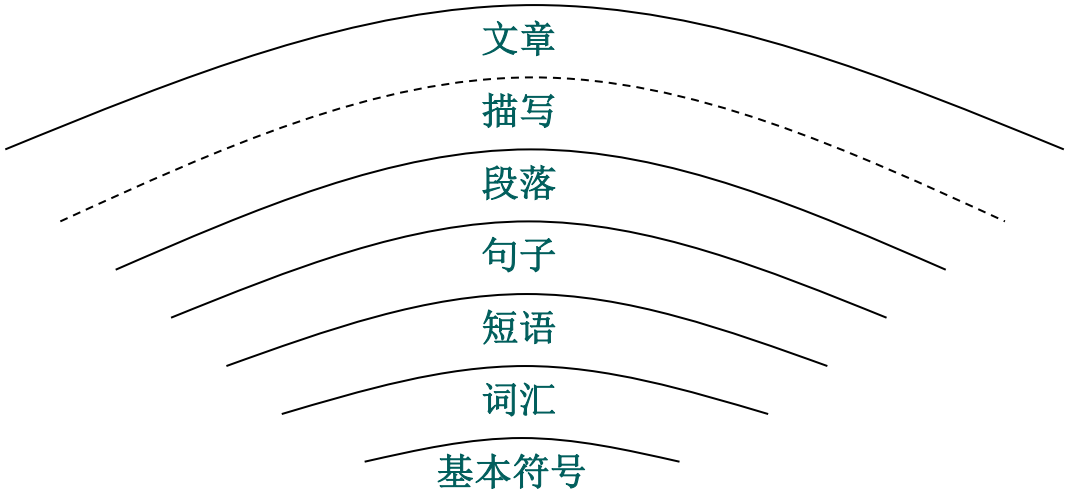
机器学习程序设计基础
Python基础操作

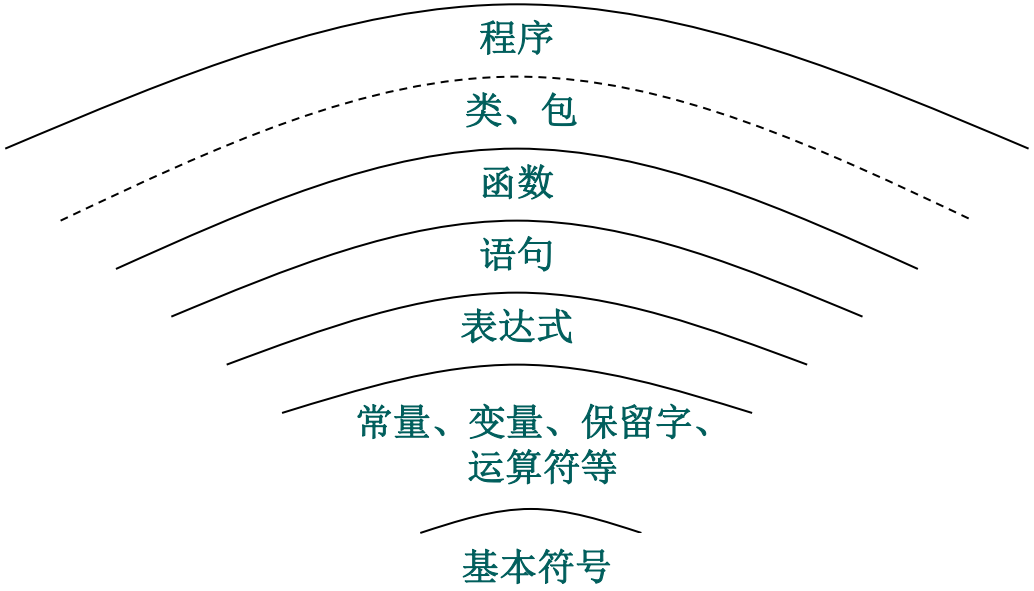
陈湘萍

中山大學
SUN YAT-SEN UNIVERSITY

程序设计语言



语言的基本体系结构



程序设计语言的基本体系结构

常量、变量和对象

- 常量
  - 常量指在程序的执行过程中不变的量
- 字面量
  - 1,2,3,4;　　1.1,　1.2,　　1.0
  - 'abc', "Python"
  - True , False
- 符号常量

  pi　#显示3.141592653589793

  e　#显示2.718281828459045

常量、变量和对象

- 变量
  - 变量在程序中就是用一个变量名表示，变量名必须是大小写英文、数字和_的组合，且不能用数字开头，比如：
  - 在Python中，等号=是赋值语句，可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量，例如：
  - a = 123　　# a是整数
  - a = 'ABC'　　# a变为字符串

