# 数据存储

## 课程目标

* 能够创建带有3个数据的列表对象
* 能够使用append方法完成向列表中添加数据的操作
* 能够通过下标来获取列表中指定位置的数据
* 能够创建带有3个数据的元组对象
* 能够通过下标获取元组中指定位置的数据
* 能够创建带有3个数据的集合对象
* 能够创建带有3组数据的字典对象
* 能够向字典中添加、修改并获取数据
* 能够完成斗地主发牌案例

## 课程内容

* 列表
* 元组
* 集合
* 字典
* 公共函数
* 切片
* 推导式

### 列表

#### 1.1 列表的概念

列表是一种存储大量数据的存储模型。

#### 1.2 列表的特点

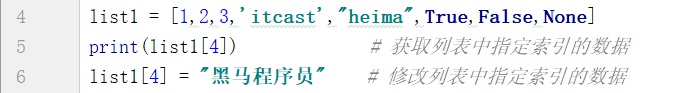
列表具有索引的概念，可以通过索引操作列表中的数据。列表中的数据可以进行添加、删除、修改、查询等操作。

#### 1.3 列表的基本语法

创建列表： 变量名 = **[**数据1，数据2，……]

获取列表数据： 变量名[索引]

修改列表数据： 变量名[索引] = 值



#### 1.4列表常用方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法名 | 功能 | 参数 | 返回值 | 范例 |
| append(data)  关键词：追加 | 在列表的末尾添加数据 | data:加入到列表中的数据 | None | list1 = [1,2,3] list1.append(**"itcast"**) |
| insert(idx,data)  关键词：插入 | 在列表的指定位置插入数据，如果索引位置超过列表数据总量，数据将插入到列表末尾 | idx:插入数据的索引位置  data:加入列表中的数据 | None | list1 = [1,2,3] list1.insert(0,**"itcast"**) |
| extend(model)  关键词：追加全部 | 在列表的末尾添加参数对象中的所有数据 | model:保存有数据的存储模型，该模型接受列表、元组、集合类型的对象 | None | list1 = [1,2,3] list2 = [4,5,6] tuple2 = (7,8,9) set2 = {10,11,12} list1.extend(list2) list1.extend(tuple2) list1.extend(set2) |
| remove(data)  关键词：删除 | 从列表中删除指定的数据，如果数据不存在将报错 | data:要从列表中删除的数据 | None | list1 = [1,2,3,**"itcast"**] list1.remove(**"itcast"**) |
| pop(idx)  关键词：获取删除 | 从列表中获取并删除指定索引位置上的数据，如果索引值超过列表数据总量将报错 | idx:要从列表中获取的数据对应的索引位置 | 获取到的数据 | list1 = [1,2,3,**"itcast"**] data = list1.pop(2) |
| clear()  关键词：清空 | 清空列表中的数据 | 无 | None | list1 = [1,2,3,**"itcast"**] list1.clear() |
| index(data)  关键词：查询位置 | 查询列表中指定数据对应的索引，如果数据不存在将报错 | data:在列表中要查询的数据 | 查询数据第一次出现的索引 | list1 = [1,2,3,**"itcast"**] idx = list1.index(**"itcast"**) |
| count(data)  关键词：统计数量 | 统计列表中指定数据出现的数量 | data:在列表中要统计数量的数据 | 统计数据出现的次数 | list1 = [1,2,3,**"itcast"**] num = list1.count(**"itcast"**) |

### 元组

#### 2.1 元组的概念

元组是一种存储固定数据的存储模型。

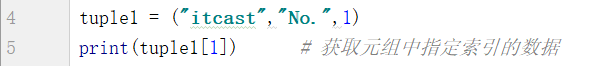
#### 2.2 元组的特点

元组具有索引的概念，可以通过索引操作元组中的数据。元组中的数据可以进行查询操作，但不能进行添加、删除、修改操作。

#### 2.3 元组的基本语法

创建元组： 变量名 = （数据1，数据2，……）

获取元组数据： 变量名[索引]



#### 2.4元组常用方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法名 | 功能 | 参数 | 返回值 | 范例 |
| index(data)  关键词：查询位置 | 查询元组中指定数据对应的索引，如果数据不存在将报错 | data:在元组中要查询的数据 | 查询数据第一次出现的索引 | tuple1 = [1,2,3,**"itcast"**] idx = tuple1.index(**"itcast"**) |
| count(data)  关键词：统计数量 | 统计元组中指定数据出现的数量 | data:在元组中要统计数量的数据 | 统计数据出现的次数 | tuple1 = [1,2,3,**"itcast"**] num = tuple1.count(**"itcast"**) |

