# 第二章 python变量、输出和输入、数字、字符串

本节所讲内容：

2.1 python 变量使用

2.2 python 输出、输入

2.3 python 数字类型

2.4 python 字符串类型

## 2.1 python变量

### 2.1.1 变量以及类型

1、变量的定义

在程序中，有时我们需要对2个数据进行求和，那么该怎样做呢？

大家类比一下现实生活中，比如去超市买东西，往往咱们需要一个菜篮子，用来进行存储物品，等到所有的物品都购买完成后，在收银台进行结账即可。



如果在程序中，需要把2个数据，或者多个数据进行求和的话，那么就需要把这些数据先存储起来，然后把它们累加起来即可，变量就是用来存东西的。

在Python中，存储一个数据，需要一个叫做变量的东西，如下示例:

num1 = 100 #num1就是一个变量，就好比一个小菜篮子。

num2 = 87 #num2也是一个变量。

result = num1 + num2 #把num1和num2这两个"菜篮子"中的数据进行累加，然后放到 result变量中。

说明:

所谓变量，可以理解为菜篮子，如果需要存储多个数据，最简单的方式是有多个变量；

程序就是用来处理数据的，而变量就是用来存储数据的；

变量起名要有意义。

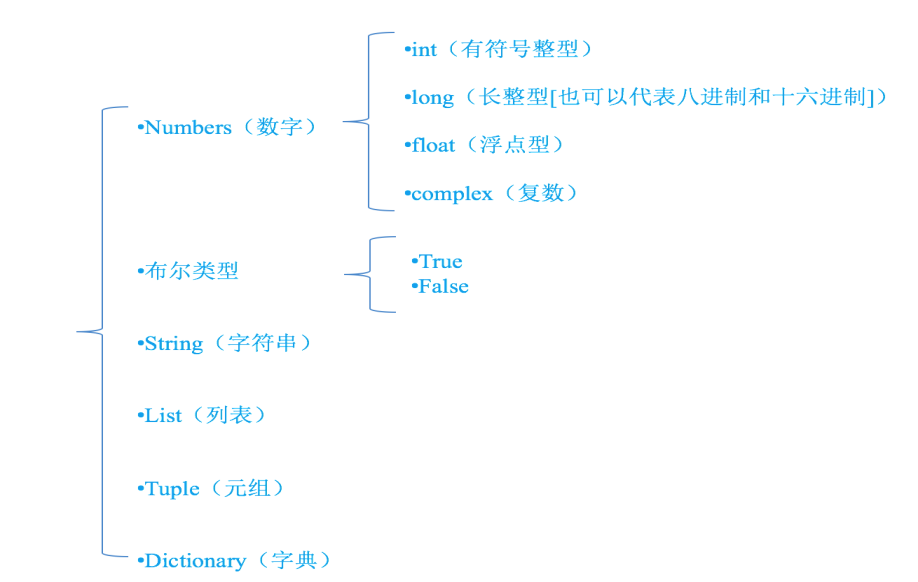
2、变量的类型

生活中的“类型”的例子:



程序中:

为了更充分的利用内存空间以及更有效率的管理内存，变量是有不同的类型的，如下所示:



怎样知道一个变量的类型呢？

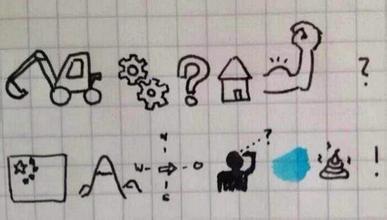
在python中，只要定义了一个变量，而且它有数据，那么它的类型就已经确定了，不需要咱们开发者主动的去说明它的类型，系统会自动辨别。

可以使用type(变量的名字)，来查看变量的类型。以上知识让我们知道了，什么是变量，变量的类型是什么？接下来让我们学习，变量的起名，以及标识符相关规范。

### 2.1.2 标示符和关键字

1、 标示符

什么是标示符，看下图:



