RYKA CBT

پدرام شاه صفی
pd.Shahsafi@gmail.com

python

به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

تابستان ۱۳۹۴

برنامه نویسی به سبک پایتون

برنامه نویسی در سطح اول

بخش اول

کلاس

فصل هشتم

فصل هفتم

فصل ششم

فصل پنجم

فصل چهارم

فصل سوم

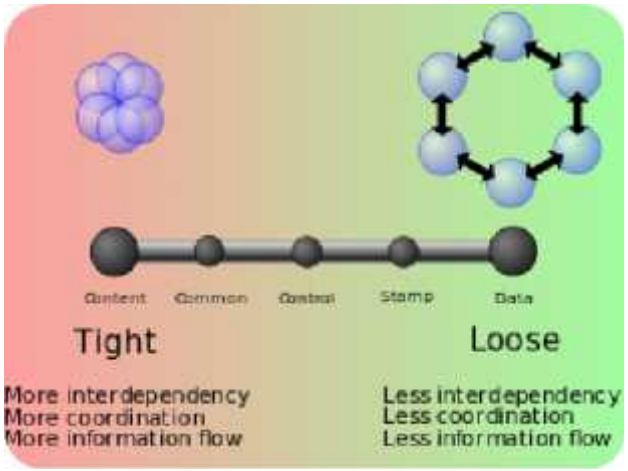
فصل دوم

فصل اول

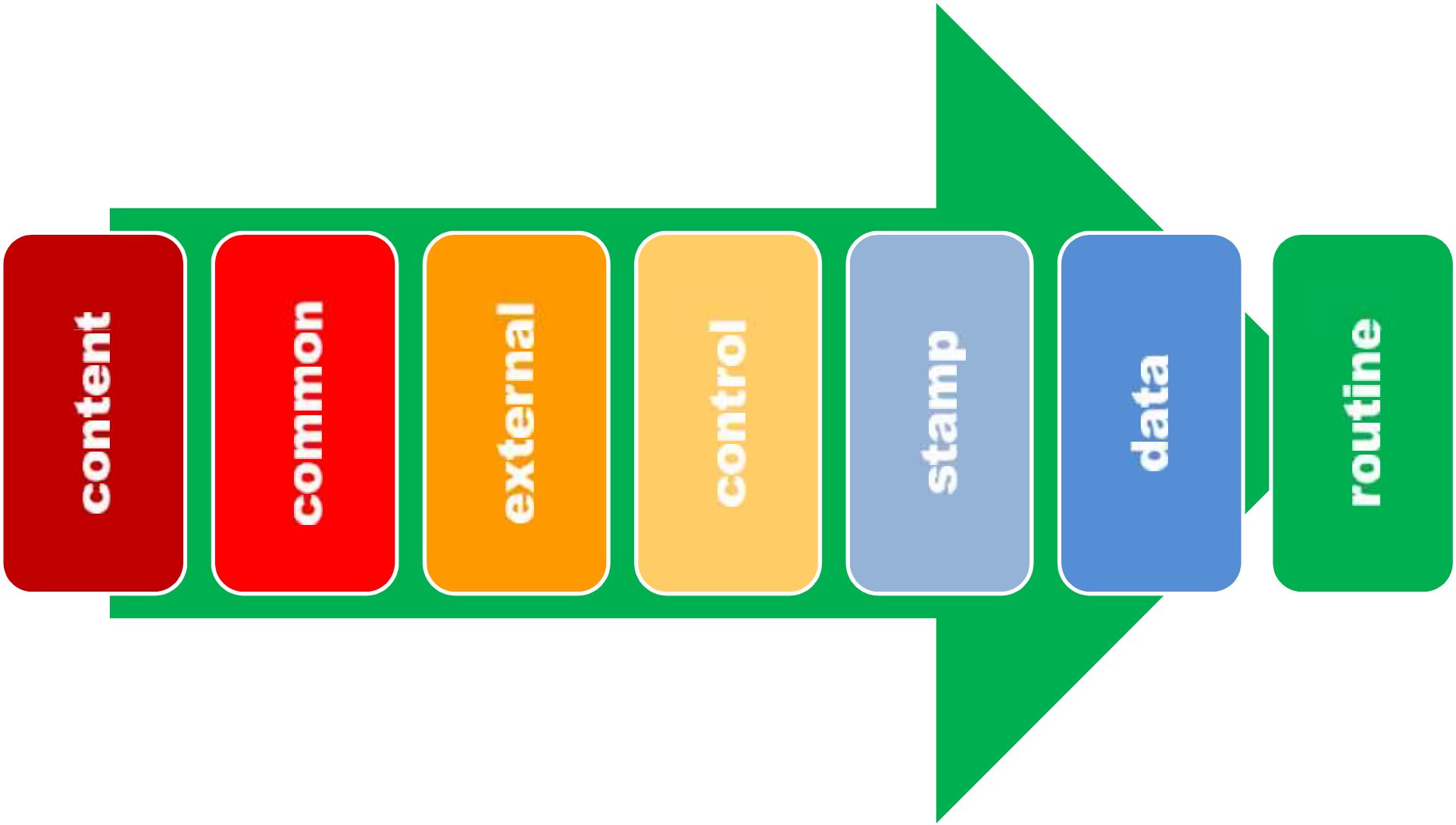
طراحی

- **Cohesion**
  - پیوستگی بین ماژول ها را گویند..
  - هر پیمانه یک single task انجام دهد.
  - برای انجام ان کمترین ارتباط را با سایرین داشته باشد.
  - هر چه Cohesion بالاتر باشد بهتر است اما سطح متوسط ان هم قابل قبول است.
  - اثر بخشی در سطح متوسط هم به قدر کافی خوب است.

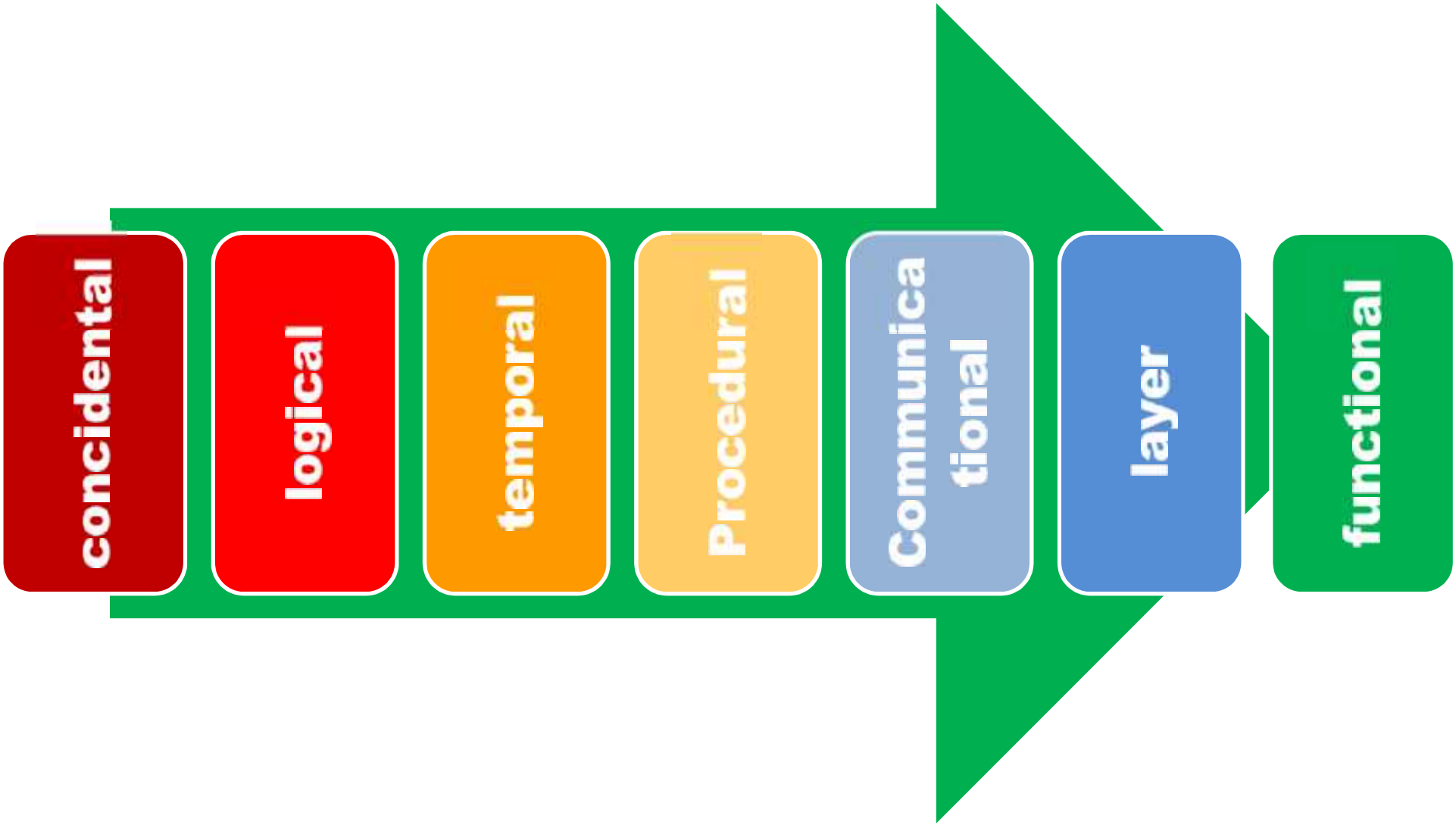
- **Coupling**
  - وابستکی بین ماژول ها را گویند.
  - هر چه ارتباط به دیگران بیشتر باشد Coupling بالاتر است.



