

Python 语言基础教程

Kuladmir Present

V 1.1.1

2025-8-17

Python 语言基础教程

目录/Contents

1 内置函数和库函数-----	2
2 基本规范和说明-----	2
3 运算符-----	5
4 选择结构-----	6
5 循环结构-----	7
6 序列结构-----	0
7 字符串操作-----	0
8 函数-----	0
9 文件操作-----	0
10 面向对象程序设计 I-----	0
11 面向对象程序设计 II-----	0
12 数据库和 Python 联用-----	0
EX.部分例题-----	0

特殊说明：此版本内页码仅供参考，因为还是不完全体....

待较完全版本确认后，页码将会被修改

[声明] 本文档仅用于个人学习，不可用以他用，例如牟利等。文档经由

https://www.bilibili.com/video/BV1Ys4y1D72T/?spm_id_from=333.788.player.switch&vd_source=6a66772e6a688ab2ca4e733f22547766&p=24，并通过个人理解形成，以及参考课程内容等整理后形成。

说明：加红字体表示提示，加绿加深字体表示 python 语句，加蓝加深字体表示内置函数和关键字。

1 内置函数和库函数

函数名	功能	举例
abs()	求绝对值	a = -5 b = abs(a) b -> 5
divmod()	取余	a = divmod(5,2) a -> (2,1)
pow()	次方	c = pow(5,2) c -> 25
sum()	求和	a = sum([1,2,3,4]) a -> 10
round()	四舍五入	a = round(1.5) a -> 2
max()	求最大值	a = max(1,2,3,4) a -> 4
min()	求最小值	a = min(1,2,3,4) a -> 1
len()	计算列表长度	len(a)
id()	打印变量的地址	print(id(a))
type()	打印变量的类型	print(type(a))
类型名()	把某类型的数据转化为期望的类型	此处的类型名可以为 float int str round bool chr ord eval
list()	将序列转化为列表	print(list(range(0,30,2)))
str()	将序列转化为字符串	print(str(range(0,30,2)))
sorted()	对元素进行排序	/
reversed()	反向列表的元素	/
enumerate()	将序列组合成索引	/

turtle 库：可以进行绘图操作

math 库：可以进行数学运算操作

random 库：可以进行随机数及相关操作

time 库：可以进行时间及相关操作

第三方库：

包管理工具 pip

语法：pip<命令>[模块名]

2 基本规范和基础操作

1) 书写规范

1.赋值时，= 两边需要间隔一个空格。例如：**a = 10**

2.字符串界定，交替使用单双引号。

2) 标识符命名规则

数字不能在开头，区分大小写，只能使用数字、下划线、字母，不能使用关键字。

3) 数据类型

整数(int): 十进制、八进制 (0o)、十六进制(0x), 在变量初始化时, 可以使用进制表示符加数字进行初始化, 打印时默认打印十进制数字。

浮点数(float): 变量初始化时, 如果小数点前后无数字, 则默认为 0, 但两边至少有一个数字。

字符型/字符串(str)。

复数(complex): 形式和数学中相同: $a + bj$ 或者 $a + bJ$ 。

复数实部和虚部的提取: `a = 5 + 6j` `print(a.real)` `print(a.imag)`

注意: 使用 `a.real` 或 `a.imag` 提取的实部或虚部, 都是 **float** 类型。

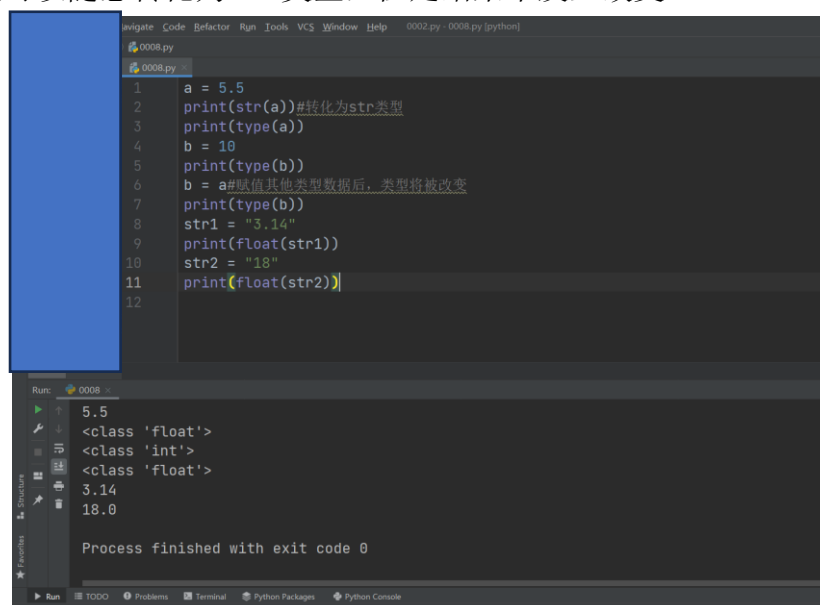
4) 数据类型转化:

float 转化为 **int**: 仅保留整数部分, 反之, 将在小数位补 0。

int 转化为 **str** 类型: 可以随意转化, 但是结果不改变, 转换后可以使用其各种函数。

如果 **str** 类型中存储的为 **float** 类型数据或 **str** 类型, 则不能转化为 **int**。

float 类型可以随意转化为 **str** 类型, 但是结果不发生改变。



```
0008.py
1 a = 5.5
2 print(str(a))#转化为str类型
3 print(type(a))
4 b = 10
5 print(type(b))
6 b = a#赋值其他类型数据后, 类型将被改变
7 print(type(b))
8 str1 = "3.14"
9 print(float(str1))
10 str2 = "18"
11 print(float(str2))
12
```

Run: 0008

```
5.5
<class 'float'>
<class 'int'>
<class 'float'>
3.14
18.0

Process finished with exit code 0
```

5) 转义字符

\n 换行, **\a** 响铃, **\t** 水平制表符, 等于 Tab, **\v** 垂直制表符, **\'("?\)** 表示输出该内容, **\b** 退格, **\r** 回车, **\f** 换行, **%%** 输出 %

格式化字符: **%s** – 字符串 **%d** – 整数 **%c** – 字符 **%f** – 浮点数 **%o** – 八进制输出 **%x** – 十六进制输出 **%e** – 科学计数法输出

可以使用 **%.nf** 选择输出的小数位数, **%0nd** 设定补零的数量, **n** 表示总位数, 补 0 的数目等于 **n** - 自身位数。

6) 注释

如果希望单行注释，在要注释的行前加入 # 符号，并且和语句内容间隔一个空格，可以在语句末尾注释，也可以使用 Ctrl 和 / 快捷注释。如果希望多行注释，则需要在被注释内容第一行的上行加入三个双引号或单引号，在被注释内容最后行的下一行加入三个双引号或单引号，多行注释不能在语句末尾生效。

7) 基础打印内容

- 1.如果希望打印文字等，需按照此语法实现：`print("待打印的内容")`
- 2.如果希望打印变量的数值等，使用此语法实现：`print(a,b)`
- 3.如果希望格式化输出，可使用 `print('%格式化字符'%变量)`实现。

例如：`print('A %s has %d legs'%(‘monkey’,4))`

在使用时，对%f数据来说可以使用%m.nf。其中，m表示输出前空的格数，n表示小数保留位数。

也可以使用如下格式化方式：

`print('{0}的年龄为{1}'.format('张三', 20))`

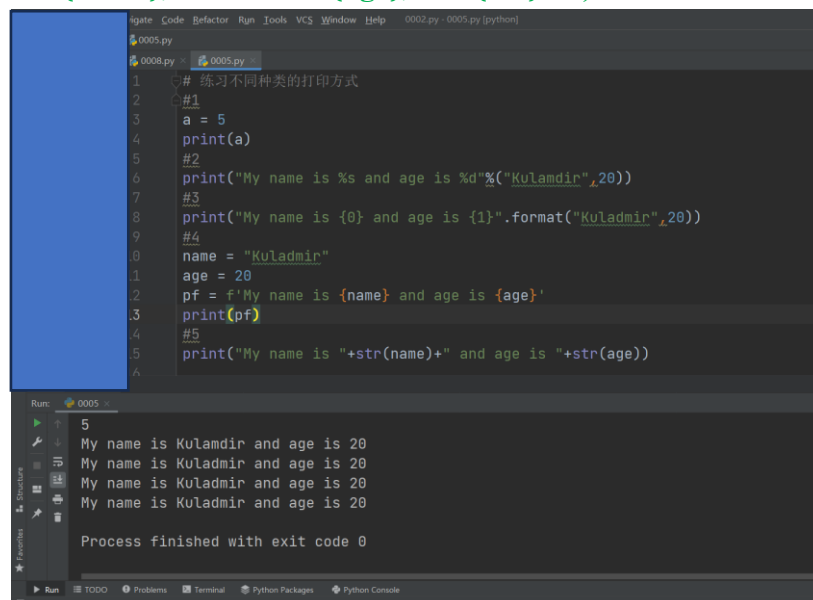
- 4.如果希望在打印的文字之间输出数字，可以使用如下方法：

`a = 20 print('张三的年龄是'+str(a)+'岁')`

f-string 格式化输出方式：

`age = 10 name = 'kula' sex = 'M'`

`print(f'我是{name},我的年龄是{age},我是{sez}生')`



The screenshot shows a Python IDE with a file named 0005.py. The code in the editor is as follows:

```
1 # 练习不同种类的打印方式
2 #1
3 a = 5
4 print(a)
5 #2
6 print("My name is %s and age is %d"%(‘Kulamdir’,20))
7 #3
8 print("My name is {0} and age is {1}".format("Kulamdir",20))
9 #4
10 name = "Kulamdir"
11 age = 20
12 pf = f'My name is {name} and age is {age}'
13 print(pf)
14 #5
15 print("My name is "+str(name)+" and age is "+str(age))
```

The output window shows the following results:

```
5
My name is Kulamdir and age is 20
My name is Kulamdir and age is 20
My name is Kulamdir and age is 20
My name is Kulamdir and age is 20
Process finished with exit code 0
```

说明：输出语句默认都会换行，如果不想换行，可以使用 `end = “结束符”`，如果想要让每个值间都有一个间隔符号。可以使用 `sep = “间隔符”`。

例如：`print(i, end=‘ ’)`

```
test.py x
1 a = 1
2 print(a)
3 a = "kuLadmir"
4 print(a)
5 for i in range(0,3,1):
6     print(i)
7 for i in range(0,3,1):
8     print(i,end="||")
9 print()
10 print('a','b','c',sep='-')
```

```
test x
D:\python\python.exe "C:\Users\kuLadmir\Desktop\事务文件夹\python\0-0500\练习\test.py"
1
kuLadmir
0
1
2
0||1||2||
a-b-c
```

8) 变量赋值

可以对多个变量同时赋值：**a,b,c=12,25,26**，如果想把字符内容赋值给变量，则需要给赋值内容加上引号，单双引号均可；三引号包含的内容可以根据代码分行显示。**如果两个变量数值相同，则这两个变量的存储地址相同**，无论是**a=b=5**的定义方式还是**a=5 b=5**的定义方式。

input()函数：如果想打印提示内容，可以在**input()**中加入文字等。
例如：**a = input("请输入")** 此时 a 的类型默认为字符类型。如果希望变量类型
为其他，则需要在**input()**外引用数据类型转换函数。例如：**a = int(input("请输入"))**

9) 异常处理

try: 可能出错的语句块 **except**: 语句块

当**try**里的语句出现错误，则会输出**except**后的内容。如果**try**里语句没有错误，则将继续执行其后的代码。

【高级应用】

try: 可能出错的语句块 **except** 错误名: 语句块 **except**: 语句块

当**try**里的语句出现错误，则会输出**except**后的内容。如果**try**里语句没有错误，则将继续执行其后的代码。如果出现的错误为错误名中的，则会对应输出该**except**内的语句，否则将输出别的**except**的内容。（类似于一个判断）

3 运算符[优先级（越小越优先）;目]

1) 加减乘除类型:

+[3;2] -[3;2] *[2;2] /[2;2] %[2;2] //整除[2;2] **幂运算[1;2]

乘法：*不仅可以用来计算两个数的乘积，也可以在输出时设定某字符输出的次数。

```
print("@"*10) # 最终输出 10 次@
```

除法：运算结果必定为浮点型。

加法：+不仅可以用来计算两个数的和，也可以用以连接两个字符串。

```
print('A'+ 'CDE') # 最终输出 ACDE
```

说明：python 不支持 a++ 等做法，但可使用 += 等做法。

2) 关系运算符[8;2]:

< > <= >= != == is[9;2]效果同== is not[9;2]效果同!=

得到的结果为 bool 类型，只有 True 和 False 两个结果。

3) 逻辑运算符:

and(&) 逻辑与：同真为真，否则为假；

or() 逻辑或：一真为真，全假为假；

not(!) 逻辑非：反转操作数逻辑状态。

3) 位运算符:

右移>>[4;2] 左移<<[4;2] 按位与&[5;2] 按位或|[7;2] 按位异或^[6;2] 按位取反~[/;1]

