安徽理工大学
ANHUI UNIVERSITY OF SCIENCE & TECHNOLOGY

• 人工智能专业 学科基础教育必修模块

Python与机器学习

Python and Machine Learning

Chapter 4:Python Basic Syntax

• Lecturer ： 孟亦凡

• E-mail：myf@aust.edu.cn

安徽理工大學
ANHUI UNIVERSITY OF SCIENCE & TECHNOLOGY

Chapter 4：Python Basic Syntax

Target

- 掌握Python核心流程控制结构
- **Master Python's core flow control structures.**
- 理解Python与C++在语法和思维上的异同
- **Understand the syntactic and conceptual similarities and differences between Python and C++.**

安徽理工大學
ANHUI UNIVERSITY OF SCIENCE & TECHNOLOGY

Chapter 3： Python Basic Data Types

CONTENTS

4.1 Branching Structure of the Program
程序的分支结构

➢ From Condition Checking to Execution Path（从条件判断到执行路径）

- **Core Purpose: To equip programs with "decision-making" capabilities. (让程序具备“决策”能力)**
- **Execute different code blocks based on different conditions（条件）.**
- **Real-world Analogy(现实世界类比):**
- **If it rains today, I will take an umbrella; otherwise, I will not.**
- **If the exam score >= 60, display "Pass"; otherwise, display "Fail".**

➢ Review of Branching Structures in C++

- **Syntax relies（依赖） on if, else if, else, and curly braces {}.**
- **Conditions（条件） must be enclosed in parentheses ().**

```
// C++ 示例
int score = 85;
if (score >= 90) {
    cout << "优秀";
} else if (score >= 60) {
    cout << "及格";
} else {
    cout << "不及格";
}
```

➢ Branching Structures in Python

- Same keywords: if, elif（else if）, else
- Key Difference: ces.（缩进替代花括号）
- Indentation (ty: mbership.
- Key Difference: es, but must end with a colon :. 号）
- The colon（冒号） : signals the start of a new block.

```python
# Python 示例
score = 85
if score >= 90:
    print("优秀")       # 属于if块
    print("做得很好!")  # 属于if块
elif score >= 60:
    print("及格")       # 属于elif块
else:
    print("不及格")     # 属于else块
print("程序结束")       # 不属于任何分支，总是执行
```

➢ Branching Structures in Python

用于条件组合的三个保留字

操作符及使用	描述
x and y	两个条件x和y的逻辑与
x or y	两个条件x和y的逻辑或
not x	条件x的逻辑非

➢ Summary：Compare With C++

特性	C++	Python
代码块界定	花括号 {}	缩进 (Indentation)
条件	必须使用 ()	不需要 ()
代码块开始	{	冒号 :
else if	else if	elif
哲学	显式、结构化	简洁、可读性强

➢ Summary: NOTICE

- **Consistency is key（注重一致性）: Stick to the same number of spaces (highly recommend 4) for indentation throughout your project.**
- **Do not mix spaces and tabs（不要混用空格和制表符。）. This is usually handled automatically in PyCharm.**
- **The colon : is easily forgotten, so remember it!（不要忘记冒号）**
- **There is no switch-case in Python. Use if-elif-else chains or dictionary mapping instead.**

4.1 Branching Structure of the Program

➢ Case Study: BM nt

- **BMI Definition:** ess.
- **Formula:**
- **Classification Cr**
  - **BMI < 18.5: 偏**
  - **18.5 ≤ BMI < 24**
  - **24 ≤ BMI < 28:**
  - **BMI ≥ 28: 肥胖**

```python
# 获取用户输入
weight = float(input("请输入体重(kg): "))
height = float(input("请输入身高(m): "))
# 计算BMI
bmi = weight / (height ** 2)
print(f"您的BMI值为: {bmi:.2f}") # 格式化输出，保留2位小数
# 分支判断
if bmi < 18.5:
    print("体重状况: 偏瘦")
    category = "Underweight"
elif bmi < 24:
    print("体重状况: 正常")
    category = "Normal"
elif bmi < 28:
    print("体重状况: 超重")
    category = "Overweight"
else:
    print("体重状况: 肥胖")
    category = "Obese"
print(f"International category: {category}")
```