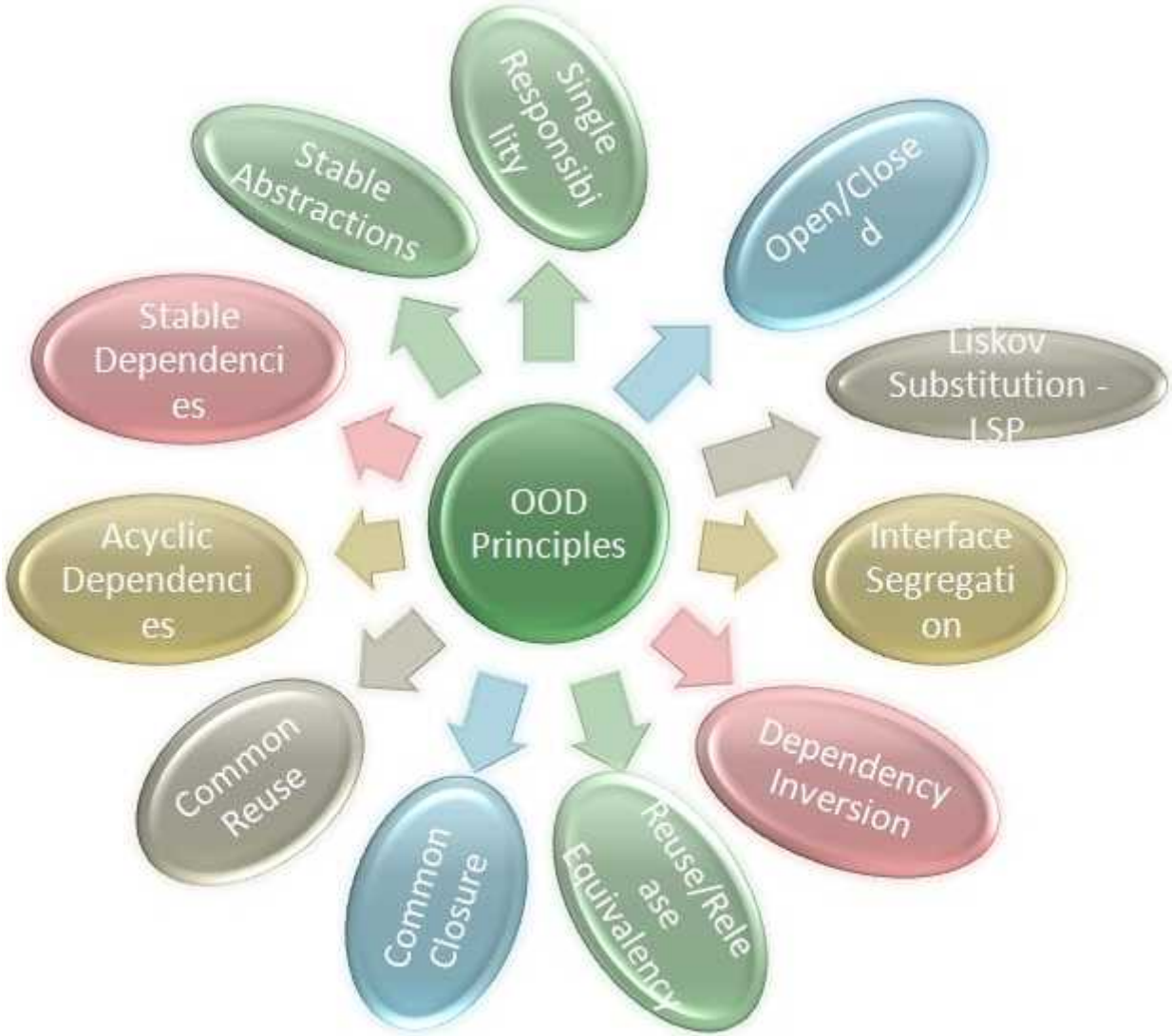
وابستگی



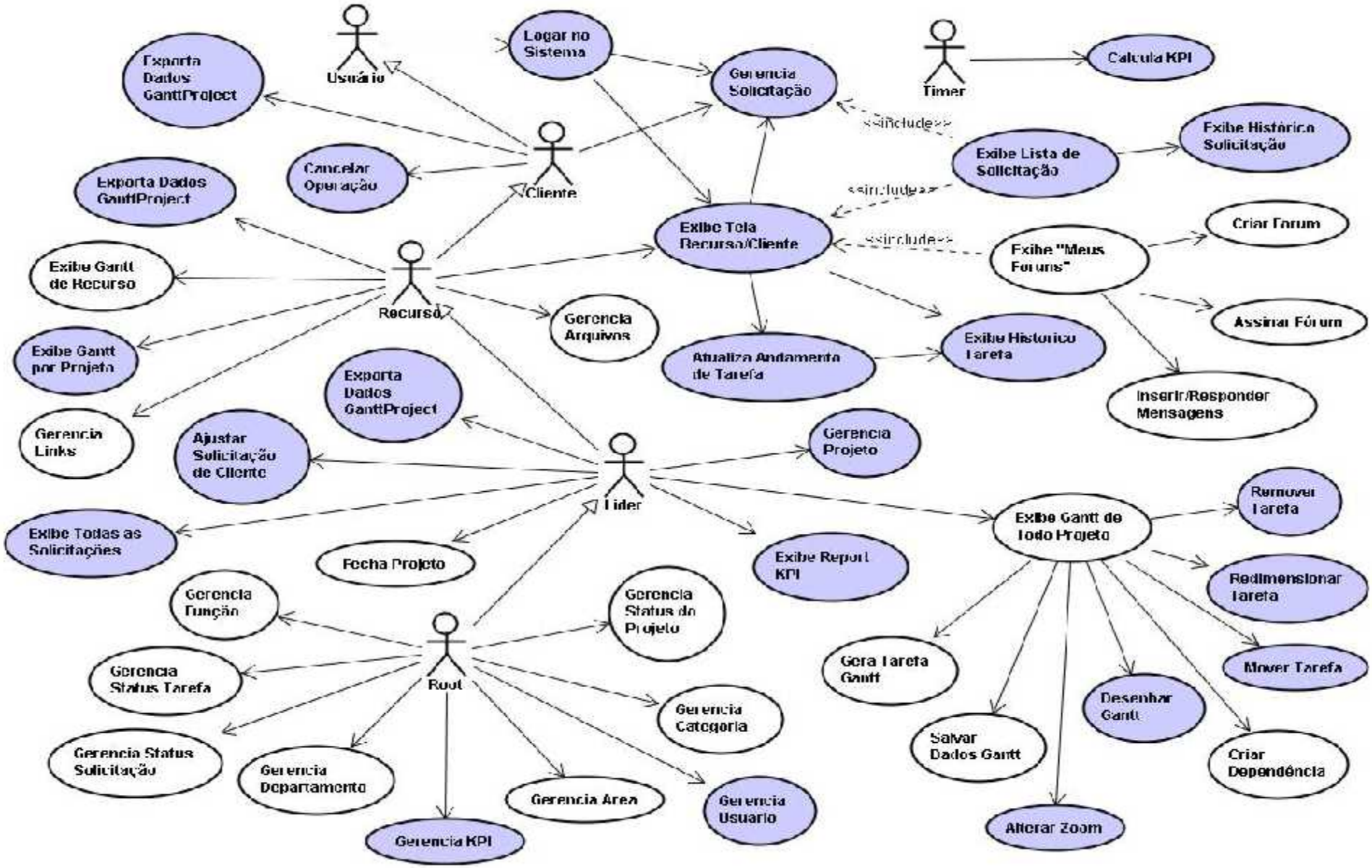
پیوستگی


concidental
logical
temporal
Procedural
Communica tional
layer
functional

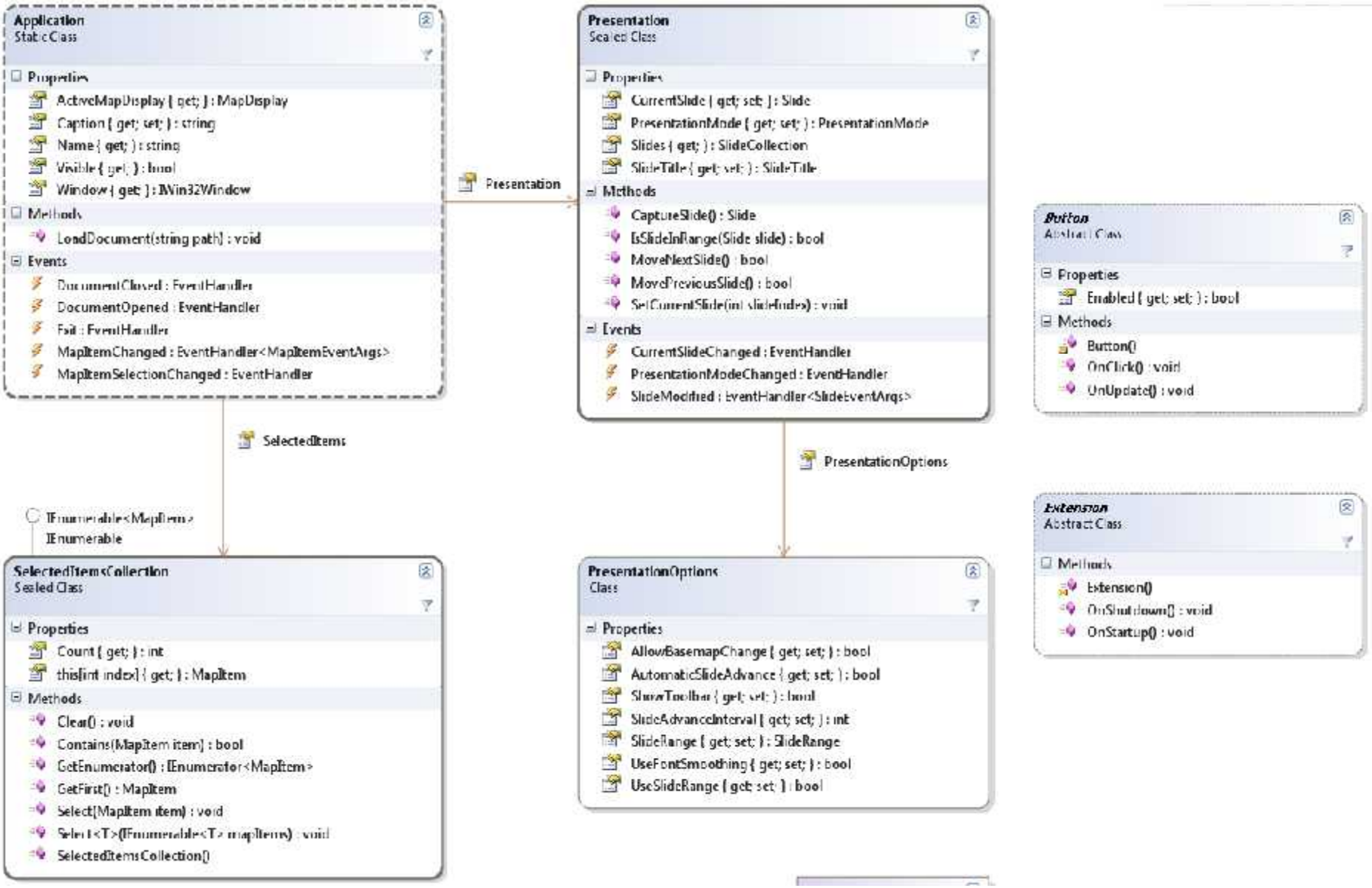
طراحی شی گرا


OOD Principles
Single Responsibility
Open/Closed
Liskov Substitution - LSP
Interface Segregation
Dependency Inversion
Reuse/Release Equivalency
Common Closure
Common Reuse
Acyclic Dependencies
Stable Dependencies
Stable Abstractions

نمودار کاربرد



نمودار کلاس

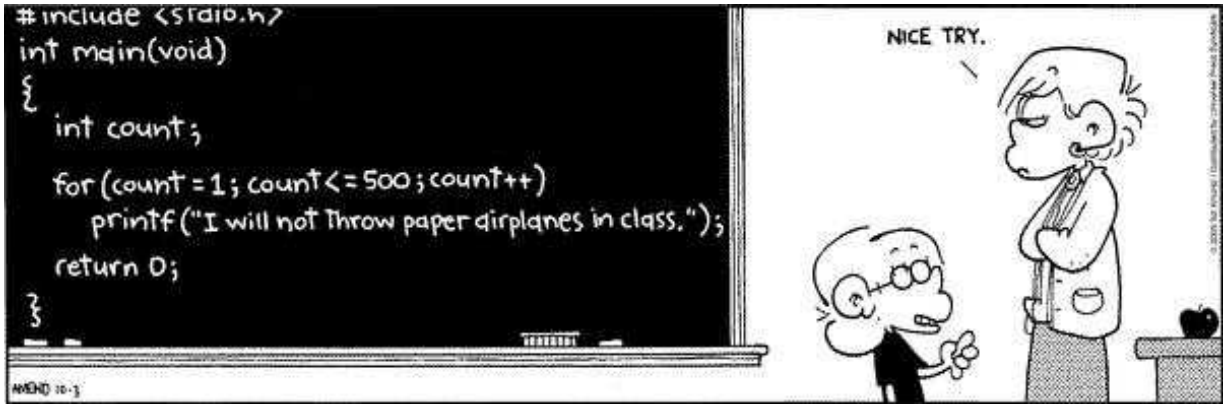

Application
Static Class
Properties
ActiveMapDisplay { get; } : MapDisplay
Caption { get; set; } : string
Name { get; } : string
Visible { get; } : bool
Window { get; } : IWin32Window
Methods
LoadDocument(string path) : void
Events
DocumentClosed : EventHandler
DocumentOpened : EventHandler
Exit : EventHandler
MapItemChanged : EventHandler<MapItemEventArgs>
MapItemSelectionChanged : EventHandler
Presentation
Sealed Class
Properties
CurrentSlide { get; set; } : Slide
PresentationMode { get; set; } : PresentationMode
Slides { get; } : SlideCollection
SlideTitle { get; set; } : SlideTitle
Methods
CaptureSlide() : Slide
IsSlideInRange(Slide slide) : bool
MoveNextSlide() : bool
MovePreviousSlide() : bool
SetCurrentSlide(int slideIndex) : void
Events
CurrentSlideChanged : EventHandler
PresentationModeChanged : EventHandler
SlideModified : EventHandler<SlideEventArgs>
Presentation
SelectedItems
PresentationOptions
IEnumerable<MapItem>
IEnumerable
SelectedItemCollection
Sealed Class
Properties
Count { get; } : int
this[int index] { get; } : MapItem
Methods
Clear() : void
Contains(MapItem item) : bool
GetEnumerator() : IEnumerator<MapItem>
GetFirst() : MapItem
Select(MapItem item) : void
Select<T>(IEnumerable<T> mapItems) : void
SelectedItemsCollection()
PresentationOptions
Class
Properties
AllowBasemapChange { get; set; } : bool
AutomaticSlideAdvance { get; set; } : bool
ShowToolbar { get; set; } : bool
SlideAdvanceInterval { get; set; } : int
SlideRange { get; set; } : SlideRange
UseFontSmoothing { get; set; } : bool
UseSlideRange { get; set; } : bool
Button
Abstract Class
Properties
Enabled { get; set; } : bool
Methods
Button()
OnClick() : void
OnUpdate() : void
Extension
Abstract Class
Methods
Extension()
OnShutdown() : void
OnStartup() : void

طراحی شی گرا، چرا ؟

- ملموس و قابل درک چون از دنیای واقعی الهام گرفته.
- بصورت ذاتی از اصول طراحی حمایت میکند.
- coupling پایین، cohesion بالا، تجرید و پیمانه سازی و پنهان سازی.
- قابلیت استفاده مجدد.
- کاهش هزینه و زمان .
- افزایش اطمینان.

کلاس

- کلاس یعنی دسته بندی اشیای مشابه.
- اشیای یک کلاس میتوانند در مقادیر با هم متفاوت باشند.
- کلاس توصیف کننده ی اشیای خود است اما به صرف تعریف کلاس شی تشکیل نمیشود .



شی

- ویژگی های یک شی واقعی در دنیای واقعی :
  - State
    - active
      - خصوصیاتی که تغییر میکند.
    - passive
      - خصوصیاتی که ثابت اند.
  - Behaviour
    - ارتباط بین اشیا که معمولا موجب تغییر در State ان شی میشود.
  - Identity
    - خصوصیاتی که شی را از سایرین متمایز میکند.

مفاهیم کلاس ها

- Attribute:
  - state و Identity یک شی مربوط به مقادیر Attribute ها میشود.
- Method:
  - رفتاریی شی انجام میدهد را مشخص میکند.
- instance:
  - صرف تعریف کلاس شی ای تشکیل نمیشود و حافظه ای به ان اختصاص داده نمیشود.(در پایتون این قضیه برقرار نیست!) وقتی از کلاس شی می سازیم این شی یک فضا را اشغال میکند. شی عینیت یافته را instance گویند.
- Inheritance
  - Attribute و Method مشترک بین اشیا.

مفاهیم کلاس ها

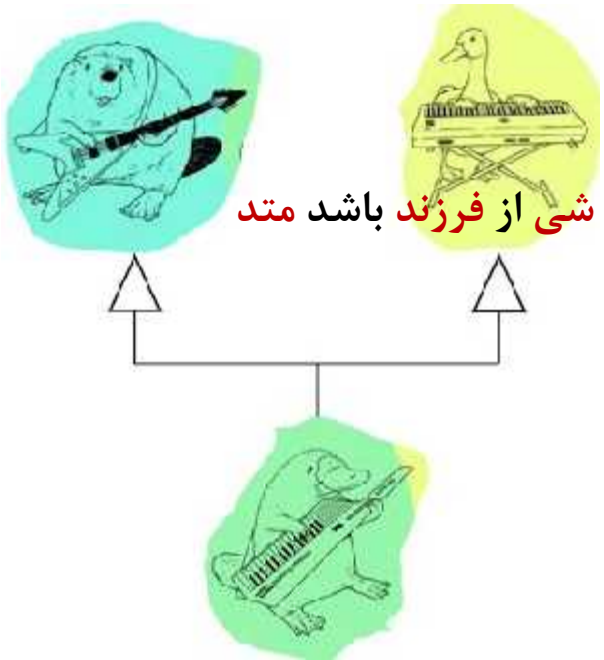
- Encapsulate:
  - بسته بندی atttibute ها و methodها که حاصل این بسته بندی کلاس است.
  - Infortmation hiding اشیا به داخل هم دسترسی ندارند بلکه از طریق متد های Interface با هم ارتباط میگیرند => کاهش coupling => افزایش استفاده مجدد reausability.

- public:
  - قابل دست رس برای همه.
- private:
  - قابل دست رس فقط برای اعضای همان کلاس.
- protect:
  - private + قابل دست رس برایفرزندان.

visibility / keyword	Containing Classes	Derived Classes	Containing Assembly	Anywhere outside the containing assembly
public	yes	yes	yes	yes
protected internal	yes	yes	yes	no
protected	yes	yes	no	no
private	yes	no	no	no
internal	yes	no	yes	no

مفاهیم کلاس ها

- Ploymorphism:
  - با یک متد چند رفتار متفاوت وابسته به شرایط خاص داریم.
  - در ابر کلاس به شکل abstract تعریف میکنیم در زیر کلاس ها به شکل دقیق تر تعریف میکنیم. دیتل : انجام کار درست در فرخوانی متدهای هم نام.
- Ovearloading:
  - چند متد هم نام با ورودی های متفاوت داریم با توجه به نوع ورودی ای ک میدیم خودش متد درست را صدا میزند
  - در پایتون نیست؛ چون نیازی به ان نیست !
- Ovearride:
  - کلاس متدی هم نام با پدر دارد اگر شی از پدر باشد متد پدر و اگر شی از فرزند باشد متد فرزند اجرا میشود



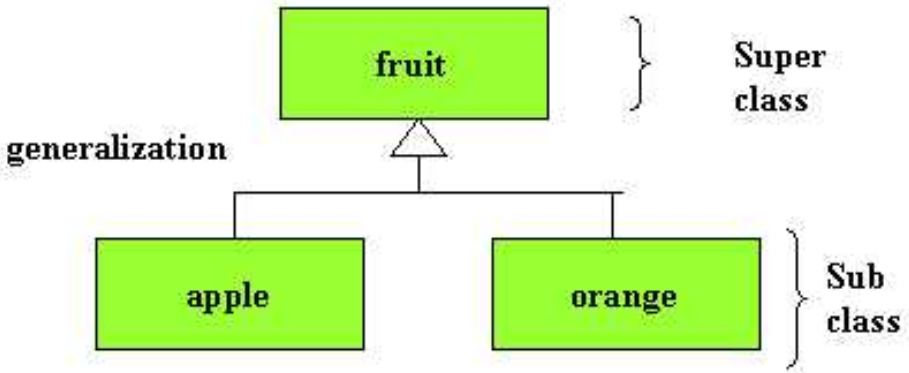
مفاهیم تابع Vs کلاس

Structured/Procedural	Object-Oriented
structure type definitions	classes
variables of structure types	instances (objects)
structure (component) vars	attributes
functions and procedures	methods
arguments (on function calls)	messages (on method calls)

ارث بری

- **Inheritance**
  - هر کلاس به طور جداگانه پیاده سازی شود و صفات مشترک در همه انها کپی شود.
  - یک کلاس از مقادیر مشترک ایجاد کنیم و بقیه ارث بری کنند.
  - ارث بری از دنیای واقعی الهام گرفته.
- انواع ارث بری
  - **Single inheritance**.
  - **Multiple inheritance** .
- **Subclass**
  - کلاس ارث برنده.
- **Superclass**
  - کلاس ارث برده شده.