按位与：二进制位同 1 则 1，否则为 0；按位或：二进制位有 1 则 1，否则为 0；

按位异或：二进制位相同为 0，否则为 1；按位取反：二进制为 0 改为 1，反之。

说明：bool 类型的数据也可参与运算：True 表示 1，False 表示 0，如果一个算数中，有多种类型数据，则在输出时的数据为最高级的类型。

5) 三目运算符： 语句 1 if 条件 else 语句 2

理解：如果条件满足则执行 1，否则执行 2。

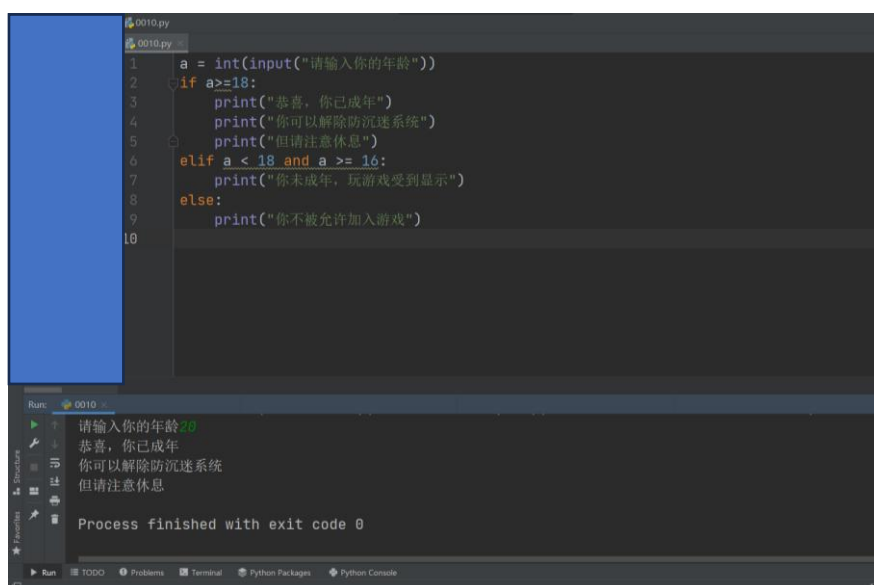
4 选择结构

1) 语法结构：if 表达式: 语句 else: 语句

2) 基本语句：if.....else 语句；if.....elif.....else 语句；

3) 语句嵌套：if if.....else else if.....else

说明：注意缩进，这是判断是否包含在 if.....else 语句的重要依据。分支条件可为逻辑判断语句，也可以为赋值等语句，也可以为 bool 值等。



```
0010.py
1 a = int(input("请输入你的年龄"))
2 if a >= 18:
3     print("恭喜, 你已成年")
4     print("你可以解除防沉迷系统")
5     print("但请注意休息")
6 elif a < 18 and a >= 16:
7     print("你未成年, 玩游戏受到显示")
8 else:
9     print("你不被允许加入游戏")
10
```

Run: 0010

请输入你的年龄20
恭喜, 你已成年
你可以解除防沉迷系统
但请注意休息

Process finished with exit code 0

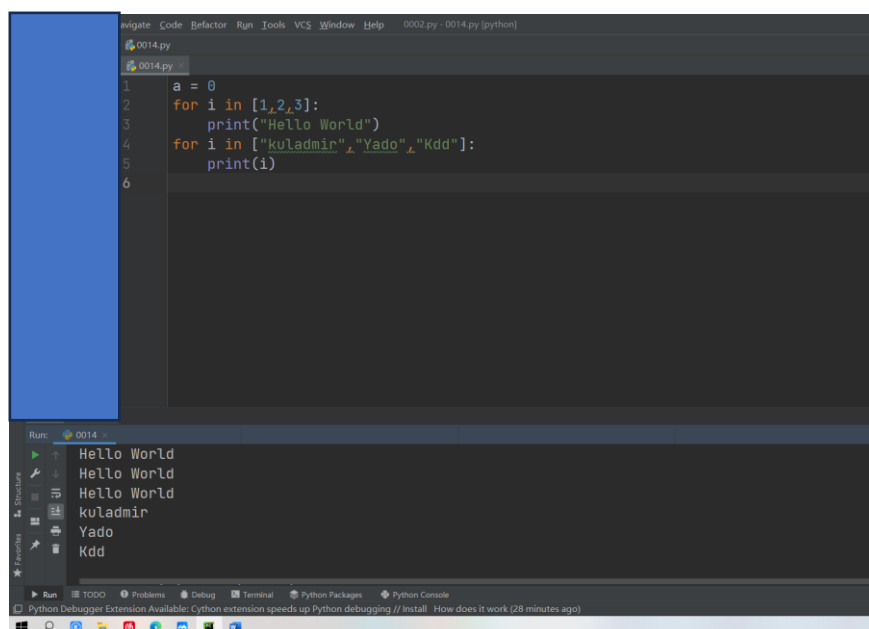
5 循环结构

1) **while** 语句结构【无限循环】: **while** 条件 : 语句块 (需要有变量调整条件)
当条件满足时, 才输出内容, 否则退出循环。

2) **for** 语句结构【遍历循环】: 存在两种形式。

1. 数值循环: **for** 变量 **in** [内容]:

内容里可以为数字, 也可以为字符串。利用数值循环可以输出其他内容, 也可以输出列表中的内容。



```
0014.py
1 a = 0
2 for i in [1,2,3]:
3     print("Hello World")
4 for i in ["kuladmir", "Yado", "Kdd"]:
5     print(i)
6
```

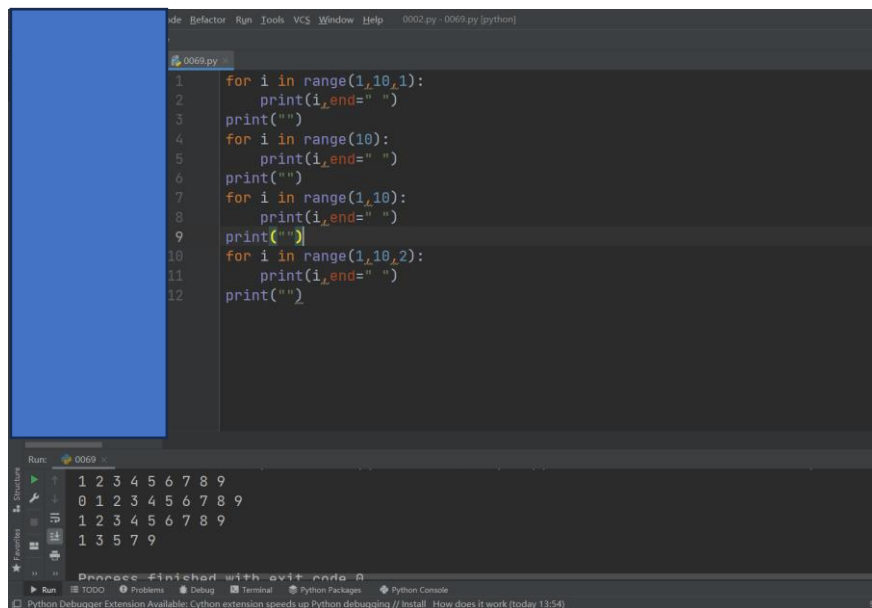
Run: 0014

Hello World
Hello World
Hello World
kuladmir
Yado
Kdd

2. 数值循环, 和 **range** 联用: **for** 变量 **in range**(初始值, 结束值, 步长):

注意: 如果 **range** 中只有一个变量, 则表示为结束值 **end**, 表示循环范围为[0, **end**-1], 步长为 1; 如果 **range** 中有两个变量, 则第一个为初始值 **start**, 第二个为结束值 **end**, 步长默认为 1。不管如何设定, **range** 的范围都不包含结束值 **end**。

特殊用法：可以在字符串里对某一字符的出现次数进行计数：



```
0069.py
1 for i in range(1,10,1):
2     print(i,end=" ")
3     print("")
4     for i in range(10):
5         print(i,end=" ")
6         print("")
7         for i in range(1,10):
8             print(i,end=" ")
9             print("")
10        for i in range(1,10,2):
11            print(i,end=" ")
12        print("")
```

Run: 0069

```
1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 5 7 9
```

Process finished with exit code 0

3) 中断语句：

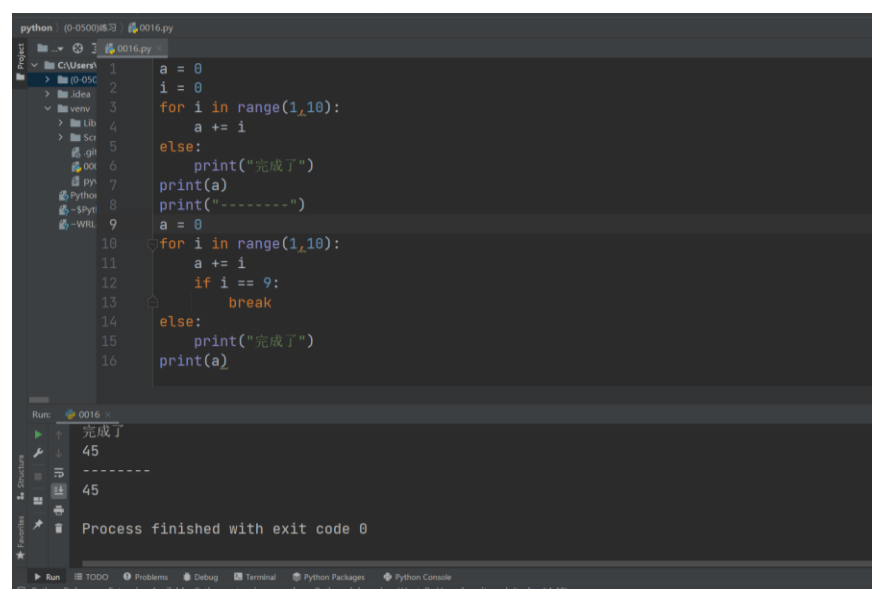
break：可以直接将此循环结束。**continue**：直接跳过此循环后面的内容。**pass**：用来占位，即不产生任何作用。

4) **while** / **for.....else** 语句：

语法结构：**while** 条件：语句结构 **else**：语句结构

for 循环变量 **in** [内容] / **range()**：语句结构 **else**：语句结构

说明：如果 **for** 或者 **while** 的语句结构内，没有 **break** 结束循环，那么循环结束后，也会执行 **else** 语句。



```
python 0-0500练习 0016.py
0016.py
1 a = 0
2 i = 0
3 for i in range(1,10):
4     a += i
5 else:
6     print("完成了")
7 print(a)
8 print("-----")
9 a = 0
10 for i in range(1,10):
11     a += i
12     if i == 9:
13         break
14 else:
15     print("完成了")
16 print(a)
```

Run: 0016

```
完成了
45
-----
45
```

Process finished with exit code 0

6 序列结构（一块存放多个值的连续内存空间，包含四个部分）

1) 列表操作

1.列表定义和删除

C = ["内容"] #字符串加双引号，字符加单引号，内容为空时空列表。

del C #可删除此列表

2.索引:

每个列表单位都与一个编号绑定，类似于 C 中的数组下标，这个编号从前到后依次为 0、1、2.....而且该编号可以为负，从后到前依次是-1、-2.....

列表元素	1	2	3	4	5	6	7
编号：正	0	1	2	3	4	5	6
编号：负	-7	-6	-5	-4	-3	-2	-1

3.切片:

语法: `print(sname[start:end:step])`

sname 为列表名，start 为起始点，end 为终止点，step 为步长。三者不设置时分别默认为 0，全长，1,以此得到从编号为 start 到编号 end（包含）之间的所有元素。如果不设置起点，步长为负，则起点默认为-1。切片不会对原列表产生影响，输出仍为列表。

4.列表相加:

`print(c1 + c2)`#c1 c2 为两个列表，列表内容类型可以不同，但 c1 c2 类型必须相同（同为列表或元组等），输出仍为列表。

5.列表乘法:

`print(c1 * n)`#c1 为一个列表，此操作将把 c1 中内容打印 n 遍，输出仍为列表。

6.判断成员:

`print(search in c1)`#search 为待查找内容，c1 为列表，存在返回 True，否则返回 False。

7.计算列表的长度、最值等:

使用基本函数 `len()`、`max()`、`min()`进行计算。

```
0024.py
1 b = [1901,1951,1964,1954,1965]
2 a = ["Kula"]
3 c = ["Kuladmin", "Apex", "Legend", 2019, 3306, 951]
4 #索引
5 print(c[2],c[-2])#Legend_3306
6 #切片
7 print(c[1:4])#Apex_Legend_2019_3306
8 #乘法
9 print(a * 5)#Kula_Kula_Kula_Kula_Kula
10 #序列相加
11 print(a + b)
12 #检查成员
13 print("Apex" in c)
14 print("Kulepati" in c)
15 #序列长度
16 print(len(b),max(b),min(b))

Run: 0024
Legend 3306
['Apex', 'Legend', 2019]
['Kula', 'Kula', 'Kula', 'Kula', 'Kula']
['Kula', 1901, 1951, 1964, 1954, 1965]
True
False
5 1965 1901
```

