CS50x Iran

Loops

Week note

پاییز و زمستان ۱۴۰۴

1 فهرست

- بطور کلی، حلقه ها یا Loops، روشی برای انجام کاری برای چندین بار هستند. شما می‌توانید یک کار را بارها و بارها تکرار کنید.
- برای شروع `code cat.py` را در پنجره‌ی terminal تایپ کنید و با کد زیر شروع کنید:

cat.py

```
print("meow")
print("meow")
print("meow")
```

- با تایپ دستور `python cat.py`، شاهده می‌کنید که برنامه سه بار عبارت meow را چاپ می‌کند.
- به عنوان یک برنامه‌نویس، شما می‌خواهید بدانید چگونه می‌توانید کد خود را بهینه‌تر کنید و بهبود ببخشید. برای مثال به جای تکرار یک تکه کد، آن را تنها یک بار بنویسید. تصور کنید که می‌خواهید ۵۰۰ بار عبارت "meow" را چاپ کنید، آیا منطقی است که عبارت `print("meow")` را ۵۰۰ بار تایپ کنید؟
- حلقه ها به شما این امکان را می‌دهند که یک تکه کد را یک بار بنویسید و هر چند بار که می‌خواهید اجرا شود.

- حلقه‌ی while در تمامی زبان های برنامه‌نویسی تقریباً استفاده می‌شود.
- این حلقه، یک بلوک کد را چندین بار تکرار می‌کند.
- می‌توانید کد اسلاید قبل را با استفاده از حلقه while به این صورت بنویسید:

cat.py

```
i = 3
while i != 0:
    print("meow")
```

توجه کنید که با وجود اینکه این کد چندین بار دستور print("meow") را اجرا می‌کند، اما هرگز متوقف نمی‌شود و به طور بی پایان ادامه خواهد داشت. حلقه while در هر بار اجرا شدن می‌پرسد آیا شرط حلقه True است؟ در این حالت، مفسر پایتون می‌پرسد " آیا i برابر صفر نیست؟" و تا زمانی که این شرط برقرار باشد، تکرار حلقه را ادامه خواهد داد.

وقتی در حلقه ای گیر می‌کنید که برای همیشه اجرا می‌شود، می‌توانید Ctrl + C را روی صفحه کلید خود فشار دهید تا از برنامه خارج شوید.

- برای این که این حلقه بی پایان نباشد، می‌توانیم کد خود را به صورت زیر بنویسیم:

```
cat.py

i = 3
while i != 0:
  print("meow")
  i = i - 1
```

در کدی که ویرایش کرده‌ایم، متوجه می‌شویم که در این حالت، برنامه مقدار `i` را با هر بار اجرا شدن حلقه، یک واحد کاهش می‌دهد و به درستی اجرا می‌شود. اصطلاح ایتریشن (iteration) که در برنامه نویسی کاربرد زیادی دارد به معنای یک دور اجرا شدن حلقه می‌باشد. به عبارت دیگر، به هر گردش در حلقه، یک ایتریشن می‌گویند. اولین ایتریشن، ایتریشن "صفرم" است و دومین ایتریشن، ایتریشن "اول" است.

در برنامه نویسی، شمارش از صفر شروع می‌شود و به ترتیب، شماره‌گذاری ایتریشن‌ها از صفر، یک، دو و به همین ترتیب ادامه می‌یابد.

- برای این که کد خود را بهبود ببخشیم می‌توانیم به این صورت نیز آن را بنویسم:

cat.py

```
i = 1
while i <= 3:
    print("meow")
    i = i + 1
```

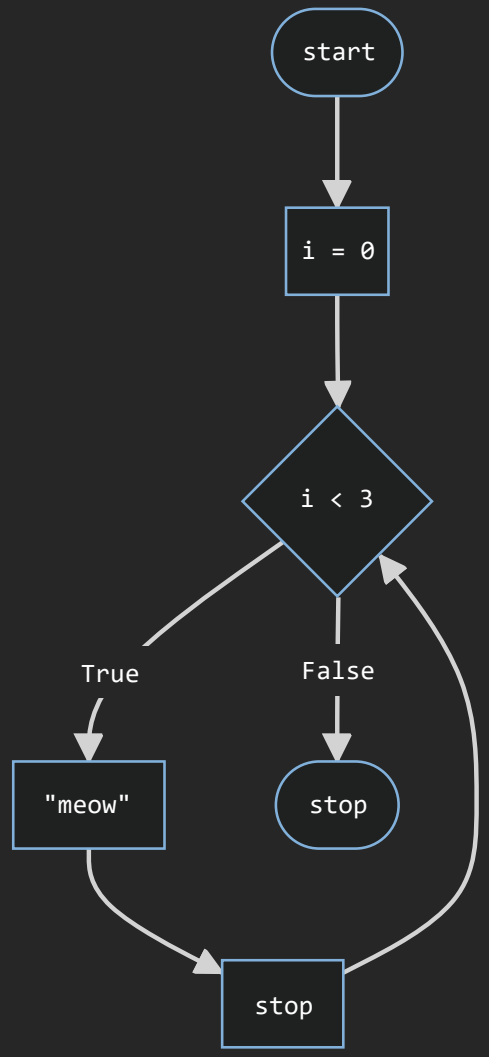
توجه کنید که وقتی کد `i = i + 1` را نوشتیم، مقدار `i` را از راست به چپ نسبت دادیم. در کد بالا، مقدار `i` را با 1 شروع کردیم که شبیه به شمارشی است که برای انسان ها مرسوم است و از یک شروع می‌شود (1, 2, 3, ...). اگر کد بالا را اجرا کنید، سه بار meow را مشاهده خواهید کرد اما در برنامه نویسی، بهتر است با شمارش از 0 شروع کنیم.

- بنابراین کد خود به این صورت مینویسیم تا همان طور که به آن اشاره شد کد ما از 0 شروع شود:

cat.py

```
i = 0
while i < 3:
    print("meow")
    i += 1
```

توجه کنید که با تغییر شرط حلقه به `i < 3` کد ما به درستی عمل می‌کند. با صفر شمارش را شروع کرده‌ایم و این کد سه بار در حلقه ما اجرا می‌شود و سه بار `"meow"` را می‌نویسد. همچنین توجه کنید که `i += 1` همان معنی `i = i + 1` را دارد.



کد ما تا به اینجا، به شکل رو به رو نشان داده شده است:

توجه کنید که حلقه ما i را تا 3 شمرده اما زمانیکه i برابر با 3 می‌شود دیگر شرط حلقه برقرار نیست بنابراین از حلقه خارج می‌شود.

- حلقه‌ی for نوع دیگری از حلقه هاست.
- برای درک بهتر حلقه for، بهتر است با صحبت درباره‌ی نوع متغیر جدیدی که در پایتون به آن لیست (List) گفته می‌شود شروع کنیم. بگذارید مثال هایی در زندگی خودمان بزنیم: لیست خرید، لیست کارهایی که باید انجام دهیم و غیره...
- حلقه for همه آیتم های لیست را ایتریت می‌کند. به عنوان مثال، کد cat.py خود را به شکل زیر ویرایش کنید:

cat.py

```
for i in [0, 1, 2]:
    print("meow")
```

دقت کنید با استفاده از حلقه for چگونه کدی خواناتر نسبت به قبل نوشته شده. در این کد، متغیر i با مقدار 0 آغاز شده و پس از اجرای دستور چاپ "meow"، به متغیر i مقدار 1 داده می‌شود و دستور چاپ دوم "meow" اجرا می‌شود و همین عملیات برای متغیر i با مقدار 2 نیز انجام می‌شود، سپس حلقه به پایان می‌رسد.

- کد قبلی هرچند که کاری را که می‌خواهیم را انجام می‌دهد، اما می‌توانیم آن را بهبود ببخشیم و کد بهتری بنویسیم. در نگاه اول، کد ما بسیار خوب به نظر می‌رسد. اما چه می‌شود اگر بخواهید قطعه کدتان را یک میلیون بار تکرار کنید؟ بهتر است کدی بنویسیم که بتواند با موارد این چنینی نیز کار کند و در عین حال بهینه و تمیز باشد. بنابراین، می‌توانیم کد خود را به شکل زیر بهبود بخشیم:

cat.py

```
for i in range(3):
    print("meow")
```

دقت کنید که مقدار بازگشتی `range(3)` همان معنای قبلی را دارد اما به صورت بهینه تر و مطلوب تری نوشته است و همچنان همان مقدار `[0, 1, 2]` را برمی‌گرداند. این کد نیز سه بار اجرا خواهد شد و عبارت `"meow"` را نیز همان طور که انتظار می‌رود سه بار چاپ خواهد کرد.

