Python 基础

2. Python数据结构

Python中内置的数据结构有六种：Number（数值）、String（字符串）、List（列表）、Tuple（元组）、Dictionary（字典）、Set（集合）。除此以外，数组、矩阵等结构需要导入工具包后才可以实现

2.1 数值

Python3里支持int、float、bool、complex（复数）几种类型的数值。

In [1]:

```
a = 5
b = 2
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a//b)
print(a%b)
print(a**b)
```

```
7
3
10
2.5
2
1
25
```

2.2 字符串

Python中字符串是一个由多个字符组成的序列，字符的个数即为字符串的长度。Python中没有字符，单个字符被认作长度为1的字符串。

In [3]:

```
# 创建字符串
s1 = "python" # 单引号与双引号创建的字符串完全一样
s2 = 'python'
s3 = """
       python

    """ # 三引号字符串可以是多行字符串
s = "
    python
    "
```

```
  File "<ipython-input-3-52a1069bd1dc>", line 8
    s = "
        ^
SyntaxError: EOL while scanning string literal
```

In [10]:

```
s = "o"
type(s)
```

Out[10]:

```
str
```

转义字符

In [4]:

```
# 使用单引号和双引号创建多行字符串时可以使用转义字符\n实现
s = "py\nthon"
print(s)
```

```
py
thon
```

In [5]:

```
# \还可以屏蔽掉符号的含义
s = "python\"" # 屏蔽掉第二个引号的功能
print(s)
```

```
python"
```

In [6]:

```
# 在字符串中打印\
s = "python\\"
print(s)
```

```
python\
```

原始字符串

在字符串前加入字母r

In [8]:

```
s = r"p\y\tho\n"
print(s)
```

```
p\y\tho\n
```

字符串的操作

字符串的元素访问，字符串是一个有序（元素有对应的下标）且不可变（定以后不能修改）的结构

In [9]:

```
s = "python"
s[0] # 通过下标访问元素，对应下标为0~n-1
```

Out[9]:

```
'p'
```

In [10]:

```
# 不可变，修改其中元素会报错
s[0] = "l"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-072a2635df67> in <module>
      1 # 不可变，修改其中元素会报错
----> 2 s[0] = "l"

TypeError: 'str' object does not support item assignment
```

字符串的运算

In [12]:

```
# 加法运算，拼接字符串
a = "hello"
b = "world"
print(a+b)
```

```
helloworld
```

In [13]:

```
# 乘法运算，字符串重复
a = "hello"
b = 2
print(a*b)
```

```
hellohello
```

字符串的常用操作

字符串常用的操作都提供了对应的方法，如切割、大小写替换、拼接等。这些操作并没有修改原有的字符串，而是生成了新的字符串。

In [14]:

```
# 切割
# str.split(str1)：以为str1为分隔符对字符串切割
s = "python"
s.split('h')
```

Out[14]:

```
['pyt', 'on']
```

In [15]:

```
# 替换
# str.replace(str1,str2)：将字符串中的str1替换为str2生成新的字符串
s = 'python'
s.replace('py','PY')
```

Out[15]:

```
'PYthon'
```

In [16]:

```
# 大小写转换
# str.lower()：将字符串中的大写字符转化为小写
# str.upper()：将字符串中小写字符转化为大写
s = "PYthon"
print(s.lower())
```

```
print(s.upper())
```

```
python
PYTHON
```

In [17]:

```
# 拼接
# str.join(iter)：将所给参数中的每个元素以指定的字符连接生成一个新的字符串
"-".join("python")
```

Out[17]:

```
'p-y-t-h-o-n'
```

格式化输出

格式化输出，是指将指定的数据按照给定的格式进行输出。

In [19]:

```
# 格式化操作符（百分号%）、字符串转换类型和格式化操作符辅助指令实现格式化输出
'My name is %s , age is %d' %('AI', 63)
```

Out[19]:

```
'My name is AI , age is 63'
```

2.3 列表

列表是一个序列，其中的元素可以是任意的数据类型，并且可以随时的添加或者删除元素。List = [obj1, obj2, ...]

In [21]:

```
l = [1,2,3] # 创建列表
l[1] # 访问元素
```

Out[21]:

```
2
```

In [22]:

```
# 修改元素
l[1] = 0
print(l)
```

```
[1, 0, 3]
```

列表运算符

In [1]:

```
a = [1,2,3]
b = [4,5]
```

列表中的运算符和字符串相同

In [2]:

```
# 乘法
print(a*2)
# 加法
print(a+b)
```

```
[1, 2, 3, 1, 2, 3]
[1, 2, 3, 4, 5]
```

列表中提供了很多方法，用于列表数据的操作

In [1]:

```
animals = ['cat', 'dog', 'monkey']
# list.append(obj)：在列表末尾添加新的对象。
animals.append('fish')# 追加元素
print(animals)
```

```
['cat', 'dog', 'monkey', 'fish']
```

