



Python实践教学





3.1 字符串介绍



- 字符串介绍

- 想一想：

当打来浏览器登录某些网站的时候，需要输入密码，浏览器把密码传送到服务器后，服务器会对密码进行验证，其验证过程是把之前保存的密码与本次传递过去的密码进行对比，如果相等，那么就认为密码正确，否则就认为不对；服务器既然想要存储这些密码可以用数据库（比如MySQL），当然为了简单起见，咱们可以先找个变量把密码存储起来即可；那么怎样存储带有字母的密码呢？

- 答：

字符串

- <1>python中字符串的格式
- 如下定义的变量a，存储的是数字类型的值
- `a = 100`
- 如下定义的变量b，存储的是字符串类型的值
- `b = "hello itcast.cn"`
- 或者
- `b = 'hello itcast.cn'`
- 小总结：
- 双引号或者单引号中的数据，就是字符串



3.2 字符串的使用



字符串

- 字符串输出
- Demo

```
name = 'xiaoming'  
position = '讲师'  
address = '北京市昌平区建材城西路金燕龙办公楼1层'  
  
print('-----')  
print("姓名：%s"%name)  
print("职位：%s"%position)  
print("公司地址：%s"%address)  
print('-----')
```

- 结果：

```
-----  
姓名： xiaoming  
职位： 讲师  
公司地址： 北京市昌平区建材城西路金燕龙办公楼1层  
-----
```

- 字符串输入
- 之前在学习input的时候，通过它能够完成从键盘获取数据，然后保存到指定的变量中；
- 注意：input获取的数据，都以字符串的方式进行保存，即使输入的是数字，那么也是以字符串方式保存

- 字符串的拼接

```
In [8]: a = "lao"
In [9]: b = "wang"
In [10]: c = "zhao"
In [11]: d = a+b
In [12]: d
Out[12]: 'laowang'

In [13]: A = 100
In [14]: B = 200
In [15]: C = A+B
In [16]: C
Out[16]: 300

In [17]: e = "===" + a + b + "==="
In [18]: e
Out[18]: '===laowang==='

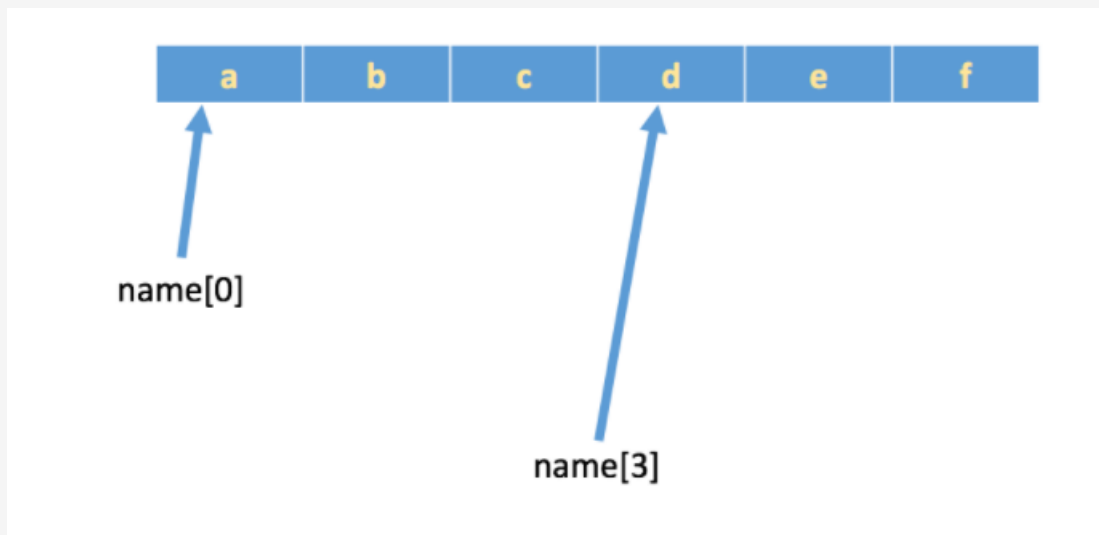
In [19]:
In [19]: f = "===%s==="%(a+b)
In [20]: f
Out[20]: '===laowang==='
```


