Python 编程实训教程

V3.0

目录

1. Python 介绍

1.1 起源

Python 是由荷兰计算机科学家吉多·范罗苏姆（Guido van Rossum）于 1989 年末开发、1991 年首次发布的高级编程语言，凭借简洁易读的语法、强大的生态与广泛的应用，成为全球最流行的开源编程语言之一。

发展历程与版本：

- 诞生与早期：1989 年末，吉多为解决 “系统管理需要更简洁的脚本语言” 问题，以喜剧《巨蟒剧团之飞翔的马戏团》命名，开始研发 Python；1991 年发布首个公开版本（0.9 版）。
- 重要里程碑：

✓ 2000 年：Python 2.0 发布，引入垃圾回收优化、Unicode 支持、列表推导式等关键特性。

✓ 2008 年：Python 3.0 发布，对语言进行 “现代化重构”（如 print 变为函数、强化 Unicode 支持），但不完全兼容 Python 2。

✓ 2010 年：Python 2.7 发布（作为 2.x 系列最后一个版本，支持向 3.x 过渡）；

✓ 2020 年，Python 官方停止对 2.x 的维护。

✓ 近年更新：3.x 系列持续迭代（如 3.11 优化性能、3.13 引入更多语法糖与性能改进），保持语言活力。

1.2 应用领域

Python 的灵活性与生态优势，使其渗透到众多领域：

Web 开发：借助 Django（大而全，适合复杂项目）、Flask（轻量灵活）等框架，快速搭建后端服务、API 接口。

数据科学与人工智能：

数据分析：用 Pandas 清洗 / 分析数据，Matplotlib/Seaborn 可视化结果。

机器学习 / 深度学习：Scikit-learn 实现传统算法，TensorFlow/PyTorch 支持神经网络、大模型开发。

自动化与脚本：

办公自动化：批量处理 Excel/Word、自动发送邮件。

测试自动化：Selenium（UI 测试）、pytest（接口 / 单元测试）等框架广泛用于软件测试。

网络爬虫：Requests+BeautifulSoup/Scrapy 批量抓取网页数据。

其他领域：金融科技（算法交易、风控建模）、游戏开发（Pygame）、教育（编程入门首选）、物联网等。

1.3 特点

- **语法简洁，易读易维护**

用缩进替代大括号定义代码块，语法接近自然语言，降低了学习门槛（如 “if 条件: 执行逻辑” 的结构直观易懂）。

- **跨平台与可移植**

能在 Windows、macOS、Linux 等主流操作系统上运行，字节码 “一次编写，多平台执行”，无需针对不同系统重复编译。

- **多编程范式支持**

同时支持面向对象编程（通过类、对象封装逻辑）和函数式编程（如高阶函数、匿名函数、装饰器），灵活适配不同开发场景。

- **丰富的库与生态**

标准库：覆盖文件操作、网络通信、数据处理等常见任务（如 os 管理系统、requests 处理 HTTP 请求），无需额外安装第三方库即可完成大量工作。

第三方库：生态极其繁荣，涵盖各领域需求 —— 数据分析有 Pandas/NumPy，机器学习有 TensorFlow/PyTorch，Web 开发有 Django/Flask 等，开发者可 “站在巨人肩膀上” 快速实现功能。

- **解释型与动态类型**

解释型：代码逐行解释执行，开发过程更灵活（修改后无需重新编译即可运行）。

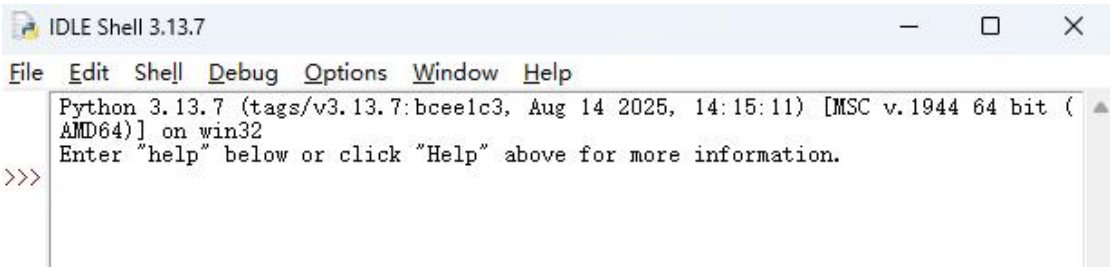
动态类型：变量类型在运行时自动确定，无需显式声明（如 x = 10 是整数，x = "hello" 又变为字符串），提升编码效率。

2. Python 基础

2.1 开始第一个 Python 程序

编写 python，使用自带的 IDLE 工具编写。

路径：开始菜单>Python 3.x>IDLE(Python 3.x 64-bit)



现在可以开始第一个 Python 程序了，

```
>>> print("Hello world!!")
Hello world!!
```

2.2 输入输出

2.2.1 输出（print）

Python 的打印，用 print()在括号中加上字符串，就可以向屏幕上输出指定的文字。除了上面的第一个程序打印的字符串，print()也可以打印整数，或者计算结果：

```
>>> print(100+200)
300
```

print()函数也可以接受多个字符串，用逗号“,”隔开，就可以连成一串输出，遇到逗号就会打印一个空格：

```
>>>print("100+200=",100+200)
100+200= 300
```

注意：python2 中 print 为一个语句，如 print "hello word"；python3 中的 print 为一个函数，函数的调用必须带括号。

2.2.2 输入(input)

