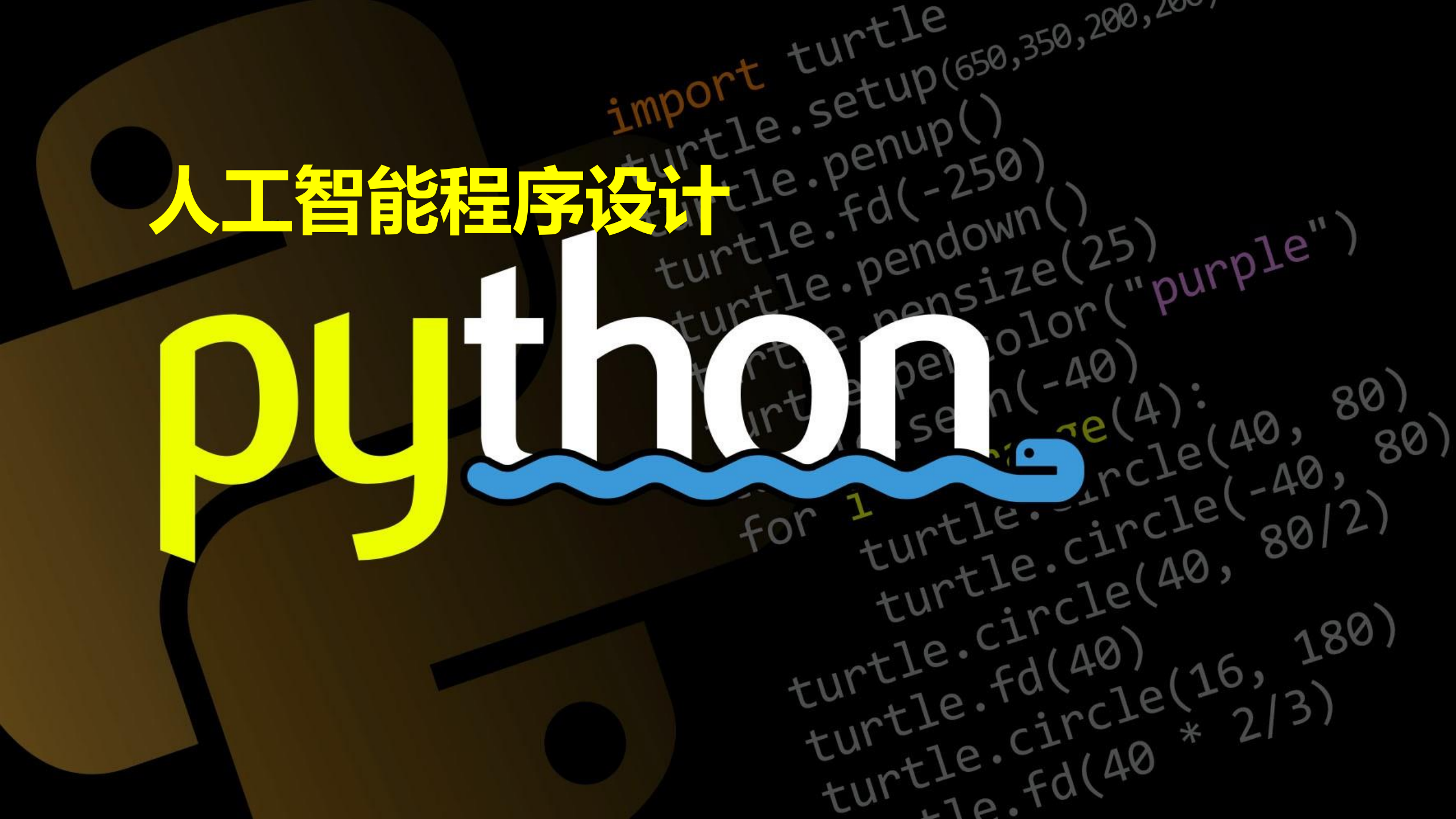
人工智能程序设计


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

14.3 NLP应用实践

北京石油化工学院 人工智能研究院

刘 强

NLP项目开发流程

通过构建完整的NLP应用项目，将学习的技术有机结合：

1. 需求分析和系统设计
2. 数据收集和预处理
3. 模型选择和训练
4. 系统集成和测试
5. 部署和优化

14.3.1 智能文本分析系统

系统功能：

- 文本预处理和清洗
- 文本分类
- 关键词提取
- 文本摘要

系统架构设计

模块化架构包括:

