IMPERIAL

BIOE50010 – Programming 2

Computer Lab 5

Binghuan Li Department of Chemical Engineering

Maria Portela Department of Bioengineering

Wenhao Ding Department of Bioengineering

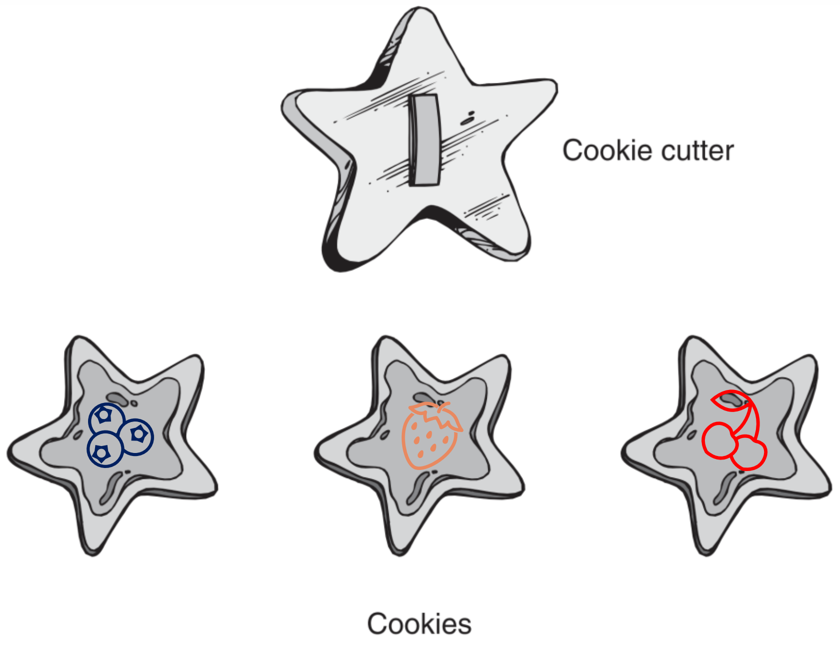
1 November, 2024

Object-Oriented Programming

🟥 new terminology!

- Two most commonly-used programming paradigms:
  - **Procedural**: programs are composed of one or more functions, executed serially;
  - **Object-oriented**: programs based on the **objects**, where data and functions are 'packed' into a user-defined **data structure**.
- **Examples of objects**: `str`, `list`, `dict`…
  - These are the **data structures**, rather than the real data!
- The prototype / blueprint of an object is structured by the **class definition**.

A ***cookie cutter*** can be used to make different ***cookies***. ⟺ A ***class definition*** can be used to make several ***objects***.



