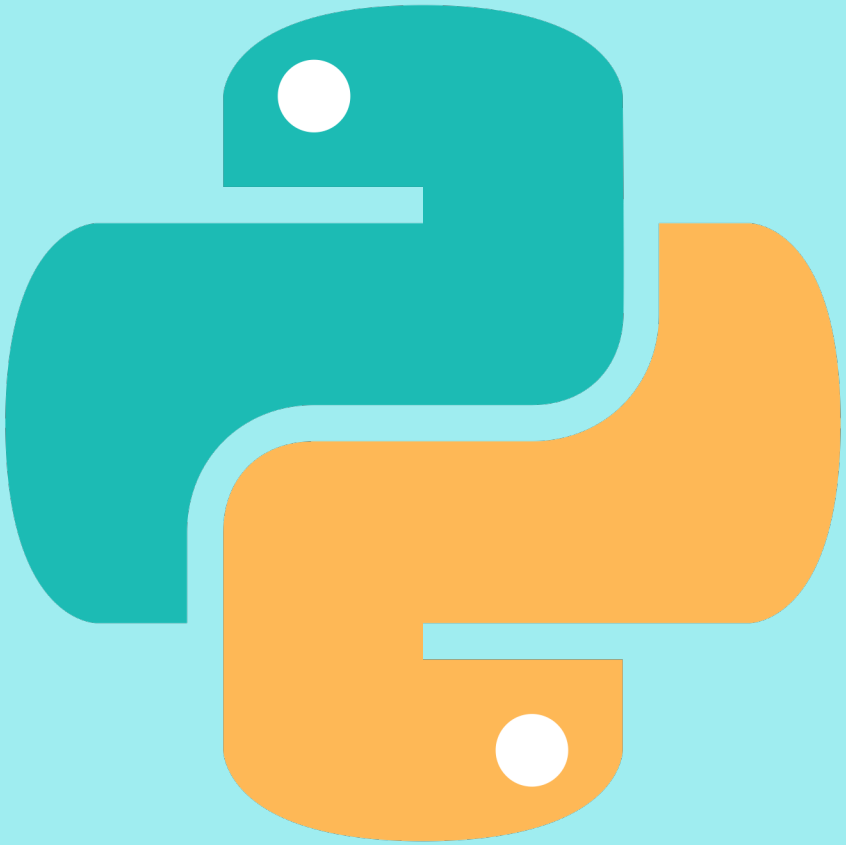
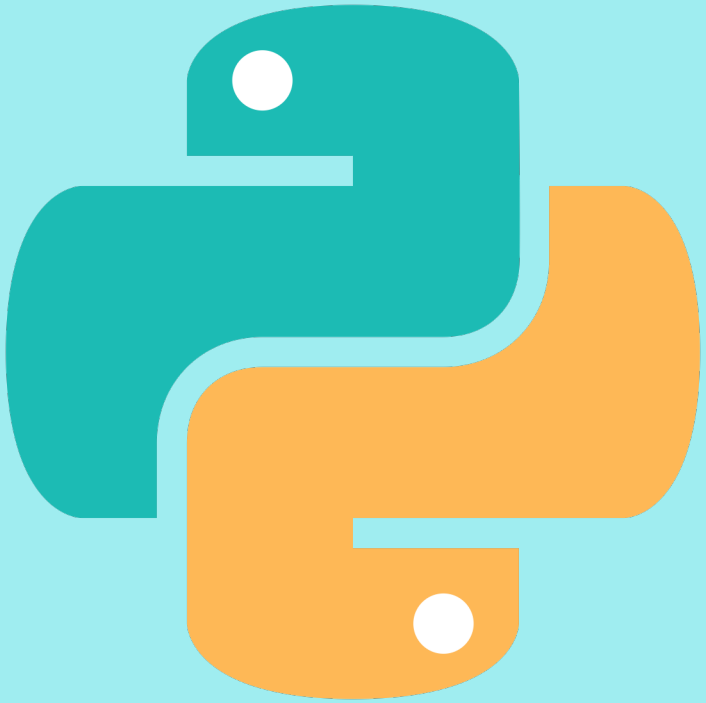


پایتون پیشرفته

علیرضا کیانی

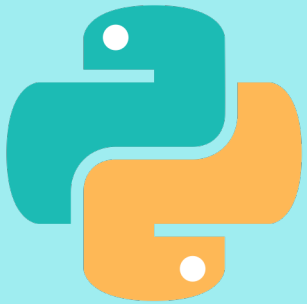
https://www.linkedin.com/in/alireza-kiani/

