Dispatching on Type

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

```
square = Square(2, 6, 7)
circle = Circle(2, 6, 7)


draw(square)
draw(circle)



draw_square(square)
draw_circle(circle)



square.draw()
circle.draw()
```

Given objects of different types...

functions can't be overloaded on argument type – `draw()` would need internal logic to detect the type and act accordingly

Use distinct function names?

The implementation selected depends on the type of the object giving polymorphism

Drawing Shapes

Draw shapes in Scalable Vector Graphics (SVG)

How to organize the drawing code?

Methods - dispatch on type

Poor separation of concerns, so refactor!

- Switch emulation
- Reflection/introspection
- The `@singledispatch` decorator

Implementing multiple dispatch

shapes > shapes.svg

shapes

Project

shapes

External Libraries

Scratches and Consoles

shapes.py | shapes.svg

```
1  <svg viewBox="300 10 400 415" xmlns="http://www.w3.org/2000/
2  <polygon points="300 425 580 425 560 155 510 155 500 285 450
3  <rect x="430" y="330" width="50" height="40" stroke="#2a9fbc
4  <rect x="350" y="330" width="50" height="40" stroke="#2a9fbc
5  <g>
6  <circle cx="550" cy="120" r="30" fill="#d8d8d8" />
7  <circle cx="600" cy="90" r="40" fill="#d8d8d8" />
8  <circle cx="650" cy="60" r="50" fill="#d8d8d8" />
9  </g>
10 </svg>
11
```

1:1

16x16 SVG (32-bit color) 593 B

Run: shapes

```
/Users/rjs/.virtualenvs/shapes/bin/python /Users/rjs/visning-servers/core-python-advanced-flow-control/build/build/shape

Process finished with exit code 0
```

