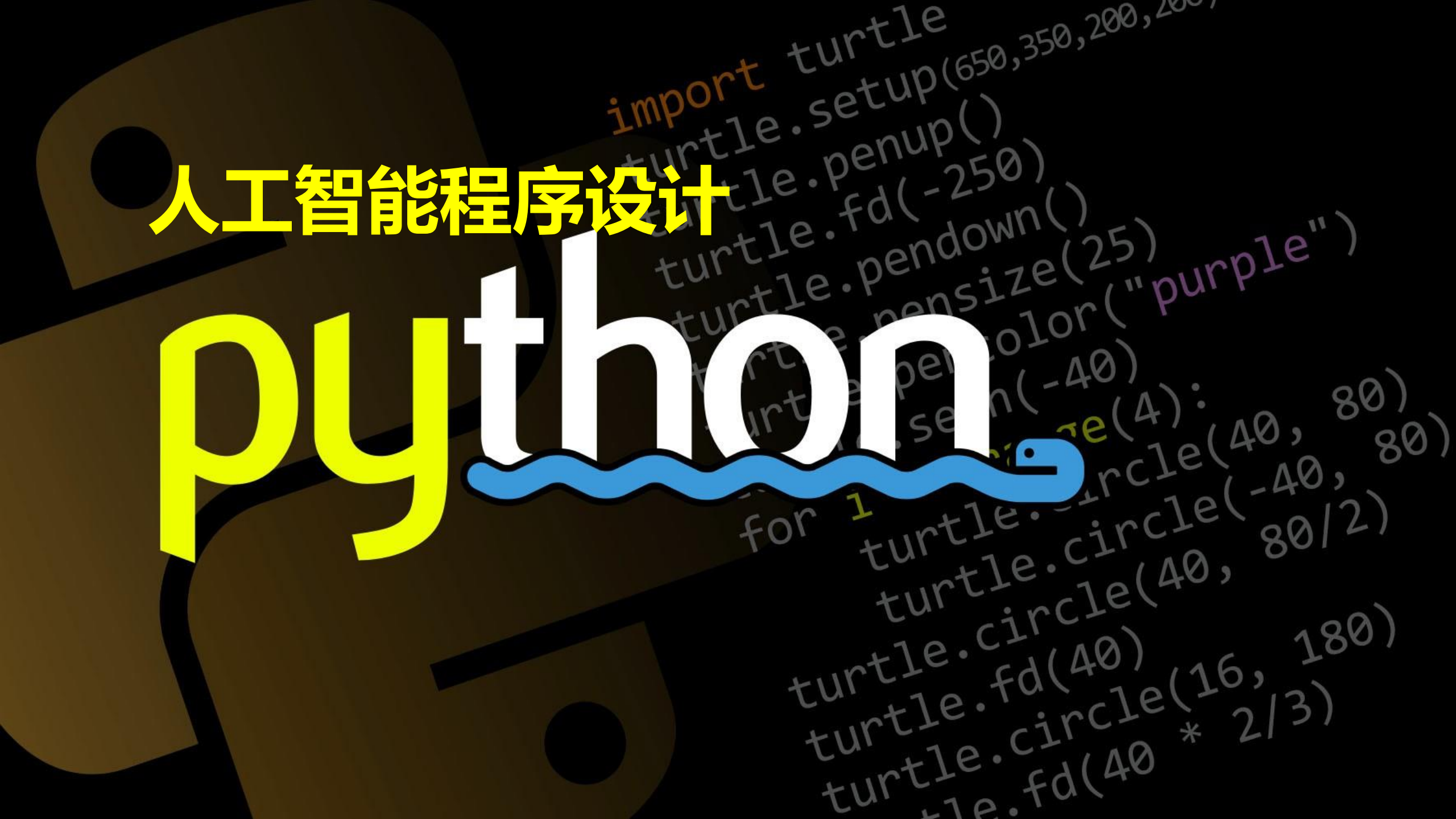
人工智能程序设计


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

16.3 大模型集成开发：LANGCHAIN

北京石油化工学院 人工智能研究院

刘 强

学习内容

- LangChain框架核心组件
- 向量数据库与文档检索
- RAG检索增强生成系统

16.3.1 LangChain框架入门

学习内容：

- LangChain核心概念
- 主要组件介绍
- 基础功能实现

LangChain简介

LangChain是构建大语言模型驱动应用的开源框架。

核心目标：

- 解决大模型能力单一问题
- 整合外部资源和工具
- 实现复杂AI应用的快速搭建

开发者可像搭积木一样组合模块化组件。

LangChain核心组件

组件类别	核心功能	常见用途
模型集成	对接主流大模型	调用GPT-4、文心一言等
提示工程	管理提示模板	动态注入变量
文档处理	加载、分割文档	构建知识库
检索系统	向量数据库检索	语义搜索

LangChain核心组件（续）

组件类别	核心功能	常见用途
链	串联多个处理步骤	RAG问答链
代理	自主决策调用工具	智能助手
记忆	存储对话历史	多轮对话
工具	预设可调用工具	计算、API调用