سرفصل ها



برنامه‌نویسی شی گرا

Object Oriented Programming



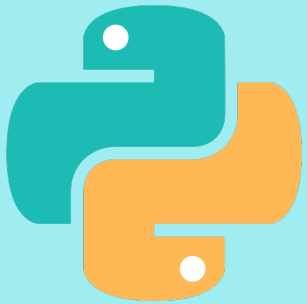
Object Oriented Programming

«برنامه‌نویسی شی‌گرا» (Object-Oriented Programming یا به اختصار OO)
Pیک الگو یا شیوه تفکر در برنامه‌نویسی است که برگرفته از دنیای واقعی بوده و از ده ۱۹۶۰ ه میلادی مطرح گشته است. به زبانی که از این الگو پشتیبانی کند، «زبان شی‌گرا» گفته می‌شود

رویکرد برنامه‌نویسی شی‌گرا «از پایین به بالا» (Bottom-Up) است؛ یعنی ابتدا واحدها یی کوچک از برنامه ایجاد می‌شود و سپس با پیوند این واحدها، واحدهایی بزرگتر و در نها یت شکلی کامل از برنامه به وجود می‌آید. برنامه‌نویسی شی‌گرا در قالب دو مفهوم «کلاس » (class)و «شی» (Object) ارایه می‌گردد.
هر کلاس واحدی از برنامه است که تعدادی داده و عملیات را در خود نگهداری می‌کند و ه ر شی نیز حالتی (State) مشخص از یک کلاس می‌باشد.

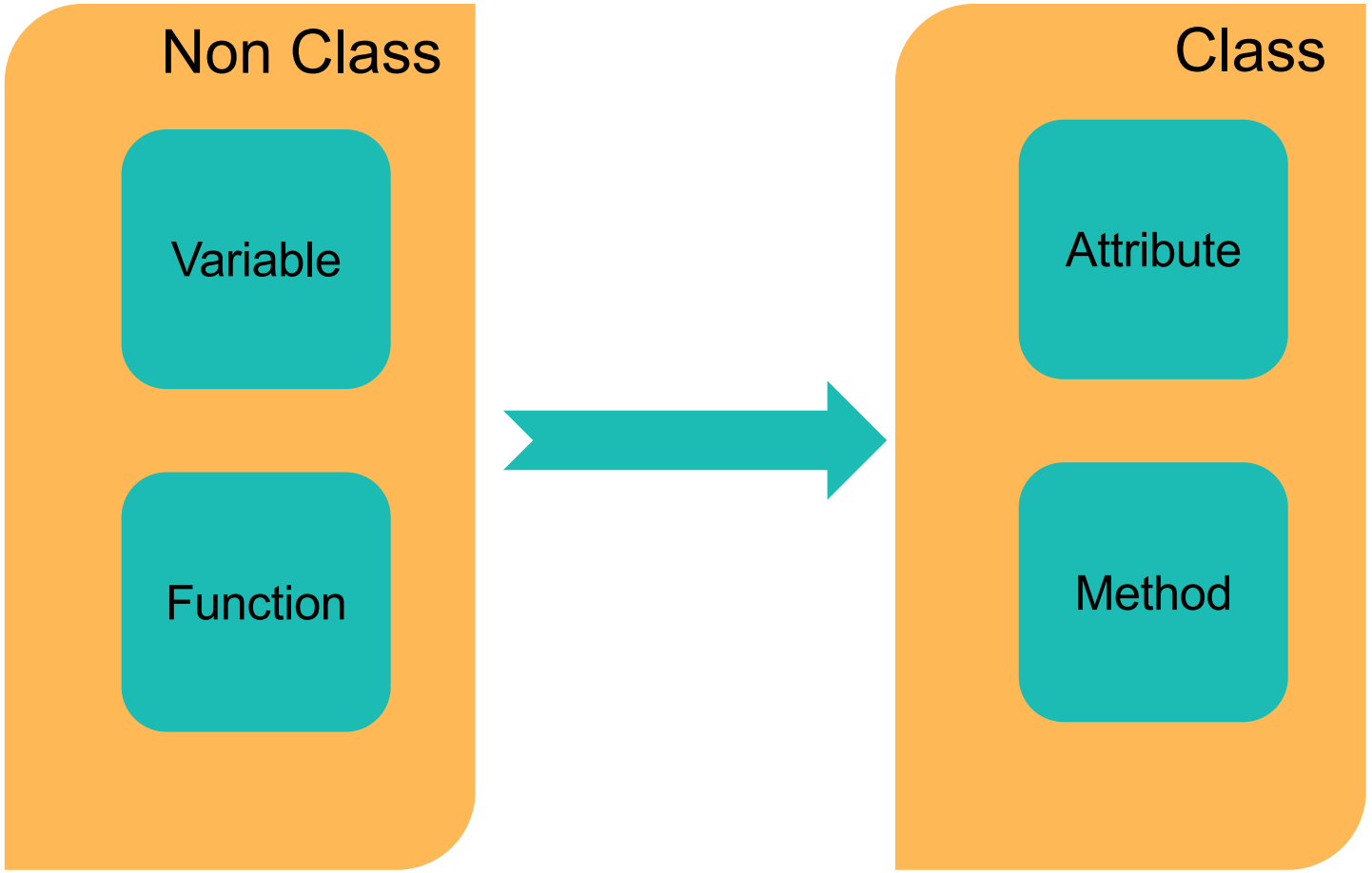
در برنامه‌نویسی شی‌گرا، هر برنامه در قالب موجودیت‌های کوچکی که در واقع همان اشیا هستند و با یکدیگر تعامل دارند در نظر گرفته می‌شود.