如果要让用户从电脑输入一些字符怎么办？Python 提供了一个 input()函数，可以让用户输入字符串，并存放到一个变量里。当你按下回车的时候，就在等待你的输入，你可以输入任意字符串，然后回车：

```
>>>name = input("Enter your name: ")
Enter your name: Harry    #键盘输入
>>>print(name)
Harry
```

注意：任意输入的内容都会被当作字符串，包括数字。

2.3 变量和表达式

2.3.1 变量赋值

变量的概念基本上和初中代数的方程变量是一致的，只是在计算机程序中，变量不仅可以是数字，还可以是任意数据类型。变量在程序中就是用一个变量名表示了，变量名必须是大小写英文、数字和_的组合，且不能用数字开头，比如下面的，

```
>>>a = 100                    #变量为 int 型
>>>b = "test"                  #变量为字符型
>>>x,y,z = 1,2, "string"     #多元赋值
>>>x=y=z=1                    #多重赋值
>>>print(x,y,z)
1 1 1
```

注意：变量赋值的=，不是数学计算中的等于。比如当你输入 a=100，那么其实 Python 解释器干了两件事情：

1) 在内存中创建了一个整数值 100；

2) 在内存中创建了一个名为 a 的变量，并且将 a 指向了内存中的整数值 100。

前面讲过，Python 的变量不需要申明，通过上面的方式给变量赋值以后，变量就拥有了类型，类型还可以转换，比如申明了一个字符串变量 a="100"，此时你用 a+10，会报错。那么你有两种方式处理 int(a)+10 和 a+str(10)，

```
>>> a = "100"
>>> int(a)+10
110
>>> a+str(10)
 '10010'
```

常见的类型转换：

转换方法	描述
str(a)	将 a 变量转换成字符串类型
int(a)	将 a 变量转换成整形
float(a)	将 a 变量转换成浮点型

2.3.2 表达式

表达式是由值，变量和运算符组成，如：

```
>>> a+3
```

常见运算符：（下表实例中 a=10；b=21）

运算符	描述	实例

+	加 - 两个对象相加	a + b 输出结果 31
-	减 -一个数减去另一个数	a - b 输出结果 -11
*	乘 - 两个数相乘	a * b 输出结果 210
/	除 - x 除以 y	b / a 输出结果 2.1
%	取余 - 返回除法的余数	b % a 输出结果 1
**	幂 - 返回 x 的 y 次幂	a**b 为 10 的 21 次方
//	取整除 - 返回商的整数部分	9//2 输出结果 4 , 9.0//2.0 输出结果 4.0

2.3.3 Python - 引号与注释

引号：

python 不区分单引号和双引号，可以嵌套使用，但不能交叉。谁在外谁在内都不重要，重要的是不能交叉。

```
>>>print('there is a "pig"')
there is a “pig”
```

注释：

注释分单行和多行注释，单行注释为“#”，多行注释为三组单引号或者双引号。

转义：

python 用反斜杠(\)转义字符，如\\，\"。另外\还有些其他用法：\(在行尾时)表示续航符，\n 表示换行符等，

```
>>> print("我有换行符\n后面的在第二行")          #加上转义符试试
我有换行符
后面的在第二行
>>> print("一行写不完\
    接着写 ")
一行写不完接着写
```

2.4 基本数据类型

2.4.1 数值类型（Number）

用于表示数值，包括整数、浮点数和复数，支持常见的数学运算（加、减、乘、除等）。

子类型	定义	关键特点	示例
`int`	整数（正负整	Python 3 中无大小限制（可表示任	`10`、`-5`、`0`、

子类型	定义	关键特点	示例
	数、0）	意大的整数），无需声明“长整数”（Python 2 的 long 已合并）	999999999999999
float	浮点数（带小数点的数）	基于二进制存储，存在精度误差（避免直接用==比较浮点数），支持科学计数法	3.14、-0.5、1.2e3（即 1200）
complex	复数（实部+虚部，虚部用 j 表示）	实部和虚部均为浮点数，主要用于科学计算（如信号处理、线性代数）	3+4j、-2.5j、1.0-2.0j

举例代码：

```
# int 运算
a = 10
b = 3
print(a + b)    # 13（加法）
print(a // b)    # 3（整除）
print(a % b)    # 1（取余）

# float 注意精度
c = 0.1
d = 0.2
print(c + d)    # 0.30000000000000004（而非 0.3）
print(round(c + d, 1))    # 0.3（用 round()解决精度问题）

# complex 运算
e = 3 + 4j
print(e.real)    # 3.0（获取实部）
print(e.imag)    # 4.0（获取虚部）
print(e * 2)    # (6+8j)（复数乘法）
```

2.4.2 布尔类型（bool）

用于表示“真”或“假”，是 int 类型的子类，一个布尔值只有 True、False 两种值，要么是 True（真，对应整数 1），要么是 False（假，对应整数 0），在 Python 中，可以直接用 True、False 表示布尔值（请注意大小写）

应用场景：

1) 条件判断（if、while 语句的条件）。
2) 比较运算的结果（如 3 > 2 返回 True，5 == 6 返回 False）。
3) 逻辑运算（and、or、not）。

示例代码：

```
# 比较运算结果
print(5 > 3)      # True
print('a' == 'A')# False

# 逻辑运算
x = True
y = False
print(x and y)   # False（与：均真才真）
print(x or y)    # True（或：一真即真）
print(not x)      # False（非：取反）

# 布尔值与整数的关系
print(True + 1)   # 2（True 视为 1）
print(False * 5) # 0（False 视为 0）
```

