برنامه نویسی شی گرا

برنامه نویسی شی گرا (OOP) یکی از موانع اصلی برای مبتدیان در زمان شروع یادگیری پایتون است.

در این درس، با ساختار OOP در پایتون با استفاده از مباحث زیر آشنا خواهید شد:

- اشیاء
- استفاده از کلمه کلیدی *class*
- ساخت و تعریف ویژگی های کلاس
- ساخت متدها در یک کلاس
- آموزش در مورد Inheritance
- آموزش در مورد Polymorphism
- آموزش در مورد Special Methods برای کلاس ها
- و بسیاری دیگر ...

با یادآوری درباره اشیاء پایه پایتون، شروع به یادگیری OOP خواهید کرد. به عنوان مثال:

In [4]:

```
lst  = [1, 2, 3]
```

بخاطر بیاورید چگونه متدهای یک لیست را می توانیم فراخوانی کنیم

In [5]:

```
type(lst)
```

Out[5]:

```
list
```

In [6]:

```
lst.count(2)
```

Out[6]:

```
1
```

ساخت اشیاء در پایتون

در این درس، ما به بررسی این می پردازیم که چگونه می توانیم یک نوع شیء مانند لیست ایجاد کنیم. قبلاً درباره ساخت توابع یاد گرفته بودیم. بنابراین بیایید به طور کلی درباره اشیاء صحبت کنیم:

اشیاء

در پایتون، *همه چیز شیء است*. به خاطر درس های قبل، ما می توانیم از تابع `type()` برای بررسی نوع شئ استفاده کنیم:

In [7]:

```
print(type(1))
print(type(''))
print(type([]))
print(type(()))
print(type({}))
```

```
<class 'int'>
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
```

پس ما می دانیم که همه این چیزها شیء هستند، پس چگونه می توانیم نوع شئ خود را ایجاد کنیم؟ اینجاست که کلید واژه `class` وارد می شود.

شروع کار با کلاس های پایتون

پایتون یک زبان برنامه نویسی چند الگویی است که با کلاس هایی که می توانید با کلید واژه class تعریف کنید، برنامه نویسی شی گرا (OOP) را پشتیبانی می کند. می توانید یک کلاس را به عنوان یک قطعه کد در نظر بگیرید که داده ها و رفتار را مشخص می کند و یک نوع خاص از شئ را نمایش می دهد و مدل می کند.

کلاس در پایتون چیست؟ یک شباهت رایج این است که یک کلاس مانند طرح بلوپرینت خانه است. شما می توانید از طرح بلوپرینت برای ایجاد چندین خانه و حتی یک محله کامل استفاده کنید. هر خانه بتنی یک شئ یا نمونه است که از طرح بلوپرینت به دست آمده است.

هر نمونه ممکن است دارای ویژگی های خود باشد، مانند رنگ، صاحب، و طراحی داخلی. این ویژگی ها حالت شئ را به خود مشخص می کنند. نمونه ها همچنین ممکن است رفتار های مختلف داشته باشند، مانند قفل کردن درب ها و پنجره ها، باز کردن درب گاراژ، روشن و خاموش کردن نور، آب دادن به باغچه و غیره.

در OOP، به طور عمومی از عبارات ویژگي ها (attributes) برای اشاره به خصوصیات یا داده های مرتبط با يك شئ خاص از يك كلاس داده شده استفاده مي شود. در پایتون، ویژگي ها به عنوان متغیرهای تعریف شده در داخل يك كلاس با هدف ذخیره کلیه داده های لازم برای کار کلاس تعریف شده از کلاس کاربردي استفاده مي شود.

بطور مشابه، به متدها برای اشاره به رفتار های مختلف که شئ ها نشان خواهند داد، استفاده خواهیم کرد. متد ها تابع هستند که در یک کلاس تعریف می شود. این توابع به طور عمده بروی یک نمونه یا کلاس پشتیبانی شده یک سطح عمل کرده و یا با ویژگی های نمونه یا کلاس پایه کار می کنند. ویژگی ها و متد ها به طور کلی به عنوان اعضای یک کلاس یا شئ ارجاع داده می شوند.

با استفاده از کلاس ها می توانید جهان واقعی را مدل کنید. این کلاس ها به شما در سازماندهی بهتر کد و حل مسائل برنامه نویسی پیچیده کمک خواهند کرد.