4.X Homework

➢ （1）扩展练习开发

- 增加输入验证，确保输入体重、身高为正值；
- 根据年龄调整BMI的标准。

4.2 Loop structure of the program
程序的循环结构

➢ Loop Structure：The Engine for Automating Repetitive Tasks

- **Core Purpose: To enable programs to repeatedly execute specific code blocks.**
  （让程序能够重复执行特定代码块）
- **Avoid code redundancy（冗余）, improve efficiency and processing capability.**
- **Real-world Analogy:**
- **Printing 100 exam papers → Repeat the "print" action 100 times.**
- **Calculating the average score of 50 students → Repeat the "add score" action 50 times.**

4.2 Loop Structure of the Program

➢ Review Loop Structure in C ++

- **for loop: When the number of iterations is known.**
- **while loop: Executes as long as condition is true.**

```cpp
// C++ for循环示例
for (int i = 0; i < 5; i++) {
    cout << "循环次数: " << i << endl;
}
```

```cpp
// C++ while循环示例
int count = 0;
while (count < 5) {
    cout << "Count: " << count << endl;
    count++;
}
```

➢ Iteration loops（遍历循环）

Iterate through a structure to form a loop.

for <循环变量> in <遍历结构>：

<语句块>

Extracting elements from an iterable and putting them into a loop variable one by one.

(从遍历结构中逐一提取元素，放在循环变量中。)

➢ Iteration loops（遍历循环）

for <循环遍历> in <遍历结构>:

<语句块>

Composed of the reserved words 'for' and 'in', it loops through all elements until complete.

（由保留字for和in组成，完整遍历所有元素后结束。）

During each loop iteration, an element is obtained and placed into the loop variable, and then the statement block is executed.

（每次循环，所有获得元素放入循环变量，并执行一次语句块。）

➢ Iteration loops（遍历循环）

Counting cycle (N times)

```
for i in range(N):
    <语句块>
```

Iterating over a sequence of numbers produced by the range() function creates a loop.

（遍历由range()函数产生的数字序列，产生循环。）

➢ Iteration loops（遍历循环）

```
>>>for i in range(5):
>>>    print(i)

0
1
2
3
4
```

```
>>>for i in range(5):
>>>    print("hello:", i)

hello: 0
hello: 1
hello: 2
hello: 3
hello: 4
```

➢ Iteration loops（遍历循环）

String Iteration Loop

```
for c in s:
    <语句块>
```

Given a string s, iterating over each character in the string produces a loop.

（s是字符串，遍历字符串每个字符，产生循环。）

➢ Iteration loops（遍历循环）

String Iteration Loop

```
>>> for c in “Python123”:
>>>     print(c, end=“,”)
```

P,y,t,h,o,n,1,2,3,

➢ Iteration loops（遍历循环）

List Iteration Loop

```
>>> for item in ls:
        <语句块>
```

Given a list ls, iterating over each element in the list produces a loop.

（ls是一个列表，遍历其每个元素，产生循环。）

➢ Iteration loops（遍历循环）

List Iteration Loop

```
>>> for item in [123, “yyds”, 456]:
>>>     print(item, end=“,”)

123,yyds,345,
```

