استرینگ (str)

آخرین به‌روزرسانی: ۲۶ اردیبهشت ۱۴۰۵

نوع داده استرینگ یا رشته (str) در پایتون، نوع داده‌ای است که برای نمایش متن (دنباله‌ای از کاراکترها) استفاده می‌شود. رشته‌ها در پایتون تغییرناپذیر هستند؛ یعنی پس از ساخته شدن، نمی‌توان محتوای آن‌ها را مستقیماً تغییر داد. پایتون از استاندارد یونیکد پشتیبانی کامل دارد، بنابراین هر رشته‌ای می‌تواند شامل کاراکترهای هر زبانی از جمله فارسی، عربی، چینی، ایموجی و نمادهای ریاضی باشد. نوع داده str یکی از پرکاربردترین انواع داده در پایتون است و متدهای بسیار غنی و متنوعی برای پردازش متن دارد.

فهرست مطالب:

ایجاد استرینگ در پایتون

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

۱) نوشتار مستقیم با کوتیشن تک یا دوتایی:

```
>>> s1 = 'Hello, World!'
>>> s2 = "Hello, World!"
```

۲) رشته‌های چندخطی با سه کوتیشن:

```
>>> multiline = """This is
... a multiline
... string."""
```

۳) رشته‌های خام با پیشوند r: در رشته‌های خام (Raw String)، کاراکترهای escape مثل `\n` یا `\t` به صورت تحت‌اللفظی تفسیر می‌شوند. کاربرد اصلی آنها در regex و آدرس‌دهی به مسیرهای ویندوزی است:

```
>>> path = r"C:\Users\name\Documents"
>>> print(path)
C:\Users\name\Documents

>>> normal = "C:\Users\name\Documents"
>>> print(normal)
C:\Users
ame\Documents
```

در مثال بالا مشاهده می‌کنید که عدم استفاده از پیشوند r باعث اشتباه مفسر شده و کاراکترهای `\n` به عنوان خط جدید قلمداد شده است.

۴) تابع str: سازنده نوع داده استرینگ در پایتون، تابع `str` است:

```
str(object)
str(object, encoding, errors="strict")
```

مقدار بازگشتی تابع str همیشه یک رشته متنی (استرینگ) از نوع str است.

راهنمای جامع تابع str

در مورد تابع str که سازنده یا Constructor برای استرینگ در پایتون است بیشتر بدانید: **آشنایی با تابع str در پایتون**

نکته: رشته‌های متنی در پایتون کاملاً از استاندارد یونیکد پشتیبانی می‌کنند. می‌توانید مستقیماً متن فارسی، ایموجی یا هر کاراکتر یونیکدی را در استرینگ خود بنویسید.

عملیات پایه روی استرینگ

در ادامه، تعدادی مثال از عملیات روی رشته‌های متنی را با هم مرور خواهیم کرد.

الحاق و تکثیر: عملگرهای + و * به ترتیب عملیات الحاق (Concatenation) و تکثیر را روی استرینگ‌ها انجام می‌دهند:

```
>>> "Hello" + " " + "World"
'Hello World'
>>> "ha" * 3
'hahaha'
>>> "-" * 20
'--------------------'
```

دسترسی با اندیس و برش: رشته‌ها دنباله‌ای از کاراکترها هستند و مثل لیست از اندیس‌گذاری پشتیبانی می‌کنند:

```
>>> s = "Python"

>>> s[0]       # first character
'P'
>>> s[-1]      # last character
'n'
>>> s[1:4]     # slicing
'yth'
>>> s[:3]      # from beginning to index 3
'Pyt'
>>> s[3:]      # from index 3 to end
'hon'
>>> s[::-1]    # reverse string
'nohtyP'
>>> s[::2]     # one in between
'Pto'
```

طول رشته و بررسی عضویت: با استفاده از عملگرهای منطقی in و not in می‌توان بررسی کرد که آیا یک رشته از کاراکترها درون یک رشته بزرگتر قرار دارند یا خیر:

```
>>> s = "Hello, World!"

>>> len(s)
13
>>> "World" in s
True
>>> "Python" not in s
True
```

مقایسه رشته‌ها: در پایتون، با استفاده از عملگردهای مقایسه‌ای مانند > یا < می‌توان عملیات مقایسه بین رشته‌های متنی را انجام داد. مقایسه رشته‌ها بر اساس ترتیب یونیکد کاراکترها انجام می‌شود:

```
>>> "apple" == "apple"
True
>>> "apple" < "banana"
True
>>> "abc" < "abd"
True
>>> "Z" < "a"
True
```

یادآوری: در ترتیب کاراکترهای یونیکدی، حروف الفبای انگلیسی بزرگ قبل از حروف کوچک قرار دارند.

