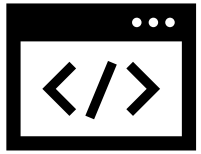
IMPERIAL



BIOE50010 – Programming 2

Computer Lab 6: Object-Oriented Programming

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

10 November, 2025

Feedback on Week 5 – **2D Board Structure**

Example: declare the 2-D nested structure

```
N_ROW = 3
N_COL = 4
board = []
for i in range(N_ROW):
    r = []
    for j in range(N_COL):
        r.append(' . ')
    board.append(r)
```

1. **Create a row**: Start with an empty list and append the same element to it `N_COL` times. (imagine this is a row vector with N_{col} elements, represents N_{col} columns per row.)
2. **Build the board**: Append the initialized row to the main board structure for `N_ROW` times. (so, you obtain a $N_{row} \times N_{col}$ matrix.)

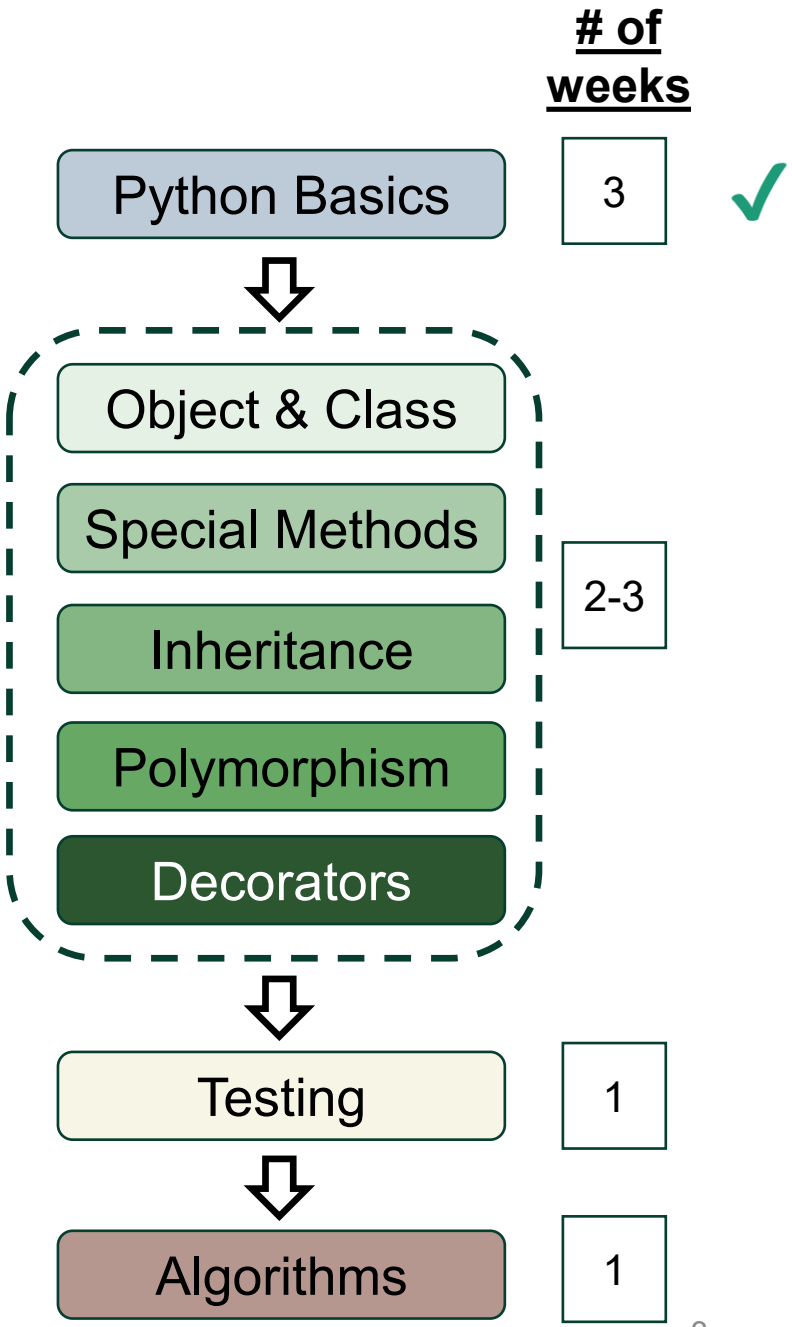
```
[[' . ', ' . ', ' . ', ' . '], [' . ', ' . ', ' . ', ' . '], [' . ', ' . ', ' . ', ' . ']]
```

- Think of it as building layers: each row adds a layer to the board.
- However, this is **not** formatting. To print the rectangular board structure, you need to use nested `for` loops to loop over each element in the board. (week 4 slides)

* See in-class example here.