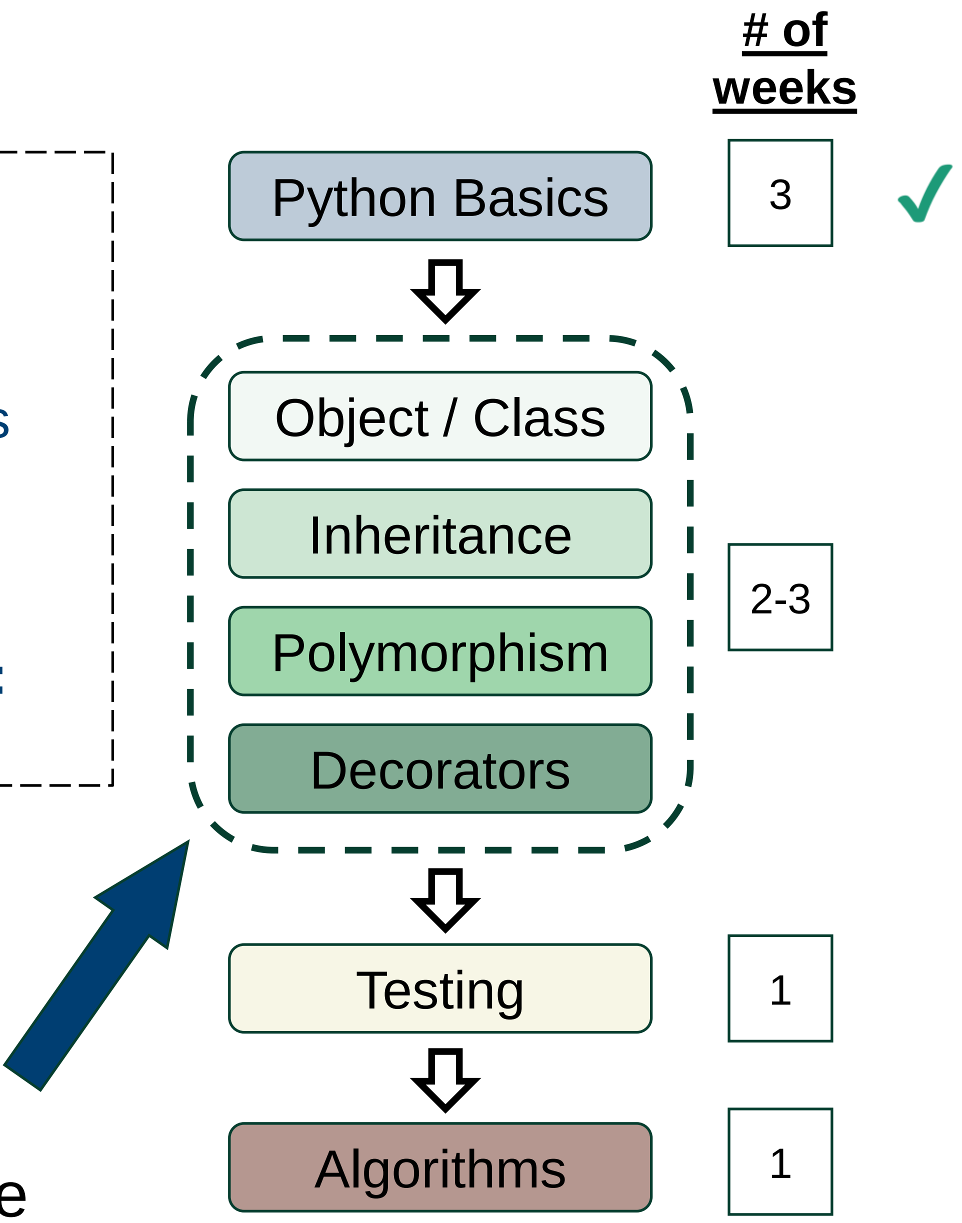
Source: Starting Out with Python, 4th Ed.

- Sometimes, **object** is also referred to as **instance**.

Progress Check

Checklist: you should have mastered…

- **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.*



Week 5:
we are here

Four Pillars of OOP

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

Four Pillars of OOP

🟧 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.

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)

Abstraction & Encapsulation

🟥 new terminology!

1 Abstraction

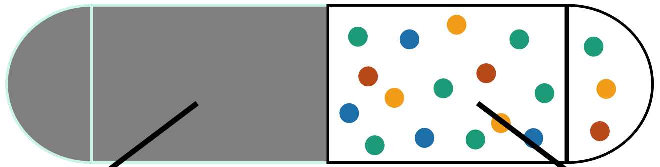
OOP focuses on the essential characteristics of an object from the **users'** perspectives

⟺

2 Encapsulation

OOP hides the details of the implementation of an object.

a paracetamol capsule



user side

developer side

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

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

Inheritance & Polymorphism

🟧 new terminology!

