

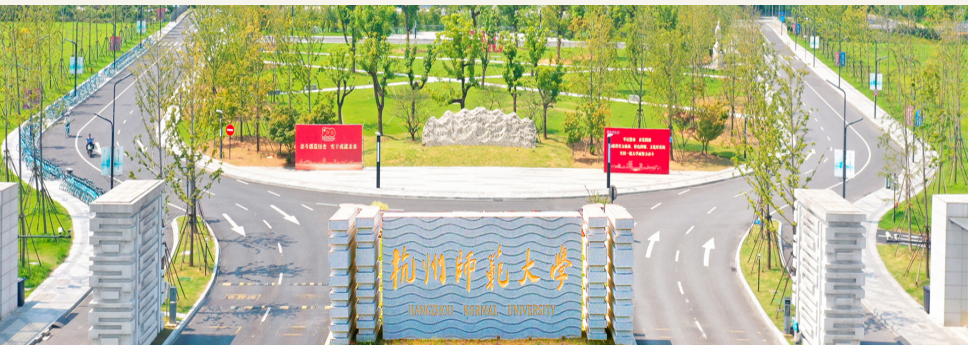
第二章 - Python 快速入门

张建章

阿里巴巴商学院

杭州师范大学

2025-09



- 1 Python 交互式解释器
- 2 Jupyter Notebook
- 3 表达式
- 4 变量
- 5 常见的 Python 数据类型
- 6 保存和执行程序
- 7 写程序注意事项
- 8 函数
- 9 输入和输出
- 10 模块
- 11 流程控制

两种代码执行方式

交互式

```
[1]: # 计算BMI指数

[2]: Height = float(input("请输入身高(m): "))
      请输入身高(m): 1.75

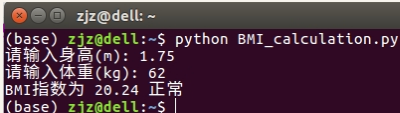
[3]: Weight = float(input("请输入体重(kg): "))
      请输入体重(kg): 62

[4]: BMI = round(Weight/Height**2, 2)

[5]: if BMI>=23.9:
      print("BMI指数为", BMI, "体质偏重")
      elif BMI<=18.5:
      print("BMI指数为", BMI, "体质偏轻")
      else:
      print("BMI指数为", BMI, "正常")

      BMI指数为 20.24 正常
```

脚本式

A terminal window with a dark background and light text. The prompt is 'zjz@dell: ~'. The user enters '(base) zjz@dell:~\$ python BMI_calculation.py'. The program outputs '请输入身高(m): 1.75', '请输入体重(kg): 62', and 'BMI指数为 20.24 正常'. The prompt returns to '(base) zjz@dell:~\$'.

```
zjz@dell: ~
(base) zjz@dell:~$ python BMI_calculation.py
请输入身高(m): 1.75
请输入体重(kg): 62
BMI指数为 20.24 正常
(base) zjz@dell:~$
```

推荐使用交互式解释器学习 Python 程序设计

安装 Python 3.X 并启动: 本课程安装的 Anaconda 3.X 已内含 Python 3.X, 启动 Jupyter-lab 即启动 Python;



初学者使用本地交互式解释器的一些常见问题

(1) jupyter-lab 启动后无法成功运行

- ① 在使用 jupyter-lab 的过程中，确保 cmd 黑框框处于打开状态；
- ② 如果 cmd 黑框框显示 Bad File Descriptor 错误，类似下图

```
To access the notebook, open this file in a browser:
file:///C:/Users/%E5%B8%85%E5%B8%85%E9%A3%9E%E7%8C%AA/AppData/Roaming/j
Or copy and paste one of these URLs:
http://localhost:8888/?token=47cf2aaa44780278c4e644e8c277c5088e44a5cca0
or http://127.0.0.1:8888/?token=47cf2aaa44780278c4e644e8c277c5088e44a5cca0
[I 18:56:16.832 NotebookApp] 302 GET / (::1) 0.000000ms
[W 18:56:44.320 NotebookApp] 404 GET /nbextensions/widgets/notebook/js/extension
00ms referer=http://localhost:8888/notebooks/TFPractise/Untitled.ipynb
Bad file descriptor (C:\projects\libzmq\src\epoll.cpp:100)
Bad file descriptor (C:\projects\libzmq\src\epoll.cpp:100)
```

