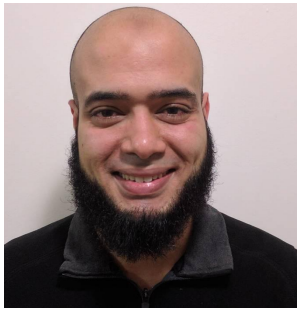
# Python Programming

# Class and Static Methods

**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)



# Static Methods

- So far we used **Instance** Methods
  - def something (self).
    - Self is an instance of object. We can access/change the attributes
- **Static methods** are defined at the class level not the object
  - They don't get self object ⇒ they can't change object attributes
  - You shouldn't use to alter class static variables
- Best usage: as a **utility** that neither depend on the object or the class
  - filter_duplicates(lst)
  - is_even(n)
  - get_position_neigbours(x, y, cnt)

# Static Methods

```python
class Person:
    def __init__(self, name):
        self.first, self.last = Person.process(name)
    def __repr__(self):
        return f'Person first name: {self.first} - last name: {self.last}'

    @staticmethod
    def process(name):  # No self - no interaction with class/objects
        """Convert to lower, get first word as first name, remaining as last"""
        first, *last = name.lower().split()
        last = ' '.join(last)
        return first, last

if __name__ == '__main__':
    print(Person('Mostafa Saad Ibrahim Mohamed'))
    # Person first name: mostafa - last name: saad ibrahim mohamed
```

# Class Methods

- **Class methods** are at the class level not the object
  - They don’t get self object ⇒ they can’t change object attributes
  - They get an object of the class type
    - They may access/modify the class attributes
- Best usage:
  - A **factory method** to generate objects from the class
    - This is a popular simple design pattern to create objects
  - A shared method among objects to manipulate attributes

# Class Methods

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first, self.last = first_name, last_name

    def __repr__(self):
        return f'Person first name: {self.first} - last name: {self.last}'

    @classmethod
    def get_person_from_full_name(cls, full_name):
        first, last = cls.process(full_name)
        return cls(first, last)

    @staticmethod
    def process(name):...

if __name__ == '__main__':
    per = Person.get_person_from_full_name('Mostafa Saad Ibrahim Mohamed')
    print(per)
    # Person first name: mostafa - last name: saad ibrahim mohamed
```

# Class Methods

- A few remarkable things about it
  - The method depends on passed argument cls for the class itself
    - If the class name changed, the method won’t :)
    - DRY principle
  - Soon, we learn about inheritance
    - If you implemented the method at the parent the level, it is visible for the child too!
    - Static method doesn’t have this great feature. It only can use Person.somestatic

# For educational purpose

- Similar to the property class, we can create without the decorator

```python
class Person:
    def __init__(self, name):
        self.first, self.last = Person.process(name)
    def __repr__(self):
        return f'Person first name: {self.first}  -  last name: {self.last}'

    def myprocess(name):
        first, *last = name.lower().split()
        last = ' '.join(last)
        return first, last

if __name__ == '__main__':
    # staticmethod: Convert a function to be a static method.
    Person.process = staticmethod(Person.myprocess)

    print(Person('Mostafa Saad Ibrahim Mohamed'))
    # Person first name: mostafa  -  last name: saad ibrahim mohamed
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."