2.4.3 空值 NoneType

空值是 Python 里一个特殊的值，用 None 表示。None 表示“空值”或“无结果”，不能理解为 0，因为 0 是有意义的，而 None 是一个特殊的空值。

应用场景：

(1) 初始化变量（暂时没有具体值时）。
(2) 函数无明确返回值时（默认返回 None）。
(3) 表示“不存在”（如字典中键不存在时用 get()返回 None）。

注意：None 不等于空字符串（""）、0（0）或空列表（[]），需用 is 判断（而非==）。

示例代码：

```
# 初始化变量
name = None
print(name is None)   # True
```

查看数据类型：

type()，Python 中可以用 type()函数来查看数据类型。

```
>>>a = True
>>>type(a)
<class ‘bool’ >
```

2.5 字符串类型（str）

字符串是 Python 中最常用的数据类型。我们可以使用引号(’或")来创建字符串。前面已经接触到字符串变量和打印了。这一节将正式介绍一下字符串。

首先创建一个字符串变量，变量名为 name，值为 Harry 字符串。例如：

```
>>>name = “Harry”
```

2.5.1 格式化字符串

打印带变量的字符串，试一试下面的操作！

```
>>>x, y =1, "A"
>>>print("%d is a number, %s is a string" % (x,y))
1 is a number, A is a string
```

字符串的格式化是通过%来实现的。其中%d（十进制整数）打印数字，%s（字符串）只能打印字符串，如果无法确定种类则用%r。

```
>>>lst = [1,2,3]
>>>print("这个列表内容是 %r" % lst)
这个列表内容是 [1, 2, 3]
```

2.5.2 字符串操作

字符串也可以有加和乘，只是与数字运算不同。根据下表中的操作符，（实例中的 a=”Hello”；b=” Python” ）

操作符	描述	实例
+	字符串连接	a + b 输出结果：HelloPython
*	重复输出字符串	a*2 输出结果：HelloHello
[]	通过索引获取字符串中字符	a[1] 输出结果 e
[:]	截取字符串中的一部分	a[1:4] 输出结果 ell
in	成员运算符 - 如果字符串中包含给定的字符返回 True	"H" in a 输出结果 True

not in	成员运算符 - 如果字符串中不包含给定的字符返回 True	"M" not in a 输出结果 True
r/R	原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母"r"（可以大小写）以外，与普通字符串有着几乎完全相同的语法。	print(r"中间有换行符\n前面带r就不用转义")

2.5.3 字符串常见方法/函数

除了通过常见的运算符对字符串操作，Python 还提供了很多函数和方法，见下表（每个都加粗的都是重点，要重点练习！）

方法/函数	说明
len(str)	**返回字符串长度**
str.count(sub, start= 0,end=len(str))	返回 sub 在 str 里面出现的次数,如果 start 或者 end 指定则返回指定范围内 sub 出现的次数
str.startswith(prefix, start=0,end=len(str))	**检查字符串是否是以 prefix 开头，是则返回 True，否则返回 False。如果 start 和 end 指定值，则在指定范围内检查。**
str.endswith(suffix, start=0, end=len(str))	**检查字符串是否以 suffix 结束，如果 start 或者 end 指定则检查指定的范围内是否以 suffix 结束，如果是，返回 True,否则返回 False**
str.find(sub, start=0 end=len(str))	**检测 sub 是否包含在字符串中 str 中，如果 start 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1**
str.index(sub, start=0, end=len(str))	跟 find()方法一样,只不过如果 sub 不在 str 中会报一个异常
str.rfind(sub, start=0,end=len(str))	类似于 find()函数，不过是从右边开始查找.
str.rindex(sub, start=0, end=len(str))	类似于 index()，不过是从右边开始.
str.lstrip()	默认删除字符串左边的空格。
str.rstrip()	默认删除字符串右边的空格.
str.strip()	**在字符串上执行 lstrip()和 rstrip()，就是删除左右空格。**
str.replace(old, new, max=str.count(old))	**将字符串中的 old 替换成 new,如果 max 指定，则替换不超过 max 次。**
str.join(seq)	**以 str 作为分隔符，将 seq 中所有的元素合并为一个新的字符串**
str.split(sub="", num=str.count(obj))	**以 sub 为分隔符截取字符串，如果 num 有指定值，则截**

	取 num 次
str.splitlines(keepends=False)	按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。

2.6 列表 list

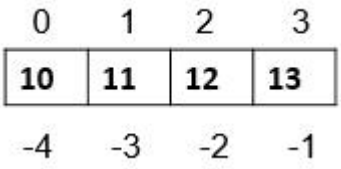
Python 内置的一种数据类型是列表：list。list 是一种有序的集合，可以随时添加和删除其中的元素。

以下就是 list 的赋值方式，

```
>>> A=[10,11,12,13]                    #元素为整数
>>> B=["red","blue","green"]      #元素为字符串
>>> C=[]                                    #定义空列表
>>> E=A+B                                  #两个列表相加
>>>L=[1,2,3,[4,5,6]]                    #二维数组
```

2.6.1 list 的基本操作

在 python 中，list 可以通过下标取值、修改元素等操作。其下标有正向和负向，有正向的下标从 0 开始，0 代表第一个；负向从-1 开始，-1 代表倒数第一个。



试一试：

```
>>> b=A[1]     #取 A 列表的第 2 个元素
>>> A[2]=5     #修改列表第三个元素的值
>>> L[3][1]      #多维数组取值
```

