دوره جامع برنامه نویسی پایتون

سر فصل مطالب :

1. معرفی و نصب ابزار ها
2. انواع داده ها، تعریف متغیر و اصول نامگذاری، ساختارهای داده (مفاهیم پایه)
3. عملگرهای مقایسه ای و منطقی
4. دستوالعمل ها
5. متد ها و توابع
6. برنامه نویسی شیء گرا
7. ماژول ها و پکیج ها
8. خطاها و مدیریت استثناء ها
9. Decorators and Generators
10. ماژول های پیشرفته
11. مطالب تکمیلی و پیشرفته
12. پروژه ها

فصل ۶

برنامه نویسی شیء گرا

- مقدمه
- class و attributes
- ساخت متد در کلاس
- وراثت
- چندریختی
- متدهای خاص

مقدمه

- برنامه نویسی شیءگرا یا Object Oriented Programming یا به اختصار OOP به برنامه نویس این قابلیت را میدهد که object خود به همراه ویژگی ها و متدهای خاص خود را بسازد
- همانطور که تاکنون با ساخت داده های از نوع رشته، لیست، دیکشنری و یا دیگر objectها میتوانستیم با صدا زدن متدهای خاص آنها طبق رابطه زیر از قابلیت های این متد ها استفاده کنیم

```
object.method(arg1, arg2, arg3, ...)
```

- به طور کلی متدها، توابعی هستند که برای یک شیء تعریف شده اند و طبق خصوصیات آن شیء بعد از فراخوانی نتیجه ای را برمیگردانند

مقدمه

- توابع کدهای مارا repeatable و از نوشتن دستورات تکراری جلوگیری میکردند، oop برای مدیریت بیشتر و قابلیت استفاده مجدد بیشتر در کدها و برنامه های بزرگ، راه حل بهتری نسبت به توابع است
- تکرار وظایف و اشیاء در برنامه نویسی به روش شیءگرا بسیار کاربردی و بااهمیت است
- Object یا شیء چیست؟ در پایتون همه چیز یک object است

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

Class و Attributes

- میتوان با استفاده از کلمه کلیدی class اشیاء(object) خود را بسازیم
- کلاس ماهیت یک شیء را تعریف می کند
- از کلاس ها می توانیم نمونه هایی(instances) بسازیم
- یک نمونه یک شیء خاص است که از یک کلاس خاص ایجاد می شود. به عنوان مثال، ساخت یک لیست خاص از کلاس لیست
- اصول نامگذاری class مانند نامگذاری متغیر و تابع است با این تفاوت که در نامگذاری کلاس از _ استفاده نمیشود و از نامگذاری PascalCase استفاده میشود

```
camelCase
snake_case
PascalCase
```

class and instance

```
class Sample:
    pass

# Instance of Sample
x = Sample()

print(type(x))
```

Class و Attributes

- Attribute یک مشخصه یا ویژگی از object است
- Method عملیاتیست که میتوان با شیء انجام داد
- برای مثال کلاس Car را در نظر بگیرید:
- یک ماشین چه ویژگی هایی دارد؟
- یک ماشین چه کارهایی میتواند انجام دهد؟
- آیا ویژگی ها و عملیات ها با هم در ارتباط هستند؟
- سینتکس تعریف یک attribute به صورت زیر است:

```
self.attribute_name = something
```

Class و Attributes

- یک متد خاص با نام __init__() در ابتدای کلاس تعریف میشود که برای مقداردهی اولیه ویژگی های یک شیء استفاده میشود
- نماد __ خوانده میشود dunder که کوتاه شده Double Underscore است و در متدهای خاص یا مجیک متدها استفاده میشود
- تابع __init__() یک تابع سازنده(Constructor) است و هر زمان یک شیء جدید از کلاس نمونه سازی شود این تابع فراخوانی میشود
- توابع سازنده نوعی از توابع هستند که در برنامه نویسی شیءگرا استفاده میشوند و معمولا برای مقداردهی اولیه متغیر ها به کار میروند

```
self.attribute_name = something
```

- سینتکس تعریف کلاس به صورت زیر است:

```
class ClassName():

    def __init__(self,param1,param2,...):
        self.param1 = param1
        self.param2 = param2

    def some_method(self):
        #do something
        print(self.param1)
```

Attributes و Class

class

```
class Person:

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

    def hello(self):
        print("Hello " + self.name)
        print(f"you're {self.age} years old")

student = Person("Ali", 25)

student.hello()
student.name
```

ساخت متد در کلاس

- متدها توابعی هستند که در داخل بدنه یک کلاس تعریف می شوند
- متدها برای انجام عملیات با ویژگی های اشیاء ما استفاده می شوند. متدها یک مفهوم کلیدی از پارادایم OOP هستند. آنها برای تقسیم مسئولیت ها در برنامه نویسی، به ویژه در برنامه های کاربردی بزرگ ضروری هستند.
- اساساً می‌توانی متدها را به‌عنوان توابعی در نظر گرفت که روی یک شیء عمل می‌کنند
- برای دسترسی به خود شیء در این توابع لازم و ضروری است همه توابعی که به عنوان متد تعریف میشوند آرگمان ورودی با نام self داده شود
- Self به خود شیء اشاره دارد و با استفاده از آن میتوان به ویژگی های تعریف شده در شیء دسترسی داشت و یا حتی آن ها را به صورت داینامیک تغییر داد

ساخت متد در کلاس


```
class Circle:
    pi = 3.14
    def __init__(self, radius=1):
        self.radius = radius

    def circumference(self, r):
        self.radius = r
        print(f"Circumference with radius {r}: {2*self.pi*r}")

    def area(self, r):
        self.radius = r
        print(f"Area with radius {r}: {self.pi*r**2}")



c = Circle()
c.circumference(3)
c.area(3)
```

ساخت متد در کلاس

```
class Circle:
    pi = 3.14
    def __init__(self, radius=1):
        self.radius = radius
        self.area = Circle.pi*radius**2
        self.circumference = 2*Circle.pi*radius

    def showInfo(self):
        print(f"Radius = {self.radius}")
        print(f"Area = {self.area}")
        print(f"circumference = {self.circumference}")



c = Circle(3)
c.showInfo()
c.circumference
c.area
```