常量、变量和对象

- 变量的取名不能跟保留字同名

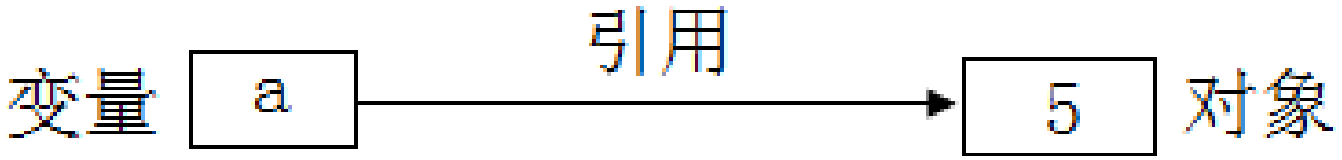
and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

常量、变量和对象

- 程序中的常量
  - 在Python中，通常用全部大写的变量名表示常量
    - PI = 3.14159265359
  - 但事实上PI仍然是一个变量，Python根本没有任何机制保证PI不会被改变，
  - 用全部大写的变量名表示常量只是一个习惯上的用法，实际上是可以改变变量PI的值，是一个变量

常量、变量和对象

- 变量
  - 指向对象的值的名称就是变量
  - 变量是一个标识符，通过等号（=）赋值运算创建



    - 多个变量可以引用同一个对象，
    - 一个变量也可以引用不同的对象。
    - 引用不同的对象时，id也就不同的

常量、变量和对象

- 对象
  - 一切皆对象，1,2,3,a,b,c,　sin(x)
  - 对象是某个类型事物的一个具体的实例
  - 每一个对象都有一个唯一的身份标识
  - 对象的id号

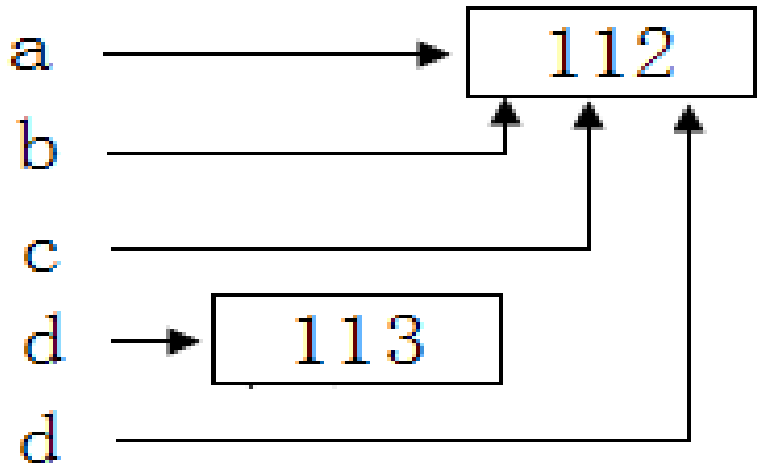
    id(11)
  - 对象的类型

    type(12)

常量、变量和对象

- id相同，就是相同的对象

```
>>> a=112
>>> b=112
>>> c=b
>>> d=113
>>> d=112
```



- Python中的变量
- 不需要声明
- 可以随时赋不同类型的值

对象和数据的不同

- 数据纯粹是数据，
- 对象包含属性和功能
- 例子
  - 数据4、3
  - 直角边分别是4和3的直角三角形
    - 直角
    - 边长4、3
    - 周长
    - 面积
    - 位置
    - 第三边长度

对象具有属性和方法，不同类型的对象具有的属性和方法是不同的

对象的方法

<变量名>.<方法>(<参数>)