字符串

- 下标和切片
- 1. 下标索引
- 所谓“下标”，就是编号，就好比超市中的存储柜的编号，通过这个编号就能找到相应的存储空间
- 生活中的“下标”
- 超市储物柜



- 字符串中"下标"的使用
- 列表与元组支持下标索引好理解，字符串实际上就是字符的数组，所以也支持下标索引。
- 如果有字符串:`name = 'abcdef'`，在内存中的实际存储如下：



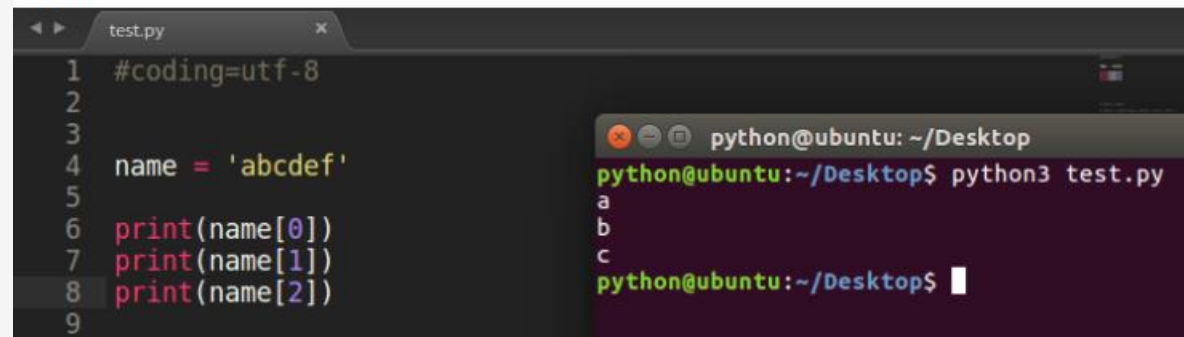
- 如果想取出部分字符，那么可以通过下标的方法，（注意python中下标从 0 开始）

- 如果想取出部分字符，那么可以通过下标的方法，（注意python中下标从 0 开始）

```
name = 'abcdef'

print(name[0])
print(name[1])
print(name[2])
```

运行结果:



The image shows a code editor window titled 'test.py' with the following code:

```
1 #coding=utf-8
2
3
4 name = 'abcdef'
5
6 print(name[0])
7 print(name[1])
8 print(name[2])
9
```

Next to the code editor is a terminal window titled 'python@ubuntu: ~/Desktop'. It shows the command 'python3 test.py' being executed, with the output:

```
python@ubuntu:~/Desktop$ python3 test.py
a
b
c
python@ubuntu:~/Desktop$
```

- 2. 切片
- 切片是指对操作的对象截取其中一部分的操作。字符串、列表、元组都支持切片操作。
- 切片的语法：[起始:结束:步长]
- 注意：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）。
- 我们以字符串为例讲解。
- 如果取出一部分，则可以在中括号[]中，使用：

```
name = 'abcdef'

print(name[0:3]) # 取 下标0~2 的字符
```

- `name[:2]` 取从头开始带下标为1的字符

```
name = 'abcdef'

print(name[2:]) # 取 下标为2开始到最后的字符
```

- 其他使用如下:

```
>>> a = "abcdef"
>>> a[:3]
'abc'
>>> a[::2]
'ace'
>>> a[5:1:-2]
''
>>> a[1:5:2]
'bd'
>>> a[::-2]
'fdb'
>>> a[5:1:-2]
'fd'
```



3.3 字符串常用操作



- 如有字符串 `mystr = 'hello world itcast and itcastcpp'`，以下是常见的操作
- **<1>find**
- 检测 `str` 是否包含在 `mystr` 中，如果是返回开始的索引值，否则返回 -1

```
mystr.find(str, start=0, end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.find("itcast")
12
>>>
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.find("itcast", 0, 10)
-1
>>>
```

- **<2>index**
- 跟find()方法一样，只不过如果str不在 mystr中会报一个异常。

```
mystr.index(str, start=0, end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.find("itcast",0,10)
-1
>>> mystr.index("itcast",0,10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>>
```

- **<3>count**
- 返回 str在start和end之间 在 mystr里面出现的次数

```
mystr.count(str, start=0, end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.count("itcast")
2
>>>
```


