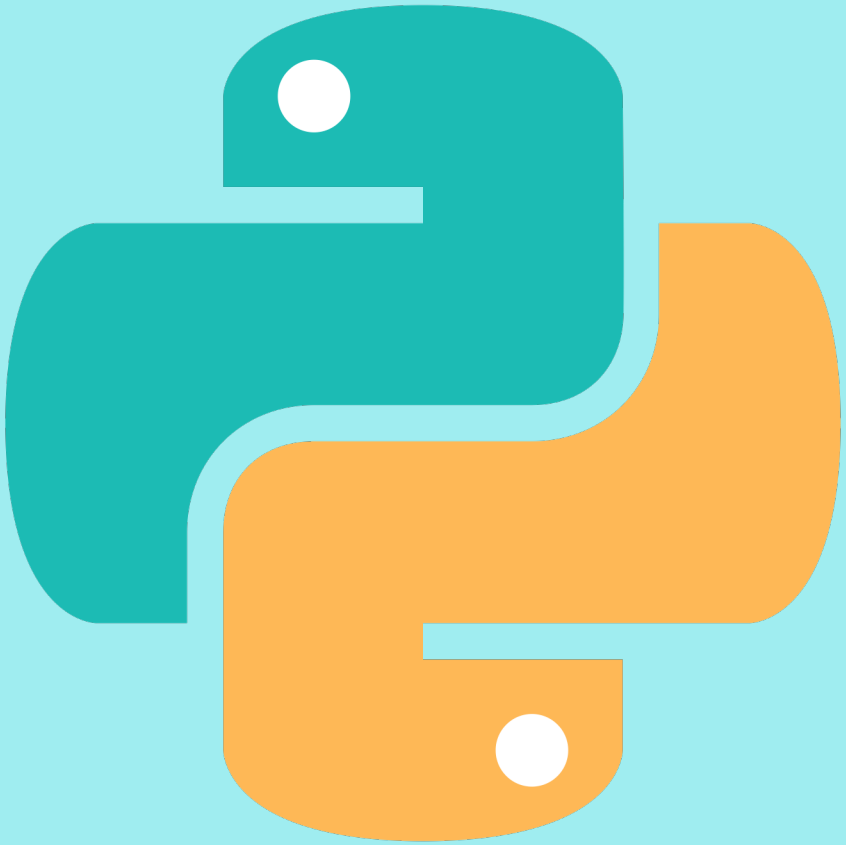
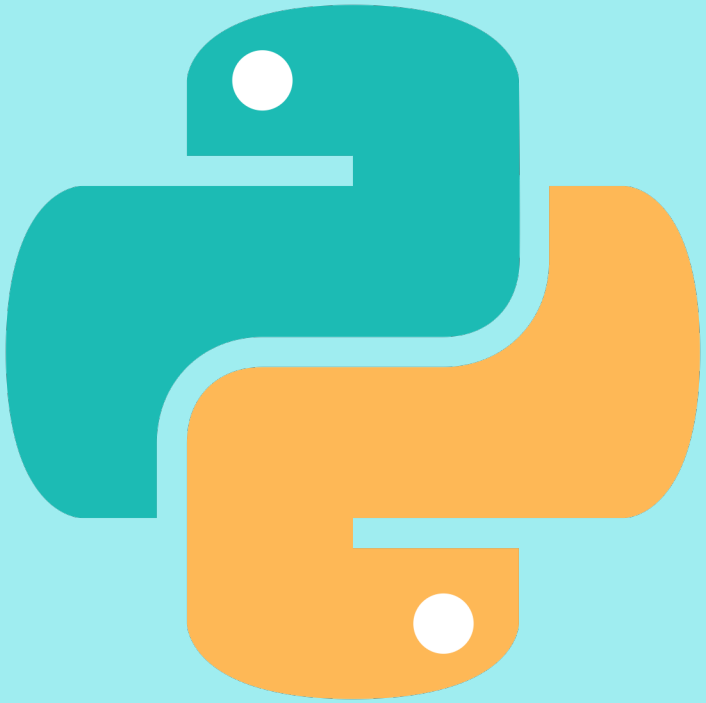


پایتون پیشرفته

علیرضا کیانی

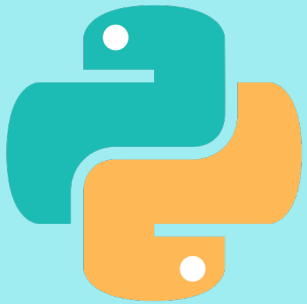
https://www.linkedin.com/in/alireza-kiani/

سرفصل ها



برنامه‌نویسی شی گرا

Object Oriented Programming



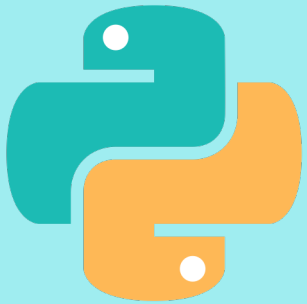
Object Oriented Programming

«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.

