python基本数据结构以及用法

周亭亭 2021-11-19整理 1.0版本

一、字符串

字符串是Python中常用的表示非数值型数据的对象，其基本操作和列表相似，但是不可以进行改变，具体操作在第二部分列表会详细阐明。

1.1创建拼接复制

```
str1='hello'#定义一个字符串
str2='world'
```

```
str3=str1+" "+str2
print(str3)
```

```
hello world
```

```
print(str3*2)#字符串复制连接
```

```
hello worldhello world
```

1.2切片和索引

```
print("'m' in str3","m" in str3)#判断字符是否在字符串中
print("'e' in str3","e" in str3)
```

```
'm' in str3 False
'e' in str3 True
```

```
print("提取第三个字符",str3[2])#提取第三个字符
print("提取第四到第七个字符（左闭右开区间）",str3[3:7])#提取第四到第七个字符（左闭右开区间）
print("str1[1:2]",str1[1:2])
print(str1[-3:])#提取倒数第三个之后的所有
```

```
提取第三个字符 l
提取第四到第七个字符（左闭右开区间） lo w
str1[1:2] e
llo
```

```
print(str1[::2])#带步长的切片
```

```
hlo
```

```
print(str1[::-1])#倒着输出
print(str1[5:0:-1])#倒着输出
print(str1[4:0:-1])#倒着输出，注意这三种的区别
```

```
olleh
olle
olle
```

1.3字符串方法

center

```
ss='1q,q,q,q'
ss.center(20,"0")#表示在字符串两端填上某个字符使得达到某个长度
```

```
'0000001q,q,q,q000000'
```

find

```
ss1='7jdauig'
print("当元素存在的时候，返回索引位置",ss1.find("g"))
print("当元素不存在的时候，返回",ss1.find("z"))
```

```
当元素存在的时候，返回索引位置 6
当元素不存在的时候，返回 -1
```

join

```
#join 一般和其他序列一起使用
ss2="abcdefg"
lists2=list(ss2)
tups2=tuple(ss2)
print(ss2)
print(lists2)
print(tups2)
```

```
abcdefg
['a', 'b', 'c', 'd', 'e', 'f', 'g']
('a', 'b', 'c', 'd', 'e', 'f', 'g')
```

```
print("_".join(ss2))
print("0".join(lists2))
print("".join(tups2))
```

```
a_b_c_d_e_f_g
a0b0c0d0e0f0g
abcdefg
```

lower 和 upper

```
str4='MIninfov'
str5=str4.upper()#转成大写，有返回值，要用变量名承接
print("转成大写",str5)
str6=str4.lower()#转成小写
print('转成小写',str6)
```

```
转成大写 MININFOV
转成小写 mininfov
```

title

```
words='my name is liu'
words.title()#首字母大写
```

```
'My Name Is Liu'
```

count

```
print(str3.count("l"))#统计字符l的个数
print(str3.count("ll"))#统计字符ll的个数
str7="llellll"
print(str7.count("ll"))#统计字符串ll的个数
```

```
3
1
3
```

replace

```
print(str3.replace("e","i"))#替换操作，有返回值
print(str3)
str8=str3.replace("e","i")
print(str8)
print(str7.replace("ll","i"))
```

```
hillo world
hello world
hillo world
ieii
```

translate

```
#和replace类似，可以进行多字符替换
trans=str.maketrans("xyz",'abc')#创建转换表
sss='xyjfnszldajf'
sss.translate(trans)
```

```
'abjfnscldajf'
```

strip

```
print('去除开头的字符he',str3.strip("he"))#去除开头的字符he
print('去除末尾的字符ld',str3.strip("ld"))#去除末尾的字符ld
print('无法去除中间的字符',str3.strip('llo'))#无法去除中间的字符
```

```
去除开头的字符he llo world
去除末尾的字符ld hello wor
无法去除中间的字符 hello world
```

```
str10='  nia  hfnv  '
print('去除首尾的空格符',str10.strip())#去除首尾的空格符
```

```
去除首尾的空格符 nia  hfnv
```

split

```
ss3='minaj miahe mccs'
print(ss3.split())
print(ss3.split(" "))#都是按照空格进行切割，返回列表
```

```
['minaj', 'miahe', 'mccs']
['minaj', 'miahe', 'mccs']
```

```
print(ss3.split("m"))#用m切割
#如果要每个字符都被分割，直接用list方法
print(list(ss3))
```

```
['', 'inaj ', 'iahe ', 'ccs']
['m', 'i', 'n', 'a', 'j', ' ', 'm', 'i', 'a', 'h', 'e', ' ', 'm', 'c', 'c', 's']
```