1:1

Refactoring to Separate Concerns

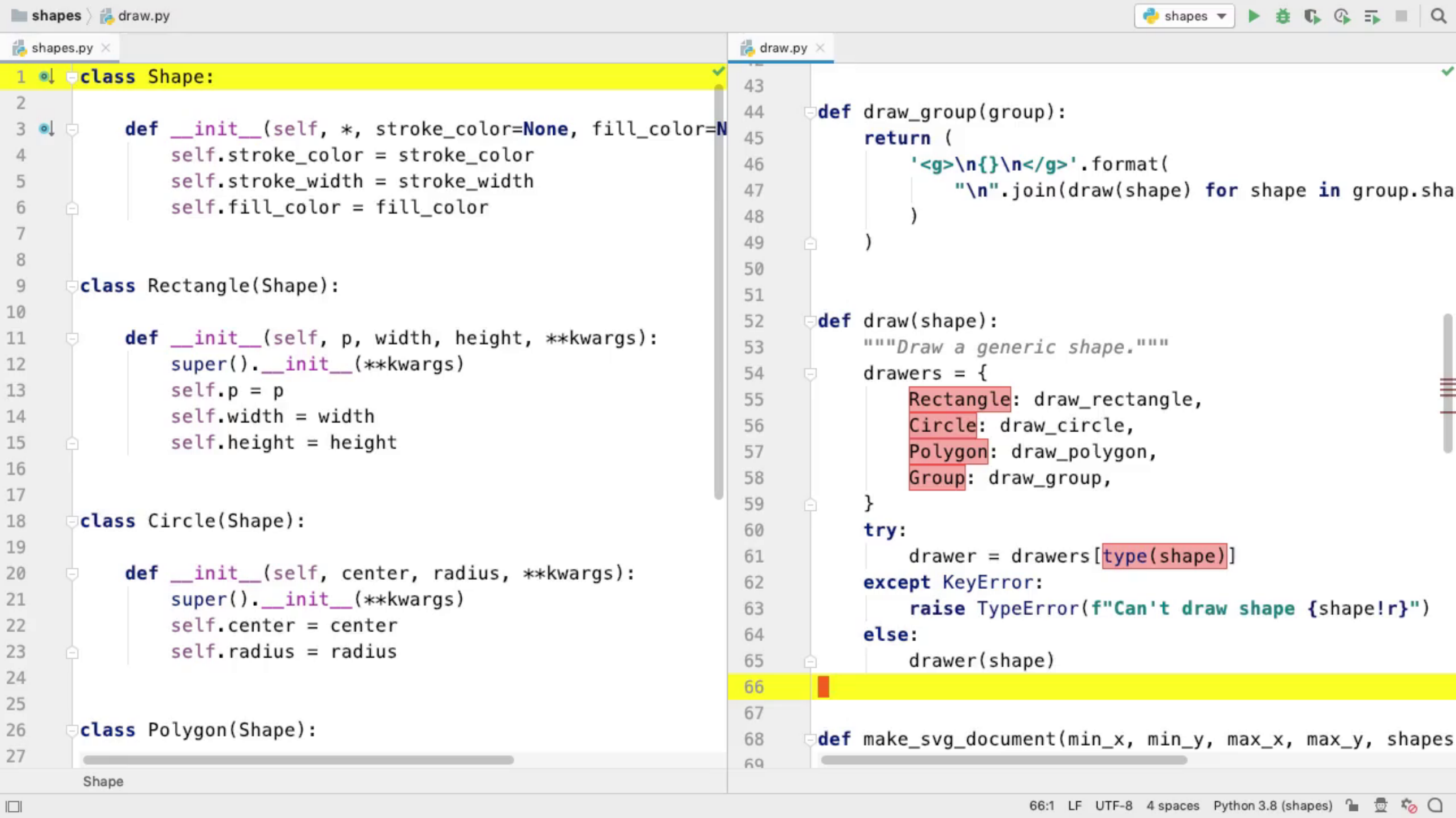
shapes.py

```
17
18  class Circle(Shape):
19
20      def __init__(self, center, radius, **kwargs):
21          super().__init__(**kwargs)
22          self.center = center
23          self.radius = radius
24
25
26  class Polygon(Shape):
27
28      def __init__(self, points, **kwargs):
29          super().__init__(**kwargs)
30          self.points = points
31
32
33  class Group:
34
35      def __init__(self, shapes):
36          self.shapes = shapes
37
```

draw.py

```
43
44  def draw_group(group):
45      return (
46          '<g>\n{}\n</g>'.format(
47              "\n".join(draw(shape) for shape in group.sha
48          )
49      )
50
51
52  def draw(shape):
53      """Draw a generic shape."""
54      if isinstance(shape, Rectangle):
55          draw_rectangle(shape)
56      elif isinstance(shape, Circle):
57          draw_circle(shape)
58      elif isinstance(shape, Polygon):
59          draw_polygon(shape)
60      elif isinstance(shape, Group):
61          draw_group(shape)
62      else:
63          raise TypeError(f"Can't draw shape {shape!r}")
64
65
66  def make_svg_document(min_x, min_y, max_x, max_y, shapes
67      """Make an SVG document from a collection of shapes.
68      return (
69          '<svg viewBox="{min_x} {min_y} {width} {height}"
```

Dictionary Dispatch


shapes
draw.py
shapes
shapes.py
draw.py
class Shape:
def __init__(self, *, stroke_color=None, fill_color=N
self.stroke_color = stroke_color
self.stroke_width = stroke_width
self.fill_color = fill_color
class Rectangle(Shape):
def __init__(self, p, width, height, **kwargs):
super().__init__(**kwargs)
self.p = p
self.width = width
self.height = height
class Circle(Shape):
def __init__(self, center, radius, **kwargs):
super().__init__(**kwargs)
self.center = center
self.radius = radius
class Polygon(Shape):
Shape
def draw_group(group):
return (
'<g>\n{}\n</g>'.format(
"\n".join(draw(shape) for shape in group.sha
)
)
def draw(shape):
"""Draw a generic shape."""
drawers = {
Rectangle: draw_rectangle,
Circle: draw_circle,
Polygon: draw_polygon,
Group: draw_group,
}
try:
drawer = drawers[type(shape)]
except KeyError:
raise TypeError(f"Can't draw shape {shape!r}")
else:
drawer(shape)
def make_svg_document(min_x, min_y, max_x, max_y, shapes
66:1 LF UTF-8 4 spaces Python 3.8 (shapes)