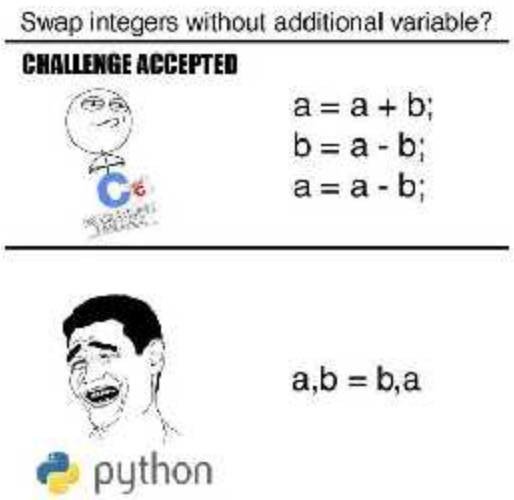
- Apple is-a fruit
- Orange is-a fruit



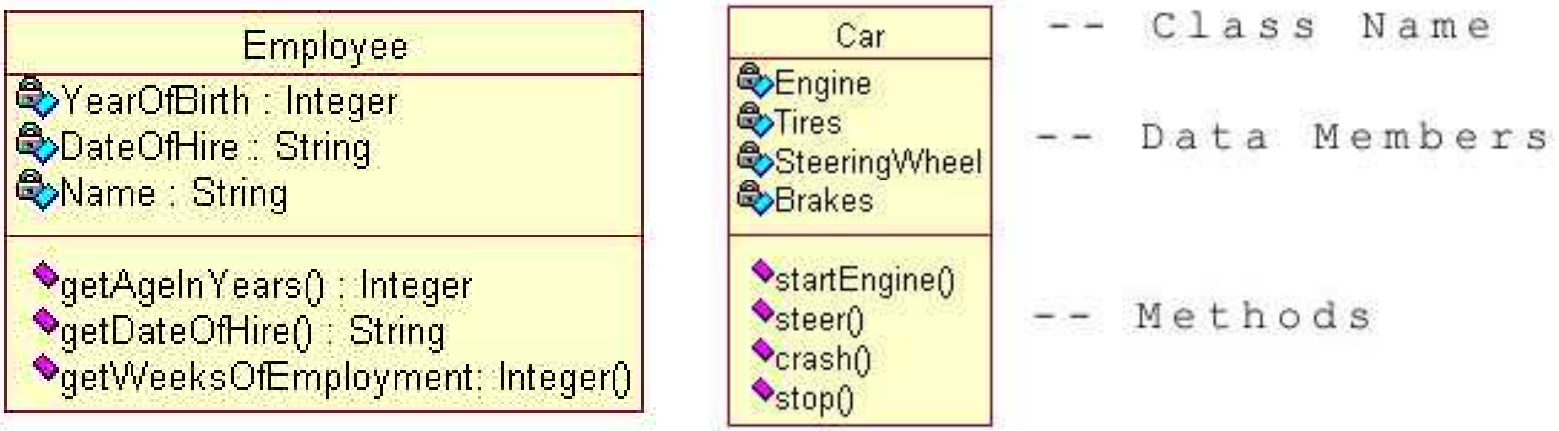
ارث بری چندگانه

- **Multiple inheritance**
- طراح باید سعی کند از ارث بری چندگانه **اجتناب** کند . اما **گاهی راه دیگری نیست**!

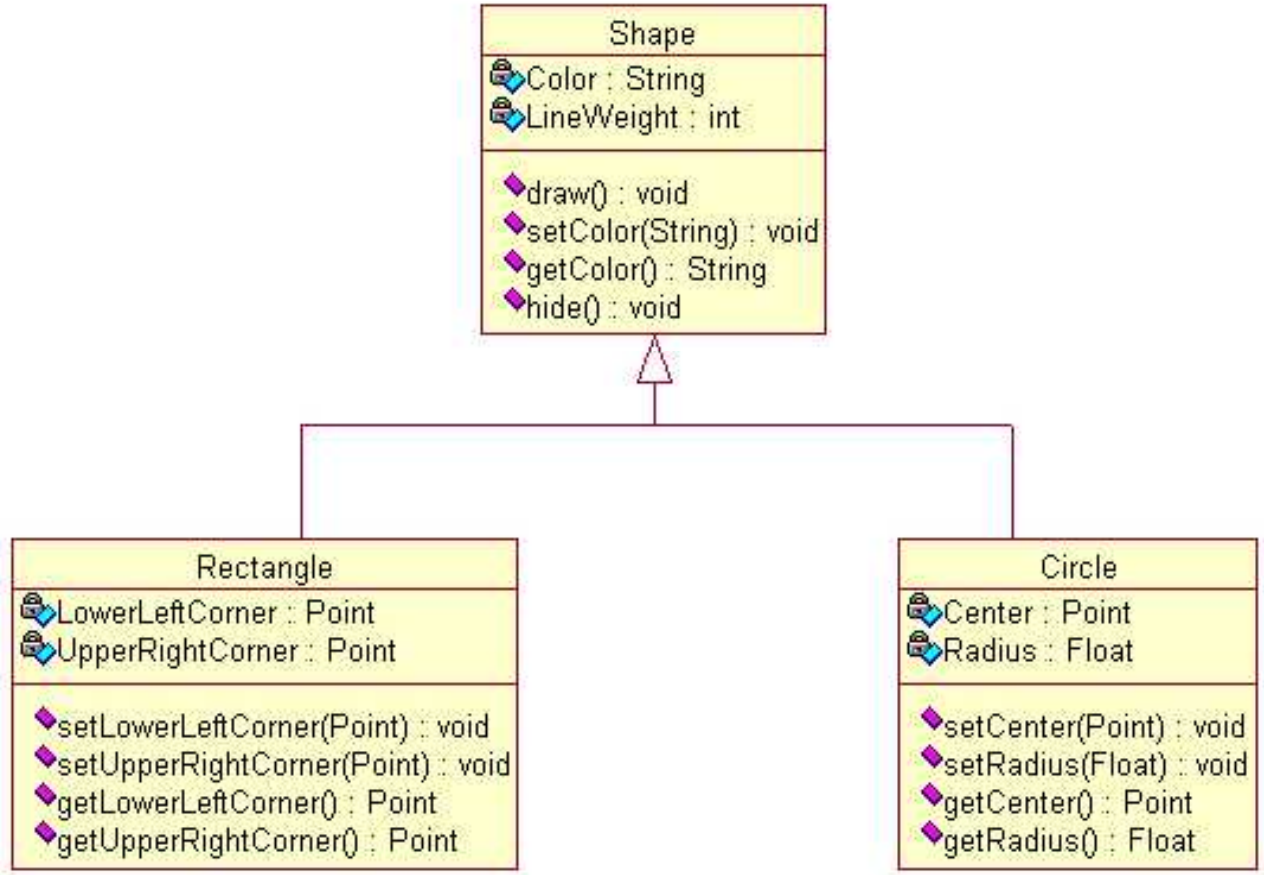
- چرا؟
- **name collosion** متد ها **هم نام** از کدام کلاس اجرا شوند.
- **Repeat inheritance** ارث بری از **دو** مسیر.
- (این دو مشکل درپایتون **حل شده**).
- تغییر و نگهداری ساختار **پیچیده** است.
- در زبان های **C#** و **جاوا** وراثت چندگانه **نداریم** اما در **پایتون داریم.**
- همیشه **نمی توان** کلاسی که که ارث بری **چندگانه دارد** را به ارث بری **یگانه** تبدیل کرد.



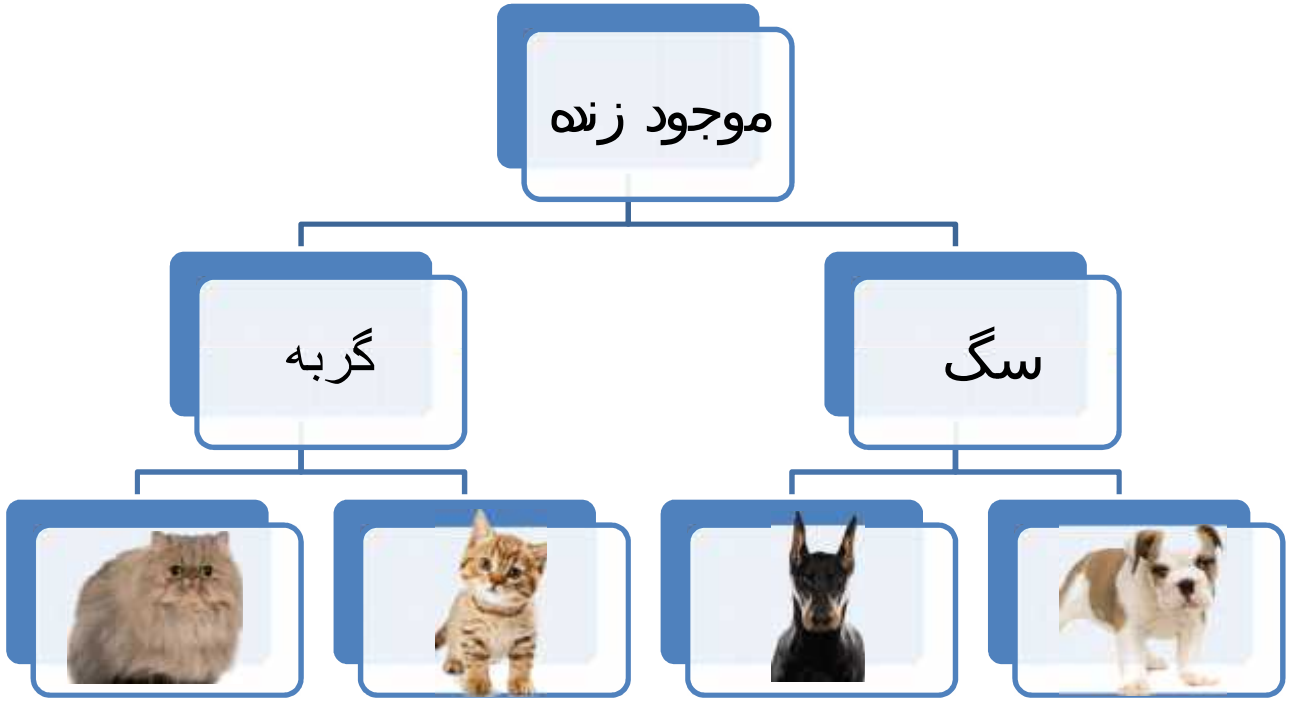
نمودار کلاس


Employee
YearOfBirth : Integer
DateOfHire : String
Name : String
getAgeInYears() : Integer
getDateOfHire() : String
getWeeksOfEmployment: Integer()
Car
Engine
Tires
SteeringWheel
Brakes
startEngine()
steer()
crash()
stop()
-- Class Name
-- Data Members
-- Methods

نمودار ارث بری



مثلا



پیاده سازی

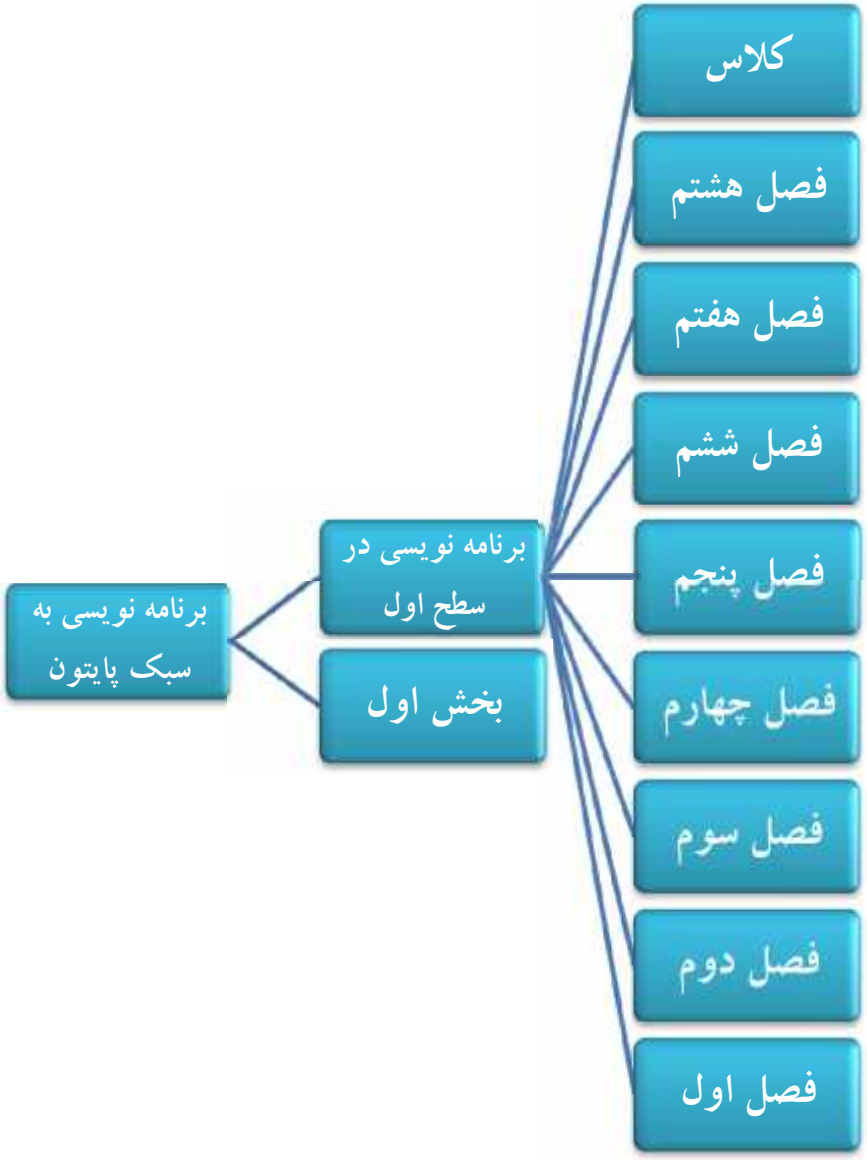
```
abstract class LivingCreature
        private weight
        private age
        private ...

        public breathe()
                breathing

        public abstract move()
                #no implementation

public class doberman(LivingCreature)
        attribute of doberman
        method    of doberman

        public override move()
                move_like_doberman
```

