INHERITANCE AND super() IN PYTHON

What you will learn:

- Introduction to inheritance in Python
- Using the super() function
- Single inheritance and the super() function
- Multiple inheritance and the super() function

Warning: all info in this course is based on Python 3, the same concepts are in Python 2, but the syntax is different in some cases

TABLE OF CONTENTS

OBJECTS AND CLASSES

- An object is a way of grouping data and methods on that data together
- Objects often map to things in the real world:
  - Person: name, address, add_to_course(), save_data()
  - Balance Sheet: assets[], liabilities[], total_assets()
- A class defines how to make an object
- Use a class to create objects

THE SIMPLEST CLASS

```
# in shapes.py
class Square:
    pass

>>> from shapes import Square
>>> square = Square()
>>> square.length = 3
>>> dir(square)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', 'length']
>>> square.__class__
<class '__main__.Square'>
```

ADD PARAMETERS AND METHODS

```python
class Square:
    def __init__(self, length):
        self.length = length

    def area(self):
        return self.length * self.length

    def perimeter(self):
        return 4 * self.length
```

ADD PARAMETERS AND METHODS

```
>>> from shapes import Square
>>> square = Square(3)
>>> square.length
3
>>> square.area()
9
>>> dir(square)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'area', 'length', 'perimeter']
```

ANOTHER SHAPE

```python
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * self.length + 2 * self.width
```

```
>>> from shapes import Rectangle
>>> rectangle = Rectangle(2, 4)
>>> rectangle.area()
8
```

SQUARES ARE SPECIAL RECTANGLES

```python
class Square:
    def __init__(self, length):
        self.length = length

    def area(self):
        return self.length * self.length

    def perimeter(self):
        return 4 * self.length
```

✗

```python
class Square(Rectangle):
    def __init__(self, length):
        super().__init__(length, length)
```

✓

SQUARE WITH INHERITANCE

```python
class Square(Rectangle):
    def __init__(self, length):
        super().__init__(length, length)

>>> from shapes import Square
>>> square = Square(3)
>>> square.__class__
<class 'Square'>
>>> dir(square)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'area', 'length', 'perimeter', 'width']
>>> square.__class__.__bases__
(<class 'Rectangle'>,)
```

Real Python

TABLE OF CONTENTS

ACCESSING METHODS

```python
class Cube(Square):
    # same parameters as Square, no need to redefine __init__

    def surface_area(self):
        face_area = self.area()
        return face_area * 6

    def volume(self):
        face_area = super().area()
        return face_area * self.length
```

```
>>> from shapes import Cube
>>> cube = Cube(3)
>>> cube.surface_area()
54
>>> cube.volume()
27
```

CALLING AN OBJECT'S METHOD

- When you call a method on an object, Python looks for a method with that name on the current object
    - If it finds it, it calls it
    - If it doesn't find it, it tries to find a method with that name in the parent object
    - It keeps going up the inheritance chain until it finds the method or if it never finds it an AttributeError will be thrown

MODIFIED RECTANGLE

```
class Rectangle:
    …
    def what_am_i(self):
        return 'Rectangle'
```

```
>>> from shapes import Rectangle, Square
>>> rectangle = Rectangle(2, 4)
>>> rectangle.what_am_i()
'Rectangle'
>>> square = Square(3)
>>> square.what_am_i()
'Rectangle'
```

MODIFIED SQUARE & CUBE

```
class Square(Rectangle):
    …
    def what_am_i(self):
        return 'Square'

class Cube(Square):
    …
    def what_am_i(self):
        return 'Cube'

>>> from shapes import Rectangle, Square, Cube
>>> rectangle = Rectangle(2, 4)
>>> rectangle.what_am_i()
'Rectangle'
>>> square = Square(3)
>>> square.what_am_i()
'Square'
>>> cube.what_am_i()
‘Cube'
```

FORMS OF super()

- super() called within a class method gives you access to the parent object
- super() can also be called with parameters indicating the class and object to access
    - `super(class, object)`
    - This form doesn't even have to be inside the object method
- Inside a class method "super()" is a shortcut for "super(*my_class*, self)"

ACCESSING A PARENT'S METHODS

```
>>> from shapes import Cube
>>> cube = Cube(3)
>>> cube.what_am_i()
'Cube'
>>> super(Cube, cube).what_am_i()
'Square'
>>> super(Square, cube).what_am_i()
'Rectangle'
```

