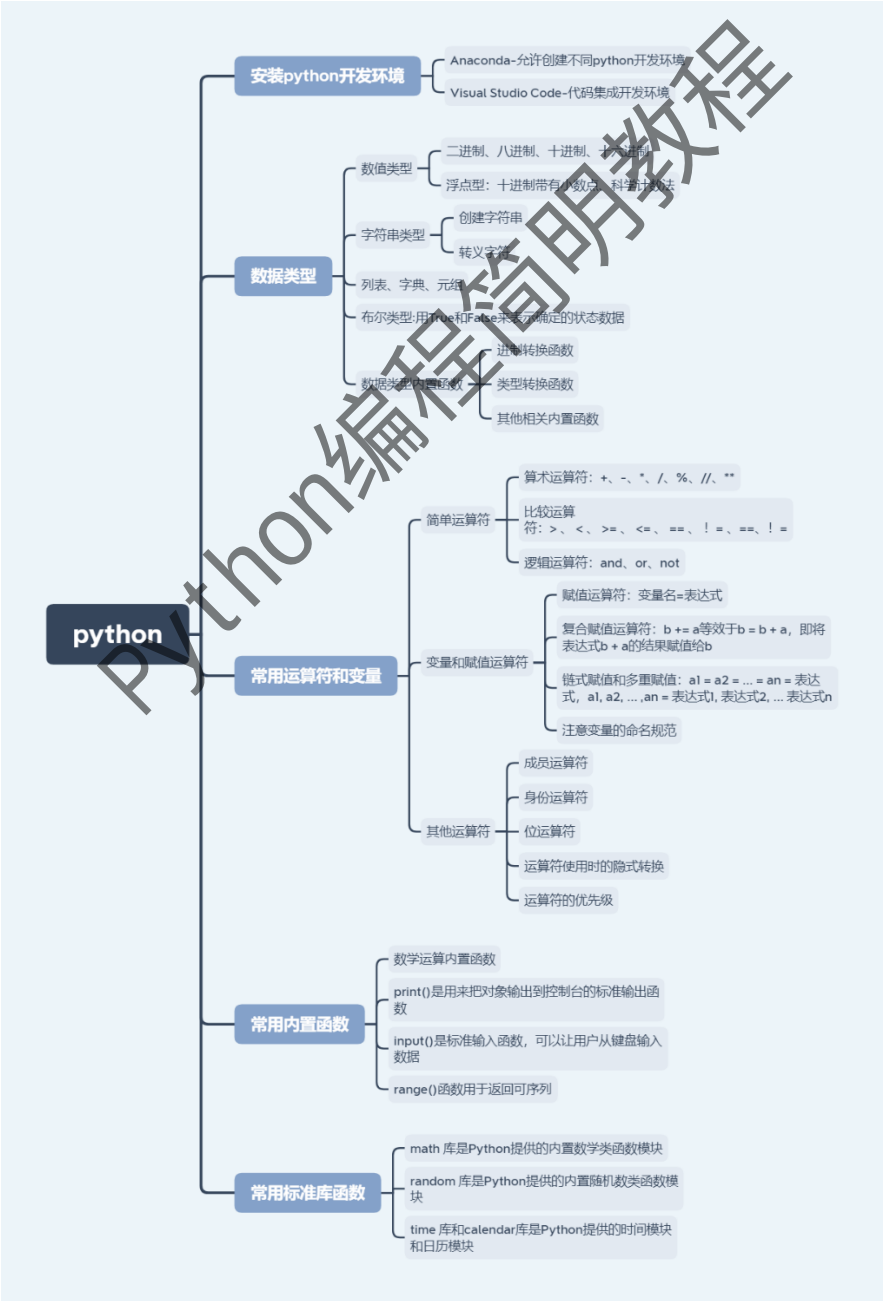
1. Python 语言基础

✍ 知识目标：	💻 技能目标
1. 能完成 Anaconda 和 Visual Studio code 编程环境的搭建。 2. 掌握常用数据类型、运算符和变量 3. 展望常用内置函数和标准库函数	1. 具有初步的编程能力 2. 具有合理命名变量和使用注释的能力 3. 能视情况合理使用不同的数据类型、运算符 4. 能视情况合理使用

⌘ 学习地图：



⌘

1.1. 体验 Python

Python 是一款解释型、面向对象、动态数据类型的高级语言。Python 组织直观，又具有开源，扩展库，是初学编程的好选择，也是目前工业界和学术界广泛使用的程序语言。

运行 Python 程序有两种方式：

1) 交互式：Python 解释器即时响应每条输入代码给出结果。常用于少量代码的调试。
2) 文件式：在 Python 解释器中建立后缀名为 .py 的程序文件，调用并执行该文件。

打开 Python 官网 https://www.python.org/shell/ ，如图 1-1。其中就内嵌了交互式环境。

图 1-1 Python 官网在线开发平台

【示例-1】在 Python 官网交互式编程环境中，执行如下操作。

```
01  >>> print('hello python!')
    hello python!
02  >>> 1 + 1.0
    2.0
03  >>> 2 < 3
    True
```

语句 01、02、03 中含 python 命令提示符 >>> ，表示后面可直接输入要执行的命令。阴影部分为【回车键】执行语句 01、02、03 后的输出结果。具体解释如下：

- ✓ 语句 01："hello python!" 是字符串（string）[1]；print() 是 python 中的固有函数，调用该函数可执行输出显示字符串的内容，字符串在函数中称为参数。
- ✓ 语句 02：称 1 + 1.0 为表达式。其中整数 1 为整型（int），小数 1.0 为浮点型（float），+ 为算术运算符，与数学中作用相同。表达式在交互式环境运行后直接输出结果。
- ✓ 语句 03：2 < 3 同样为表达式。< 为比较运算符，与数学中作用相同。表达式执行后输出 True，表示正确；如错误输出 False。True 和 False 属布尔类型（bool）。

[1] 字符串两边用双引号或单引号包裹以区分其他代码。

2

1.2. 开发环境安装

本节介绍如何在本机安装 Python 开发环境。本书使用 Anaconda 和 Visual Studio Code。

1.2.1. 安装和应用 Anaconda

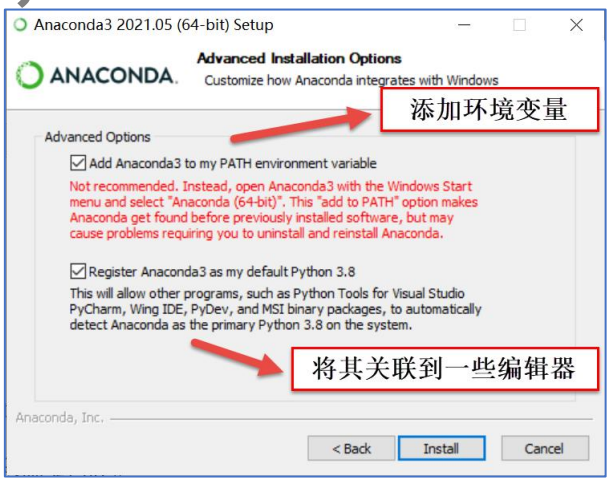
Anaconda 是一个开源的 Python 发行版本，集成了 NumPy、SciPy、Pandas、Matplotlib 和 scikit-learn 等多个科学包，且可使用 conda 命令创建多个环境和安装第三方库。

1. 安装 Anaconda

如图 1-2，在 https://www.anaconda.com/products/individual，完成 Anaconda 个人版下载。

图 1-2 下载 Anaconda

在 Windows 安装 Anaconda 十分方便[1]。如图 1-3，建议添加环境变量和关联编辑器。



[1] 本教程主要介绍 windows 环境下的安装，对于 linux 环境和 mac 环境，读者可参阅其他书籍。

3

图 1-3 安装 Anaconda

2. 环境管理

使用独立稳定的 Python 环境很重要，Anaconda 允许创建不同的 python 开发环境。如图 1-4，在开始菜单栏，按步骤选择 Anaconda Prompt 命令运行 Anaconda 环境。

图 1-4 运行 Anaconda

输入如下命令：

```
conda create -n env_name python=version anaconda
```

其中，`env_name` 为设置环境的名字，如命名为 `iflytek`；`python=version` 为 python 版本，如 `python=3.7`；`anaconda` 为可省命令，省略时仅安装一些 Python 基础包，添加时则具有 Anaconda 的所有包，建议添加。

创建后用如下命令激活环境，如将 `activate` 改为 `deactivate` 则退出环境。：

```
conda activate env_name
```

通过 `conda list` 命令可查看当前环境安装的包：

```
conda list
```

通过 `pip` 命令则可进行包的卸载和安装，其中 `uninstall` 为卸载， `install` 为安装。如安装

4

下载过慢，建议使用国内镜像。现使用百度镜像，完成 lxml 扩展包的重装[1]。

```
pip uninstall lxml
pip install lxml –i https://mirror.baidu.com/pypi/simple
```

3. 测试运行

安装好 iflytek 环境后，输入 python，就可打开与图 1-1 类似的交互式编程环境。使用 Ctrl+Z 键可退出交互式编程环境。

1.2.2. 安装 Visual Studio Code

Visual Studio Code 是微软的一个免费代码集成开发环境，本书使用该关键执行文件式 Python 程序。

1. 下载和安装

Visual Studio Code 下载地址为：https://code.visualstudio.com/。单击 Download for Windows 即可。其安装也十分简单，建议勾选所有附加任务。

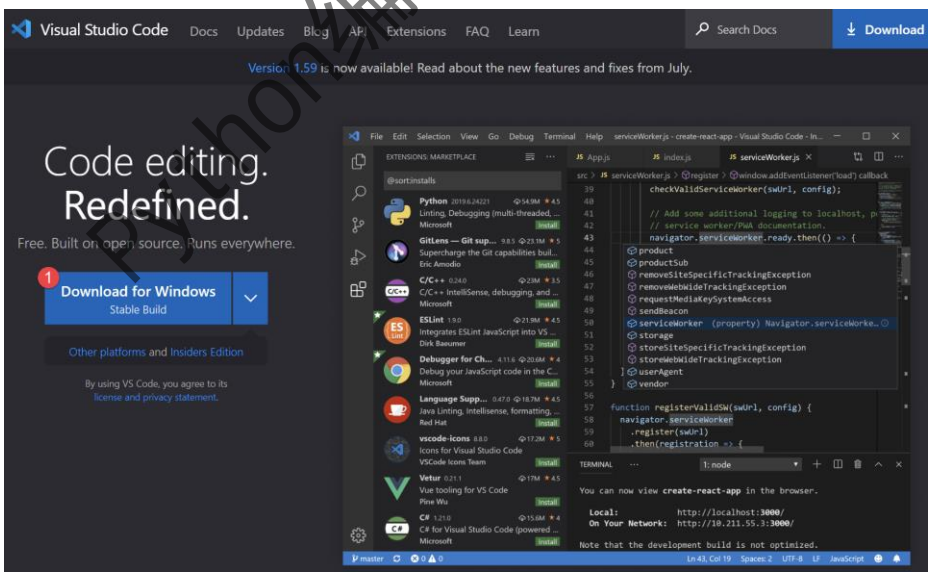


图 1-5 Visual Studio Code 下载页面

2. 配置插件

打开 Visual Studio Code ,可完成颜色主题设置。并如图 1-6 进入插件管理页面。在商店

[1] 建议大家重新安装该扩展包，原始的包有时在后续操作中会报错。

5

中搜索 Chinese（Simplied） Lang，安装重启后即可完成中文设置。其他建议安装的插件，如表 1-1。插件安装完成，为防止中文乱码，建议在设置搜索框输入：Files.autoGuessEncoding，如图 1-7。

图 1-6 配置中文插件

图 1-7 防止中文乱码

表 1-1 其他常用插件

插件	作用描述
Python	能自动识别本地解释器以及 conda 或 pipenv 的虚拟环境
Prettier	Beautify 插件可以快速格式化你的代码格式
ESLint	代码检查工具

6

Trailing Spaces	提醒不必要的空格
Bracket Pair Colorizer 2	自动检查括号
Kite	代码补全
indent-rainbow	高亮（每行代码之前的）缩进
Path Intellisense	自动路径补全
Regex Previewer	正则校验测试
Data Preview	支持直接在 vscode 对 csv，xlsx 等数据做可视化、统计探索
open in browser	在文件右键 可以选择在浏览器预览

3. 运行程序

现用 Visual Studio Code 运行程序，如图 1-8。创建文件夹，右键 Code 打开。

图 1-8 创建文件夹

打开后按图 1-9 步骤进行操作，创建章节目录 chap01，在其中创建 python 源文件 ex01.py，选择 Anaconda 中创建好的 iflytek 环境。。

