11-Python序列数据

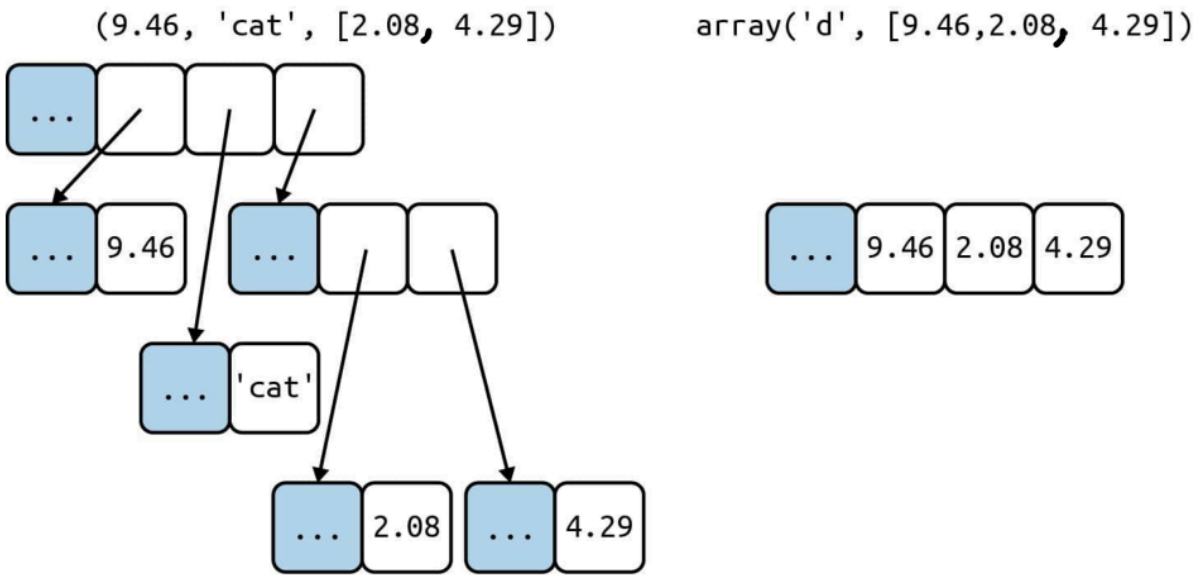
大纲

- 概述
- 列表推导式与生成器表达式
- 拆包与嵌套拆包
- 序列模式匹配
- 切片
- 其他序列数据结构： 数组，memeoryview, 双端队列和其他队列

概述

序列数据可以分为：

- 容器序列（container sequence）：可以存放不同类型的元素，包括嵌套的容器，例如： list, tuple, collections.deque
- 扁平序列（flat sequence）：可以存放一种简单类型(int, float, byte)的元素。例如：str, bytes, array.array



任何Python对象在内存中都有一个包含元数据的标头。最简单的Python对象，例如一个float，内存标头中有一个值字段和两个元数据字段。

- ob_refcnt：对象的引用计数
- ob_type：指向对象类型的指针
- ob_fval：一个 C 语言 double 类型值，存放 float 的值

序列数据还可以分为：

- 可变序列（mutable sequence）：list, bytearray, array.array, collections.deque, memoryview
- 不可变序列（immutable sequence）：tuple, str, bytes
- 可变序列继承不可变序列的所有方法，另外还多实现了几个方法。