解决办法：关闭当前 cmd 黑框框，重新打开 cmd 黑框框，在确保网络连通的情况下，依次执行如下两条命令 `pip uninstall pyzmq`；`pip install pyzmq==19.0.2 --user`，两条命令成功运行后（运行时没有出现 Error 信息），重新启动 jupyter-lab 即可。

(2) jupyter-lab 的浏览器界面需要输入 token

请复制你 cmd 黑框框中的 **http 开头的网址** (两个网址中的任意一个, 类似下图) 到浏览器打开。

```
To access the notebook, open this file in a browser:
    file:///C:/Users/Administrator/AppData/Roaming/jupyter/runtime/nbserver-58048-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=adda914e1a9f67dc96d78601ae42e1dae7cb10efa3bc06b3
    or http://127.0.0.1:8888/?token=adda914e1a9f67dc96d78601ae42e1dae7cb10efa3bc06b3
[W 23:49:14.572 LabApp] Could not determine jupyterlab build status without nodejs
[I 23:49:16.360 LabApp] Kernel started: bde3aa9e-ea4a-4fa0-a89c-e46129c63b26
```

(3) 运行 BMI_calculation 代码时, 输入身高体重后无法继续计算

确保在输入身高体重信息后, 按 **Enter 键确认**, 因为 `input` 函数在接受键盘输入后, 需要用户确认输入, 以继续运行后续程序代码。

(4) 明明在 jupyter-lab 里写了代码, 却在本机上找不到

请在启动 jupyter-lab 后, 进入到桌面 **Desktop**, 然后新建 **Notebook**, **重命名为有意义的英文名字**, 再写代码, 写代码过程中, **一定要多按保存键**, 快捷键为 **Ctrl + S**。

(5) 课程配套代码的打开与运行

本地：

- ①将课程网站的代码下载到本机，建议下载到桌面，便于查找；
- ②启动 jupyter-lab，从左侧文件导航栏找到课程代码，双击打开；
- ③ 点击▶，运行选中的当前代码行；点击▶▶，运行全部代码。

云端：

- ①将课程网站的代码下载到本机，建议下载到桌面，便于查找；
- ② 打开魔搭在线 jupyter 环境，点击↑，上传课程代码到云端；
- ③从左侧文件导航栏找到刚才上传的课程代码，双击后打开；
- ④ 点击▶，运行选中的当前代码行；点击▶▶，运行全部代码。

注意：一定要动手编写和运行课程代码，只看代码几乎没用。

表达式和语句

表达式（Expression）是由操作符和操作数组成的代码单元，用于计算和**返回一个值**。

`3 + 5` # 是一个算术表达式，使用了算数操作符 `+`

`abs(-7)` # `abs(-7)` 是一个函数调用表达式，使用了内置函数 `abs` 求绝对值

表达式可以包含常量、变量、函数调用和操作符等。根据使用的操作符类型，表达式可以分为不同的类别。

语句（Statement）是一条完整的指令，通常执行操作或更改程序的状态，语句**不一定返回值**。

`x = 5` # 赋值语句

算数表达式

算术表达式涉及数字和算术操作符（如 +、-、*、/等），用于执行数学运算。

表 1: Python 常用算术操作符

操作符	含义	优先级	示例	结果
()	括号（最高优先级）	最高	<code>(3 + 2) * 4</code>	<code>20</code>
<code>**</code>	幂运算	高	<code>2 ** 3</code>	<code>8</code>
<code>*</code>	乘法	中	<code>3 * 4</code>	<code>12</code>
<code>/</code>	除法	中	<code>10 / 2</code>	<code>5.0</code>
<code>//</code>	整数除法	中	<code>10 // 3</code>	<code>3</code>
<code>%</code>	取余	中	<code>10 % 3</code>	<code>1</code>
<code>+</code>	加法	低	<code>3 + 5</code>	<code>8</code>
<code>-</code>	减法	低	<code>10 - 4</code>	<code>6</code>

