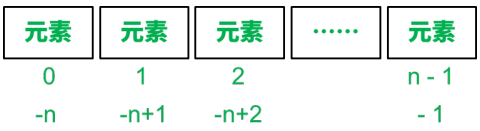
第 2 讲　复合数据类型

知识脉络

○ 序列的共通特性

字符串、列表、元组都属于序列，包含多个有先后次序的数据（元素），通过下标（索引）来访问



1. 运算符

① 访问运算符

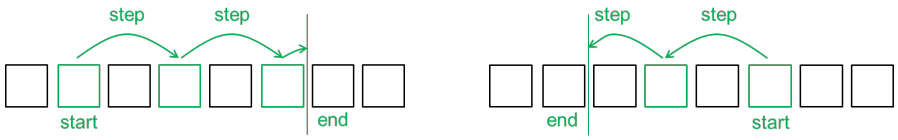
```
a[index]                # 返回序列 index 处的元素，index 为整数
```

· 如果 index 为负数，表示从序列的最后一个元素往前引用（见上面的图）

· 设序列长度为 n，如果下标 index 不在-n ~ n-1 的范围内则会抛出异常 IndexError

② 切片运算符

```
a[start: end]
a[start: end: step]    # 返回下标为 start 至 end（不含）的元素组成的列表，步长为 step
```



· step 默认值为 +1　step > 0 时，start 默认值为 0，end 默认值为末尾元素（含）

　　step < 0 时，start 默认值为-1，end 默认值为首个元素（含）

③ 算术运算符

```
seq1 + seq2  # 将序列 seq2 接在序列 seq1 后面，生成新序列，两个序列的类型要相同
```

```
seq * n      # 将 n 个序列 seq 首尾连接生成新序列，n 必须是整数
```

④ 关系运算符

· > >= == != <= < ：从两个序列的第一个元素开始逐一比较，规则与字符串相同

· in 和 not in

```
e in seq      # 如果 e 是序列 seq 中的元素，结果为 True，反之为 False
```

```
e not in seq  # 如果 e 不是序列 seq 中的元素，结果为 True，反之为 False
```

注意：**字符串可以判断某个字符串是不是它的一部分（子串），但其它类型的序列只能判断单个元素**

2. 常用方法

①
```
seq.index(value, start, stop)   # 返回 value 在 start~stop（不含）范围内首次出现时的下标
```

· 如果 value 不存在则会产生 ValueError，start 和 stop 的默认值是从头到尾

②
```
seq.count(x)                    # 返回序列中元素 x 出现的次数（对于字符串，x 可以是子串）
```

一　列表

1. 表示

· 由任意元素组成的序列，元素可以是不同类型，且可以被修改。

```
[1, 3, 5, 7, 11, 13]
```

```
[]    # 空列表
```

2. 运算符（用 xxxx 表示列表本身）

① **访问/切片运算符**

· 在序列共有的访问、切片功能上，可结合赋值修改元素

```
xxxx[index] = value
```

```
xxxx[start:end] = value    # value 必须是序列
```

② **删除运算符**

```
del xxxx[index]            # 删除 xxxx 中下标为 index 的元素
```

3. 常用方法

第一组：增

```
① xxxx.append(e)           # 无返回值，在列表尾部追加元素 e
② xxxx.extend(x)           # 无返回值，将序列 x 接到列表 xxxx 的后面
③ xxxx.insert(i, value)    # 在下标 i 处插入 value（该处原有的元素后移），i 超出范围则加到最后
```

第二组：删

```
④ xxxx.clear()             # 移除列表的所有元素
⑤ xxxx.pop(index)          # 返回并删除下标为 index 的元素，默认是最后一个
⑥ xxxx.remove(value)       # 删除第一个值为 value 的元素，如果 value 不存在则会发生错误
```

第三组：改

```
⑦ xxxx.copy(x)             # 返回列表的备份（关于为什么要备份见第 3 讲）
⑧ xxxx.reverse()           # 将列表进行反转
⑨ xxxx.sort(key, reverse)  # 将列表中元素按照 key 的规则，reverse 的顺序排序
```

