

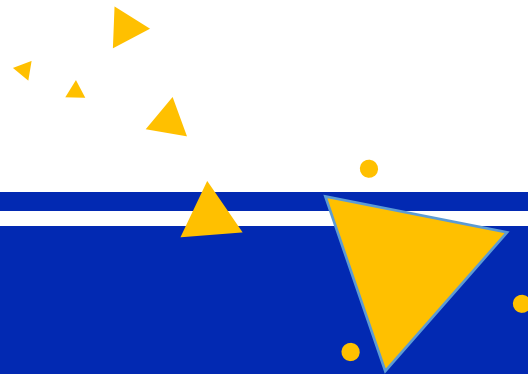


# Python编程基础

## --序列（列表、元组、字典）

主讲：万永权

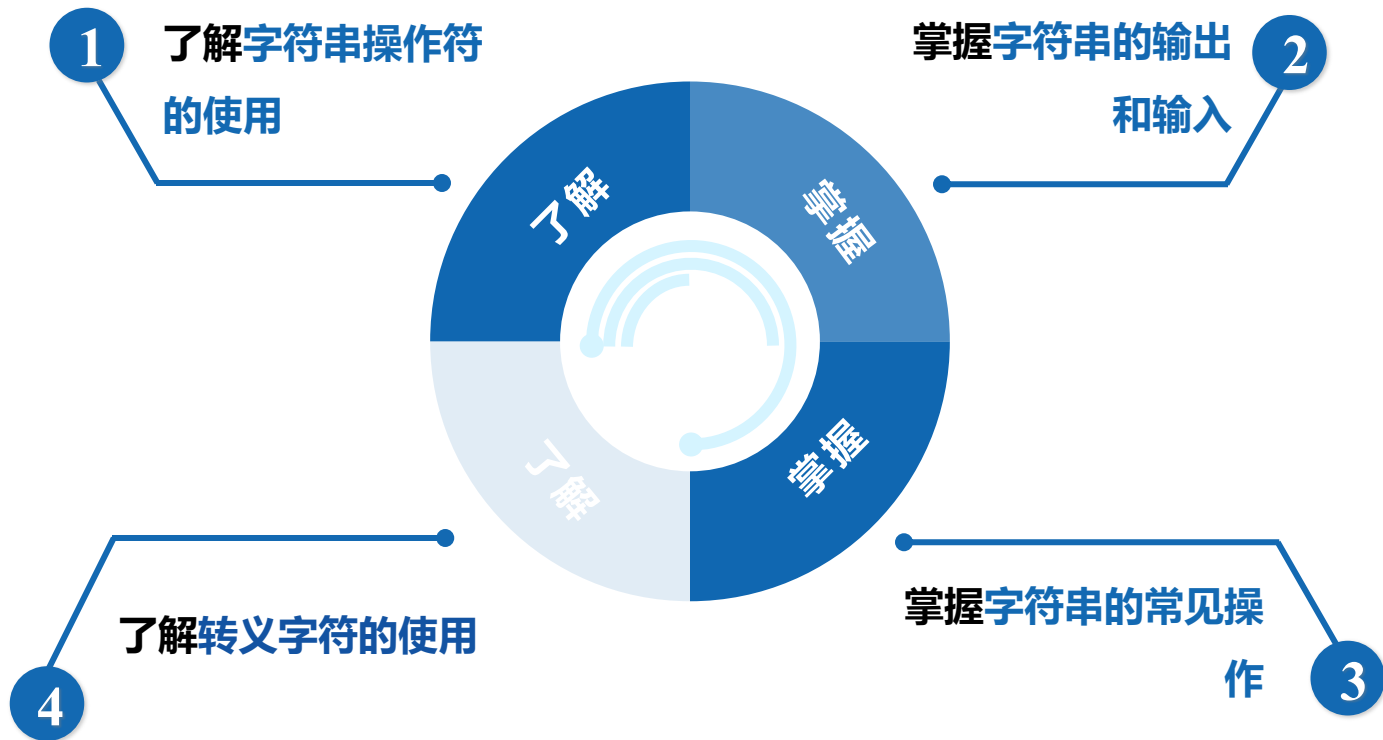




# 字符串



# 学习目标



# 什么字符串

字符串是一种表示文本数据的类型。

使用单引号

'a' 、 '123'

使用双引号

"a"、 "123"

使用三引号

.....

