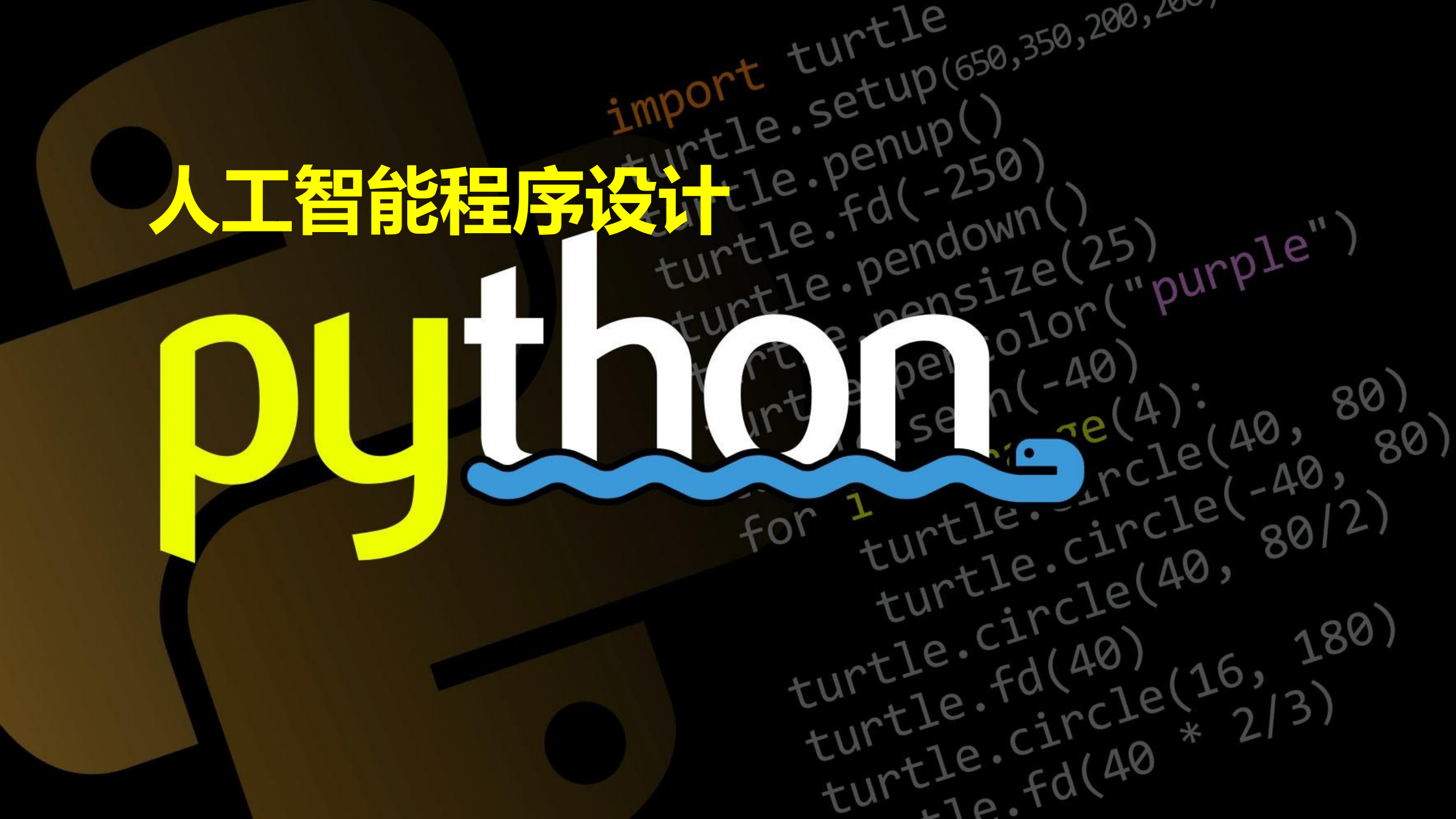
人工智能程序设计


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

7.2 异常处理机制

北京石油化工学院 人工智能研究院

刘 强

7.2 异常处理机制

在程序运行过程中，经常会遇到各种错误情况，如文件不存在、网络连接失败、用户输入错误等。

Python提供了统一的异常处理机制，包括try-except-finally语句、异常层次结构和自定义异常支持，能够妥善处理这些错误，提高程序的健壮性。

异常处理不仅能够防止程序崩溃，还能为用户提供清晰的错误信息，帮助开发者快速定位和解决问题。

7.2.1 什么是异常

异常是程序执行过程中发生的错误事件，它会中断程序的正常执行流程。在Python中，异常是一个对象，包含了错误的类型和相关信息。

常见异常示例

```
print(10 / 0)          # ZeroDivisionError: 除零错误
print(int("abc"))      # ValueError: 值错误
print(list[10])        # NameError: 名称错误
```