متدهای پرکاربرد استرینگ

نوع `str` در پایتون بیش از ۴۰ متد دارد. در اینجا مهم‌ترین و پرکاربردترین آنها را بررسی می‌کنیم.

۱) متدهای تغییر حالت حروف:

- متد `upper` – تبدیل حروف کوچک به بزرگ
- متد `lower` – تبدیل حروف بزرگ به کوچک
- متد `capitalize` – حرف اول استرینگ بزرگ، باقی حروف کوچک
- متد `title` – حرف اول همه کلمات بزرگ، باقی حروف کوچک
- متد `swapcase` – تبدیل حالت هر حرف به عکس آن

```
>>> "Hello World".upper()
'HELLO WORLD'

>>> "Hello World".lower()
'hello world'

>>> "hello world".capitalize()
'Hello world'

>>> "hello world from python".title()
'Hello World From Python'

>>> "Hello World".swapcase()
'hELLO wORLD'
```

۲) متدهای جستجو و بررسی:

- متد `find` و `rfind` – یافتن اندیس اولین (یا آخرین) رخداد زیررشته
- متد `index` و `rindex` – مثل `find` ولی در صورت نیافتن، خطای `ValueError` می‌دهد
- متد `count` – شمارش تعداد رخداد زیررشته
- متد `startswith` و `endswith` – بررسی شروع یا پایان رشته

```
>>> s = "Hello, Hello, World"

>>> s.find("Hello")
0
>>> s.rfind("Hello")
7
>>> s.find("Python")
-1                              # not found
>>> "Hello".index("xyz")
ValueError: substring not found   # not found
>>> "banana".count("a")
3
>>> "Hello, World".startswith("Hello")
True
>>> "image.jpg".endswith((".jpg", ".png", ".gif"))
True
```

۳) متدهای حذف فضای خالی:

- متد `strip` و `lstrip` و `rstrip` – حذف فضای خالی (یا کاراکترهای دلخواه) از ابتدا و انتها
- متد `removeprefix` و `removesuffix` – حذف پیشوند یا پسوند دقیق (پایتون 3.9+)

```
>>> "  Hello, World!  ".strip()
'Hello, World!'

>>> "  Hello".lstrip()
'Hello'

>>> "Hello  ".rstrip()
'Hello'

>>> "***Hello***".strip("*")
'Hello'

>>> "Hello, World!".removeprefix("Hello, ")
'World!'

>>> "report_2024.pdf".removesuffix(".pdf")
'report_2024'
```

۴) متدهای جایگزینی و تبدیل:

- متد `replace` – جایگزینی زیررشته
- متد `translate` – جایگزینی کاراکتر به کاراکتر با استفاده از جدول ترجمه

```
>>> "Hello, World!".replace("World", "Python")
'Hello, Python!'

>>> table = str.maketrans("aeiou", "12345")
>>> "Hello, World!".translate(table)
'H2ll4, W4rld!'

>>> remove_table = str.maketrans("", "", "aeiou")   # حذف حروف صدادار
>>> "Hello, World!".translate(remove_table)
'Hll, Wrld!'
```

۵) متدهای تقسیم و اتصال:

- متد `split` – تقسیم رشته به لیست
- متد `splitlines` – تقسیم بر اساس کاراکترهای خط جدید
- متد `partition` و `rpartition` – تقسیم به سه بخش (قبل، جداکننده، بعد)
- متد `join` – اتصال عناصر یک ایتریبل با جداکننده

```
>>> "Hello World Python".split()
['Hello', 'World', 'Python']

>>> "a,b,c,d".split(",")
['a', 'b', 'c', 'd']

>>> "line1\nline2\nline3".splitlines()
['line1', 'line2', 'line3']

>>> "Hello: World".partition(":")
('Hello', ':', ' World')

>>> "Hello: World: Python".rpartition(":")
('Hello: World', ':', ' Python')

>>> ", ".join(["Python", "Java", "C++"])
'Python, Java, C++'

>>> "".join(["a", "b", "c"])
'abc'
```


نکته مهم: برای الحاق تعداد زیادی رشته، `join` بسیار بهینه‌تر از عملگر + در حلقه است. توصیه می‌شود همیشه از متد `join` به جای عملگر += در حلقه استفاده کنید.

