
python
import turtle
turtle.setup(650,350,200,200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for 1 in range(4):
turtle.circle(40, 80)
turtle.circle(-40, 80)
turtle.fd(40)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)

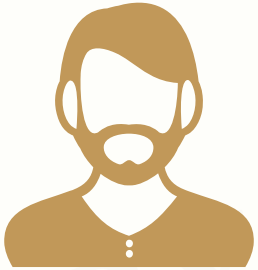
Python语言程序设计

序列类型及操作

嵩 天
北京理工大学

单元开篇

序列类型及操作



- 序列类型定义
- 序列处理函数及方法
- 元组类型及操作
- 列表类型及操作
- 序列类型应用场景

python
语言程序设计

序列类型定义

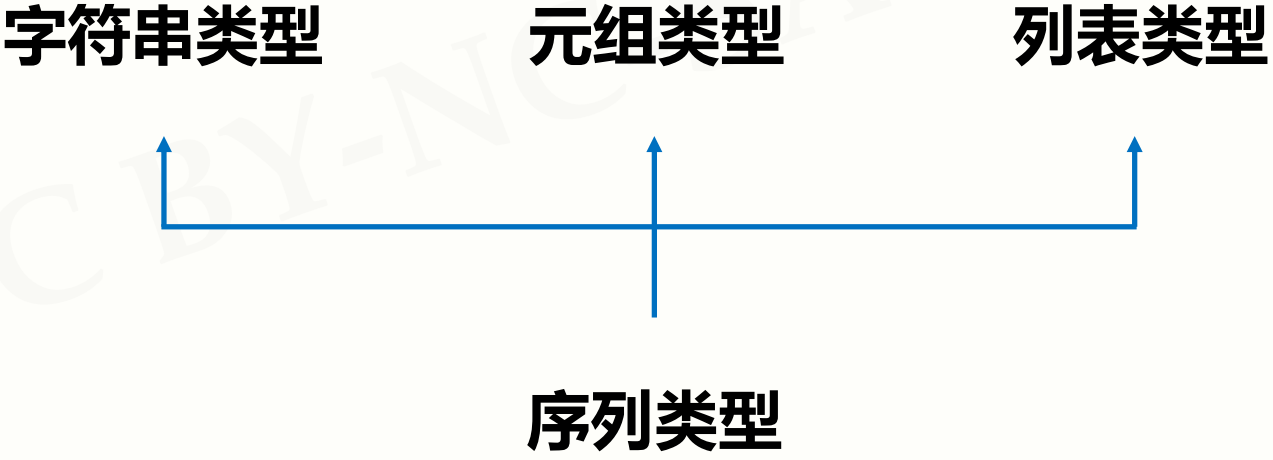
序列类型定义

序列是具有先后关系的一组元素

- **序列是一维元素向量，元素类型可以不同**
- **类似数学元素序列：** $s_0, s_1, \cdots, s_{n-1}$
- **元素间由序号引导，通过下标访问序列的特定元素**

