رشته

پایتون نوع داده‌ای به نام کاراکتر ندارد و حتی اگر بخواهید یک کاراکتر را ذخیره کنید، باید از رشته استفاده کنید.رشته ساختار داده‌ای تغییرناپذیر (*immutable*) است. به این معنی که پس از ساخت نمی‌توان آن را تغییر داد. برای تغییر دادن مقدار داخل یک متغیر از نوع رشته، باید رشته‌ای جدید ساخته و رشته قبلی را مساوی رشته جدید قرار دهیم. رشته‌ها در پایتون بین دو علامت `'` یا `"` قرار می‌گیرند. که بسته به استفاده‌ی آن رشته می‌توانید از هرکدام استفاده کنید.

- **نکته**: رشته‌هایی که با کوتیشن دوتایی باز و بسته می‌شوند، می‌توانند شامل کوتیشن تکی باشند و برعکس:

Copy Python

```
"This is a string."
'This is also a string.'
" ' " #=> strings between "" can include '
' " ' #=> strings between '' can include "
```

رشته‌ها و متغیرهایی که حاوی رشته هستند را می‌توان با عملگر `+` به هم چسباند و رشته‌ها بدون هیچ عملگری بین آنها نیز به هم می‌چسبند ولی چسبیدن بدون عملگر روی متغیرها کار نمی‌کند:

```
>>> "Hello " + "world!"
'Hello world!'
>>>  "Hello " "world!"
'Hello world!'
>>> c = "hi"
>>> c += "bye"
>>> c
'hibye'
>>> a = "Hello "
>>> b = "world!"
>>> a+b
'Hello world!'
>>> a b #this can't be done on varibles
  File "<stdin>", line 1
    a b
      ^
SyntaxError: invalid syntax
```

با رشته‌ها می‌توان مانند لیست برخورد کرد و با اندیس به عناصر آنها دسترسی داشت. همچنین اگر اندیس منفی باشد، از آخر رشته یا لیست حساب می‌شود. مثلا اندیس -1، به آخرین کاراکتر یک رشته یا آخرین عنصر یک لیست اشاره می‌کند.

- **نکته**: `slicing` به بریدن بخشی از یک لیست گفته می‌شود که در درسنامه‌ی لیست به طور کامل به آن اشاره شده است.

```
>>> "This is a string"[0]
'T'
>>> "This is a string"[-1]
'g'
>>> a = "Salam"
```

```
>>> a[0:2] #string slicing
'Sa'
```

تابع `len` طول رشته را برمی‌گرداند:

```
>>> len("This is a string")
16
```

قالب‌دهی به رشته‌ها (*String Formatting*)

تابع `format` پس از هر رشته، آرگومان‌های خود را به ترتیب جایگزین `{}` داخل رشته می‌کند:

```
>>> "{} can be {}".format("Strings", "interpolated")
'Strings can be interpolated'
```

میتوان داخل `{}` شماره‌ی آرگومانی که باید جایگزین شود را وارد کرد. و میتوان به شکل `.nf` عددی را تا `n` رقم اعشار جایگزین کرد:

```
>>> "{0} be nimble, {0} be quick, {0} jump over the {1}".format("Jack", "candle stick")
'Jack be nimble, Jack be quick, Jack jump over the candle stick'
>>> "{0:.3f} is a number with 3 decimal digits".format(0.123456789)
'0.123 is a number with 3 decimal digits'
```

میتوان به جای استفاده از شماره‌ی آرگومان، از کلید آرگومان استفاده کرد. آرگومان‌های کلیدی(keyword arguments) جلوتر توضیح داده خواهد شد:

```
>>> "{name} wants to eat {food}".format(name="Bob", food="lasagna")
'Bob wants to eat lasagna'
```

این شکل از قالب دهی به رشته‌ها کاملا مانند تابع `format` عمل می‌کند ولی متعلق به پایتون ۲.۵ به قبل است و پیشنهاد نمی‌شود:

```
>>> "%s can be %s the %s way" % ("Strings", "interpolated", "old")
'Strings can be interpolated the old way'
```

پایتون از نسخه‌ی ۳.۶ به بعد از قابلیتی به نام `f-string` پشتیبانی می‌کند. هر رشته ای که در ابتدای آن `f` نوشته شود، داخل `{}` را یک `statement` در نظر گرفته و جواب آن را در جای `{}` قرار می‌دهد. برای مثال در کد پایین یک متغیر و تابعی از یک متغیر به عنوان `statement` نوشته شده است:

```
>>> name = "Reiko"
>>> f"She said her name is {name}."
'She said her name is Reiko.'
>>> f"{name} is {len(name)} characters long."
'Reiko is 5 characters long.'
```

کلیدواژه `in` مشخص می‌کند که رشته‌ی سمت چپ در رشته‌ی سمت راست وجود دارد یا نه:

```
>>> "sa" in "salam"
True
>>> "lam" in "salam"
True
>>> "l" in "salam"
True
>>> "pif" in "salam"
False
```

رشته‌های چند خطی (*Multiline Strings*)

می‌توانید رشته‌های چند خطی را در پایتون با `"""` یا `'''` تعریف کنید. (از این رشته‌ها می‌توان به عنوان کامنت هم استفاده کرد.)

```
>>> a = """ this i a
... multi line string.
... it is used as comment too."""
>>> a
' this i a \nmulti line string.\nit is used as comment too.'
```

توابع معروف رشته (*String Methods*)

تابع `strip()`

تابع `strip`، فاصله(`whitespace`)ها(`space`، `tab`، `enter` و ...) را از ابتدا و انتهای رشته حذف می‌کند.

```
>>> a = "\n   \n Hello, World!      "
>>> a
'\n   \n Hello, World!      '
>>> a.strip()
'Hello, World!'
>>> "\t salam   \t".strip()
'salam'
```

تابع `lower()`

تابع `lower` همه‌ی حروف رشته را کوچک می‌کند.

```
>>> a = "Hello World"
>>> a.lower()
'hello world'
```

تابع `upper()`

تابع `upper` همه‌ی حروف رشته را بزرگ می‌کند.

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

تابع `replace()`

تابع `replace` در داخل یک رشته آرگومان اول خود را به هر تعدادی که باشد، پیدا می‌کند و با آرگومان دوم خود جایگزین می‌کند.

```
>>> a = "Hello, World!"
>>> a.replace("o" , "j")
'Hellj, Wjrld!'
```

تابع `split()`

متد `split` رشته را بر اساس آرگومان اول خود جدا می‌کند و یک لیست از قسمت‌هایش برمی‌گرداند. اگر آرگومان اول آن وجود نداشته باشد، رشته را بر اساس `whitespace` جدا می‌کند.

```
>>> a = "Hello, World!"
>>> a.split(",")
['Hello', ' World!']
>>> a.split("o")
['Hell', ', W', 'rld!']
>>> a.split()
['Hello,', 'World!']
```