International Scientific League of PAYA 2024

دفترچه پیش آزمون مرحله مقدماتی

لیگ علوم کامپیوتر و برنامه نویسی

مقطع متوسطه دوم (پایه‌های دهم و یازدهم)

اسفند ۱۴۰۲

خیابان فاطمی غربی، بین سیندخت و جمالزاده شمالی، پلاک ۲۵۰، خانه ریاضی تهران
شماره تماس: ۶۷۴۱۹۰۰۰
کانال تلگرام خانه ریاضی تهران: @Mathhome
کانال ایتا و تلگرام پایا: @payaleague

لیست

لیست در پایتون

لیست یک ساختمان داده مهم در پایتون است. به کمک لیست‌ها می‌توانیم دنباله‌ای از داده‌ها را در یک متغیر ذخیره کرده و روی آن‌ها عملیات‌های مختلفی را اجرا کنیم.

نوع داده‌ای لیست یکی از انواع داده مجموعه‌ای یا Collections در این زبان محبوب است. لیست شامل مجموعه‌ای از عناصر به صورت ترتیبی و قابل تغییر است. لیست در پایتون با کروشه یا (براکت bracket) به شکل []مشخص می‌شود؛ به طوری که با علامت کروشه باز [لیست آغاز شده و تا علامت کروشه بسته] ادامه می‌یابد.

هر عنصر درون لیست به وسیله کاما (ویرگول انگلیسی یا comma) به شکل , از یکدیگر جدا شده و می‌تواند از هر نوع داده دلخواهی باشد.

```
lst = ["Omid", 217, 22.5, "mathhome"]
```

در یک خط کد بالا، یک لیست با نام lst ایجاد کرده‌ایم. سپس درون آن چهار مقدار با نوع داده‌ای مختلف قرار داده‌ایم. به همین راحتی!

کاربرد لیست

از لیست در هر جایی که بخواهید می‌توانید استفاده کنید! هر کجا که دنباله‌ای معنی‌دار یا بدون معنی از داده‌ها دارید، لیست‌ها سریع‌ترین و شاید بهترین انتخاب شما باشند.

مزیت لیست‌ها در پایتون این است که به سادگی تعریف شده، اعضای آن اضافه یا کم می‌شوند. همچنین اعمال بین لیست‌ها بسیار ساده و سریع است. در ادامه توابع و تکنیک‌هایی به شما معرفی می‌کنیم که به راحتی هر بلایی که خواستید سر لیست‌ها در بیاورید!

دسترسی به اعضای لیست در پایتون

همانطور که پیش‌تر نیز گفته شد، ساده‌ترین نوع تعریف یک لیست در پایتون، استفاده از دو علامت کروشه در کنار هم است.

با این تعریف، متوجه خواهید شد که امکان ایجاد یک لیست خالی نیز وجود دارد. در نتیجه حتماً نیازی نیست که در ابتدای کار تمام مقادیر درون لیست را داشته باشیم؛ بلکه در حین برنامه خواهیم توانست مقادیر دلخواه خود را به آن اضافه یا از آن کم کنیم.

دسترسی به اعضای لیست بر اساس اندیس‌های آن صورت می‌گیرد. اندیس‌ها اعدادی هستند که موقعیت هر عنصر در لیست را مشخص می‌کنند.

اولین عنصر یک لیست دارای اندیس صفر (۰) و به طور کلی n اُمین عنصر دارای اندیس n-1 خواهد بود.

اندیس لیست (ایندکس یا نمایه)

فرض کنید یک لیست از اسامی افراد مختلف را درون لیست خود داریم.

```
lst = ["sara", "omid", "amir", "roya" ,"ehsan"]
```

برای دسترسی به هر عنصر از لیست، با داشتن موقعیت یا اندیس آن عنصر و آوردن اندیس در جلوی نام متغیر لیست، به عنصر مورد نظرمان دسترسی خواهیم داشت.

یعنی قطعه کد زیر به ما عنصری که در اندیس شماره ۳ قرار دارد با خروجی می‌دهد؛ یعنی چهارمین عنصر در لیست که معادل roya است.

```
lst[3]
```

به همین سادگی خواهیم توانست به تمام عناصر درون یک لیست دسترسی پیدا کنیم.

توجه کنید که اندیس یک مقدار از نوع عددی صحیح (integer) است. در نتیجه فراخوانی اندیس اعشاری یا رشته‌ای ما را با خطا مواجه خواهد کرد.

در لیست بالا ما پنج عنصر داریم. بازه اندیس‌های ما از 0 تا 4 خواهد بود. وارد کردن عددی بزرگ‌تر از ۴ ما را با خطای اندیس (IndexError) مواجه خواهد کرد.

اندیس منفی لیست در پایتون

زبان برنامه نویسی پایتون تعریف اندیس برای اعضای یک لیست را امکان‌پذیر کرده است. به این‌صورت که آخرین عنصر هر لیست دارای اندیس 1-، عنصر یکی مانده به آخر 2-و همین طور تا عنصر اول...

به کمک اندیس منفی در پایتون می‌توانیم به عناصر یک لیست از انتهای آن دسترسی داشته باشیم.

بنابراین برای چاپ اسم ehsan در لیست مفروض خود، کدی شبیه به کد زیر خواهیم داشت.

```
print( lst[-1] )
# Output: ehsan
```

سوال: با توجه به این توضیحات، به نظرتان بازه اندیس منفی مجاز در پایتون در یک لیست با n عضو چیست؟

دسترسی به بخشی از لیست (List) برش زدن

یکی از ویژگی‌ها و قابلیت‌های جذاب لیست‌ها در پایتون، توانایی **برش زدن لیست** یا Slicing است.

بگذارید این موضوع را همراه با یک مثال برایتان توضیح دهم. فرض کنید لیست زیر را در برنامه خود ایجاد کرده‌ایم.

```
lst = ['m', 'a', 't', 'h', 'h', 'o', 'm', 'e']
```

یک لیست با اندازه ۱۰. اگر بخواهیم فقط به بخشی از این لیست دسترسی داشته باشیم باید چگونه عمل کنیم؟

ویژگی slicing یا برش زدن list این امکان را به ما می‌دهد که فقط به بخشی از یک لیست دسترسی داشته باشیم.

دسترسی به لیست با تعریف بازه اندیس

برای تعریف یک بازه اندیس در هنگام فراخوانی عناصر لیست، کافی است به جای وارد کردن یک شماره اندیس، بازه آن را مشخص کنیم. مشخص کردن بازه با علامت دو نقطه (:) انجام می‌شود.

```
lst[3:7]
# Output: ['h', h 'o', 'm']
```

در قطعه کد بالا، عناصری که اندیس آن‌ها در بازه 3 تا 7 بود به ما برگردانده شد. توجه کنید که بازه اندیس‌ها چگونه تعریف شده است:

اگر بازه به صورت i:jتعریف شود، این تعریف شامل عنصر iتا عنصر j-1 خواهد شد و عنصر j در فراخوانی آورده نمی‌شود.

اگر هر یک از کران بازه را مشخص نکنیم، به طور پیش‌فرض تا انتهای آن را در نظر خواهد گرفت. یعنی برای فراخوانی اعضای لیست از عنصر دوم به بعد به صورت زیر عمل می‌کنیم.

```
lst[2:]
# Output:
# ['t', 'h', 'h', 'o, 'm', 'e']
```

در قطعه کد بالا، عناصر از اندیس دوم تا انتهای لیست به ما نمایش داده می‌شود.

تکنیک: دسترسی به میان لیست در پایتون

همانطور که احتمالاً تا به اینجای کار برداشته کرده‌اید، می‌توانیم از اندیس‌های مثبت و منفی در هنگام برش دادن لیست استفاده کنیم.

اگر این دو نوع اندیس را به طور همزمان استفاده کنیم، خواهیم توانست خروجی‌های جالب و شگفت‌انگیزی از لیست خود بگیریم!

برای مثال، فرض کنید در لیست دلخواه خود، می‌خواهیم دو عنصر اول و دو عنصر آخر لیست را حذف کرده و بقیه را نشان دهیم. برای این کار به راحتی کد زیر عمل خواهیم کرد. به این روش می‌توانید لیست را به هر صورتی که مایلید برش دهید!

```
lst[2:-2]
# Output:
# ['h', 'h', 'o', 'm']
```

تکنیک: دسترسی دنباله‌ای به اعضای لیست (همراه با گام)

فرض کنید یک لیست از اعداد ۱ تا ۲۰ در اختیار دارید و می‌خواهید اعداد را دو در میان از آن خارج کنید. اولین راهی که احتمالاً به ذهنمان می‌رسد استفاده از **حلقه‌های پایتونی** نظیر (for) برای پیمایش لیست است.

این موضوع را جلوتر بررسی خواهیم کرد. اما این جا یک ویژگی جذاب در هنگام تعریف بازه لیست در پایتون را با هم یاد خواهیم گرفت.

