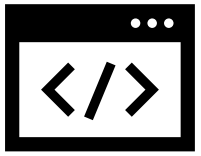
IMPERIAL



BIOE50010 – Programming 2

Computer Lab 5: OOP & Inheritance

Binghuan Li, Maria Portela, Gauthier Boeshertz, Samuel George-White, Yilin Sun, Kamrul Hasan, Wenhao Ding, Siyu Mu, Lito Chatzidavari

2 November, 2025

Feedback on Week 4 - `__mul__` & `__rmul__`

- Although `pt * 2` and `2 * pt` should give you the same result, their implementations in Python are different.
  - `pt * 2` : "2 has been multiplied to the variable `pt`", the object `pt` is the left-hand operand – use `__mul__`
  - `2 * pt` : "the variable `pt` has been multiplied to 2" , the object `pt` is the right-hand operand – use `__rmul__`

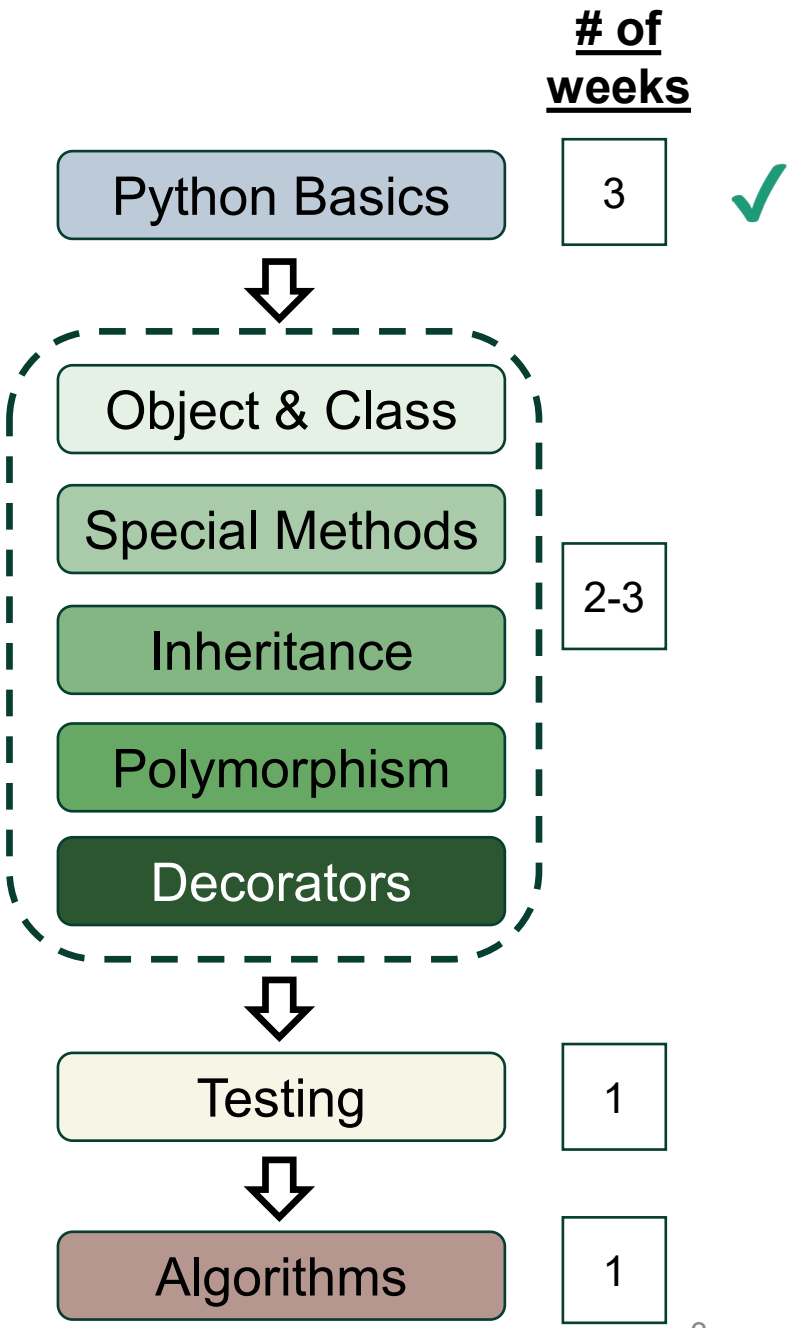
- **Why?** Math operations are *not* always symmetric! Consider the following example:

$$\vec{v}_1 = \begin{pmatrix}1\\0\\0\end{pmatrix} \text{ and } \vec{v}_2 = \begin{pmatrix}0\\1\\0\end{pmatrix} \Longrightarrow \vec{v}_1 \times \vec{v}_2 = \begin{pmatrix}0\\0\\1\end{pmatrix} \quad \vec{v}_2 \times \vec{v}_1 = \begin{pmatrix}0\\0\\-1\end{pmatrix} \Longrightarrow \boxed{\vec{v}_1 \times \vec{v}_2 \neq \vec{v}_2 \times \vec{v}_1}$$

