Python 程序设计入门指引

Python Learning

—a study guide for freshmen

学 业 支 持 中 心

Academic Support Center

编者的话

同学们，你们好！为了帮助大家 Python 语言程序设计课程的学习，学业支持中心的志愿者们参考了各大网站相关资源，编写了本册《Python 程序设计入门指引》。

本册指南主要讲解了 Python 语言的编程基础内容和简单的面向对象编程，涵盖了 Python 语言学习初期的主要知识点，推荐辅助教学课程进行预复习。希望本册对你的学习有所帮助。

如有知识不当或错误之处，欢迎读者多多批评指正，以便于我们今后进一步改进。

联系方式

邮件：bjtuwhascwk@163.com

电话：0631-3806520

地址：思源东楼 508 室

QQ：967348142

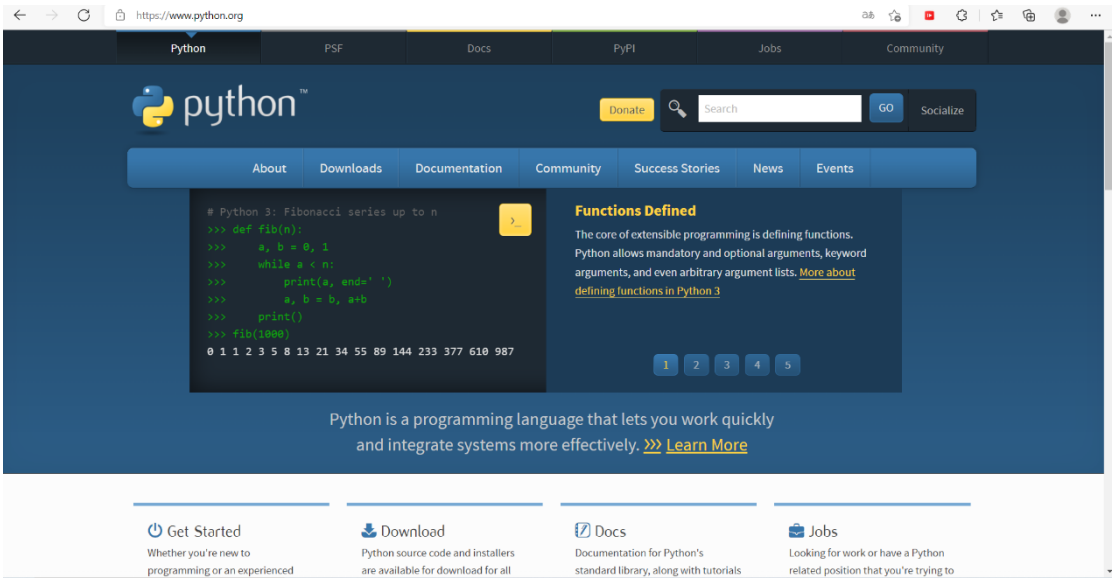
北京交通大学威海校区学业支持中心

目录

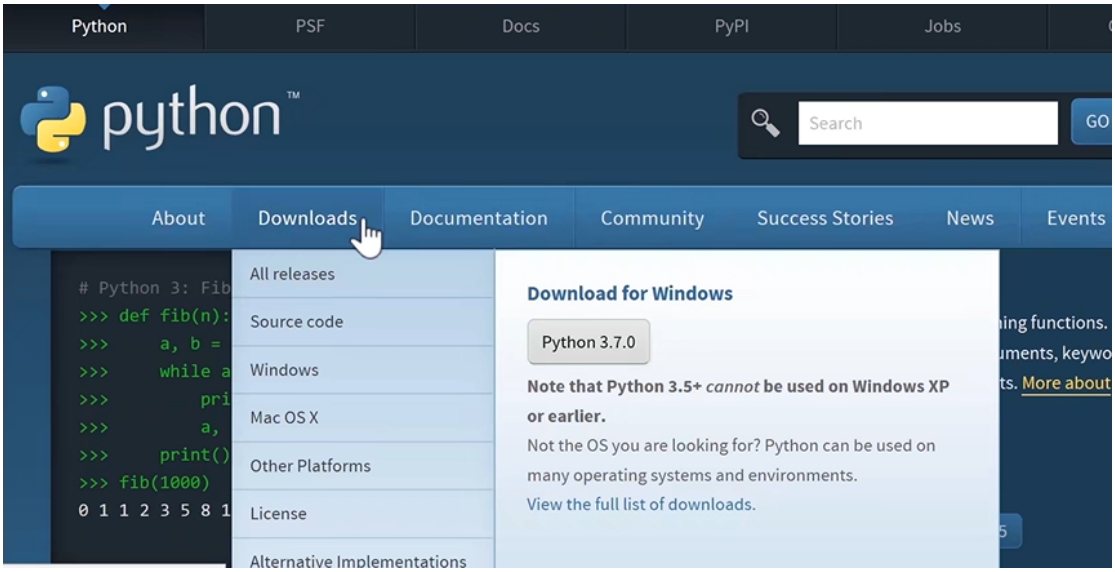
CHAPTER1 起始步

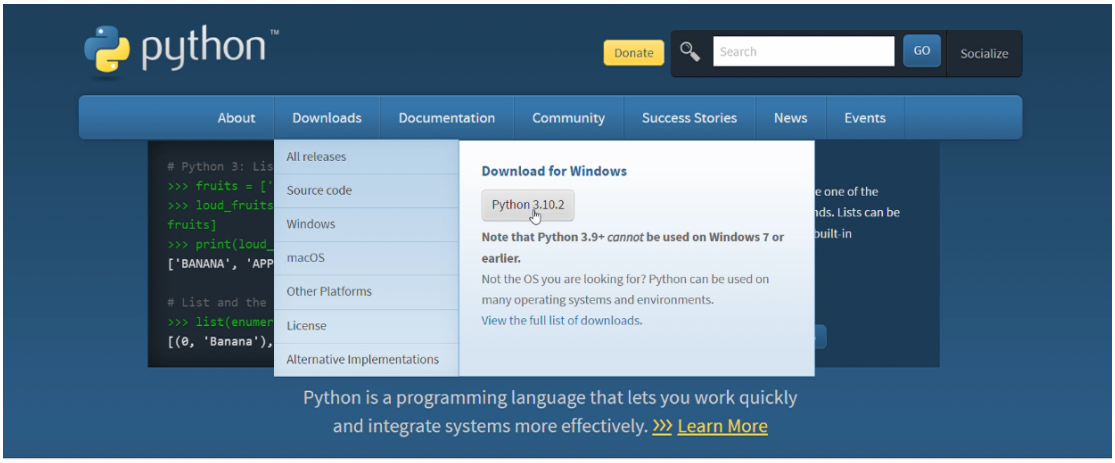
1.1 安装 Python

1. 打开浏览器输入 http://python.org 如下：

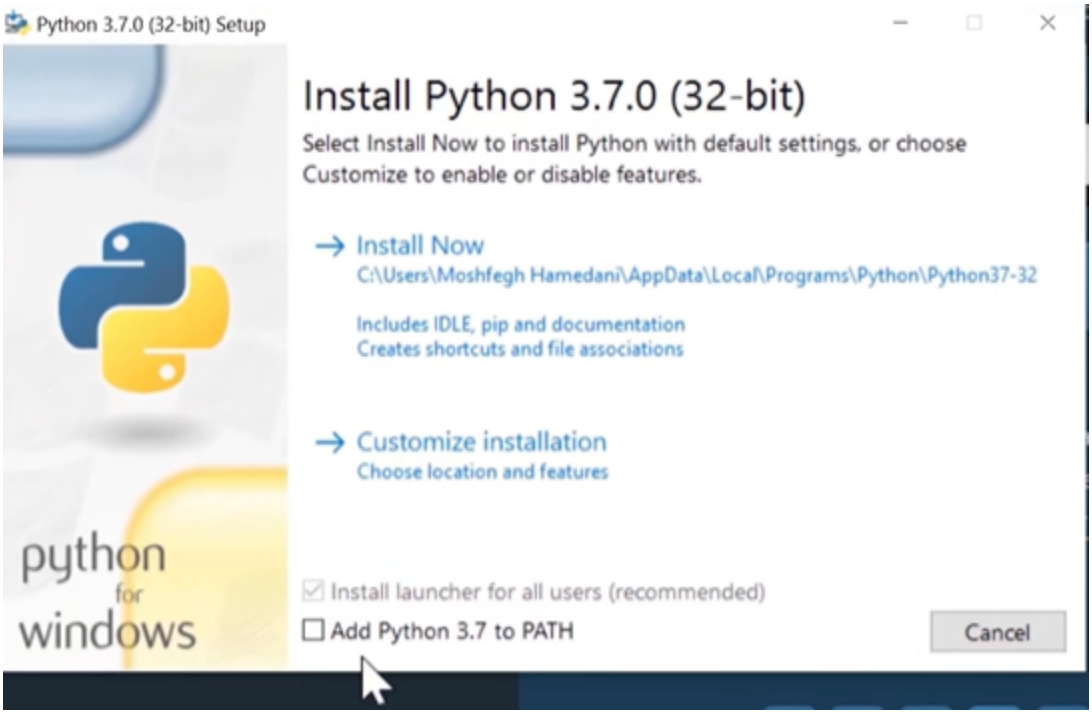


2. 点击 Downloads：





此处为 python 最新版本，继续点击此处下载 Python。
如果是 windows 系统要选择添加 python 到路径（此处截图示例为 3.7 版本）。



添加到路径后继续安装（图中的 Install Now）