- در کد قبلی، توجه کنید که ما هرگز از `i` به صورت صریح در کد خود استفاده نکردیم. به عبارت دیگر، پایتون برای آن که عدد ایتریشن را در جایی ذخیره کند به متغیر `i` نیاز دارد و ما از آن در قسمت دیگری از کد برای هدف دیگری استفاده نخواهیم کرد. اگر چنین متغیری در کد ما ارزش دیگری نداشته باشد، می‌توانیم این متغیر را به عنوان یک آندرلاین (_) نمایش دهیم. بنابراین، شما می‌توانید کد خود را به شکل زیر ویرایش کنید:

cat.py

```
for _ in range(3):
    print("meow")
```

توجه کنید که تغییر دادن `i` به _ تأثیری بر عملکرد برنامه ما ندارد.

- کد ما همچنان هم می‌تواند بهتر شود. پایتون این قابلیت را به شما می‌دهد که کد خود را به شکل منعطف تر و خلاصه تر به این صورت بنویسید:

cat.py
```
print("meow" * 3)
```

- توجه کنید که این کد سه بار meow را چاپ می‌کند، اما خروجی برنامه meowmeowmeow خواهد بود. راه حلی بیابید که چگونه می‌توانید در انتهای هر عبارت meow یک خط جدید ایجاد کنید؟ برای این کار کد خود را به این صورت ویرایش کنید:

cat.py
```
print("meow\n" * 3, end="")
```

در کد بالا می‌توان مشاهده کرد که این کد سه بار "meow" را در سه خط جداگانه چاپ می‌کند.
با اضافه کردن end="" و \n به کد، به مفسر پایتون می‌گوییم که یک خط جدید به انتهای هر "meow" اضافه کند.

- شاید بخواهیم از کاربر ورودی دریافت کنیم. ما می‌توانیم از حلقه ها به عنوان یک روش برای اعتبار سنجی آن چه کاربر وارد کرده استفاده کنیم.
- یک الگوی رایج در پایتون استفاده از حلقه `while` برای اعتبار سنجی ورودی کاربر است.
- به عنوان مثال، بیایید از کاربر، یک عدد بزرگتر یا مساوی با صفر، درخواست کنیم:

cat.py

```
while True:
    n = int(input("What's n? "))
    if n < 0:
        continue
    else:
        break
```

- توجه کنید که دو کلمه کلیدی جدید که با آن آشنا شدیم `break` و `continue` است. `continue` به صراحت به پایتون می‌گوید که به تکرار بعدی حلقه بروید. از سوی دیگر، `break` به پایتون می‌گوید که سریعا از حلقه خارج شود، قبل از اینکه همه تکرارهای خود را به پایان برساند.

- در این حالت، هنگامیکه n کوچکتر از 0 باشد، به تکرار بعدی حلقه ادامه می‌دهیم و در نهایت با "What's n? " از کاربر می‌خواهیم مجددا ورودی را وارد کند. اگر اما n بزرگتر یا مساوی 0 باشد، از حلقه خارج شده و بقیه بخش های برنامه اجرا می‌شود.
- در واقع، کلمه کلیدی continue در این حالت اضافی است. می‌توانیم کد را به صورت زیر خلاصه کنیم:

cat.py

```
while True:
    n = int(input("What's n? "))
    if n > 0:
        break

for _ in range(n):
    print("meow")
```

دقت کنید که این حلقه while تا زمانیکه n بزرگتر از 0 نباشد، همیشه اجرا می‌شود (برای همیشه بصورت بی‌نهایت).
وقتی که n بزرگتر از 0 باشد، حلقه متوقف می‌شود.

- با بهره گیری از مطالبی که پیش تر آموختیم، می‌توانیم از توابع برای بهبود کدمان استفاده کنیم:

cat.py

```
def main():
    number = get_number()
    meow(number)

def get_number():
    while True:
        n = int(input("What's n? "))
        if n > 0:
            break
    return n

def meow(n):
    for _ in range(n):
        print("meow")
```

توجه کنید که نه تنها کد شما را به چندین تابع تقسیم کردیم، بلکه از دستور `return` برای بازگرداندن مقدار `n` به تابع `main` استفاده کردیم.