火星文：挖掘机技术哪家强，中国山东找蓝翔。

标示符是自己定义的，直指程序员的本心，就是开发人员在程序中自定义的一些符号和名称，如变量名 、函数名等。

2、 标示符的命名规则

标示符由字母、下划线和数字组成，且数字不能开头。

思考：下面的标示符哪些是正确的，哪些不正确为什么

fromNo12

from#12 #错误

my\_Boolean

my-Boolean #错误

Obj2

2ndObj #错误

myInt

test1

Mike2jack

My\_tExt

\_test

test!32 #错误

haha(da)tt #错误

int #错误

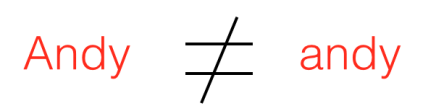
jack\_rose

jack&rose #错误

GUI

G.U.I #错误

python中的标识符是区分大小写的：



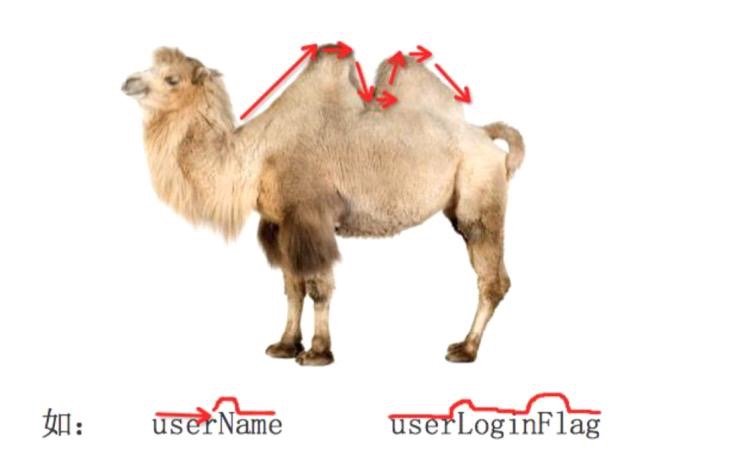
3、 命名规则和方法

见名知意：

起一个有意义的名字，尽量做到看一眼就知道是什么意思(提高代码可读性) 。

比如: 名字 就定义为 name , 定义学生 用 student。

驼峰命名法：



小驼峰式命名法（lower camel case）： 第一个单词以小写字母开始；第二个单词的首字母大写，例如：myName、aDog

大驼峰式命名法（upper camel case）： 每一个单字的首字母都采用大写字母，例如：FirstName、LastName

不过在程序员中还有一种命名法比较流行，就是用下划线“\_”来连接所有的单词，比如send\_buf

Python推荐就是用下划线“\_”来连接所有的单词。

注意：一定要注意关键字命名。

查看关键字：

>>> import keyword

>>> keyword.kwlist

['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']

上面咱们学习了命名，接下来学习变量的赋值。

4、变量赋值的三种方式

传统赋值

name = “xuegod”

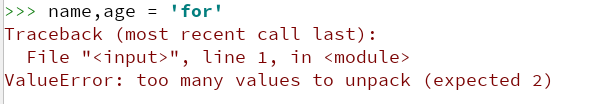
链式赋值

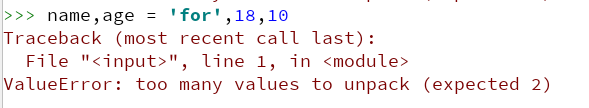
name = user = “xuegod”

序列解包赋值

name,age = “xuegod”，10

注意：两边的变量和常量要对等。





### 2.1.3 python 虚拟机（面试题）

1、垃圾回收机制

当一个常量被生成，会占用一份内存，这时候如果有变量指向该常量，那么该常量的引用计数为1，python 虚拟机规定，当一个常量的引用计数为0，也就是没有变量指向的时候，该常量占用的内存会被回收。

例如： name = user = ‘xuegod’

就是变量name，变量user 指向常量‘xuegod’这个内存。

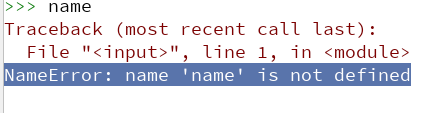
如何查看一个变量的id呢？

内置函数id（）

Id(name) 和 id（user）对比，你会惊讶的发现指向的id竟然相同，那么就说明有两个指针指向‘xuegod’这个内存。

例子：

del(name) 删除了指针name  
 del(user) 删除了指针user  
 我们再来访问，name和user就会发现如下图：



也就是指向‘xuegod’的内存的所有指针都已经被删除了，引用计数 0

2、python是强类型的动态脚本语言

强类型：不允许不同类型相加。例如：整形+字符串会报类型错误。

动态：不使用显示数据类型声明，且确定一个变量的类型是在第一次给它赋值的时候。

脚本语言：一般是解释性语言，运行代码只需要一个解释器，不需要编辑。

A = 1  
 A =‘xuegod’