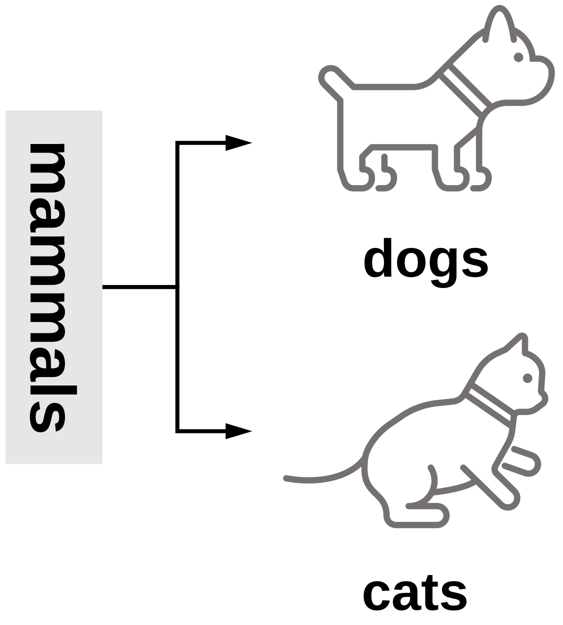
3 Inheritance

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.*

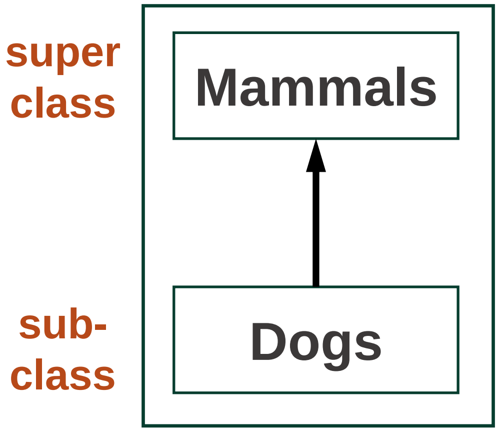
- Dogs: good sense of smell
- Cats: cannot taste sweetness

polymorphism

Coding Example

🟥 new terminology!

