لیست و رشته

به علت شباهت ساختاری لیست و رشته بسیاری از دستورات و عملگرهای لیست روی رشته هم کار می‌کنند.

پلاک منفی لیست

اگر پلاک منفی به یک لیست بدهیم، از آخر **لیست** شروع به شمارش می‌کند. یعنی a[-1] خانه آخر لیست و a[-2] خانه یکی مانده به آخر لیست است. مانند:

```
a = [0, 10, 20, 30, 40, 50]
i = a[-1]        # 50
j = a[-2]        # 40
```

نکته: در رشته هم قابل استفاده است. مانند:

```
st = "012345"
ch = st[-1]      # "5"
```

پایان : آغاز لیست

از یک لیست جدید ایجاد می‌کند و بخشی از **لیست** را در آن کپی می‌کند.

مانند range تا یک خانه قبل از عدد **پایان** در نظر گرفته می‌شود. مانند:

```
a = [0, 10, 20, 30, 40, 50]
b = a[1:5]       # [10, 20, 30, 40]
c = a[0:len(a)]  # [0, 10, 20, 30, 40, 50]
```

باز هم مانند range می‌توانیم **آغاز** را ننویسیم، که در این حالت آغاز صفر نظر گرفته می‌شود. مانند:

```
b = a[:5]        # [0, 10, 20, 30, 40]
```

می‌توانیم **پایان** را ننویسیم، که در این حالت len یا انتهای لیست در نظر گرفته می‌شود. مانند:

```
b = a[1:]        # [10, 20, 30, 40, 50]
```

نکته: استفاده از اعداد منفی برای آغاز و پایان درست است. مانند:

```
b = a[:-2]       # [0, 10, 20, 30]
b = a[2:-2]      # [20, 30]
b = a[-4:-2]     # [20, 30]
```

نکته: برای کپی گرفتن از کل یک لیست می‌توان از این دستور استفاده کرد:

```
b = a[:]         # [0, 10, 20, 30, 40, 50]
```

نکته: در رشته هم قابل استفاده است. مانند:

```
string1 = "012345"
string2 = a[1:5] # "1234"
```

لیست.reverse()

لیست را معکوس می‌کند. مانند:

```
a = [0, 10, 20, 30, 40, 50]
a.reverse()      # [50, 40, 30, 20, 10, 0]
```

نکته: دقت کنید که این یک دستور است و هیچ خروجی ندارد. فقط لیست را معکوس می‌کند.

بنابراین نمی‌توان نوشت:

```
❌ b = a.reverse()
❌ print(a.reverse())
```

نکته: در رشته هم قابل استفاده نیست!

```
❌ string1.reverse()
```

مقدار in لیست

برای بررسی این‌که **مقدار** در یک **لیست** هست یا نه از in یا not in استفاده می‌شود. مانند:

```
a = [0, 10, 20, 30, 40]
b = int(input())
if b in a:
     print ("hast!")
if b not in a:
     print ("nist!")
```

نکته: در رشته هم قابل استفاده است. مانند:

```
q = input()
if "helli" in q:
     print("OK!")
```

لیست.index(مقدار)

موقعیت **مقدار** را در **لیست** اعلام می‌کند.

نکته: اگر بیش از یک بار **مقدار** در **لیست** وجود داشته باشد، موقعیت اولی را بازمی‌گرداند. مانند:

```
a = [20, 40, 60, 20, 40, 60]
b = a.index(40)       # 1
```

نکته: اگر **مقدار** در **لیست** وجود نداشته باشد، برنامه با پیام خطا متوقف می‌شود. بنابراین بیشتر اوقات پیش از استفاده از این دستور باید از in استفاده کرد.

نکته: در رشته هم قابل استفاده است. مانند:

```
hex = "0123456789abcdef"
q = input()
b = a.index(s)
if s in hex:
     print(hex.index(q))
```

for متغیر in لیست:

یک روش سریع‌تر و بهتر برای دسترسی به تمام اعضای **لیست**. در برنامه زیر نتیجه هر دو حلقه یکی است:

```
a = [20, 40, 60, 20, 40, 60]
for x in a:
     print(x)
for i in range(len(a)):
     print(a[i])
```

نکته: در مواردی که با شماره اعضاء لیست هم کار داریم، بهتر است از همان روش قدیمی range استفاده کنیم.