二、列表

列表是Python中常用语表示向量的对象。

```
a='is'
b="nice"
list1=["my","list",a,b]
list2=[[1,2,3,4],[5,6,7,8]]
print(list1)
print(list2)
```

```
['my', 'list', 'is', 'nice']
[[1, 2, 3, 4], [5, 6, 7, 8]]
```

```
print('拼接后',list1+list2)#列表拼接
print('复制后',list1*3)#列表重复复制
```

```
拼接后 ['my', 'list', 'is', 'nice', [1, 2, 3, 4], [5, 6, 7, 8]]
复制后 ['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice', 'my', 'list',
'is', 'nice']
```

2.1索引

```
print('提取第二个元素',list1[1])#提取第二个元素
print('倒数第一个元素',list1[-1])#倒数第一个元素
print('倒数第二个元素',list1[-2])#倒数第二个元素
print('第一行第二个数',list2[0][1])#第一行第二个数
```

```
提取第二个元素 list
倒数第一个元素 nice
倒数第二个元素 is
第一行第二个数 2
```

2.2切片

```
print('提取第2到3个元素',list1[1:3])#提取第2到3个元素
print('第二行第2到3个数',list2[1][1:3])#第二行第2到3个数
```

```
提取第2到3个元素 ['list', 'is']
第二行第2到3个数 [6, 7]
```

```
print("从后面往前找",list1[-3:-1])#从后面往前找
```

```
从后面往前找 ['list', 'is']
```

```
print(list1[-3:0])#如果想要找倒数第三个到倒数第一个，不能这么写。
print(list1[-3:])
```

```
[]
['list', 'is', 'nice']
```

```
print(list1[:3])#相当于list1[0:3]
print(list1[:])#整个序列
```

```
['my', 'list', 'is']
['my', 'list', 'is', 'nice']
```

使用步长进行切片

```
list5=[i for i in range(10)]
print(list5)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
print('步长为1的切片',list5[0:10:1])#和不用步长没有区别
print("步长为2的切片",list5[0:10:2])#相当于每隔一个数取一个值
print("步长为2的切片",list5[::2])
print("步长为4的切片",list5[::4])#表示在进行切片的时候从0位开始一直找到末位结束
```

```
步长为1的切片 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
步长为2的切片 [0, 2, 4, 6, 8]
步长为2的切片 [0, 2, 4, 6, 8]
步长为4的切片 [0, 4, 8]
```

```
#步长不能为0 但是可以为负数，表示从右往左进行移动,步长为负数时，第一个索引必须比第二个索引大。
print('步长为-1的切片(错误)',list5[0:10:-1])
print('步长为-1的切片',list5[10:0:-1])
print('步长为-1的切片',list5[9:0:-1])#表示从9到1，同样是左开右闭区间。
print('步长为-1的切片',list5[::-1])#注意这三种的区别
print("步长为-2的切片（错误）",list5[3:8:-2])
print("步长为-2的切片",list5[8:3:-2])
```

```
步长为-1的切片(错误) []
步长为-1的切片 [9, 8, 7, 6, 5, 4, 3, 2, 1]
步长为-1的切片 [9, 8, 7, 6, 5, 4, 3, 2, 1]
步长为-1的切片 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
步长为-2的切片（错误） []
步长为-2的切片 [8, 6, 4]
```

2.3修改元素

```
list1=["my","list",a,b]
list5=[i for i in range(10)]
print(list1)
print(list5)
```

```
['my', 'list', 'is', 'nice']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
list1[0]=1#修改列表的值
print(list1)
```

```
[1, 'list', 'is', 'nice']
```

```
list5[0:4]=[2,4,1,5]
print(list5)#修改切片的值
```

```
[2, 4, 1, 5, 4, 5, 6, 7, 8, 9]
```

```
list1[1:1]=[1,41,4]#使用切片方式还可以在不替换的情况下进行插入操作
print(list1)
```

```
[1, 1, 41, 4, 'list', 'is', 'nice']
```

```
list1[1:3]=list("fjig")#使用切片还可以将原位置替换为不同长度的元素
print(list1)
```

```
[1, 'f', 'j', 'i', 'g', 4, 'list', 'is', 'nice']
```

```
del list1[1:5]#删除元素（切片、索引）
print(list1)
```

```
[1, 4, 'list', 'is', 'nice']
```