به نام پروردگار دانایی

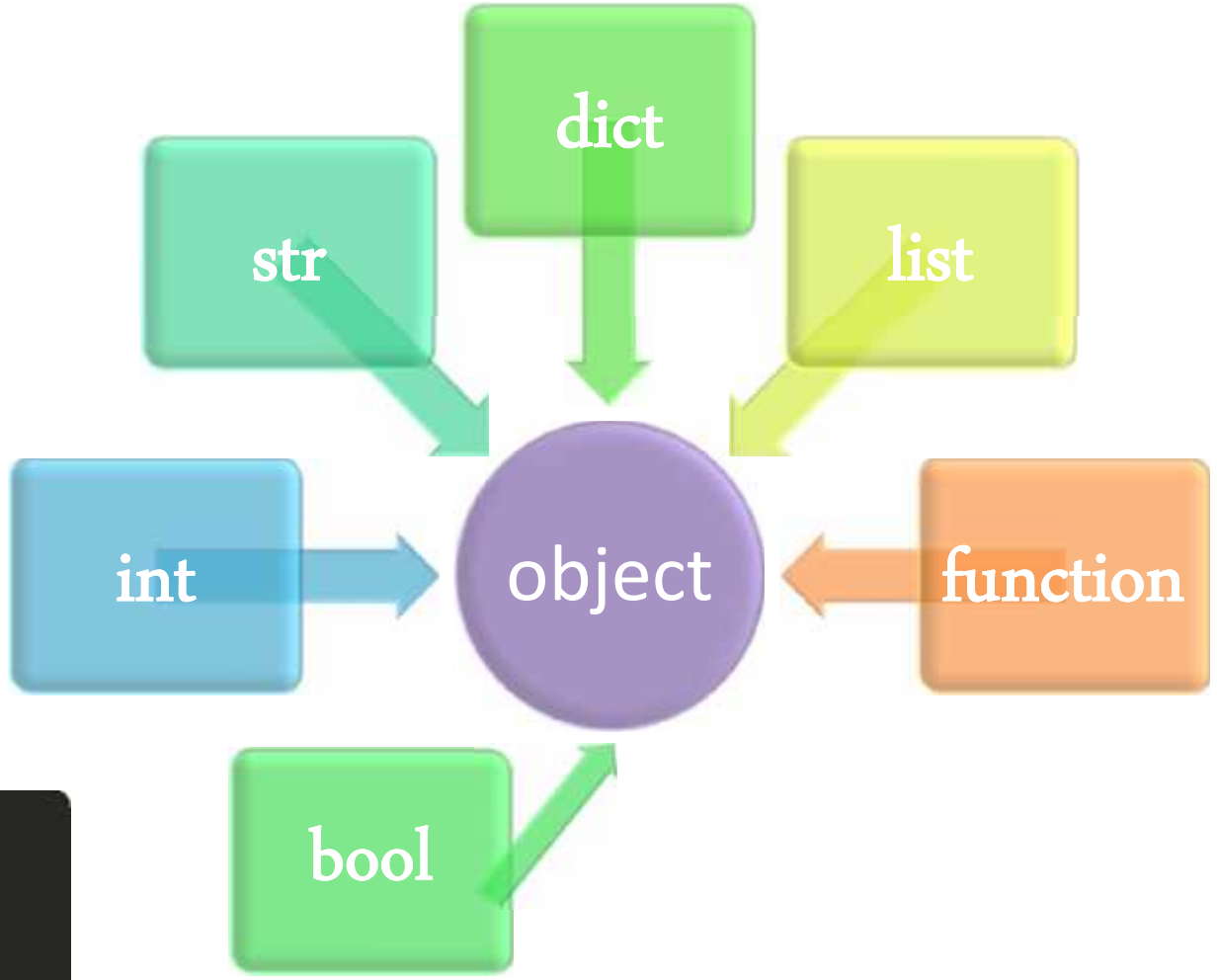
برنامه نویسی به سبک پایتون

پدرام شاه صفی

تابستان ۱۳۹۴

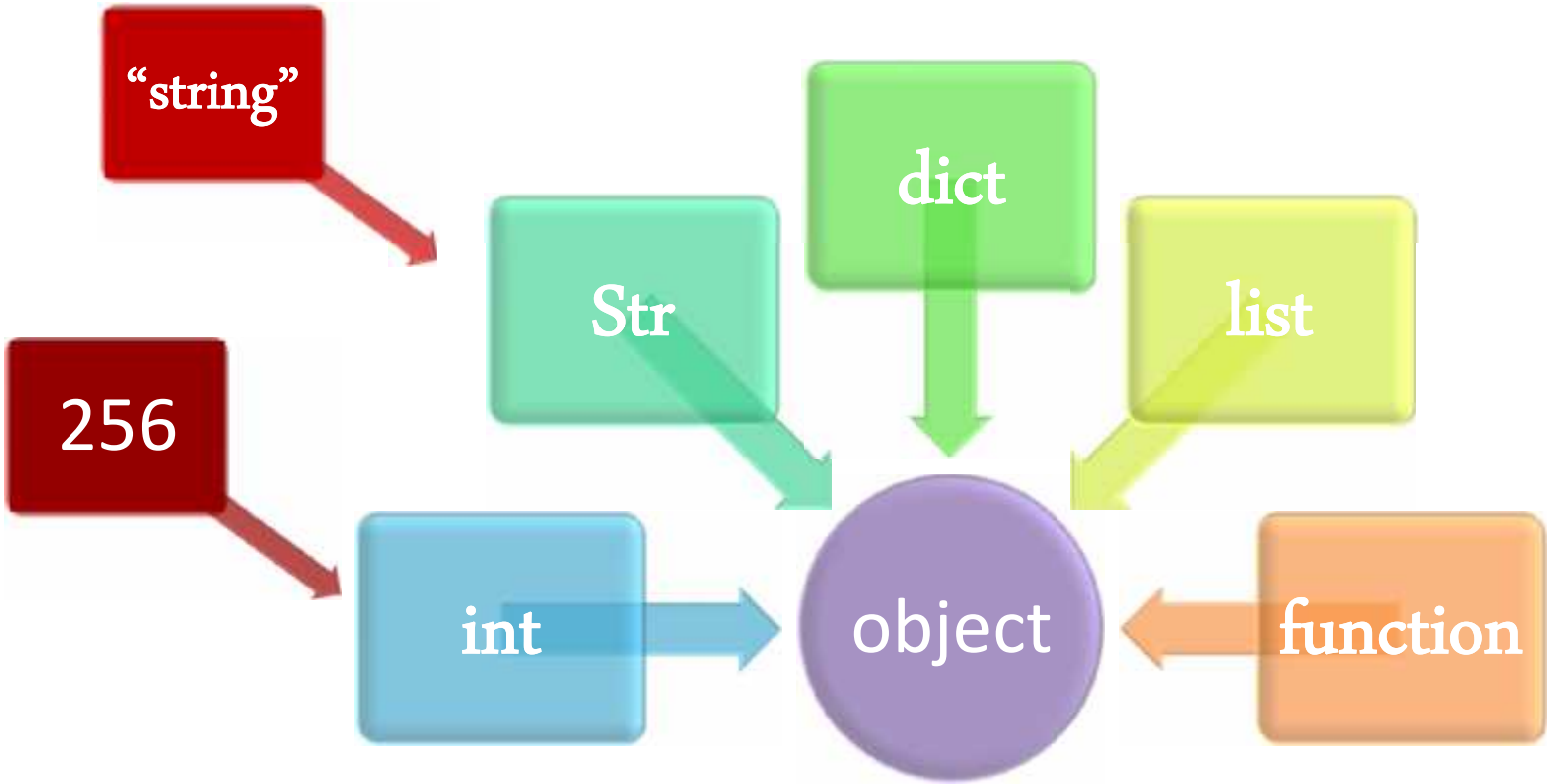

برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
کلاس
فصل هشتم
فصل هفتم
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

چیدمان کلاس ها

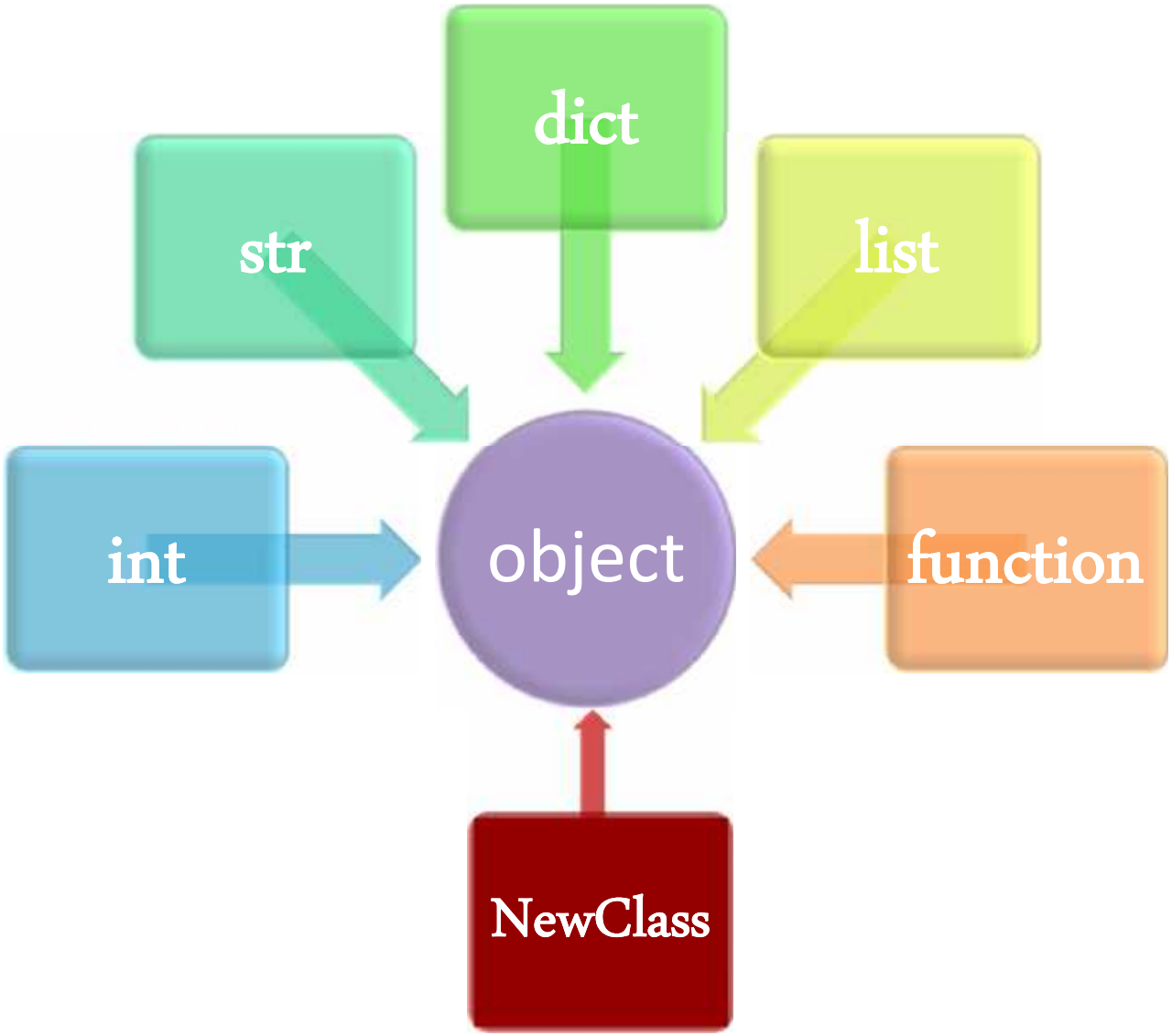



```
isinstance(int,object)
isinstance(bool,int)
issubclass(bool, int)
```

