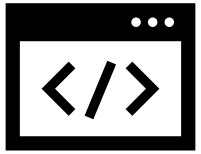
Imperial College London



BIOE50010 – Programming 2

Computer Lab 8

Binghuan Li | *Department of Chemical Engineering*

binghuan.li19@imperial.ac.uk

November 27, 2023

Meme of the week ☺

Source: https://www.pinterest.com/pin/462322717972269253/

Four Pillars of OOP

1) ***Abstraction*** focuses on the essential characteristics of an object relative to the perspective of the viewer.

2) ***Encapsulation*** hides the details of the implementation of an object.

3) ***Inheritance*** allows for a derived object type to inherit features from another object type.

4) ***Polymorphism*** allows for overriding any inherited method by creating your own method within its own class.

- We also very briefly mentioned ***aggregation*** and ***composition***, which are the alternatives to ***inheritance*** that describe the relationships between classes.

super()

- The **super()** function is used to give access to methods and properties of a parent (or sibling) class.

Example

```
class Board:
    def __init__(self, rows, cols):
        for i in range(rows):
            self.board.append(['.']*cols)

class TicTacToe(Board):
    def __init__(self):
        super().__init__(3, 3)
```

- `Board` is a parent (***super***) class that can be suited into any board games with of ***any*** dimension.
- `TicTacToe` is a child (***sub***) class of `Board` with **3 columns** and **3 rows**.

triggers the **`__init__`** method in **`Board`**, passes parameters `rows = 3` and `cols = 3` to the superclass, hence initialise the `board` with the desired dimension

Shout your questions from Lab 7!

you should now understand…

- The concepts of abstraction, encapsulation and inheritance in OOP;
- How to build a class constructor, `__init__`;
- The purpose for using the implicit keyword self in Python OOP;
- How to use other special methods (*e.g.*, `__str__`);
- How to implement inheritance in Python OOP.

we encourage you to understand…

- The concepts of polymorphism, aggregation, and composition;
- Best class design patterns and design architecture planning.

Decorators

A *decorator* is a special type of function that is used to modify the behaviour of another function or method.

Example from decorators.py

```
def debug_timer(some_function):

    import time

    def wrapper_function(*args, **kwargs):
        t0 = time.time()
        some_function(*args, **kwargs)
        dt = time.time() - t0
        print(f'Elapsed time: {dt} seconds')
    return wrapper_function

@debug_timer
def original_function(data1, data2):
    print(f'running fcn with {data1} and {data2}')

original_function('happy', 1)
```

Flow of execution

1. **original_function** is decorated with **@debug_timer**, so when it is triggered, it is replaced by the **wrapper_function**
2. **wrapper_function** is called with the arguments **'happy', 1**
3. Execution returns to **wrapper_function**, and the elapsed time is calculated

Static Methods

- Sometimes in OOP, we want a function that does ***not*** need to access to any attributes of the current object
  - *i.e.*, a method that does not depend on variables followed by `self`.
- There are two possible ways to implement such a function:
  - Example: check if someone is adult

Example with *standalone function*

```
class Person:
    def __init__(self, age):
        self.age = age;
        self.adult = is_adult(age);

def is_adult(age):
    return age > 18;
```

Example with *static method*

```
class Person:
    def __init__(self, age):
        self.age = age;
        self.adult = self.is_adult(age);

    @staticmethod
    def is_adult(age):
        return age > 18;
```

revoke with `self`

decorator (must have)

no need `self`

Class Methods

- In OOP, a class method can **modify a class state** that would apply across all the instances of the class.
  - Example: set the age of a person

Example

```
from datetime import date

class Person:
    def __init__(self, age = 0):
        self.age = age

    @classmethod
    def fromBirthYear(cls, year):
        return cls(date.today().year - year)
```

decorator (must have)

`cls` is an implicit name that refers to the class

Return the calculated age and assign to `self.age`

Driver code

```
p1 = Person(20)
print(p1.age)

p2 = Person.fromBirthYear(2001);
print(p2.age)
```

Console

```
20
22
```

Your Task Today

- 4 mini-tasks, featuring the exercises of
  - Computer animation with command prompts,

    `os.system('cls') / os.system('clear')`
  - Wrapper functions
  - Time module in Python
  - Decorators

  } **Your lecture live coding example**

- Read the sample code and console output carefully before you start.

Questions?