2.4列表函数

```
list1=["my","list",a,b]
list5=[i for i in range(10)]
```

```
print("列表长度",len(list1))#列表长度
print('最大值',max(list5))
print("最小值",min(list5))
print("相加值",sum(list5))
print("列表排序",sorted(list1))
print('判断is是否在list1中',"is" in list1)#判断is是否在list1中
```

```
列表长度 4
最大值 9
最小值 0
相加值 45
列表排序 ['is', 'list', 'my', 'nice']
判断is是否在list1中 True
```

2.5列表方法

```
list4=['af','a',' ',3,1,1]
print(list4)
list1=["my","list",a,b]
print(list1)
```

```
['af', 'a', ' ', 3, 1, 1]
['my', 'list', 'is', 'nice']
```

append

```
list1.append("i")#列表后加值，无返回值，改变原列表
print(list1)
```

```
['my', 'list', 'is', 'nice', 'i']
```

extend

```
list4=['af','a',' ',3,1,1]
list4.append(list1)
print(list4)
```

```
['af', 'a', ' ', 3, 1, 1, ['my', 'list', 'is', 'nice', 'i']]
```

```
list4=['af','a',' ',3,1,1]
list4.extend(list1)
print(list4)
```

```
['af', 'a', ' ', 3, 1, 1, 'my', 'list', 'is', 'nice', 'i']
```

copy

```
list1_1=list1.copy()#进行拷贝，这样在改变list1_1的同时，list1不会被更改
print(list1_1)
list1_1[1]='aa'
print(list1_1)
print(list1)
```

```
['my', 'list', 'is', 'nice', 'i']
['my', 'aa', 'is', 'nice', 'i']
['my', 'list', 'is', 'nice', 'i']
```

```
list10=[[1,2],[12,4]]
list10_1=list10.copy()
list10_1[1]='aa'
print(list10)
print(list10_1)#浅拷贝在上一维度中有效
```

```
[[1, 2], [12, 4]]
[[1, 2], 'aa']
```

```
list10=[[1,2],[12,4]]
list10_1=list10.copy()
list10_1[1][1]='aa'
print(list10)
print(list10_1)#浅拷贝在二维列表中失效
```

```
[[1, 2], [12, 'aa']]
[[1, 2], [12, 'aa']]
```

```
from copy import deepcopy#使用深拷贝
list10=[[1,2],[12,4]]
list10_1=deepcopy(list10)
list10_1[1][1]='aa'
print(list10)
print(list10_1)#深拷贝有效
```

```
[[1, 2], [12, 4]]
[[1, 2], [12, 'aa']]
```

index

```
list4=['af','a',' ',3,1,1]
```

```
print('寻找元素1的位置',list4.index(1))#寻找元素位置(只能找到最先出现的位置)
```

```
寻找元素1的位置 4
```

count

```
list4=['af','a',' ',3,1,1]
print(list4)
```

```
['af', 'a', ' ', 3, 1, 1]
```

```
print('计算元素个数',list4.count('is'))#计算元素个数
```

```
计算元素个数 0
```

```
print('计算元素个数',list4.count(1))#计算元素个数
```

```
计算元素个数 2
```

clear

```
list4.clear()#清除列表内容
print(list4)
```

```
[]
```

remove

```
list1=["my","list",a,b]
print(list1)
```

```
['my', 'list', 'is', 'nice']
```

```
list1.remove("is")#删除某个值，无返回值，改变原列表
print(list1)
```

```
['my', 'list', 'nice']
```

reverse

```
list1=["my","list",a,b]
print(list1)
list1.reverse()
print(list1)
```

```
['my', 'list', 'is', 'nice']
['nice', 'is', 'list', 'my']
```

insert

```
list1=["my","list",a,b]
print(list1)
list1.insert(0,'!')#在指定位置加元素，无返回值，会改变原列表
print(list1)
list1.insert(5,6)
print(list1)
```

```
['my', 'list', 'is', 'nice']
['!', 'my', 'list', 'is', 'nice']
['!', 'my', 'list', 'is', 'nice', 6]
```