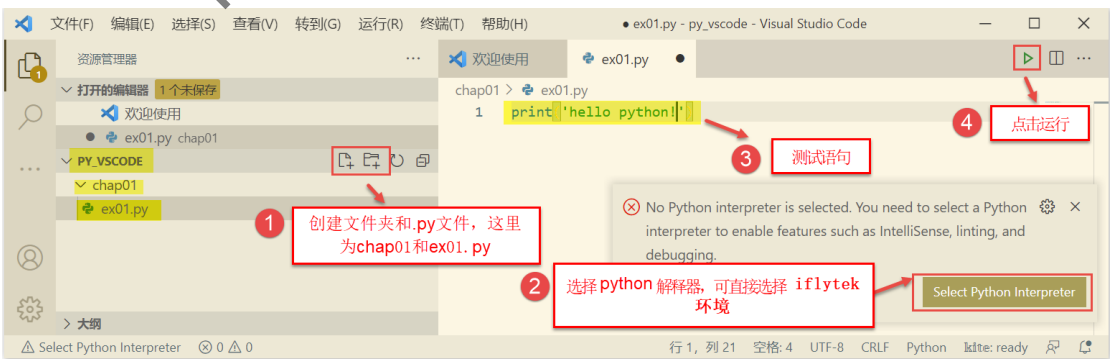


图 1-9 运行 python

在主窗口输入如下语句，

```
01  print('hello  python!')
02  1+1.0
```


7

```
03    2 > 3
```

点击运行，此时以文件式执行 python 文件。终端输出如下：

```
01    hello python!
```

对照【示例-1】，语句 02 和语句 03 并不能直接输出相应的结果。换句话说，文件式执行时必须使用 print()输出函数，才能在控制台得到相应的输出。

1.3. 数据类型[1]

【示例-1】中的字符串、整数、浮点型、布尔类型在 Python 中统称数据类型。除上述外，python 中的数据类型还包括列表，元组和字典。

1.3.1. 数值类型

数值类型用于存储数据，常用的有整型和浮点型。该类型支持算术运算、比较运算。

1. 整型

整型是不含小数点的数值。整型表示方式包括：二进制、八进制、十进制和十六进制。

表 1-2 整型的 4 种进制[2]

进制	引导符号	输入	对应输出（默认十进制）
十进制	无	365	365
二进制	0B 或 0b	0b101101101	365
八进制	0O 或 0o	0o555	365
十六进制	0X 或 0x	0x16d	365

2. 浮点型

浮点型是含小数点的数值。浮点型的表示有两种：

1) 十进制带有小数点：如 1.2 和-1.2。
2) 科学计数法：使用字母 E 或 e 作为幂的符号。

[1] 对于字符串、列表、字典本章只进行简要介绍。详细介绍将在第 2 章进行。
[2] 内置函数 hex()，oct()，bin()可分别将十进制转换为十六进制、八进制和二进制，这里不再举例。

8

1.3.2. 字符串类型

很多编程语言都包括字符串（string），字符串是由引号包裹起来的一个或多个字符。

1. 创建字符串

根据字符串是否自动换行，可将字符串分为单行字符串和多行字符串。其中用单引号（'）、双引号（"）等包裹起来是单行字符串[1]；用三单引号（'''）或三双引号（"""）等包裹起来的是多行字符串。

【示例-2】字符串演示[2]。

```
print('')      #创建空字符串
print('爱')     #Python没有单独的字符类型，一个字符就是长度为1的字符串
print("好好学习，天天向上！")     #双引号字符串，支持文本
print("Let's go!")
print('He said "Hello world!"')
print('''我和我的祖国   一刻也不能分割
无论我走到哪里   都流着一首赞歌
''')
```

点击运行。终端输出如下：

```
爱
好好学习，天天向上！
Let's go!
He said "Hello world!"
我和我的祖国   一刻也不能分割
无论我走到哪里   都流着一首赞歌
```

2. 转义字符

但如果字符串中既含单引号又含双引号，如 She said "Let's go!"，则此时需对该字符串使用反斜杠（\）来转义字符。\ 不计入字符串内容。表 1-3 给出了常用的转义字符：

表 1-3 反斜杠“\”常用转义特殊字符

转义字符	\在行尾	\\	\'	\"	\a	\b	\n	\v	\t
输出描述	续行符	反斜杠	单引号	双引号	响铃	退格	换行	纵向制表符	横向制表符

而在一些特殊情况下，不想转义字符生效，如希望输出文件路径：“D:\today\new_list”，

[1] 既用单引号又用双引号界定，可防止字符串本身含单引号或双引号。此时根据情况选用，避免出错。
[2] #后面的为注释语句，用于提高程序的可读性。该段代码不会被执行。

9

此时**可用 r 或 R 来取消反斜杠转义**，这样的字符串称为**原始字符串**。

【示例-2 续】转义字符演示

```
09  print("He said\"I'm OK!\"")
10  print("水果：\t 苹果\t 梨\t 香蕉\n 个数：\t10\t9\t8")
11  print('D:\today\new_list')
12  print(r'D:\today\new_list')
```

点击运行。终端输出如下：

```
09  He said"I'm OK!"
10  水果：  苹果    梨      香蕉
    个数：  10      9       8
11  D:      oday
    ew_list
12  D:\today\new_list
```

1.3.3. 列表和字典

很多情况下，单一的数据会组成复合的数据类型。常用的包括列表、元组和字典。

1) 列表（list）由一系列按特定顺序排列的元素组成。列表中相邻的数据元素间用半角字符逗号（,）隔开，所有数据元素被一对半角字符中括号（[]）包裹。
2) 元组（tuple）与列表相似。不同的是元组中的数据元素不可变，所有数据元素被一对半角字符小括号（()）包裹。
3) 字典（dict）是一种无序的数据类型。字典中的元素使用键值对（key-value）的形式并用半角字符冒号（:）冒号来分隔，相邻的数据元素间用半角字符逗号（,）隔开，所有数据元素被一对半角字符花括号（{}）包裹。

【示例-3】演示列表和字典

```
01  >>> [] #创建空列表
    []
02  >>> {} #创建空字典
    {}
03  >>> [100, 90, 80]  #创建学生成绩列表
    [100, 90, 80]
04  >>> {'语':100, '数':90, '外':80}  #创建语数外成绩字典
    {'语': 100, '数': 90, '外': 80}
```


10

1.3.4. 布尔类型

布尔类型用来表示两个确定的状态数据，它有真（True）和假（False）两个值。需注意的是，python 中任何值为 0 或空的数据，如一个空字符串，一个空列表，一个空字典，它们的布尔值均为 False。

1.3.5. 数据类型内置函数（Built-in Functions, BIF）

此外，python 内置了一些函数用于查看数据的类型或完成数据类型的转换。

1. 进制转换函数

通过进制转换函数，可完成进制间的转换。如表 1-4 所示

表 1-4 进制转换

函数名	功能描述	输入示例	对应输出
bin(x)	将整数 x 转换为二进制数	bin(365)	0b101101101
oct(x)	将整数 x 转换为八进制数	oct(365)	0o555
hex(x)	将整数 x 转换为十六进制数	hex(365)	0x16d

2. 类型转换函数

通过类型转换函数，可以完成多个数据类型间的相互转换。如表 1-5 所示

表 1-5 类型转换函数

函数名	功能描述
int([x[, base]])[1]	将相关类型转换为整数；base 表示进制（默认为 10），可以是[2,6]范围内的值或者 0。
bool([x])	将 x 转换为布尔数据类型。如果无参数则返回 False
float([x])	将一个字符串或数返回为浮点数据类型。如果无参数则返回 0.0。
str([object])	将对象（object）转换为字符串类型。
eval(str)	将字符串 str 当成有效的表达式求值并返回计算结果。

【示例-4】类型转换函数演示

```
01    >>> print(bool(0), bool(1), bool('a'), bool([]))
      False True True False
```

[1] 函数中参数用半角字符（,）进行分割，参数被半角字符中括号（[]）包裹，表示该参数可缺省时，缺省时使用默认值。同时逗号后，运算符两边、关键词后常添加空格，以便阅读方便，无实际语法意义。

```
02  >>> print(int(True), int(3.14), int('2'), int('0x10', 0))
    1 3 2 16
03  >>> print(int('0x10'))  #无法解析该字符串，程序报错
    Traceback (most recent call last): File "<stdin>", line 1, in <module>
    ValueError: invalid literal for int() with base 10: '0x10'
04  >>> print(str(True), str(10), str(10.6), str([1,2,3]))
    True 10 10.6 [1, 2, 3]
05  >>> print(eval('2*4'), eval('[0, 1, 2]'), eval("{'x':1, 'y':2}"), eval('int(3.14)'))
    8 [0, 1, 2] {'x': 1, 'y': 2} 3
```

3. 其他相关内置函数

此外，这里还介绍几个与数据类型相关的内置函数。如表 1-6 所示：

表 1-6 数据类型相关的内置函数

函数名	功能描述	输入示例	对应输出
help([object])	显示有关对象的帮助信息	help(str)	略
dir([object])	返回当前范围内的变量或对象的详细列表	dir(str)	略
id(object)	返回对象的唯一标识符	id(10)	140716475492416
type(object)	返回该对象的数据类型	type(10)	<class 'int'>
isinstance(object, classinfo)	如果对象类型与参数类型（classinfo）相同返回 True，否则返回 False。	isinstance(10,int)	True

1.4. 常用运算符和变量

通过上例，不难看出 python 能轻松完成计算器的一些常用功能。下面介绍 python 中常用的运算符及其优先级。

1.4.1. 简单运算符

1. 算术运算符

Python 中的算术运算符，支持整型，浮点型间的运算。除常用的四则运算符： + （加），- （减）， * （乘）， / （除）外，还包括如下 3 种。

表 1-4 算术运算符举例

运算符	描述	输入示例	返回值
%	取模，返回除法的余数，常用于具有周期规律的场景	365 % 7	1
//	取整除，返回商的整数部分，常用于具有周期规律的场景	365 // 7	52
**	返回 x 的 y 次幂	3.5 ** 2	12.25

12

		2 ** 0.5	1.4142135623730951

2. 比较运算符

同样，python 还支持常用的比较运算符：＞（大于），＜（小于），>=（大于等于），<=（小于等于），以及 ==（等于），！=（不等于）。

表 1-5 比较运算符举例

运算符	描述	输入示例	返回值
==	比较运算符左右是否相等，注意用 ==，而非数学中的 =（= 在编程语言中有特殊作用）	10 == 10	True
！=	比较运算符左右是否不相等	10！= 10	False

需要强调的是，浮点数运算存在“不确定尾数”[1]。== 比较需慎用。

【示例-4】“不确定尾数”的影响

```
>>> 2.01 + 2.01 == 4.02
True
>>> 2.01 + 3.02 == 5.03
False
```

3. 逻辑运算符

Python 同样支持逻辑运算符。

表 1-6 逻辑运算符举例

运算符	描述	输入示例	返回值
and	称为布尔与运算符，该运算符左右 2 端如果全为 True 返回 True，否则返回 False。	True and True	True
		True and False	False
or	称为布尔或运算符，该运算符左右 2 端如果全为 False 返回 False，否则返回 True。	True or False	True
not	称为非运算符，该运算符只有右端值，返回相反的 bool 值	not True	False

1.4.2. 变量和赋值运算符

程序中常用变量来存放数据，如某人名，年龄，身高等。Python 中的变量可以想象成一个名字，用来获取你之前存储的值。

[1] 在计算机内部，浮点数的保存其实是一个保留位数的无限小数。如浮点数 0.1 在保存时只是无限接近 0.1。

13

1. 赋值运算符

那么，数据是如何与变量发生联系的呢。这就需要用到赋值运算符 = ，该赋值过程如图 1-10 所示。

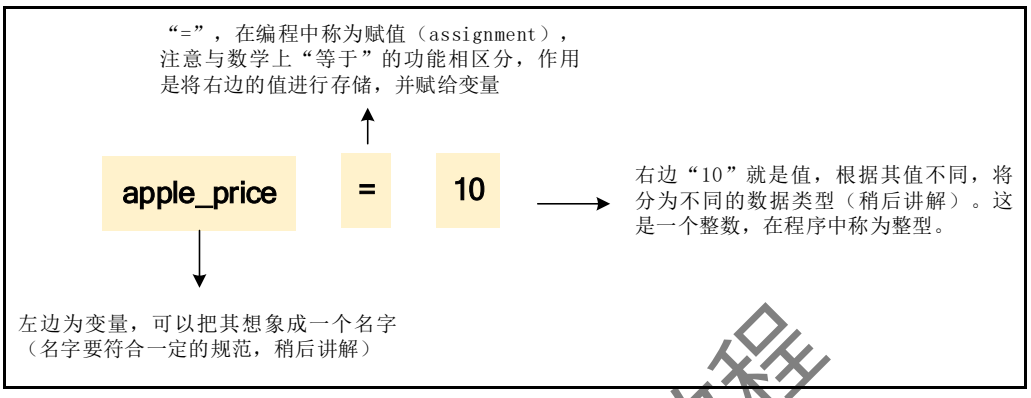


图 1-10 变量赋值的描述

其基本的语法结构如下，即将右边表达式所求出的结果，赋值给左边的变量：

```
变量名称 = 表达式
```

【示例-5】 在 Code 中，根据长方形的长和宽，求取长方形的面积和周长。

```
01   length = 4 ; width = 3
02   size = length * width
03   print(f"长方形的面积为{size}。")
```