نکته: در رشته هم قابل استفاده است و هر دفعه یک حرف از رشته را بازمی‌گرداند. مانند:

```
for ch in "abc":
     print(ch)
```

خروجی:

```
a
b
c
```

رشته.split(جداکننده)

رشته را به لیستی از رشته‌های کوچک‌تر تقسیم می‌کند. مانند:

```
a = "yek-do-se"
b = a.split("-")  # ["yek", "do", "se"]
```

می‌توانیم جداکننده را مشخص نکنیم. در این حالت فاصله (" ")، tab ("\t") و enter ("\n") جداکننده خواهند بود. مانند:

```
a = "m\tcm mm\nKg     gr"
b = a.split()     # ['m', 'cm', 'mm', 'Kg', 'gr']
```

نوع داده

type(x)

این تابع نوع یک مقدار یا متغیر را مشخص می‌کند. بیشترین کاربرد این دستور در مقایسه است. مانند:

```
If type(a) == type(b):
      print("OK!")
```

برای این‌که بررسی کنیم یک متغیر از نوع لیست است یا خیر دو راه وجود دارد. اول این‌که نوع آن را با نوع یک لیست مقایسه کنیم. مانند:

```
if type(a) == type([]):
      print("is list")
```

راه دوم که راه بهتری است این است که نوع آن را با کلمه کلیدی list مقایسه کنیم:

```
if type(a) == list:
      print("is list")
```

در راه دوم برای عدد صحیح از کلمه کلیدی int و برای رشته از کلمه کلیدی str استفاده می‌شود.

کتابخانه time

time.time()

این یک عدد اعشاری بازمی‌گرداند که نشانگر زمان برحسب ثانیه است. مبداء این زمان در ویندوز و لینوکس متفاوت است. ولی خیلی برای ما مهم نیست، چون استفاده ما از این تابع محاسبه زمان سپری شده برای اجرای یک قطعه کد است. به این صورت که زمان پیش و پس از اجرای قطعه کد مورد نظر را از هم کم می‌کنیم تا فاصله این دو زمان مشخص شوند. نمونه:

```
import time
t0 = time.time()
f = 1
for i in range(10000000):
      f=f*i
t1 = time.time()
print(t1 - t0)
```

time.sleep(زمان)

این دستور برنامه را برای مدت **زمان** مشخصی بیکار نگه می‌دارد و سپس به اجرای برنامه را ادامه خواهد داد.

زمان بر حسب ثانیه است.

مثال:

```
import time
print(time.time())
time.sleep(3)
print(time.time())
```

رخداد در turtle

```
turtle.mainloop()
```

هر برنامه‌ای برای این‌که تمام نشود نیاز به یک حلقه بی‌پایان دارد. در این حلقه برنامه مدام فعالیت‌های کاربر، شبکه، ساعت و ... را بررسی می‌کند و هر وقت لازم شد تابع‌هایی را فراخوانی خواهد کرد.

این حلقه، حلقه اصلی یا **mainloop** و فعالیت‌های یاد شده **رخداد** نامیده می‌شوند.

برخی کتابخانه‌ها و ابزارها مانند turtle برای ساده کردن کار برنامه نویس حلقه اصلی را در یک تابع می‌نویسند و فقط کافی است آن را فراخوانی کنید. البته قبل از فراخوانی حلقه لازم است با دستوراتی به حلقه بگوییم که در هر رخداد کدام تابع را فراخوانی کند.

بدیهی است که اگر فقط رخدادها را تنظیم کنیم ولی حلقه اصلی را فراخوانی نکنیم، نه تنها برنامه هیچ کاری نخواهد کرد، بلکه پایان خواهد یافت.

```
turtle.onscreenclick(تابعِ رخداد)
```

به turtle.mainloop اعلام می‌کند در صورتی کاربر روی صفحه turtle کلیک کرد، کدام تابع فراخوانی شود.

نکته: کلیک رخدادی است که دو پارامتر عرض و ارتفاع نقطه کلیک شده را به همراه دارد. بنابراین **تابعِ رخداد** باید شامل دو پارامتر شود. مانند:

```
import turtle
turtle.onscreenclick(turtle.goto)
turtle.mainloop()
```

نکته: نیازی نیست که **تابعِ رخداد** از تابع‌های turtle باشد. می‌توانیم خودمان تابعی برای این کار بنویسیم. مانند:

```
import turtle
def circle10(x, y):
      turtle.pu()
      turtle.goto(x, y)
      turtle.pd()
      turtle.circle(10)
turtle.onscreenclick(circle10)
turtle.mainloop()
```

نکته: برای از کار افتادن رخداد کلیک می‌توان از همین دستور استفاده کرد. فقط باید **تابعِ رخداد** را None قرار داد. مانند:

```
turtle.onscreenclick(None)
```

turtle.onkeypress(کلید, تابعِ رخداد)

به turtle.mainloop اعلام می‌کند در صورتی کاربر یکی از کلیدهای صفحه کلید را فشرد، کدام تابع فراخوانی شود.

می‌توان برای هر **کلید**، یک **تابعِ رخداد** تعریف کرد. کلید یک رشته است مانند "a"، یا "1" یا "&".

نکته: کلید به حروف کوچک و بزرگ حساس است. بنابراین دکمه "c" با "C" متفاوت است. یکی بدون Shift یا Caps Lock است و دیگری برعکس. همچنین برای تعریف برخی کلیدها باید نام آن‌ها را نوشت. مانند "Up", "Down", "Left", "Right" برای کلیدهای جهت نما، "F1" تا"F12"، "Escape"، "Space"، "BackSpace" و نیز "Return" برای دکمهٔ Enter.