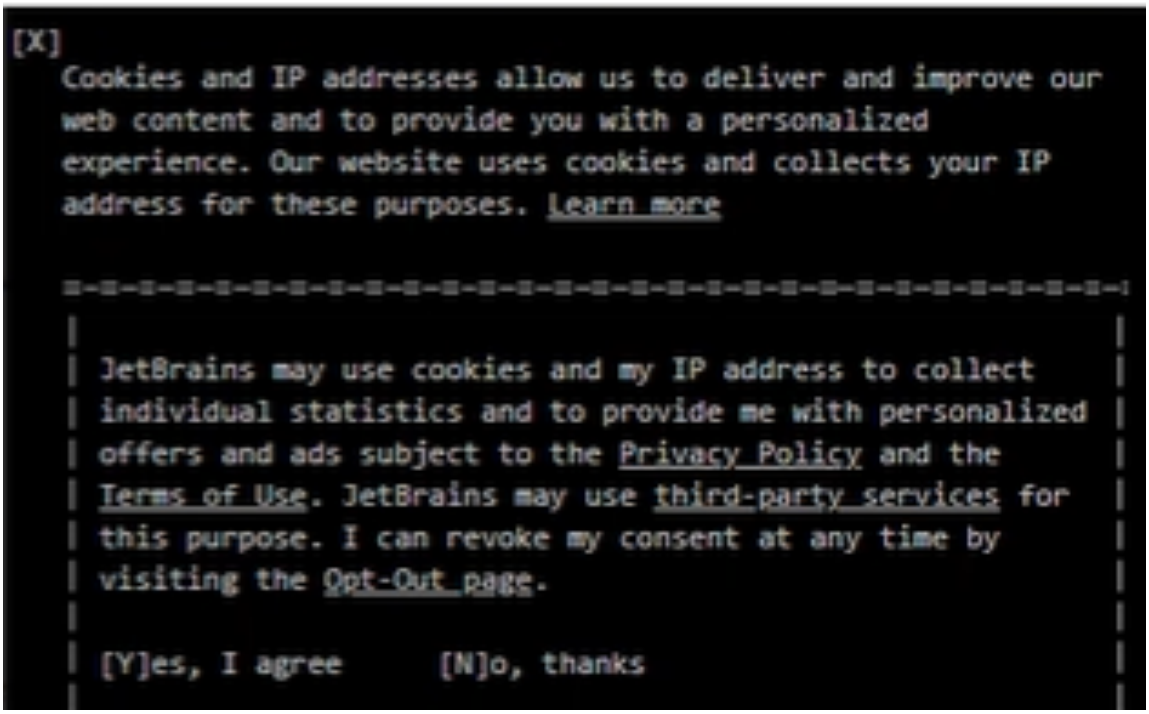
Python 下载完成！

1.2 下载安装 PyCharm

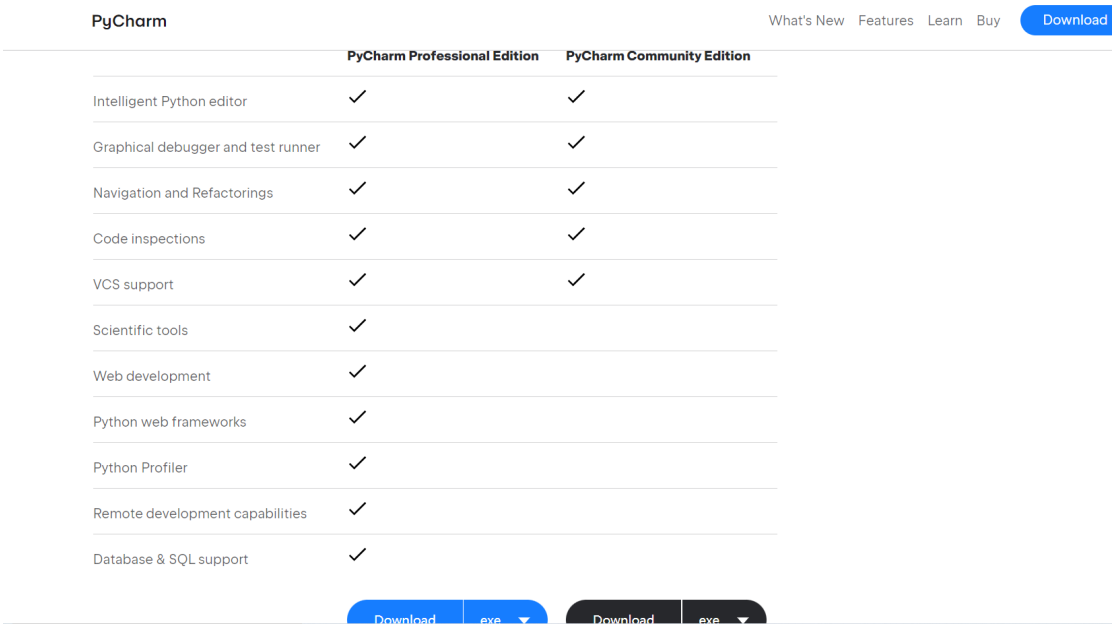
下载步骤：

1. 打开浏览器输入 PyCharm: the Python IDE for Professional Developers by JetBrains 如下：

2. 右下角弹出 cookies 选项点 No, thanks：

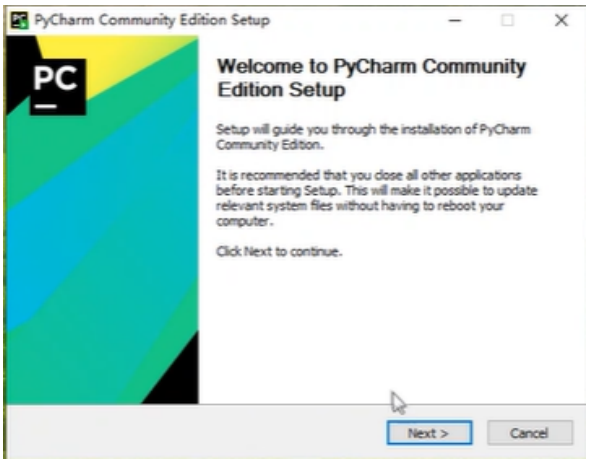


3. 向下翻，下载 Community 版本（Professional 版本付费）。

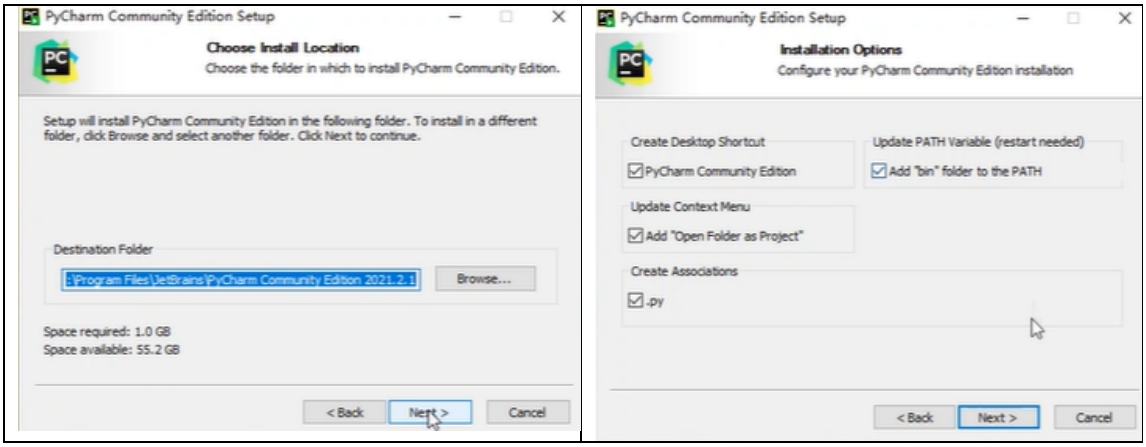


安装步骤：

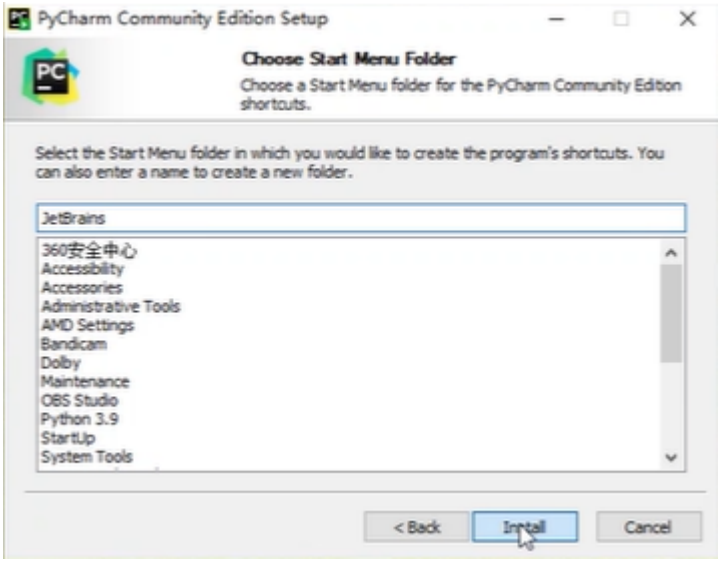
1. 双击桌面图标，弹出以下界面：



2. 点击 Next



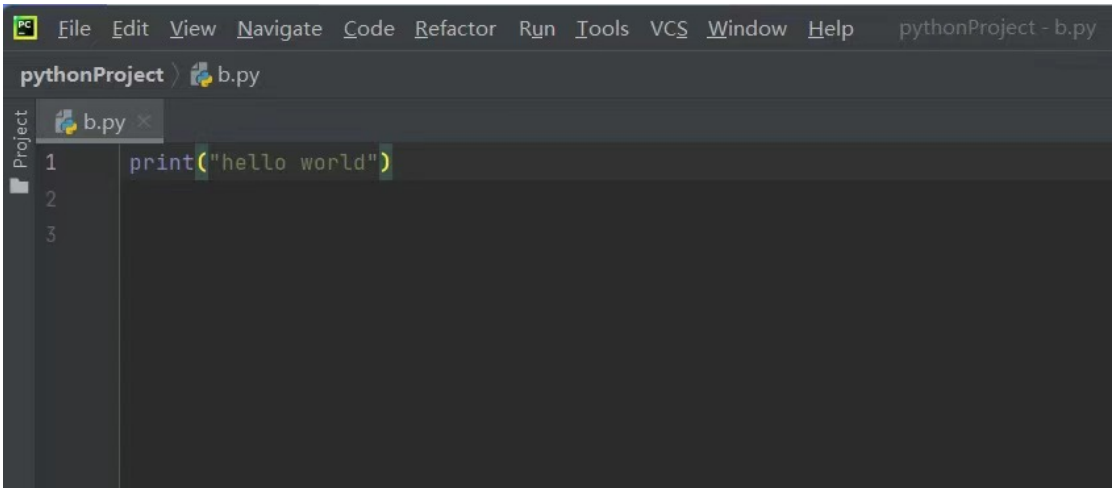
3. 全选，然后点 Next，弹出



4. 点击 Install，安装完毕（安装完最好重启）

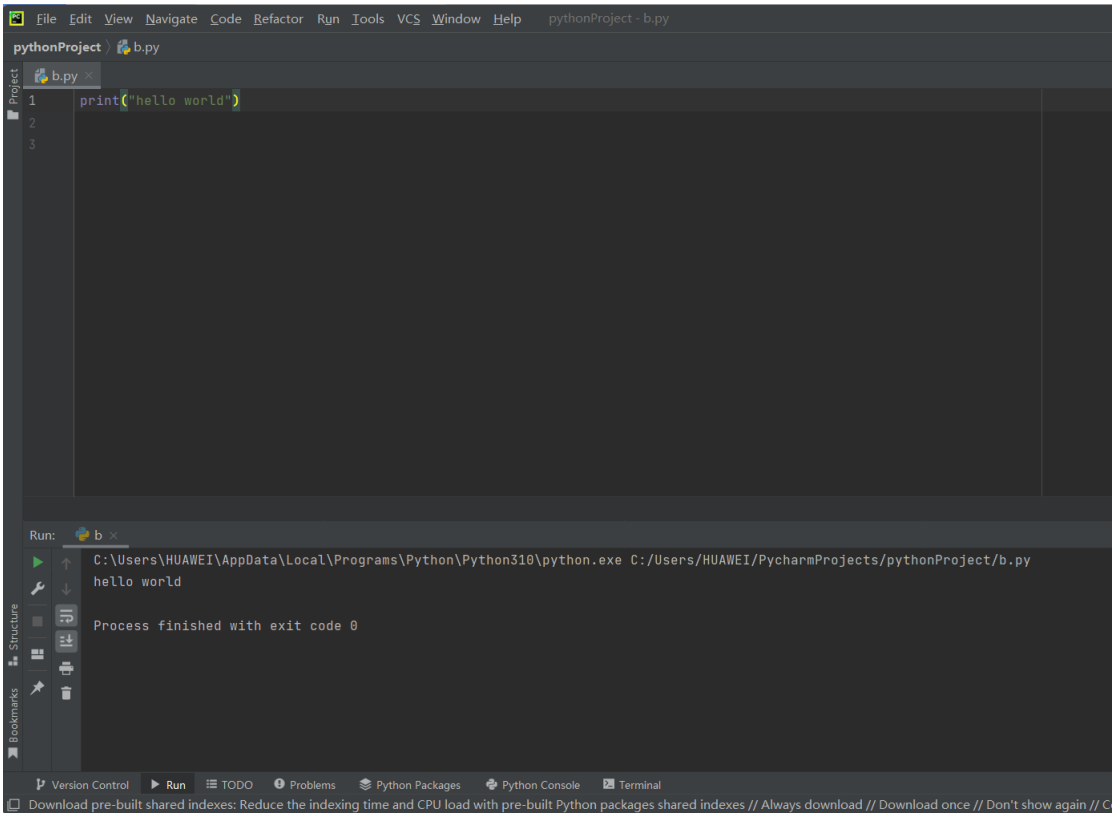
1.3 输出第一个程序：“hello world”

选择编程语言为 Python：



表示“输出”的基本内置函数在 Python 中基本格式为：print(“XXX”)

如图希望输出结果为 hello world，如此输入：



点击快捷键后显示结果“hello world”。

CHAPTER2 数据类型

可以用 type(变量名) 的方式查看它的数据类型，例如:

```
a = 1