实例操作

数据类型

数据类型

程序 = 算法 + 数据结构

- 数据是程序的一个重要组成部分，在程序中首先需要对要处理的数据进行描述。
- 数据的描述是通过数据类型来实现的，每个数据都属于一种数据类型。
- 在程序中区分数据类型的好处是便于实现对数据的可靠、有效处理。

数据类型

- 一种数据类型由两个集合构成：
  - 值集：规定了该数据类型能包含哪些值（包括这些值的结构）。
  - 操作（运算）集：规定了对值集中的值能实施哪些运算。
  - 例如整数类型就是一种数据类型
    - 值集：由整数构成
    - 操作集：加、减、乘、除等运算。

静态类型语言和动态类型语言

- 静态类型语言（statically typed languages）
  - 在静态的程序（运行前的程序）中必须为每个数据指定类型。
  - 程序通常采用编译方式执行。
- 动态类型语言（dynamically typed languages）
  - 在程序运行中数据被用到时才确定它们的类型。
  - 程序通常采用解释方式执行。

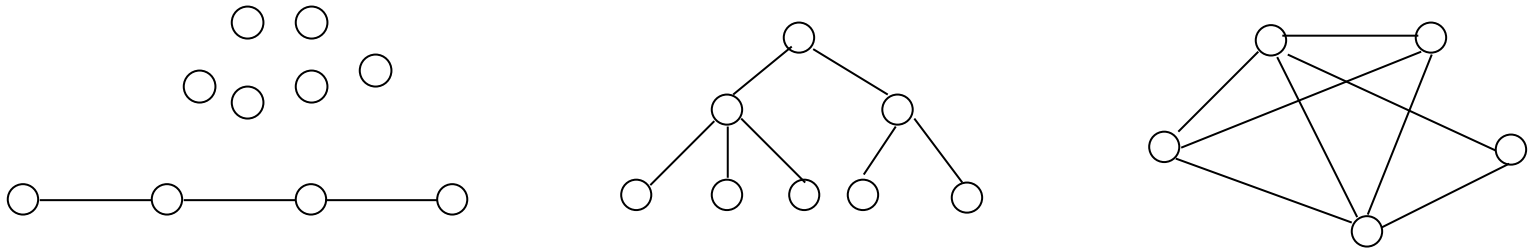
静态类型语言和动态类型语言

- 静态类型语言的好处
  - 提高程序的可靠性，便于编译程序自动进行类型一致性检查。
  - 便于产生高效的可执行代码。
- 例如，对于“x+y”，如果在程序运行前不知道x和y的类型，则
  - 无法知道它是否合法
  - 无法生成指令
- C/C++是一种静态类型语言
- Python是一种动态类型语言

数据类型

- 数据类型一般可以分为
  - 简单数据类型：值集中的数据是不可再分解的简单数据，如：整数类型、实数类型等；
  - 复合数据类型：值集中的数据是由其它类型的数据按照一定的方式组织而成，如：表、向量、矩阵、学生信息等。

数据结构



- 数据结构(Data Structure)：是指相互之间具有(存在)一定联系(关系)的数据元素的集合。元素之间的相互联系(关系)称为逻辑结构。
- 数据元素之间的逻辑结构有四种基本类型
  - ① 集合：结构中的数据元素除了“同属于一个集合”外，没有其它关系
  - ② 线性结构：结构中的数据元素之间存在一对一的关系
  - ③ 树型结构：结构中的数据元素之间存在一对多的关系
  - ④ 图状结构或网状结构：结构中的数据元素之间存在多对多的关系

Python数据类型

- 基本数据类型
  - 语言预先定义好的数据类型，常常又称为标准数据类型或内置数据类型（built-in types）。
  - 它们都是简单类型。
- 构造数据类型
  - 利用语言提供的类型构造机制从其它类型构造出来的数据类型。
  - 它们大多为复合数据类型。
- 抽象数据类型
  - 利用数据抽象机制把数据的表示对使用者隐藏起来的数据类型。
  - 它们一般为复合数据类型。