pop

```
list1=["my","list",a,b]
print(list1)
x=list1.pop()#删除最后一个值，有返回值的同时改变原列表的值,如果要删除某个位置的值，应该在pop()里加上索引
print(x)#返回最后一个值
print(list1)
```

```
['my', 'list', 'is', 'nice']
nice
['my', 'list', 'is']
```

```
x=list1.pop(2)
print(x)
print(list1)
```

```
is
['my', 'list']
```

sort

```
list3=[1,3,1,61,6,1]
list3.sort()
print(list3)#升序排列
```

```
[1, 1, 1, 3, 6, 61]
```

```
list3=[1,3,1,61,6,1]
list3.sort(reverse=True)
print(list3)#降序排列
```

```
[61, 6, 3, 1, 1, 1]
```

```
list6=[[13,1],[0,1],[10,5]]
list6.sort(key=lambda x:x[1],reverse=True)
print(list6)#key表示某个规则，表示以什么为标准进行排序，这里表示以第二个数值的大小进行排序
```

```
[[10, 5], [13, 1], [0, 1]]
```

```
list6=[[13,1],[0,1],[10,5]]
list6.sort(key=sum,reverse=True)
print(list6)#以数组相加的大小进行排序
```

```
[[10, 5], [13, 1], [0, 1]]
```

```
list7=[['afaf','11',2],['fAA'],['minar']]
list7.sort(reverse=True)
print(list7)
```

```
[['minar'], ['fAA'], ['afaf', '11', 2]]
```

```
list7=[['afaf','11',2],['fAA'],['minar']]
list7.sort(key=lambda x:len(x[0]))
print(list7)#表示按照每个内层列表的第一个元素的长度进行排序
```

```
[['fAA'], ['afaf', '11', 2], ['minar']]
```

三、元组

元组与列表类似，但是不能对数据进行增删改

```
tup1=(1,2,2,3)
tup2=('a','b',1,2,3)
tup3=1,2,4
print(tup1)
print(tup2)
print(tup3)
```

```
(1, 2, 2, 3)
('a', 'b', 1, 2, 3)
(1, 2, 4)
```

```
tup4=(50)#只有一个元素的时候，这样写会将括号显示称为运算符
tup5=(50,)#这样写才会识别为元组
print(tup4)
print(tup5)
```

```
50
(50,)
```

```
print(type(tup4))
print(type(tup5))
```

```
<class 'int'>
<class 'tuple'>
```

```
del tup1 #可以删除整个元组
```

```
list1=list(tup2)#元组转化为列表
tup21=tuple(list1)#列表转化为元组
print(list1)
print(tup21)
```

```
['a', 'b', 1, 2, 3]
('a', 'b', 1, 2, 3)
```

四、字典

4.1函数dict

```
items=[("name","liuchen"),("id1",111),("age",18)]
```

```
d=dict(items)
print(d)
```

```
{'name': 'liuchen', 'id1': 111, 'age': 18}
```

```
print(d["name"])
print(d["id1"])
```

```
liuchen
111
```

```
d2=dict(name="zh",id1=121,age=18)
print(d2)
```

```
{'name': 'zh', 'id1': 121, 'age': 18}
```

```
d3={"w":1,"e":2}
print(d3)
```

```
{'w': 1, 'e': 2}
```

4.2字典基本操作

```
d=dict(items)
len(d)
```

```
3
```

```
d["name"]
```

```
'liuchen'
```

```
d["name"]="liulage"#修改键对应的值
print(d)
```

```
{'name': 'liulage', 'id1': 111, 'age': 18}
```

```
del d["id1"]#删除键代表的项
print(d)
```

```
{'name': 'liulage', 'age': 18}
```

```
print("name" in d)
print("school" in d)#查看字典中是否有某个键
```

```
True
False
```

```
d["scll"]='swaaa'#添加键值对
print(d)
```

```
{'name': 'liulage', 'age': 18, 'scll': 'swaaa'}
```

