From C To Python

目录

前言

- 本教程旨在帮助人们在掌握 C 语言的情况下，快速学会 Python。（关于 C 语言可以参考我的“C 语言教程”（https://juejin.cn/post/7418134213654642688）。
- Python 和 C 语言一样属于编程语言，不过 Python 要更加地易用、易编写和易学习，因此有非常多的人（包括中学生、不以编程作为职业的人）都选择以 Python 作为自己的入门语言和主要使用语言。
- 本教程并没有涵盖所有的 Python 内容，但是可以在以后的实践过程中渐渐掌握其他的知识点。
- 阅读时不要忽略脚注中的内容。
- 文档中有些地方可以点击跳转到相应的章节或网页链接，如“另见……”等。
- 已为关键概念添加英文名。学习编程的同时，学会对应的英文术语也很重要。

第一章 运行

运行	C	Python
预处理 （Preprocessing）	C 语言属于**编译型语言**，需要通过**编译器**（Compiler），**编译**（Compile）成二进制的机器指令才能运行	Python 属于**解释型语言**，不需要**编译**（Interpret），但需要通过**解释器**（Interpreter）运行
入口点 （Entrypoint）	`main()`函数	从源文件的第一行开始运行

（终端）**运行**	`# gcc 为 C 语言的编译器`
	`gcc -i main.c -o main.exe`
	`main.exe`
		`# python3 为 Python 的解释器`
		`python3 main.py`

注：

- 另见附录：**安装 Python**
- 另见附录：**REPL**

第二章 部分语法

语法	C	Python
语句 （Statement）	每条语句之间 必须用**分号**（Semicolon）分隔	一般情况下，语句之间**按行**分隔，不需要通过分号分隔。 但是如果需要在同一行内同时写入多行代码，就需要使用分号分隔

注释
（Comments）
	`// 单行注释`
	
	`/*`
	`  多行注释`
	`*/`
		`# 单行注释`
		
		`"""`
		`多行注释`
		`"""`
		
		`'''`
		`多行注释`
		`'''`

代码块 （Code Block）	`{` `}`	`statement:` `  pass`

注：

- Python 的代码块由**缩进**（Indent）决定，只有缩进相同且相邻的代码才属于同一代码块
- Python 的代码块无法独立存在，只能在特定语句[1]后出现
- Python 的代码块内必须有代码（若代码块内不需要执行任何代码，用关键字 `pass` 作为占位符）
- 如果 Python 的代码块内只有一行代码，可以将其和冒号写于同一行

[1]控制语句、函数实现等

第三章 变量

§3.1 变量的声明、赋值和使用

C	Python
C 语言中的变量，必须先**声明**（Declare），再**赋值**（Assign），最后**使用**。	Python 的变量[2]不需要声明，但是必须先**赋值**，再**使用**。[3]
C 语言的变量在声明时必须指明类型，且之后类型永远不会改变。[4]	Python **不需要**指明变量类型，并且变量的类型可以发生**改变**。[5]

§3.2 常用变量总览

注:

- Python 的整数理论上没有大小范围，浮点数的绝对值的大小范围约为$[2.225 \times 10^{-308}, 1.8 \times 10^{308}]$
- 虽然 Python 没有**数组**，但是 Python 有功能比数组更强大的**列表**[6]

类型	C	Python	Python 名

整数（Integer）	`short x = 1;`
	`int x = 1;`
	`long long x = 1;`
	`unsigned short x = 1;`
	`unsigned int x = 1;`
	`unsigned long long x = 1;`
		`x = 1`	`int`

浮点数 (Floating Point Number)	`float x = 3.14;` `double x = 3.14;`	`x = 3.14`	`float`
复数 （Complex Number）	无	`x = complex(1, 2) # 1 + 2i`	`complex`
字符（Character）	`char x = 'a';`	无	无
布尔[7]（Boolean）	`bool x = true;`	`x = True`	`bool`
字符串（String）	`char x[] = "Hello\nWorld";` `char* x = "Hello\nWorld";`	`x = "Hello\nWorld"`	`str`
数组（Array）	`int x[3] = {1, 2, 3};`	无	无
列表（List）	无	`x = [1, 2, 3]`	`list`
元组（Tuple）	无	`x = (1, 2, 3)` `x = (1,) # 仅有一个元素的元组`	`tuple`
集合（Set）	无	`x = {1, 2, 3}`	`set`

字典（Dictionary）	无	`x = {`
		`  "name": "Mike",`
		`  "age": 18`
		`}`
			`dict`

区间[8]（Range）	`for (int i = 0; i < 5; i += 1);`
	`for (int i = 2; i < 5; i += 1);`
	`for (int i = 0; i < 5; i += 2);`
	`for (int i = 5; i > 0; i += -1);`
		`x = range(5)`
		`x = range(2, 5)`
		`x = range(0, 5, 2)`
		`x = range(5, 0, -1)`
			`range`

指针（Pointer）	`int* p = &x;`	无	无
空[9]	`NULL`	`None`	`NoneType`

[2]函数和类同理
[3]可以理解为赋值的同时也声明了这个变量
[4]因此 C 属于静态类型语言
[5]因此 Python 属于动态类型语言
[6]将在后文详细介绍
[7]并非 C 语言原生类型，需要 `#include <stdbool.h>`
[8]C 语言并**没有**区间，此处仅用于理解。区间特指**整数区间**

§ 3.3 特殊字面量

注:

- **字面量**（Literal Value）指一个固定值的表示方法
- Python 和 C 语言一样支持**转义字符**（Escape Character）

字面量	C	Python	备注

整数	`0b1011 // 二进制`
	`075 // 八进制`
	`0x7F // 十六进制`
		`0b1011 # 二进制`
		`0o75 # 八进制`
		`0x7F # 十六进制`
			C 语言的二进制字面量需要 C11 及以上版本

浮点数	`1e9`	`1e9`	“e9”表示“$\times 10^9$”

字符串	`"Hello\nWorld"`	`# 单行字符串`
		`"Hello\nWorld"`
		`'Hello\nWorld'`
		`# 多行字符串`
		`"""Hello`
		`World"""`
		`'''Hello`
		`World'''`
			无

§ 3.4 变量类型转换

从	到	C	Python	备注
浮点数	整数	`(int)x`	`int(x)`	无
整数	浮点数	`(float)x`	`float(x)`	无

整数	字符串	`char s[256];`
		`sprintf(s, "%d", x); // 十进制`
		`// 无法直接进行二进制转换`
		`sprintf(s, "%o", x); // 八进制`
		`sprintf(s, "%X", x); // 十六进制`
			`str(x) # 十进制`
			`bin(x) # 二进制`
			`oct(x) # 八进制`
			`hex(x) # 十六进制`
				C 语言需要 `#include <stdio.h>`

字符串	整数	`strtol(x, NULL, 10); // 十进制`
		`strtol(x, NULL, 2); // 二进制`
		`strtol(x, NULL, 8); // 八进制`
		`strtol(x, NULL, 16); // 十六进制`
			`int(x) # 十进制`
			`int(x, 2) # 二进制`
			`int(x, 8) # 八进制`
			`int(x, 16) # 十六进制`
				C 语言需要 `#include <stdlib.h>`

任意类型	布尔	无	`bool(x)`	无
可迭代对象[10]	列表	无	`list(x)`	无
可迭代对象[10]	元组	无	`tuple(x)`	无
可迭代对象[10]	集合	无	`set(x)`	无
可迭代对象[10]	字典	无	`dict(x)`	需要满足特定格式的序列，如 `[['k1', 1], ['k2', 2]]`