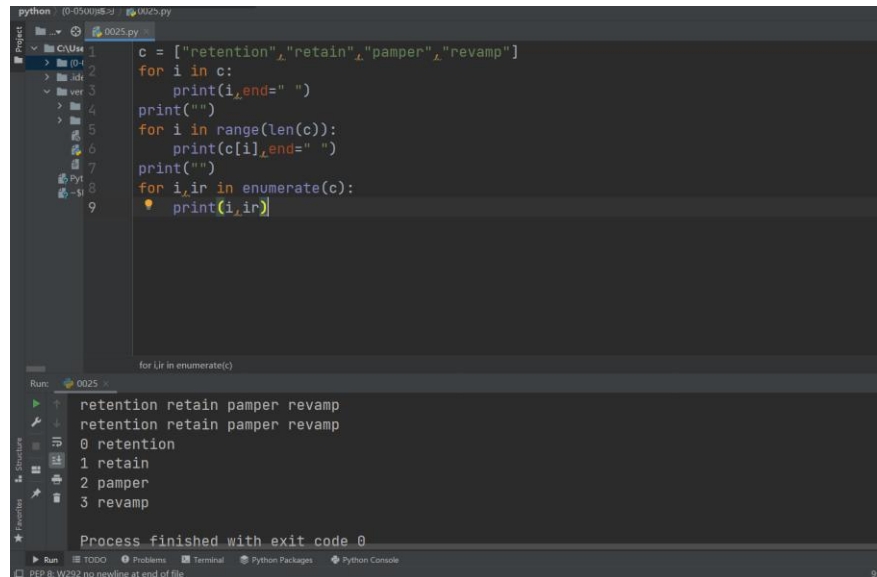
8.遍历列表:

for i in 列表名: print(i)

for i in range(len(列表名)): print(列表名[i])

如果希望以索引的形式输出, 可以采用如下方法:

for i,ir in enumerate(列表名): print(i,ir)

A screenshot of a Python IDE (likely PyCharm) showing a script named '0025.py'. The script defines a list 'c' with elements 'retention', 'retain', 'pamper', and 'revamp'. It then demonstrates three different ways to iterate over the list: 1. A simple 'for' loop printing each element. 2. A 'for' loop with 'range(len(c))' printing the index and element. 3. A 'for' loop with 'enumerate(c)' printing the index and element. The output window shows the results of these iterations. The first iteration prints the elements directly. The second iteration prints the index and element separated by a space. The third iteration prints the index and element separated by a space. The process finished with exit code 0.

```
python 0025.py
c = ["retention", "retain", "pamper", "revamp"]
for i in c:
    print(i, end=" ")
print("")
for i in range(len(c)):
    print(c[i], end=" ")
print("")
for i, ir in enumerate(c):
    print(i, ir)
```

Run: 0025

```
retention retain pamper revamp
retention retain pamper revamp
0 retention
1 retain
2 pamper
3 revamp
```

Process finished with exit code 0

9.增加元素 (会对原列表造成影响): (假设 c 已经为定义的列表)

c.append(内容)方法。此法可以将内容加到列表内容最后, 可以把整个列表内容加入进去, **返回值为 None**。

c.extend(内容)方法。此法可以将内容逐个拆开, 添加到列表最后, 如果内容是个序列, 则会将序列中的内容逐个加入, **返回值为 None**。

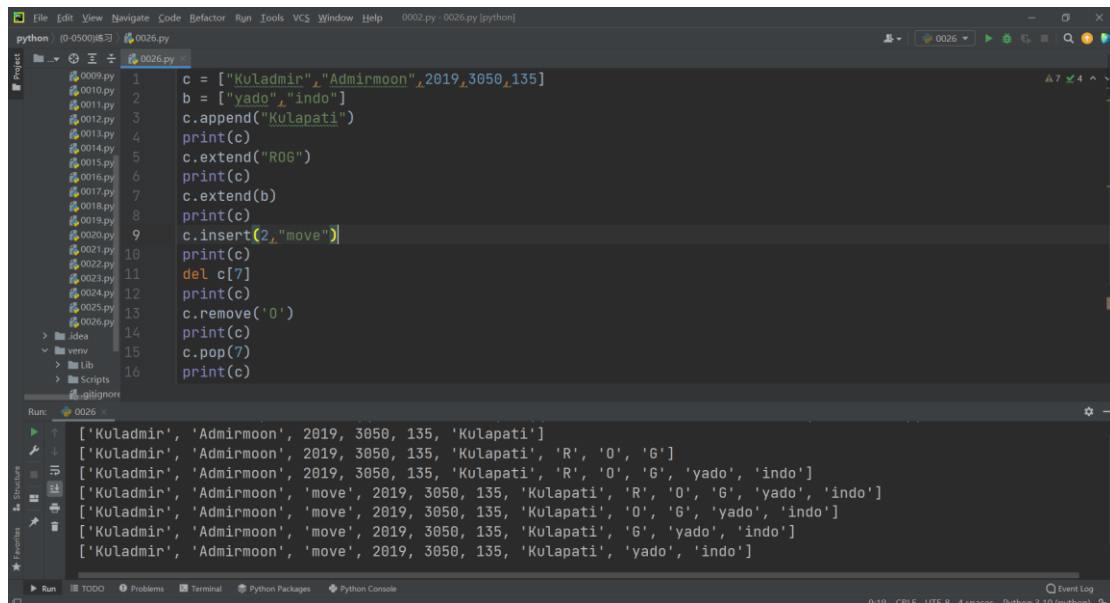
c.insert(定位数,内容)方法。此法可以在指定位置添加内容。定位数对应每个元素编号, 会在该元素前添加内容, **返回值为 None**。

10.删除元素 (会对原列表造成影响): (假设 c 已经为定义的列表)

通过访问元素编号删除元素: **del c[编号]**

通过访问元素值删除元素: **c.remove(某元素的值)** **#如果某元素重复出现, 只会删除先出现的**

c.pop(元素编号)删除法。如果不加参数, 则默认删除最后一个元素, **结果为列表**。



```
python 0-0500练习 0026.py
1 c = ["Kuladmir", "Admirmoon", 2019, 3050, 135]
2 b = ["yado", "indo"]
3 c.append("Kulapati")
4 print(c)
5 c.extend("R06")
6 print(c)
7 c.extend(b)
8 print(c)
9 c.insert(2, "move")
10 print(c)
11 del c[7]
12 print(c)
13 c.remove('0')
14 print(c)
15 c.pop(7)
16 print(c)

Run: 0026
['Kuladmir', 'Admirmoon', 2019, 3050, 135, 'Kulapati']
['Kuladmir', 'Admirmoon', 2019, 3050, 135, 'Kulapati', 'R', '0', 'G']
['Kuladmir', 'Admirmoon', 2019, 3050, 135, 'Kulapati', 'R', '0', 'G', 'yado', 'indo']
['Kuladmir', 'Admirmoon', 'move', 2019, 3050, 135, 'Kulapati', 'R', '0', 'G', 'yado', 'indo']
['Kuladmir', 'Admirmoon', 'move', 2019, 3050, 135, 'Kulapati', '0', 'G', 'yado', 'indo']
['Kuladmir', 'Admirmoon', 'move', 2019, 3050, 135, 'Kulapati', 'G', 'yado', 'indo']
['Kuladmir', 'Admirmoon', 'move', 2019, 3050, 135, 'Kulapati', 'yado', 'indo']
```

11.修改和查找元素：（假设 c 已经为定义的列表）

修改：直接通过编号修改。**c[n] = 新内容**

倒序：将列表内容反置。**c.reverse()** #会对原列表进行修改

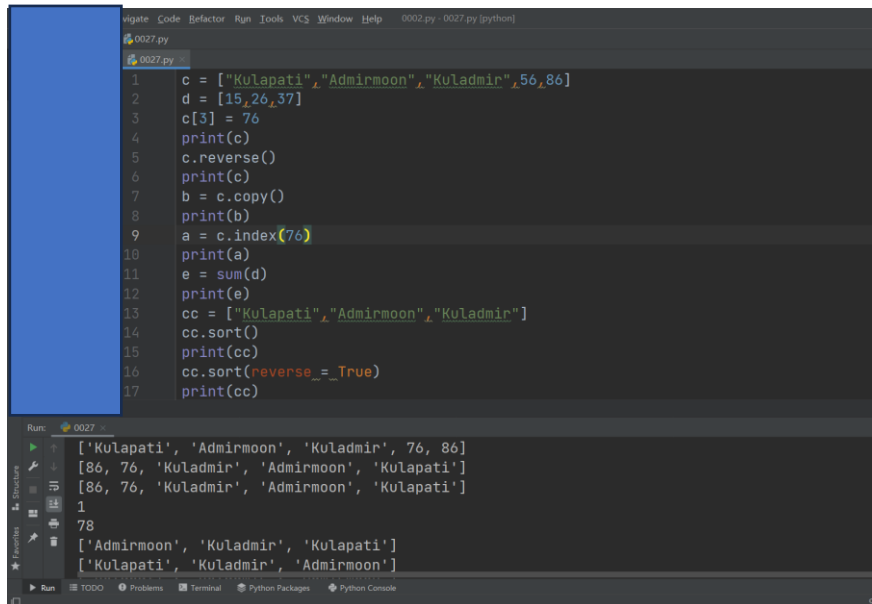
复制：可以复制某列表内容。**c.copy()**

查找：**c.index(查找对象[,start,end])**方法，可以设置查找的范围，如果不设置默认为全列表，找到返回对应元素编号，否则报错。**c.count(内容)**方法：此方法可以计算被查找元素的出现次数。如果使用 **rindex** 函数，则将会从右到左查找。

12.列表元素统计和排序：（假设 c 已经为定义的列表）

统计（对数字列表）：**a = sum(c)** 可计算列表中数字之和

排序：**c.sort([reverse=True/False])**#设置 reverse 时，true 表示倒序输出，否则为正序输出，**会改变原列表顺序**。**sorted(列表名,[reverse = True/False])**方法，**不会改变原列表顺序，但是会产生新的副本**。



```
0027.py
1 c = ["Kulapati", "Admirmoon", "Kuladmir", 56, 86]
2 d = [15, 26, 37]
3 c[3] = 76
4 print(c)
5 c.reverse()
6 print(c)
7 b = c.copy()
8 print(b)
9 a = c.index(76)
10 print(a)
11 e = sum(d)
12 print(e)
13 cc = ["Kulapati", "Admirmoon", "Kuladmir"]
14 cc.sort()
15 print(cc)
16 cc.sort(reverse=True)
17 print(cc)
```

Run: 0027

```
['Kulapati', 'Admirmoon', 'Kuladmir', 76, 86]
[86, 76, 'Kuladmir', 'Admirmoon', 'Kulapati']
[86, 76, 'Kuladmir', 'Admirmoon', 'Kulapati']
1
76
['Admirmoon', 'Kuladmir', 'Kulapati']
['Kulapati', 'Kuladmir', 'Admirmoon']
```

2) 元组操作

1.元组的定义:

方式 1. C = ("Kula", "dmir", "Admir", "moon")

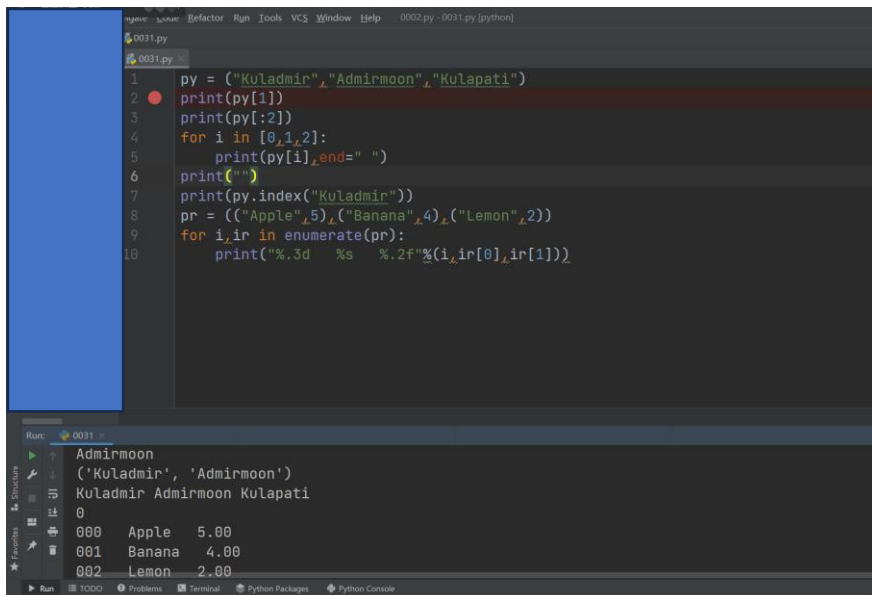
方式 2. C = "Kula", "dmir", "Admir", "moon"

如果 **只有一个元素**时: C = ("Kula",), 无内容则为空元组。

2.元组删除: del 元组名

3.元组修改: 可用赋值方式直接修改方式

4.支持索引、切片、遍历、最值计算和长度计算:



```
0031.py
1 py = ("Kuladmir", "Admirmoon", "Kulapati")
2 print(py[1])
3 print(py[:2])
4 for i in [0, 1, 2]:
5     print(py[i], end=" ")
6 print("")
7 print(py.index("Kuladmir"))
8 pr = (("Apple", 5), ("Banana", 4), ("Lemon", 2))
9 for i, ir in enumerate(pr):
10     print("%3d %s %.2f"%(i, ir[0], ir[1]))
```

Run: 0031

```
Admirmoon
('Kuladmir', 'Admirmoon')
Kuladmir Admirmoon Kulapati
0
000 Apple 5.00
001 Banana 4.00
002 Lemon 2.00
```

3) 字典操作

1.字典的定义:

方式 1. c = {'A': 'a', 'B': 'b'}

方式 2. `c = dict(zip(A,B))` #A、B 表示两个序列，其中对应内容会一一对应连接，A 中内容在前，B 中内容在后。

方式 3. `c = dict(元素键 1=元素值 1,元素键 2=元素值 2,元素键 3=元素值 3)`

列表、字典、集合不能放入字典作为元素出现，此方法会替换之前的内容。

2.字典遍历：

方式 1. `print(c[A])` #此处 A 为一个元素键

方式 2. `print(c.get("A"],["default"]))` #default 表示一个默认值，可以设置，当该字典里没有元素键 A 时，将返回 default 值

方式 3. `for item in c.items(): print(item)`

3.字典增加：`c["元素键"] = 元素值`

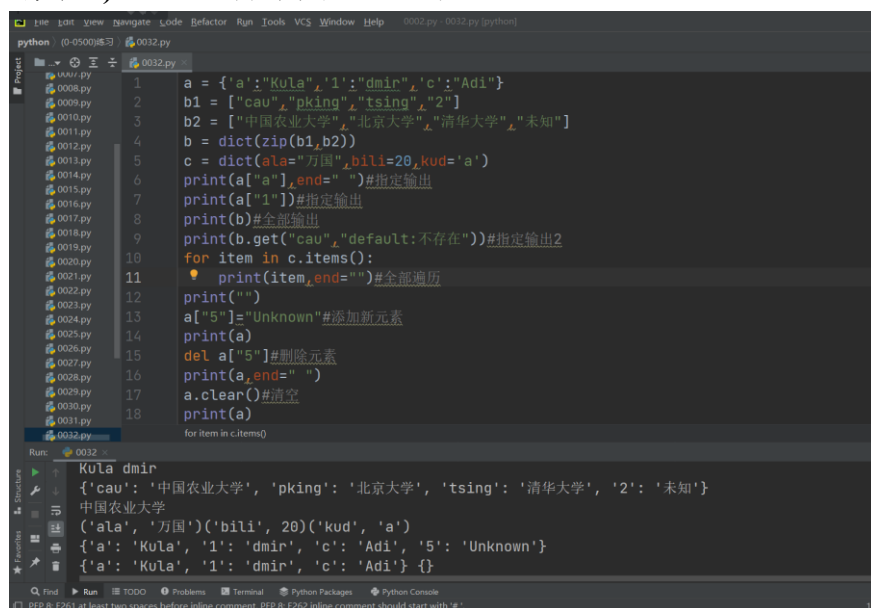
4.字典删除：

`del c` #用以删除字典

`del c["元素键"]` #用以删除字典中的一个映射

`c.clear()` #用于清除字典全部内容

`c.pop("元素键")` #用以删除字典中的一个映射



```
python 0032.py
1 a = {'a': 'Kula', '1': 'dmir', 'c': 'Adi'}
2 b1 = ["cau", "pking", "tsing", "2"]
3 b2 = ["中国农业大学", "北京大学", "清华大学", "未知"]
4 b = dict(zip(b1,b2))
5 c = dict(a1a="万国",b1l=20,kud='a')
6 print(a,"_end=") #指定输出
7 print(a["1"]) #指定输出
8 print(b) #全部输出
9 print(b.get("cau","default:不存在")) #指定输出2
10 for item in c.items():
11     print(item,end=" ") #全部遍历
12 print("")
13 a["5"]="Unknown" #添加新元素
14 print(a)
15 del a["5"] #删除元素
16 print(a,end=" ")
17 a.clear() #清空
18 print(a)
for item in c.items()

Run: 0032
Kula dmir
{'cau': '中国农业大学', 'pking': '北京大学', 'tsing': '清华大学', '2': '未知'}
中国农业大学
{'a': '万国'} ('b1l', 20) ('kud', 'a')
{'a': 'Kula', '1': 'dmir', 'c': 'Adi', '5': 'Unknown'}
{'a': 'Kula', '1': 'dmir', 'c': 'Adi'} {}
```

5.字典修改：`c["元素键"] = 元素新值`

6.字典查找：

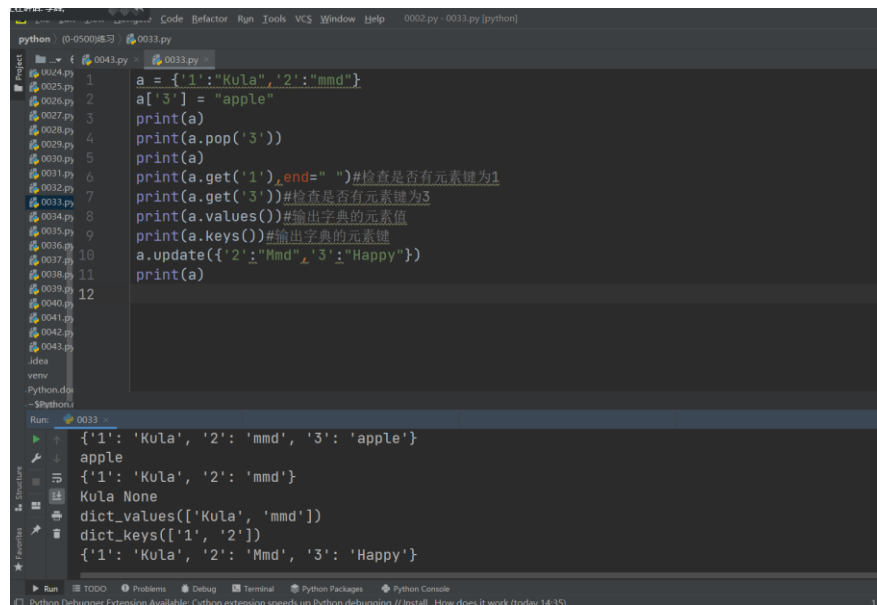
`c.get("元素键"],["default"])` #如果不存在则返回 None

`c.keys()` #输出所有的元素键

`c.values()` #输出所有的元素值

7.字典更新：

`c.update({"键": "值"})` 如果添加的键已经存在，则会替换。



```
1 a = {'1': 'Kula', '2': 'mmd'}
2 a['3'] = "apple"
3 print(a)
4 print(a.pop('3'))
5 print(a)
6 print(a.get('1'), end=" ") #检查是否有元素键为1
7 print(a.get('3')) #检查是否有元素键为3
8 print(a.values()) #输出字典的元素值
9 print(a.keys()) #输出字典的元素键
10 a.update({'2': 'Mmd', '3': 'Happy'})
11 print(a)
```

4) 集合操作

1.集合定义:

方式 1. `c = {内容 1, 内容 2, 内容 3}`

方式 2. `c = set(C)` C 为可迭代对象

集合有去重性

2.集合添加:

`c.add(添加内容)`方式。添加内容只能为字符、数字、bool 类型。

`c.update(添加内容)`方式。向集合追加的数据为序列。

3.集合删除:

`c.remove(“待删除内容”)` #可以准确删除某个内容，不存在时显示异常

`c.discard(“待删除内容”)` #可以准确删除某个内容

`c.pop()` #随机删除一个内容

`c.clear()` #清空内容

4.集合查询: `print(待查找内容 in c)` #正确返回 True, 否则返回 False

5.集合交并差:

假设 a、b 为两个集合。交: `a & b`; 并: `a | b`; 差: `a - b`

6.集合遍历: `for i in c:` 内容 #遍历顺序随机

```

python (0-0500)练习 0034.py
1 a = {'cau','tsing','peking','中国农业大学'}
2 b = "kuladmir"
3 c = set(b)
4 print(a)
5 print(b,end=' ')
6 print(c)
7 a.add('kuda')
8 print(a)
9 d = set(['中国农业大学','清华大学','北京大学'])
10 d.update(['兰州大学'])
11 print(d)
12 print(a.pop(),end=" ")
13 print(a)
14 d.clear()
15 print(d,end=' ')
16 a.remove('tsing')
17 print(a)

Run: 0034
{'tsing', 'peking', 'cau', '中国农业大学'}
kuladmir {'k', 'r', 'u', 'm', 'l', 'd', 'l', 'a'}
{'cau', 'tsing', '中国农业大学', 'peking', 'kuda'}
{'清华大学', '北京大学', '兰州大学', '中国农业大学'}
cau {'tsing', '中国农业大学', 'peking', 'kuda'}
set() {'中国农业大学', 'peking', 'kuda'}

```

总结:

	列表	元组	字典	集合
<code>print(a*n)</code>	√	√	×	×
<code>print(a+a1)</code>	√	√	×	×
<code>print(x in a)</code>	√	√	√	√
<code>for i in a:</code>	√	√	√	√
<code>print(a[n:m:d])</code>	√	√	×	×
<code>print(a.index(b))</code>	√	√	×	×
<code>print(lmm(a))</code>	√	√	×	√
<code>enumerate</code>	√	√	√	√
添加元素的方法	<code>a.append()</code> <code>a.extend()</code> <code>a.insert()</code>	<code>a+a1</code>	<code>a[“键”]=值</code> <code>a.update()</code>	<code>a.add()</code> <code>a.update()</code>
删除元素的方法	<code>del a[]</code> <code>a.remove()</code> <code>a.pop()</code>		<code>del a[“键”]</code> <code>a.pop(“键”)</code>	<code>a.remove()</code> <code>a.discard()</code> <code>a.pop()</code>

说明:

`print(a*n)`表示序列和整数乘法，即复制几遍，返回序列；

`print(a+a1)`表示序列和其自身类型相加，即拼接，返回序列；

`print(x in a)`表示某个元素是否在某一序列中，即判断是否存在，返回 True 或 False；

`for i in a:`表示遍历某个序列；

`print(a[n:m:d])`表示对序列进行切片，返回序列；

`print(a.index(b))`表示在某序列查找元素，找到返回下标，否则报错；

`print(lmm(a))`表示对序列进行 `len()`,`max()`,`min()`操作，只能在纯数字序列使用；

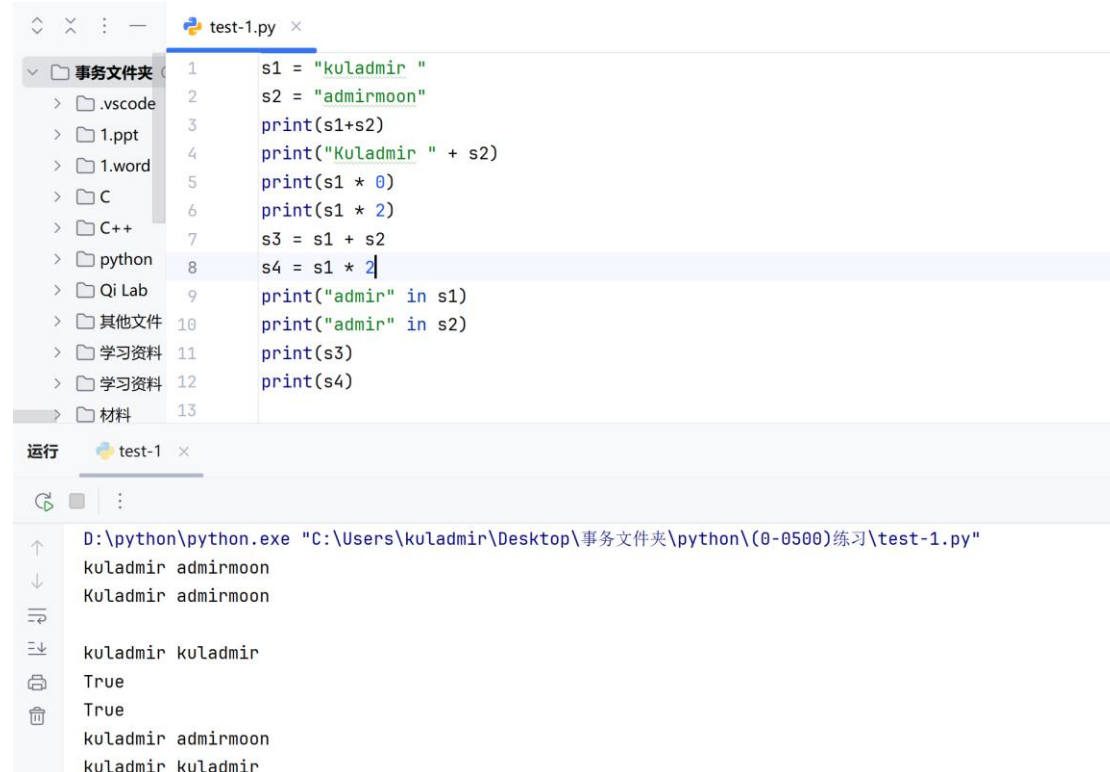
`enumerate` 表示对序列进行索引输出；

7 字符串操作

1) 字符串操作符

1. **+** 拼接 用法：两边都是字符串，+可以用来拼接；
2. **in** 检索 用法：检索一个字符串是否是另一个字符串的子串，返回 True 或者 False；
3. ***** 复制 用法：字符串 * 数字，用来多次输出某字符串，如果 n=0，则为换行，无输出。

说明： + 和 * 不是只有在 print 里才能使用，可以将其赋值给其他变量。



```
1 s1 = "kuladmir "
2 s2 = "admirmoon"
3 print(s1+s2)
4 print("Kuladmir " + s2)
5 print(s1 * 0)
6 print(s1 * 2)
7 s3 = s1 + s2
8 s4 = s1 * 2
9 print("admir" in s1)
10 print("admir" in s2)
11 print(s3)
12 print(s4)
```