执行结果如下：

```
03   长方形的面积为12。
```

语法具体解释如下

- ✓ 语句 01 在同一行使用了对多个变量进行了赋值，之间用“;”分隔开。
- ✓ 语句 02 中先完成右边表达式中乘法的运算，再将结果赋值给左边的变量。
- ✓ 语句 03 在字符串前加 f 修饰符，此时可在字符串中使用“{}”占位符，“{}”中可放置变量或表达式，此时 print() 函数直接输出该变量或表达式对应的数据内容。

2. 复合赋值运算符

除了常用的普通赋值运算符，python 中还提供了复合赋值运算符。具体包括 +=（加法赋值运算符）， -= （减法赋值运算符）， *= （乘法赋值运算符）， /= （除法赋值运算

14

符），%=（取模赋值运算符），//=（取整除赋值运算符），**=（幂赋值运算符）。

现以 += 说明该赋值运算符的语法结构：

b += a 等效于 b = b + a，即将表达式 b + a 的结果赋值给 b。

3. 链式赋值和多重赋值

如果多个变量具有相同的值，可用链式赋值。语法结构如下：

$a_1 = a_2 = \cdots = a_n =$ 表达式

如果多个变量具有不同的值，可采用多重赋值。

$a_1, a_2, \cdots, a_n =$ 表达式 1, 表达式 2, … 表达式 n

多重赋值首先计算赋值号右边的表达式，然后按顺序赋值给左边的变量。

【示例-6】 链式赋值和多重赋值演示。

```
01  a = b = c = 10      #链式赋值
02  print(f"a 的值为{a},b 的值为{b},c 的值为{c}")
03  a, b = 10, 20     #重新进行多重赋值
04  print(f"a 的值为{a}, b 的值为{b}")
05  a, b = b, a     #多重赋值实现变量的互换
06  print(f"a 的值为{a}, b 的值为{b}")
```

执行结果如下：

```
02  a 的值为 10,b 的值为 10,c 的值为 10
04  a 的值为 10, b 的值为 20
06  a 的值为 20, b 的值为 10
```

4. 变量的命名规范

对于 python 语言，在为变量命名时，需要遵守一定的规范。

- ✓ 名称可包含字母、中文、数字或下划线。但不能以数字开头。这里不建议用中文做变量名，会降低程序的移植性。
- ✓ 变量区分大小写，比如 apple 和 Apple 是 2 个不同的变量。
- ✓ 变量最好有意义，如有多个单词表示，建议用用下划线分割的小写单词来创建。
- ✓ Python 中有一些特殊意义的关键字称为保留字（如布尔类型中的 True 和 False），不能

15

用作变量；Python 中的内置函数不建议当作变量的名称（如内置输出函数 print()）[1]。

1.4.3. 其他运算符

1. 成员运算符

Python 还支持成员运算符，用于查看如字符串或列表中是否包含特定的元素。

表 1-7 成员运算符举例

运算符	描述	输入示例	返回值
in	列表、字符串中是否含某元素。有返回 True，无返回 False	'h' in 'hello'	**True**
not in	列表、字符串中是否含某元素。有返回 False，无返回 True	1 not in [1,2,3]	**False**

2. 身份运算符

常用于比较 2 个变量是否对应同一个存储单元。Python 中内置的 id()函数可用于获取存储单元的地址。

表 1-8 成员运算符举例

运算符	描述
is	is 用来判断两个变量是不是引用自同一内存单元。x is y，类似于 id(x) == id(y)
is not	is not 用来判断两个变量是不是引用自同一内存单元。x is not y，类似于 id(x) != id(y)

3. 位运算符

位运算符是把数值转换成二进制再进行计算。

表 1-9 成员运算符举例

运算符	描述	输入示例	返回值
&	按位与，2 个数值对应二进制位为 1，该位为 1 否则为 0。	35&13	1
|	按位或，2 个数值对应二进制位为 0，该位为 0 否则为 1。	35|13	47
^	按位异或，2 个数值对应二进制位相同，该位为 0 否则为 1。	35^13	46
~	按位取反	~35	-36
<<	左移运算符：运算数左移若干位，高位丢弃，低位补 0。	35<<2	140
>>	右移运算符：运算数右移若干位。	35>>2	8

[1] 关于 python 的保留字可详见附录 1；python 的内置函数详见附录 2.

16

4. 运算符使用时的隐式转换

运算符在使用时可能会改变数据类型，即常说的隐式转换。即数据类型间存在着相互扩展的关系，具体如下：

```
布尔类型 -> 整型 -> 浮点型
```

这是因为布尔类型可看成整数的 0 和 1，而整型可看成浮点型无小数点的情况。

5. 运算符的优先级

与数学运算一样，运算符同样存在优先级。如表 1-10 所示。如要改变运算顺序，可用半角字符小括号（()）。

表 1-10 运算符的优先级

运算符	描述
**	指数（最高优先级）。
~	按位取反。
*、/、%、//	乘、除、取模和取整除。
+、-	加法和减法
<<、>>	左移和右移运算符
&	按位与运算符
^、|	按位或和按位异或运算符
<=、<、>、>=	比较运算符
==、!=	比较运算符
=、%=、/=、//=、-=、+=、*=、**=	赋值运算符
is 、is not	身份运算符
in、not in	成员运算符
not、or、and	逻辑运算符

【示例-7】隐式转换与运算优先级

```
01  >>> True + 1 + 3.14
    5.140000000000001
02  >>> 2 + 3 * 4 != 12 or 2 + 3 == 5
    True
```

1.5. 常用内置函数

前面已对内置函数进行了部分讲解，这里再介绍一些其他常用的内置函数。

17

1.5.1. 数学运算内置函数

除了上述的一些基本运算符外，Python 3 还提供了一些内置的数学运算函数。

表 1-10 数学运算内置函数

函数名	描述	输入示例	返回值
abs(x)	x 的绝对值	abs(-35)	35
divmod(x, y)	(x//y,x%y)，输出为二元组的形式	divmod(35,8)	(4, 3)
pow(x, y[, z])	(x**y)%z。当 z 缺省时为 pow(x,y)，它与 x**y 相同	pow(2,3)	8
round(x[, n])	对 x 四舍五入，保留 n 位小数。round(x)返回四舍五入的整数	round(3.14,1)	3.1

1.5.2. 标准输出/输出函数

print()是用来把对象输出到控制台的标准输出函数。其基本语法为：

print(value1[, value2, …, sep=' ', end='\n']) ✓ value: print()函数可输出多个 value，多个 value 间按照 sep 分隔符进行分隔； ✓ sep: 为插入各输出值之间的分隔符，默认为空格。 ✓ end：默认为最后一个输出值后面的结束符，默认为换行符。 ✧ 无返回值

input()是标准输入函数，可以让用户从键盘输入数据。其基本语法为：

input([prompt]) ✓ prompt: 提示信息 ✧ 返回用户输入的字符串

【示例-8】输入和输出函数演示

```
01  name = input("请输入姓名：")
02  age = int(input("请输入年龄："))    #对于单个数据，可根据希望的数据类型进行相应转换
03  x,y,z = eval(input("输入语数外成绩："))   #输入多个数据，半角字符(,)分隔，用eval()完成转换
04  print('语文', x, sep = ':', end = '\t')
05  print('数学', y, sep = ':', end = '\t')
06  print('英语', z, sep = ':')
07  print(f"{name}的年龄为{age},总分为{x+y+z}")
```

执行结果如下，注意输入成绩时，使用英文半角字符（,）：

```
01  请输入姓名：张三
02  请输入年龄：15
```


18

```
03      请输入语数外3门课的成绩：78,89,100
04-     语文:78 数学:89 英语:100
06      张三的年龄为15,总分为267
07
```

1.5.3. range()函数

range()函数用于返回可序列，具体使用将在后面讲解。其基本语法如下：

range([start], stop[, step])
✓ start：开始值。省略时默认 0 ，如 range（5）等价于 range（0， 5）。
✓ stop：迭代结束值。注意不包括 stop。如 range（0，3）对应 0，1，2，而没有 3。
✓ step：步长（可以为负数），默认为 1。如 range（0， 5） 等价于 range(0, 5, 1)

1.6. 常用标准库函数

除内置函数外，python 还提供了许多默认支持的函数库叫做标准库（Standard Library）或模块。常用的模块有：数学模块（math）、随机数模块（random）、时间模块（time）和日历模块（calendar）。

标准库中的函数不能直接使用，需用关键字 import 引用该库，方式有 2 种：

表 1-11 引用标准库

方法	描述
import 库名	对库中的函数采用 库名.函数名() 形式使用
from math import <函数名>	对库中的函数直接采用 函数名() 形式使用

1.6.1. 数据模块

math 库是 Python 提供的内置数学类函数模块。常用的包括：

表 1-12 math 库常用函数

属性/函数	描述
pi / e	圆周率 3.141 592 653 589 793 / 自然对数 2.718 281 828 459 045
ceil(x) / floor(x)	向上取整 / 向下取整
modf(x)	返回 x 的整数部分和小数部分
factorial(x)	x!（注意：x 必须是整数）
exp(x)	e^x
sqrt (x)	x 的平方根
log2(x) / log10(x) / log(x[,y])	2 为底对数 / 10 为底对数 / y 为底对数;只输入 x 是返回自然对数

19

log1p(x)	ln(1 + x)
degrees(x) / radians(x)	角度 x 的弧度值转角度值 / 角度 x 的角度值转弧度值
hypot(x,y)	坐标(x,y)到原点（0，0）的距离
sin(x) / cos(x) / tan(x)	sin(x) / cos(x) / tan(x)
asin(x) / acos(x) / atan(x)	asin(x) / acos(x) / atan(x)

1.6.2. 随机数模块

random 库是 Python 提供的内置随机数类函数模块。常用的包括：

表 1-13 random 库常用函数

属性/函数	描述
random() / uniform(x,y)	随机生成一个[0,1)范围内的浮点数 随机生成一个[x,y]范围内的浮点数
randrange([start], stop[, step])	在 range([start], stop[, step])序列中随机挑选一个数，含 start 不含 stop.
choice(seq)	从序列（如字符串、列表、range 函数等）中随机挑选一个元素
sample(seq,k)	从序列（如字符串、列表、range 函数等）中随机挑选 k 个元素
shuffle(lst)	将序列的所有元素随机排序，注意该函数无返回值，直接修改 lst 参数
seed([x])	设置随机数生成器种子，在调用随机函数前调用此函数

【示例-9】random 模块演示

```
01  from random import *
02  print(random(), uniform(1, 10), choice(range(10)), sample(range(10), 3))
03  x = [1, 2, 3, 4, 5]
04  print(shuffle(x), x)
```

执行结果如下：

```
02  0.9374522432815992  2.0368211879673015  0  [0, 6, 1]
04  None  [2, 3, 4, 1, 5]
```

1.6.3. 时间模块和日历模块

time 库和 calendar 库是 Python 提供的时间模块和日历模块。常用的包括：

表 1-14 time 库和 calendar 库常用函数

属性/函数	描述
time.time()	随机生成一个[0,1)范围内的浮点数 随机生成一个[x,y]范围内的浮点数
time.asctime([tupletime])	接收时间元组并返回一个日期时间字符串。
time.strftime(fmt[, tupletime])	接收时间元组，按指定格式 fmt 返回当前的日期和时间
calendar.month(year, month)	返回指定年月的日历

20

calendar.weekday(year, month, day)	返回日期的星期代码。默认为 0（星期 1）到 6（星期日）

【示例-10】time 库和 calendar 库演示

```
import time, calendar
print(time.time())
print(time.asctime())
print(time.strftime('%Y-%m-%d %H:%M:%S'))
print(calendar.weekday(2018, 7, 31))    #1 代表星期二
print(calendar.month(2018, 7))
```