## 2.2 python 输出、输入

2.2.1 Print：输出

Python2的输出是print，而python3的输出直接变成一个函数print（）

例如：print（”hello,world !”）

print('''

へ　　　　　／|

　　/＼7　　　 ∠＿/

　 /　│　　 ／　／

　│　Z ＿,＜　／　　 /`ヽ

　│　　　　　ヽ　　 /　　〉

　 Y　　　　　`　 /　　/

　ｲ●　､　●　　⊂⊃〈　　/

　()　 へ　　　　|　＼〈

　　>ｰ ､\_　 ィ　 │ ／／

　 / へ　　 /　ﾉ＜| ＼＼

　 ヽ\_ﾉ　　(\_／　 │／／

　　7　　　　　　　|／

　　＞―r￣￣`ｰ―＿

''')

2.2.2 Input 输入

1、python2版本

在Python2中，获取键盘输入的数据的方法是采用 raw\_input ()和input()两种函数。

看如下示例:

raw\_input()

raw\_input()的小括号中放入的是提示信息，用来在获取数据之前给用户的一个简单提示；

raw\_input()在从键盘获取了数据以后，会存放到等号左边的变量中；

raw\_input()会把用户输入的任何值都作为字符串来对待。

input()

input()函数与raw\_input()类似，但其接受的输入必须是表达式；

input()接受表达式输入，并把表达式的结果赋值给等号左边的变量；

raw\_input()输入的都当成字符串（和Python3的input()功能一样）input()输出的是输入的数据类型。

2、python3版本中

没有raw\_input()函数，只有input()

并且 python3中的input与python2中的raw\_input()功能一样

案例：

name = input('请输入你的姓名:')

#python 程序当中 不print 就不会打印  
print(name)

输出结果：

请输入你的姓名:for

for

## 2.3 python数字类型

### 2.3.1 Python数字类型介绍

整型： int型， 例子：1为整型。

浮点型： float型， 例子：2.1为浮点型。

长整型： long型， 例子：2L为长整形，数字后面加L就是长整形，理论上长整型的界限为 2147483647。

Long的爱恨情仇：

因为python 2.x版本长整型的不严谨，python 3版本取消了长整型。在python 2.x版本对MySQL操作时，导出的整形数可能就是长整型。

### 2.3.2 python数字类型转换

我们可以通过类型函数（type（变量名））查看数字类型

print(type(1))  
 print(type(1.0))

输出：

<class 'int'>

<class 'float'>

也可以通过运算改变数字类型：

print(type(3//2))  
print(type(3/2))

输出：

<class 'int'>

<class 'float'>

### 2.3.3 python数字类型运算符

| 运算符 | 描述 | 实例 |
| --- | --- | --- |
| + | 加 | 两个对象相加 a + b 输出结果 7 |
| - | 减 | 得到负数或是一个数减去另一个数 a - b 输出结果 3 |
| \* | 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 10 |
| / | 除 | x除以y b / a 输出结果 2 |
| // | 取整除 | 取商，5//2得2；返回商的整数部分 9//2 输出结果 4 。 |
| % | 取余 | 返回除法的余数 a % b 输出结果 1 |
| \*\* | 幂 | 返回x的y次幂 a\*\*b 为5的2次方， 输出结果 25 |

>>> a = 5

>>> b = 2

>>> a + b

7

>>> a \* b

10

>>> a / b

2.5

>>> a // b #取整除

2

>>> a % b #取余

1

>>> 2 \*\* 3 #次方（幂）

8

>>> a \*\* 2

25

Python 内健函数讲解：

内建数值型函数

abs(x) 取x的绝对值

round(number, ndigits=0) 对数值进行'四舍五入',ndigits是小数向右取整的位数，负数表示向左取整

pow(x, y, z=None) 相当于 x \*\* y 或 x \*\* y % z

help() 函数查看帮助

help(函数名或对象名)

>>> help(abs)

## 2.4 python字符串类型

### 2.4.1 python字符串类型概述

字符串定义：

字符串是一个有序的，不可修改的，以引号包围的序列。

什么是python字符串：双引号或者单引号中的数据，就是字符串

>>> a = 100 #数字类型

>>> b = 'hello world' #字符串类型

引号：

单引号：‘ ’