همانطور که آموختید، با گذاشتن یک علامت دو نقطه (:) می‌توان بازه‌ای برای اندیس‌های لیست تعریف کرد. اگر پس از تعریف بازه اندیس، یک بار دیگر دو نقطه بگذاریم، خواهیم توانست گام حرکت خود را نیز مشخص کنیم.

یک لیست به نام numbers ایجاد کرده و اعداد ۱ تا ۲۰ را در آن قرار می‌دهیم. سپس به صورت یک در میان، از ابتدا تا انتها آن حرکت می‌کنیم.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

numbers[::2]

# Output:
# [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

به این‌صورت تمام اعداد فرد موجود در لیست را فراخوانی کرده‌ایم. به همین سادگی!

حال اگر بخواهیم از عنصر پنجم شروع کرده و سه تا سه تا حرکت کنیم چه اتفاقی می‌افتد؟ کد آن، چیزی مشابه به کد زیر خواهد شد:

```
numbers[4::3]
# Output:
# [5, 8, 11, 14, 17, 20]
```

امیدوارم به خوبی با نحوه برش لیست و تکنیک‌های آن آشنا شده باشید و از آن در زمان لازم استفاده کنید.

لیست تو در تو یا لیست چند بعدی در پایتون

عناصر لیست‌ها هر چیزی می‌توانند باشند؛ از عدد و رشته گرفته تا یک لیست دیگر! بنابراین اگر عناصر یک لیست، یک لیست دیگر باشد، **لیست‌های تو در تو** (Nested List) را خواهیم داشت.

فرض کنید یک لیست داریم که اطلاعات یک دانشجو را نگهداری می‌کند. می‌خواهیم شناسه(id) ، اسم و کلیه نمرات او را داشته باشیم.

```
student = [6214, "Mitra", [18, 17.2, 19.5, 16, 20, 19.9, 20]]
```

عنصر سوم این لیست، خودش یک لیست است. به همین صورت می‌توان لیست‌های چند سطحی را نیز ایجاد کرد.

تعریف ماتریس در پایتون

همانطور که می‌دانید ماتریس یک لیست تو در تو است که اعضای آن اندازه یکسان و مشخصی دارند.

به طور پیش‌فرض در پایتون ساختمان داده ماتریس نداریم. البته به کمک برخی کتابخانه‌ها و ابزارها می‌توان چنین ساختمان داده‌ای را ایجاد کرده و از آن استفاده کرد.

ما در این جا ساده‌ترین حالت ایجاد یک ماتریس در پایتون را با هم بررسی می‌کنیم.

برای ایجاد ماتریس، کافی است یک لیست ایجاد کنیم که هر عنصر آن یک لیست دیگر باشد.

```
mtrx = [[], [], []]
```

بهتر است در طول برنامه خود همواره بررسی کنیم که اندازه لیست‌های داخلی از اندازه مشخص شده بیشتر نشود. راه‌حل جایگزین بررسی اندازه لیست‌ها، استفاده از تابع اندازه لیست (یا تابع len()) است که در قسمت توابع کاربردی معرفی می‌شود.

مثلاً اطلاعات مجموع ساعت کاری کارمندان در سه روز از هفته را در ماتریس زیر نگهداری کرده‌ایم.

```
logs = [['Mon', '06/22', 'sina', 12],
        ['Tue', '06/23', 'ehsan', 7],
        ['Wed', '06/24', 'roya', 10],
        ['Wed', '06/24', 'sina', 5 ]]
```

کار با عناصر لیست

علاوه بر دسترسی به عناصر یک لیست، نیاز به وارد کردن، خارج کردن یا **تغییر داده‌ها در لیست** خواهیم داشت.

برای هر کدام از این کارها روش‌ها یا توابع مختلفی در پایتون وجود دارد. در ادامه به مهم‌ترین و کاربردی‌ترینِ آن‌ها را بررسی می‌کنیم.

افزودن عنصر جدید به لیست

برای افزودن یک عنصر جدید به لیست فعلی، سه روش وجود دارد. در دو روش اول، عنصر به انتهای لیست اضافه می‌شود؛ اما به کمک روش سوم می‌توانیم عنصر را در اندیس دلخواه خود قرار دهیم.

لیست nums را با سه مقدار اولیه ایجاد کرده و سپس به آن عناصری را اضافه می‌کنیم.

```
nums = [32, 15, 20]
```

افزودن عنصر جدید به لیست با تابع append

با صدا زدن تابع ()append روی لیست، مقدار ورودی خود را به عنوان عنصر جدید در انتهای لیست درج می‌کند. این متد صرفاً یک ورودی گرفته و نوع داده‌ای آن اهمیتی ندارد.

```
nums.append(11)

print(nums)
# Output:
# [32, 15, 20, 11]

nums.append(73)

print(nums)
# Output:
# [32, 15, 20, 11, 73]
```

حذف عنصر از لیست در پایتون

وقتی می‌خواهیم مقداری را از لیست حذف کنیم، دو حالت وجود دارد.

- ایندکس عنصر مورد نظر را می‌دانیم.
- مقدار عنصر مورد نظر را می‌دانیم.

برای هر یک از این حالات دو دستور متفاوت را معرفی می‌کنم. از هر کدام می‌توانید در جای مناسب خود استفاده کرده و عنصر مورد نظرتان را از لیست حذف کنید.

حذف عنصر با ایندکس به کمک دستور del

دستور del در پایتون برای حذف متغیرها استفاده می‌شود. به این‌صورت که هر گاه این کلمه کلیدی را قبل از نام متغیر بیاوریم، آن متغیر به طور کامل از درون برنامه ما حذف خواهد شد.

به کمک این دستور، می‌توان کل لیست یا یک عنصر از آن را حذف کرد.

اگر نام لیست یا لیست همراه به ایندکس را در جلوی کلمه کلیدی del قرار دهیم، لیست یا عنصر مورد نظر به طور کامل حذف خواهد شد.

```
lst = ['m', 'a', 't', 'h', 'h', 'o, 'm', 'e']
del lst[3]
print(lst)
# ['m', 'a', 't', 'h', 'o', 'm', 'e']

del lst
print(lst)

# Traceback (most recent call last):
```

```
#   File "<pyshell#79>", line 1, in <module>
#     print(lst)
# NameError: name 'lst' is not defined
```

در خط دوم قطعه کد بالا، اقدام به حذف عنصر موجود در ایندکس 3 لیست (معادل با عنصر h) کردیم. همانطور که در نتیجه مشاهده می‌کنید، این عنصر از درون لیست حذف شده است.

در خط ششم، کل متغیر لیست را حذف کرده‌ایم. پس از اقدام برای چاپ لیست، با خطای عدم **تعریف متغیر در پایتون** رو به رو می‌شویم.

استفاده از متد remove برای حذف عنصر از لیست

این متد بر روی لیست صدا زده می‌شود و یک ورودی می‌گیرد. آنچه که به عنوان ورودی گرفته را در لیست جستجو کرده و اولین موردی که مطابقت داشت را حذف می‌کنید.

در لیست زیر، ما دو بار حرف a را داریم. با صدا زدن متد remove('a') اولین موردی که در لیست پیدا می‌شود (یعنی ایندکس ۱) حذف خواهد شد.

```
lst = ['m', 'a', 't', 'h', 'h', 'o', 'm', 'e']
lst.remove('m')
print(lst)
# ['a', 't', 'h', 'h', 'o', 'm', 'e']
```

تغییر عنصر در لیست

برای تغییر مقدار موجود در ایندکس مورد نظر، می‌توان پس از فراخوانی آن خانه از لیست، اقدام به تغییر آن کنیم. این کار با علامت انتصاب یا = انجام می‌شود.

```
lst = ["sara", "omid", "amir", "roya" ,"ehsan"]
lst[2] = "hamed"
print(lst)
# ['sara', 'omid', 'hamed', 'roya', 'ehsan']
```

اگر عنصر لیست ما یک مقدار عددی باشد و بخواهیم آن را کاهش یا افزایش بدهیم، به صورت مشابه انجام خواهد شد.

```
nums = [15, 24, 60]
nums[1] += 2

print(nums)
# [15, 26, 60]
```

جستجو در لیست

در هنگام کار با لیست‌ها در پایتون، بارها پیش می‌آید که بخواهیم عنصری را در لیست جستجو کنیم. معمولاً این نوع جستجو برای یک عبارت شرطی استفاده می‌شود.