به عنوان مثال، شما می توانید از کلاس ها برای ایجاد شئ هایی استفاده کنید که انسان، حیوانات، وسایل نقلیه، کتاب ها، ساختمان ها، خودروها یا دیگر اشیاء را شبیه سازی می کنند. همچنین می توانید اشیاء مجازی را نمایش دهید، مانند یک سرور وب، درخت دایرکتوری، چت بات، مدیر فایل و غیره.

تعریف کلاس با پایتون

برای تعریف یک کلاس در پایتون، باید از کلمه کلیدی class و نام کلاس و دو نقطه استفاده کنید، همانطور که برای دیگر عبارات ترکیبی در پایتون انجام می دهید. سپس باید بدنه کلاس را تعریف کنید، که در سطح بعدی تورفتگی شروع خواهد شد:

In [9]:

```
# Create a new object type called Sample
class Sample:
    pass

# Instance of Sample
x = Sample()

print(type(x))
```

```
<class '__main__.Sample'>
```

در بدنه یک کلاس، می توانید ویژگی ها و متد ها را به عنوان نیاز تعریف کنید. همانطور که قبلاً یاد گرفتید، ویژگی ها متغیرهایی هستند که داده های کلاس را نگه می دارند، در حالی که متد ها توابعی هستند که رفتار را فراهم می کنند و به طور عمده بر روی داده های کلاس عمل می کنند.

توجه: در پایتون، بدنه یک کلاس به عنوان یک فضای نام عمل می کند که در آن ویژگی ها و متد ها زندگی می کنند. شما فقط می توانید به این ویژگی ها و متد ها از طریق کلاس یا شئ های آن دسترسی داشته باشید.

به عنوان مثال برای تعریف ویژگی ها و متد ها، فرض کنید نیاز به یک کلاس دایره برای شبیه سازی دایره های مختلف در یک برنامه نقاشی دارید. در ابتدا، کلاس شما یک ویژگی تک رادیوس برای نگه داری خواهد داشت. همچنین یک روش برای محاسبه مساحت دایره خود خواهید داشت:

In [10]:

```
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return round(math.pi * self.radius ** 2, 2)
```

در این قطعه کد، شما با استفاده از کلمه کلیدی class، کلاس دایره را تعریف می کنید. در داخل کلاس، دو متد را می نویسید. متد `__init__()` در کلاس های پایتون معنای خاصی دارد. این متد به عنوان مبدأ شیء شناخته می شود زیرا مقادیر اولیه و ویژگی های شما را تعریف و تنظیم می کند. در بخش ویژگی های نمونه بیشتر در مورد این متد یاد خواهید گرفت.

متد دوم دایره به نام `.calculate_area()` است و با استفاده از شعاع خود، مساحت یک دایره خاص را محاسبه خواهد کرد. برای توصیف عملکرد متد، معمول است که نام های متد حاوی فعل باشند، مانند `.calculate` در این مثال، شما از ماژول `math` برای دسترسی به ثابت `pi` استفاده کرده اید زیرا در آن ماژول تعریف شده است.

توجه: در پایتون، اولین آرگومان بسیاری از متد ها `self` است. این آرگومان یک ارجاع به شئ فعلی را نگه می دارد تا بتوانید آن را در داخل کلاس استفاده کنید. در بخش متد های نمونه با `self` بیشتر درباره این آرگومان یاد خواهید گرفت.

خب! شما اولین کلاس خود را نوشتید. حالا چگونه می توانید از این کلاس در کد خود استفاده کنید تا چندین دایره واقعی را نشان دهید؟ خب، باید کلاس خود را نمونه سازی کنید تا شئ های دایره خاص را از آن بسازید.

ایجاد شئ های جدید از یک کلاس در پایتون

عمل ساخت شئ های واقعی از یک کلاس موجود به عنوان نمونه سازی شناخته می شود. با هر نمونه سازی، یک شئ جدید از کلاس هدف ایجاد می شود. برای آغاز، با اجرای کد زیر در جلسات REPL پایتون، چندین نمونه Circle بسازید:

In [12]:

```
circle_1 = Circle(42)
circle_2 = Circle(7)
```

برای ایجاد یک شیء از یک کلاس پایتون مانند دایره، باید با جفت پرانتز و مجموعه مناسبی از آرگومان ها، سازنده کلاس `Circle()` را فراخوانی کنید. چه آرگومان هایی؟ در پایتون، سازنده کلاس همان آرگومان های `.__init__()` را قبول می کند. در این مثال، کلاس دایره آرگومان شعاع را انتظار دارد.