- مدرسه هاگوارتز را از دنیای معروف هری پاتر در نظر بگیرید. در ترمینال `code hogwarts.py` را تایپ کنید. همچنین کد زیر را بنویسید:

hogwarts.py

```
students = ["Hermoine", "Harry", "Ron"]

print(students[0])
print(students[1])
print(students[2])
```

دقت کنید که یک لیست از دانش آموزان با نام هایشان داریم. سپس دانش آموزی که در موقعیت صفرم لیست قرار دارد، یعنی `"Hermoine"` چاپ می‌شود. سایر دانش آموزان هم در موقعیت اول و دوم لیست چاپ می‌شوند.

- همانطور که قبلاً نشان داده شد، می‌توانیم از یک حلقه استفاده کنیم تا یک لیست را ایتریت کنیم. می‌توانید کد خود را خلاصه تر و به این شکل بنویسید:

hogwarts.py

```
students = ["Hermoine", "Harry", "Ron"]

for student in students:
    print(student)
```

توجه کنید که برای هر دانش آموز در لیست دانش آموزان، نام دانش آموز مورد نظر چاپ می‌شود. شاید بپرسید چرا از (_) همانطور که کمی قبلتر راجع به آن بحث شد استفاده نکردیم. این کار را انجام ندادیم زیرا کلمه `student` به صراحت در کد ما استفاده شده است.

- برای مطالعه مطالب بیشتر در رابطه با لیست ها در پایتون می‌توانید به این لینک مراجعه کنید.

- می‌توانیم از تابع `len` برای بررسی طول لیستی به نام `students` استفاده کنیم.
- تصور کنید که می‌خواهید نام دانش آموز و همچنین موقعیت دانش آموز در لیست را چاپ کنید. برای انجام این کار، می‌توانید کد خود را به شکل زیر ویرایش کنید:

hogwarts.py

```
students = ["Hermoine", "Harry", "Ron"]

for i in range(len(students)):
    print(i + 1, students[i])
```

- توجه کنید که زمانی که این کد اجرا شود، علاوه بر دریافت موقعیت هر دانش آموز در لیست و اضافه کردن یک، با استفاده از `i + 1`، نام هر دانش آموز نیز چاپ می‌شود. تابع `len` به شما این امکان را می‌دهد که به طور پویا طول لیست دانش آموزان را مشاهده کنید، بدون توجه به این که چقدر بزرگ شود.

- برای مطالعه بیشتر در ارتباط با تابع `len` می‌توانید به این لینک مراجعه کنید.

- `dict` ها یا دیکشنری ها یک ساختار داده هستند که به شما اجازه می‌دهند تا داده ها را به صورت `key-value` درج کنید.
- یک `list`، یک لیستی از چندین مقدار است اما یک `dict` یک کلید (key) را با یک مقدار (value) مرتبط می‌کند.
- با توجه به گروه های هاگوارتز، ممکن است دانش آموزان خاصی را در گروه های خاصی قرار دهیم.

Hermione	Harry	Ron	Draco
Gryffindor	Gryffindor	Gryffindor	Slytherin

- ما می‌توانیم `list` ها را به تنهایی به این صورت استفاده کنیم:

hogwarts.py

```
students = ["Hermoine", "Harry", "Ron", "Draco"]
houses = ["Gryffindor", "Gryffindor", "Griffindor", "Slytherin"]
```

- توجه داشته باشید که می‌توانیم خاطر جمع باشیم که همیشه این لیست ها را مرتب نگه داریم. فردی که در جایگاه اول `students` قرار دارد که مربوط به خانه موجود در جایگاه اول لیست `houses` است و ... . با این حال، با بزرگ شدن لیست های ما، این مورد می‌تواند بسیار دست و پا گیر شود!

- ما می‌توانیم کد خود را با استفاده از یک `dict` به صورت زیر بهتر کنیم:

hogwarts.py

```
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
print(students["Hermoine"])
print(students["Harry"])
print(students["Ron"])
print(students["Draco"])
```

- توجه کنید که چگونه از `{}` برای ایجاد یک دیکشنری استفاده می‌کنیم. در حالی که از اعداد برای پیمایش در `list` ها استفاده می‌شود. `dict` ها به ما اجازه می‌دهند تا از کلمات استفاده کنیم.