7.2.2 try-except语句详解

基本语法

try-except语句是Python异常处理的核心机制，它允许我们捕获和处理特定的异常。

```
try:
    # 可能出现异常的代码
    risky_operation()
except ExceptionType:
    # 处理异常的代码
    handle_exception()
```

7.2.2 try-except语句详解

捕获单个异常

最简单的异常处理方式是捕获特定类型的异常。

```
def safe_division(a, b):
    """安全的除法运算"""
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("错误：不能除以零")
        return None

## 使用示例
print(safe_division(10, 2))    # 输出：5.0
print(safe_division(10, 0))    # 输出：错误：不能除以零，None
```

7.2.2 try-except语句详解

捕获多个异常：可以使用多个except子句来处理不同类型的异常。

```
def safe_input_processing():
    """安全的输入处理"""
    try:
        filename = input("请输入文件名: ")
        with open(filename, "r", encoding="utf-8") as file:
            content = file.read()
            number = int(content.strip())
            result = 100 / number
            print(f"结果: {result}")

    except FileNotFoundError:
        print("错误：文件不存在")
    except ValueError:
        print("错误：文件内容不是有效数字")
    except ZeroDivisionError:
        print("错误：文件中的数字不能为零")
    except Exception as e:
        print(f"发生未知错误: {e}")
```

```
## 使用示例
safe_input_processing()
```

获取异常信息

可以使用as关键字获取异常对象，从中提取详细的错误信息。

```
def detailed_error_handling():
    """详细的错误处理"""
    try:
        numbers = [1, 2, 3]
        index = int(input("请输入索引: "))
        print(f"数字是: {numbers[index]}")

    except (ValueError, IndexError) as e:
        print(f"输入错误: {type(e).__name__}: {e}")
    except Exception as e:
        print(f"未预期的错误: {type(e).__name__}: {e}")
```

异常信息获取说明：

- 使用as关键字将异常对象赋给变量
- type(e).__name__获取异常类型名称
- str(e)或直接使用e获取异常消息

7.2.3 finally子句

finally子句无论是否发生异常都会执行，常用于清理资源。

```
def safe_file_operation():
    """安全的文件操作"""
    try:
        with open("data.txt", "r", encoding="utf-8") as file:
            content = file.read()
            print("文件读取成功")
            return content
    except FileNotFoundError:
        print("文件不存在")
        return None
    finally:
        print("操作完成") # 无论成功失败都会执行

## 使用示例
result = safe_file_operation()
```

7.2.4 常见异常类型

ZeroDivisionError - 除零错误

当尝试将数字除以零时触发。

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("错误：不能除以零")
```

ValueError - 值错误

当函数接收到正确类型但值不合适的参数时触发，通常是类型转换失败。

```
    try:
    age = int(input("请输入年龄: "))
except ValueError:
    print("错误：请输入有效的数字")
```

7.2.5 异常的层次结构

Python的异常类型形成了一个层次结构，所有异常都继承自BaseException类。理解异常的层次结构有助于我们更好地捕获和处理异常。

```
BaseException
 ├── SystemExit
 ├── KeyboardInterrupt
 └── Exception
      ├── ArithmeticError
      │    └── ZeroDivisionError
      ├── LookupError
      │    ├── IndexError
      │    └── KeyError
      ├── ValueError
      └── OSError
           ├── FileNotFoundError
           └── PermissionError
```

层次结构说明：

- BaseException：所有异常的基类
- Exception：大多数异常的基类，应该捕获这个类型
- ArithmeticError：所有算术错误的基类
- LookupError：所有查找错误的基类
- OSError：所有操作系统相关错误的基类

7.2.5 异常的层次结构

使用建议：

- 通常应该捕获Exception或其子类，而不是BaseException
- 捕获具体的异常类型比捕获通用的Exception更好
- 可以捕获父类异常来处理多种相关的异常类型

```
## 捕获父类异常的示例
try:
    numbers = [1, 2, 3]
    print(numbers[10])  # IndexError
except LookupError:  # 可以捕获IndexError和KeyError
    print("查找错误：索引或键不存在")
```

7.3 Ask AI：文件处理与异常的高级应用

当想要深入了解文件处理和异常管理的更多高级特性时，可以向AI助手提出以下问题：

高级文件操作

- "如何处理大型文件？什么是文件的流式读取？"
- "Python中如何处理二进制文件？"
- "pathlib模块相比os.path有什么优势？"

实践练习

练习 7.2.1：安全数字转换器

编写一个函数，接收字符串输入并尝试转换为整数或浮点数，使用异常处理确保转换过程安全，返回转换结果或错误信息。

练习 7.2.2：批量文件处理器

创建一个程序，尝试读取指定目录中的多个文本文件，对每个文件的读取操作进行异常处理，并生成处理报告。

练习 7.2.3：数据解析器

设计一个程序，从用户输入的文本中解析数字并进行计算，使用异常处理来处理格式错误、计算异常等情况。