# 认识python

1. python解释器是用C语言实现的，并能够调用C语言的库文件
2. 解释器：
   1. 计算机不能直接理解任何除机器语言以外的语言。所以出现了把其他语言翻译成机器语言的工具，叫做编译器
   2. 编译器翻译的方式有两种：编译和解释。两者的区别在于翻译时间点不同。当编译器以解释的方式运行时，也可以把它叫做解释器。
   3. 编译型：如C或C++。源代码-》编译器-》最终可执行文件-》操作系统-》CPU。效率要更高。
   4. 解释型：如Python和JAVA。源代码-》解释器（逐行解释每一句源代码）-》操作系统-》CPU。具有跨平台的优势，只需要在不同平台装不同解释器就可以运行同一份源代码。

# Python起步

**程序输出**

1. 可以使用print函数在显示器直接输出字符串，亦可以输出变量的值
   1. str = ‘hello world’

print(str)

* 1. print(‘hello world’)
  2. 引号表示这是一个字符串，在python中既可以用单引号也可以用双引号，但需要成对出现。

2、下划线（\_）在解释器中有特别的含义，表示最后一个表达式的值。

3、Python3的print函数，如果要达到和C语言printf一样的效果，可以通过format()

a ) print(“{0} is {1}”.format(“a”,”b”)

i.其中a对应0位置，b对应1位置。即下标通过{}来定

**程序输入**

1. 从用户得到数据输入最容易的方法就是使用input()内建函数。它读取标准输入，并将读取到的数据赋给指定的变量。
2. 例如：
   1. a = input(“”)
   2. input()函数中的参数所表示的字符串一般就是输入的指示文字
3. 通过input()得到的数据是字符串，可以通过int()转化为整型
   1. a = input()

b = int(a) \* 2

1. 在学习python的过程中，如果需要得到一个生疏函数的帮助，只需要对它使用内建函数help()就可以得到相关的帮助信息：
   1. help(print)

**注释**

1. 和大部分脚本和Unix-shell语言一样，python也是用#符号表示注释
2. 从#开始，知道一行结束的内容都是注释。也就是行级注释
3. 也有一种叫文档字符串的特别注释。
   1. 在模块、类、函数的其实添加一个字符串。注意必须是**起始位置**
   2. 起到了在线文档的功能
   3. 类似于java的/\*\*

**运算符**

1. +-\*/%，加减乘除余都是标准运算符。
2. Python有两种除服运算符：
   1. 单斜杠（/）：传统除法，如果两个操作数都是整数，结果取比商小的最大整数
   2. 双斜杠（//）：浮点除法，真正的除法，对结果四舍五入
3. Python还有乘方运算符：
   1. 用双星号（\*\*）：x\*\*y，即x的y次方
   2. 类似于其他语言的pow(x,y)
4. Python也有标准的比较运算符
   1. <、<=、>、>=、==、!=、<>
   2. 表达式返回的结果是布尔值
   3. !=是C风格的不等，<>是ABC/Pascal风格的不等，<>慢慢地被淘汰了。
5. Python也提供逻辑运算符：
   1. 与and、或or、非not
   2. 和java的&&、||、!稍有不同
   3. 像3<4<5这样的表达式在其他语言是不合法的，但在python等价于3<4 and 4<5，是合法的，返回布尔值。
6. 同其他语言一样，使用括号可以改变运算符运算顺序，同时也增强代码的可读性

**变量与赋值**

1. python变量名规则和其他大多语言一样，主要是收C语言影响
2. 变量名开头只允许字母和下划线（\_）。其他字符可以是数字、字母、下划线。
3. Python的变量名是大小写敏感的
4. Python是动态型语言，不需要预先声明变量的类型。
5. 变量的类型和值在赋值的那一刻被初始化
6. 变量赋值通过等号（=）来执行
7. Python也支持增量赋值
   1. n = n \* 10 -》 n \*= 10
   2. n = n + 1 -》 n += 1
8. 但是python不支持自增（++）和自减（--）

**数字**

1. python支持五种基本数字类型：
   1. int：有符号整数
   2. long：长整数
   3. bool：布尔值
   4. float：浮点数
   5. complex：复数
2. Python的长整数所能表达的范围远远超过C语言的长整数
   1. 仅仅受限于用户计算机的虚拟内存总数
   2. 类似于JAVA的BigInteger
   3. 整型和长整型慢慢无缝结合，后缀的L基本变得可有可无
3. 布尔值是特殊的整数
   1. 和C语言类似，和JAVA不同
   2. True可以当成1，False可以当成0
4. Python2.4增加了十进制浮点数类型decimal

**字符串**

1. python中字符串被定义为引号间的字符集合
2. Python支持使用成对的单引号或双引号或三引号（三个连续的单引号或双引号），来包含特殊字符。
3. 使用索引运算符（[]）或者切片运算符（[:]）可以得到子字符串
4. 字符串索引规则：
   1. 第一个字符的索引是0，第二个之后递增
   2. 最后一个字符的索引是-1，之后递减
5. 加号（+）可以用于字符串连接运算：
   1. “hello” + “world”连接为”helloworld”
6. 星号（\*）可以用于字符串重复：
   1. “python” \* 2可以变为”pythonpython”

**列表和元组**

1. 可以将列表和元组当成普通的“数组”，能保存**任何数量**、**任何类型**的python对象。
2. 和数组一样，从0开始的索引。
3. 列表和元组中存储的对象可以**类型各不相同**
4. 列表和元组的区别：
   1. 列表元素用中括号（[]）包裹，元组元素用小括号（()）包括
   2. 列表元素个数和值可以改变，元组元素不可以更改（内容可以改）
   3. 元组可以看成只读的列表
5. 列表和元组都可以通过切片运算（[]和[:}）得到自己，和字符串的使用方法相同。

**代码块及缩紧对齐**

1. python表达代码逻辑的方式和其他语言不同：
   1. c和java等通过大括号
   2. Python通过缩进对其的方式
2. 这样做的好处是没有了额外的字符，程序可读性更高。
   1. 至少不会出现}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}的情况
3. 代码块当然和其他语言一样，可以只有一个语句构成。
4. 这种缩进对齐的方式是简洁和可读性好的体现

**if语句**

1. 标准if语句语法：

if 表达式:

if代码块

1. 表达式可以是数字或者布尔值，如果为非0或True则会执行if代码块，否则执行下一条语句。
2. Python的条件表达式不需要括号（()），但末尾需要冒号（:）
3. Python的else语句语法：

if 表达式:

if代码块

else:

else代码块

1. python的elif（即else-if）语句语法：

if 表达式1:

if代码块

elif 表达式2:

elif代码块

else:

else代码块

**while循环**

1. while条件循环语句语法：

while 表达式:

while代码块

1. 若表达式为非0或True，就会一直执行while代码块，否则执行下一条语句

**for循环**

1. python的for循环和传统的计数器循环不太一样，更类似与for-each迭代，语法为：

for 自定义变量 in 列表:

for代码块

1. 我们无法改变for循环迭代一个序列的行为，当我们可以改变一个序列
2. 由于我们要迭代的范围经常会出现变化，因此可以使用一个内建函数：
   1. range(x)：生成并返回一个从0开始的，有x个数的列表

**列表解析**

1. 可以将for循环语句放入一个列表中：
   1. a = [x for x in range(4)] 则a = [0,1,2,3]
   2. a = [x+1 for x in range(4)] 则a = [1,2,3,4]

**文件**

1. 打开文件：
   1. handle = open(文件名,打开模式)
   2. 文件名如果和程序一个目录下则无需路径，如果不在一个目录下则用绝对路径
   3. 打开模式r表示读取，w表示写入，a表示添加，+表示读写，b表示二进制访问。如果没提供该参数，默认值为r
   4. 该函数返回一个文件对象的句柄，之后对文件的操作都可以通过句柄完成
   5. file()内建函数同样是打开文件：
      1. 等同于open()
      2. 但它是一个工厂函数，用于生成文件对象

