Python Programming

Interfaces and APIs

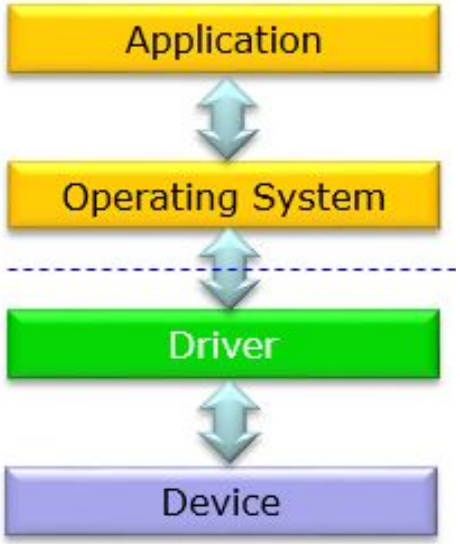
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

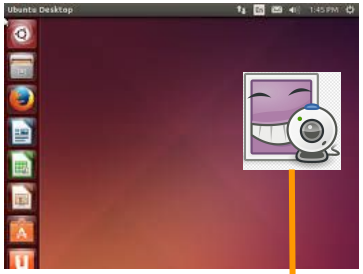
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Interfaces

- If **all** abstract class methods are abstract, we call it **interface**
  - **No state** or implemented methods
  - It can give sense of views: This class is *printable, comparable and runnable* (3 interfaces)
  - More advanced points in Python Metaprogramming
- Culture in other programming language
  - More explicit referring/need for interface.
    - Dependency on **Interfaces with Polymorphism** is a standard practice
  - **Properties**: In inheritance you think: Employee is a person
    - With Interfaces: you might think also in properties as a parent class
    - E.g. a class is Printable, Diskable(Savable, Loadable), Clonable, Comprable, etc
  - Due to duck typing, we don't use interfaces frequently

Interfaces: Device Driver





Use (What)

```python
class ICameraDevice(ABC):
    @abstractmethod
    def get_version(self):...

    @abstractmethod
    def start(self):...

    @abstractmethod
    def stop(self):...
```

Streaming 1080P Webcam

Communicate (How)

```python
class UbuntuDriverOpenSource(ICameraDevice):
    pass  # Override methods
```

Img Src

Interfaces: Device Driver

```python
class ICameraDevice(ABC):...

class UbuntuDriverOpenSource(ICameraDevice):
    pass  # Override methods

class UbuntuDriver3rdPart(ICameraDevice):
    pass  # Override methods

class Windows10Driver(ICameraDevice):
    pass  # Override methods


class UbuntuOS:
    def get_app(self, app, name):
        return UbuntuDriverOpenSource()

if __name__ == '__main__':
    os = UbuntuOS()
    device = os.get_app('camera cheese')
    device.start()
    device.stop()
```

Img Src

Coupling

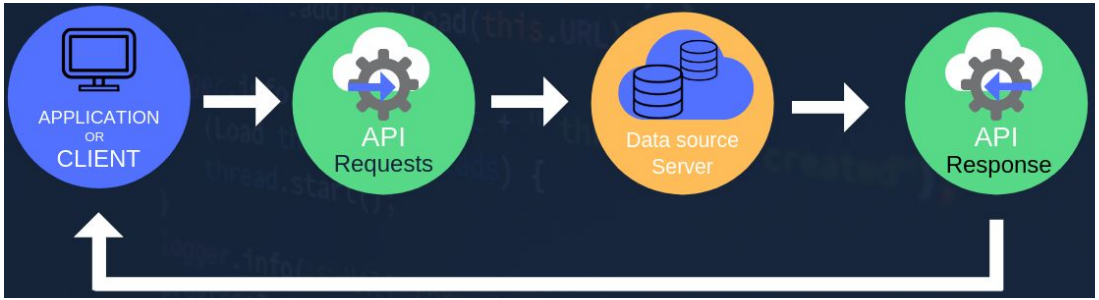
- Coupling is the measure of the degree of **interdependence** between the modules/classes. [critical SWE concept]
  - Target: **low coupling**
- If the Camera App will code for a specific driver, then if you decided to change the driver which has its own methods (start() vs run() - stop() vs shutdown()), then the system is highly coupled, which is so bad
- But if we have a common interface and each driver is following it, our camera app doesn’t need to know which driver is installed
  - Thanks also for polymorphism. This is very visible in C++/Java