نکته: پس از تعریف کلیدها با دستور turtle.listen() باید turtle را گوش به زنگ فشردن کلیدها نمود.

مثال:

```
import turtle
turtle.onkeypress(turtle.bye, "Escape")
turtle.onkeypress(turtle.clear, "c")
turtle.listen()
turtle.mainloop()
```

turtle.ontimer(تابعِ رخداد, زمان)

به turtle.mainloop اعلام می‌کند که مدتی بعد **تابعِ رخداد** را فراخوانی کند.

زمان بر حسب میلی ثانیه (هزارم ثانیه) است. و می‌تواند حذف شود که در این صورت تابع مذکور بلافاصله اجرا خواهد شد.

نکته: برای تکرار شدن یک کد می‌توان آن را درون **تابعِ رخداد** قرار داد. و در انتهای **تابعِ رخداد** نیز، دوباره دستور ontimer را فراخوانی نمود. مانند:

```
import turtle
def repeat():
      turtle.fd(100)
      turtle.ontimer(repeat, 1000)
turtle.ontimer(repeat)
turtle.mainloop()
```

فایل

```
file1 = open(آدرس فایل, نوع دسترسی)
```

آدرس فایل یک رشته است. مثل "/home/user/phone.txt" در لینوکس یا "C:/folder/phone.txt" در ویندوز. اگر فقط نام فایل را بنویسیم مانند "phone.txt" مسیر فایل مسیری که خود برنامه در آن قرار دارد در نظر گرفته می‌شود.

نوع دسترسی رشته است و حالات زیادی دارد. ولی مهم‌ترین آن‌ها (که برای آزمون هم مهم است) این موارد است:

- "r" برای خواندن فایل.
- "w" برای نوشتن فایل از ابتدا.
- "a" برای نوشتن در ادامه فایل.

نکته: پیش فرض **نوع دسترسی** خواندنی ("r") است. یعنی می‌شود نوع دسترسی را ننوشت. این دو دستور معادل هم هستند:

```
file1 = open(آدرس فایل)
file1 = open(آدرس فایل, "r")
```

```
file1.close()
```

فایل را می‌بندد و دسترسی برنامه را از فایل قطع می‌کند.

پرسش: چرا از این دستور استفاده می‌کنیم؟

پاسخ: در برخی از انواع دسترسی، دسترسی کاربر و بقیه برنامه‌ها به اطلاعات فایل ناممکن می‌شود. همچنین موقع نوشتن اطلاعات در فایل، گاهی اطلاعات در حافظه میانی قرار می‌گیرد و به فایل منتقل نمی‌شود. با بستن فایل اطلاعات به فایل منتقل شده و دسترسی سایرین به فایل ممکن می‌شود.

نکته: با پایان برنامه تمام فایل‌هایی که باز شده‌اند، به صورت خودکار بسته می‌شوند.

```
file1.read()
```

این دستور کل محتوای یک فایل را می‌خواند و به صورت یک رشته به ما تحویل می‌دهد. مانند:

```
file1 = open("story.txt")
string1 = file1.read()
```

story.txt

```
Salam.
Yeki bood yeki nabood.
...
Payan!
```

نتیجه:

```
string1 = "Salam.\nYeki book yeki nabood.\n...\nPayan!"
```

file1.readlines()

کل محتوای یک فایل را می‌خواند و به صورت یک لیست از رشته‌ها تحویل می‌دهد. مانند:

```
file1 = open("story.txt")
list1 = file1.readlines()
```

نتیجه:

```
list1 = ["Salam.\n", "Yeki book yeki nabood.\n", "...\n", "Payan!"]
```

file1.readline()

هر بار اجرای این دستور فقط یک خط از فایل را می‌خواند و به صورت یک رشته به ما تحویل می‌دهد. مانند:

```
file1 = open("story.txt")
string1 = file1.read()
string2 = file1.read()
```

نتیجه:

```
string1 = "Salam.\n"
string2 = "Yeki book yeki nabood.\n"
```

file1.write(متن)

متن را در فایل می‌نویسید. این دستور مشابه دستور print است با چند تفاوت:

- فقط یک ورودی می‌گیرد. برای نوشتن چند مقدار باید با استفاده از + آنها را تبدیل به یک رشته کرد.
- سر خط جدید نمی‌رود. در صورت نیاز باید از علامت "\n" برای انتهای خط استفاده کرد.
- متن ورودی فقط می‌تواند رشته باشد. برای نوشتن عدد باید با دستور str آن را تبدیل به رشته کرد.

مانند:

```
file1 = open("a.txt", "w")
a = 10*10+10
file1.write("Sal")
file1.write("am.\n")
file1.write("a = " + str(a))
file1.close()
```

a.txt
Salam. a = 110