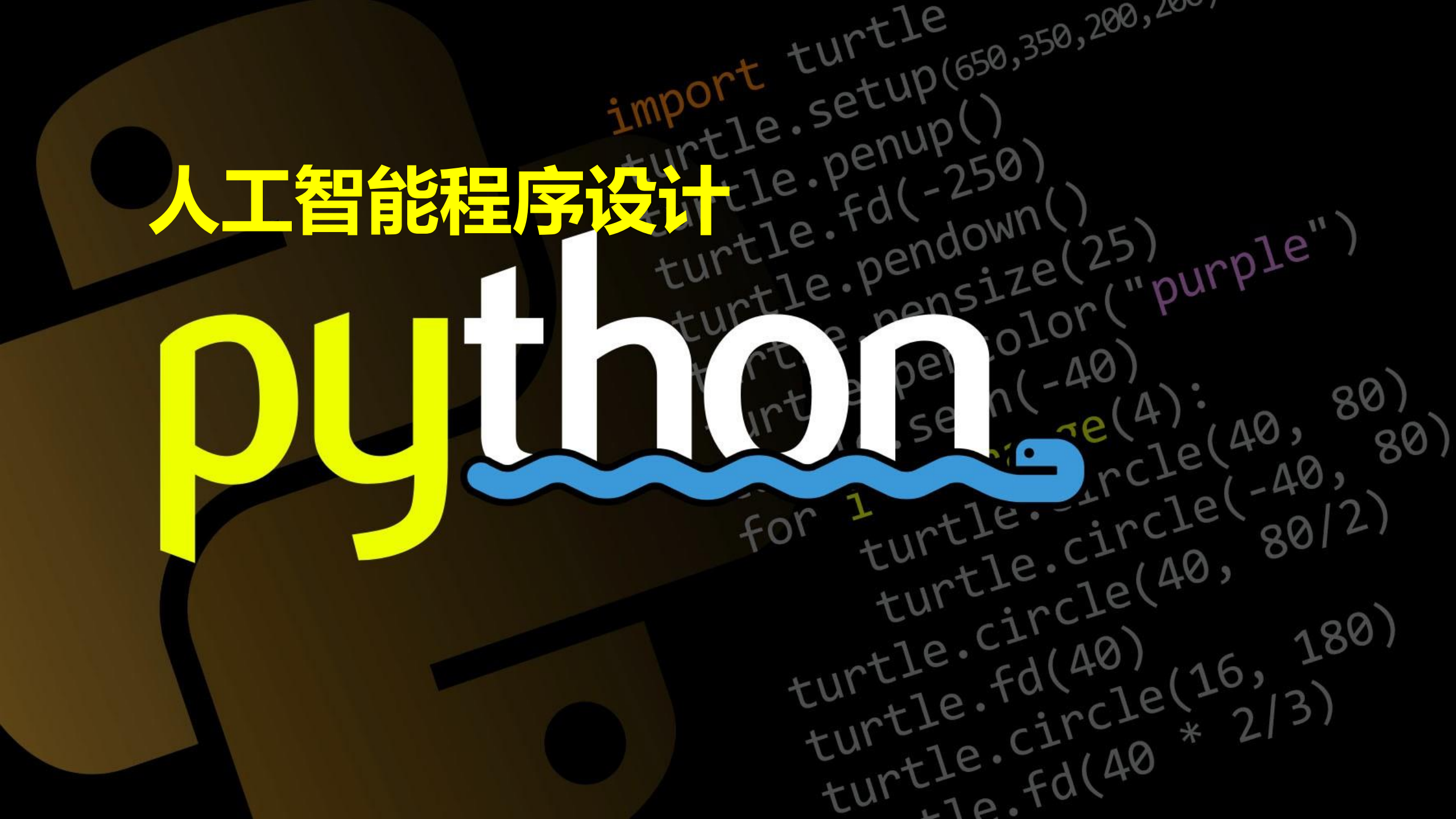
人工智能程序设计


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

第7章 文件与异常

北京石油化工学院 人工智能研究院

刘 强

第7章 文件与异常

Python提供了简洁实用的文件处理功能，支持文本文件、二进制文件等多种格式，能够处理各种文件操作需求。

异常处理是编写健壮程序的关键。在实际应用中，程序经常会遇到各种意外情况：文件不存在、数据格式错误、类型转换失败、权限不足等。通过合理的异常处理，我们可以让程序在面对这些问题时不会崩溃，而是妥善处理错误并给用户清晰的反馈