type(a)
print(type(a))
```

输出结果：

<class 'int'>

2.1 数字

int - 整型/整数

用于存储整数数据，如：年龄,人数,游戏等级

声明方式：变量名 = 整数值，例如：

```
a = 1
```

一次性给多个变量赋值：变量名 1，变量名 2=数值 1，数值 2，例如：

```
c, d = 3, 4
```

float - 浮点型/小数

用于存储带小数的数据，如：金额余额,身高,体重,π

声明方式：变量名 = 浮点数值，例如：

```
my_money = 99999.99
```

注意:

在 Python 中不动用模块的情况下，1.2-1 并不是等于 0.2（其他情况也是这样的）。

这是因为小数在内存中是以二进制形式存储的,小数点后面的部分在转换成二进制时很有可能是一串无限循环的数字,无论如何都不能精确表示,所以小数的计算结果一般都是不精确的。例如：

```
a = 1.2 - 1
print(a)
```

输出结果：

0.19999999999999996

complex- 复数

Python 还支持复数，复数由实数部分和虚数部分构成，复数的实部 a 和虚部 b 都是浮点型（赋值若时不加小数，Python 会将数值的第一位小数储存为 0）。

注：虚数不能单独存在，它们总是和一个值为 0.0 的实数部分一起构成一个复数，同时虚数部分必须有后缀 j 或 J。

声明方式：变量名= a + bj 或者变量名=complex(a,b)

查看实部和虚部

变量名 .real （实部）

变量名 .imag （虚部）

```
t = 4 + 5j
print(t.real)
print(t.imag)
```

输出结果：
4.0
5.0

str-字符串

