# Classes in Python

**Session III - Introduction to ABM**

Ali Seyhun Saral (Uni. Bologna)

14 June 2022

### Table-Based Workflow

| Name | Awards | Partner |
|---|---|---|
| Art | 10 | Paul |
| Cher | 20 | Sonny |
| Paul | 20 | Art |
| Sonny | 5 | Cher |

### Table-Based Workflow

| Name | Awards | Partner |
|---|---|---|
| Art | 10 | Paul |
| Cher | 20 | Sonny |
| Paul | 20 | Art |
| Sonny | 5 | Cher |

```
def addAwards(name):
  table[name, 'Awards'] += 1
```

```
addAwards("Cher")
```

```
def make_partner(name1, name2):
  table[name1, 'Partner'] += name2
  table[name2, 'Partner'] += name1
```

```
}
```

```
make_partner("Art", "Paul")
```

### Object-Oriented Programming

```
cher.add_award()

art.make_partner(paul)
```

. . .

```
cher.awards

## output: 10
```

. . .

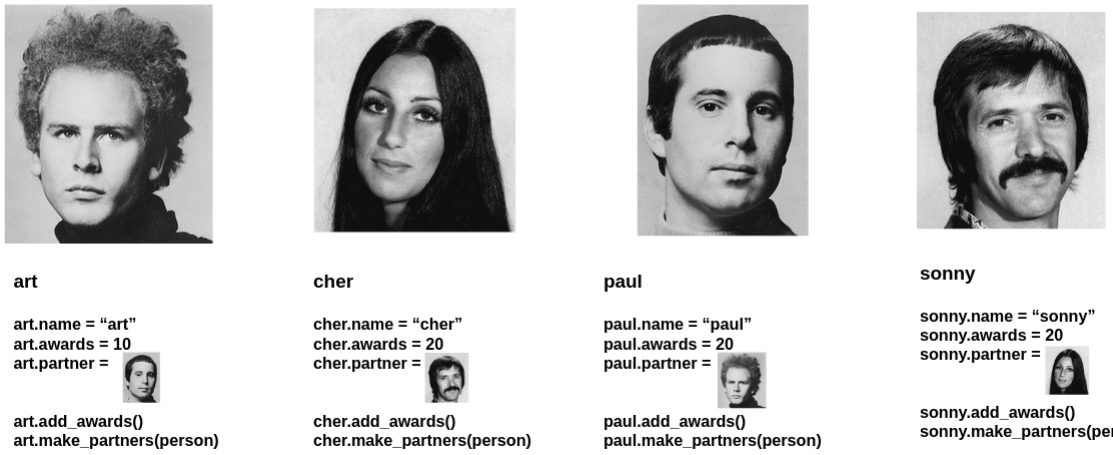
```
art.partner   # paul
paul.partner # art

art.partner.partner # art
art.partner.add_award() # paul gets an award
```