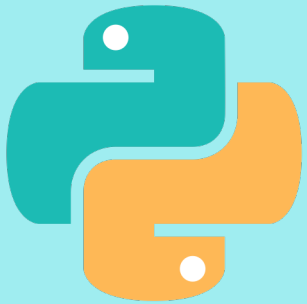
برای داشتن این اشیا می‌بایست ابتدا کلاس‌های برنامه را تعریف نماییم؛
از یک کلاس می‌توان هر تعداد که بخواهیم شی ایجاد نماییم. هر شی بیانگر یک «حالت» یا یک «نمونه» (Instance)از کلاس خود است.



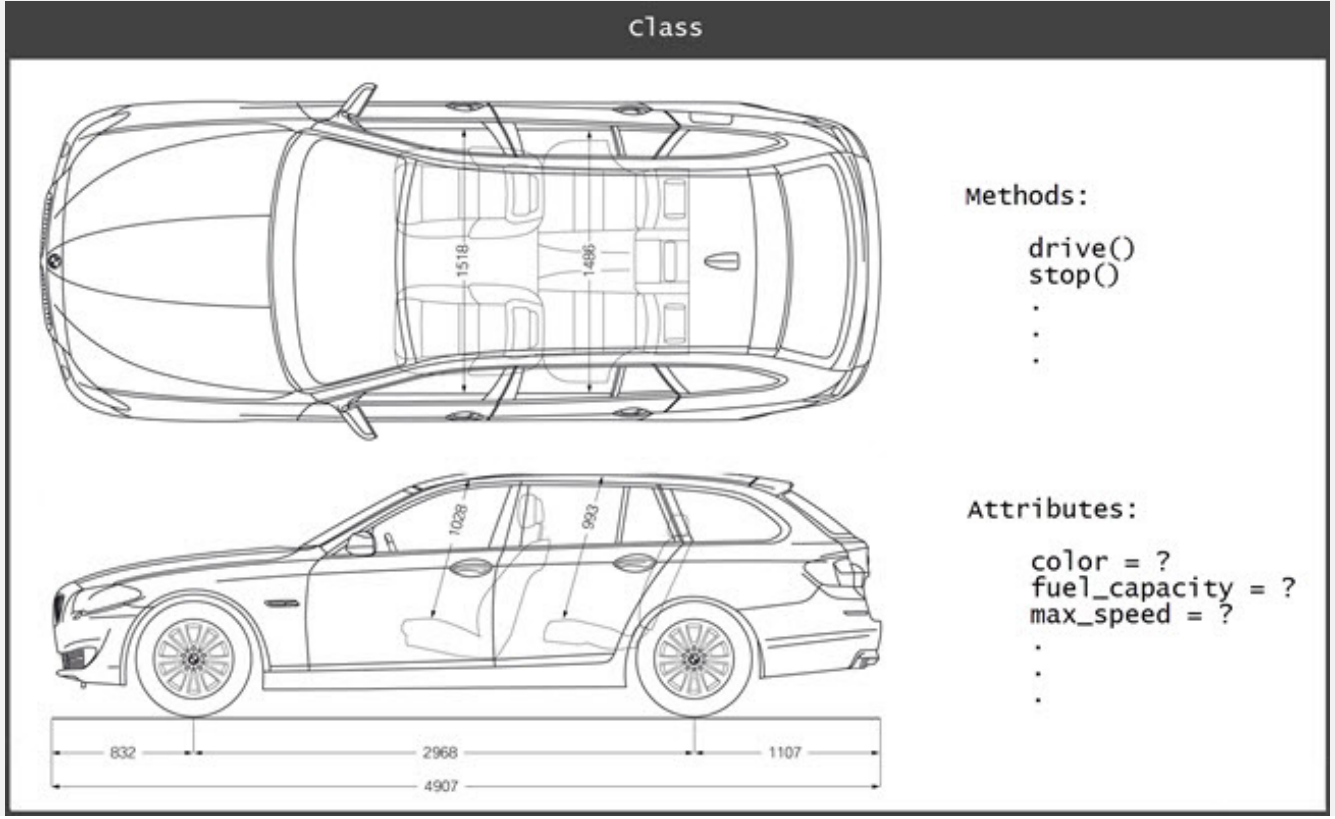
Object Oriented Programming Cont.

