



OOP

Walter Cazzola

OOP

Introduction
Why
class definition
Inheritance
polymorphism
duck typing
conclusions

References

Object-Oriented Programming in Python

Classes, Inheritance & Polymorphism

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
e-mail: cazzola@dico.unimi.it



Slide 1 of 9



OOP

Walter Cazzola

OOP

Introduction
Why
class definition
Inheritance
polymorphism
duck typing
conclusions

References

Object-Oriented Programming

Introduction

Python is a multi-paradigm programming language.

Many claims that:

Python is object-oriented

Python is just **Object-Based** But we can use it as if it is object-oriented.

Look at

Reference

Peter Wagner.

Dimensions of Object-Based Language Design.

In Proceedings of OOPSLA'87, pp. 168-182, October 1987.

for the differences.



Slide 2 of 9



OOP

Walter Cazzola

OOP

Introduction
Why
class definition
Inheritance
polymorphism
duck typing
conclusions

References

Object-Oriented Programming

Designing Philosophy

Data-Centric View

- an application can be viewed as data+algorithms

Reference

Niklaus Wirth.

Algorithms + Data Structures = Programs.

Prentice-Hall, 1976.

- data and operations on them should be strictly related
- principles: encapsulation, data hiding and abstraction

Divide & Conquerer

- a problem is divided in subproblems to simplify the solution;
- each class solves a subproblem the interaction of several objects will originate the solution to the problem;
- this is true also for modules, functions, ...



Slide 3 of 9



OOP

Walter Cazzola

OOP

Introduction
Why
class definition
Inheritance
polymorphism
duck typing
conclusions

References

Object-Oriented Programming

Class Definition: Rectangle

```
class rectangle:
def __init__(self, width, height):
    self._width=width
    self._height=height
def calculate_area(self):
    return self._width*self._height
def calculate_perimeter(self):
    return 2*(self._height+self._width)
def __str__(self):
    return "I'm a Rectangle! My sides are: {0}, {1}\nMy area is {2}.".format(self._width,self._height, self.calculate_area())
```

```
[13:08]cazzola@ulik:~/esercizi-pa>python3
>>> from rectangle import rectangle
>>> r = rectangle(7,42)
>>> print(r)
I'm a Rectangle! My sides are: 7, 42
My area is 294
```



Slide 4 of 9



Object-Oriented Programming

Inheritance

OOP

Walter Cazzola

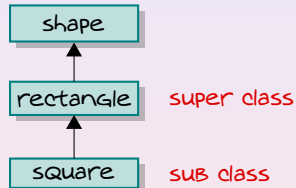
OOP

Introduction
Why's
class definition

Inheritance
polymorphism
duck typing
conclusions

References

Inheritance permits to reuse and specialize a class.



```
class shape:
    def calculate_area(self): pass
    def calculate_perimeter(self): pass
    def __str__(self): pass
```

```
from rectangle import rectangle
```

```
class square(rectangle):
    def __init__(self, width):
        self._width=width
        self._height=width
    def __str__(self):
        return \
            "I'm a Square! My side is: {0}\n \
            My area is {1}".format( \
                self._width, self.calculate_area())
```

A square is a rectangle that is a shape

ADAPT LAB

Slide 5 of 9



Object-Oriented Programming

Inheritance & Polymorphism

OOP

Walter Cazzola

OOP

Introduction
Why's
class definition

Inheritance
polymorphism
duck typing
conclusions

References

```
[22:24]cazzola@ulik:~/esercizi-pa>python3
>>> from rectangle import rectangle
>>> from square import square
>>> from circle import circle
>>> shapes = [square(7), circle(3.14), rectangle(6,7), square(5),
              circle(.7), rectangle(7,2), square(2)]
>>> shapes
[<square.square object at 0x80c698c>, <circle.circle object at 0x80c69ac>,
<rectangle.rectangle object at 0x80c69cc>, <square.square object at 0x80c69ec>,
<circle.circle object at 0x80c6a0c>, <rectangle.rectangle object at 0x80c6a2c>,
<square.square object at 0x80c6a4c>]
>>> for i in shapes: print(i)
...
I'm a Square! My side is: 7
My area is 49
I'm a Circle! My ray is: 3.14
My area is 30.9748469273
I'm a Rectangle! My sides are: 6, 7
My area is 42
I'm a Square! My side is: 5
My area is 25
I'm a Circle! My ray is: 0.7
My area is 1.53938040026
I'm a Rectangle! My sides are: 7, 2
My area is 14
I'm a Square! My side is: 2
My area is 4
```

ADAPT LAB

Slide 6 of 9



Object-Oriented Programming

Inheritance & Polymorphism Duck Typing

OOP

Walter Cazzola

OOP

Introduction
Why's
class definition

Inheritance
polymorphism
duck typing
conclusions

References

... But is shape really necessary? No

```
class rectangle:
    def __init__(self, w, h):
        self._width=w
        self._height=h
    def calculate_area(self):
        return \
            self._width*self._height
    def calculate_perimeter(self):
        return \
            2*(self._height+self._width)
    def __str__(self):
        return ...
```

```
class circle:
    def __init__(self, ray):
        self._ray=ray
    def calculate_area(self):
        return self._ray**2*math.pi
    def calculate_perimeter(self):
        return 2*self._ray*math.pi
    def __str__(self):
        return ...
```

```
class square(rectangle):
    def __init__(self, width):
        self._width=width
        self._height=width
    def __str__(self):
        return ...
```

```
[22:28]cazzola@ulik:~/esercizi-pa>python3
>>> from rectangle import rectangle
>>> from square import square
>>> from circle import circle
>>> shapes = [square(7), circle(3.14), rectangle(6,7), square(5),
              circle(.7), rectangle(7,2), square(2)]
>>> for i in shapes: print(i)
I'm a Square! My side is: 7
My area is 49
...
```

Duck Typing

ADAPT LAB

Slide 7 of 9



Object-Oriented Programming

Summarizing

OOP

Walter Cazzola

OOP

Introduction
Why's
class definition

Inheritance
polymorphism
duck typing
conclusions

References

The meaning of class is changed

- super classes do not impose a behavior (no abstract classes or interfaces)
- super classes are used to group and reuse functionality

Late Binding quite useless

- no static/dynamic type
- duck typing

Class vs instance members

- no real distinction between fields and methods
- class is just the starting point
- a member does not exist until you use it (dynamic typing)



Slide 8 of 9



References

OOP

Walter Cazzola

OOP

Introduction

Why?

Class definition

Inheritance

Polymorphism

Duck typing

Conclusions

References

- ▶ Jennifer Campbell, Paul Gries, Jason Montojo, and Greg Wilson.
Practical Programming: An Introduction to Computer Science Using Python.
The Pragmatic Bookshelf, second edition, 2009.
- ▶ Mark Pilgrim.
Dive into Python 3.
Apress*, 2009.
- ▶ Mark Summerfield.
Programming in Python 3: A Complete Introduction to the Python Language.
Addison-Wesley, October 2009.