算数表达式

涉及括号、指数运算、整数除法、取余和多级嵌套运算

```
(3 + 5 * 2) ** 2 // (4 % 3 + 2 * (5 - 3))
```

该表达式包含多个优先级相同的运算符，优先级相同则从左到右计算

```
5 ** 3 // 7 % 4 + 6 * (7 // 2)
```

该表达式包含多个不同优先级的操作符，需要掌握优先级表并正确使用括号

```
(2 ** 3) * (3 // 2 + 4 % 3) ** 2 - 5 / 2 + 7
```

关系表达式

关系表达式（Relational Expressions）用于比较两个操作数，并返回一个布尔值，即 `True` 或 `False`。

表 2: Python 常用关系操作符

操作符	含义	示例	结果
<code>==</code>	等于	<code>5 == 5</code>	<code>True</code>
<code>!=</code>	不等于	<code>5 != 3</code>	<code>True</code>
<code>></code>	大于	<code>10 > 5</code>	<code>True</code>
<code><</code>	小于	<code>3 < 7</code>	<code>True</code>
<code>>=</code>	大于或等于	<code>4 >= 4</code>	<code>True</code>
<code><=</code>	小于或等于	<code>6 <= 8</code>	<code>True</code>

逻辑表达式

逻辑表达式由逻辑操作符和布尔值组成，计算结果为布尔值 `True` 或 `False`。

表 3: Python 中的逻辑操作符

操作符	含义	优先级
<code>and</code>	逻辑与：仅当所有操作数为 <code>True</code> 时返回 <code>True</code>	中等
<code>or</code>	逻辑或：只要有一个操作数为 <code>True</code> 就返回 <code>True</code>	低
<code>not</code>	逻辑非：将 <code>True</code> 变为 <code>False</code> ，反之亦然	高

表达式 `not (True or False and True)` 会按照以下顺序计算：

- ① 首先执行 `and`，因为它的优先级高于 `or`，结果是 `False`。
- ② 然后执行 `or`，其结果为 `True`。
- ③ 最后，`not` 将结果反转为 `False`。

逻辑表达式

逻辑运算符的短路特性 (Short-circuiting): 逻辑运算符会从左到右进行求值, 一旦能够确定整个表达式的最终结果, 就会立即停止计算剩余的部分。

```
# 逐步解析每个表达式, 理解 not、and 和 or 的结合使用如何影响最终结果
```

```
x = 8; y = 20; z = 15
```

```
not (x > 5 and y == 20) or (z < 10)
```

```
p = 3; q = 6; r = 9
```

```
(p + q > r) and (q != r or p <= q)
```

```
# 结合短路特性, 理解下面表达式的执行顺序
```

```
x = 0; y = 4; z = 7
```

```
x > y or (y < z and z > x) or not y
```

身份运算符 (代码示例)

Python 的身份运算符 `is` 和 `is not` 用于判断两个变量是否引用了内存中的同一个对象，它通过比较对象的唯一标识 (`id()`) 来实现。

当两个变量引用同一个对象时，`is` 返回 `True`。例如：

```
a = [1, 2, 3]
b = a
print(a is b) # 输出: True
c = [1, 2, 3]
print(a is not c) # 输出: True
```

这里，`a` 和 `b` 引用的是同一个列表对象，所以 `a is b` 为 `True`，即使 `a` 和 `c` 的内容相同，但它们在内存中引用了不同的对象。

