**برنامه نویسی کامپیوتر ـ زمستان ۹۷**
**تدریس توسط: حسین جوهری**
**دانشگاه خواجه نصیرالدین طوسی**

# خلاصه هفته یازدهم

**مفاهیم:** **مجموعه و دیکشنری، چند مثال کاربردی**

در این هفته با ساختارهای داده مجموعه set و دیکشنری dictionary در پایتون آشنا می‌شویم و چند مسئله کاربردی را با استفاده از این ساختارها حل می‌کنیم.

## ۱ مجموعه set در پایتون

ساختارهای داده‌ای مثل لیست، تاپل و رشته یک ویژگی مشترک دارند و آن اینست که عناصری که در این ساختارها ذخیره می‌شوند دارای ترتیب هستند. به همین دلیل عباراتی چون اولین عنصر لیست، دومین عنصر یک تاپل یا چهارمین کاراکتر یک رشته معنی‌دار هستند. ساختار داده مجموعه همانند لیست و تاپل یک سری عناصر را در خود جای می‌دهد اما ترتیب خاصی برای این عناصر وجود ندارد. این ساختار همانند مفهوم مجموعه در ریاضی است. یک مجموعه شامل تعدادی عناصر است که ترتیب خاصی ندارند. طبیعتا یک مجموعه عناصر یکسان و تکراری ندارد. یک مجموعه را در پایتون می‌توان با کلمه set تعریف کرد.

قطعه کد زیر یک مجموعه تهی بنام a تعریف می‌کند. در واقع عبارت set() در پایتون معادل با یک مجموعه تهی است.

```
a = set()
print(a)  # prints set()
```

می توان یک مجموعه غیر تهی را با استفاده از دو کاراکتر { و } تعریف کرد. کد زیر یک مجموعه شامل عناصر $1$ و $2$ و $5$ را تعریف می‌کند.

```
a = {1, 2, 5}
```

دقت کنید که دستور $a = \{\}$ یک دیکشنری تهی را تعریف می‌کند. ساختار دیکشنری را بزودی در این درس تعریف خواهیم کرد.

می‌توان با استفاده از متد add یک عنصر به یک مجموعه اضافه کرد.

```
a = set()
a.add("banana")
a.add("apple")
a.add("cherry")
print(a)  # prints {"banana","apple","cherry"}
a.add("banana")
print(a)  # prints {"banana","apple","cherry"}
```

در مثال بالا می‌بینید در صورتی که عنصری به یک مجموعه اضافه شود که قبل عضو آن بوده تغییری در وضعیت مجموعه ایجاد نمی‌کند.

می‌توان با استفاده از متد remove عنصری را از مجموعه حذف کرد.

```
a.remove("banana")
print(a)  # prints {"apple","cherry"}
```

می‌توان با استفاده از متد clear یک مجموعه را از عناصر خالی کرد.

```
a.clear()
print(a)  # prints set()
```

تعداد عناصر یک مجموعه را می‌توان با تابع len بدست آورد.

```
a = {"banana","apple","cherry"}
print(len(a))  # prints 3
```

## ۱.۱ اشتراک، اجتماع و تفاضل

با استفاده از متد intersection می‌توان اشتراک میان دو مجموعه را بدست آورد. مقدار برگشتی متد intersection خود یک مجموعه است.

```
a = {"banana","apple","cherry"}
b = {"orange","apple","cherry"}
c = a.intersection(b)
print(c)  # prints {"apple","cherry"}
```

با استفاده از متد union می‌توان اجتماع دو مجموعه را بدست آورد.

```
a = {"banana","apple","cherry"}
b = {"orange","apple","cherry"}
c = a.union(b)
print(c)  # prints {'banana', 'orange', 'apple', 'cherry'}
```

با استفاده از متد difference می‌توان تفاضل میان دو مجموعه را حساب کرد. $A - B$ حاوی عناصری از $A$ است که در $B$ نیست.

```
a = {"banana","apple","cherry"}
b = {"orange","apple","cherry"}
c = a.difference(b)
print(c)  # prints {'banana'}
```

### ۲.۱ حلقه for و مجموعه

می‌توان عناصر یک مجموعه را با استفاده از حلقه for خواند.

```
a = {"banana","apple","cherry"}
for x in a:
    print(x)
```

