**字典和集合**

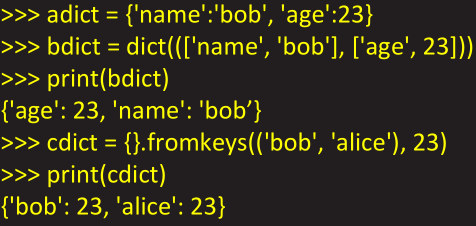
字典基础操作

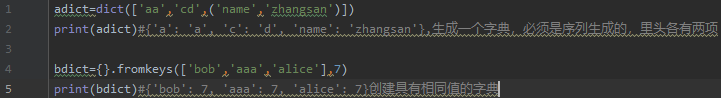
创建字典

通过{}操作符创建字典

通过dict()工厂方法创建字典

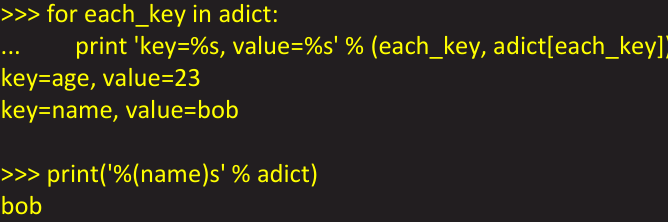
通过fromkeys()创建具有相同值的默认字典

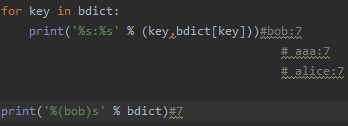




访问字典

字典是映射类型，意味着它没有下标，访问字典中的值需要使用相应的键



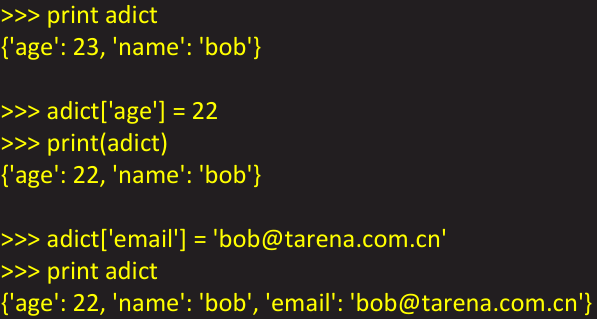


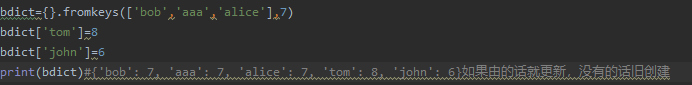
更新字典

通过键更新字典

如果字典中有该值，则更新相关值

如果字典中没有该值，则向字典中添加新值



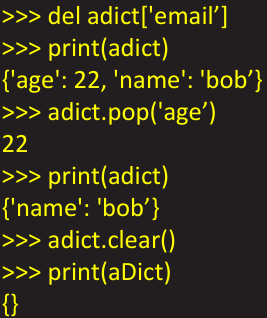


删除字典

通过del可以删除字典中的元素或整个字典

使用内部方法clear()可以清空字典

使用pop()方法可以弹出字典中的元素

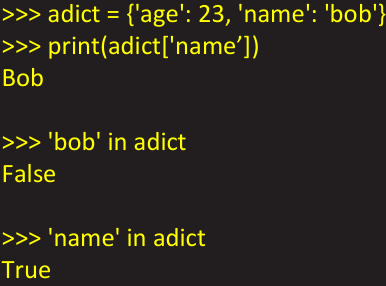


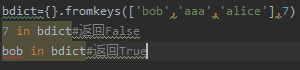


字典操作符

使用字典键查找操作符[]，查找所对应的值

使用in和not in判断键是否存在于字典中



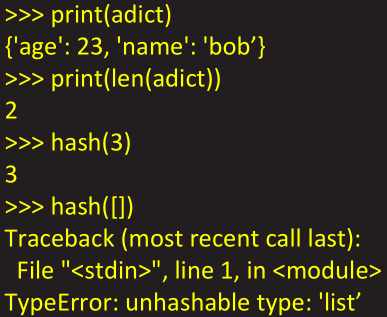