双引号： “ ”

三单引号：’’’(多用于代码的注释)

三双引号：”””(多用于代码的注释)

单引号和双引号区别？

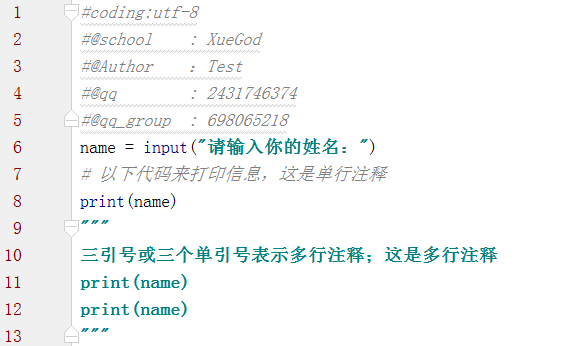
都是字符串的标准格式，只是为了区分英语中的一些语义

print("for's name is for")  
 三引号：字符串内容可换行（多用于注释）。

print('''  
1  
2  
3  
''')  
'''我是多行注释，可以写很多很多行的功能说明

这就是我牛X之处  
哈哈哈'''

注释：



字符串存储的另外一种方式：

>>> a = 'for'

>>> b = ' is'

>>> c = ' cool'

>>> d = a+b+c

>>> d

'for is cool'

字符串当中的特殊字符：

特殊字符就是在字符串当中起到特殊含义的字符。

“\” 转义符 将字符串当中的具有特殊含义的字符的特殊含义取消掉和续行；

“\n”换行；

“\t”水平制表符，tab键。

|  |  |
| --- | --- |
| \(在行尾时) | 续行符 |
| \\ | 反斜杠符号 |
| \' | 单引号 |
| \" | 双引号 |
| \a | 响铃 |
| \b | 退格(Backspace) |
| \000 | 空 |
| \n | 换行 |
| \v | 纵向制表符 |
| \t | 横向制表符，也就是键盘的table键 |
| \r | 回车 |
| \f | 换页 |
| \ | 将字符串当中的具有特殊含义的字符的特殊含义取消掉 |

（\a的时候win会颤抖下，发出不明觉厉的声响）

字符串格式化操作：

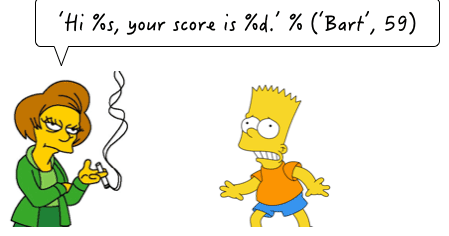
在字符串当中以指定的格式符号进行占位，然后我们将指定的数据传入字符串

%s 字符串占位符

%d 数字占位符

%f 浮点型数字占位符

%.2f 控制浮点型数字占位符



（对于学渣的我来说，知道分数和不知道分数同样纠结），当然只要我足够快乐，父母的巴掌就打不到我。

#打印字符串  
print('My name is %s'%('for'))  
#打印整数  
print('I am %d years old'%(25))  
#打印浮点数  
print('His height is %f m'%(1.70))  
#打印浮点数（保留两位小数）  
print('His height is %.2f'%(1.70))