In [2]:

```
# list.remove(obj)：移除列表中某个值的第一个匹配项。
animals.remove('fish')# 删除元素fish
print(animals)
```

```
['cat', 'dog', 'monkey']
```

In [3]:

```
# list.insert(index, obj)：用于将指定对象插入列表的指定位置。index：插入位置
animals.insert(1,'fish')    # 在下标1的地方插入元素fish
print(animals)
```

```
['cat', 'fish', 'dog', 'monkey']
```

In [4]:

```
# list.pop([index=-1])：要移除列表中对下标对应的元素（默认是最后一个）。Index：下标
animals.pop(1)      # 删除下标为1的元素
print(animals)
```

```
['cat', 'dog', 'monkey']
```

In [6]:

```
#遍历并获取元素和对应索引
# enumerate(sequence) ：将一个可遍历的数据对象组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。
for i in enumerate(animals):
    print(i)
```

```
(0, 'cat')
(1, 'dog')
(2, 'monkey')
```

In [7]:

```
#列表推导式
squares = [x*2 for x in animals] # 批量生成符合规则的元素组成的列表
print(squares)
```

```
['catcat', 'dogdog', 'monkeymonkey']
```

In [8]:

```
list1 = [12,45,32,55]
# list.sort(cmp=None, key=None, reverse=False): cmp为可选参数，如果指定了该参数，会使用该参数的方法进行排序。key是用来进行比较的元素。reverse为排序规则，False为升序。
list1.sort()       # 对列表进行排序
print(list1)       # 输出[12,32,45,55]
```

```
[12, 32, 45, 55]
```

In [9]:

```
# list.reverse(): 反向列表中元素。
list1.reverse()      # 对列表进行逆置
print(list1)       # 输出[55,45,32,12]
```

```
[55, 45, 32, 12]
```

切片与索引

In [13]:

```
# 索引
l = [1,2,3,[4,5]]
l[3][0] # 索引的访问从0到n-1 ，所有有序序列都能使用
```

Out[13]:

```
4
```

In [14]:

```
# 切片
# list[start:end:step],start为0时可以省略，end为n-1时可以省略，step为1时可以省略
l = [0,1,2,3,4,5]
l[::]
```

Out[14]:

```
[0, 1, 2, 3, 4, 5]
```

In [15]:

```
l[1:4:2]
```

Out[15]:

```
[1, 3]
```

2.4元组

元组是一个不可变的有序序列，类似于字符串，但是不同的是，元组中的元素可以是任意对象。元组的形式为小括号包裹元素：(obj1,obj2.....)

In [11]:

```
# 创建元组
t = (1,[1,2],"python")
print(t)
print(type(t))
```

```
print(type(t))
```

```
(1, [1, 2], 'python')
<class 'tuple'>
```

In [12]:

```
# 创建单个元素的元组
t1 = (5)
t2 = (5,) # 创建单个元素的元组需要使用‘，’来标明是元组
print("t1",type(t1))
print("t2",type(t2))
```

```
t1 <class 'int'>
t2 <class 'tuple'>
```

通过下标访问元素

In [16]:

```
t = (1,2,3)
print(t[1])
```

```
2
```

In [17]:

```
# 元组不可变，修改数据会报错
t[1] = 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-8ea75fcadaa8> in <module>
      1 # 元祖不可变，修改数据会报错
----> 2 t[1] = 0

TypeError: 'tuple' object does not support item assignment
```

In [18]:

```
# 变相的修改元组数据
t = (1,2,[3,4]) # 不推荐这样使用
t[2][1] = 5
t
```

Out[18]:

```
(1, 2, [3, 5])
```

2.5字典

字典的每个元素由两部分组成－键和值，所以字典的元素也被称为键值对。其中键是不可变且唯一的，如果字典有相同的键，则后面的键对应的值会将前面的值覆盖。数据量大时，字典数据的访问速度比列表块。字典由一对花括号包裹，元素由逗号隔开：{key:value}。

In [2]:

```
# 字典的三种赋值操作
x = {'food':'Spam','quantity':4,'color':'pink'}
x2 =dict(food='Spam',quantity=4, color='pink')
x3 = dict([("food", "Spam"),("quantity", 4),("color","pink")])

print(x)
```

```
print(x2)
print(x3)
```

```
{'food': 'Spam', 'quantity': 4, 'color': 'pink'}
{'food': 'Spam', 'quantity': 4, 'color': 'pink'}
{'food': 'Spam', 'quantity': 4, 'color': 'pink'}
```