- **<4>replace**
- 把 mystr 中的 str1 替换成 str2,如果 count 指定,则替换不超过 count 次.

```
mystr.replace(str1, str2, mystr.count(str1))
```

```
>>> name="hello world ha ha"
>>> name.replace("ha", "Ha")
'hello world Ha Ha'
>>> name.replace("ha", "Ha", 1)
'hello world Ha ha'
>>> 
```

- **<5>split**
- 以 str 为分隔符切片 mystr, 如果 maxsplit有指定值,则仅分隔 maxsplit 个子字符串

```
mystr.split(str=" ", 2)
```

```
>>> name="hello world ha ha"
>>> name.split(" ")
['hello', 'world', 'ha', 'ha']
>>> name.split(" ", 2)
['hello', 'world', 'ha ha']
>>> 
```

- **<6>startswith**
- 检查字符串是否是以 obj 开头, 是则返回 True, 否则返回 False

```
mystr.startswith(obj)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.startswith("hello")
True
>>> mystr.startswith("Hello")
False
>>>
```

- **<7>endswith**
- 检查字符串是否以obj结束, 如果是返回True, 否则返回 False.
- 个子字符串

```
mystr.endswith(obj)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.endswith('cpp')
True
>>> mystr.endswith('app')
False
>>>
```

字符串

- **<8>isalpha**
- 如果 mystr 所有字符都是字母 则返回 True,否则返回 False

```
mystr.isalpha()
```

```
>>> mystr = 'abc'
>>> mystr.isalpha()
True
>>> mystr = '123'
>>> mystr.isalpha()
False
>>> mystr = 'abc 123'
>>> mystr.isalpha()
False
>>>
```

- **<9>isdigit**
- 如果 mystr 只包含数字则返回 True 否则返回 False.

```
mystr.isdigit()
```

```
>>> mystr = 'abc'
>>> mystr.isdigit()
False
>>> mystr = '123'
>>> mystr.isdigit()
True
>>> mystr = 'abc123'
>>> mystr.isdigit()
False
>>>
```

字符串

- **<10>isalnum**
- 如果 mystr 所有字符都是字母或数字则返回 True, 否则返回 False

- **<11>isspace**
- 如果 mystr 只包含空格则返回 True 否则返回 False.

```
mystr.isalnum()
```

```
>>> mystr = '123'
>>> mystr.isalnum()
True
>>> mystr = 'abc'
>>> mystr.isalnum()
True
>>> mystr = 'abc123'
>>> mystr.isalnum()
True
>>> mystr = 'abc 123'
>>> mystr.isalnum()
False
>>>
```

```
mystr.isspace()
```

```
>>> mystr = 'abc123'
>>> mystr.isspace()
False
>>> mystr = ''
>>> mystr.isspace()
False
>>> mystr = ' '
>>> mystr.isspace()
True
>>> mystr = '   '
>>> mystr.isspace()
True
```

字符串

- **<12>lower**
- 转换 mystr 中所有大写字符为小写

```
mystr.lower()
```

```
>>> mystr = 'HELLO world itcast and itcastcpp'
>>> mystr.lower()
'hello world itcast and itcastcpp'
>>>
```

- **<13>upper**
- 转换 mystr 中的小写字母为大写

```
mystr.upper()
```

```
>>> mystr = 'HELLO world itcast and itcastcpp'
>>> mystr.upper()
'HELLO WORLD ITCAST AND ITCASTCPP'
>>>
```

字符串

- `<14>join`
- `mystr` 中每个字符后面插入`str`,构造出一个新的字符串

```
mystr.join(str)
```

```
>>> str = " "  
>>> li = ["my", "name", "is", "dongGe"]  
>>> str.join(li)  
'my name is dongGe'  
>>> str = "_"   
>>> str.join(li)  
'my_name_is_dongGe'  
>>>
```

```
mystr.splitlines()
```

```
>>> mystr="hello\nworld"
>>> print mystr
hello
world
>>> mystr.splitlines()
['hello', 'world']
>>>
```

- **<15>splitlines**
- 按照行分隔，返回一个包含各行作为元素的列表

- **<16>partition**
- 把mystr以str分割成三部分,str前，str和str后

```
mystr.partition(str)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.partition("itcast")
('hello world ', 'itcast', ' and itcastcpp')
>>>
```