运行结果如下：

My name is for

I am 25 years old

His height is 1.700000 m

His height is 1.70

### 2.4.2 python字符串的索引(index)

超市储物柜：

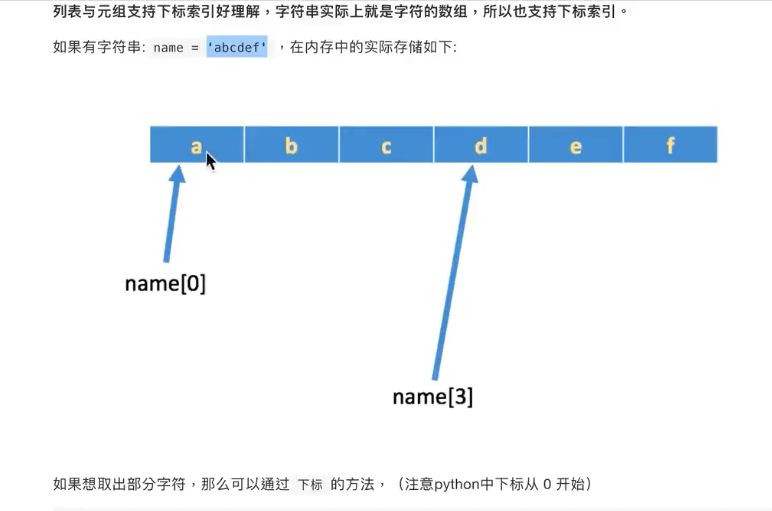


在python当中所有有序的序列都是由索引概念的，它们的区别在于序列是否可以被修改；索引在我们初学的时候我们可以理解为字符串的下标；

字符串里的每一个个体都被称作字符也是该字符串的一个元素，每一个元素都对应一个索引值（下标）；

在这里可以用len() 方法看一个序列的长度

索引的用法，取单个元素时，使用字符串[索引值] 索引值为对应元素的索引号；



比如字符串‘while’，可以按照下图理解其下标概念，索引号从0开始；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| w | h | i | l | e |
| 0 | 1 | 2 | 3 | 4 |

字符串截取：字符串[start:end]，得到对应索引范围的元素，该范围包含起始端，不包含结尾端，默认截取的方向是从左往右的；

name = 'while'  
print(name[1:3])

步长截取：字符串[start:end:step] 按照step步长进行隔取；

切片的语法：[起始:结束:步长]

注意：选取的区间属于左闭右开型，即从"起始"位开始，到"结束"位的前一位结束（不包含结束位本身)。

老师是想说 可以理解为包头不包尾

str\_test = 'hello world'  
print(str\_test[0:7:2])

'hlow'

默认取法：字符串[start:end:step] 这三个参数都有默认值、start；默认值为0；end 默认值未字符串结尾元素；step 默认值为1

若  step > 0, 则表示从左向右进行切片。 此时，start必须小于end才有结果，否则为空。

若  step < 0, 还是表示从左到右只不过反过来切片，此时，start必须大于end才有结果，否则为空。

str\_test = 'hello world'

print(str\_test[0:7])  
print(str\_test[:7])  
print(str\_test[2:])  
print(str\_test[:])  
print(str\_test[::2])

反取：字符串[负数]，从右往左取(从后面取)

print(str\_test[::-1])  
print(str\_test[::-2])  
print(str\_test[1:9:-1])  
print(str\_test[9:1:-2])

### 2.4.3 python字符串的方法

|  |  |  |
| --- | --- | --- |
| 字符串的查找 | count | 计数功能，返回自定字符在字符串当中的个数 |
| find | 查找，返回从左第一个指定字符的索引，找不到返回-1 |
| rfind | 查找，返回从右第一个指定字符的索引，找不到返回-1 |
| index | 查找，返回从左第一个指定字符的索引，找不到报错 |
| rindex | 查找，返回从右第一个指定字符的索引，找不到报错 |
| 字符串的分割 | partition | 把mystr以str分割成三部分,str前，str自身和str后 | |
| rpartition | 类似于 partition()函数,不过是从右边开始. | |
| splitlines | 按照行分隔，返回一个包含各行作为元素的列表,按照换行符分割 | |
| split | 判断字符串的分隔符切片 | |
| 字符串的替换 | replace | 从左到右替换指定的元素，可以指定替换的个数，默认全部替换 | |

str\_test = 'hello world'  
print(str\_test.count('l'))  
print(str\_test.find('world'))  
print(str\_test.rfind('world'))  
print(str\_test.index('o'))  
print(str\_test.rindex('o'))

运行结果如下：

3

6

6

4

7

|  |  |  |
| --- | --- | --- |
| 字符串的分割 | partition | 把mystr以str分割成三部分,str前，str自身和str后 |
| rpartition | 类似于 partition()函数,不过是从右边开始. |
| splitlines | 按照行分隔，返回一个包含各行作为元素的列表,按照换行符分割 |
| split | 判断字符串的分隔符切片 |
| 字符串的替换 | replace | 从左到右替换指定的元素，可以指定替换的个数，默认全部替换 |

str\_test = 'hello world'  
print(str\_test.partition('o'))  
  
print(str\_test.rpartition('o'))  
  
my\_str = 'hello\n world \n python\n'  
print(my\_str)  
print(my\_str.splitlines())  
print(str\_test.replace('h','w'))