2.6.2 切片

截取序列（还记得字符串的操作么？）的一部分作为新的序列返回。格式：list[x:y] 返回索引从 x 到 y-1 的元素。（实例中 A=[10,11,12,13]）

```
>>>A[0:2]            #取索引为 0，1 的元素
>>>A[-3:-1]          #取索引倒数第 3 个到倒数第 2 个的元素
>>>A[:2]             #取索引为 0，1 的元素
```

```
>>>A[-3:]          #取索引倒数第 3 个到倒数第 1 个的元素
>>>A[1:-1]         #取索引 2 个元素到倒数第 2 个元素
>>>A[2:-3]         #空列表
```

2.6.3 列表常见的函数与方法

List 函数：

函数	说明
len(list)	**列表元素个数**
max(list)/min(list)	返回列表元素最大值/最小值
list(tuple)	将元组转换为列表

List 方法：

方法	说明
list.append(obj)	**在列表末尾添加新的对象**
list.count(obj)	统计某个元素在列表中出现的次数
list.extend(seq)	在列表末尾一次性追加另一个序列（seq 是一个序列）
list.index(obj)	**从列表中找出某个值第一个匹配项的索引位置**
list.insert(index, obj)	将对象插入列表
list.pop(index)	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
list.remove(obj)	移除列表中某个值的第一个匹配项
list.reverse()	反向列表中元素
list.sort()	对原列表进行排序
list.clear()	清空列表
list.copy()	复制列表

2.7 元组 tuple

另一种有序列表叫元组：tuple。tuple 和 list 非常类似，但是 tuple 一旦初始化就不能修改。也用 tuple 试一试 list 中修改元素的方法。

```
>>> f = (2,3,4,5)                    #整数元组
>>> g = ()                             #空元组
>>> h = (2, [3,4], (10,11,12))        #多维列表
>>> i = (1,)                         #i=(1)与 i=1 相同，只有一个元素的元组加逗号
```

操作：

```
>>> x = f[1]       # 将 f[1]的元素值赋值 x = 3
>>> y = f[1:3]         # 获得索引为 1,2 的元素
```

```
>>> z = h[1][1]        # 二维数组看待 z = 4
```

特色：

a) 与 list 类似，最大的不同元组是一种只读且不可变更的数据结构

b) 不可取代元组中的任意一个元素，因为它是只读不可变更

c) list 与 tuple 可以互相转换，tuple(list)、 list(tuple)

2.8 字典 Dict

字典是另一种可变容器模型，且可存储任意类型对象。字典的每个键值(key=>value)对用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号({})中。

```
>>> a = {}              # 定义空字典
>>> b = { "x": 3, "y": 4 }
>>> c = { "uid": 105,
        "login": "beazley",
        "name" : "David Beazley"
      }
```

字典中，键必须是唯一的，但值则不必。值可以取任何数据类型，但键必须是不可变数据类型，如字符串，数字或元组。

2.8.1 常用操作

字典可以根据键取值，也可以根据键修改、新增键值对。

```
>>>u = c["uid"]                    # 根据索引读取元素值
>>>c["name"] = "Harry"      # 根据键修改值
>>>c["new"] = "新值"             #增加新的值对
>>>len(c)                          #字典键值对个数
```

2.8.2 常用方法

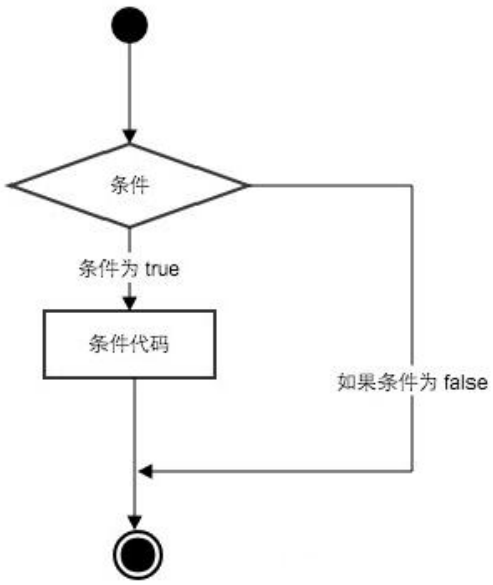
方法	说明
key in dict	**如果键在字典 dict 里返回 True，否则返回 False**
dict.items()	**以列表返回可遍历的(键, 值) 元组**
dict.keys()	**以列表返回一个字典所有的键**
dict.values()	**以列表返回字典中的所有值**
dict.update(dict2)	把字典 dict2 的键/值对添加到 dict 里

dict.get(key,default=None)	返回指定键的值，如果键不在字典中返回 default 值
dict.setdefault(key, default=None)	和 get()类似，但如果键不存在于字典中，将会添加键并将值设为 default 的值
dict.pop(key[,default])	删除键 key 对应的值,返回被删除的值。key 不存在返回 default 值。如果 key 不存在，default 未传值，则报错。
dict.copy()	返回一个字典的浅复制
dict.clear()	删除字典内所有元素

2.9 条件语句

2.9.1 if 语句

条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。可以通过下图来简单了解条件语句的执行过程:



if…elif…else 语句

```
if  y > 0 :
    print("y > 0")
elif  y == 0 :
    print("y == 0")
else :
    print("y < 0 ")
```

注意缩进!