#### 2.5注意事项

元组中的数据如果是非引用类型数据，不允许修改

元组中的数据如果是引用类型对象，该对象不允许替换，而对象的属性值可以发生改变

#### 2.6元组的特殊应用

**自动组包**：将多个数据赋值给一个变量时，多个数据将包装成一个元组，将元组赋值给变量，该过程称为自动组包。

应用场景：设置返回值为多个值

return 1,2,3 等同于 return (1,2,3)

应用场景：动态字符串赋值传值方式

print(“坐标%d,%d” % (10,20))

**自动解包**：将元组赋值给多个变量时，如果元组数据的数量与变量的数量相同，元组将被自动拆分成多个值，并赋值给对应变量，该过程称为自动解包。

场景：多个变量接收返回值

a,b,c = func()等同于 a,b,c = (1,2,3)等同于a,b,c = 1,2,3

场景：动态字符串赋值取值方式

print(“坐标%d,%d” % (10,20)) 将10给第一个%d,将20给第二个%d

### 集合

#### 3.1 集合的概念

集合是一种存储大量无序不重复数据的存储模型。

#### 3.2 集合的特点

集合没有索引的概念。集合中的数据可以进行添加、删除等操作。

#### 3.3 集合的基本语法

创建集合： 变量名 = **{**数据1，数据2，……}



#### 3.4集合常用方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法名 | 功能 | 参数 | 返回值 | 范例 |
| add(data)  关键词：添加 | 在集合中添加数据 | data:加入到集合中的数据 | None | set1 = {1,2,3} set1.append(**"itcast"**) |
| remove(data)  关键词：删除 | 从集合中删除指定的数据，如果数据不存在将报错 | data:要从集合中删除的数据 | None | set1 = {1,2,3,**"itcast"**} set1.remove(**"itcast"**) |
| pop()  关键词：获取删除 | 从集合中获取并删除第一个数据 | 无 | 获取到的数据 | set1 = {1,2,3,**"itcast"**} data = set1.pop() |
| clear()  关键词：清空 | 清空集合中的数据 | 无 | None | set1 = {1,2,3,**"itcast"**} set1.clear() |

### 字典

#### 4.1 字典的概念

字典是一种使用“键值对结构”存储数据的存储模型。

#### 4.2 字典的特点

字典不具有索引的概念，字典使用键key代替索引，可以通过键操作字典中存储的数据值value。字典可以根据键key进行数据的添加、删除、修改、查询操作。

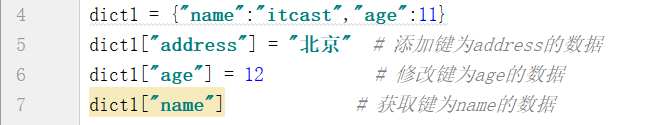
#### 4.3 字典的基本语法

创建字典： 变量名 = {键1：值1，键2：值2，……}

添加数据： 变量名[键] = 值 （字典中没有对应的键）

获取数据： 变量名[键]

修改数据： 变量名[键] = 值 （字典中存在对应的键）



#### 4.4字典常用方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法名 | 功能 | 参数 | 返回值 | 范例 |
| pop(key)  关键词：删除获取 | 从字典中删除指定键key对应的键值对，如果键key不存在将报错 | key:要删除的值value对应的键key | 被删除的值value | dict1 = {**"name"**:**"itcast"**,**"age"**:11} v = dict1.pop(**"name"**) |
| popitem()  关键词：删除 | 从字典中删除指定键key对应的键值对，如果键key不存在将报错 | key:要删除的键值对对应的键key | 被删除的键值对，以元组的形式返回 | dict1 = {**"name"**:**"itcast"**,**"age"**:11} v = dict1.popitem() |
| clear()  关键词：清空 | 清空字典中的数据 | 无 | None | dict1 = {**"name"**:**"itcast"**,**"age"**:11} dict1.clear() |
| setdefault(key,value)  关键词：检测添加 | 添加新的键值对，如果存在对应的键，则忽略该操作 | key:要添加的新键值对对应的键key  value: 要添加的新键值对对应的值value | 字典中key对应的值，如果是添加则返回参数value，如果不是添加，返回原始key对应的value | dict1 = {**"name"**:**"itcast"**,**"age"**:11} dict1.setdefault(**"age"**,22) |
| update(dict)  关键词：更新数据 | 使用新字典中的数据对原始字典数据进行更新 | dict：新字典对象 | None | dict1 = {**"name"**:**"itcast"**,**"age"**:11} dict2 = {**"address"**:**"北京"**,**"age"**:22} dict1.update(dict2) |
| get(key)  关键词：获取 | 根据键key查询字典中对应的值，如果键key不存在将返回None | key:要查询的键key | 根据键key在字典中查询的值value | dict1 = {**"name"**:**"itcast"**,**"age"**:11} v = dict1.get(**"age"**) |
| keys()  关键词：获取键列表 | 获取字典中所有的键key组成的列表数据 | 无 | 由所有键组成的列表 | dict1 = {**"name"**:**"itcast"**,**"age"**:11} dict1.keys() |
| values()  关键词：获取值列表 | 获取字典中所有的值value组成的列表数据 | 无 | 由所有值组成的列表 | dict1 = {**"name"**:**"itcast"**,**"age"**:11} dict1.values() |
| items()  关键词：获取键值对列表 | 获取字典中所有的键值对列表数据 | 无 | 由键值对组成的列表，键值对的格式是元组数据 | dict1 = {**"name"**:**"itcast"**,**"age"**:11} dict1.items() |