**错误和异常**

1. 编译时会检查错误的语法
2. Python也和java一样允许在程序运行时检测错误，语法：

try:

要捕捉的代码块

except 异常:

异常处理代码块

1. 可以通过raise故意引发一个异常，类似于java的throw

**函数**

1. 和其他语言一样，python中的函数使用小括号（()）调用
2. 函数在调用之前必须先定义
3. 如果函数中没有return语句，则会自动返回none对象
4. Python是通过引用调用函数的：
   1. 函数内对参数的改变会影响到原始对象。不可变对象就和按值调用类似，不会改变
5. 函数定义语法：

def 函数名(参数列表):

“文档注释”

函数体

1. 参数列表可以为空
2. 而其他语言一样，函数调用时，小括号（()）不能省略，即使没参数。
3. 和java不同，和C相同，函数的参数可以是默认值，在函数调用时没提供该参数有效

**类**

1. python并不强求以面向对象的方式编程
   1. 类似于C++，区别于JAVA
2. 定义类：

Class 类名(父类列表):

“文档注释”

静态成员声明

方法声明

1. 父类列表可以使用object作为几类
2. 文档注释可有可无
3. \_\_init\_\_()方法比较特殊：
   1. 所有名字开始或结束都有两个下划线的方法都是特殊方法
   2. 类似于构造方法，当一个类实例被创建时，\_\_init\_\_()就会自动执行
   3. 和其他语言稍有不同，它不创建实例，而仅仅是对象创建后执行的第一个方法
   4. 它的目的就是执行一些必要的初始化工作。
4. self类似于其他语言的this，是类实例自身的引用
5. 创建一个类实例就像调用一个函数，加上小括号（()）创建

**模块**

1. 模块是一种组织形式，可以包含可执行代码、函数、类以及它们的组合
   1. 可以参考java的包,C++的头文件
2. 一个python源文件就是一个模块，模块名就是不带.py后缀的文件名
3. 导入模块语法：

import 模块名

1. 导入完成后，模块的属性和变量可以通过点属性标识法访问
   1. module.function()
   2. module.variable

**实用的函数**

1. dir([对象])：
   1. 可以没有参数
   2. 显示对象的属性，如果没提供参数，显示全局变量的名字
2. help([对象])：
   1. 可以没有参数
   2. 显示对象的文档字符串。如果没提供参数，进入交互式帮助
3. int(对象)：
   1. 将一个对象转换为整数
4. len(对象)：
   1. 返回对象的长度
5. open(文件名,模式)：
   1. 以指定模式打开一个文件
6. range(start,end,step):
   1. 返回一个**整数**列表
   2. start默认为0，step默认为1，结束值为end-1
7. intput(str)：
   1. 等待用户输入一个字符串，str作为特使信息
8. str(obj)：
   1. 将一个对象转换为字符串
9. type(obj)：
   1. 返回对象的类型

# Python基础

**语句和语法**

1. python语句中的基本规则和特殊字符：
   1. 井号（#）表示之后的字符为python注释
   2. 换行（\n）是标准的行分隔符
   3. 反斜线（\）继续上一行
   4. 分号（;）将两个语句连接在一行中
   5. 冒号（:）将代码块的头和体分开
   6. 代码块用缩进的方式体现
   7. 不同的缩进深度分割不同的代码块
   8. Python文件以模块的形式组织
2. 注释（#）：
   1. 注释可以在一行的任何地方开始
   2. 解释器会忽略掉改行#之后的所有内容
3. 继续（\）：
   1. 一行语句过长时可以用反斜杠（\）分割成几行
   2. 比如：

if (a == 1) and \

(b == 1):

print(“a”)

* 1. 有两种例外：
     1. 含有小括号、中括号、大括号时可以多行书写
     2. 三引号包括下的字符串也可以跨行书写

1. 多个语句构成代码组（:）：
   1. 缩进相同的一组语句构成一个代码块，也可以称为代码组
   2. 如if、while、def、class这样的
2. 代码组由不同的缩进分割：
   1. 代码的层次关系通过同样深度的空格或制表符体现
   2. 同一代码组的代码必须严格左对齐，否则会被当成另一个组，或者导致语法错误
   3. 缩进一般推荐用四个空格宽度，而避免使用制表符。因为不同文本编辑器的制表符代表的空白宽度不一致
   4. 没有缩进的代码块是最高层次，被称作脚本的main部分
3. 同一行书写多个语句（;）：
   1. 分号（;）允许将多个语句写在同一行上，语句之间用分号隔开
   2. 但这些语句不能在这行开始一个新的代码块
   3. 这样做会降低代码可读性，因此不推荐
4. 模块：
   1. 每一个python脚本文件都可以被当成一个模块
   2. 模块是以磁盘文件形式存在的。
   3. 模块里的代码可以是一段直接执行的脚本，亦可以是库函数，从而被其他模块导入
   4. 当一个模块变得过大，或驱动太多功能时，应考虑拆些代码出来另外组建一个模块。

**变量赋值**

1. python语言中，等号（=）是主要的赋值运算符
   1. 对象是通过引用传递的，在赋值时，不管这个对象是新创建还是已经存在，都是将该对象的引用赋值给变量。
   2. Python的赋值语句不会返回值
2. Python的增量赋值：
   1. x += 1
   2. x \*= 1
   3. =、-、\*、/、%、\*\*、<<、>>、&、^、/都可以
   4. Python不支持++、--这样的前置\后置自增自减
3. python允许多重复值
   1. a = b = c = 2
4. 多元赋值：
   1. 将多个变量同时赋值的方法
   2. 等号两边的对象都是元组
   3. 比如：a, b, c = 1, 2, ‘hello’
   4. 通常元组需要用小括号括起来，因此推荐用：
      1. (a, b, c) = (1, 2, ‘hello’)
   5. Python通过多元赋值无需中间变量就可以交换两个变量的值：
      1. (x, y) = (y, x)

## 标识符

1. 合法的python标识符
   1. 第一个字符必须是字母或下划线（\_）
   2. 剩下的字符可以是字母、数字、下划线
      1. 除了下划线，其他字符都不允许使用
      2. 不能以数字开头
   3. 大小写敏感
   4. 不能用关键字做标识符
2. 关键字：
   1. Python的关键字列表和iskeyword()函数都被放入keyword模块以便查阅
   2. Python的关键字：
      1. and、as、break、class、continue、def、del、elif、else、excep、exec、finally、for、from、global、if、import、in、is、lambda、not、or、pass、print、raise、return、try、while、with、yield、（None）
3. 内建：
   1. 除了关键字以外，python还有可以在任何一级代码使用的内建名字
   2. 这些名字由编译器设置或使用，在程序开始时由解释器自动导入，是\_\_builtins\_\_模块成员
   3. 虽然它们不是关键字，但应该当做系统保留字，最好不用做标识符
   4. Python不支持重载标识符，任何时刻都只有一个名字绑定
   5. 可以把它们看做使用在任何一级python代码的全局变量
4. 专用下划线标识符：
   1. Python用下划线作为变量前缀或后缀指定特殊变量
   2. \_xxx：不需要’from module import \*’导入
   3. \_\_xxx\_\_：系统定义名字
   4. \_\_xxx：类中的私有变量名
   5. 避免用下划线作为变量名的开始，因为它们对解释器有特殊意义。