执行结果如下：

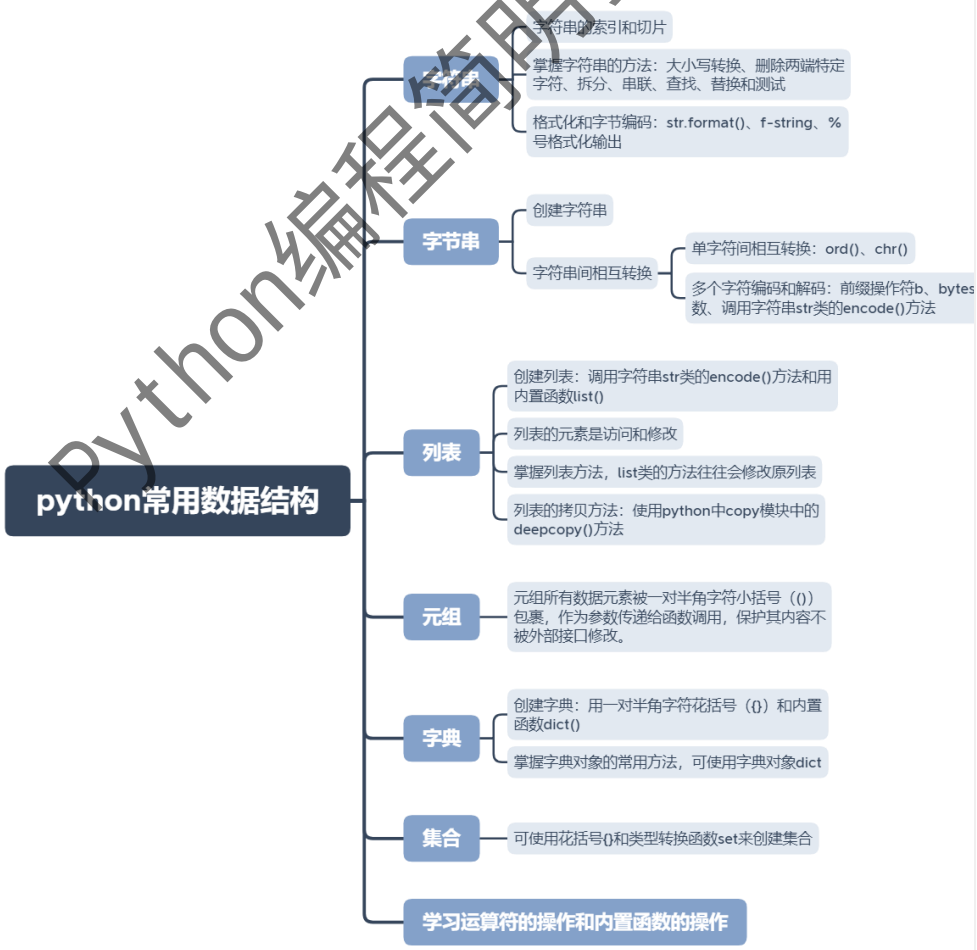
```
1629276939.961824
Wed Aug 18 16:55:39 2021
2021-08-18 16:55:39
1
    July 2018
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```


21

2 Python 常用数据结构

✍ 知识目标:	💻 技能目标
1， 掌握字符串的索引切片，字符串用对象 str 类的方法进行操作 2， 掌握字节串，列表，元组，字典和集合的创建和常用类的方法 1. 3， 结合所学，了解运算符和内置函数的操作	1， 具有初步的编程能力 2，对字符串，字节串，列表，元组，字典，集合具有更多的了解，并会使用不同的方法进行修改 3， 能根据情况使用不同的运算符和函数

⌘ 学习地图:



数据是计算机处理的基础。数据结构是通过某种方式组织在一起的数据元素的集合，这

些元素既可是前文所讲的基本数据类型，也可本身就是数据结构。

Python 内置的常用数据结构：列表、元组、字典、集合和字符串。此外，还有 bytes 字节串、bytearray 字节数组等。这些数据结构都是可迭代对象。

从数据结构中元素是否可写，这些数据结构可分为：

- ✓ 列表、字典、集合和 bytearray 字节数组是可变数据结构，可直接操作其中的元素。
- ✓ 元组、字符串和 bytes 是不可变数据，不可直接操作其中的元素。

从数据结构中数据元素排列规律，这些数据结构可分为：

- ✓ 列表、元组和字符串都是有序数据结构，也称序列。通过下标进行索引。
- ✓ 字典、集合是无序数据。其中字典可以通过键名进行索引映射。

2.1 字符串

字符串是由一组字符组成的内置不可写序列，本节将在上节基础上介绍字符串的索引与切片、字符串格式化、字符串操作符、字符串处理函数和处理方法进行介绍。

2.1.1字符串的索引和切片

字符串是由一组字符组成的内置不可写序列，因此可以通过**索引（Index）**，即下标对字符串进行访问。Python 提供了两种索引，如表 2-1 所示。

表 2-1 字符串索引表示

字符串	H	e		s	a	i	d	\"	l	'	m		O	K	!	\"
从左往右索引	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
从右往左索引	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

其中：由左往右，**索引以 0 开始**，依次递增；由右往左，**索引以-1 开始**，依次递减。需强调，转义字符（\）不计入字符串内容，而空格计入字符。

通过操作符（[]）可获取字符串中某个字符；通过切片可获取部分字符串，语法如下。

```
字符串名称[起始索引值:结束索引值:间隔值]
```

其中，起始索引默认为 0，结束索引默认为最右端，间隔值默认为 1。**需强调的是切片不含**

结束值。表 2-2 为字符串 `var8 = "He said\"I'm OK!\""`在不同切片时的返回值。

表 2-2 字符串切片及其输出

语法	var8[0:5:1]	var8[0:5]	var8[:5]	var8[::4]	var8[::-4]	var8[::-1]	var8[:]
作用	取字符串索引值 0~4			从左/右以 4 为间隔选取		反向输出字符串	复制字符串
返回	He sa			HalO	" "s	"!KO m'I"dias eH	He said"I'm OK!"

2.1.2 字符串方法

对于创建好的字符串，可用字符串对象 str 类的方法来操作，本节列举一些主要的方法。详细的信息可用内置函数 help()查询，如查询 str.title()方法，对应的语句为 help(str.title)。**再次强调字符串是不可变数据类型，这些方法都不会改变原字符串，如有修改操作，实际都是返回一个新的字符串。**

1. 字符串大小写转换

str 类提供了一系列字符串大小写转换的方法，如表 2-3。

表 2-3 字符串大小写转换

方法	描述	输入示例	返回新字符串
str.title()	将字符串中每个单词的首字母大写	"let's go".title()	Let'S Go
str.capitalize()	将字符串首字符大写	"let's go".capitalize()	Let's go
str.upper()	将字符串全部大写	"let's go".upper()	LET'S GO
str.lower()	将字符串全部小写	"LET'S GO".lower()	let's go

2. 删除字符串两端特定字符

用户在输入数据时，常会无意输入多余的空格（如制表符或换行符）或特殊字符，此时常需去除这些字符。str 类中提供了如下 3 个方式去除字符串首尾空格或特殊字符。

表 2-4 去除字符串两端特定字符

方法	描述	输入示例	返回值
str.strip([chars])	去除两侧字符。chars 为要去除的字符；默认为去除空白字符，包括（\n、\t）等	" \naa bb\t".strip()	aa bb
		"***abc**".strip('*')	abc
str.lstrip([chars])	去除左侧特定字符……	" abc ".lstrip()	abc
str.rstrip([chars])	去除右侧特定字符……	" abc ".rstrip()	abc

【示例-1】央视的域名为 www.cctv.com，去除域名的前缀和后缀。

```
01  print("www.cctv.com".strip("wcom."))
02  print("www.cctv.com".lstrip("w.").rstrip("com."))
```

执行结果如下：

```
01  tv
02  cctv
```

此处，不能直接按语句 01 通过 strip("wcom.")处理，因为如去除前缀"www."，"cctv"中的"cc"就在最外则，也会被去除。所以可按语句 02，先去除左侧的"www."，再去除右侧的".com"。

3. 字符串拆分

有时需要把一段字符串拆分为多个。str 类中提供了如下方式进行字符串拆分。这里需注意的是，拆分后的返回值是一个列表。

表 2-5 字符串拆分

方法	描述
str.split([sep, [maxsplit]])	从左到右分割字符串。sep 为分割字符，默认为所有空白字符；maxsplit 参数控制分割次数，默认-1，表示无限制。**返回类型为列表，元素为去掉 sep 参数的字符子串。**
str.rsplit(…,…)	从右到左分割字符串，当 maxsplit 有限定参数时，与 str.split()输出可能不一致。

【示例-2】观察下列字符串的拆分，思考拆分的结果，并与输出相比较。

```
01  print("道路自信  理论自信  制度自信  文化自信".split())
02  print("www.iflytek.com".split("."))
03  print("www.iflytek.com".split(".",  1))
04  print("www.iflytek.com".rsplit(".",  1))
```

执行结果如下[1]，

```
01  ['道路自信', '理论自信', '制度自信', '文化自信']
02  ['www', 'iflytek', 'com']
03  ['www', 'iflytek.com']
04  ['www.iflytek', 'com']
```

如要保留分割字符，可使用 str.partition()方法和 str.rpartition()方法，这里不再讨论。

[1] 这里返回的为列表数据结构，也是序列的一种，关于列表的讲解请见下节，在这里只要有所概念即可。

4. 串联字符串

前面讲解了字符串的分割，那么分割后的字符串同样可以合并。

表 2-6 字符串合并

方法	描述	输入示例	返回值
str.join(seq)	使用字符串 str 将序列 seq 中元素合并成**新字符串**。	"+".join(["1","2","3"]	1+2+3

【示例-3】学生名单中有多个连续空格，请整理并保留一个。

```
01  print(" ".join("小明    小红    小张   小李  小赵".split()))
```

执行结果如下

```
01  小明 小红 小张 小李 小赵
```

5. 字符串的查找和替换

下面介绍字符串查找和替换的基本方法。

表 2-7 字符串查找和替换

方法	描述
str.find(substr [,start, [end]])	从左到右查找。substr 为要查找字符，start 与 end 为原字符串查找范围，默认为整个字符串。如查找不到，返回-1；否则，返回第 1 次查找到字符出现的位置。
str.rfind(…,…,…)	从右到左查找，其余与 str.find()一致……。
str.count(substr [,start, [end]])	统计子字符串 substr 在原字符串中出现的次数。如没有匹配的子串返回 0，否则返回子串出现的次数。
str.replace(oldstr, newstr [,count])	oldstr 为要替换的字符，newstr 为用来替换的字符，count 参数指定被替换的次数，默认为-1，表示全部替换。

【示例-4】观察下列字符串的查找和替换，并分析结果。

```
01  print("I come from China!".find("o"))
02  print("I come from China!".rfind("o"))
03  print("I come from China!".count("o"))
04  print("I come from China!".replace("o","O"))
```

执行结果如下[1]，

```
3
9
2
I cOme frOm China!
```

此外，还可用 str.index()方法和 str.rindex()方法，这里不再描述。

6. 字符串测试

字符串测试用于检测字符串是否为特定格式，其返回值为 bool 值。

表 2-8 字符串测试

方法	描述
str.startswith(substr [,start, [end]])	原字符串 str 以指定的 substr 开头，返回 True
str.endswith(substr [,start, [end]])	原字符串 str 以指定的 substr 结尾，返回 True
str.isalnum()	字符串只包含字母和数字时返回 True。
str.isalpha()	字符串只包含字母时返回 True。
str.isdecimal()	字符串只包含十进制数字字符时返回 True。
str.isdigit()	字符串只包含数字字符时返回 True。
str.istitle()	字符串符合大小写标题要求，返回 True。

【示例-5】观察下列字符串，去除开头空白字符后，观察是否以域名‘www’开头。

```
print('    www.cctv.com'.strip().lower().startswith('www'))
```

执行结果如下[2]，

```
True
```

此外，还可用 str.index()方法和 str.rindex()方法，这里不再描述。

7. 获取指定宽度的字符串

可通过字符串对齐，或填充的方式，获取指定宽度的字符串。

[1] 这里返回的为列表数据结构，也是序列的一种，关于列表的讲解请见下节，在这里只要有所概念即可。
[2] 这里返回的为列表数据结构，也是序列的一种，关于列表的讲解请见下节，在这里只要有所概念即可。

表 2-9 获取指定宽度的字符串

方法	描述	输入示例	返回值
str.center(n[,fillchar])	返回宽度为 n，str 居中的新字符串；fillchar 为填充字符，默认为空格填充。	'py'.center(6,"–")	--py--
str.ljust(…,…)	……，str 居于左边，……。	'py'.ljust(6,"–")	py----
str.rjust(…,…)	……，str 居于右边，……。	'py'.rjust(6,"–")	----py

2.1.3字符串格式化和字节编码

字符串格式化用于解决字符串和变量同时输出的格式安排，。其实现方法有如下 3 种。

1. 字符串方法 str.format()

Python 使用 str.format()方法，实现字符串格式化处理。其基本格式如下：

模板 str.format(arg1, arg2, arg3, ……)
✓ 模板 str：由一系列的槽“{}”组成，槽中可填写整数表示顺序，默认为从 0 开始依次递增的。
✓ format 后面的参数按照序号关系替换模板字符串中的槽。

【示例-6】观察下列字符串格式化的结果。

```
01  str1 = "{}：{}岁，{}m；{}：{}岁，{}m。".format('张三', 35, 1.82, '李四', 35, 1.82)
02  str2 = "{0}：{2}岁，{3}m；{1}：{2}岁，{3}m。".format('张三', '李四', 35, 1.82)
03  print(str1), print(str2)
```

执行结果如下，