2.9.2 布尔表达式

布尔表达式的值为真和假。真为 True，假为 False

```
a = 2 ;  b = 4 ; c = 6
if b >= a and b <= c :
    print("b is between a and c ")
if not (b < a or b > c ):
    print("b is still between a and c ")
```

逻辑运算符：

操作符	描述	操作符	描述
<	小于	and	与，同时为 True
<=	小于或等于	or	或，其中一个为 True
>	大于	not	非，为 False 时
>=	大于或等于	in	包含
==	等于，比较对象是否相等	not in	不包含
!=	不等于		

2.10 循环语句

2.10.1 While 语句（循环直到条件不满足）

while 语句，当条件为 True 时会一直循环，直到条件为 False 为止，结束循环。Python 中

```
while 判断条件：
    语句
else：
    语句
```

while 语句的一般形式（else 子句并不是必须的）：

试一试例子：

```
i = 5
While i > 0:
    print("hello world ! %s" %i )
    i = i-1
else：
```

```
    print("循环结束！")
```

2.10.2 for 语句(遍历序列的元素)

for 语句主要用来遍历序列（如 list、tuple 等），当我们看到连续的数字，或者序列需要遍历操作的时候，首先应该想到用 for 语句。

```
for i in [3, 4, 10, 25]:
    print(i)
```

前面讲字符串的时候提到，字符串也有 list 的特性，因此字符串输出也可以用 for，

```
for c in "Hello World":
    print(c)
```

range()函数：

在用 for 语句的很多时候，我们想多次循环 ，比如要完成 1+2+3…+9999 的时候，不可能预先去定义这样一个 9999 个元素的 list，这时候就需要用到 range()函数。range(start, end，step)函数，用来生成给一个整数的序列。其中 range()函数有三个参数分别是 start，end，step 分别表示起始数值，结束数值，加减值。比如 range(1，10，2)，表示从 1 开始，到 9 结束，每次循环加 2。

```
for i in range(1, 10, 2):
    print(i)
for i in range(10):
    print(i)
for i in range(3, 10):
    print(i)
for i in range(0, -10, -1):
    print(i)
```

2.10.3 控制流终止

continue：

continue 语句被用来告诉 Python 跳过当前循环块中的剩余语句,然后继续进行下一轮循环。

```
for i in range(1,5):
    print(i)
    if i == 2 :
       print("我准备跳了！")
       continue
```

break ：

break 语句是用来终止循环语句的，如果循环中遇到 break 则结束循环。

```
for i in range(1,5):
```

```
    if i == 2 :
        break
    print(i)
```

注意：不要滥用 break 和 continue 语句。break 和 continue 会造成代码执行逻辑分叉过多，容易出错。

3. 函数

3.1 函数的申明

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。函数能提高应用的模块性，和代码的重复利用率。你已经知道 python 提供了许多内建函数，比如 print()。但你也可以自己创建函数，这被叫做用户自定义函数。

函数的定义需要用到 def 语句：

```
def 函数名(参数 1,参数 2…):
    语句块
```

试一试，创建一个函数并调用。

```
def add(p1 ,p2):                                #声明函数
    print("%d + %d = %d" % (p1,p2,p1+p2))       #定义函数中的语句块
add(1,2)                                        #调用函数
```

在 python 中，函数参数是没有类型的，只有调用传参的时候才会根据你传入参数的类型去定义。试一试传入不同类型的参数值！

```
def type_para(t):
    print(type(t))
```

3.2 返回值

何谓返回值，返回值就是指调用函数，函数内部语句块执行后，通过 return 语句将函数内的执行结果，返回给调用者。如果函数中没有 return 语句，返回值为 None。

返回值可以有多个，试一试用对应返回值个数的变量来取返回值和用一个变量来取所有返回值！

```
def divide(a, b):
    q = a / b
    r = a - b
    return q, r
x, y = divide(42,5)
a = divide(42,5)
```

3.3 函数的参数

3.3.1 位置参数（必选参数）

位置参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。
但是我们可以在调用时加上参数名，就可以修改顺序了。如：

```
def func(name, score):
    print("%s 的成绩是%d" % (name, score))
func("zhansan", 90)                 #必须这样调用，否则输出就是错的
func(score=90, name="zhansan")  #或者这样调用
func(90, name="zhansan")          #不能这样调用
```

3.3.2 默认参数

在函数声明时就加入默认值，调用函数时，如果没有传递参数，则会使用默认参数，如果调用时给默认参数传递了参数，则会替换默认值而使用传递的值，

```
def func(name, score=90):
    print("%s 的成绩是%d" % (name, score))
func("zhangsan")                    #可以这样调用
func("zhansan", 80)           #也可以这样
func(score=90, name="zhansan")  #依然可以这样
```

注意：一定要保证位置参数在前，默认参数在后，否则 Python 的解释器会报错。

3.3.3 可变参数

要定义出这个函数，我们必须确定输入的参数。由于参数个数不确定，我们首先想到可以把 a，b，c……作为一个 list 或 tuple 传进来。比如下面的例子，不确定需要多少个数相加，可以这样写。

```
def func(*args):
    sum = 0
    for n in args:
        sum = sum+n
    return sum
func(1,2,3,4,5)                                #可以加任意多个参数值
```

可变参数就是传入的参数个数是可变的，可以是 1 个、2 个到任意个，还可以是 0 个。
那么接收可变参数的函数内部是什么样的呢：

其实在函数内部 args 接收到的是一个 tuple；我们要使用这些参数就必须按照 tuple 的方式处理。

另外，如果你已经有一个 list 或者 tuple 了，那你想调用的时候怎么办？调用的时候在参数前面加*，如：func(*tuple)，就可以将 tuple 拆分成多个参数。以上面的例子。

```
number=(1,2,3,4,5)
func(*number)
```