## 基本风格指南

1. 注释：
   1. 注释对于自己和后来人都是非常重要的
   2. 特别是对那些很久没有被动过的代码而言
   3. 既不能缺少，也不能过多。尽可能简洁明了，放在最合适的地方
   4. 要确保注释的准确性，为每个人节省时间和精力
2. 文档：
   1. Python可以通过\_\_doc\_\_特别变量，动态获得文档字符串
   2. 通过xx.\_\_doc\_\_访问模块、类或函数在第一行声明的文档注释
3. 缩进：
   1. 通常空4格式最流行的。这样代码更容易读
   2. 一般不用制表符tab，因为文本编辑器对它的设置大不相同
4. 选择标识符名称：
   1. 尽量短且意义丰富
   2. 模块的命名同样要尽量短且意义丰富
5. 模块结构和布局：
   1. 一种合理的模块布局结构：
      1. 起始行：#/usr/bin/env python
         1. 通常只在类UNIX环境下才使用，这样可以直接输入脚本名字执行脚本，无需使用解释器
      2. 模块文档：”this is test module”
         1. 简要介绍模块哦功能和重要全局变量的含义
         2. 通过module.\_\_doc\_\_访问这些内容
      3. 模块导入：import sys
         1. 导入当前模块需要的所有模块
         2. 每个模块仅导入一次（在当前模块被加载时）
         3. 函数内部的模块导入代码只有在函数正在被执行时才执行
      4. 变量定义：a = 1
         1. 这里定义的变量为全局变量，本模块中所有函数都可以直接使用
         2. 除非必须，少用全局变量，因为不易维护，且占用内存
      5. 类定义:class b(object):
         1. 所有类在这个位置被定义，当模块被导入时类也就被定义
         2. 类的文档变量是class.\_\_doc\_\_
      6. 函数定义:def c():
         1. 此处定义的函数可以通过module.function()被其他模块访问
         2. 当模块被导入时def语句会被执行，即函数被定义号
         3. 函数的文档变量是function.\_\_doc\_\_
      7. 主程序if \_\_name\_\_ = ‘\_\_main\_\_’:
         1. 无论该模块是被别的模块导入，还是自己作为脚本执行，这部分代码都会被执行。
         2. 通常这里不需要太多功能性代码，而是根据执行的模式调用不同函数
         3. 这是放入测试代码的好地方。
   2. 绝大部分模块创建的目的都是为了被别人调用而不是独立执行的脚本，只有包含主程序的一个模块会被执行
   3. Python文件中没有缩进的代码行在模块被导入时就会被执行，不管是不是真的需要执行
      1. 因此，比较安全的方式就是除了真正需要执行的代码外，其他功能代码都放在函数或类中。
   4. \_\_name\_\_指示模块应如何被加载：
      1. 主程序代码无论模块是被导入还是被直接执行都会运行。
      2. \_\_name\_\_系统变量能在运行时检测该模块是被执行还是被导入
         1. 如果是被导入，\_\_name\_\_的值为模块名字
         2. 如果是被直接执行，\_\_name\_\_的值为’\_\_main\_\_’
6. 在主程序中书写测试代码：
   1. 测试代码仅当改文件被直接执行时运行。而被导入时不会执行
   2. 因此可通过\_\_name\_\_变量判断是被导入还是被执行
   3. 这些代码可以先封装入自定义test()函数中，然后在主程序执行
   4. 这些测试代码应该随着测试条件及测试结果的变更及时修改。
7. 使用局部变量替换模块变量:
   1. 对引用一个模块的一个变量，需要解释器做两次查询：
      1. 查找模块确认它是一个模块
      2. 在这个模块查找变量
   2. 给模块变量取一个别名，就会快很多，而且也可以有更简短的名字了
8. 通常把能提到循环外面的重复操作都尽量提到循环外，避免每次循环重复计算。

## 内存管理

1. 变量无需实现声明：
   1. 变量无需指定类型
   2. 程序员不用关心内存管理
   3. 变量名会被回收
   4. Del语句能够直接释放资源
2. 变量定义：
   1. 和其他大部分语言不同，python无需显式变量声明语句
   2. 变量在第一次被赋值时自动声明
   3. 和其他大多数语言一样，变量至于被创建和赋值后才能被使用
3. 动态类型：
   1. Python中无需类型声明
   2. 对象的类型和内存的占用都是运行时确定的
   3. 在创建即赋值时，解释器根据语法和右侧操作数决定新对象的类型
   4. 在对象创建后，一个该对象的引用就会被赋值给左侧变量
4. 内存分配：
   1. 为变量分配内存时，是在借用系统资源
   2. 用完后，应该释放借用的系统资源。
   3. Python解释器承担了内存管理的复杂任务，和java虚拟机一样
5. 引用计数：
   1. 引用计数是为了保持追踪内存中的对象，记录了所有使用中的对象各有多少引用。
   2. 当对象被创建时，就创建一个引用计数
   3. 当对象不再需要时，对象的引用计数变为0，就被垃圾回收
   4. 增加引用计数：
      1. 当对象被创建并将引用赋值给变量时，该对象的引用计数就被置1
      2. 当同一个对象的引用传递到其它对象时，引用计数自增1
      3. 有几种增加的情况：
         1. 被创建时
         2. 引用变量赋值给其它变量时
         3. 作为参数传递给函数时
         4. 称为容器中的一个元素时
   5. 减少引用计数：
      1. 当对象的引用被销毁时，引用计数会减小
      2. 有几种减少的情况
         1. 引用该对象的函数结束时
         2. 对象的别名通过del被显式销毁时
         3. 其他对象赋值给对象的一个别名时
         4. 对象被从一个容器移除时
         5. 对象所属的容器本身被销毁时
   6. del语句
      1. del语句会删除对象的一个引用：
         1. del obj1,[obj2,...]
      2. del y会产生两种结果：
         1. 从现在的名字空间删除y
         2. y所引用的对象引用计数减1
6. 垃圾收集：
   1. 不再被使用的内存通过垃圾收集机制释放
   2. 解释器跟踪对象的引用计数，而释放内存的工作有垃圾收集器负责
   3. 垃圾收集器用来寻找引用计数为0的对象，也负责那些引用计数大于0但也应该被销毁的对象（循环引用导致的）
   4. 因为循环引用的问题，只靠引用计数是不够的。Python的垃圾收集器实际就是一个引用计数器和一个循环垃圾收集器。
   5. 循环垃圾收集器会寻找那些虽然引用计数不为0，但是不可达的对象

## 相关模块和开发工具

1. 调式模块pbd
   1. 允许设置（条件）断点，代码逐行执行，检查堆栈
   2. 还支持时候调试
2. 日志模块logging:
   1. 定义了一些函数和类帮助程序实现灵活的日志系统
   2. 共有五种日志级别：紧急、错误、警告、信息、调试
3. 性能测试模块：
   1. Profile：最早的，最慢的，用python写成，用来测试函数的执行时间以及每次脚本执行的总时间。
   2. Hotshot：修复了profile模块的一些错误，用C语言写的，性能更高，重点解决性能测试过载的问题，却需要更多的时间生成结果
   3. cProfile：最新的，用C语言写的，缺点是：
      1. 需要花较长时间从日志文件载入分析结果
      2. 不支持子函数状态细节
      3. 某些结果不准

# Python对象

## Python对象

1. python使用对象模型来存储数据
   1. 构造任何类型的值都是一个对象，包括基本数据类型
   2. Python完全能够不适用任何类和实例
   3. Python对象拥有的三个特性：
      1. 身份：每个对象都有一个唯一的身份标识自己，任何对象的身份可以用内建函数id()来获得。这个值可以被认为对象的内存地址
      2. 类型：对象的类型决定了该对象可以保存什么类型的值，可以进行什么操作，以及遵循什么规则。
         1. 可以用内建函数type()查看python对象的类型。而类型也是对象，因此type()函数返回的是对象而不是字符串
      3. 值：对象表示的数据项。
   4. 对象在创建时就被赋值，身份和类型都是只读的。
2. 对象属性：
   1. 某些python对象有属性、值和相关联的可执行代码，比如方法
   2. Python用点（.）标记法访问属性，这点和java一样

