# python开发准备

## 1.python简介



2. 免费,开源

3. 应用领域广泛

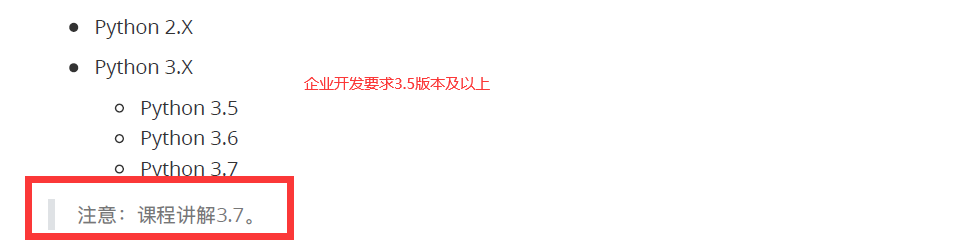
备注：以下知名框架均是Python语⾔言开发

Google开源机器学习框架：TensorFlow

开源社区主推学习框架：Scikit-learn

百度开源深度学习框架：Paddle

Python发展历史：<https://baike.baidu.com/item/Python/407313?fr=aladdin>



## 2. 解释器的作用

Python解释器作用：运行文件

Python解释器种类

CPython，C语言开发的解释器[官方]，应用广泛的解释器。

IPython，基于CPython的一种交互式解释器。

其他解释器:

PyPy，基于Python语言开发的解释器。

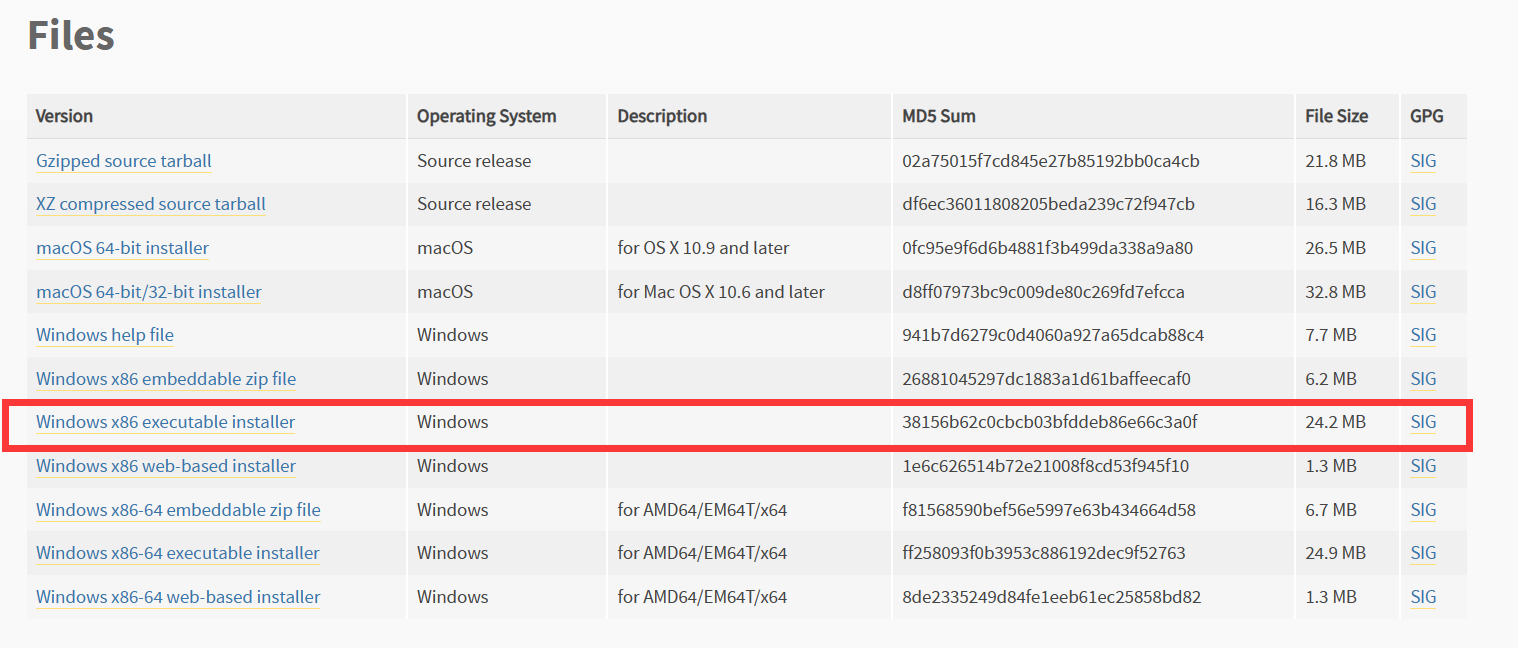
Jython，运行在Java平台的解释器，直接把Python代码编译成Java字节码执行。

IronPython，运行在微软.Net平台上的Python解释器，直接把Python代码编译成.Net的字节码。

### 2.1 下载Python解释器

下载地址：<https://www.python.org/downloads/release/python-372/>

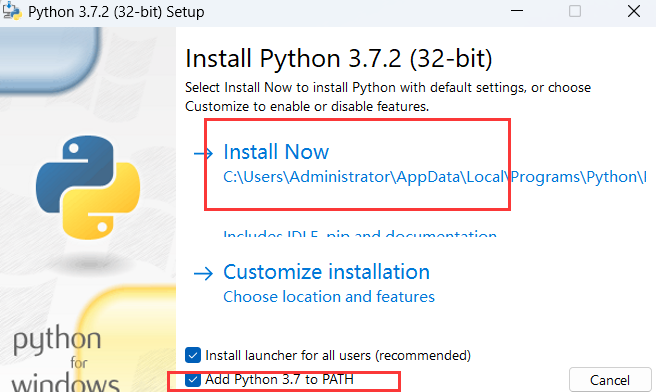
[单击上述链接] -- 查找目标文件：Windows x86-64 executable installer -- 单击即可下载

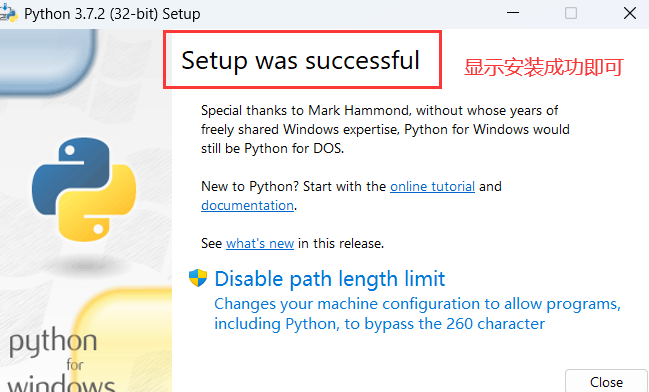


### 2.2 安装Python解释器

双击可执行文件 — 勾选[pip] -- [Next] -- [勾选添加环境变量] -- [Install]，按提示操作

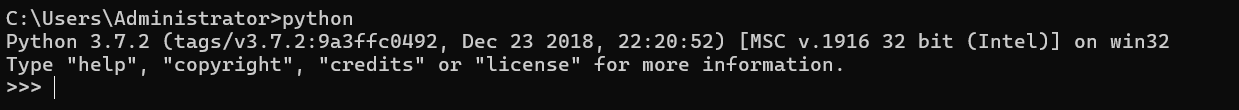






### 2.3 检验安装

Win+r打开cmd命令窗口，输入python，进入到解释器页面即为安装成功



**注**：如有多版本，进入的是2开头的版本，可输入exit()命令退出解释权窗口





然后输入python3（为文件的名字）重新进入





## 3.编辑器PyCharm

### 3.1 pycharm的作用

PyCharm是一种Python IDE（集成开发环境），带有一整套可以帮助用户在使⽤用Python语言开发时

提高其效率的工具，内部集成的功能如下：

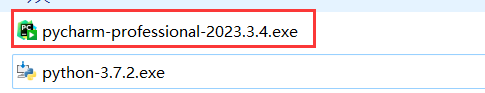
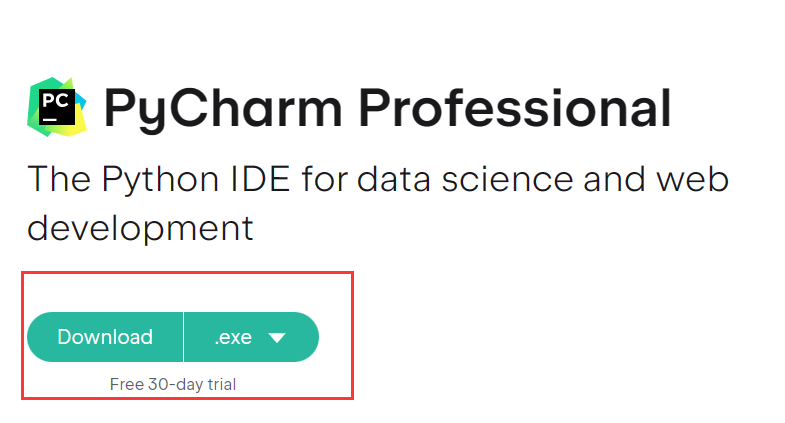
Project（项目）管理 智能提示 语法高亮 代码跳转 调试代码 解释代码(解释器)

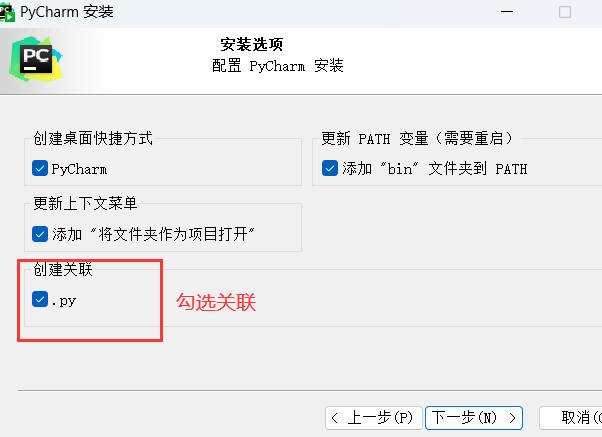
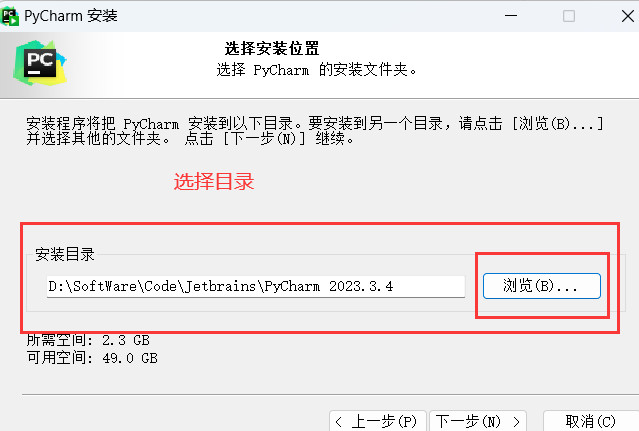
框架和库（社区版没有）