### Object-Oriented Programming

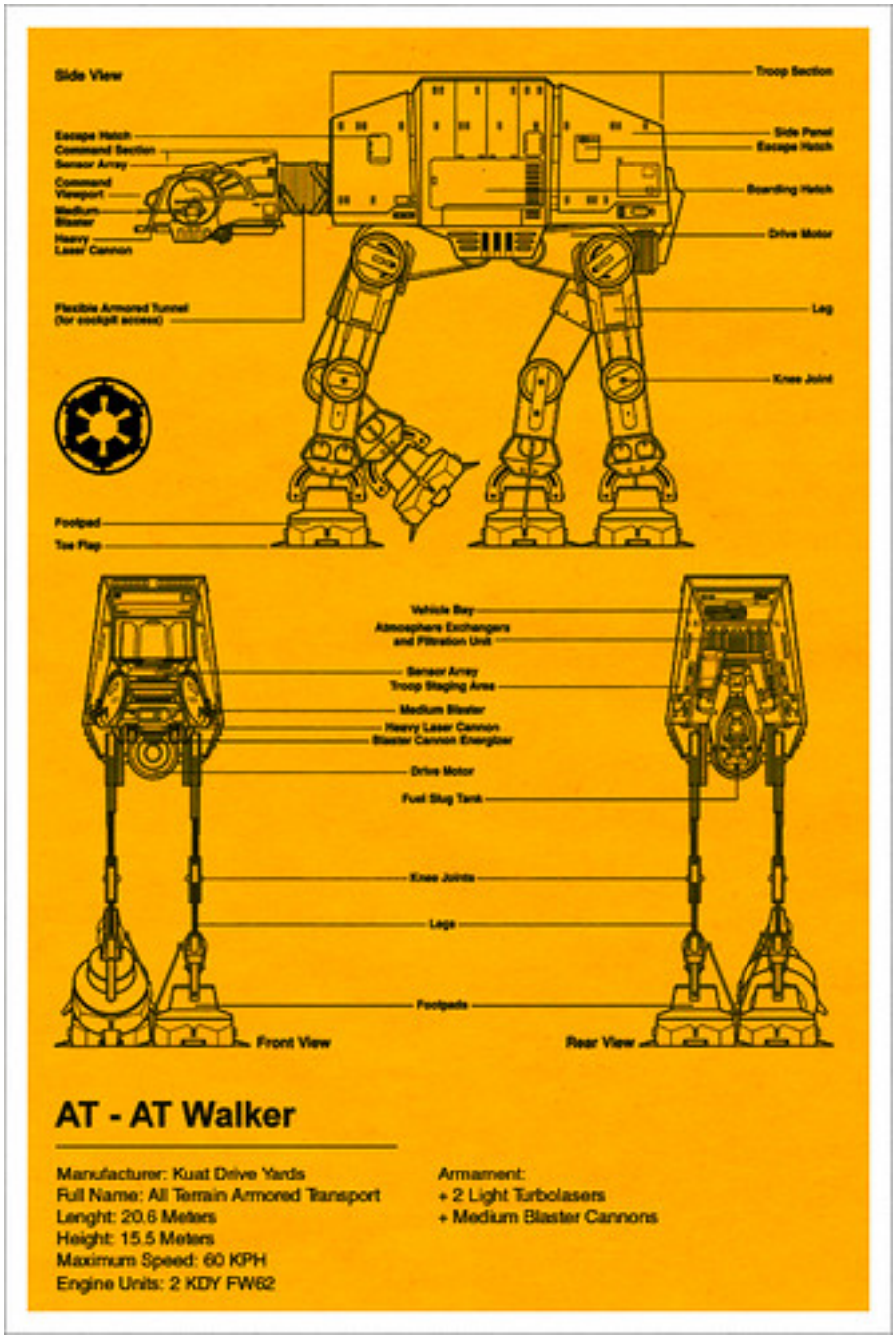
- Objects have some variables `attribute` (`art.award`, `art.partner`)
- Objects have some methods functions attached to them `method` (`art.add_award()`)
- Object are independent units.

## Object-Oriented Programming




## Classes

- Classes are blueprints in which an object is generated.
- An object created wrom a class is called an `instance`
- Usually a class ansers four questions:
  - What are the variables related to it? (`attributes`)
  - What are the functions related to it? (`methods`)
  - What to do when I first create an object? (`__init__`)
  - Is there another class I should inherit from? (`parent class`)


Side View
Troop Section
Escape Hatch
Command Section
Sensor Array
Command Viewport
Medium Blaster
Heavy Laser Cannon
Side Panel
Escape Hatch
Boarding Hatch
Drive Motor
Flexible Armored Tunnel (for cockpit access)
Leg
Knee Joint
Footpad
Toe Flap
Vehicle Bay
Atmosphere Exchangers and Filtration Unit
Sensor Array
Troop Staging Area
Medium Blaster
Heavy Laser Cannon
Blaster Cannon Energizer
Drive Motor
Fuel Slug Tank
Knee Joints
Legs
Footpads
Front View
Rear View
AT - AT Walker
Manufacturer: Kuat Drive Yards
Full Name: All Terrain Armored Transport
Lenght: 20.6 Meters
Height: 15.5 Meters
Maximum Speed: 60 KPH
Engine Units: 2 KDY FW62
Armament:
+ 2 Light Turbolasers
+ Medium Blaster Cannons

## Classes

Generic Singer: Singer class

**Singer**

**self.name =**
**self.awards =**
**self.partner =**

**self.add_awards()**
**self.make_partners(person)**

## Class

Generic Singer: Singer class

**Singer**

**self.name =**
**self.awards =**
**self.partner =**

**self.add_awards()**
**self.make_partners(person)**

pavarotti

vocal_type = “tenor”

