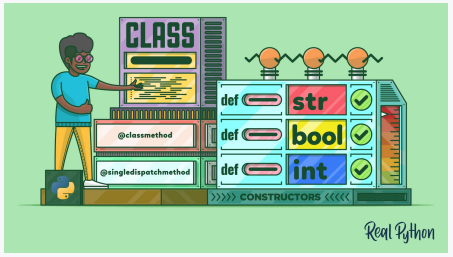


Data Classes in Python 3.7+ (Guide)

Additional Resources



Python's property(): Add Managed Attributes to Your Classes



Providing Multiple Constructors in Your Python Classes

Python Basics: Building Systems With Classes