基本数据类型

- 数值型
- 字符类型
- 逻辑类型
- 空值类型

数值型

- 数值类型：用于存储数值。
  - 整型(int)：通常被称为是整型或整数，是正或负整数，不带小数点
  - 长整型(long)：无限大小的整数，整数最后是一个大写或小写的L。在Python3里，只有一种整数类型 int，没有Python2 中的 Long
  - 浮点型(float)：浮点型由整数部分与小数部分组成，浮点型也可以使用科学计数法表示（2.78e2就是$2.78 \times 10^2 = 278$）
  - 复数(complex)：复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示，复数的虚部以字母j或J结尾。如：2+3j

数值型

- 数字类型（数值类型）
  - 整数类型
  - 开头是0X或0x，被认为是十六进制的常量
    - 如0x22,表示的数是十进制的
  - 开头是0O或0o，则被认为是八进制的常量
    - 如0O22,表示的数是十进制的
  - 开头是0B或0b，则被认为是二进制的常量
    - 如0B1101,表示的数是十进制的

实例操作

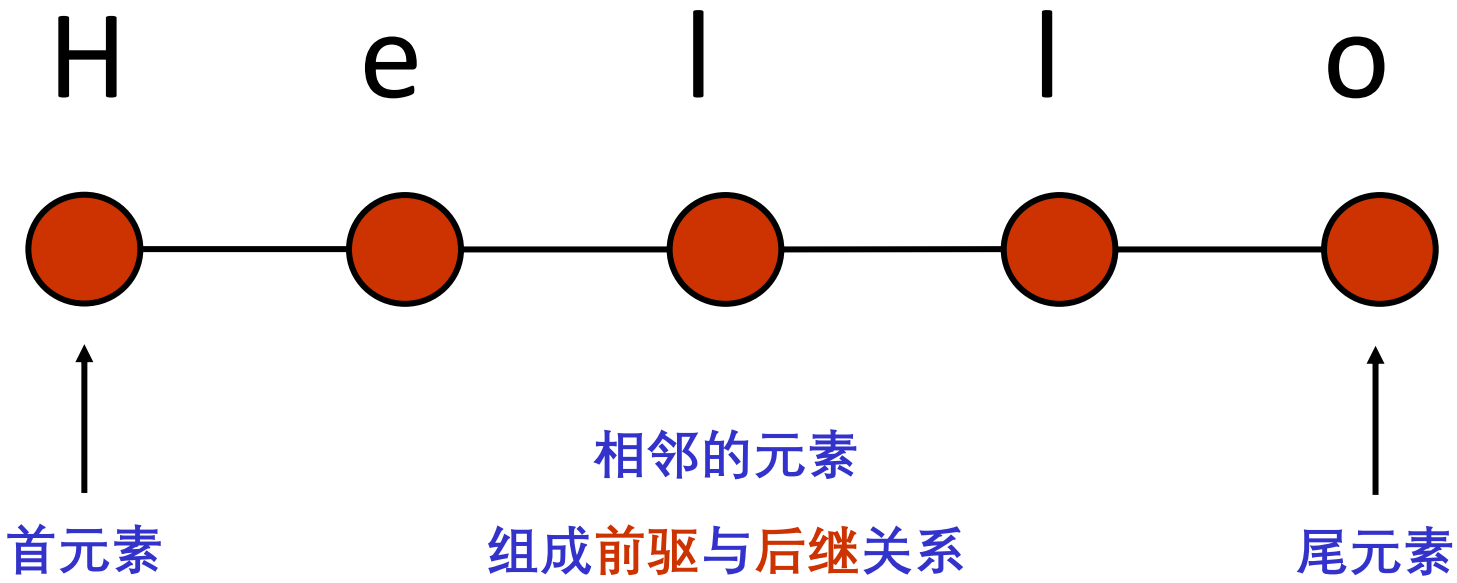
基本数据类型

- 数值型
- 字符类型
- 逻辑类型
- 空值类型

字符串

- Python使用单引号和双引号来表示字符串是一样的。
  - 创建和访问字符串
  - var1 = 'Hello World!'
  - var2 = “Python Programming”
  - Var3= ‘‘‘ Python
  - Programming’’’