### 3.2 下载pycharm

PythonCharm分为专业版（professional）和社区版（community）

下载地址：<http://www.jetbrains.com/pycharm/download/#section=windows>

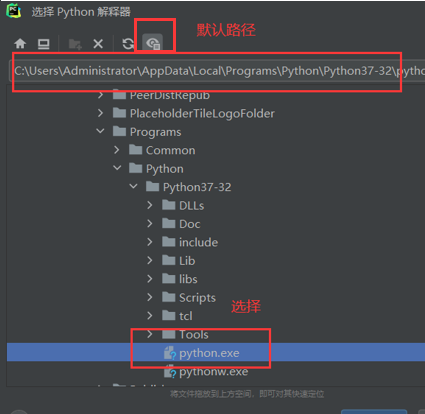
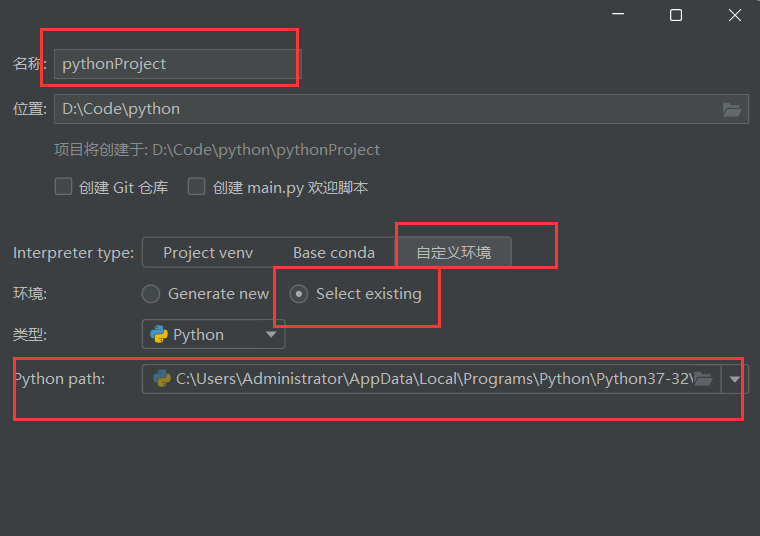




### 3.3 pycharm的使用

#### 3.3.1 新建项目

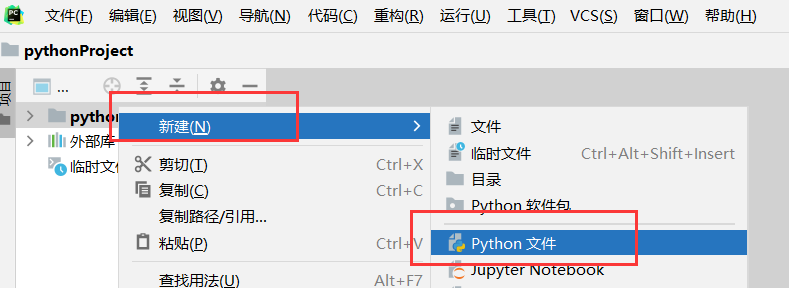
打开PyCharm -- [Create New Project] -- 选择项目根目录和解释器版本 -- [Create]，即可完成新建一个项目



#### 3.3.2 新建文件并书写代码

项目根目录或根目录内部任意位置 — 右键 -- [New] -- [Python File] -- 输入文件名 -- [OK]

注：如果是将来要上传到服务器的文件，那么文件名切记不能用中文

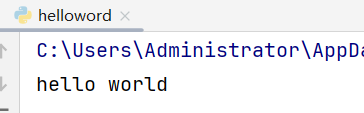
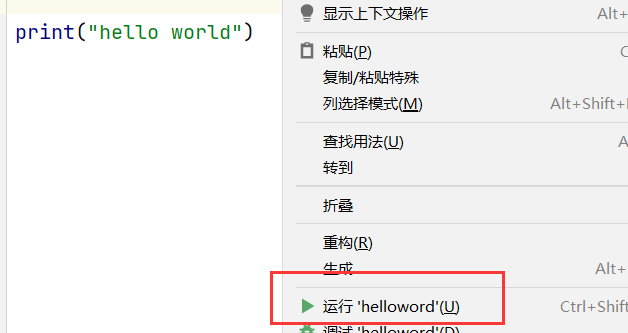


双击打开文件，并书写一个最简单的Python代码：



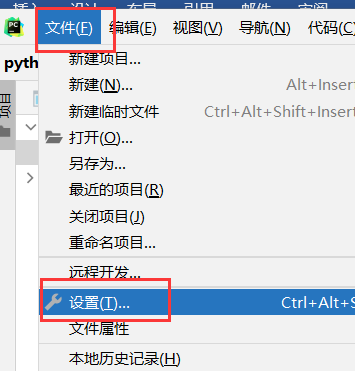
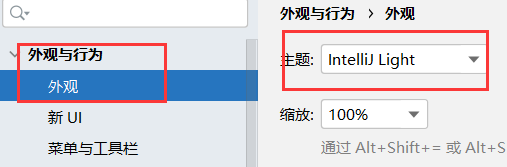
#### 3.3.3 运行文件

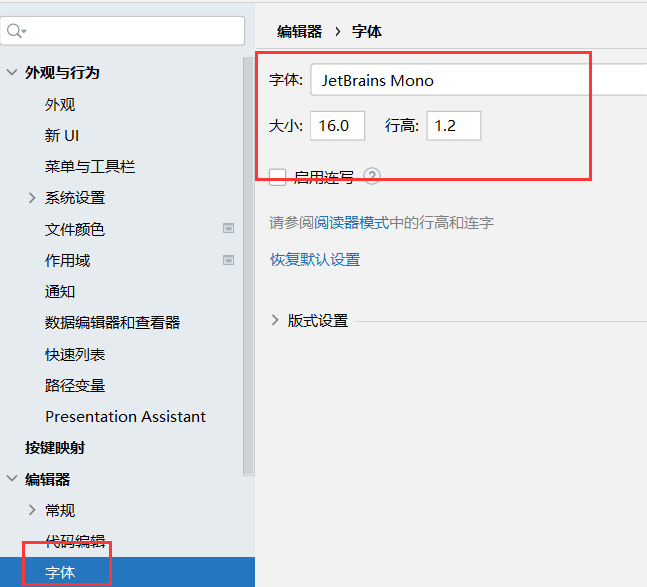
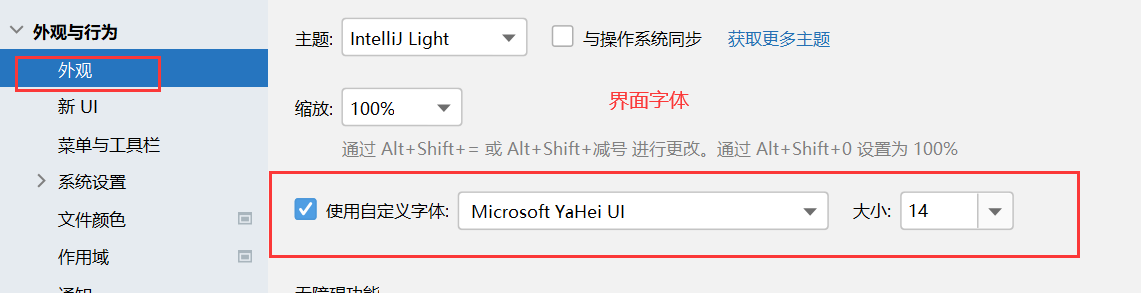
文件打开状态 -- 空白位置 -- 右键 -- Run -- 即可调出Pycharm的控制台输出程序结果。



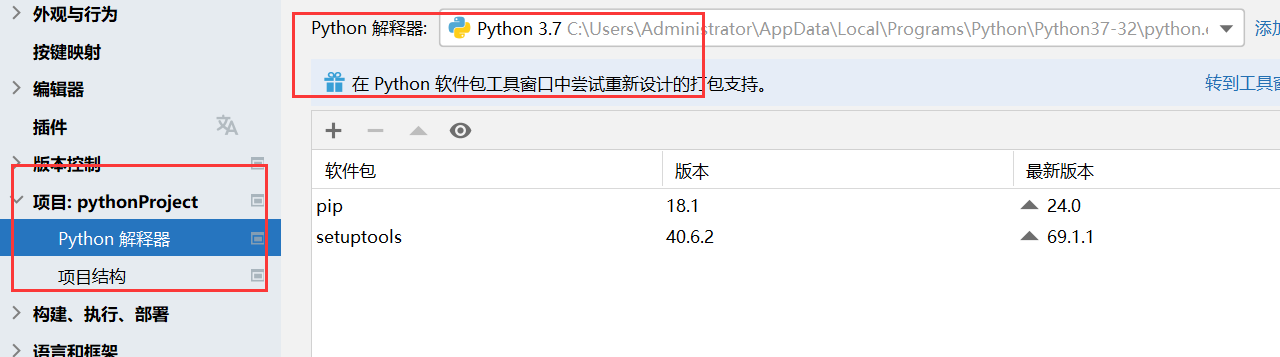
#### 3.3.4 pycharm的基本设置

[file] -- [Settings]/[Default Settings]

[Project: 项目名称] -- [Project Interpreter] -- [设置图标] -- [Add] -- 浏览到目标解释器

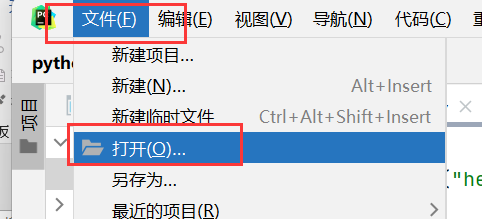
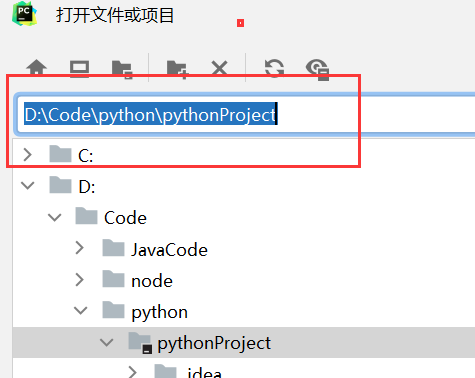


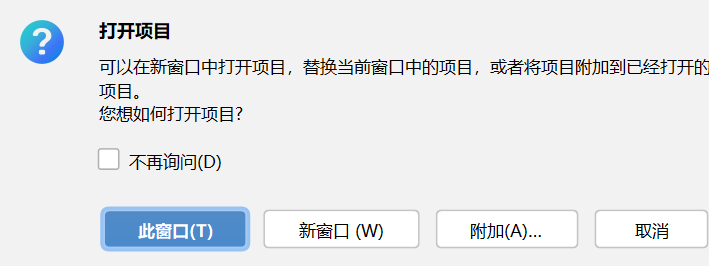
#### 3.3.5 项目管理

**打开项目**

[File] -- [Open] -- 浏览选择目标项目根目录 -- [OK] -- 选择打开项目方式。

打开项目的方式共三种，分别如下：

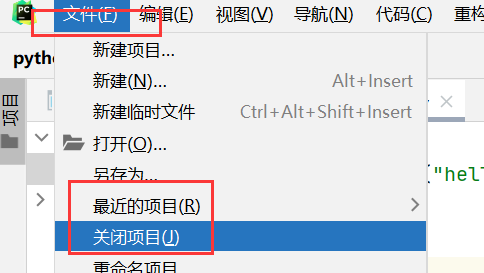


This Window（此窗口）覆盖当前项目，从而打开目标项目

New Window（新窗口）在新窗口打开，则打开两次PyCharm，每个PyCharm负责一个项目

Attach（附加）将项目添加到已打开的项目当中

**关闭项目**



### 4. 注释

通过用自己熟悉的语言，在程序中对某些代码进行标注说明，这就是注释的作用，能够大大增强程序的可读性

注释分为两类：单行注释 和 多行注释

快捷键为：CTRL+/

CTRL+SHIFT+/（多行）

单行注释：只能注释一行内容，语法如下：

# 注释内容

多行注释：可以注释多行内容，一般用在注释一段代码的情况

""" '''

第一行注释 注释1

第二行注释 注释2

第三行注释 注释3

""" '''

