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

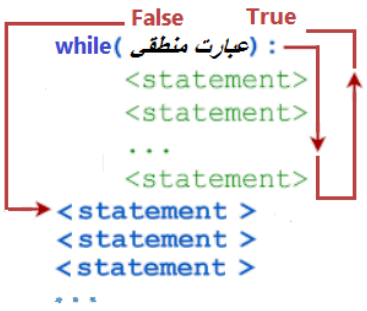
خلاصه هفته ششم

مفاهیم:
حلقه while ـ معرفی لیست list در پایتون

در این هفته با دستور while و بطور کلی با ساختار حلقه و تکرار در پایتون آشنا می شویم. در ادامه ساختار داده مهم و پرکاربرد لیست list را معرفی می‌کنیم.

۱ حلقه while

یکی از ابزارهای اصلی در برنامه‌نویسی ساختار حلقه است. از حلقه loop برای تکرار دستورالعملها استفاده می‌شود. شکل زیر ساختار حلقه while را نشان می‌دهد. به مجموعه دستورات زیر while که از اول خط فاصله دارند (دستورات سبزرنگ) بدنه حقله گفته می‌شود. به عبارت منطقی اصطلاحا شرط حلقه گفته می‌شود.
این ساختار تا حد زیادی مشابه ساختار دستور if است با این تفاوت که اجرای دستورات بدنه حلقه تا زمانیکه عبارت منطقی (شرط حلقه) ارزش True داشته باشد تکرار می‌شود. در دستور if فقط یک بار این دستورات اجرا می‌شود. بعد از هر تکرار عبارت منطقی چک می‌شود. اگر ارزش False داشته باشد، کنترل به اولین دستور بعد از بدنه حلقه منتقل می‌شود.



طبیعتا شرط حلقه یک عبارت ثابت نیست و حاوی متغیرهایی است که در بدنه حلقه تغییر می‌یابند. به نمونه زیر دقت کنید. با اجرای این قطعه کد، دستور print("hello") دوباره و دوباره اجرا می‌شود و هیچ گاه اجرای به دستور print("Goodbye") نمی‌رسد. به این یک حلقه بینهایت گفته می‌شود. برنامه نویس باید از حلقه‌های بی نهایت در برنامه اجتناب کند مگر اینکه در داخل حلقه مکانیزمی برای خروج در نظر گرفته شده باشد.

```
while (1==1):
   print("hello")
print("Goodbye")
```

مثال: برنامه‌ای بنویسید که به تعداد 100 تا رشته hello را چاپ کند.

```
i = 1
while (i <= 100):
      print("hello")
      i = i+1
```

معمولا حلقه‌ها یک متغیر دارند که تعداد تکرار را کنترل می‌کند. اصطلاحا به این متغیر شمارنده حلقه گفته می‌شود. در مثال بالا متغیر i شمارنده حلقه است. مقدار i از 1 شروع شده و هر بار یکی به آن اضافه می‌شود تا اینکه به حد 100 برسد. زمانی که i به 101 رسید عبارت منطقی داخل پرانتز ارزش False خواهد داشت و اجرای حلقه خاتمه می‌یابد.

مثال: برنامه‌ای بنویسید که اعداد 1 تا 100 زیر هم چاپ کند.

```
i = 1
while (i <= 100):
      print(i)
      i = i+1
```

اگر بخواهیم اعداد با فاصله از هم در یک خط چاپ شوند می‌توانیم پارامتر end در دستور print را خودمان مشخص کنیم تا با هر بار اجرای دستور print به خط بعدی نرود.

```
i = 1
while (i <= 100):
      print(i, end=" ")
      i = i+1
```

مثال: برنامه‌ای بنویسید که اعداد از 100 تا 1 را بصورت نزولی زیر هم چاپ کند.

```
i = 100
while (i > 0):
      print(i)
      i = i-1
```

مثال: برنامه‌ای بنویسید که اعداد فرد بین 1 تا 100 را زیر هم چاپ کند.

```
i = 1
while (i <= 100):
      print(i)
      i = i + 2
```