Hello

.....

# 转义字符

看下面的代码：

```
>>>'let's go! go'
File "<input>", line 1
    'let's go! go'
      ^
SyntaxError: invalid syntax
```

对于单引号或者双引号这些特殊的符号，我们可以对他们进行**转义**。例如，对字符串中的单引号进行转义：

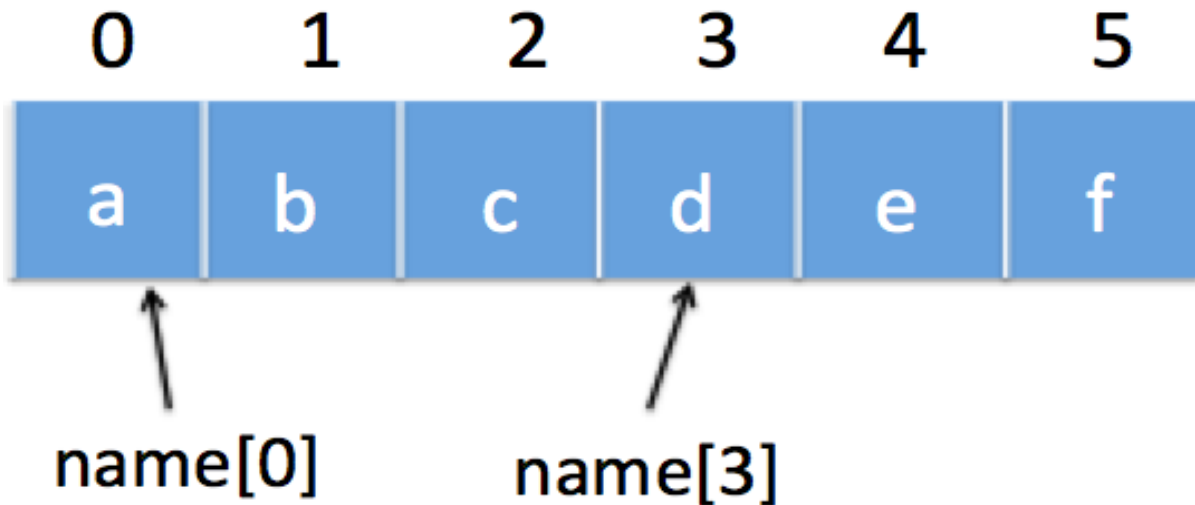
```
>>>'let\'s go! go'
"let's go! go"
```

# 转义字符

转义字符	代表含义
\\(在行尾时)	反斜杠符号
\\	反斜杠符号
\"	双引号
\\n	换行
\\b	退格
\\t	横向制表符

## 字符串的存储方式

字符串中的每个字符都对应一个下标，下标编号是从0开始的。



# 什么是切片？

切片的语法格式如下所示：

[起始:结束:步长]

切片选取的区间属于左闭右开型，即从"起始"位开始，到"结束"位的前一位结束（不包含结束位本身）



## 使用切片截取字符串


假设有字符串 `name= "abcdef"` , 则:

`name[0:3]`            `abc`

`name[3:5]`            `de`

`name[1:-1]`            `bcde`

`name[2:])`            `cdef`

`name[::-2]`            `fdb`

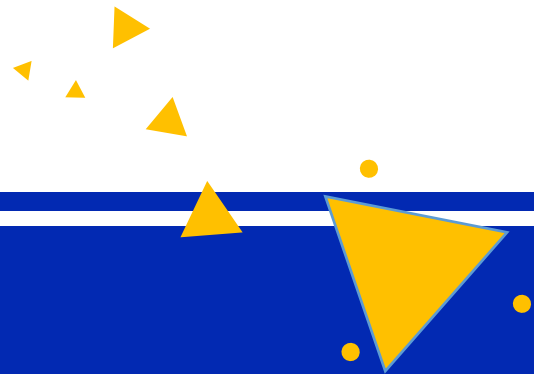
# 字符串内建函数

■ split函数：通过指定分隔符对字符串进行切片

```
str.split(str=" ", num=string.count(str))
```

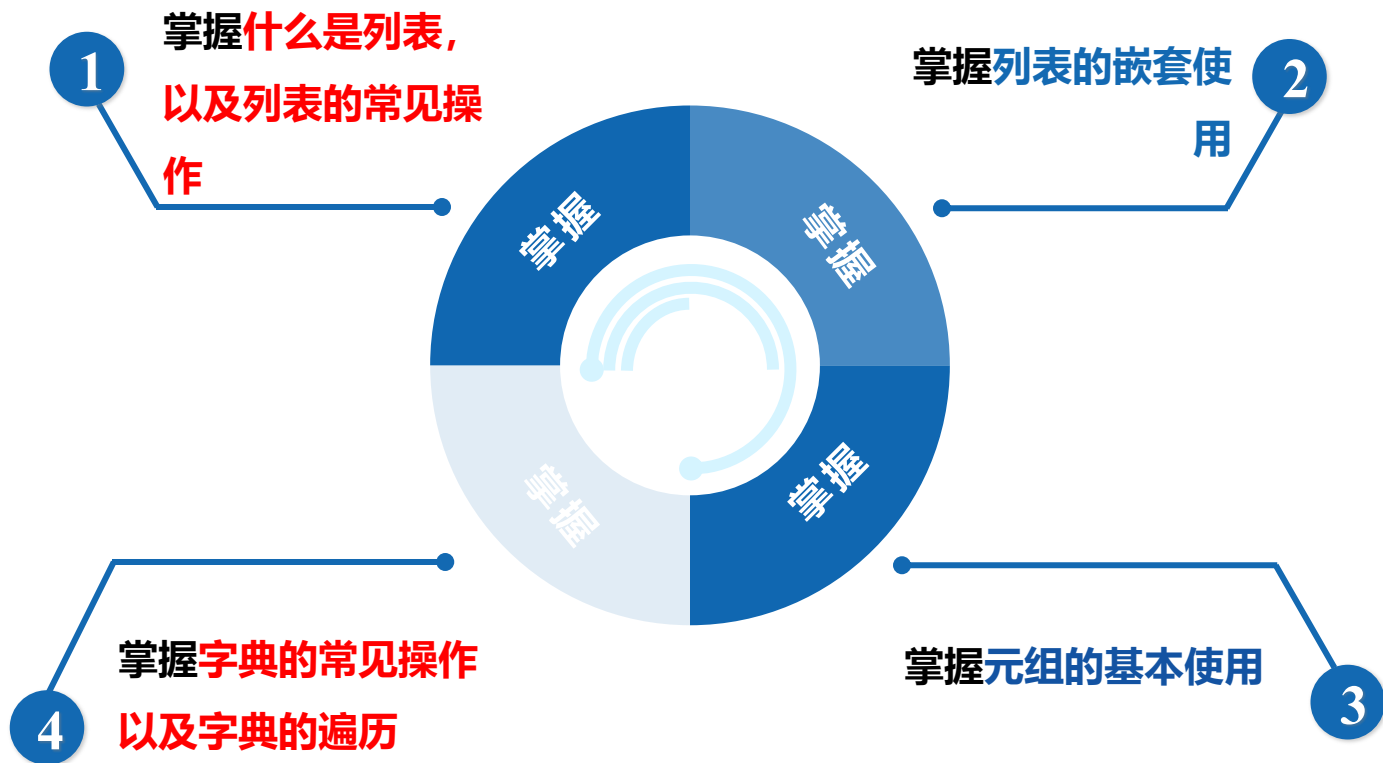
参数如下：

- str -- 分隔符。默认为所有空字符。
- num -- 分割次



# 序列

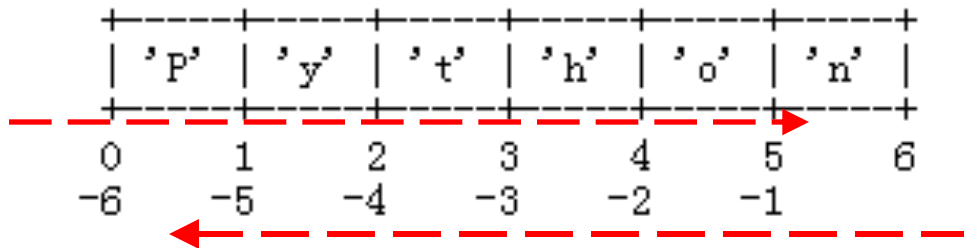
# 学习目标



# [概念]

**序列**是一块用于存放**多个**值的连续内存空间，并且按一定顺序排列，可以通过**索引**取值。

- 列表、元组、字符串支持**双向索引**，第一个元素下标为0，第二个元素下标为1，以此类推；最后一个元素下标为-1，倒数第二个元素下标为-2，以此类推。





# 索引

1

# 序列相加

3

# 检查某个 元素是否 是序列的 成员

5

# 切片

2

# 乘法

4

# 计算序列 的长度、 最大值和 最小值

6

# 列表

- ◆列表是Python中最基本的数据结构,是一种线性表的表示方式,也是最常用的Python 数据类型。
- ◆列表的数据项的**数据类型可以各不相同**,可以同时分别为整数、实数、字符串等基本类型,甚至是列表、元组、字典、集合以及其他自定义类型的对象。
- ◆Python对列表也提供了非常便捷的操作,如 创建、访问、切片、增加、扩展、更新、删除等



遍历  
列表

列表的  
创建和  
删除

二维列  
表的使用

访问列  
表元素

添加、修  
改和删除  
列表元素





## 1 使用赋值运算符直接创建列表

```
numList = [10, 20, 30, 40]
```

## 2 创建空列表

```
emptyList = list()
```



# 访问 列表元素



## 列表访问

列表索引是从0开始的，我们可以通过下标索引的方式来访问列表中的值。