在 python 中，文字存储通过字符串的方式来进行，也可以用于存储带文本(汉字、字母、特殊符号)的数据。字符串的表现形式为用一对引号(单，双，三引号)包裹起来的数据里面可以存储汉字、字母、数字、特殊符号等数据。

定义方式：变量名 = ″字符串内容″，例如：

```
name = "张三"
names = "张三，李四"
a = "123456"
```

注：不同引号之间不能混搭，如 name = ‘张三″。

单引号，双引号一起嵌套使用：

```
print("老师说："大家都很棒"") #报错，Python 不知道哪对引号是一对
print('老师说： "大家很棒"')   #外层换成单引号包裹
```

三引号：

三个单引号或者三个双引号，用于多行注释。例如：

```
"""
12345
上山打老虎
"""
'''
老虎没打着
打着小松鼠
'''
```

同时，可用于建立多行字符串，例如：

```
str = '''
hello world
'''
print(str)
```

输出结果：

hello world

2.2 字符串

字符串格式化：

语法:

在字符串前面加上一个 f/F,把要输出的变量用大括号{}进行包裹，print(f″XXXX{变量},XXX{变量}″)。

例如：

```
name = "张三"
age = 18
print(f"大家好，我是{name}，今年{age}岁。")
```

输出结果：

大家好，我是张三，今年 18 岁。

格式化中 format 方法：

语法：

print('XX{}XX{}'.format(数据 1,数据 2)) 注： 传入的数据类型是不限的。例如：

```
print("大家好，我是{}，今年{}岁。".format("张三", 18))
```