其实对于位置参数和默认参数，我们也可以用这种方法，先组装一个 tuple，然后在调用的时候拆成单个的参数。以 3.3.1 位置参数的例子，试一试结果！

```
student=("zhansan", 80)
func(*student)
```

3.3.4 关键字参数

可变参数允许你传入 0 个或任意个参数,这些可变参数在函数调用时自动组装为一个 tuple。而关键字参数允许你传入 0 个或任意个含参数名的参数,这些关键字参数在函数内部自动组装为一个 dict。与可变参数相同，需要使用 dict 的操作方法来操作这个 dict。

```
def func(**kwargs):
    print(kwargs)
func(name="zhansan", score=90, age=30)
```

在调用函数时，我们可以将参数组装成 dict，通过 func(**dict)的方式传参。

```
def func(name, age=20)
    print("%s' s age is %d"% (name,age))
person = {"name":" Bob"," age": 27}
func(**person)
```

3.3.5 参数组合

Python 中的各种类型的参数可以进行组合。

```
def func(para, *args, para2=None, **kwargs)
```

思考：当顺序为位置参数、默认参数、可变参数和关键字参数。形式：

```
def func(para, para2=None,*args, **kwargs)
```

这时候默认参数相当于失效了，就无法使用到默认值，且必须传参，为什么？

4. 面向对象

4.1 类和实例

什么是类?

类就是一类东西，如门，车均指一类东西

类也可以是一类东西的模板(Template)

类可以定义属性和方法，也可以继承父类的方法

属性指：车的马力，速度，轴距，或者门的宽度，厚度等

方法指：车可以运行，停止，鸣笛，门可以开，关等

什么是对象?

对象 (Object) 是类的实例 (Instance)，所以”对象”=“实例”

车牌为”川 A12345”的车是”车”的一个实例

东方广场 C 座 7 楼门牌号为 701 的门是”门”的一个实例

一个实例可以实现类的所有属性和方法

类的申明：

class Person(): #class 关键字申明类，后面接类名（单词首字母大写）

类的属性及实例化：

类的属性，就是指某一类事物的特有属性，每个对象的属性可能是不同的。实例化的过程其实就是初始化类为实例的过程，这个过程中会调用__init__方法为实例属性赋值。

另外有一种类的属性，这种属性可以用“类名.属性”的方式访问，类名不能访问实例属性，只能访问类属性，实例化的对象可以访问类属性也可以访问实例属性。

```
class Person():
    x = "class attribute"  #类属性
    def __init__(self, name, age):   #初始化，绑定实例属性
        self.name = name
        self.age = age
Person.x        #调用类属性
p = Person ("zhansan", 30)
p.name          #调用实例属性
p.x              #调用类属性
```

上例中我们在__init__方法中传入了参数，__init__是一种特殊写法，前后双下划线这种写法都是 python 中类的特殊方法，有特殊的用途；我们前面讲过__init__是类的初始化方法，在对象实例化的过程中给属性赋值,但是__init__方法有参数,就表明需要给传参才能赋值，因此如果你实例化的时候不传入参数，则无法给属性绑定值，也就无法实例化。那么就是说，只要类的__init__有参数值，就必须在实例化的时候传参。

类的方法：

对于人这种类来说，人的行走、说话、吃饭，都可以看作是人的动作（方法）。

```
class Person ():
    def __init__(self, name, age):   #初始化，绑定实例属性
        self.name = name
        self.age = age
    def walk(self):
        print("%s 都已经%d 岁了，因此要多走路多运动！" %(self.name, self.age))
p = Person ("zhansan", 30)
p.walk()
```

除了上例中的实例化必须传参，方法本身也可以有参数，那么此时方法自身的参数只对当前方法有效。一定要注意区别实例化(__init__)的参数和方法自身的参数。

```
class Person ():
    def walk(self, name, age):
        print("%s 都已经%d 岁了，因此要多走路多运动！" % (name, age))
p = Person ()
p.walk("zhansan", 30)
```

4.2 继承

何谓继承?

门是一个类，它规定了所有门的方法和属性

铁门，木门，防盗门是门的一个分类，它也有所有门都共有的属性和方法，但同时它还有一些自己特有的属性和方法

可以为铁门，木门和门分别建三个类，各自实现自己的方法和属性

也可以为门建立类，让铁门和木门继承门的方法和属性

```
class Father():                                  #申明父类
    def __init__(self, a, b)
        self.a = a
        self.b = b
    def add(self):
        return self.a + self.b
    def sub(self):
        return self.a  -  self.b
class Son(Father):                               #申明子类，继承父类
    def print_add(self):
        print(self.add())                  #调用父类的方法
    def sub(self):                         #重写父类的同名方法
```

```
        return self.a * 2 - self.b    #调用父类的属性
son = Son(6,3)
son.sub()
son.add()
```

类的继承，表示子类继承父类的所有属性和方法。

- 这些属性和方法可以通过“self.属性”或者“self.方法（）”的方式在子类中调用；
- 父类的属性和方法，子类的实例化对象也可以直接调用；
- 子类可以重写父类的同名方法，重写以后，父类的方法并不会改变，但是子类在调用该方法的时候自动调用重写后的方法。