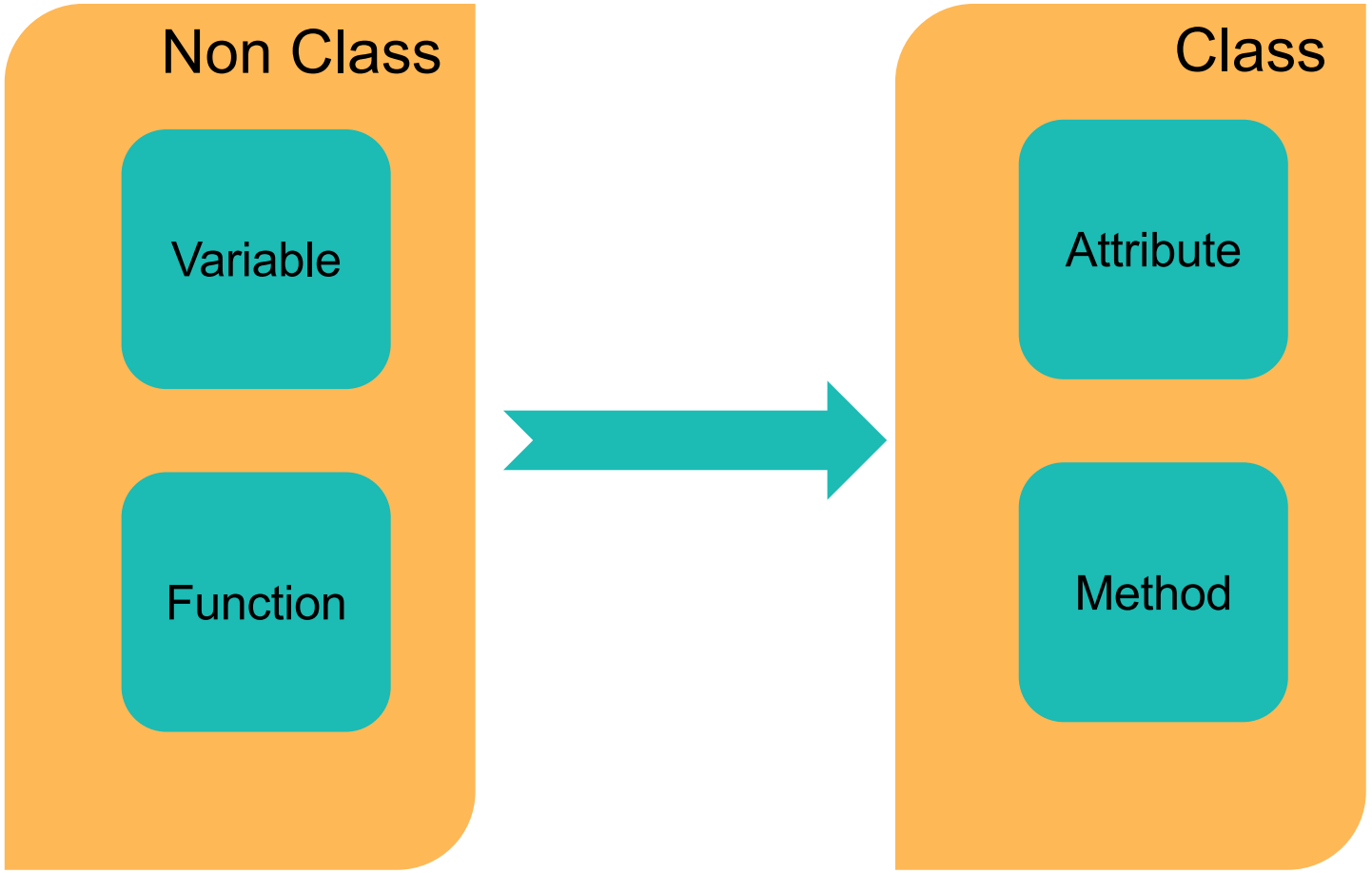
در برنامه‌نویسی شی‌گرا، هر برنامه در قالب موجودیت‌های کوچکی که در واقع همان اشیا هستند و با یکدیگر تعامل دارند در نظر گرفته می‌شود.

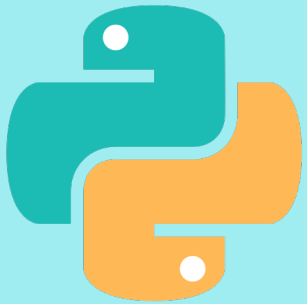
برای داشتن این اشیا می‌بایست ابتدا کلاس‌های برنامه را تعریف نماییم؛
از یک کلاس می‌توان هر تعداد که بخواهیم شی ایجاد نماییم. هر شی بیانگر یک «حالت» یا یک «نمونه» (Instance)از کلاس خود است.



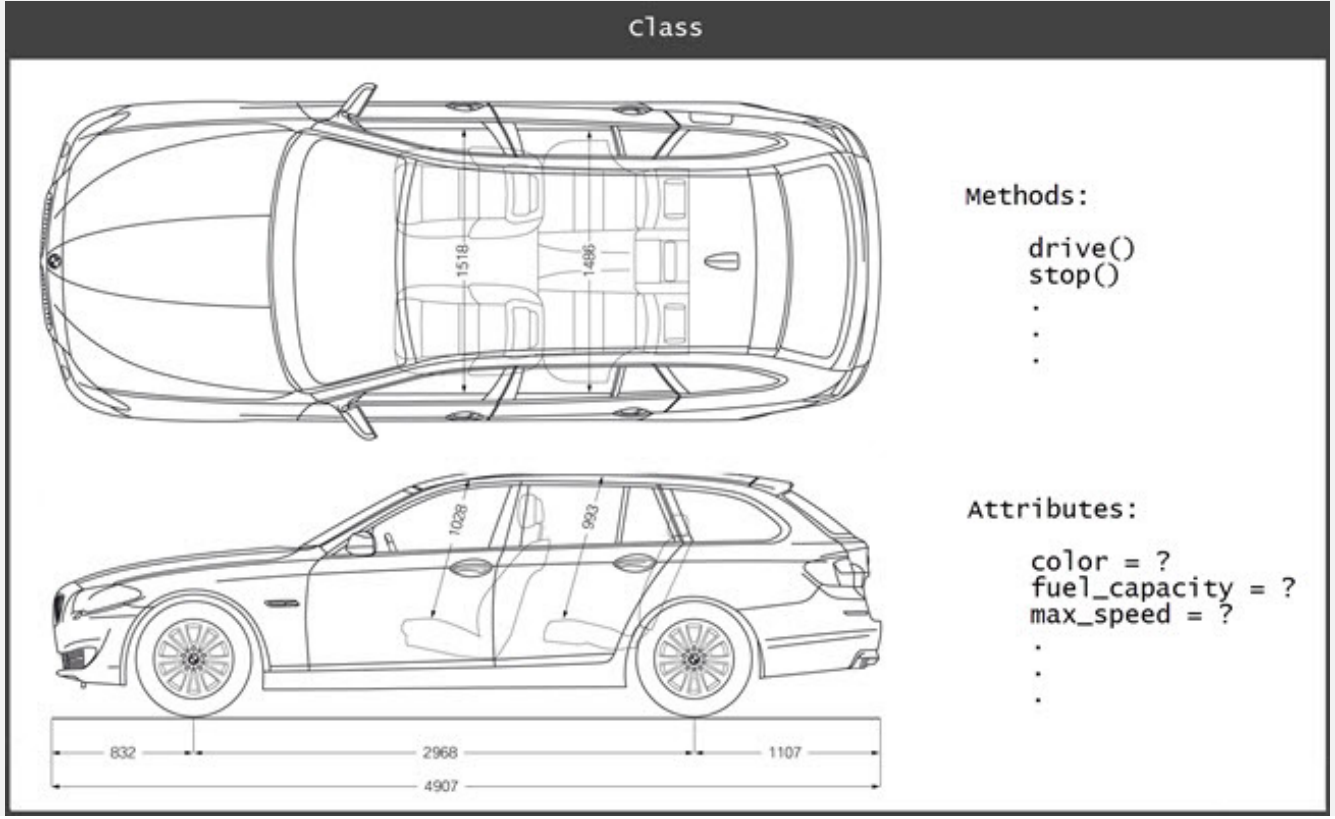
Object Oriented Programming Cont.