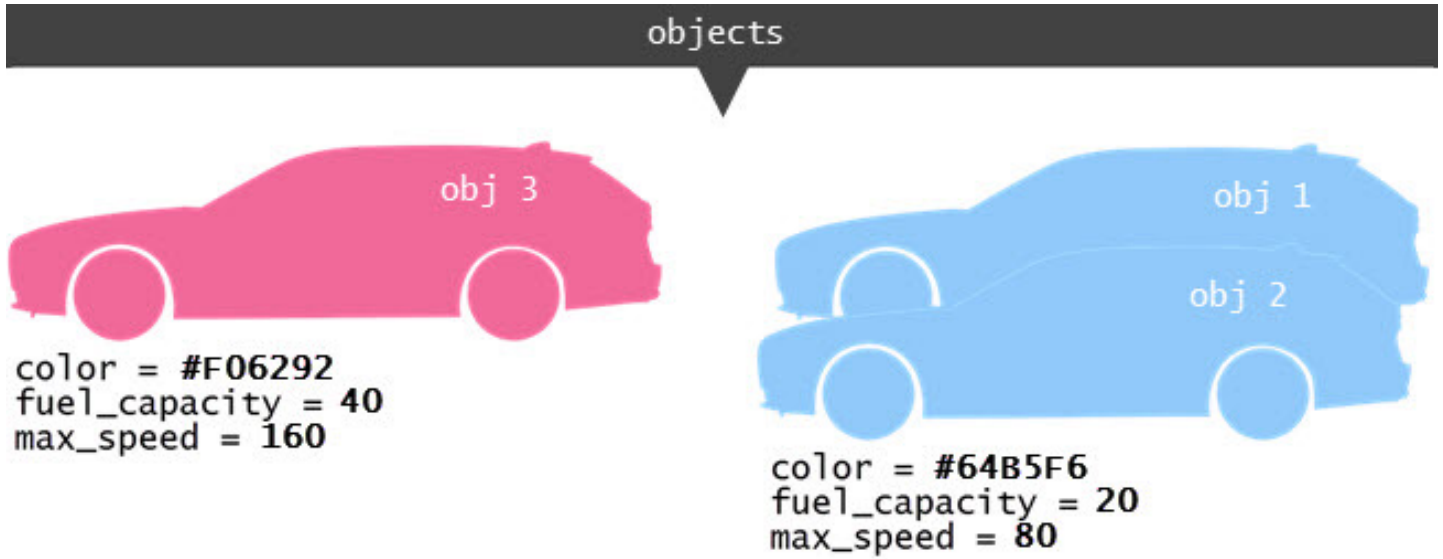
هر کلاس از تعدادی داده و عملیات در درون خود نگهداری می‌کند

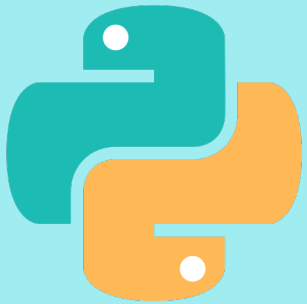




Class to Object







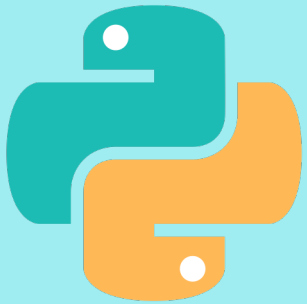
Why OOP

- کپسوله‌سازی: داده‌ها و توابع مرتبط در یک کلاس محفوظ بمانند و از دسترسی غیرمجاز جلوگیری شود.

- چندریختی: توابع با یک نام ولی با پارامترها ی مختلف تعریف شوند.

- ارث‌بری: امکان استفاده مجدد از کد را فراهم می‌آورد و به توسعه‌دهندگان اجازه می‌دهد تا ویژگی‌های یک کلاس را به کلاس‌های دیگر منتقل کنند.