- **<17>rpartition**
- 类似于 partition()函数,不过是从右边开始.

```
mystr.rpartition(str)
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.partition("itcast")
('hello world ', 'itcast', ' and itcastcpp')
>>> mystr.rpartition("itcast")
('hello world itcast and ', 'itcast', 'cpp')
>>>
```


- **<18>rfind**
- 类似于 find()函数，不过是从右边开始查找.

```
mystr.rfind(str, start=0,end=len(mystr) )
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.rfind("itcast")
23
>>> 
```

- **<19>rindex**
- 类似于 index(), 不过是从右边开始.

```
mystr.rindex( str, start=0,end=len(mystr))
```

```
>>> mystr = 'hello world itcast and itcastcpp'
>>> mystr.rindex("IT")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> 
```



3.4 列表介绍



- 列表介绍

想一想：

前面学习的字符串可以用来存储一串信息，那么想一想，怎样存储咱们班所有同学的名字呢？

定义100个变量，每个变量存放一个学生的姓名可行吗？有更好的办法吗？

答：

列表

- <1>列表的格式
- 变量A的类型为列表

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

- 比C语言的数组强大的地方在于列表中的元素可以是不同类型的

```
testList = [1, 'a']
```

- <2>打印列表

demo:

```
namesList = ['xiaoWang','xiaoZhang','xiaoHua']  
print(namesList[0])  
print(namesList[1])  
print(namesList[2])
```

结果：

```
xiaoWang  
xiaoZhang  
xiaoHua
```

列表

- <3>列表的下标

```
[50]: name = "laowang"
```

```
[51]: name[0]
```

```
[51]: 'l'
```

```
[52]: names
```

```
[52]: ['八戒', '沙僧', '老李', '老刘', '老赵', '悟空', '八戒', '老王', '沙僧', '老李']
```

```
[53]: names[0]
```

```
[53]: '八戒'
```

```
[54]: names[1]
```

```
[54]: '沙僧'
```

```
[55]: names[2]
```

```
[55]: '老李'
```

```
[56]: names[-1]
```

```
[56]: '老李'
```

```
[57]: names[2:5]
```

```
[57]: ['老李', '老刘', '老赵']
```

```
[58]:
```



3.5 列表的循环遍历



- 1. 使用for循环
- 为了更有效率的输出列表的每个数据，可以使用循环来完成
- demo:

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

- 结果

```
xiaoWang  
xiaoZhang  
xiaoHua
```


- 2. 使用while循环
- 为了更有效率的输出列表的每个数据，可以使用循环来完成
- demo:

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

length = len(namesList)

i = 0

while i<length:
    print(namesList[i])
    i+=1
```

- 结果

```
xiaoWang
xiaoZhang
xiaoHua
```



3.6 列表的常用操作



- 列表的相关操作
- 列表中存放的数据是可以进行修改的，比如"增"、"删"、"改"“
- <1>添加元素("增"append, extend, insert)
- **append**
- 通过append可以向列表添加元素
- demo:

```
#定义变量A，默认有3个元素
A = ['xiaoWang', 'xiaoZhang', 'xiaoHua']

print("-----添加之前，列表A的数据-----")
for tempName in A:
    print(tempName)

#提示、并添加元素
temp = input('请输入要添加的学生姓名:')
A.append(temp)

print("-----添加之后，列表A的数据-----")
for tempName in A:
    print(tempName)
```

结果:

```
-----添加之前，列表A的数据-----
xiaoWang
xiaoZhang
xiaoHua
请输入要添加的学生姓名:dongGehahahaha
-----添加之后，列表A的数据-----
xiaoWang
xiaoZhang
xiaoHua
dongGehahahaha
```

- **extend**
- 通过**extend**可以将另一个集合中的元素逐一添加到列表中

```
>>> a = [1, 2]
>>> b = [3, 4]
>>> a.append(b)
>>> a
[1, 2, [3, 4]]
>>> a.extend(b)
>>> a
[1, 2, [3, 4], 3, 4]
```

- **insert**
- **insert(index, object)** 在指定位置index前插入元素object

```
>>> a = [0, 1, 2]
>>> a.insert(1, 3)
>>> a
[0, 3, 1, 2]
```

- <2>修改元素("改")
- 修改元素的时候，要通过下标来确定要修改的是哪个元素，然后才能进行修改
- demo:

```
#定义变量A，默认有3个元素
A = ['xiaoWang','xiaoZhang','xiaoHua']

print("-----修改之前，列表A的数据-----")
for tempName in A:
    print(tempName)

#修改元素
A[1] = 'xiaoLu'

print("-----修改之后，列表A的数据-----")
for tempName in A:
    print(tempName)
```

结果:

```
-----修改之前，列表A的数据-----
xiaoWang
xiaoZhang
xiaoHua
-----修改之后，列表A的数据-----
xiaoWang
xiaoLu
xiaoHua
```

- <3>查找元素("查"in, not in, index, count)
- 所谓的查找，就是看看指定的元素是否存在
- in, not in
- python中查找的常用方法为：
- 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
- demo

```
#待查找的列表
nameList = ['xiaoWang','xiaoZhang','xiaoHua']

#获取用户要查找的名字
findName = input('请输入要查找的姓名:')

#查找是否存在
if findName in nameList:
    print('在字典中找到了相同的名字')
else:
    print('没有找到')
```

结果1:(找到)

请输入要查找的姓名:xiaoWang
在字典中找到了相同的名字

结果2:(没有找到)

请输入要查找的姓名:xiaowanghaha
没有找到

- **index, count**
- index和count与字符串中的用法相同

```
>>> a = ['a', 'b', 'c', 'a', 'b']
>>> a.index('a', 1, 3) # 注意是左闭右开区间
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 'a' is not in list
>>> a.index('a', 1, 4)
3
>>> a.count('b')
2
>>> a.count('d')
0
```

- <4>删除元素("删"**del, pop, remove**)
- 类比现实生活中，如果某位同学调班了，那么就应该把这个条走后的学生的姓名删除掉；在开发中经常会用到删除这种功能。
- 列表元素的常用删除方法有：
- **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
- **demo:(del)**

```
movieName = ['加勒比海盗','骇客帝国','第一滴血','指环王','霍比特人','速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

del movieName[2]

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```


- **demo:(del)**

```
movieName = ['加勒比海盗', '骇客帝国', '第一滴血', '指环王', '霍比特人', '速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

del movieName[2]

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```

结果:

```
----- 删除之前 -----
加勒比海盗
骇客帝国
第一滴血
指环王
霍比特人
速度与激情
----- 删除之后 -----
加勒比海盗
骇客帝国
指环王
霍比特人
速度与激情
```

- **demo:(pop)** pop也可以根据下标删除元素

```
movieName = ['加勒比海盗', '骇客帝国', '第一滴血', '指环王', '霍比特人', '速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

movieName.pop()

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```

结果:

----- 删除之前 -----

加勒比海盗

骇客帝国

第一滴血

指环王

霍比特人

速度与激情

----- 删除之后 -----

加勒比海盗

骇客帝国

第一滴血

指环王

霍比特人

- **demo:(remove)**

```
movieName = ['加勒比海盗', '骇客帝国', '第一滴血', '指环王', '霍比特人', '速度与激情']

print('-----删除之前-----')
for tempName in movieName:
    print(tempName)

movieName.remove('指环王')

print('-----删除之后-----')
for tempName in movieName:
    print(tempName)
```

结果:

```
----- 删除之前 -----
加勒比海盗
骇客帝国
第一滴血
指环王
霍比特人
速度与激情
----- 删除之后 -----
加勒比海盗
骇客帝国
第一滴血
霍比特人
速度与激情
```

- <5>排序(sort, reverse)
- sort方法是将list按特定顺序重新排列，默认为由小到大，参数reverse=True可改为倒序，由大到小。
- reverse方法是将list逆置。

```
>>> a = [1, 4, 2, 3]
>>> a
[1, 4, 2, 3]
>>> a.reverse()
>>> a
[3, 2, 4, 1]
>>> a.sort()
>>> a
[1, 2, 3, 4]
>>> a.sort(reverse=True)
>>> a
[4, 3, 2, 1]
```



3.7 列表的嵌套



- 1. 列表嵌套
- 类似while循环的嵌套，列表也是支持嵌套的
- 一个列表中的元素又是一个列表，那么这就是列表的嵌套

```
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']]
```