➢ Iteration loops（遍历循环）

File Iteration Loop

```
>>> for line in fi:
    <语句块>
```

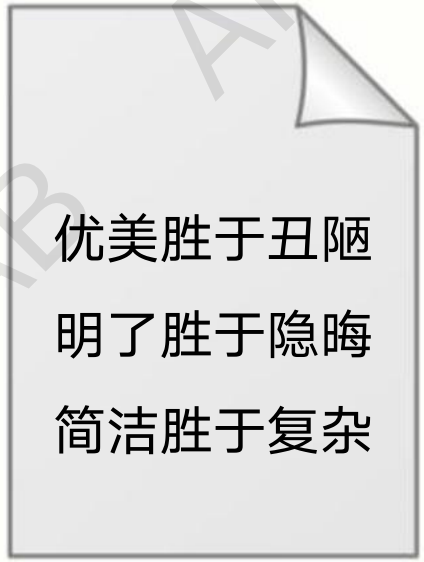
Given a file fi, iterating over each line in the file produces a loop.

（fi是一个文件，遍历其每行，产生循环。）

➢ Iteration loops（遍历循环）

File Iteration Loop

```
>>> for line in fi:
>>>         print(line)
    优美胜于丑陋
    明了胜于隐晦
    简洁胜于复杂
```



4.2 Loop Structure of the Program

➢ Iteration loops（遍历循环）

- 计数循环(N次)
- 计数循环(特定次)
- 字符串遍历循环
- 列表遍历循环
- 文件遍历循环
- ……

➢ Conditional Loop（条件循环）

Loop operation method controlled by conditions.

while <条件>:

<语句块>

- 反复执行语句块，直到条件不满足时结束。

➢ Conditional Loop（条件循环）

```
>>> a = 3
>>> while a > 0:
>>>     a = a - 1
>>>     print(a)

2
1
0
```

```
>>> a = 3
>>> while a > 0:
>>>     a = a + 1
>>>     print(a)

4
5
...
```

➢ Loop control reserved word（循环控制保留字）

break & continue

- **break:** jump out and end the whole current loop, execute the statement after the loop.
- **continue**: end the current loop, continue to execute the following times loop.
- **break** and **continue** can be used in conjunction（配合） with **for** and **while** loops.

➢ Loop control reserved word（循环控制保留字）

break & continue

```
>>> for c in “PYTHON”:
        if c == “T”:
            continue
        print(c, end=“”)

PYHON
```

```
>>> for c in “PYTHON”:
        if c == “T”:
            break
        print(c, end=“”)

PY
```

➢ Loop control reserved word（循环控制保留字）

```
>>> s = "PYTHON"
>>> while s != "" :
        for c in s:
            print(c, end="")
        s = s[:-1]
PYTHONPYTHOPYTHPYTPYP
```

```
>>> s = "PYTHON"
>>> while s != "" :
        for c in s:
            if c == "T":
                break
            print(c, end="")
        s = s[:-1]
PYPYPYPYPYP
```

➢ Advanced use of loops（循环的高级用法）

Loop and else

```
for <变量> in <遍历结构>:
    <语句块1>
else:
    <语句块2>
```

```
while <条件>:
    <语句块1>
else:
    <语句块2>
```

➢ Advanced use of loops（循环的高级用法）

Loop and else

– 当循环没有被break语句退出时,执行else语句块

–else语句块作为“正常”完成循环的奖励

➢ Advanced use of loops（循环的高级用法）

```
>>> for c in “PYTHON”:
        if c == “T”:
            continue
        print(c, end=“”)
    else:
        print(“正常退出”)


PYHON正常退出
```

```
>>> for c in “PYTHON”:
        if c == “T”:
            break
        print(c, end=“”)
    else:
        print(“正常退出”)


PY
```

➢ Summary

- for … in 遍历循环:计数、字符串、列表、文件……
- while 无限循环
- continue和break保留字:退出当前循环层次
- 循环else的高级用法:与break有关

➢ Summary

特性	C++	Python
for循环哲学	基于索引的计数	基于迭代器的遍历
range()函数	需要手动创建计数器	内置range()生成序列
自增操作	i++	i += 1
循环else	不支持	支持for-else/while-else
代码块	花括号{}	缩进

4.X Homework

（2）编程练习

打印出所有的“水仙花数”，所谓“水仙花数”是指一个三位数，其各位数字立方等于该数本身。例如：153是一个“水仙花数”，因为 153=1^3+5^3+3^3.