输出结果：

大家好，我是张三，今年 18 岁。

转义字符

有一些符号,字符是具有特别含义,特别功能。

\n 换行

例如：

```
print("大家好\n我是张三")
```

输出结果：

大家好

我是张三

\t 相当于 tab,一般为 4 个空格。

```
print("大家好\t我是张三")
```

输出结果：

大家好 我是张三

\b 代表退格，即\b 后面的一字符替换\b 前面的一个字符；若\b 后面的是空格则效果上是删除了\b 前面的一个字符，其实是\b 后面的一各空格替换了\b 前面的那个字符；如果\b 后面没有任何字符，实际上是没有效果的。

例如：

```
print("大家好我是\b 张三")
```

输出结果：

大家好我张三

```
print("大家好我是\b\b 张三")
```

输出结果：

大家好张三

\ 在字符串行尾的续行符，即一行未完，转到下一行继续写。例如：

```
print("大家好我是\
张三")
```

输出结果：

大家好我是张三

\\　　正常输出一个反斜杠
\″　　正常输出一个引号

字符串函数：（下面列举常见函数）

字符串函数	描述	用法
len()	判断字符串长度	len(变量名)
islower()	判断字符串是否全由小写字母组成，输出值为 True 或 False	变量名.islower()
isdigit()	判断字符串是否全数字，输出值为 True 或 False	变量名.isdigit()
salpha()	判断字符串是否全字母，输出值为 True 或 False	变量名.salpha()
upper()	字符串中所有字母转大写字母	变量名.upper()
swapcase()	字符串中字母大写转小写，小写转大写	变量名.swapcase()
lower()	将字符串中所有字母转小写字母，只支持英文字母	变量名.lower()
title()	字符串中每个单词首字母大写，其余字母小写（区分单词以空格区分）	变量名.title()
capitalize()	字符串首个字母大写，其余字母小写	变量名.capitalize ()

例如：

len()

```
a = "ABCDE"
print(len(a))
```

输出结果：

5

lower()

```
a = "ABCDE"
print(a.lower())
```

输出结果：

abcde

title()

```
a = "student teacher apple"
print(a.title())
```

输出结果：

Student Teacher Apple

capitalize()

```
a = "student teacher apple"
print(a.capitalize())
```

输出结果：

Student teacher apple

数学函数：

min()和 max()可用于查找可迭代(列表，元祖，字典，集合中的数字)的最小值或最大值。

```
x = max(5, 15, 25)
y = min(5, 15, 25)

print(x)
print(y)
```

输出结果：

25

5

pow(x,y)　函数将 x 的值返回为 y 的幂（即 x 的 y 次方）。

```
x = pow(4, 3)
print(x)
```

输出结果：

64

Python 还有一个名为 math 的内置模块，该模块扩展了数学函数的列表。如要使用，需要导入，里面的函数不再一一列举。

方法： import 模块名

```
import math
```

CHAPTER3 运算

简单运算

以下假设两个变量，变量 a=10，变量 b=21。

运算符	描述	实例
+	加—两个对象相加	a + b 输出结果 31
-	减—得到负数或是一个数减去另一个数	a - b 输出结果 -11
*	乘—两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 210
/	除—x 除以 y	b / a 输出结果 2.1
%	取模—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—向下取接近商的整数	>>> 9//2
		4
		>>> -9//2
		-5

赋值运算符

对于简单的算术运算，Python 提供了简单的赋值运算符，以加法赋值运算符为例：

运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a

Python 中简单运算的结果用如下方式输出：

```
print(1+2)
print(2-1)
print(2*3)
print(6/2)
```

注意：这种情况下应该注意的是在 print 的括号里应该**直接输入**，如果用引号引起来就不会输出结果，而是输出了字符串，这就类似于打出“ hello, world” 无论引号里有什么都会原封不动的输出，比如：

```
print('1+2')
```

输出结果：
1+2

注意：
a)必须使用英文字符，中文字符会报错。
b)不要在 Python 中随便打空格，Python 中使用 Tab 键（相当于四个空格）来进行缩进。
c)Python 中的四则混合运算符合数学中的混合运算顺序，即括号优于开方、乘方，开方、乘方优于乘除，乘除优于加减。

CHAPTER4 选择

4.1 布尔表达式

布尔表达式（Boolean expression）是能计算出一个布尔值 True 或 False 的表达式。

例如：

```
print(1 < 10)
```

输出结果：

True

```
print(1 > 10)
```

输出结果：

False

常用的比较运算符：

运算符	<	<=	>	>=	==	!=
描述	小于	小于等于	大于	大于等于	等于	不等于

（比较运算符的相等是两个等号，单个等号是用来赋值的。）

4.2 if 语句

如果条件正确就执行一个单向 if 语句。

当且仅当条件为 True 时，一条单向 if 语句执行一个动作。单向 if 语句的语法如下：

if boolean-expression（上文所提及的布尔表达式）：**（冒号是必不可缺的）**

    statement(s) （命令语句）

这里的 statement(s)必须相对于 if 向右缩进，通常缩进 4 个空格(tab)。

例如：

```
a = 2
if a > 1:
    print("当 a 大于 1 时这句话将会被输出")
```

4.3 双向判断 if-else 语句

双向判断 if-else 是 if 语句的衍生，语句除了执行条件为真下的命令，还可以进一步来执行条件为假下的命令。

if-else 语句根据条件是 True 还是 False 指定不同的动作。if-else 语句的语法如下：

if boolean-expression（布尔表达式）：

    statement(s)-for-the-true-case（条件为真情况下的命令）

else:**（冒号是必不可缺的）**

    statement(s)-for-the-false-case（条件为假情况下的命令）

例一：

```
a = 0
if a > 1:
    print("当 a 大于 1 时这句话将会被输出")
else:
    print("当 a 小于 1 时这句话将会被输出")
```

例二：计算圆的面积

```
import math
radius = eval(input("Enter a number:"))

if radius >= 0:
    area = radius * radius * math.pi
    print("The area for the circle of radius", radius, "is", area)
else:
    print("Negative input")
```

例三：判断一个数的奇偶性

```
number = eval(input("Enter an integer:"))

if number % 2 == 0:
     print(number, "is even.")
else:
     print(number, "is odd.")
```

4.4 嵌套 if 和多向判断 if-elif-else 语句

将一个 if 语句放在另一个 if 语句中就形成了一个嵌套语句。

例如，下面的语句是一个嵌套 if 语句：

```
a = 1
if a > 0:
    if a < 2:
        print("当 a 小于 2，大于 0 时这句话将会被输出")
else:
    print("当 a 大于等于 2 或小于等于 0 时这句话将会被输出")
```

嵌套 if 语句可以用来实现多种选择，例如：根据分数的大小，给变量 grade 赋值一个字母值。

```
score = eval(input("Enter a number:"))

if score >= 90.0:
    grade = 'A'
else:
    if score >= 80.0:
        grade = 'B'
    else:
        if score >= 70.0:
            grade = 'C'
        else:
            if score >= 60.0:
                grade = 'D'
            else:
                grade = 'F'
```