Progress Check

Week 6:
we are here



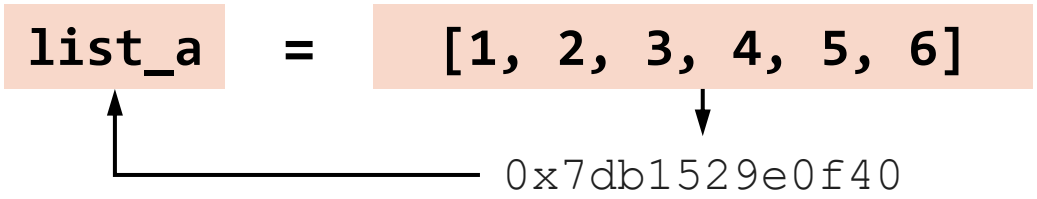
Revision Points (from weeks 5)

- Concepts of "four pillars" in OOP: **abstraction**, **encapsulation**, **inheritance**, **polymorphism**.
- **Syntax/concepts of coding inheritance in Python:** super class, sub-class, `super()` function
- **Inheritance can be in many forms:** single, multiple, multi-level.

deepcopy

* See weekly coding example here.

- During variable assignment, the variable name does *not* hold the object itself.
- Instead, Python links the **memory address** of the object to the variable name.



- Therefore, when assigning one variable to another (*e.g.*, `list_b = list_a`) **does *not* create a new copy** of the object.
  - Both variables point to the same memory address.
  - Any change in `list_a` will also reflect in `list_b`.

- To create a *true* copy of an object, one needs to use a **deep copy** (*e.g.*, `list_b = copy.deepcopy(list_a)`), so that everything within `list_a` is duplicated recursively into `list_b`, each in its own independent memory space.

Handling Exceptions

* See weekly coding example here.

- An **exception** is a fatal event that happens during the execution of a program.
  - Exception: usually from the programme-level, *e.g.*, bugs.
  - Error: usually from the system-level, *e.g.*, not enough memory.
- Programme may be able to *catch* and *handle* exceptions, but not errors.
- In Python, exceptions can be handled using the **`try…except…`** clause:

Example: use of `try…except…` clause

```
while True:
    try:
        x = int(input("Please enter a number (1-9): "))
        break
    except ValueError:
        print("That was not valid number. Try again...")
```

`ValueError` raises due to the failure of typecasting, *e.g.*, `int("hello!")`

- A more advanced exception handling syntax is **`try…except…finally…`**

Your task today

Provide an **object-oriented** implementation for handling and manipulating DNA sequences.

- Manipulating the DNA data: concatenation (`__add__`, `__radd__`), indexing (`__getitem__`), counting nucleotides.
- File I/O: load and read the contents from an external `.fna` file.

To start…

- Recall the **string/list methods** and **file I/O methods** you have used; (lab 2 slides)
- Recall the class **special methods** and operator overloading you have used;
- Read all information and the **sample output** provided in the lab carefully;
- Try to integrate **exception handling** into your code: *e.g.*, "open a file and read in the data; *if* your attempt fails, return an empty data structure."

Questions?

That's it for now.

You can now proceed to the Lab 6 exercises.

Appendix 1: **Summary of Common Exceptions**

Exception	Description
`AttributeError`	Accessing an undefined attribute in a **class**.
`ImportError`	**Module** import fails.
`IndexError`	Accessing an out-of-range index in a **list** or **tuple**.
`KeyError`	Accessing a non-existent **dictionary** key.
`NameError`	Using a **variable** that hasn't been defined.
`TypeError`	Performing an operation on an inappropriate **data type**.
`ValueError`	Passing a valid type but **invalid value**.
`ZeroDivisionError`	Dividing by zero.
`SyntaxError`	Code contains a **syntax** error.
`RuntimeError`	**Generic** error for code execution issues.

