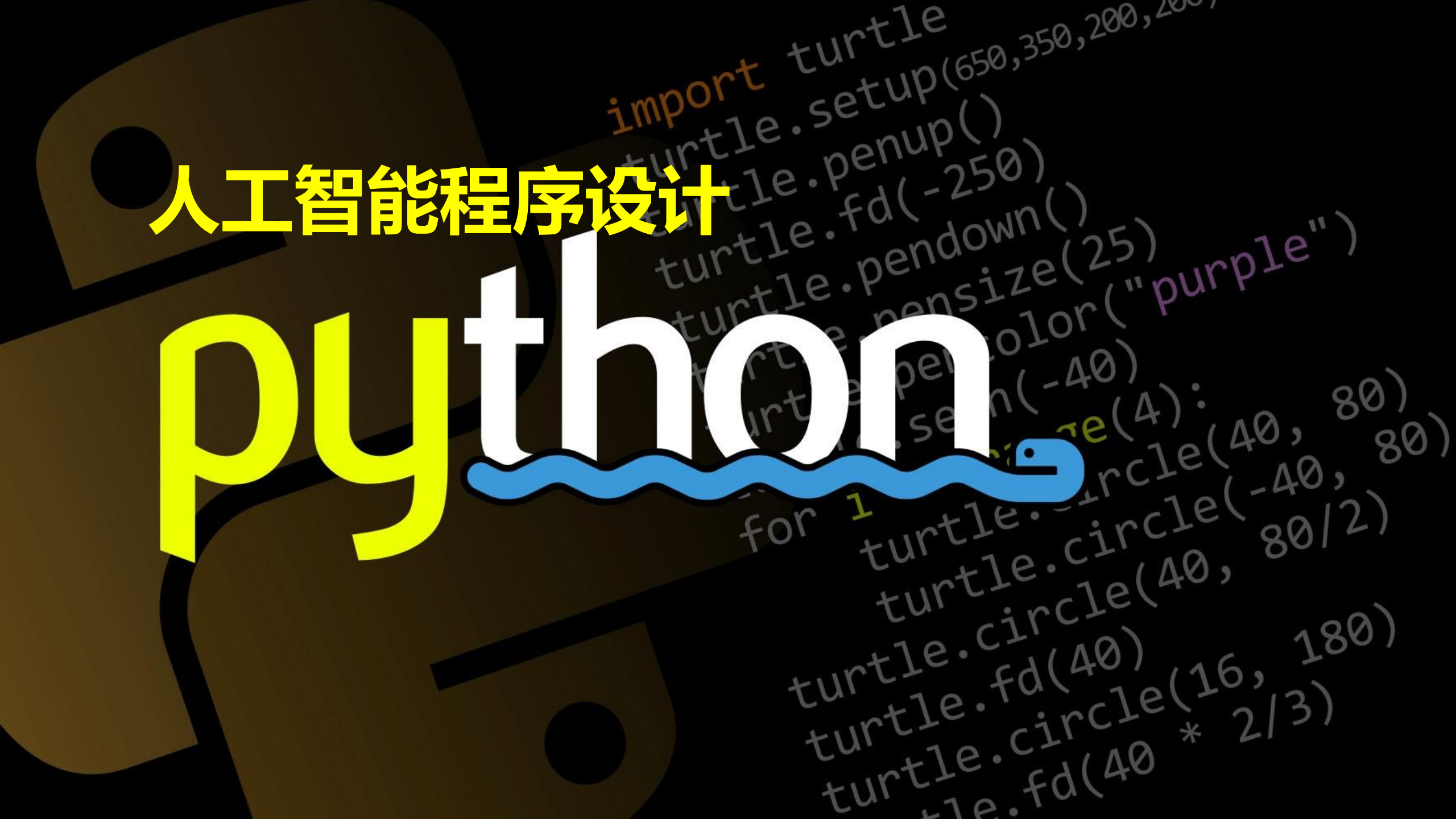
人工智能程序设计


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

17.3 多智能体系统：CREWAI协作框架

北京石油化工学院 人工智能研究院

刘 强

学习内容

- CrewAI智能体团队构建
- 角色分工与任务协调
- 多智能体项目部署

17.3.1 CrewAI智能体团队构建

学习内容：

- 智能体角色定义
- 团队协作机制
- 协作流程设计

CrewAI简介

CrewAI是专门用于构建多智能体协作系统的框架。

核心特点：

- 通过角色分工和任务协调
- 实现团队化的问题解决
- 让多个专业化智能体高效协作

智能体角色定义

CrewAI中的每个智能体都有明确的角色、目标和技能。

```python
from crewai import Agent

# 创建研究员智能体
researcher = Agent(
    role="研究员",
    goal="收集和分析相关信息",
    backstory="专业的信息收集和分析专家",
    verbose=True
)

# 创建写作者智能体
writer = Agent(
    role="写作者",
    goal="创作高质量内容",
    backstory="经验丰富的内容创作者",
    verbose=True
)
```