· key：函数对象，将每个元素送入该函数后的返回值作为比较依据，默认依据是元素本身，要求元素之间能比较

· reverse 为 True 时为从大到小排序（降序），False 时为从小到大排序（升序）

二　元组

· 元组是**不可修改**的可以表达任何类型、任意数量的数据的有序序列，无特殊方法

· 单个元素的元组必须后跟逗号，否则会被程序无视括号，不认为是元组

```
(10, 20, 30, 40, 50)  (1,)
```

三　字符串

1. **常用方法**（用 x 表示字符串本身）

第一组：大小写

① `x.title()`　# 返回将字符串 x 中每个英语单词的首字母改为大写后的字符串

② `x.lower()`　# 返回将字符串 x 中所有英文字母改为小写后的字符串

③ `x.upper()`　# 返回将字符串 x 中所有英文字母改为大写后的字符串

第二组：删除空格

④ `x.strip()`　# 返回删除字符串 x 开头和末尾空格后的字符串

⑤ `x.rstrip()`　# 返回删除字符串 x 末尾空格后的字符串

⑥ `x.lstrip()`　# 返回删除字符串 x 开头空格后的字符串

第三组：子串

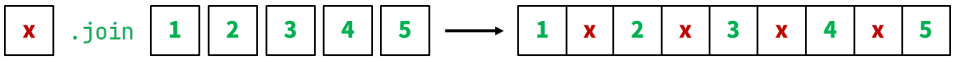
⑦ `x.replace(old, new)`　　# 返回将字符串中的 old 子串用 new 替换后的字符串

⑧ `x.find(sub, start, end)`　# 返回子串 sub 在 start~end（不含）区间首次出现的位置，没有则返回-1

· start 默认为 0，end 默认为最后一个元素（能取到）

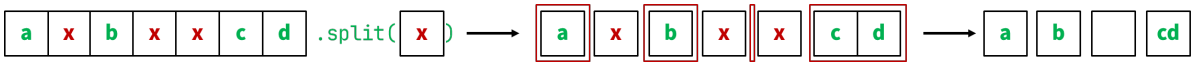
第四组：列表转换

⑨ `x.join(seq)`　# 将序列 seq 中的元素转换为字符串后用 x 连接成的新字符串



⑩ `x.split(sep)`　# 将字符串拆分成根据 sep 串分隔开的列表，sep 默认值为空格

· 若 x 中 sep 连续出现，中间没有其它字符，则会产生一个空字符串加入列表



四　集合

1. **表示**

· 集合用花括号{}表示，其元素**没有先后顺序、值不重复**且**必须是不可变对象**

`{1, 5, 7}`　# 创建时自动去重

`set()`　# 空集合，注意 **{} 不代表空集合而是空字典**

2. **运算符**

① **关系运算符**

运算符	True	运算符	True
in	元素 e 在集合 s 中	not in	元素 e 不在集合 s 中
s1 == s2	s1 与 s2 相等	s1 != s2	s1 与 s2 不相等
s1 < s2	s1 是 s2 的真子集	s1 > s2	s2 是 s1 的真子集
s1 <= s2 s1.issubset(s2)	s1 是 s2 的子集	s1 >= s2 s1.issuperset(s2)	s2 是 s1 的子集

② **算术运算符**

运算符	功能	等价方法
s1 | s2	并集	s1.union(s2)
s1 & s2	交集	s1.intersection(s2)
s1 - s2	差集（s1 有 s2 无）	s1.difference(s2)
s1 ^ s2	对称差（排除掉交集元素）	s1.symmertric_difference(s2)

· 尤其需要注意的是，**集合不能够访问特定的元素**，只能通过 for 循环迭代访问（访问的顺序随机）

3. **方法**（用 xxxx 表示集合本身）

① `xxxx.add(e)` # 将元素 e 加入集合中

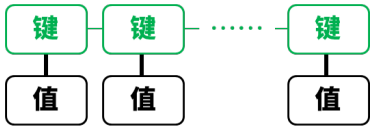
② `xxxx.remove(e)` # 将元素 e 移出集合，若 e 不存在则报错