# 二．Python基础

## 1.变量

程序中，数据都是临时存储在内存中，为了更快速的查找或使用这个数据，通常我们把这个数据在内存中存储之后定义一个名称，这个名称就是变量

### 1.1 定义变量

变量名 = 值

变量名自定义，要满足标识符命名规则

**标识符：**

标识符命名规则是Python中定义各种名字的时候的统一规范，具体如下：

* 由数字、字母、下划线组成
* 不能数字开头
* 不能使用内置关键字
* 严格区分大小写

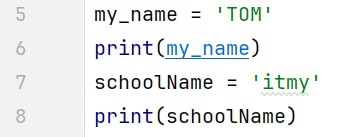
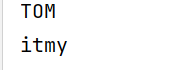
**内置关键字：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| False | None | True | and | as | assert | break | class |
| continue | def | del | elif | else | except | finally | for |
| from | global | if | import | in | is | lambda | nonlocal |
| not | or | pass | raise | return | try | while | with |
| yield |  |  |  |  |  |  |  |

**命名习惯：**

* 见名知义。
* 大驼峰：即每个单词首字母都大写，例如： MyName
* 小驼峰：第二个（含）以后的单词首字母大写，例如： myName
* 下划线：例如： my\_name

### 1.2 使用变量

## 2.bug

所谓bug，就是程序中的错误。如果程序有错误，需要程序员排查问题，纠正错误

### 2.1 Debug工具

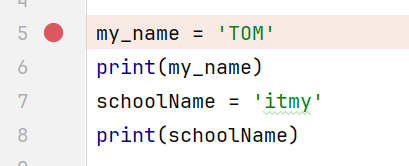
Debug工具是PyCharm IDE中集成的用来调试程序的工具，在这里程序员可以查看程序的执行细节和流程或者调解bug。

Debug工具使用步骤： 1. 打断点 2. Debug调试

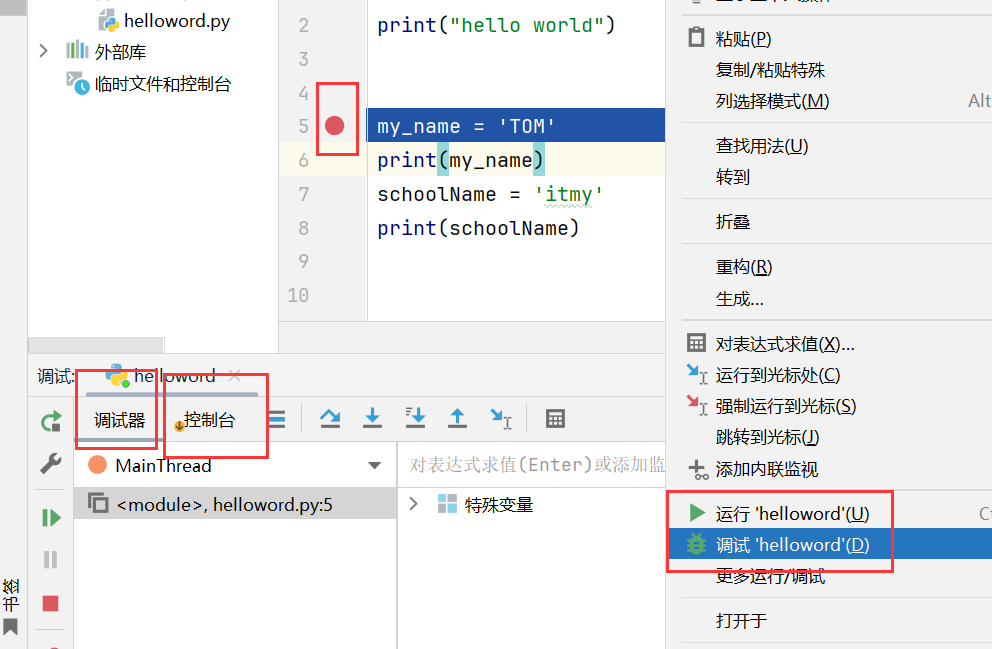
### 2.2 打断点

**断点位置**：目标要调试的代码块的第一行代码即可，即一个断点即可

**打断点的方法**：单击目标代码的行号右侧空白位置



打成功断点后，在文件内部任意位置 -- 右键 -- Debug'文件名' -- 即可调出Debug工具面板 -- 单击StepOver/F8，即可按步执行代码。



**面板输出解释**：

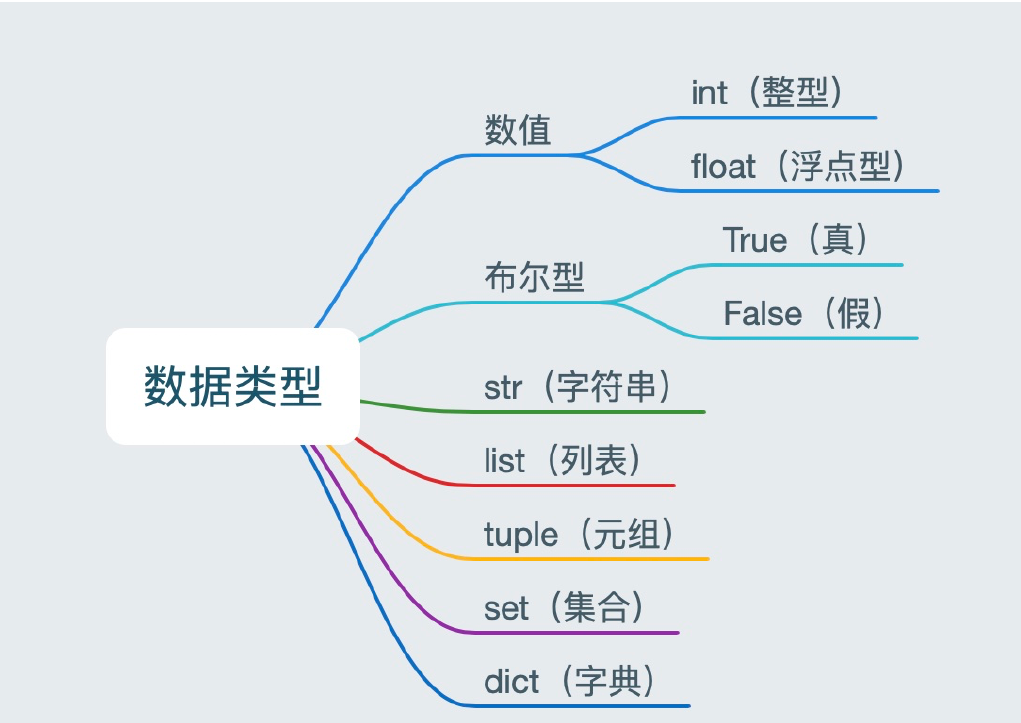
Debugger（调试器）显示程序执行过程的变量和变量的细节

Console（控制台）输出内容

## 3.数据类型

整型：int 浮点型：float 字符串：str 布尔型：bool

元组：tuple 集合：set 字典：dict



### 3.1 检测数据的方法

检测数据类型的方法： type()

a = 1

print(type(a)) # <class 'int'> -- 整型

b = 1.1

print(type(b)) # <class 'float'> -- 浮点型

c = True

print(type(c)) # <class 'bool'> -- 布尔型

d = '12345'

print(type(d)) # <class 'str'> -- 字符串串

e = [10, 20, 30]

print(type(e)) # <class 'list'> -- 列列表

f = (10, 20, 30)

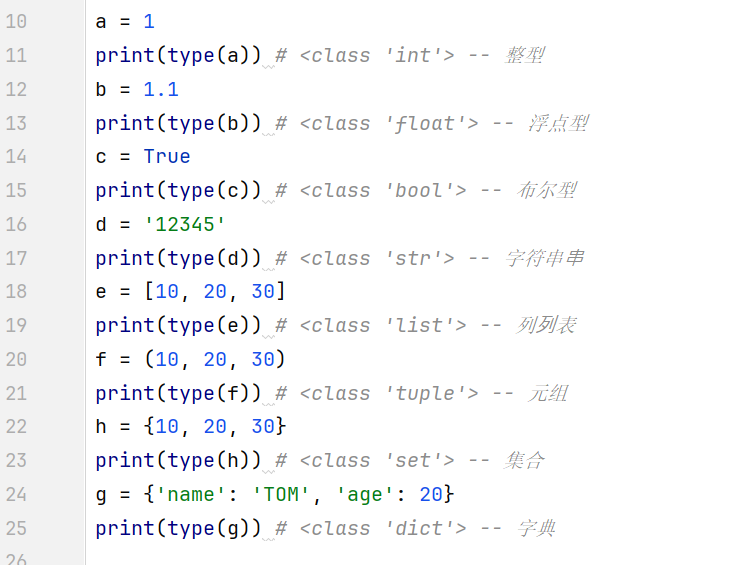
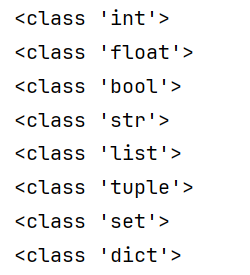
print(type(f)) # <class 'tuple'> -- 元组

h = {10, 20, 30}

print(type(h)) # <class 'set'> -- 集合

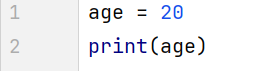
g = {'name': 'TOM', 'age': 20}

print(type(g)) # <class 'dict'> -- 字典

## 4. 输出print

作用：程序输出内容给用户



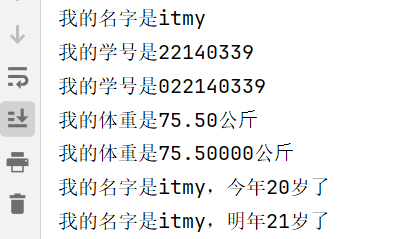
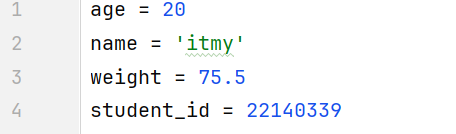
### 4.1 格式化输出

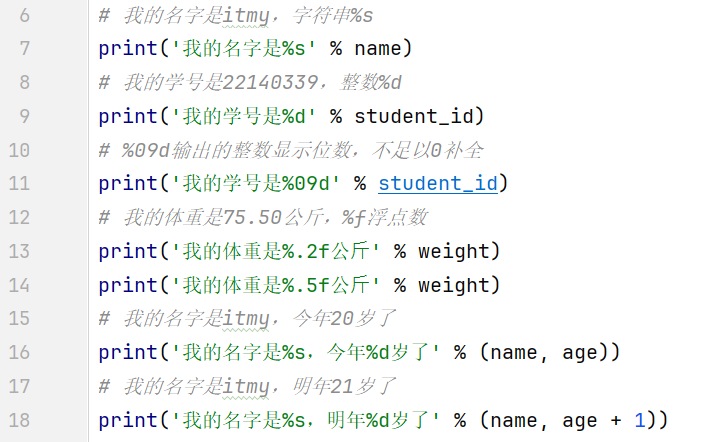
|  |  |
| --- | --- |
| 格式符号转换 | 格式符号转换 |
| %s | %s 字符串 |
| %d | %d 有符号的十进制整数 |
| %f | %f 浮点数 |
| %c | %c 字符 |
| %u | %u 无符号十进制整数 |
| %o | %o 八进制整数 |
| %x | %x 十六进制整数（小写ox） |
| %X | %X 十六进制整数（大写OX） |
| %e | %e 科学计数法（小写'e'） |
| %E | %E 科学计数法（大写'E'） |
| %g | %g %f和%e的简写 |
| %G | %G %f和%E的简写 |

**技巧**：

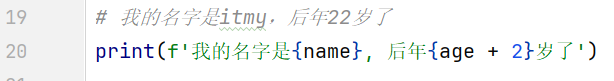
%06d，表示输出的整数显示位数，不足以0补全，超出当前位数则原样输出

%.2f，表示小数点后显示的小数位数。





格式化字符串除了%s，还可以写为f'{表达式}'