## 标准类型

1. 数字、整型、布尔型、长整型、浮点型、复数型、字符串、列表、元组、字典
2. 这些python内建的标准类型也称为基本数据类型

## 其他内建类型

1. 类型（type）、Null对象（None）、文件、集合/固定集合、函数/方法、模块、类
2. 类型对象和type类型对象：
   1. 在python中把数据类型本身也看做对象
   2. 所有类型对象的类型都是type，它是所有python类型的根和所有Python标准类的默认元类。
3. None，python的null对象
   1. Python有一个特殊的类型，被称为Null对象，它只有一个值，就是None
   2. 它不支持任何运算，也没有任何内建方法
   3. None类型类似于C语言的void，None类型的值类似于C语言的NULL
   4. None的布尔值总是False
   5. 布尔值：
      1. 所有标准对象均可用于布尔测试
      2. 同类型的对象之间可以比较大小
      3. 以下对象的布尔值是False，其他所有对象布尔值都是True
         1. None
         2. False
         3. 所有值为0的数
            1. 0（整型）
            2. 0.0（浮点型）
            3. 0L（长整型）
            4. 0.0+0.0j（复数）
         4. “”（空字符串）
         5. []（空列表）
         6. ()（空元组）
         7. {}（空字典）

## 内部类型

1. 一般程序员不会直接和这些对象打交道
2. 代码对象：
   1. 代码对象是编译过得Python源代码片段，是可执行对象
   2. 通过调用内建函数compile()可以得到代码对象。
   3. 代码对象可以被exec命令或eval()内建函数执行
   4. 代码对象本身不包含任何执行环境信息，它是用户自定义函数的核心，在被执行时动态获得上下文。
   5. 一个函数有的属性：
      1. 代码对象
      2. 函数名
      3. 文档字符串
      4. 默认参数
      5. 全局命名空间等
3. 帧对象：
   1. 帧对象表示python的执行栈帧。
   2. 帧对象包含python解释器在运行时需要知道的所有信息
   3. 它的属性包括上一帧的链接，正在被执行的代码对象，本地以及全局名字空间字典，当前指令等
   4. 每次函数调用产生一个新的帧，每个帧对象都会创建一个C栈帧
4. 跟踪记录对象
   1. 当代码出错时，python就会引发一个异常
   2. 如果异常未被捕获或处理，解释器就会退出脚本执行，显示诊断信息
   3. 当异常发生时，一个包含针对异常的跟踪记录对象被创建。如果一个异常有自己的处理程序，处理程序就可以访问这个跟踪记录对象
5. 切片对象：
   1. 当使用python扩展的切片语法时，就会创建切片对象
   2. 切片对象可以由内建函数slice()生成
   3. 多维切片语法：sequence[start1,end1,start2,end2]
   4. 步进切片语法：sequence[start,end,step]
   5. 将列表转置可以list[::-1]
6. 省略对象：
   1. 省略对象用于扩展切片语法中，起记号作用。
   2. sequence[...,start1,end1]
   3. 它有一个唯一的名字Ellipsis，布尔值始终为True
7. Xrange对象：
   1. 调用内建函数xrange()会生成一个Xrange对象
   2. xrange()类似于range()，用于需要节省内存使用或range()无法完成超大数据集场合

## 标准类型运算符

1. 对象值的比较：
   1. 比较运算符可以判断同类型对象==,!=,>,<,>=,<=等
   2. 所有内建类型均支持比较运算
   3. 比较运算返回布尔值True或False
   4. 实际进行的比较运算因类型而异：
      1. 数字类型根据大小和正负比较
      2. 字符串按照字符串序列值比较
   5. 多个比较操作可以在同一行进行，求值顺序为从左到右
      1. 4<3<5!=2<7
   6. 比较的目标是对象的数值而不是对象本事
   7. 未来很可能不再支持<>，建议一直使用!=
2. 对象身份比较：
   1. Python也支持对象本身的比较。
   2. 对象可以通过引用赋值到另一个对象，当任何一个引用使内容发生改变，其他引用因为指向同一个对象也当然改变
   3. a= b = 1，除了多重赋值外，其实也是将a和b同时指向1这个对象
   4. Python提供了is和not is运算符测试两个变量是否指向同一个对象
      1. a is b等价于id(a) == id(b)
   5. 整数对象和字符串对象是不可变对象，python会很高效地缓存它们。
      1. Python缓存的整数范围是（-1,100），不过不确定现在是不是
   6. 一个字符串如果没有被引用，那么也不会被缓存
3. 布尔类型
   1. 布尔逻辑运算符and，or，not都是python关键字
   2. 优先级顺序是：not》and=or，比所有比较运算符低一级

## 标准类型内建函数

1. cmp(obj1,obj2)：比较obj1和obj2，用于python2，python3不可用
   1. 不管是标准类型对象还是用户自定对象都可以用此方法比较
   2. 如果是用户自定义对象，cmp()会调用该类的特殊方法\_\_cmp\_\_()
   3. 如果obj1<obj2，返回负数
   4. 如果obj1>obj2，返回正数
   5. 如果obj1==obj2，返回0
2. Python3可以导入operator模块用一些函数替换cmp()：
   1. 返回都是布尔值
   2. operator.gt(a,b)：大于
   3. operator.lt(a,b)：小于
   4. operator.le(a,b)：小于等于
   5. operator.ge(a,b)：大于等于
   6. operator.ne(a,b)：不等于
   7. operator.eq(a,b)：等于
3. repr(obj)，或`obj`：返回一个对象的字符串表示
   1. 可以通过eval()重新得到该对象
   2. 并不是所有返回的字符串都是用eval()得到原来的对象
   3. 目前`obj`的方式不再鼓励了
4. str(obj)：返回对象可读性好的字符串表示
   1. 返回的结果无法用于eval()求值
   2. 适合用于print()输出
5. type(obj)：得到对象的类型对象，并返回类型对象
6. type()和isinstance()：
   1. Python不支持方法或函数重载
   2. isinstance()接受一个或多个对象作为参数，如果参数是同一类，则返回True，否则返回false

## 类型工厂函数

1. 所有内建类型也是类
2. int()、type()、list()都成了工厂函数，也就是说它们看上去像函数，但实质上都是类
   1. 调用它们的时候，实际是生成了该类型的一个实例
3. 工厂函数：
   1. int(),long(),float(),complex(),bool()
   2. str(),unicode(),basestring()
   3. list(),turple(),dict(),set(),frozenset()
   4. type()
   5. object(),classmethod(),staticmethod(),super(),property(),file()

## 标准类型的分类

1. 存储模型：
   1. 看看这种类型的对象能保存多少个对象
   2. 分为：
      1. 标量/原子类型：数值、字符串
      2. 容器类型：列表、元组、字典
2. 更新模型：
   1. 对象创建之后，它的值是否更新
   2. 分为：
      1. 可变类型：列表，字典
      2. 不可变类型：数字、字符串、元组
3. 访问模型：
   1. 根据访问存储的数据的方式进行分类
   2. 分为：
      1. 直接访问：数字
      2. 顺序访问：字符串、列表、元组
      3. 映射访问：字典
         1. 索引并不适用顺序的数字偏移量取值，元素无序存放，通过一个唯一的key访问

## 不支持的类型

1. python目前还不支持的数据类：
   1. char：可以用长度为1的字符串表示
   2. byte：可以用整型表示
   3. 指针：pyhton解释器管理内存，因此没必要访问指针

# 数字

## 数字类型

1. 数字提供了标量贮存和直接访问
2. 数字是不可变类型，也就是说变更数字的值会生成新的对象。
3. Python的数字类型：整型、长整型、布尔型、双精度浮点型、十进制浮点型、复数
4. 对数字的更新实际上是生成了一个新的数值对象，并得到它的引用
5. 数字对象无法真正自己删除，仅仅是不再使用
   1. 可以通过del语句删除一个数值对象的引用

