Python数据容器

一、课前准备

1. 完成上节课的作业并提交；
2. 根据课程大纲提前复习Python基础语法；

二、课堂主题

本小节主要学习Python的基础，比如：列表、元组、字典、集合等，Python数据容器的基础知识点。

三、课堂目标

1. 掌握Python数据容器的的创建和使用；
2. 各个容器对数据的存储方式和特点。

四、知识要点

1. 字符串（str）

1.1 字符串的定义

我们在介绍数据类型的时候,简单介绍了一下字符串类型.因为字符串是Python语言中特别重要的概念(不仅仅是Python,在其他语言中也有着举重若轻的位置), 我们详细的讲解一下字符串的用法。

我们已经知道了,单引号,双引号,包括三引号包围的字符组,就是字符串,例如

```
str = 'hello'#定义字符串变量
str = "world"#定义字符串变量
str = """hello
       Python"""#定义多行字符串变量
```

1.2 下标和切片

1. 下标索引

下标在Python中的概念就是编号的意思,`字符串` `元组` `列表`都会经常用到下标的概念,我们可以**根据下标找到它们所对应的元素**.就好像生活中你要准备去看电影,电影票上的座位号找到对应的**位置.**

我们现在考虑这样的一个问题,例如我们创建了一个字符串`name = zhang`,我现在想取到名为`name`字符串里面的a字符.如何去取呢?,其实我们可以通过我们讲过的for循环来遍历这个字符串,通过这种方法来取出字符串里的字符,但是`Python`给我们提供了更为简便的方法,我们就可以用下标来取出`a`字符

```
    name = 'zhangsan'

    print(name[2])
```

运行结果:

```
a
```

2. 切片

我们可以利用下表索引取到字符串里面对应的一个元素,但如果想要截取一段元素就要用到切片片.

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]

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

运行结果:
abc
name = 'abcdef'

print(name[3:5]) # 取 下标为3、4 的字符

运行结果 :
de
name = 'abcdef'

print(name[2:]) # 取 下标为2开始到最后的字符

运行结果:
cdef
```

支持负数

```
name = 'abcdef'

print(name[1:-1]) # 取 下标为1开始 到 最后第2个  之间的字符

运行结果:
bcde
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'
```

1.3 字符串的常见操作

如有字符串`mystr = 'hello world kkb'`，以下是常见的操作

四类,增删改查

1.3.1 查

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))
```

例如:

```
mystr = 'hello world kkb'
mystr.find("kkb")
运行结果为:12
mystr = 'hello world kkb'
mystr.find("kkb",0,10) #在mstr字符串0-10下标范围查询
运行结果:-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))
```

例如:

```
mystr = 'hello world kkb'
mystr.index("ab")
运行结果:控制台会直接报错(Vale Error: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))
```

例如:

```
mystr = '   hello world kkb and kkb   '
mystr.count('kkb')
运行结果:2
```

4. startswith

检查字符串是否是以 '`hello`'开头, 是则返回 `True`，否则返回 `False`

```
mystr.startswith('hello')
```

5. endswith

检查字符串是否以`obj`结束，如果是返回`True`,否则返回 `False`.

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

6. 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) )
```

7. rindex

类似于 `index()`，不过是从右边开始.

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

1.3.2 增

1. join

`mystr` 中每个元素后面插入`str`,构造出一个新的字符串

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

1.3.3 删

1. lstrip

删除 `mystr` 左边的空白字符

```
mystr.lstrip()
```

2. rstrip

删除 `mystr` 字符串末尾的空白字符

```
mystr.rstrip()
```

3. strip

删除 `mystr` 字符串两端的空白字符

```
 a = "\n\t kkb \t\n"
 a.strip()
 运行结果:
'kkb'
```

1.3.4 改

1. 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))
```

2. 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)
```

3、capitalize

把字符串的第一个字符大写

```
mystr.capitalize()
```

4. title

```
a = "hello kkb"
a.title()
运行结果
'Hello Kkb'
```

5. lower

转换 `mystr` 中所有大写字符为小写