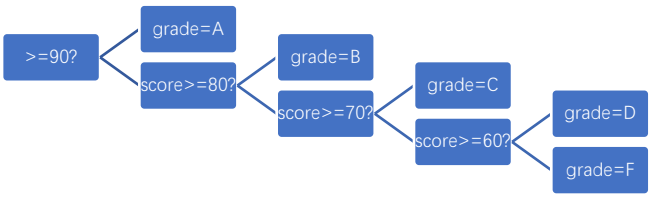
a)

```
score = eval(input("Enter a number:"))

if score >= 90.0:
    grade = 'A'
elif score >= 80.0:
    grade = 'B'
elif score >= 70.0:
    grade = 'C'
elif score >= 60.0:
    grade = 'D'
else:
    grade = 'F'
```

b)

a)和 b）是相等的，推崇使用多向 if-elif-else 语句。
此外，复杂的多层嵌套往往不好写，此处提供一个技巧——画树形图，以此段代码为例：



画出树状图后可直接写出代码。

4.5 选择语句中的常见错误

选择语句中的大多数常见错误都是由不正确的缩进问题导致的。
如下，在 b） 中，print 语句不在 if 语块内，要把它放进 if 块中，必须像 a） 中那样将 print 语句缩进。

```
import math
radius = eval(input("Enter a number:"))

if radius >= 0:
    area = radius * radius * math.pi
    print("The area for the circle of radius", radius, "is", area)
```

a） 正确

```
import math
radius = eval(input("Enter a number:"))

if radius >= 0:
    area = radius * radius * math.pi
print("The area for the circle of radius", radius, "is", area)
```

b） 错误

因为（i>j）是 false，所以 c）中的代码会显示 B，但是 d）中的语句什么也不显示。

```
i = eval(input("i="))
j = eval(input("j="))
k = eval(input("k="))

if i > j:
    if i > k:
        print('A')
else:
    print('B')
```

c）

```
i = eval(input("i="))
j = eval(input("j="))
k = eval(input("k="))

if i > j:
    if i > k:
        print('A')
    else:
    print('B')
```

d）

4.6 逻辑运算符

逻辑运算符 not、and 和 or 都可以用来创建一个组合条件。

运算符	描述
not	逻辑否
and	逻辑和
or	逻辑或

not 运算符：对 True 取反得 False，对 False 取反得 True。

and 运算符：当且仅当两个操作数都为真时，两个操作数的 and 操作为真。

or 运算符：至少有一个操作数为真时，两个操作数的 or 操作结果才为真。

```
number = eval(input("Enter an integer:"))

if number % 2 == 0 and number % 3 == 0:
    print(number, "is divisible by 2 and 3")

if number % 2 == 0 or number % 3 == 0:
    print(number, "is divisible by 2 or 3")

if(number % 2 == 0 or number % 3 == 0) and \
    not(number % 2 == 0 and number % 3 == 0):
    print(number, "is divisible by 2 or 3,but not both")
```

```
Enter an integer:18
18 is divisible by 2 and 3
18 is divisible by 2 or 3
```

4.7 条件表达式

条件表达式是根据某个条件计算一个表达式。

句法结构如下：

expression1 if boolean-expression else expression2

如果布尔逻辑表达式为真，那么这个条件表达式的结果就是 expression1；否则，这个结果就是 expression2。

```
number = eval(input("Enter a integer:"))
print("number is even" if number % 2 == 0 else "number is odd")
```

CHAPTER5 循环

5.1 while 循环

while 循环是一种条件控制循环，它是根据一个条件的真假控制的，当一个条件保持为真时 while 循环重复执行语句。

while 循环的语法是：

while loop-continuation-condition（条件语句）：**（标点符号是必不可缺的）**

statement(s)（命令）

如果 while 后面的条件语句为 false 时，则执行 else 的语句块。语法格式如下：

```
while loop-continuation-condition:
statement(s)
else:
additional_statement(s)
```

“program is fun! ”一百次的循环是一个 while 循环的例子：

```
count = 0
while count < 100:  # while 进入循环
    print("program is fun!")
    count = count + 1
```

注意**循环的次数**，若被错误地写成如下所示：

```
count = 0
while count <= 100:
    print("program is fun!")
    count = count + 1
```

这个循环显示“program is fun! ”101 次而不是 100 次。

求 1 到 9 的和是另一个例子：

```
sum = 0
i = 1
while i < 10:
    sum = sum + i
    i = i + 1
print("sum is", sum)
```

注意**整个循环体必须被内缩进到循环内部**，若被错误地写成如下所示：

```
sum = 0
i = 1
while i < 10:
    sum = sum + i
i = i + 1
print("sum is", sum)
```

这里的语句 i=i+1 不在循环体内，i 一直是 1，总满足 i<10，这是一个无限循环。

5.2 for 循环

Python 的 for 循环用来依次取出序列中的内容。

通常，for 循环的语法是：

```
for variable in sequence（变量在一定范围内进行迭代）：
    statement(s)
```

例如：

```
languages = ["C", "C++", "Perl", "Python"]
for x in languages:
    print(x)
```

输出结果：

C

C++

Perl

Python

5.3 range（）函数

Range（a，b）函数返回一系列连续**整数** a、a+1、・・・、b-2 和 b-1，b 是界限值，序列中最后一个数必须小于 b，range（a）与 range（0，a）功能一样，在 range（a，b，k）中 k 被用作步长值，例如：

```
for v in range(3, 9, 2):
    print(v)
```

输出结果：

3

5

7

若 range（a，b，k）中的 k 为负数，则可以反向计数，最后一个数必须大于 b。例如：

```
for v in range(5, 1, -1):
    print(v)
```

输出结果：

5

4

3

2

5.4 嵌套循环

一个循环可以嵌套到另一个循环里。

嵌套循环是由一个外层循环和一个或多个内层循环构成。每次重复外层循环时，内层循环都被重新进入并且重新开始。

```
for i in range(1, 3):
    for j in range(1, 4):
        print((i, j))
```

输出结果：

(1, 1)

(1, 2)

(1, 3)

(2, 1)

(2, 2)

(2, 3)

5.5 关键字 break 和 continue

关键字 break 和 continue 提供了另一种控制循环的方式。

我们可以在循环中使用关键字 break 来立即终止循环。break 只跳出最内层循环，无论输入多少个 break，都是一样的效果。如下：

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('循环结束。')
```

输出的结果为：

4

3

循环结束。

关键词 continue 会终止当前的迭代并控制程序转到循环体的最后。如下：

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        continue
    print(n)
print('循环结束。')
```