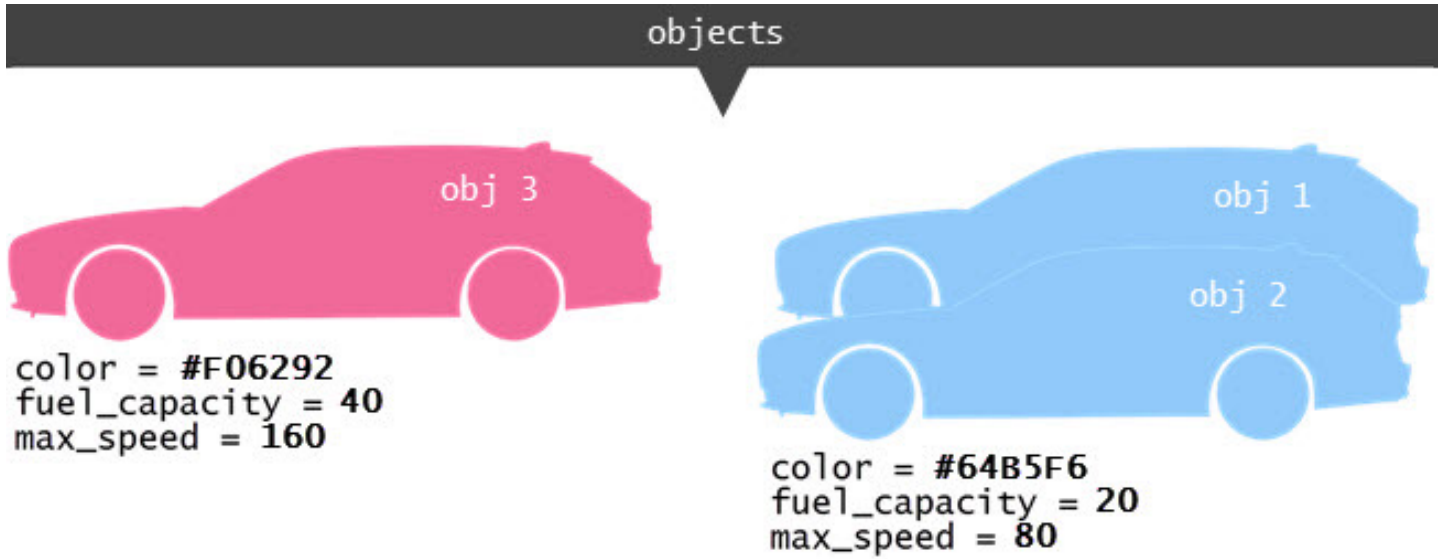
هر کلاس از تعدادی داده و عملیات در درون خود نگهداری می‌کند

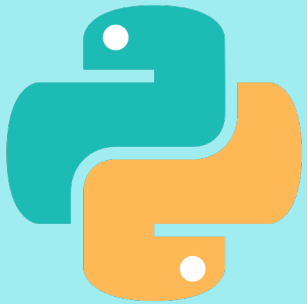




Class to Object







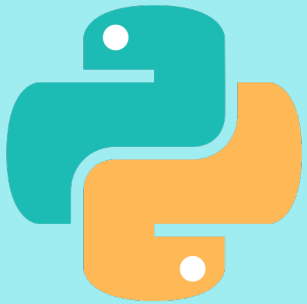
Why OOP

- 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



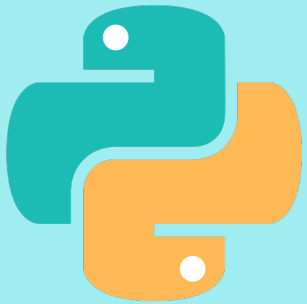
Class and Object Definition

- تعریف کلاس:

```
Class ClassName:
    …
```

- تعریف شی:

```
Obj1=ClassName()
```

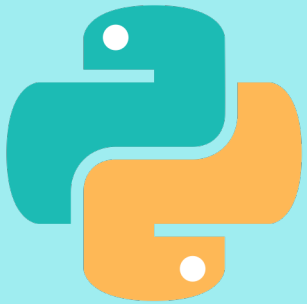


Class Variables VS. Instance Variables

```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
.

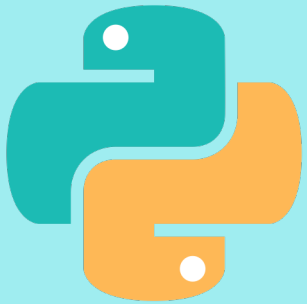
متغیرهای نمونه`Instance Variables`: این نوع متغیرها مختص نمونه‌ی ایجاد شده از کلا
س هستند و در متد سازنده یا سایر متدهای کلاس تعریف می‌شوند. هر شی متغیر نمونه‌ی م
خصوص خود دارد.



Constructors in Classes

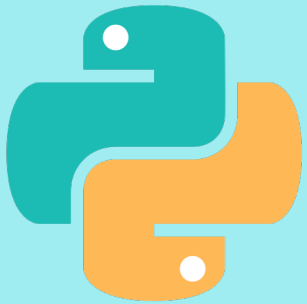
```python
class EmployeeClass:
    baseSalary=1000

