lecture_13

September 27, 2022

1 Lecture 13

1.1 Polymorphism

[1]:
```
def foo(x):
    return x**2


def foo(x, y):
    return x**y
```

[2]:
```
foo(2, 4)
```

[2]:
```
16
```

[3]:
```
foo(2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 foo(2)

TypeError: foo() missing 1 required positional argument: 'y'
```

[4]:
```
def foo(x, y=None):
    return x**2 if y is None else x**y
```

[5]:
```
foo(2, 4)
```

[5]:
```
16
```

[6]:
```
foo(2)
```

[6]:
```
4
```

[7]:
```
def foo(*args):
    if len(args) == 0:
        raise TypeError("at lease one argument required")
    elif len(args) == 1:
        return args[0]**2
    else:
        return args[0]**args[1]
```

[8]:
```
foo(2, 4)
```

[8]:
```
16
```

[9]:
```
foo(2)
```

[9]:
```
4
```

[10]:
```
def foo(*args):
    def _foo_square(x):
        return x**2

    def _foo_power(x, y):
        return x**y

    if len(args) == 1:
        return _foo_square(*args)
    elif len(args) == 2:
        return _foo_power(*args)
    else:
        raise TypeError("at lease one argument required")
```

[11]:
```
foo(2, 4)
```

[11]:
```
16
```

[12]:
```
foo(2)
```

[12]:
```
4
```

[14]:
```
class Foo:
    def bar(self, *args):
        if len(args) == 1:
            return self.__foo_square(*args)
        elif len(args) == 2:
            return self.__foo_power(*args)
        else:
            raise TypeError("at lease one argument required")

    @staticmethod
    def __foo_square(x):
        return x**2

    @staticmethod
    def __foo_power(x, y):
        return x**y
```

[15]:
```
a = Foo()
```

[16]:
```
a.bar(2)
```

[16]:
```
4
```

[17]:
```
a.bar(4, 2)
```

[17]:
```
16
```

[18]:
```
def foo(x):
    if isinstance(x, int):
        return x + 1
    elif isinstance(x, str):
        return x + "1"
    else:
        raise TypeError()
```

[19]:
```
foo(1)
```

[19]:
```
2
```

[20]:
```
foo("1")
```

[20]:
```
'11'
```

1.2 Abstraction

[22]:
```
from functools import total_ordering
```

[23]:
```
@total_ordering
class Shape:
    def area(self):
        raise NotImplementedError()

    @staticmethod
    def calculate_area(*args):
        raise NotImplementedError()

    def __lt__(self, other):
        return self.area() < other.area()
    def __eq__(self, other):
        return self.area() == other.area()

    def __bool__(self):
        return self.area() > 0

    def __add__(self, other):
        if isinstance(other, (int, float)):
            return self.area() + other
        elif isinstance(other, self.__class__):
            return self.area() + other.area()
        else:
            raise TypeError(f"Can't add Rectangle with {other.__class__}")

    def __radd__(self, other):
        return self.__add__(other)

    def __iadd__(self, other):
        raise TypeError("+= not supported for this object")

    def __int__(self):
        return int(self.area())

    def __repr__(self):
        value_mapping = [f'{attr_name}={attr_value}' for attr_name, attr_value␣
 ↪in vars(self).items()]
        return f"{self.__class__.__name__}({', '.join(value_mapping)})"
```

[24]:
```
a = Shape()
```

[25]:
```
a.area()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Input In [25], in <cell line: 1>()
----> 1 a.area()

Input In [23], in Shape.area(self)
      3 def area(self):
----> 4     raise NotImplementedError()

NotImplementedError:
```

[26]:
```
class Rectangle(Shape):
    def __init__(self, width, length):
        if width < 0 or length < 0:
            raise Exception("width and length should be positive numbers")
        self.width = width
        self.length = length

    def area(self):
        return self.calculate_area(self.width, self.length)

    @staticmethod
    def calculate_area(width, length):
        return width * length
```

[27]:
```
import math


class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return self.calculate_area(self.radius)

    @staticmethod
    def calculate_area(radius):
        return math.pi * radius **2
```

[28]:
```
class Square(Rectangle):
    def __init__(self, width):
        super().__init__(width, width)
```

[29]:
```
from abc import ABC, abstractmethod
```

[36]:
```
class Shape(ABC):
    @abstractmethod
    def area(self):
        return NotImplemented
```

[37]:
```
a = Shape()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [37], in <cell line: 1>()
----> 1 a = Shape()

TypeError: Can't instantiate abstract class Shape with abstract method area
```

[38]:
```
class Rectangle(Shape):
    def __init__(self, width, length):
        self.width = width
        self.length = length

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

[39]:
```
a = Rectangle(10, 20)
```

[40]:
```
a.area()
```

[40]:
```
200
```

[41]:
```
from abc import abstractclassmethod

@total_ordering
class Shape(ABC):
    @abstractmethod
    def area(self):
        raise NotImplementedError()

    @abstractclassmethod
    def calculate_area(*args):
        raise NotImplementedError()

    def __lt__(self, other):
        return self.area() < other.area()

    def __eq__(self, other):
        return self.area() == other.area()

    def __bool__(self):
        return self.area() > 0

    def __add__(self, other):
        if isinstance(other, (int, float)):
            return self.area() + other
        elif isinstance(other, self.__class__):
            return self.area() + other.area()
        else:
            raise TypeError(f"Can't add Rectangle with {other.__class__}")

    def __radd__(self, other):
        return self.__add__(other)

    def __iadd__(self, other):
        raise TypeError("+= not supported for this object")

    def __int__(self):
        return int(self.area())

    def __repr__(self):
        value_mapping = [f'{attr_name}={attr_value}' for attr_name, attr_value␣
↪in vars(self).items()]
        return f"{self.__class__.__name__}({', '.join(value_mapping)})"
```

[42]:
```
a = Shape()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [42], in <cell line: 1>()
----> 1 a = Shape()

TypeError: Can't instantiate abstract class Shape with abstract methods area,␣
 ↪calculate_area
```

[43]:
```
class Rectangle(Shape):
    def __init__(self, width, length):
        if width < 0 or length < 0:
            raise Exception("width and length should be positive numbers")
        self.width = width
        self.length = length

    def area(self):
        return self.calculate_area(self.width, self.length)

    @staticmethod
    def calculate_area(width, length):
        return width * length
```

[44]:
```
a = Rectangle(10, 20)
```

[45]:
```
a.area()
```

[45]:
```
200
```

[46]:
```
a + a
```

[46]:
```
400
```

[82]:
```
class Foo(ABC):
    def greet(self):
        return f"Hello, {self.name}!"

    @property
    @abstractmethod
    def name(self):
        ...
```

[83]:
```
a = Foo()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [83], in <cell line: 1>()
----> 1 a = Foo()

TypeError: Can't instantiate abstract class Foo with abstract method name
```

[84]:
```
class Bar(Foo):
    def __init__(self, name):
        self.__name = name

    @property
    def name(self):
        return self.__name
```

[85]:
```
a = Bar("Adam")
```

[86]:
```
a.greet()
```

[86]:
```
'Hello, Adam!'
```

[87]:
```
class Baz(Foo):
    name = None

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

[88]:
```
b = Baz("Adam")
```

[89]:
```
b.greet()
```

[89]:
```
'Hello, Adam!'
```

[90]:
```
class Bar(Foo):
    def name(self):
        return "ADAM"
```

```
[91]: c = Bar()
```

```
[92]: c.greet()
```

```
[92]: 'Hello, <bound method Bar.name of <__main__.Bar object at 0x10a535c70>>!'
```

```
[ ]:
```