## ۲ دیکشنری

یک دیکشنری (یا فرهنگ داده) نوع خاصی از مجموعه است. هر عنصر یک دیکشنری بصورت یک زوج کلید و مقدار است که با کاراکتر دو نقطه : از هم جدا شده‌اند. قطعه کد زیر یک دیکشنری به نام weekdays تعریف می‌کند.

```
weekdays = {"Mon":1, "Tue":2, "Wed":3, "Thur":4, "Fri":5, "Sat":6, "Sun":7}
```

در این مثال Mon, Tue, Wed, Thur, Fri, Sat, Sun کلیدهای دیکشنری weekdays هستند و اعداد 1 تا 7 مقادیر مربوط به این کلیدها هستند. می‌توان با ذکر کلید خاصی مقدار متناظر با آن کلید را بدست آورد.

```
weekdays["Wed"] = 3
weekdays["Fri"] = 5
```

کلیدهای یک دیکشنری باید منحصر بفرد باشند (هیچ کلیدی تکرار نمی‌شود) اما مقادیر می‌توانند تکرار شوند و لزوما منحصر بفرد نیستند.

```
nature = {"cat": "animal", "dog":"animal", "wheat":"plant", "grass":"plant"}
```

کد زیر یک دیکشنری تهی به نام thisdict را تعریف می‌کند.

```
thisdict = {}
```

اگر بخواهیم یک زوج کلید و مقدار را به یک دیکشنری اضافه کنیم می‌توانیم این کار را بصورت زیر انجام دهیم.

```
thisdict["banana"] = 345
print(thisdict) # prints {"banana":345}
thisdict["sandwich"] = 56
print(thisdict) # prints {"banana":345, "sandwich":56}
```

اگر کلیدی که می‌خواهیم اضافه کنیم قبلا در دیکشنری موجود باشد، مقدار متناظر با آن آپدیت می‌شود.

```
thisdict["banana"] = 12
print(thisdict) # prints {"banana":12, "sandwich":56}
```

می‌توان با استفاده از متد pop یک کلید را از دیکشنری حذف کرد.

```
thisdict.pop("banana")
print(thisdict) # prints {"sandwich":56}
```

## ۱.۲ لیست کلیدها، مقادیر و زوج کلید_مقدار

با استفاده از متد keys می‌تواند لیست کلیدهای یک دیکشنری را بدست آورد. البته باید دقت کرد که حاصل متد keys مانند یک لیست معمولی نیست و نمی‌توان با دادن اندیس دلخواه عناصر آن را بدست آورد.

```
nature = {"cat": "animal", "dog":"animal", "wheat":"plant", "grass":"plant"}
k = nature.keys()
print(k)  # prints dict_keys(['cat', 'dog', 'wheat', 'grass'])
print(k[0])  # Error!!
```

با استفاده از متد values می‌توان لیست مقادیر یک دیکشنری را بدست آورد. همانند متد keys حاصل values یک لیست معمولی نیست و نمی‌توان با دادن اندیس دلخواه عناصر آن را بدست آورد.

```
nature = {"cat": "animal", "dog":"animal", "wheat":"plant", "grass":"plant"}
k = nature.values()
print(k)  # prints dict_values(['animal', 'animal', 'plant', 'plant'])
```

با استفاده از متد items می‌توان لیست زوج کلید:مقدار را بدست آورد.

```
nature = {"cat": "animal", "dog":"animal", "wheat":"plant", "grass":"plant"}
k = nature.items()
print(k)  # prints dict_items([('cat', 'animal'), ('dog', 'animal'), ('wheat', 'plan
```

## ۲.۲ حلقه for و دیکشنری

با استفاده از حلقه for می‌توان کلیدهای داخل یک دیکشنری را خواند. در قطعه کد زیر x هر بار برابر با یکی از کلیدهای دیکشنری nature است. nature[x] مقدار متناظر با کلید x است.

```
for x in nature:
    print(nature[x])
```

کد زیر همان خروجی کد بالا را تولید می‌کند.

```
for x in nature.values():
    print(x)
```

اگر بخواهیم حلقه for روی زوج کلید و مقدار حرکت کند:

```
for x,y in nature.items():
    print(x,y)
```

# ۳ چند مسئله کاربردی

در این قسمت چند مسئله را با استفاده از ساختارداده‌های مجموعه و دیکشنری حل می‌کنیم.