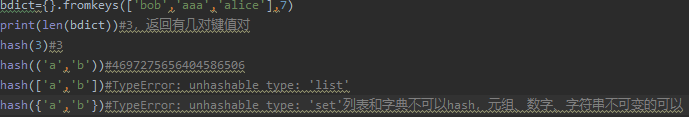
字典相关函数

作用于字典的函数

len():返回字典中元素的数目

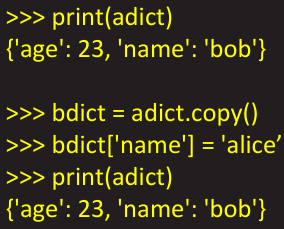
hash():本身不是为字典设计的，但是可以判断某个对象是否可以作为字典的键





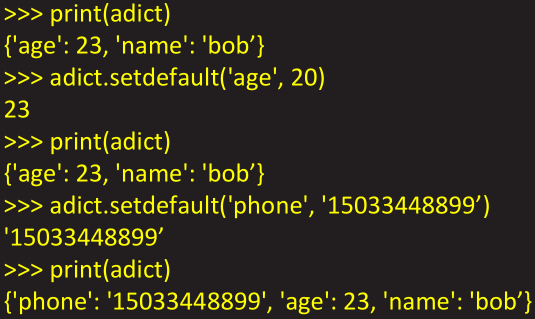
字典的内建方法

dict.copy():返回字典（深复制）的一个副本



dict.get(key,default=None)：对字典dict中的键key，返回它对应的值value，如果字典中不存在此键，则返回default的值

dict.setdefault(key,default=None)：如果字典中不存在key值，由dict[key]=default为它赋值

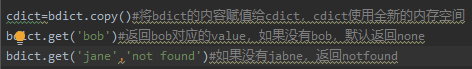


dict.items()：返回一个包含字典中（键，值）对元组的列表

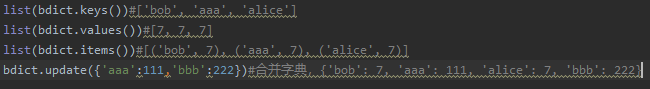
dict.keys()：返回一个包含字典中键的列表

dict.values()：返回一个包含字典中所有值的列表

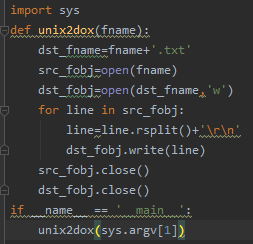
dict.update(dict2)：将字典dict2的键-值对添加到字典dict











创建集合

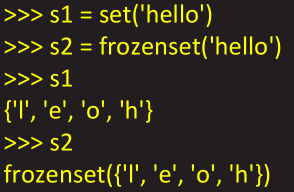
数学上，把set称作由不同的元素组成的集合，集合（set）的成员通常被称做集合元素