线性表



线性表的逻辑结构

字符串

- Python转义字符:需要在字符中使用特殊字符时，用反斜杠(\)

转义字符	描述	转义字符	描述
\(在行尾时)	续行符	\n	换行
\\	反斜杠符号	\v	纵向制表符
\'	单引号	\t	横向制表符
\"	双引号	\r	回车
\a	响铃	\f	换页
\b	退格(Backspace)	\e	转义
\oyy	八进制数，yy 代表的字符，例如：\o12 代表换行	\000	空
\xyy	十六进制数，yy 代表的字符，例如：\x0a 代表换行		

字符串基本操作

- +字符串合并
- *字符串重复

```
>>> len('abc')
3
>>> 'abc'+'def'
'abcdef'
>>> 'abc' 'def'
'abcdef'
>>> 'hello'*4
'hellohellohellohello‘
>>> 'abc'+9
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

字符串索引和分片

- 字符串是字符的有序集合，能够通过其位置来获得他们的元素
- Python中字符串中的字符是通过索引提取的
- 索引从0开始，但不同于C语言的是可以取负值，表示从末尾提取，最后一个是-1，前一个是-2，依次类推，认为是从结束处反向计数

```
>>> s = 'spam'
>>> s[0]
's'
>>> s[1]
'p'
>>> s[-1]
'm'
>>> s[-2]
'a'
```

字符串索引和分片

- 分片：从字符串中分离提取了一部分内容（子字符串）；可以用于提取部分数据，分离出前、后缀等场合。
- 当使用一对以冒号分隔的偏移索引字符串这样的序列对象时，就返回一个新的对象，其中包含了以这对偏移所标识的连续的内容。
- 左边的偏移被取作是下边界（包含下边界在内），而右边的偏移被认为是上边界（不包括上边界在内）。
- 如果被省略上下边界的默认值分别对应为0和分片对象的长度。

```
>>> s = 'spam'
>>> s[1:3]
'pa'
>>> s[1:]
'pam'
>>> s[:-1]
'sp'
>>> s[:]
'spam'
```

有错吗?

修改字符串

- 缺省情况下，字符串对象是“不可变序列”，不可变的意思是不能实地的修改一个字符串。

```
>>> s = 'spam'
>>> s[0] = 'x'
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

- 那如何改变一个字符串呢？这就要利用合并、分片这样的工具来建立并赋值给一个新的字符串；必要的话，可以将结果赋值给字符串最初的变量名。

```
>>> s = 'spam'
>>> s = s + 'SPAM'
>>> s
'spamSPAM'
>>> s = s[:4] + 'OK!' + s[-1]
>>> s
'spamOK!M'
```