[9] C 语言并没有“空”这个类型，此处仅用于理解；需要 `#include <stddef.h>`

[10] 详情另见章节：**迭代**

§3.5 特殊赋值方式

Python 还有一种特殊的赋值方法：

```
x, y = 1, 2 # 等价于 x = 1; y = 2
```

等号左边为一连串的变量，右边为一系列的值（或表达式）。 赋值过程就是将右边的值（或表达式）一一对应到左边的变量。 等号右边也可以是序列，赋值过程也是一一对应。

借助这个功能我们可以实现快速交换变量：

```
x, y = y, x
```

有的时候，我们希望左边的变量可以被赋值给多个值，这个时候就可以用*：

```
x, *y = 1, 2, 3, 4 # 等价于 x = 1; y = [2, 3, 4]
x, *y, z = 1, 2, 3, 4, 5 # 等价于 x = 1; y = [2, 3, 4]; z = 5;
```

即，将单个变量赋值完之后，再把剩余的值全部分配给 y

第四章 运算符

§ 4.1 算术运算符

注:

- Python 的“**除**”的结果永远为**浮点数**
- Python 的“**整除**”的结果永远为**整数**（结果为**保留整数部分**的商）

运算符	C	Python
加	`x + y;`	`x + y`
减	`x - y;`	`x - y`
乘	`x * y;`	`x * y`
除	`x * 1.0 / y;`	`x / y`
整除	`(int)x / (int)y;`	`x // y`
取余	`x % y;`	`x % y`
乘方[11]	`pow(x, y);`	`x ** y`

§ 4.2 逻辑运算符

运算符	C	Python
与	`x && y;`	`x and y`
或	`x || y;`	`x or y`
非	`!x;`	`not x`

§ 4.3 关系运算符

注:

- Python 的等于和不等关系的运算符可以**连用**。比如 `1 < 2 < 3, 2 != 3 != 5`

运算符	C	Python
大于	`x > y;`	`x > y`
大于等于	`x >= y`	`x >= y`
小于	`x < y;`	`x < y`
小于等于	`x <= y;`	`x <= y`
等于	`x == y;`	`x == y`
不等于	`x != y;`	`x != y`
属于	无	`x in y`
不属于	无	`x not in y`
地址相同[11]	`&x == &y;`	`x is y`
地址不同[11]	`&x != &y;`	`x is not y`

§ 4.4 位运算符

运算符	C	Python
与	`x & y;`	`x & y`
或	`x | y;`	`x | y`
取反	`~x;`	`~x`
异或	`x ^ y;`	`x ^ y`
左移	`x << y;`	`x << y`
右移	`x >> y;`	`x >> y`

[11] C 语言代码仅用于理解

第五章 控制语句

§ 5.1 分支语句

分支	C	Python
如果	`if (x) {` `}`	`if x:` `  pass`
亦或是	`else if (x) {` `}`	`elif x:` `  pass`
否则	`else {` `}`	`else:` `  pass`

选择[12]	`switch(x) {`
	`  case 0:`
	`  break`
	`  default:`
	`  break;`
	`}`
		`match x:`
		`  case 0:`
		`  pass`
		`  case _:`
		`  pass`

§ 5.2 循环语句

注:

- C 语言的 `for` 只是 `while` 的变体
- 关于 Python 的 `for` 的详情，另见章节：**迭代**

循环	C	Python
`for`	`for (int i = 0; i < n; ++i) {` `}`	`for i in range(n):` `  pass`
`while`	`while (x) {` `}`	`while x:` `  pass`

§ 5.3 跳转语句

跳转	C	Python
跳出循环	`break;`	`break`
跳过一次循环	`continue;`	`continue`
函数返回	`return x;`	`return x`
代码跳转	`goto label;`	无

[12]需要 Python 3.10 及以上版本

第六章 函数

§ 6.1 基础语法

注:

- Python 的函数无法像 C 语言一样拆分“声明”和“实现”，换句话说 Python 的函数在声明的同时就要包含实现。

参数	返回值	C	Python

无	无	`void f() {`
		`}`
		
		`void f() {`
		`  return;`
		`}`
			`def f():`
			`  pass`
			
			`def f():`
			`  return`

无	有	`int f() {`
		`  return 0;`
		`}`
			`def f():`
			`  return 0`

有	无	`void f(int x) {` `}`	`def f(x):` `  pass`

有	有	`int f(int x) {`
		`  return x;`
		`}`
			`def f(x):`
			`  return x`

多个	无	`void f(int x, int y) {` `}`	`def f(x, y):` `  pass`

§6.2 变量作用域

C 语言中，根据作用域（Scope）的不同，变量[13]被分为局部变量（Local Variable）和全局变量（Global Variable）。在代码块内声明的变量均为局部变量，否则是全局变量。一个变量只能在作用域内被访问。

在 Python 中，情况较为复杂。Python 变量的作用域和代码块无关，而与“**闭包**”（Closure）有关。“闭包”是指一个封闭的代码区域。在闭包中创建的变量，作用域会被限制在闭包中，一旦离开闭包，变量就无法被再次访问。因此在**闭包内**创建的变量均为**局部变量**，否则为全局变量。

Python 最主要的闭包就是函数体，以及后文将会提到的“方法”体。观察下面的代码：

```
def f():
  print(x)

x = 0
f() # 输出 0
```

x 没有位于闭包内，所以为全局变量。在函数 `f()`中 `print(x)`会访问全局变量 `x`。

但是函数内一旦出现了对某个变量的赋值操作（不论在函数的哪个位置），这个变量就会变成局部变量。下面的代码就展示出了局部变量和全局变量的差异：

```
def f():
  x = 1
  print(x)

x = 0
f() # 输出 1
print(x) # 输出 0
```

但是有的时候我们想修改全局变量，而非创建局部变量，这个时候就需要使用“`global`”关键字，在函数内声明该变量为全局变量：

```
def f():
  global x # 声明 x 为全局变量
  x = 1
  print(x)

x = 0
f() # 输出 1
print(x) # 输出 1
```

[13]暂不考虑静态变量

§ 6.3 Python 专有语法

§ 6.3.1 默认值

Python 的函数参数可以有默认值。 当不指定参数的值时，参数取默认值。 比如：

```
def f(x=1):
  return x

f() # 返回 1
f(0) # 返回 0
```

注：没有默认值的参数必须全都在有默认值的参数前面

§ 6.3.2 传递参数

§ 6.3.2.1 位置参数

在 C 语言里，实际参数和形式参数是根据**位置**和**顺序**一一对应的：

```
void f(int x, int y, int z);
f(1, 2, 3); // 1 对应 x，2 对应 y，3 对应 z
```

在 Python 里，这样的参数叫做**位置参数**（Positional Argument）

§ 6.3.2.2 关键字参数

在 Python 里还有一种参数叫**关键字参数**（Keyword Argument）：

```
def f(x, y, z):
  pass

f(x=1, y=2, z=3)
```

即通过“形式参数名 = 实际参数”的方式传递参数

§ 6.3.3 变长参数

有的时候，一个函数的参数的数量并不固定（就像 C 语言的 `printf`），这个时候就需要变长参数。

变长参数同时支持位置参数和关键字参数，它们可以接收任意数量的参数。变长的位置参数需要在参数前添加一个星号，变长的关键字参数需要在参数前添加两个星号，比如：

```
def f(*args, **kwargs):
  pass

f(1, 2, x=1, y=2, z=3)
```

其中：

- `*args` 为元组。例子中 `args` 为`(1, 2)`
- `**kwargs` 为字典。例子中 `kwargs` 为`{'x': 1, 'y': 2, 'z': 3}`

