تابع list

آخرین به‌روزرسانی: ۲۹ اردیبهشت ۱۴۰۵

تابع `list` در پایتون یک **تابع پیش‌ساخته** (`built-in`) است که یک ایتریبل را دریافت کرده و از عناصر آن یک لیست جدید می‌سازد، یا در صورت فراخوانی بدون آرگومان، یک لیست خالی برمی‌گرداند. لیست‌ها یکی از پرکاربردترین و انعطاف‌پذیرترین ساختارهای داده در پایتون هستند که می‌توانند عناصر از انواع مختلف را در کنار هم نگه دارند.

فهرست مطالب:

سینتکس تابع `list`

سینتکس کلی تابع `list` به صورت زیر است:

```
list(iterable)
```

پارامترها یا آرگومان‌های این سینتکس عبارتند از:

- آرگومان اختیاری `iterable` هر شیء قابل پیمایش است مانند استرینگ، تاپل، رنج، مجموعه، دیکشنری، جنریتور یا هر آبجکت دیگری که پروتکل `iteration` را پیاده‌سازی کرده باشد. اگر آرگومانی وارد نشود، یک لیست خالی `[]` برگردانده می‌شود.

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

راهنمای جامع لیست (list)

در مورد نوع داده لیست یا همان list در پایتون بیشتر بدانید: **لیست (list)**

یادآوری: نوع داده لیست در پایتون، یک داده قابل تغییر (mutable) است؛ یعنی می‌توانید پس از ساخت، عناصر آن را اضافه، حذف یا ویرایش کنید.

کاربردهای تابع list

ساده‌ترین مثال از کاربرد این تابع می‌تواند حالت زیر باشد:

```
>>> list()
[]
>>> list((1, 2, 3))
[1, 2, 3]
>>> list("Hello")
['H', 'e', 'l', 'l', 'o']
>>> list({3, 1, 4, 1, 5, 9, 2, 6})
[1, 2, 3, 4, 5, 6, 9]
```

یا می‌توان خروجی توابعی مانند range یا map یا filter را که ایتریتور هستند، با تابع پایتونی list به یک لیست تبدیل کرد:

```
>>> list(range(1, 6))
[1, 2, 3, 4, 5]
>>> list(map(lambda x: x ** 2, range(1, 6)))
[1, 4, 9, 16, 25]
>>> list(filter(lambda x: x % 2 == 0, range(1, 11)))
[2, 4, 6, 8, 10]
```

رایج‌ترین کاربردهای تابع list عبارتند از:

- تبدیل تاپل، مجموعه، رنج یا جنریتور به لیست قابل تغییر،
- مادی‌سازی خروجی ایتریتورهایی مانند map و filter و zip (مثال واقعی را ببینید!)،
- ایجاد یک کپی سطحی از یک لیست موجود،

- تبدیل استرینگ به لیست کاراکترها برای پردازش حرف به حرف.

تبدیل تاپل به لیست: تاپل‌ها تغییرناپذیر (`immutable`) هستند. اگر بخواهید روی عناصر یک تاپل عملیات تغییر انجام دهید، ابتدا باید آن را به لیست تبدیل کنید:

```
>>> coordinates = (35.6892, 51.3890)
>>> coords_list = list(coordinates)
>>> coords_list
[35.6892, 51.389]
>>> coords_list.append(1200)
>>> coords_list
[35.6892, 51.389, 1200]
```

تبدیل دیکشنری به لیست: به صورت پیش‌فرض، اثر دادن تابع پایتونی `list` روی یک دیکشنری فقط کلیدها را برمی‌گرداند:

```
>>> person = {"name": "ali", "age": 28, "city": "Tehran"}
>>> list(person)
['name', 'age', 'city']

# برای دریافت مقادیر
>>> list(person.values())
['ali', 28, 'Tehran']

# برای دریافت جفت کلید-مقدار
>>> list(person.items())
[('name', 'ali'), ('age', 28), ('city', 'Tehran')]
```

کپی کردن لیست با تابع `list`: یکی از کاربردهای مهم تابع `list` ساختن یک کپی مستقل از لیست موجود است. این روش یک کپی سطحی یا اصطلاحاً `shallow copy` ایجاد می‌کند:

```
>>> original = [1, 2, 3, 4, 5]
>>> copy = list(original)
>>> copy.append(6)

>>> print(original)
[1, 2, 3, 4, 5]

>>> print(copy)
[1, 2, 3, 4, 5, 6]
```

نکته مهم: اگر لیست شامل آبجکت‌های تودرتو (مانند لیست داخل لیست) باشد، تابع `list` فقط یک کپی سطحی می‌سازد و آبجکت‌های داخلی همچنان به منبع اصلی اشاره دارند. برای کپی کامل باید از تابع `deepcopy` ماژول `copy` استفاده کنید.