In [1]:
```
from collections import abc
issubclass(tuple, abc.Sequence)
```

Out[1]: True

In [3]:
```
issubclass(list, abc.MutableSequence)
```

Out[3]: True

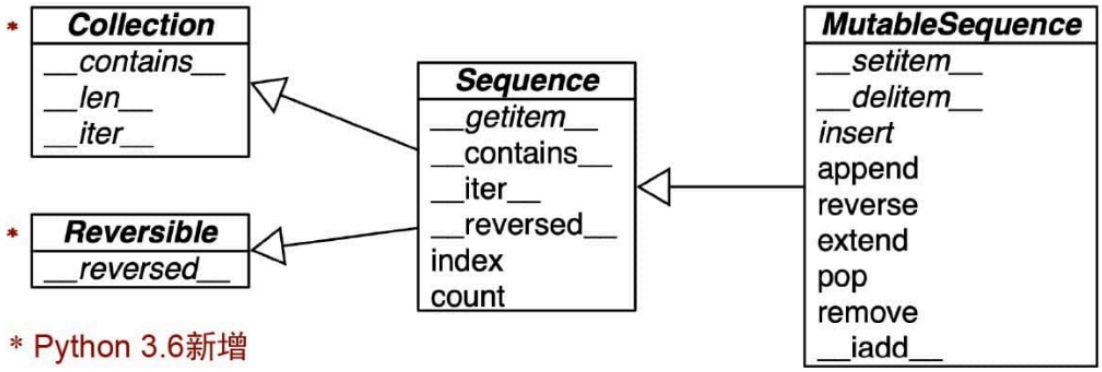


图 2-2：collections.abc 中部分类的简化 UML 类图（左边是超类；箭头从子类指向超类，表示继承；以斜体显示的名称是抽象类和抽象方法）

列表推导式与生成器表达式

- Python 会忽略 []、{} 和 () 内部的换行。因此，列表、列表推导式、元 组、字典等结构完全可以分成几行来写，无须使用续行转义符 \。
- 使用这3种括号定义字面量时，项与项之间使用逗号分隔，末尾的逗号将被忽略。
- 跨多行定义列表字面量时，最好在最后一项后面添加一个逗号。

In [5]:
```
# 打印符号的Unicode码位

symbols = '$¢£¥€¤'
codes = []
for symbol in symbols:
    codes.append(ord(symbol))
codes
```

Out[5]: [36, 162, 163, 165, 8364, 164]

In [6]:
```
# 与上面的代码等效

symbols = '$¢£¥€¤'
codes = [ord(symbol) for symbol in symbols]
codes
```

Out[6]: [36, 162, 163, 165, 8364, 164]

In [7]:
```
# 列表推导式和生成器表达式的局部作用域

x = 'ABC'
codes = [ord(x) for x in x]
x
```

Out[7]:
```
'ABC'
```

In [8]:
```
codes = [last:=ord(c) for c in x]
last
```

Out[8]:
```
67
```

In [9]:
```
c
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [9], in <cell line: 1>()
----> 1 c

NameError: name 'c' is not defined
```

- x毫发无损，绑定的值仍然是'ABC'
- 使用海象运算符赋值:= 的 last 仍然可以访问
- c消失了，因为它只存在于列表推导式的内部

生成器表达式

In [12]:
```
symbols = '$¢£¥€¤'
my_gen = (ord(symbol) for symbol in symbols)
tuple(my_gen)
```

Out[12]:
```
(36, 162, 163, 165, 8364, 164)
```

生成器可以作为函数的参数，如果生成器表达式是唯一的参数，可以省略括号。

In [10]:
```
symbols = '$¢£¥€¤'
tuple(ord(symbol) for symbol in symbols)
```

Out[10]:
```
(36, 162, 163, 165, 8364, 164)
```

In [11]:
```
import array
array.array('I', (ord(symbol) for symbol in symbols))
```

Out[11]:
```
array('I', [36, 162, 163, 165, 8364, 164])
```

序列与可迭代对象拆包（unpack）

最典型的拆包就是并行赋值（parallel assignment）

In [13]:
```
lax_coordinates = (33.9425, -118.408056)
latitude, longitude = lax_coordinates
latitude
```

Out[13]:
```
33.9425
```

In [14]:
```
longitude
```

Out[14]:
```
-118.408056
```

In [16]:
```
# 利用拆包交换变量的值
a = 10
b = 20
a, b = b, a
print(a, b)
```

```
20 10
```

利用 * 运算符可以拆包任意长度的可迭代对象

In [17]:
```
divmod(20, 8)
```

Out[17]:
```
(2, 4)
```

In [18]:
```
t = (20, 8)
divmod(*t)
```

Out[18]:
```
(2, 4)
```

嵌套拆包

In [ ]:
```
metro_areas = [
    ('Tokyo', 'JP', 36.933, (35.689722, 139.691667)),
    ('Delhi NCR', 'IN', 21.935, (28.613889, 77.208889)),
    ('Mexico City', 'MX', 20.142, (19.433333, -99.133333)),
    ('New York-Newark', 'US', 20.104, (40.808611, -74.020386)),
    ('São Paulo', 'BR', 19.649, (-23.547778, -46.635833)),
]

def main():
    print(f'{"":15} | {"latitude":>9} | {"longitude":>9}')
    for name, _, _, (lat, lon) in metro_areas:  # <2>
        if lon <= 0:  # <3>
            print(f'{name:15} | {lat:9.4f} | {lon:9.4f}')

if __name__ == '__main__':
    main()
```