4.3 访问限制

在 Class 内部，可以有属性和方法，而外部代码可以通过直接调用实例变量的方法来操作数据，这样，就隐藏了内部的复杂逻辑。但是，从前面 Person 类的定义来看，外部代码还是可以自由地修改一个实例的 name、age 属性：

```
p.age = 50
p. walk()
```

如果要让内部的属性不能外部随便访问，那么就在属性名称前加两个下划线__。方法或属性前面加双下划线表示的是私有类型(private)的变量，只能允许这个类本身进行访问了，其子类都不能访问。依然以 Person 类为例。

```
class Person ():
    def __init__(self, name, age):   #初始化，绑定实例属性
        self.name = name
        self.__age = age
    def walk(self):
        print("%s 都已经%d 岁了，因此要多走路多运动" % (self.name, self.__age))
p = Person ("zhansan", 30)
p.__age=50
p. walk()
```

通过外部 p.__age 是无法修改类中的__age 属性的，实际上虽然没有报错，但是其实是给 p 这个对象新增了一个__age 属性，而原本的__age 属性并没有被修改。原本的__age 已经被解释器自动改成了_Person__age 了。

4.4 编码规范

Python 的 PEP8 规范了一些 python 的编码规则，此节时截取一部分重要的规范，如下：

I. 缩进。4 个空格的缩进（编辑器都可以完成此功能），不使用 Tab，更不能混合使用

Tap 和空格；

II. 尽量避免单独使用小写字母‘l’，大写字母‘O’等容易混淆的字母；

III. 类命名尽量以大写字母开头，多个单词组成，每个单词首字母大写；

IV. 函数命名使用全部小写的方式，可以使用下划线；

V. 常量命名使用全部大写的方式，可以使用下划线；

VI. 类的属性（方法和变量）命名使用全部小写的方式，可以使用下划线；

VII. 变量、函数、属性等命名尽可能采用有意义的英文单词，多单词由下划线连接。

5. 模块

5.1 引入模块

我们讲 python 中有丰富的第三方库，那么当我们下载了第三方库之后，需要怎么引入进我们的代码呢？python 提供了两种引入方式 import *package/class/def* 与 from *package* import *class/def/**：

两种方式都可以引入模块、包、类、方法、函数。import 只能引入顶层的，from 可以引入到方法/函数级别。我们在用的时候，如果没有命名冲突，我们可以通过 from 引入到类或者方法/函数级别，方便我们调用。

示例，打印当前时间，试一试并注意观察使用时的调用方式！

```
import time
print(time.ctime())
from time import ctime
print(ctime())
```

5.2 引入自己写的模块

除了引入第三方模块,我们还可以引入我们自己编写的 py 文件,在同一目录下,直接“import 文件名”或“from 文件名 import 类/方法/函数”。

同一目录调用示例：

```
Project/
 |——calc.py
 |——calc_add.py
```

先键一个文件 calc. py，内容如下：

```
def add(a,b):
    return a+b
```

calc_add.py 调用 calc.py 中的 add()函数，calc_add.py 文件内容如下：

```
from calc import add
print(add(2,3))
```

如果是跨目录的情况，只需要在 import 或者 from…import…时，在文件名称前面加上包的名称。

跨目录调用示例：

```
Project/
 |——model/
          |——calc.py
 |——calc_add.py
```

calc_add.Py 调用 calc.py 中的 add()函数，calc_add.py 文件内容如下：

```
from model.calc import add
print(add(2,3))
```

6. 异常处理

6.1 try 语句

高级语言通常都内置了一套 try...except...的错误处理机制。当我们认为某些代码可能会出错时，就可以用 try 来运行这段代码，如果执行出错，则后续代码不会继续执行，而是直接跳转至错误处理代码，即 except 语句块，执行完 except 后，如果有 finally 或 else 语句块，则执行 finally 或 else 语句块，至此，执行完毕。

try 语句，当我们打印一个我们没有定义的变量 name 时，一定会抛出异常，此时可以用 try 捕获这个异常，并且抛出我们定义的错误，且不会影响后续语句的执行。

```
try:
     print(name)
except NameError:
     print("name 不存在！")
```

6.2 try…except…finally

不论是否发生异常，都执行 finally 中的语句。

```
try:
    print(name)
```

```
except NameError as e:
    print(e)
finally:
    print("我必然会出现")
```

6.3 try…except…else

try...except 还可以和 else 联用，不发生异常时则执行 else 中的语句。

```
try:
   print(name)
except NameError as e:
    print(e)
else:
    print("没有异常我才会出现")
```

6.4 try…except…except

try…except…except，except 可以有很多个，类似 if 语句中的 elif，对于捕获到的不同的异常进行不同的处理。

```
try:
    f = open(name, 'r')
except NameError:
    print("参数未定义")
except IOError:
    print("文件不存在")
else:
    print("没有异常")
```

6.5 抛出异常

Python 可以在需要抛出异常的地方，通过 raise 语句直接抛出异常。

```
raise NameError("这是一个名称错误！")
```

6.6 常见的异常分类

异常	说明
AssertionError	assert（断言）语句失败
AttributeError	试图访问一个对象没有的属性，比如 foo.x ，但是 foo 没有 x 这