PARENT METHODS INSIDE AN OBJECT

```
class Cube(Square):
    def family_tree(self):
        # super() is a shortcut for super(Cube, self)
        return self.what_am_i() + ' child of ' + super().what_am_i()

>>> from shapes import Cube
>>> cube = Cube(3)
>>> cube.family_tree()
'Cube child of Square'
```

TABLE OF CONTENTS

A NEW BASE SHAPE

```python
# in shapes.py
class Triangle:
    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

    def what_am_i(self):
        return 'Triangle'
```

MULTIPLE INHERITANCE

```python
class RightPyramid(Triangle, Square):
    def __init__(self, base, slant_height):
        self.base = base
        self.slant_height = slant_height

    def what_am_i(self):
        return 'RightPyramid'
```

```
>>> from shapes import RightPyramid
>>> rightpyramid = RightPyramid(2, 4)
>>> super(RightPyramid, rightpyramid).what_am_i()
'Triangle'
>>> rightpyramid.__class__
<class 'shapes.RightPyramid'>
>>> rightpyramid.__class__.__bases__
(<class 'shapes.Triangle'>, <class 'shapes.Square'>)
>>> RightPyramid.__mro__
(<class 'shapes.RightPyramid'>, <class 'shapes.Triangle'>, <class 'shapes.Square'>, <class
'shapes.Rectangle'>, <class ‘object'>)
```

METHOD RESOLUTION ORDER

- MRO dictates the order of name look-up
- Your inheriting classes must co-operate
- Solutions to method name clashes:
  - Re-write code so there are no name clashes:

    ```
    square.square_area() & triangle.triangle_area()
    ```
  - Careful use of inheritance declaration:

    ```
    class RightPyramid(Triangle, Square):
      vs
    class RightPyramid(Square, Triangle):
    ```
  - Directly access the class to make a call

    ```
    Square.area(self)
    ```

MRO AND MULTIPLE INHERITANCE

```python
# in chain.py
class A:
    def __init__(self):
        print('A')
        super().__init__()

class B(A):
    def __init__(self):
        print('B')
        super().__init__()

class X:
    def __init__(self):
        print('X')
        super().__init__()

class Forward(B, X):
    def __init__(self):
        print('Forward')
        super().__init__()
```

```python
class Backward(X, B):
    def __init__(self):
        print('Backward')
        super().__init__()
```

```
>>> from chain import Forward, Backward
>>> forward = Forward()
Forward
B
A
X
>>> backward = Backward()
Backward
X
B
A
```

ALL THE super()

```python
class Rectangle:
    def __init__(self, length, width, **kwargs):
        self.length = length
        self.width = width
        super().__init__(**kwargs)

class Square(Rectangle):
    def __init__(self, length, **kwargs):
        super().__init__(length=length, width=length, **kwargs)

class Triangle:
    def __init__(self, base, height, **kwargs):
        self.base = base
        self.height = height
        super().__init__(**kwargs)

class RightPyramid(Square, Triangle):
    def __init__(self, base, slant_height, **kwargs):
        self.base = base
        self.slant_height = slant_height
        kwargs["height"] = slant_height
        kwargs["length"] = base
        super().__init__(base=base, **kwargs)
```

MIXINS

```python
class SurfaceAreaMixin:
    def surface_area(self):
        surface_area = 0
        for surface in self.surfaces:
            surface_area += surface.area(self)

        return surface_area

class RightPyramid(Square, Triangle,
        SurfaceAreaMixin):
    def __init__(self, base, slant_height):
        self.base = base
        self.slant_height = slant_height
        self.height = slant_height
        self.length = base
        self.width = base

        self.surfaces = [Square, Triangle,
            Triangle, Triangle, Triangle]
```

```python
class Cube(Square, SurfaceAreaMixin):
    def __init__(self, length):
        super().__init__(length)
        self.surfaces = [Square, Square,
            Square, Square, Square, Square]

>>> from shapes import Cube
>>> cube = Cube(3)
>>> cube.surface_area()
54
```

INHERITANCE AND super() IN PYTHON

- Overview of how objects and classes work in python
- Object inheritance
- How to use super() with single inheritance
- Multiple inheritance and the Method Resolution Order in Python
- Complexities of multiple inheritance
- Mixins