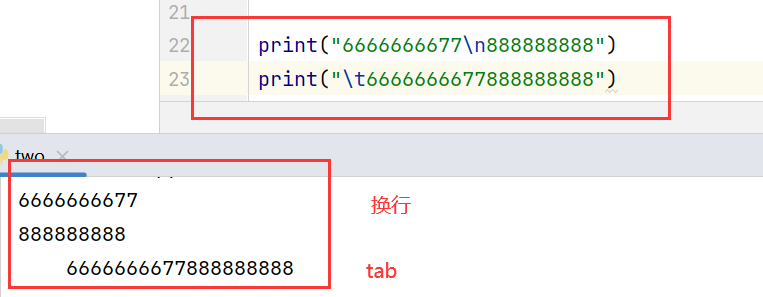




### 4.2 转义字符

\n ：换行

\t ：制表符，一个tab键（4个空格）的距离



### 4.3 结束符

在Python中，print()， 默认自带end="\n" 这个换行结束符，所以导致每两个print 直接会换行

展示，用户可以按需求更改结束符。

print('输出的内容', end="\n")

## 5. 输入input

在Python中，程序接收用户输入的数据的功能即是输入

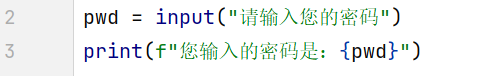
**语法**：input("提示信息")

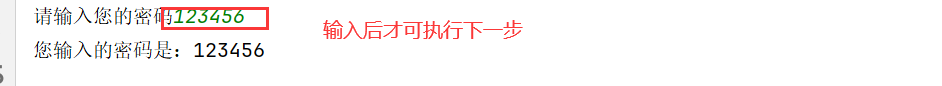
**特点**：

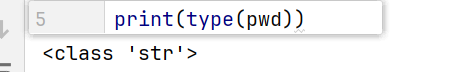
当程序执行到input ，等待用户输入，输入完成之后才继续向下执行。

在python中， input 接收用户输入后，一般存储到变量，方便使用。

在Python中， input 会把接收到的任意用户输入的数据都当做字符串处理。

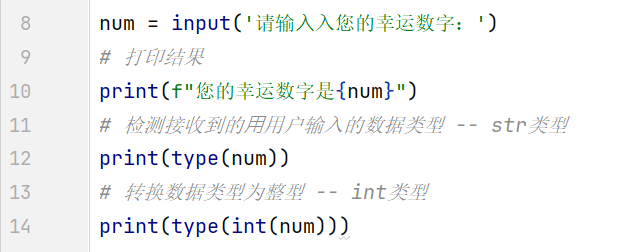


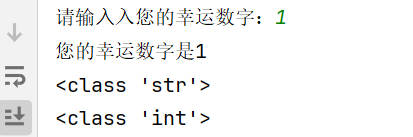




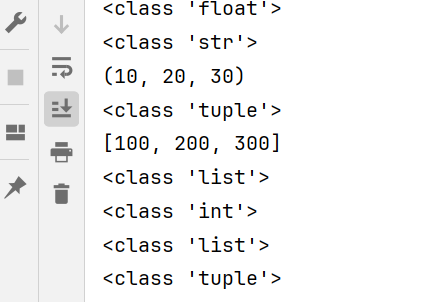
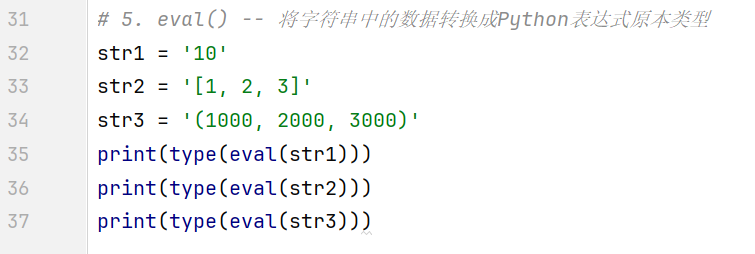
## 6. 转换数据类型

|  |  |
| --- | --- |
| 函数 | 说明 |
| int(x [,base ]) | 将x转换为一个整数 |
| float(x ) | 将x转换为一个浮点数 |
| complex(real [,imag ]) | 创建一个复数，real为实部，imag为虚部 |
| str(x ) | 将对象 x 转换为字符串 |
| repr(x ) | repr(x ) 将对象 x 转换为表达式字符串 |
| eval(str ) | 用来计算在字符串中的有效Python表达式,并返回一个对象 |
| tuple(s ) | 将序列 s 转换为一个元组 |
| list(s ) | 将序列 s 转换为一个列表 |
| chr(x ) | 将一个整数转换为一个Unicode字符 |
| ord(x ) | 将一个字符转换为它的ASCII整数值 |
| hex(x ) | 将一个整数转换为一个十六进制字符串 |
| oct(x ) | 将一个整数转换为一八个进制字符串 |
| bin(x ) | 将一个整数转换为一个二进制字符串 |









# 1. float() -- 转换成浮点型

num1 = 1

num11 = float(num1)

print(type(num11))

# 2. str() -- 转换成字符串类型

num2 = 10

print(type(str(num2)))

# 3. tuple() -- 将⼀一个序列转换成元组

list1 = [10, 20, 30]

print(tuple(list1))

print(type(tuple(list1)))

# 4. list() -- 将⼀一个序列转换成列表

t1 = (100, 200, 300)

print(list(t1))

print(type(list(t1)))

# 5. eval() -- 将字符串中的数据转换成Python表达式原本类型

str1 = '10'

str2 = '[1, 2, 3]'

str3 = '(1000, 2000, 3000)'

print(type(eval(str1)))

print(type(eval(str2)))

print(type(eval(str3)))

## 7.运算符

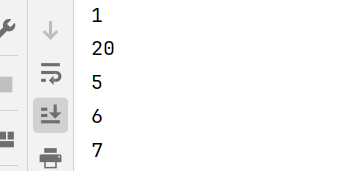
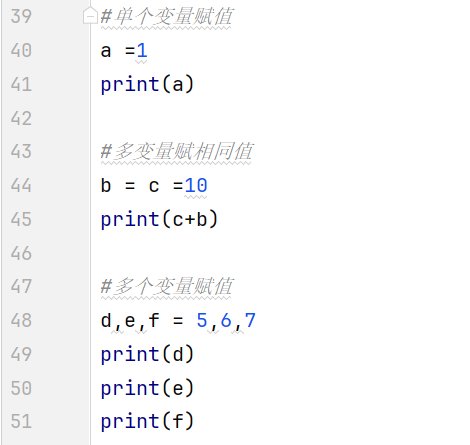
### 7.1 算术运算符

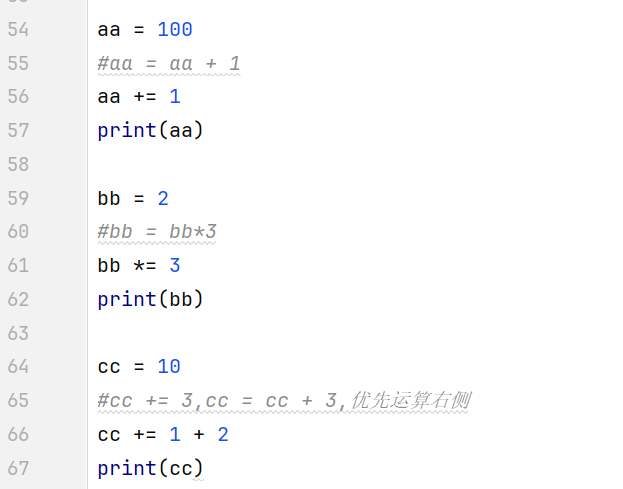
|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| + | 加 | 1 + 1 输出结果为 2 |
| - | 减 | 1-1 输出结果为 0 |
| \* | 乘 | 2 \* 2 输出结果为 4 |
| / | 除 | 10 / 2 输出结果为 5 |
| // | 整除 | 整除9 // 4 输出结果为2 |
| % | 取余 | 9 % 4 输出结果为 1 |
| \*\* | 指数 | 2 \*\* 4 输出结果为 16，即 2 \* 2 \* 2 \* 2 |
| () | 小括号 | 小括号用来提高运算优先级，即 (1 + 2) \* 3 输出结果为 9 |

注：混合运算优先级顺序： () 高于 \*\* 高于 \* / // % 高于 + -

### 7.2 赋值运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| = | 赋值 | 将=右侧的结果赋值给等号左侧的变量 |
| += | 加法赋值运算符 | c += a 等价于 c = c + a |
| -= | 减法赋值运算符 | c -= a 等价于 c = c- a |
| \*= | 乘法赋值运算符 | c \*= a 等价于 c = c \* a |
| /= | 除法赋值运算符 | c /= a 等价于 c = c / a，除法得到的结果一定是小数 |
| //= | 整除赋值运算符 | c //= a 等价于 c = c // a |
| %= | 取余赋值运算符 | c %= a 等价于 c = c % a |
| \*\*= | 幂赋值运算符 | c \*\* = a 等价于 c = c \*\* a |

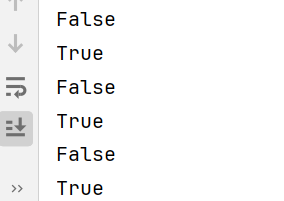
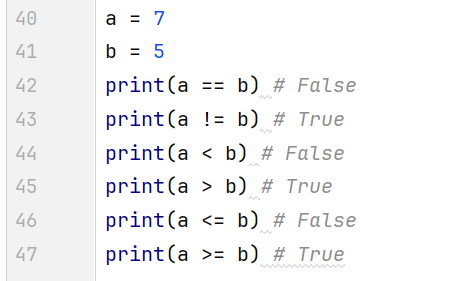




### 7.3 比较运算符

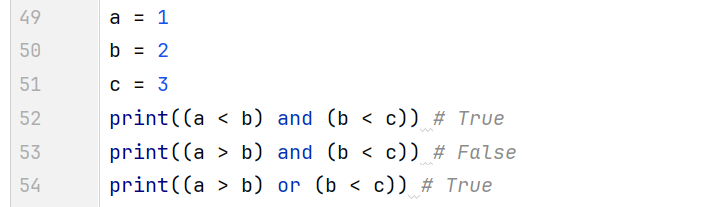
比较运算符也叫关系运算符， 通常用来判断。

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| == | 判断相等。如果两个操作数的结果相等  则条件结果为真(True)，否则条件结果为假(False) | 如a=3,b=3  则（a == b) 为 True |
| ！= | 不等于 。如果两个操作数的结果不相等  则条件为真(True)，否则条件结果为假(False) | 如a=3,b=3则(a == b)为 True  如a=1,b=3则(a != b) 为 True |
| > | 运算符左侧操作数结果是否大于右侧操作数结果，  如果大于，则条件为真，否则为假 | 如a=7,b=3  则(a > b) 为 True |
| < | 运算符左侧操作数结果是否小于右侧操作数结果，  如果小于，则条件为真，否则为假 | 如a=7,b=3  则(a < b) 为 False |
| >= | 运算符左侧操作数结果是否大于等于右侧操作数结  果，如果大于，则条件为真，否则为假 | 如a=7,b=3则(a < b)为 False  如a=3,b=3，则(a >= b) 为 True |
| <= | 运算符左侧操作数结果是否小于等于右侧操作数结  果，如果小于，则条件为真，否则为假 | 如a=3,b=3  则(a <= b) 为 True |



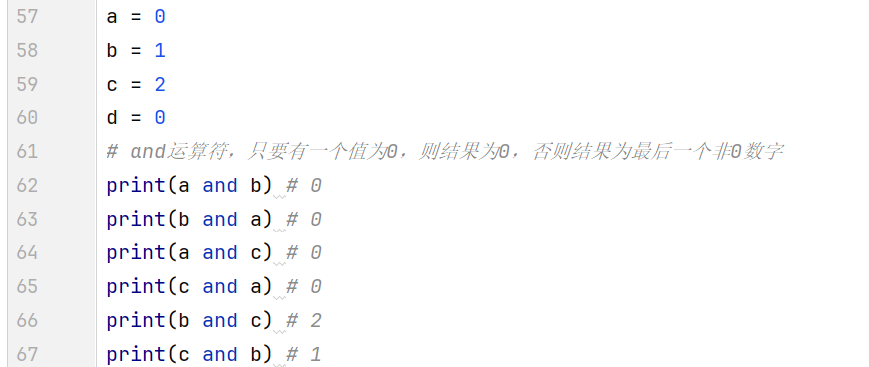
### 7.4 逻辑运算符

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **逻辑表达式** | **描述** | **实例** |
| And | X and y | 布尔"与"：如果 x 为 False，x and y 返回  False，否则它返回 y 的值 | True and False  返回False |
| Or | X or y | 布尔"或"：如果 x 是 True，它返回 True，否则它返回 y 的值 | False or True  返回True |
| Not | Not x | 布尔"非"：如果 x 为 True，返回 False  如果 x为 False，它返回 True | not True 返回 False not False 返回 True |