    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

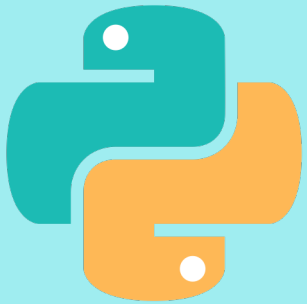
```python
from datetime import date

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.



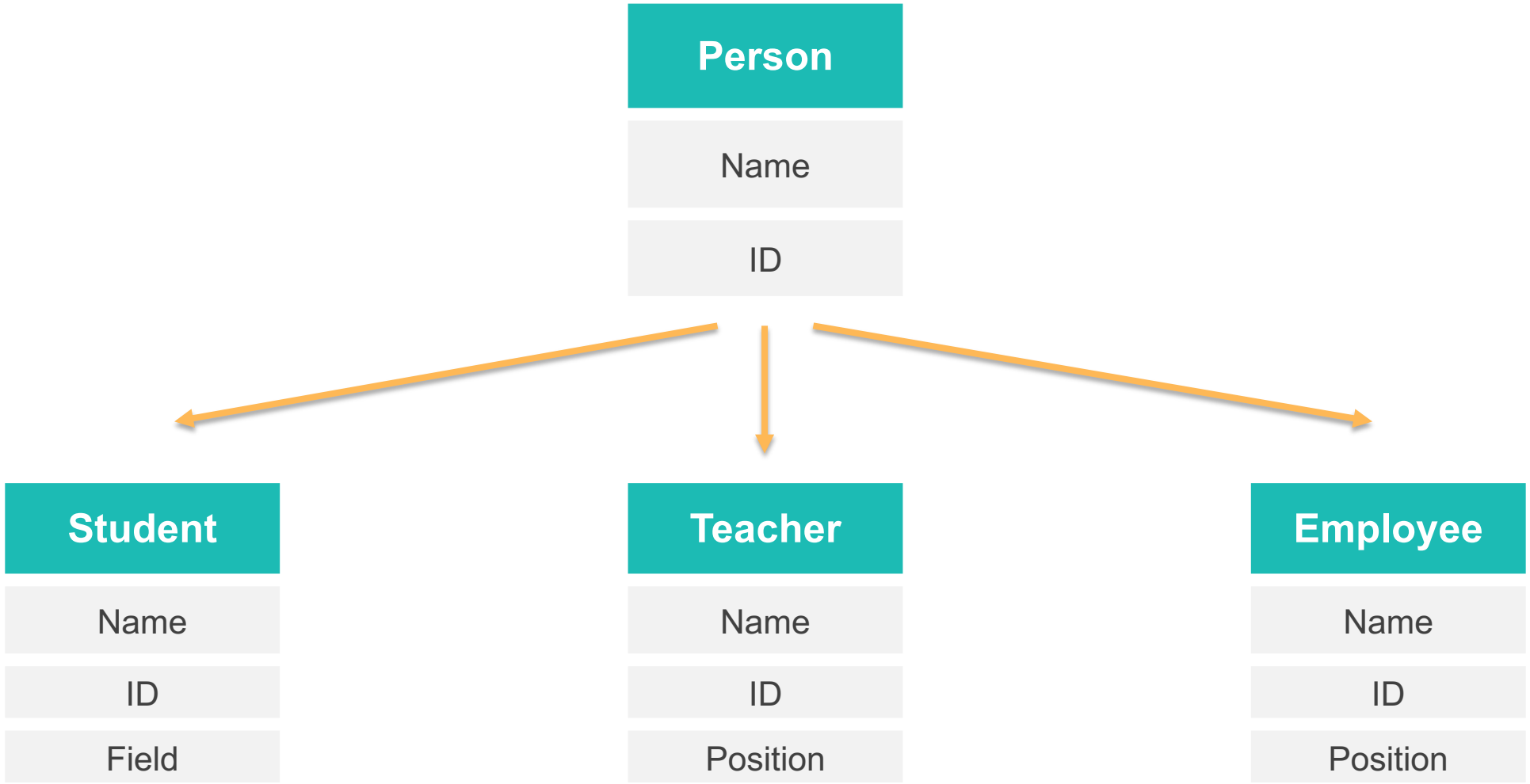
OOP Sample

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 ?



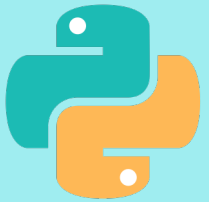
Inheritance In Python

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:

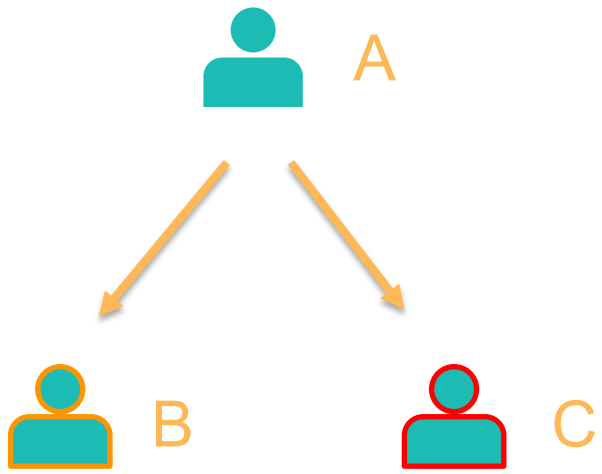
- کلاس والد یا SuperClass
- زیرکلاس یا فرزند یا SubClass



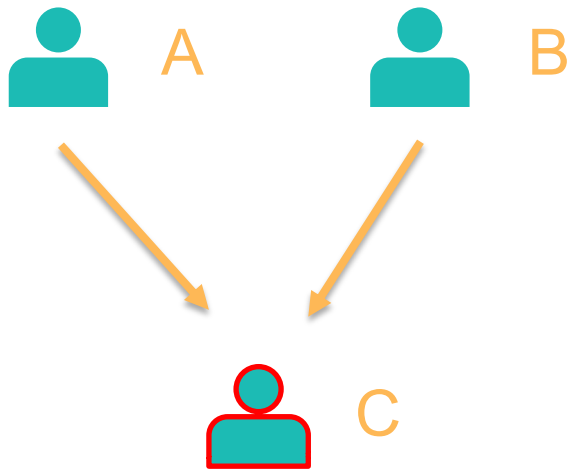
Type of Inheritance in Python

Single

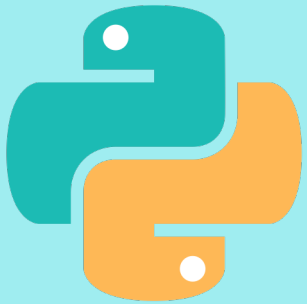
Multilevel



Heretical



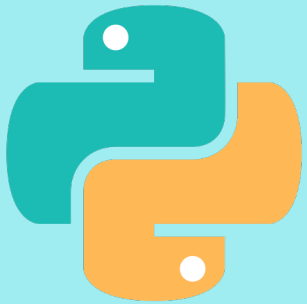
Multiple



```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

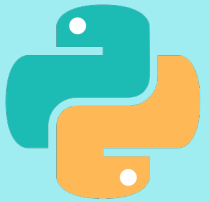
یکی از مفاهیم بنیادی برنامه نویسی شی گرا، کپسوله سازی است.
کپسوله سازی از بسته‌بندی داده‌ها و متدهایی که در داخل یک واحد کار می‌کنند، به وجود آمده است.
این کار کمک می‌کند تا دسترسی مستقیم به متغیرها و متدها محدود شده و از تغییر تصادفی داده‌ها جلوگیری شود.
در این روش، تغییر متغیر یک شی فقط با متد همان 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.

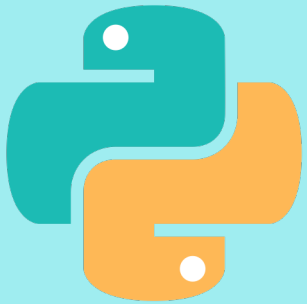
اعضای محافظت شده (Protected members)

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.