“single” inheritance



Example

```python
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(Mammals):
    def __init__(self, name):
        super().__init__(name)

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

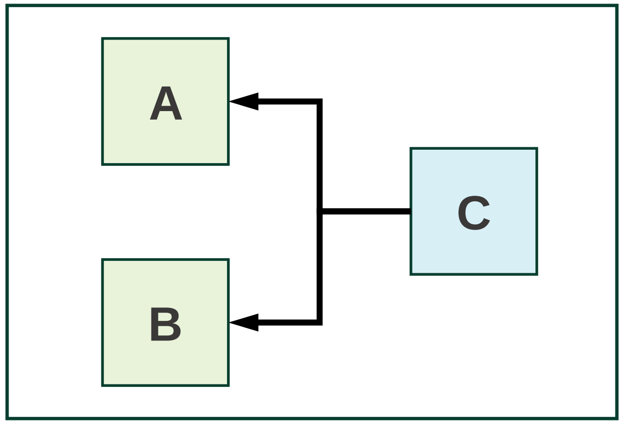
speak() method in Dog overrides the speak() method in Mammal: **polymorphism**

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

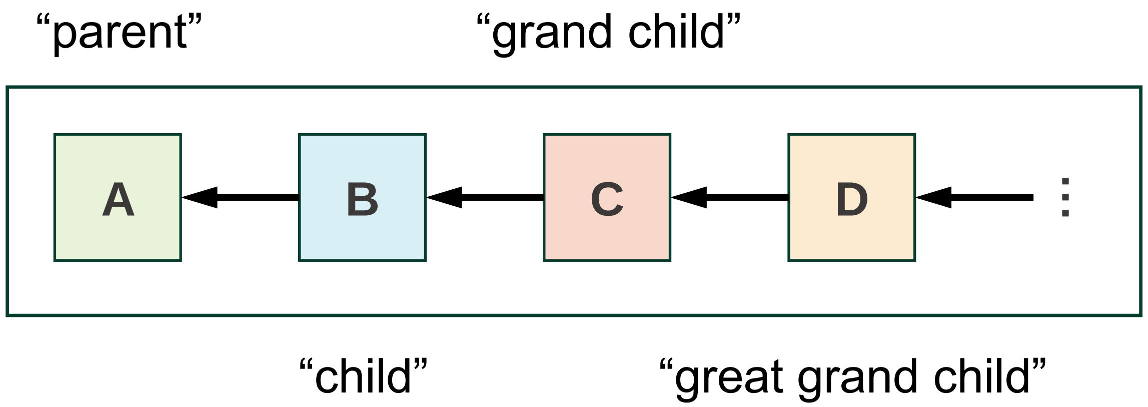
See Mammal_example.py on Bb

Inheritance Can Be in Many Forms

“multiple” inheritance

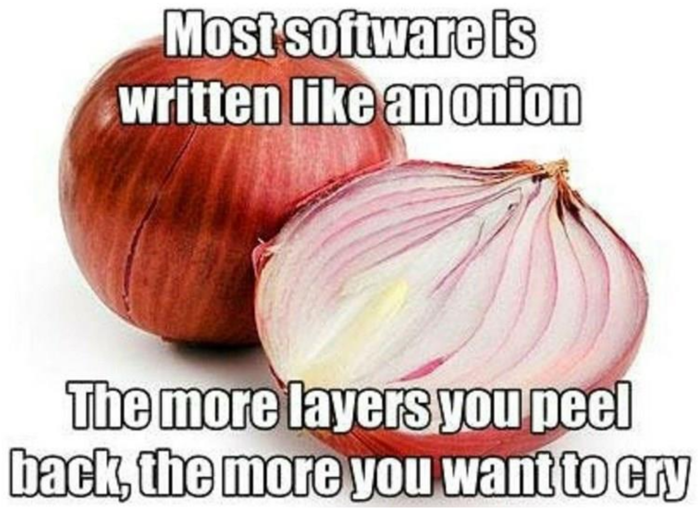


“multi-level” inheritance



⚠ Be very cautious about [the yo-yo problem]() when using multi-level inheritance!

- Excessive maintenance challenges
- Compensated readability


Most software is written like an onion
The more layers you peel back, the more you want to cry

