لیست ها

در بحث رشته‌ها، مفهوم *دنباله* را در پایتون معرفی کردیم. لیست‌ها می‌توانند به عنوان نسخه‌ی عمومی‌ترین *دنباله* در پایتون در نظر گرفته شوند. بر خلاف رشته‌ها، آنها قابل تغییر هستند، به این معنا که عناصر درون یک لیست می‌توانند تغییر کنند!

در این بخش، موارد زیر را خواهیم آموخت:

1. ایجاد لیست‌ها
2. ایندکس‌گذاری و برش لیست‌ها
3. متدهای پایه‌ای لیست
4. تو در تو کردن لیست‌ها
5. مقدمه‌ای بر فهم لیست‌ها

لیست‌ها با استفاده از براکت‌ها [] و کاماها بین هر عنصر در لیست ساخته می‌شوند.

بیایید شروع به ساختن لیست‌ها کنیم!

In [1]:

```
# Assign a list to an variable named my_list
my_list = [1,2,3]
```

ما تازه یک لیست از اعداد صحیح ایجاد کردیم، اما در واقع لیست‌ها می‌توانند شامل انواع مختلفی از اشیاء باشند. به عنوان مثال:

In [2]:

```
my_list = ['A string',23,100.232,'o']
```

مانند رشته‌ها، تابع len() تعداد عناصر موجود در دنباله لیست را به شما می‌گوید.

In [3]:

```
len(my_list)
```

Out[3]:

```
4
```

ایندکس گذاری و برش

عمل ایندکس‌گذاری و برش مشابه با رشته‌ها عمل می‌کند. بیایید یک لیست جدید بسازیم تا به یادآوری نحوه کار این عملیات بپردازیم.

==========

In [1]:

```
my_list = ['one','two','three',4,5]
```

In [5]:

```
# Grab element at index 0
my_list[0]
```

Out[5]:

```
'one'
```

In [6]:

```
# Grab index 1 and everything past it
my_list[1:]
```

Out[6]:

```
['two', 'three', 4, 5]
```

In [7]:

```
# Grab everything UP TO index 3
my_list[:3]
```

Out[7]:

```
['one', 'two', 'three']
```

In [ ]:

```
my_list[::-1]
```

بررسی عضویت یا عدم عضویت در لیست با عملگر in

In [2]:

```
'one' in my_list
```

Out[2]:

```
True
```

In [3]:

```
'one' not in my_list
```

Out[3]:

```
False
```

همچنین می‌توانیم از + برای اتصال لیست‌ها استفاده کنیم، درست مانند کاری که برای رشته‌ها انجام دادیم.

In [8]:

```
my_list + ['new item']
```

Out[8]:

```
['one', 'two', 'three', 4, 5, 'new item']
```

توجه: این در واقع لیست اصلی را تغییر نمی دهد!

In [9]:

```
my_list
```

Out[9]:

```
['one', 'two', 'three', 4, 5]
```

برای دائمی شدن این تغییر، باید لیست جدید را مجدد در متغیر ذخیره نمایید!

In [10]:

```
# Reassign
my_list = my_list + ['add new item permanently']
```

In [11]:

```
my_list
```

Out[11]:

```
['one', 'two', 'three', 4, 5, 'add new item permanently']
```

همچنین می‌توانیم از * برای تکرار یک لیست، مشابه رشته‌ها استفاده کنیم:

In [12]:

```
# Make the list double
my_list * 2
```

Out[12]:

```
['one',
 'two',
 'three',
 4,
 5,
 'add new item permanently',
 'one',
 'two',
 'three',
 4,
 5,
 'add new item permanently']
```

In [13]:

```
# Again doubling not permanent
my_list
```

Out[13]:

```
['one', 'two', 'three', 4, 5, 'add new item permanently']
```

متدهای پایه لیست ها

اگر با یک زبان برنامه‌نویسی دیگر آشنایی دارید، ممکن است شروع کنید به تشبیه بین آرایه‌ها در آن زبان و لیست‌ها در پایتون. اما لیست‌ها در پایتون به دو دلیل از آرایه‌ها در زبان‌های دیگر انعطاف‌پذیرتر هستند: آنها اندازه ثابتی ندارند (بدین معنی که نیازی نداریم اندازه یک لیست را مشخص کنیم) و محدودیت نوع ثابتی ندارند (همانطور که در بالا دیدیم).