Interfaces Guidelines

- Interface = contract. Don’t break it.
  - Changes may cause compilation errors. Consider backward compatibility.
  - Think deeply about method signature
- A minimal public interface
  - Doesn’t include your common or private functions
  - Avoid irrelevant functionalities / hard to get
- Principle of Least Surprise
  - Most guys don’t read documentation. Expected resulting behaviour = match function name
- Think from client/user perspective:
  - What are their needs?
    - Make their life easy as possible
  - Intuitive/minimal usage for your interface = default/fair behaviour

Application programming **interface** (API)

- Software **intermediary** (interface) that allows **two applications** to talk to each other.
  - Mobile apps like Facebook, Hangout, Weather are using an API.
  - **Messenger** ⇒ Facebook Messenger API ⇒ **Facebook backend**
    - Facebook backend provides this API to a few specific functionalities
  - They contact remote API
    - Communication + Request (function: param+return) + Response (e.g. JSON or XML)
  - Future readings: API design, Backend as API



Img Src

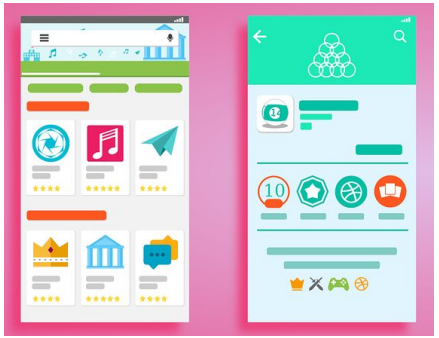
Application programming **interface** (API)

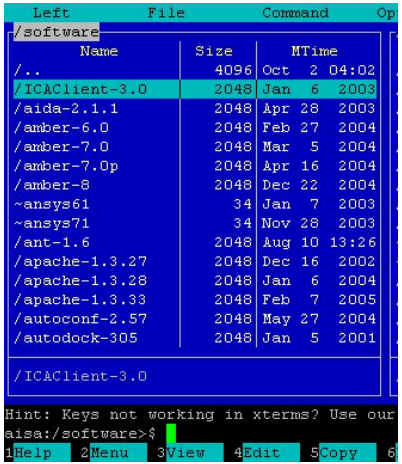
- Example: Payment API (top ones)
  - Many software we develop allows user payments (e.g. credit/debit cards)
  - It is waste of time to build several codes to contact/verify by yourself
  - We use payment APIs (e.g. Paypal or Stripe APIs)
  - They do the verifications, take user info + money to withdraw
    - We **pay** them some subscription (e.g. 0.1 dollar per transaction)
  - All the APIs will provide similar functionalities, but different style
    - E.g. different function names, parameters, call orders, authentication
- Example: Airlines API (top ones)
  - You want to go from Cairo to Vancouver
  - Go to expedia. Expedia call the API for different airlines
    - You can go through Toronto (AirCanada API), Istanbul (Turkish airlines API)
    - Also you can go through 3 cities in Germany (Lufthansa airlines API)

API vs Library vs SDK vs Framework

- **Library**: Functions/Classes ready to use (e.g. sorted / dict)
- Software development kit (**SDK**): **Collection** of tools in **one** installable package to make development and debugging easy (E.g. JDK, IOS SDK)
- **Framework**: High level - group of libraries, *typically* with **inversion of control** (IoC) pattern. Typically some **abstract** design, with more behavior built in.
  - You **insert** your code in a few places (e.g. subclassing, let's call it Class C)
  - The framework e.g. has specific pipeline: A, B, **AbstractC**, D, E
    - A, B, D are already implemented and fixed. You may override E.
    - Framework calls them in order, but for AbstractC calls your Class C (Polymorphism)
      - E.g. Your class C prepare specific data for the web-page to view
  - Your code calls library, but Framework calls your code (inverse of the flow)

Relevant: User interface (UI) & User experience (UX)





- UI/UX is not CS job
  - Frontend dev use UI
  - Console / GUI not frequent now
  - Web & Mobile
  - UI design is all about how the product's interfaces look and function.
  - UX design is all about the overall feel of the experience
  - Reading

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."