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.

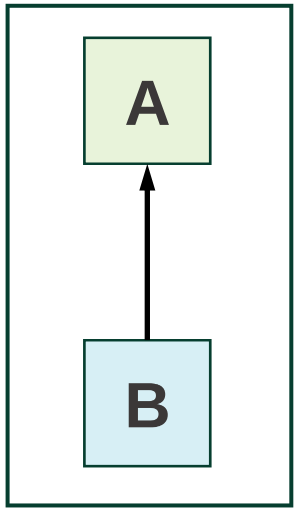
… and a **`main()`** function to drive the Tic Tac Toe game.

super class

`Board()`

sub-class

`TicTacToe()`



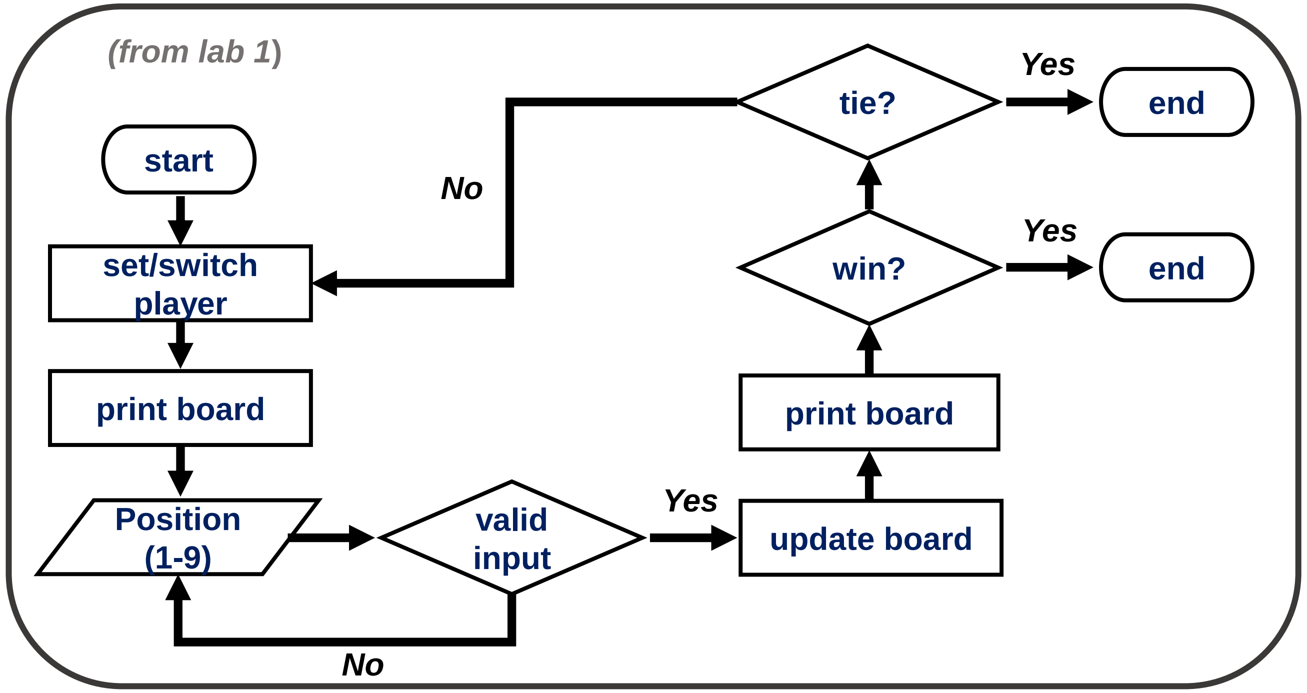
To start…

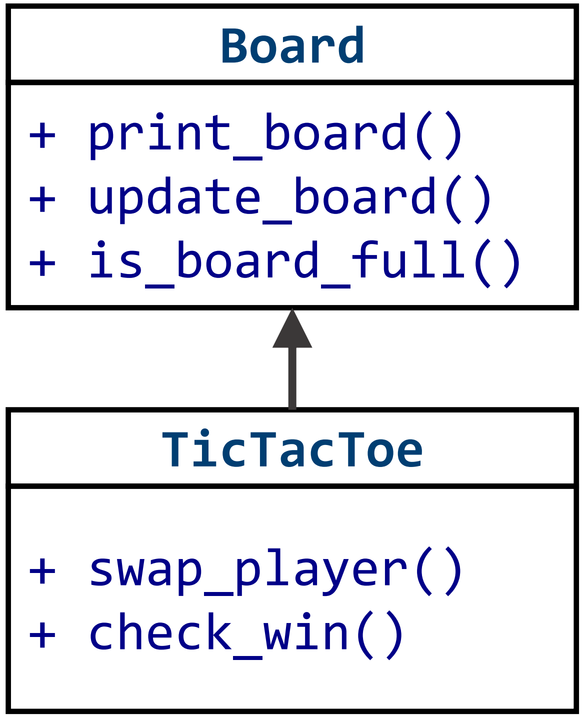
- Revise your worked solution to Lab session 1. What features/procedures are <u>common</u> for all board games? What features/procedures are <u>unique</u> for Tic Tac Toe only?
- Study the sample scripts for the syntax of inheritance of OOP.

Questions?

That’s it for now.

You can now proceed to the Lab 5 exercises.


(from lab 1)
start
set/switch player
print board
Position (1-9)
valid input
No
Yes
update board
print board
win?
Yes
end
tie?
Yes
end
No


Board
+ print_board()
+ update_board()
+ is_board_full()
TicTacToe
+ swap_player()
+ check_win()