بررسی وجود داشتن عنصر در لیست با کلمه in

یکی از کلمات کلیدی که برای جستجو یک عنصر در لیست پایتون استفاده می‌شود، عبارت in است.
همانطور که از معنی این کلمه کلیدی مشخص است، بررسی می‌کند که آیا یک مقدار خاص درون لیست مورد نظر ما وجود دارد یا خیر؟ و نتیجه را به صورت True و False به ما می‌دهد.
روش استفاده از آن بسیار ساده و راحت است. کافی است آنچه را می‌خواهید به زبان پایتون بگویید!
فرض کنید لیست عددی زیر را در اختیار داریم:

```
nums = [15, 24, 60, 55, 74, 33]
```

می‌خواهیم بررسی کنیم آیا اعداد 24 و 77 درون این لیست وجود دارند یا خیر؟
در صورتی که هر کدام از آن اعداد وجود داشت یک پیغام متناسب با آن در خروجی چاپ می‌کنیم.

```
if 24 in nums:
    print("24 is in python list!")

if 77 in nums:
    print("77 is in python list!")

# output:
# 24 is in python list!
```

به همین سادگی!
اگر یک لیست با مقادیر رشته‌ای داشتیم؛ باز هم نحوه کار به همین شکل می‌شد.

```
colros = ["red", "purple", "green", "blue", "yellow", "orange"]

if "red" in colors and "blue" in colors:
    print("A mix of these colors can be Violet!")

# output:
# A mix of these colors can be Violet!
```

جستجوی ایندکس عنصر در لیست

گاهی لازم است شماره اندیس عنصر موجود در لیست را به دست بیاوریم. یا همزمان با جستجوی یک مقدار، در صورت وجود داشتن آن، اندیس آن را در اختیار داشته باشیم.
برای انجام چنین کاری می‌توان از متد ()index در لیست پایتون استفاده کرد.
این متد یک مقدار ورودی گرفته و آن را درون لیست ما جستجو می‌کند.

- اگر مقدار درون لیست پیدا شود، شماره اندیس خانه مورد نظر را به عنوان خروجی به ما می‌دهد.
- اگر مقدار مورد نظر دورن لیست وجود نداشته باشد، با خطایی از نوع ValueError مواجه خواهیم شد.

با **مدیریت خطای زمان اجرا در پایتون**، خواهیم توانست به راحتی، علاوه بر جستجو، ایندکس خانه مورد جستجو را نیز پیدا کنیم.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]

try:
    target = names.index("omid")
except ValueError:
    print("name not found!")

# result:
# target = 1
```

توابع مهم کار با لیست پایتون

در هنگام کار با لیست‌ها در پایتون با توابع بسیار زیادی رو به رو هستید! توابعی که کارهای بسیاری برایتان انجام می‌دهند.
در این بخش از مقاله به معرفی چند مورد از مهم‌ترین و پرکاربردترین توابع لیست پایتون می‌پردازیم.

اندازه لیست با تابع len

یکی از لازم‌ترین ویژگی‌های یک لیست، اندازه آن است. از جمله پرکاربردترین توابع هم، همین تابع len() است.
این تابع، لیست را به عنوان ورودی از ما گرفته و اندازه آن را به ما می‌دهد. منظور از اندازه، تعداد عناصر موجود در لیست است.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
print(len(names))

# output:
# 5
```

خارج کردن آخرین عنصر از لیست با متدpop

با کمک تابع pop() که روی لیست صدا زده می‌شود، می‌توانیم آخرین عنصر (عنصر موجود در خانه 1-) را از آن خارج کنیم.
همزمان با خارج کردن این عنصر، تابع pop() آن را برای ما باز می‌گرداند. در نتیجه خواهیم توانست آخرین عنصر را قبل از دور ریختن، درون متغیری ریخته یا با آن کار دیگری انجام دهیم.
در قطعه کد زیر روند کار با این متد را می‌بینید. ابتدا آخرین عضو موجود در لیست names را بیرون انداخته و چاپ می‌کنیم. سپس لیست را چاپ کرده تا نتیجه عملیات را مشاهده کنیم.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
print( names.pop() )
print( names )

# output:
# ehsan
# ['sara', 'omid', 'amir', 'roya']
```

برعکس کردن لیست با متد reverse (وارونه کردن)

گاهی نیاز داریم یک لیست را سر و ته کنیم! یعنی عضو اول تبدیل به آخرین عضو و آخرین عضو تبدیل به اولین عضو شود! شاید اولین راه‌کار ایجاد یک لیست جدید و افزودن عناصر از انتهای لیست قبلی به لیست جدید باشد.

به کمک صدا زدن متد ()reverse روی **لیست پایتون**، می‌توان به راحتی لیست را وارونه کرد.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
names.reverse()
print( names )

# output:
# ['ehsan', 'roya', 'amir', 'omid', 'sara']
```

تابع ()reverse لیست را برعکس می‌کند و به عبارتی لیست اصلی از بین می‌رود. اگر فقط می‌خواهید لیست را وارونه در اختیار داشته باشید یا لیست فعلی را به صورت برعکس درون متغیر دیگری ذخیره کنید، از روش زیر استفاده کنید.

معکوس کردن لیست به صورت حرفه‌ای با گام منفی

حال که بحث از وارونه کردن یا معکوس کردن لیست در پایتون شد؛ اجازه دهید یک روش بسیار جالب و به کمک معلومات قبلی خود به شما بگویم.

در قسمت دسترسی به عناصر لیست، روشی برای حرکت با یک گام مشخص در لیست را آموختیم. اگر گام حرکت در لیست را برابر 1-قرار دهیم، حرکت ما وارونه خواهد شد!

در این صورت فقط به لیست برعکس شده دسترسی خواهیم داشت اما آن را برعکس نخواهیم کرد.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
print( names[::-1] )
print( names )

# output:
# ['ehsan', 'roya', 'amir', 'omid', 'sara']
# ['sara', 'omid', 'amir', 'roya', 'ehsan']
```

همانطور که می‌بینید به کمک این تکنیک، لیست برعکس را در خط دوم چاپ کردیم؛ اما لیست اصلی تغییری نکرده است.

حذف تمام عناصر لیست با clear

اگر از اعضای لیست خود خسته شده‌اید و می‌خواهید تمام اعضای آن را از صحنه برنامه‌تان حذف کنید، متد ()clear برای شماست !

با صدا زدن متد ()clear روی یک لیست، تمام اعضای آن حذف شده و در نهایت یک لیست خالی خواهید داشت.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
names.clear()
print( names )

# output:
# []
```

می‌توانید تمام توابع و متدهای لیست‌ها در پایتون را در این لینک مشاهده کنید.

پیمایش List در پایتون

پیمایش لیست در پایتون یکی از مهم‌ترین مباحث در بررسی لیست‌هاست. دو راه برای پیمایش عناصر موجود در لیست داریم.

در ادامه هر دو مورد را با مثال برایتان شرح می‌دهیم.

پیمایش در لیست با حلقه for و ایندکس‌ها

در این روش برای دسترسی به عناصر لیست، از ایندکس هر خانه استفاده می‌کنیم.

شیوه کار به این صورت است که متغیر مورد نظر را در بازه مجاز تغییر داده و در هر بار، به عنصر خانه مورد نظر دسترسی می‌یابیم.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
for i in range(0, 5):
    print( names[i] )
```

در مثال فوق، متغیر `i` را به نوبت معادل 0 تا 4 قرار دادیم و در هر دور از حلقه، مقدار موجود در خانه‌های 0 تا 4 لیست را چاپ کردیم.

اگر اندازه لیست ثابت نباشد، از تابع len() برای مشخص کردن اندازه آن استفاده خواهیم کرد. قطعه کد زیر، ساختار بهتری برای این نوع پیمایش در لیست‌ها را نشان می‌دهد.

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
for i in range( len(names) ):
    print( names[i] )
```

حرکت در لیست با for in

اگر به اندیس هر خانه احتیاجی ندارید و فقط می‌خواهید به مقادیر موجود در لیست دسترسی داشته باشید، راه ساده‌تری نیز وجود دارد.

به کمک عبارت کلیدی for in خواهیم توانست درون یک لیست پیمایش کنیم.

برای درک بهتر، به مثال زیر توجه کنید:

```
names = ["sara", "omid", "amir", "roya" ,"ehsan"]
for name in names:
    print("Hello " + name)
```

در حلقه موجود در این قطعه کد، در هر دور اجرای حلقه، مقدار موجود در تک تک خانه‌های لیست درون متغیر name ریخته می‌شود.

از این متغیر در طول حلقه می‌توانیم استفاده کنیم؛ آن را چاپ کرده، مقایسه کرده و یا عملیات خاصی روی آن انجام دهیم.

تابع

تعریف تابع در پایتون

توابع در پایتون (Python Functions) دارای ساختار مشخصی هستند. برای این که هر قطعه کد را شناسایی کنیم، باید برای آن‌ها یک اسم در نظر بگیریم. پس تمام توابع دارای یک اسم یکتا هستند.

برای تعریف تابع در پایتون از کلمه کلیدی def قبل از اسم آن استفاده می‌شود.

فرض کنید می‌خواهیم یک تابع با نام ()myprint بنویسیم. این تابع یک رشته متنی را در خروجی پایتون چاپ می‌کند. برای چاپ از ()print استفاده می‌کنیم.