۶) متدهای تراز کردن و پُر کردن:

- متد `center` و `ljust` و `rjust` – تراز کردن متن در عرض مشخص
- متد `zfill` – پُر کردن با صفر از سمت چپ (برای اعداد)

```
>>> "Hello".center(20, "-")
'-------Hello--------'

>>> "Hello".ljust(20, ".")
'Hello...............'

>>> "Hello".rjust(20, ".")
'...............Hello'

>>> "42".zfill(8)
'00000042'
```

۷) متدهای بررسی محتوا: این متدها فقط `True` یا `False` بر می‌گردانند.

- متد `isalnum` – آیا تمام کاراکترهای رشته فقط حروف الفبا و اعداد هستند یا خیر
- متد `isalpha` – آیا تمام کاراکترهای رشته فقط حروف الفبا هستند یا خیر
- متد `isdigit` – آیا تمام کاراکترهای رشته فقط اعداد هستند یا خیر
- متد `isspace` – آیا تمام کاراکترهای رشته فقط فاصله، تب و فضای خالی هستند یا خیر
- متد `istitle` – آیا فقط حرف اول همه کلمات رشته به حالت بزرگ هستند یا خیر
- متد `isnumeric` – آیا تمام کاراکترهای رشته مقادیر عددی معتبر هستند یا خیر
- متد `isupper` و `islower` – آیا تمام کاراکترهای رشته حروف بزرگ/کوچک هستند یا خیر

```
>>> "Hello123".isalnum()
True
>>> "Hello 123".isalnum()
False
>>> "Hello123".isalpha()
False
>>> "123.45".isdigit()
False
>>> "  \t\n".isspace()
True
>>> "Hello World".istitle()
True
>>> "HELLO".isupper()
True
>>> "hello".isnumeric()
False
```

۸) متدهای کدگشایی و کدگذاری: استرینگ‌های پایتونی یونیکد هستند، اما برای ارسال از طریق شبکه یا ذخیره در فایل باید به بایت تبدیل شوند. این عملیات و معکوس آن، توسط متدهای encode و decode انجام می‌شود.

```
>>> "سلام".encode("utf-8")
b'\xd8\xb3\xd9\x84\xd8\xa7\xd9\x85'

>>> "Hello".encode("ascii")
b'Hello'

>>> "سلام".encode("ascii")
UnicodeEncodeError: ...

>>> "سلام".encode("ascii", errors="replace")
b'????'

>>> b'\xd8\xb3\xd9\x84\xd8\xa7\xd9\x85'.decode("utf-8")
'سلام'
```

فرمت‌بندی استرینگ در پایتون

پایتون چندین روش برای فرمت‌بندی استرینگ‌ها دارد که مدرن‌ترین، خواناترین و سریع‌ترین آنها، استفاده از f-string است که از نسخه 3.6 به بعد پایتون، وارد این زبان شده است.

علاوه بر f-string، دو متد format و format_map نیز برای فرمت‌بندی استرینگ‌ها در پایتون وجود دارد. متد format_map تقریباً مشابه format عمل می‌کند ولی از یک دیکشنری یا mapping استفاده می‌کند:

```
>>> "{0} and {1}".format("Python", "Java")
'Python and Java'

>>> "{name} is {age} years old".format(name="Ali", age=30)
'Ali is 30 years old'

>>> "{:.2f}".format(3.14159)
'3.14'

>>> data = {"name": "Ali", "city": "Tehran"}
>>> "{name} lives in {city}".format_map(data)
'Ali lives in Tehran'
```

مثال واقعی از کاربرد استرینگ

در یک سناریوی فرضی، شما داده‌های خام کاربران را از یک فایل CSV دریافت می‌کنید. این داده‌ها دارای مشکلاتی مثل فضاهای اضافه، حالت حروف نادرست و ایمیل‌های نامعتبر هستند. باید آن‌ها را پاکسازی و اعتبارسنجی کنید:

```python
raw_users = [
    "  ALI REZAEI  |  ali.rezaei@gmail.com  |  09123456789  ",
    "mina mohammadi|MINA.M@YAHOO.COM|09987654321",
    "  Reza Ahmadi  |  invalid-email  |  0911111111  ",
    "MARYAM HOSSEINI|maryam@outlook.com|09355555555",
]

def clean_name(name):
    return name.strip().title()

def clean_email(email):
    return email.strip().lower()

def clean_phone(phone):
    p = phone.strip()
    return p if p.startswith("09") and len(p) == 11 and p.isdigit() else None

def is_valid_email(email):
    return "@" in email and "." in email.split("@")[-1]

print(f"{'Name':<20} {'Email':<30} {'Phone':<15} {'Valid'}")
print("-" * 75)

for row in raw_users:
    parts = row.split("|")
    if len(parts) != 3:
        continue

    name  = clean_name(parts[0])
    email = clean_email(parts[1])
    phone = clean_phone(parts[2])
    valid = is_valid_email(email) and phone is not None

    phone_display = phone if phone else "INVALID"
```

```
    status = "YES" if valid else "NO"

    print(f"{name:<20} {email:<30} {phone_display:<15} {status}")
```

