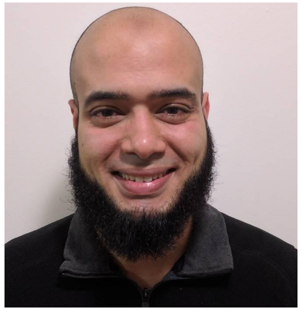
Python Programming
Abstract Classes

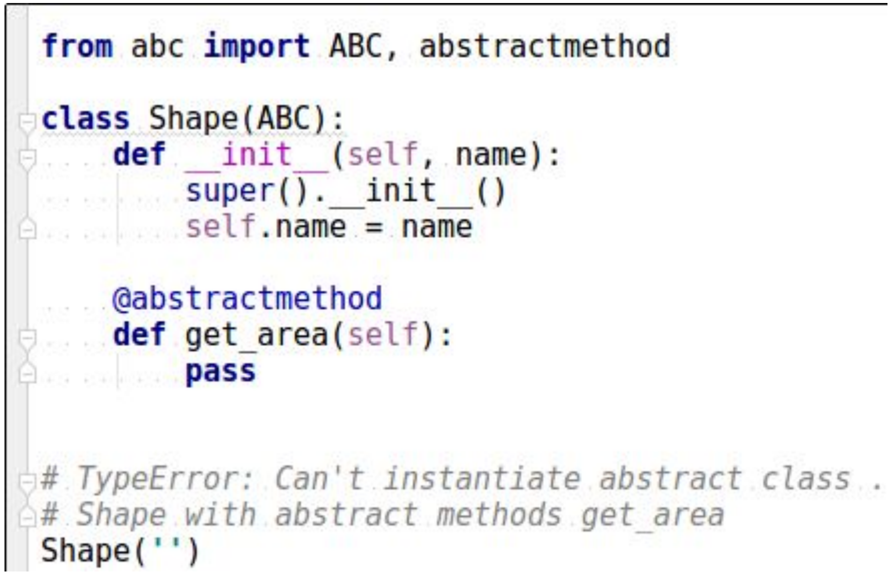
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

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



Abstract class

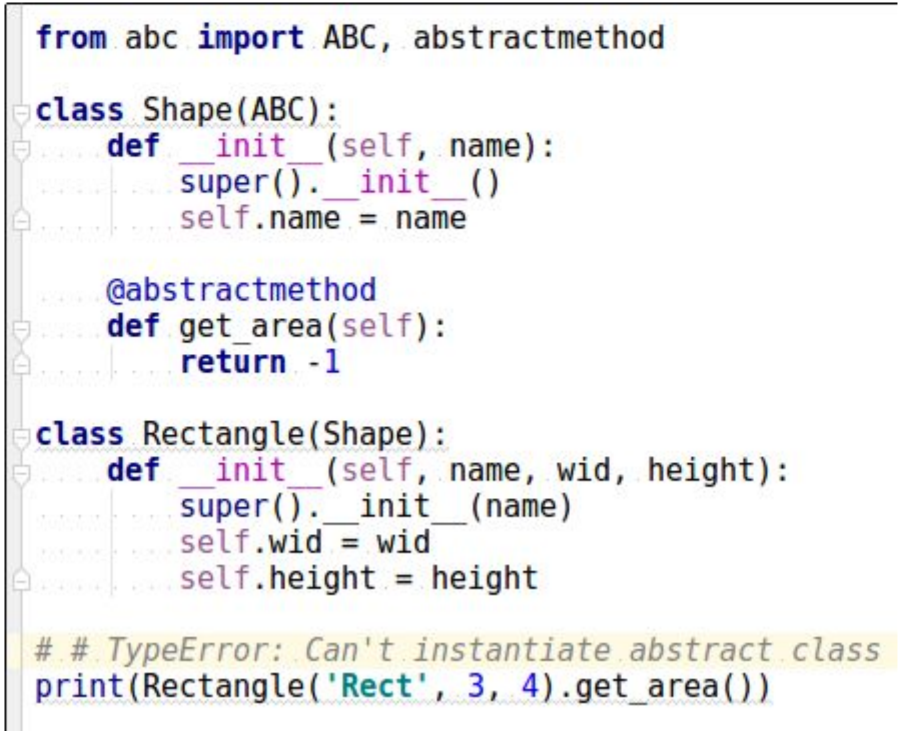
- Inheriting from ABC = marks class as abstract class
- Using decorator abstract method, we mark method abstract
- If an abstract class has a single abstract method, we can’t create object
- An abstract class can has non-abstract methods



```python
from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self, name):
        super().__init__()
        self.name = name

    @abstractmethod
    def get_area(self):
        pass


# TypeError: Can't instantiate abstract class .
# Shape with abstract methods get_area
Shape('')
```