فراخوانی سازنده کلاس با مقادیر آرگومان مختلف به شما اجازه می دهد تا اشیاء یا نمونه های مختلفی از کلاس هدف ایجاد کنید. در مثال بالا، circle_1 و circle_2 نمونه های جداگانه از `Circle` هستند. به عبارت دیگر، آنها دو دایره متفاوت و ملموس هستند، همانطور که می توانید از خروجی کد استنباط کنید.

عالی! شما قبلاً یاد گرفتید چگونه با فراخوانی سازنده کلاس مورد نظر با آرگومان های لازم، اشیاء یک کلاس موجود را ایجاد کنید. حال، چگونه می توانید به ویژگی ها و متد های یک کلاس داده شده دسترسی پیدا کنید؟ این چیزی است که در بخش بعدی خواهید آموخت.

دسترسی به ویژگی ها و متدها

در پایتون، شما می توانید با استفاده از نشانه نقطه با عامل نقطه به ویژگی ها و متدهای یک شیء دسترسی پیدا کنید. قطعه کد زیر نحو لازم را نشان می دهد:

- obj.attribute_name

توجه کنید که نقطه (.) در این نحو به طور کلی به معنای دسترسی به ویژگی یا متد زیر این شیء است. خط اول مقدار ذخیره شده در ویژگی هدف را برمی گرداند، در حالی که خط دوم به روش هدف دسترسی پیدا می کند تا بتوانید آن را فراخوانی کنید.

توجه: به یاد داشته باشید که برای فراخوانی یک تابع یا متد، باید از جفت پرانتز و سری آرگومان ها استفاده کنید، اگر قابل اجرا باشد.

حال به شیء های دایره خود بازگردید و کد زیر را اجرا کنید:

In [13]:
```
1 circle_1.radius
```

Out[13]:

42

In [14]:
```
1 circle_2.radius
```

Out[14]:

7

In [15]:
```
1 circle_1.calculate_area()
```

Out[15]:

5541.77

In [16]:
```
1 circle_2.calculate_area()
```

Out[16]:

153.94

In [ ]:

```
1
```

در چند خط اول پس از نمونه سازی، شما به ویژگی `.radius` در شیء circle_1 خود دسترسی پیدا می کنید. سپس با فراخوانی متد `.calculate_area()` مساحت دایره را محاسبه می کنید. در جفت دوم از بیانیه ها، شما همان کار را انجام می دهید اما در شیء circle_2.

شما همچنین می توانید از نقطه نوتاسیون و یک بیانیه اختصاص به تغییر مقدار فعلی یک ویژگی استفاده کنید:

In [17]:

```
1 circle_1.radius = 100
2 circle_1.radius
```

Out[17]:

```
100
```

In [18]:

```
1 circle_1.calculate_area()
```

Out[18]:

```
31415.93
```

حال شعاع circle_1 به طور کامل متفاوت است. هنگام فراخوانی `.calculate_area()` ، نتیجه به طور فوری این تغییر را نشان می دهد. شما وضعیت داخلی یا داده های شیء را تغییر داده اید که به طور معمول بر رفتارهای یا متد های آن تأثیر می گذارد.

مثال دیگر:

In [19]:

```
1 class Dog:
2     def __init__(self, breed):
3         self.breed = breed
4 
5 sam = Dog(breed = 'Lab')
6 frank = Dog(breed = 'Huski')
```

In [20]:

```
1 sam.breed
```

Out[20]:

```
'Lab'
```

In [21]:

```
frank.breed
```

Out[21]:

```
'Huski'
```

توجه کنید که ما پس از نام نژاد هیچ پرانتزی نداریم؛ این به این دلیل است که یک ویژگی است و هیچ آرگومانی را نمی گیرد.