字符串转化

- Python不允许字符串和数字直接相加。

```
>>> "15" + 1
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

- 这是有意设计的，因为+既能够进行加法运算也能够进行合并运算，这样的语法会变得模棱两可，因此，Python将其作为错误处理，在Python中，如果让操作变得复杂或含糊，就会避免这样的语法。

字符串

Python字符串运算符

a = 'Hello'　　b = 'Python'

操作符	描述	实例
+	字符串连接	a + b 输出结果：HelloPython
*	重复输出字符串	a*2 输出结果：HelloHello
[]	通过索引获取字符串中字符	a[1] 输出结果 e
[:]	截取字符串中的一部分	a[1:4] 输出结果 ell
in	成员运算符，如果字符串中包含给定的字符返回 True	'H' in a 输出结果 True
not in	成员运算符，如果字符串中不包含给定的字符返回 True	'M' not in a 输出结果 True
r 或 R	原始字符串，原始字符串：所有的字符串都是直接按照字面的意	print(r'\n prints \n') 和

布尔类型

- Python支持布尔类型的数据，布尔类型只有True和False两种值
  - and与运算：只有两个布尔值都为 True 时，计算结果才为True
  - or或运算：只要有一个布尔值为 True，计算结果就是 True
  - not非运算：把True变为False，或者把False变为True
- 在Python中，布尔类型还可以与其他数据类型做and、or和not运算

布尔类型

- 下面的几种情况会被认为是FALSE
  - 为0的数字，包括0,0.0
  - 空字符串' '，""
  - 表示空值的None
  - 空集合，包括空元祖()，空序列[]，空字典{}
- 其他的值都为TRUE
  - a = 'python'
  - print (a and True)　# 结果是 True
  - b = ''
  - print (b or False)　# 结果是 False

布尔类型

- 在Python中，逻辑值True和False作为数值，则分别是整型值1和0参与运算。例如：
- >>> x=False
- >>>a=x+(5>4)　#结果a是1
- >>> b=x+5　#结果b是5

空值

- 空值是Python里一个特殊的值，用None表示。
- 不支持任何运算也没有任何内置函数方法
- None和任何其他的数据类型比较永远返回False
- 在Python 中未指定返回值的函数会自动返回None。

实例操作

Python的数据结构

- 序列对象
  - 字符串 str
  - List 列表 list
  - Tuple 元组 tuple
- 字典类型 dict
- 集合对象set

序列数据结构

- 数据结构是计算机存储、组织数据的方式
- 序列是Python中最基本的数据结构
- Python内置序列类型最常见的是列表、元组、字符串
- Python提供了字典和集合这样的数据结构，它们属于无顺序的数据集合体，不能通过位置索引号来访问数据元素

序列对象

- 字符串、列表和元组的对象类型均属于称为序列的Python对象,一种可使用数字化索引进行访问其中元素的对象 。
- 可用算术运算符联接或重复序列。 (+/*)
- 比较运算符(<，<=，>，>=，!=，==)也可用于序列 。
  - a=“123” b = “456” a和b的比较关系

序列对象

- 可通过下标，切片和解包来访问序列的某部份。
  - a=“123456”
  - print a[1],a[3:],a[:3],a[2:4]
- in运算符可判断当有对象是否序列对象成员
  - ‘1’ in a
- 可通过循环运算符对序列对象进行迭代操作。
  - for x in range(1,10):

索引和分片

- 索引（s[i]）获取特定偏移的元素
  - 第一个元素的偏移为0
  - 负偏移索引意味着从最后或右边反向进行计数
  - s[0]获取第一个元素，s[-2]获取倒数第二个元素
- 分片（s[i:j]）提取对应的部分作为一个序列
  - 上边界并不包含在内
  - 分片的边界默认为0和序列的长度，如果没有给出的话
  - s[1:3]获取从偏移为1开始，直到但不包含偏移为3的元素
  - s[1:]获取了从偏移为1直到末尾之间的元素
  - s[:3]获取从偏移为0直到但不包含偏移为3的元素
  - s[:-1]获取从偏移为0直到但不包含最后一个元素之间的元素
  - s[:]获取从偏移为0直到末尾之间的所有元素

列表的主要性质

- 任意对象的有序集合：从功能是看，列表就是收集其他对象的地方，可以他们看成数组；同时，列表所包含的每一项都保持了从左到右的位置顺序（也就是说，它们是序列）。
- 通过偏移读取：和字符串一样，可以通过列表对象的偏移对其进行索引，从而读取对象的一部分内容。当然也可以执行诸如分片和合并之类的操作
- 可变长度、异构以及任意嵌套：和字符串不同，列表可以根据需要增长或缩短（长度可变），并且可以包含任何类型的对象，并支持任意的嵌套。
- 可变序列：列表支持在原处的修改，也可以响应所有针对字符串序列的操作，如索引、分片以及合并。
  - 序列操作在列表与字符串中工作方式相同
  - 区别是：当合并或分片应用于列表时，返回新的列表而不是新的字符串，并且支持某些字符串不支持的操作。

常用列表常量和操作

操作	解释
L1=[]	一个空的列表
L2 = [0, 1, 2, 3]	四元素列表
L3 = ['abc',10,['def', 'ghi']	嵌套列表
L2[i]	索引
L3[i][j]	索引的索引
L2[i:j]	分片
len(L2)	求长度
L1 + L2	合并
L2 * 3	重复

列表的方法

- append(x)

  把一个元素添加到列表的结尾，相当于a[len(a):] = [x]

- extend(L)

  通过添加指定列表的所有元素来扩充列表，相当于a[len(a):]=L

- insert(i,x)

  在指定位置插入一个元素。第一个参数是准备插入到其前面的那个元素的索引，例如a.insert(0,x)会插入到整个链表之前，而a.insert(len(a), x)相当于a.append(x)。

列表的方法

- remove(x)

  删除链表中值为x的第一个元素。如果没有这样的元素，就会返回一个错误。

- pop([i])

  从链表的指定位置删除元素，并将其返回。如果没有指定索引，a.pop()返回最后一个元素。元素随即从链表中被删除。（方法中i两边的方括号表示这个参数是可选的，而不是要求输入一对方括号，会经常在Python库参考手册中遇到这样的标记。）

- del x[]
  - 可以删除某个索引的元素或切片元素

列表的方法

- index(x)

  返回链表中第一个值为x的元素的索引。如果没有匹配的元素就会返回一个错误。

- count(x)

  返回x在链表中出现的次数。

- sort()

  对链表中的元素进行适当的排序。

- reverse()

  倒排链表中的元素。

List对象的操作

方法	描述
append(x)	在列表尾部追加单个对象x。使用多个参数会引起异常。
count(x)	返回对象x在列表中出现的次数。
extend(L)	将列表L中的表项添加到列表中。返回None。
Index(x)	返回列表中匹配对象x的第一个列表项的索引。无匹配元素时产生异常。
insert(i,x)	在索引为i的元素前插入对象x。如list.insert(0,x)在第一项前插入对象。返回None。
pop(x)	删除列表中索引为x的表项，并返回该表项的值。若未指定索引，pop返回列表最后一项。
sort()	对列表排序，返回none。bisect模块可用于排序列表项的添加和删除。
remove(x)	删除列表中匹配对象x的第一个元素。匹配元素时产生异常。返回None。
reverse()	颠倒列表元素的顺序。