```
03  张三:35 岁，1.82m；李四：35 岁，1.82m。
    张三:35 岁，1.82m；李四：35 岁，1.82m。
```

除上述使用方法外，str.format()方法中还可包含多种格式限定符，附带在“{}”操作符的“:”符号的后面。格式控制标记包括如下字段

- ✓ 宽度、对齐、填充是 3 个相关字段。宽度指当前槽设定的输出字符宽度；对齐指参数在宽度内输出时的对齐方式，包括左对齐“<”，右对齐“>”和“^”居中对齐；填充是指宽度内除了参数外的字符采用什么方式表示，默认为空格。
- ✓ 数字的千位分割符“,”，用于显示数字类型的千位。
- ✓ 精度，由小数点“.”开头，对于浮点数，精度表示小数部分输出的有效位数。对于字符串表示输出的字符串长度。

✓ 类型，表示输出整数和浮点数类型的格式规则。整数类型包括二进制整数“b”，十进制整数“d”，八进制整数“o”，十六进制整数“x”或“X”，或整数对应 Unicode 字符“c”；浮点数类型包括标准浮点形式“f”，百分比形式“%”,指数形式“e”或“E”。

【示例-7】观察下列字符串格式化的结果。

```
str3 = "{:20}".format('好好学习')
str4 = "{:-<20}".format('好好学习')
str5 = "{:*>20,}".format(123456789)
str6 = "{:-^20,.3f}".format(12345.6789)
str7 = "{0:b},{0:d},{0:o},{0:x},{0:c}".format(90)
str8 = "{0:.2e},{0:.2f},{0:.2%}".format(12345.6789)
print(str3),print(str4),print(str5),print(str6),print(str7),print(str8)
```

执行结果如下，

```
好好学习
好好学习----------
*********123,456,789
-----12,345.679-----
1011010,90,132,5a,Z
1.23e+04,12345.68,1234567.89%
```

2. 使用 f-string 方法

f-string 是方法在第 1 章中有所涉及，该方法同样可包含多种格式限定符。

【示例-8】观察下列字符串格式化的结果。

```
import math
print(f"半径为{1}的圆，周长为{2*math.pi:.2f}，面积为{math.pi:.2f}。")
```

执行结果如下：

```
半径为1的圆，周长为6.28，面积为3.14
```

3. %号格式化输出

在 Python 中同样支持通过“%”完成格式化输出，%在字符串中表示格式化字符，它后面必须附加一个格式化符号。

表 2-10 python 格式化符号列表

方法	描述	指令	描述
%c	格式化为字符（ASCII）码，适用于数字和字符。	-	左对齐显示，默认右对齐。
%d	格式化有符号十进制整数，适用于数字。	+	正数前显示加号（+）。
%u	格式化无符号十进制整数，适用于数字。	0	显示的数字前填充 0，而不是默认的空格。
%o	格式化无符号八进制整数，适用于数字。		
%x	格式化无符号十六进制整数，适用于数字。	#	在八进制前显示零（‘0’），在十六进制前显示（‘0x’）。
%r	使用 repr()函数格式化显示。		
%s	使用 str()函数格式化显示。	m.n	m 表示最小显示的总宽度，如果超出，则原样输出；n 表示可保留的小数点后的位数或者字符串的个数。
%f	格式化浮点数，可指定小数点后位数。		
%e	格式化为科学计数法，仅适用于数字。		
%%	输出%。		

【示例-9】观察下列字符串格式化的结果。

```
01  s='p=%10.3f,p=%-10.3f,p=%+10.3f,p=%010.3f'%(math.pi,math.pi,math.pi,math.pi)
02  print(s)
```

执行结果如下：

```
02  pi=     3.142, pi=3.142     ,pi=    +3.142,pi=000003.142
```

2.2 字节串

字节串（bytes）是 Python3 新增的一种数据类型，也称字节序列，是不可变序列。存储以字节为单位的数据。其与字符串比较如下：

- ✓ 字符串由多个字符构成，默认使用 Unicode 字符，不能直接存储和传递。
- ✓ 字节串由多个字节构成，以字节为单位，可方便在计算机中存储和传递

字符串与字节串间可相互转换，其映射关系称为编码和解码。表 2-11 是最常见的几种。

表 2-11 常见编码方式

方法	描述
ASCII	美国信息交换码，使用 7 个或 8 个二进制进行编码，可给 26 个大小写字母、10 个数字、标点符号、控制字符等分配数值。
Unicode	又称统一码、万国码，Unicode 为每种语言的每个字符设定了唯一的二进制编码，规定所有的字符和符号最少由 2 个字节来表示。

UTF-8	是对 Unicode 编码的优化，他将所有的字符和符号进行分类：ASCII 码中的内容用 ASCII 码直接保存、欧洲字符用 2 个字节保存，东亚字符用 3 个字节保存...
GB2312	是一个简体中文字符集，由 6763 个常用汉字和 682 个全角的非汉字字符组成。GB2312 使用 2 个字节表示 1 个汉字。
GBK	GBK 是对 GB2312 的扩展，也采用双字节。收录汉字 21003 个，符号 883 个。

2.2.1字节串的创建

字节串有如下 2 种常用的创建方式。

✓ 对于可用 ASCII 字符集的字符串，可通过添加前缀操作符 b，完成创建。

✓ 对于一般的字符串，可通过 bytes()函数返回一个新的 bytes 对象。方法如下：

```
bytes([source[, encoding[, errors]]])
```

✓ 如 source 为整数，则返回一个长度为 source 的初始化字节序列；

✓ 如 source 为字符串，则按照指定的 encoding 将字符串转换为字节序列；

✓ 如没有输入任何参数，默认就是初始化空的字节串。

【示例-10】观察下列字符串转字节串方法。

```
print(b'China', b'\x41\x42\x43')
print(bytes(range(65,91)), bytes('中国:China','utf-8'))
```

执行结果如下：

```
b'China'        b'ABC'
b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'        b'\xe4\xb8\xad\xe5\x9b\xbd:China'
```

2.2.2字节串与字符串相互转换

1. 单字符与 unicode 相互转换

Python 中内置了 ord()函数以获取字符的 Unicode 码，内置了 chr()函数以把 Unicode 码转换为对应的字符。

【示例-11】观察下列字符串使用 ord()函数和 chr()函数的结果。

```
print(ord('2'), ord('a'), ord('国'))
```

```
02    print(chr(50), chr(97), chr(22269))
```

执行结果如下：

```
01    50 97 22269
02    2 a 国
```

2. 多个字符编码和解码

对多个字符的字符串编码，除了可用前缀操作符 b 和 bytes()函数，还可用如下方法。

✓ 调用字符串 str 类的 encode()方法，将字符串编码成字节串。

```
str.encode(encoding='UTF-8',errors='strict')
```

✓ 参数 encoding 表示要进行编码的类型，默认为 utf-8。

✓ 参数 errors 设置不同的错误处理方案，默认 strict，表示遇到非法字符就抛出异常；其他如 ignore 表示忽略，replace 表示用? 代替。

➢ 返回 2 进制编码的字符串

✓ 调用字节串 bytes 类的 decode()方法，将字符串编码成字节串。

```
bytes.decode(encoding='UTF-8',errors='strict')
```

【示例-12】使用 encode()方法和 decode()的结果。

```
01    byt1 = '中国:china'.encode(), byt2 = '中国:china'.encode('gb2312')
02    print(byt1,byt2)
03    str1 = byt1.decode(), str2 = byt2.decode('gb2312')
04    print(str1,str2)
```

执行结果如下：

```
02    b'\xe4\xb8\xad\xe5\x9b\xbd:china'    b'\xd6\xd0\xb9\xfa:china'
04    中国:china    中国:china
```

2.3 列表

列表数据类型（list）是另一种 Python 内置序列，与字符串不同，列表可写。

2.3.1列表的创建

除上一章调用字符串 str 类的 encode()方法创建列表，还可用内置函数 list()完成列表的创建。且列表中元素数据类型可不同[1]，几乎所有的简单数据类型和数据结构都可作列表数据元素。

【示例-13】演示用中括号定义列表。

```
01  list1 = list(range(5))                          #list 方法将 range 对象进行强制转换
02  list2 = list('China')                           #list 方法将字符串对象进行强制转换
03  list3 = list({'x':1,'y':2,'z':3})     #list 方法将字典对象进行强制转换，默认转换键
04  print(f"list1:{list1}\nlist2:{list2}\nlist3:{list3}")
05  scores = [60 , 70.5 , 80 , 90 ]
06  names = ["张三" , "李四" , "王五" , "赵六"]
07  students = [names , scores]
08  print(f'students:{students}')
```

执行结果如下：

```
04  list1:[0, 1, 2, 3, 4]
    list2:['C', 'h', 'i', 'n', 'a']
    list3:['x', 'y', 'z']
08  students:[['张三', '李四', '王五', '赵六'], [60, 70.5, 80, 90]]
```

2.3.2列表的元素的访问和修改

与字符串相同，列表同样支持索引和切片。且列表本身可变，所以还可完成元素的修改。

1. 简单列表元素的访问

表 2-12 为列表 `lis = list('abcdefgh')`在不同切片时的返回值。

表 2-12 字符串切片及其输出

语法	lis[0:5:1]	lis[0:5]	lis[:5]	lis[::4]	lis[::-4]	lis[::-1]
作用	取列表索引值 0~4			从左/右以 4 为间隔选取		反向输出字符串
返回	['a', 'b', 'c', 'd', 'e']			['a', 'e']	['g', 'c']	['g', 'f', 'e', 'd', 'c', 'b', 'a']

[1] 从 Python 代码的可读性和程序的执行效率考虑，还是建议统一列表元素的数据类型。

【示例-13 续】 请输出嵌套列表 students 中倒数第 2 个人的姓（输出结果略）。

```
13  print(students[0][-2][0])
```

2. 列表的解包

解包操作指从可迭代对象中把元素逐个取出赋值给对应的变量。一般而言，变量的个数与可迭代对象中元素的个数保持一一对应。当变量个数少于元素个数时，常需在某个变量前加上*号，意思是用于接收剩余的所有元素。

【示例-13 续】 解包操作演示。

```
01  a,b = [1,2]
02  c,*d = [1,2,3]
03  print(f'{a}\t{b}\t{c}\t{d}')
```

对应的输出为：

```
03  1       2       1       [2, 3]
```

3. 列表元素的修改

由于列表是可写序列，因此可通过直接赋值的方法，完成列表元素的修改。

【示例-13 续】 将 students 列表中最后一个人的姓名，修改为‘钱七’。

```
01  students[0][-1] = "钱七"
02  print(f'students:{students}')
```

执行结果如下：

```
02  students:[['张三', '李四', '王五', '钱七'], [60, 70.5, 80, 90]]
```

2.3.3列表的方法

对于创建好的列表，同样可使用列表对象 list 类的方法来操作。与**字符串方法不同，列表是可变数据类型，list 类的方法往往会修改原列表。**

1. 常用方法

现将 list 类的常用方法总结如下。

表 2-13 列表对象的常用方法（设原 lis=[1,-1,10]）

方法	描述	示例	操作后 lis	返回值
list.append(obj)	将参数 obj 的值直接添加在列表末尾	lis.append(5)	[1,-1,10,5]	None
list.extend(iterable)	将可迭代对象 iterable 的所有元素添加到列表末尾	lis.extend('ab')	[1,-1,10,'a','b']	None
list.insert(index, obj)	index 表示插入索引的位置，obj 表示要插入对象	lis.insert(1,5)	[1,5,-1,10]	None
list.remove(obj)	删除第 1 个匹配参数 obj 的值，若无报错	lis.remove(10)	[1,-1]	None
list.pop([index])	删除并返回 index 指定对象，默认最后一个。若 index 超出范围，则报错。	lis.pop()	[1,-1]	10
list.clear()	清空列表	lis.clear()	[]	None
list.reverse()	将列表倒序	lis.reverse()	[10,-1,1]	None
list.sort([reverse])	对列表进行排序，默认 reverse=False 进行降序排列	lis.sort()	[-1,1,10]	None
list.index(x[, start[, end])	从列表中找出参数 obj 第一个匹配项的索引位置。start 与 end 默认为整个列表。	lis.index(-1)	[1,-1,10]	1
list.count(obj)	统计参数 obj 出现的次数	lis.count(100)	[1,-1,10]	0
list.copy()	对列表进行浅拷贝	lis.copy()	[1,-1,10]	[1,-1,10]

2. 列表的拷贝方法

列表的拷贝，包括引用拷贝，浅拷贝，深拷贝。其中深拷贝需要使用 python 中 copy 模块中的 deepcopy()方法，关于这 3 种拷贝可用如下示例说明

【示例-14】 列表的三种拷贝方法。

```
01  import copy
02  lis = [1,2,[10,100]]
03  lis1 = lis    #引用拷贝
04  lis2 = lis.copy() #浅拷贝
05  lis3 = lis[:]  #功能与浅拷贝相同
06  lis4 = copy.deepcopy(lis)  #深拷贝
07  print(id(lis), id(lis1), id(lis2), id(lis3), id(lis4))
08  print(id(lis[2]), id(lis1[2]), id(lis2[2]), id(lis3[2]), id(lis4[2]))
09  lis[0] = 10;lis[-1][-1] = 10
10  print(f"{lis}\n{lis1}\n{lis2}\n{lis3}\n{lis4}")
```

执行结果如下：