- 2. 应用
- 一个学校，有3个办公室，现在有8位老师等待工位的分配，请编写程序，完成随机的分配

```
#encoding=utf-8

import random

# 定义一个列表用来保存3个办公室
offices = [[],[],[ ]]

# 定义一个列表用来存储8位老师的名字
names = ['A','B','C','D','E','F','G','H']

i = 0
for name in names:
    index = random.randint(0,2)
    offices[index].append(name)

i = 1
for tempNames in offices:
    print('办公室%d的人数为:%d'%(i,len(tempNames)))
    i+=1
    for name in tempNames:
        print("%s"%name,end='')
    print("\n")
    print("-"*20)
```

运行结果如下:

办公室 1 的人数为 :4
ABCE

办公室 2 的人数为 :3
DGH

办公室 3 的人数为 :1
F



3.8 元组



- 元组
- Python的元组与列表类似，不同之处在于元组的元素不能修改。元组使用小括号，列表使用方括号。

```
>>> aTuple = ('et',77,99.9)
>>> aTuple
('et',77,99.9)
```

- <1>访问元组

```
>>> tuple=('hello',100,3.14)
>>> tuple[0]
'hello'
>>> tuple[1]
100
>>> tuple[2]
3.14
>>> █
```

- <2>修改元组

```
[>>> tuple[2]=188
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> █
```

说明：**python**中不允许修改元组的数据，包括不能删除其中的元素。

- <3>元组的内置函数count, index
- index和count与字符串和列表中的用法相同

```
>>> a = ('a', 'b', 'c', 'a', 'b')
>>> a.index('a', 1, 3) # 注意是左闭右开区间
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: tuple.index(x): x not in tuple
>>> a.index('a', 1, 4)
3
>>> a.count('b')
2
>>> a.count('d')
0
```



3.9 字典介绍



- 列表介绍
- 学生信息列表，每个学生信息包括学号、姓名、年龄等，如何从中找到某个学生的信息？

```
>>> studens = [[1001, "王宝强", 24], [1002, "马蓉", 23], [1005, "宋喆", 24], ...]
```

字典

- <1>生活中的字典



- <2>软件开发中的字典
- 变量info为字典类型：

```
info = {'name':'班长', 'id':100, 'sex':'f', 'address':'地球亚洲中国北京'}
```

- 说明：
- 字典和列表一样，也能够存储多个数据
- 列表中找到某个元素时，是根据下标进行的
- 字典中找到某个元素时，是根据'名字'（就是冒号:前面的那个值，例如上面代码中的'name'、'id'、'sex'）
- 字典的每个元素由2部分组成，键:值。例如 'name':'班长', 'name'为键，'班长'为值

- <3>根据键访问值

```
info = {'name': '班长', 'id': 100, 'sex': 'f', 'address': '地球亚洲中国北京'}

print(info['name'])
print(info['address'])
```

- 结果:

```
班长
地球亚洲中国北京
```

- 若访问不存在的键，则会报错:

```
>>> info['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```


- 在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：

```
>>> age = info.get('age')
>>> age # 'age'键不存在，所以age为None
>>> type(age)
<type 'NoneType'>
>>> age = info.get('age', 18) # 若info中不存在'age'这个键，就返回默认值18
>>> age
18
```



3.9 字典常用操作1



- <1>修改元素
- 字典的每个元素中的数据是可以修改的，只要通过key找到，即可修改

- demo:

```
info = {'name':'班长', 'id':100, 'sex':'f', 'address':'地球亚洲中国北京'}
```

```
newId = input('请输入新的学号')
```

```
info['id'] = int(newId)
```

```
print('修改之后的id为%d:%s'%info['id'])
```

- 结果:

```
请输入新的学号 88  
修改之后的 id为： 88
```

- <2>添加元素
- demo:访问不存在的元素

```
info = {'name':'班长', 'sex':'f', 'address':'地球亚洲中国北京'}  
  
print('id为:%d'%info['id'])
```

结果:

```
MacBook-Pro 01-python基础班-资料$ python test.py  
File "test.py", line 10  
    print 'id为:' info['id']  
                  ^  
SyntaxError: invalid syntax
```

- 如果在使用 变量名['键']=数据 时，这个“键”在字典中，不存在，那么就会新增这个元素

- demo:添加新的元素