获取字典数据

In [3]:

```
# 使用键获取值
print(x["food"])
print(x["x"]) # 如果访问不存在的键，会报错
```

```
Spam
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-3-fbbd3c5dc2a3> in <module>
      1 # 使用键获取值
      2 print(x["food"])
----> 3 print(x["x"]) # 如果访问不存在的键，会报错

KeyError: 'x'
```

In [5]:

```
# 使用get(key,["str"])方法获取数据
print(x.get("food"))
print(x.get("x")) # 使用get方法获取数据时，如果键不存在会返回指定的值str
print(x.get("x","x不存在"))
```

```
Spam
None
x不存在
```

In [4]:

```
# 查看所有的键
print(x.keys())
# 查看所有的值
print(x.values())
# 查看所有的键值对
print(x.items())
```

```
dict_keys(['food', 'quantity', 'color'])
dict_values(['Spam', 4, 'pink'])
dict_items([('food', 'Spam'), ('quantity', 4), ('color', 'pink')])
```

In [5]:

```
# 向字典中插入数据
x["x"] = "x"
print(x)
```

```
{'food': 'Spam', 'quantity': 4, 'color': 'pink', 'x': 'x'}
```

In [6]:

```
# 修改字典中的数据
x["x"] = 0
print(x)
```

```
{'food': 'Spam', 'quantity': 4, 'color': 'pink', 'x': 0}
```

2.6 集合

集合中的元素是唯一的，重复的元素会被删除。集合是由一个花括号包裹，内部元素以逗号隔开：{obj1, obj2,....}。

In [8]:

```
sample_set = {'Prince', 'Techs'}
print('Data' in sample_set)     #in的作用是检查集合中是否存在某一元素
```

```
False
```

In [9]:

```
# set.add(obj): 给集合添加元素，如果添加的元素在集合中已存在，则不执行任何操作。
sample_set.add('Data')
print(sample_set)
print(len(sample_set))
```

```
{'Techs', 'Prince', 'Data'}
3
```

In [10]:

```
# set.remove(obj): 移除集合中的指定元素。
sample_set.remove('Data')     # 删除元素Data
print(sample_set)
```

```
{'Techs', 'Prince'}
```

In [11]:

```
list2 = [1,3,1,5,3]
print(list(set(list2)))# 利用集合元素的唯一性进行列表去重
sample_set = frozenset(sample_set)# 不可变集合
```

```
[1, 3, 5]
```

2.7 数据拷贝

在Python中对于数据的拷贝可以根据拷贝形式的不同分为深拷贝和浅拷贝

-浅拷贝（copy()），即对数据的表面结构进行拷贝，如果数据为嵌套的结构，则嵌套结构里面的元素是对之前数据的引用。修改之前的数据会影响拷贝得到的数据。

-深拷贝，解决了嵌套结构中深层结构只是引用的问题，它会对所有的数据进行一次复制，修改之前的数据则不会改变拷贝得到的数据。

In [15]:

```
# 数据赋值
a = [1,2,3,[4,5]]
b = a # 赋值
c = a.copy() # 浅拷贝

import copy
d = copy.deepcopy(a) # 深拷贝
```

In [16]:

```
a[0] = 6 # 修改a的值
print("a:",a)
print("b:",b)
print("c:",c)
print("d:",d)
```

```
a: [6, 2, 3, [4, 5]]
b: [6, 2, 3, [4, 5]]
c: [1, 2, 3, [4, 5]]
d: [1, 2, 3, [4, 5]]
```

In [17]:

```
# 修改嵌套列表的值
a[3][0] = 7
print("a:",a)
print("b:",b)
print("c:",c)
print("d:",d)
```

```
a: [6, 2, 3, [7, 5]]
b: [6, 2, 3, [7, 5]]
c: [1, 2, 3, [7, 5]]
d: [1, 2, 3, [4, 5]]
```

2.8 运算符

In [32]:

```
# 赋值运算符
a=5
a+=1
print("a+=1:",a)
a-=1
print("a-=1:",a)
a*=2
print("a*=2:",a)
a/=2
print("a/=2:",a)
```

```
a+=1: 6
a-=1: 5
a*=2: 10
a/=2: 5.0
```

In [40]:

```
# 比较运算符
print(1>=2)
print(1<=2)
print(1!=2)
print(1==2)
```

```
False
True
True
False
```

In [33]:

```
# 逻辑运算
True and False
```

Out[33]:

```
False
```

In [34]:

```
True or False
```

Out[34]:

```
True
```

In [35]:

```
not True
```

Out[35]:

```
False
```

In [36]:

```
# 成员运算符
a = [1,2,3]
1 in a
```

Out[36]:

```
True
```

In [37]:

```
4 not in a
```

Out[37]:

```
True
```

In [41]:

```
# 身份运算符
a = [1,2]
b = a
c = a.copy()
```

In [43]:

```
print(a is b)
print(a is c)
print(a == b == c)
```

```
True
False
True
```

In [ ]:

```

```