مثال واقعی از تابع `list`

در یک سناریوی فرضی، شما یک سیستم مدیریت دانش‌آموزان دارید. اطلاعات از یک فایل CSV خوانده می‌شود و باید آن‌ها را پردازش کنید. در این مسیر از تابع `list` برای مادی‌سازی (`materialization`) خروجی `iterator` ها استفاده می‌کنید:

```python
# data from a CSV file
raw_data = "Jafar,18,85.5;Ali,17,92.0;Reza,19,78.3"

# convert CSV data from string to a list
records = list(raw_data.split(";"))

# convert each record to a dictionary
students = list(map(
    lambda r: dict(zip(["name", "age", "score"], r.split(","))),
    records
))

# filter for top students with score >= 80
top_students = list(filter(
    lambda s: float(s["score"]) >= 80,
    students
))

for s in top_students:
    print(f"{s['name']} - score: {s['score']}")
```

نمونه خروجی این کد به صورت زیر خواهد بود:

```
Jafar - score: 85.5
Ali - score: 92.0
```

در این مثال، تابع `list` در سه جای مختلف استفاده شده: برای تبدیل خروجی `split`، برای مادی‌سازی نتیجه `map` و برای مادی‌سازی نتیجه `filter`؛ و هر بار یک ساختار قابل استفاده و قابل تغییر ایجاد کرده است.

تفاوت تابع `list` با `list comprehension` و `tuple`

هر سه روش می‌توانند مجموعه‌ای از عناصر را نگه دارند، اما تفاوت‌های مهمی دارند:

```python
numbers = list(range(1, 6))
# output: [1, 2, 3, 4, 5]

squares = [x ** 2 for x in range(1, 6)]
# output: [1, 4, 9, 16, 25]

immutable = tuple(range(1, 6))
# output: (1, 2, 3, 4, 5)
```

به طور کلی:

- تابع `list` بهترین انتخاب برای تبدیل یک ایتریبل موجود (مانند تاپل، رنج یا ایتریتور) به لیست است.
- وقتی می‌خواهید هم‌زمان با ساخت لیست، تبدیل یا فیلتری روی عناصر اعمال کنید، `list comprehension` انتخاب بهتری است.
- تابع `tuple` زمانی مناسب‌تر است که مجموعه‌ای از داده‌ها داشته باشید که نباید تغییر کنند؛ مثل مختصات جغرافیایی یا رنگ‌های RGB ثابت.

سوالات متداول

1. تفاوت تابع `list` و `[]` در پایتون چیست؟

هر دو یک لیست می‌سازند، اما `[]` برای ساخت لیست خالی یا لیست با عناصر `literal` (مثل `[1, 2, 3]`) استفاده می‌شود. تابع `list` زمانی به کار می‌رود که بخواهید یک ایتریبل موجود را به لیست تبدیل کنید. از نظر سرعت، `[]` کمی سریع‌تر است.

---

2. آیا تابع `list` یک کپی واقعی از لیست می‌سازد؟

این تابع یک کپی سطحی ایجاد می‌کند. این یعنی لیست جدید یک آبجکت مستقل است، اما اگر لیست حاوی آبجکت‌های قابل تغییر (مانند لیست‌های تودرتو) باشد، آن آبجکت‌های داخلی هنوز به منبع اصلی اشاره دارند.

---

3. تفاوت `list` و `tuple` چیست؟

هر دو دنباله‌ای مرتب از عناصر هستند، اما `list` قابل تغییر (`mutable`) است؛ یعنی می‌توان عناصر را اضافه، حذف یا ویرایش کرد ولی `tuple` تغییرناپذیر (`immutable`) است. تاپل‌ها معمولاً کمی سریع‌تر هستند و برای داده‌هایی که نباید تغییر کنند مناسب‌ترند.

---

4. آیا می‌توان تابع `list` را روی یک جنریتور اعمال کرد؟

بله. یکی از رایج‌ترین کاربردهای تابع `list` مادی‌سازی جنریتورها و ایتریتورها است. وقتی `list(generator)` می‌نویسید، تمام مقادیر جنریتور یک بار محاسبه شده و در لیست ذخیره می‌شوند.

جهت کسب اطلاعات بیشتر می‌توانید به **مستندات رسمی پایتون برای تابع `list`** مراجعه کنید.

دسته: توابع پیش‌ساخته - پایتون - دانش‌نامه برنامه‌نویسی