需求：打印所有“水仙花数”。

思路：利用for循环控制100-999个数，每个数分解出个位，十位，百位。

4.X Homework

（3）编程练习

打印九九乘法表

需求：九九乘法口诀表。

思路：分行与列考虑，共9行9列，i控制行，j控制列。

```
1x1=1
1x2=2   2x2=4
1x3=3   2x3=6   3x3=9
1x4=4   2x4=8   3x4=12  4x4=16
1x5=5   2x5=10  3x5=15  4x5=20  5x5=25
1x6=6   2x6=12  3x6=18  4x6=24  5x6=30  6x6=36
1x7=7   2x7=14  3x7=21  4x7=28  5x7=35  6x7=42  7x7=49
1x8=8   2x8=16  3x8=24  4x8=32  5x8=40  6x8=48  7x8=56  8x8=64
1x9=9   2x9=18  3x9=27  4x9=36  5x9=45  6x9=54  7x9=63  8x9=72  9x9=81
```

4.3 Exception handling of programs
(程序的异常处理)

4.3 Exception handling of programs

➢ Exception

```
num = eval(input("请输入一个数字:"))
print(num**2)
```

当用户没有输入数字时，会产生异常，怎么处理？

```
请输入一个数字:ab
Traceback (most recent call last):
  File "/Users/mengyifan/Documents/Studio/Software/PythonML/test1.py", line 1, in <module>
    num = eval(input("请输入一个数字:"))
  File "<string>", line 1, in <module>
NameError: name 'ab' is not defined. Did you mean: 'abs'?

进程已结束，退出代码为 1
```

➢ What is exception?

- **Exception: An error event that occurs during program execution.**
  (程序运行期间发生的错误事件)
- **Interrupts the normal flow of instructions.**
- **Common Exception Examples:**
- **除零错误； 文件不存在： open("nonexistent.txt")； 类型错误:**
  "hello" + 5； 索引越界： list[10] (列表只有3个元素)； 键不存在

➢ Basic usage of exception handling?

```
try:
    <语句块1>
except:
    <语句块2>
```

```
try:
    <语句块1>
except <异常类型>:
    <语句块2>
```

4.3 Exception handling of programs

➢ Basic usage of exception handling?

```
try:
    num = eval(input("请输入一个数字:"))
    print(num**2)
except:
    print("输入的不是数字")
```

```
请输入一个数字: dd
输入的不是数字
```

➢ Basic usage of exception handling?

```python
# Python 基本异常处理
try:
    num = int(input("请输入一个数字: "))
    result = 10 / num
    print(f"结果是: {result}")
except ZeroDivisionError:
    print("错误：除数不能为零!")
except ValueError:
    print("错误：请输入有效的数字!")
```

➢ Complete exception handling structure:

- **try: Code to attempt（尝试执行的代码）**
- **except: Catch specific exceptions（捕获特定异常）**
- **else: Execute if no exceptions（没有异常时执行）**
- **finally: Always execute（无论是否异常都执行）**

➢ Complete exception handling structure:

```
try:
    file = open("data.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("文件不存在!")
else:
    print("文件读取成功!")
    print(content)
finally:
    # 确保文件被关闭
    file.close() if 'file' in locals() else None
    print("清理完成")
```

➢ C++ Exception Handling：Look Before You Leap (LBYL)（先检查，后使用）

- **Check all possible error conditions before operation（在操作前检查所有可能的错误条件）**

```
// C++ 风格：先检查
if (file_exists("data.txt")) {
    if (has_permission("data.txt")) {
        // 然后使用
        open_file("data.txt");
    }
}
```

➢ Python Exception Handling：Easier to Ask for Forgiveness than Permission (EAFP)（先使用，后处理异常）

