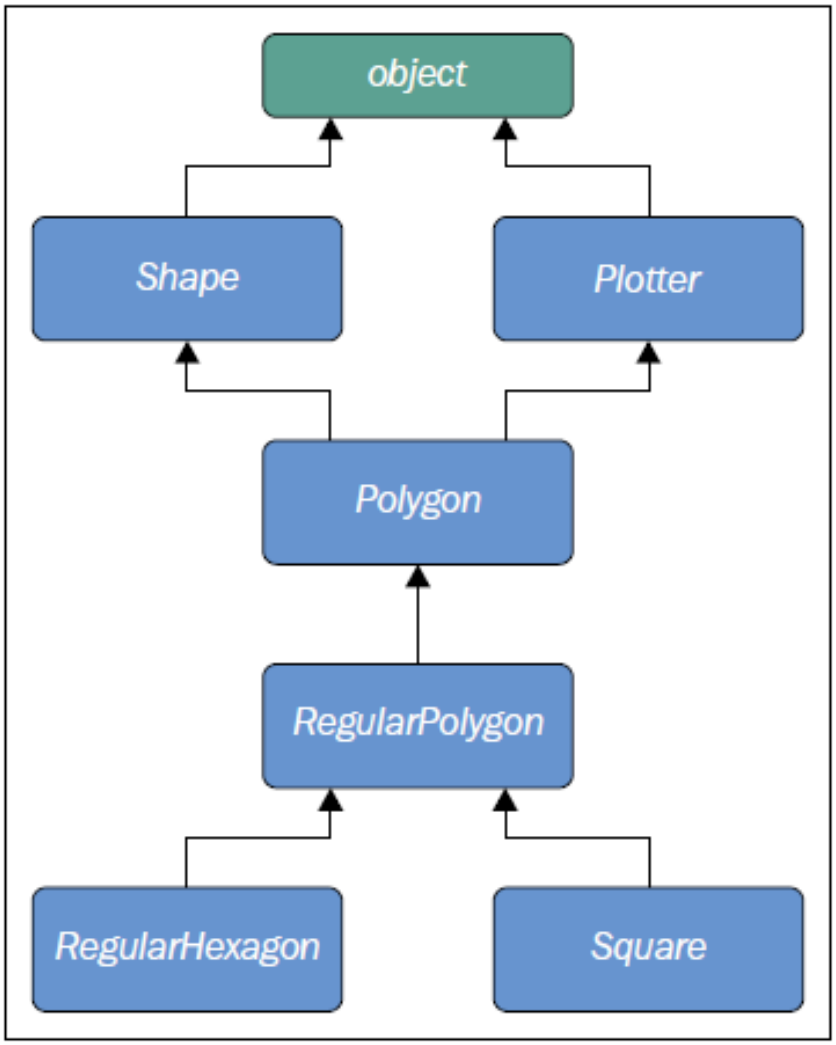
CES 22 – aula 6

Herança Múltipla, Decoradores

Objetivos

- Herança múltipla
- Decoradores

Herança Múltipla



```
class Shape:
    geometric_type = 'Generic Shape'

    def area(self):  # This acts as placeholder for the interface
        raise NotImplementedError

    def get_geometric_type(self):
        return self.geometric_type

class Plotter:

    def plot(self, ratio, topleft):
        # Imagine some nice plotting logic here...


        print('Plotting at {}, ratio {}.'.format(
            topleft, ratio))

class Polygon(Shape, Plotter):  # base class for polygons
    geometric_type = 'Polygon'
```

```
class RegularPolygon(Polygon):  # Is-A Polygon
    geometric_type = 'Regular Polygon'

    def __init__(self, side):
        self.side = side

class RegularHexagon(RegularPolygon): # Is-A RegularPolygon
    geometric_type = 'RegularHexagon'

    def area(self):
        return 1.5 * (3 ** .5 * self.side ** 2)

class Square(RegularPolygon):  # Is-A RegularPolygon
    geometric_type = 'Square'

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

hexagon = RegularHexagon(10)
print(hexagon.area())  # 259.8076211353316
print(hexagon.get_geometric_type())  # RegularHexagon
hexagon.plot(0.8, (75, 77))  # Plotting at (75, 77), ratio 0.8.

square = Square(12)
print(square.area())  # 144
print(square.get_geometric_type())  # Square
square.plot(0.93, (74, 75))  # Plotting at (74, 75), ratio 0.93.
```

MRO (Method Resolution Order)

```
print(square.__class__.__mro__)
# prints:
# (<class '__main__.Square'>, <class '__main__.RegularPolygon'>,
#  <class '__main__.Polygon'>, <class '__main__.Shape'>,
#  <class '__main__.Plotter'>, <class 'object'>)
```

```
class A:
    label = 'a'

class B(A):
    label = 'b'

class C(A):

    label = 'c'

class D(B, C):
    pass

d = D()
print(d.label)  # Hypothetically this could be either 'b' or 'c'
```

```
print(d.__class__.mro())  # notice another way to get the MRO
# prints:
# [<class '__main__.D'>, <class '__main__.B'>,
#  <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
```

Exercício

- Crie extensões no exemplo de classes de polígonos regulares de modo a avaliar os diferentes MROs possíveis.