A child class, but still abstract

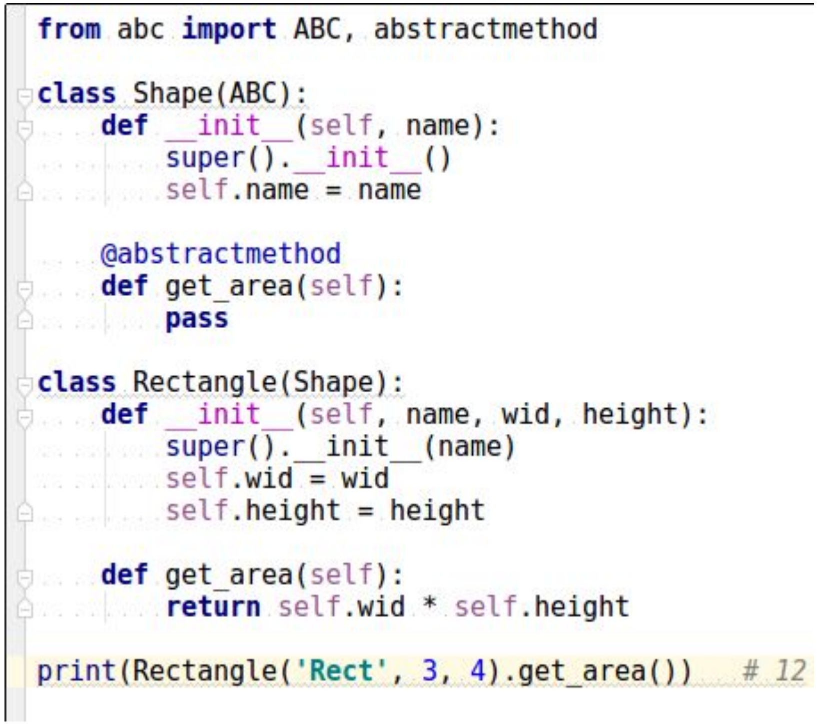
- If the child class doesn't provide implementation to ALL abstract method of an abstract class, then it is also abstract class
    - Even if the abstract method has default implementation
        - You can use super().something



```python
from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self, name):
        super().__init__()
        self.name = name

    @abstractmethod
    def get_area(self):
        return -1

class Rectangle(Shape):
    def __init__(self, name, wid, height):
        super().__init__(name)
        self.wid = wid
        self.height = height

# # TypeError: Can't instantiate abstract class
print(Rectangle('Rect', 3, 4).get_area())
```

Complete Class

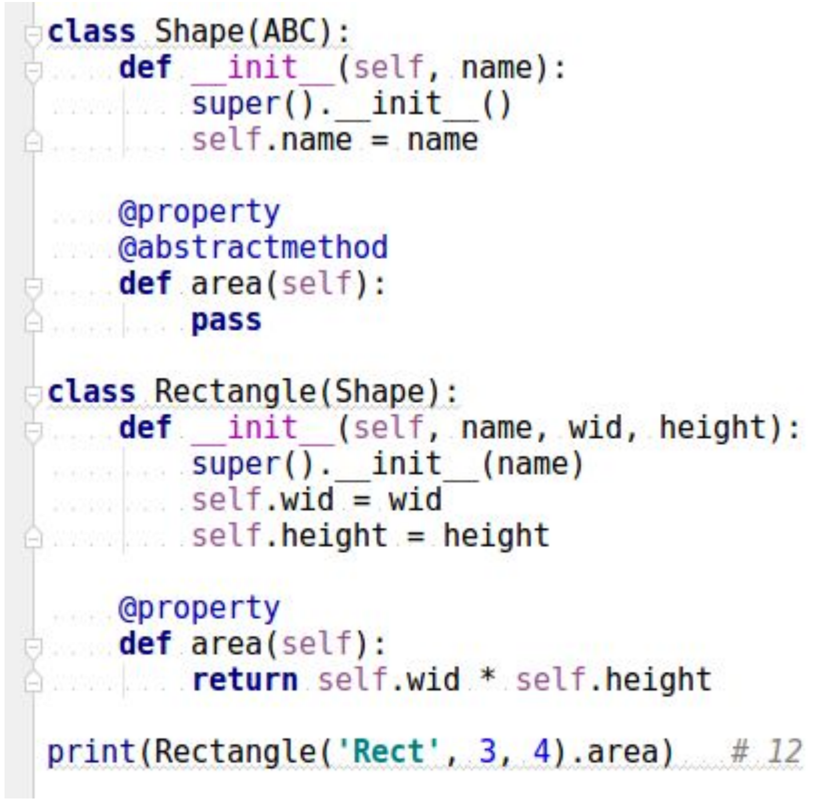
- Now Rectangle class is a **complete class**
- It already provides impl



```python
from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self, name):
        super().__init__()
        self.name = name

    @abstractmethod
    def get_area(self):
        pass

class Rectangle(Shape):
    def __init__(self, name, wid, height):
        super().__init__(name)
        self.wid = wid
        self.height = height

    def get_area(self):
        return self.wid * self.height

print(Rectangle('Rect', 3, 4).get_area())    # 12
```

