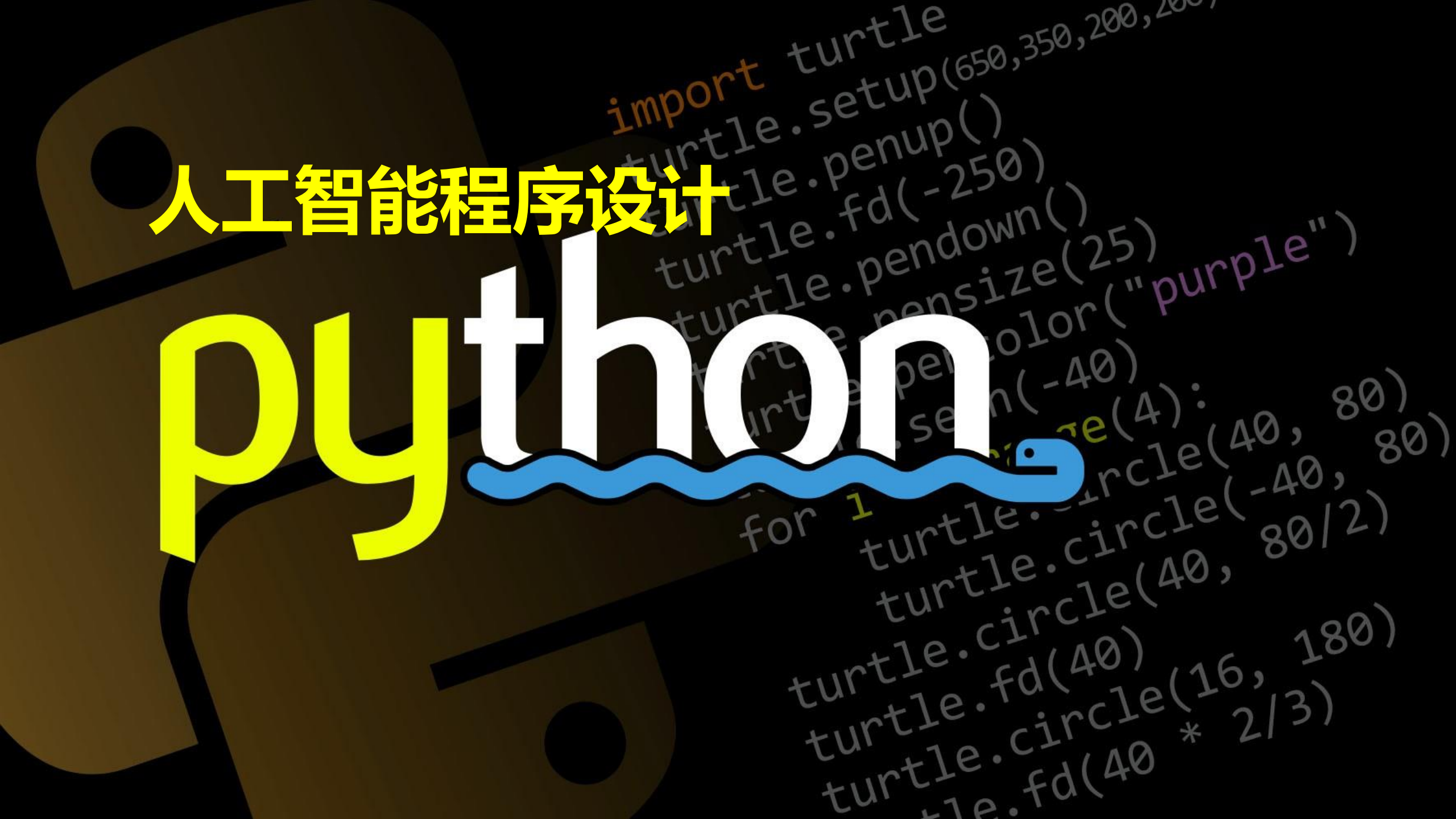
人工智能程序设计


python
import turtle
turtle.setup(650,350,200,200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for 1 in range(4):
turtle.circle(40, 80)
turtle.circle(-40, 80)
turtle.fd(40)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)