Introspective Lookup


shapes
draw.py
shapes.py
draw.py
class Shape:
def __init__(self, *, stroke_color=None, fill_color=N
self.stroke_color = stroke_color
self.stroke_width = stroke_width
self.fill_color = fill_color
class Rectangle(Shape):
def __init__(self, p, width, height, **kwargs):
super().__init__(**kwargs)
self.p = p
self.width = width
self.height = height
class Circle(Shape):
def __init__(self, center, radius, **kwargs):
class Polygon(Shape):
Fragile with respect to refactoring
def draw_polygon(polygon):
return '<polygon points="{points}" {attrs} />'.forma
points=" ".join(f"{p[0]} {p[1]}" for p in polygo
attrs=attrs(polygon)
def draw_group(group):
return (
'<g>\n{}\n</g>'.format(
"\n".join(draw(shape) for shape in group.sha
def draw(shape):
"""Draw a generic shape."""
suffix = type(shape).__name__.lower()
drawer_name = f"draw_{suffix}"
try:
drawer = globals()[drawer_name]
except KeyError:
raise TypeError(f"Can't draw shape {shape!r}")
else:
drawer(shape)
Shape
make_svg_document()
66:1
LF
UTF-8
4 spaces
Python 3.8 (shapes)

The `@singledispatch` Decorator

The `functools` Standard Library Module

```
from functools import singledispatch
```

Core Python:
Functions and Functional Programming

on

PLURALSIGHT

Generic Functions

Multiple functions with the same name

Distinguished by the type of argument

Each version is an overload

We talk of overloading a function

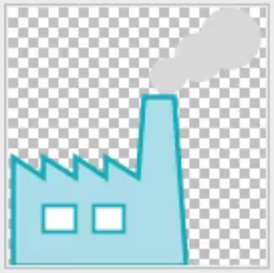
Common terms in statically typed languages

shapes 〉 draw.py

shapes.py | shapes.svg

```svg
<svg viewBox="300 10 400 415" xmlns=
<polygon points="300 425 580 425 560
<rect x="430" y="330" width="50" heig
<rect x="350" y="330" width="50" heig
<g>
<circle cx="550" cy="120" r="30" fill
<circle cx="600" cy="90" r="40" fill=
<circle cx="650" cy="60" r="50" fill=
</g>
</svg>
```

2-bit color) 593 B



draw.py

```python
    )


@draw.register(Group)
def _(group):
    return (
        '<g>\n{}\n</g>'.format(
            "\n".join(draw(shape) for shape in group.sha
        )
    )


def make_svg_document(min_x, min_y, max_x, max_y, shapes
    """Make an SVG document from a collection of shapes.
    return (
        '<svg viewBox="{min_x} {min_y} {width} {height}"
```