With properties

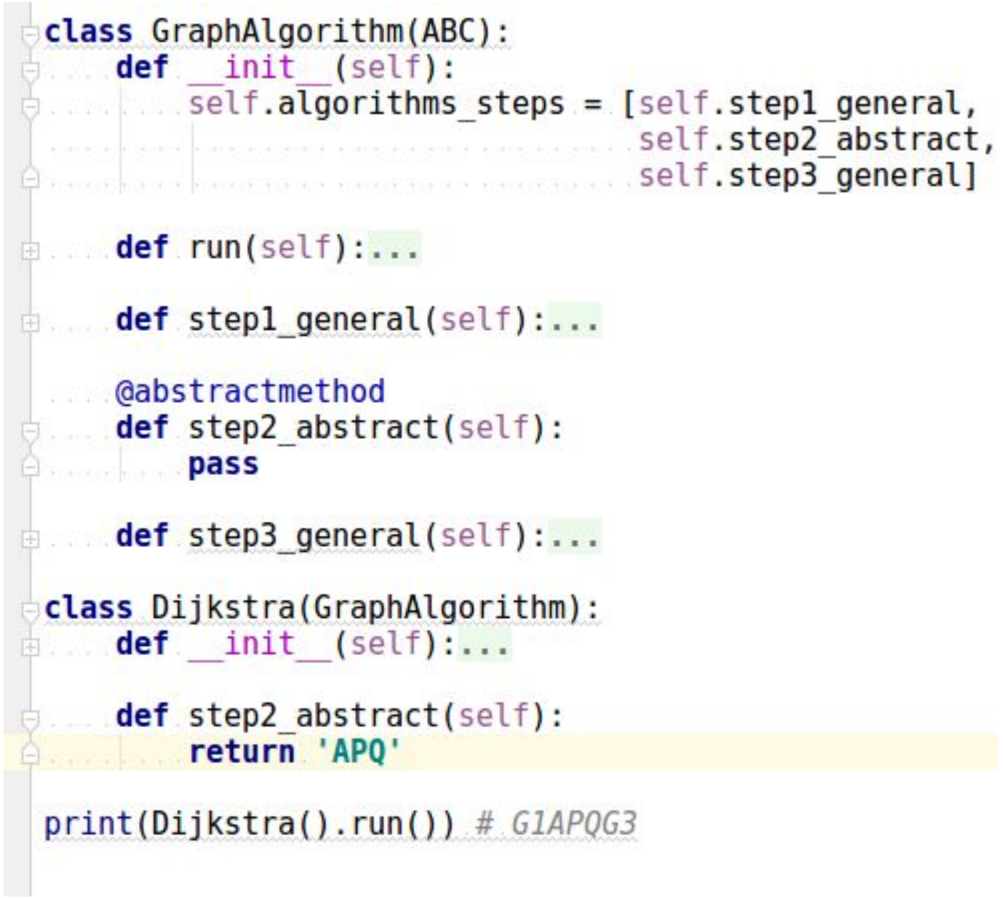
- In python 3, just add abstract method decorator normally
- Follow **this order**:
    - @property [FIRST]
    - @abstractmethod
- The same with @classmethod, @staticmethod



```
class Shape(ABC):
    def __init__(self, name):
        super().__init__()
        self.name = name

    @property
    @abstractmethod
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, name, wid, height):
        super().__init__(name)
        self.wid = wid
        self.height = height

    @property
    def area(self):
        return self.wid * self.height

print(Rectangle('Rect', 3, 4).area)    # 12
```

Messages

- Abstract classes make code more cleaner
  - We avoid raise exception
    We avoid not writing the method
- Communicate intentions
  - When we know the class is abstract, we understand this is incomplete
  - We need to provide ALL abstract methods to have a complete class



```
class GraphAlgorithm(ABC):
    def __init__(self):
        self.algorithms_steps = [self.step1_general,
                                 self.step2_abstract,
                                 self.step3_general]

    def run(self):...

    def step1_general(self):...

    @abstractmethod
    def step2_abstract(self):
        pass

    def step3_general(self):...

class Dijkstra(GraphAlgorithm):
    def __init__(self):...

    def step2_abstract(self):
        return 'APQ'

print(Dijkstra().run()) # G1APQG3
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."