در مقابل نام تابع علامت دو نقطه قرار داده و بدنه تابع را با یک فرورفتگی می‌نویسیم:

```
def myprint():
    print("Some Text from mathhome.com")
```

تبریک! شما اولین تابع خود را با زبان پایتون نوشتید! حالا بیایید و یک تابع حرفه‌ای ایجاد کنیم...

برای فراخوانی کردن تابع (یا همان اجرا کردن تابع) فقط کافی است در هر کجای برنامه که خواستیم، نام تابع را صدا زده و اگر لازم است ورودی‌های آن را تعریف کنیم.

در خط دوم و دقیقاً پس از تعریف تابع، با قرار دادن یک رشته‌ی متنی کامنت شده با '''، می‌توانیم توضیحاتی درباره تابع بنویسیم. این توضیحات دلخواه بوده و برای توسعه‌های بعدی یا هنگام کار با IDEهای برنامه‌نویسی کاربرد خواهد داشت.

در قطعه کد زیر، من یک توضیح ساده برای تابع خود نوشته‌ام.

```
def myprint():
    ''' This Simple Function Will Print a Static Text '''
    print("Some Text from Mathhome.com")
```

ورودی تابع

تابع ()myprint که در اینجا تعریف کردیم، همیشه یک متن ثابت را چاپ می‌کند. اگر بخواهیم رشته‌ی مورد نظر را در زمان فراخوانی به تابع بدهیم، باید برای تابع خود ورودی تعریف کنیم.

تعریف ورودی برای تابع پایتون بسیار ساده است. درون پرانتز جلوی نام تابع، نام ورودی را مشخص می‌کنیم.

اگر می‌خواستید چند ورودی برای تابع تعریف کنید، می‌توان آن‌ها را با علامت , از یکدیگر جدا کرد؛ مشابه تابع زیر:

```
def myFunction( arg1, arg2, arg3 ):
    # Some Codes Here
```

وقتی یک تابع با ورودی‌هایش صدا زده می‌شود، مقادیر ورودی درون این متغیرها قرار می‌گیرند. در نتیجه برای خواندن آن مقادیر در بدنه اصلی تابع از این متغیرها (در این جا arg1، arg2 و arg3) کمک می‌گیریم. (بیشتر بدانید :انواع متغیر در پایتون)

خروجی تابع

برخی از توابع باید یک خروجی به ما بدهند. برای مثال فرض کنید تابع add() را برای جمع کردن دو عدد تعریف می‌کنیم. این تابع باید دو ورودی و یک خروجی داشته باشد.

ورودی‌ها همان اعداد مورد نظر برای عملیات جمع بوده و خروجی، حاصل جمع ما خواهد بود.

برای تعریف خروجی تابع در پایتون از کلمه کلیدی return استفاده می‌شود. return هر مقداری که در جلویش قرار داشته باشد را به عنوان خروجی تابع به ما باز می‌گرداند.

```
def add( a, b ):
    return a + b
```

برای اینکه معنی دقیق خروجی تابع را متوجه شوید، به مثال زیر توجه کنید.

```
a = input("Enter First Number: ")
b = input("Enter Second Number: ")
result = add(a,b)
print(result)
```

در قطعه کد بالا از روش **گرفتن ورودی در پایتون** استفاده کرده و دو مقدار را از کاربر می‌گیرم. سپس روی آن‌ها محاسباتی کرده و نتیجه را در متغیر result نگهداری می‌کنیم. در نهایت نیز مقدار را چاپ می‌کنیم.

همانطور که دیدید، اگر در تعریف تابع add() از return استفاده نمی‌شد، با اجرای خط سوم کد بالا، حاصل جمع درون متغیر result قرار نمی‌گرفت.

ترفندهای کار با توابع پایتون

تعریف و استفاده از توابع در پایتون به همین سادگی است! در ادامه ۲ ترفند و ۲ نکته برای حرفه‌ای‌تر شدن در هنگام کار با توابع با شما مطرح می‌کنم.

ارسال پارامتر با ارجاع (رفرنس)

تمام پارامترهایی که به صورت شئ هستند، در زبان پایتون با ارجاع (Reference) فراخوانی می‌شوند. برای درک بهتر موضوع، اجازه دهید با یک مثال ادامه دهیم.

فرض کنید تابعی دارید که یک لیست به عنوان ورودی می‌گیرد. سپس آن را مرتب کرده و در محیط برنامه چاپ می‌کند.

```
def list_sort( lst ):
    lst.sort()
    print( "Sorted List is: ", lst )
```

حال یک لیست با مقادیر به هم ریخته ایجاد کرده و آن را به تابع می‌دهیم تا نتیجه را به ما نمایش دهد.

```
mylist = [21, 15, 36, 18, 27]
list_sort( mylist )
```

خروجی کد بالا چیزی شبیه زیر را به ما خواهد داد:

```
Sorted List is:  [15, 18, 21, 27, 36]
```

حال اگر مقدار mylist را چاپ کنیم؛ با کمال تعجب می‌بینیم که لیست ما که بیرون از تابع بوده هم به صورت صعودی مرتب شده است!

```
print( mylist )
# [15, 18, 21, 27, 36]
```

اما چرا؟
دلیل این اتفاق، ارسال پارامتر با ارجاع است. وقتی که ما تابع را همراه با لیست صدا می‌زنیم، لیست به عنوان یک شئ و به صورت Reference به بدنه تابع فرستاده می‌شود. پس هر تغییری که در آن ایجاد کنیم، در لیست اصلی هم اعمال خواهد شد.

دیکشنری

دیکشنری در زبان برنامه نویسی پایتون یک نوع داده‌ای محبوب و نسبتاً پر کاربرد است. این نوع داده‌ای در اصل یک نوع لیست انجمنی (Associative) یا لیست کلید-مقدار (key-value) هست. در این آموزش به طور جامع کار با دیکشنری در پایتون را یاد می‌گیریم.
معمولاً از دیکشنری پایتون در جاهایی استفاده می‌کنیم که نیاز داریم مقادیری را با استفاده از یک کلید مرتبط با هر کدام از آن‌ها شناسایی کنیم. این مقادیر هر چیزی می‌توانند باشند؛ برای مثال، رشته متنی، عدد، شئ و. ...
در ابتدا به این موضوع می‌پردازیم که دیکشنری در پایتون چیست و پس از آن یک دیکشنری ساخته و کار با دیکشنری پایتون را خواهیم آموخت. همچنین در انتها در مورد نوع کلیدها و مقادیر بیشتر صحبت کرده و چند ترفند کاربردی را با هم مرور می‌کنیم.
دیکشنری (Dictionary) در زبان فارسی معادل فرهنگ لغت یا واژه‌نامه است. در یک کتاب فرهنگ لغت، یک کلمه با معادلش مرتبط می شود. این معادل می‌تواند یکی بوده یا چندین مورد باشد، اما کلمه اصلی یکی است! در اصل یک یا چند معنی معادل را به یک کلمه مرتبط کرده‌ایم.
در دیکشنری پایتون هم دقیقاً مشابه چنین کاری را انجام می‌دهیم؛ یک کلمه (که آن را به عنوان کلید می‌شناسیم) با یک مقدار(string)، عدد، رفرنس شئ و ... مرتبط خواهد شد.
معمولاً اعضای یک دیکشنری مواردی مرتبط با هم هستند و هر یک از اعضا یک خصیصه یا ویژگی از مورد اصلی را بیان می‌کنند. اما این مسئله تماماً وابسته به تعاریف و نوع برنامه‌نویسی شما به عنوان برنامه‌نویس خواهد بود و هیچ قاعده و قانون خاصی در مورد مرتبط بودن اعضای یک دیکشنری نداریم!

ساخت دیکشنری در پایتون

یک دیکشنری با علامت }شروع شده و با { خاتمه می‌یابد. اعضای کی دیکشنری به صورت کلید و مقدار (key value) در درون آن تعریف شده و مشابه سایر ساختارهای رایج ذخیره‌سازی در پایتون، با استفاده از کاما (,) از یکدیگر جدا می‌شوند.
برای مثال فرض کنید می‌خواهیم مشخصات فردی را درون یک دیکشنری داشته باشیم؛ دیکشنری را به صورت زیر تعریف می‌کنیم.

```
person  = {
```

```
    "name": "hesam",
    "job": "graphist",
    "car": "BMW x6",
    "age": 24,
    "code": 134
}
```

به همین سادگی توانستیم یک دیکشنری حاوی اطلاعات فردی به نام hesam را ایجاد کنیم!

تعداد عناصر موجود در یک دیکشنری پایتون نامحدود است. ما می‌توانیم یک دیکشنری با هزاران عنصر داشته یا فقط درون آن یک عنصر را نگهداری کنیم. فقط باید قوانین تعریف کلید و مقدار برای آن‌ها رعایت شود. (که در انتهای مقاله در مورد آن بحث می‌کنیم.)