## 整型

1. 布尔型：
   1. 只有True和False两个值
2. 标准整数类型：
   1. 32位机器范围是-2^31~2^31-1
   2. 64位机器范围是-2^63~2^63-1
   3. python的整型等价于C的有符号长整型
   4. Python支持八进制或十六进制表示整数，默认十进制
      1. 八进制以数字0开始
      2. 十六进制以0x或0X开始
3. 长整型
   1. 它和C的长整型不同，Python的长整数类型能表达很大的数，仅仅与机器支持的内存大小有关
   2. 在一个整数后面加上L或l，表示这个整数是长整数
4. 整型和长整型的统一
   1. 这两种类型正在逐渐统一为一种
   2. Pyhton2.2以后标准整数类型不再会因为超出取值范围溢出了

## 双精度浮点数

1. 类似于C语言的double类型，可以直接用十进制或者科学计数法
2. 每个浮点数占8个字节，52个比特用于底，11个比特用于指数，1个比特用于符号
3. 浮点数值通常有一个小数点和一个可选的后缀E\e。
   1. 在e和指数之间可以用+或-表示指数的正负，其中+可以省略

## 复数

1. 叙述不能单独存在，总是和一个0.0的示数部分一起构成一个复数
2. 复数由实数部分和虚数部分构成
3. 表示复数的语法：real+imagej
4. 虚数部分必须有后缀j或J
5. 实数部分和虚数部分都是浮点数
6. 复数的内建属性：
   1. 复数对象拥有实部real和虚部image两个数据属性
   2. 复数拥有conjugate方法，返回共轭复数对象

## 运算符

1. 混合模式运算符：
   1. 相加：
      1. 当两个整数相加时，+号表示整数加法；
      2. 当两个浮点数相加时，+号表示浮点数加法；
      3. 当两个字符串相加时，表示连接操作，生成新的字符串
      4. 当一个整数和一个浮点数相加时，系统会决定使用浮点数加法
   2. 如果某种数据类型支持某种运算符，该数据类型就必须告诉Python，+运算符该如何工作，类似于C++的重载
   3. 两种不同类型相加时遵循的规则：
      1. 如果能够转换，解释器会自动进行转换
      2. 也可以用coerce()实现这种转换：
         1. 如果一个操作数是复数，另一个操作数转换为复数
         2. 否则，如果一个操作数是浮点数，另一个操作数转换为浮点数
         3. 否则，如果一个操作数是长整数，另一个操作数转换为长整数
         4. 否则，如果都是普通整数，则无需转换
2. 标准类型运算符：
   1. 标准运算符都可以用于数值类型。
3. 算数运算符：
   1. 单目运算符：
      1. 正号（+）
      2. 负号（-）
   2. 双目运算符：
      1. 加法（+）
      2. 减法（-）
      3. 乘法（\*）
      4. 除法（/）：对浮点操作数执行真正的除法
      5. 整除（//）：不管操作数何种类型，总是舍去小数部分
      6. 取余（%）
      7. 幂运算（\*\*）：比左侧一元操作符优先级低，比右侧一元操作符优先级高
   3. 优先级：\*\*》单目+》单目-》\*》/》//》%》双目+》双目-
4. 位运算符（只适合与整数）：
   1. 位运算操作符：
      1. 取反（~）
      2. 按位与（&）
      3. 或（|）
      4. 异或（^）
      5. 左移（<<）
      6. 右移（>>）
   2. 负数会被当成正数的二进制补码处理
   3. 对长整数，位运算符使用一种经修改的二进制补码形式，使符号位可以无限向左扩展
   4. 优先级（从高到低）：~ << >> & ^ |

## 内建函数与工厂函数

1. 标准类型函数：
   1. cmp()、str()、type()
2. 数字类型函数：
   1. 转换工厂函数：
      1. int(obj,base=10)：将其他类型转换为整型，接受字符串参数
         1. 在转换字符串时，接受进制参数。数字类型间的转换不适用
      2. long(obj,base=10)：将其他类型转换为长整型，接受字符串参数
         1. 在转换字符串时，接受进制参数。数字类型间的转换不适用
      3. float(obj)：将其他类型转换为浮点型，接受字符串参数
      4. complex(real,image=0.0)：将其他类型转换为复数，接受字符串参数
      5. bool(obj)：将整数值1和0转换为True和False
      6. 以上内建函数都成为工厂函数
   2. 功能函数 ：
      1. abs(obj)：返回给定参数的绝对值
         1. 如果是复数，返回math.sqrt(num.real^2+num.image^2)
      2. coerce(num1,num2)：不依赖与python解释器，自定义两个数值类型转换方法
         1. 如果一个操作数是复数，另一个操作数转换为复数
         2. 否则，如果一个操作数是浮点数，另一个操作数转换为浮点数
         3. 否则，如果一个操作数是长整数，另一个操作数转换为长整数
         4. 否则，如果都是普通整数，则无需转换
      3. divmod(被除数,除数)：返回一个包含商和余数的元组
         1. 对整数来说，是地板除和取余
         2. 对浮点数来说，是真正除法
         3. 对复数来说，是math.floor((num1/num2).real)
      4. pow(底,幂,[余])：和\*\*一样，可以用于指数运算
         1. 如果有第三个参数，指数运算完会和第三个参数进行取余运算
      5. round(obj,小数位)：对浮点数进行四舍五入计算
         1. 如果没小数位，返回正数
         2. 如果有小数位，结果精确到小数位指定的位数
      6. int()、math.floor()、round()的区别：
         1. int()：返回整数，直接截去小数部分
         2. floor()：返回浮点数，返回小于原数的最大整数
         3. round()：返回浮点数，返回最接近原数的整数
   3. 仅用于整数的函数
      1. oct(obj)：接受任意进制整数，返回8进制的字符串
      2. hex(obj)：接受任意进制整数，返回16进制的字符串
      3. chr(obj)：接受单字节（0~255）整数，返回ascii对应字符
      4. ord(obj)：接受字符，返回对应ascii码
      5. unichr(obj)：接受Unicode码值，返回对应Unicode字符

## 其他数字类型

1. 布尔数：
   1. 布尔类型的主要概念：
      1. 有两个永不改变的值True或False
      2. 是整型的子类，但是不能再被继承
      3. 没有\_\_nonzero\_\_()方法的对象的默认值是True
      4. 对于值为0的任何数字或空集，在python中的布尔值都是False
      5. 在数字运算中，True对应1，False对应0
      6. True和False是关键字（看是哪个版本的python）
   2. 所有Python对象都有一个内建的True或False值
2. 十进制浮点数：
   1. 和其他数值类型一样，可以用同样的算数运算符
   2. 0.1的二进制表示是0.11001100110011...
   3. 0.1的十进制表示就是0.1000000000

## 相关模块

1. decimal模块：十进制浮点运算类
2. array模块：高效数值数组（字符，整数，浮点数等）
3. math/cmath：标准c库数学运算函数
   1. 常规数学运算在math模块
   2. 复数运算在cmath模块
4. operator：数学运算符的函数实现
   1. 比如operator.sub(m,n)等价于m-n
5. random：多种伪随机数生成器
   1. 以当前的时间戳为随机数种子

