دنیای جدیدی از عملکردها

در درسنامه قبلی، با مفهوم **Dunder Methods** آشنا شدیم و به بررسی تعدادی از آن‌ها پرداختیم. در این بخش، به معرفی تعدادی از **پرکاربردترین متدهای جادویی** خواهیم پرداخت که می‌توانند عملکردهای پیچیده و جذاب‌تری را به برنامه‌های شما اضافه کنند.

متد add: جمع اشیاء با عملگر +

در پایتون، عملگر + می‌تواند عملکردهای مختلفی بر اساس نوع داده‌ها داشته باشد. به‌طور مثال، اگر دو عدد را جمع کنیم، حاصل جمع ریاضی آن‌ها نمایش داده می‌شود. اگر دو رشته را به هم بچسبانیم یا دو لیست را با هم ترکیب کنیم، این عملگر عملکردهای مختلفی از خود نشان می‌دهد. این تفاوت‌ها به دلیل پیاده‌سازی‌های خاص متد `__add__` است.

برای مثال، در کلاس `FootballTeam` ، از این متد برای جمع تعداد جام‌های قهرمانی استفاده می‌کنیم:

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

    def __add__(self, other_object):
        return self.cups_number + other_object.cups_number
```

با استفاده از متد `__add__` ، می‌توانیم عملگر + را برای جمع کردن جام‌های دو تیم فوتبال استفاده کنیم:

```
bayern_munich = FootballTeam('FC Bayern Munich', 6)
barcelona = FootballTeam('FC Barcelona', 5)
print(bayern_munich + barcelona)  # 11
```

این روش به شما این امکان را می‌دهد که عملکردهایی مشابه جمع را با استفاده از عملگرهای مختلف برای انواع داده‌ها تعریف کنید:

نام عملگر	عبارت	داندر متد
جمع	a + b	**add**
تفریق	a - b	**sub**
ضرب	a * b	**mul**
تقسیم اعشاری	a / b	**truediv**
تقسیم صحیح	a // b	**floordiv**
باقی‌مانده	a % b	**mod**
توان	a ** b	**pow**
شیفت چپ	a << b	**lshift**
شیفت راست	a >> b	**rshift**
AND	a & b	**and**
OR	a	b
XOR	a ^ b	**xor**
NOT	~a	**invert**

متد len: محاسبه طول اشیاء

شاید بارها از تابع `len()` برای محاسبه طول لیست‌ها یا دیگر اشیاء استفاده کرده‌اید. این تابع، بر اساس نحوه پیاده‌سازی متد `__len__` ، می‌تواند به شیوه‌های مختلف عمل کند. به‌عنوان مثال، اگر بخواهیم هنگام استفاده از `len()` برای یک لیست تعداد عناصر یکتا را حساب کنیم، می‌توانیم متد `__len__` را به این شکل بازنویسی کنیم:

```
class CustomList(list):
    def __len__(self):
```

```
        return len(set(self))  # We are only counting unique elements
```

حالا اگر لیستی با تکرار عناصر داشته باشیم:

```
my_list = CustomList([1, 1, 1, 2, 3, 3])
print(len(my_list))  # 3
```

با این روش، تابع `len()` فقط تعداد عناصر یکتا را به شما نشان می‌دهد.

متدهای iter و next: پیمایش در اشیاء

در پایتون، برای پیمایش در اشیاء، مانند لیست‌ها، دیکشنری‌ها و غیره، از دستور `for` استفاده می‌کنیم. اما این دستور در پشت‌صحنه از متدهای `__iter__()` و `__next__()` برای پیمایش استفاده می‌کند.

متد `__iter__()` باید یک پیمایش‌گر (Iterator) را بازگرداند و متد `__next__()` باید هر بار که فراخوانی شود، یک عنصر از مجموعه را بازگرداند. وقتی تمام عناصر پیمایش شدند، باید خطای `StopIteration` را ایجاد کند.

مثال زیر نشان می‌دهد که چگونه می‌توانیم کلاس خود را برای پیمایش پیاده‌سازی کنیم:

```
import random

class RandomShuffle:
    def __init__(self, data):
        self.data = data

    def __iter__(self):
        return self

    def __next__(self):
        if len(self.data) == 0:
            raise StopIteration

        index = random.randint(0, len(self.data)-1)
        return self.data.pop(index)
```

حالا اگر از این کلاس استفاده کنیم، می‌توانیم مجموعه را به‌طور تصادفی پیمایش کنیم:

```
number = [1, 2, 3, 4, 5, 6]
random_shuffle = RandomShuffle(number)

for item in random_shuffle:
    print(item)
```

خروجی این کد می‌تواند به شکل زیر باشد:

```
4
3
1
6
2
5
```

در این بخش از درسنامه، برخی از متدهای جادویی پایتون که به شما این امکان را می‌دهند تا اشیاء خود را به‌طور سفارشی‌تری تعریف کنید، معرفی شدند. این متدها شامل متدهای ریاضی مانند `__add__` و `__sub__` ، متد پیمایش مانند `__iter__` و `__next__` ، و متد `__len__` برای محاسبه طول اشیاء بودند. با استفاده از این متدها می‌توانید تجربه برنامه‌نویسی پایتون خود را به سطح جدیدی ارتقا دهید.