集合对象是一组无需排列的可哈希的值

集合由两种类型

可变集合set

不可变集合frozenset



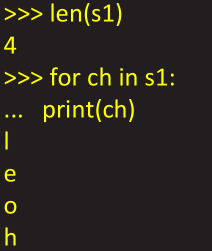
集合类型操作符

集合支持用in和not in操作符检查成员

能够通过len()检查集合大小

能够使用for迭代集合成员

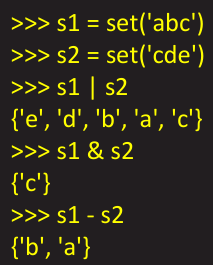
不能取切片，没有键



|：联合，取并集

&：交集

-：差补

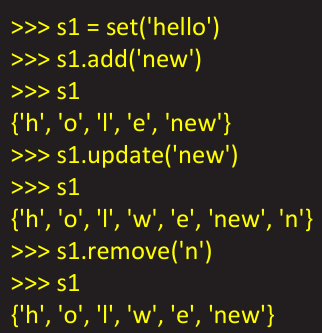


集合内建方法

set.add()：添加成员

set.update()：批量添加成员

set.remove()：移除成员



s.issubset(t)：如果s是t的自己，则返回true，否则返回false

s.issuperset(t)：如果t是s的超集，则返回true，否则返回false

s.union(t)：返回一个新集合，该集合是s和t的并集

s.intersection(t)：返回一个新集合，该集合是s和t的交集

s.difference(t)：返回一个新集合，该集合是s的成员,但不是t的成员

**时间方法**

time模块

时间表示方法

时间戳timestamp：表示的是从1970年1月1日00:00:00开始按秒计算的偏移量

UTC（世界协调时间）也是格林威治天文时间，世界标准时间。在中国为UTC+8。DST即夏令时

元组：由9个元素组成

struct\_time元组

|  |  |  |
| --- | --- | --- |
| 索引 | 属性 | 值 |
| 0 | tm\_year | 2000 |
| 1 | tm\_mon | 1-12 |
| 2 | tm\_mday | 1-31 |
| 3 | tm\_hour | 0-23 |
| 4 | tm\_min | 0-59 |
| 5 | tm\_sec | 0-61 |
| 6 | tm\_wday | 0-6（0表示周一） |
| 7 | tm\_yday（一年中的第几天） | 1-366 |
| 8 | tm\_isdst（是否为dst时间） | 默认为-1 |

time模块方法

time.localtime([secs])：将一个时间戳转换为当前时区的struct\_time。secs参数未提供，则以当前时间为准

time.gmtime([secs])：和localtime()方法类似，gmtime()方法是将一个时间戳转换为UTC时区（0时区）的struct\_time

time.time()：返回当前时间的时间戳

time.mktime(t)：将一个struct\_time转换为时间戳

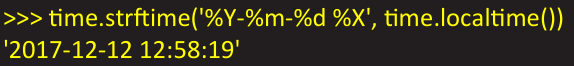
time.sleep(secs)：线程推迟指定时间运行。单位为秒

time.asctime([t])：把一个表示时间的元组或者struct\_time表示为这种形式：’sun jun 20 23:21:05 1993’。如果没有参数，将会将time.localtime()作为参数传入

time.ctime([secs])：把一个时间戳（按秒计算的浮点数）转换为time.asctime()的形式

time.strftime(format[,t])：把一个代表时间的元组或者struct\_time(如由time.localtime()和time.gmtime()返回)转换为格式化的时间字符串。如果t未指定，将传入time.localtime()

time.strptime(string[,format])：把一个格式化时间字符串转换为struct\_time。实际上它和strftime()是逆操作



时间样式

|  |  |  |  |
| --- | --- | --- | --- |
| 格式 | 含义 | 格式 | 含义 |
| %a | 本地简化星期名称 | %m | 月份（01-12） |
| %A | 本地完整星期名称 | %M | 分钟数（00-59） |
| %b | 本地简化月份名称 | %p | 本地am或者pm的相应符 |
| %B | 本地完整月份名称 | %s | 秒（01-61） |
| %c | 本地相应的日期和时间 | %U | 一年中的星期数（00-53，星期日是一个星期的开始） |
| %d | 一个月中的第几天（01-31） | %w | 一个星期中的第几天（0-6，0是星期天） |
| %H | 一天中的第几个小时（24小时制，00-23） | %x | 本地相应日期 |
| %I | 第几个小时（12小时制，01-12） | %X | 本地相应时间 |
| %j | 一年中的第几天（001-366） | %y | 去掉世纪的年份（00-99） |
| %Z | 时区的名字 | %Y | 完整的年份 |

datetime模块

datetime模块方法

datetime.today()：返回一个表示当前本地时间datetime对象

datetime.now([tz])：返回一个表示当前本地时间的datetime对象，如果提供了参数tz，则获取tz参数所指时区的本地时间

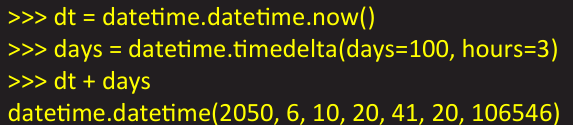
datetime.strptime(date\_string,format)：将格式字符串转换为datetime对象