```
A = ['xiaoWang', 'xiaoZhang', 'xiaoHua']  
print(A[0])  
print(A[1])
```

# 成员资格判断

- 使用 **in** 关键字来判断一个值是否存在于列表中，返回结果为“True”或“False”。
- in 操作也适用于 **元组** 和 **字典**。

```
>>> aList
```

```
[3, 4, 5, 5.5, 7, 9, 11, 13, 15, 17]
```

```
>>> 3 in aList
```

```
True
```

```
>>> 18 in aList
```

```
False
```

```
>>> bList = [[1], [2], [3]]
```

```
>>> 3 in bList
```

```
False
```

```
>>> 3 not in bList
```

```
True
```

```
>>> [3] in bList
```

```
True
```

```
>>> aList = [3, 5, 7, 9, 11]
```

```
>>> bList = ['a', 'b', 'c', 'd']
```

```
>>> (3, 'a') in zip(aList, bList)
```

```
True
```

```
>>> for a, b in zip(aList, bList):  
        print(a, b)
```



切片

# 列表的切片操作

- 切片适用于列表、元组、字符串、range对象等类型，但作用于列表时功能最强大。可以使用切片来**截取**列表中的任何部分，得到一个新列表，也可以通过切片来**修改**和**删除**列表中部分元素，甚至可以通过切片操作为列表对象**增加**元素。
- 切片使用**2个冒号分隔的3个数字**来完成：
  - ✓ **第一个数字**表示切片开始位置（默认为0）。
  - ✓ **第二个数字**表示切片截止（但**不包含**）位置（默认为列表长度）。
  - ✓ **第三个数字**表示切片的步长（默认为1），当步长省略时可以顺便省略最后一个冒号。
- 切片操作不会因为下标越界而抛出异常，而是简单地在列表尾部截断或者返回一个空列表，代码具有**更强的健壮性**。