区别：相等性运算符 `==` 检查的是值 (value) 的一致性。两个对象的值可以相等，但它们在内存中可能是两个独立的对象。

成员关系运算符

Python 的成员关系运算符 `in` 和 `not in` 用于测试一个值是否存在于一个容器 (container) 或可迭代对象中, 如果值存在于容器中, 返回 `True`, 否则返回 `False`。

```
fruits = ['apple', 'banana', 'cherry']
print('banana' in fruits)  # 输出: True
print('mango' not in fruits)  # 输出: True
```

这两个运算符不仅适用于序列 (如列表、元组、字符串), 还适用于其他重要的容器类型, 如字典、集合。

```
my_dict = {'name': 'Alice', 'age': 30}
print('age' in my_dict)  # 输出: True
print('Alice' in my_dict)  # 输出: False (因为 'Alice' 是值,
↪ 不是键)
```

操作符的优先级

在 Python 中，操作符的优先级决定了在表达式中各个操作符的运算顺序。高优先级的操作符会先计算，除非使用括号明确改变运算顺序。以下是 Python 中所有操作符的优先级表，从最高到最低优先级。

表 4: Python 操作符优先级 (从高到低)

优先级	操作符	描述
1	<code>()</code>	圆括号用于改变优先级
2	<code>**</code>	指数运算（从右到左）
3	<code>+x</code> , <code>-x</code> , <code>~x</code>	一元加、减、按位取反
4	<code>*</code> , <code>/</code> , <code>//</code> , <code>%</code>	乘法、除法、取整除、取余
5	<code>+</code> , <code>-</code>	加法、减法
6	<code><<</code> , <code>>></code>	位移运算符
7	<code>&</code>	按位与
8	<code>^</code>	按位异或
9	<code> </code>	按位或
10	<code>==</code> , <code>!=</code> , <code>></code> , <code><</code> , <code>>=</code> , <code><=</code> , <code>is</code> , <code>is not</code> , <code>in</code> , <code>not in</code>	比较运算符、身份运算符、成员运算符
11	<code>not</code>	逻辑非
12	<code>and</code>	逻辑与
13	<code>or</code>	逻辑或

Python 中，变量是一个名称 (name)，它指向内存中的一个对象 (object)。真正存储数据的是对象，而变量只是引用该对象的方式。对象可以存储不同类型的数据，如，整数、字符串、布尔值等。

```
x = 10          # 整数
name = "Alice"  # 字符串
is_active = True # 布尔值
```

通过赋值操作符 `=`，可以将某个值赋给变量 (将变量绑定到某个对象)，也可以同时为多个变量赋值 (将多个名称同时绑定到多个对象上，也称为解包)：

```
age = 25
a, b, c = 1, 2, 3
```

变量命名

变量的名称可以是字母、数字和下划线的组合，但**不能以数字开头**。此外，变量命名应遵循一系列最佳实践和约定，以确保代码的可读性、可维护性以及团队协作时的易理解性：

- **使用描述性和有意义的名称：**变量名称应清晰地描述变量的用途或内容。避免使用简短、不具描述性的名称（如 `x`、`y`），而应使用如 `student_age` 这样的名称，便于理解。
- **遵循命名约定：**Python 推荐使用 **snake_case** 作为变量命名规范，即用小写字母并以下划线分隔单词，如 `first_name`。
- **避免使用保留关键字：**不要使用 Python 的保留字（如 `if`、`else`、`for`）作为变量名，以免导致语法错误。[查看 Python 保留关键字](#)

变量命名

问题: 根据 Python 的变量命名规则, 找出每个变量名中的问题并进行纠正。

提示: 变量名不能以数字开头、不能包含特殊字符 (如 # 和空格), 并且避免使用 Python 的保留关键字。

```
2ndPlaceWinner = "John"  
total#ofBooks = 150  
User Email = "example@domain.com"  
float = 7.5  
VARiable = 100
```