بیایید به بررسی تعدادی از متدهای ویژه برای لیست‌ها بپردازیم:

In [4]:

```
# Create a new list
list1 = [1,2,3]
```

متد append یک و تنها یک عضو به انتهای لیست اضافه می کند:

In [5]:

```
# Append
list1.append('append me!')
```

In [6]:

```
# Show
list1
```

Out[6]:

```
[1, 2, 3, 'append me!']
```

اگر بخواهیم چند عضو را بصورت همزمان به لیست اضافه کنیم دیگر متد append پاسخگوی نیاز ما نمی باشد

In [7]:

```
list1.append([333,444])
```

In [8]:

```
list1
```

Out[8]:

```
[1, 2, 3, 'append me!', [333, 444]]
```

در این حالت باید از متد extend استفاده نماییم

In [9]:

```
# add multiple items to end of list
list1.extend([999, 1111])
```

In [11]:

```
list1
```

Out[11]:

```
[1, 2, 3, 'append me!', [333, 444], 999, 1111]
```

درج عضو جدید در ایندکس مورد نظر با استفاده از متد insert

In [17]:

```
# insert an item to specific index

list1.insert(2, 'Inserted item')
```

In [18]:

```
list1
```

Out[18]:

```
[1, 2, 'Inserted item', 3, 'append me!', [333, 444], 999, 1111]
```

افزودن چندین عضو به ابتدا یا انتها با استفاده از عملگر +=

In [19]:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

a += ['grault', 'garply']
```

In [20]:

```
a
```

Out[20]:

```
['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'grault', 'garply']
```

In [21]:

```
a = [10, 20] + a
```

In [22]:

```
a
```

Out[22]:

```
[10, 20, 'foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'grault', 'garply']
```

یک نکته جالب

In [34]:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux']
a += 'corge'
a
```

Out[34]:

```
['foo', 'bar', 'baz', 'qux', 'quux', 'c', 'o', 'r', 'g', 'e']
```

متد pop برای حذف و بازگرداند یک عضو از لیست مورد استفاده قرار می گیرد. بصورت پیش فرض این متد آخرین ایندکس یا -1 را حذف کرده و مقدار آن را برای استفاده در سایر محاسبات برمیگرداند

In [23]:

```
list1 = [1,2,3]
```

In [17]:

```
# Pop off the 0 indexed item
list1.pop(0)
```

Out[17]:

```
1
```

In [18]:

```
# Show
list1
```

Out[18]:

```
[2, 3, 'append me!']
```

In [19]:

```
# Assign the popped element, remember default popped index is -1
popped_item = list1.pop()
```

In [20]:

```
1 popped_item
```

Out[20]:

```
'append me!'
```

In [21]:

```
1 # Show remaining list
2 list1
```

Out[21]:

```
[2, 3]
```

حذف اعضا از لیست به چندین روش مقدور می باشد.

1. استفاده از متد `remove`
2. استفاده از تابع `del`
3. استفاده از روش انتساب چند ایندکسی

In [28]:

```
1 # remove a first occurance of value
2 list1 = [1,2,3,4,5,6,1,1]
3
4 list1.remove(1)
```

In [29]:

```
1 list1
```

Out[29]:

```
[2, 3, 4, 5, 6, 1, 1]
```

In [30]:

```
1 del list1[2]
```

In [31]:

```
1 list1
```

Out[31]:

```
[2, 3, 5, 6, 1, 1]
```

In [32]:

```
1 del list1[0:2]
```

In [33]:

```
1 list1
```

Out[33]:

```
[5, 6, 1, 1]
```

همچنین باید توجه داشت که اگر در ایندکس مورد نظر عنصری وجود نداشته باشد، عمل ایندکس‌گذاری در لیست خطا خواهد داد. به عنوان مثال:

In [22]:

```
1 list1[100]
```

```
---------------------------------------------------------------------------
-
IndexError                                Traceback (most recent call las
t)
<ipython-input-22-af6d2015fa1f> in <module>()
----> 1 list1[100]

IndexError: list index out of range
```

می‌توانیم از متد sort و متد reverse نیز برای تغییر لیست‌های شما استفاده کنیم:

In [23]:

```
1 new_list = ['a','e','x','b','c']
```

In [24]:

```
1 #Show
2 new_list
```

Out[24]:

```
['a', 'e', 'x', 'b', 'c']
```

In [25]:

```
1 # Use reverse to reverse order (this is permanent!)
2 new_list.reverse()
```

In [26]:

```
1 new_list
```

Out[26]:

```
['c', 'b', 'x', 'e', 'a']
```

In [27]:

```
# Use sort to sort the list (in this case alphabetical order, but for numbers it wil
new_list.sort()
```

In [28]:

```
new_list
```

Out[28]:

```
['a', 'b', 'c', 'e', 'x']
```

تغییر چندین مقدار در لیست با عملگر انتساب

In [35]:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

In [36]:

```
a[1:4]
```

Out[36]:

```
['bar', 'baz', 'qux']
```

1. تعداد موارد انتخاب شده با موارد جایگزین برابر باشند

In [37]:

```
a[1:4] = [100, 200, 300]
```

In [38]:

```
a
```

Out[38]:

```
['foo', 100, 200, 300, 'quux', 'corge']
```

2. تعداد موارد انتخاب شده کمتر از موارد جایگزین باشد

In [39]:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

a[1:4] = [111, 222, 333, 444, 555, 666]

a
```

Out[39]:

```
['foo', 111, 222, 333, 444, 555, 666, 'quux', 'corge']
```

3. تعداد موارد انتخاب شده بیشتر از موارد جایگزین باشد

In [40]:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

a[1:5] = [-111, -222]

a
```

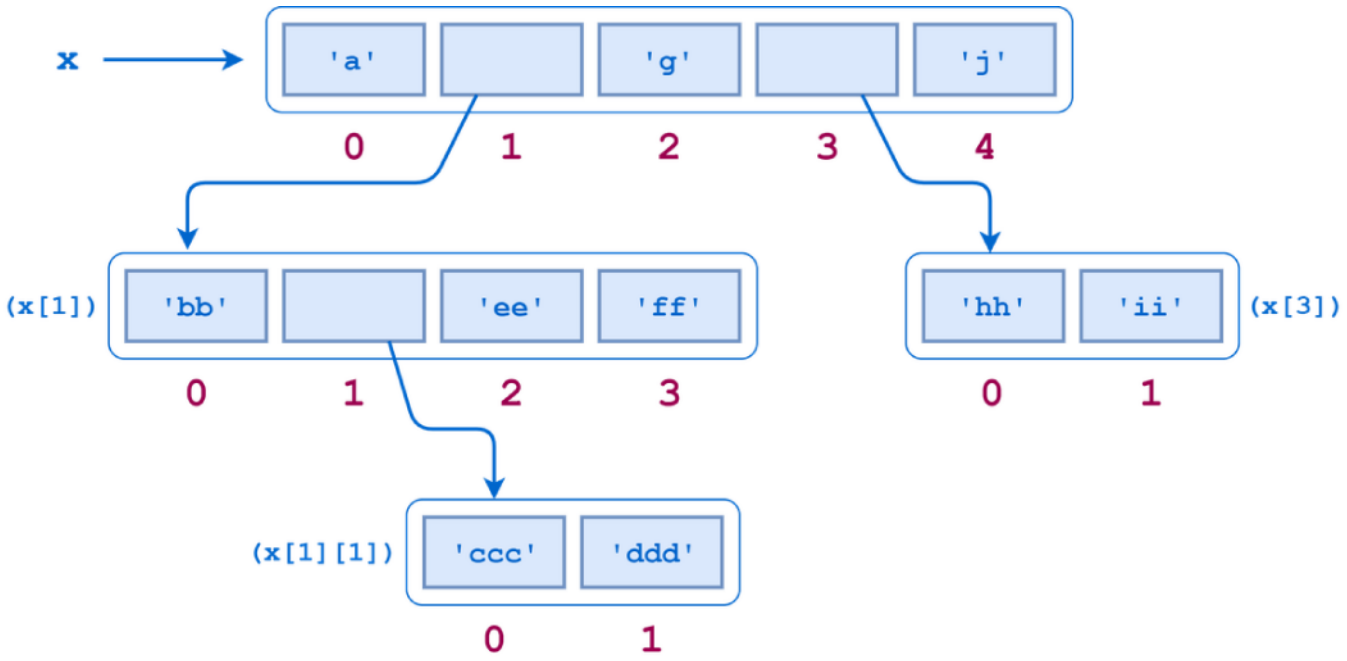
Out[40]:

```
['foo', -111, -222, 'corge']
```

لیست های تودرتو

یک ویژگی بزرگ از ساختارهای داده در پایتون این است که آنها از تو در تو کردن پشتیبانی می‌کنند. این بدان معنی است که می‌توانیم ساختارهای داده را درون ساختارهای داده دیگر داشته باشیم. به عنوان مثال: یک لیست درون یک لیست.

بیایید ببینیم چگونه کار می‌کند!



In [29]:

```
# Let's make three lists
lst_1=[1,2,3]
lst_2=[4,5,6]
lst_3=[7,8,9]

# Make a list of lists to form a matrix
matrix = [lst_1,lst_2,lst_3]
```

In [30]:

```
# Show
matrix
```

Out[30]:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

ما می‌توانیم باز هم از عمل ایندکس‌گذاری استفاده کنیم تا عناصر را بگیریم، اما اکنون دو سطح برای ایندکس وجود دارد.
عناصر درون شیء ماتریس و سپس عناصر درون آن لیست!

In [31]:

```
# Grab first item in matrix object
matrix[0]
```

Out[31]:

```
[1, 2, 3]
```

In [32]:

```
# Grab first item of the first item in the matrix object
matrix[0][0]
```

Out[32]:

```
1
```

In [34]:

```
first_col
```

Out[34]:

```
[1, 4, 7]
```

In [12]:

```
x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
```

In [13]:

```
1 x[1]
```

Out[13]:

```
['bb', ['ccc', 'ddd'], 'ee', 'ff']
```

In [14]:

```
1 x[1][1]
```

Out[14]:

```
['ccc', 'ddd']
```

In [15]:

```
1 x[1][1][0]
```

Out[15]:

```
'ccc'
```

In [16]:

```
1 x[1][1][0].upper()
```

Out[16]:

```
'CCC'
```

بریم سراغ توپل ها!