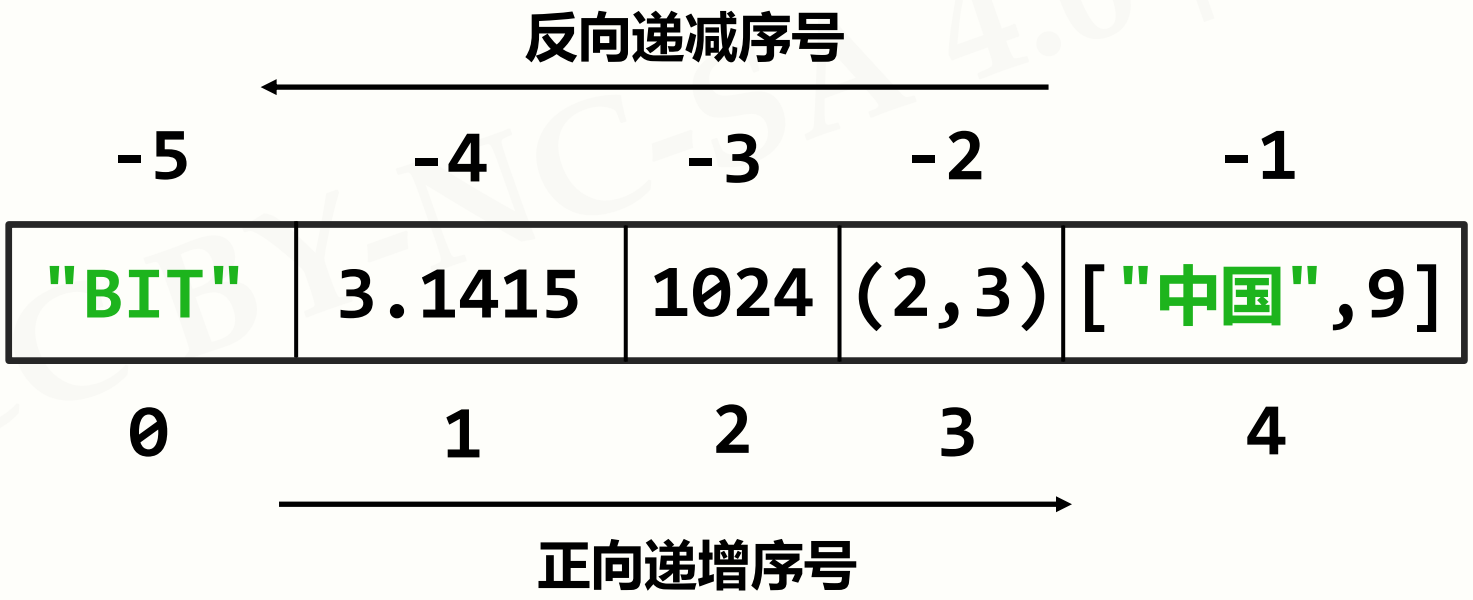
序列类型定义

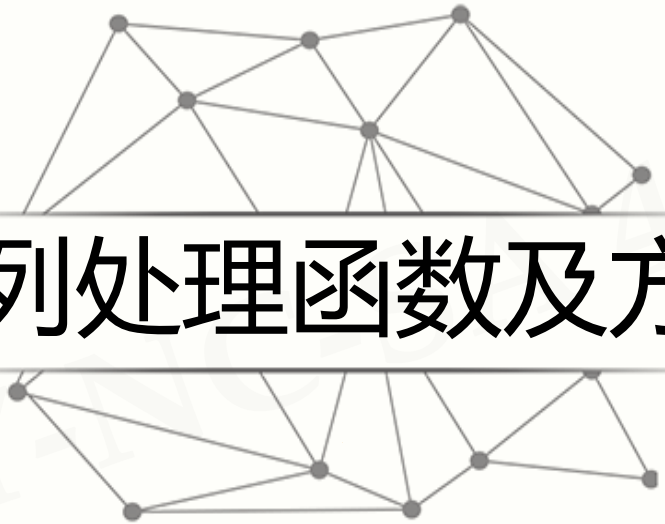
序列是一个基类类型



序列类型定义

序号的定义





序列处理函数及方法

序列类型通用操作符

6个操作符

操作符及应用	描述
x in s	如果x是序列s的元素，返回True，否则返回False
x not in s	如果x是序列s的元素，返回False，否则返回True
s + t	连接两个序列s和t
s*n 或 n*s	将序列s复制n次
s[i]	索引，返回s中的第i个元素，i是序列的序号
s[i: j] 或 s[i: j: k]	切片，返回序列s中第i到j以k为步长的元素子序列

序列类型操作实例

```
>>> ls = ["python", 123, ".io"]
>>> ls[::-1]
['.io', 123, 'python']
>>> s = "python123.io"
>>> s[::-1]
'oi.321nohtyp'
```

序列类型通用函数和方法

5个函数和方法

函数和方法	描述
len(s)	返回序列s的长度
min(s)	返回序列s的最小元素，s中元素需要可比较
max(s)	返回序列s的最大元素，s中元素需要可比较
s.index(x) 或 s.index(x, i, j)	返回序列s从i开始到j位置中第一次出现元素x的位置
s.count(x)	返回序列s中出现x的总次数

序列类型操作实例

```
>>> ls = ["python", 123, ".io"]
>>> len(ls)
3
>>> s = "python123.io"
>>> max(s)
'y'
```

元组类型及操作

元组类型定义

元组是序列类型的一种扩展

- 元组是一种序列类型，一旦创建就不能被修改
- 使用小括号 () 或 tuple() 创建，元素间用逗号，分隔
- 可以使用或不使用小括号

```
def func():
    return 1,2
```

元组类型定义

```
>>> creature = "cat", "dog", "tiger", "human"
>>> creature
('cat', 'dog', 'tiger', 'human')
>>> color = (0x001100, "blue", creature)
>>> color
(4352, 'blue', ('cat', 'dog', 'tiger', 'human'))
```

元组类型操作

元组继承序列类型的全部通用操作

- 元组继承了序列类型的全部通用操作
- 元组因为创建后不能修改，因此没有特殊操作
- 使用或不使用小括号

元组类型操作

```
>>> creature = "cat", "dog", "tiger", "human"
>>> creature[::-1]
('human', 'tiger', 'dog', 'cat')
>>> color = (0x001100, "blue", creature)
>>> color[-1][2]
'tiger'
```