- … that explains why Python requires both `__mul__` and `__rmul__` methods to handle left and right operands appropriately.

Progress Check

Week 5:
we are here



Revision Points (from weeks 4)

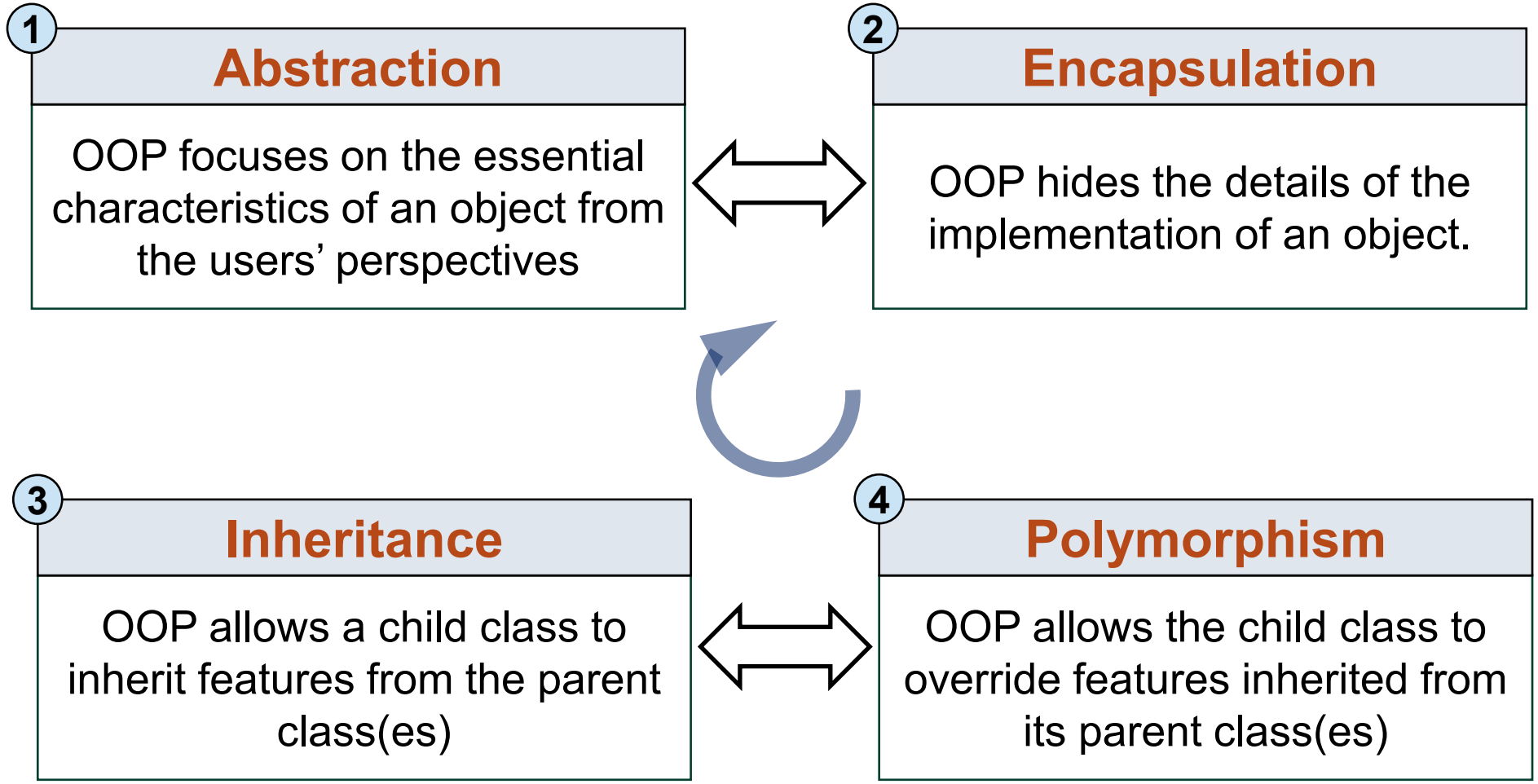
- **Concepts of OOP and definition of the terminologies:** class, object, instance, abstraction, encapsulation, attributes, methods
- **Basic OOP syntax:** `__init__`, `self`, how to instantiate an object, call methods, …
- **Special methods and operator overloading:** `__str__`, `__add__`, `__radd__`, `__eq__`, *etc.*