提示模板

提示模板是LangChain的核心功能，允许创建可重用的提示结构。

```
# 创建提示模板
template = ChatPromptTemplate.from_messages([
    ("system", "你是一个专业的翻译助手"),
    ("human", "请将以下{source_lang}文本翻译成{target_lang}: \n{text}")
])

# 创建链
chain = LLMChain(llm=chat_model, prompt=template)

# 使用
result = chain.run(
    source_lang="英文",
    target_lang="中文",
    text="Machine learning is transforming the world."
)
```

链式调用

链式调用将多个处理步骤串联，形成复杂工作流。

```
# 创建分析链
chain1 = LLMChain(
    llm=chat_model,
    prompt=PromptTemplate(
        input_variables=["code"],
        template="分析这段代码的功能：\n{code}"
    )
)

# 创建建议链
chain2 = LLMChain(
    llm=chat_model,
    prompt=PromptTemplate(
        input_variables=["analysis"],
        template="基于分析结果提供改进建议：\n{analysis}"
    )
)
```

组合顺序链

将多个链组合成顺序执行的工作流：

每个链的输出成为下一个链的输入，实现多步骤自动化处理。

```
# 组合成顺序链
overall_chain = SimpleSequentialChain(chains=[chain1, chain2])

# 执行：代码分析 -> 改进建议
result = overall_chain.run("def add(a, b): return a + b")
```

记忆机制

记忆机制使大模型能够维护对话历史，实现上下文感知的多轮对话。

```
# 创建记忆和对话链
memory = ConversationBufferMemory()
conversation = ConversationChain(llm=chat_model, memory=memory)

# 使用
response1 = conversation.predict(input="我叫张三")
response2 = conversation.predict(input="我的名字是什么？")
# 模型能记住用户之前说过的名字
```

16.3.2 向量数据库与文档检索

学习内容：

- 文档加载与分割
- 向量存储与检索
- 语义相似度搜索

文档加载与分割

文档处理是构建知识库的第一步。

分割保证语义完整性，满足模型输入长度限制。

```
# 加载文档
loader = TextLoader("document.txt")
documents = loader.load()

# 分割文档
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,    # 每块1000字符
    chunk_overlap=200   # 块之间重叠200字符
)
texts = text_splitter.split_documents(documents)
```

向量存储

向量存储将文档转换为高维向量表示并建立索引。

支持基于语义相似度的快速检索，是实现智能问答的关键技术。

```
# 创建向量存储
vectorstore = FAISS.from_documents(texts, embeddings)

# 相似性搜索
query = "什么是机器学习？"
docs = vectorstore.similarity_search(query, k=3)
```

16.3.3 RAG检索增强生成系统

学习内容：

- RAG系统原理
- 基本RAG实现
- 流式RAG响应

RAG系统原理

RAG（Retrieval-Augmented Generation）结合了检索和生成。

工作流程：

1. 用户提问
2. 从知识库检索相关文档
3. 将文档作为上下文提供给大模型
4. 大模型基于上下文生成回答

结合检索的准确性和生成的灵活性。

示例 16.3.1：知识库问答系统

RAG系统通过检索相关文档片段，使大模型基于特定知识库回答问题。

```
# 创建检索器
retriever = vectorstore.as_retriever(search_kwargs={"k": 3})

# 自定义提示模板
template = """
基于以下上下文信息回答问题：

{context}

问题：{question}
答案：
"""

prompt = PromptTemplate(
    template=template,
    input_variables=["context", "question"]
)
```

创建RAG链

将检索器和提示模板组合成完整的RAG问答链：

```
# 创建RAG链
qa_chain = RetrievalQA.from_chain_type(
    llm=chat_model,
    chain_type="stuff",
    retriever=retriever,
    chain_type_kwargs={"prompt": prompt}
)

# 使用
answer = qa_chain.run("什么是深度学习？")
```

流式RAG响应

流式RAG响应可以实时输出生成内容，提供更好的用户体验。

用户无需等待完整回答生成完毕。

```
# 创建流式RAG链
streaming_qa = RetrievalQA.from_chain_type(
    llm=streaming_model,
    retriever=retriever
)

# 流式回答 - 用户即时看到回答进展
streaming_qa.run("解释人工智能的发展历程")
```

16.3.4 LangChain应用开发流程

开发流程：

1. 需求分析与架构设计
2. 模型选择与配置
3. 组件开发与集成
4. 测试优化
5. 部署维护

核心组件选择指南

应用需求	推荐组件
简单文本生成	LLMChain
多步骤处理	SimpleSequentialChain
外部知识支持	RAG相关组件
对话式应用	Memory + ConversationChain

16.3.5 LangChain典型应用场景

场景	说明
知识库问答	企业文档检索+智能回答
智能代理	自主联网、数据分析
多模态应用	语音+文本+图像处理
多轮对话	记忆上下文的对话系统

实践练习

练习 16.3.1：LangChain组件实验

分别实现本章介绍的提示模板、链式调用和记忆机制功能，比较它们与直接API调用的差异。

实践练习

练习 16.3.2：文档知识库构建

使用LangChain的文档处理功能，将一组相关文档（如技术手册）构建成可检索的向量数据库。

实践练习

练习 16.3.3：简单RAG系统

结合向量数据库和RAG功能，构建一个基于特定文档集合的问答系统，测试检索准确性。