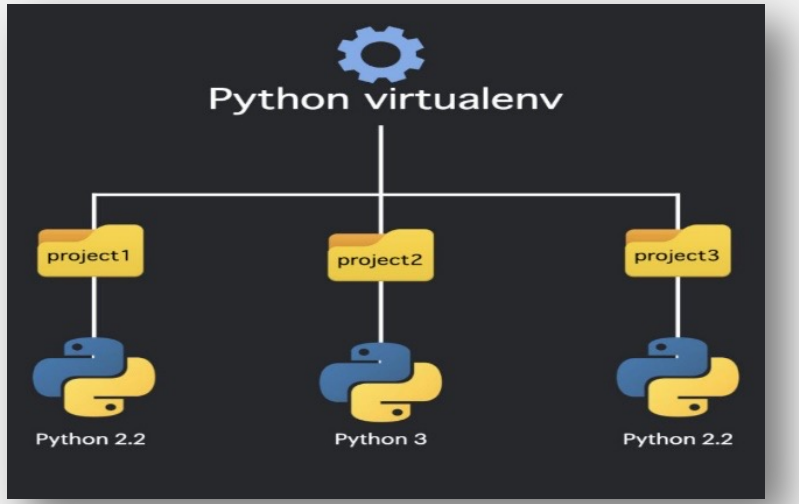




Polymorphism

```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

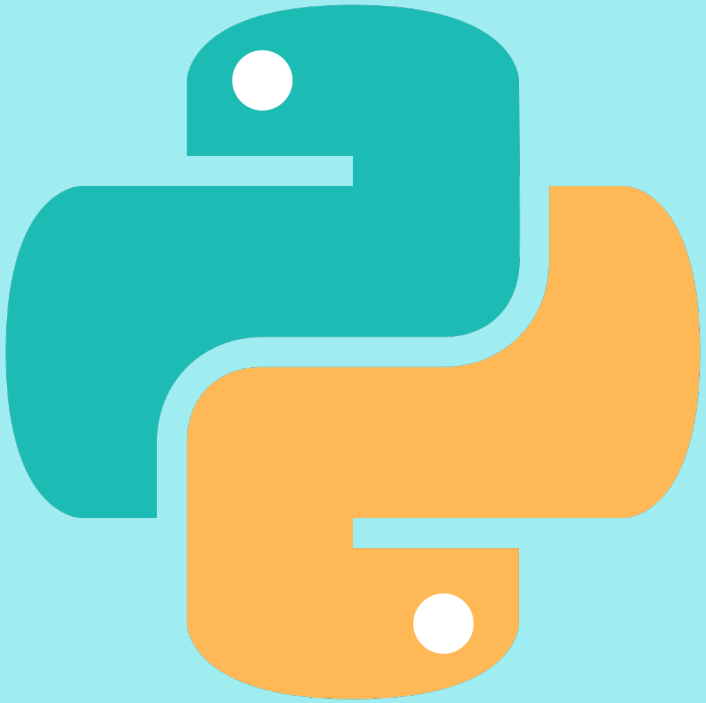
5- using virtual environment is vscode: code .

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

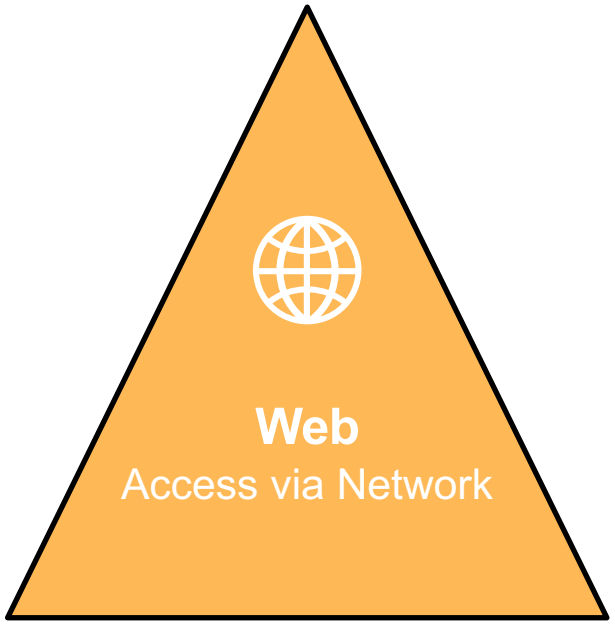
9- to deactivate : c:\prj_1\scripts\deactivate.bat



رابط کاربری

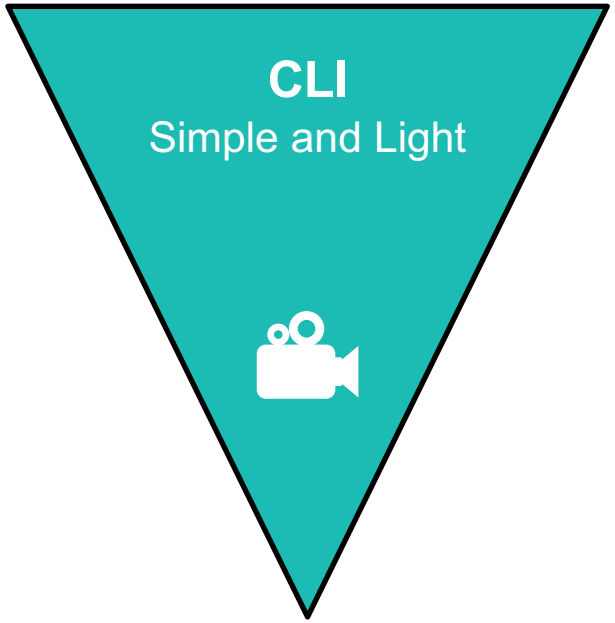
User Interface

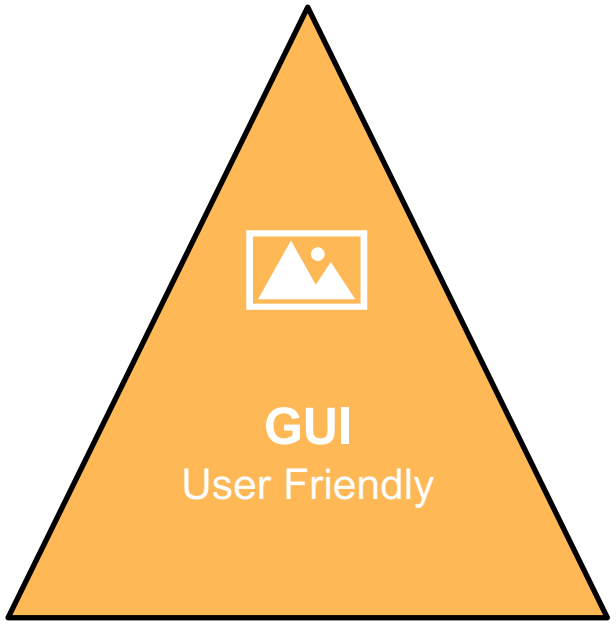
Software Layout



Too many library and option is available

No Library need





Too many library available

GUIs

Package	Target	Latest version	Latest update
PyQt	Qt	6.3.1	2022-06-17
TkInter	Tk	Included in Python standard library	
Kivy[MD]	Multi	2.1.0	2022-03-06
PySimpleGUI	Multi	5.0.2	2024-02-14
Flet	Multi	0.21.2	2024-03-18

https://wiki.python.org/moin/GuiProgramming

TKinter

- Standard Library

- Code Sample:

```
from tkinter import *
mainform=Tk()
mainloop()
```

- Widget:

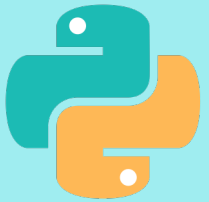
  Button, CheckButton, Message, Text, Menu,..

- Widget Properties:

  bg, bd, font, Dimension, color,

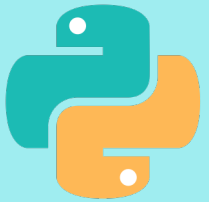
- Widget Methods:

  Status(), maxsize(),



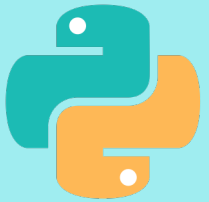
Tkinter Widget

کنترل	شرح
Button	برای نمایش دکمه‌ها در برنامه گرافیکی استفاده می‌شود.
Canvas	برای کشیدن شکل‌های مختلف مانند: خطوط، دایره، چند ضلعی و مستطیل در برنامه استفاده می‌شود.
Checkbutton	برای نمایش چند انتخاب به صورت check box است. کاربر می‌تواند چند گزینه را همزمان انتخاب کند.
Entry	نمایش جعبه یک خطی برای دریافت متن از کاربر
Frame	به عنوان یک نگهدارنده و مرتب کننده دیگر کنترل‌ها استفاده می‌شود.
Label	برای نمایش عنوان تک خطی برای دیگر کنترل‌ها استفاده می‌شود. همچنین می‌تواند تصویر را نیز نمایش دهد.
Listbox	لیستی از گزینه‌ها را برای کاربر مهیا می‌کند.
Menubutton	برای نمایش منو در برنامه استفاده می‌شود.
Menu	دستورات مختلفی را برای کاربر فراهم می‌کند. این دستورات درون Menubutton ها قرار گرفته‌اند.
Message	برای نمایش فیلدهای متنی چند خطی، به منظور دریافت ورودی از کاربر استفاده می‌شوند.
Radiobutton	تعدادی گزینه را به صورت دکمه‌های رادیویی نمایش می‌دهد. کاربر تنها یک گزینه را می‌تواند انتخاب کند.



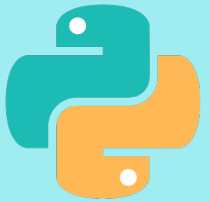
Tkinter Widget Cont.

Scale	برای نمایش کنترل slider استفاده می‌شود.
Scrollbar	برای اضافه کردن امکان اسکرول به کنترل‌ها مختلف از قبیل listbox ها استفاده می‌شود.
Text	برای نمایش متن چند خطی استفاده می‌شود.
Toplevel	امکان قرار گرفتن و نمایش چند پنجره مجزا را در یک برنامه فراهم می‌کند.
Spinbox	نوع دیگری از entry است که امکان انتخاب از میان چند گزینه ثابت را فراهم می‌کند.
PanedWindow	به عنوان نگهدارنده کنترل‌ها دیگر استفاده می‌شود و می‌تواند آنها را به صورت افقی یا عمودی مرتب کند.
LabelFrame	یک نگهدارنده ساده است که هدف اولیه آن به عنوان جدا کننده یا دربرگیرنده کنترل‌ها دیگر می‌باشد.
tkMessageBox	برای نمایش پنجره‌های پیام به کاربر استفاده می‌شود.



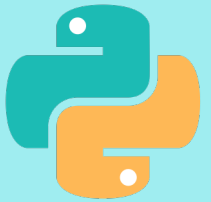
Start with TKinter

```
from tkinter import *
mainform=Tk()
Mainform.title("Python GUI")
mainform.resizable(0, 0)
mainform.geometry("500x500+100+100")
Mainform.configure(bg=“black”)
```



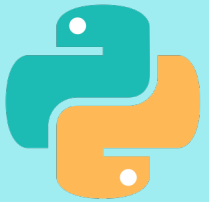
MainForm Properties

تنظیمات	شرح
bg	رنگ پس زمینه پنجره را مشخص می‌کند.
bd	اندازه خط کناری را بر اساس پیکسل مشخص می‌کند. پیش فرض 0 است.
cursor	شکل نشانگر ماوس را زمانی که درون پنجره باشد، مشخص می‌کند.
font	فونت پیش فرض برای متن‌هایی که درون این عنصر قرار می‌گیرند.
fg	رنگ استفاده شده برای متن و bitmap درون این عنصر را مشخص می‌کند.
height	ارتفاع پنجره را مشخص می‌کند.
relief	در حالت عادی این نوع پنجره دارای حاشیه سه بعدی در اطراف خود نیست. برای بدست آوردن حاشیه سایه دار، گزینه bd را بیشتر از صفر قرار داده و relief را به یکی از ثابت‌هایش مقداردهی کنید.
Width	عرض پنجره را مشخص می‌کند.



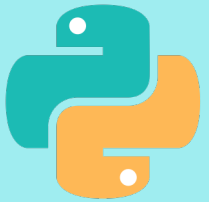
Text Widget

```
text1=Text(mainform,height=10,width=10)
text1.pack()
```



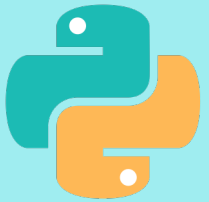
Entry Widget

```
entry1 = Entry(mainform, width=10, font=("", 20, ""))
entry1.config(bg="#96eef2", fg="black")
entry1.pack()
```



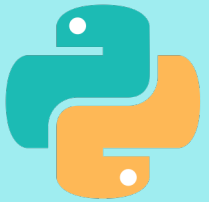
Label Widget

```
label1 = Label(mainform, text="FirstName", width=10,fg="yellow", font=(""
, 20, ""))
label1.pack()
```



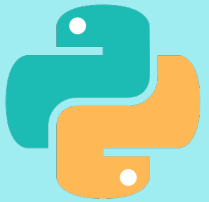
Checkbox Widget