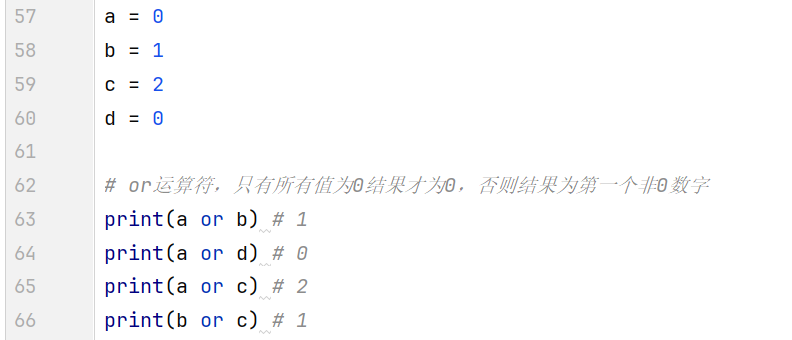


### 7.5 数字之间的逻辑运算（拓展）

and运算符，只要有一个值为0，则结果为0，否则结果为最后一个非0数字



or运算符，只有所有值为0结果才为0，否则结果为第一个非0数字



## 8. 条件语句

条件成立执行某些代码，条件不成立则不执行这些代码

### 8.1 if语句

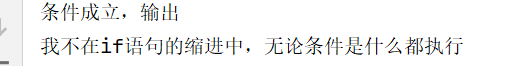
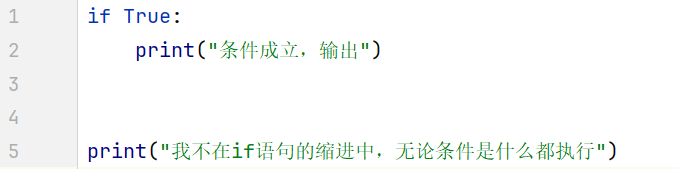
语法：if 条件:

条件成立执行的代码1

条件成立执行的代码2

......（注：只会执行if语句后带缩进的代码）

注：if语句后无缩进的代码不会执行，也代表if语句结束



### 8.2 if…else…语句

条件成立执行if下方的代码; 条件不成立执行else下方的代码

语法：if 条件:

条件成立执行的代码1

条件成立执行的代码2

......

else:

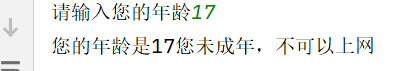
条件不成⽴立执行的代码1

条件不成⽴立执行的代码2

......

#### 案例：输入年龄，判断是否可以上网



### 8.3 多重判断

多重判断可以和else配合使用。

一般else放到整个if语句的最后，表示以上条件都不成立的时候执行的代码

语法：if 条件1:

条件1成立执行的代码1

条件1成立执行的代码2

......

elif 条件2：

条件2成立执行的代码1

条件2成立执行的代码2

......

......

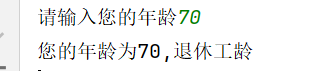
else:

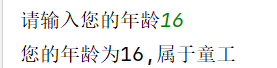
以上条件都不成立执行执行的代码

#### 案例：工龄判断

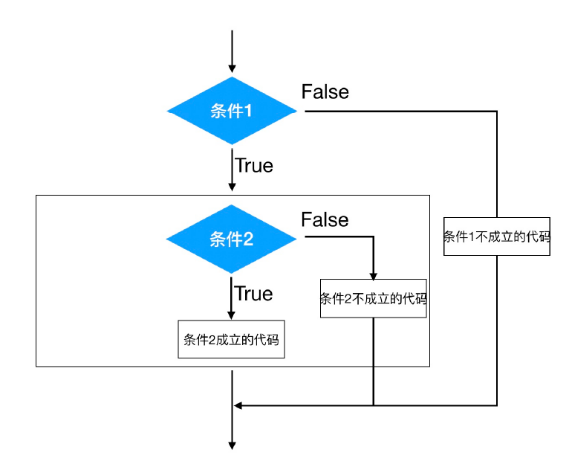
拓展：如：age>18,age<20，可以写成18<age<20





### 8.4 if嵌套循环

语法：if 条件1：

条件1成立执行的代码

条件1成立执行的代码

Else:

if 条件2：

条件2成立执行的代码

条件2成立执行的代码

Else:

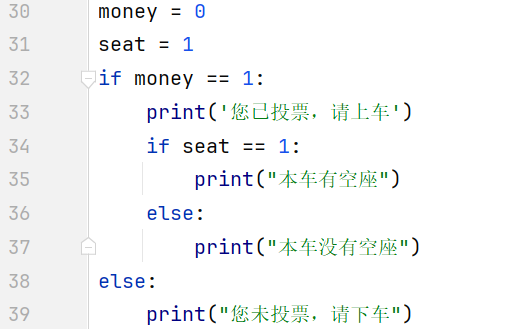
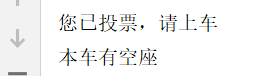
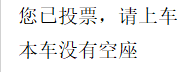
...

Else：

#### 案例：1. 如果有钱，则可以上车，如果没钱，不能上车

2. 上车后，如果有空座，可以坐下，上车后，如果没有空座，则站着等空座位

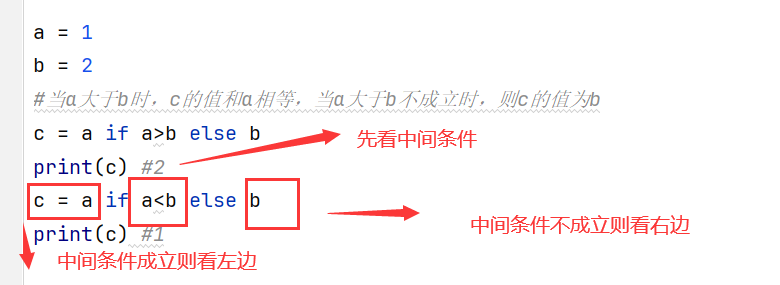
假设用 money = 1 表示有钱, money = 0表示没有钱; seat = 1 表有空座，seat = 0 表没空座

## 9.三目运算符

三目运算符也叫三元运算符或三元表达式

条件成立执行的表达式 if 条件 else 条件 不成立执行的表达式



## 10.循环语句

循环的作用：让代码更高效的重复执行

循环分为while 和for 两种，最终实现效果相同

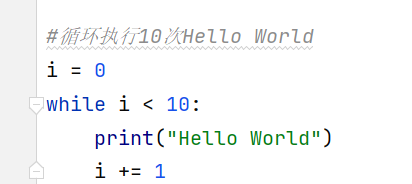
### 10.1 while语句

语法：while 条件:

条件成立重复执行的代码1

条件成立重复执行的代码2

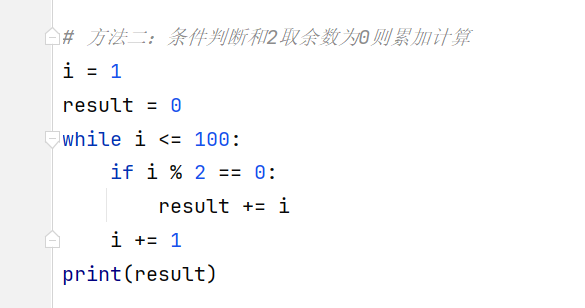
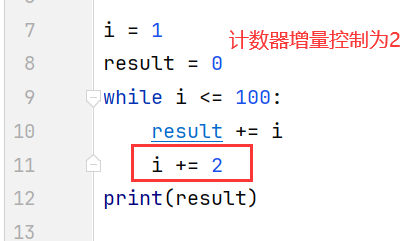
......



#### 案例：计算1到100的和



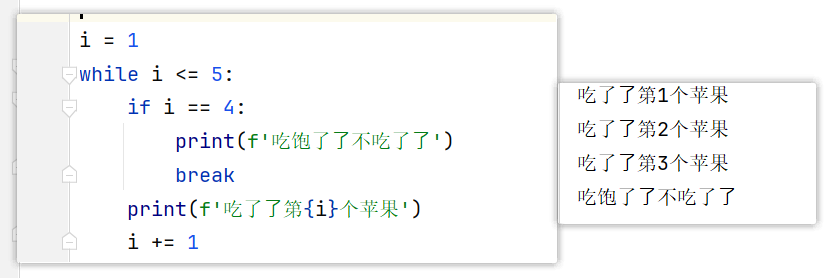
#### 案例：计算1到100的偶数和



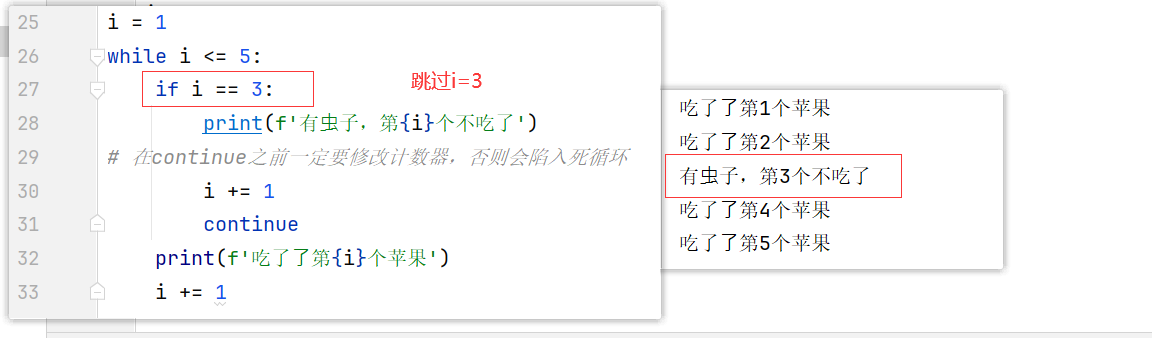
### 10.2 while的break和continue

break和continue是循环中满足一定条件退出循环的两种不同方式

break是终止此循环



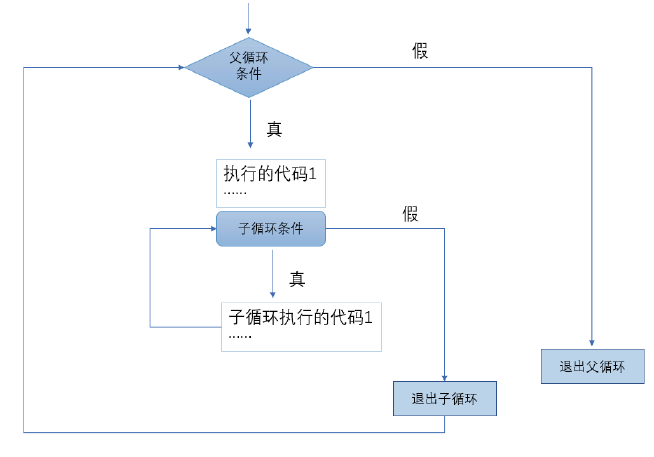
continue退出当前一次循环继而执行下一次循环代码



**拓展**：建议在实际开发中，使用循环时，i以0开始，i小于多少，就执行多少次

### 10.3 while循环嵌套

while循环嵌套，就是一个while里面嵌套一个while的写法，每个while的基础语法是相同的

语法：while 条件1:

条件1成立执行的代码

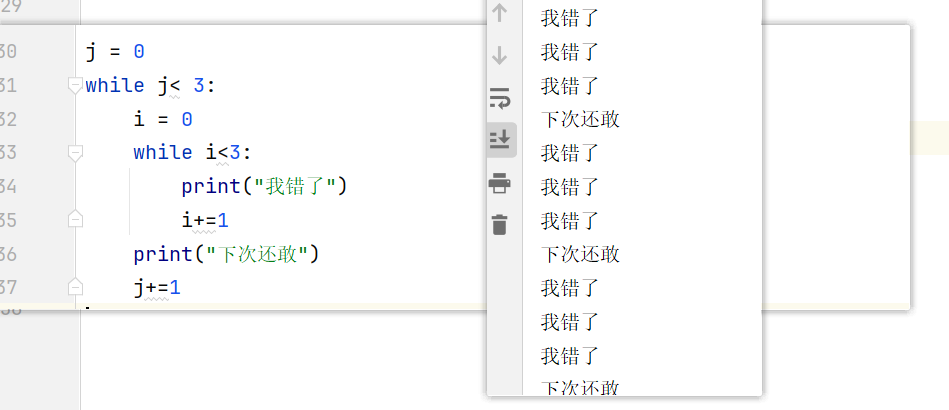
......

while 条件2:

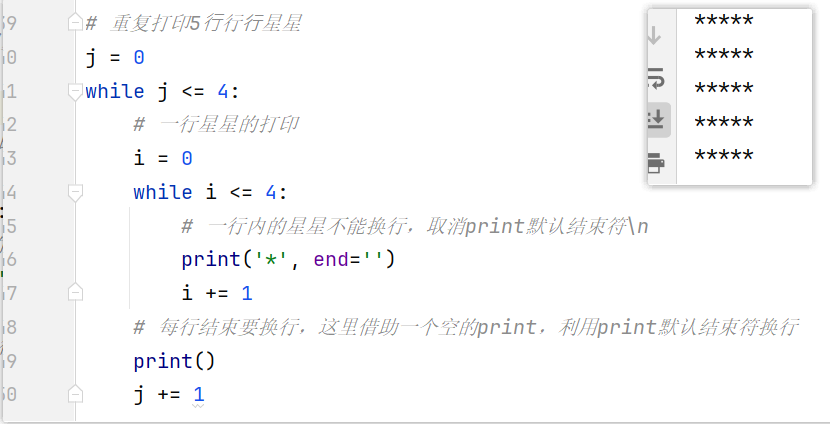
条件2成立执行的代码

......

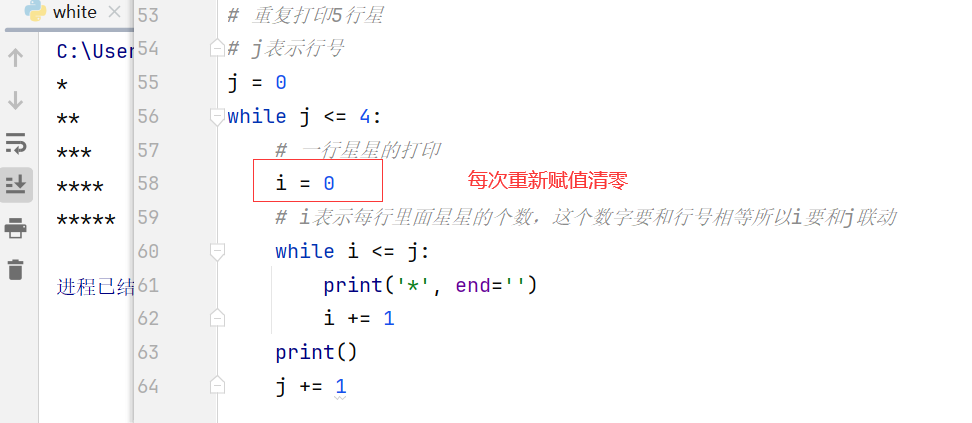
#### 案例：连续3次输出--我错了3次



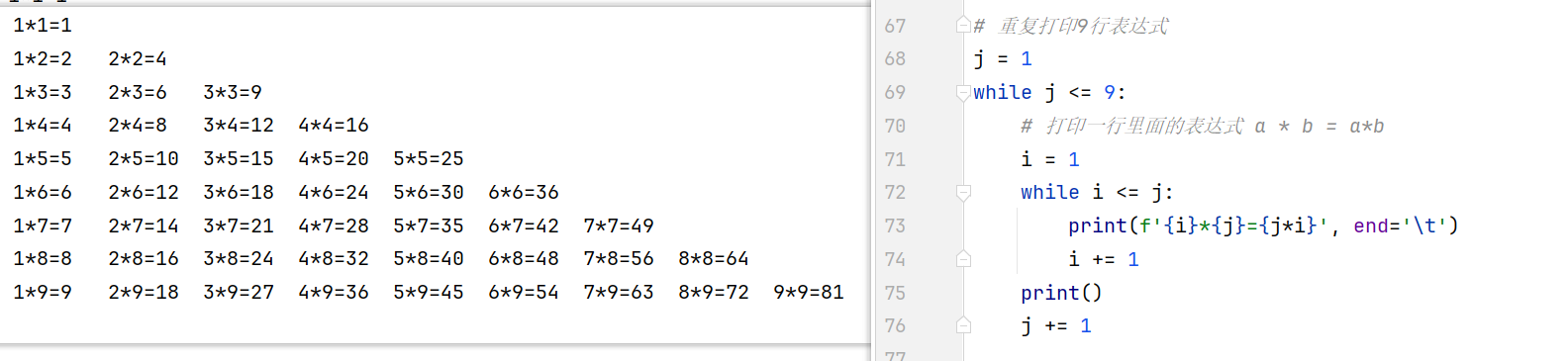
#### 案例：打印星星（正方形）



#### 案例：打印金字塔



#### 案例：九九乘法表



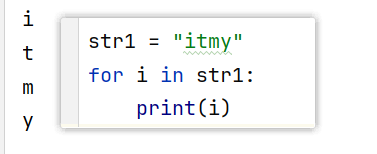
### 10.4 for循环语句

语法：for 临时变量 in 序列:

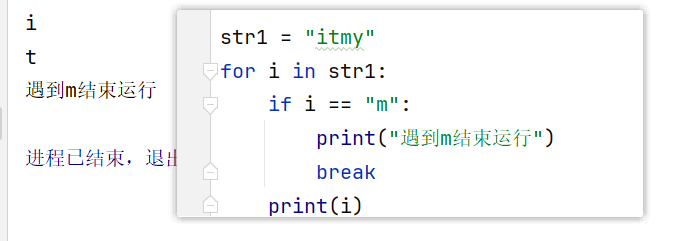
重复执行的代码1

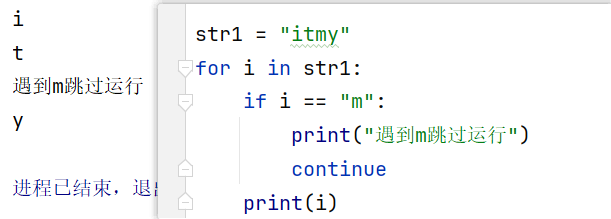
重复执行的代码2

......



### 10.5 for的break和continue





### 10.6 while…else

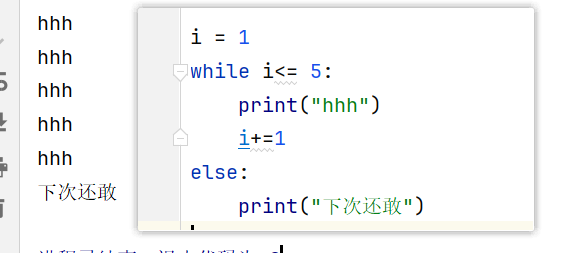
循环可以和else配合使用，else下方缩进的代码指的是当循环正常结束之后要执行的代码

语法： while 条件:

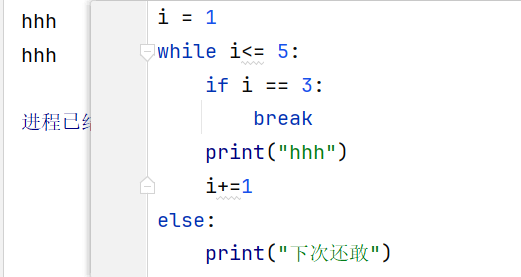
条件成立重复执行的代码

else:

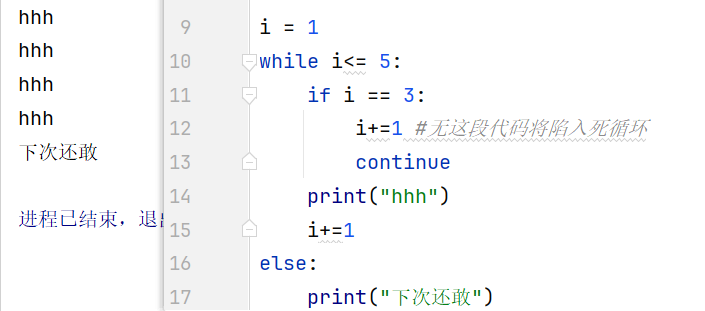
循环正常结束之后要执行的代码



配合break使用：break终止循环的情况，else下方缩进的代码将不执行



配合continue使用：因为continue是退出当前一次循环，继续下一次循环，所以该循环在continue控制下是可以正常结束的，当循环结束后，则执行了else缩进的代码



### 10.7 for…else

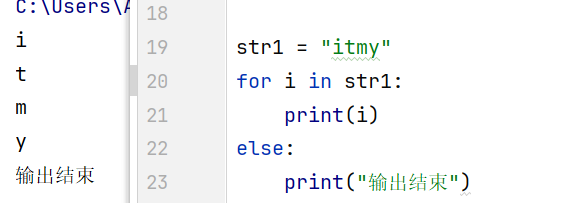
语法： for 临时变量 in 序列:

重复执行的代码

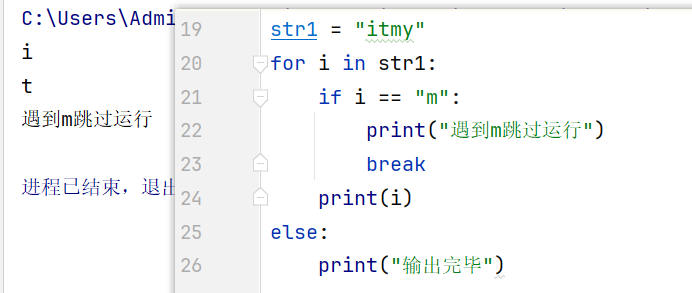
...

else:

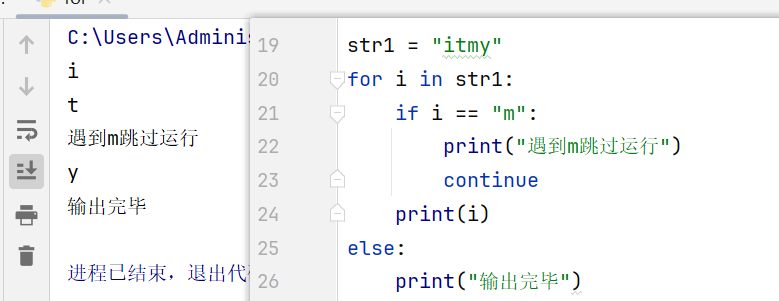
循环正常结束之后要执行的代码



配合break使用：

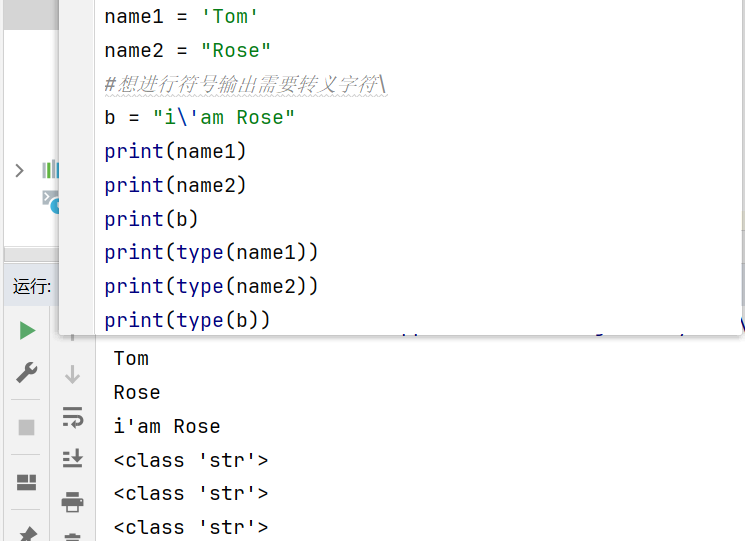


配合continue使用：



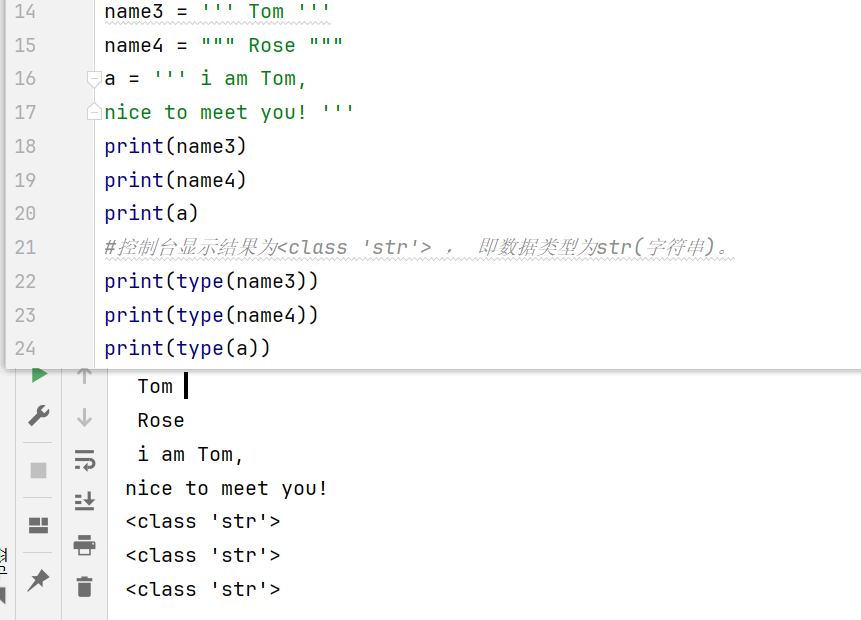
## 11. 字符串

### 11.1 一对引号字符串



### 11.2 三引号字符串串

三引号形式的字符支持换行



### 11.3 字符串输出

[4. 输出print](#_4._输出print)



### 11.4 字符串输入

[5. 输入input](#_5._输入input)

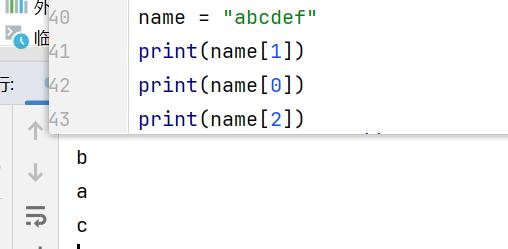


### 11.5． 下标（索引）

“下标” 又叫“索引” ，就是编号。

**注意**：下标从0开始

**示例**：字符串name = "abcdef" ，取到不同下标对应的数据





### 11.6. 切片

切片是指对操作的对象截取其中一部分的操作。字符串、表、元组都支持切片操作。

**语法**：序列 [ 开始位置下标 : 结束位置下标 : 步长 ]

注：

1. 不包含结束位置下标对应的数据， 正负整数均可；

2. 步长是选取间隔，正负整数均可，默认步长为1。



### 11.7 查找

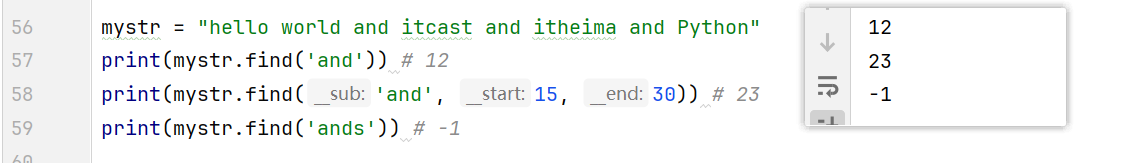
所谓字符查找⽅方法即是查找子串在字符串中的位置或出现的次数

#### find()（某个子串是否包含在这个字符串）检测某个子串是否包含在这个字符串中，如果在则返回这个子串开始的位置下标，否则则返回-1。

rfind()： 和find()功能相同，但查找方向为右侧开始。

**语法**：字符串序列.find(子串, 开始位置下标, 结束位置下标)

**注意**：开始和结束位置下标可以省略，表示在整个字符序列中查找。

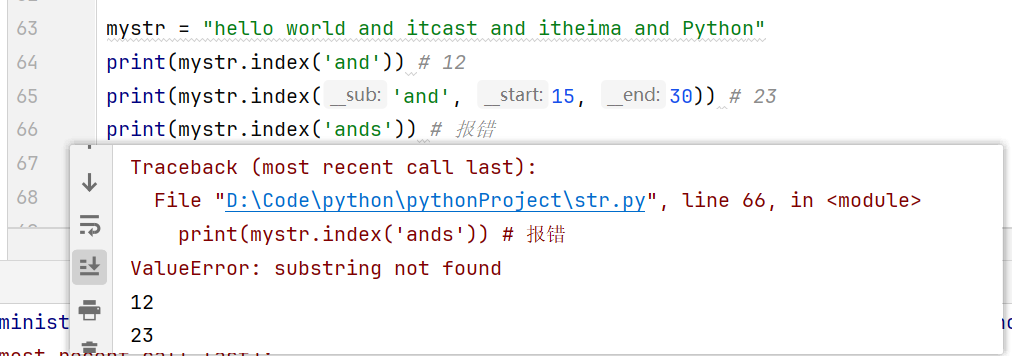


#### index()（子串是否包含在这个字符）检测某个子串是否包含在这个字符中，如果在返回这个子串开始的位置下标，否则则报异常。

rindex()：和index()功能相同，但查找方向为右侧开始。

**语法**：字符串序列.index(子串, 开始位置下标, 结束位置下标)

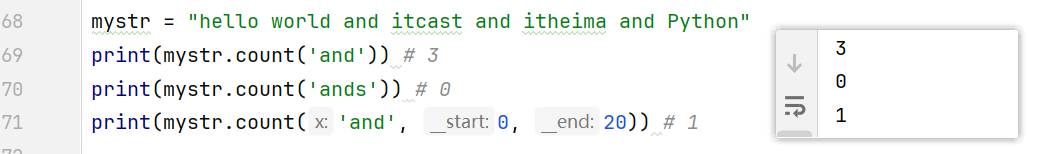
**注意**：开始和结束位置下标可以省略，表示在整个字符序列中查找。



#### count()（子串在字符串中出现的次数）返回某子串在字符串中出现的次数

**语法**：字符串序列.count(子串, 开始位置下标, 结束位置下标)

**注意**：开始和结束位置下标可以省略，表示在整个字符序列中查找。



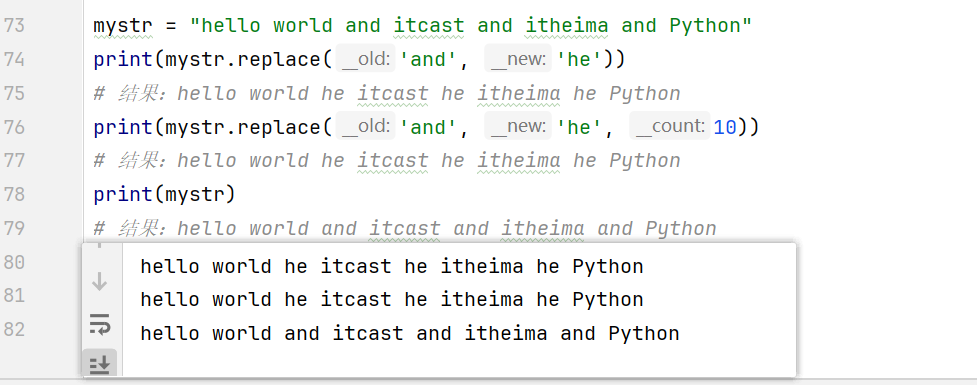
### 11.8 修改

所谓修改字符串，指的就是通过函数的形式修改字符串中的数据

#### replace()：替换

**语法**：字符串序列.replace(旧子串, 新子串, 替换次数)

**注意**：替换次数如果查出子串出现次数，则替换次数为该子串出现次数

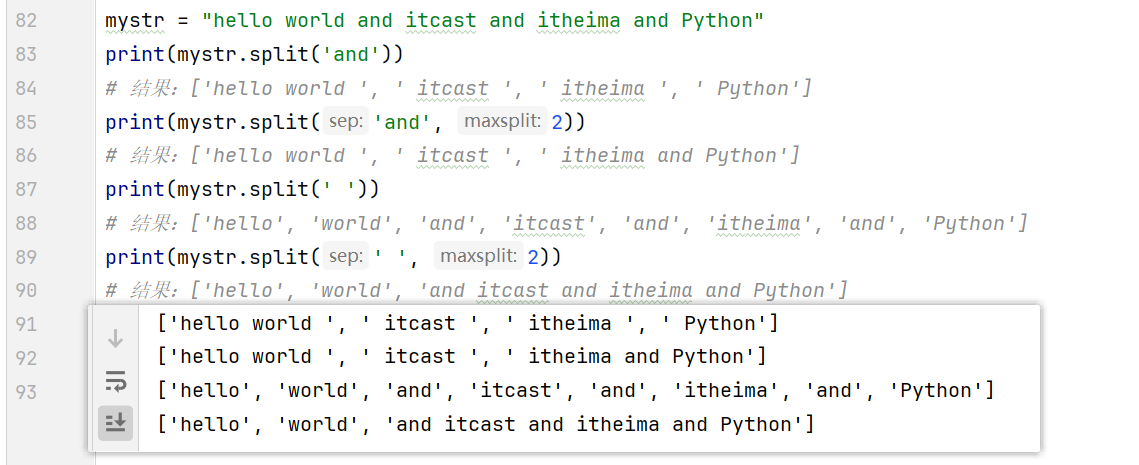


**注意**：数据按照是否能直接修改分为可变类型和不可变类型两种。字符串类型的数据修改的时候不能改变原有字符串，属于不能直接修改数据的类型即是不可变类型。

#### split()：按照指定字符分割字符串

**语法**：字符串序列.split(分割字符, num)

**注意**：num表示的是分割字符出现的次数，即将来返回数据个数为num+1个。



**注意**：如果分割字符是原有字符串中的子串，分割后则丢失该子串。

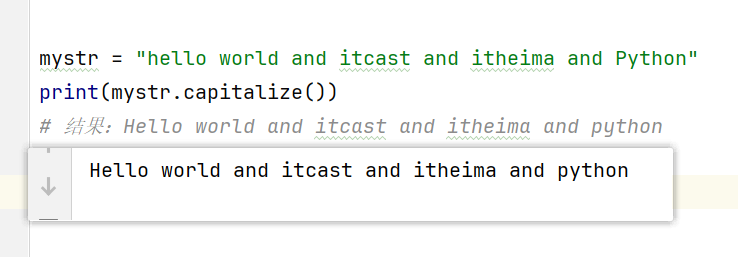
#### join()：用⼀个字符或子串合并字符串，即是将多个字符串合并为一个新的字符串。