Four Pillars of OOP

Source: https://www.reddit.com/r/ProgrammerHumor/comments/gu4k3y/oops/

Four Pillars of OOP

new terminology!



Abstraction & Encapsulation

new terminology!

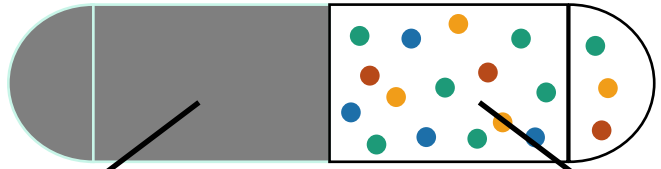
1 **Abstraction**

OOP focuses on the essential characteristics of an object from the **users'** perspectives

2 **Encapsulation**

OOP hides the details of the implementation of an object.

a paracetamol capsule



user side

perspective of the **patient**: a pain-killer capsule

developer side

perspective of the **pharmaceutical company**: the effective ingredients

Inheritance & Polymorphism

new terminology!

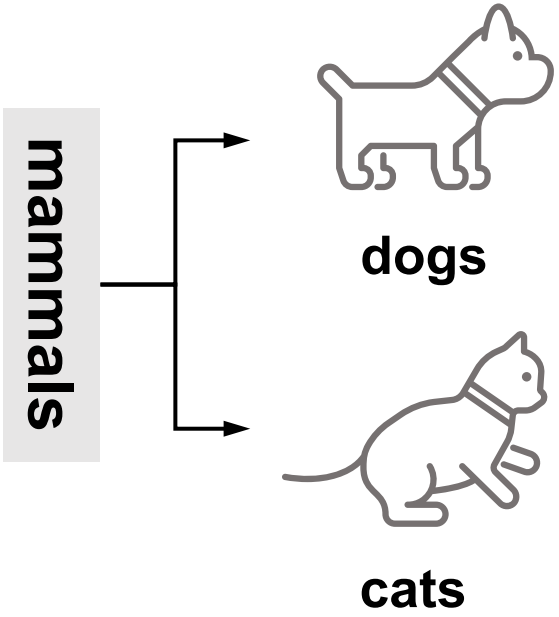
3 Inheritance

OOP allows a child class to inherit features from the parent class(es)

4 Polymorphism

OOP allows the child class to override features inherited from its parent class(es)

"***having many forms***"



All mammals have some **common** characteristics, *e.g.*

- warm-blooded
- feed their babies with milk

inheritance

Dogs and cats have **unique** characteristics, *e.g.*

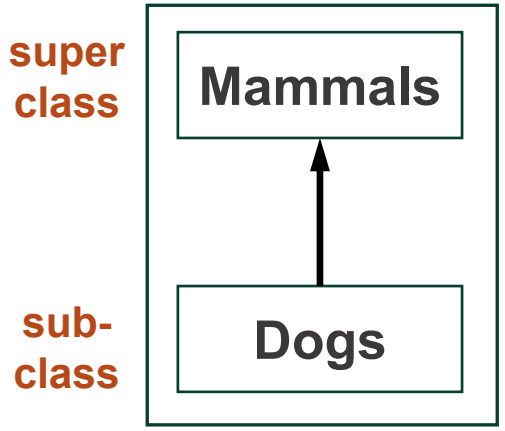
- Dogs: good sense of smell
- Cats: cannot taste sweetness

polymorphism

Coding Example

new terminology!

