لیست (list)

آخرین به‌روزرسانی: ۲۹ اردیبهشت ۱۴۰۵

نوع داده لیست (list) در پایتون یکی از پرکاربردترین و انعطاف‌پذیرترین ساختارهای داده این زبان است. لیست‌ها، دنباله‌هایی مرتب و تغییرپذیر هستند که می‌توانند عناصری از انواع داده مختلف مانند اعداد، رشته‌های متنی، بولین‌ها، آبجکت‌ها و حتی لیست‌های دیگر را همزمان در خود نگه دارند. برخلاف آرایه‌ها در بسیاری از زبان‌های دیگر، لیست‌های پایتون اندازه پویا دارند و نیازی به تعریف ظرفیت از قبل ندارند. list به عنوان یک نوع دنباله‌ای قابل تغییر، هم از عملیات مشترک دنباله‌ها مثل اندیس‌گذاری و برش پشتیبانی می‌کند، هم مجموعه‌ای غنی از متدهای اختصاصی برای تغییر محتوا دارد.

فهرست مطالب:

ایجاد لیست در پایتون
عملیات پایه روی لیست
متدهای پرکاربرد لیست
لیست‌های تودرتو در پایتون
ساخت پشته و صف در پایتون
مثال واقعی از کاربرد لیست
سوالات متداول

ایجاد لیست در پایتون

داده‌هایی از نوع لیست در پایتون را می‌توان به روش‌های مختلفی ساخت.

۱) نوشتار مستقیم با کروشه:

```
>>> empty = []
>>> numbers = [1, 2, 3, 4, 5]
>>> mixed = [1, "hello", 3.14, True, None]
>>> nested = [[1, 2], [3, 4], [5, 6]]
```

۲) روش List Comprehension: پایتونیک‌ترین و بهینه‌ترین روش برای ساخت این نوع داده به حساب می‌آید:

```
>>> squares = [x ** 2 for x in range(1, 6)]
>>> squares
[1, 4, 9, 16, 25]

>>> words = ["hello", "world", "python"]
>>> upper_long = [w.upper() for w in words if len(w) > 4]
>>> upper_long
['HELLO', 'WORLD', 'PYTHON']

>>> matrix = [[i * j for j in range(1, 4)] for i in range(1, 4)]
>>> matrix
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

۳) تابع list: سازنده نوع داده `list` در پایتون، تابع `list` است:

```
list(iterable)
```

مقدار بازگشتی تابع `list` همیشه یک لیست از نوع `list` است.

راهنمای جامع تابع list

در مورد تابع `list` که سازنده یا `Constructor` برای لیست‌ها در پایتون است بیشتر بدانید: **آشنایی با تابع list در پایتون**

نکته: لیست‌ها می‌توانند عناصر تکراری داشته باشند و ترتیب درج عناصر همیشه حفظ می‌شود. این دو ویژگی آنها را از `set` و `dict` متمایز می‌کند.

عملیات پایه روی لیست

در ادامه، تعدادی مثال از عملیات روی list در پایتون را با هم مرور خواهیم کرد.

الحاق و تکثیر: عملگرهای + و * به ترتیب عملیات الحاق و تکثیر را روی لیست‌ها انجام می‌دهند:

```
>>> [1, 2] + [3, 4]
[1, 2, 3, 4]

>>> [0] * 5
[0, 0, 0, 0, 0]
```

نکته: در لیست‌های تودرتو یعنی زمانی که یک list حاوی آبجکت‌های تغییرپذیر (مثل لیست دیگری) باشد، تکثیر با عملگر * کپی نمی‌سازد بلکه ارجاع را تکرار می‌کند:

```
>>> matrix = [[0] * 3] * 3
>>> matrix[0][1] = 99
>>> matrix
[[0, 99, 0], [0, 99, 0], [0, 99, 0]]
```

طول لیست و بررسی عضویت: با استفاده از عملگرهای منطقی in و not in می‌توان بررسی کرد که آیا یک آیتم درون list قرار دارد یا خیر:

```
>>> 3 in [1, 2, 3, 4]
True

>>> 10 not in [1, 2, 3]
True

>>> len([1, 2, 3, 4, 5])
5
```

دسترسی با اندیس و برش: listها دنباله‌ای از آیتم‌ها هستند و مثل استرینگ‌ها از اندیس‌گذاری پشتیبانی می‌کنند:

```
>>> fruits = ["apple", "banana", "cherry", "date", "elderberry"]

>>> fruits[0]          # first item
'apple'

>>> fruits[-1]         # last item
'elderberry'

>>> fruits[1:4]        # slicing
['banana', 'cherry', 'date']

>>> fruits[:3]         # from beginning to index 3
['apple', 'banana', 'cherry']

>>> fruits[2:]         # from index 2 to end
['cherry', 'date', 'elderberry']

>>> fruits[::2]        # one in between
['apple', 'cherry', 'elderberry']

>>> fruits[::-1]       # reverse list
['elderberry', 'date', 'cherry', 'banana', 'apple']
```

دقت داشته باشید که در آخرین خط مثال که از سینتکس `fruits[::-1]` برای معکوس کردن `list` استفاده شده، یک لیست جدید ساخته می‌شود و ارجاع به `list` اصلی نیست.

تغییر عناصر با اندیس و برش: چون لیست‌ها تغییرپذیر (`mutable`) هستند، می‌توانید مستقیماً عناصر را دستکاری کنید:

```
>>> nums = [1, 2, 3, 4, 5]

>>> nums[0] = 99
>>> nums
[99, 2, 3, 4, 5]

>>> nums[1:3] = [20, 30]
>>> nums
[99, 20, 30, 4, 5]

>>> nums[1:3] = []
>>> nums
[99, 4, 5]

>>> nums[1:1] = [100, 200]
>>> nums
[99, 100, 200, 4, 5]
```


متدهای پرکاربرد لیست

نوع `list` در پایتون دارای ۱۱ متد اختصاصی است که عملیات افزودن، حذف، جستجو، مرتب‌سازی و کپی را پوشش می‌دهند.

۱) متدهای افزودن عنصر:

- متد `append` – افزودن یک عنصر به انتهای لیست (لیست اصلی را تغییر می‌دهد)
- متد `insert(i, x)` – درج عنصر `x` در موقعیت اندیس `i` (عناصر بعدی یک جایگاه به جلو می‌روند)
- متد `extend` – افزودن تمام عناصر یک ایتریبل به انتهای لیست. (معادل عملگر `+=` است)

```
>>> fruits = ["apple", "banana"]

>>> fruits.append("cherry")
>>> fruits
['apple', 'banana', 'cherry']

>>> fruits.insert(2, 'orange')
>>> fruits
['apple', 'banana', 'orange', 'cherry']

>>> fruits.insert(-1, 'coconut')
>>> fruits
['apple', 'banana', 'orange', 'coconut', 'cherry']

>>> fruits.extend(['grape', 'peach'])
>>> fruits
['apple', 'banana', 'orange', 'coconut', 'cherry', 'grape', 'peach']
```

تفاوت کلیدی متدهای append و extend: همانطور که در مثال زیر مشاهده می‌کنید، متد append آن لیست را به عنوان یک عنصر واحد اضافه می‌کند اما متد extend هر عنصر را جداگانه اضافه می‌کند.

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]

>>> a.append([4, 5])
>>> a
[1, 2, 3, [4, 5]]

>>> b.extend([4, 5])
>>> b
[1, 2, 3, 4, 5]
```

۲) متدهای حذف عنصر:

- متد remove(x) – حذف اولین رخداد مقدار x (اگر مقدار پیدا نشود، خطای ValueError می‌دهد)

- متد `pop(i)` – حذف عنصر در اندیس `i` که پیش‌فرض آن، آخرین عنصر است
- متد `clear` – حذف تمام عناصر. لیست خالی باقی می‌ماند اما شیء لیست در حافظه پابرجاست

```
>>> fruits = ["apple", "banana", "cherry", "date", "apple"]

>>> fruits.remove("apple")
>>> fruits
['banana', 'cherry', 'date', 'apple']

>>> fruits.remove("orange")
ValueError: list.remove(x): x not in list

>>> fruits.pop()
'apple'

>>> fruits
['banana', 'cherry', 'date']

>>> fruits.pop(0)
'banana'

>>> fruits
['cherry', 'date']

>>> fruits.clear()
>>> fruits
[]
```


تذکر مهم: همه متدهایی که لیست را تغییر می‌دهند (به جز متد `pop`) مقدار `None` برمی‌گردانند!