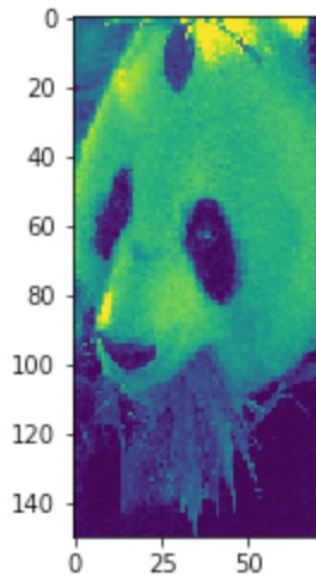
# 切片操作

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList[0:9:1]                                #前9个元素
[3, 4, 5, 6, 7, 9, 11, 13, 15]
>>> aList[3:6]                                    #下标在[3, 6)之间的所有元素
[6, 7, 9]
>>> aList[:,2]                                    #偶数位置, 隔一个取一个; 前两个参数采用默认值
[3, 5, 7, 11, 15]
>>> aList[1::2]                                    #奇数位置, 隔一个取一个; 第二个参数采用默认值
[4, 6, 9, 13, 17]
>>> aList[3::]                                    #从下标3开始的所有元素; 后2个参数采用默认值
[6, 7, 9, 11, 13, 15, 17]
>>> aList[100:]                                    #下标100之后的所有元素, 自动截断
[]
>>> aList[::-1]                                    #逆序的所有元素
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
```

# 切片

## ■ 只显示脸部的图像

```
In [11]: plt.imshow(panda_tensor[25:175,60:130,0].numpy());
```







# 遍历列表



# 列表的循环遍历

## 1. 使用for循环遍历列表

```
namesList = ['xiaoWang','xiaoZhang','xiaoHua']  
for name in namesList:  
    print(name)
```



# 列表的循环遍历

## 2. 使用while循环遍历列表

```
namesList = ['xiaoWang', 'xiaoZhang', 'xiaoHua']  
length = len(namesList)  
i = 0  
while i < length:  
    print(namesList[i])  
    i += 1
```



## 添加、修改和删除列表元素



# 在列表中增加元素

在列表中增加元素的方式有多种，具体如下：

- 通过`append`可以向列表添加元素
- 通过`extend`可以将另一个列表的元素添加到列表中。
- 通过`insert`在指定位置`index`前插入元素`object`。



# 在列表中查找元素

在列表中查找元素的方法包括：

- in（存在）,如果存在那么结果为true， 否则为false。
- not in（不存在）, 如果不存在那么结果为true， 否则false。



## 在列表中修改元素

列表元素的修改，也是通过**下标**来实现的。

```
A = ['xiaoWang','xiaoZhang','xiaoHua']
```

```
A[1] = 'xiaoLu'
```



# 在列表中删除元素

列表元素的常用删除方法有三种，具体如下：

- `del`：根据下标进行删除
- `pop`：删除最后一个元素
- `remove`：根据元素的值进行删除





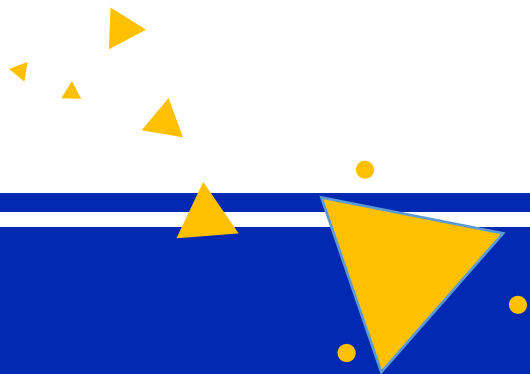
# 二维列表的 使用



# 列表的嵌套

**列表的嵌套**指的是一个列表的元素又是一个列表。

```
schoolNames = [['北京大学','清华大学'],  
                ['南开大学','天津大学','天津师范大学'],  
                ['山东大学','中国海洋大学']]
```



# 元组

# 元组与列表的区别

- ❖ 元组一旦定义就**不允许更改**。
- ❖ 元组没有append()、extend()和insert()等方法，**无法向元组中添加元素**。
- ❖ 元组没有remove()或pop()方法，也无法对元组元素进行del操作，**不能从元组中删除元素**。
- ❖ 从效果上看，tuple()冻结列表，而list()融化元组。



# 元组

## 不可变序列

（不能添加、修改和删除元素，可以整体替换）

## 支持切片的操作

（只能访问元组中的元素）

## 元组访问速度快

## 元组可以作为字典键

# 列表

## 可变序列

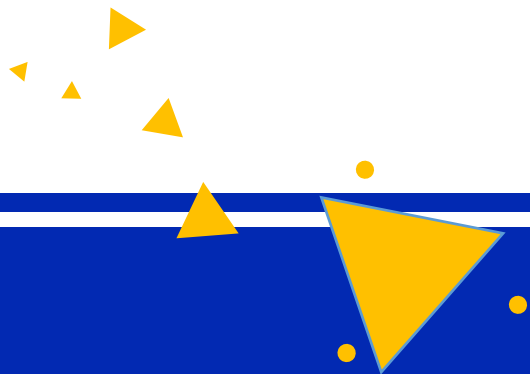
（随时添加、修改或者删除）

## 支持切片的操作

（可以访问、修改元组中的元素）

## 列表访问速度慢

## 不能作为字典的键



# 字典



字典的  
创建和  
删除

1

遍历  
字典

3

通过键  
值对访  
问字典

2

添加、修  
改和删除  
字典元素

4



# 字典介绍

字典是一种存储数据的容器，它和列表一样，都可以存储多个数据。