نمونه خروجی این کد به صورت زیر خواهد بود:

```
Name                 Email                          Phone           Valid
-------------------------------------------------------------------------
Ali Rezaei           ali.rezaei@gmail.com           09123456789     YES
Mina Mohammadi       mina.m@yahoo.com               09987654321     YES
Reza Ahmadi          invalid-email                  INVALID         NO
Maryam Hosseini      maryam@outlook.com             09355555555     YES
```

در این مثال از ترکیب متدهای `isdigit` – `startswith` – `split` – `lower` – `title` – `strip` و `f-string` برای پاکسازی و نمایش داده‌ها استفاده شد.

سوالات متداول

1. چرا رشته‌ها در پایتون تغییرناپذیر هستند؟

تغییرناپذیری رشته‌ها چند مزیت مهم دارد: اول، رشته‌ها می‌توانند کلید دیکشنری باشند چون قابل هش هستند؛ دوم، پایتون می‌تواند رشته‌های یکسان را در حافظه به اشتراک بگذارد که باعث کاهش مصرف حافظه می‌شود؛ سوم، ایمنی در محیط‌های `multi-threading` تضمین می‌شود. هر متدی که روی رشته فراخوانی می‌شود، رشته‌ی جدیدی برمی‌گرداند، نه اینکه رشته‌ی اصلی را تغییر دهد.

---

2. تفاوت متدهای `find` و `index` چیست؟

هر دو اندیس اولین رخداد زیررشته را برمی‌گردانند، اما وقتی زیررشته پیدا نشود رفتار متفاوتی دارند: `find` مقدار `-1` برمی‌گرداند، در حالی که `index` خطای `ValueError` ایجاد می‌کند. در موقعیت‌هایی که نبود زیررشته حالت عادی است، از `find` استفاده کنید. اگر نبود آن نشانه خطا است، `index` مناسب‌تر است.

---

3. بهترین روش فرمت‌بندی استرینگ در پایتون چیست؟

در پایتون مدرن (3.6+)، `f-string` توصیه می‌شود چون هم خواناتر و هم سریع‌تر از `format` و عملگر `%` است. برای موقعیت‌هایی که قالب را باید در زمان اجرا تعیین کنید (مثلاً قالب از یک فایل پیکربندی می‌آید)، از متدهای `format` یا `format_map` استفاده کنید.

---

4. تفاوت متدهای `split` و `partition` چیست؟

`split` رشته را به لیستی از زیررشته‌ها تقسیم می‌کند و می‌توانید تعداد تقسیم‌ها را محدود کنید. `partition` همیشه دقیقاً سه مقدار برمی‌گرداند: بخش قبل از جداکننده، خود جداکننده و بخش بعد از آن. اگر جداکننده پیدا نشود، `partition` رشته اصلی و دو رشته خالی را برمی‌گرداند.

---

5. چطور یک استرینگ را معکوس کنم؟

پایتون متد مستقیمی برای معکوس کردن رشته ندارد، اما از `slicing` می‌توان استفاده کرد: `s[::-1]`.

---

6. تفاوت متدهای `isdigit` و `isdecimal` و `isnumeric` چیست؟

هر سه بررسی می‌کنند که آیا استرینگ عددی است، اما در نوع کاراکترهای قابل قبول تفاوت دارند. `isdecimal` سخت‌گیرترین است و فقط ارقام اعشاری استاندارد (0-9) را می‌پذیرد. `isdigit` اندکی گسترده‌تر بوده و اعداد توان‌دار یونیکد را هم می‌پذیرد. `isnumeric` گسترده‌ترین است و کاراکترهایی مثل کسرهای یونیکد (½، ¾) را هم شامل می‌شود. برای بررسی ورودی کاربر و تبدیل با `int`، استفاده از `isdecimal` توصیه می‌شود.

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

دسته: انواع داده - پایتون - دانش‌نامه برنامه‌نویسی