خلاصه نویسی: جلوگیری از کد نویسی زیاد



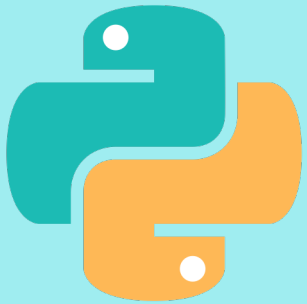
Class and Object Definition

- تعریف کلاس:

```
Class ClassName:
    …
```

- تعریف شی:

```
Obj1=ClassName()
```

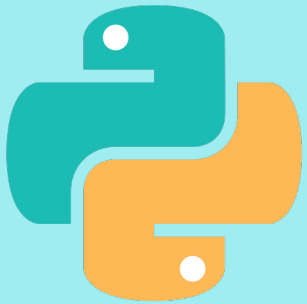


Class Variables VS. Instance Variables

```python
class Television:
    shape='Rectangle'
    inputs=['hdmi','usb3']

    def playMusic(self):
        print("Play music")

#1
print(Television.shape)

#2
lg=Television()
print(lg.shape)

#3
print(getattr(Television,'shape'))
```

متغیرهای کلاس `class Variables`: این نوع متغیر در بدنه‌ی کلاس تعریف شده و در تما
می شی‌های ایجاد شده از این کلاس، مشترک است. از این متغیرها به ندرت استفاده می‌شود
.

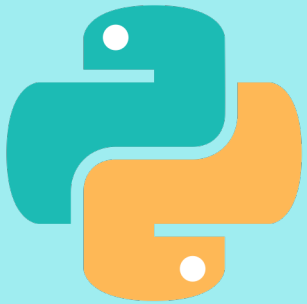
متغیرهای نمونه`Instance Variables`: این نوع متغیرها مختص نمونه‌ی ایجاد شده از کلا
س هستند و در متد سازنده یا سایر متدهای کلاس تعریف می‌شوند. هر شی متغیر نمونه‌ی م
خصوص خود دارد.



Constructors in Classes

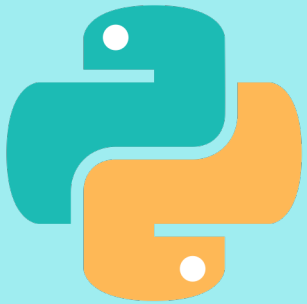
```python
class EmployeeClass:
    baseSalary=1000

    def __init__(self,name,salary):
        self.name=name
        self.salary=salary+self.baseSalary

    def displayBaseSalary(self):
        print(f"Total Employee: {self.baseSalary}")

    def displayEmployee(self):
        print(f"Name: {self.name}, Salary: {self.salary}")

emp1=EmployeeClass("Alireza",7000)

print(emp1.displayEmployee())
```

یکی از این متدها که سازنده‌ی کلاس نیز محسوب می‌شود، متد init است. مقداردهی اولیه ش ی و نیز دستورالعمل‌هایی که در زمان ایجاد شی باید اجرا شود، به وسیله‌ی این متد انجام می‌پذ یرد.



Instance Method
VS.
Class Method

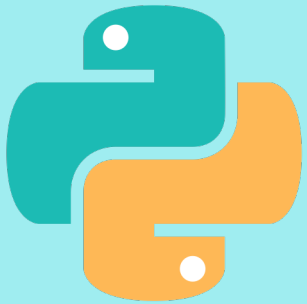
```python
from datetime import date

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

    @classmethod
    def calculate_age(cls, name, birth_year):
        # calculate age an set it as a age
        # return new object
        return cls(name, date.today().year - birth_year)

    def show(self):
        print(self.name + "'s age is: " + str(self.age))


joy = Student.calculate_age("Joy", 1995)
joy.show()
```



Static Method

```python
class MathOperations:
    PI = 3.14159

    @staticmethod
    def circle_area(radius):
        return MathOperations.PI * (radius ** 2)


# استفاده از متد بدون نیاز به نمونه‌سازی از کلاس
radius = 5
area = MathOperations.circle_area(radius)
print(f"The area of the circle with radius {radius} is {area}")
```

زمانی استفاده می‌شود که بخواهید یک متد بدون نیاز به نمونه‌سازی از کلاس فراخوانی شود.
این متد به `self` یا `cls` نیاز ندارد و به طور مستقیم به کلاس مرتبط است.