```
mystr.lower()
```

6. upper

转换 `mystr` 中的小写字母为大写

```
mystr.upper()
```

7. ljust

返回一个原字符串左对齐,并使用空格填充至长度 `width` 的新字符串

```
mystr.ljust(width)
```

8. rjust

返回一个原字符串右对齐,并使用空格填充至长度 `width` 的新字符串

```
mystr.rjust(width)
```

9. center

返回一个原字符串居中,并使用空格填充至长度 `width` 的新字符串

```
mystr.center(width)
```

10. partition

把`mystr`以`str`分割成三部分,`str`前，`str`和`str`后

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

11. rpartition

类似于 partition()函数,不过是从右边开始.

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

12. splitlines

按照行分隔，返回一个包含各行作为元素的列表

```
mystr.splitlines()
```

2、列表（list）

2.1 列表简介

> Python内置的一种数据类型是**列表**：`list`。`list`是一种**有序的集合**，可以随时**添加和删除**其中的元素，写在方括号之间、用逗号分隔开的数值列表。列表内的项目不必全是相同的类型。

列如：

```
list1 = ['Mike', '张三', 25000, 99.99, True]
```

注意：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。

2.2 列表的常见操作

1. 列表的长度

```
#用len()函数可以获得list元素的个数：
namesList = ['xiaoWang','xiaoZhang','xiaoHua']
len(namesList)
```

2. 列表的访问

用索引来访问`list`中每一个位置的元素，记得**索引是从0开始的**：

```
namesList = ['Tony','Rose','Lucy']
print(namesList[0])
print(namesList[1])
print(namesList[2])
print(namesList[3])

 结果：
Tony
Rose
Lucy
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-54-6f5bf781f8ad> in <module>
      3 print(namesList[1])
      4 print(namesList[2])
----> 5 print(namesList[3])

IndexError: list index out of range
```

注意：当索引**超出**了范围时，Python会报一个`IndexError`错误，所以，要确保索引不要越界，记得最后一个元素的索引是`len(classmates) - 1`。

如果要取最后一个元素，除了计算索引位置外，还可以用**-1**做索引，直接获取**最后一个**元素：

```
namesList = ['Tony','Rose','Lucy']
print(namesList[-1])
结果：
Lucy
```

以此类推，可以获取倒数**第2个**、倒数**第3个**：

```
namesList = ['Tony','Rose','Lucy']
print(namesList[-1])
print(namesList[-2])
print(namesList[-3])
结果：
Lucy
Rose
Tony
```

3. 列表的切片

切片: 根据下标的范围获取一部分数据，比如: 列表，字符串可以使用切片。

```
切片的使用格式
数据[起始下标:结束下标:步长]
提示： 起始下标默认0， 结束下标是不包含， 步长默认是1
```

```
# 使用切片的方式获取一部分数据
my_str = ['Mike', '张三', 25000, 99.99, True]
result = my_str[1:4:1]
print(result)

#前三个
result = my_str[0:3]
print(result)
result = my_str[:3]
print(result)
```

4. 添加元素(`append`, `extend`, `insert`)

通过`append`可以向列表添加元素

```
#定义变量A，默认有3个元素
namesListA = ['ZhangSan','LiSi','WangWu']

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

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

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

通过`extend`可以将另一个集合中的元素逐一添加到列表中

```
#append:
a = [1, 2, 3]
b = [4, 5, 6]
a.append(b)
a
结果:
[1, 2, 3, [4, 5, 6]]

#extend:
a = [1, 2, 3]
b = [4, 5, 6]
a.extend(b)
a
结果:
[1, 2, 3, 4, 5, 6]

#insert
```

```
insert(index, object)` 在指定位置`index`前插入元素`object
a = [1, 2, 3]
a.insert(1,100)
a
结果:
[1, 100, 2, 3]
```

5. 修改元素

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

```
#定义变量namesListA，默认有4个元素
namesListA = ['ZhangSan','LiSi','WangWu','ZhaoLiu']

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

#修改元素
namesListA[1] = '张三'

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


结果:
-----修改之前，列表A的数据-----
ZhangSan
LiSi
WangWu
ZhaoLiu
-----修改之后，列表A的数据-----
ZhangSan
张三
WangWu
ZhaoLiu
```