۳) متدهای جستجو و شمارش:

- متد `index(x)` – برگرداندن اندیس اولین رخداد `x` (اگر پیدا نشود، خطای `ValueError`)
- متد `count(x)` – شمارش تعداد رخداد مقدار `x`

```
>>> colors = ["red", "green", "blue", "green", "yellow"]

>>> colors.index("green")       # search from beginning
1

>>> colors.index("green", 2)    # search from index 2
3

>>> colors.count("green")
2
```

۴) متدهای مرتب‌سازی و معکوس‌سازی:

- متد sort – مرتب‌سازی درجا
- متد reverse – معکوس‌سازی درجا

```
>>> nums = [3, 1, 4, 1, 5, 9, 2, 6]

>>> nums.sort()
>>> nums
[1, 1, 2, 3, 4, 5, 6, 9]

>>> nums.sort(reverse=True)
>>> nums
[9, 6, 5, 4, 3, 2, 1, 1]

>>> nums.reverse()
>>> nums
[1, 1, 2, 3, 4, 5, 6, 9]
```

متد sort یک آرگومان key دریافت می‌کند که به وسیله آن می‌توان رفتار پیش‌فرض متد را عوض کرد.

```
>>> words = ["banana", "apple", "cherry", "date"]
>>> words.sort()                # مرتب‌سازی بر اساس حروف الفبا
>>> words
['apple', 'banana', 'cherry', 'date']

>>> words.sort(key=len)         # مرتب‌سازی بر اساس طول رشته
>>> words
['date', 'apple', 'banana', 'cherry']
```

در این مثال، به جای رفتار پیش‌فرض که ترتیب الفبایی کلمات را لحاظ می‌کند، از متد sort خواسته شده که مرتب‌سازی را بر اساس طول استرینگ انجام دهد.

۵) متد کپی: متد copy برای ایجاد یک کپی سطحی (shallow copy) از list در پایتون استفاده می‌شود.

```
>>> original = [1, 2, 3, [4, 5]]

>>> shallow = original.copy()
>>> shallow
[1, 2, 3, [4, 5]]
```

سه روش دیگر نیز برای ایجاد کپی سطحی وجود دارد:

```
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> c = a[:]
>>> d = list(a)
```

برای کپی کامل باید از تابع deepcopy ماژول copy استفاده کنید.

۶) توابع پیش‌ساخته مرتبط با list: علاوه بر متدهای اختصاصی لیست‌ها، بسیاری از توابع پیش‌ساخته پایتون نیز می‌توانند روی آنها اعمال شوند.

```
>>> nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]

>>> len(nums)
10

>>> min(nums)
1

>>> max(nums)
9

>>> sum(nums)
39

>>> sorted(nums)                        # new sorted list
[1, 1, 2, 3, 3, 4, 5, 5, 6, 9]

>>> list(reversed(nums))                # new reversed list
[3, 5, 6, 2, 9, 5, 1, 4, 1, 3]

>>> list(enumerate(["a", "b", "c"], start=1))
[(1, 'a'), (2, 'b'), (3, 'c')]

>>> list(zip([1, 2, 3], ["a", "b", "c"]))
[(1, 'a'), (2, 'b'), (3, 'c')]

>>> list(filter(lambda x: x > 3, nums))
[4, 5, 9, 6, 5, 3]

>>> list(map(lambda x: x ** 2, [1, 2, 3, 4]))
[1, 4, 9, 16]
```

لیست‌های تودرتو در پایتون

لیست‌ها می‌توانند شامل listهای دیگر باشند که برای نمایش ماتریس‌ها و ساختارهای چندبعدی به کار می‌روند:

```
>>> matrix = [
... [1, 2, 3],
... [4, 5, 6],
... [7, 8, 9],
... ]

>>> matrix[0]
[1, 2, 3]
>>> matrix[1][2]
6
>>> [row[1] for row in matrix]
[2, 5, 8]
```

ساخت پشته و صف در پایتون

لیست‌ها در پایتون می‌توانند مستقیماً به عنوان پُشته (Stack) استفاده شوند:

```
>>> stack = []
>>> stack.append("A")
>>> stack.append("B")
>>> stack.append("C")

>>> stack
['A', 'B', 'C']

# Stacks: Last In First Out (LIFO)
>>> stack.pop()
'C'
>>> stack
['A', 'B']
```