- کد خود را اجرا کنید و مطمئن شوید که خروجی شما به صورت رو به رو است:

```
$ python hogwarts.py
Gryffindor
Gryffindor
Gryffindor
Slytherin
```

- ما می‌توانیم کد بهتری به این صورت بنویسیم:

hogwarts.py

```
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
for student in students:
    print(student)
```

- توجه داشته باشید که چگونه با اجرای این کد، حلقه for فقط در کلیدها یعنی اسامی دانش آموزان پیمایش می‌کند و در نتیجه لیستی از نام دانش آموزان ایجاد می‌شود. چگونه می‌توانیم هر دوی مقادیر یعنی نام گروه دانش آموزان و کلیدها یعنی اسامی آن ها را چاپ کنیم؟

- کد خود را به صورت زیر تغییر دهید:

hogwarts.py

```
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
for student in students:
    print(student, students[student])
```

- توجه داشته باشید که چگونه `students[student]` به سراغ کلید (key) هر دانش آموز می‌رود و مقدار (value) خانه‌اش را پیدا میکند. کد خود را اجرا کنید و متوجه خواهید شد که چگونه خروجی کمی به هم ریخته است.

- ما می‌توانیم کد خود را به این صورت بهبود ببخشیم تا تمیزتر شود:

توجه کنید که چگونه خروجی ما چگونه با اضافه کردن یک " , " به برنامه مرتب تر شده است.

hogwarts.py

```
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
for student in students:
    print(student, students[student], sep=", ")
```

- اگر `python hogwarts.py` را اجرا کنید، خروجی زیر را مشاهده خواهید کرد:

```
$ python hogwarts.py
Hermoine, Gryffindor
Harry, Gryffindor
Ron, Gryffindor
Draco, Slytherin
```

- اگر اطلاعات بیشتری در مورد دانش آموزان داشته باشیم چه می‌شود؟ چگونه می‌توانیم داده های بیشتری را برای هر دانش آموز اضافه کنیم؟

	name	house	patronus
0	Hermione	Gryffindor	Otter
1	Harry	Gryffindor	Stag
2	Ron	Gryffindor	Jack Russell terrier
3	Draco	Slytherin	

- می‌خواهیم مطابق جدول بالا داده هایی را به عنوان مقدار (value) برای هر دانش آموز (key) اضافه کنیم:

hogwarts.py

```
students = [
    {"name": "Hermoine", "house": "Gryffindor", "patronus": "Otter"},
    {"name": "Harry", "house": "Gryffindor", "patronus": "Stag"},
    {"name": "Ron", "house": "Gryffindor", "patronus": "Jack Russell terrier"},
    {"name": "Draco", "house": "Slytherin", "patronus": None},
]
```

- توجه کنید که چگونه یک لیست از دیکشنری‌ها (dict) ایجاد کردیم. لیستی که students نامیده شده و چهار dict درون خود دارد، یکی برای هر دانش آموز. همچنین، توجه داشته باشید که می‌توانید از None زمانی که مقداری نمی‌خواهید بدهید استفاده کنید.

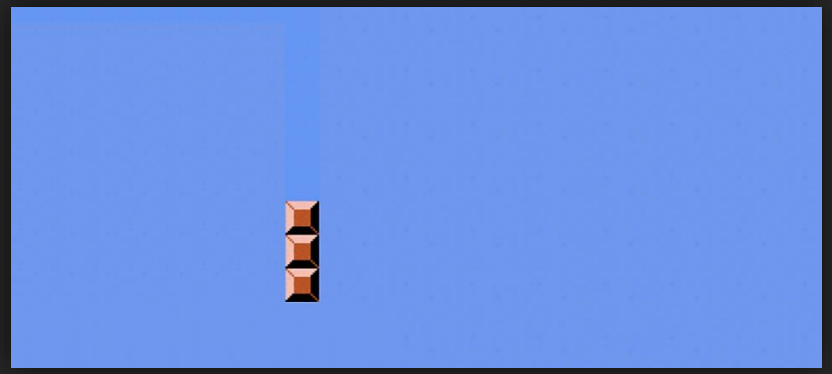
- اکنون، شما به مجموعه ای از داده های جالب در مورد این دانش آموزان دسترسی دارید. اکنون کد خود را به صورت زیر تغییر دهید:

hogwarts.py

```
students = [
    {"name": "Hermoine", "house": "Gryffindor", "patronus": "Otter"},
    {"name": "Harry", "house": "Gryffindor", "patronus": "Stag"},
    {"name": "Ron", "house": "Gryffindor", "patronus": "Jack Russell terrier"},
    {"name": "Draco", "house": "Slytherin", "patronus": None},
]

for student in students:
    print(student["name"], student["house"], student["patronus"], sep=", ")
```

- توجه کنید که چگونه حلقه `for` داخل هریک از `dict` های موجود در `list` ای به نام `students` را پیمایش می‌کند.
- برای اطلاعات بیشتر در رابطه ها ساختار داده دیکشنری یا `dict`، می‌توانید به این لینک مراجعه کنید.