- 每个元组是一个四字段记录，最后一个字段是坐标对。
- 把最后一个字段赋值给一个嵌套元组，拆包坐标对。
- lon<=0是测试条件，只选取了西半球的城市。

切片

切片和区间排除最后一项是一种 Python 风格约定，这与Python、C 和很多其他语言中从零开始的索引相匹配。排除最后一项可以带来以下好处。

- 在仅指定停止位置时，容易判断切片或区间的长度。例如，range(3) 和my_list[:3] 都只产生 3 项。
- 同时指定起始和停止位置时，容易计算切片或区间的长度，做个减法即可：stop - start。
- 方便在索引 x 处把一个序列拆分成两部分而不产生重叠，直接使用 my_list[:x]和 my_list[x:] 即可。例如：

In [20]:
```
l = [10, 20, 30, 40, 50, 60]
l[:2]
```

Out[20]:
```
[10, 20]
```

In [21]:
```
l[2:]
```

Out[21]:
```
[30, 40, 50, 60]
```

使用双端队列deque

- list 只有在末尾附加（append）数据时是高效的
- 使用 deque 在头和尾都可以高效地操作数据

In [1]:
```
from collections import deque
dq = deque(range(10))
dq.append(11)
dq.appendleft(-1)  # appendleft效率比insert(0, -1)高
print(dq)

dq.pop()
dq.popleft()
print(dq)
```

```
deque([-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11])
deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [2]:
```
dq.index(5)
```

Out[2]:
```
5
```

In [3]:
```
dq.extend([11, 12, 13])
dq.extendleft([-1, -2, -3])
print(dq)
```

```
deque([-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13])
```

In [4]:
```
dq.rotate(1)  # 向右旋转1位
print(dq)

dq.rotate(-4)  # 向左旋转1位
print(dq)
```

```
deque([13, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12])
deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, -3, -2, -1])
```

match/case 语句

- Python 3.10引入了新的 match/case 语句
- match/case 语句类似于C和Java中的 switch/case 语句，但更加强大，可以匹配更多的模式
- 具体可以参考官方文档PEP 636 – Structural Pattern Matching: Tutorial (https://peps.python.org/pep-0636/)
- 或者参考《Fluent Python》一书中这些小节：
  - Pattern Matching with Sequences
  - Pattern Matching with Mapping
  - Pattern Matching Class Instances
  - Pattern Matching in lis.py: A Case Study

In [ ]:
```
# 输入格式为: action object
# 例如:
# move north
# get sword
# attack orc
command = input("What are you doing next? ")
```

In [ ]:
```
match command.split():
    case ["quit"]:
        print("Goodbye!")
        print("quit_game()")
    case ["look"]:
        print("current_room.describe()")
    case ["get", obj]:
        print(f"Get a {obj}")
    case ["go", direction]:
        print(f"Go to the {direction}")
    case _:
        print("Unknown command.")
```

In [ ]:
```
metro_areas = [
    ('Tokyo', 'JP', 36.933, (35.689722, 139.691667)),
    ('Delhi NCR', 'IN', 21.935, (28.613889, 77.208889)),
    ('Mexico City', 'MX', 20.142, (19.433333, -99.133333)),
    ('New York-Newark', 'US', 20.104, (40.808611, -74.020386)),
    ('São Paulo', 'BR', 19.649, (-23.547778, -46.635833)),
]

def main():
    print(f'{"":15} | {"latitude":>9} | {"longitude":>9}')
    for record in metro_areas:
        match record:  # <1>
            case [name, _, _, (lat, lon)] if lon <= 0:  # <2>
                print(f'{name:15} | {lat:9.4f} | {lon:9.4f}')

if __name__ == '__main__':
    main()
```

匹配对象同时满足下面的条件时能匹配序列模式：

- 匹配对象是序列。
- 匹配对象和模式的项数相等。
- 对象的项相互匹配，包括嵌套的项。

标准库中以下类型与序列模式兼容：

- list
- memoryview
- array.array
- tuple
- range
- collections.deque

在 match/case 上下文中，str、bytes 和 bytearray 实例不作 为序列处理。match 把这些类型视为"原子"值，就像整数 987 整体被视 为一个值，而不是数字序列。倘若把这三种类型视为序列，就可能会由于 意外匹配而导致 bug。如果想把这些类型的对象视为序列，则要在 match 子句中转换，例如以下示例中的 tuple(phone)。

```
    match tuple(phone):
        case ['1', *rest]: # 北美洲和加勒比地区
        ...
        case ['2', *rest]: # 非洲
        ...
        case ['3' | '4', *rest]: # 欧洲
        ...
```