数字-整数

1. **二进制 (Binary)** 通过在数字前加前缀 `0b` 或 `0B` 表示二进制。例如，`0b1010` 表示二进制数 1010，其十进制值为 10。
2. **八进制 (Octal)** 使用前缀 `0o` 或 `0O` 来表示八进制数。例如，`0o123` 表示八进制数 123，对应的十进制值为 83。
3. **十进制 (Decimal)** 默认情况下，整数以十进制形式表示，无需添加任何前缀。例如，`123` 即表示十进制数 123。
4. **十六进制 (Hexadecimal)** 前缀 `0x` 或 `0X` 用于表示十六进制数。例如，`0x1A` 表示十六进制数 1A，对应的十进制值为 26。

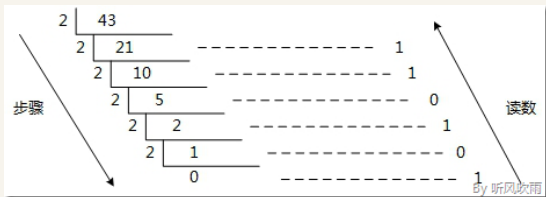


图 1: 十进制转换为 n 进制的除 n 取余法 (43→0b101011)

Python 提供了内置函数来进行进制转换：

- `bin()`：将整数转换为二进制字符串表示，例如 `bin(10)` 结果为 `'0b1010'`。
- `oct()`：将整数转换为八进制字符串表示，例如 `oct(83)` 结果为 `'0o123'`。
- `hex()`：将整数转换为十六进制字符串表示，例如 `hex(26)` 结果为 `'0x1a'`。

`int()` 函数可以将字符串形式的数值转换为指定进制的整数：

```
int("0b1010", 2)  # 结果为 10
int("0o123", 8)   # 结果为 83
int("0x1A", 16)   # 结果为 26
```

[更多示例代码](#)

数字-浮点数 (代码示例)

浮点数 (float) 用于表示带有小数部分的实数, 由于浮点数在计算机中是用二进制表示的, 某些十进制小数在二进制中不能精确表示, 因此, Python 浮点数存在精度限制。由于这种固有的不精确性, 应避免使用 `==` 直接比较两个浮点数是否相等:

```
# 0.1 在 Python 内部的表示并非精确的 0.1, 而是一个近似值
1-0.9 == 0.1 # False

# 使用 math.isclose() 是比较浮点数的推荐方法
import math
math.isclose(0.1, 1-0.9) # True
```

科学计数法的形式为 $N \times 10^M$, 在 Python 中表示为 `NeM` 或 `NEM`, 例如 `1.23e4` 表示的数值是 1.23×10^4 (即 12300)。当数值大于 `1e16` 或小于 `1e-4` 时, Python 会自动将浮点数以科学计数法显示。

数字-复数 (代码示例)

复数表示为 $a + bj$ 的形式，其中 a 是实部， b 是虚部， j 表示虚数单位（即 $\sqrt{-1}$ ）。Python 中用 j 代替了传统的 i 来表示虚部。可以通过直接赋值和 `complex` 函数两种方式在 Python 中定义复数，使用 `.real` 和 `.imag` 属性，分别访问复数的实部和虚部。虚部为 1 时，不可以省略不写。

```
# 直接赋值
```

```
z = 3 + 4j
```

```
# 使用 complex() 函数，该函数接受两个参数，分别表示实部和虚部
```

```
z = complex(3, 4)
```

```
z = 3 + 4j
```

```
print(z.real) # 输出: 3.0
```

```
print(z.imag) # 输出: 4.0
```

布尔值 (代码示例)

布尔值 (Boolean) 是一种表示逻辑真值的数据类型，只有两个可能的取值：`True` 和 `False`。

布尔值可以通过比较运算符 (如 `<`, `>`, `==` 等) 来产生。例如，表达式 `5 > 3` 会返回 `True`，而 `2 == 3` 则返回 `False`。

除了直接的布尔值，Python 中的任何数据类型都可以被转换为布尔值，非零数字、非空对象会被视为 `True`，而 `0`、`None` 和空对象 (如空字符串、空列表) 会被视为 `False`。

```
bool(1)      # 返回 True
bool(0)      # 返回 False
bool("")     # 返回 False
bool("abc")  # 返回 True
```