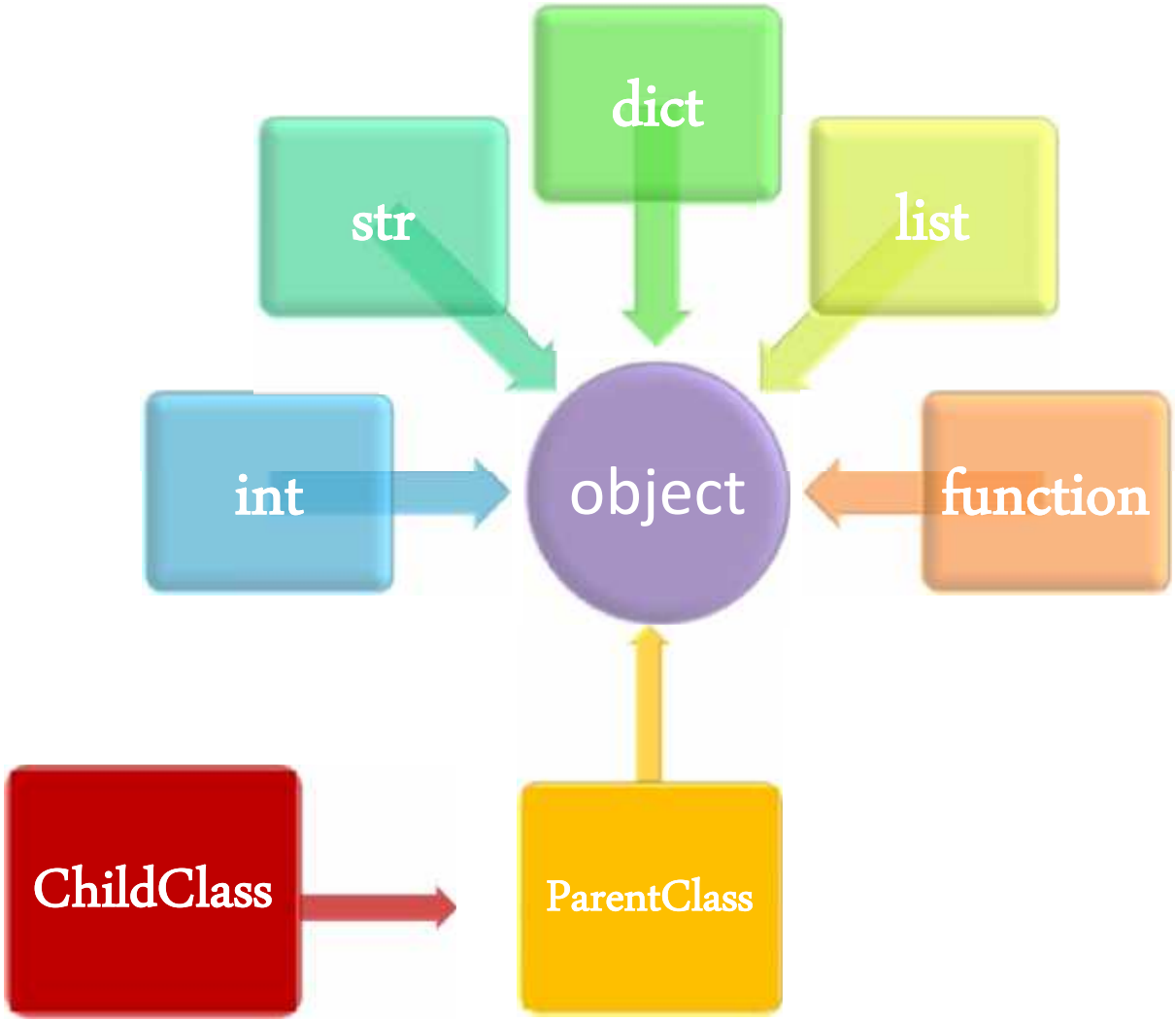
اشیا



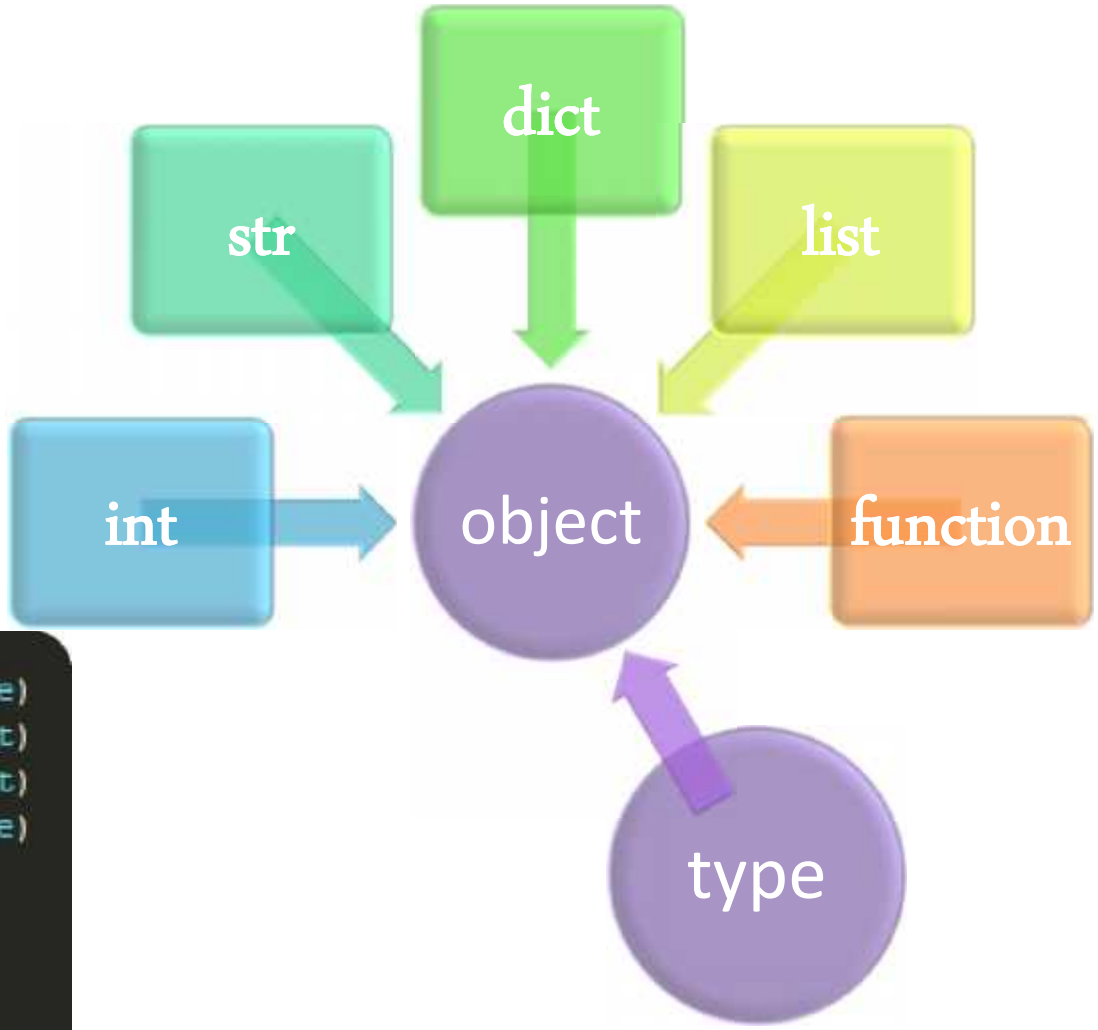
کلاس جدید



ارث بری

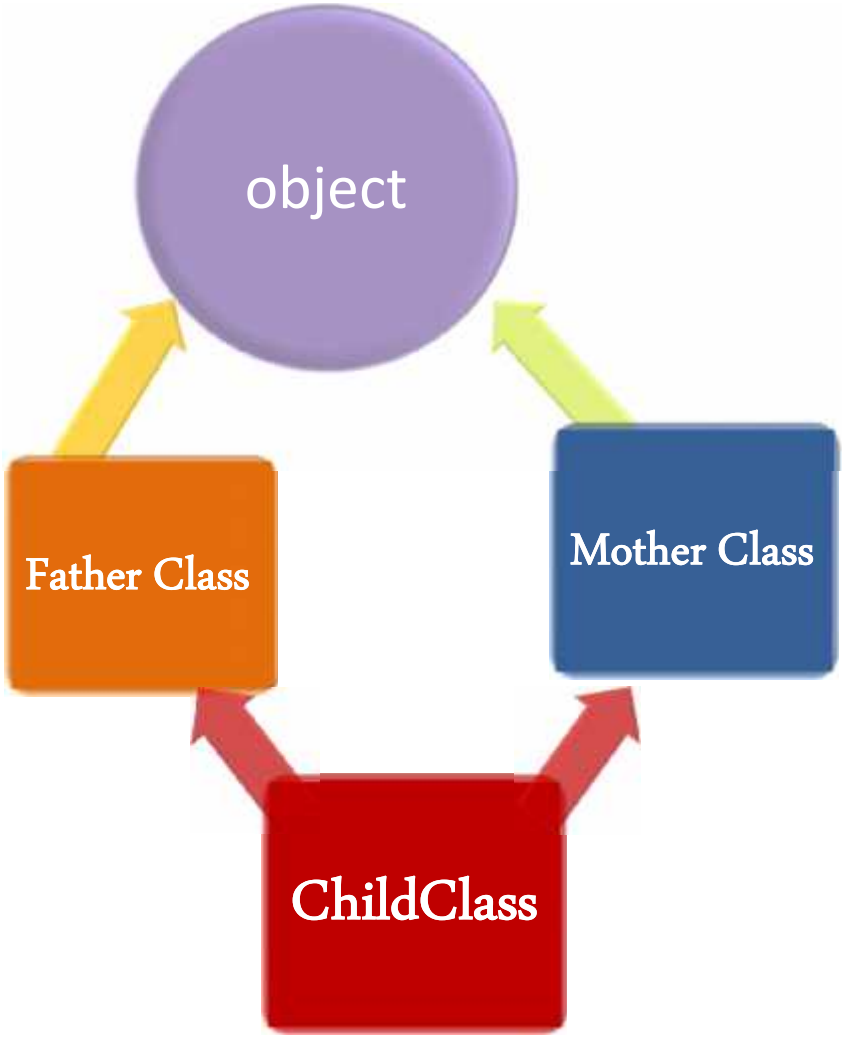


چیدمان واقعی

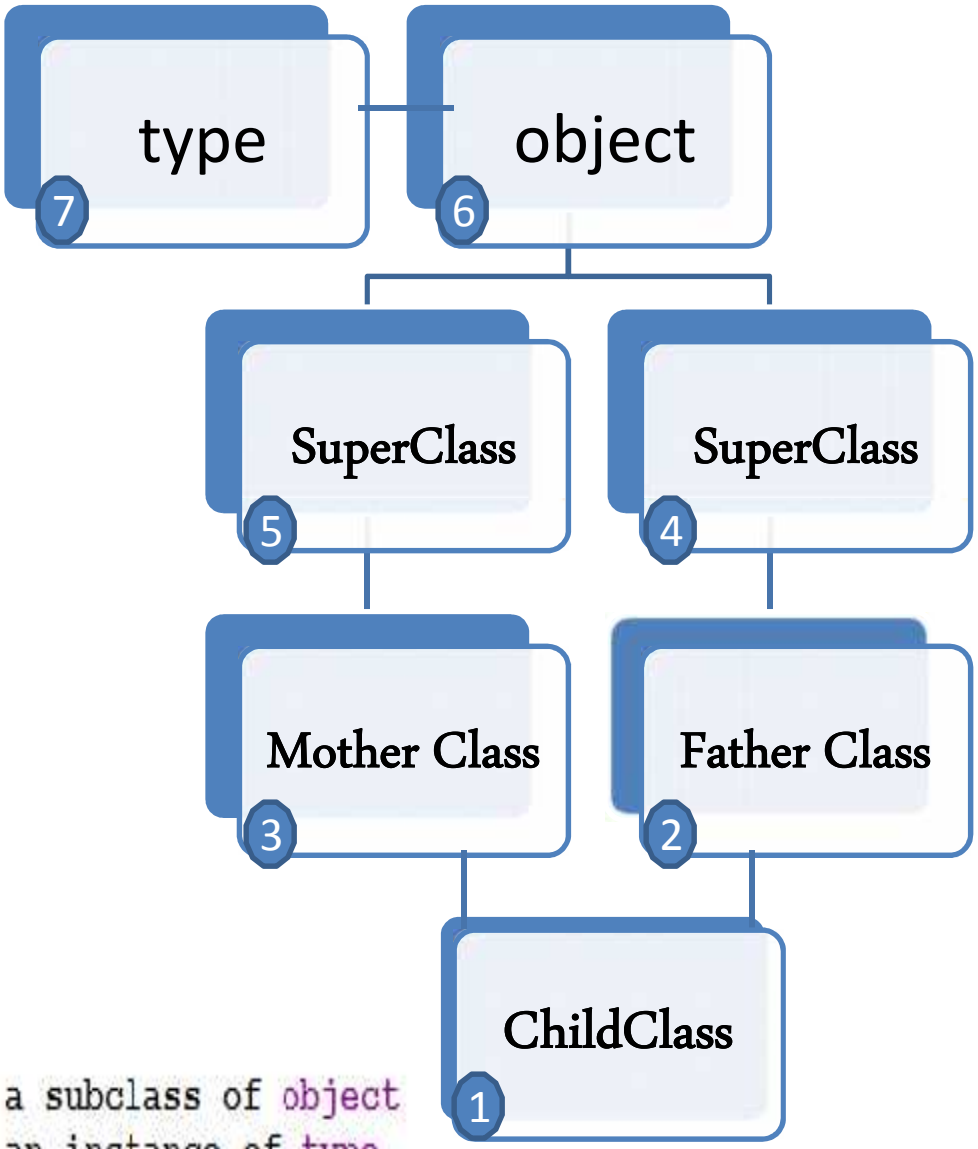



```
isinstance(object,type)
isinstance(type,object)
issubclass(type,object)
issubclass(object,type)
type(object)
type(type)

isinstance(int,type)
issubclass(int,type)
type(str)
```

ارث بری



ترتیب ارث بری



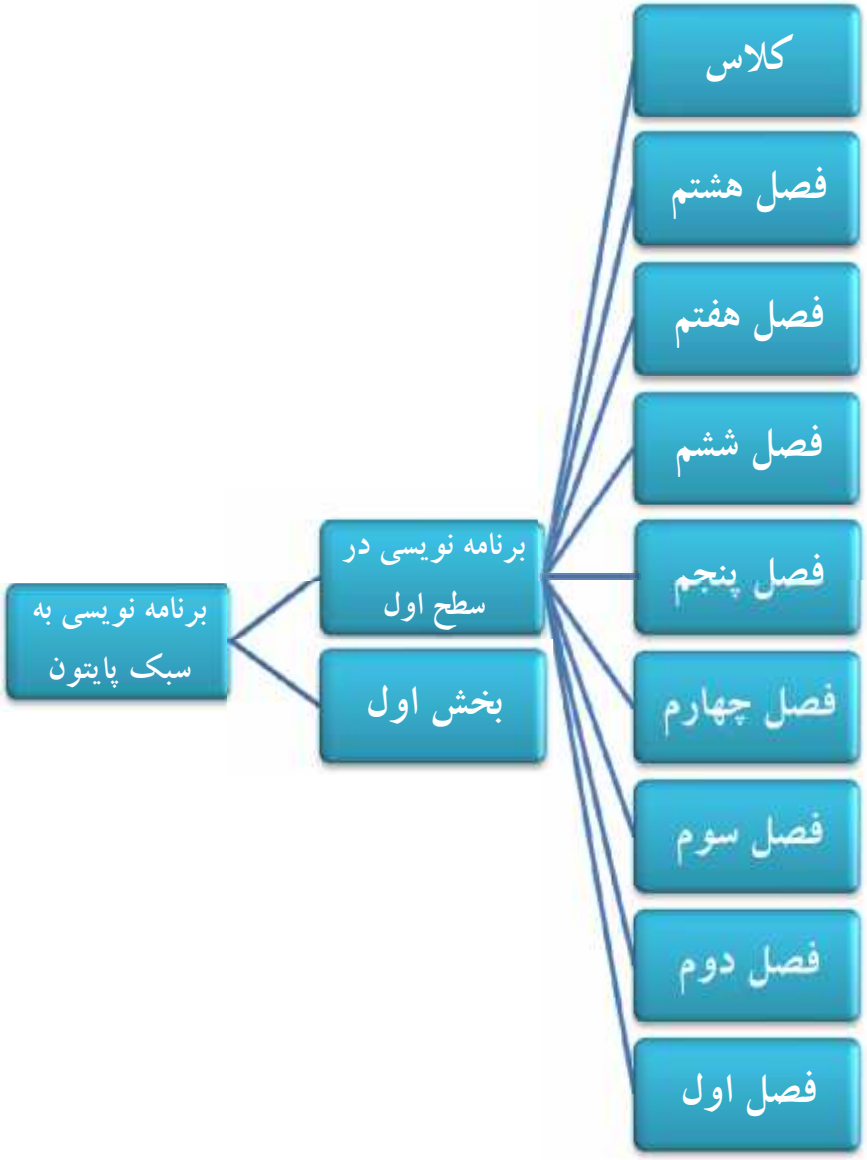

```
Every class is a subclass of object
Every class is an instance of type
```

به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

تابستان ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
کلاس
فصل هشتم
فصل هفتم
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

تعریف کلاس

```
class CamelCase(object):
    '''
    Doc
    '''
    class_attribute

    def class_method(self):
        #method body
        class ClassInFunction():
            #class_attribute
            #class_method
            pass
```

مقدار دهی اولیه

```
class animal(object):

    def __init__(self,name):
        '''
            initializer
        '''
        self.name='instance variable = '+name

    def print_name(self):
        print(self.name)

instance_cat=animal('Cat')
instance_dog=animal('Dog')
instance_cat.print_name()
instance_dog.print_name()
```

With explicit references, there is no need to have a special syntax for method definitions nor do you have to worry about complicated semantics concerning variable lookup. Instead, one simply defines a function whose first argument corresponds to the instance, which by convention is named "self."