- **Try the operation directly, handle exceptions if it fails（直接尝试操作，如果失败则处理异常）**

```
# Python 风格：先尝试
try:
    file = open("data.txt")
    # 处理文件
except FileNotFoundError:
    print("文件不存在")
except PermissionError:
    print("没有权限")
```

➢ Python Exception Handling：NOTICE

- **Be specific about exception types, avoid bare except.（具体化异常类型，避免裸except）**

```
# 不好的做法
try:
    # 代码
except:  # 捕获所有异常，难以调试
    pass

# 好的做法
try:
    # 代码
except (ValueError, TypeError) as e:
    print(f"输入错误: {e}")
```

➢ Python Exception Handling：NOTICE

- **Log exception information （在日志中记录异常信息）**

```
import logging
try:
    risky_operation()
except Exception as e:
    logging.error(f"操作失败: {e}", exc_info=True)
```

➢ Python Exception Handling：NOTICE

- **Log exception information （在日志中记录异常信息）**

```
import logging
try:
    risky_operation()
except Exception as e:
    logging.error(f"操作失败: {e}", exc_info=True)
```

- **保持try块的精简**
  - **Keep try blocks concise**
  - **只包含可能抛出异常的代码**

➢ Python Exception Handling：NOTICE

- From "Defensive Programming" to "Graceful Degradation"（从"防御性编程"到"优雅降级"）
- Trust Python's exception mechanism, focus on normal flow（信任Python的异常机制，专注于正常流程）
- Use exceptions to make code clearer and more robust（利用异常使代码更清晰、更健壮）

4.4 Basic Usage of Function
函数的基本用法

➢ What is a Function?

- **Encapsulated reusable code block that performs a specific task.**

（封装的可重用代码块，执行特定任务）

- **Purposes（目的）:**
- **Code Reuse（代码复用）: Avoid writing the same code repeatedly.（避免重复编写相同代码）**
- **Modularization（模块化）: Break complex problems into smaller ones.（将复杂问题分解为小问题）**

➢ What is a Function?

- **Improved Readability（提高可读性）: Meaningful function names make code easier to understand.（有意义的函数名使代码更易理解）**
- **Easier Debugging（便于调试）: Each function can be tested independently.（可以单独测试每个函数）**

4.4 Basic Usage of Function

➢ Review of C++ Function

- **Requires declaration of return type and parameter types.需要声明返回类型和参数类型**

```
// C++ 函数示例
#include <iostream>
#include <string>
using namespace std;
// 函数声明
string greet(string name, int age);
// 函数定义
string greet(string name, int age) {
    return "Hello, " + name + "! You are " + to_st
ring(age) + " years old.";
}
int main() {
    string message = greet("Alice", 25);
    cout << message << endl;
    return 0;
}
```

➢ Basic usage of Python Function

- **Use def keyword to define functions.**
- **No need to specify parameter types or return type.**

```
# Python 函数定义
def greet(name, age):
    message = f"Hello, {name}! You a
    re {age} years old."
    return message
# 函数调用
result = greet("Alice", 25)
print(result)
```

➢ Basic usage of Python Function

- **Dynamic Typing（动态类型）: Parameters and return values can be of any type.**

```python
def process_data(data):
    """处理各种类型的数据"""
    if isinstance(data, str):
        return data.upper()
    elif isinstance(data, list):
        return len(data)
    elif isinstance(data, dict):
        return list(data.keys())
    else:
        return f"Unknown type: {type(data)}"
# 同一个函数处理不同类型数据
print(process_data("hello"))    # HELLO
print(process_data([1, 2, 3]))  # 3
print(process_data({"a": 1}))   # ['a']
```