第七章 结构体与类

§7.1 基础概念

C 语言的**结构体**和 Python 的**类**类似，都可以用于创建自定义的变量类型，让变量拥有自己的**属性**[14]（Property）。

在 Python 中，通过某个类创建的变量被称作这个类的“**实例**”（Instance）或“**对象**”（Object）

§7.2 方法

“**方法**”（Method）指类和实例专有的函数，这是 Python 的类和 C 语言的结构体的**最核心、最根本的区别**。

类专有的函数被称作“**静态方法**”，实例专有的函数被称作“**实例方法**”。一般情况下，方法都指**实例方法**

Python 和 C 一样，访问属性和方法都使用“.”（点号）

方法的调用方式和函数类似，区别在于需要在方法前面加上“类.”或“实例.”，比如：

```
A.f() # A 为类
x.f() # x 为实例
```

§7.2.1 魔术方法

Python 有这样一类方法，我们并不直接通过实例名.方法(...)的方式调用它，这样的方法就是**魔术方法**（Magic Method）。

魔术方法的名字都有一个特点：__方法名__，即名字前后都有两个下划线。

最常用的魔术方法是__init__，它会在创建对象的时候被调用，专门用来初始化对象的属性等。

还有一些魔术方法，如__add__、__sub__、__mul__等，它们分别对应了+、-、*等运算符。对类的实例使用这些运算符的时候，实际上就相当于调用这些方法。

在类中定义和运算符相对应的魔术方法的过程，被称作“**运算符重载**”（Operator Overloading）

§7.3 区别

区别	C	Python
属性	在定义的时候就必须明确和固定	可以在任何时候添加和删除
方法	没有	有
初始化（Initialize）	只能由用户手动初始化	主要通过__init__方法

[14] 也称“字段”（Field）

§7.4 基础语法

注:

- Python 的实例方法的第一个参数为实例自身，并且通常起名为“self”
- Python 的类的属性和 Python 的变量一样，都不需要声明，但需要先赋值后使用

结构体与类	C	Python

基础定义	`struct Point {`
	`  int x;`
	`  int y;`
	`};`
		`class Point:`
		`  def __init__(self, x, y):`
		`  self.x = x`
		`  self.y = y`

创建实例方法[15]	`int Point_add(struct Point* self) {`
	`  return self->x + self->y;`
	`}`
		`class Point:`
		`  def __init__(self, x, y):`
		`  self.x = x`
		`  self.y = y`
		
		`  def add(self):`
		`  return self.x + self.y`

创建变量	`struct Point p = {`
	`  .x = 1,`
	`  .y = 2`
	`};`
		`p = Point(1, 2)`

使用方法	`Point_add(&p);`	`p.add()`

§7.5 继承

有的时候，两个类可能会有共用的属性和方法。这个时候就可以通过“**继承**”（Inheritance），将一个类的属性和方法都复制到另一个类之中。比如:

```
class Person:
  def __init__(self, name):
    self.name = name

  def introduce(self):
    print(f"Hi, I'm {self.name}!")

class Student(Person): pass
class Teacher(Person): pass
```

这里 Student 和 Teacher 都继承了 Person 类，它们都拥有 Person 的`__init__`方法和属性 `name`。其中被继承的类（Person）被称为**父类**（Parent Class），继承者（Student 和 Teacher）被称为**子类**（Child Class）。

子类可以覆盖父类的属性和方法，比如:

```
class Student(Person):
  def introduce(self):
    print(f"Hi, I'm Student {self.name}!")
```

如果在覆盖了父类的方法后，还想要在子类的方法中调用父类的方法，可以使用 super:

```
class Student(Person):
  def introduce(self):
    super().introduce()
    print("I'm a student!")
```

[15] C 语言没有方法的概念。此处代码仅用作理解使用

第八章 迭代

§8.1 迭代

“**迭代**”（Iteration）指重复执行一组代码的过程。比如，在 C 语言中遍历数组：

```
int a[5];
for(int i = 0; i < 5; ++i)
  printf("%d\n", a[i]);
```

每一次循环都可以被看作是对数组的一次迭代。在 Python 里有许多种可以被迭代的变量，它们被统称为“**可迭代对象**”（Iterable）。最常见的可迭代对象就是列表，它和 C 语言的数组类似，但是它有许多独特的功能和特性（详情请阅读后面的序列章节）。

§8.1.1 for 循环

所有可迭代对象都可以使用 for 循环遍历，其语法为“`for 元素 in 可迭代对象`”。比如：

```
for i in [4,5,6]:
  print(i)
```

输出结果为：

```
4
5
6
```

§8.1.2 yield 关键字与生成器

有的时候，我们希望函数可以返回多个值，这个时候就可以使用“`yield`”关键字。比如：

```
def f():
  yield 1
  yield 2
  yield 3
```

`yield` 和 `return` 类似，都可以让函数返回一个值，但不同的是 `yield` 不会让函数停止执行，因此我们可以借助它让函数返回多个值。

调用有 yield 的函数之后，它会返回一个名为“**生成器**”（Generator）的对象。顾名思义，“生成器”就是可以生成多个值的对象。 由于它存储了多个值，因此它可以被遍历，属于可迭代对象。 我们可以用 `for` 循环遍历生成器生成的值。比如：

```
for i in f():
  print(i)
```

输出结果为：

```
1
2
3
```

§8.1.3 生成器的简化写法

生成器还有一个简化写法：

```
(表达式 for 变量 in 可迭代对象)
# 比如 (i * i for i in [1, 2, 3]) 结果为 1, 4, 9
```

这种写法从前往后被分为两部分：**表达式部分**和**循环部分**。表达式部分就是需要生成的值，循环部分就是表达式中的循环变量。

生成器中的“循环部分”还可以是多重循环，比如：

```
(i + j for i in [1, 2] for j in [3, 4]) # 生成结果为 4, 5, 5, 6
```

“循环部分”的后面还可以加入 `if` 判断进行元素筛选：

```
(i for i in [1, 2, 3, 4] if i % 2 == 0) # 生成结果为 2, 4
```

§8.1.4 惰性求值

假如你尝试输出（`print`）一个生成器，会发现输出结果为“`<generator object <genexpr> at ...>`”。实际上，生成器并不会立刻生成出所有值，而是只有被迭代的时候才会生成值。这种特性被称为“**惰性求值**”（Lazy Evaluation），意思是它不会积极地求出值，而是只有等到被需要的时候再求。

§8.2 容器

“**容器**”（Container）指可以存储多个值的变量类型。它支持运算符 `in` 和 `not in`，即用来判断某个值是否存在于容器中。

§8.2.1 元素个数

所有容器都可以使用 `len(x)`函数获取容器内的元素个数。

注：当使用 `if` 判断容器内是否有元素时，可以直接这么写：

```
a = [1, 2]
if a:
  print('Yes')
```

实际上“`if a`”调用了“`bool(a)`”，即将 `a` 转换为布尔值后再进行的判断。只要容器内有元素，`bool(a)`就是 `True`，否则为 `False`。

§8.2.2 索引/下标

（注：并非所有容器都支持通过索引/下标的方式访问元素）

和 C 语言的数组类似，访问容器中的元素，需要借助“索引/下标”（Index）。

每种类型的索引将在后文中介绍。

§8.3 集合（Collection）

“**集合**”（Collection）指**可迭代**的**容器**。

§8.3.1 集合（Set）

集合（Set）和数学中的集合类似，不能有重复的元素。同时集合中的元素永远保持有序[16]，但是集合**无法使用**索引等相关功能。

集合创建另见章节：**常用变量总览**

集合的常用方法：

功能	参数	举例	结果

添加元素	`# obj: 元素`
	`add(obj)`
		`a = {1, 2}`
		`a.add(3)`
		`print(a)`
			`{1, 2, 3}`