布尔值参加算数运算时，`True` 和 `False` 分别被视作数字 1 和 0。

类型转换 (代码示例)

类型转换： 调用内置函数将一种数据类型转换为另一种数据类型。

```
num_str = "123"  
num_int = int(num_str) # 将字符串 '123' 转换为整数 123
```

通过类型转换，可以明确指定期望的数据类型，以防止由于类型不匹配导致的错误，例如，使用用户输入的数据进行算数运算时，需要将字符串转换为整数。常见的类型转换函数包括：

- `int()`：将其他类型转换为整数；
- `float()`：将其他类型转换为浮点数；
- `str()`：将其他类型转换为字符串。

其他重要数据类型 (本次课了解即可)

- ❶ 字符串: 用于存储文本数据, 例如, `name = "Alice"`
- ❷ 列表: 有序的、可变的元素集合, 例如,
`fruits = ["apple", "banana", "cherry"]`
- ❸ 元组: 有序的、不可变的元素集合, 例如, `coordinates = (10, 20)`
- ❹ range: 用于生成一系列数字的范围对象, 例如 `range(0, 10)`
- ❺ 字典: 用于存储键值对, 例如,
`person = {"name": "Alice", "age": 25}`
- ❻ 集合: 无序且唯一的元素集合, 例如,
`unique_numbers = {1, 2, 3}`
- ❼ 不可变的集合: 不可变的集合, 元素不可更改的集合。
- ❽ None: 表示“空”或“无值”的特殊对象。

- 命令行运行 **Python** 脚本:
`python BMI_calculation.py`
- 使用 **IDE** 运行 **Python** 脚本:
自行尝试
- 使用 **Jupyter Notebook** 编写和运行代码:
这是第一节课后作业

唯手熟尔

写代码初期，遇到错误，多看看这几条：

- 避免拼错标识符，如变量名，函数，语句等；
- 避免使用中文符号，如引号，逗号，括号等；
- 引号、括号通常成对使用，如，有左括号也要有右括号，左边有引号，右边引号也别漏；
- 注意书写格式 (冒号，缩进，对齐)。

后面小节的内容本次课了解即可
本课程后续章节会展开详细讲解

函数是一个用于组织和重用代码的核心工具。函数将相关的代码逻辑封装在一个命名的代码块中，使程序更加模块化和易于维护。

函数通过 `def` 关键字定义，后面跟随函数名称和圆括号。圆括号中可以包含参数，这些参数是函数在调用时需要传入的值。以下是一个简单的函数示例：

```
# 定义函数
def add(a, b):
    return a + b

# 调用函数
result = add(5, 3)  # result 的值为 8
```

常用内置函数

Python 中的内置函数提供了许多常用的功能，帮助简化编程任务。

表 5: 常用 Python 内置函数

函数	描述	示例
<code>print()</code>	将对象打印到控制台输出	<code>print("Hello, World!")</code>
<code>input()</code>	从用户输入读取一行字符串	<code>name = input("Enter your name: ")</code>
<code>abs()</code>	返回数字的绝对值	<code>abs(-7)</code> 返回 7
<code>pow()</code>	计算第一个参数的幂次,支持模运算	<code>pow(2, 3)</code> 返回 8
<code>len()</code>	返回对象的长度 (如字符串、列表等)	<code>len("Python")</code> 返回 6
<code>sum()</code>	返回可迭代对象中所有元素的和	<code>sum([1, 2, 3])</code> 返回 6
<code>min()</code>	返回可迭代对象中的最小值	<code>min([5, 3, 9])</code> 返回 3
<code>max()</code>	返回可迭代对象中的最大值	<code>max([5, 3, 9])</code> 返回 9
<code>round()</code>	将浮点数四舍五入到指定的小数位数	<code>round(3.456, 2)</code> 返回 3.46
<code>type()</code>	返回对象的数据类型	<code>type(42)</code> 返回 <code><class 'int'></code>