```
07  2417365455808 2417365455808 2417365456064 2417365456128 2417365770560
08  2417365456000 2417365456000 2417365456000 2417365456000 2417365704960
10  [10, 2, [10, 10]]
    [10, 2, [10, 10]]
    [1, 2, [10, 10]]
    [1, 2, [10, 10]]
    [1, 2, [10, 100]]
```

这里“=”实现的是引用拷贝，即 lis1 和 lis 指向的是同一内存地址，其中一个变化，都会引起另一个的变化；而 deepcopy 对应深拷贝，此时 lis 和 lis4 之间不在有任何关联，是相互独立的变量；而特别需要注意的就是浅拷贝，即 list.copy()方法，该方法在内存中新建了一个原列表对象的副本，但不会对原列表中的元素进行新建。

其实关于引用拷贝，浅拷贝，深拷贝的原理，只需查看列表的 id，列表中元素的 id 是否变化，就可完成这一过程的理解。

2.4 元组

元组（tuple）与列表类似，不同的是元组中的数据元素不可变，所有数据元素被一对半角字符小括号（()）包裹。由于元组不可变，其主要作用是作为参数传递给函数调用，保护其内容不被外部接口修改。

需强调，对于单个元素的元组，创建时必须注意小括号（()）中的（,）不可缺省。

【示例-15】 元组的创建。

```
01  print(type((1)),type((1,)))
```

对应的输出为：

```
01  <class 'int'> <class 'tuple'>
```

与 list 类方法相比，tuple 类的方法相对较少。

表 2-13 元组对象的常用方法（设原 tup=(1,-1,10)）

方法	描述	示例	返回值
tuple.index(x[, start[, end])	从元组中找出参数 obj 第一个匹配项的索引位置。start 与 end 默认为整个元组。	tup.index(-1)	1
tuple.count(obj)	统计参数 obj 出现的次数	tup.count(100)	0

2.5 字典

字典（dict）是 Python 的另外一种可变的数据结构。与列表不同，字典没有索引，使用键值对（key-value）的形式来存储数据。

2.5.1 字典的创建

与列表类似，除上一章用一对半角字符花括号（{}）创建字典外，还可用内置函数 dict()完成字典的创建。需注意的是，字典的键必须为不可变数据类型，否则会报错。

【示例-16】 字典的创建[1]。

```
01  d1 = {'x':1,'y':2,'z':3}
02  d2 = dict(enumerate([1,10,100]))
03  d3 = dict(zip(['a','b','c'],[1,2,3]))
04  print(f'{d1}\t{d2}\t{d3}\t')
```

对应的输出为：

```
04  {'x': 1, 'y': 2, 'z': 3}    {0: 1, 1: 10, 2: 100}    {'a': 1, 'b': 2, 'c': 3}
```

2.5.2 使用字典的键操作字典元素

字典同样支持元素的访问和修改。但字典本身是无序的数据结构，上述操作一般使用字典的键来完成。

【示例-16 续】 修改示例 16 中的变量 d1。

```
01  d1['x'] = 10
02  d1['X'] = 5
03  print(f"{d1}\t{d1['x']}")
```

对应的输出为：

```
03  {'x': 10, 'y': 2, 'z': 3, 'X': 5}    10
```

需注意的是，如果输入的键不在字典中，程序会报错。

[1] python 提供了内置函数 enumerate()函数和 zip()函数用来生成可迭代对象，下章在 for 循环进行详解。

2.5.3 字典对象的常用方法

对于创建好的字典，同样可使用字典对象 dict 类的方法来操作。字典对象的方法同样会修改原字典。

1. 常用方法

表 2-14 字典对象的常用方法（设原始 d={'a':1,'b':2}）

方法	描述	示例	操作后 d	返回
dict.clear()	清除列表中的所有元素	d.clear()	{}	None
dict.keys()	返回可迭代的键列表	d.keys()	{'a': 1, 'b': 2}	dict_keys(['a', 'b'])
dict.values()	返回可迭代的值列表	d.values()	{'a': 1, 'b': 2}	dict_values([1, 2])
dict.items()	返回可迭代键-值对列表	d.values()	{'a': 1, 'b': 2}	dict_items([('a', 1), ('b', 2)])
dict.get(k [, default])	通过键返回值；当不存在时，返回默认的值	d.get('a')	{'a': 1, 'b': 2}	1
		d.get('c',0)	{'a': 1, 'b': 2}	0
dict.pop(k[,v])	删除键 k 对应的键值对，并返回该值；若 k 不存在，返回值 v	d.pop('a',0)	{'b': 2}	1
		d.pop('c',0)	{'a':1,'b':2}	0
dict.popitem()	删除字典中最后一对键值对，并以元组的形式返回该键值对	d.popitem()	{'b': 2}	('b', 2)
dict.update(d1)	将字典 d1 添加到 dict 中，若 dict 中有键与 d1 重复，则更新重复的键	d.update({'b':10,'c':0})	{'a': 1, 'b': 10, 'c': 0}	None

2. 字典的深浅拷贝

与列表相同，字典同样存在深浅拷贝

【示例-17】 字典的三种拷贝方法。

```
01  import copy
02  d = {'a': 1, 'b': [10,100]}
03  d1 = d        #引用拷贝
04  d2 = d.copy()        #浅拷贝
05  d3 = copy.deepcopy(d)       #深拷贝
06  print(id(d), id(d1), id(d2), id(d3))
07  print(id(d['b']), id(d1['b']), id(d2['b']), id(d3['b']))
08  d['a'] = 0
09  d['b'].pop()
```

```
10  print(f"{d}\n{d1}\n{d2}\n{d3}")
```

执行结果如下：

```
06  2214838146624 2214838146624 2214838146048 2214838146304
07  2214843307136 2214843307136 2214843307136 2214843047744
10  {'a': 0, 'b': [10]}
    {'a': 0, 'b': [10]}
    {'a': 1, 'b': [10]}
    {'a': 1, 'b': [10, 100]}
```

2.6 集合

Python 中的集合（set）其定义与数学中的集合相类似，集合是无序的不重复元素的数据集，其最常用的功能是去除重复值。可使用花括号{}和类型转换函数 set 来创建集合，需强调的是，一个空集合必须用函数 set()进行创建，而不能用{}，后者实际表示创建一个字典。

【示例-18】 求字符串中的字符。

```
01  print(set('I come from China!'))
```

执行结果如下：

```
01  {'h', 'n', 'C', 'a', 'o', 'c', ' ', 'f', 'I', 'm', 'r', 'i', '!', 'e'}
```

2.7 运算符的操作

对于本章介绍的数据结构，运算符同样可以对其实现操作。如第一章的成员运算符（in，not in）[1]，身份运算符（is，is not）对于所有的数据结构都是通用的。

2.7.1 序列常用运算符

对于序列（包括元组，字典，字符串）常用的运算符，如表 2-15 所示。

表 2-15 序列常用运算符

运算符	描述	输入示例	返回值
+	连接 2 个序列	"Hello"+" "+"world!"	**Hello world!**

[1] 对于字典成员运算符用于检查字典中是否含有键，而非对值进行检查。

*	将序列重复 n 次	"Hello"*2	**HelloHello**
<, >, ==	字符串：比较 ASCII 码，多个字符串从左往右比较	"he" > "He"	True
	对列表中的元素逐个比较大小	[1,2,3] > [1,2]	True

2.7.2 集合常用运算符

对于集合常用的运算符，如表 2-16 所示。

表 2-16 集合常用运算符

运算符	描述	输入示例	返回值
==，!=	比较 2 个集合是否相同。	{1,2,3} == {1,2}	**False**
<，<=	查看集合是否是子集、真子集。	{1,2} < {1,2,3}	**True**
&, |, –, ^	求集合的交集、并集、差集、对称差	{1,2,3} & {1,2,4}	{1, 2}

2. 8 内置函数的操作

对于本章介绍的数据结构，同样可以用 python 中的内置函数。如第一章的 help()，id()，dir()，type()……，都是通用的，也有一些是各个类型所特有的如各个类型转换函数 str()，bytes()，list()，tuple()，set()。除此之外，常用的还包括：

表 2-15 复合数据结构常用内置函数

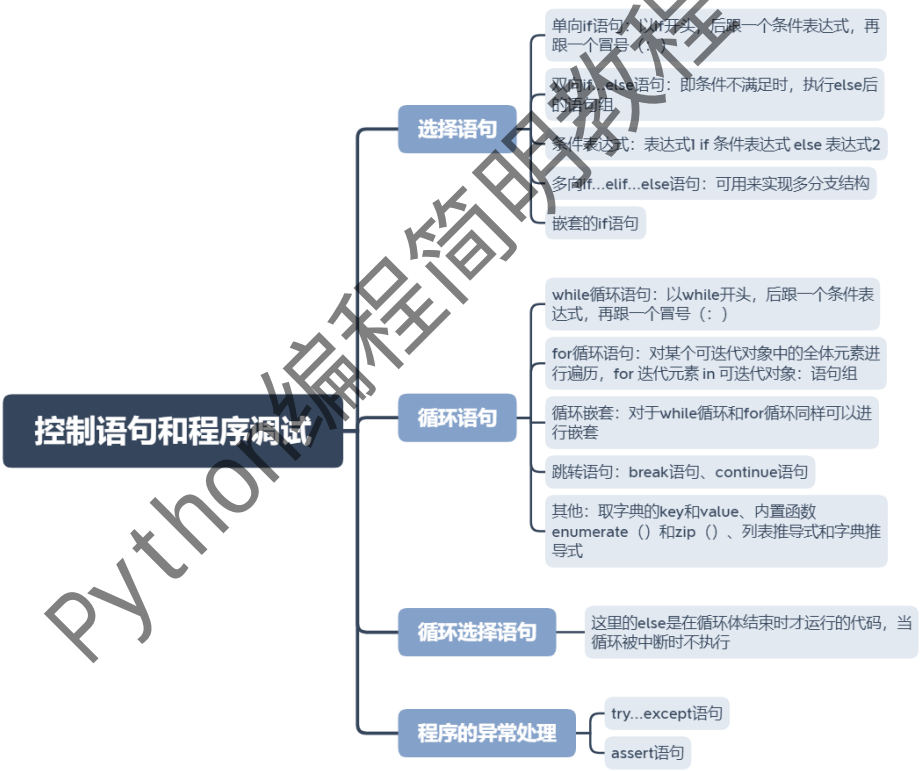
方法	描述	输入示例	返回值
len(s)	返回本章数据结构长度。	len({'a':1,'b':2})	2
max(iterable [,args…][key]) min(iterable [,args…][key])	返回本章数据结构最大值/最小值（字符串按 unicode 编码，字典返回键）。	max({'a':1,'b':2}) min("好好学习")	'b' 习
all(iterable)	元素都为真，返回 True,否则返回 False	all([1,'','h'])	
any(iterable)	有元素为真，返回 True,否则返回 False	any([1,'','h'])	
sum(iterable[,start])	对序列，集合求和。	sum([1,2,3])	6
sorted(iterable[, cmp[, key[, reverse]]])[1]	对序列进行排序，reverse 默认 False。返回一个新的重新排序的列表。	sorted('hello')	['e', 'h', 'l', 'l', 'o']
reversed(iterable)[2]	对序列进行翻转，reversed()函数返回翻转后的迭代器对象。	略	略

[1] 该内置函数比较复杂，这里仅提供 2 个常用参数，其他常用参数将在第 4 章函数中进行讲解。
[2] 该函数的返回对象是一个字符

3. 控制语句和程序调试

知识目标：	技能目标
1，掌握 if 语句，if...else 语句，条件表达式，if...elif...else 语句以及嵌套的 if 语句 2，掌握不同的循环语句 3，学习使用对程序异常的处理	1，具有编写选择和循环语句的开发能力 2，具有面对程序异常的处理能力，解决实际问题

⌘ 学习地图：



结构化程序设计认为程序流程可用三种基本逻辑结构（顺序结构[1]，选择结构，循环结构）来编制，用来管理程序流程的语句称为控制语句。python 中常用控制语句有包括：

- ✓ 选择结构语句：if 语句、if…else 语句和 if…elif…else 语句
- ✓ 循环结构语句：for 语句、while 语句、break 中断语句、continue 继续执行语句

[1] 本书前 2 章所用的顺序结构，这里不再说明。

✓ 选择循环嵌套语句：for…else 语句，while…else 语句

此外，在程序运行中，常会出现一些异常情况。因此，掌握 vs code 下的程序调试，快速修复错误十分重要。Python 同样内置了一系列异常处理语句，以避免程序因这些问题而终止运行。如 assert 语句、raise 语句、try…except…else…finally 语句

3.1. 选择语句

选择语句又称为分支语句，条件语句。该语句常用来判定条件是否满足，根据判断的结果（即 True 或 False）决定是否执行给出的操作。常用的语句有 if 语句、if…else 语句和 if…elif…else 语句。

3.1.1. 单向 if 语句

if 语句是最为简单的单选结构。其基本的语法结构如下：