清空集合	`clear()`	`a = {1, 2, 3}`
		`a.clear()`
		`print(a)`
			`[]`

复制集合	`copy()`	`a = {1, 2, 3}`
		`b = a.copy()`
		`print(b)`
			`{1, 2, 3}`

扩充集合	`# iterable: 可迭代对象`
	`update(iterable)`
		`a = {1, 2}`
		`a.update({3, 4})`
		`print(a)`
			`{1, 2, 3, 4}`

删除元素	`# value: 元素`
	`remove(value)`
	`# 元素不存在不会导致报错`
	`discard(value)`
	`# 移除最小的元素并返回它`
	`pop()`
		`a = {1, 2}`
		`a.remove(2)`
		`print(a)`
		`print(a.pop())`
		`print(a)`
			`{1}`
			`1`
			`{}`

集合运算	`union(*others) # 求多个集合的并集`
	`intersection(*others) # 求多个集合的交集`
	`difference(*others) # 求多个集合的差集`
		`{1}.union({2}, {3})`
		`{1, 2}.intersection({2, 3})`
		`{1, 2}.difference({2, 3})`
			`{1, 2, 3}`
			`{2}`
			`{1}`

集合关系	`issubset(other) # 是否是 other 的子集` `issuperset(other) # 是否是 other 的超集`	`{1, 2}.issubset({1, 2, 3})` `{1, 2, 3}.issuperset({1, 2})`	`True` `True`

[16]指元素始终保持从小到大排序。对于无法直接排序的值，Python 会将其转换为哈希值后再排序（哈希值：https://baike.baidu.com/item/Hash）

§8.3.2 字典

字典（Dictionary）存储的元素为一个值到另一个值的**映射**（Mapping）关系。比如对于下面的字典 a：

```
a = {
  "name": "Mike",
  "age": 18
}
```

`"name"`被映射为了`"Mike"`，`"age"`被映射为 `18`。并且我们把映射前的值称作“**键**”（Key），映射后的值称为“**值**”（Value），由它们二者组成的整体被称为“**键值对**”（Key-Value Pair）。因此我们也可以把字典看成由键值对组成的集合（Collection）。

字典和数学中的（单值）函数类似，一个键只能被映射成一个值（当然不同的键可以被映射为同一个值），因此字典中存储的键是不重复的，并且我们可以用“键”作为**索引**来访问“值”。

与集合（Set）相同或相似的方法：

字典	集合	备注
`clear()`	`clear()`	无
`copy()`	`copy()`	无
`popitem()` `pop(key, default=None)`	`pop()`	字典的 `popitem()`和集合的 `pop()`功能一样； 字典的 `pop` 有参数 `key` 和 `default`， 它的 `pop` 会删除 `key` 所对应的键值对，并返回对应的值。 如果该键不存在则返回 `default`
`update(other)`	`update(other)`	添加字典 `other` 中的所有键值对。 如果键已存在则**覆盖**原有的值

字典的常用方法和操作：

功能	参数	举例	结果

添加元素（映射）	`a[key] = value`	`a = {}`
		`a[1] = 2`
		`print(a)`
			`{1: 2}`

获取字典中的所有元素	`# 获取所有键值对`
	`items()`
	`# 获取所有键`
	`keys()`
	`# 获取所有值`
	`values()`
		`dic = {`
		`  1: 2,`
		`  2: 3`
		`}`
		`dic.items()`
		`dic.keys()`
		`dic.values()`
			`dict_items([(1, 2),(2, 3)])`
			`dict_keys(['name', 'age'])`
			`dict_values(['S', 18])`

获取值或默认值	`get(`
	`  # 键`
	`  key,`
	`  # 键不存在时返回的默认值`
	`  default=None`
	`)`
		`dic = {1:'1', 2:'2'}`
		
		`dic.get(0, 99)`
		`dic.get(1, 99)`
			
			`99`
			`'1'`

设置默认值	`setdefault(`
	`  key,`
	`  # 键不存在时让键映射为 default`
	`  default=None`
	`)`
		`dic = {1:'1', 2:'2'}`
		`dic.setdefault(0, 5)`
		`print(dic)`
		`dic.setdefault(1, 5)`
		`print(dic)`
			
			`{0:5, 1:'1', 2:'2'}`
			
			`{0:5, 1:'1', 2:'2'}`

§8.4 序列

“**序列**”（Sequence）指**线性存储**的**集合**。（线性存储是指元素按照 0, 1, 2, …的顺序存储的。）

§8.4.1 索引

序列的索引均为为**整数**。但与 C 语言不同的是，Python 的整数索引可以为**负数**。负数的含义就是从最后一个元素开始往前数。比如：

```
s = 'Hello'
s[-1] # 最后一个字符，即 o
s[-3] # 倒数第三个字符，即 l
```

§8.4.2 切片

“**切片**”（Slice）指**截取**序列中的多个元素，其语法为“[开始索引:结束索引]”。比如，假设 `s = "abcde"`，那么 `s[1:3]`就表示从索引 1 开始，直到索引 3（不含）的所有元素，即`"bc"`。其中开始索引和结束索引均可以省略，默认值分别为序列的第一个索引和最后一个索引。切片的索引也可以使用负整数。

切片还支持第三个参数——步长，其表示索引每次的变化量。比如`[0:10:2]`就表示索引为 `0, 2, 4, 6, 8` 的元素组合成的切片。步长也可以是负数，但此时开始索引和结束索引需要交换位置。步长的默认值为 1。

对于列表来说，还可以借助切片来修改列表。比如，假设 `a = [1, 2, 3, 4, 5]`，那么 `a[1:3] = [5, 6, 7]` 就会让 a 变成`[1, 5, 6, 7, 4, 5]`。

需要注意的是，切片会创建一个新的序列，因此修改切片不会对原序列产生影响。

§8.4.3 序列总览

以下几种均为序列，但有**区别**：

类型	存储元素	可变性[17]
字符串（String）	字符	不可变
列表（List）	任意	可变
元组（Tuple）	任意	不可变
区间（Range）	整数	不可变

[17]指能否往容器内添加、插入、删除元素

§8.4.4 字符串

字符串的常用方法和操作：

功能	参数	举例	结果
连接	无	`'Hello' + 'World'`	`'HelloWorld'`
重复	无	`'Hello' * 3`	`'HelloHelloHello'`

分割字符串	`# sep: 分隔符（默认为空白符，包含空格、\n、\r、\t 等）`
	`# maxsplit: 最多分割几次（默认为无限次）`
	`split(sep, maxsplit)`
		`'A B C'.split()`
		`'A,B,C'.split(',')`
		`'A,B,C'.split(',', 1)`
			`['A', 'B', 'C']`
			`['A', 'B', 'C']`
			`['A', 'BC']`

大小写转换	`lower() # 全部转小写` `upper() # 全部转大写`	`'ABCdef'.lower()` `'abcDEF'.upper()`	`'abcdef'` `'ABCDEF'`

统计某字符串出现次数	`# sub: 字符串`
	`# start: 开始索引（默认为 0）`
	`# end: 结束索引（默认为最后一个索引）`
	`count(sub)`
	`count(sub, start)`
	`count(sub, start, end)`
		`'abcabcabc'.count('abc')`
		`'abcabcabc'.count('abc', 1)`
		`'abcabcabc'.count('abc', 1, 7)`
			`3`
			`2`
			`1`

移除首尾字符	`# chars: 要移除的字符，默认为空格` `strip(chars)`	`'  abc  '.strip()` `'aabbccbbaa'.strip('ab')`	`'abc'` `'cc'`

（方法风格）格式化	`# args: 用于替换的参数`
	`format(*args)`
		`'Hello, {}!'.format('world')`
		`# 0 表示第一个参数，1 同理`
		`'{0}{1}{0}'.format('x', 'y')`
		`'pi = {:.2f}'.format(3.14159)`
			`'Hello, world!'`
			