**语法**：字符或子串.join(多字符串组成的序列)

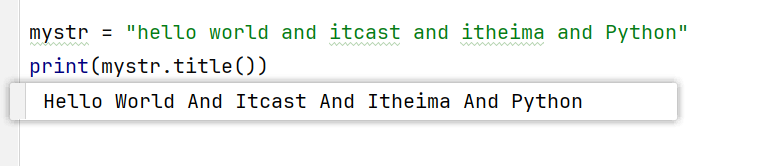


#### capitalize()：将字符串第一个字符转换成大写

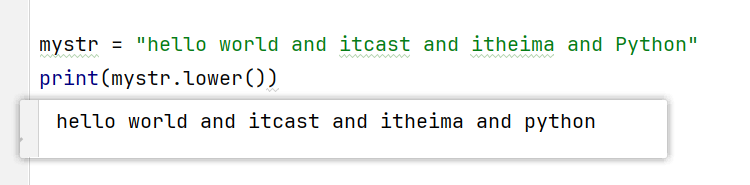
**注意**：capitalize()函数转换后，只字符串第一个字大写，其他的字符全都⼩小写。



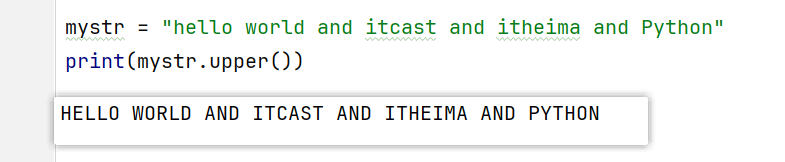
#### title()：将字符每个单词首字母转换成大写。



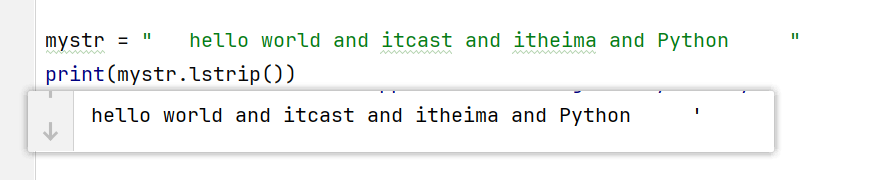
#### lower()：将字符串中大写转小写。



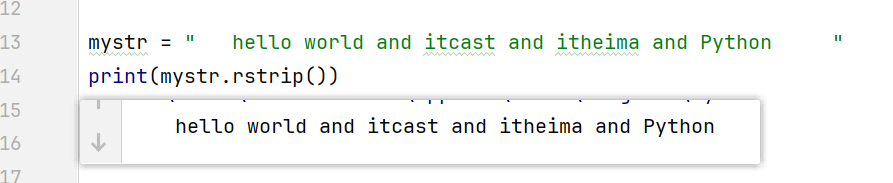
#### upper()：将字符串中小写转大写



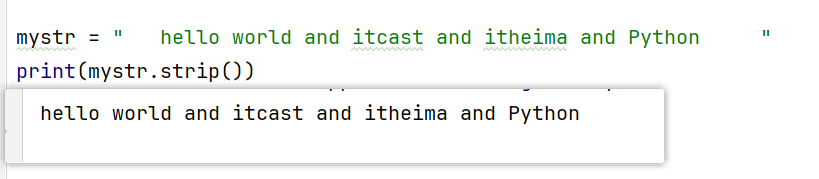
#### lstrip()：删除字符串左侧空白字符



#### rstrip()：删除字符串右侧空白字符

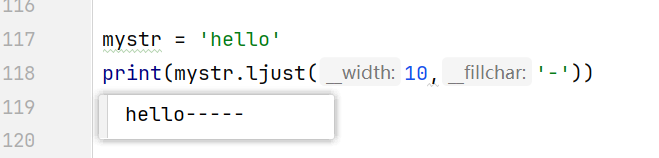


#### strip()：删除字符串两侧空白字符



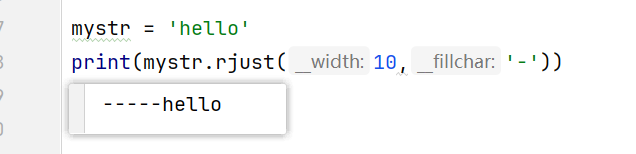
#### **ljust()：返回一个原字符串左对齐,并使用指定字符(默认空格)填充至对应长度的新字符串**

**语法**：字符串序列.ljust(长度, 填充字符)



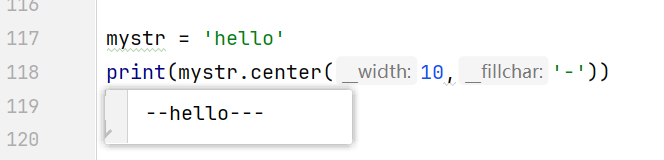
#### **rjust()：返回一个原字符串右对齐,并使用指定字符(默认空格)填充至对应长度的新字符串，语法和ljust()相同。**

**语法**：字符串序列.rjust (长度, 填充字符)



#### center()：返回一个原字符串居中对齐,并使用指定字符(默认空格)填充至对应长度 的新字符串，语法和ljust()相同。

**语法**：字符串序列.center (长度, 填充字符)

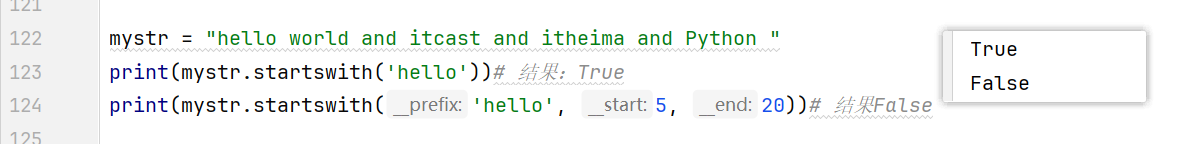


### 11.9 判断

所谓判断即是判断真假，返回的结果是布尔型数据类型：True 或 False

#### startswith()检查字符串是否是以指定子串开头，是则返回 True，否则返回 False。如果设置开始和结束位置下标，则在指定范围内检查。

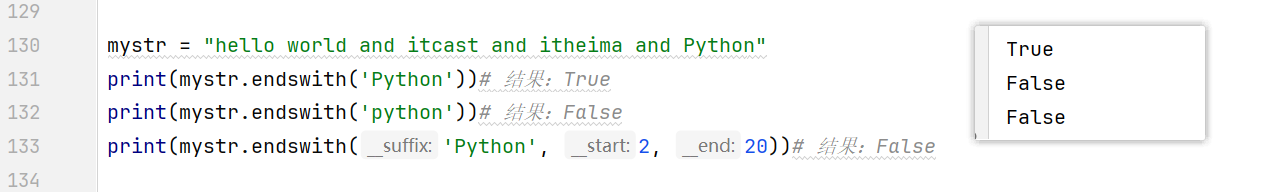
**语法**：字符串序列.startswith(子串, 开始位置下标, 结束位置下标)



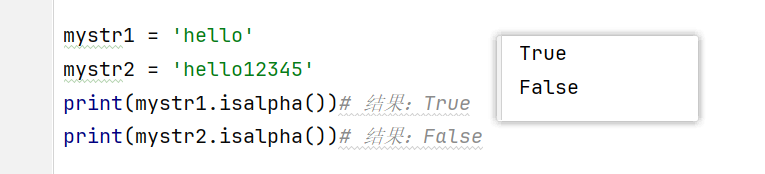
#### endswith()检查字符是否是以指定子串结尾，是则返回 True，否则返回 False。如果设置开

始和结束位置下标，则在指定范围内检查。

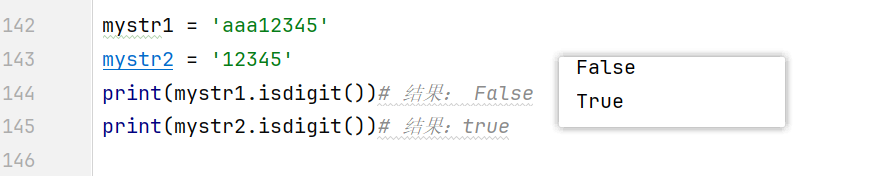
**语法**：字符串序列.endswith(子串, 开始位置下标, 结束位置下标)



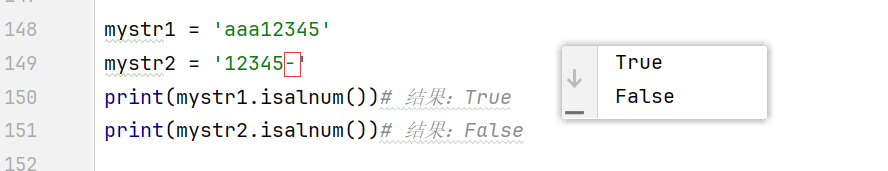
#### isalpha()如果字符串至少有一个字符并且所有字符都是字母则返回True, 否则返回 False



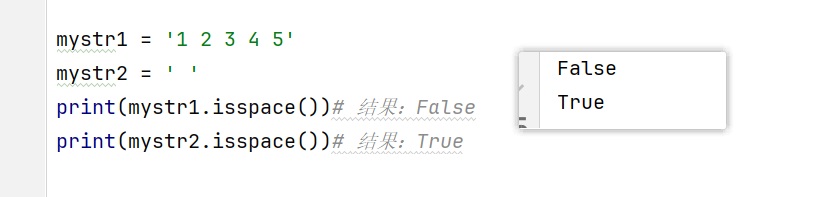
#### isdigit()如果字符串只包含数字则返回 True 否则返回 False。



#### isalnum()如果字符串至少有一个字符并且所有字符都是字母或数字则返回 True,否则返回False。



#### isspace()如果字符串中只包含空白，则返回 True，否则返回 False



# 综合案例

## 案例1：猜拳游戏

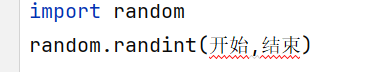
需求分析：

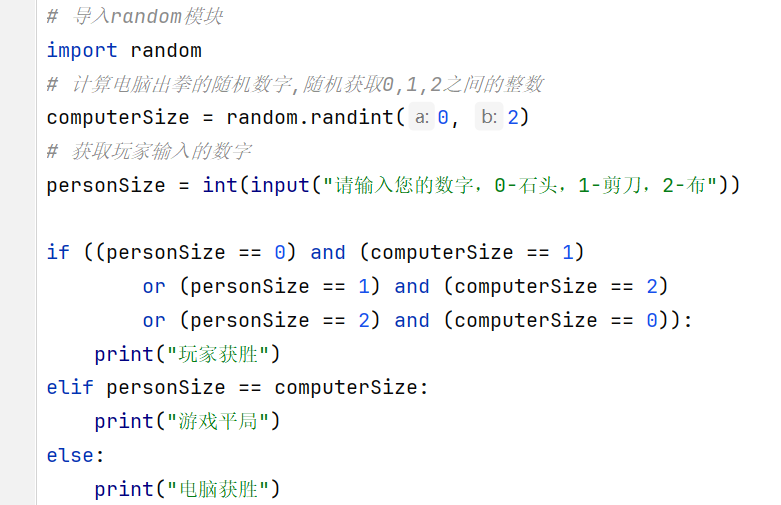
参与游戏的角色 玩家：手动出拳 电脑：随机出拳



平局 玩家出拳 和 电脑出拳相同

**1.需要导入random模块---**使用random模块中的随机整数功能





import random # 导入random模块

computerSize = random.randint(0, 2) # 计算电脑出拳的随机数字,随机获取0,1,2之间的整数

personSize = int(input("请输入您的数字，0-石头，1-剪刀，2-布")) # 获取玩家输入的数字

if ((personSize == 0) and (computerSize == 1)

or (personSize == 1) and (computerSize == 2)

or (personSize == 2) and (computerSize == 0)):

print("玩家获胜")

elif personSize == computerSize:

print("游戏平局")

else:

print("电脑获胜")