运行 test-1

```
D:\python\python.exe "C:\Users\kuladmir\Desktop\事务文件夹\python\练习\test-1.py"
kuladmir admirmoon
Kuladmir admirmoon

kuladmir kuladmir
True
True
kuladmir admirmoon
kuladmir kuladmir
```

2) 字符串索引和切片

假设字符串长度为 n，则该字符串的索引长度为[0, n-1]（正向索引），也可以使用反向索引，-1 表示倒数第一个，-2 同理。正向索引和反向索引同样适用与切片。如果想要调用/访问某个位置的字符，则可以使用 `s_var[n]` #s_var 为一个字符串变量

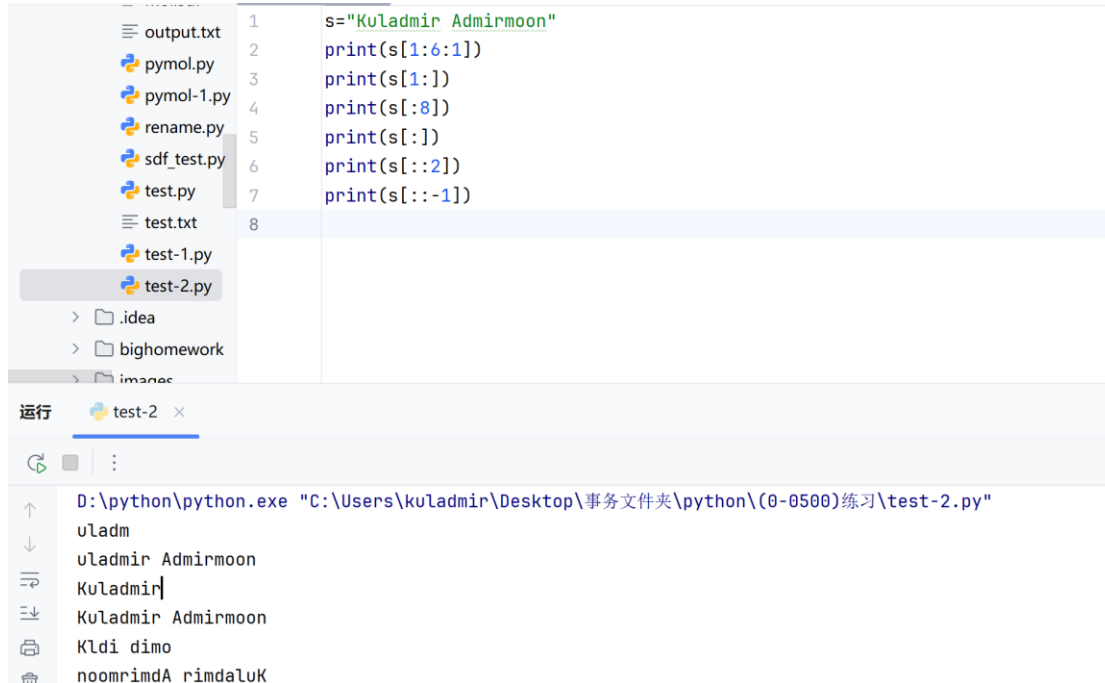
字符串切片：切片不会影响被切片的字符串，除非有重新赋值。可以使用 `s_var[s: e: p]` #s_var 为一个字符串变量

其中，s 是索引初始值，可省略；e 是索引结束值，可省略；p 是步长，可省略。故存在以下情况：`s_var = "Kuladmir Admirmoon"` # 定义一个 17 长度字符串

- 1.`print(s[1:6:1])` # 标准定义，输出 `uladm` ## 这是输出第二个字符到第五个字符，遵循区间左闭右开

- 2.`print(s[1:])` # 省略，输出 `uladmir Admirmoon` ## 默认步长为 1，输出到结束

- 3.`print(s[:8])` # 省略, 输出 **Kuladmir** ## 默认步长为 1, 从头开始输出
- 4.`print(s[:])` # 省略, 全部输出
- 5.`print(s[::2])` # 省略, 输出 **Kldi dron** ## 间隔 2 个输出
- 6.`print(s[::-1])` # 输出 **noomrimdA rimdaluK** ## 逆序输出



The screenshot shows an IDE with a file explorer on the left containing files like `output.txt`, `pymol.py`, `test.py`, etc. The main editor displays a Python script:

```

1 s="Kuladmir Admirmoon"
2 print(s[1:6:1])
3 print(s[1:])
4 print(s[:8])
5 print(s[:])
6 print(s[::2])
7 print(s[::-1])
8

```

Below the editor, the '运行' (Run) console shows the output of the script:

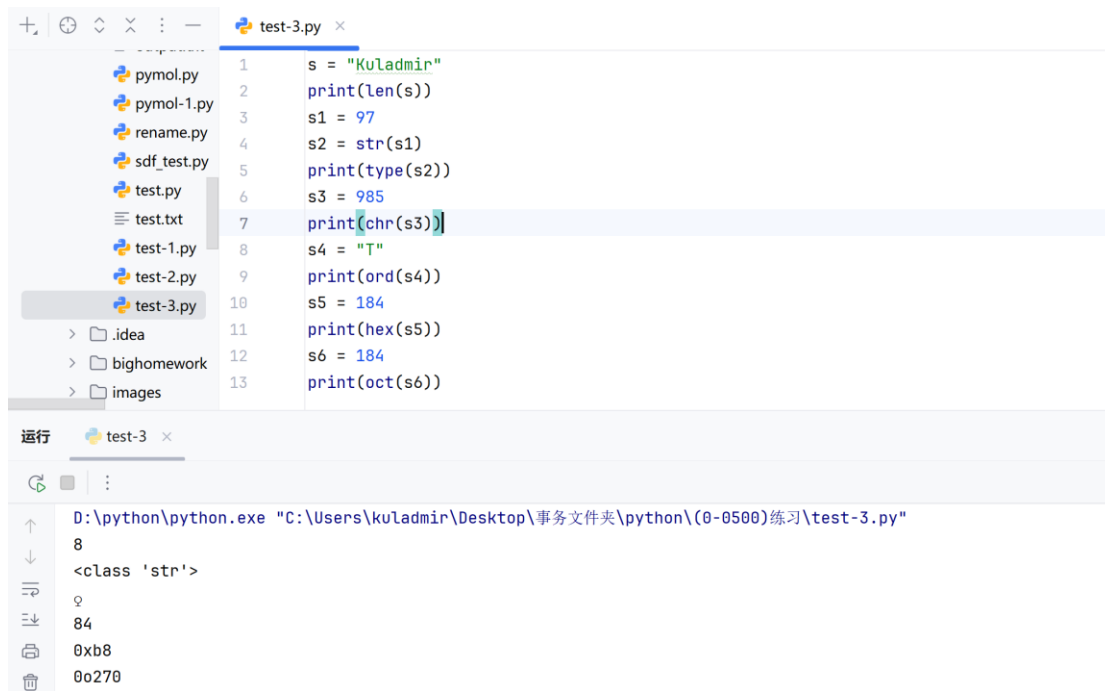
```

D:\python\python.exe "C:\Users\kuladmir\Desktop\事务文件夹\python\0-0500\练习\test-2.py"
uladm
uladmir Admirmoon
Kuladmir|
Kuladmir Admirmoon
Kldi dimo
noomrimdA rimdaluK

```

3) 字符串函数

- 1.`len(x)` 计算某个字符串的长度;
- 2.`str(x)` 将任意一个数据类型的数据, 转化为字符串;
- 3.`chr(x)` 将一个数值, 转化为 Unicode 编码里对应的字符;
- 4.`ord(x)` 将给定的字符, 转化为 Unicode 编码里对应的数值;
- 5.`hex(x)` 将一个数值转化为十六进制;
- 6.`oct(x)` 将一个数值转化为八进制。



4) 字符串处理方法（也是函数）

1. 字符串替换：`c.replace()`

语法结构：`c.replace(旧值, 新值[, 指定替换次数])`

不指定时默认全部替换。

2. 字符串分割：`c.split()`

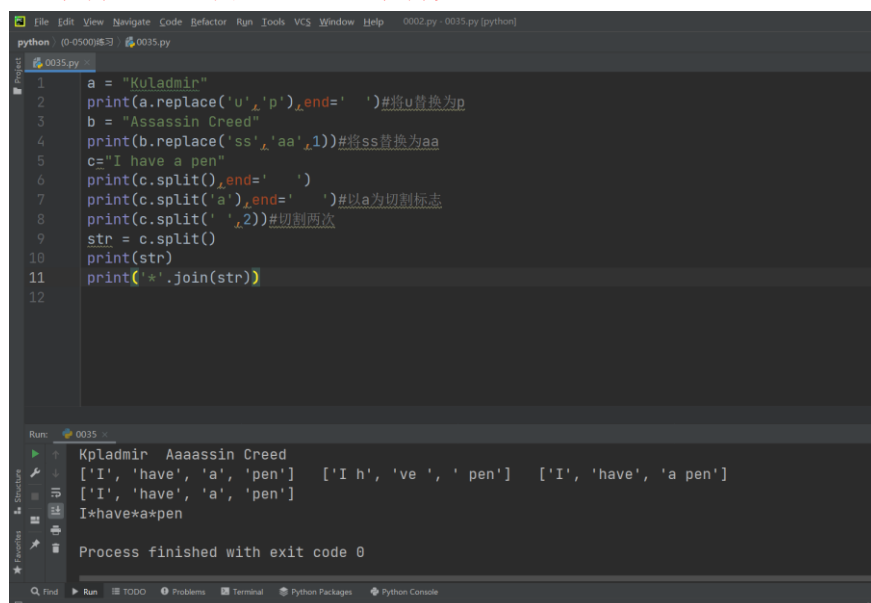
语法结构：`c.split('分割标志'[, 指定分割次数])`

不指定时默认全部分割，且默认分割标志为空格，分隔标志将消失。

3. 字符串连接：`c.join()`

语法结构：`print("结合标志".join(被分割后的字符串))`

被分割字符串中的内容必须为字符类型。



4.统计个数: **c.count()**

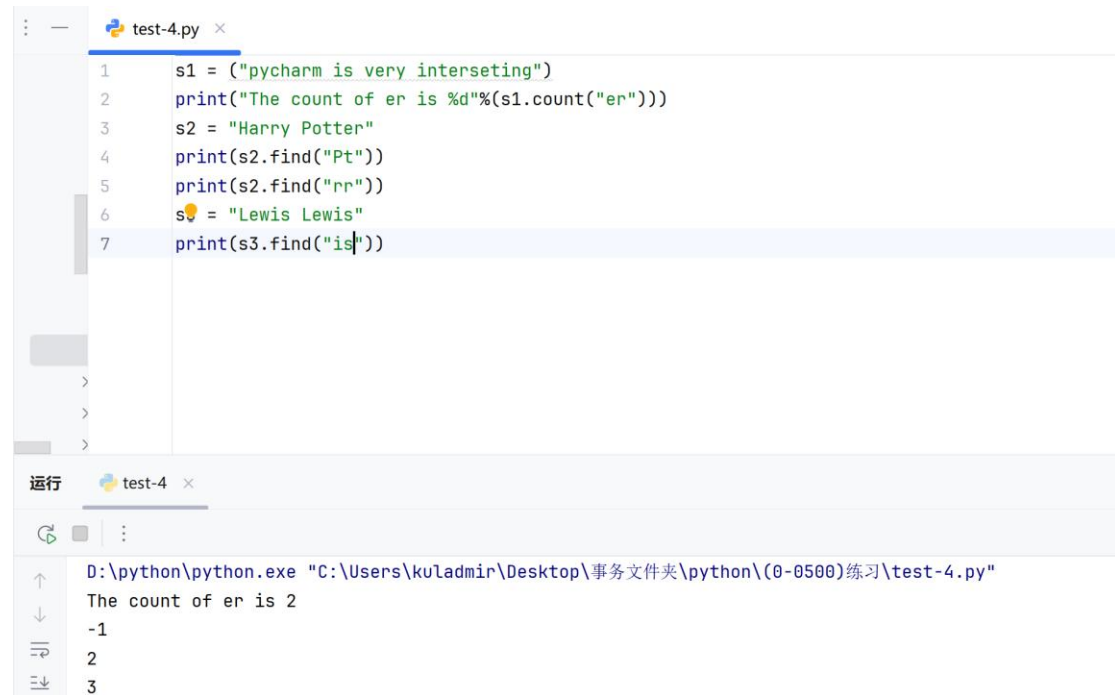
语法结构: **c.count("字符串")**

返回被查找字符串在目标字符串内的出现次数。

5.查找位置: **c.find()**

语法结构: **c.find("子串",[起始位置,终止位置])**

找到返回位置下标, 未找到返回-1; 如果有多次出现, 只会返回第一次出现的位置。



```
test-4.py
1 s1 = ("pycharm is very interseting")
2 print("The count of er is %d"%(s1.count("er")))
3 s2 = "Harry Potter"
4 print(s2.find("Pt"))
5 print(s2.find("rr"))
6 s3 = "Lewis Lewis"
7 print(s3.find("is"))

运行 test-4
D:\python\python.exe "C:\Users\ku\ladminr\Desktop\事务文件夹\python\0-0500\练习\test-4.py"
The count of er is 2
-1
2
3
```

6.字符串转换:

1> **capitalize()**: 将字符串第一个字符大写, 其余小写, 只对字母有效。

语法结构: **c.capitalize()**

2> **title()**: 将字符串中每个单词首字母大写。

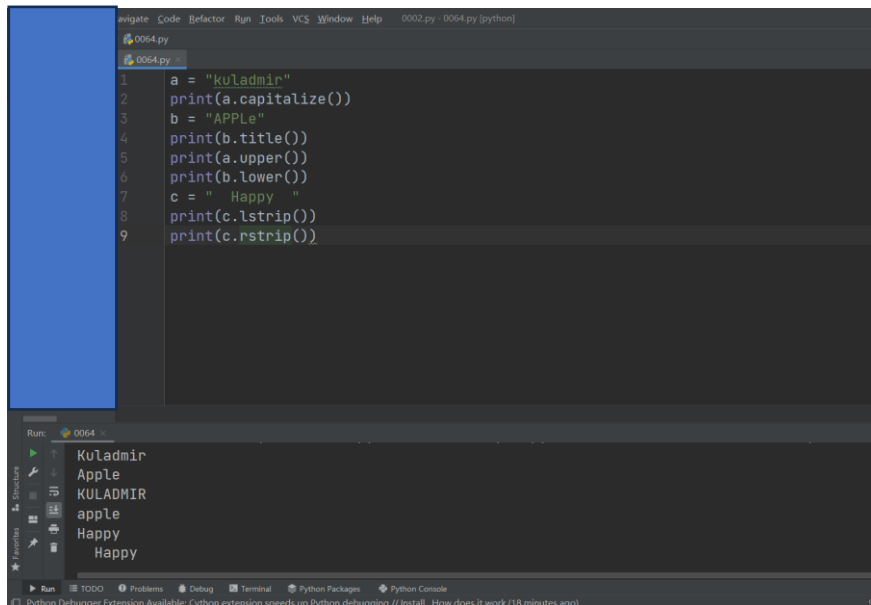
语法结构: **c.title()**

3> **upper()**和 **lower()**: 将字符串中字母全部大写/小写。

语法结构: **c.upper()/c.lower()**

4> **lstrip()**和 **rstrip()**: 用于删除字符串左边、右边字符。

语法结构: **c.lstrip()/c.rstrip()**



```
0064.py
1 a = "kuladmir"
2 print(a.capitalize())
3 b = "APPLe"
4 print(b.title())
5 print(a.upper())
6 print(b.lower())
7 c = " Happy "
8 print(c.lstrip())
9 print(c.rstrip())
```

Run: 0064

```
Kuladmir
Apple
KULADMIR
apple
Happy
Happy
```

5> **strip()**: 用于删除字符串两侧指定字符。

语法结构: **c.strip(['删除指定字符'])**

如果不传参, 则去除左右两侧空白。可以传多个字符。同时可以控制删除左右多少个字符。

c.[l/r]strip('\$') # l 表示删除左侧字符, r 表示删除右侧字符

如果在 **strip()** 中填写多个字符, 则会将两边内容中和填写的内容相同的内容删除, 直到遇到为填写的未知。

6> **rjust()**、**ljust()** 和 **center()**: 用于左右对齐和两边对齐。

语法结构: **c.rjust(n, '对齐字符')/c.ljust(n, '对齐字符')/center(n, '对齐字符')**

如果不传对齐字符, 则用空格对齐。如果 n 的值小于字符串长度, 则默认返回原字符串。

7> **startswith()** 和 **endswith()**:

语法结构: **c.startswith("待判断字符", 开始位置, 终止位置)/c.endswith("待判断字符", 开始位置, 终止位置)**

用以判断是否以.....开头(结尾), 正确返回 **True**, 否则返回 **False**, 范围为左闭右开。

```
test-5.py x
1 s1 = "--MuMu is a good game--"
2 print(s1.strip("-"))
3 print(s1.strip("/"))
4 s2 = "-/* MuMu is a good game--"
5 print(s2.strip("-/"))
6 print(s1.rjust( _width: 30, _fillchar: "*"))
7 print(s1.ljust( _width: 30, _fillchar: "*"))
8 print(s1.ljust(10))
9 print(s1.center( _width: 30, _fillchar: "*"))
10 print(s1.startswith("***"))
11 print(s1.endswith("--"))

运行 test-5 x
D:\python\python.exe "C:\Users\kuladmir\Desktop\事务文件夹\python\0-0500\练习\test-5.py"
MuMu is a good game
--MuMu is a good game--
* MuMu is a good game
*****--MuMu is a good game--
--MuMu is a good game-----
--MuMu is a good game--
****--MuMu is a good game-----
False
True
```

8> **isalpha()**:

语法结构: **c.isalpha()**

用以判断字符串是否全部由字母组成, 是返回 **True**, 否则返回 **False**。

9> **isdigit()**:

语法结构: **c.isdigit()**

用以判断字符串是否全部由数字组成, 是返回 **True**, 否则返回 **False**。

10> **isalnum()**:

语法结构: **c.isalnum()**

用以判断字符串是否全部由数字或字母组成, 是返回 **True**, 否则返回 **False**。

11> **isspace()**:

语法结构: **c.isspace()**

用以判断字符串是否全部由空格组成, 是返回 **True**, 否则返回 **False**。

12> **isnumeric()**和 **isdecimal()**:

语法结构: **c.isnumeric()/c.isdecimal()**

用以判断字符串是否由数字, 是返回 **True**, 否则返回 **False**。注意: **isnumeric()** 包含了 **Unicode** 的所有数字字符: 123/一二三/上标等; 而 **isdecimal()** 只包含了数字。

13> **isidentifier()**:

语法结构: **c.isidentifier()**

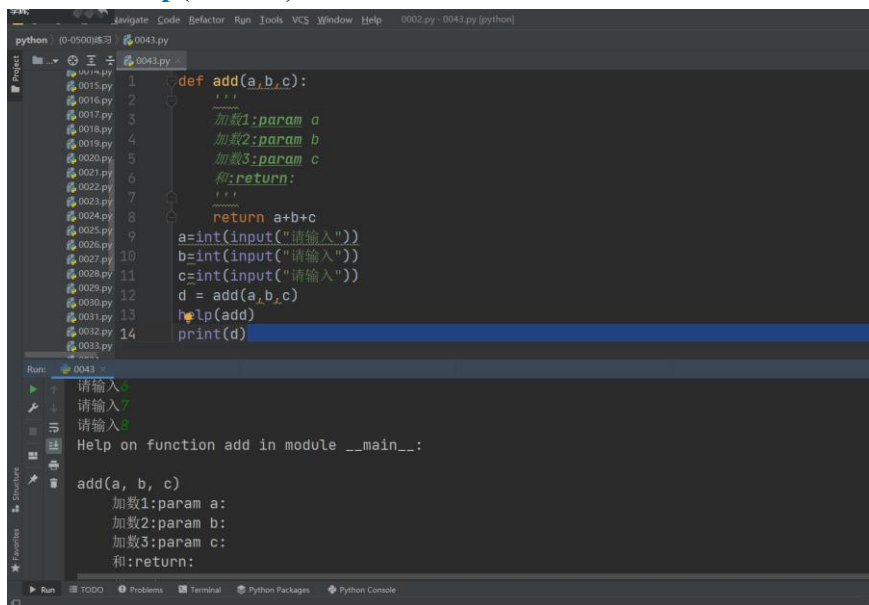
用以判断是否为标识符, 是返回 **True**, 否则返回 **False**。

8 函数

1) 函数的定义和调用

def 函数名(函数参数): `[" 注释 "]` 函数体

在主函数里使用 **help**(函数名), 可以显示函数的注释



2) 传值调用和传址调用

当给函数的实参传入不可变对象, 则为传值调用, 形参变化和实参无关; 如果给函数的实参传入可变对象, 则为传址调用, 形参变化会影响实参变化。

3) 位置参数函数和关键字参数函数

在使用位置参数和关键字参数时, 需要注意: **实参数量和形参数量相等, 位置需要和定义时一致**。使用关键字参数时, 由于使用了键值对, 因此在传参时不需要对齐, 而可乱序, **但是关键字参数必须在位置参数后**。

【kimi 老师的详细解释:】

1. 先算「数量」

无论位置还是关键字, 最终传进去的实参总数必须等于形参总数 (除非有缺省值)。
例:

```
Python
def f(a, b, c=0): ...
f(1, 2)          # OK, 两个位置实参 + 一个缺省值, 总数=3
f(1, 2, 3, 4)    # TypeError, 实参 4 > 形参 3
```

2. 再说「位置」

纯位置参数必须一一对应, 不能错位。

```
Python
def g(x, y): ...
g(10, 20)        # x=10, y=20
g(20, 10)        # x=20, y=10, 顺序颠倒结果就反了
```

3. 关键字参数可以「乱序」

因为带了名字, 解释器按名字匹配, 所以顺序无所谓:

```
Python
g(y=20, x=10)    # 与 g(10, 20) 结果相同
```

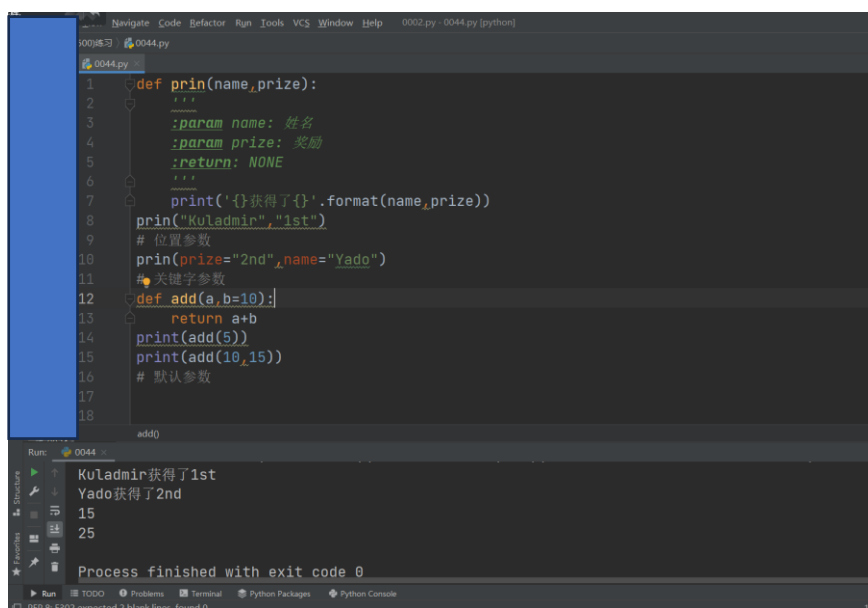
4. 关键字参数必须「在后」

一旦某个实参用了关键字写法, 它之后的所有实参都必须是关键字, 不能回到位置写法。

```
Python
def h(a, b, c, d): ...
h(1, 2, c=3, d=4) # OK
h(1, b=2, 3, 4)   # SyntaxError: 3 又成了位置参数, 在关键字 b=2 之后出现, 非法
```

4) 默认参数函数

在函数定义时, 可以让形参默认为一个数, 因此传参时可以省略。默认值必须在非默认值之后。



```
1 def prin(name,prize):
2     """
3     :param name: 姓名
4     :param prize: 奖励
5     :return: NONE
6     """
7     print('{}获得了{}'.format(name,prize))
8     prin("Kuladmir","1st")
9     # 位置参数
10    prin(prize="2nd",name="Yado")
11    # 关键字参数
12    def add(a,b=10):
13        return a+b
14    print(add(5))
15    print(add(10,15))
16    # 默认参数
17
18    add()
```

Run: 0044

Kuladmir获得了1st
Yado获得了2nd
15
25

Process finished with exit code 0

5) 函数返回值

如果函数执行后，需要返回，则应该使用 **return**，并在后面写下函数体内定义的变量。同时，函数外应该有对应数量的变量用以去接收函数的返回值。**注意，一旦函数执行了 return 语句，则认为函数已经结束。**

6) 函数变量作用域

函数内部定义的变量，即为局部变量；函数外定义的变量，即为全局变量。局部变量无法在范围外进行使用，否则会产生错误。

如果局部变量和全局变量名一致，对局部变量或对全局变量的值的改变，不会影响对应的变量。

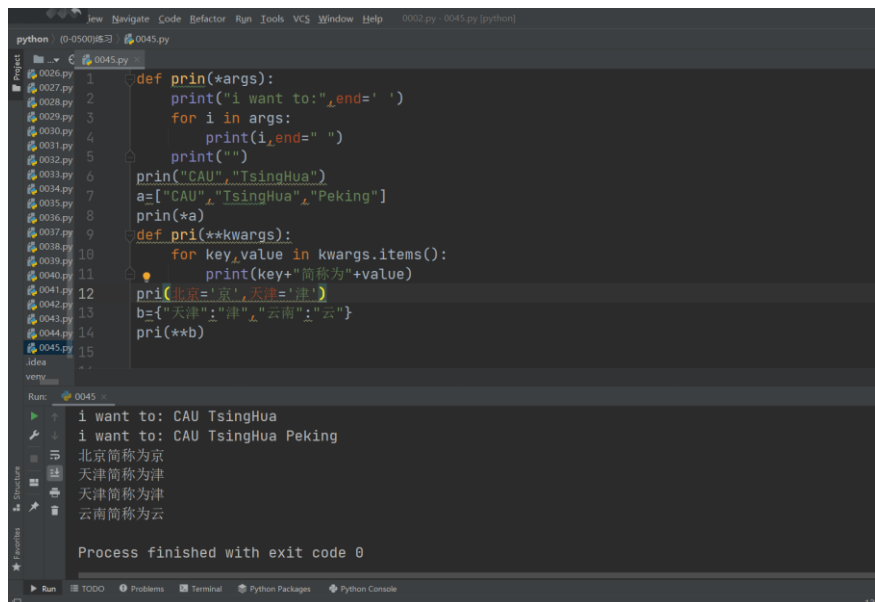
如果想在函数内对全局变量的值进行调整，则需要在变量前加 **global**。

7) 不定参数函数

函数定义过程中，可以将形参设为不定形式。

1.***args** 形式：表示接收任意多参数并放入一个元组。可以将一个列表作为参数传入函数。

2.****kwargs** 形式：表示接收任意多参数并放入一个字典。可以将一个字典作为参数传入函数。



```
python (0-0500)练习 0045.py
1 def prin(*args):
2     print("i want to:"_end=' ')
3     for i in args:
4         print(i,end=" ")
5     print("")
6     prin("CAU","TsingHua")
7     a=["CAU","TsingHua","Peking"]
8     prin(*a)
9     def pri(**kwargs):
10         for key,value in kwargs.items():
11             print(key+"简称为"+value)
12     pri(北京='京',天津='津')
13     b={"天津":"津","云南":"云"}
14     pri(**b)
15
```