|  |  |  |
| --- | --- | --- |
| 字符串的修饰 | center | 让字符串在指定的长度居中，如果不能居中左短右长，可以指定填充内容，默认以空格填充 |
| ljust | 让字符串在指定的长度左齐，可以指定填充内容，默认以空格填充 |
| rjust | 让字符串在指定的长度右齐，可以指定填充内容，默认以空格填充 |
| zfill | 将字符串填充到指定的长度，不足地方用0从左开始补充 |
| format | 按照顺序，将后面的参数传递给前面的大括号 |
| strip | 默认去除两边的空格，去除内容可以指定 |
| rstrip | 默认去除右边的空格，去除内容可以指定 |
| lstrip | 默认去除左边的空格，去除内容可以指定 |

str\_test = 'for'  
print(str\_test.center(10))

#center这个里面可以加入两个参数  
print(str\_test.center(10,'x'))

运行结果如下：

for

xxxforxxxx

ljust这个里面可以加入两个参数

print(str\_test.ljust(10))  
print(str\_test.ljust(10,'x'))  
print(str\_test.rjust(10,'x'))

运行结果如下：

for

forxxxxxxx

xxxxxxxfor

print(str\_test.zfill(10))

运行结果如下：

0000000for

注意：如果传入的大小小于字符串的长度，字符串不变。

str\_test = ' for '

print(str\_test.strip())

'for'

print(str\_test.rstrip())

' for'

print(str\_test.lstrip())

'for '

python = "{0} is {1}"  
print(python.format('for','cool'))

运行结果如下：

'for is cool'

字符串的变形

|  |  |  |
| --- | --- | --- |
| 字符串的变形 | upper | 将字符串当中所有的字母转换为大写 |
| lower | 将字符串当中所有的字母转换为小写 |
| swapcase | 将字符串当中所有的字母大小写互换 |
| title | 将字串符当中的单词首字母大写，单词以非字母划分 |
| capitalize | 只有字符串的首字母大写 |
| expandtabs | 把字符串中的 tab 符号('\t')转为空格，tab 符号('\t')默认的空格数是 8，可以试下8,12 |

print('hello'.upper())  
*#将字符串当中所有的字母转换为小写*print('HELLO'.lower())  
*#将字符串当中所有的字母大小写互换*print('Hello'.swapcase())  
*#将字串符当中的单词首字母大写，单词以非字母划分*print('hello,world'.title())  
*#只有字符串的首字母大写*print('hello world'.capitalize())  
*#把字符串中的 tab 符号('\t')转为空格，tab 符号('\t')默认的空格数是 8*print('for \t is \t cool'.expandtabs(10))  
print('for \t is \t cool'.expandtabs())  
print('for \t is \t cool'.expandtabs(4))

运行结果如下：

HELLO

hello

hELLO

Hello,World

Hello world

for is cool

for is cool

for is cool

字符串的判断

|  |  |  |
| --- | --- | --- |
| 字符串的判断 | isalnum | 判断字符串是否完全由字母或数字组成 |
| isalpha | 判断字符串是否完全由字母组成 |
| isdigit | 判断字符串是否完全由数字组成 |
| isupper | 判断字符串当中的字母是否完全是大写 |
| islower | 判断字符串当中的字母是否完全是小写 |
| istitle | 判断字符串是否满足title格式 |
| isspace | 判断字符串是否完全由空格组成 |
| startswith | 判断字符串的开头字符，也可以截取判断 |
| endswith | 判断字符串的结尾字符，也可以截取判断 |

*#判断字符串是否完全由字母或数字组成*print('12345a'.isalnum())  
*#判断字符串是否完全由字母组成*print('abcdef'.isalpha())  
*#判断字符串是否完全由数字组成*print('12345'.isdigit())  
*#判断字符串当中的字母是否完全是大写*print('HELLO'.isupper())  
*#判断字符串当中的字母是否完全是小写*print('hello'.islower())  
*#判断字符串是否满足title格式*print('Hello'.istitle())  
*#判断字符串是否完全由空格组成*print(' '.isspace())  
*#判断字符串的开头字符，也可以截取判断*print('for is long'.startswith('f'))  
print('for is long'[3:].startswith('f'))  
*#判断字符串的结尾字符，也可以截取判断*print('for is long'.endswith('long'))  
  
print('for is long'.split())

判断字符串的分隔符切片

>>> 'for is cool'.split(' ')

['for', 'is', 'cool']