برای صف (Queue) می‌توانید از تابع deque ماژول collections استفاده کنید:

```
from collections import deque


>>> queue = deque(["A", "B", "C"])
>>> queue.append("D")

# Queues: First In First Out (FIFO)
>>> queue.popleft()
'A'
>>> queue
deque(['B', 'C', 'D'])
```

مثال واقعی از کاربرد لیست

در یک سناریوی فرضی، شما سیستمی برای مدیریت و تحلیل نمرات دانش‌آموزان می‌نویسید که باید نمرات را پاکسازی، تحلیل و رتبه‌بندی کند. در این مثال از list comprehension برای پاکسازی نمرات None، از متد sort و append برای ساخت لیست نتایج، از ترکیب sort با key و lambda برای رتبه‌بندی نهایی و از list comprehension تودرتو برای یکپارچه‌سازی همه نمرات استفاده شده است.

کدهای این مثال به همراه خروجی نهایی را می‌توانید از اینجا دانلود کنید.

سوالات متداول

1. تفاوت append و extend چیست؟

متد append آرگومانش را به عنوان یک عنصر واحد به انتهای list اضافه می‌کند؛ اگر لیستی بدهید، آن list به عنوان یک عنصر درج می‌شود. extend آرگومانش را به عنوان یک ایتریبل می‌بیند و تک‌تک عناصر آن را اضافه می‌کند. از extend برای الحاق دو list به جای حلقه‌ای از append استفاده کنید.

---

2. چرا list.sort مقدار None برمی‌گرداند؟

این طراحی عمدی است تا از اشتباه رایج sorted_list = my_list.sort جلوگیری شود. برنامه‌نویس مجبور می‌شود بین دو رویکرد آگاهانه انتخاب کند: list.sort که درجا مرتب می‌کند (بهینه از نظر حافظه)، یا sorted که list جدید می‌سازد و list اصلی را دست‌نخورده می‌گذارد.

---

3. تفاوت remove و pop چیست؟

متد remove یک مقدار می‌گیرد و اولین رخداد آن را حذف می‌کند؛ اگر مقدار وجود نداشته باشد ValueError می‌دهد و مقداری برنمی‌گرداند. متد pop یک اندیس می‌گیرد، عنصر را حذف می‌کند و آن را برمی‌گرداند؛ اگر اندیس خارج از محدوده باشد IndexError می‌دهد. وقتی به مقدار حذف‌شده نیاز دارید pop مناسب است. وقتی می‌دانید مقدار چیست ولی اندیسش را نمی‌دانید remove مناسب است.

---

4. متد copy چه تفاوتی با deepcopy دارد؟

list.copy یک کپی سطحی می‌سازد: لیست جدیدی ایجاد می‌شود اما عناصر درون آن همان ارجاع‌های اصلی هستند نه کپی از آن‌ها. اگر list شامل عناصر تغییرپذیر مثل لیست‌های تودرتو باشد، تغییر آن عناصر در هر دو نسخه منعکس می‌شود. copy.deepcopy تمام سلسله مراتب شیء را بازسازی می‌کند و دو نسخه کاملاً مستقل می‌شوند.

---

5. آیا لیست می‌تواند کلید دیکشنری باشد؟

خیر. لیست‌ها تغییرپذیر هستند و بنابراین قابل هش نیستند، پس نمی‌توانند کلید دیکشنری یا عضو مجموعه (set) باشند. اگر به دنباله‌ای نیاز دارید که کلید دیکشنری باشد، از تاپل استفاده کنید که تغییرناپذیر و قابل هش است.

---

6. بهترین روش حذف عناصر از list در حین پیمایش چیست؟

هرگز در حین پیمایش مستقیم با حلقه `for` از لیست، عناصری را حذف نکنید؛ اندیس‌گذاری داخلی به هم می‌ریزد و عناصری نادیده گرفته می‌شوند. روش توصیه‌شده، ساخت `list` جدید با `list comprehension` است.

جهت کسب اطلاعات بیشتر می‌توانید به **مستندات رسمی پایتون برای نوع داده لیست (`list`)** مراجعه کنید.

دسته: انواع داده - پایتون - دانش‌نامه برنامه‌نویسی