➢ Basic usage of Python Function

- **Multiple Return Values（多返回值）: Actually returns a tuple（元组）.**

```
def calculate_stats(numbers):
    """计算统计信息"""
    total = sum(numbers)
    count = len(numbers)
    average = total / count if count > 0 else 0
    return total, count, average  # 返回元组 (total, count, average)
# 接收多个返回值
total, count, avg = calculate_stats([10, 20, 30, 40])
print(f"总和: {total}, 数量: {count}, 平均值: {avg}")
```

4.4 Basic Usage of Function

➢ Document String of Python Function

- **Use triple quotes to add function documentation.（使用三引号添加函数文档）**

```
def calculate_bmi(weight, height):
    """
    计算身体质量指数(BMI)

    Args:
        weight (float): 体重，单位千克
        height (float): 身高，单位米

    Returns:
        tuple: (bmi_value, category)
            - bmi_value (float): BMI数值
            - category (str): 体重类别
    """
    bmi = weight / (height ** 2)

    if bmi < 18.5:
        category = "偏瘦"
    elif bmi < 24:
        category = "正常"
    elif bmi < 28:
        category = "超重"
    else:
        category = "肥胖"

    return bmi, category
# 查看文档
help(calculate_bmi)
```

➢ Function Parameter Passing

- **Object Reference Passing（对象引用传递）**
- **All parameters are passed by "object reference".**
- **Key Concepts:**
- **Immutable objects (numbers, strings, tuples): Modifications inside function don't affect original.（不可变对象（数字、字符串、元组）：函数内修改不影响原始值）**

➢ Function Parameter Passing

- 可变对象（列表、字典、集合）： 函数内修改会影响原始值
- **Mutable objects (lists, dicts, sets): Modifications inside function affect original.**

➢ Function Parameter Passing

- Most basic parameter passing method.（最基本的参数传递方式）
- Matched by parameter position order.（按参数位置顺序进行匹配）

```
def student_info(name, age, grade):
    print(f"姓名: {name}, 年龄: {age}, 年级: {grade}")
# 位置参数调用
student_info("张三", 18, "高三")  # 姓名: 张三, 年龄: 18, 年级: 高三
# 参数顺序必须正确
student_info(18, "张三", "高三")  # 错误: 姓名: 18, 年龄: 张三, 年级: 高三
```

➢ Function Parameter Passing：Default Parameters

- Provide default values for parameters.

  （为参数提供默认值）

- Parameters with default values can be omitted when calling.

  （调用时可省略有默认值的参数）

➢ Function Parameter Passing: Variable Parameter

- *args: Accept any number of positional arguments. （*args：接收任意数量的位置参数）
- **kwargs: Accept any number of keyword arguments.（接收任意数量的关键字参数）

```
def calculate_sum(*args):
    print(f"参数: {args}, 类型: {type(args)}")
    return sum(args)
print(calculate_sum(1, 2, 3))      # 6
print(calculate_sum(10, 20, 30, 40)) # 100
print(calculate_sum())             # 0
```

```
def create_model_config(**kwargs):
    print(f"配置参数: {kwargs}")
    for key, value in kwargs.items():
        print(f"  {key}: {value}")
create_model_config(
    layers=5,
    activation='relu',
    optimizer='adam',
    dropout=0.3
)
```

➢ Function Parameter Passing

- Combination rules for various parameter types.（各种参数类型的组合规则）

```
a=1, b=2, c=3
args=(4, 5)
d=25, e=30
kwargs={'f': 100, 'g': 200}

进程已结束，退出代码为 0
```

```python
def complex_function(a, b, c=10, *args, d=20, e=30, **kwargs):
    """
    参数顺序:
    1. 位置参数: a, b
    2. 默认参数: c=10
    3. 可变位置参数: *args
    4. 关键字参数: d=20, e=30
    5. 可变关键字参数: **kwargs
    """
    print(f"a={a}, b={b}, c={c}")
    print(f"args={args}")
    print(f"d={d}, e={e}")
    print(f"kwargs={kwargs}")
# 调用示例
complex_function(1, 2, 3, 4, 5, d=25, f=100, g=200)
```

4.4 Basic Usage of Function

➢ NOTICE!