五 字典

1. **表示**

· 字典是用“键”做索引来存储的数据的集合

一个键（key）和它所对应的值（value）形成字典的一个条目




```
{'math': 1, 'python': 2, 'c': 3}  # 可变对象（列表，字典等）不可以作为字典的键
```

2. **运算符**

① **访问/编辑运算符**

```
dictionary[key]            # 获得 key 对应的 value，若 key 不在字典中则报错 KeyError
dictionary[key] = value    # 将 key 所对应的数据修改为 value，如果 key 不存在则增加该项
```

② **删除**

```
del dictionary[key]        # 删除 key 对应的条目，若 key 不在字典中则报错 KeyError
```

③ **关系运算符**

运算符	True	运算符	True
in	键 key 在字典中	not in	键 key 不在字典中
==	两个字典键值对相同	!=	两个字典键值对不相同

④ **解包运算符**

```
{**dict1, **dict2}   # 合并字典，若字典有相同的 key，则后面的字典条目覆盖前面的
```

4. **常用方法**（用 xxxx 表示字典本身）

（一）生成遍历

① `xxxx.keys()` # 返回字典的**键**组成的**可迭代序列**，可以给 for 遍历

② `xxxx.values()` # 返回字典的值组成的可迭代序列，可以给 for 遍历或转换成列表

③ `xxxx.items()` # 返回字典的(键,值)元组组成的可迭代序列，可以给 for 遍历或转换成元组

（二）删与查

④ `xxxx.pop(key)` # 返回键 key 所对应的值，同时删除这个条目

⑤ `xxxx.clear(key)` # 清空字典

⑥ `xxxx.get(key, value)` # 返回这个键所对应的值，如找不到返回 value(默认为 None)

六 复合数据类型的内置函数

1.复合类型函数（用 x 表示复合类型本身）

① `len(x)` # 返回**序列或集合** x 内部元素的个数或字典 x 内部键值对个数

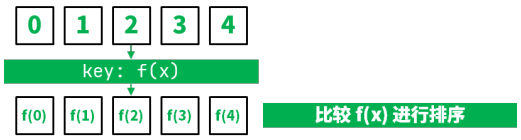
② `max(x)` # 返回**序列或集合** x 中的最大值，要求序列中的元素可以相互比较

③ `min(x)` # 返回**序列或集合** x 中的最小值，要求序列中的元素可以相互比较

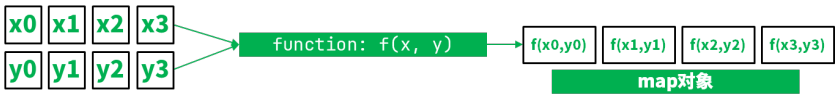
④ `sum(x)` # 返回**序列或集合** x 所有元素之和，要求序列中的元素可以相互相加

⑤ `sorted(x, key, reverse)` # 将 x 中的元素排序，返回一个新建的列表

- key：**函数对象**，代表**排序依据**，默认比较元素自身
  各元素的顺序取决于输入该函数后的返回值大小顺序
- reverse：排序方向，True 为降序，False 为升序（默认）



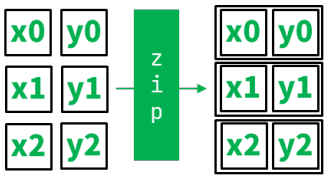
⑥ `map(func, x1, x2, …)` # 将复合类型 x 中的元素送入函数对象 func，创建以返回值为元素的 map 对象



- 可以接收一个或多个 x，function 是一个函数对象，其接收元素的形参个数应与提供的 x 个数相同
- 注意：map 函数的返回值是 map 对象，需要通过类型转换函数（如 list()）转换为其它类型

⑦ `zip(x1, x2, …)` # 将复合类型 x1,x2…中相同位置的元素打包成元组，返回由元组组成的 zip 对象

- 若各个 x 长度不一致，以最短的长度为准
  （其它 x 超出该长度的都被截掉）
- 如果只提供一个 x，同样会打包成元组（ele, ）
- 应用：可以将两个列表打包后传给 dict 生成字典