### 2.4.4 python字符串的编码

encode是编码  
decode是解码

| 编码方式 | ASCII | Unicode | UTF-8 |
| --- | --- | --- | --- |
| 二进制表示字母‘A’ | 01000001 | 00000000 01000001 | 01000001 |
| 二进制表示中文‘中’ |  | 01001110 00101101 | 11100100 10111000 10101101 |
| 十六进制形式 |  | \u4e2d | b\xe4\xb8\xad |
| 编码范围 | 只有英文 | 所有语言 | 所有语言 |
| 一个字符占字节数 | 1 | 2（少数偏僻字4） | 英文1，中文3（少数偏僻字4-6） |
| 特点 |  | 统一，英文就在ASCII前面补一个字节 | 节省，英文跟ASCII一样只有一个字节 |
| 用处 |  | 内存中，服务器中，比较统一 | 保存在硬盘时，传输时，需要节省 |

编码方式对比：

python字符串是一种数据类型，但是字符串比较特殊，它有一个编码的问题。

因为计算机只能处理数字，如果要处理文本，就必须先把文本转换为数字才能处理。最早的计算机在设计时采用8个比特（bit）作为一个字节（byte），所以，一个字节能表示的最大的整数就是255（二进制11111111=十进制255），如果要表示更大的整数，就必须用更多的字节。比如两个字节可以表示的最大整数是65535，4个字节可以表示的最大整数是4294967295。

由于计算机是英语国家发明的，最早只有127个字母被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为ASCII编码，比如大写字母A的编码是65，小写字母a的编码是97。

但是要处理中文显然一个字节是不够的，至少需要两个字节，而且还不能和ASCII编码冲突，所以，中国制定了GB2312编码，用来把中文编进去。

全世界有上百种语言，日本把日文编到Shift\_JIS里，韩国把韩文编到Euc-kr里，各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。

因此，俗称“万国码”的Unicode应运而生。Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

Unicode标准也在不断发展，但最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。现代操作系统和大多数编程语言都直接支持Unicode。

现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

字母A用ASCII编码是十进制的65，二进制的01000001；

字符0用ASCII编码是十进制的48，二进制的00110000，注意字符'0'和整数0是不同的；

汉字中已经超出了ASCII编码的范围，用Unicode编码是十进制的20013，二进制的01001110 00101101。

如果把ASCII编码的A用Unicode编码，只需要在前面补0就可以，因此，A的Unicode编码是00000000 01000001。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用 UTF-8编码就能节省空间：

UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作。搞清楚了ASCII、Unicode和UTF-8的关系，我们就可以总结一下现在计算机系统通用的字符编码工作方式：

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件。浏览网页的时候，服务器会把动态生成的Unicode内容转换为UTF-8再传输到浏览器。

由于python的诞生比Unicode标准发布的时间还要早，所以最早的Python只支持ASCII编码，普通的字符串’ABC’在Python内部都是ASCII编码的。Python提供了ord()和chr()函数，可以把字母和对应的数字相互转换，len()函数可以查字符串的长度：

u = '学神'  
*#python 编码是‘gbk*str1 = u.encode('gbk')  
print(str1)  
*#python3 编码 ‘utf-8’*str2 = u.encode('utf-8')  
print(str2)  
*#解码*u1 = str1.decode('gbk')  
print(u1)  
*#解码*u2 = str2.decode('utf-8')  
print(u2)

运行结果如下：

b'\xd1\xa7\xc9\xf1'

b'\xe5\xad\xa6\xe7\xa5\x9e'

学神

学神

 在python中解决编码问题

由于Python源代码也是一个文本文件，所以，当你的源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为UTF-8编码。当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行（python3默认为utf-8 所以没必要加文档头）：

1、# -\*- coding: utf-8 -\*-

2、#coding=utf-8

总结：

2.1 python 变量使用

2.2 python 输出、输入

2.3 python 数字类型

2.4 python 字符串类型

练习作业：

BMI指数(Body Mass Index) 以称身体质量指数

BMI值计算公式: BMI = 体重(公斤) / 身高的平方(米)

例如:

一个人69公斤，身高是173公分

BMI = 69 / 1.73\*\*2 = 23.05

标准表:

BMI < 18.5 体重过轻

18.5 <= BMI < 24 体重正常

BMI > 24 体重过重

要求: 输入身高的体重，打印出BMI的值并打印体重状况