```
var1=0
check3=Checkbutton(mainform,text="Enabled",variable=var1)
check3.select()
check3.pack()
```



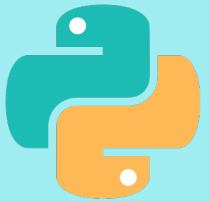
RadioButton Widget

```
v = ""
R1 = Radiobutton(mainform, text="Python", variable=v, value=1)
R1.pack()
R2 = Radiobutton(mainform, text="JAVA", variable=v, value=2)
R2.pack()
```



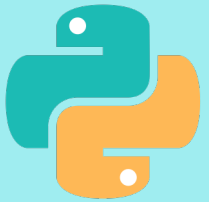
Button Widget

```
b1 = Button(mainform, text="Exit", width=10, font=("", 20, ""),
command=mainform.destroy)
b1.pack()
```



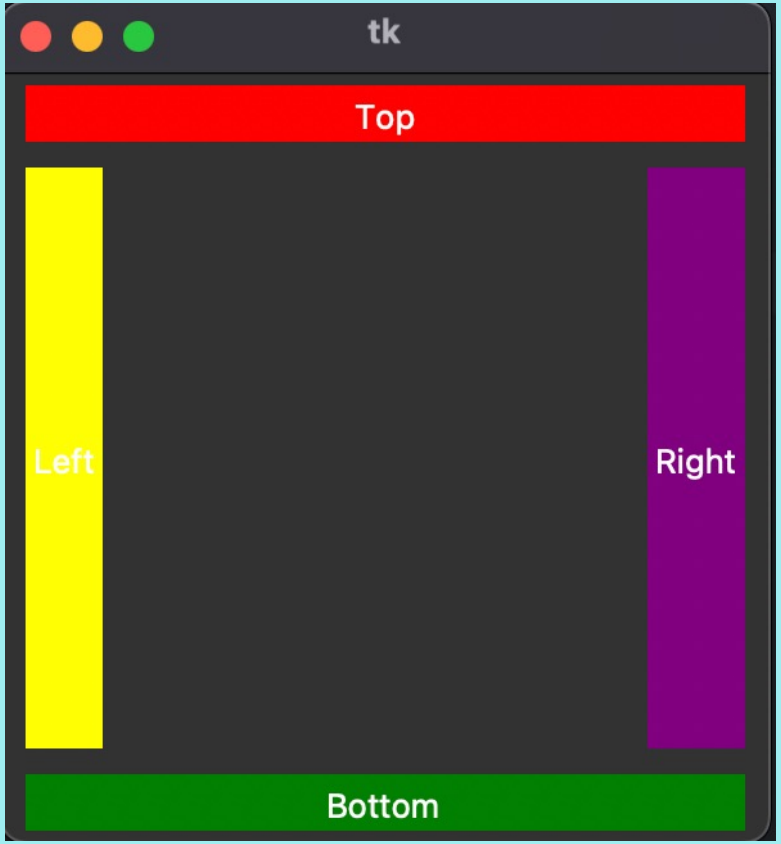
Widgets Design

Pack
Grid
Place



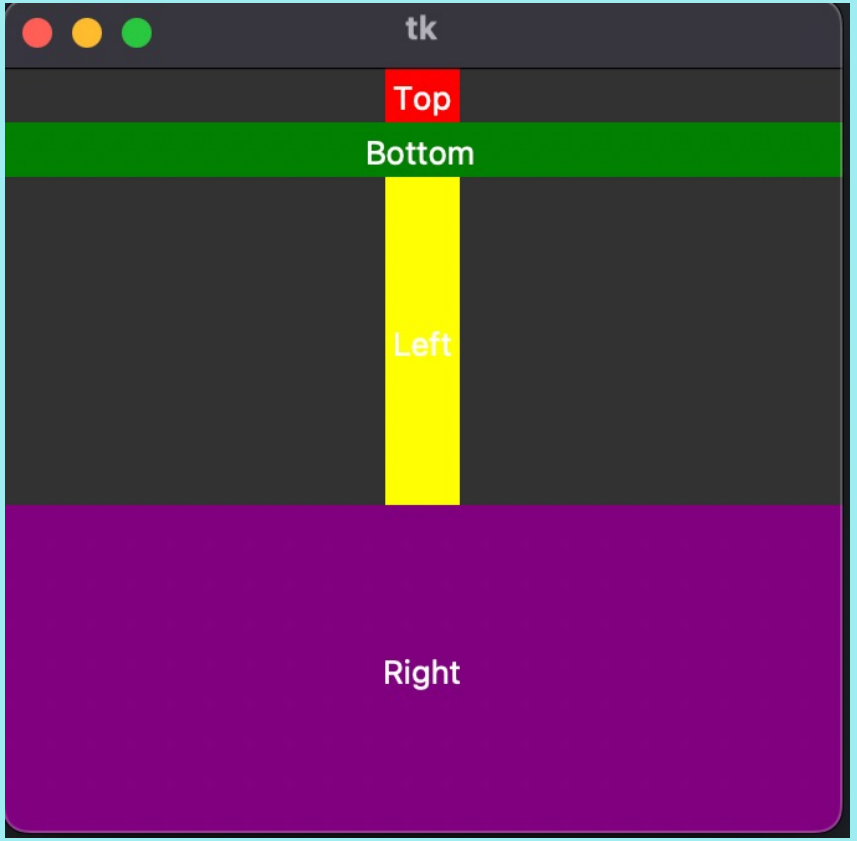
Widgets Design - Pack

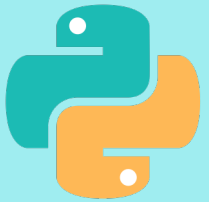
```
import tkinter as tk

root = tk.Tk()
root.geometry("300x300")
label1 = tk.Label(root, text="Top",bg="red")
label1.pack(side=tk.TOP, fill=tk.X, padx=10, pady=5)

label4 = tk.Label(root, text="Bottom",bg="green")
label4.pack(side=tk.BOTTOM, fill=tk.X, padx=10, pady=5)

label2 = tk.Label(root, text="Left",bg="yellow")
label2.pack(side=tk.LEFT, fill=tk.Y, padx=10, pady=5)

label3 = tk.Label(root, text="Right",bg="purple")
label3.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=5)

root.mainloop()
```



Widgets Design - Pack

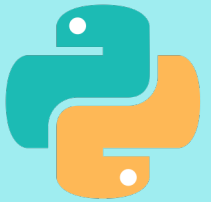
```
import tkinter as tk

root = tk.Tk()
root.geometry("300x300")

label1 = tk.Label(root, text="Top",bg="red")
label1.pack(fill=tk.NONE)

label4 = tk.Label(root, text="Bottom",bg="green")
label4.pack(fill=tk.X)