- Use meaningful function and parameter names.(使用有意义的函数名和参数名)
- Add docstrings to functions.（为函数添加文档字符串)
- Avoid using mutable objects as default parameters.(避免使用可变对象作为默认参数)

4.4 Basic Usage of Function

➢ NOTICE!

```python
# 错误做法
def append_to_list(item, target_list=[]):  # 危险!
    target_list.append(item)
    return target_list

# 正确做法
def append_to_list(item, target_list=None):
    if target_list is None:
        target_list = []
    target_list.append(item)
    return target_list
```

4.X Summary

- **4.1 Branching Structure（程序的分支结构）**
  - if-elif-else结构，缩进定义代码块
- **4.2 Loop Structure（程序的循环结构）**
  - for遍历循环，while条件循环，循环控制语句
- **4.3 Exception Handling（程序的异常处理）**
  - try-except-else-finally，EAFP编程哲学
- **4.4 Basic Function Usage（函数的基本使用）**
  - def定义函数，多返回值，文档字符串
- **4.5 Function Parameter Passing（函数的参数传递）**
  - 位置参数、默认参数、可变参数、关键字参数

4.X Homework

➢ （4）编程练习

- Exercise Title: Intelligent Grade Analysis System（智能成绩分析系统）
- Core Functional Requirements（核心功能需求）：
- 1. Input multiple students' grade data（输入多个学生成绩数据）
- 2. Calculate various statistical metrics（计算各项统计指标）
- 3. Generate grade analysis report（生成成绩分析报告）
- 4. Detect abnormal grades and provide alerts（检测异常成绩并提醒）

4.X Homework

- Exercise Title: Intelligent Grade Analysis System（智能成绩分析系统）

```
请输入学生数量: 5
请输入第1个学生的姓名和成绩(格式: 姓名 成绩): 张三 85
请输入第2个学生的姓名和成绩(格式: 姓名 成绩): 李四 92
请输入第3个学生的姓名和成绩(格式: 姓名 成绩): 王五 78
请输入第4个学生的姓名和成绩(格式: 姓名 成绩): 赵六 105
错误: 成绩应在0-100范围内, 请重新输入
请输入第4个学生的姓名和成绩(格式: 姓名 成绩): 赵六 65
请输入第5个学生的姓名和成绩(格式: 姓名 成绩): 钱七 88
```

4.X Homework

- Exercise Title: Intelligent Grade Analysis System（智能成绩分析系统）
- Processing Logic:
- 使用字典存储学生数据 {姓名: 成绩}
- 实现输入验证和异常处理
- 计算平均分、最高分、最低分、标准差

4.X Homework

- Exercise Title: Intelligent Grade Analysis System（智能成绩分析系统）
- Processing Logic:
- 等级划分：

```
def get_grade_level(score):
    if score >= 90: return "A"
    elif score >= 80: return "B"
    elif score >= 70: return "C"
    elif score >= 60: return "D"
    else: return "E"
```

- Anomaly Detection Rules（异常检测规则）：
  - 成绩＞95分：优秀学生提醒
  - 成绩＜60分：需要关注提醒
  - 成绩＞平均分＋2*标准差：异常高分
  - 成绩＜平均分－2*标准差：异常低分

4.X Homework

- 结果示例：

```
学生数据：
张三：85分(B)
李四：92分(A)
王五：78分(C)
赵六：65分(D)
钱七：88分(B)

统计信息：
平均分：81.6分
最高分：92分(李四)
最低分：65分(赵六)
标准差：10.2分

等级分布：
A：1人 (20.0%)
B：2人 (40.0%)
C：1人 (20.0%)
D：1人 (20.0%)
E：0人 (0.0%)

特别提醒：
🎉 李四同学成绩优秀(92分)！
💡 赵六同学需要额外关注(65分)！
```