```
if 条件表达式：
    语句组
```

if 语句以 if 开头，后跟一个条件表达式（由**关系运算符**或**逻辑运算符**按一定语法规则组成的式子，输出**布尔值**）；再后跟一个冒号（:），该冒号标志着 if 条件控制的开始，只有**布尔表达式**为 True，才执行下面的语句组。而需执行的语句组前会用 Tab 键插入 4 个半角的空格，称为缩进。Python 通过缩进的方式区隔 if 语句中的程序代码块。

下面用流程图说明这个 if 语句。

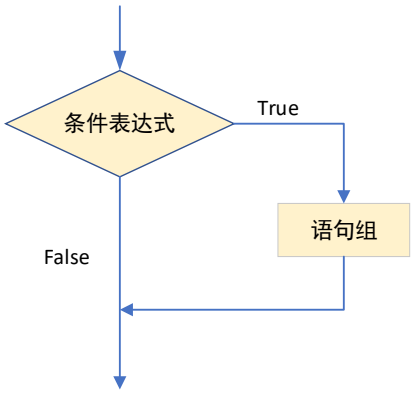


图 2-1 if 语句的流程图

【示例-1】 输入 2 个整数，比较大小，按从小到大的顺序输出。

```
num1,num2 = eval(input('请输入 2 个整数:'))
if num1 > num2:
    num1,num2 = num2, num1
print('输入的 2 个整数从小到大排序为：', num1, num2)
```

执行结果如下：

```
请输入 2 个整数:20,30
输入的 2 个整数从小到大排序为： 20 30
```

3.1.2. 双向 if…else 语句

与 if 语句不同，if…else 语句是最为简单的双选结构，即条件不满足时，执行 else 后的语句组。其基本的语法结构如下：

```
if 条件表达式：
    语句组 1
else:
    语句组 2
```

即如果条件表达式为 True，即执行语句组 1。布尔表达式为 False，即执行语句组 2。

下面用流程图说明这个 if…else 语句。

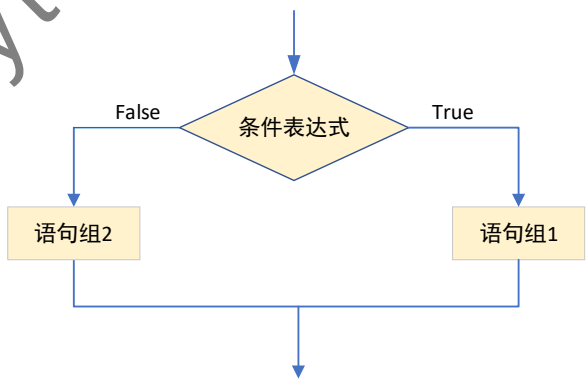


图 2-2 if…else 语句的流程图

【示例-2】 判断是否年满 18 周岁，已经是成人。

```
age = int(input('判断是否成年，请输入当前实际年龄：'))
if age<18:
    print('还未成年！！！')
else:
```

```
05          print('已成年！！！')
```

执行结果如下（读者可自行输入年龄，查看结果）：

```
01      判断是否成年，请输入当前实际年龄：16
✓       还未成年！！！
```

3.1.3. 条件表达式

除此之外，python 还提供了条件表达式用于赋值。其基本的语法结构如下：

```
表达式 1 if 条件表达式 else 表达式 2
```

如果条件表达式为 True，则使用表达式 1；如果条件表达式为 False，则使用表达式 2。

【示例-5】求 2 个数的最大值。

```
01  num1, num2 = eval(input('请输入 2 个整数:'))
02  max = num1 if num1 > num2 else num2
03  print('2 个数的最大值为：',max)
```

执行结果如下（读者可自行输入，查看结果）：

```
01      请输入 2 个整数:10,100
03      2 个数的最大值为： 100
```

3.1.4. 多向 if…elif…else 语句

elif 语句与 if 语句配合使用，可用来实现多分支结构。其基本的语法结构如下：

```
if 条件表达式 1：
    语句组 1
elif 条件表达式 2：
    语句组 2
elif 条件表达式 3：
    语句组 3
…
else：
    语句组 n
```

即如果条件表达式 1 为 True 则执行语句组 1，然后离开条件判断；否则检查条件表达式 2，如果为 True 则执行语句组 2，然后离开条件判断；否则继续坚持条件表达式 3……，如果

所有条件表达式都不满足，则执行 else 语句。

下面用流程图说明这个多向 if…elif…else 语句。

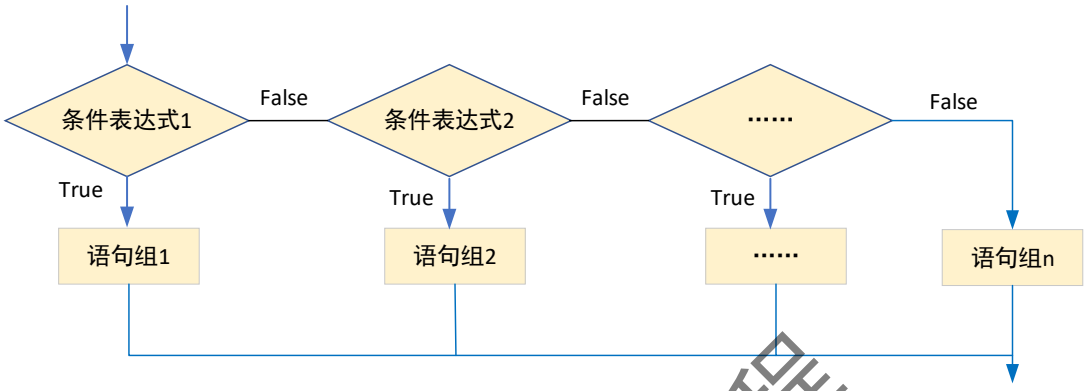


图 2-3 if…elif…else 语句的流程图

【示例-3】 输入分数成绩，自动转换为优、良、中、及格和不及格等级。

- **方法 1**

```
score = int(input('请输入 0~100 范围内的分数：'))
if score >= 90:
    print('优秀')
elif score >= 80:
    print("良好")
elif score >= 70:
    print("中等")
elif score >= 60:
    print("及格")
else:
    print("不及格")
```

执行结果如下（读者可自行输入分数，查看结果）：

```
请输入 0~100 范围内的分数：85
良好
```

- **方法 2：实际中也可使用字典来实现（执行结果略）。**

```
score = int(input('请输入 0~100 范围内的分数：'))
dic = {10:'优秀',9:'优秀',8:'良好',7:'中等',6:'及格'}
if score < 60:
    print("不及格")
else:
    print(dic[score//10])
```

【示例-4】 设计算法，实现华氏温度和摄氏温度的转换。其转换算法如下：① C=(F-32)/1.8 ; ② F=C*1.8+32。

```
tempStr = input('请输入带符号的温度值:').strip()
if tempStr[-1] in 'Ff':
    c = (eval(tempStr[:-1]) - 32)/1.8
    print(f"转换后的温度为{c:.2f}C。")
elif tempStr[-1] in 'Cc':
    f = 1.8 * eval(tempStr[:-1]) + 32
    print(f"转换后的温度为{f:.2f}F。")
else:
    print('输入格式错误。')
```

执行结果如下（读者可自行输入年龄，查看结果）：

```
请输入带符号的温度值:28C
转换后的温度为 82.40F。
```

3.1.5. 嵌套的 if 语句

除此之外，上述的选择语句还可进行相互嵌套。通过示例说明如下：

【示例-5】判断闰年，能够被 400 整除，或者能够被 4 整除但不能被 100 整除。

```
year = int(input('请输入年份：'))
if year%400 == 0:
    print(year,'是闰年')
else:
    if year%100 == 0:
        print(year,'不是闰年')
    elif year%4 == 0:
        print(year,'是闰年')
    else:
        print(year,'不是闰年')
```

执行结果如下（读者可自行输入年份，查看结果）：

```
请输入年份：1996
1996 是闰年
```

3.2. 循环语句

循环语句能够使部分语句组重复执行，是另一种常见的语句结构。Python 提供了 while 循环和 for 循环两种循环类型。

3.2.1. while 循环语句

while 语句用于循环执行程序，其基本的语法结构如下：

```
while 条件表达式：
    语句组
```

即如果条件表达式为 True，循环执行语句组。布尔表达式为 False，跳出循环。

下面用流程图说明这个 while 语句。

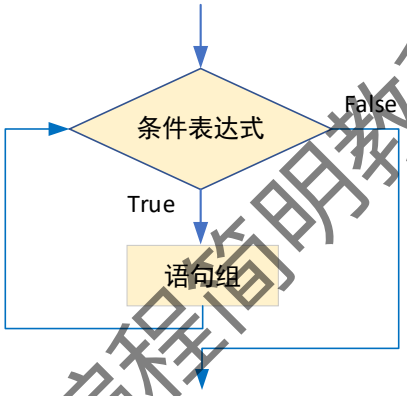


图 2-3 while 循环语句的流程图

【示例-6】求 1~100 的和。

```
01  sum = 0
02  i = 1
03  while i <= 100:
04      sum = sum + i
05      i = i+1
06  print("1~100 的和为", sum)
```

执行结果如下（读者可自行输入，查看结果）：

```
06      1~100 的和为 5050
```

【示例-7】输入某个整数 a，用迭代法求平方根，对应的公式为 $x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$，其中 $x_0=\frac{a}{2}$，且求出的精度为前后项的差的绝对值小于 10^{-5}。

```
01  a = int(input('请输入一个数：'))
```

```
x0 = a/2
x1 = (x0+a/x0)/2
while abs(x1-x0) > 0.00001:
    x0 = x1
    x1 = (x0+a/x0)/2
print("对应的平方根为：", x1)
```

执行结果如下（读者可自行输入，查看结果）：

```
请输入一个数：9
对应的平方根为： 3.0
```

3.2.2. for 循环语句

for 循环是另一种常见的循环结构，它会对某个可迭代对象中的全体元素进行遍历。如第 1 章提及的字符串就是可迭代对象，第 3 章的列表、元组、字典都是可迭代对象。其基本的语法结构如下：

```
for 迭代元素 in 可迭代对象：
    语句组
```

对应的执行过程，每次循环，都从可迭代对象中依次取出元素。

For 语句的流程图

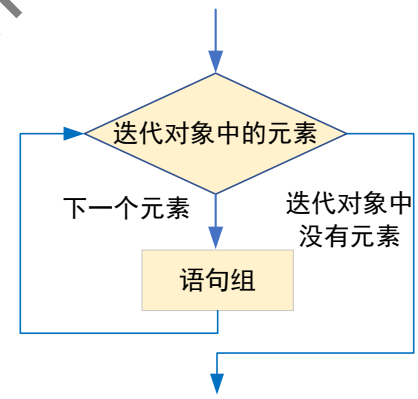


图 2-3 for 循环语句流程图

【示例-8】取出字符串中的相应元素。

```
for i in '中国人':
    print(i)
```

执行结果如下：

```
中
国
人
```

此外可迭代对象都可用于 for 循环，如内置函数 range()其返回值同意为。

【示例-9】使用 for 循环，求 1~100 的和。

```
x,y = 0,''
for i in range(1,101):
    x += i
print(f"1~100 的和为：{x}！！！")
for j in reversed('hello'):
    y += j
print(f'hello 的反向字符串为：{y}!!!')
```

执行结果如下：

```
1~100 的和为：5050！！！
hello 的反向字符串为：olleh!!!人
```

3.2.3. 循环嵌套

对于 while 循环和 for 循环同样可以进行嵌套。这里不再阐述，示例如下：

【示例-10】使用 for 循环，输出九九乘法表（for 循环嵌套）。

```
for i in range(1,10):
    for j in range(1,i+1):
        print(f"{j}*{i}={j*i}",end="\t")
    print()
```

执行结果如下：

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12 4*4=16
1*5=5  2*5=10 3*5=15 4*5=20 5*5=25
1*6=6  2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7  2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8  2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9  2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

【示例-11】 files 列表中含一系列文件名，提取“.py”结尾的 python 程序，并放到新列表。

方法 1

```
files = ['ex01.py','ex02.py','ex04.py','ex01.doc','ex02.txt']
py = []
for file in files:
    if file.endswith('.py'):
        py.append(file)
print(py)
```

执行结果如下：

```
['ex01.py', 'ex02.py', 'ex04.py']
```

方法 2（该方法是错误方法，可以思考下为什么）

```
files = ['ex01.py','ex02.py','ex04.py','ex01.doc','ex02.txt']
for file in files:
    if not file.endswith('.py'):
        files.remove(file)
print(files)
```

执行结果如下（为什么没删除 ex02.txt）：

```
['ex01.py', 'ex02.py', 'ex04.py', 'ex02.txt']
```

3.2.4. 跳转语句

跳转语句能够改变程序的执行顺序，可以实现程序的跳转。本节重点介绍循环中常用的 break 跳转语句和 continue 语句。

1. break 语句

break 语句可用于本节介绍的 while 和 for 循环结构，它的作用是某些条件发生时强行退出循环体，不再执行循环体中剩余的语句。这个指令常与 if 语句一起使用。

【示例-12】使用 for 循环，完成密码输入判断。（1）当输入用户名和密码小于 3 次：输入正确，显示“欢迎登陆系统”；输入错误，显示“用户名或密码错误，请再次输入，剩余输入？次”。（2）当输错三次后退出。

方法 1（使用 for 循环）