```
help(list)
help(list.count)
```

实例操作

元组简介

- 一个元组由数个逗号分隔的值组成

```
>>> t = 12345, 54321, 'hello'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello')
>>> u = t, (1, 2, 3)
>>> u
((12345, 54321, 'hello'), (1, 2, 3))
```

- 元组在输出时总是有括号的，以便于正确表达嵌套结构。
- 在输入时，有或没有括号都可以，不过经常括号都是必须的（如果元组是一个更大的表达式的一部分）。

元组

- 元组有很多用途。例如(x, y)坐标点，数据库中的员工记录等等。
- 元组就像字符串，不可改变：不能给元组的一个独立的元素赋值（尽管可以通过联接和切片来模仿）
- 可以通过包含可变对象来创建元组，例如链表。

```
>>> lst = [1, 2, 3]
>>> t = tuple(lst)
>>> t
(1, 2, 3)
```

元组

- 一个特殊的问题是构造包含零个或一个元素的元组
  - 一对空的括号可以创建空元组
  - 要创建一个单元素元组可以在值后面跟一个逗号（在括号中放入一个单值是不够的）

```
>>> emp = ()
>>> emp
()
>>> single = 'a',  #<-- note trailing comma
>>> len(emp)
0
>>> len(single)
1
>>> single
('a',)
```

元组封装和解封

- 语句t = 12345, 54321, 'hello!' 是元组封装（sequence packing）的一个例子：值12345，54321 和'hello!' 被封装进元组。其逆操作可能是这样：

```
>>> t = (1, 2, 3)
>>> x, y, z = t
>>> print x, y, z
1 2 3
```

- 这个调用被称为序列拆封非常合适。序列拆封要求左侧的变量数目与序列的元素个数相同。

元组封装和解封

- 拆封和封装一点不对称：封装多重参数通常会创建一个元组，而拆封操作可以作用于任何序列。