6. 查找元素

所谓的**查找**，就是看看指定的元素**是否存在。**

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`

```
#待查找的列表
namesListA = ['ZhangSan','LiSi','WangWu','ZhaoLiu']

#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 namesListA:
    print('在列表中找到了相同的名字')
else:
    print('没有找到')
```

`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
```

7. 删除元素

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

(1) del

```
list1 = ['a','b','c','d','e','f']

    print('------删除之前------')
    for tempName in list1:
        print(tempName)

    del list1[2]

    print('------删除之后------')
    for tempName in list1:
        print(tempName)
结果:

    ------删除之前------
    a
    b
    c
    d
    e
    f
```

```
    ------删除之后------
    a
    b
    d
    e
    f
```

(2) pop

```
list2 = ['a','b','c','d','e','f']

print('------删除之前------')
for tempName in list2:
    print(tempName)

list2.pop()

print('------删除之后------')
for tempName in list2:
    print(tempName)

结果:

    ------删除之前------
    a
    b
    c
    d
    e
    f
    ------删除之后------
    a
    b
    c
    d
    e
```

(3) remove

```
list3 = ['a','b','c','d','e','f']

print('------删除之前------')
for tempName in list3:
    print(tempName)

list3.remove('e')

print('------删除之后------')
for tempName in list3:
    print(tempName)


结果:
------删除之前------
a
b
```

```
c
d
e
f
------删除之后------
a
b
c
d
f
```

(4) 排序

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

2.3 列表的遍历

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

```
namesList = ['ZhangSan','LiSi','WangWu','ZhaoLiu']
for name in namesList:
    print(name)

结果:
    ZhangSan
    LiSi
    WangWu
    ZhaoLiu
```

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

```
namesList = ['ZhangSan','LiSi','WangWu','ZhaoLiu']

length = len(namesList)

i = 0
```

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

结果:
    ZhangSan
    LiSi
    WangWu
    ZhaoLiu
```

3、元组（tuple）

另一种**有序列表叫元组**：`tuple`。`tuple`和`list`非常类似，但是`tuple`一旦初始化就**不能修改**，比如同样是列出同学的名字：

```
>>> classmates = ('Michael', 'Bob', 'Tracy')
```

现在，`classmates`这个`tuple`不能变了，它也**没有**`append()`，`insert()`这样的方法。其他获取元素的方法和list是一样的，你**可以正常地使用**`classmates[0]`，`classmates[-1]`，但不能赋值成另外的元素。

不可变的tuple有什么**意义**？因为tuple不可变，所以代码更**安全**。如果可能，能用`tuple`代替`list`就尽量用`tuple`。

如果要定义一个空的`tuple`，可以写成`()`：

```
>>> t = ()
>>> t
()
```

但是，要定义一个只有1个元素的`tuple`，如果你这么定义：

```
>>> t = (1)
>>> t
1
```

定义的**不是**tuple，是1这个数！这是因为括号()既可以表示tuple，又可以表示**数学公式中的小括号**，这就产生了歧义，因此，Python规定，这种情况下，按小括号进行计算，计算结果自然是1。

所以，只有1个元素的tuple定义时**必须加一个逗号**，来消除歧义：

```
>>> t = (1,)
>>> t
(1,)
```

Python在显示只有1个元素的`tuple`时，也会加一个逗号，以免你误解成数学计算意义上的括号。

最后来看一个“**可变的**”tuple：

```
>>> t = ('a', 'b', ['A', 'B'])
>>> t[2][0] = 'X'
>>> t[2][1] = 'Y'
>>> t
('a', 'b', ['X', 'Y'])
```

4、字典（dict）

4.1 字典简介

字典是另一种可变容器模型，且可存储任意类型对象。

字典的每个键值(key=>value)对用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号{}中

举个例子，假设要根据同学的名字查找对应的成绩，如果用`list`实现，需要两个`list`：

```
names = ['Michael', 'Bob', 'Tracy']
scores = [95, 75, 85]
```

给定一个名字，要查找对应的成绩，就先要在`names`中找到对应的位置，再从`scores`取出对应的成绩，`list`**越长**，**耗时**越长。

如果用`dict`实现，只需要一个**“名字”-“成绩”**的对照表，直接根据名字查找成绩，无论这个表有多大，查找速度都不会变慢。用Python写一个`dict`如下：

```
>>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}
>>> d['Michael']
95
```