# 序列：字符串、列表、元组

## 序列

1. 序列：成员是有序排列的。包括字符串、列表、元组这三种类型
2. 序列类型有相同的访问模式：
   1. 每一个元素可以通过指定一个偏移量的方式得到。
   2. 多个元素可以通过切片操作的方式一次得到
   3. 下标偏移量从0开始
   4. 下标偏移量到总元素数-1结束
3. 标准类型操作符：
   1. 标准类型操作符一般都能适用于所有序列类型
4. 序列类型操作符：
   1. 优先级从上到下：
      1. seq[ind]：获得下标为ind的元素
      2. 切片操作符seq[ind1:ind2]：获得下标从ind1到ind2间的元素集合
         1. 返回的是左闭右开区间，即包含ind1而不包含ind2
         2. 起始索引和终止索引都是可选的
            1. 没有起始索引，则从0开始
            2. 没有终止索引，则一直到结束
         3. 还可以有第三个参数，步长：seq{ind1:ind2:step]
            1. 默认步长为1
            2. 步长为-1则表示翻转序列
            3. 步长为2则表示隔一个取一个
      3. 重复操作符seq \* expr：序列重复expr次
         1. expr必须是一个整数，但不能是长整数
         2. 返回一个新的包含多份原对象拷贝的对象
      4. 连接操作符seq1 + seq2：序列seq1连接序列seq2
         1. 对字符串来说不如join()节约内存
         2. 对列表来说不如extend()
      5. 成员关系操作符obj in seq：判断obj元素是否在seq中
      6. 成员关系操作符obj not in seq：判断obj元素是否不在seq中
5. 内建函数：
   1. 类型转换：
      1. list(iter)：把一个可迭代对象转化为一个列表
      2. str(obj)：把一个obj对象转化为字符串
      3. unicode(obj)：把对象转换为unicode字符串
      4. basestring()：抽象工厂，str()和unicode()的父类，无法实例化
      5. tuple(iter)：把一个可迭代对象转换为一个元组对象
      6. 可迭代对象包含但不仅限与序列对象
      7. 以上实际都是工厂函数，把参数内容浅拷贝到新生成的对象中
      8. 浅拷贝即只拷贝了对对象的索引，而不是重新创建对象
   2. 操作函数：
      1. enumerate(iter)：返回一个迭代器enumerate0
      2. len(seq)：返回seq的长度
      3. max(iter,key=None)或max(arg1,arg2,...,key=None)：返回最大值
         1. key必须是一个可以传给sort()的用于比较的函数
      4. min(iter,key=None)或min(arg1,arg2,...,key=None)：返回最小值
         1. key必须是一个可以传给sort()的用于比较的函数
      5. reversed(seq)：返回一个以逆序访问的迭代器
      6. sorted(iter,func=None,key=None,reverse=False)：返回一个有序的列表
      7. sum(seq,init=0)：返回序列和init的总和
      8. zip([iter0,iter2,...,iterN])：返回一个列表，各元素对应各迭代器的第一个元素组成的元组

## 字符串

1. 字符串一些基本特性：
   1. 可以通过在引号间包含字符的方式创建字符串
   2. python里面单引号和双引号的作用是相同的
   3. 字符串是不可变类型，也就是改变一个字符串时会创建一个新的字符串，这一点和JAVA类似
   4. 字符串是由独立的字符组成的，这些字符可以通过切片操作顺序地访问
2. 字符串的创建和赋值：
   1. 就像使用标量一样直接创建并赋值：str = ‘hello world’
   2. 也可以通过str()创建一个字符串并赋给一个变量
3. 字符串前面加u表示unicode字符串：
   1. u”hello world”
4. 如何访问字符串的值（字符和子串）
   1. Python没有字符这个类型，可以用长度为1的字符串表示
   2. 通过索引str[0]可以得到某个字符
   3. 通过切片str[0:5]可以得到子串
5. 如何改变字符串：
   1. 改变字符串和创建字符串类似，实际原理也并没什么区别。因为字符串是不可变类
6. 如何删除字符和字符串：
   1. 要去掉某个字符，进行切片和连接操作即可：
      1. str = str[:3]+str[4:]，这样可以删除第4个字符

## 字符串和操作符

1. 标准类型操作符：
   1. 可以用所有比较操作符
   2. 字符串比较操作时，是按照ascii值的大小来比较的
2. 序列操作符：
   1. 可以访问[0,len(str)-1]在内的所有索引
   2. 切片的start和end和step都可以不指定
3. 成员操作符：
   1. 可以通过in和not in判断一个字符或子串是否出现在另一个字符串中
   2. 通过连接符（+）连接两个字符串，组合成新的字符串
   3. Python的语法允许在源码中把几个字符串连在一起写，以此构建为新的字符串：
      1. str = ‘hello’’world’
   4. Unicode字符串语法：
      1. U”hello”或u”hello”
   5. 带Unicode的字符串和普通字符串连接，最终会连接为Unicode字符串
   6. 重复操作符（\*）：创建一个包含多个原有字符串的拷贝的新字符串

## 只适用于字符串的操作符

1. 格式化操作符（%）：
   1. 类似于C语言里面的printf()函数的字符串格式化
   2. 这一部分先跳过
2. 字符串模板：
   1. 跳过
3. 原始字符串操作符：
   1. 跳过
4. Unicode字符串操作符
   1. 跳过

## 内建函数

1. 标准类型函数
   1. cmp(str1,str2)：同比较操作符一样，根据字符串的ascii码值比较
2. 序列类型函数：
   1. len(str)：返回字符串的字符数
   2. max(str)：返回ascii码值最大的字符
   3. min(str)：返回ascii码值最小的字符
   4. enumerate(str)：返回字符串的迭代器
   5. zip(str1,str2)：返回str1匹配str2的二元组列表
3. 字符串类型函数：
   1. input(str)：给定str字符串提示用户输入，并返回用户输入的字符串
   2. str(obj)和unicode(obj)：工厂函数，分别给出对象可打印和unicode的字符串表示
   3. chr()和unichr()：分别根据码值返回ascii字符或unicode字符
   4. ord()：根据ascii字符或unicode字符返回码值
4. 字符串内建函数（只记下了经常用的）：
   1. string.capitalize()：把字符串第一个字符大写
   2. string.count(str,beg=0,end=len(string))：返回范围内str在string中出现的次数
   3. string.endswith(obj,beg=0,end=len(string))：返回范围内字符串是否以obj结束
   4. string.startswith(obj,beg=0,end=len(string))：返回范围内字符串是否以obj开始
   5. string.find(str,beg=0,end=len(string))：检查范围内是否包含str串
      1. 包含则返回str的索引
      2. 不包含则返回-1
   6. string.index(str,beg=0,end=len(string))：和find类似
      1. 不同在于如果不包含则报错，而不是返回-1
   7. string.join(seq)：以string为分隔符，将seq所有元素合并为一个新的字符串
   8. string.lower()：转换所有大写字符为小写
   9. String.upper()：转换所有小写字符为大写
   10. string.replace(str1,str2,num=string.count(str1))：把str1替换成str2,
       1. 替换不超过num次
   11. string.split(str=””,num=string.count(str))：以str为分隔符切片string
       1. 仅分割num个字符串
   12. string.strip([obj])：删除string前面和后面的空格

## 字符串的特殊特性

1. 特殊字符和控制字符
   1. \n换行
   2. \r回车
   3. \t制表符
   4. \0空字符Nul
   5. \”双引号
   6. \’单引号
   7. [\\反斜杠](\\\\反斜杠)
   8. \连字符
   9. \000~\0177是八进制值
   10. \x00~\xFF是十六进制值
   11. 不可打印不经常使用的ascii比较实用用来作为定界符
   12. 其他的不列举了
2. 三引号：
   1. 允许一个字符串跨多行，其中可以包含换行符、制表符以及其他特殊字符
3. 字符串不变性：
   1. 即上面提过的字符串时不可变类
   2. 对字符串的字符或切片改变是不允许的

## Unicode

跳过该部分

## 字符串相关模块

1. string：字符串操作相关函数和工具
2. re：正则表达式
   1. 之后学习
3. struct：字符串和二进制之间的转换
4. 其他的暂不列举

## 列表

1. 创建列表类型数据并赋值：
   1. 手工创建：list = [a,b,[c,d]]
   2. 工厂方法创建：list = list(‘hello’)
2. 访问列表中的值：
   1. 列表的切片操作和字符串相同
   2. list[0]访问某个元素，list[0,len(list)]访问多个元素
3. 更新列表：
   1. 可以直接改变某个元素：list[0] = 2
   2. 也可以追加元素：list.append(2)
4. 删除列表元素
   1. 通过索引删除：del list[1]
   2. 通过值删除：list.remove(123)