ساخت دیکشنری با سازنده dict

را صدا زده و تمام dict() آن نیز استفاده کرد. به این منظور تابع (Constructor) برای ایجاد دیکشنری می‌توان از سازنده .کلیدها و مقادیر آن‌ها را به صورت پارامترهای جداگانه به عنوان ورودی به تابع می‌دهیم:

```
person = dict( name="hesam", job="graphist", car="BMW x6", age=24, code=134 )
```

نتیجه ذخیره شده در متغیر person نیز مشابه حالت قبل است. در حقیقت تفاوت خاصی در دیکشنری ایجاد شده در این دو حالت وجود ندارد. شما می‌توانید با هر روشی که راحت‌تر هستید از آن‌ها استفاده کنید.

البته همان‌طور که می‌دانید، گاهی اوقات شرایط برنامه و موقعیت‌های پیش رو نحوه استفاده از آن را برای ما تعیین می‌کند.

کار با دیکشنری در پایتون

تا به این جای کار توانستیم یک دیکشنری در پایتون بسازیم. قطعاً لازم است بتوانیم با عناصر موجود در آن عملیات‌هایی انجام داده تا برنامه خود را پیش ببریم. در ادامه روش‌های کار با عناصر دیکشنری را می‌خوانیم.

دسترسی به مقادیر دیکشنری

برای دسترسی به مقادیر (عناصر) ذخیره‌شده در دیکشنری، دو راه وجود دارد.

راه اول و ساده‌ترین راه‌حل مشابه دسترسی به خانه‌های یک لیست در پایتون است. به این صورت که با استفاده از علامت []می‌توان به تک تک عناصر دسترسی داشت؛ با این تفاوت که به جای مقدار عددی برای اَندیس، از مقدار کلیدها به عنوان ایندکس استفاده خواهیم کرد.

اگر در دیکشنری‌ای که ابتدای کار ساختیم بخواهیم به نام فرد دسترسی پیدا کنیم، به صورت زیر عمل خواهیم کرد:

```
print( "Name: " + person["name"] )
```

خروجی قطعه کد بالا چیزی شبیه زیر خواهد شد:

```
Name: hesam
```

راهکار دوم برای دسترسی به مقدار یک کلید در دیکشنری پایتون، استفاده از تابع get() روی دیکشنری است. این تابع یک ورودی می‌گیرد که همان کلید مورد نظر ماست و سپس مقدار مرتبط با کلید را بازمی‌گرداند.

```
print( "Name: " + person.get("name") )
```

```
# output: Name: hesam
```

خطای KeyError در دیکشنری پایتون

اگر در هنگام فراخوانی یک مقدار از دیکشنری، کلیدی را صدا بزنیم که وجود نداشته باشد، با خطایی از نوع KeyErrorمواجه خواهیم شد. (یادگیری بیشتر : مدیریت خطا در پایتون)

```
info = {'name': 'omid', 'job': 'programmer', 'code': 177}
print( info['name'] )
print( info['family'] )

# omid
# KeyError: 'family'
```

تغییر مقادیر دیکشنری در پایتون

برای به‌روزرسانی مقدار هر یک از کلیدها مشابه تخصیص مقدار به ایندکس‌های مختلف یک لیست عمل می‌کنیم.

در قطعه کد زیر، مقدار 23 را در کلید age از دیکشنری person قرار داده‌ایم.

```
person["age"] = 23
```

همان‌طور که احتمالاً درست حدس زده‌اید، می‌توان برای به‌روزرسانی مقدار یک کلید در دیکشنری از همان مقدار یا مقدارهای دیگر درون آن دیکشنری نیز استفاده کرد.

مثلاً در مثال زیر، سن فعلی کاربر را به اندازه یک واحد افزایش می‌دهیم.

```
person["age"] = person["age"] + 1
```

افزودن عنصر جدید به دیکشنری پایتون

افزودن عضو جدید به دیکشنری در پایتون بسیار آسان و ساده است. برای این کار، مشابه تغییر مقدار یک کلید در دیکشنری عمل می‌کنیم، با این تفاوت که کلید صدا زده شده، کلید جدید و مقداری که به آن تخصیص می‌یابد مقدار مورد نظرمان خواهد بود.

```
person["mobile"] = "Samsung Galaxy Note 7"
```

حذف عناصر از دیکشنری

برای حذف یک کلید و مقدار مرتبط با آن، دو روش خواهیم داشت.

روش اول استفاده از دستور del است. در جلوی این دستور کلید مورد نظر در دیکشنری را فراخوانی می‌کنیم.

```
print( person )
del person["mobile"]   # delete 'mobile' key in dictionary
print( person )
```

در خروجی قطعه کد بالا، در پرینت دوم می‌بینیم که کلید mobile و مقدار مربوط به آن از درون دیکشنری حذف شده است.

```
{'name': 'hesam', 'job': 'graphist', 'car': 'BMW x6', 'age': 24, 'code': 134, 'mobile': 'Samsung Ga
laxy Note 7'}
{'name': 'hesam', 'job': 'graphist', 'car': 'BMW x6', 'age': 24, 'code': 134}
```

روش دوم برای حذف یک المان از دیکشنری استفاده از متد pop() در دیکشنری پایتون است. این تابع به صورت متد روی دیکشنری مورد نظر صدا زده شده و یک ورودی می‌گیرد.

ورودی آن همان مقدار کلید مورد نظر ماست که می‌خواهیم از دیکشنری حذف کنیم.

```
person.pop("car")
```

حذف کل دیکشنری در پایتون

گاهی لازم است تا کل دیکشنری را حذف کنیم. منظور از کل دیکشنری، حذف تمام کلیدها و مقدارها و پس از آن حذف متغیر دیکشنری است.

برای این کار کافی است تا نام دیکشنری را در مقابل دستور del بنویسیم. با این کار کل دیکشنری حذف خواهد شد.

```
del person   # delete person dictionary object
```

حذف تمام عناصر دیکشنری

اما گاهی نیاز است که صرفاً دیکشنری را خالی کنیم؛ یعنی بدون حذف خود دیکشنری، فقط کلیدها و مقدارها را حذف کنیم.

برای این کار متد clear() را روی شیء دیکشنری صدا می‌زنیم.

با صدا زدن این تابع، تمام کلیدهای موجود در دیکشنری حذف شده و در نهایت یک دیکشنری خالی خواهیم داشت.

```
person.clear() # delete all keys in Dictionary
print( person )

# output: {}
```

قوانین تعریف کلید و مقدار در دیکشنری پایتون

کلیدهای دیکشنری در پایتون باید دو ویژگی اصلی داشته باشند.

۱. یکتا باشند (تکراری نباشند)

۲. تغییرناپذیر باشند.

مورد اول که تقریباً واضح است. در حقیقت به ازای هر کلید در دیکشنری نمی‌توان بیشتر از یک مقدار داشت. در صورتی که یک کلید دو بار در دیکشنری تعریف شود، مقدار دوم آن جایگزین اولی شده و همواره مقدار دوم را خواهیم داشت. (به نوعی تغییر مقدار صورت می‌گیرد)

در مورد دومین قانون، کلید را به عبارتی هویت مقدار (value) مرتبط با آن می‌دانند. می‌دانیم که هویت را نمی‌توان تغییر داد. بنابراین کلیدها در دیکشنری پایتون می‌بایست یک مقدار تغییرناپذیر باشند. در نتیجه نمی‌توان از یک متغیر به عنوان کلید استفاده کرد.

کپی کردن دیکشنری در پایتون

همان‌طور که می‌دانید، متغیرهایی که به اشیاء اشاره می‌کنند حاوی رفرنسی به آن شئ هستند. پس اگر یک دیکشنری را با استفاده از علامت تخصیص (=) به متغیر دیگری بدهیم، دیکشنری کپی نمی‌شود؛ بلکه فقط رفرنس آن، درون متغیر دوم ریخته خواهد شد. در نتیجه با تغییر دیکشنری اول، دیکشنری دوم ما نیز تغییر می‌کند.

```
person_backup = person   # not a true way to copy dictionary
```

برای کپی کردن دیکشنری در پایتون می‌توان از متد ()copyروی شیء دیکشنری استفاده کرد. (یادگیری بیشتر :اصول برنامه نویسی شی گرا)

```
person_backup = person.copy()
```

حالا اگر دیکشنری person یا person_backup را تغییر دهیم، محتوای دیکشنری دیگر هیچ تغییری نخواهد کرد.