خویش

```
class spam():
    def __init__(self, egg):
        self.egg=egg
    def get_egg(self):
        return self.egg

egg=spam('Ostrich egg') #egg is instance of spam
isinstance(egg,spam)
egg.get_egg() #Automatic self
spam.get_egg(egg)#Handy self

#self = first argument or bunded object
spam.get_egg(spam('chicken egg'))#self=spam instance
spam('chicken egg').get_egg()#self argument is automatically set
egg=spam('chicken egg').get_egg
egg
egg==egg.__self__.get_egg
egg==spam.get_egg
```

متغییرهای کلاس VS متغییرهای شی

```
class Dog:

    tricks = []             # mistaken use of a class variable

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

    def add_trick(self, trick):
        self.tricks.append(trick)
        #or
        Dog.tricks.append(trick)

class DogCorrectDesign:
    '''
    Correct design of the class
    '''

    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)
```

```
d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
print(d.tricks )# unexpectedly shared by all dogs
```

روی هوا !

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

d = Dog('Fido')
d.trick='roll over'
e = Dog('Buddy')

print('Dog name= ', d.name,'.trick/s : ', d.trick)
print(e.name)
print(e.trick) #AttributeError
```

مثلا

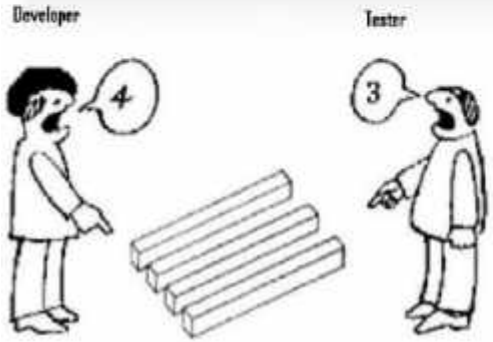
```python
class animal(object):
    def print_name(self):
        print(self.name)

animal_instance=animal()
animal_instance.print_name()
#AttributeError: 'animal' object has no attribute 'name'
animal.name="Cat" # add attribute !
animal_instance=animal()
animal_instance.print_name()


class C():
   pass

#Define a global function:
def meth(myself, arg):
   myself.val = arg
   return myself.val

#Poke the method into the class:
C.meth = meth
C.meth(C,12)
```

محدوده

```python
name='module_variable'

class animal(object):

    name="class variable"

    def __init__(self,name):
        self.name='instance variable = '+name

    def print_name(self):
        name='local method variable'
        print(self.name)
        print(animal.name)
        print(name) #local method variable or
                    #module variable

instance_cat=animal('Cat')
instance_dog=animal('Dog')
instance_cat.print_name()
instance_dog.print_name()
print(name)
```

ترکیب محدوده & روی هوا

```
class Test(object):
    i = 3 #This is a class attribute

x = Test()
x.i = 12                    #Attempt to change the value of the class attribute using x instance
print(   x.i == Test.i  )   #different
print(   Test.i == 3    )   #Test.i was not affected
print(   x.i == 12      )   #x.i is a different object than Test.
```

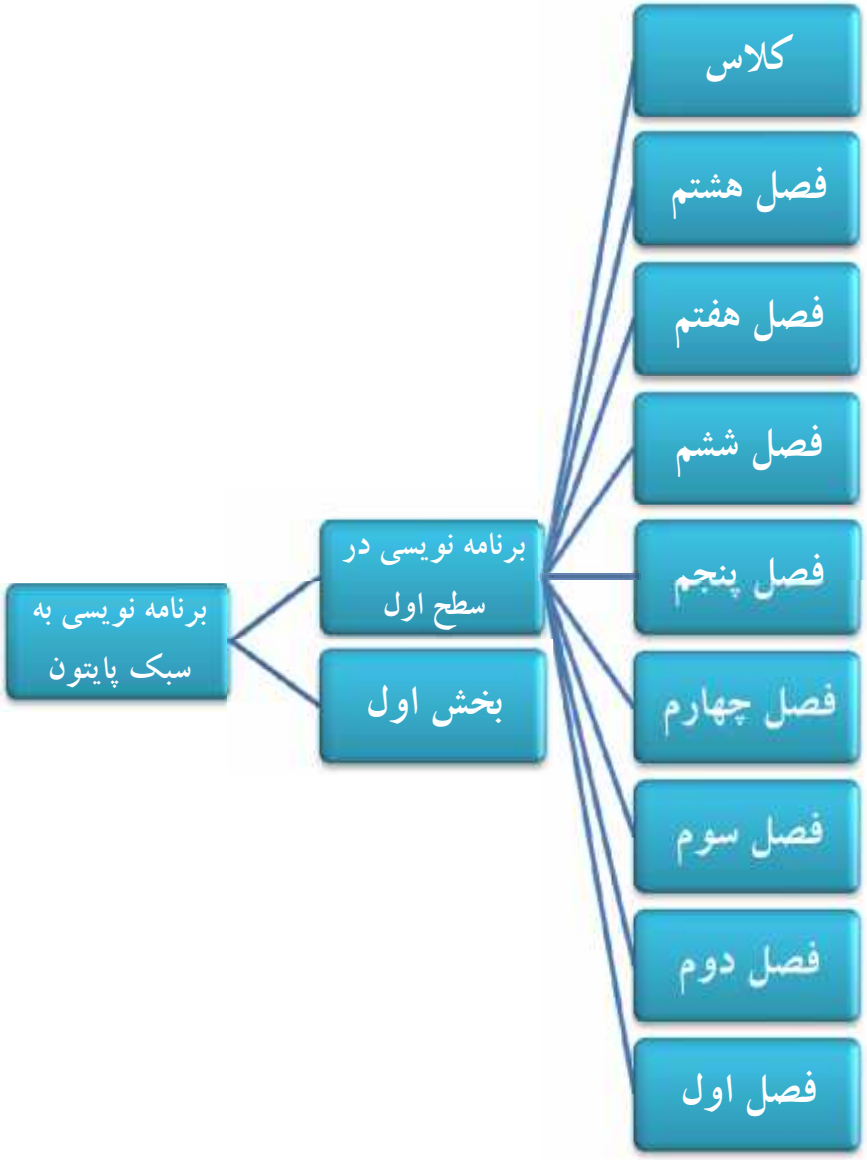


به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

تابستان ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
کلاس
فصل هشتم
فصل هفتم
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

پنهان سازی در پایتون نداریم !!!

```python
class Doberman():
    _age=None
    def __init__(self):
        self._name='Doberman'
        Doberman._age=3
        self.public='see You!'

    def _Print(self):
        print(self._name ,self._age)
```

ارث بری

```
class LivingThings():
    name=''
    age=0
    def print_livingthings_info(self):
        print('name= ',self.name)
        print('age= ',self.age)

class Shape():
     def Shape(self):
         print('If it looks like a duck \
          and walks like a duck, it is a duck!')
class Dog(LivingThings, Shape):
    breed=''

    def Print(self):
        print('breed= ',self.breed)
```

```
dog_instance=Dog()
dog_instance.Print()
dog_instance.breed='pitbull'
dog_instance.Print()
dog_instance.name
dog_instance.name='pit'
dog_instance.name
dog_instance.Shape()
```

Attribute Name	Meaning	Attribute Name	Meaning
__doc__	The function's documentation string, or None if unavailable; not inherited by subclasses. Writable.	__code__	The code object representing the compiled function body. Writable.
__name__	The function's name. Writable.	__defaults__	A tuple containing default argument values for those arguments that have defaults, or None if no arguments have a default value. Writable.
__qualname__	The function's *qualified name* *New in version 3.3.* Writable.	__globals__	A reference to the dictionary that holds the function's global variables — the global namespace of the module in which the function was defined. Read-only.
__module__	The name of the module the function was defined in, or None if unavailable. Writable.	__dict__	The namespace supporting arbitrary function attributes. Writable.
__closure__	None or a tuple of cells that contain bindings for the function's free variables. Read-only.	__annotations__	A dict containing annotations of parameters. The keys of the dict are the parameter names, and 'return' for the return annotation, if provided. Writable.

مثلا

```
class spam():
    def __init__(self,name):
        self.name=name
    def __str__(self):
        return(self.name)


x=spam('pd')
print(x)
```

دست رسی فرزند به والد

```
class LivingThings():
    name=''
    age=0
    def print_livingthings_info(self):
        print('name= ',self.name)
        print('age= ',self.age)

class Shape():
     def Shape(self):
         print('If it looks like a duck\
          and walks like a duck, it is a duck!')
class Dog(LivingThings, Shape):
    breed=''

    def Print(self):
        print('breed= ',self.breed)
        super().Shape()
        super().print_livingthings_info()

dog_instance=Dog()
dog_instance.name='bully'
dog_instance.age=5
dog_instance.breed='bulldog'
dog_instance.Print()
```

دست رسی والد به فرزند

```python
class Dog():

    def __init__(self):
        self.name='Dog'
        Doberman.age='Dog'

class Doberman(Dog):
    age=None
    def __init__(self):
        self.name='Doberman'
        Doberman.age='Doberman'
        super().__init__()#COMMENT !!

dob=Doberman()
dob.name
dob.age
```

VS


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

class bar(foo):
    def __init__(self, name, age):
        self.age=age
        super().__init__(name)

x=foo('x')
x is foo
x==foo
y=foo('y')
'x is y: ', x is y
x==y
isinstance(x,foo)
isinstance(y,foo)
y=bar('y',3)
isinstance(y,foo)
isinstance(y,bar)
foo.__subclasses__()
bar.__bases__
bar.__mro__
```

پیاده سازی صحیح

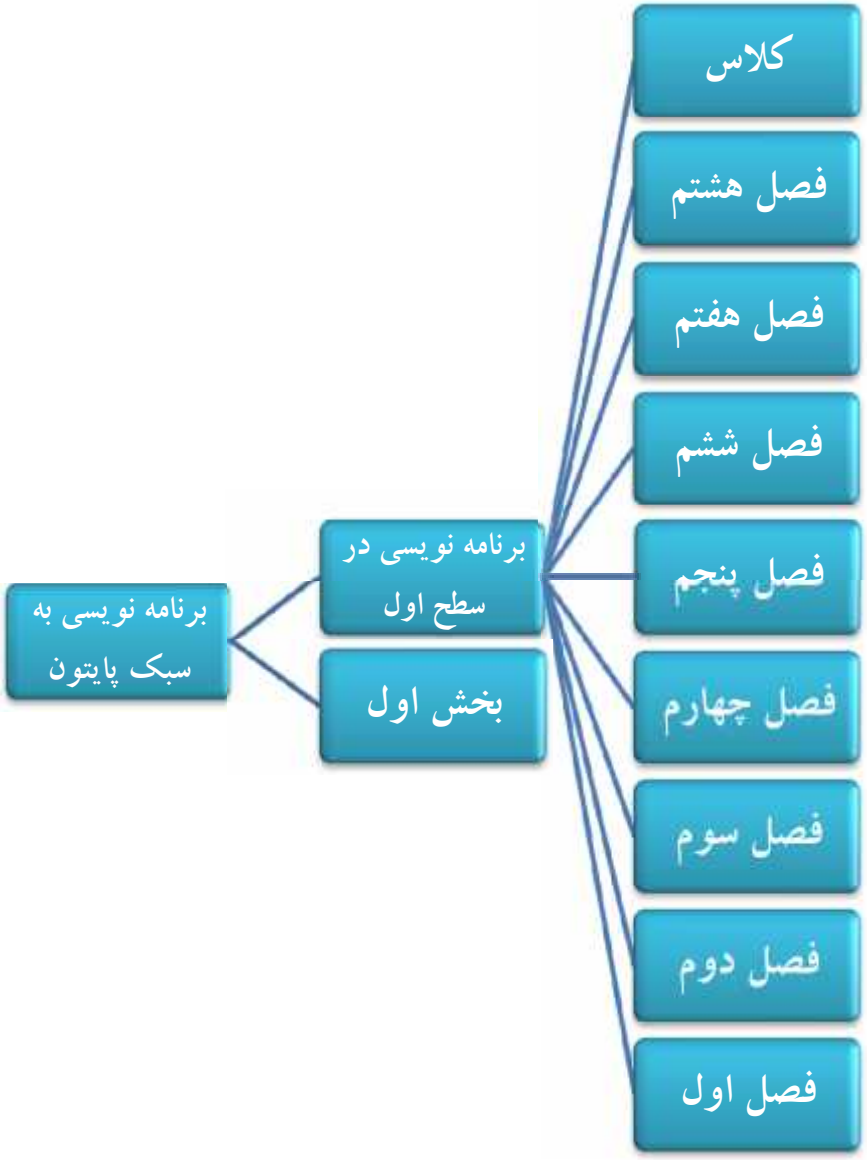
```
same='module'
class LivingThings():
    same='LivingThings'
    def __init__(self, name, age):
        self.name=name
        self.age=age

    def print_livingthings_info(self):
        print('name= ',self.name)
        print('age= ',self.age)

class Shape ():
    same='Shape'
    def shape(self):
        print('If it looks like a duck \
          and walks like a duck, it is a duck!')

class Dog(LivingThings, Shape):
    same='Dog' #lets delete same!
    def __init__(self, name, age, breed):
        super().__init__(name, age)
        self.breed=breed

    def Print(self):
        super().print_livingthings_info()
        print('breed= ',self.breed)
```

به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

مهر ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
کلاس
فصل هشتم
فصل هفتم
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

اضافه بار در پایتون

```python
class Dog():
    '''
    Overloading
    Test
    '''
    Dog_counter=0
    def __init__(self):
        Dog.Dog_counter+=1

    def Print(self):
        print('by self',self.Dog_counter)

    def Print():
        print('by class',Dog.Dog_counter)


obj=Dog()
Dog.Print() # Print() = Second Print method => Error
obj.Print()
obj=Dog()
obj=Dog()
obj.Print()
```

اضافه بار روی مقدار اولیه

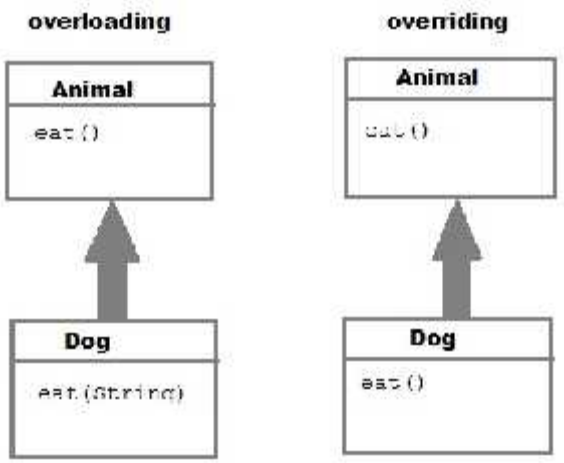
```python
class Dog():
    '''
    Ovearloading
    init
    '''
    Dog_counter=0
    def __init__(self):
        Dog.Dog_counter+=1

    def __init__(self,name):
        Dog.Dog_counter+=1
        self.name=name
d=Dog('dobi')
d=Dog()#TypeError
```

