object oriented programming

OOP - Python

1 What is OOP ?

- Paradigm (Coding style)
  - procedural
  - OOP
  - Functional
- Python
  - procedural : sequence of steps (All previous exercises)
  - OOP : Interactions of objects
    - Reusable
    - Readable
    - Maintainable
    - Great for packages

- Everything in python is object
  list, string, tuple, dict., ... etc

2 What is object ?

object = state + behavior

- state → Attributes (Variables)
- behavior → Methods (Functions)

Encapsulation ← Bundling data with code operating on it

- create class (blueprint)
- create instanance (Copy)

```
Class player:
    pass
```

} Class (blueprint) — Template

```
p1 = player()
p2 = player()
```

} instances (Copies)

```
Class player:                                  # create class
    def set_name(self, name):                  # create method
        self.name = name                       # create attribute
```

Not Recommended

maybe any name

refer to the instance

first argument of any method

```
p1 = player()
p1.set_name("Ahmed")        # p1.name = "Ahmed"
print(p1.name)              ⇒ "Ahmed"
dir(player)                 ⇒ shows attributes and Methods
```

* __init__() Constructor

→ To add data once you create an instance from the class

```
class player:

    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = player('Ahmed', 29)
p2 = player('Anas', 5)
```

```
class Customer:

    def __init__(self, name, job, balance):
        self.name = name
        self.job = job
        self.balance = balance
```

Created once instance is created

```
c1 = Customer('Abdo', 'Engineer', 20000)

print(c1.job)      >> 'Engineer'
print(c1.balance)  >> 20000
```

OOP Core principles

① Encabsulation : (Data + Code)

② Inheritance : Extending functionality to childs

③ polymorphism : creating unified interface

④ Abstraction : dealing with outer interface (hide details)

* Inheritance

```
class player:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def set_position(self, position):
        self.position = position

    def set_celebration(self, celebration):
        self.celebration = celebration

class AfricanPlayer(player):
    def __init__(self, color, hair_style):
        player.__init__(self, name, age)
        self.color = color
        self.hair = hair_style

    def set_speed(self, speed):
        self.speed = speed
```

child ← AfricanPlayer(player) → Parent

Parent functionality → player.__init__(self, name, age)

more functionality → self.color = color, self.hair = hair_style, def set_speed(self, speed): self.speed = speed

*Polymorphism

```
class player:

    def set_name(self, fname, lname):
        self.first_name = fname
        self.last_name = lname
        print(f"Name is set to {self.first_name} {self.last_name}")


class Employee:
    def set_name(self, name):
        self.name = name
        print(f"Hello Mr. {name}")

p1 = player()
E1 = Employee()

p1.set_name('Ahmed', 'Mostafa')
    >> Name is set to Ahmed Mostafa

E1.set_name('Ahmed')
    >> Hello Mr. Ahmed
```