审核者智能体

Agent类定义了智能体的基本属性，为智能体行为提供明确指导。

```
# 创建审核者智能体
reviewer = Agent(
    role="审核者",
    goal="确保内容质量和准确性",
    backstory="严谨的质量控制专家",
    verbose=True
)
```

团队协作机制

CrewAI通过定义清晰的协作流程，让不同角色有序配合。

Crew类管理智能体团队的协作流程。

```
from crewai import Crew

# 组建智能体团队
team = Crew(
    agents=[researcher, writer, reviewer],
    process="sequential",  # 顺序执行
    verbose=True
)
```

团队工作流程

定义研究、写作、审核三阶段的协作流程，每个智能体按顺序完成各自专业任务。

```
def team_workflow(topic):
    # 研究阶段
    research_result = researcher.execute("研究{}的相关信息".format(topic))

    # 写作阶段
    content = writer.execute("基于研究结果写作关于{}的文章".format(topic))

    # 审核阶段
    final_result = reviewer.execute("审核并完善文章内容")

    return final_result
```

17.3.2 角色分工与任务协调

学习内容：

- 任务定义与分配
- 协调策略配置
- 执行模式选择

任务定义与分配

CrewAI通过任务对象明确定义每个智能体的具体工作。

```python
from crewai import Task

# 定义研究任务
research_task = Task(
    description="研究人工智能在教育领域的应用现状和发展趋势",
    agent=researcher,
    expected_output="详细的研究报告"
)

# 定义写作任务
writing_task = Task(
    description="基于研究结果撰写一篇技术文章",
    agent=writer,
    expected_output="高质量的技术文章"
)
```

审核任务定义

Task类定义了任务的具体要求和期望输出。

```
# 定义审核任务
review_task = Task(
    description="审核文章的准确性和可读性",
    agent=reviewer,
    expected_output="经过审核的最终文章"
)
```

协调策略配置

CrewAI支持多种协调策略：

```
# 顺序协作模式
sequential_crew = Crew(
    agents=[researcher, writer, reviewer],
    tasks=[research_task, writing_task, review_task],
    process="sequential"
)

# 层次化协作模式
parallel_crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, writing_task],
    process="hierarchical"
)

# 执行任务
result = sequential_crew.kickoff()
```

协作模式对比

模式	特点	适用场景
sequential	顺序执行	有依赖关系的任务
hierarchical	层次化管理	需要监督协调的任务

17.3.3 多智能体项目部署

学习内容：

- 系统架构设计
- 监控与管理
- 性能优化

系统架构设计

多智能体系统通常采用分布式架构。

```
class MultiAgentSystem:
    def __init__(self):
        self.agents = {}
        self.message_queue = []
        self.coordinator = None

    def register_agent(self, agent_id, agent):
        self.agents[agent_id] = agent

    def coordinate_task(self, task):
        # 任务分解
        subtasks = self.decompose_task(task)
        # 分配给合适的智能体
        results = []
        for subtask in subtasks:
            agent_id = self.select_agent(subtask)
            result = self.agents[agent_id].execute(subtask)
            results.append(result)
        # 整合结果
        return self.integrate_results(results)
```

监控与管理

多智能体系统需要完善的监控机制。

```
class AgentMonitor:
    def __init__(self):
        self.metrics = {}
        self.alerts = []

    def track_performance(self, agent_id, task_id, duration, success):
        if agent_id not in self.metrics:
            self.metrics[agent_id] = []

        self.metrics[agent_id].append({
            "task_id": task_id,
            "duration": duration,
            "success": success,
            "timestamp": time.time()
        })
```

获取智能体状态

帮助管理员了解系统运行状况和优化性能。

```
def get_agent_status(self, agent_id):
    if agent_id in self.metrics:
        recent_tasks = self.metrics[agent_id][-10:]
        success_rate = sum(1 for t in recent_tasks if t["success"]) / len(recent_tasks)
        avg_duration = sum(t["duration"] for t in recent_tasks) / len(recent_tasks)
        return {"success_rate": success_rate, "avg_duration": avg_duration}
    return {"success_rate": 0, "avg_duration": 0}
```

17.3.4 Ask AI：多智能体协作优化

想要学习更多多智能体协作的高级技巧，可以向AI助手询问：

- "如何设计高效的多智能体通信协议？"
- "多智能体系统中如何处理冲突和协调？"

实践练习

练习 17.3.1：团队构建

创建一个包含不同专业角色的智能体团队，实现内容创作的完整流程。

实践练习

练习 17.3.2：任务协调

设计一个多智能体协作系统，处理需要多个步骤和不同专业技能的复杂任务。

实践练习

练习 17.3.3：系统部署

实现一个简单的多智能体管理系统，包括智能体注册、任务分配和性能监控功能。