7.1 文件读写

在编程中，经常需要从文件中读取数据或将数据保存到文件中。Python提供了简单易用的文件操作方法，支持文本文件的读写操作。

文件操作包括三个基本步骤：打开文件、操作文件（读取或写入）、关闭文件。

7.1.1 打开和关闭文件

使用open()函数打开文件

open()函数是Python中打开文件的基本方法。它需要指定文件名和打开模式，返回一个文件对象用于后续操作。

基本语法

```
file = open("文件名", "模式")
```

常用的文件模式：

- 'r'：只读模式（默认）
- 'w'：写入模式（会覆盖原文件）
- 'a'：追加模式（在文件末尾添加内容）

7.1.1 打开和关闭文件

使用with语句（推荐）

with语句是处理文件的最佳实践。它会自动管理文件的打开和关闭，确保文件在使用完毕后被正确关闭，即使程序出现异常也不例外。

```
## 推荐使用with语句，自动关闭文件
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
## 文件会自动关闭
```

with语句的优势：

- 自动关闭文件，即使出现错误
- 代码更简洁、更安全

7.1.2 读取文件

读取整个文件

read()方法可以一次性读取文件的全部内容，适用于小文件。注意指定编码格式以正确处理中文字符。

```
## 读取整个文件内容
with open("data.txt", "r", encoding="utf-8") as file:
    content = file.read()
    print(content)
```

7.1.2 读取文件

逐行读取文件

对于大文件，逐行读取更加高效。Python提供了两种主要方法来实现逐行读取。

```
## 方法1：使用readline()
with open("data.txt", "r", encoding="utf-8") as file:
    line = file.readline()
    while line:
        print(line.strip())  # strip()去除换行符
        line = file.readline()
```

7.1.2 读取文件

逐行读取文件

对于大文件，逐行读取更加高效。Python提供了两种主要方法来实现逐行读取。

```
## 方法2：使用for循环（推荐）
with open("data.txt", "r", encoding="utf-8") as file:
    for line in file:
        print(line.strip())
```

7.1.2 读取文件

读取所有行到列表

readlines()方法将文件的所有行读取到一个列表中，每一行作为列表的一个元素。这种方法适合需要多次处理文件内容的情况。

```
with open("data.txt", "r", encoding="utf-8") as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())
```

7.1.3 写入文件

文本的写入

使用write()方法可以将文本写入文件。写入模式会覆盖原文件内容，而追加模式会在文件末尾添加新内容。

```
## 写入模式（覆盖原文件）
with open("output.txt", "w", encoding="utf-8") as file:
    file.write("Hello, Python!\n")
    file.write("这是第二行\n")

## 追加模式（在文件末尾添加）
with open("output.txt", "a", encoding="utf-8") as file:
    file.write("这是追加的内容\n")
```

7.1.3 写入文件

写入多行内容

当需要写入多行内容时，可以使用writelines()方法。它接受一个字符串列表，将所有字符串依次写入文件。

```
lines = ["第一行\n", "第二行\n", "第三行\n"]
with open("output.txt", "w", encoding="utf-8") as file:
    file.writelines(lines)
```

示例 7.1.1：文件行数统计器

这个函数演示了如何统计文件的行数。

```
def count_lines(filename):
    """统计文件行数"""
    with open(filename, "r", encoding="utf-8") as file:
        lines = file.readlines()
        return len(lines)

## 使用示例
line_count = count_lines("data.txt")
print(f"文件共有 {line_count} 行")
```

示例 7.1.2：简单记事本程序

这个程序展示了文件读写的综合应用，结合了用户交互、文件创建和文件读取等多个知识点。

```
def simple_notepad():
    """简单记事本程序"""
    print("=== 简单记事本 ===")
    print("1. 新建文件")
    print("2. 读取文件")

    choice = input("请选择操作 (1/2): ")

    if choice == "1":
        filename = input("请输入文件名: ")
        content = input("请输入内容: ")

        with open(filename, "w", encoding="utf-8") as file:
            file.write(content)
        print(f"内容已保存到 {filename}")

    elif choice == "2":
        filename = input("请输入要读取的文件名: ")
        with open(filename, "r", encoding="utf-8") as file:
            content = file.read()
            print(f"\n文件内容:\n{content}")

## 运行程序
simple_notepad()
```

7.1.5 文件编码

在处理包含中文字符的文件时，必须指定正确的编码格式。UTF-8是目前最通用的编码格式，建议在所有文件操作中使用。

```
## 指定UTF-8编码
with open("chinese.txt", "r", encoding="utf-8") as file:
    content = file.read()
```

实践练习

练习 7.1.1：文本文件备份

编写程序读取一个文本文件，在每行前面加上行号，然后保存到新文件中。

练习 7.1.2：单词统计器

创建一个程序，读取文本文件并统计其中的单词数量、字符数量和行数，将统计结果保存到另一个文件中。

练习 7.1.3：配置文件管理

编写程序创建和读取简单的配置文件，支持保存用户的姓名、年龄、城市等信息，并能从文件中加载这些配置。