## ۱.۳ ترجمه کلمه به کلمه

دیکشنریها همانطور که از اسمشان برمی‌آید می‌توانند برای نگهداری کلمات و معانی متناظر با آنها استفاده شوند. برای مثال در دیکشنری EF برای بعضی از کلمات انگلیسی معادل فرانسوی آنها ذکر شده است.

```
EF = {'I':'je', 'he':'il', 'had':'eu', 'a':'une', 'an':'un', 'car':'voiture'}
```

اگر بخواهیم از این دیکشنری برای ترجمه از انگلیسی به فرانسه استفاده کنیم می‌توانیم یک جمله را بصورت زیر ترجمه کنیم.

```
EF = {'I':'je', 'he':'il', 'had':'eu', 'a':'une', 'an':'un', 'car':'voiture'}
english = "I had a car"
words = sentence.split()
french =  "I had a car"
for x in words:
    french = french  + EF[x] +  " "

print(french)  # prints    je eu une voiture
```

## ۲.۳ نام کاربری و کلمه عبور

از آنجا که کلیدهای دیشکنری متمایز هستند، از قالب دیکشنری می‌توان برای ذخیره‌سازی مشخصات کاربران شامل زوج نام کاربری و کلمه عبور استفاده کرد.

```
users = {'ali24':'ksdhfie23A', 'hamid23':'kwerh432', 'reza77':'1234394'}
```

با ذکر نام کاربری می‌توان براحتی کلمه عبور متناظر با آن را پیدا کرد.

```
print(users['ali24'])   # prints 'ksdhfie23A'
```

## ۳.۳ تشخیص وجود مثلث در یک رابطه

یک رابطه مجموعه‌ای از زوجهاست. برای مثال مجموعه $A$ یک رابطه است.

$$A = \{(0,1),(1,0),(2,3),(3,2),(5,4),(4,5),(7,5),(5,7),(4,7),(7,4)\}$$

می‌خواهیم بدانیم آیا یک مثلث در رابطه داده شده وجود دارد یا نه. یک مثلث متشکل از سه عنصر متمایز $x$ و $y$ و $z$ است بطوریکه زوجهای $(x,y)$ و $(x,z)$ و $(y,z)$ در رابطه وجود داشته باشند. فرض ما بر این است که رابطه داده شده متقارن است. به عبارت دیگر اگر $(x,y)$ در رابطه باشد آنگاه $(y,x)$ هم در رابطه است. در مثال بالا $A$ یک رابطه متقارن است. همچنین فرض می‌کنیم هیچ عنصری با خودش در رابطه نیست.

فرض کنید رابطه $A$ بصورت لیستی از تاپلها داده شده است.

```
A = [(0,1), (1,0), (2,3), (3,2), (5,4), (4,5), (7,5), (5,7), (4,7), (7,4)]
```

همه عناصری که با $x$ در رابطه هستند را همسایه‌های $x$ می‌نامیم. با این تعریف عناصر $x$ و $y$ در یک مثلث هستند اگر $x$ و $y$ حداقل یک همسایه مشترک داشته باشند.

در قدم اول، برای هر عنصر مجموعه همسایه‌هایش را بدست می‌آوریم. لیست $N$ را تعریف می‌کنیم بطوریکه $N[i]$ مجموعه همسایه‌های $i$ باشد. در ابتدا مجموعه همسایه‌های هر عنصر تهی است. فرض می‌کنیم عناصر اعداد $0$ تا $n-1$ هستند.

```
N = [set()] * n
```

کد بالا لیستی از مجموعه‌های تهی را ایجاد می‌کند. چون $n$ تعداد عناصر است، در اینجا $n$ مجموعه تهی داریم. $N[i]$ متناظر با مجموعه همسایه‌های $i$ است.

حال مجموعه همسایه‌ها را محاسبه می‌کنیم.

```
for x in A:
   N[x[0]].add(x[1])
   N[x[1]].add(x[0])
```

بعد از اجرای کد بالا، برای مثال داده شده خواهیم داشت:

```
N[0] = {1}
N[1] = {0}
N[2] = {3}
N[3] = {2}
N[4] = {5,7}
N[5] = {4,7}
N[6]= set()
N[7]= {5,4}
```

حال براحتی می‌توانیم چک کنیم که مثلثی در رابطه بالا وجود دارد و یا نه.

```
for x in A:
   s = N[x[0]].intersection(N[x[1]])
   if (len(s) > 0):
      print("Found a triangle!")
```