列表类型及操作

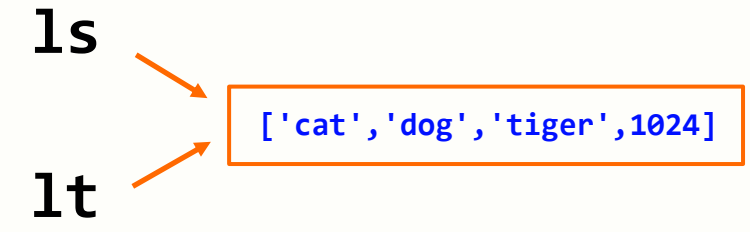
列表类型定义

列表是序列类型的一种扩展，十分常用

- **列表是一种序列类型，创建后可以随意被修改**
- **使用方括号 [] 或list() 创建，元素间用逗号，分隔**
- **列表中各元素类型可以不同，无长度限制**

列表类型定义

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> ls
['cat', 'dog', 'tiger', 1024]
>>> lt = ls
>>> lt
['cat', 'dog', 'tiger', 1024]
```



方括号 [] 真正创建一个列表，赋值仅传递引用

列表类型操作函数和方法

函数或方法	描述
ls[i] = x	替换列表ls第i元素为x
ls[i: j: k] = lt	用列表lt替换ls切片后所对应元素子列表
del ls[i]	删除列表ls中第i元素
del ls[i: j: k]	删除列表ls中第i到第j以k为步长的元素
ls += lt	更新列表ls，将列表lt元素增加到列表ls中
ls *= n	更新列表ls，其元素重复n次

列表类型操作

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> ls[1:2] = [1, 2, 3, 4]
['cat', 1, 2, 3, 4, 'tiger', 1024]
>>> del ls[::3]
[1, 2, 4, 'tiger']
>>> ls*2
[1, 2, 4, 'tiger', 1, 2, 4, 'tiger']
```

列表类型操作函数和方法

函数或方法	描述
ls.append(x)	在列表ls最后增加一个元素x
ls.clear()	删除列表ls中所有元素
ls.copy()	生成一个新列表，赋值ls中所有元素
ls.insert(i,x)	在列表ls的第i位置增加元素x
ls.pop(i)	将列表ls中第i位置元素取出并删除该元素
ls.remove(x)	将列表ls中出现的第一个元素x删除
ls.reverse()	将列表ls中的元素反转

列表类型操作

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> ls.append(1234)
['cat', 'dog', 'tiger', 1024, 1234]
>>> ls.insert(3, "human")
['cat', 'dog', 'tiger', 'human', 1024, 1234]
>>> ls.reverse()
[1234, 1024, 'human', 'tiger', 'dog', 'cat']
```

列表功能默写

- 定义空列表lt
- 向lt新增5个元素
- 修改lt中第2个元素
- 向lt中第2个位置增加一个元素
- 从lt中第1个位置删除一个元素
- 删除lt中第1-3位置元素
- 判断lt中是否包含数字0
- 向lt新增数字0
- 返回数字0所在lt中的索引
- lt的长度
- lt中最大元素
- 清空lt

列表功能默写

- 定义空列表lt
- 向lt新增5个元素
- 修改lt中第2个元素
- 向lt中第2个位置增加一个元素
- 从lt中第1个位置删除一个元素
- 删除lt中第1-3位置元素

```
>>> lt = []
>>> lt += [1,2,3,4,5]
>>> lt[2] = 6
>>> lt.insert(2, 7)
>>> del lt[1]
>>> del lt[1:4]
```

列表功能默写

```
>>> 0 in lt
>>> lt.append(0)
>>> lt.index(0)
>>> len(lt)
>>> max(lt)
>>> lt.clear()
```

- 判断lt中是否包含数字0
- 向lt新增数字0
- 返回数字0所在lt中的索引
- lt的长度
- lt中最大元素
- 清空lt

序列类型应用场景

序列类型应用场景

数据表示：元组 和 列表

- 元组用于元素不改变的应用场景，更多用于固定搭配场景
- 列表更加灵活，它是最常用的序列类型
- 最主要作用：表示一组有序数据，进而操作它们

序列类型应用场景

元素遍历

```
for item in ls :
    <语句块>
```

```
for item in tp :
    <语句块>
```

序列类型应用场景

数据保护

- 如果不希望数据被程序所改变，转换成元组类型

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> lt = tuple(ls)
>>> lt
('cat', 'dog', 'tiger', 1024)
```

单元小结

序列类型及操作

- 序列是基类类型，扩展类型包括：字符串、元组和列表
- 元组用()和tuple()创建，列表用[]和set()创建
- 元组操作与序列操作基本相同
- 列表操作在序列操作基础上，增加了更多的灵活性