Run: draw

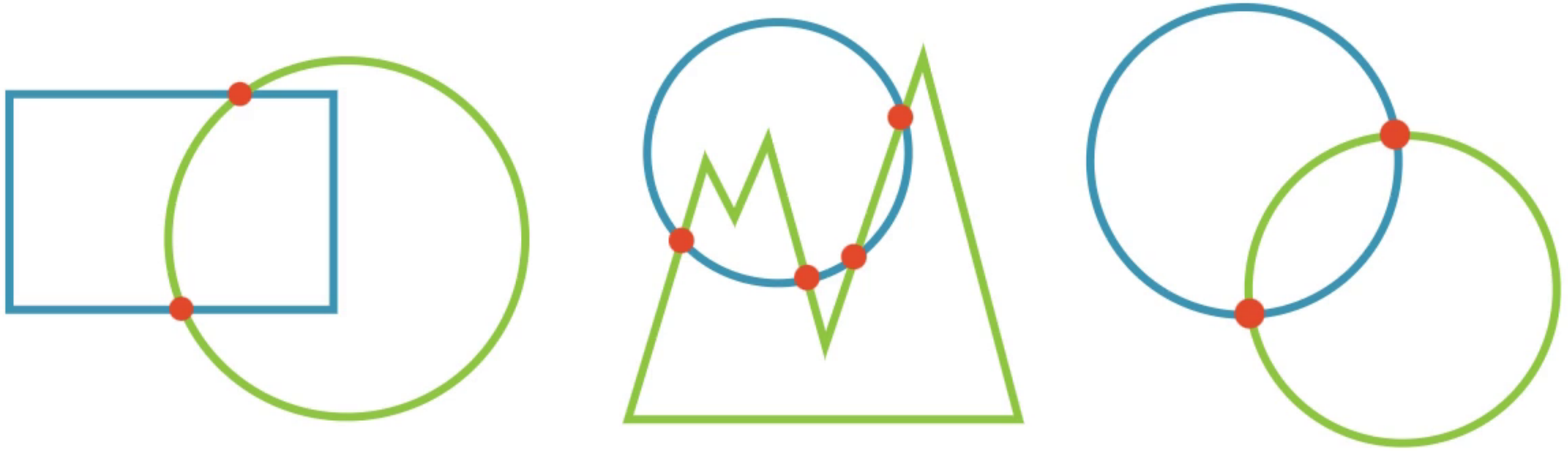
```
/Users/rjs/.virtualenvs/shapes/bin/python /Users/rjs/visning-servers/core-python-advanced-flow-control/build/build/shape

Process finished with exit code 0
```

Do one thing and do it well.