输出的结果为：

4

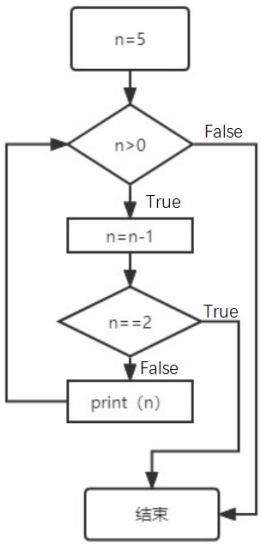
3

1

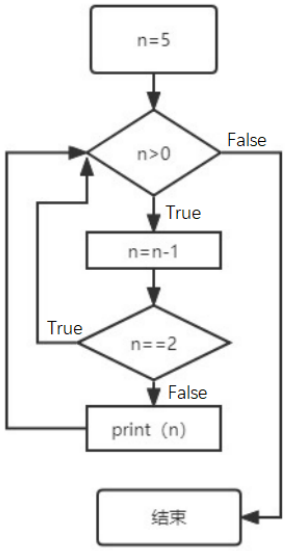
0

循环结束。

上述 break 执行流程图：



上述 continue 执行流程图：



5.6 pass 语句

Python 中 pass 是空语句，是为了保持程序结构的完整性。
当它被执行时，什么都不发生，一般用做占位语句。它适合当语法上需要一条语句但并不需要执行任何代码时用来临时占位在编写代码时只写框架思路，具体实现还未编写就可以用 pass 进行占位，使程序不报错，不会进行任何操作。

例如：以下代码若不写 pass 会发生 IndentationError。

```
for i in range(5):
    if i == 3:
        pass  # 此处必须写 pass
else:
        print(i, "is not 3")
```

CHAPTER6 列表

6.1 列表

列表是 Python 中最基本的**数据结构**。我们可以在一个列表中存放一系列的数字、字符串这些 Python 中的基本数据类型。

要创建一个列表，我们用方括号 [ ] 将要存放的元素括起来，如下：

```
newList = ['String', 123, False, [4, 5, 6]]
```

注意，由于列表本身也是基本数据类型之一，列表也可以作为列表中的项。如上述代码中列表 newList 的第四项为一个存放了三个数字的列表。

6.2 索引

列表中的每一项元素都有对应的**索引**，来表示这些元素在列表中的位置。其中第一项的索引是 **0**，第二项的索引是 **1**，依此类推。对 4.1 中列表 newList 我们可得：

对应值	'String'	123	False	[4, 5, 6]
索引	0	1	2	3

要访问列表中的一项，我们可以使用列表名[索引]。如要打印 newList 的第一项：

```
newList = ['String', 123, False, [4, 5, 6]]
print(newList[0])
```

考虑到 newList 中有列表作为元素的情况，则对于子列表 **newList[3]**，如要输出其中的元素 **4**：

```
newList = ['String', 123, False, [4, 5, 6]]
print(newList[3][0])
```

当索引值为负数时，从列表的最后开始选取元素。其中列表最后一项的负数索引值为**-1**，倒数第二项为**-2**，依此类推。

对应值	'String'	123	False	[4, 5, 6]
负数索引	-4	-3	-2	-1

6.3 对列表进行切片

对列表进行**切片**，换言之就是取出原有列表中的一部分，形成一个**新的列表**。
当我们要选取原有列表中连续的一段时，我们可以使用列表名[起始索引值:终止索引值]。其中终止索引值对应的元素**不会**计入新列表。如下：

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
my_numbers = numbers[2:6]
print(my_numbers)
```

输出结果即新列表 my_numbers 中的所有项，为**[2, 3, 4, 5]**。它们是列表 numbers 中索引值从 2 到 5 的所有项，而在新列表中的索引值依次为从 0 到 3。

对应值	0	1	2	3	4	5	6	7
numbers 中的索引	0	1	2	3	4	5	6	7
my_numbers 中的索引			0	1	2	3		

当我们要取出索引值从 2 开始的**所有项**时，我们可以将终止索引值**省略**。如下：

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
my_numbers = numbers[2:]
print(my_numbers)
```

输出结果为[2, 3, 4, 5, 6, 7]。

类似地，当我们要取出索引值小于 6 的所有项时，我们可以将起始索引值省略。如下：

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
my_numbers = numbers[:6]
print(my_numbers)
```

输出结果为[0, 1, 2, 3, 4, 5]。

当起始索引值、终止索引值均省略时，我们能得到一个与原列表一模一样的副本。如下：

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
my_numbers = numbers[:]
print(my_numbers)
```

输出结果为[0, 1, 2, 3, 4, 5, 6, 7]，与原列表相同。

当我们要从原有列表中每隔一个选一个、或者隔两个以上选一个时，我们可以加上步长参数，即列表名[起始索引值:终止索引值:步长]。步长即取出的相邻两个元素在原列表中索引值的差。若我们要取出 numbers 中所有索引值为奇数的值时：

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
my_numbers = numbers[1::2]
print(my_numbers)
```

输出结果为[1, 3, 5, 7]。

对应值	0	1	2	3	4	5	6	7
numbers 中的索引	0	1	2	3	4	5	6	7
my_numbers 中的索引	＼	0	＼	1	＼	2	＼	3

利用步长值为负数的情况，我们可以将原有列表倒序获得新列表。如下：

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
my_numbers = numbers[::-1]
print(my_numbers)
```

输出结果为[7, 6, 5, 4, 3, 2, 1, 0]。注意步长值为 0 时，程序报错。

6.4 列表的加法与乘法

Python 中定义了列表的**加法**和列表与整数之间的**乘法**。

当两个列表相加时，我们能得到两个列表**拼接**而成的新列表。如下：

```
first = [1, 2, 3]
second = [4, 5, 6]
whole = first + second
print(whole)
```

输出结果即列表 first 和 second 拼接而成的新列表 whole，为**[1, 2, 3, 4, 5, 6]**。

当一个列表乘以一个整数 n 时，我们能得到这个列表**重复 n 次**而成的新列表。如下：

```
zeros = [0] * 10
print(zeros)
```

输出结果即包含 10 个整数 0 的列表 zeros，为[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]。

6.5 在列表上的循环

在 Python 的 for 循环中，我们需要提供一个**可迭代对象**进行遍历。由于列表也是一个可迭代对象，如果要输出一个列表中的一些项，我们可以将其直接写进 for 循环语句：

```
items = ['apple', 'banana', 'car', 'dog']
for item in items[:2]:
        print(item)
```

输出结果即列表 items 中索引值**小于 2** 的项，为 items 的前两项。
输出结果是：
apple
banana

当我们想把任意一个可迭代对象转化为列表时，我们可以使用 Python 提供的 list 函数。这个函数需要一个可迭代对象作为参数，形如 list(可迭代对象)。如下：

```
numbers = list(range(10))
print(numbers)
```

输出结果为[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]。

6.6 常用列表函数

Python 常用以下列表函数：

len(列表名)	返回列表元素个数
max(列表名)	返回列表元素最大值
min(列表名)	返回列表元素最小值
sorted(列表名)	返回原列表排序后的副本

如当我们想要得到一个列表的上述信息时：

```
events = [1917, 1848, 1789, 1871, 1524, 1861]
print(f'Count:{len(events)} First:{min(events)} Last:{max(events)}')
print(sorted(events))
```