人工智能程序设计

6.2 属性与方法

北京石油化工学院 人工智能研究院

刘 强

6.2 属性与方法

类的属性和方法是面向对象编程的基础要素。属性存储对象的状态信息，方法定义对象的行为。

理解不同类型的属性和方法对于编写高质量的面向对象代码至关重要。

6.2.1 实例属性和类属性

实例属性

实例属性属于特定的对象实例，每个对象都有自己的实例属性副本。

```
class Product:
    def __init__(self, name, price):
        self.name = name      # 实例属性
        self.price = price    # 实例属性
product1 = Product("笔记本电脑", 5999)
product2 = Product("无线鼠标", 89)
print(product1.name)   # 输出：笔记本电脑
print(product2.name)   # 输出：无线鼠标
```

6.2.1 实例属性和类属性

类属性

类属性属于整个类，所有实例共享同一个类属性。

```
class Employee:
    company = "科技有限公司"    # 类属性
    def __init__(self, name, department):
        self.name = name
        self.department = department
## 所有实例共享类属性
emp1 = Employee("王明", "技术部")
emp2 = Employee("李华", "销售部")
print(emp1.company)  # 输出：科技有限公司
print(emp2.company)  # 输出：科技有限公司
print(Employee.company)   # 输出：科技有限公司
```

6.2.1 实例属性和类属性

属性访问优先级

实例属性优先于类属性：

```
class Counter:
    count = 0  # 类属性
    def __init__(self):
        self.count = 1  # 实例属性
counter = Counter()
print(counter.count)    # 输出：1 (实例属性)
print(Counter.count)    # 输出：0 (类属性)
```

6.2.2 实例方法和类方法

实例方法

实例方法是最常见的方法类型，需要通过对象实例调用。

```
class Calculator:
    def add(self, a, b):        # 实例方法
        return a + b
    def multiply(self, a, b):   # 实例方法
        return a * b
calc = Calculator()
result = calc.add(3, 5)  # 通过实例调用
print(result)  # 输出：8
```

6.2.2 实例方法和类方法

类方法

类方法使用@classmethod装饰器，第一个参数是cls（代表类本身）。装饰器是一种修改函数行为的特殊语法，它使用@符号标记，可以在不改变函数代码的情况下为函数添加额外的功能。

```
class Library:
    location = "市中心图书馆"

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

    @classmethod
    def get_location(cls):        # 类方法
        return cls.location

## 通过类直接调用
print(Library.get_location())  # 输出：市中心图书馆
```

6.2.3 特殊方法（魔术方法）

特殊方法以双下划线开头和结尾，用于定义对象的特殊行为。

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):          # 字符串表示
        return f"Point({self.x}, {self.y})"

    def __eq__(self, other):    # 相等比较
        return self.x == other.x and self.y == other.y

    def __add__(self, other):   # 加法运算
        return Point(self.x + other.x, self.y + other.y)
```

```
## 使用特殊方法
p1 = Point(1, 2)
p2 = Point(3, 4)

print(p1)          # 输出：Point(1, 2)
print(p1 == p2)    # 输出：False
print(p1 + p2)     # 输出：Point(4, 6)
```

6.2.3 特殊方法（魔术方法）

重要的特殊方法

- __init__：构造方法
- __str__：字符串表示（用户友好）
- __repr__：对象表示（开发者友好）
- __len__：支持len()函数
- __eq__：相等比较
- __lt__：小于比较

实践练习

练习 6.2.1：音乐播放器类

创建一个MusicPlayer类，使用类属性记录播放历史总数，实现播放、暂停和切换歌曲的方法。

练习 6.2.2：文件工具类

设计一个FileHelper类，使用静态方法实现文件操作（如获取文件扩展名、计算文件大小格式化显示）（**Ask AI: 什么是静态方法**）

练习 6.2.3：坐标点类

创建一个Point类，实现坐标点的距离计算、中点计算和字符串表示。