输入与输出是程序与用户交互的基础功能。Python 提供了内置的 `input()` 和 `print()` 函数，分别用于接收用户输入和屏幕输出结果。

`input()` 函数用于从用户处接收输入。调用该函数时，程序会暂停运行并等待用户输入，直到用户按下回车键。输入的内容会作为字符串返回。如果需要将用户输入转换为其他类型（如整数或浮点数），可以使用类型转换函数，例如：

```
age = int(input("Enter your age: ")) # 将输入转换为整数
```

`print()` 函数用于向控制台输出结果。它可以接收多个参数，并将它们以空格分隔输出。`print()` 还可以使用可选的 `sep` 和 `end` 参数，来控制输出的分隔符和结束符。例如：

```
print("Hello", "World", sep=", ", end="!") # 输出: Hello, World!
```


模块是 Python 中一种将相关功能组织在一起的方式。通过导入模块，可以使用模块中定义的函数、常量和对象。例如，Python 自带的 `math` 模块就是一个常用的数学模块，它提供了基本的数学运算函数和常量。

要使用 `math` 模块，首先需要通过 `import` 语句将其导入。导入后，可以通过 `math.` 前缀访问模块中的函数和常量。

```
import math

result = math.sqrt(16) # 使用 math 模块计算平方根
print(result) # 输出: 4.0
```

流程控制（Control Flow）是指程序代码的执行顺序。通常情况下，Python 程序会从上到下、逐行执行代码。但通过使用流程控制语句，程序可以根据特定条件、循环结构等来控制代码的执行顺序。主要流程控制结构包括：顺序执行、选择（条件语句）和重复（循环语句）。

条件语句是 Python 编程中控制流程的基础，它允许程序根据特定条件执行不同的代码块。常见的条件语句包括 `if`、`elif` 和 `else`。

条件语句

`if` 语句检查条件是否为真，如果条件成立，则执行相应的代码块：

```
if condition:  
    # 条件为真的情况下执行的代码
```

当需要处理多个条件时，可以使用 `elif` 来扩展条件判断。

`else` 则用于在所有条件都不成立时执行默认代码块：

```
if condition1:  
    # 条件 1 为真时执行  
elif condition2:  
    # 条件 2 为真时执行  
else:  
    # 所有条件为假时执行
```

假设我们在一个电子商务平台上，根据销售金额对产品进行分类。可以使用条件语句判断产品属于“低销量”、“中等销量”还是“高销量”：

```
sales = 15000 # 假设这是产品的销售金额

if sales > 20000:
    print(" 高销量产品")
elif 10000 <= sales <= 20000:
    print(" 中等销量产品")
else:
    print(" 低销量产品")
```

循环语句

for 循环和 **while** 循环是 Python 中常用的控制流程结构，广泛用于商业数据分析中的多种任务，如遍历数据集或动态处理数据。

for 循环用于遍历一个序列（如列表、元组或字符串），每次迭代提取一个元素。常见的使用场景包括逐行读取数据、遍历数据集中每一条记录等。

while 循环根据条件执行代码块，直到条件不再满足为止。它通常用于需要不确定的迭代次数时，例如数据监控或数据收集过程。

例如，假设有一个代表公司季度销售额的列表，可以用 `for` 循环来计算总销售额：

```
sales = [12000, 18000, 25000, 30000]
total_sales = 0

for sale in sales:
    total_sales += sale

print(f" 总销售额为: {total_sales}")
```

例如，假设我们想持续监控某一产品的库存量，当库存低于某个阈值时自动停止销售：

```
stock = 50  # 初始库存
threshold = 10  # 库存阈值

while stock > threshold:
    print(f" 当前库存: {stock}, 继续销售...")
    stock -= 5  # 每次销售 5 个
print(" 库存低于阈值, 停止销售")
```

THE END