```
#encoding=utf-8

#**************** Part 1: 元组 *****************
#实际上元组是跟列表非常相近的另一种容器类型.元组和列表看起来不同的一点是元组用
#的是圆括号而列表用的是方括号。而功能上,元组和列表相比有一个很重要的区别,元组是
一种
#不可变类型.正因为这个原因,元组能做一些列表不能做的事情... 用做一个字典的 key.另外
当
#处理一组对象时,这个组默认是元组类型.

#创建一个元组并给它赋值
#创建一个元组并给他赋值实际上跟创建一个列表并给它赋值完全一样,除了一点,只有一个
#元素的元组需要在元组分割符里面加一个逗号(,)用以防止跟普通的分组操作符混淆.

#tuple1=(123, 'abc', 4.56, ['inner', 'tuple'], 7-9j)
#tuple2=(None, 'something to see here')
#print(tuple1)
#print(tuple2)
#-->
#(123, 'abc', 4.56, ['inner', 'tuple'], (7-9j))
#(None, 'something to see here')
#
#tuple3=(None,)
#tuple4=('Dave')
#print(tuple3)
#print(tuple4)
#--> 注意这里有逗号和没逗号的区别
#(None,)
#Dave

#访问元组中的值
#元组的切片操作跟列表一样,用方括号作为切片操符([]),里面写上索引值或者索引范围.
#tuple1=(123, 'abc', 4.56, ['inner', 'tuple'], 7-9j)
#print(tuple1[1:2],tuple1[3][1])
#-->
#('abc',) tuple
```

```
#更新元组
#跟数字和字符串一样,元组也是不可变类型,就是说你不能更新或者改变元组的元素,
#可以通过现有元祖的片段再构造一个新元祖的方式解决的
#tuple1=(123, 'abc', 4.56, ['inner', 'tuple'], 7-9j)
#tuple1 = tuple1[0], tuple1[1], tuple1[-1]
#print(tuple1)
#-->
#(123, 'abc', (7-9j))
#
#tup1 = (12, 34.56)
#tup2 = ('abc', 'xyz')
#tup3 = tup1 + tup2
#print(tup3)
#-->
#(12, 34.56, 'abc', 'xyz')


#移除一个元组的元素以及元组本身
#删除一个单独的元组元素是不可能的,当然,把不需要的元素丢弃后, 重新组成一个元组是没有问题的.
#要显示地删除一整个元组,只要用 del 语句减少对象引用计数.当这个引用计数达到 0 的时候,该对象就会被析构.
#大多数时候,我们不需要显式的用 del 删除一个对象,一出它的作用域它就会被析构,Python编程里面用到显式删除元组的情况非常之少.
#del aTuple


#**************** Part 2: 元组操作符和内建函数 *****************

#标准类型操作符,序列类型操作符和内建函数.
#元组的对象和序列类型操作符还有内建函数跟列表的完全一样.你仍然可以对元组进行切
#片操作,合并操作,以及多次拷贝一个元组,还可以检查一个对象是否属于一个元组,进行元组之间的比较等.

#创建,重复,连接操作
#t = (['xyz', 123], 23, -103.4)
#print(t)
#print(t * 2)
#t = t + ('free', 'easy')
#print(t)
#
#-->
#(['xyz', 123], 23, -103.4)
#(['xyz', 123], 23, -103.4, ['xyz', 123], 23, -103.4)
```

```
#(['xyz', 123], 23, -103.4, 'free', 'easy')

#成员关系操作,切片操作
#t = (['xyz', 123], 23, -103.4)
#print(23 in t)
#print(123 in t)
#print(t[0][1])
#print(t[1:])
#-->
#True
#False
#123
#(23, -103.4)

#内建函数
#t = (['xyz', 123], 23, -103.4)
#print(str(t))
#print(len(t))
#print(list(t))
#-->
#(['xyz', 123], 23, -103.4)
#3
#[['xyz', 123], 23, -103.4]

#下面几种语法在 Python 2.x 里支持
#print(max(t))
#print(min(t))
#print(cmp(t, (['xyz', 123], 23, -103.4, 'free', 'easy')))


#操作符
#print((4, 2) < (3, 5))
#print((2, 4) < (3, -1))
#print((2, 4) == (3, -1))
#print((2, 4) == (2, 4))
#-->
#False
#True
#False
#True

#元组类型操作符和内建函数,内建方法
#像列表一样,元组也没有它自己专用的运算符和内建函数.列表方法都跟列表对象的可变性有关,比如说排序,替换,添加等等,
```

#因为元组是不可变的,所以这些操作对元组来说就是多余的,这些方法没有被实现.

