ارث‌بری در کلاس‌ها

ارث‌بری در کلاس‌ها (Inheritance)

مقدمه

ارث‌بری یکی از مفاهیم بنیادین در برنامه‌نویسی شیءگرا است که به یک کلاس اجازه می‌دهد ویژگی‌ها و رفتارهای کلاس دیگری را به ارث ببرد. در این ساختار، کلاس دریافت‌کننده ویژگی‌ها را **کلاس فرزند** و کلاس ارائه‌دهنده ویژگی‌ها را **کلاس والد** می‌نامند.

نحوه تعریف ارث‌بری در پایتون

در پایتون، برای ارث‌بری کافی است هنگام تعریف کلاس فرزند، نام کلاس والد را در پرانتز قرار دهیم. به عنوان مثال:

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

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

در این مثال، کلاس `Child` از کلاس `Parent` ارث‌بری کرده و تمامی ویژگی‌های آن را دریافت می‌کند. همچنین، اگر برای کلاسی والد مشخص نکنیم، به طور پیش‌فرض از کلاس `object` ارث‌بری می‌کند:

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

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

مثال: ارث‌بری برای کلاس‌های مرتبط با مشاغل

فرض کنید می‌خواهیم کلاس‌هایی برای مشاغل مختلف مانند کارگر، معلم و مهندس تعریف کنیم. چون این مشاغل ویژگی‌های مشترکی دارند (مانند نام و سن)، بهتر است ابتدا کلاس `Person` را ایجاد کنیم و کلاس‌های دیگر از آن ارث‌بری کنند:

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

class Worker(Person):
    pass

class Teacher(Person):
    pass

class Engineer(Person):
    pass
```

استفاده از `super()` در ارث‌بری

برای استفاده از متدهای کلاس والد در کلاس فرزند، می‌توانیم از تابع `super()` استفاده کنیم. این روش موجب افزایش خوانایی کد و کاهش مشکلات احتمالی در ارث‌بری چندگانه می‌شود.

```
class Person:
    def __init__(self):
        print('Initialize from class Person')

class Engineer(Person):
    def __init__(self):
        print('Initialize from class Engineer')
        super().__init__()

eng = Engineer()
```

خروجی:

```
Initialize from class Engineer
Initialize from class Person
```

همچنین در صورتی که متدی در کلاس فرزند بازنویسی شده باشد اما بخواهیم نسخه کلاس والد را نیز اجرا کنیم، می‌توان از `super()` استفاده کرد:

```
class Person:
    def output(self):
        print('Output from class Person')

class Engineer(Person):
    def output(self):
        print('Output from class Engineer')
        super().output()

sepehr = Engineer()
sepehr.output()
```

خروجی:

```
Output from class Engineer
Output from class Person
```

توابع کاربردی در ارثبری

تابع `isinstance()`

این تابع بررسی می‌کند که آیا یک شیء از نوع کلاس مشخصی هست یا خیر.

```
print(isinstance(5, int))  # True
print(isinstance(5, str))  # False
```

```
print(isinstance("Hello", (float, str, int, list, dict, tuple)))  # True
```

تابع issubclass()

این تابع بررسی می‌کند که آیا یک کلاس، زیرکلاس کلاس دیگری است یا خیر.

```
class Person:
    pass

class Worker(Person):
    pass

class Animal:
    pass

print(issubclass(Worker, Person))  # True
print(issubclass(Animal, Person))  # False
```

با استفاده از این مفاهیم، می‌توان برنامه‌هایی منظم‌تر، انعطاف‌پذیرتر و قابل توسعه‌تر طراحی کرد.

ارث‌بری چندگانه در پایتون

یکی از ویژگی‌های منحصر به فرد پایتون نسبت به بسیاری از زبان‌های دیگر، توانایی ارث‌بری چندگانه است. در این نوع ارث‌بری، یک کلاس می‌تواند همزمان از چند کلاس دیگر ویژگی‌ها و متدهای خود را به ارث ببرد. برای انجام این کار، کافیست تا هنگام تعریف کلاس جدید، نام کلاس‌های والد را به ترتیب بنویسید.

مثال اولیه:

```
class Rooster:
    pass

class Hen:
    pass


```

```
class Chick(Rooster, Hen):
    pass
```

در این مثال، کلاس `Chick` از کلاس‌های `Rooster` و `Hen` ارث می‌برد و می‌تواند از ویژگی‌ها و متدهای هر دو استفاده کند.

برخورد با ویژگی‌های مشترک

اگر کلاس‌هایی که از آن‌ها ارث می‌بریم، ویژگی‌های مشابه یا یکسانی داشته باشند، ممکن است این سوال پیش بیاید که وقتی ویژگی مشترک صدا زده می‌شود، از کدام کلاس استفاده خواهد شد؟ پاسخ این سوال ساده است: ویژگی از کلاسی که اول در لیست ارث‌بری آمده است، انتخاب می‌شود.

مثال:

```
class Rooster:
    def name(self):
        print('In Rooster class.')

class Hen:
    def name(self):
        print('In Hen class.')
    def hello(self):
        print('Hello my baby!')

class Chick(Rooster, Hen):
    pass

x = Chick()
x.name()
x.hello()
```

در این مثال، چون کلاس `Chick` ابتدا از کلاس `Rooster` ارث‌بری کرده است، وقتی متد `name` فراخوانی می‌شود، ابتدا به سراغ کلاس `Rooster` می‌رود و از آن استفاده می‌کند. بعد از آن، متد `hello` از کلاس `Hen` اجرا می‌شود زیرا در کلاس `Rooster` چنین متدی وجود ندارد.

استفاده از شئ `super` در ارث‌بری چندگانه

در ارث‌بری چندگانه، گاهی اوقات ممکن است از چندین پدر یک متد مشابه فراخوانی شود و این می‌تواند منجر به تکرار عملکردها شود. برای حل این مشکل، می‌توانیم از شئ `super()` استفاده کنیم تا تنها یک بار از هر کلاس والد استفاده شود.

مثال با استفاده از `super` :

```
class Shape:
    def __init__(self):
        print('Initialize from class Shape')

class Rectangle(Shape):
    def __init__(self):
        print('Initialize from class Rectangle')
        super().__init__()

class Rhombus(Shape):
    def __init__(self):
        print('Initialize from class Rhombus')
        super().__init__()

class Square(Rectangle, Rhombus):
    def __init__(self):
        super().__init__()

squ = Square()
```

خروجی:

```
Initialize from class Rectangle
Initialize from class Rhombus
Initialize from class Shape
```

در این مثال، استفاده از `super()` از تکرار فراخوانی متد `__init__` در کلاس `Shape` جلوگیری می‌کند و باعث می‌شود که فقط یک بار این متد صدا زده شود.