4.3字典方法

clear

```
d2=dict(name="zh",id1=121,age=18)
d2.clear()
print(d2)
```

```
{}
```

copy

```
print(d)
d1_1=d.copy()#进行浅拷贝，这样子可以在对d1_1进行修改的时候不改变d
print(d1_1)
```

```
{'name': 'liulage', 'age': 18, 'scll': 'swaaa'}
{'name': 'liulage', 'age': 18, 'scll': 'swaaa'}
```

```
d1_1["name"]="fhajf"
print(d1_1)
print(d)
```

```
{'name': 'fhajf', 'age': 18, 'scll': 'swaaa'}
{'name': 'liulage', 'age': 18, 'scll': 'swaaa'}
```

```
d4={"name":[1,3,5],"age":[52,6,1]}
d4_1=d4.copy()
d4_1["name"]=[51,4,1,1]
print(d4)
print(d4_1)#此时d4没有被修改
```

```
{'name': [1, 3, 5], 'age': [52, 6, 1]}
{'name': [51, 4, 1, 1], 'age': [52, 6, 1]}
```

```
d4={"name":[1,3,5],"age":[52,6,1]}
d4_1=d4.copy()
d4_1["name"][2]=[34]
print(d4)
print(d4_1)#此时d4被修改,因为copy（）是浅拷贝，如果要做到d4不变就要用深拷贝

```

```
{'name': [1, 3, [34]], 'age': [52, 6, 1]}
{'name': [1, 3, [34]], 'age': [52, 6, 1]}
```

```
from copy import deepcopy
d4={"name":[1,3,5],"age":[52,6,1]}
d4_1=deepcopy(d4)
d4_1["name"][2]=[34]
print(d4)#d4没有被修改
print(d4_1)
```

```
{'name': [1, 3, 5], 'age': [52, 6, 1]}
{'name': [1, 3, [34]], 'age': [52, 6, 1]}
```

fromkeys

```
#方法fromkeys创建一个新字典，其中包含指定的键，且每个键对应的值都是None。
dict.fromkeys(["ada","xgb"])
```

```
{'ada': None, 'xgb': None}
```

```
dict.fromkeys(["ada","xgb"],"000")#指定空字典的值
```

```
{'ada': '000', 'xgb': '000'}
```

get

```
#在字典中找到某个键的值,使用“索引”方法在键不在字典中的时候，会报错
d.get("name")
```

```
'liulage'
```

```
print(d.get("lll"))#如果键不在其中，没有返回值
```

```
None
```

```
print(d.get("111",0))#指定键不在的时候的返回值
```

```
0
```

items

```
it=d.items()#返回值属于一种名为字典视图的特殊类型
print(it)
```

```
dict_items([('name', 'liulage'), ('age', 18), ('scll', 'swaaa')])
```

```
#视图的一个优点是不复制，它们始终是底层字典的反映，即便你修改了底层字典亦如此
d["sss"]=11
print(it)
```

```
dict_items([('name', 'liulage'), ('age', 18), ('scll', 'swaaa'), ('sss', 11)])
```

```
list(d.items())
```

```
[('name', 'liulage'), ('age', 18), ('scll', 'swaaa'), ('sss', 11)]
```

keys

```
itk=d.keys()#返回值属于一种名为字典视图的特殊类型
print(itk)
```

```
dict_keys(['name', 'age', 'scll', 'sss'])
```

values

```
itv=d.values()#返回值属于一种名为字典视图的特殊类型
print(itv)
```

```
dict_values(['liulage', 18, 'swaaa', 11])
```

pop

```
#删除键值对,有返回值，也会改变原来的字典
d=dict(items)
print(d)
x=d.pop("age")
print(x)
print(d)
```

```
{'name': 'liuchen', 'id1': 111, 'age': 18}
18
{'name': 'liuchen', 'id1': 111}
```

popitem

```
#删除末尾的键值对，有返回值，也会改变原来的字典
d=dict(items)
print(d)
x=d.popitem()
print(x)
print(d)
```

```
{'name': 'liuchen', 'id1': 111, 'age': 18}
('age', 18)
{'name': 'liuchen', 'id1': 111}
```

setdefault

和get类似，它也获取与指定键相关联的值，但除此之外，setdefault还在字典不包含指定的键时，在字典中添加指定的键-值对。

```
d=dict(items)
x=d.setdefault("name","000")
print(d)
print(x)
```

```
{'name': 'liuchen', 'id1': 111, 'age': 18}
liuchen
```

```
d=dict(items)
xx=d.setdefault("sc","000")
print(d)
print(xx)
```

```
{'name': 'liuchen', 'id1': 111, 'age': 18, 'sc': '000'}
000
```

update

使用另一个字典的值来更新一个字典

```
y1={"a":1,"b":4}
y2={"a":2,"c":7}
y1.update(y2)#当两个字典的键相同时，将y2的值赋值给y1，当y2的键不在y1中时，直接将该键值对添加到y1中|
```

```
print(y1)
```

```
{'a': 2, 'b': 4, 'c': 7}
```