输出结果共两行，第一行为字符串 Count:6 First:1524 Last:1917。第二行为排序后的列表[1524, 1789, 1848, 1861, 1871, 1917]。

6.7 常用列表方法

Python 常用以下列表方法：

增加	列表名.append(元素)	在列表末尾添加新的元素，无返回值
	列表名.insert(索引，元素)	在所给的索引处添加新的元素，无返回值
删除	列表名.pop(索引)	删除索引处的元素(默认最后一个)，返回该元素
	列表名.remove(元素)	删除列表中第一个所给的元素，无返回值
	列表名.clear()	清空列表，无返回值
查找	列表名.count(元素)	返回列表中所给的元素出现的次数
	列表名.index(元素)	返回列表中第一个所给的元素的索引
排序	列表名.sort(元素)	将原列表排序，无返回值

6.8 常用列表关键词

当我们想得知列表中是否含有某个元素时，我们可以使用关键词 **in** 和 **not in**。当列表中含有所给的元素时，表达式 **元素** in **列表名** 返回 **True**，否则为 **False**。关键词 **not in** 的使用方法类似，返回值的情况则相反。如当我们想得知列表中是否含有某个元素及其索引时：

```
events = [1917, 1848, 1789, 1871, 1524, 1861]
if 1848 in events:
    print(events.index(1848))
```

输出结果即 1848 的索引值，为 1。用上述办法可以避免若元素不在列表中时，使用 index 方法导致程序报错。

当我们要删除列表中的一部分时，我们可以使用关键词 **del**，并用类似列表切片的办法实现。**del 列表切片** 将把切片选取到的内容从原列表中删除。如下：

```
events = [1917, 1848, 1789, 1871, 1524, 1861]
del events[3:]
print(events)
```

输出结果即从列表 events 中删去索引值不小于 3 的所有元素，为**[1917, 1848, 1789]**。通过和列表切片类似的形式，我们能以固定的步长删去元素。

CHAPTER7 函数

7.1 函数定义与调用

函数在 Python 中就是把具有独立功能的代码块组织成为一个小模块，在需要的时候调用。
定义函数的格式为：Def + 函数名称 +（变量名称）：（不要忘记反括号后的冒号）

```
def cal(a, b): # 函数的创建
    sum = a + b
    return sum

ret = cal(99, 89) # 函数的调用
print(ret)
```

函数分为：
定义函数 - 具有独立封装的功能（需要自定义）
函数包括函数头和函数体。函数头以一个 def 关键字开始，后面紧接着函数名以及形参**并以冒号结束**。
函数头中的参数被称为形式参数（简称形参）。
调用函数 - 享受封装的成果（可以直接调用 Python 里的函数，如上图）
参数就像一个占位符：当调用函数时，就将一个值传递给参数，这个值被称为实际参数（简称实参）。

7.2 return

返回值是函数完成工作后，最后给调用者的一个结果，使用 return 关键字可以返回结果。

```
def cal(a, b):
    sum = a + b
    return sum
```

7.3 可更改与不可更改对象

在 Python 中，数据类型分为 mutable 和 immutable。
strings, tuples 和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的。
不可更改的变量：如函数下面的变量的值更改后 例如 fun(a)，传递的只是 a 的值。在 fun(a) 内部修改 a 的值，不会影响 a 本身。

```
def fun(a):
    a = 10
b = 2
fun(b)
print (b)
```

输出结果：
2

7.4 默认值参数

Python 中给形参设置默认值，只有与默认值不符的时候才需要传实参。例如：b 为默认值参数

```
def fun(a,b=10):
    print(a,b)
# 函数的调用
fun(100) # 只传一个参数 100 给 a，b 采用默认值 10.
fun(20,30)
```

输出结果：

100 10

20 30

（只传一个参数，a=100，b 用默认值 b=10。所以结果为 100，10

30 将默认值 10 替换。结果为 20，30）

注意：**具有默认值的形参最好放在参数列表的靠后，也就是最右边。**

```
def cal(a=10, b):
    sum = a + b
    return sum

ret = cal(3)
print(ret) # 报错
```

因为第 1 个实参总是对应第 1 个形参，所以，3 赋值给 a，替换掉默认值 10，参数 b 没有传值，最终函数调用报错。

```
def cal(a,b=10):
    sum = a + b
    return sum

ret = cal(3)
print(ret)
```

输出结果：

13

7.5 位置实参和关键词实参

当调用函数时，需要将实参传递给形参。实参有两种类型：位置参数和关键词参数。

位置实参:根据形参的位置进行实参传递。

```
def cal(a, b): # a,b 称为形式参数，形参的位置是在函数的定义处
# a,b 在括号里没有具体的值
    sum = a + b
    return sum

sum = cal(10, 15) # 10,15 称为实际参数的值，实参的位置也是函数的调用处
print(sum)
```

输出结果：

25

（相当于 a=10，b=15。）

关键字实参：根据形参名称进行实参传递。

```
def cale(a, b):
    c = a + b
    return c

ret = cale(b=77, a=99) # = 左侧的变量名称称为关键字参数
print(ret)
```

输出结果：

176

输出结果不再按照位置一一对应，而是根据关键字进行实参传递。例如：a 为形参名称，99 为实参值。

7.6 不定长参数

不定长参数分为个数可变的位置函数和个数可变的关键字形参。

个数可变的位置参数：定义函数时可能无法事先确定传递的位置实参的个数，可以使用可变的位置参数，即*定义个数可变的位置形参，结果是一个元组。

```
def fun(*zzz): # zzz 为参数名称
    print(zzz)
fun(10)
fun(10,99)
fun(56,45,78)
```

输出结果：

(10,)

(10, 99)

(56, 45, 78)

个数可变的关键字形参：

定义函数时无法事先确定传递的关键字实参的个数时可以使用可变的关键字形参，即**定义个数可变的关键字形参，结果为一个字典。

```
def fun(**zzz):
    print(zzz)
fun(a=52)
fun(a=56,b=111)
```

输出结果：

{'a': 52}

{'a': 56, 'b': 111}

注意：无论是个数可变的位置参数还是个数可变的关键字参数都只能定义一个，否则程序会报错。

```
def fun1(*zzz,*yyy):
    #程序报错
def fun2(*zzz,**yyy):
    pass
def fun3(**yyy,*zzz)
    #程序报错
```

在一个函数的定义中，如果想要既有可变的关键字形参又有可变的位置形参，就要将个数可变的位置形参放在个数可变的关键字形参之前。

制作人员

• 参与制作名单 •

第一章：田宜麟 李彦妮
第二章：郭冠麟 王静
第三章：郭冠麟 丁雅清
第四章：王一攀
第五章：王一攀
第六章：杨柏翰
第七章：黄欣怡

• 编审人员名单 •

李明翀 徐家嵘 王一攀 杨柏翰

关注微信公众号

获取更多资讯