			`'xyx'`
			`'pi = 3.14'`

（C 风格）格式化	无	`'Hello, %s!' % 'world'`
		`'%s %s' % ('x', 'y')`
		`'pi = %.2f' % 3.14159`
			`'Hello, world!'`
			`'x y'`
			`'pi = 3.14'`

（字面量风格）格式化[18]	无	`name = 'world'`
		`x, y = 1, 2`
		`pi = 3.14159`
		
		`f'Hello, {name}!'`
		`f'Point({x}, {y})'`
		`f'pi = {pi:.2f}'`
			`'Hello, world!'`
			`'Point(1, 2)'`
			`'pi = 3.14'`

拼接可迭代对象[19]	`# iterable: 可迭代对象` `join(iterable)`	`','.join([1, 2, 3])` `'+'.join([1, 2, 3])`	`'1,2,3'` `'1+2+3'`

查找字符串[20]	`# sub: 字符串`
	`# start: 开始索引（默认为 0）`
	`# end: 结束索引（默认为最后一个索引）`
	`find(sub)`
	`find(sub, start)`
	`find(sub, start, end)`
		`'Hello, world!'.find('world')`
		`'Hello, world!'.find('世界')`
		`'Hello, world!'.find('world', 8)`
			`7`
			`-1`
			`-1`

替换字符串	`# old: 原字符串`
	`# new: 新字符串`
	`# count: 最大替换次数（默认为无限次）`
	`replace(old, new, count)`
		`'Hello'.replace('l', 'x')`
		`'Hello'.replace('l', 'x', 1)`
			`'Hexxo'`
			`'Hexlo'`

[18] 又被称为 f-string，即在字符串字面量前面加上前缀“f”

[19] 以字符串本身作为分隔符

[20] 找到则返回第一次出现的开始索引，找不到返回-1

功能	参数	举例	结果

特性判断	# 是否以 prefix 开头
	startswith(prefix)
	
	# 是否以 suffix 结尾
	endswith(suffix)
	
	# 是否只包含:
	isalpha() # 字母
	isdigit() # 数字
	isalnum() # 字母和数字
	islower() # 小写字母
	isupper() # 大写字母
	isspace() # 空格
		'www.baidu.com'.startswith('www')
		'file.txt'.endswith('.txt')
		'abc'.isalpha()
		'123'.isdigit()
		'123abc'.isalnum()
		'abc'.islower()
		'ABC'.isupper()
		' '.isspace()
			True
			True
			True
			True
			True
			True
			True
			True

§8.4.5 列表

列表创建另见章节：**常用变量总览**

与集合相同或相似的方法：

列表	集合	备注
append(obj)	add(obj)	无
clear()	clear()	无
copy()	copy()	无
remove(value)	remove(value)	无
extend(other)	update(other)	无

与字典相同或相似的方法：

列表	字典	备注
pop(index=-1)	pop(key)	列表的 pop()没有 default 参数

列表的的常用方法和操作：

功能	参数	举例	结果
连接	无	`[1] + [2]`	`[1, 2]`
重复	无	`[1] * 3`	`[1, 1, 1]`
统计元素个数	`# value: 元素` `count(value)`	`a = [1, 1, 3]` `print(a.count(1))`	`2`

获取元素下标	`index(`
	`  value, # 元素`
	`  start=0, # 开始索引`
	`  stop=2**63-1 # 结束索引`
	`) # 仅返回第一个相等元素的下标`
		`a = [2, 1, 3, 4, 1]`
		`print(a.index(1))`
			`1`

插入元素	`insert(`
	`  index, # 插入位置的索引`
	`  obj # 元素`
	`)`
		`a = [1, 2]`
		`a.insert(0, 100)`
		`print(a)`
			`[100, 1, 2]`

翻转列表	`reverse()`	`a = [1, 2]`
		`a.reverse()`
		`print(a)`
			`[2, 1]`

排序	`sort(`
	`  # 是否降序排序`
	`  reverse=False,`
	`  # 排序依据（将元素映射为其他值）`
	`  key=None`
	`)`
		`def f(x): return -x`
		`a = [3, 1, 4, 2, 5]`
		`a.sort(); print(a)`
		`a.sort(reverse=True); print(a)`
		`a.sort(key=f); print(a)`
			`[1, 2, 3, 4, 5]`
			`[5, 4, 3, 2, 1]`
			`[5, 4, 3, 2, 1]`

创建列表的时候，可以借助简化的生成器表达式来指定列表内的元素。比如：

```
a = [i*i for i in [1, 2, 3]]
# 等价于 a = [1, 4, 9]
```

这种列表创建方式被称为“**列表推导式**”（List Comprehension）。

类似地，集合（Set）和字典也可以使用这种生成式写法。

§8.4.6 元组

元组和列表相比，几乎唯一的区别就是元组是**不可变**[21]（Immutable）的。 因此元组仅拥有列表的 `count` 和 `index` 方法以及“连接（`+`）”和“重复（`*`）”的操作。

元组的创建另见章节：**常用变量总览**

§8.4.7 区间

区间有三种创建方式：

```
range(end)
range(start, end)
range(start, end, step)
```

其中 start，end，step 分别表示起始值，结束值和步长。步长指相邻两个数之间的变化量。并且 start 默认为 0，step 默认为 1。

使用方法另见章节：**常用变量总览**

[21]指元组的长度和元素不能发生改变

第九章 内存

§9.1 引用

在 C 语言中，内存分为栈内存和堆内存。其中，通过声明的方式创建的变量使用的是栈内存，而通过 `malloc()` 等方式创建的指针使用的是堆内存。

在 Python 中，一切对象都是在堆内存中创建的，而变量则是对内存中的对象的“**引用**”（Reference）。引用类似于指针，就是让变量指向内存中的对象

比如 `x = "Hello"`这段代码，Python 会先在内存中创建`"Hello"`这个值，然后让 x 引用这个值。

§9.2 实际参数与形式参数

由于 Python 中的变量实际上是对内存中的某个值的引用，因此在函数传递参数的过程中，传递的是**引用**而非实际值。因此下面的代码会输出“0”：

```
def f(x): # x 与 a 指向了同一个值“0”
  x = 1 # 在内存中创建整数“1”，并让 x 指向它，但外界代码中 a 依旧指向原来的“0”

a = 0 # 在内存中创建整数“0”，并让 a 指向它
f(a)
print(a) # 输出“0”
```

而下面的代码会输出“[0]”：

```
def f(x): # x 与 a 指向了内存中的同一个列表“[0]”
  x.append(0) # 让 x 指向的列表进行 append 操作

a = []
f(a)
print(a) # 输出“[0]”
```

§9.3 缓存

如果每次进行 `x = 0` 的操作时，都在内存上创建一个整数，会浪费大量内存。因此 Python 会预先创建好 $[-5, 256]$范围内的整数（即缓存这些整数）。如果有变量被赋值成这些整数中的其中一个，Python 就会直接让它们指向这些缓存的整数之一。可以用下面的代码验证：

```
a = 5
b = 5
print(id(a), id(b)) # 可能输出 2392361140592 2392361140592
print(id(a) == id(b)) # 输出 True
```