این برنامه را می‌توان به طریق دیگری هم پیاده‌سازی کرد.

```
i = 1
while (i <= 100):
      if (i % 2 == 1):
           print(i)
      i = i + 1
```

مثال: برنامه‌ای بنویسید که الگوی زیر را چاپ کند.

```
1:100 - 2:99 - 3:98 - 4:97 - ... - 50:51 -
```

```
i = 1
while (i <= 50):
      print(i,101-i, sep=":", end=" - ")
      i = i + 1
```

مثال: برنامه‌ای بنویسید که مجموع زیر را محاسبه کند و مقدار آن را چاپ کند.

$$\sum_{i=1}^{100} i^2$$

```
i = 1
sum = 0
while (i <= 100):
      sum = sum + i*i
      i = i + 1
print("sum=",sum)
```

مثال: برنامه‌ای بنویسید که مجموع زیر را محاسبه کند و مقدار آن را چاپ کند.

$$\sum_{i=1}^{100} \frac{1}{i}$$

```
i = 1
sum = 0
while (i <= 100):
      sum = sum + 1/i
      i = i + 1
print("sum=",sum)
```

مثال: برنامه‌ای بنویسید که مقدار $n!$ را محاسبه کند.

$$n! = 1 \times 2 \times 3 \times 4 \times \ldots \times n-1 \times n$$

```
i = 1
f = 1
while (i <= n):
      f = f * i
      i = i + 1
print("factorial=",f)
```

مثال: برنامه‌ای بنویسید که مجموع زیر را محاسبه کرده و مقدار آن را چاپ کند.

$$\sum_{i=1}^{10} i!$$

در برنامه قبلی در هر مرحله مقدار f برابر با $i!$ است. پس کافی است مقدار f را به sum اضافه کنیم.

```
i = 1
f = 1
while (i <= n):
      f = f * i
      i = i + 1
      sum = sum + f
print("sum=",sum)
```

مثال: برنامه‌ای بنویسید که عددی صحیح را از کاربر گرفته و تشخیص دهد که عدد داده شده اول است یا مرکب.

برای تشخیص اول بودن n، عدد n را بر همه اعداد 2 تا $n-1$ تقسیم می‌کنیم. اگر در یکی از تقسیم ها مقدار باقیمانده صفر شد، این بدین معنی است که عدد n مرکب است چون یک مقسوم علیه صحیح غیر از خودش و 1 دارد. در غیر این صورت اگر هیچ تقسیمی باقیمانده صفر تولید نکرد پس با اطمینان می‌توانیم بگوییم که n اول است.

در حقیقت کافی است تنها اعداد 1 تا $\sqrt{n}$ را برای مقسوم علیه امتحان کنیم چون وقتی $n = k \times m$ آنگاه حتما k یا m کمتر از یا مساوی با $\sqrt{n}$ خواهد بود.

```
flag = True
n = int (input ("Please enter a positive integer: "))
i = 2
while ( i <= n ** 0.5):
    r = n % i
    if ( r == 0):
        print(n," is a composite number.")
        flag = False
        break
    i = i + 1

if (flag):
   print(n," is a prime number.")
```

چند نکته در مورد برنامه بالا:

- به محض اینکه باقیمانده تقسیمی صفر شد می‌دانیم که n اول نیست. در این حالت ادامه حلقه غیر ضروری است. در اینجا چاپ می‌شود که n عددی مرکب است. برای خروج اضطراری از حقله از دستور break استفاده کرده‌ایم. دستور break اجرای بدنه حلقه را قطع کرده و کنترل برنامه به اولین دستور بعد از بدنه حلقه می‌سپارد.