#### 4.5注意事项

字典中的键是唯一的

#### 4.6字典的作用

1.当需要存储少量数据，并且期望在编程期以最快的速度获取单个数据， 推荐选择字典。

2.当需要使用非对象格式保存单个对象的属性值，推荐选择字典。

### 格式转换

|  |  |  |  |
| --- | --- | --- | --- |
|  | 转换成列表 | 转换成元组 | 转换成集合 |
| 列表list | — | **tuple**(列表对象） | **set**(列表对象） |
| 元组tuple | **list**(元组对象） | — | **set**(元组对象） |
| 集合set | **list**(集合对象） | **tuple**(集合对象） | — |

### for循环

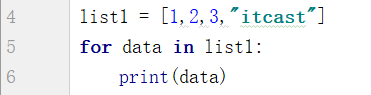
#### 6.1for循环作用

for循环用于对数据存储模型进行访问遍历

#### 6.2for循环基本语法

**for** 变量名 **in** 列表**:**

\_\_\_\_**变量相关操作**



### Range

#### 7.1range的功能

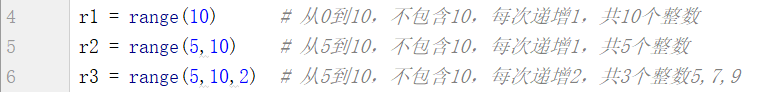
创建连续的整数

#### 7.2range的基本语法

格式1：range(m) 生成 0 到 m-1 的整数

格式2：range(m，n) 生成 m 到 n-1 的整数

格式3：range(m，n，s) 生成 m 到 n-1 的整数，整数间隔为s



#### 7.3range的应用场景

1.配合for循环构造指定次数的循环

**for \_ in range(..):**

\_\_\_\_循环执行的代码

2.快速创建由连续的整数作为数据的列表、元组、集合对象

list(range(..))

tuple(range(..))

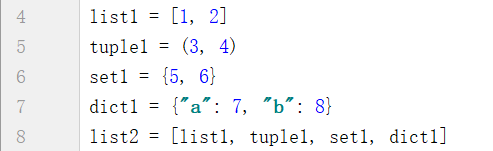
set(range(..))

### 数据存储结构嵌套

#### 8.1数据存储结构嵌套概念

数据结构嵌套指一种数据结构中包含的数据是另一种数据结构

#### 8.2范例



### 公共方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法名 | 功能 | 参数 | 返回值 | 范例 |
| len(model)  关键词：数据总量 | 获取容器模型中的数据总量 | model:保存有数据的存储模型，该模型接受各种容器 | 容器模型中数据的总量 | list1 = [1,2,3,**'itcast'**,**"heima"**] length = len(list1) |
| max(model)  关键词：最大值 | 获取容器模型中的最大值，对于字典获取字典的键key的最大值 | model:保存有数据的存储模型，该模型接受各种容器 | 容器模型中数据的最大值 | list1 = [1,2,3,4,5] max\_value = max(list1) |
| min(model)  关键词：最小值 | 获取容器模型中的最小值，对于字典获取字典的键key的最小值 | model:保存有数据的存储模型，该模型接受各种容器 | 容器模型中数据的最小值 | list1 = [1,2,3,4,5] min\_value = min(list1) |

### 切片

#### 10.1切片的作用

获取列表、元组或字符串中的局部数据

#### 10.2切片基本语法

容器对象[开始索引:结束索引:步长]

#### 10.3特殊格式：

省略开始索引：默认开始索引为0

范例：list1[:5:1] 等同于 list1[0:5:1]

省略结束索引：默认结束索引为数据总量

范例：list1[0::1] 等同于 list1[0:len(list1):1]

省略步长：每次递增索引数为1

范例：list1[0:5:] 等同于 list1[0:5:1]

负数步长： 反向操作/反向步长操作，需要开始索引、结束索引逆序输入

范例：list1[5:0:-1]