در پایتون همچنین *ویژگی های شیء کلاس* وجود دارد. این ویژگی های شیء کلاس برای هر نمونه از کلاس یکسان است. به عنوان مثال، ما می توانیم ویژگی *گونه* را برای کلاس سگ ایجاد کنیم. سگ ها، بدون توجه به نژاد، نام یا ویژگی های دیگر خود، همیشه پستانداران خواهند بود. ما این منطق را به صورت زیر اعمال می کنیم:

In [35]:

```
class Dogs:
    species = 'mammal'
    def __init__(self, breed, name):
        self.breed = breed
        self.name = name
        #print(Dog.species)
        print(type(self).species)

sam = Dog(breed = 'Lab', name = 'sam')
frank = Dog(breed = 'Huski', name = 'frank')
```

```
mammal
mammal
```

In [23]:

```
sam.name
```

Out[23]:

```
'sam'
```

In [24]:

```
sam.breed
```

Out[24]:

```
'Lab'
```

توجه کنید که ویژگی شیء کلاس خارج از هر متد در کلاس تعریف می شود. همچنین به طور معمول، آنها را قبل از init قرار می دهیم.

In [25]:

```
sam.species
```

Out[25]:

```
'mammal'
```

In [26]:

```
1 frank.species
```

Out[26]:

```
'mammal'
```

In [27]:

```
1 Dog.species
```

Out[27]:

```
'mammal'
```

In [28]:

```
1 Dog.breed
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
Cell In[28], line 1
----> 1 Dog.breed

AttributeError: type object 'Dog' has no attribute 'breed'
```

In [38]:

```
1  class Dog:
2      species = 'mammal'
3      def __init__(self, breed, name):
4          self.breed = breed
5          self.name = name
6          #print(Dog.species)
7          print(type(self).species)
8      def bark(self):
9          print('Wooof!')
10
11 sam = Dog(breed = 'Lab', name = 'sam')
12 frank = Dog(breed = 'Huski', name = 'frank')
```

```
mammal
mammal
```

In [39]:

```
1 sam.bark()
```

```
Wooof!
```

وراثت

ارث بری یک روش برای ایجاد کلاس های جدید با استفاده از کلاس هایی است که قبلاً تعریف شده اند. کلاس های تازه شکل داده شده کلاس های مشتق شده نامیده می شوند و کلاس هایی که از آنها به دست می آوریم، کلاس های پایه نامیده می شوند. مزایای مهم ارث بری استفاده مجدد از کد و کاهش پیچیدگی برنامه است. کلاس های مشتق شده (فرزندان) عملکرد کلاس های پایه (پدران) را بازنویسی یا گسترش می دهند.

بیایید با یک مثال به کلاس سگ بپردازیم:

In [55]:

```
class Animal:
    def __init__(self):
        print('Animal Created')

    def whoAmI(self):
        print('Animal')

    def eat(self):
        print('Eating')

class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Dog Created")
    def whoAmI(self):
        print('Dog')

    def bark(self):
        print('Woof!')

class Lion(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Lion Created")
```

In [56]:

```
d = Dog()
```

```
Animal Created
Dog Created
```

In [57]:

```
d.whoAmI()
```

```
Dog
```

In [58]:

```
d.eat()
```

```
Eating
```

In [59]:

```
1 d.bark()
```

```
Woof!
```

In [60]:

```
1 a = Animal()
```

```
Animal Created
```

In [46]:

```
1 a.bark()
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
Cell In[46], line 1
----> 1 a.bark()

AttributeError: 'Animal' object has no attribute 'bark'
```

In [54]:

```
1 l = Lion()
2
3 l.whoAmI()
```

```
Animal Created
Lion Created
Animal
```

در این مثال، دو کلاس داریم: Animal و Dog. Animal کلاس پایه است، Dog کلاس مشتق شده است.

کلاس مشتق شده متدهای کلاس پایه را به ارث می برد.

- این با روش eat() نشان داده شده است.

کلاس مشتق شده رفتار موجود کلاس پایه را تغییر می دهد.

- با روش whoAmI() نشان داده شده است.

در نهایت، کلاس مشتق شده عملکرد کلاس پایه را گسترش می دهد، با تعریف یک روش جدید bark().