- بازی کلاسیک ماریو که دارای قهرمانی است که از روی آجرها می‌پرد را به یاد دارید؟ بیایید این بازی را به صورت متنی پیاده سازی کنیم.

- با کد رو به رو شروع کنید:

mario.py

```
print("#")
print("#")
print("#")
```

توجه کنید که چگونه یک کد را بارها و بارها کپی و جای گذاری می‌کنیم.

- می‌توانیم کد را به صورت رو به رو بهتر کنیم:

```mario.py
for _ in range(3):
    print("#")
```

این کد کاری را که می‌خواهیم انجام می‌دهد اما بسیار تمیز تر از کد قبلیست.

- در نظر بگیرید که آیا می‌توانیم برای حل مسائل پیچیده تر بعداً از این کد استفاده کنیم؟ کد خود را به صورت زیر تغییر دهید:

```mario.py
def main():
    print_column(3)

def print_column(height):
    for _ in range(height):
        print("#")

main()
```

توجه داشته باشید که اکنون می‌توانیم هر اندازه که بخواهیم ستون چاپ کنیم بدون آن که هیچ کد غیر بهینه بنویسیم.

Hard code

- اکنون، بیایید سعی کنیم یک ردیف را به صورت افقی چاپ کنیم. کد خود را به صورت زیر تغییر دهید:

mario.py

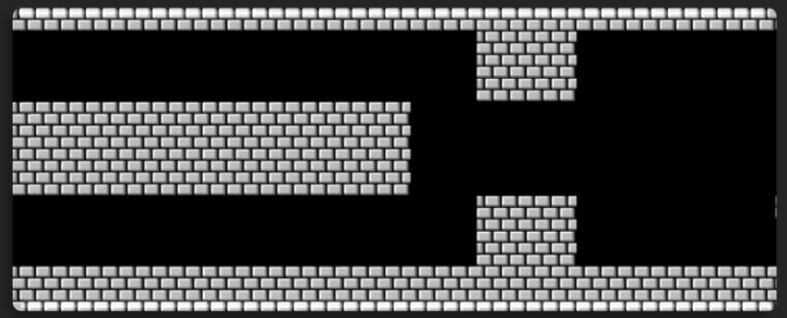
```
def main():
    print_row(4)

def print_row(width):
    print("?" * width)

main()
```

توجه کنید اکنون کدی داریم که می‌تواند بلوک های چپ به راست ایجاد کند.

- با بررسی تصویر زیر، متوجه شوید که چگونه ماریو دارای هر دوی ردیف و ستون از بلوک‌ها است.



- در نظر بگیرید، چگونه می‌توانیم هم ردیف ها و هم ستون ها را در کد خود پیاده سازی کنیم؟ کد خود را به صورت زیر تغییر دهید:

```
mario.py

def main():
    print_square(3)

def print_square(size):

    # For each row in square
    for i in range(size):

        # For each brick in row
        for j in range(size):

            #  Print brick
            print("#", end="")

        # Print blank line
        print()

main()
```

توجه کنید که ما یک حلقه برای هر ردیف در مربع داریم. سپس، یک حلقه دیگر درون آن داریم که در هر ردیف یک آجر چاپ می‌کند. در نهایت، یک دستور چاپ داریم که یک خط خالی چاپ می‌کند.

می‌توانیم کد خود را کوتاه تر کنیم:

mario.py

```
def main():
    print_square(3)

def print_square(size):
    for i in range(size):
        print_row(size)

def print_row(width):
    print("#" * width)

main()
```

در این هفته مباحث زیر را مورد بررسی قرار دادیم:

- ✓ حلقه ها
- ✓ حلقه `while`
- ✓ حلقه `for`
- ✓ Improving with user input
- ✓ More about lists
- ✓ Length
- ✓ Dictionaries
- ✓ Mario

حالا یک توانایی به لیست رو به افزایش توانایی های شما در زبان پایتون اضافه شد. این رشد ادامه داره...

CS50x Iran

Harvard's Computer Science 50x Iran