Overloading Methods

Shape to Shape Intersections

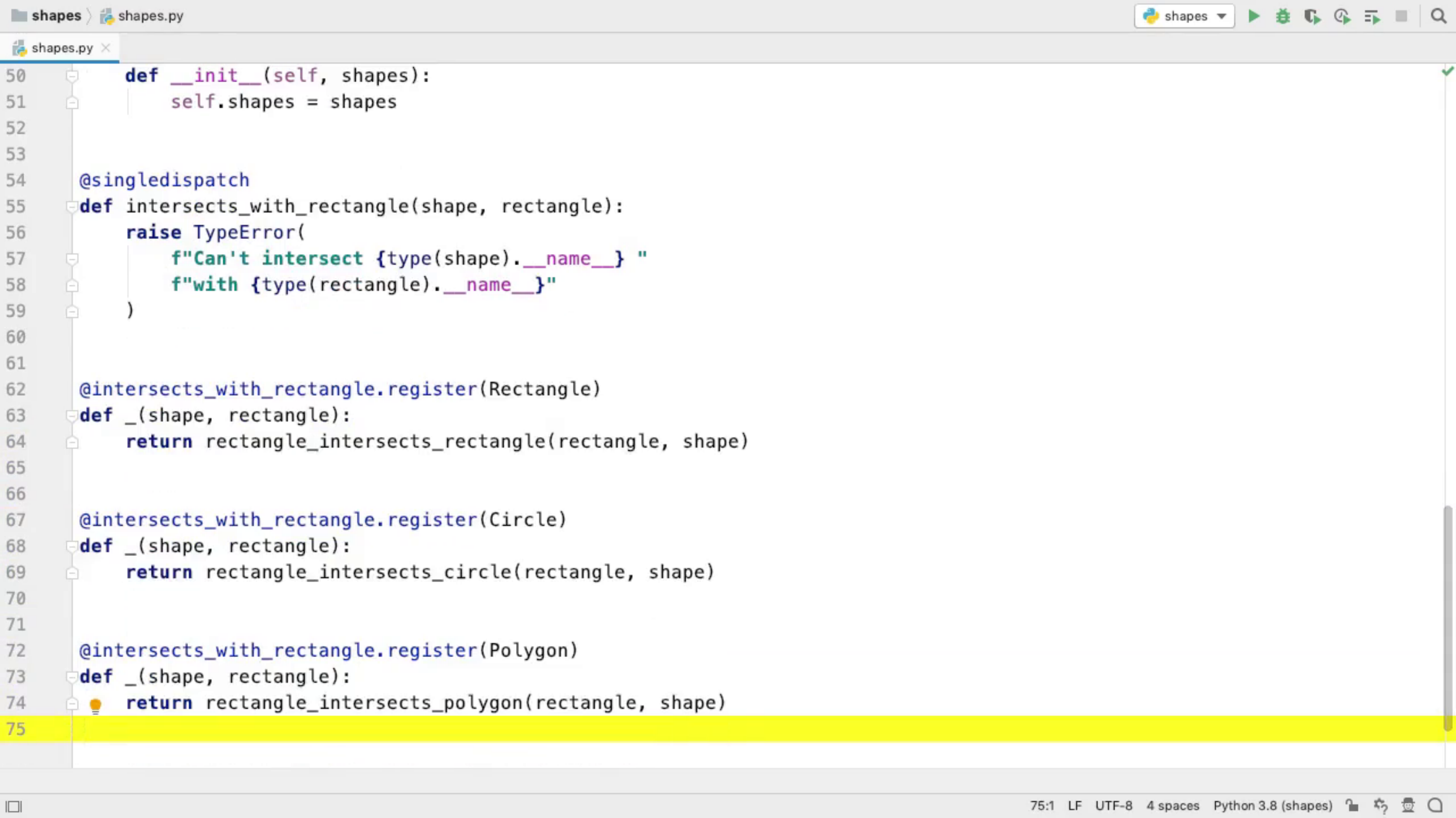


Select an Algorithm Based on Argument Types

```
intersects(circle, polygon)

intersects(rectangle, polygon)

intersects(shape_a, shape_b)
```



```
    def __init__(self, shapes):
        self.shapes = shapes


@singledispatch
def intersects_with_rectangle(shape, rectangle):
    raise TypeError(
        f"Can't intersect {type(shape).__name__} "
        f"with {type(rectangle).__name__}"
    )


@intersects_with_rectangle.register(Rectangle)
def _(shape, rectangle):
    return rectangle_intersects_rectangle(rectangle, shape)


@intersects_with_rectangle.register(Circle)
def _(shape, rectangle):
    return rectangle_intersects_circle(rectangle, shape)


@intersects_with_rectangle.register(Polygon)
def _(shape, rectangle):
    return rectangle_intersects_polygon(rectangle, shape)

```

Equivalent Method Invocations

```
d = my_rect.intersects(my_circle)

d = type(my_rect).intersects(my_rect, my_circle)

d = Rectangle.intersects(self=my_rect, shape=my_circle)
```

Overloading Methods

- The `singledispatch` decorator does not work with methods
- Relocate the generic function to global scope
- **Delegate** from a regular method to the generic function, **reversing the arguments**

Multiple Dispatch

```python
@singledispatch
def intersects_with_polygon(shape, polygon):
    raise TypeError(
        f"Can't intersect {type(shape).__name__} "
        f"with {type(polygon).__name__}"
    )


@intersects_with_polygon.register(Rectangle)
def _(shape, polygon):
    return rectangle_intersects_polygon(shape, polygon)


@intersects_with_polygon.register(Circle)
def _(shape, polygon):
    return circle_intersects_polygon(shape, polygon)


@intersects_with_polygon.register(Polygon)
def _(shape, polygon):
    return polygon_intersects_polygon(polygon, shape)


def intersects(shape_a, shape_b):
    return shape_a.intersects(shape_b)

```