Run: 0045

```
i want to: CAU TsingHua
i want to: CAU TsingHua Peking
北京简称为京
天津简称为津
天津简称为津
云南简称为云
Process finished with exit code 0
```

8) 递归函数和函数嵌套

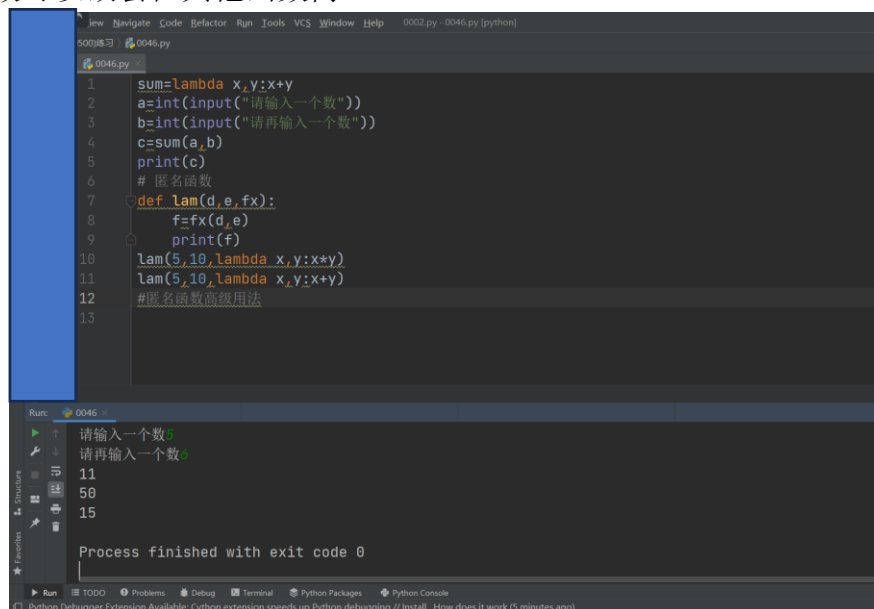
函数嵌套：在函数内部使用嵌套其他函数；

递归函数：在函数内部继续调用自身。**注意，递归必须有结束条件，且每次循环都要向结束条件靠近。**

9) 匿名函数

定义方式：**result = lambda 参数:表达式** #result 负责对该函数进行调用

匿名函数可以嵌套在其他函数内。



```
5000练习 0046.py
1 sum=lambda x,y:x+y
2 a=int(input("请输入一个数"))
3 b=int(input("请再输入一个数"))
4 c=sum(a,b)
5 print(c)
6 # 匿名函数
7 def lam(d,e,fx):
8     f=fx(d,e)
9     print(f)
10 lam(5,10,lambda x,y:x*y)
11 lam(5,10,lambda x,y:x+y)
12 #匿名函数高级用法
13
```

Run: 0046

```
请输入一个数
请再输入一个数
11
50
15
Process finished with exit code 0
```

10) 高阶函数

内置函数 **reduce()** 函数结构：**reduce(fx,para)**

fx 表示一个二元函数，可以为匿名函数，para 表示一个集合。改函数可以持续执行 fx 函数，直至 para 中的参数被全部使用。

内置函数 **filter()** 函数结构：**filter(fx,para)**

fx 表示一个一元函数，且属于判断函数，para 表示一个集合。改函数可以持续执行 fx 函数，直至 para 中的参数被全部使用。

```
1 from functools import reduce
2 def add(a,b):
3     return a+b
4 c = reduce(add,[1,2,3,4,5,6,7])
5 print(c)
6 def odd(a):
7     return a%2==0
8 d = filter(odd,[1,2,3,4,5,6,7])
9 print(list(d))
```

28
[2, 4, 6]
Process finished with exit code 0

9 文件操作

1) 文件的打开: `file = open(name,mode)`

name 表示文件名, mode 表示访问模式 (读 r, 写 w, 追加 a)

r: 文件只读, 必须存在; w: 文件可读写, 不存在时会新建, 存在时会清除内容。

a: 文件追加, 文件不存在时新建, 存在时追加。

2) 文件的关闭: `file.close()`

文件每次打开后, 都需要在使用后关闭。

3) 文件读取 (文件操作模式需要为 r)

file.read(num)

num 表示读取的字符串长度, 如果没有传入, 则读取所有内容, 如果字符被全部读取完, 则返回空字符串。

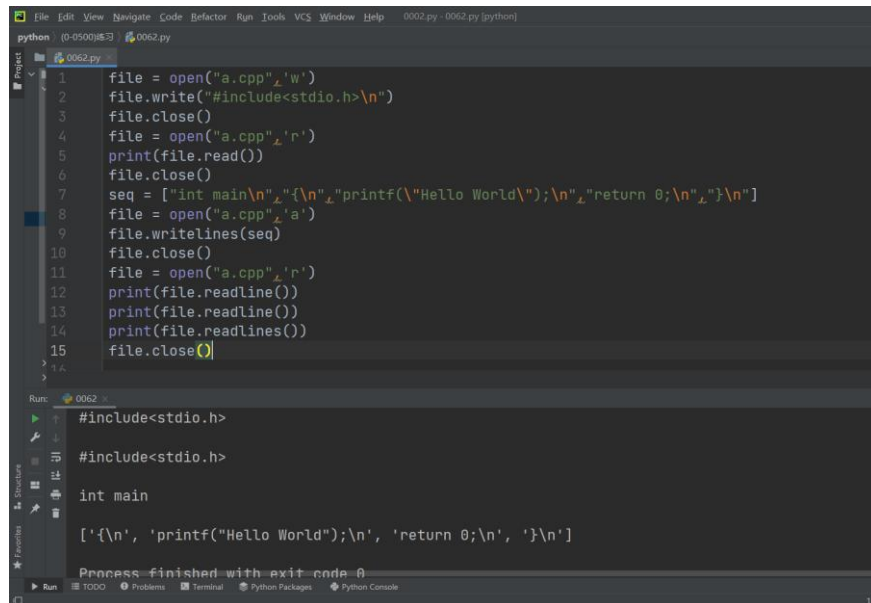
file.readline() 可以读出一行内容, 读取后文件指针移动一行

file.readlines() 可以读出每一行内容, 每行内容会放入一个列表。

4) 文件写入 (文件操作模式需要为 w)

file.write(内容)

file.writelines(p) p 表示一个多行内容



```
1 file = open("a.cpp", 'w')
2 file.write("#include<stdio.h>\n")
3 file.close()
4 file = open("a.cpp", 'r')
5 print(file.read())
6 file.close()
7 seq = ["int main\n", "{\n", "printf(\"Hello World\");\n", "return 0;\n", "}\n"]
8 file = open("a.cpp", 'a')
9 file.writelines(seq)
10 file.close()
11 file = open("a.cpp", 'r')
12 print(file.readline())
13 print(file.readline())
14 print(file.readlines())
15 file.close()
```

Run: 0062

```
#include<stdio.h>

#include<stdio.h>

int main

[ '{\n', 'printf("Hello World");\n', 'return 0;\n', '}\n']

Process finished with exit code 0
```

5) 文件指针偏移

file.seek(offset[,hen]) 其中最至少有一个参数 offset，最多两个。

offset 表示开始的偏移值。hen 表示参考点，默认为 0，表示从文件开头计算，如果为 1，表示从当前位置开始，如果为 2，表示从文件末开始。

6) 其余文件操作

先引入头文件 **import os**

os.rename(旧文件名,新文件名) 可以实现文件名修改，也可以移动文件位置

os.remove(文件名) 可以实现文件的删除，但是不能删除文件夹

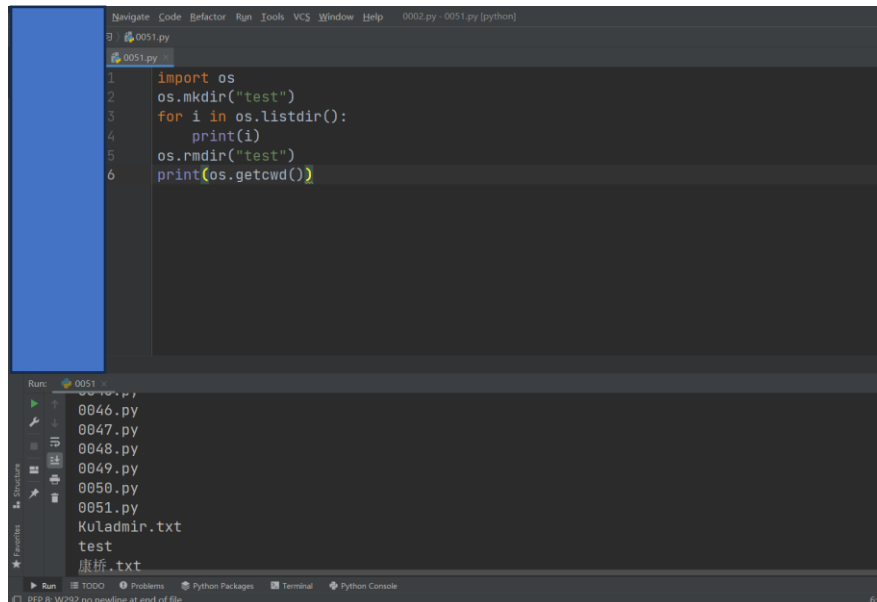
7) 文件夹操作

os.mkdir(文件夹名) 用以创建一个文件夹，但不能创建多级

os.rmdir(文件夹名) 用以删除一个文件夹

os.getcwd() 用以显示当前工作目录

for I in os.listdir():print(i) 用以列举该目录下所有文件



```
1 import os
2 os.mkdir("test")
3 for i in os.listdir():
4     print(i)
5 os.rmdir("test")
6 print(os.getcwd())
```

Run: 0051

0046.py
0047.py
0048.py
0049.py
0050.py
0051.py
Kuladmir.txt
test
康桥.txt

10 面向对象程序设计 I

1) 类和对象:

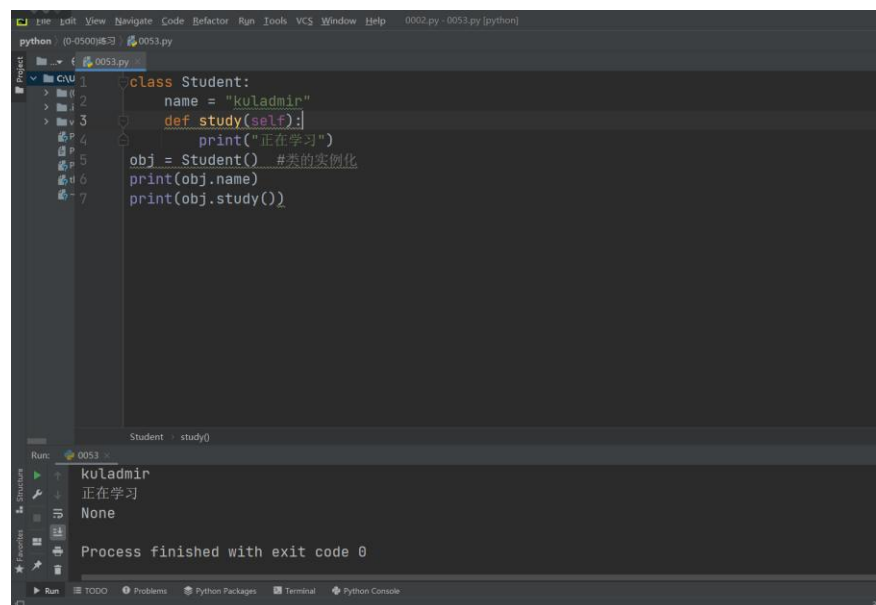
类: 用来描述相同的属性和方法的对象集合;

对象: 某个具体的实物。

2) 类的使用: class 类名: 类体

类名通常为大写字母开头, 类的成员包括成员属性和描述操作函数的函数成员。

3) 类的实例化:



```
1 class Student:
2     name = "kuladmir"
3     def study(self):
4         print("正在学习")
5 obj = Student() #类的实例化
6 print(obj.name)
7 print(obj.study())
```