مثالی از دیکشنری

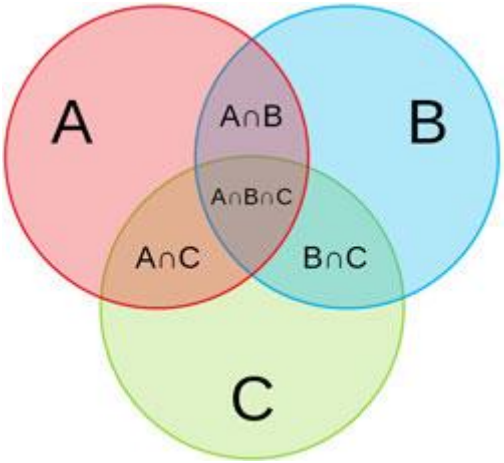
```
# Defining a dictionary
my_dict = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Accessing values using keys
print(my_dict["name"])  # Output: John
print(my_dict["age"])  # Output: 30
print(my_dict["city"])  # Output: New York

# Adding a new key-value pair
my_dict["job"] = "Engineer"
print(my_dict)  # Output: {'name': 'John', 'age': 30, 'city': 'New York', 'job':
'Engineer'}

# Removing a key-value pair
del my_dict["age"]
print(my_dict)  # Output: {'name': 'John', 'city': 'New York', 'job': 'Engineer'}

# Length of the dictionary
print(len(my_dict))  # Output: 3

# Iterating through key-value pairs
for key, value in my_dict.items():
    print(key, ":", value)
```

همانطور که در مثال بالا می‌بینیم :

- برای تعریف دیکشنری از {} استفاده می‌شود.
- برای دسترسی به یک آیتم خاص my_dict[key] استفاده می‌شود .
- برای اضافه کردن یک زوج کلید-مقدار از my_dict[key] = value استفاده می‌شود.
- برخلاف تاپل ها دیکشنری ها قابلیت تغییر دارن و می‌توان آنها را تغییر داد .(مانند my_dict[key] = new_value.
- برای حذف کردن یک کلید خاص از del استفاده می‌شود.
- مانند تاپل ها می‌توان با استفاده از تابع len(my_dict) تعداد زوج کلید-مقدار ها را در دیکشنری پیدا کرد.
- در دیکشنری ها به ازای هر کلید یکتا یک مقدار وجود دارد.
- همین طور تنها مجاز به استفاده از داده ساختارهایی به عنوان کلید هستیم که تغییرناپذیر هستند. مانند(رشته ها ، اعداد، ...)

مجموعه (Set)

از مجموعه در پایتون برای نگهداری داده‌ها ا ستفاده می شود. مجموعه یا set یک نوع داده‌ای م شابه لی ست ا ست که می‌تواند داده‌های از یک نوع یا نوع‌های مختلف را درون خود نگه دارد. دو تفاوت اصـــلی مجموعه‌ها، عدم ترتیب عناصـــر و غیر تکراری بودن اعضای آن است. در این آموزش با مجموعه‌ها و ترفندهای کار با مجموعه پایتون آشنا خواهید شد.
اگر بخواهم مجموعه‌های پایتون (python set) را خیلی ســاده معرفی کنم، شــما را به نظریه مجموعه‌ها در ریاضــی ارجاع می‌دهم! مطمئناً در درس ریاضی با مجموعه‌ها آشنا شده‌اید. اگر هم الآن حضور ذهن ندارید، جای هیچ نگرانی نیست.
در مجموعه‌ها گروهی از عنا صر را نگهداری می‌کنیم. می‌توان به هر مجموعه اعضای جدید ا ضافه کرد یا اعضای قبلی را حذف کرد ولی اعضای موجود را نمی‌توان تغییر داد. همچنین بین دو مجموعه عملگرهای مختلفی تعریف می‌شوند؛ عملگرهایی نظیر اجتماع، اشتراک، تفاضل.



مجموعه (Venn) نمایش نمودار ون

در برنامه‌نویسی پایتون نیز از مجموعه‌ها دقیقاً به همین منظور استفاده می‌شود. ابتدا با ویژگی‌های setدر پایتون آشنا شده و نحوه تعریف آن را یاد می‌گیریم. سپس توابع اصلی کار با مجموعه را معرفی خواهم کرد.

مجموعه در پایتون

set یک ساختمان داده built-in است. اعضای یک مجموعه پایتون دارای سه ویژگی اصلی زیر هستند:

- **نامرتب(unordered)**
- **غیرقابل تغییر(unchangeable)**
- **غیر تکراری(non-duplicate)**

منظور از نامرتب بودن این است که نمی‌توان ترتیب خاصی برای عناصر درون مجموعه در نظر گرفت. علاوه بر این، مجموعه برخلاف لیست در پایتون دارای اندیس (ایندکس) نیستند. در نتیجه نمی‌توان به یکی از اعضای آن به وسیله اندیس دسترسی داشت.

اعضای یک مجموعه غیر قابل تغییر هستند؛ بنابراین نمی‌توان یک مقدار را تغییر داد. در حقیقت چون نمی‌توانیم به یک عنصر مشخص دسترسی داشته باشیم، نمی‌توانیم آن را تغییر دهیم.

همه اعضای یک مجموعه در پایتون به صورت یکتا هستند. یعنی هیچ دو عضوی از یک مجموعه وجود ندارد که با هم برابر باشند. در جلوتر می‌بینیم که اگر در هنگام تعریف مجموعه سعی کنیم مقادیر تکراری داشته باشیم، به صورت خودکار فقط یکی از آن‌ها باقی می‌ماند.

تعریف مجموعه پایتون

دو نوع ساختار تعریف برای setدر پایتون داریم. زمانی از اولین حالت استفاده می‌کنیم که حداقل یک مقدار از مجموعه را در اختیار داشته باشیم. برای تعریف مجموعه، از علامت آکولاد استفاده می‌شود. با {مجموعه را شروع کرده و با }خاتمه می‌دهیم. هر عضو در مجموعه نیز با استفاده از کاما (,) از یکدیگر جدا می‌شوند.

```
num_set = {17, 81, 4, 11}
str_set = {"Negar", "Omid", "Ehsan"}
```

برای تعریف مجموعه نمی‌توانیم به طور تنها از علامت {}استفاده کنیم. برعکس لیست‌ها که با گذاشتن []هم تعریف می‌شدند، اگر فقط از آکولاد استفاده کنیم، یک دیکشنری پایتونی
خواهیم داشت. به کد زیر دقت کنید:

```
my_set = {17, 81, 4, 11}
print( type(my_set) )   # <class 'set'>

test = {}
print( type(test) )   # <class 'dict'>
```

روش دوم برای تعریف مجموعه در پایتون استفاده از سازنده set() است. با صدا زدن این تابع، یک مجموعه خالی ایجاد می‌شود که به کمک توابعی که در قسمت بعدی می‌بینیم خواهیم توانست مقادیر جدیدی به آن اضافه کنیم.

```
X = set()
```

تابع **set()** می‌تواند به عنوان ورودی یک مجموعه داده تکرارگر یا iterator در پایتون دریافت کند. داده‌های تکرارگر داده‌هایی هستند که قابل پیمایش هستند. بنابراین می‌توانیم یک لیست، تاپل یا رشته را به عنوان ورودی این تابع داده و آن را به مجموعه تبدیل کنیم.

```
lst = ["Shiraz", "Tehran", "Tabriz", "Mashhad"]
cities = set(lst)
print(cities)
# outpuut:
# {'Mashhad', 'Tabriz', 'Tehran', 'Shiraz'}
```

می‌توان به جای لیست، یک رشته متنی را به set() داده و مجموعه‌ای از کاراکترهای یکتای رشته مورد نظر را دریافت کنیم.

```
site_name = "mathhome.com"
site_chars = set(site_name)
print(site_chars)
# output:
# {'m', 'a', 't', 'h', 'h', 'o', 'm', 'e', '.', 'c', 'o', 'm'}
```

دقت کنید که در مثال بالا، برخی از کاراکترها مثل s که چندین بار در رشته به کار گرفته شده‌اند، فقط یک بار در مجموعه قرار می‌گیرند.

یادآوری می‌کنم که ویژگی‌های اصلی مجموعه در پایتون Python Set نامرتب بودن، غیرتکراری بودن و غیر قابل تغییر بودن

اعضای مجموعه است. این نوع داده‌ای مبتنی بر ساختمان داده جدول هش (Hash Table) است.

کار با مجموعه‌های پایتون

داده‌هایی که در یک مجموعه قرار می‌دهیم می‌توانند از یک نوع یا نوع‌های مختلف باشند. برای مثال در قطعه کد زیر می‌بینید که سه set مختلف با داده‌های متنوع ایجاد شده است.

```
my_set1 = {6, 17, 28}
my_set2 = {"omid", 2557, "mathhhome", (2,5)}
my_set3 = {2.4, "Hello", (4,7,9)}
```

گفتم که اعضای مجموعه در پایتون تغییر ناپذیر (immutable) هستند. به این معنی که مقدار یک عنصر را در مجموعه نمی‌توان تغییر داد. اما توجه کنید که set در پایتون تغییرپذیر است .یعنی می‌توانیم اعضای جدیدی به آن اضافه کرده یا برخی از اعضا را حذف کنیم. در ادامه با توابعی آشنا می‌شویم که چنین کارهایی انجام خواهند داد.