- 文本预处理模块
- 特征提取模块
- 模型推理模块
- 结果展示模块

工作流程:

文本输入 → 预处理清洗 → 分词标注 → 特征提取 → 模型分析 → 结果输出

文本清洗函数

实现文本清洗功能，去除噪声数据：

```
## 文本清洗
def clean_text(text):
    # 去除特殊字符
    text = re.sub(r'[^\w\s]', '', text)
    # 转换为小写
    text = text.lower()
    # 去除多余空格
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

中文分词函数

实现中文分词功能，过滤停用词：

这个函数将中文文本切分为词语列表，并过滤掉单字词。

```
## 中文分词
def segment_chinese(text):
    words = jieba.cut(text)
    return [word for word in words if len(word) > 1]
```

特征提取函数

使用TF-IDF进行文本特征提取：

TF-IDF（词频-逆文档频率）是经典的文本特征提取方法。

```
## 特征提取
def extract_features(texts):
    vectorizer = TfidfVectorizer(max_features=1000)
    features = vectorizer.fit_transform(texts)
    return features, vectorizer
```

文本分类实现

使用朴素贝叶斯算法训练分类模型：

MultinomialNB是适合文本分类的朴素贝叶斯变体。

```
## 训练分类模型
def train_classifier(features, labels):
    classifier = MultinomialNB()
    classifier.fit(features, labels)
    return classifier
```

文本分类预测

完整的文本分类预测流程：

```
## 文本分类预测
def predict_category(text, vectorizer, classifier):
    # 预处理
    cleaned_text = clean_text(text)
    words = segment_chinese(cleaned_text)
    processed_text = ' '.join(words)

    # 特征提取
    features = vectorizer.transform([processed_text])

    # 预测分类
    prediction = classifier.predict(features)[0]
    probability = classifier.predict_proba(features)[0]

    return prediction, max(probability)
```

TF-IDF关键词提取

使用TF-IDF算法提取文本关键词：

```
## TF-IDF关键词提取
def extract_keywords_tfidf(text, top_k=10):
    words = segment_chinese(text)
    text_processed = ' '.join(words)

    vectorizer = TfidfVectorizer(max_features=1000)
    tfidf_matrix = vectorizer.fit_transform([text_processed])

    feature_names = vectorizer.get_feature_names_out()
    tfidf_scores = tfidf_matrix.toarray()[0]

    # 获取top_k关键词
    top_indices = tfidf_scores.argsort()[-top_k:][::-1]
    keywords = [(feature_names[i], tfidf_scores[i]) for i in top_indices]

    return keywords
```

TextRank关键词提取

使用TextRank算法提取关键词：

TextRank基于图算法，不依赖语料库。

```
## TextRank关键词提取
def extract_keywords_textrank(text, top_k=10):
    keywords = textrank4zh.TextRank4Keyword()
    keywords.analyze(text=text, lower=True, window=2)

    result = []
    for item in keywords.get_keywords(top_k, word_min_len=2):
        result.append((item.word, item.weight))

    return result
```

14.3.2 中文情感分析项目

情感分析判断文本的情感倾向：

- 正面（positive）
- 负面（negative）
- 中性（neutral）

应用场景：

- 商品评论分析
- 舆情监控
- 客户反馈分析

基于词典的情感分析

加载情感词典：

```
## 加载情感词典
def load_sentiment_dict():
    positive_words = set()
    negative_words = set()

    # 加载正面词汇
    with open('positive.txt', 'r', encoding='utf-8') as f:
        positive_words = set(line.strip() for line in f)