راه حل

```
class Dog():
    '''
    Overloading
    init
    '''
    Dog_counter=0
    def __init__(self, name=None, age=None):
        Dog.Dog_counter+=1
        if name and age:
            self.name=name
            self.age=age
        elif age:
            self.age=age
        elif name:
            self.name=name
        else:
            pass
    def set_dog_name_age(self,name,age):
        self.name=name
        self.age=age
    def set_dog_age(self,age):
        self.age=age
    def set_dog_name(self,name):
        self.name=name
```

```
dood=Dog()
dood.set_dog_name('dood')
dood.name
dood.Dog_counter
dobi=Dog('dobi',10)
dobi.Dog_counter
dobi.name
```

پایتون مادرزاد اضافه بار دارد!




```
class Dog():
    '''
    Ovearloading
    init
    '''
    Dog_counter=0
    def __init__(self):
        Dog.Dog_counter+=1

dood=Dog()
dood.name='dood'
dood.age=12
dobi=Dog()
dobi.name #dobi has no attribute 'name'
Dog.name=''
dobi=Dog()
dobi.name
dobi.name='dobi'
```

بازنویسی/متد انتزاعی

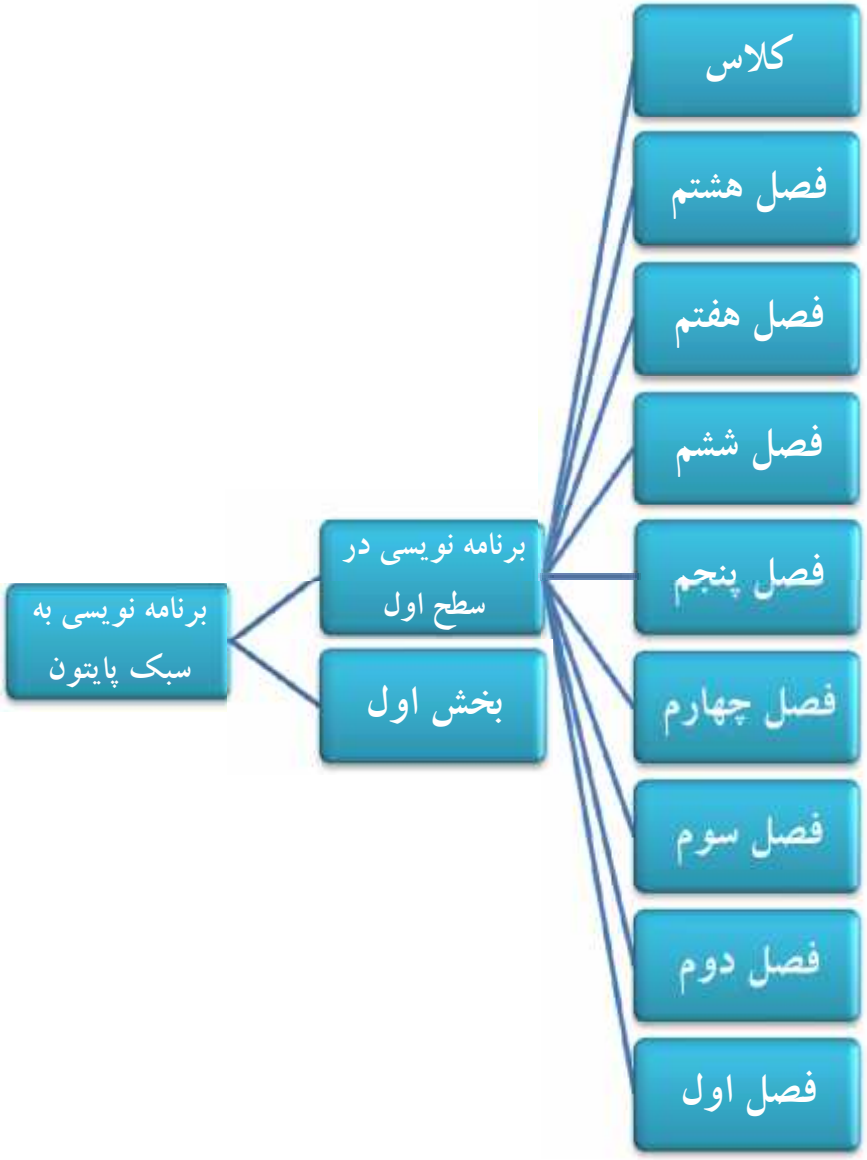
```python
class SuperClass(object):

    def method_for_override(self):
        raise AssertionError('override this method')

class Child(SuperClass):
    def method_for_override(self):
        print('override!!!')

check=SuperClass()
check.method_for_override()
#AssertionError: override this method
check=Child()
check.method_for_override()
```

به نام پروردگار دانایی

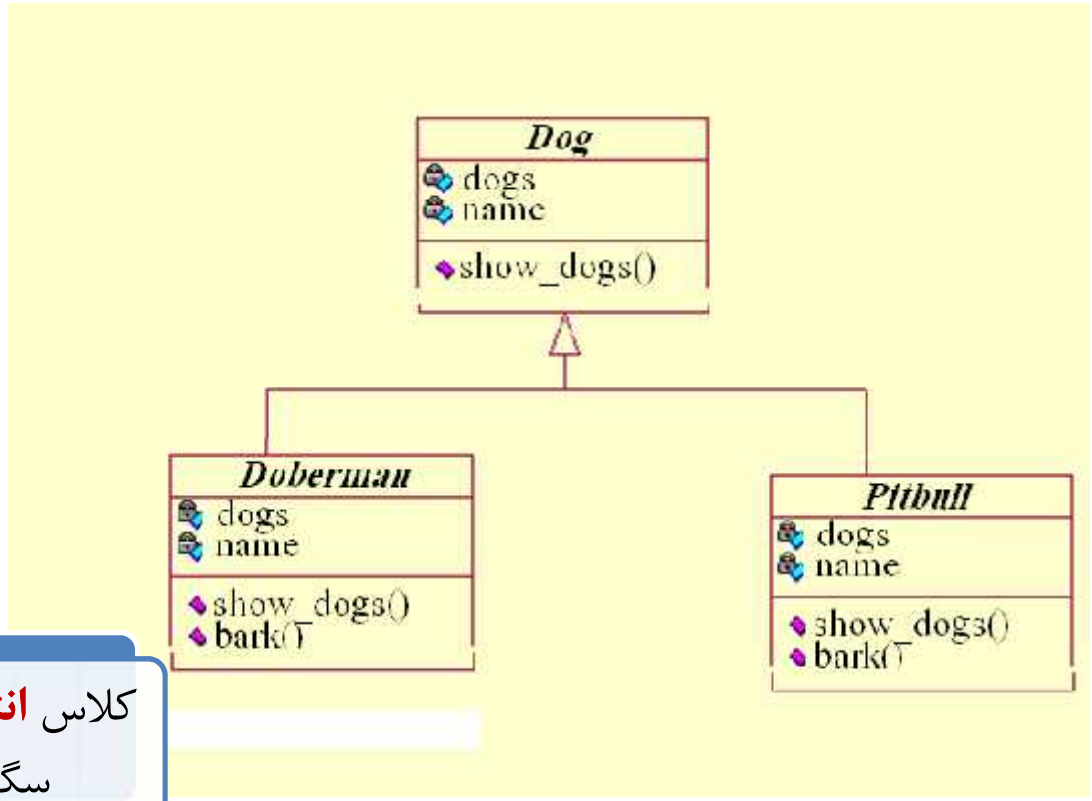
برنامه نویسی به سبک پایتون

پدرام شاه صفی

مهر ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
کلاس
فصل هشتم
فصل هفتم
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

پیاده سازی


Dog
dogs
name
show_dogs()
Doberman
dogs
name
show_dogs()
bark()
Pitbull
dogs
name
show_dogs()
bark()


کلاس انتزاعی
سگ

متغییرهای مشترک

```
class Dog:
    age=int()
    name=list()

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

    def __str__(self):
        return "age: "+str(Dog.age)+"\nname"+str(Dog.name)

doberman_instance=Dog(15,'doberman')
pitbull_instance =Dog(16,'pitbull')
print(doberman_instance)
```

JavaScript

متدهای شی

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

    def __str__(self):
        return_srting=str(self.__class__)
        return_srting+="\nage: "+str(self.age)
        return_srting+="\nname: "+str(self.name)
        return return_srting


doberman_instance=Dog(5,'doberman')
print(doberman_instance)
```

یک مشکل

```
class Dog:
    dogs = [] # this is a class variable

    def __init__(self, name):
        self.name = name #self.name is an instance variable
        Dog.dogs.append(name)

    def bark(self, n): # this is an instance method
        print("{} says: {}".format(self.name, "woof! " * n))

    def dog_list(): #this is implicitly a class method (see comments below)
        for dog in Dog.dogs:
            print(dog,end=', ')

doberman_instance=Dog('doberman')
pitbull_instance =Dog('pitbull')
Dog.dog_list()
#doberman,pitbull,
doberman_instance.dog_list()
#Error!
```

راه حل

```python
class Dog:
    dogs = [] # this is a class variable
    def __init__(self, name):...
    def bark(self, n):...

    @staticmethod
    def dog_list(): #this is implicitly a class method (see comments below)
        for dog in Dog.dogs:
            print(dog,end=', ')

doberman_instance=Dog('doberman')
pitbull_instance =Dog('pitbull')
Dog.dog_list()
doberman_instance.dog_list()
```

C++

متدهای استاتیک

staticmethod

- این متد ها هیچ کاری با کلاس و instance ان کلاس ندارد.
- یکی از اهداف دسته بندی این متد ها در کلاس ها مربوط به خودشان است.
- گر static method جاوا دارید فکر میکنید باید بگویم کاملا در اشتباهید. کلا قضیه متفاوت است!
- این متد مربوط به instanceخاصی نمیشود پس هیچ instance ای هم به ان متصل نیست.
- پایتون این موضوع را میفهمد که قرار نیست به این متد هیچ instanceای عنوان ورودی selfارسال شود.
- سعی کنید متد را از کلاس خارج کنید و در خارج کلاس باید همان کاری را کند که در داخل ان میکرد. اما نکته مهم این است که این متد ها کاری با کلاس و instanceندارند .

```
class Dog:
    dogs = [] # this is a class variable
    def __init__(self, name):...
    def bark(self, n):...

def dog_list():
    for dog in Dog.dogs:
        print(dog,end=', ')

doberman_instance=Dog('doberman')
pitbull_instance =Dog('pitbull')
dog_list()#doberman,pitbull
```

کاربرد صحیح متدهای استاتیک

```python
class Date(object):
    def __init__(self, day, month, year):
        date=str(day)+str(month)+str(year)
        if Date._check(date):
            self.day = day
            self.month = month
            self.year = year

    @staticmethod
    def _check(date):
        if date.isdigit():
            return True
        print('Date fortmat is: "dd-mm-yyyy ')
        return False
```

متدهای کلاس

classmethod

- به عنوان اولین ورودی خود یک کلاس را میگیرند.
- اولین ورودی انها ثابت است و ان را با cls نشان میدهیم.
- این متد ها مربوط به instance نمیشود.
- این متد ها برای کار با کلاس ها میشود.

Transformation	Called from an Object	Called from a Class
function	f(obj, *args)	f(*args)
staticmethod	f(*args)	f(*args)
classmethod	f(type(obj), *args)	f(klass, *args)

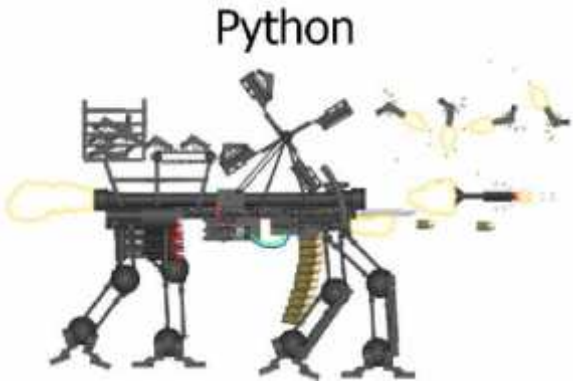
مثلا

```
class Dog():
    _dogs = []
    def __init__(self, name):
        Dog._dogs.append(name)
    def print_dogs(self):
        print(Dog._dogs)

class Doberman(Dog):
    _doberman_dogs = []

    def __init__(self, name):
        self.name=name
        Doberman._doberman_dogs.append(name)
        super().__init__(name)
    def print_dogs(self):
        print(Doberman._doberman_dogs)

class Pitbull(Dog):
    _pitbull_dogs = []

    def __init__(self, name):
        self.name=name
        Pitbull._pitbull_dogs.append(name)
        super().__init__(name)
    def print_dogs(self):
        print(Pitbull._pitbull_dogs)
```

Java/C#

طراحی صحیح

```python
class Dog():
    _dogs = []
    def __init__(self, name):
        self._dogs.append(name)

    @classmethod
    def show(cls):
        print( cls._dogs)

class Doberman(Dog):
    _dogs = []
    def __init__(self, name):
        super().__init__(name)

class Pitbull(Dog):
    _dogs = []
    def __init__(self, name):
        super().__init__(name)
```



VS

```python
class Dog():
    _dogs = []
    def __init__(self, name):
        self._dogs.append(name)

    def show(self):
        print( self._dogs)

class Doberman(Dog):
    _dogs = []
    def __init__(self, name):
        super().__init__(name)

class Pitbull(Dog):
    _dogs = []
    def __init__(self, name):
        super().__init__(name)
```

```python
dp=Doberman('dobi')
doberman=Doberman('dob')
pitbull=Pitbull('pit')
dp.show()#['Doberman puppy', 'Doberman1']
pitbull.show()#['Pitbull puppy']
Doberman.show()#Error!
Dog.show()#error
Doberman.show(dp) #['dobi', 'dob']
                  #Doberman.show()
```

کلاس انتزاعی

```
class ABS(object):
    def __init__(self):
        self.check()