类似地，Python 的字符串也会被缓存。

第十章 库与模块

§10.1 概念

Python 中的**库**（Library）是通过“**模块**”（Module）组织的。 一个 Python 文件就可以被当作是一个模块。

§10.2 导入

注：**标识符**（Identifier）指变量、函数、类等

C	Python

`#include "头文件.h"`	`import 模块`
	`import 模块 as 别名`
	
	`import 模块.子模块.子模块`
	`import 模块.子模块.子模块 as 别名`
	
	`from 模块 import 标识符`
	`from 模块 import 标识符 as 别名`
	
	`from 模块 import 标识符 1, 标识符 2, 标识符 3`
	`from 模块 import * # 导入所有标识符`

§10.3 举例

假设有一个 Python 文件“lib.py”：

```python
def add(x, y):
  return x + y

def sub(x, y):
  return x - y
```

就可以在其他 Python 文件中用下面的方式导入它并使用其中的函数：

```python
import lib

lib.add(1, 2)
lib.sub(1, 2)
```

```python
import lib as l

l.add(1, 2)
l.sub(1, 2)
```

```python
from lib import add

add(1, 2)
```

```python
from lib import add as f

f(1, 2)
```

```python
from lib import add, sub

add(1, 2)
sub(1, 2)
```

§10.4 常用内置函数

函数	功能	举例	结果

`input(`
`  prompt=None`
`)`
	从终端读入字符串	`input()`
		
		`input('请输入数字：')`
			`> hello`
			`'hello'`
			`> 请输入数字：123`
			`'123'`

`print(`
`  # 需要输出的值`
`  *args,`
`  # 值之间的分隔符`
`  sep=' ',`
`  # 结尾字符`
`  end='\n'`
`)`
	向终端输出内容	`print(1)`
		`print(1, 2, 3)`
		`print(1, 2, 3, sep=',')`
		`print(1, 2, 3, end='$')`
			`1`
			`1 2 3`
			`1,2,3`
			`1 2 3$`

`abs(x)` `round(x)`	求绝对值 四舍五入[22]	`abs(-1)` `round(3.14)`	`1` `3`
`ord(x)` `chr(x)`	字符转 ASCII 值 ASCII 值转字符	`ord('a')` `chr(97)`	`97` `'a'`
`eval(x)` `exec(x)`	计算表达式 执行语句	`eval('1 + 2')` `exec('print(1 + 2)')`	`3` `3`
`id(x)`	获取 x 的地址	`x = 1` `id(x)`	`140725698102184`

`isinstance(x, t)`	判断 x 的类型
	是否为 t
		`def f(x):`
		`  print(isinstance(x,int))`
		`f(1)`
		`f('1')`
			`True`
			`False`

`max(*args)` `min(*args)`	获取最大/小值	`max(4, 5)` `min(4, 5, 6)`	`5` `4`
`type(x)`	获取 x 的类型	`type(1.23)`	`float`

`open(`
`  # 文件路径`
`  filename,`
`  # 打开模式`
`  mode='r',`
`  # 编码`
`  encoding=None`
`)`
	打开文件
	（另见章节：**文件操作**）
		`open(`
		`  '1.txt',`
		`  'r',`
		`  encoding='utf-8'`
		`)`
			无

[22]严格来说不是四舍五入，但是被广泛认作是四舍五入

与可迭代对象相关的内置函数：

函数	功能	举例	结果
`all(iterable)` `any(iterable)`	是否所有元素都为 True[23] 是否有元素为 True[23]	`max([1, 2, 3])` `min([1, 2, 3])`	`3` `1`
`max(iterable)` `min(iterable)`	获取最大/小值	`max([1, 2, 3])` `min([1, 2, 3])`	`3` `1`
`sum(iterable)`	对所有元素**求和**	`sum([1, 2, 3])`	`6`

`filter(`
`  # 过滤函数`
`  function,`
`  # 可迭代对象`
`  iterable`
`)`
	过滤元素[24]	`def f(x): return x % 2 == 0`
		`list(filter(f, range(7)))`
			`[0, 2, 4, 6]`

`map(`
`  # 映射函数`
`  function,`
`  # 可迭代对象`
`  iterable`
`)`
	把每一个元素**映射**成另一个值[24]	`def f(x): return x * x`
		`list(map(f, range(4)))`
			`[0, 1, 4, 9]`

`reversed(`
`  # 可迭代对象`
`  iterable`
`)`
	返回**翻转**后的可迭代对象[24]	`list(reversed([1, 2, 3]))`	`[3, 2, 1]`

`# 参数解释同列表的 sort 方法`
`sorted(`
`  iterable,`
`  reverse=False,`
`  key=None`
`)`
	以列表形式得到对所有元素**排序**后的可迭代对象	`sorted([3,1,5,2,4])`
		`sorted([3,1,5,2,4],reverse=True)`
		
		`def f(x): return -x`
		`sorted([3, 1, 5, 2, 4], key=f)`
			`[1, 2, 3, 4, 5]`
			`[5, 4, 3, 2, 1]`
			
			`[5, 4, 3, 2, 1]`

[23]会先通过 `bool()`将元素转换为布尔值后再判断
[24]返回的是生成器。为了便于看到函数的执行结果，举例中将生成器转换为了列表

§10.5 文件操作

§10.5.1 打开和关闭文件

操作	C	Python
打开文件	`FILE* f = fopen("1.txt", "r");` `FILE* f = fopen("1.txt", "w");`	`f = open('1.txt', 'r', encoding='utf-8')` `f = open('1.txt', 'w', encoding='utf-8')`
关闭文件	`fclose(f);`	`f.close()`

§10.5.2 读取文件

方法	功能	举例	结果
`read()`	读取全部内容	`f.read()`	`'123\n456\n789\n'`
`readline()`	读取一行内容	`f.readline()`	`'123\n'`
`readlines()`	按行读取全部内容	`f.readlines()`	`['123\n', '456\n', '789\n']`

§10.5.3 文件写入

方法	功能	举例	结果
`write(s)`	写入字符串 s	`f.write('Hello')`	`Hello`

`writelines(lines)`	按行写入字符串	`f.writelines([`
		`  '123',`
		`  '456',`
		`  '789'`
		`])`
			`123`
			`456`
			`789`

§10.6 常用标准库

注：

· **标准库**（Standard Library），指 Python 自带的库

名称	作用	文档
os	操作系统接口	https://docs.python.org/zh-cn/3.11/library/os.html
sys	解释器接口	https://docs.python.org/zh-cn/3.11/library/sys.html
json	处理 JSON 数据	https://docs.python.org/zh-cn/3.11/library/json.html
datetime	处理日期和时间	https://docs.python.org/zh-cn/3.11/library/datetime.html
time	时间相关操作	https://docs.python.org/zh-cn/3.11/library/time.html
math	数学运算	https://docs.python.org/zh-cn/3.11/library/math.html
random	随机数	https://docs.python.org/zh-cn/3.11/library/random.html
re	正则表达式	https://docs.python.org/zh-cn/3.11/library/re.html
itertools	实用的枚举函数	https://docs.python.org/zh-cn/3.11/library/itertools.html
functools	实用的函数工具	https://docs.python.org/zh-cn/3.11/library/functools.html
collections	实用的容器数据类型	https://docs.python.org/zh-cn/3.11/library/collections.html

§10.7 第三方库

要想**安装**第三方库，可以在终端中运行以下命令：

```
pip install 库名
```

比如，有一个专门用于发起网络请求的库：requests，要想安装它就可以运行：

```
pip install requests
```

然后就可以在代码中**使用**这个库：

```
import requests # 导入
r = requests.get('https://baidu.com') # 请求 baidu.com
print(r.text) # 输出网页内容
```

由于 Python 的第三方库的下载网站位于国外，访问速度会受到限制。可以在终端中运行下面的命令，将下载网站改为国内网站：

```
pip config set global.index-url https://mirrors.tuna.tsinghua.edu.cn/pypi/web/simple
```