    # 加载负面词汇
    with open('negative.txt', 'r', encoding='utf-8') as f:
        negative_words = set(line.strip() for line in f)

    return positive_words, negative_words
```

词典情感分析实现

基于词典统计情感倾向：

```
## 情感分析
def analyze_sentiment(text, positive_words, negative_words):
    words = segment_chinese(text)

    positive_count = sum(1 for word in words if word in positive_words)
    negative_count = sum(1 for word in words if word in negative_words)

    if positive_count > negative_count:
        return "positive", positive_count - negative_count
    elif negative_count > positive_count:
        return "negative", negative_count - positive_count
    else:
        return "neutral", 0
```

深度学习情感分析

使用预训练BERT模型进行情感分析：

```
## 使用预训练模型
def sentiment_analysis_bert(text):
    # 加载中文情感分析模型
    classifier = pipeline("sentiment-analysis",
                          model="uer/roberta-base-finetuned-chinanews-chinese")

    result = classifier(text)

    # 转换标签
    label_map = {"LABEL_0": "negative", "LABEL_1": "positive"}
    sentiment = label_map.get(result[0]['label'], result[0]['label'])
    confidence = result[0]['score']

    return sentiment, confidence
```

批量情感分析

处理多条文本的情感分析:

```
## 批量情感分析
def batch_sentiment_analysis(texts):
    results = []
    for text in texts:
        sentiment, confidence = sentiment_analysis_bert(text)
        results.append({
            'text': text,
            'sentiment': sentiment,
            'confidence': confidence
        })
    return results
```

情感趋势分析

分析文本情感随时间的变化趋势：

```python
## 情感趋势分析
def sentiment_trend_analysis(data):
    # data格式: [{'text': '...', 'timestamp': '...'}]

    results = []
    for item in data:
        sentiment, confidence = sentiment_analysis_bert(item['text'])
        results.append({
            'timestamp': item['timestamp'],
            'sentiment': sentiment,
            'confidence': confidence,
            'text': item['text']
        })

    # 按时间分组统计
    df = pd.DataFrame(results)
    df['date'] = pd.to_datetime(df['timestamp']).dt.date

    daily_sentiment = df.groupby(['date', 'sentiment']).size().unstack(fill_value=0)

    return daily_sentiment
```

词典法 vs 深度学习法

选择建议：

- 简单场景用词典法
- 高精度需求用深度学习

方法	优点	缺点
词典法	简单、可解释	依赖词典质量
深度学习	准确率高	需要GPU资源

14.3.3 Ask AI：多模态NLP探索

多模态NLP发展方向：

- 文本与图像结合
- 文本与语音结合
- 跨模态检索和生成

代表技术：

- CLIP：图文匹配
- DALL-E：文本生成图像
- Whisper：语音转文本

跨模态检索

应用场景：

- 以文搜图：用文本描述搜索图像
- 以图搜文：用图像搜索相关文本
- 图像问答：回答关于图像的问题

核心技术：

将文本和图像映射到统一的特征空间

Ask AI学习建议

向AI助手询问以下进阶内容：

1. **"如何构建多模态的文本-图像理解系统？"**
2. **"CLIP模型是如何实现跨模态表示学习的？"**
3. **"如何评估NLP模型在实际应用中的性能？"**

实践练习

练习 14.3.1：文本分析系统

构建完整的文本分析系统，包括：

- 文本分类功能
- 关键词提取功能
- 摘要生成功能

实践练习

练习 14.3.2：情感分析项目

开发中文情感分析应用：

- 支持实时分析
- 支持批量处理
- 包含可视化展示

实践练习

练习 14.3.3：多模态应用

探索文本与图像结合的应用场景：

- 图像标题生成
- 视觉问答系统
- 跨模态检索

实践练习

练习 14.3.4：系统优化

优化NLP系统的处理速度和准确率：

- 分析不同算法的性能
- 模型压缩和加速
- 缓存和批处理优化