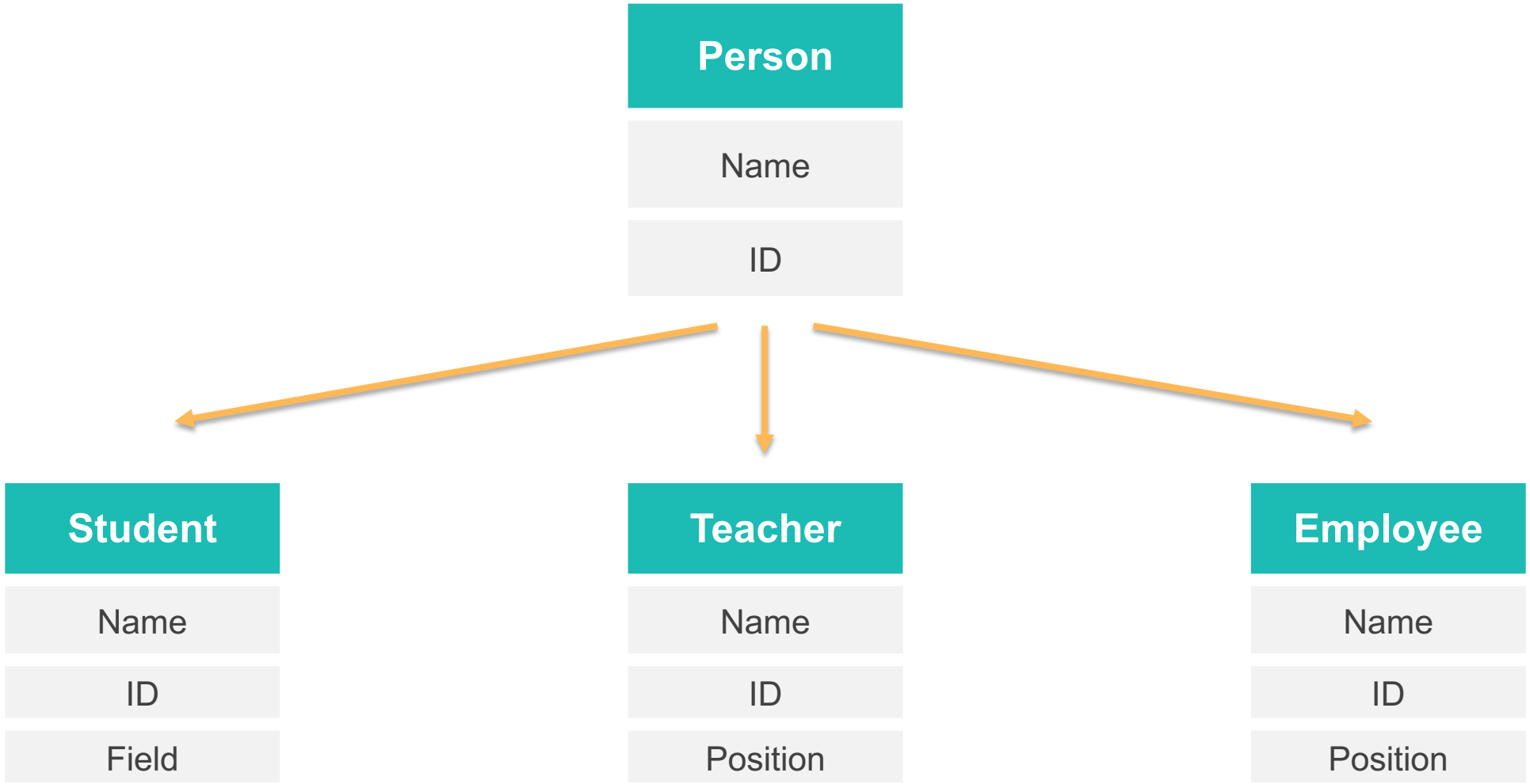
OOP Sample

app_2.py > DBManager > __init__

```python
import sqlite3

class DBManager:
    def __init__(self):
        self.connect1=sqlite3.connect("mydb_1.db")
        self.cur=self.connect1.cursor()
        self.cur.execute("CREATE TABLE IF NOT EXISTS table1(id INTEGER PRIMARY KEY ,\
            name TEXT,salary INTEGER)")
        self.connect1.commit()
    def insert_db(self,name,salary):
        data=[]
        data.append(name)
        data.append(salary)
        self.cur.execute("INSERT INTO table1(name,salary) VALUES(?,?)",data)
        self.connect1.commit()
    def export_db(self):
        self.cur.execute("SELECT * FROM table1")
        print(self.cur.fetchall())


db_m_1=DBManager()
db_m_1.insert_db(name="Alireza",salary=7000)
db_m_1.insert_db("Reza",6000)
db_m_1.export_db()
```

```python
import sqlite3
connect1=sqlite3.connect("mydb.db")
cur=connect1.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS table1(id INTEGER PRIMARY KEY , \
    name TEXT,salary INTEGER)")
connect1.commit()
data=["Alireza",7000]
cur.execute("INSERT INTO table1(name,salary) VALUES(?,?)",data)
connect1.commit()

cur.execute("SELECT * FROM table1")
output=cur.fetchall()
print(output)
```

What is Inheritance ?