datetime.ctime(datetime对象)：返回时间格式字符串

datetime.strftime(format)：返回指定格式字符串

时间计算

使用timedelta可以很方便的在日期上做天days，小时hour，分钟，秒，毫秒，为妙的时间计算



**异常处理**

异常

什么是异常

当python检测到一个错误时，解释器就会指出当前流已经无法继续执行下去，这时候就出现了异常

异常是因为程序出现了错误而在正常控制流以外采取的行为

这个行为又分为两个阶段

首先是引起异常发生的错误

然后是检测（和采取可能的措施）阶段

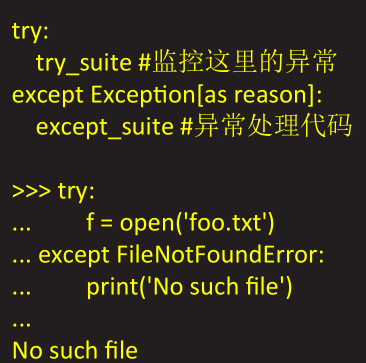
python中的异常

当程序运行时，因为遇到未解的错误而导致中止运行，便会出现traceback消息，打印异常

|  |  |
| --- | --- |
| 异常 | 描述 |
| NameError | 未申明/初始化对象 |
| IndexError | 序列中没有此索引 |
| SyntaxError | 语法错误 |
| KeyboardInterrupt | 用户中断执行 |
| EOFError | 没有内建输入，达到EOF标记 |
| IOError | 输入/输出操作失败 |

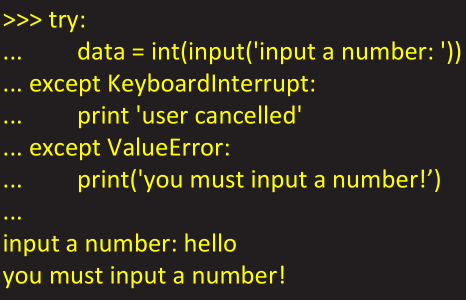
try-except语句

定义了进行异常监控的一段代码，并且提供了处理异常的机制



带有多个expect的try语句

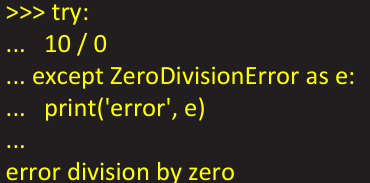
可以把多个except语句连接在一起，处理一个try块中可能发生的多种异常



异常参数

异常也可以有参数，异常引发后它会被传递给异常处理器

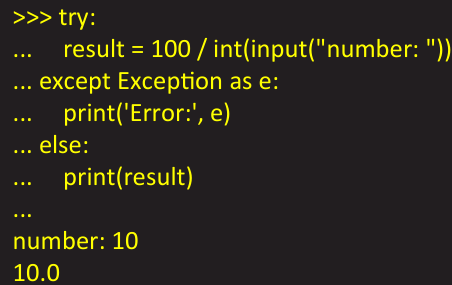
当异常被引发后参数是作为附加帮助信息传递给异常处理器的



else子句

在try范围内没有异常被检测到时，执行else子句

在else范围中的任何代码运行前，try范围内的所有代码必须完全成功



finally子句

finally子句是无论异常是否发生，是否捕捉都会执行的一段代码

如果打开文件后，因为发生异常导致文件没有关闭，可能会发生数据损坏。使用finally可以保证文件总是能正常的关闭

触发异常

raise语句

要想引发异常，最简单的形式就是输入关键字raise，后面跟要引发的异常的名称

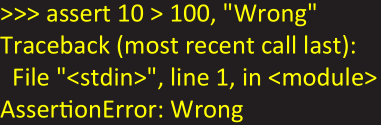
执行raise语句时，python会创建指定的异常类的一个对象

raise语句还可以指定对异常对象进行初始化的参数

断言

断言是一句必须等价于布尔值为真的判定

此外，发生异常也意味着表达式为假



**OS相关模块**

os模块

os模块简介

对文件系统的访问大多通过python的os模块实现

该模块是python访问操作系统功能的主要接口

有些方法，如copy等，并没有提供，可以使用shutil模块作为补充

os模块方法

|  |  |
| --- | --- |
| 函数 | 作用 |
| symlink() | 创建符号链接 |
| listdir() | 列出指定目录的文件 |
| getcwd() | 返回当前工作目录 |
| mkdir() | 创建目录 |
| chmod() | 改变权限模式 |
| getatime() | 返回最近访问时间 |
| chdir() | 改变工作目录 |

pickle模块

pickle模块简介

把数据写入文件时，常规的文件方法只能把字符串对象写入。其他数据虚先转成字符串再写入文件

python提供了一个标准的模块，称为pickle。使用它可以在一个文件中存储任何python对象，之后又可以把它完整无缺的取出来

pickle模块方法

分别调用dump()和load()可以存储、写入