```
info = {'name':'班长', 'sex':'f', 'address':'地球亚洲中国北京'}

# print('id为:%d'%info['id'])#程序会终端运行，因为访问了不存在的键

newId = input('请输入新的学号')

info['id'] = newId

print('添加之后的id为:%d'%info['id'])
```

- 结果:

```
请输入新的学号188
添加之后的id为: 188
```

- <3>删除元素
- 对字典进行删除操作，有以下几种：
- del
- pop
- demo:del删除指定的元素

```
info = {'name': '班长', 'sex': 'f', 'address': '地球亚洲中国北京'}

print('删除前,%s'%info['name'])

del info['name']

print('删除后,%s'%info['name'])
```

结果

```
MacBook-Pro 01-python基础班-资料$ python test.py
删除前, 班长
删除后,
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print '删除后',info['name']
KeyError: 'name'
```

删除后不
能访问

- pop: 根据key值删除数据
- dic.pop(key)



3.9 字典常用操作2



- **<1>len()**

测量字典中，键值对的个数

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> len(dict)  
2  
>>> █
```

- **<2>keys**

返回一个包含字典所有KEY的列表

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> dict.keys()  
['name', 'sex']  
>>> █
```

- **<3>values**

返回一个包含字典所有value的列表

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> dict.values()  
['zhangsan', 'm']  
>>> █
```

- **<4>items**

返回一个包含所有 (键，值) 元祖的列表

```
[>>> dict = {"name": 'zhangsan', 'sex': 'm'}  
[>>> dict.items()  
[('name', 'zhangsan'), ('sex', 'm')]  
>>> █
```



3. 10 字典的遍历



- 遍历
- 通过for ... in ...:的语法结构，我们可以遍历字符串、列表、元组、字典等数据结构。
- 注意python语法的缩进

- 字符串遍历

```
>>> a_str = "hello itcast"
>>> for char in a_str:
...     print(char,end=' ')
...
h e l l o   i t c a s t
```

- 列表遍历

```
>>> a_list = [1, 2, 3, 4, 5]
>>> for num in a_list:
...     print(num,end=' ')
...
1 2 3 4 5
```

- 元组遍历

```
>>> a_tuple = (1, 2, 3, 4, 5)
>>> for num in a_tuple:
...     print(num,end=" ")
1 2 3 4 5
```

- 字典遍历
- <1> 遍历字典的key（键）
- <2> 遍历字典的value（值）
- <3> 遍历字典的项（元素）
- <4> 遍历字典的key-value（键值对）

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for key in dict.keys():
...     print key
...
name
sex
>>>
```

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for value in dict.values():
...     print value
...
zhangsan
m
>>>
```

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for item in dict.items():
...     print item
...
('name', 'zhangsan')
('sex', 'm')
>>>
```

```
>>> dict = {'name': 'zhangsan', 'sex': 'm'}
>>> for key, value in dict.items():
...     print("key=%s, value=%s"%(key, value))
...
key=name, value=zhangsan
key=sex, value=m
>>>
```



3. 11 附加



- **python**内置函数
- Python包含了以下内置函数

序号	方法	描述
1	cmp(item1, item2)	比较两个值
2	len(item)	计算容器中元素个数
3	max(item)	返回容器中元素最大值
4	min(item)	返回容器中元素最小值
5	del(item)	删除变量

- 公共方法
- 运算符

运算符	Python 表达式	结果	描述	支持的数据类型
+	[1, 2] + [3, 4]	[1, 2, 3, 4]	合并	字符串、列表、元组
*	'Hi!' * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	复制	字符串、列表、元组
in	3 in (1, 2, 3)	True	元素是否存在	字符串、列表、元组、字典
not in	4 not in (1, 2, 3)	True	元素是否不存在	字符串、列表、元组、字典

- 多维列表/元祖访问的示例

```
>>> tuple1 = [(2,3),(4,5)]
>>> tuple1[0]
(2, 3)
>>> tuple1[0][0]
2
>>> tuple1[0][2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
>>> tuple1[0][1]
3
>>> tuple1[2][2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> tuple2 = tuple1+[(3)]
>>> tuple2
[(2, 3), (4, 5), 3]
>>> tuple2[2]
3
>>> tuple2[2][0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not subscriptable
```




3.12 总结 & 实践环节