##********** Part 3: 元组的特殊特性 **********

#不可变性给元组带来的影响
#在三个标准不可变类型里面--数字，字符串和元组字符串--元组是受到影响最大的,一个数
#据类型是不可变的,简单来讲,就意味着一旦一个对象被定义了,它的值就不能再被更新,除非重
#新创建一个新的对象.对数字和字符串的影响不是很大,因为它们是标量类型,当它们代表的值
#改变时，这种结果是有意义的，是按照你所想要的方式进行访问的，而对于元组，事情就不是
#这样了。
#因为元组是容器对象，很多时候你想改变的只是这个容器中的一个或者多个元素，不幸的
#是这是不可能的，切片操作符不能用作左值进行赋值。这和字符串没什么不同，切片操作只能
#用于只读的操作。
#不可变并不是坏事，比如我们把数据传给一个不了解的 API 时，可以确保我们的数据不会
#被修改。同样地，如果我们操作从一个函数返回的元组，可以通过内建 list()函数把它转换成一个列表.

#元组也不是那么“不可变”
#虽然元组是被定义成不可变的，但这并不影响它的灵活性。比如说，既然我们可以把字符串组合在一起形成一个大字符串。
#重复操作，用一个 list 函数调用把一个元组变成一个可变的列表。

#默认集合类型
#所有的多对象的，逗号分隔的，没有明确用符号定义的，比如说像用方括号表示列表和用
#圆括号表示元组一样，等等这些集合默认的类型都是元组。
#>>> 'abc', -4.24e93, 18+6.6j, 'xyz'
#('abc', -4.24e+093, (18+6.6j), 'xyz')
#>>>
#>>> x, y = 1, 2
#>>> x, y
#(1, 2)

#所有函数返回的多对象（不包括有符号封装的）都是元组类型。注意，有符号封装的多对象集合其实是返回的一个单一的容器对象，比如：
#def foo1():
#...
#return obj1, obj2, obj3

```
#
#def foo2():
#...
#return [obj1, obj2, obj3]
#
#def foo3():
#...
#return (obj1, obj2, obj3)


#单元素元组

#print(type(('xyz')))
#-->
#<class 'str'>
#注意这里返回的是 string,而不是 tuple,即不存在单元素元祖，除非在第一个元素后加逗号.

#print(type(('xyz',)))
#-->
#<class 'tuple'>


#字典的关键字
#不可变对象的值是不可改变的。这就意味着它们通过 hash 算法得到的值总是一个值。这是作为字典键值的一个必备条件。

#核心笔记：列表 VS 元组
#最好使用不可变类型变量的一个情况是，如果你在维护一些敏感的数据，并且需要把这些
#数据传递给一个并不了解的函数(或许是一个根本不是你写的 API),作为一个只负责一个软件
#某一部分的工程师，如果你确信你的数据不会被调用的函数篡改，你会觉得安全了许多。
#一个需要可变类型参数的例子是，如果你在管理动态数据集合时。你需要先把它们创建出
#来，逐渐地或者不定期的添加它们，或者有时还要移除一些单个的元素。这是一个必须使用可
#变类型对象的典型例子。幸运的是，通过内建的 list()和 tuple()转换函数,你可以非常轻松
#的在两者之间进行转换.
#list()和 tuple()函数允许你用一个列表来创建一个元组,反之亦然.如果你有一个元组变
#量,但你需要一个列表变量因为你要更新一下它的对象,这时 list()函数就是你最好的帮手.如
#果你有一个列表变量,并且想把它传递给一个函数,或许一个 API,而你又不想让任何人弄乱你
#的数据,这时 tuple()函数就非常有用。


#********** Part 4: 拷贝 Python 对象: 浅拷贝和深拷贝 **********
```

#对象赋值实际上是简单的对象引用。也就是说当你创建一个对象，然后把它赋给另一个变量的时候，
#Python 并没有拷贝这个对象，而是拷贝了这个对象的引用。

#对一个对象进行浅拷贝其实是新创建了一个类型跟原对象一样,其内容是原来对象元素的引用,换句话说,
#这个拷贝的对象本身是新的,但是它的内容不是.序列类型对象的浅拷贝是默认类型拷贝,
#并可以以下几种方式实施:(1)完全切片操作[:],(2)利用工厂函数,比如 list(),dict()等,(3)使用 copy 模块的 copy 函数.

#完全拷贝或者说深拷贝--创建一个新的容器对象,包含原有对象元素（引用）全新拷贝的引用--需要 copy.deepcopy()函数.
#x=5
#y=x
#print(x,y)
#print(id(x),id(y))
#-->
#5 5
#506090144 506090144

#x=6
#print(x,y)
#print(id(x),id(y))
#-->
#6 5
#506090176 506090144
#这里我们看到 x,y 的值不一样了。 y 是对 x 的引用，当我们修改 x 的值，会重新生成一个对象。 但是 y 的引用还是 x 的旧值。 所以此时的 x,y 不一样。

#以上就是浅拷贝

#import copy
#x=5
#y=copy.deepcopy(x)
#print(x,y)
#print(id(x),id(y))

#核心模块: copy
#copy 模块中只有两个函数可用:copy()进行浅拷贝操作,而 deepcopy()进行深拷贝操作.

#以下有几点关于拷贝操作的警告。
#第一,非容器类型(比如数字,字符串和其他"原子"类型的对象,像代码,类型和 xrange 对象等)

没有被拷贝一说,
#浅拷贝是用完全切片操作来完成的.第二,如果元组变量只包含原子类型对象,对它的深拷贝将不会进行.