برای خروج اضطراری از حلقه از دستور break استفاده می‌کنیم

- وقتی n اول است هیچگاه دستور break اجرا نمی‌شود و حلقه بصورت طبیعی به پایان می‌رسد. بعد از اتمام حلقه چگونه بدانیم که حلقه بصورت طبیعی تمام شده است یا با دستور break ؟ برای این منظور از متغیر اضافی flag استفاده کرده‌ایم. در آغاز کار flag برابر با True است. این بدین معنی است که بطور پیش فرض پیش بینی می‌کنیم که n اول باشد. هر جا که مرکب بودن n اثبات شد، متغیر flag را False می‌کنیم تا بعد از اجرای حلقه بدانیم که n مرکب است و دستور break اجرا شده است.

- در انتهای برنامه مقدار flag را چک می‌کنیم. اگر مقدار flag برابر با True باشد، یعنی اینکه عدد داده شده اول است در غیر این صورت قبلا پیام مناسب دال بر مرکب بودن n چاپ شده است.

۱.۱ حلقه تودرتو

مثال: برنامه‌ای بنویسید که خروجی زیر را چاپ کند. m سطر ستاره و هر سطر شامل n ستاره باشد.

```
******* ... ****
******* ... ****
******* ... ****
.
.
.
******* ... ****
******* ... ****
```

قطعه کد زیر یک سطر ستاره، متشکل از n ستاره چاپ می‌کند.

```
i = 1
while ( i <= n):
    print("*", sep = "",end="")
    i = i + 1
```

می‌خواهیم کد بالا m بار اجرا شود. پس تمام کد بالا در درون یک حلقه دیگر می‌گذاریم.

```
j = 1
while(j <= m):

   i = 1
   while (i <= n):
      print("*", sep = "",end="")
      i = i + 1

   print("")
   j = j + 1
```

به نمونه کد بالا یک حلقه تو در تو گفته می‌شود.
دقت کنید بعد از حلقه وسطی یک دستور print خالی گذاشته‌ایم تا برای چاپ سطر جدید به اول خط بعدی برود.
برنامه بالا را می‌توانیم با استفاده از ابزارهای پایتون تنها با یک حلقه بنویسیم. دستور print("*" * n) به تعداد n ستاره در یک سطر چاپ می‌کند.

```
j = 1
while(j <= m):

   print("*" * n)

   j = j + 1
```

۲ لیست در پایتون

فرض کنید می‌خواهیم برنامه‌ای بنویسیم که در آن کاربر n نمره را وارد کند و در انتها میانگین نمرات چاپ شود. این کار با استفاده از حلقه while به آسانی قابل پیاده‌سازی است.

```
n = 20
i = 1
sum = 0
while (i <= n):
    grade = float(input("please enter the next grade: "))
    sum = sum + grade
    i = i + 1
avg = sum / n
print("average =",avg)
```

حال فرض کنید بخواهیم همه نمراتی که بالای میانگین هستند را چاپ کنیم. چگونه این کار را انجام دهیم؟ اگر از کد بالا استفاده کنیم یک مشکل این است که زمانی که میانگین محاسبه می‌شود، نمرات وارد شده از دست رفته است. متغیر grade تنها ظرفیت نگهداری یک نمره را دارد. از طرفی دیگر زمانی نمرات وارد می‌شود هنوز میانگین محاسبه نشده است تا مقادیر آنها را با میانگین مقایسه کنیم.

برای حل این مشکل نیاز داریم که نمرات در جایی ذخیره شوند تا بعدا با میانگین مقایسه شوند. برای ذخیره سازی مجموعه‌ای از اعداد (و بطور کلی اشیا) می‌توان از ساختار داده لیست list استفاده کرد. در اینجا به معرفی این ساختار می‌پردازیم.