Run: 0053

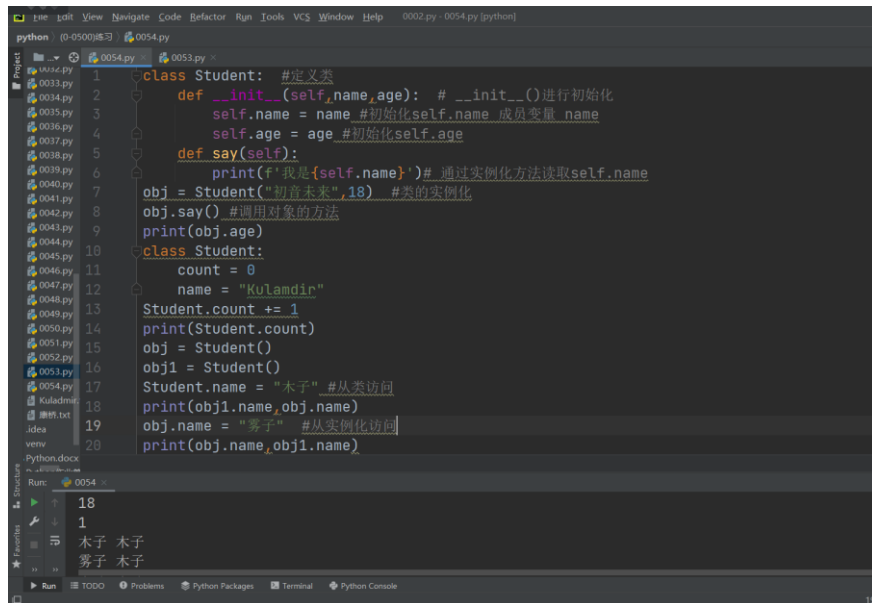
kuladmir
正在学习
None

Process finished with exit code 0

可以使用 `def __init__(self,变量名):` 的方法进行从初始化。

类实例化后, 如果使用类, 则会自动调用一次 `def __init__(self)` 函数, 之后使用类将不会再调用。

也可以通过类名去修改类的属性值, 需要注意是通过类修改还是实例化修改。



```
python / 0-0500练习 0054.py
1 class Student: #定义类
2     def __init__(self,name,age): # __init__()进行初始化
3         self.name = name #初始化self.name 成员变量 name
4         self.age = age #初始化self.age
5     def say(self):
6         print(f'我是{self.name}')# 通过实例化方法读取self.name
7 obj = Student("初音未来",18) #类的实例化
8 obj.say() #调用对象的方法
9 print(obj.age)
10 class Student:
11     count = 0
12     name = "Kulamdir"
13     Student.count += 1
14     print(Student.count)
15     obj = Student()
16     obj1 = Student()
17     Student.name = "木子" #从类访问
18     print(obj1.name,obj.name)
19     obj.name = "雾子" #从实例化访问
20     print(obj.name,obj1.name)
```

Run: 0054

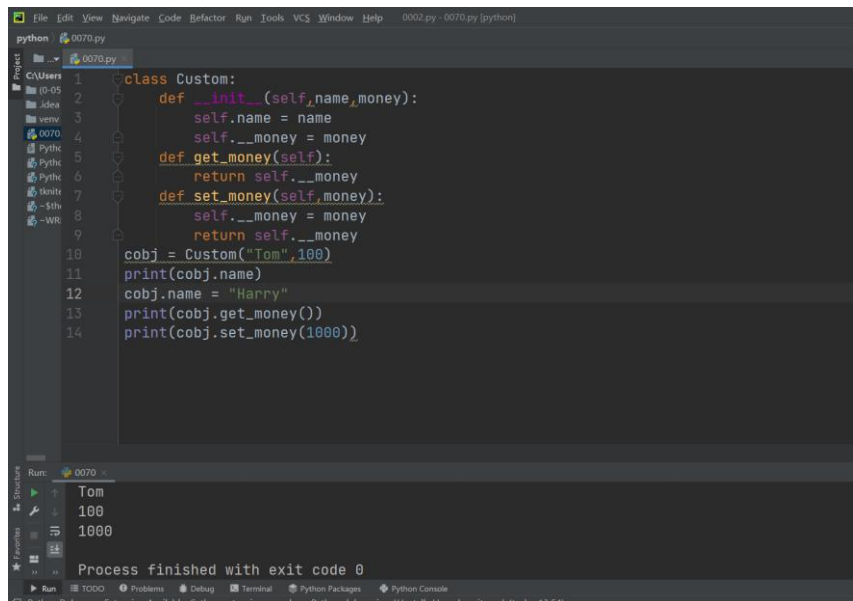
18
1
木子 木子
雾子 木子

4) 定义类的私有属性:

定义方式: `__成员名` #不能被直接使用, 需要使用 `get` 或 `set` 等使用。

5) 私有方法的定义和访问

定义方式: `def __方法名(self):` #不能被直接使用, 需要间接使用



```
python / 0070.py
1 class Custom:
2     def __init__(self,name,money):
3         self.name = name
4         self.__money = money
5     def get_money(self):
6         return self.__money
7     def set_money(self,money):
8         self.__money = money
9         return self.__money
10 cobj = Custom("Tom",100)
11 print(cobj.name)
12 cobj.name = "Harry"
13 print(cobj.get_money())
14 print(cobj.set_money(1000))
```

Run: 0070

Tom
100
1000

Process finished with exit code 0

类方法: 类对象拥有的方法, 其参数第一个必须为类对象 `cls`。

静态方法: 不需要传入默认参数, 即可使用。

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help 0002.py - 0004.py (python)
python (0-0500)练习 0004.py
1 class People:
2     con = "China"
3     @classmethod #类方法
4     def getCon(cls):
5         return cls.con
6     @staticmethod
7     def GetCon():
8         return People.con
9
10 p = People
11 print(p.con)
12 print(p.getCon()) #通过实例化引用 类cls此时被代替为p的类,即为People,然后执行People.con
13 print(People.getCon()) #通过类名引用 类cls即为类People,然后执行People.con
14 print(p.GetCon())
15 print(People.GetCon())

Run: 0004
China
China
China
China
China
Process finished with exit code 0
```

6) 类的继承: (继承: 描述事物之间所属关系)

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help 0002.py - 0007.py (python)
python (0-0500)练习 0007.py
1 class Animal():
2     def __init__(self,name,age):
3         self.name = name
4         self.age = age
5     def eat(self,food):
6         print(f"它是{self.name}, 年龄是{self.age}, 爱吃{food}")
7 class Dog(Animal):
8     def __init__(self,name,age,color):
9         Animal.__init__(self,name,age)
10        self.color = color
11    def eat(self,food):
12        Animal.eat(self,food)
13        print(f'一只名为{self.name},颜色是{self.color},爱吃{food}')
14
15 lala = Animal("laplado",3)
16 lala.eat("Fish")
17 jinmao = Dog("jinmao",2,"Golden")
18 jinmao.eat("Meat")

Run: 0007
它是laplado, 年龄是3, 爱吃Fish
它是jinmao, 年龄是2, 爱吃Meat
一只名为jinmao,颜色是Golden,爱吃Meat
Process finished with exit code 0
```

7) 模块

模块是一组 python 源程序代码, 能够提供计算功能的代码单元成为模块, 导入并使用这些模块的程序, 称为客户程序。模块和客户程序之间遵守应用程序接口 (API), 其中描述了模块中提供得到函数或类的调用方法。

每个模块的定义中, 都包含一个记录模块名称的变量 `__name__`, 它是 Python 内置属性, 用以表示当前模块的名, 如果 py 文件被作为模块调用, `__name__` 的属性值为模块文件的名, 如果模块独立运行, 其属性值为 `__main__`。

8) 模块导入

import 模块名

from 模块名 import 函数名

from 模块名 import 函数名 as 可以将改模块名改为 as 后的名

9) 包

包是一个有层次的文件目录结构，通常将一组功能相近的模块组织在一个目录下，定义了模块和子包组成的 Python 应用程序执行环境。

11 数据库和 python 联用

import pymysql as pm #进行头文件引入

1) 数据库连接

db=pymysql.connect(host='localhost',user='root',password='',db='jxgl',charset='utf8') #打开数据库连接

cursor = db.cursor() #创建游标对象

cursor.execute("SELECT VERSION()") #执行 SQL 查询

data = cursor.fetchone() #获取单条数据

print("Database version : %s " % data)

db.close() #关闭连接

2) 数据表创建

cursor.execute("SQL 语句") #也可以为一个 SQL 语句的字符串

3) 表的操作

插入多条数据:

cursor.executemany("insert into books(name, category, price, publish_time) values (%s,%s,%s,%s)", data) #data 为一个序列，包含其中数据

db.commit() #数据提交

12 FastAPI 开发

1) FastAPI 数据请求方式

路径参数(Path)

需要使用带装饰器的@**app.get("/item/{id_value}")**方法注册路由，用方法里的参数接受 URL 地址传递过来的参数。用于比较 URL 地址中传递的数据，匹配成功后，执行紧贴路由后的内容。

有类型的路径参数

相对于简单路径参数而言，需要指定变量类型，例如 **int float str** 等，可以使用 Python 类型提示的语法。

路由访问顺序

当 FastAPI 程序中使用装饰器注册路由路径时，路由路径按照代码中的顺序保存到后端服务器的路由表中，当后端服务器注册了多个路由路径时，FastAPI 为访问 URL 路径的匹配提出的访问顺序要求。

使用枚举(Enum)可以对路径参数中接受的值进行验证,并转换为枚举类型的数据。

查询参数(Query):

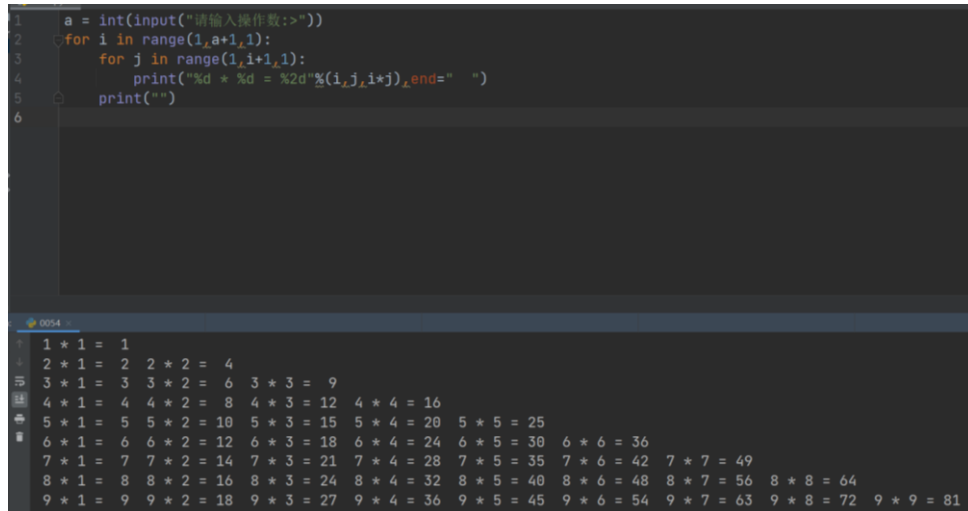
网络跟踪器(Cookie):

请求头(Header):

EX.部分例题

1.输出乘法表

```
1 a = int(input("请输入操作数:>"))
2 for i in range(1,a+1,1):
3     for j in range(1,i+1,1):
4         print("%d * %d = %2d"%(i,j,i*j),end=" ")
5     print("")
6
```



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

2.判断闰年

```
1 a = int(input("请输入一个年份:>"))
2 if((a % 4 == 0 and a % 100 != 0) or (a % 400 == 0)):
3     print("是闰年")
4 else:
5     print("不是闰年")
```



```
请输入一个年份:> 2023
不是闰年
```

3.输出斐波那契数列


```

1  a = int(input("请输入要计算第几个数:>"))
2  b = 1
3  c = 1
4  if(a==1 or a==2):
5      print(1)
6  else:
7      for i in range(3,a+1,1):
8          d = c + b
9          b = c
10         c = d
11     print(d)
12
else

```

0055

请输入要计算第几个数:>

8

4.分数计算器

```

1  a = []
2  length = int(input("请输入数据个数:>"))
3  print("请输入%d个数据"%length)
4  for i in range(length):
5      b = float(input())
6      a.append(b)
7  print("总和为%.2f"%sum(a))
8  print("最大值为%.2f"%max(a))
9  print("最小值为%.2f"%min(a))
10 print("平均值为%.2f"%(sum(a)/length))

```

0056

请输入4个数据

10

9.9

9.4

9.8

总和为39.30

最大值为10.00

最小值为9.60

平均值为9.82

5.匿名函数高级应用

```

1  def function(a,b,fx):
2      print("%.2f"%fx(a,b))
3  function(5,10,lambda a,b:a+b)
4  function(5,10,lambda a,b:a*b)
5  function(5,10,lambda a,b:a/b)
6  function(5,10,lambda a,b:a-b)

```

0057

15.00

50.00

0.50

-5.00

6.递归求阶乘

```
1 def matiy(a):
2     if a == 1:
3         return 1
4     else:
5         return matiy(a-1) * a
6 b = int(input("请输入操作数:>"))
7 a = matiy(b)
8 print(a)
```

0058

请输入操作数:>10

3628800

7.根据条件求一个数

```
1 for i in range(1,50):
2     for j in range(1,50):
3         if ((i+j)*(i-j)==68):
4             print(i,j)
5             break
6 a = 18*18-168
7 print("原数字是:>%d"%a)
```

0072

18 16

原数字是:>156

8.判断素数