第十一章 异常与 with 关键字

§11.1 异常的概念

注：“异常”仅存在于 Python 中，不存在于 C 语言中。

异常（Exception）指代码执行过程中出现的错误。这种错误往往会导致程序直接终止运行。比如，运行下面的代码：

```
def f(x, y):
  return x / y

a = f(1, 0)
print(a)
```

就会出现下面的错误信息：

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    a = f(1, 0)
  File "main.py", line 2, in f
    return x / y
ZeroDivisionError: division by zero
```

错误信息包含了以下几点：

1. “Traceback”指**代码的调用栈**，用于追溯代码发生异常的位置。此处就是执行 `a = f(1, 0)`之后，执行 `x / y` 的过程中出现了异常。
2. “ZeroDivisionError:”指**异常的类型**。此处的意思是零作除数的异常。
3. “division by zero”指**错误消息**。错误消息可以帮助开发者更快了解异常的错误原因。

异常会直接终止程序，打断后续的代码的执行，因此后面的 `print(a)`不会被执行。

§11.2 抛出异常

有的时候，某些不符合逻辑的地方我们也希望抛出异常，就可以使用 `raise` 关键字抛出自己的异常。比如：

```
def add(x, y):
  if type(x) != type(y):
    raise TypeError("变量 x 的类型与 y 不同")

  return x + y

add('1', 1)
```

很明显，如果 x 和 y 的类型不同，就无法进行 `x + y` 的操作。因此我们在加法操作前加了一个 `if` 判断，并添加了 `raise ...`语句。执行上面的代码会出现：

```
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    add('1', 1)
    ~~~^^^^^^^^
  File "main.py", line 3, in add
    raise TypeError("变量 x 的类型与 y 不同")
TypeError: 变量 x 的类型与 y 不同
```

§ 11.3 捕获异常

有的时候，我们不希望程序遇到异常就终止执行；还有的时候我们希望由我们自己来处理异常，而不是让 Python 自行处理。这时就需要在代码中主动捕获异常。

语法如下：

```
try:
  代码块 # 可能出现异常的代码
except 异常类型 1 as 变量名: # 捕获异常类型 1，执行代码块
  代码块
except 异常类型 2 as 变量名: # 捕获异常类型 1，执行代码块
  代码块
except 异常类型 3 | 异常类型 4 as 变量名: # 捕获异常类型 3 或 4，执行代码块
  代码块
finally: # 执行完 try 或 except 的代码块后，执行该代码块
  代码块
```

其中“异常类型”可以省略，表示捕获任何异常；“as 变量名”也可以省略。

举例：

```
def f(x, y):
  return x / y

try:
    f('1', 1)
    f(1, 0)
except ZeroDivisionError:
    print('出现了除数为 0 的错误')
except TypeError as e:
    print('出现了类型错误:', e)
except RuntimeError | SyntaxError:
    print('无法运行代码')
finally:
    print('Done')
```

运行结果：

```
出现了类型错误: unsupported operand type(s) for /: 'str' and 'int'
Done
```

§ 11.4 with 关键字

我们知道，成功打开文件之后，代码的最后必须关闭文件。但是假如这之间出现了异常，就会导致程序直接终止，无法执行关闭文件的代码。因此我们需要用 finally 确保不论有没有异常都关闭文件：

```
f = open('1.txt')

try:
  print(f.read())
finally:
  f.close()
```

从中我们可以发现，文件对象有一个特点：必须同时有创建对象（打开文件）和销毁对象（关闭文件）的操作，并且不应该被异常打断销毁操作。这时就可以用 with 简化代码逻辑：