Appendix 2a: **“Is-a” or “Has-a”?**

Optional Topic

new terminology!

- Consider the following associations between two objects:

“Dog **is a** breed of mammal” ⇨ Compatible with *inheritance*

“The car **has an** engine.” ⇨ Not compatible with inheritance!

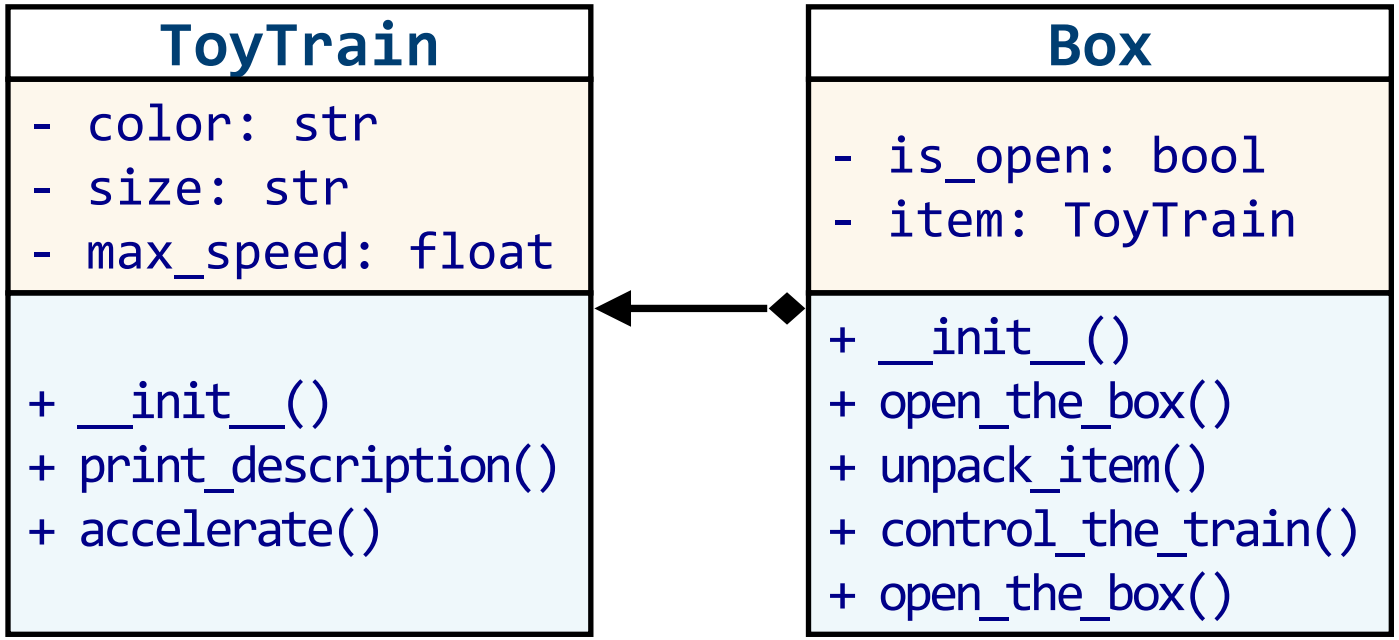
- In the second situation, the engine is a component of the car (composite).
- The “***has-a***” association is more appropriate for depicting cases in which one object forms a component or part of another, rather than being a type of that object.
- This relation is known as the **composition**.

Appendix 2b: **Composition**

Optional Topic

new terminology!

- **Composition** is a form of *association* where a class contains objects of another class as part of its internal structure.
- The following code example: A toy train (object 1) within a box (object 2).



- **`Box.item`** is an attribute holding a **`ToyTrain`** object.
- Therefore, the following expressions are functionally equivalent:

  `ToyTrain.accelerate()`

  `Box.item.accelerate()`

* See weekly coding example here.