常见格式：

范例：list1[:5:] 获取索引5之前的数据（不含5）

范例：list1[4::] 获取索引4之后的数据（含4）

### 通用运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 功能 | 格式 | 适用范围 |
| + | 将两个容器数据合并放入第一个容器 | list1 + list2 | 列表之间或元组之间  （列表与元组之间报错） |
| \* | 将容器数据复制n次放入容器中 | list1 **\*** n | 列表、元组 |
| in | 判断容器中是否包含数据 | data in list1 | 列表、元组、集合、字典  （字典判断数据是否在keys（）中） |
| not in | 判断容器中是否不包含数据 | data not in list1 | 列表、元组、集合、字典  （字典判断数据是否不在keys（）中） |
| >、>=、==、<=、< | 比较两个容器中的数据关系 | list1 <= list2 | 列表、元组、集合 |

### for…else

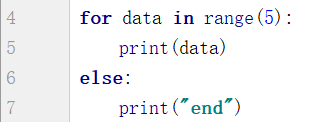
#### 12.1 for…else基本语法

for 变量名 in 列表:

变量相关操作

else:

循环正常运行结束后执行的操作



#### 12.2注意事项

1.如果for循环中执行了break语句，则else中的代码将不执行

2.while循环同样具有while…else语法格式

### 推导式

#### 13.1推导式基本语法

基础语法格式： 循环变量 for循环

范例：list1 = [data **for** data **in** range(5)]

数据处理语法格式： 表达式 for循环

范例：list1 = [data\*5 **for** data **in** range(5)]

数据过滤语法格式： 表达式 for循环 if判断

范例：list1 = [data **for** data **in** range(5) if data > 200]

#### 13.2推导式的作用

推导式可以快速生成数据存储结构中的数据

范例1：创建包含1到100所有整数的列表

list1 = [data **for** data **in** range(1,101)]

范例2：创建包含1到10的平方和的列表

list2 = [data\*\*2 for data in range(1,11)]

范例3：创建字典，key从1到5，value是key的平方

dict1 = [(data,data\*\*2) for data in range(1,6)]

### 案例：斗地主

*"""  
案例斗地主  
分析：  
1.扑克牌作为对象呈现  
2.创建未发牌的牌堆的列表  
3.创建三个玩家牌堆的列表  
4.创建底牌的元组  
5.最原始的牌堆初始化，将54张牌加入到牌堆  
6.创建洗牌操作  
7.创建发牌操作  
"""*import random  
class Poke:  
 pokes = []  
 player1 = []  
 player2 = []  
 player3 = []  
 last = None  
 def \_\_init\_\_(self,flower,num):  
 self.flower = flower  
 self.num = num  
  
 def \_\_str\_\_(self):  
 return "%s%s" % (self.flower,self.num)  
  
 *# 初始化牌* @classmethod  
 def init\_pokes(cls):  
 flowers = ("♠","♥","♣","♦")  
 nums = ("2","3","4","5","6","7","8","9","10","J","Q","K","A")  
 kings = {"big":"大王","small":"小王"}  
 for flower\_ in flowers :  
 for num\_ in nums:  
 p = Poke(flower\_,num\_)  
 cls.pokes.append(p)  
 cls.pokes.append(Poke(kings["big"],""))  
 cls.pokes.append(Poke(kings["small"], ""))  
  
 *#洗牌* @classmethod  
 def wash(cls):  
 *# 洗牌是从牌堆中找出一张固定的牌，与随机的一张牌交换位置  
 # 迭代牌堆，找出一张牌  
 # 产生随机数，作为被交换牌  
 # 交换* for idx in range(54):  
 idxx = random.randint(0,53)  
 cls.pokes[idx],cls.pokes[idxx] = cls.pokes[idxx],cls.pokes[idx]  
  
 *#发牌* @classmethod  
 def send\_poke(cls):  
 for \_ in range(0,17):  
 cls.player1.append(cls.pokes.pop(0))  
 cls.player2.append(cls.pokes.pop(0))  
 cls.player3.append(cls.pokes.pop(0))  
 *# 将剩余的3张牌做成底牌last* cls.last = tuple(cls.pokes)  
  
 *# 临时方法：展示牌堆* @classmethod  
 def show(cls):  
 for poke in cls.pokes:  
 print(poke ,end=" ")  
 print()  
  
 @classmethod  
 def show\_player(cls):  
 print("玩家1",end = ":")  
 for poke in cls.player1:  
 print(poke, end=" ")  
 print()  
 print("玩家2", end=":")  
 for poke in cls.player2:  
 print(poke, end=" ")  
 print()  
 print("玩家3", end=":")  
 for poke in cls.player3:  
 print(poke, end=" ")  
 print()  
 print("底牌", end=":")  
 for poke in cls.last:  
 print(poke, end=" ")  
 print()  
  
Poke.init\_pokes()  
*# Poke.show()*Poke.wash()  
*# Poke.show()*Poke.send\_poke()  
Poke.show\_player()