label2 = tk.Label(root, text="Left",bg="yellow")
label2.pack(fill=tk.Y, expand=1)

label3 = tk.Label(root, text="Right",bg="purple")
label3.pack(fill=tk.BOTH, expand=1)

root.mainloop()
```





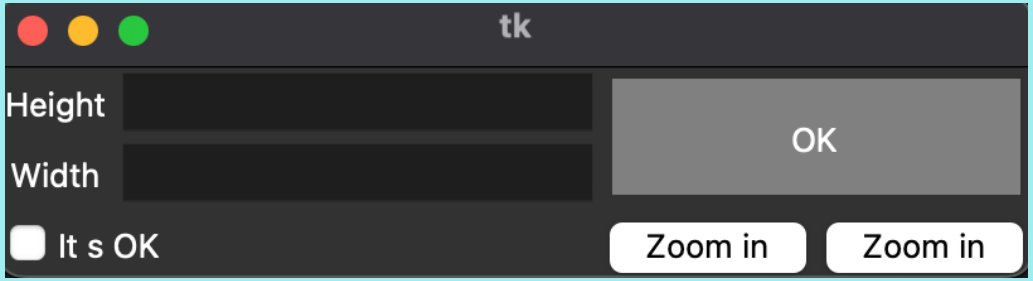
Widgets Design - Grid

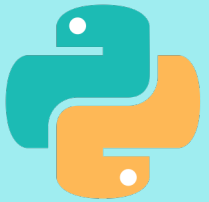
خاصیت	توضیح
column	مشخص می‌کند که فرم به چند ستون برای قرار گرفتن عناصر تقسیم شود.
row	مشخص می‌کند که فرم به چند سطر برای قرار گرفتن عناصر تقسیم شود.
padx	فاصله بین عناصر در محور افقی را مشخص می‌کند.
pady	فاصله بین عناصر در محور عمودی را مشخص می‌کند.
ipadx	فاصله بین سلول و عنصری که در داخل سلول قرار دارد، در محور افقی را مشخص می‌کند.
ipady	فاصله بین سلول و عنصری که در داخل سلول قرار دارد، در محور عمودی را مشخص می‌کند.
sticky	اگر اندازه سلول از اندازه عنصری که در داخل آن قرار گرفته است، بزرگ‌تر باشد، این خاصیت مشخص می‌کند که عنصر در چه جای سلول قرار بگیرد: شمال، شمال غربی و ... . مقادیری هم که این خاصیت دریافت می‌کند عبارتند از : SW، SE، NW، NE، W، S، E، N و center



Widgets Design - Grid

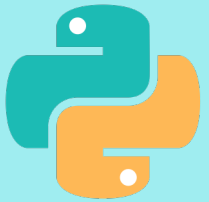
```
from tkinter import *
root = Tk()
label1=Label(text="Height")
label1.grid(row=0, column=0)
label2=Label(text="Width")
label2.grid(row=1, column=0)
entry1=Entry()
entry1.grid(row=0, column=1)
entry2=Entry()
entry2.grid(row=1, column=1)
checkbox1=Checkbutton(text="It s OK")
checkbox1.grid(columnspan=2, row=2, column=0, sticky=W)
button1=Button(text="Zoom in")
button1.grid(row=2, column=2)
button2 = Button(text="Zoom in")
button2.grid(row=2, column=3)
label3 = Label(text="OK", bg="gray")
label3.grid(row=0, column=2, columnspan=2, rowspan=2, sti
cky=W+E+N+S, padx=5,pady=5)
root.mainloop()
```





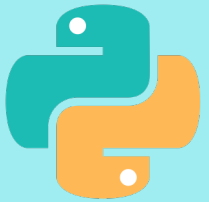
Events And Event Control

– رویدادها (event) رفتارها یا اتفاقاتی هستند مه هنگام اجرای برنامه به وقوع می پیوندند مانند: فشردن کلیدها، حرکت ما وس، کلیک ماوس و...
– کنترل رویداد فرایند نظارت بر وقوع یک رویداد مشخص می باشد با استفاده از تابع bind() از کنترل کننده رویداد برای اضافه کردن یک قابلیت و پاسخ به کاربر استفاده می کند
– بدون کنترل کننده رویداد فرم ها و رابط گرافیکی تا حد زیادی بدون استفاده هستند
– با استفاده از mainloop() برنامه منتظر یک رویداد می ماند
– تابع bind() یک تابع را به یک رویداد وصل می کند مثلا کلیک کردن ربط می دهد



Events And Event Control

```
from tkinter import *

window = Tk()
window.geometry('200x200')


def showlocation(event):
          print(event.x, event.y)


window.bind('<1>', showlocation)

mainloop()
```



Events And Event Control

خاصیت	کاربرد در رویدادهای	توضیح
.x,y	ماوس	موقعیت فعلی نشانگر ماوس را با توجه به پنجره برنامه بر حسب پیکسل نشان می‌دهد.
.x_root,y_root	ماوس	موقعیت فعلی نشانگر ماوس را با توجه به گوشه بالا و سمت چپ مانیتور بر حسب پیکسل نشان می‌دهد.
.num	ماوس	نشان می‌دهد که کدام دکمه ماوس فشرده شده است.
.delta	ماوس	مقدار 120 را برای یک یک بار چرخش دکمه وسط ماوس رو به بالا و مقدار 120- را برای یک بار چرخش رو به پایین دکمه وسط ماوس بر می گرداند و همراه با رویداد MouseWheel به کار می رود.
.time	ماوس	برای تشخیص فاصله بین دو بار کلیک کردن ماوس بر حسب میلی ثانیه به کار می رود.
.char	کیبورد	متن دکمه فشرده شده توسط کاربر را نشان می‌دهد. مثلاً اگر کاربر دکمه a را بفشارد، رشته a چاپ می‌شود. این خاصیت فقط دکمه‌های a-z، A-Z، اعداد 0-9 و نمادهای ریاضی را شامل می‌شود.
.keysym	کیبورد	متن دکمه فشرده شده توسط کاربر را نشان می‌دهد. مثلاً اگر کاربر دکمه a را بفشارد، رشته a چاپ می‌شود. این خاصیت تمام دکمه‌های کیبورد را شامل می‌شود.
.keysym_num	کیبورد	عدد متن دکمه فشرده شده توسط کاربر را نشان می‌دهد. مثلاً اگر کاربر دکمه a را بفشارد، عدد 97 چاپ می‌شود. این خاصیت تمام دکمه‌های کیبورد را شامل می‌شود.
.keycode	کیبورد	کد اسکی دکمه فشرده شده کیبورد را بر می‌گرداند.
.width,height	configure	اندازه کنترل را بر حسب پیکسل بر می‌گرداند.