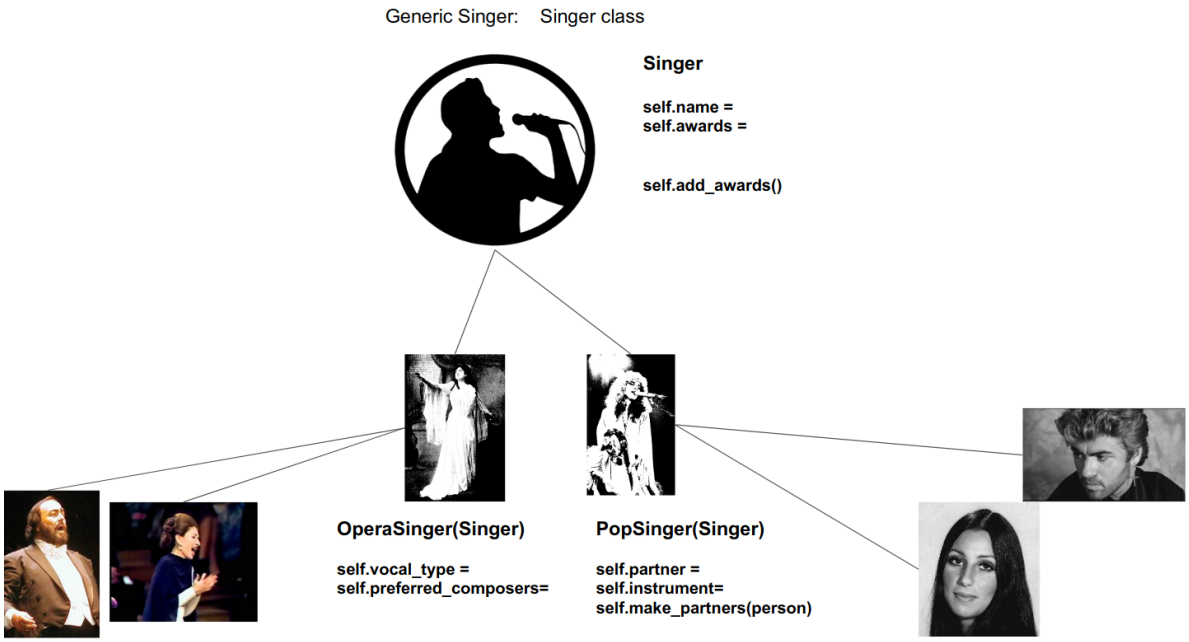
preferred_composers = [“verdi”, “puccini”]

callas

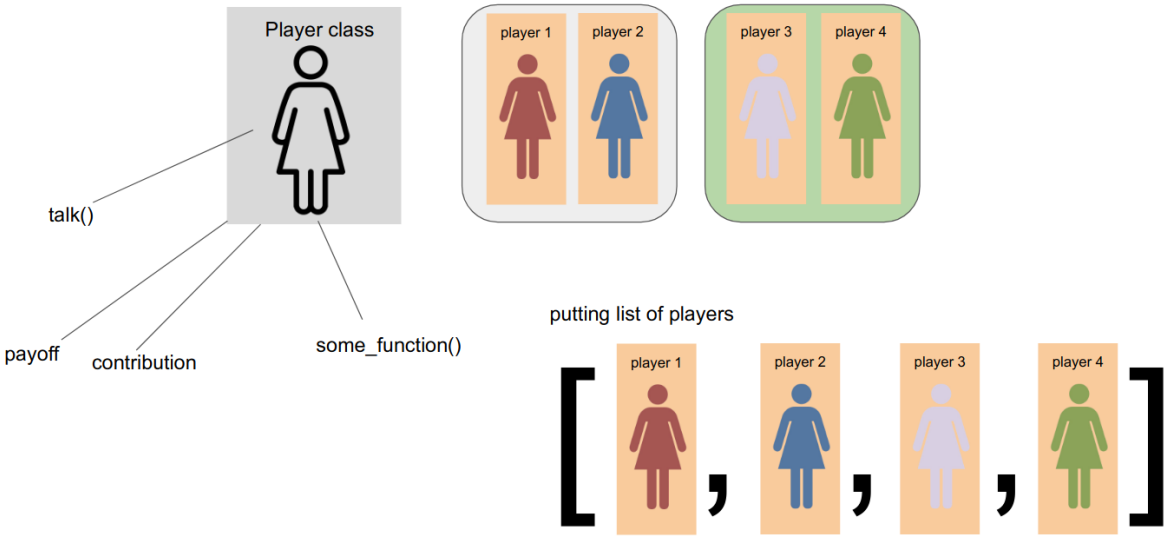
vocal_type = “soprano”

preferred_composers = [“donizetti”, “gluck”]

## Class


Generic Singer: Singer class
Singer
self.name =
self.awards =
self.add_awards()
OperaSinger(Singer)
self.vocal_type =
self.preferred_composers=
PopSinger(Singer)
self.partner =
self.instrument=
self.make_partners(person)

## Class


Player class
talk()
payoff
contribution
some_function()
player 1
player 2
player 3
player 4
putting list of players
player 1
player 2
player 3
player 4

### Object-Orented Programming Properties

1. Encapsulation: Self-contained pieces
2. Abstractation: What is going on underneath is abstracted away
3. Inheritance: Classes can be organized hierarchially
4. Polymorphism: Classes can change into different forms from the parents

### How to create Classes in Python

```
class Singer:
  def __init__(self):
    self.awards = 0

  def sing(self):
    print("la la la laaa!")

  def win_award(self):
    self.award = self.award + 1
```

. . .

- Two unfamiliar things:
  - `__init__`: basically a function run automatically when an instance is created
  - `self`: can be read **myself**. If refers to the instance that is created from it.