یادآور می‌شوم که نمی‌توانیم به طور مستقیم و از طریق index به اعضای موجود در مجموعه دسترسی داشته باشیم.

معمولاً یکی از کارهایی که با داده‌های مجموعه‌ای انجام می‌دهیم، محاسبه اندازه آن‌هاست. منظور از اندازه، تعداد اعضای درون یک مجموعه است. برای محاسبه اندازه مجموعه در پایتون از تابع len() استفاده می‌کنیم.

فقط کافی است set را به عنوان ورودی به این تابع بدهیم تا تعداد اعضای مجموعه را داشته باشیم.

```
city_set = {"Shiraz", "Tehran", "Tabriz", "Mashhad", "Isfahan"}
print( len(city_set) )
# output:
# 5
```

بررسی موجود بودن عضوset

یکی از کارهای متداول دیگری که با مجموعه‌ها انجام می‌دهیم، بررسی وجود یک مقدار در مجموعه است. این فرآیند معمولاً در دستورات شرطی پایتون و برای گرفتن یک تصمیم انجام می‌شود.

برای اینکه وجود یک مقدار در set را بررسی کنیم، از کلمه کلیدی in استفاده می‌کنیم. به مثال زیر توجه کنید:

```
print( "Shiraz" in city_set )
# output:
# True

test_set = {1, 2, 3, 5, 8}
if 7 in test_set:
   print("Yes!")
else:
   print("No!")
```

در قسمت دوم این کد، پس از تعریف مجموعه test_set، در یک شرط if بررسی کرده‌ایم که آیا مقدار ۷ در مجموعه قرار دارد یا خیر. با اجرای کد، مقدار No! چاپ می‌شود.

برای اینکه عدم وجود عنصر در مجموعه را بررسی کنیم، از عبارت not in استفاده خواهیم کرد. در قطعه کد زیر، عدم وجود nazanin در مجموعه را بررسی کرده‌ایم:

```
names_set = {"negar", "omid", "roya", "ehsan", "roya"}
if "nazanin" not in names_set:
   print("OK!")
```

سؤال: در تعریف این مجموعه، اسم roya دو بار تکرار شده است. اگر مقدار names را چاپ کنیم، خواهیم دید که خروجی شبیه تصویر زیر است. به نظر شما چرا ؟!

نتیجه چاپ

names_set

```
>>> names_set
{'negar', 'roya', 'ehsan', 'omid'}
```

افزودن عضو به مجموعه

گفتیم برخلاف عناصر مجموعه، خود مجموعه‌ها قابل تغییر هستند. برای افزودن یک عضو جدید به مجموعه در پایتون از متد add() استفاده می‌کنیم. این متد به صورت تابعی روی نوع داده مجموعه تعریف شده است.

بنابراین برای استفاده از آن، باید متد را روی مجموعه مورد نظر صدا بزنیم. این متد یک ورودی می‌گیرد که مقدار عنصر جدید را مشخص می‌کند.

```
test_set.add(11)   # {1, 2, 3, 5, 8, 11}
```

گاهی نیاز داریم چند عضو جدید را از یک لیست به مجموعه خود اضافه کنیم. روش اول استفاده از یک حلقه for برای افزودن همه عناصر list به set است.

روش دوم و بهتر، استفاده از متد ()update است. این متد یک لیست را به عنوان ورودی گرفته و تمام اعضای آن را به مجموعه ما اضافه می‌کند.

```
new_nums = [9, 15, 13]
test_set.update(new_nums)

print( test_set )
# {1, 2, 3, 5, 8, 9, 11, 13, 15}
```

حذف از set پایتون

برای حذف یک عنصر از مجموعه در پایتون از متدهای ()discard و ()remove استفاده می‌شود. این دو متد یک مقدار را به عنوان ورودی گرفته و از درون مجموعه مورد نظر ما حذف می‌کنند.

این دو متد تنها یک تفاوت با هم دارند: اگر مقدار مورد نظر در مجموعه وجود نداشته باشد، ()discard هیچ کار خاصی انجام نمی‌دهد اما ()remove خطا (error) می‌دهد.

```
test_set.remove(8)
# {1, 2, 3, 5, 9, 11, 13, 15}

test_set.remove(7) # Error!

test_set.discard(7)
# {1, 2, 3, 5, 9, 11, 13, 15}
```

عملگرهای مجموعه در پایتون

اگر از درس ریاضی به خاطر داشته باشید، بین مجموعه‌ها عملیات‌های مختلفی انجام می‌دادیم. گاهی لازم است برخی از آن کارها را در پایتون هم انجام دهیم. در ادامه ۳ عملگر پر استفاده را بررسی می‌کنیم.

در این جا برای درک ساده‌تر مثال‌ها، از عناصر عددی استفاده می‌کنم. اما نوع‌های داده‌ای مجموعه‌ها هر چیزی می‌توانند باشند.

(بیشتر بیاموزید :کار با عدد در پایتون) دو مجموعه A و B را در نظر بگیرید:

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
```

اجتماع دو مجموعه(union)

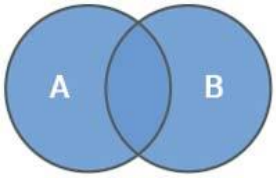
اجتماع دو مجموعه عبارت است از مجموعه‌ای از تمام اعضای هر دو مجموعه. برای اجتماع‌گیری روی مجموعه‌ها در پایتون از عملگر | یا متد ()union استفاده می‌شود.

عملگر | بین دو مجموعه قرار می‌گیرد و مشابه عملگرهای ریاضی، نتیجه نهایی را به ما خواهد داد. متد ()union روی یکی از مجموعه‌ها صدا زده شده و مجموعه دیگر به عنوان ورودی به آن داده می‌شود.

```
A_union_B = A | B
# A_union_B = {1, 2, 3, 4, 5, 6, 7, 8}
```

```
print( A.union(B) )
# {1, 2, 3, 4, 5, 6, 7, 8}
```

نمودار عمل اجتماع بین دو مجموعه



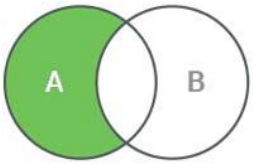
تفاضل دو مجموعه در پایتون

تفاضل یا تفاوت مجموعه B از A یک مجموعه از تمام اعضایی است که فقط در A قرار دارند. به عبارت دیگر اعضایی از A که در B حضور ندارند.

برای محاسبه تفاوت دو set از عملگر تفریق (-) یا متد ()difference استفاده می‌شود. به مثال زیر توجه کنید:

```
print( A - B )
# print( A - B )

print( A.difference(B) )
# {1, 2, 3}

print( B.difference(A) )
# {8, 6, 7}
```

نمودار تفاضل دوset



دقت کنید که در حالت عمومی مجموعه A-B برابر B-A نیست.

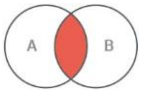
اشتراک مجموعه(intersection)

اشتراک دو مجموعه، مجموعه‌ای از اعضایی است که در هر دو مجموعه حضور دارند. در صورتی که دو مجموعه هیچ عنصر مشترکی نداشته باشند، اشتراک آن‌ها خالی (تهی) خواهد بود.

برای انجام عملیات اشتراک روی دو مجموعه در پایتون از عملگر & یا متد ()intersection استفاده می‌شود.

```
print( A & B )
# {4, 5}

print( A.intersection(B) )
# {4, 5}
```

نمودار عملیات اشتراک بین دو مجموعه



مثالی از set

```
# Define a set
my_set = {1, 2, 3, 4, 5}
# Print the set
print(my_set)  # Output: {1, 2, 3, 4, 5}
# Add elements to the set
my_set.add(6)
my_set.add(7)
print(my_set)  # Output: {1, 2, 3, 4, 5, 6, 7}
# Adding a duplicate element (which will be ignored)
my_set.add(5)
print(my_set)  # Output: {1, 2, 3, 4, 5, 6, 7}
# Remove an element from the set
my_set.remove(3)
print(my_set)  # Output: {1, 2, 4, 5, 6, 7}
# Check if an element is in the set
print(2 in my_set)  # Output: True
print(10 in my_set)  # Output: False
# Length of the set
print(len(my_set))  # Output: 6
# Initialize a set with different data types
mixed_set = {1, 2.5, "hello", (1, 2, 3)}
print(mixed_set)  # Output: {1, 2.5, 'hello', (1, 2, 3)}
```

در این مثال:

- در مجموعه‌ها عضو تکراری وجود ندارد.
- در مجموعه‌ها ترتیب عناصر براساس وارد شدن آن‌ها نیست.
- مجموعه‌ها در پایتون تغییرپذیر هستند.
- با استفاده از تابع len می‌توان سایز مجموعه را پیدا کرد.
- با استفاده از دستورات add , remove می‌توان عضوی را از مجموعه حذف یا اضافه کرد.
- با استفاده کلید واژه in می‌توان فهمید عضوی در مجموعه وجود دارد یا خیر.

تاپل(Tuple):

تاپل یکی از ساختارهای داده‌ای در زبان برنامه‌نویسی پایتون است که برای ذخیره‌سازی چندین مقدار در یک متغیر استفاده می‌شود. تاپل‌ها مانند لیست‌ها هستند، اما با این تفاوت که تاپل‌ها غیرقابل تغییرند؛ یعنی پس از ایجاد، مقادیر آنها نمی‌توانند تغییر کنند. تاپل‌ها معمولاً با استفاده از پرانتز گرد () تعریف می‌شوند. این ساختار داده‌ای برای ذخیره‌سازی اطلاعاتی که قرار است در طول زمان تغییر نکنند، مفید است.