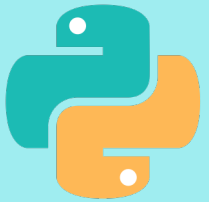
Inheritance In Python

وراثت یا ارث‌بری در شی گرایی، امکانی هست که یک کلاس می‌تواند خصوصیات یک کلاس دیگر را به ارث ببرد و علاوه بر آن خصوصیات دیگری نیز داشته باشد. بعضی از مزایای ارث‌بری:

•قابلیت استفاده‌ی مجدد از کد را مهیا کرده و از تکرار کدهای مشابه جلوگیری می‌کند و به ما امکان می‌دهد بدون تغییر کلاس، ویژگی‌های جدیدی به آن اضافه کنیم.
•ماهیت آن انتقالی است، به این معنی که اگر کلاس B از کلاس A ارث می‌برد پس تمام زیر کلاس‌های B نیز از کلاس A ارث می‌برند.

المان ها:

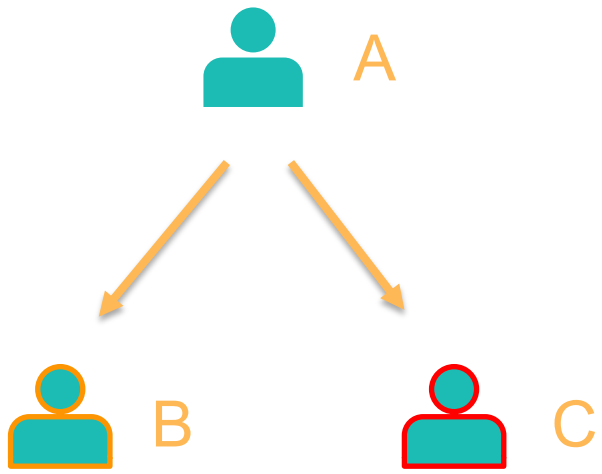
- کلاس والد یا SuperClass
- زیرکلاس یا فرزند یا SubClass



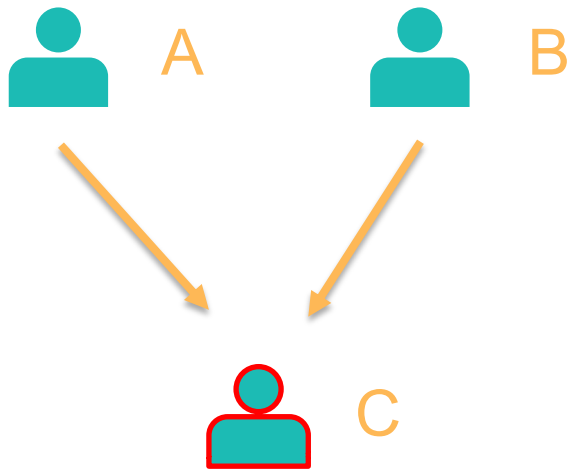
Type of Inheritance in Python

Single

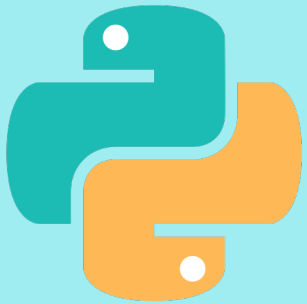
Multilevel



Heretical



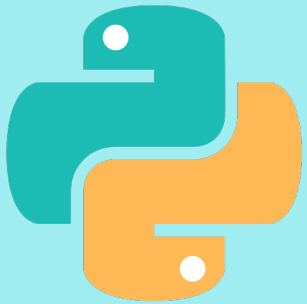
Multiple



```python
class Person:
    def __init__(self,name,id):
        self.Name=name
        self.ID=id
    def exportPersonInfo(self):
        print(f"Name: {self.Name} , ID: {self.ID} ")

class Student(Person):
    def __init__(self, name, id,field):
        super().__init__(name, id)
        self.Field=field

    def exportStudentInfo(self):
        print(f"Name: {self.Name} , ID: {self.ID} , Unit: {self.Field} ")

s1=Student("Ali",123,"Computer")
s1.exportStudentInfo()
s1.exportPersonInfo()
```

Basic Of Inheritance

اصول وراثت
Super
Init فرزند
Allfather: Object



Privacy in Inheritance

```python
class Father:
    def __init__(self):
        self.ToJibi = 21
        self.__Hoghogh = 42
class Child(Father):
    def __init__(self):
        self.Makharej = 84
        Father.__init__(self)
farzande_vasat = Father()
print(farzande_vasat.Hoghogh)
```

ایجاد محدودیت در دسترسی توسط فرزندان !!