由于一个`key`只能对应一个`value`，所以，多次对一个`key`放入`value`，**后面的值会把前面的值冲掉**：

```
>>> d['Jack'] = 90
>>> d['Jack']
90
>>> d['Jack'] = 88
>>> d['Jack']
88
```

如果key不存在，dict就会报错：

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

4.2 字典的常见操作

（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

```
info = {'name':'kkb', 'id':100, 'sex':'f', 'address':'中国北京'}

new_id = input('请输入新的学号:')

info['id'] = int(new_id)

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

(2) 添加元素

```
访问不存在的元素
info = {'name':'kkb', 'sex':'f', 'address':'中国北京'}

print('id为:%d' % info['id'])
结果:

>>> info = {'name':'kkb', 'sex':'f', 'address':'中国北京'}
>>>
>>> print('id为:%d' % info['id'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'id'
```

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。**

添加新的元素

```
info = {'name':'kkb', 'sex':'f', 'address':'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，有一下几种：

- `del`
- `clear()`

`del`删除**指定的元素**

```
info = {'name':'kkb', 'sex':'f', 'address':'中国北京'}

print('删除前,%s' % info['name'])

del info['name']
```

```
print('删除后,%s' % info['name'])
结果
>>> info = {'name':'kkb', 'sex':'f', 'address':'中国北京'}
>>>
>>> print('删除前,%s' % info['name'])
删除前,kkb
>>>
>>> del info['name']
>>>
>>> print('删除后,%s' % info['name'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'name'
```

del删除整个字典

```
info = {'name':'monitor', 'sex':'f', 'address':'China'}

print('删除前,%s' % info)

del info

print('删除后,%s' % info)
```

clear清空整个字典

```
info = {'name':'monitor', 'sex':'f', 'address':'China'}

print('清空前,%s' % info)

info.clear()

print('清空后,%s' % info)
```

(4) len()

测量字典中，**键值对的个数**

```
d1 = {'name':'abc','age':'18', 'class':'cnh'}
print(len(d1))
结果:
3
```

(5) keys

返回一个包含字典**所有key**的列表

```
d1 = {'name':'abc','age':'18', 'class':'cnh'}
print(list(d1.keys()))

结果:
['name', 'age', 'class']
```

(6) values

返回一个包含字典**所有value**的列表

```
d1 = {'name':'abc','age':'18', 'class':'cnh'}
print(list(d1.values()))

结果:
['abc', '18', 'cnh']
```

(7) items

返回一个包含**所有（键，值）**元祖的列表

```
d1 = {'name':'abc','age':'18', 'class':'cnh'}
print(list(d1.items()))

结果:
[('name', 'abc'), ('age', '18'), ('class', 'cnh')]
```

(8) has_key （Python3 已取消）

`dict.has_key(key)` 如果key在字典中，返回 `True`，否则返回 `False`

5、集合（set）

集合（set）是一个无序的不重复元素序列。

可以使用大括号 **{ }** 或者 **set()** 函数创建集合，注意：创建一个空集合必须用 **set()** 而不是 **{ }**，因为 **{ }** 是用来创建一个空字典。

```
my_set = {1, 4, 'abc', 'hello'}
# 不支持下标赋值和取值
# my_set[0] = 3
# value = my_set[0]
# print(value)

#通过遍历获取数据
my_set = {1, 5, 7}
for value in my_set:
    print(value)

for index,value in enumerate(my_set):
    print(index,value)
# 定义空的集合的时候不能直接使用{}

my_set = set()
my_set.add(1)
my_set.add(1)
print(my_set, type(my_set))

# 集合可以对容器类型数据去重
my_list = [1, 1, 3, 5, 3]
# 把列表转成集合，会把数据去重
my_set = set(my_list)

print(my_set)

# 列表，元组， 集合 三者之间可以相互转换
```

```
my_tuple = (5, 3)
print(my_tuple, type(my_tuple))
```

五、总结

1. 本节课的所有知识点全是重点，后面的学习离不开基础语法；
2. 需要着重掌握的是各个数据容器的创建方法和对数据的操作；
3. 所有的代码都要多敲几遍，练习是学习编程最简单的途径！！！