## 列表操作符

1. 标准类型操作符：
   1. 比较操作符，用的内建的cmp()函数
   2. 基本比较逻辑：两个列表的元素分别比较，知道有一方的元素比较成功。各元素的比较就按照两个元素类型进行比较
   3. 元组类型在进行比较时跟列表遵循相同的逻辑
2. 序列类型操作符：
   1. 列表的切片操作返回的是一个对象或几个对象的集合
   2. 和字符串类似，列表的切片操作也遵从正负索引规则，开始索引、结束索引、步数规则基本类似
   3. 和字符串不同，对切片的结果也可以直接改变
   4. 成员关系操作（in，not in）：
      1. 可以检查一个对象是否是一个列表（或元组）的成员
   5. 连接操作符（+）：
      1. 列表的连接操作只能在列表间进行
      2. 可以用extend()方法替代连接操作符。它和连接操作符的区别是：
         1. extend()方法把新列表添加到了原有的列表里
         2. 连接操作时创建一个新的列表，把原来需要连接的移到新列表里
      3. 连接操作符不能替换替换向列表添加新元素的操作。需要用append(obj)
   6. 重复操作符（\*）：
      1. 重复操作符可以使列表重复扩展成n倍
   7. 列表支持复合赋值运算：
      1. list \*= 2
      2. list += ‘hello’
3. 列表类型操作符和列表解析：
   1. Python中没有专门用于列表类型的操作符，但列表可以使用大部分对象和序列类型的操作符。
   2. 但只有列表能使用的列表解析：
      1. [i \* 2 for i in [2, 3, 6]] -> [4,6,12]
      2. [i for i in range(8) if i % 2 == 0] -> [0,2,4,6]

## 列表内建函数

1. 标准类型函数：
   1. cmp()：列表和元组用的算法相同。算法关键点如下：
      1. 对两个列表的元素进行比较
      2. 如果比较的元素同类型，则标胶其值，返回结果
      3. 如果比较的元素不同类型，则检查是否数字
         1. 如果是数字，执行必要的强制转换后比较
         2. 否则，如果有一方是数字，则另一方大（数字最小）
         3. 否则，通过字母顺序进行比较
      4. 如果有一个列表首先到达末尾，则另一个列表大
      5. 如果两个列表比较完都是相等的，则返回0
2. 序列类型函数：
   1. len()：返回列表的元素个数，和元组相同
   2. max()：返回列表中最大的元素，但列表结构越复杂，结果准确性越差
   3. min()：返回列表中最小的元素，但列表结构越复杂，结果准确性越差
   4. sorted()：排序列表，对字符串使用的是ascii码序，而不是字母序
   5. reversed()：转置列表
   6. enumerate()、zip()：和字符串类似
   7. sum()：将列表所有元素相加
   8. list(iter)、tuple(iter)：通过浅拷贝创建一个新的列表或元组
      1. 大多用于列表元组之间的转换
3. 列表类型内建函数：
   1. Python没有特定用于列表的内建函数

## 列表类型的内建函数

1. 这些函数涉及到对列表更改，因此不适应于元组
2. list.append(obj)：想列表添加一个对象
3. list.count(obj)：返回一个对象在列表中出现的次数
4. list.extend(seq)：把序列内容添加到列表中
5. list.index(obj,beg=0,end=len(list)：返回范围内值为obj的索引
   1. 找不到则抛出异常
6. list.insert(index,obj)：在索引index位置插入obj
7. list.pop(index=-1)：删除并返回指定位置的对象，默认是最后一个对象
8. list.remove(obj)：从列表删除obj对象
9. list.reverse()：翻转列表
10. list.sort(func=None,key=None,reverse=False)：对列表排序
11. 一般像reverse()、sort()这类可以改变对象的方法是没有返回值的，python并不是每个方法都有返回值。而字符串因为不可变，方法是有返回值。

## 列表的特殊特性

1. 用列表构建其它数据结构：
   1. 堆栈：这是一个后进先出的数据结构。类似于数组一样，给出pop()和push()方法
   2. 队列：这是一个先进先出的数据结构。类似于数组一样，给出pop()和push()方法

## 元组

1. 元组和列表非常相近，只有些许差别：
   1. 元组用小括号（()），列表用中括号（[]）
   2. 元组不可变，里诶包可变
2. 元组能做的事：
   1. 用作一个字典的key
   2. 当处理一组对象时，这个组默认是元组
3. 创建一个元组并赋值：
   1. 和列表类似tuple = (1,2,3)
   2. 只有一个元素的时候要在末尾加逗号（,），用以区分：tuple=(1,)
4. 访问元组中的值：
   1. 和列表类似，用切片操作符：tuple[1],tuple[1:4]
5. 更新元组：
   1. 因为是不可变对象，更新后实际上指向的就是另一个对象了
6. 删除元组元素：
   1. 同样，只能足够一个新的元组，不加上要删除的元素即可

## 元组操作符和内建函数

1. 创建、重复、连接、成员关系、切片等操作和列表几乎完全相同
2. 操作符和列表也几乎相同
3. str(t)、len(t)、max(t)、min(t)、cmp(t)、list(t)等操作也和列表几乎相同
4. 因为元组不可变，排序、替换、添加等方法没有被实现

## 元组的特殊特性

1. 不可变性：
   1. 一旦一各对象被定义了，它的值就不能再被更新，除非创建新的对象
   2. 切片操作符不能用作左值进行赋值，只能用于只读操作
   3. 这样可以确保数据不会被修改
2. 灵活性不变：
   1. 连接操作和重复操作等都可以用，只不过是生成一个新的元组而已
   2. 虽然元组本身不可变，但可以修改元组中的可变元素。
3. 默认集合类型：
   1. 所有多对象的、逗号分隔的、没有明确用符号定义的，这些集合默认的类型都是元组
   2. 所有函数返回的多对象（除了有符号封装的）都是元组类型：
      1. 比如return obj1,obj2,obj3
      2. 而return [obj1,obj2,obj3]就不是
   3. 为了避免歧义，建议总是用显式的圆括号表示或创建一个元组
      1. 比如 4,2<3,5 本意是要比较两个元组，但会变成(4,True,5)
4. 单元素元组：
   1. 因为由圆括号包裹的单一元素首先会被作为分组操作
   2. 因此，单元素元组只能在后面加一个逗号，表示是元组：
      1. (1,)
5. 字典的关键字：
   1. 为了确保hash算法得到的值总是一个值，因此元组适合做字典的键

## 与序列类型相关的模块

1. copy：提供浅拷贝和深拷贝的能力
2. operator：包含函数调用形式的序列操作符
   1. 比如operator.concat(m,n)等价于连接操作m+n
3. re：perl风格的正则表达式
4. collections：高性能容器数据类型

## 浅拷贝和深拷贝

1. 对一个对象进行浅拷贝其实是新创建了一个类型和原对象一样，内容是原来对象元素的引用。也就是拷贝的对象本身是新的，内容是原来的。
2. 序列类型对象的默认拷贝类型是浅拷贝，浅拷贝为copy.copy()。
3. copy.deepcopy()可以进行深拷贝：对原对象的内容也递归地创建并拷贝，而不是直接引用原来的对象
4. 只有容器类型才有拷贝的概念，浅拷贝是用完全切片操作完成的
5. 如果元组中的对象都是不可变类，那么对它的深拷贝将不会进行