```
with open('1.txt') as f:
  print(f.read())
```

with 也被称作“上下文管理器”（Context Manager）。 with 可以确保变量能够被销毁。

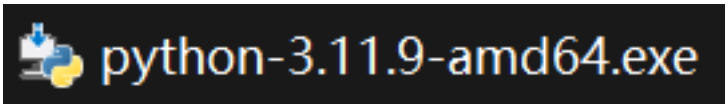
第十二章 附录

§12.1 安装 Python

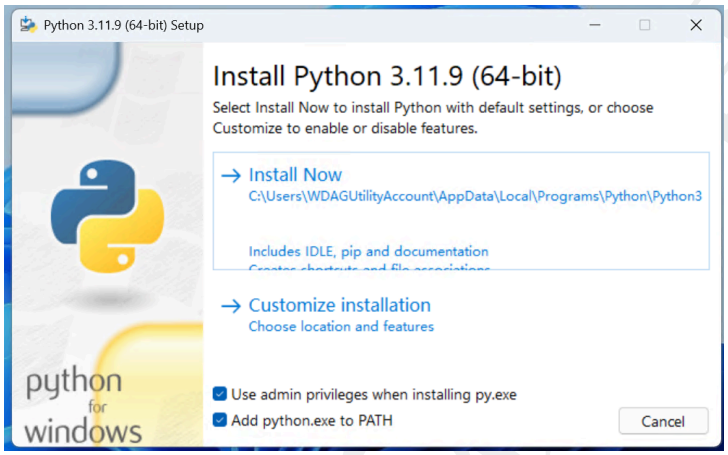
1. 下载 Python 安装文件

https://mirrors.aliyun.com/python-release/windows/python-3.11.9-amd64.exe

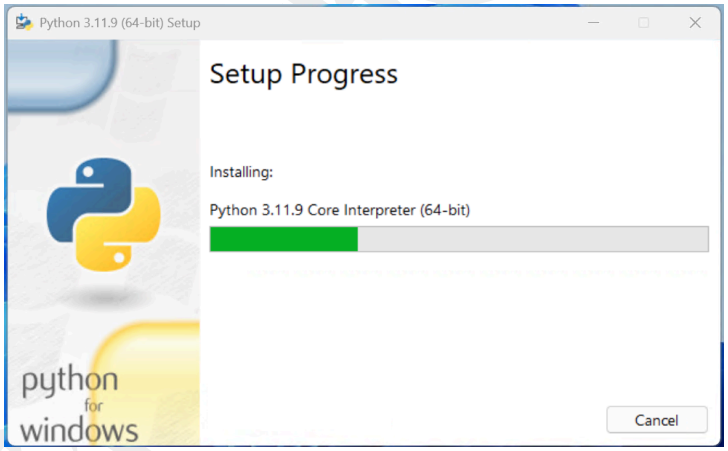
2. 打开下载好的文件



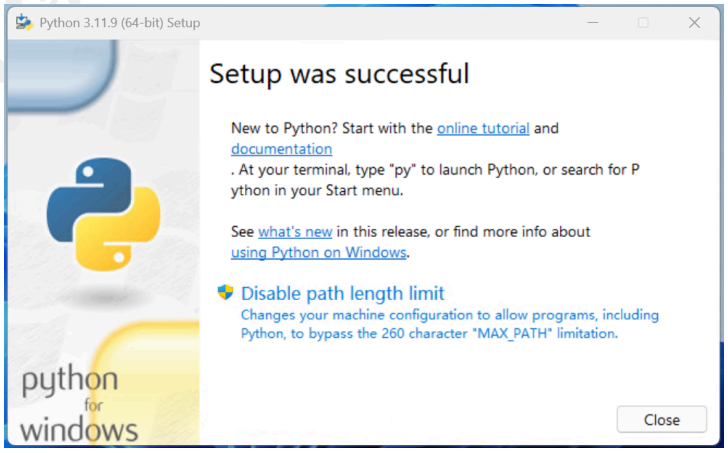
3. 勾选下方的“Add python.exe to PATH”，点击“Install Now”



4. 等待安装完成



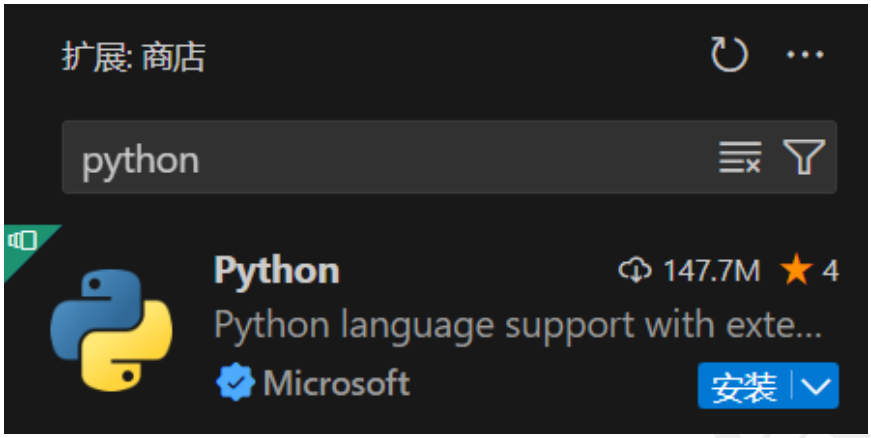
5. 点击“Close”完成安装（可选：点击“ Disable path length limit”[25]）



[25]Windows 操作系统默认限制了文件路径的长度不能超过 260 个字符，该选项可以禁用这个限制，推荐点击

§12.2 在 VSCode 中使用 Python

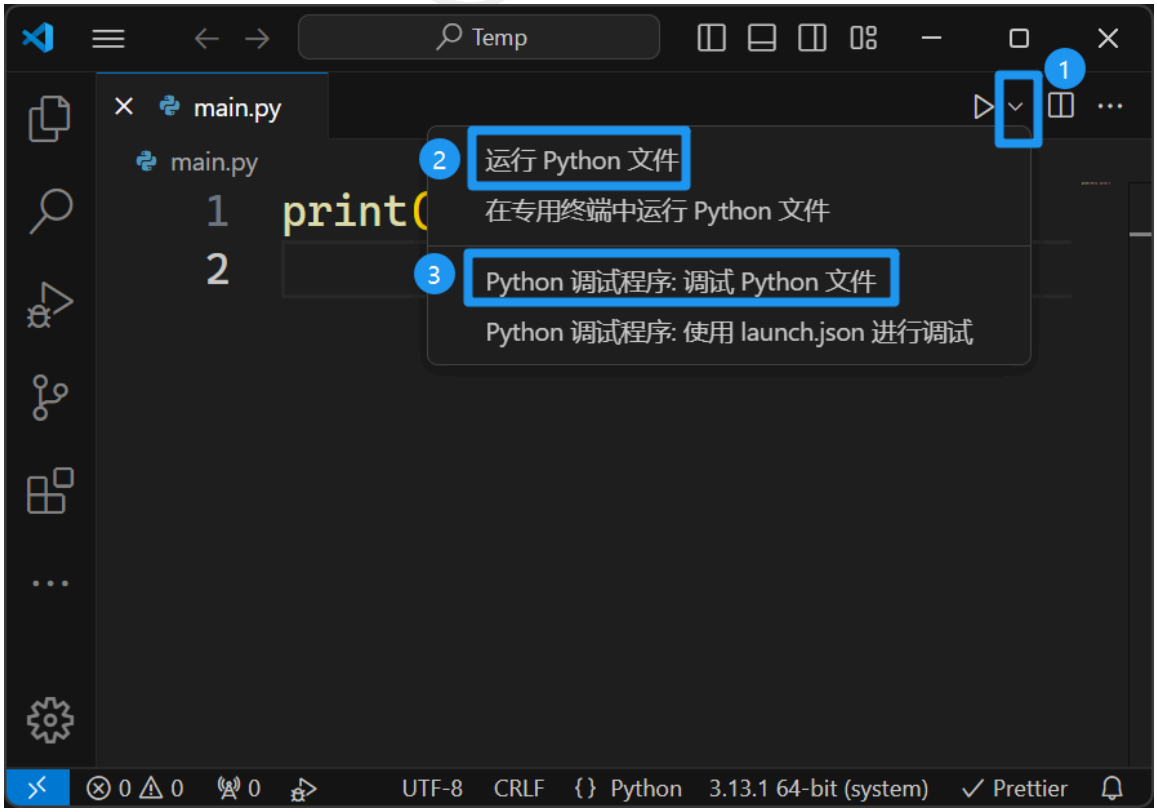
1. 在 VSCode 的“扩展”（Extensions）面板中搜索“Python”，点击“安装”（Install）



2. 打开一个空文件夹，新建 `main.py` 文件

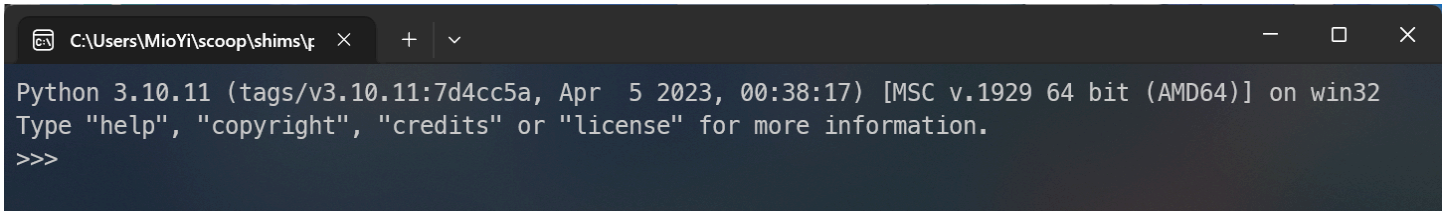
3. 编辑 `main.py`

4. 点击图片中的“①”，若要**运行**请点击“②”，若要**调试**请点击“③”

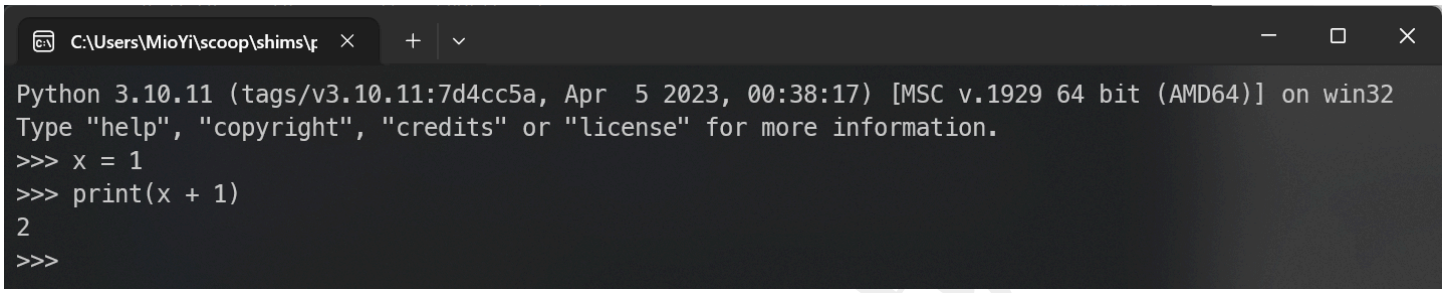


§12.3 REPL

在终端中直接执行 `python` 命令，或是直接运行 `python.exe`，会出现下图所示的界面：



这个就是 Python 的 REPL 模式。“REPL”指“Read-Eval-Print Loop”，即“读取-求值-输出 循环”。它是一种交互式的编程环境，用户每输入一行代码就会立刻执行并显示结果。比如：



REPL 的好处有很多：

1. 方便用户立刻看到代码的作用。
2. 无需创建 Python 文件就可以直接执行 Python 代码。
3. 方便进行一些简单的功能实现和测试。
4. 可以把它当作计算器使用：