```
info = {'name':'班长', 'sex':'f', 'address':'北京'}
```

每个元素都是由两部分组成的，分别是键和值。

‘name’为键，‘班长’为值。





# 创建字典

## 1 使用赋值运算符直接创建字典

```
a_dict = {'server': 'db.diveintopython3.org', 'database': 'mysql'}
```

## 2 创建空字典

```
x1 = dict()
```

```
x2 = dict('server': 'db.diveintopython3.org', 'database': 'mysql')
```



# 字典的常见操作

## 1. 根据键访问值

```
info = {'name':'班长', 'id':100, 'sex':'f', 'address':'北京'}  
print(info['name'])  
print(info['address'])
```

**注意：**如果使用的是不存在的键，则程序会报错。



# 字典的常见操作

如果我们不确定字典中是否存在某个键而又想获取其值时，可以使用get方法，还可以设置默认值。

```
info = {'name':'班长', 'id':100, 'sex':'f', 'address':'北京'}  
age = info.get('age')  
print(age) # 'age'键不存在，所以age为None  
print(type(age))  
age = info.get('age', 18) # 若info不存在'age'，返回默认值18  
print(age)
```



# 字典的常见操作

## 2. 修改字典的元素

```
info = {'name':'班长', 'id':100, 'sex':'f', 'address':'北京'}  
newId = input('请输入新的学号')  
info['id'] = int(newId)  
print('修改之后的id为: %d'%info['id'])
```



# 字典的常见操作

## 3. 添加字典元素

```
info = {'name':'班长', 'id':100, 'sex':'f', 'address':'北京'}  
newId = input('请输入新的学号')  
info['id'] = newId  
print(info)
```



# 字典的常见操作

## 4. 删除字典元素

- **del**: 用于删除字典；删除后，字典完全不存在了，无法再根据键访问字典的值。
- **clear**: 只是清空字典中的数据，字典还存在，只不过没有元素。



# 字典的常见操作

## 6. 获取字典中键的列表

**keys()**方法返回在字典中的所有可用的键的列表。

```
dict = {'Name': 'Zara', 'Age': 7};  
print(dict.keys())
```



# 字典的常见操作

## 7. 获取字典中值的列表

**values()**方法返回在字典中的所有可用的值的列表

```
dict = {'Name': 'Zara', 'Age': 7};  
print(dict.values())
```





# 字典的常见操作

## 8. 计算字典中键值对的个数

**items()**方法返回字典的(键, 值)元组对的列表

```
dict = {'Name': 'Zara', 'Age': 7}
print("Value : %s" % dict.items())
```



# 遍历字典



# 字典的遍历

## 1. 遍历字典的键key

```
dict = {'Name': 'Zara', 'Age': 7}
for key in dict.keys():
    print(key)
```



# 字典的遍历

## 2. 遍历字典的值value

```
dict = {'Name': 'Zara', 'Age': 7}
for value in dict.values():
    print(value)
```



# 字典的遍历

## 3. 遍历字典的元素

```
dict = {'Name': 'Zara', 'Age': 7}
for item in dict.items():
    print(item)
```



# 字典的遍历

## 4. 遍历字典的键值对

```
dict = {'Name': 'Zara', 'Age': 7}
for key,value in dict.items():
    print("key=%s, value=%s"%(key,value))
```