# 映像和集合类型

## 映射类型：字典

1. 字典是python中唯一映射类型。
2. 映射类型对象里，哈希值（键）和指向的对象（值）是一对多的关系
3. 一个字典对象是可变的，它是一个容器类型，能存储任意多个python对象，当然也能存储其他容器类型。
4. 序列类型只用数字作键（索引）；映射类型可以用其他对象类型作键，最常用字符串。
5. 映射类型的键不再是序列化排序的，所以映射类型中的数据是无序排序的。虽然是无序的，但可以直接通过键映射到值。
6. 创建字典和给字典赋值：
   1. 创建字典只需要把字典赋给一个变量，不管是否包含元素：
      1. dict = {}
      2. dict = {‘name’:’jim’,’score’:1}
   2. 工厂方法dict()也可以创建字典：
      1. dict = dict(([‘x’,1],[‘y’,2]))
   3. 内建方法fromkeys()可以创建默认字典，字典元素具有相同的值，如果没给出默认为None：
      1. dict = {}.formkeys((‘a’,’b’,’c’),1)
7. 访问字典中的值：
   1. 要遍历一个字典，只需要循环它的键：
      1. for key in dict.keys()
      2. 然后通过键索引：dict[key]
   2. 也可以直接循环字典本身，效果相同：
      1. for key in dict
   3. 如果访问字典不存在的键，将会报错
      1. 通过has\_key()和in和not in可以先判断字典是否有该键（has\_key将被弃用）
8. 更新字典：
   1. 字典是可变类
   2. 可以根据键更改值
   3. 可以直接创建新的键值关系
9. 删除字典元素：
   1. 删除某个条目：del dict[‘xxx’]]
   2. 清空字典所有条目：dict.clear()
   3. 删除整个字典：del dict
   4. 删除并返回某个条目：dict.pop(‘xxx’)
10. 注意，像dict、list、file、bool、str、inpu、len这些本身就是方法名，应该避免使用它们作为变量的名字

## 映射类型操作符

1. 标准类型操作符：
   1. 像比较操作符等都是支持的
2. 映射类型操作符：
   1. 查找操作符（[]）：用键来查询值，比如dict[key]
   2. 成员关系操作（in，not in）：验证某个键是否存在于字典中，比如’key’ in dict

## 映射类型的内建函数和工厂函数

1. 标准类型函数：
   1. type()、str()、cmp()：字典的比较一般不是很有用
2. 映射类型相关的函数：
   1. dict()：工厂函数被用来创建字典
      1. 不提供参数会生成空字典
      2. 如果是容器对象作为参数，那其中的元素必须成对出现
         1. 比如dict([[‘x’,1],[‘y’,x]])
      3. 如果参数是另一个字典对象，那么会进行浅拷贝
         1. 和copy.copy()相同，但速度慢
   2. len()：返回字典键值对的数目
   3. hash()：判断某个对象是否可以做一个字典的键
      1. 如果可哈希，产生整数，表示可以做字典的键
      2. 如果不可哈希，会产生异常

## 映射类型内建方法

1. dict.keys():返回另一个列表，包含字典中所有键
2. dict.values():返回另一个列表，包含字典中所有值
3. dict.items():返回一个列表，包含所有键值对元组
4. dict.clear():删除字典所有元素
5. dict.copy():返回字典的一个浅拷贝
6. dict.fromkeys(seq,val=None):创建并返回一个字典，以seq中的元素作为键，val作为所有键对应的初始值
7. dict.get(key,default=None):对字典中的键key，返回它对应的值，如果不存在返回default的值
8. dict.pop(key,[default]):如果字典key存在，删除并返回dict[key]
   1. 如果key不存在，且没有给出default，引发异常
9. dict.setdefault(key,default=None):如果不存在key，设置dict[key]=default
10. dict.update(dict2)：将字典dict2的键值对添加到dict
    1. 如果键原来已有，则新的值覆盖原来的值

## 字典的键

1. 不允许一个键对应多个值：
   1. 当有键发生冲突，即键重复赋值，取最后的赋值
2. 键必须是可哈希的：
   1. 所有不可变的类型都是可哈希的
   2. 值相等的数字表示相同的键
      1. 比如整型数字1和浮点数1.0，它们是相同的键
   3. 如果键发生变化，哈希函数会映射到不同的地址，这样不能可靠地存储
   4. 如果要用元组做键，那么它的元素必须都是不可变的

## 集合类型

1. python引入了数学上的集合：
   1. 集合对象是一组无序排序的可哈希的值
   2. 集合成员可以做字典中的键
   3. 因为是无序的
      1. 可以用in、not in、len、for循环迭代集合成员
      2. 不可以创建索引、切片，也没有键来获取集合中的元素的值
   4. 集合有可变集合和不可变集合：
      1. 可变集合：可以添加删除元素，不是可哈希的
      2. 不可变集合：不可以添加删除元素，可哈希
2. 创建集合类型和给集合赋值：
   1. 唯一方法：用集合的工厂方法：
      1. set()：可变集合
      2. frozenset()：不可变集合
3. 访问集合中的值：
   1. 可以看某对象是否是一个集合中的值：
      1. ‘k’ in set
      2. ‘k’ not in set
   2. 遍历集合：
      1. for i in set:
4. 更新集合：
   1. 添加元素：
      1. set.add(‘a’)
   2. 更新集合元素：
      1. set.update(‘abc’)
      2. 如果有添加，没有则不做任何操作
5. 删除集合元素：
   1. set.remove(‘z’)

## 集合类型操作符

1. 标准类型操作符：
   1. 成员关系(in,not in)：判断一个元素是否是一个集合的成员
   2. 集合等价/不等价(==,!=)：集合中的元素是否全部相同
      1. 与集合的类型和集合成员的顺序无关
   3. 子集/超集（<,<=,>,>=）：和数学上的子集和超集意义相同
2. 集合类型操作符：
   1. 联合(|):合并两个集合，取并集
      1. 等价于union()
   2. 交集(&):取两个集合共有的元素
      1. 等价于intersection()
   3. 差集(-):a-b，取属于a且不属于b的元素
      1. 等价于defference()
   4. 对称差分(^):a^b，取要么属于a，要么属于b，而不同时属于ab的元素
      1. 等价于symmetric\_difference()
   5. 混合集合类型操作符：
      1. 如果两个集合类型相同，产生的结果仍然是这个类型
      2. 如果两个集合类型不同，产生的结果与操作符左侧的集合类型相同
3. 集合类型操作符（仅适用于可变集合）
   1. 合并更新(|=)：从已存在的集合添加多个成员
      1. 等价于update()
   2. 保留更新(&=)：只保留与右侧集合的共同成员
      1. 等价于intersection\_update()
   3. 差更新(-=)：去掉右侧集合的元素
      1. 等价于difference\_update()
   4. 对称差分更新(^=)：加上没有的成员，同时去掉右侧有的成员
      1. 等价于symmetric\_difference\_update()

## 内建函数

1. 标准类型函数：
   1. len(set)：返回集合中元素的个数
2. 集合类型工厂函数：
   1. set()：生成可变集合
   2. frozenset()：生成不可变集合
   3. 不提供参数时默认生成空集合
   4. 提供的参数必须是可迭代的，即序列、迭代器或支持迭代的对象（文件、字典等）

## 集合类型内建方法

1. 适用于所有集合的方法：
   1. set.issubset(t)：判断集合是否是t的子集
   2. set.issuperset(t)：判断集合是否是t的超集
   3. set.union(t)：返回集合和t的并集
   4. set.intersection(t)：返回集合和t的交集
   5. set.difference(t)：返回集合和t的差集
   6. set.symmetric\_difference(t)：返回集合和t的对称差分集
   7. set.copy()：返回集合的浅复制
2. 仅适用于可变集合的方法：
   1. 上面提到过的update()、intersection\_update()、difference\_update()、symmetric\_difference\_update()等
   2. s.add(obj)：在集合中添加对象obj
      1. 参数必须是可哈希的
   3. s.remove(obj)：在集合中删除对象obj
      1. 如果不存在，引发错误
      2. 参数必须是可哈希的
   4. s.discard(obj)：在集合中删除对象obj
      1. 参数必须是可哈希的
      2. 不存在则不执行任何操作
   5. s.pop()：删除集合任意一个对象，并返回
   6. s.clear()：删除集合所有元素

## 相关模块

1、set模块，可继承Set或ImmuteableSet来生成子类