	个属性。
IOError	输入/输出异常，基本上是无法打开文件。
ImportError	无法引入模块或者包，基本上是路径问题
IndentationError	语法错误，代码没有正确对齐
IndexError	下标索引超出序列边界，比如当 x 只有三个元素，却试图访问 x[5]
KeyError	试图访问字典里不存在的键
KerboardInterrupt	Ctrl + C 被按下，主要针对无限循环
NameError	使用一个还未被赋值予对象的变量
SyntaxError	Python 代码非法，代码不能解释
TypeError	传入对象类型与要求的不符
UnboundLocalError	试图访问一个还未被设置的局部变量，基本上是由于另一个同名的全局变量，导致你以为正在访问它
ValueError	传入一个调用者不期望的值，即使值的类型是正确的
FileNotFoundError	试图打开一个不存在的文件或目录

如果你不知道该 except 后跟什么 Error 的时候，可以使用 Exception。 Exception 为所有 Error 的基类。

```
try:
    print(name)
except Exception as e:
    print(e)
```

7. 文件读写

进行文件读写，你必须要以读或者写文件的模式打开一个文件对象，使用 python 内置的 open()函数，传入文件名和标示符。

7.1 读取文件

读取文件，open()第一个参数表示文件位置，可以用绝对路径或者相对路径，当使用相对路径的时候，表示相对于当前脚本的路径。标示符'r'表示读，这样我们就成功地打开了一个文件：

```
f = open("E:\\test.txt", "r")   #只读方式打开文件
```

如果文件打开成功，接下来，调用 read()方法可以一次读取文件的全部内容，Python 把内容读到内存：

```
data = f.read()                 #读取文件所有内容
```

最后一步是调用 close()方法关闭文件。文件使用完毕后必须关闭，因为文件对象会占用操作系统的资源，并且操作系统同一时间能打开的文件数量也是有限的：

```
f.close()
```

还有按行读取的方式：

```
f = open("E:\\test.txt", "r")
line = f.readline()          #读取一行，返回的是字符串
lines = f.readlines()        #按行读取整个文件，并返回列表，一个元素一行
```

注意：文件读写中其实存在一个文件指针。当你用 r 标识符打开文件时，指针在文件开始的地方，通过读取方法（read， readline， readlines）读取时，文件指针会向后移动。比如用 readline()，第一次调用读取的是第一行，第二次调用就只能从第二行开始读，就算用 read()或者 readlines()，也只能从第二行开始读。因为此时文件指针已经移动到第二行的开头位置。

7.2 文件的写入

文件写入，打开方式有标识符 w 和 a，w 表示打开一个文件进行写，并且文件指针在文件开头，写入内容会覆盖原来文件内容，并且如果文件不存在，会创建一个文件进行写入；a 表示追加，文件指针在文件的末尾，写入内容追加在原来的文件内容之后，同样如果文件不存在，则创建一个文件进行写入。下面以 w 方式介绍：

```
f = open("E:\\test.txt", "r")
f.write("Hello World")    #将字符串写入文件
f.close()                   #关闭文件，写入内容保存到硬盘
```

写入文件后，需要 close()文件，才会将文件内容写入硬盘中（也就是关闭文件，你才能通过浏览硬盘中的文件查看写入的内容）。如果你不想关闭文件，又想看结果，可以使用 flush()方法，直接将内容写入文件。

```
f.flush()
```

7.3 常见的文件操作类型

以下是常见的文件读写类型，都是从 r，w，a 三种标识符演化的，比如 rb，wb 中的 b 表示二进制，即以二进制的方法进行读写。如果你操作的文件不是文本文件，则读取写入都要以二进制的方式。

打开类型	说明
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。
w	打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。

	如果该文件不存在，创建新文件。
w+	打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

8. 数据库操作

Python 的课程中，只讲通过 python 操作 mysql 数据库，其他数据库的操作方式大同小异。操作 mysql 数据库，比如安装第三方的库 PyMySQL。PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个库

安装 PyMySQL 方式如下，打开 CMD（命令提示符）：

```
pip install PyMySQL                          #注意大小写
```

数据库查询示例：

```
import pymysql                          #引入 pymysql
config = {                               #组装一个数据库连接的字典
    'host': '127.0.0.1',                        #数据库服务器地址
    'port': 3306,                             #端口，mysql 一般默认 3306
    'user': 'root',                            #数据库账号
    'password': 'root',                       #数据库密码
    'db': 'ecshop',                           #数据库名
     'charset': 'utf8'                      #数据库编码，否则取出的中文可能乱码
     'cursorclass': pymysql.cursors.DictCursor  #以字典的方式返回结果
    }
db = pymysql.connect(**config)       #传入组装的字典
cur = db.cursor()                      #创建一个游标对象
sql = "SELECT * FROM ecs_admin_user"       #组装 SQL 语句
```

```
cur.execute(sql)                          #执行 sql
data = cur.fetchall()       # fetchall()获取所有数据
db.close()                  #关闭数据库连接
```

数据库操作：

connect()： 打开数据库连接，需要传入数据库的相关信息。

cursor()： 创建一个可以执行 sql 语句的游标对象。

execute()： 执行 sql 语句。

fetchone()： 该方法获取下一个查询结果集。结果集是一个 dict，列名为 key，数据为 value

fetchall()： 接收全部的返回结果行。返回一个 list，list 元素由每行行成的 dict 组成

rowcount： 这是一个只读属性，并返回执行 execute()方法后影响的行数。

Python 查询 Mysql 使用 fetchone() 方法获取单条数据, 用 fetchall() 方法获取多条数据。

注意：'cursorclass'设置不同的参数，返回的结果类型是不同的。默认不设置的时候是 fetchone()是元组，而 fetchall()是二维元组。