تاپل‌ها

در پایتون، تاپل‌ها بسیار شبیه به لیست‌ها هستند، با این تفاوت که، برخلاف لیست‌ها، آنها *غیرقابل تغییر* هستند و یعنی نمی‌توان آنها را تغییر داد. شما از تاپل‌ها برای نمایش اشیاءی که نباید تغییر کنند، مانند روزهای هفته یا تاریخ‌ها در تقویم استفاده خواهید کرد.

در این بخش، مروری کوتاهی بر موارد زیر خواهیم داشت:

1. ساخت تاپل‌ها
2. متدهای پایه‌ای تاپل
3. غیرقابل تغییر بودن
4. زمان استفاده از تاپل‌ها

بر اساس آنچه درباره لیست‌ها یاد گرفته‌اید، شما به نحوی به تاپل‌ها آشنا خواهید بود. ما می‌توانیم با رفتار بسیار مشابهی از آنها استفاده کنیم، با این تفاوت که تاپل‌ها غیرقابل تغییر هستند.

ساخت تاپل‌ها

ساخت تاپل‌ها با استفاده از پرانتز () و عناصری که توسط کاما از هم جدا شده‌اند، انجام می‌شود. به عنوان مثال:

In [1]:

```
# Create a tuple
t = (1,2,3)
```

In [2]:

```
# Check len just like a list
len(t)
```

Out[2]:

```
3
```

In [3]:

```
# Can also mix object types
t = ('one',2)

# Show
t
```

Out[3]:

```
('one', 2)
```

In [4]:

```
# Use indexing just like we did in lists
t[0]
```

Out[4]:

```
'one'
```

In [5]:

```
# Slicing just like a list
t[-1]
```

Out[5]:

```
2
```

متدهای پایه تاپل ها

تاپل‌ها دارای متدهای داخلی هستند، اما تعداد آنها به اندازه لیست‌ها نیست. بیایید به دو متد آنها نگاهی بیندازیم:

In [6]:

```
# Use .index to enter a value and return the index
t.index('one')
```

Out[6]:

```
0
```

In [7]:

```
# Use .count to count the number of times a value appears
t.count('one')
```

Out[7]:

```
1
```

غیرقابل جهش بودنImmutability

اهمیت غیرقابل تغییر بودن تاپل‌ها را نمی‌توان به اندازه کافی تاکید کرد. برای بیان این نکته:

In [8]:

```
t[0]= 'change'
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-8-1257c0aa9edd> in <module>()
----> 1 t[0]= 'change'

TypeError: 'tuple' object does not support item assignment
```

به دلیل این غیرقابل تغییری، تاپل‌ها قادر به رشد نیستند. یکباری که یک تاپل ساخته شد، نمی‌توانیم به آن عناصر اضافه کنیم.

In [9]:

```
1 t.append('nope')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call las
t)
<ipython-input-9-b75f5b09ac19> in <module>()
----> 1 t.append('nope')

AttributeError: 'tuple' object has no attribute 'append'
```

نکته بسیار مهم، تاپل ها فقط در سطح اول خود غیرقابل جهش می باشند و عملگرد سطوح بعدی به نوع داده آن سطح بستگی دارد

In [1]:

```
1 t = (1, 2, 3, (5,6), [10, 11, 12])
```

In [2]:

```
1 t[3]
```

Out[2]:

```
(5, 6)
```

In [4]:

```
1 t[3] = (55, 66)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
Cell In[4], line 1
----> 1 t[3] = (55, 66)

TypeError: 'tuple' object does not support item assignment
```

In [5]:

```
1 t[3][1] = 66
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
Cell In[5], line 1
----> 1 t[3][1] = 66

TypeError: 'tuple' object does not support item assignment
```

In [6]:

```
1 t[4]
```

Out[6]:

```
[10, 11, 12]
```

In [7]:

```
1 t[4][0] = 222
2
3 t
```

Out[7]:

```
(1, 2, 3, (5, 6), [222, 11, 12])
```

چه زمانی از تاپل ها

شاید بپرسید: "چرا باید از تاپل‌ها استفاده کنیم وقتی که تعداد کمتری از متدها در دسترس هستند؟" صادقانه بگوییم، تاپل‌ها به همان اندازه که لیست‌ها در برنامه‌نویسی استفاده می‌شوند، اما زمانی که غیرقابل تغییر بودن ضروری است، تاپل‌ها مورد استفاده قرار می‌گیرند. اگر در برنامه خود شما یک شیء را انتقال می‌دهید و نیاز دارید مطمئن شوید که آن تغییر نمی‌کند، در این صورت تاپل بهترین گزینه است. آنها منبع مناسبی از سلامت داده را فراهم می‌کنند.

اکنون باید قادر باشید تاپل‌ها را ایجاد و استفاده کنید و درکی از غیرقابل تغییری آنها داشته باشید.

در ادامه، دیکشنری‌ها!