. . .

- Creating an insance

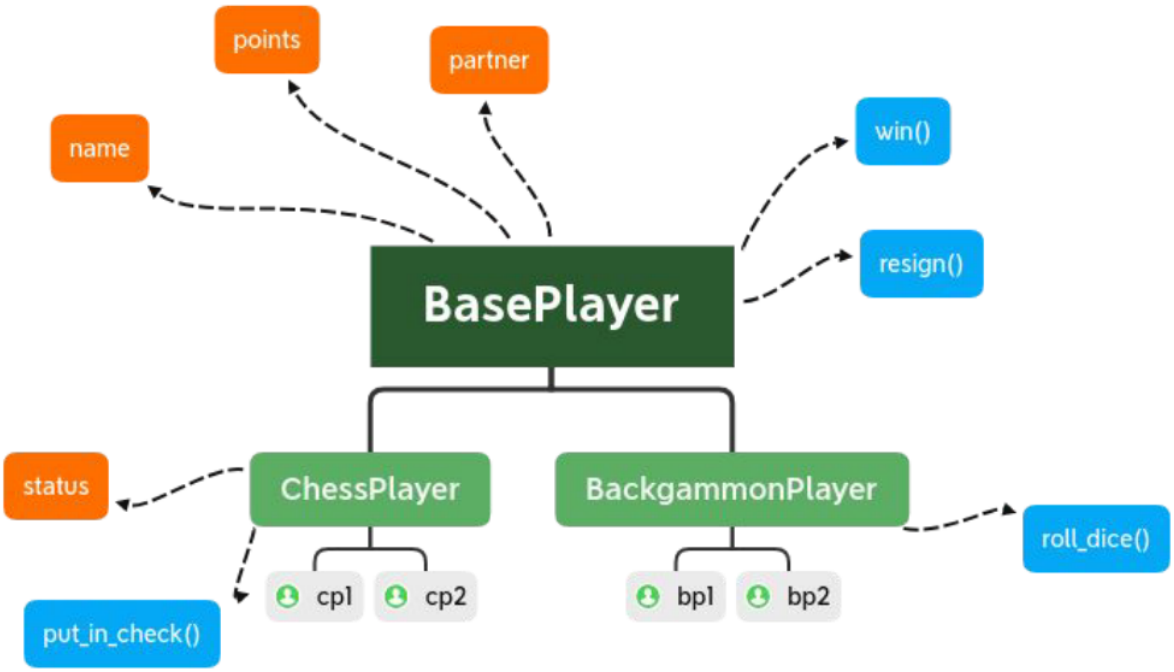
```
ali = Singer()

ali.sing()
```

```
la la la laaa!
```

## Practice

Ex7_classes.ipynb




## Class Inheritance

When you create a class with inheritance, put the name of the parent class inside the parantheses.

```
class Singer:
  def __init__(self):
    self.awards = 0

  def sing(self):
    print("la la la laaa!")

  def win_award(self):
    self.award = self.award + 1
```

. . .

```
class OperaSinger(Singer):
    def sing_aria(self):
      print("Ridi, Pagliaccio... sul tuo amore infranto!")
```

### Class Inheritance

```
angelo = OperaSinger()

angelo.sing()
```

```
la la la laaa!
```

. . .

```
angelo = OperaSinger()

angelo.sing_aria()
```

```
Ridi, Pagliaccio... sul tuo amore infranto!
```

### Class Inheritance

- If you need to reach the parent class from a child class, you can call `super()` function.
- If you define a method with the same name in the child class, the method of the parent will be overwritten.
- If this is not the behavior you want, you can call the super function and extend the method over it.

### Class Inheritance: `super()`

```
class Singer:
  def __init__(self):
    self.awards = 0

  def sing(self):
```

```
    print("la la la laaa!")

  def win_award(self):
    self.award = self.award + 1
```

. . .

```
class OperaSinger(Singer):
    def __init__(self):
      self.vocal_range = "tenor"

    def sing_aria(self):
      print("Ridi, Pagliaccio... sul tuo amore infranto!")
```

. . .

```
angelo = OperaSinger()

angelo.awards
```

```
AttributeError: 'OperaSinger' object has no attribute 'awards'
```

### Class Intheritance: `super()`

```
class Singer:
  def __init__(self):
    self.awards = 0

  def sing(self):
    print("la la la laaa!")

  def win_award(self):
    self.award = self.award + 1
```

. . .

```
class OperaSinger(Singer):
    def __init__(self):
      super().__init__()
      self.vocal_range = "tenor"
```

```

    def sing_aria(self):
      print("Ridi, Pagliaccio... sul tuo amore infranto!")
```

. . .

```
angelo = OperaSinger()

angelo.awards
```

0

### Practice

Ex8_classes2.ipynb