```
# Defining a tuple

my_tuple = ("apple", "banana", "cherry")

# Accessing elements of a tuple

print(my_tuple[0])  # Output: apple

print(my_tuple[1])  # Output: banana

print(my_tuple[2])  # Output: cherry

# Attempting to change a tuple (which will result in an error)

# my_tuple[0] = "orange"  # This line will raise an error because tuples are immutable


# Length of a tuple

print(len(my_tuple))  # Output: 3

# Iterating through a tuple

for item in my_tuple:
    print(item)

# Tuple with different data types

mixed_tuple = ("apple", 1, True)

print(mixed_tuple)  # Output: ('apple', 1, True)
```

کد بالا مثالی از استفاده از تاپل ها است. همانطور که در کد بالا مشخص است.

- برای دسترسی به آیتم‌های یک تاپل میتوان مانند لیست ها از [] استفاده کرد.
- همچنین در تاپل ها اجازه عوض کردن آیتم ها را نداریم پس مجاز به نوشتن my_tuple[0] = "orange " نیستیم.
- همچنین می‌توانیم روی آیتم‌های تاپل فور بزنیم.
- همچنین مجاز به تعریف چند نوع متغییر در تاپل ها هستیم.
- با استفاده از تابع len(my_tuple) میتوان سایز تاپل را پیدا کرد

کتابخانه یا ماژول‌ها در پایتون:

در پایتون، کتابخانه یا به انگلیسی Module یا Library، مجموعه‌ای از توابع، توابع کمکی و کلاس‌ها است که به منظور استفاده مشترک در برنامه‌های پایتون توسعه داده شده‌اند. کتابخانه‌ها در پایتون معمولاً به منظور انجام وظایف خاص مورد استفاده قرار می‌گیرند، مانند پردازش رشته، محاسبات علمی، اتصال به پایگاه‌های داده، و غیره.

مثالی از کتابخانه math:

```
import math

# Calculate the square root of a number

num = 16

square_root = math.sqrt(num)

print("Square root of", num, ":", square_root)  # Output: 4.0

# Calculate the factorial of a number

num = 5

factorial = math.factorial(num)

print("Factorial of", num, ":", factorial)  # Output: 120

# Calculate the value of pi

pi_value = math.pi

print("Value of pi:", pi_value)  # Output: 3.141592653589793
```

برای استفاده کردن از یک ماژول ابتدا آن را باید در برنامه بشناسیم بدین منظور از کلمه کلیدی import استفاده می‌شود. (در این مثال import math)

برای استفاده از توابع یک ماژول راه‌های متفاوتی است. یکی از راه‌ها در این مثال آورده شده است که از توابع sqrt که برای محاسبه جذر یک عدد و factorial برای محاسبه فاکتوریل یک عدد اسفاده میشود همچنین مقدار math.pi مقدار عدد پی را به ما برمی‌گرداند.
می‌توان از کلید as استفاده کرد و اسم دلخواه را به عنوان اسم ماژول استفاده کرد به مثال زیر دقت کنید.

```
import math as mm

print (mm.factorial(5))  # 120
```

در این مثال ما بجای استفاده از کلمه math از mm استفاده کرده‌ایم.

نامپای: (numpy)

نامپای یا NumPy یکی از کتابخانه‌های پرکاربرد در زبان برنامه‌نویسی پایتون است که برای انجام عملیات عددی و محاسبات علمی به کار می‌رود. این کتابخانه به طور گسترده در علوم مختلف مانند علوم داده، ریاضیات، مهندسی، فیزیک، آمار و... استفاده می‌شود.
NumPy قابلیت‌های مختلفی از جمله آرایه‌های چند بعدی، عملیات ماتریسی، توابع ریاضیاتی پیشرفته، تولید داده تصادفی و ... را فراهم می‌کند. این کتابخانه با استفاده از آرایه‌های NumPy، کارهای عملیاتی سریع‌تر و بهینه‌تری را انجام می‌دهد که در پردازش داده‌های بزرگ و عملیات عددی پیچیده بسیار مفید است.
برای استفاده از NumPy، معمولاً ابتدا باید آن را با استفاده از دستور import numpy as np وارد کنید، که به این شکل عموماً استفاده می‌شود:

```
import numpy as np
```

نصب numpy

در NumPy، برای ساخت آرایه از توابع مختلفی استفاده می‌شود. این آرایه‌ها می‌توانند یک بعدی یا چند بعدی باشند. در ادامه چند مثال از ساخت آرایه‌ها در NumPy آورده شده است:

```
import numpy as np

# ساخت آرایه یک بعدی از لیست
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)  # Output: [1 2 3 4 5]

# ساخت آرایه صفر از طول داده شده
arr2 = np.zeros(5)
print(arr2)  # Output: [0. 0. 0. 0. 0.]

# ساخت آرایه پر از یک
arr3 = np.ones(5)
print(arr3)  # Output: [1. 1. 1. 1. 1.]

# ساخت آرایه خالی
arr4 = np.empty(5)
print(arr4)  # Output: [0. 0. 0. 0. 0.]

# ساخت آرایه مقدار دهی شده به صورت خطی
arr5 = np.arange(1, 10, 2)  # اعداد فرد از ۱ تا ۹
print(arr5)  # Output: [1 3 5 7 9]
```

برای ساخت آرایه یک بعدی می‌توان از توابع بالا استفاده کرد.

```
import numpy as np
# ساخت آرایه دو بعدی از یک لیست ۲D
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2d)
# Output:
# [[1 2 3]
#  [4 5 6]]

# ساخت آرایه دو بعدی با اندازه خاص و مقدار دهی صفر
arr_zeros = np.zeros((3, 4))
print(arr_zeros)
# Output:
# [[0. 0. 0. 0.]
#  [0. 0. 0. 0.]
#  [0. 0. 0. 0.]]

# ساخت آرایه دو بعدی با اندازه خاص و مقدار دهی یک
arr_ones = np.ones((2, 3))
print(arr_ones)
# Output:
# [[1. 1. 1.]
#  [1. 1. 1.]]

# ساخت آرایه دو بعدی با اندازه و مقدار دهی صفر به صورت خاص
arr_custom_zeros = np.full((3, 4), 7)
print(arr_custom_zeros)
# Output:
# [[7 7 7 7]
#  [7 7 7 7]
#  [7 7 7 7]]
```

از کد های بالا هم می‌توان برا ساخت آرایه دو بعدی استفاده کرد و برای بعدهای بیشتر می‌توان از همین روش استفاده کرد. برا دسترسی به اعضای آرایه یک بعدی مانند لیست‌ها عمل کرده ولی برای بعدهای بیشتر از کاما استفاده می‌کنیم. به مثال زیر توجه فرمایید.

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('5th element on 2nd row: ', arr[1, 4])
```

در این مثال برای دسترسی به عنصر ۵ ام سطر دوم از arr[1,4] استفاده کرده ایم.

در NumPy، ماژول random برای تولید اعداد تصادفی و ایجاد داده‌های تصادفی استفاده می‌شود. این ماژول شامل توابع مختلفی برای ایجاد اعداد تصادفی از توزیع‌های مختلف، تولید آرایه‌های تصادفی، و ... است. این ماژول از تابع rand برای تولید اعداد تصادفی از توزیع یکنواخت استفاده می‌کند.

```
import numpy as np

# وارد کردن ماژول random از NumPy
from numpy import random

# تولید یک عدد تصادفی از توزیع یکنواخت بین ۰ و ۱
random_num = random.rand()
print(" ,":تعداد تصادفی از توزیع یکنواخت بین ۰ و ۱random_num)

# تولید یک عدد صحیح تصادفی در بازه [۱, ۱۰۰)
random_int = random.randint(1, 100)
print(" ,":عدد صحیح تصادفی در بازه [۱, ۱۰۰)random_int)

# انتخاب یک عنصر تصادفی از یک لیست
my_list = [1, 2, 3, 4, 5]
random_choice = random.choice(my_list)
print(" ,":یک عنصر تصادفی از لیستrandom_choice)
```