۱.۲ معرفی لیست

به بیان ساده لیست در پایتون دنباله‌ای از عناصر است. (ساختار لیست مشابه ساختار آرایه در زبانهای برنامه‌نویسی دیگر است.) لیست در پایتون با استفاده از دو کاراتر [ ] تعریف می‌شود. برای مثال در کد زیر، a یک لیست است که حاوی چند اسم می‌باشد.

```
a = ["ali","mohsen","reza","zahra","maryam"]
```

ویژگیهای مهم لیست:

- یک لیست خالی بصورت زیر تعریف می‌شود.

```
a = [ ]
```

- عناصر لیست ترتیب دارند. دو لیست زیر علارغم اینکه حاوی رشته‌های یکسان هستند اما چون ترتیب رشته‌ها فرق می‌کند برابر نیستند.

```
a = ["ali","mohsen","reza","zahra","maryam"]
b = ["ali","reza", "mohsen", "zahra","maryam"]
a == b   # False
```

- لیست می‌تواند حاوی عناصری با نوعهای مختلف باشد.

```
a = ["ali","mohsen", 1 ,"zahra", 3.4]
```

- عناصر لیست لزوما منحصر بفرد نیست. تکرار در عناصر لیست مجاز است.

```
a = ["ali","mohsen", "ali" ,"zahra", 3.4, 3.4]
```

- هر عنصر لیست یک شماره (اندیس) دارد که محل واقع شدنش درلیست را نشان می‌دهد. اندیسها از صفر شروع می‌شوند. پس عنصر اول لیست در اندیس صفر واقع شده است. برای دسترسی به عنصر واقع در اندیس i از $a[i]$ استفاده می‌شود.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> a[0]
'ali'

>>> a[2]
'reza'
```

برای لیست اندیس منفی هم تعریف شده است. $a[-1]$ عنصر آخر لیست است. $a[-2]$ عنصر قبل از آخر لیست است. به همین ترتیب $a[-i]$ عنصر i ام از آخر لیست است.

- یک قطعه از لیست را می‌توان به صورت زیر بدست آورد. اگر a یک لیست باشد، $a[i,j]$ خود یک لیست که حاوی عناصر از اندیس i تا قبل از اندیس j است. دقت کنید که شامل اندیس jام نمی‌شود.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> a[2:4]
["reza","zahra"]
```

$a[:4]$ لیستی است که از ابتدای لیست a شروع می‌شود تا قبل از اندیس چهارم.

$a[2:]$ لیستی است که از اندیس دوم a شروع می‌شود تا انتهای لیست a (شامل آخرین عنصر لیست هم می‌شود.)

- برای عملگر عضویت در لیست از in و not in استفاده می‌شود.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> "ali" in a
True

>>> "ahmad" in a
False

>>> "ahmad" not in a
True
```

- عناصر لیست قابل تغییر هستند.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> a[0] =  "ahmad"

>>> a
["ahmad","mohsen","reza","zahra","maryam"]
```

- طول لیست را می‌توان با دستور len استخراج کرد.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> len(a)
5

>>> b = [ ]

>>> len(b)
0
```

- در صورتی که به اندیسی که در لیست وجود ندارد دسترسی شود پیام خطا چاپ می‌شود.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> a[6]

Error!
```

- از علمگر + برای چسپاندن دو لیست استفاده می‌شود.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> b = [ 2, 3]

>>> a + b
["ali","mohsen","reza","zahra","maryam", 2, 3]
```

از این عملگر می‌توان برای اضافه کردن یک عنصر به انتهای لیست استفاده کرد.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> a + ["fatemeh"]

>>> a
["ali","mohsen","reza","zahra","maryam","fatemeh"]
```

- برای اضافه کردن یک عنصر به انتهای لیست می‌توان از متد append استفاده کرد.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> a.append("fatemeh")

>>> a
["ali","mohsen","reza","zahra","maryam","fatemeh"]
```

- برای حذف یک عنصر از لیست از دستور del می‌توان استفاده کرد.

```
>>> a = ["ali","mohsen","reza","zahra","maryam"]

>>> del a[1]

>>> a
["ali","reza","zahra","maryam"]
```

لیست امکانات، ویژگیهای و متدهای فراوانی دارد که می‌توان با مراجعه به مراجع برنامه‌نویسی پایتون با آنها آشنا شد. در دروس آتی با ویژگیها و امکانات دیگری از لیست آشنا می‌شویم.

در هفته بعدی خواهیم دید که چگونه از لیست می‌توان برای حل مسئله چاپ نمرات بالاتر از میانگین استفاده کرد.