```
user,password  = "abc","123"
for i in range(3):
    username = input("username:")
```

```
        password = input("password:")
        if username == user and password == password:
            print("欢迎登陆系统")
            break
        else:
            print(f'用户名或密码错误，剩余输入{2-i}次')
```

方法 2（使用 while 循环）

```
user,password  = "abc","123"
count = 0
while count < 3:
    count += 1
    username = input("username:")
    password = input("password:")
    if username == user and password == password:
        print("欢迎登陆系统")
        break
    else:
        print(f'用户名或密码错误，剩余输入{3-count}次')
```

2. continue 语句

continue 语句同样可用于本节的 while 和 for 循环结构，它的作用是某些条件发生时强行退出本次循环，不再执行循环体中剩余的语句。这个指令同样常与 if 语句一起使用。

【示例-13】 求列表中及格学生的平均成绩。

```
scores = [90,87,56,33,67,89]
s = n =0
for score in scores:
    if score < 60:
        continue
    s += score
    n += 1
print(f"及格的人数为{n}，及格学生的平均分为{s/n:.2f}")
```

执行结果如下（为什么没删除 ex02.txt）：

```
及格的人数为 4，及格学生的平均分为 83.25
```

3.2.5. 其他

除此之外，对于 for 循环还要关注如下场景。

1. 取字典的 key 和 value

对于字典中的键和值，使用 for 循环同样可以依次取出。其中 dict.keys() 返回字典的键的列表；dict.values()返回字典的值的列表；dict.items() 返回迭代字典的键-值对的列表。

【示例-14】 通过 for 循环分别取字典的 key 和 value。

```
my_dict = {'语文':89,'数学':100}
for key in my_dict.keys():   #取字典的 key
    print('key:',key)
for value in my_dict.values():   #取字典的 value
    print('value:',value)
for key,value in   my_dict.items(): #取字典的 key 和 value
    print('Key:',key)
    print('Values：',value)
```

执行结果如下（为什么没删除 ex02.txt）：

```
key: 语文
key: 数学
value: 89
value: 100
Key: 语文
Value： 89
Key: 数学
Value： 100
```


2. 内置函数 enumerate()和 zip()

Python 中内置的 enumerate()函数可以将可迭代对象数值的元素用计数值与元素配对的方式传回，返回的数据称为 enumerate 对象；Python 中内置的 zip() 函数可以将多个可迭代对象“压缩”成一个 zip 对象。所谓“压缩”，其实就是将这些序列中对应位置的元素重新组合，生成一个个新的元组。

其基本语法结构如下：

zip([iterable, ...])
enumerate(iterable[, start], stop[, step])
✓ **iterable：可迭代对象。**
✓ **start：开始值。省略时默认 0**

【示例-15】观察下列字符串使用 enumerate()函数和 zip()函数的结果。

```
print(type(enumerate([10,100])),type(zip(['a','b'],[10,100])))
print(list(enumerate([10,100])),list(zip(['a','b'],[10,100])))
for i,j in enumerate([10,100]):
    print(i,j)
for i,j in zip(['a','b'],[10,100]):
    print(i,j)
```

执行结果如下：

```
<class 'enumerate'> <class 'zip'>
[(0, 10), (1, 100)] [('a', 10), ('b', 100)]
0 10
1 100
a 10
b 100
```

3. 列表推导式

列表推导式是 Python 构建列表（list）的一种快捷方式，可以使用简洁的代码就创建出一个列表。其基本语法格式如下：

```
列表变量 = [表达式 for 循环变量 1 in 可迭代对象 1 for 循环变量 2 in 可迭代对象 2 in 条件表达式]
```

✓ 可迭代对象：如列表、字符串、元组和字典等。

【示例-16】 生成如下运算结果[1,4,9,16,25,36,49,64]和[1,4,16,25,49,64]。

```
print([i*i for i in range(1,9)])
print([i*i for i in range(1,9) if i%3!=0])
```

执行结果如下：

```
[1, 4, 9, 16, 25, 36, 49, 64]
[1, 4, 16, 25, 49, 64]
```

【示例-17】 6个包子，男人吃3个包子，女人吃2个包子，小孩吃1个包子，求组合情况。

```
print([(x,y,6-3*x-2*y) for x in range(3) for y in range(4) if 3*x+2*y<=6])
```

执行结果如下：

```
[(0, 0, 6), (0, 1, 4), (0, 2, 2), (0, 3, 0), (1, 0, 3), (1, 1, 1), (2, 0, 0)]
```

4. 字典推导式

与列表推导式相对应，字典同样具有推导式。其基本语法格式如下：

字典变量={键:值 for 循环变量1 in 可迭代对象1 for 循环变量2 in 可迭代对象2 in 条件表达式}
✓ 可迭代对象：如列表、字符串、元组和字典等。

【示例-18】 字典推导式示例。

```
d = {'a':10,'b':20,'c':3,'A':2,'B':5,'l':10}
d1 = {k:v for k,v in d.items() if v>=10}    #提取值大于等于10的键值对
d2 = {k:v for k,v in d.items() if k.islower()}    #提取键为小写字母的键值对
print(f"{d1}\n{d2}")
```

执行结果如下：

```
{'a': 10, 'b': 20, 'l': 10}
{'a': 10, 'b': 20, 'c': 3}
```

3.3. 循环选择语句

对于 while 和 for 循环，它们都还可以和 else 语句一起使用，与 if 语句中的 else 不同。s[1]。

带循环的 else 语句流程图如下：

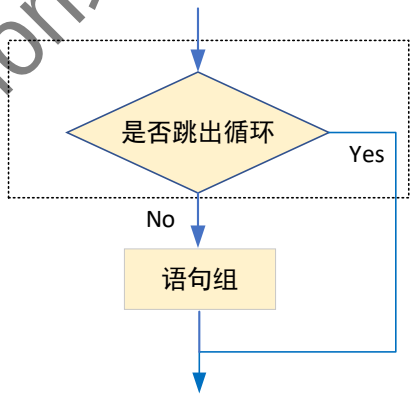


图 2-3 循环语句中含 else 对应流程图

【示例-19】 生成随机数，进行 1~10 以内的乘法测试，3 道答对，测试通过。

```
from random import *
for _ in range(3):    #迭代变量不使用，可用下划线表示
    i,j = randrange(1, 10),randrange(1, 10)
    k = int(input(f"{i}*{j}="))
```

[1] 如 break，return，抛出异常，其中 return 和抛出异常将在后面讲解。

```
05          if k != i*j:
06              print(f"本题错误，正确答案为{i}*{j} = {i*j}")
07              break
08      else:
09          print('3道答对，测试通过')
```

正确执行时，结果如下：

```
4*1=4
7*8=56
4*3=12
3道答对，测试通过
```

错误执行时，结果如下：

```
4*2=8
5*2=9
本题错误，正确答案为5*2=10
```

3.4. 程序的异常处理

除了上述的选择语句和循环语句外，程序运行时往往会出现错误，一旦碰上程序将终止执行。而程序的异常处理模块，可在发生异常时及时捕捉执行异常处理程序，使得程序可继续执行。

3.4.1. try…except 语句

1. 常见异常

程序异常是非常常见的现象，如下例所示。

【示例-20】 分别输入除数和被除数，完成除法运算。

```
01  x,y = eval(input('输入除数和被除数：'))
02  print(f'{x}除以{y}的值为{x/y}。')
```

当输入除数为 0 时，此时返回异常提示：①异常发生的代码为第 2 行；②异常类型为 ZeroDivisionError，具体内容为 division by zero。

```
01      输入除数和被除数：1,0
        Traceback (most recent call last):
```

```
        File "D:/ python_vs/chap03/ex20.py", line 2, in <module>
          print(f"{x}除以{y}的值为{x/y}。")
    ZeroDivisionError: division by zero
```

当输入中文符号(，)，此时返回异常提示：①异常发生的代码为第 1 行；②异常类型为 SyntaxError，具体内容为 invalid character in identifier。

```
    输入除数和被除数：1，0
    Traceback (most recent call last):
        File "D:/ python_vs/chap03/ex20.py", line 1, in <module>
          x,y = eval(input('输入除数和被除数：'))
        File "<string>", line 1
          1，0
           ^
    SyntaxError: invalid character in identifier
```

常见异常如表 3-1 所示。

表 3-1 常用异常列表

异常类	描述
ValueError	实际值与期望值不符
IndentationError	代码缩进错误
IndexError	序列索引不存在
AssertionError	断言失败
NameError	名字不存在
KeyError	映射（如字典）中的键不存在
AttributeError	属性错误（对象无指定名字的属性）
TypeError	类型出错
SyntaxError	代码语法错误
OSError	操作系统未能执行指定任务
ZeroDivisionError	除 0 错误

2. try…except 语句

异常类型时 python 异常中最重要的部分，也是程序处理异常的依据。Python 使用 try…except 实现异常处理。其基本的语法格式如下：

```
try:
    语句组 1
except [<异常名>,<异常名>,…] [as 别名]:
    语句组 2
except [<异常名>,<异常名>,…] [as 别名]:
    语句组 3
```

```
except [<异常名>,<异常名>,…] [as 别名]:
    语句组 4
……
主语句组
```

其中：异常名为可选参数，如果不指定异常类型，则表示捕获全部异常类型。设置别名，可方便异常对象的引用。其基本过程为：执行 try 下的语句组 1，如正常则跳离所有 except 语句组部分；如语句组 1 有错误异常，except 将捕获该异常，并将 except 中的异常与引发的异常相匹配，再执行该 except 后的语句组。当异常处理完，程序继续执行主语句组。

【示例-21】 try…except 语句分别输入除数和被除数，完成除法运算。

- **方法 1**

```
while True:
    try:
        x,y = eval(input('输入除数和被除数：'))
        print(f"{x}除以{y}的值为{x/y}。")
        break
    except (SyntaxError, ZeroDivisionError) as e:
        print(e.__str__)
    print("程序继续执行")
```

- **方法 2**

```
while True:
    try:
        x,y = eval(input('输入除数和被除数：'))
        print(f"{x}除以{y}的值为{x/y}。")
        break
    except SyntaxError as e:
        print(e.__str__)
    except ZeroDivisionError as e:
        print(e.__str__)
    print("程序继续执行")
```

对应的输出为：

```
输入除数和被除数：1,0
<method-wrapper '__str__' of ZeroDivisionError object at 0x0000022A6AACD860>
程序继续执行
输入除数和被除数：1，0
<method-wrapper '__str__' of SyntaxError object at 0x0000022A6AA94A40>
程序继续执行
输入除数和被除数：1,1
```

```
1 除以 1 的值为 1.0。
```

3. 增加 else

此外，还可在 try…except 语句后增加 else，该指令主要目的是 try 内的语句组正确时，可执行 else 内的语句组。当 try 内的语句组错误时，不执行 else 内的语句组。其语法结构如下：

```
try:
    语句组 1          #可能发生异常的语句组
except [<异常名>,<异常名>,…] [as 别名]:
    语句组 2          #异常发生时执行的语句组
……
else:
    语句组 3          #异常不发生时执行的语句组
```

【示例-22】 try…except…else 语句分别输入除数和被除数，完成除法运算。

```
num = 0
while True:
    try:
        x,y = eval(input('输入除数和被除数：'))
        print(f"{x}除以{y}的值为{x/y}。")
    except (SyntaxError, ZeroDivisionError) as e:
        num += 1
        print(e.__str__)
        print("程序继续执行")
    else:
        print(f"一共用了{num}次完成了异常的处理")
        break
```

对应的输出为：

```
输入除数和被除数：1,0
<method-wrapper '__str__' of ZeroDivisionError object at 0x000002157823D8B0>
程序继续执行
输入除数和被除数：1,1
1 除以 1 的值为 1.0。
一共用了 1 次完成了异常的处理
```

4. 增加 finally

此外，还可在 try 语句后增加 finally，如 try 后有 except 或 else，则 finally 必须放在最后，同时不论是否有异常都必须执行 finally 中的代码。这个需求在后面数据采集、文件存

储、网络爬虫时都很有用。

3.4.2. 使用 assert 语句

assert 语句称为断言，它的语法格式为：

```
assert 条件表达式, '字符串'
```

其中，条件表达式为 True 时，程序跳过字符串继续执行；条件为 False 时，程序终止同时将字符串的内容输出到 Traceback 的字符串内。

【示例-23】 assert 语句查看学生成绩，成绩必须是 0~100 之间。

```
01  try:
02      score = float(input("请输入学生成绩："))
03      assert score>=0 and score<=100, '学生成绩不在指定范围内'
04      print(f'当前学生的成绩为{score}。')
05  except ValueError as e:
06      print(e.__str__)
```

对应的输出为（这里只示范 assert）：

```
请输入学生成绩：101
Traceback (most recent call last):
  File "g:/BaiduNetdiskWorkspace/python 基础/版本 1/python_vs/chap03/ex23.py", line 3, in <module>
    assert score>=0 and score<=100, '学生成绩不在指定范围内'
AssertionError: 学生成绩不在指定范围内
```