# 数据结构与算法

## 1.1解压赋值给多个变量

任何的序列（或者是可迭代对象）可以通过一个简单的赋值语句解压并赋值给多个变量。唯一的前提就是变量的数量必须跟序列元素的数量是一样的。

|  |
| --- |
| 例1：  name,age,sex,(year,mom)=[‘liusx’,23,’man’,(2021,12)]  例2：  a, b, c, d, e = ‘hello’  例3：  \_, shares, price, \_ = [ 'ACME', 50, 91.1, (2012, 12, 21) ] #只解压部分 |

## 1.2星号在解压赋值的妙用

|  |
| --- |
| 例1：解压赋值给N个变量，b变量永远为列表  a,\*b,c = list(range(10))  例2：可变长度数据赋值，可用for循环  records = [('foo', 1, 2),('bar', 'hello'),('foo', 3, 4)]  (a,\*b),\*c = records  for tag, \*args in records:  if tag == 'foo':  print(args)  例3：  line = 'nobody:\*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false'  uname, \*fields, homedir, filename = line.split(':')  例4：丢弃赋值元素  name, \*\_, (\*\_, year) = ('ACME', 50, 123.45, (12, 18, 2012))  例5：递归求和  def sum(items):  head, \*tail = items  return head + sum(tail) if tail else head **#当tail为[]时返回head**  sum([1,2,3,4,5]) |

## 1.3保留最后N个元素

|  |
| --- |
| 在多行上面做简单的文本匹配，并返回匹配所在行的最后N 行：  from collections import deque  def search(lines, pattern, history=5):  previous\_lines = deque(maxlen=history) #定义最长为5的序列  for line in lines: #循环导入的文件内容  if pattern in line: #匹配到搜索的参数  yield line, previous\_lines #返回之前的行的列表  previous\_lines.append(line) #此行添加到储存之前行列表  if \_\_name\_\_ == '\_\_main\_\_':  with open(r'../../cookbook/somefile.txt') as f:  for line, prevlines in search(f, 'python', 5): #循环匹配到的数据并输打印  for pline in prevlines:  print(pline, end='') |

## 1.4查找最大或者最小的N个元素

|  |
| --- |
| import heapq  例1：  nums = [1, 8, 2, 23, 7, -4, 18, 23, 42, 37, 2]  print(heapq.nlargest(3, nums)) # Prints [42, 37, 23]  print(heapq.nsmallest(3, nums)) # Prints [-4, 1, 2]  例2：  portfolio = [  {'name': 'IBM', 'shares': 100, 'price': 91.1},  {'name': 'AAPL', 'shares': 50, 'price': 543.22},  {'name': 'FB', 'shares': 200, 'price': 21.09},  {'name': 'HPQ', 'shares': 35, 'price': 31.75},  {'name': 'YHOO', 'shares': 45, 'price': 16.35},  {'name': 'ACME', 'shares': 75, 'price': 115.65}  ]  cheap = heapq.nsmallest(3, portfolio, key=lambda s: s['price']) # 最小  expensive = heapq.nlargest(3, portfolio, key=lambda s: s['price']) # 最大 |

## 1.5实现一个优先级队列

|  |
| --- |
| import heapq  class PriorityQueue:  def \_\_init\_\_(self):  self.\_queue = []  self.\_index = 0  def push(self, item, priority):  heapq.heappush(self.\_queue, (priority, self.\_index, item))  self.\_index += 1  print(self.\_queue)  def pop(self):  return heapq.heappop(self.\_queue)[-1]  q = PriorityQueue()  q.push('foo', 1)  q.push('bar', 5)  q.push('spam', 6)  q.push('grok', 1)  print(q.pop()) # (1, 0, 'foo')  print(q.pop()) # (1, 3, 'grok')  print(q.pop()) # (5, 1, 'bar') |

## 1.6字典中的键映射多个值

|  |
| --- |
| from collections import defaultdict  d = defaultdict(list)  d['a'].append(1)  d['a'].append(2)  d['b'].append(4)  d = defaultdict(set)  d['a'].add(1)  d['a'].add(2)  d['b'].add(4) |

## 1.7有顺序的字典

|  |
| --- |
| from collections import OrderedDict  # OrderedDict 内部维护着一个根据键插入顺序排序的双向链表，内存大2倍  d = OrderedDict()  d['foo'] = 1  d['bar'] = 2  d['spam'] = 3  d['grok'] = 4  # Outputs "foo 1", "bar 2", "spam 3", "grok 4"  for key in d:  print(key, d[key]) |