⑧ `all(x)` # 若复合类型 x 中所有元素都为 True，返回 True，否则返回 False

`any(x)` # 只要复合类型 x 中有一个元素为 True 就返回 True，否则返回 False

2.复合类型转换函数（用 x 表示复合类型本身）

① `list(x)` # 将复合类型 x 的元素拆分为列表的元素，创建列表

② `tuple(x)` # 将复合类型 x 的元素拆分为元组的元素，创建元组

③ `set(x)` # 将复合类型 x 的元素去重，创建为集合

④ `dict(x)` # 创建字典，x 为**键值对元组组成的列表**，或者是多个 key=value 的关键词参数

典型例题

例 1 列表 `lst = [12, 5, -22, -10, -26, 35, 0, 49, 3, -21]`，则下列表达式的结果是

① `lst[::]`　② `lst[::-1]`　③ `lst[::2]`　④ `lst[1::2]`

⑤ `lst[3:8:2]`　⑥ `lst[100:]`　⑦ `lst[100]`

解 ① `[12, 5, -22, -10, -26, 35, 0, 49, 3, -21]`

② `[-21, 3, 49, 0, 35, -26, 10, -22, 5, 12]`

③ `[12, -22, -26, 0, 3]`

④ `[5, -10, 35, 49, -21]`

⑤ `[-10, 35, 49]`

⑥ `[]`（切片范围可以超出列表界限，如果没有元素被切到就返回空列表）

⑦ `IndexError`（注意切片和索引的区别）

例 2 表达式 `'23' * 3` 的值为_____。

解 相当于 3 个`'23'`首尾相接，因此结果是`'232323'`

例 3 表达式 `[1,2,[3]]+[4,5]`的结果为_____。

解 注意[3]是外层列表内部的元素，不参与相加，因此结果是`[1,2,[3],4,5]`

例 4 表达式 `'23' in '1234' == True` 的返回值是 True（正确/错误）

解 `==` 的优先级高于 `in`，因此先计算`'1234' == True`，结果为 `True`

然后计算`'23' in True`，结果为 `False`

例 5 表达式 `sum((1,3,5,7,9)) / len((1,3,5,7,9))` 的返回值是_____

解 `sum((1,3,5,7,9))` 的结果是 `1+3+5+7+9 = 45`

`len((1,3,5,7,9))` 的结果是 `5`，因此返回值 `45 / 5 = 9.0`（注意是浮点数除法）

例 6 `len('3//11//2018'.split('/'))` 的结果是_____。

解 分隔符为 `'/'`，字符串里一共有 4 个`'/'`，因此会分出 5 个字符串，因此 `len` 的结果为 5

（列表为`['3', '', '11', '', '2018']`）

例 7 下列语句能打印出 smith\exam1\test.txt?

A. `print("smith\exam1\test.txt")`　B. `print("smith\\exam1\\test.txt ")`

C. `print("smith\"exam1\"test.txt")`　D. `print("smith"\exam1"\test.txt")`

解 由于\是转义字符的起手，想要输出\本身，必须用它的转义字符形式\\或者加前缀 `r`

因此只有 B 正确

例 8 已知 `a='□□□xyz□□'`（□代表空格），则 `print(a.strip(),a.rstrip(), a.lstrip())`的输出是_____。（不要加引号）

解 三个函数的返回值分别是 `'xyz'`，`'□□□xyz'`，`'xyz□□'`

`print` 函数中，参数中间会插入分隔符（默认空格，见第 5 讲）

因此输出是`'xyz□□□□xyz□xyz□□'`

例 9 `'programming'.find('r', 2)`的结果是_____。（不要加引号）

解 start 参数为 2，因此从第 3 个字符（`'o'`）开始查找，第一个找到的`'r'`是第 5 个字符

因此结果为 4

例 10 `'12 '* 3 == ' '.join(['12']*3)` 的输出是 `True`（正确/错误）

解 `'12 '* 3` 的结果为`'12 12 12 '`，`' '.join(['12']*3)` 的结果是`'12 12 12'`，两者不同

因此输出是 `False`