    @classmethod
    def check(cls):
        if not cls.__name__ in (child.__name__ for child in ABS.__subclasses__()):
            raise NotImplementedError

class SubClass(ABS):
    def __init__(self):
        super().__init__()


x=SubClass()
x=ABS()
#NotImplementedError
```

C++

طراحی نهایی

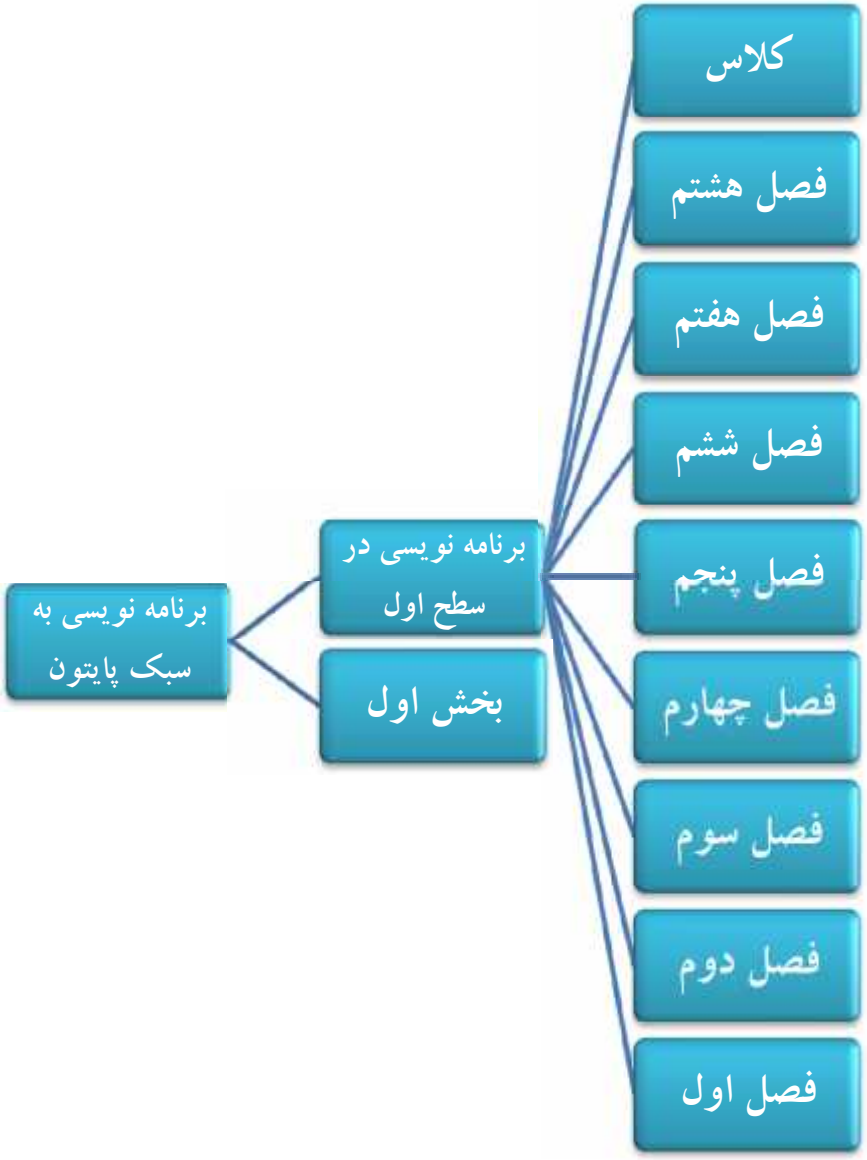
```
class Dog():
    _dogs = []
    def __init__(self, name):
        if self.check_is_child():
            self._dogs.append(name)#add to Dog subclass
            Dog._dogs.append(name)
        else:
            raise NotImplementedError
    def bark(self,n):
        print("{} says: {}".format(self.name, "woof! " * n))
    @classmethod
    def show(cls):
        return (cls._dogs)
    @classmethod
    def check_is_child(cls):
        if cls.__name__ in (child.__name__ for child in Dog.__subclasses__()):
            return True
        return False

class Doberman(Dog):
    _dogs = []
    def __init__(self, name):
        super().__init__(name)

class Pitbull(Dog):
    _dogs = []
    def __init__(self, name):
        super().__init__(name)
```

به نام پروردگار دانایی

برنامه نویسی به سبک پایتون

پدرام شاه صفی

مهر ۱۳۹۴


برنامه نویسی به سبک پایتون
برنامه نویسی در سطح اول
بخش اول
کلاس
فصل هشتم
فصل هفتم
فصل ششم
فصل پنجم
فصل چهارم
فصل سوم
فصل دوم
فصل اول

کلاس به سبک پایتون

- در اکثر زبان ها این قضیه بر قرار است :
  - کلاس توصیف کننده ی اشیای خودش است .اما به صرف تعریف کلاس شی ای تشکیل نمیشود و حافظه ای به ان اختصاص داده نمیشود.
  - وقتی از کلاس شی می سازیم این شی یک فضا را اشغال میکند. شی عینیت یافته را object گویند.

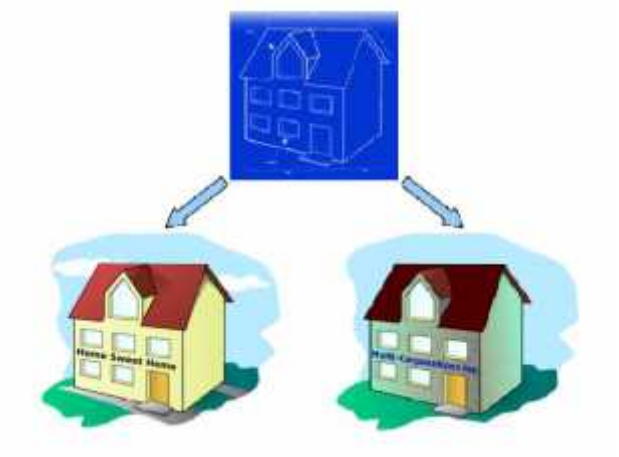
- اما در پایتون این قضیه برقرار نیست !
  - یعنی از انجایی که در پایتون نسخه ۳ همه چیز شی هست، کلاس ها هم شی اند !
  - یعنی به محض تعریف کلاس حتی قبل از ایجاد یک instance این کلاس به عنوان یک شی در حافظه قرار دارد.
  - چون یک کلاس است پس میتوان از ان instance یا object ساخت و میتوان به ان method و attribute اضافه کرد .
  - نکته مهم اینکه چون کلاس ها در پایتون شی اند میتوانید این کارها را on fly انجام بدهید.
  - چون شی است، مثل سایر اشیا در پایتون میتوان به راحتی کار با یک متغییر ساده با ان کار کرد. مثلا ان را در یک متغییر ریخت و میتوان ان را کپی کرد ویا به عنوان ورودی یک تابع پاس داد.

شی کلاس

```
class spam():
    pass
print(locals()['spam']) # class object
Object=spam()           #class instance
```

سازنده شی کلاس

```
type(name of the class,
     tuple of the parent class (for inheritance, can be empty),
     dictionary containing attributes names and values)
```

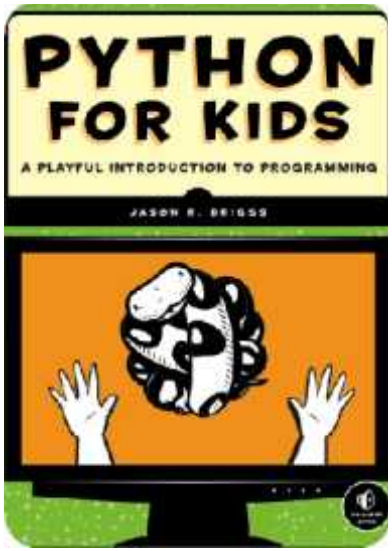


ساخت دستی شی کلاس

```
class Foo(object):
    pass

Foo = type('Foo', (), {}) # returns a class object

print(Foo)
print(Foo()) # create an instance with the class
```

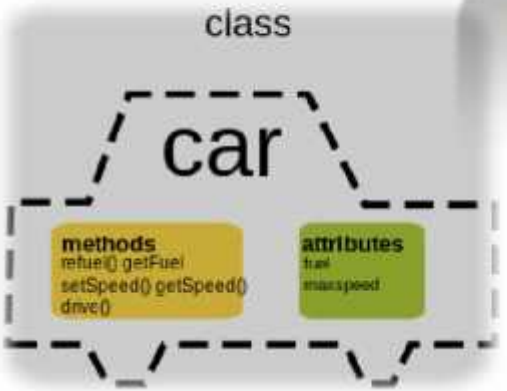
اضافه کردن صفات

```
class Foo(object):
    '''
    #type accepts a dictionary to define
    the attributes of the class.
    '''
    bar = True

#Can be translated to:
Foo = type('Foo', (), {'bar':True})
```



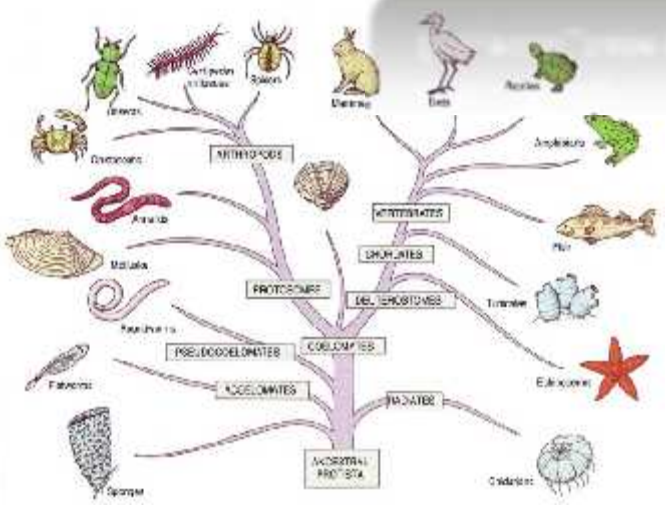
اضافه کردن پدر

```
Foo = type('Foo', (), {'bar':True})

class FooChild(Foo):
    '''
    inherit from Foo
    '''
    pass

FooChild = type('FooChild', (Foo,), {})
print(FooChild)
print(FooChild.bar) # bar is inherited from Foo
```



اضافه کردن متد

```
#add methods to your class
def echo_name(self):
    print(self.name)

Foo= type('Foo', (), {'name':'pd','echo_name': echo_name})
foo = Foo()
foo.echo_name()
```



ابر کلاس

Metaclassچیست ؟

- class of a class.
- همانطور که کلاس ایجاد گر instance است ؛ metaclass ایجادگر یک class است.
  class یک instance از metaclass است.
- metaclass در واقع type کلاس را مشخص میکند.
- در واقع پایتون وقتی به عبارت class میرسد با استفاده از صدا زدن metaclass یک classایجاد میکند. برای انجام اینکار از متد های __init__ ، __new__ استفاده میکند.

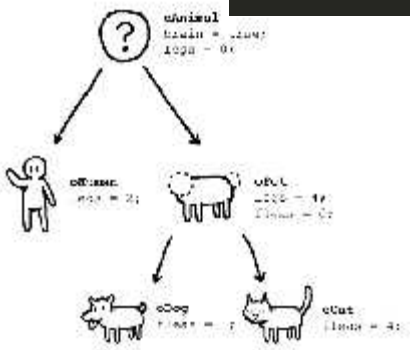
سازنده کلاس ها

- type
  - type در پایتون یک metaclass است. دیدید چطوری باهاش کلاس میشه ساخت.
  - typeخودش یک class است و چون ازش برایساخت class استفاده میشود بهش class of a class میگوییم.
  - typeاز نوع type است .
- در پایتون type را نمیتوانید تغییر دهید اما میتوانید یک metaclass دلخواه ایجاد کنید.
- پس به طور خلاصه ما برای ساخت یک instance از class کمک میگیریم ، اما از انجایی که همه چیز در پایتون نسخه 3 شی است، classها هم شی اند . چه کسی این شی class را میسازد ؟ پاسخ metaclass است.

```
MyClass = MetaClass()
MyObject = MyClass()
```

کلاسی برای کلاس

```
MyClass = MetaClass()
MyObject = MyClass()

number=int()
string=str()
dictionary=dict()
Class=type('Class',(),{})

type(number)
number.__class__  # class of number => int
number.__class__.__class__  # class of class of number or
                            # class of int => type
number.__class__.__class__.__class__  # class of class of class of number
                                      # or class of type => type
```



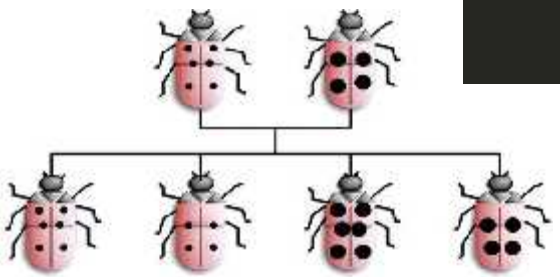
پیاده سازی ابر کلاس

```
class MyMetaClass(type):
    def __new__(cls, *args):
        print('call __new__ from MyMetaClass.')
        print('args are: ',args)
        print('cls is: ',cls)
        return type(*args)

class Foo(object, metaclass= MyMetaClass):

    def __new__(cls, *args):
        print('call __new__ from Foo.')
        print('args are: ',args)
        print('cls is: ',cls)
        return object.__new__(cls)
        #return super.__new__(cls)

    def __init__(self,name):
        print('call __init__ from Foo.')
        self.name=name
```

```
Use __new__ when you need to control the creation of a new instance.
Use __init__ when you need to control initialisation of a new instance.
__new__ is the first step of instance creation. It's called first,
 and is responsible for returning a new instance of your class.
In contrast, __init__ doesn't return anything;
it's only responsible for initializing the instance after it's been created.
In general, you shouldn't need to override __new__
unless you're subclassing an immutable type like str, int, unicode or tuple.
```

کلاس ارث نابر

```
class MyMetaClass(type):
    def __new__(mcs, name, bases, dictionary):
        print('call __new__ from MyMetaClass.')
        return type(name , () , dictionary)

class Bar():
    def Print(self):
        print('hi-tech')

class Foo(Bar, metaclass= MyMetaClass):

    def __new__(cls, *args):
        print('call __new__ from Foo.')
        return object.__new__(cls)

    def __init__(self,name):
        print('call __init__ from Foo.')
        self.name=name
```



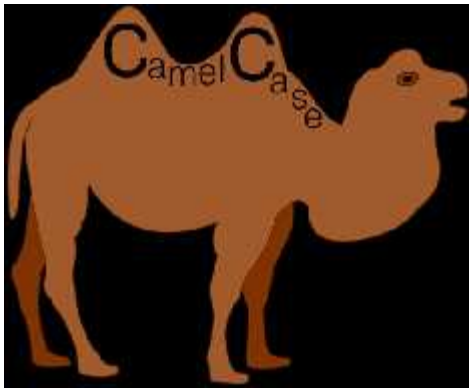
توصیفگر

```
class RevealAccess(object):
    """A data descriptor that sets and returns values
       normally and prints a message logging their access.
    """

    def __init__(self, initval=None, name='var'):
        self.val = initval
        self.name = name

    def __get__(self, obj, objtype):
        print('Retrieving', self.name)
        return self.val

    def __set__(self, obj, val):
        print('Updating', self.name)
        self.val = val
```

```
class MyClass(object):
    x = RevealAccess(10, 'var "x"')
    y = 5

m = MyClass()
m.x
m.x = 20
m.x
m.y
```

ویژگی

```
property(fget=None, fset=None, fdel=None, doc=None) -> property attribute

class C(object):
    def getx(self): return self.__x
    def setx(self, value): self.__x = value
    def delx(self): del self.__x
    x = property(getx, setx, delx, "I'm the 'x' property.")
```



پیاده سازی

```
class C(object):
    def __init__(self):
        self._name = None

    @property
    def name(self):
        """I'm the 'x' property."""
        print('@name.getter')
        return self._name

    @name.setter
    def name(self, value):
        print('@name.setter')
        self._name = value

    @name.deleter
    def name(self):
        print('@name.deleter')
        del self._name
```

```
instance=C()
instance.name='pd'
instance.name
instance. name
del instance._name
```

تمرین

- با ابزار های اماده ی پایتون یک کلاس ABS ایجاد کنید.
- کلاس انسان را پیاده سازی کنید، دو ابر کلاس دارد، پدر و مادر. اجزای بدن جزیی از بدن هستن. اجزای بدن را به دسته های سر ، بدنه، دست و پا تقسیم کنید و هر کدام کلاس خاص خود را دارند. تمام این اجزا را به روش صحیح طراحی و پیاده سازی کنید.
- موضوع تحقیق ، **Big Data .**


RYKA
CBT