"single" inheritance



Example

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

    def warm_blooded(self):
        return f"{self.name} is warm-blooded."

    def speak(self):
        return "Grrrr!"

class Dog(Mammal):
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        return "Bark!"
```

`warm_blooded()` method in `Dog` is **inherited** from the `Mammal` class

`speak()` method in `Dog` overrides the `speak()` method in `Mammal`: **polymorphism**

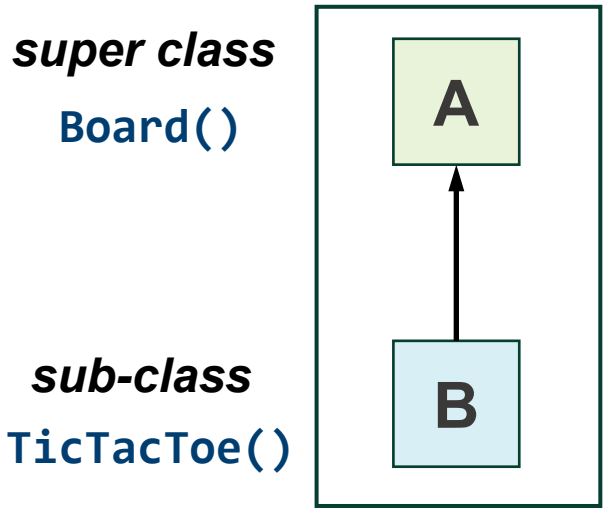
* See weekly coding example here.

Your task today

Refactor the Tic Tac Toe game using **object-oriented programming**. You are asked to define two classes

- **`Board()`** class: a class that should be able to fit into *any* board games.
- **`TicTacToe()`** class: a sub-class of **`Board()`** but also with the Tic Tac Toe-specific features.

… and a **`main()`** function to drive the Tic Tac Toe game.

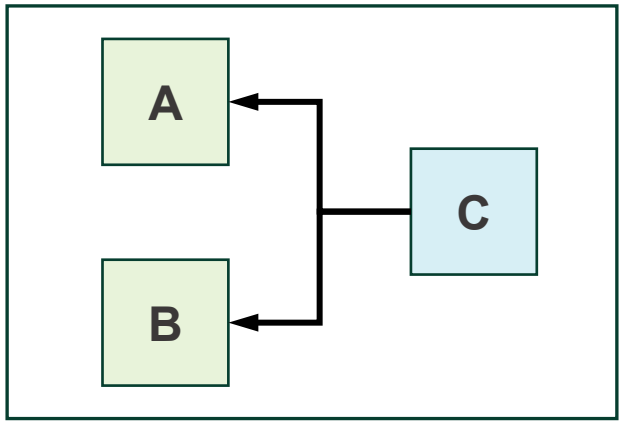


To start…

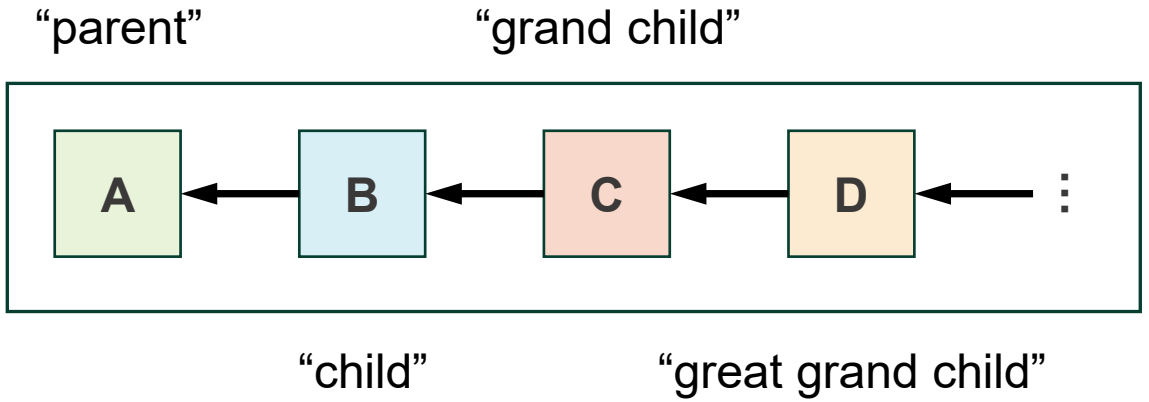
- Revise your worked solution to Lab session 1. What features/procedures are common for all board games? What features/procedures are unique for Tic Tac Toe only?
- Study the sample scripts for the syntax of inheritance of OOP.

Appendix 1: **Inheritance Can Be in Many Forms**

“multiple” inheritance



“multi-level” inheritance



⚠ Be very cautious about the yo-yo problem when using multi-level inheritance!

- Excessive maintenance challenges
- Compensated readability