چندریختی Polymorphism

ما یاد گرفتیم که در حالی که توابع می توانند با آرگومان های مختلف فراخوانی شوند، متدها متعلق به شی هایی هستند که بر روی آنها عمل می کنند. در پایتون، *polymorphism* به روشی اشاره دارد که کلاس های شی مختلف می توانند نام یکسان را به اشتراک بگذارند و این روش ها می توانند از یک مکان فراخوانی شوند، حتی اگر اشیاء مختلفی از طریق آنها منتقل شوند. بهترین راه برای توضیح این مسئله، با یک مثال است:

In [62]:

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

    def speak(self):
        return self.name + ' says WOOOF!'

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

    def speak(self):
        return self.name + ' says Meow'
niko = Dog('Niko')
felix = Cat('Felix')

print(niko.speak())
print(felix.speak())
```

```
Niko says WOOOF!
Felix says Meow
```

در اینجا یک کلاس سگ و یک کلاس گربه داریم، و هر کدام دارای یک روش `.speak()` هستند. هنگام فراخوانی، روش `.speak()` هر شیء نتیجه ای منحصر به فرد برای شیء خود باز می گرداند.

چندین روش مختلف برای نمایش polymorphism وجود دارد. اول، با یک حلقه for:

In [63]:

```
for pet in [niko, felix]:
    print(pet.speak())
```

```
Niko says WOOOF!
Felix says Meow
```

یا از طریق یک تابع این کا را انجام دهیم:

In [65]:

```
def pet_speak(pet):
    print(pet.speak())

pet_speak(niko)
pet_speak(felix)
```

```
Niko says WOOOF!
Felix says Meow
```

در هر دو مورد، ما توانستیم انواع شی مختلف را منتقل کنیم و نتایج منحصر به شیء را از همان مکان بدست آوریم.

یک شیوه کاربردی تر استفاده از کلاس های انتزاعی (abstract class) و وراثت است. یک کلاس انتزاعی یک کلاس است که هرگز انتظار ندارد نمونه سازی شود. به عنوان مثال، هرگز شیء Animal را نخواهیم داشت، فقط شیء های Dog و Cat را خواهیم داشت، اگرچه سگ ها و گربه ها از حیوانات به دست می آیند:

In [66]:

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

    def speak(self):
        raise NotImplementedError('Subclass must implement speak() method!')

class Dog(Animal):
    def speak(self): #override
        return self.name + ' says WOOOF!'

class Cat(Animal):
    def speak(self):
        return self.name + ' says Meow'

fido = Dog('Fido')
isis = Cat('Isis')

print(fido.speak())
print(isis.speak())
```

```
Fido says WOOOF!
Isis says Meow
```

مثال های واقعی از polymorphism شامل:

- باز کردن انواع مختلف فایل - ابزارهای مختلف برای نمایش فایل های pdf، Word و Excel لازم است
- اضافه کردن اشیاء مختلف - عملگر + عملیات حسابی و ادغام را انجام می دهد

متد های ویژه

در نهایت بیایید به متد های ویژه بپردازیم. کلاس ها در Python می توانند برخی از عملیات ها را با نام های متد های ویژه پیاده سازی کنند. این متد ها در واقع به صورت مستقیم فراخوانی نمی شوند، بلکه با دستور زبان خاص Python فراخوانی می شوند. به عنوان مثال، بیایید یک کلاس Book ایجاد کنیم:

In [81]:

```python
class Book:
    def __init__(self, title, author, pages):
        print('A book is created!')
        self.title = title
        self.author = author
        self.pages = pages

    def __str__(self):
        return f"Title: {self.title}, author: {self.author}, pages: {self.pages}"

    def __len__(self):
        return self.pages
    def __del__(self):
        print("A book is destroyed")
```

In [82]:

```
book = Book('Python Rocks!', 'J P', 209)
```

```
A book is created!
```

In [83]:

```
print(book)
```

```
Title: Python Rocks!, author: J P, pages: 209
```

In [84]:

```
len(book)
```

Out[84]:

```
209
```

In [85]:

```
del book
```

```
A book is destroyed
```

متد های ویژه **len**__ ،__str__ ،__init و **del**

این متد های ویژه با استفاده از زیرخط تعریف شده اند. آنها به ما اجازه می دهند تا از توابع خاص Python بر روی اشیاء ایجاد شده از طریق کلاس خود استفاده کنیم.

عالی! پس از این سخنرانی، شما باید درک پایه ای از نحوه ایجاد شی های خود با کلاس در Python داشته باشید. در پروژه مرحله بعدی خود به طور فراوان از آن استفاده خواهید کرد!