```
>>> t = [1, 2, 3]
>>> x, y, z = t
>>> print x, y, z
1 2 3

>>> s = "123"
>>> x,y,z = s
>>> print x, y, z
1 2 3
```

Dictionary字典

- 字典在某些语言中可能称为“联合内存”（“associative memories”）或“联合数组”（“associative arrays”）。
- 字典类似于通过联系人名字查找地址和联系人详细情况的地址簿，即：我们把**键**（名字）和**值**（详细情况）联系在一起。
- 注意，键必须是唯一的，就像如果有两个人恰巧同名的话，将无法找到正确的信息。

字典

- 序列是以连续的整数为索引，与此不同的是，字典以关键字为索引
- 关键字可以是任意不可变类型，通常用字符串或数值。如果元组中只包含字符串和数字，它可以做为关键字，如果它直接或间接的包含了可变对象，就不能当做关键字
- 不能用列表做关键字，因为链表可以用它们的append()和extend()方法，或者用切片、或者通过检索变量来即时改变
- 基本说来，应该只使用简单的对象作为键

字典

- 理解字典的最佳方式是把它看做无序的(关键字:值)对集合，关键字必须是互不相同的（在同一个字典之内）
- 一对大括号创建一个空的字典：{}
- 字典的主要操作是依据关键字来存储和析取值。也可以用del来删除(关键字:值)对
- 如果使用一个已经存在的关键字存储新的值或对象，以前为该关键字分配的值就会被遗忘。
- 试图析取从一个不存在的关键字中读取值会导致错误。

字典

- 字典的keys()方法返回由所有关键字组成的列表，该列表的顺序不定（如果需要它有序，只能调用返回列表的sort()方法）
- 使用字典的has_key()方法可以检查字典中是否存在某一关键字
- 字典的values()方法返回字典内所有的值
- 字典的get()方法可以根据关键字返回值，如果不存在输入的关键字，返回None

字典

- 字典的update(*anothordict*)方法类似于合并，它把一个字典的关键字和值合并到另一个，盲目的覆盖相同键的值

```
>>> tel
{'gree': 4127, 'irv': 4127, 'jack': 4098}
>>> tel1 = {'gree':5127, 'pang':6008}
>>> tel.update(tel1)
>>> tel
{'gree': 5127, 'irv': 4127, 'jack': 4098, 'pang': 6008}
```

字典

- 字典的pop()方法能够从字典中删除一个关键字并返回它的值，类似于列表的pop方法，只不过删除的是一个关键字而不是位置。

```
>>> tel
{'gree': 5127, 'irv': 4127, 'jack': 4098, 'pang': 6008}
>>> tel.pop('gree')
5127
>>> tel
{'irv': 4127, 'jack': 4098, 'pang': 6008}
>>> tel.pop('li')
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
KeyError: 'li'
```

dictionary对象的操作

方法	描述
has_key(x)	如果字典中有键x，则返回真。
keys()	返回字典中键的列表
values()	返回字典中值的列表。
items()	返回tuples的列表。每个tuple由字典的键和相应值组成。
clear()	删除字典的所有条目。
copy()	返回字典高层结构的一个拷贝，但不复制嵌入结构，而只复制对那些结构的引用。
update(x)	用字典x中的键值对更新字典内容。
get(x[,y])	返回键x，若未找到该键返回none，若提供y，则未找到x时返回y。

help(dict)

实例操作

集合

- 集合（set）是一个无序不重复元素的序列。集合基本功能是进行成员关系测试和删除重复元素。
- 可以使用**大括号({})或者 set()函数**创建集合
- 注意：创建一个空集合必须用 set() 而不是 { }，因为 { } 是用来创建一个空字典。
- student = {'Tom', 'Jim', 'Mary', 'Tom', 'Jack', 'Rose'}
- print(student) # 输出集合，重复的元素被自动去掉

集合运算

- 可以使用“-”、“|”、“&”运算符进行集合的差集、并集、交集运算。
- # set可以进行集合运算
- a = set('abcd')　　#a= {'a', 'b', 'c', 'd' }
- b = set('cdef')
- print(a)
- print("a和b的差集：", a - b)　# a和b的差集{'a', 'b' }
- print("a和b的并集：", a | b)　# a和b的并集 {'a', 'b', 'c', 'd' , 'e' }
- print("a和b的交集：", a & b)　# a和b的交集{'c', 'd' }
- print("a和b中不同时存在的元素：", a ^ b)　# a和b中不同时存在的元素