Class Encapsulation In Python

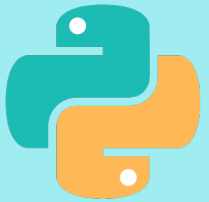
یکی از مفاهیم بنیادی برنامه نویسی شی گرا، کپسوله سازی است.
کپسوله سازی از بسته‌بندی داده‌ها و متدهایی که در داخل یک واحد کار می‌کنند، به وجود آمده است.
این کار کمک می‌کند تا دسترسی مستقیم به متغیرها و متدها محدود شده و از تغییر تصادفی داده‌ها جلوگیری شود.
در این روش، تغییر متغیر یک شی فقط با متد همان شی امکان‌پذیر است. این نوع متغیرها به عنوان متغیر خصو صی شناخته می‌شود.
یک کلاس، نمونه‌ای از کپسوله سازی است، زیرا تمامی داده‌های توابع عضو، متغیرها و غیره را در خود قرار داده است.

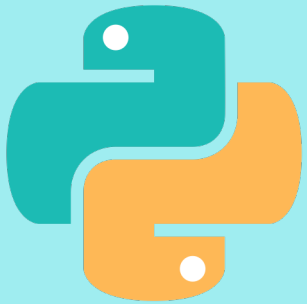
اعضای محافظت شده (Protected members)

خارج از کلاس قابل دسترسی نیستند، اما از داخل کلاس و زیرکلاس‌های آن قابل دسترسی هست ند. برای رسیدن به این مقصود در پایتون، باید قبل از نام عضو یک خط زیر "_" گذاشت.

اعضای خصوصی (Private members)

اعضای خصوصی مشابه اعضای محافظت شده هستند با این تفاوت که اعضای خصوصی فقط داخل خود کلا س قابل دسترسی‌اند و در خارج از کلاس و هیچ کلاس پایه‌ی دیگری قابل دسترسی نیستند. در پایتون، برای تعریف یک متغیر خصوصی از پیشوند دو خط زیر "__" قبل از نام متغیر استفاده می‌کنند.





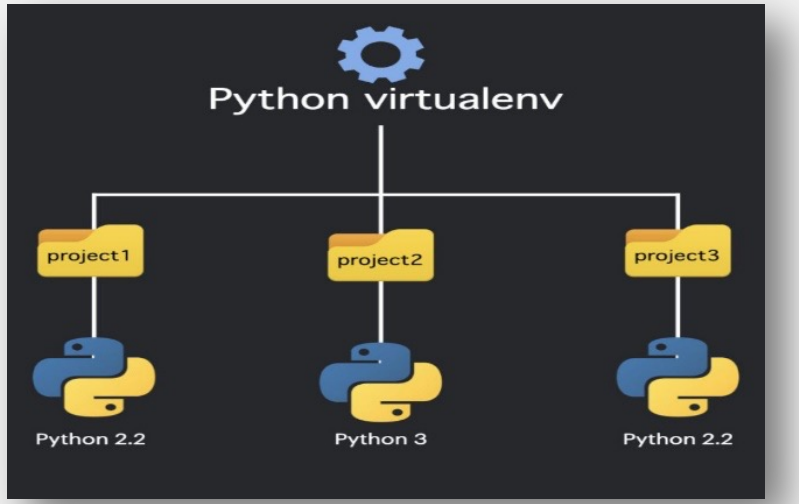
Polymorphism

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

    def info(self):
        print(f"I am a cat. My name is {self.name}. I am {self.age} years old.")

    def make_sound(self):
        print("Meow")

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

    def info(self):
        print(f"I am a dog. My name is {self.name}. I am {self.age} years old.")

    def make_sound(self):
        print("Bark")


cat1 = Cat("Dokme", 0.5)
dog1 = Dog("Temi", 2.5)

for animal in (cat1, dog1):
    animal.make_sound()
    animal.info()
    animal.make_sound()

```

مقایسه شود با چند ریختی در توابع
امکان استفاده متدهای مشترک بین چندین کلاس مختلف داده می شود

Virtual Environment

Working With Different Version of python and packages



1- make a new folder: mkdir c:\prj_1

2- install virtual environment tool: pip install virtualenv

3- make virtual environment in folder you can use different version of python:
virtualenv –p="FolderOfPython/python.exe" "c:\prj_1\"

4- activate vritual environment: c:\prj_1\scripts\activate.bat

5- using virtual environment is vscode: code .

6- create new file in project folder in vscode

7- using ctrl+shift+p and type "python environment" and select venv of prj_1

8- you can run your code in virtual environment

9- to deactivate : c:\prj_1\scripts\deactivate.bat