Decoradores

```
#Funcoes podem ser atribuidas a variaveis

def greet(name):
  return "hello "+name
greet_someone = greet
print (greet_someone("John"))

# Outputs: hello John


#Funcoes podem ser declaradas dentro de funcoes

def greet(name):
    def get_message():
        return "Hello "
    result = get_message()+name
    return result
print (greet("John") )
# Outputs: Hello John
```

```
# funcoes podem ser passadas como parametros para outras funcoes

def greet(name):
    return "Hello " + name
def call_func(func):
    other_name = "John"
    return func(other_name)
print (call_func(greet))
# Outputs: Hello John

#funcoes podem retornar outras funcoes

def compose_greet_func():
    def get_message():
        return "Hello there!"
    return get_message
greet = compose_greet_func()
print (greet() )
# Outputs: Hello there!
```

#funcoes internas possuem acesso de leitura para o escopo externo
Esta propriedade é conhecida como fechamento (Closure)

```
def compose_greet_func(name):
    def get_message():
        return "Hello there "+name+"!“
    return get_message
greet = compose_greet_func("John")
print (greet() )
# Outputs: Hello there John!
```

Decoradores são embrulhos (Wrappers) para funções

```
def get_text(name):
    return "lorem ipsum, {0} dolor sit
amet".format(name)

def p_decorate(func):
    def func_wrapper(name):
        return "<p>{0}</p>".format(func(name))
    return func_wrapper

my_get_text = p_decorate(get_text)
print (my_get_text("John") )
# <p>Outputs lorem ipsum, John dolor sit
amet</p>
```

```
get_text = p_decorate(get_text)
print (get_text("John"))
# Outputs lorem ipsum, John dolor sit
amet
```

```
def p_decorate(func):
    def func_wrapper(name):
        return "<p>{0}</p>".format(func(name))

    return func_wrapper

@p_decorate
def get_text(name):
    return "lorem ipsum, {0} dolor sit
amet".format(name)

print (get_text("John"))
```

```
# Outputs <p>lorem ipsum, John dolor sit
amet</p>
```

```
def p_decorate(func):
   def func_wrapper(name):
       return "<p>{0}</p>".format(func(name))
   return func_wrapper

def strong_decorate(func):
    def func_wrapper(name):
        return
"<strong>{0}</strong>".format(func(name))
    return func_wrapper

def div_decorate(func):
    def func_wrapper(name):
        return
"<div>{0}</div>".format(func(name))
    return func_wrapper
```

```
get_text = div_decorate(p_decorate(strong_decorate(get_text)))
```

```
@div_decorate
@p_decorate
@strong_decorate
def get_text(name):
   return "lorem ipsum, {0} dolor sit amet".format(name)

print get_text("John")

# Outputs <div><p><strong>lorem ipsum, John dolor sit
amet</strong></p></div>
```

```
def p_decorate(func):
   def func_wrapper(self):
       return
"<p>{0}</p>".format(func(self))
   return func_wrapper

class Person(object):
    def __init__(self):
        self.name = "John"
        self.family = "Doe"

    @p_decorate
    def get_fullname(self):
        return self.name+" "+self.family

my_person = Person()
print (my_person.get_fullname())
```

```
def p_decorate(func):
   def func_wrapper(*args, **kwargs):
       return "<p>{0}</p>".format(func(*args, **kwargs))
   return func_wrapper

class Person(object):
    def __init__(self):
        self.name = "John"
        self.family = "Doe"

    @p_decorate
    def get_fullname(self):
        return self.name+" "+self.family

my_person = Person()

print (my_person.get_fullname())
```

Argumentos para funções

```
def cheeseshop(kind, *arguments,
**keywords):
    print ("-- Do you have any", kind, "?")
    print ("-- I'm sorry, we're all out of",
kind)
    for arg in arguments:
        print (arg)
    print ("-" * 40)
    keys = sorted(keywords.keys())
    for kw in keys:
        print (kw, ":", keywords[kw])
```

```
cheeseshop("Limburger", "It's very runny,
sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper='Michael Palin',
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
---------------------------------------
client : John Cleese
shopkeeper : Michael Palin
sketch : Cheese Shop Sketch
```

```
def tags(tag_name):
    def tags_decorator(func):
        def func_wrapper(name):
            return "<{0}>{1}</{0}>".format(tag_name,
func(name))
        return func_wrapper
    return tags_decorator

@tags("p")
def get_text(name):
    return "Hello "+name

print (get_text("John"))

# Outputs <p>Hello John</p>
```

Referencias

- http://thecodeship.com/patterns/guide-to-python-function-decorators/

Exercícios

- Pesquise o assunto Decoradores. Desenvolva um Decorador exemplo.
- Crie um exemplo de função com lista de argumentos e dicionário de argumentos.