CS50x Iran

File I/O

Week note

پاییز و زمستان ۱۴۰۴

1 فهرست

- تا الان در تمام برنامه هایی که نوشتیم، اطلاعاتمون رو روی حافظه ذخیره میکردیم. به همین دلیل وقتی که برنامه به اتمام برسه، تمام اطلاعاتی که از کاربر گرفته شده و یا اطلاعاتی که توسط برنامه تولید شده از بین خواهند رفت.
- File I/O به قابلیتی از یک برنامه گفته می‌شود که فایلی را به عنوان ورودی دریافت کند و یا یک فایل را به عنوان خروجی ارائه دهد.
- برای شروع در ترمینال خود دستور code names.py را وارد کنید تا فایل مورد نیاز ساخته شود و سپس کدهای زیر را در فایل ساخته شده بنویسید.

```
name = input("What's your name?" )
print(f"hello, {name}")
```

خروجی این کد درست مطابق چیزیست که انتظار داریم. کاربر اسم خود را وارد کرده و برنامه متن موردنظر را چاپ میکند.

- با این حال اگر قصد داشتیم که چند اسم را از کاربر به عنوان ورودی دریافت کنیم، چطور باید آن را پیاده سازی میکردیم؟ همانطور که در درس‌های گذشته یادگرفتیم، ساختار داده list برای ذخیره چندین مقدار در یک متغیر مورد استفاده قرار میگرفت. حالا به کمک list کد خود را به شکل زیر تغییر دهید.

```
names = []

for _ in range(3):
    name = input("What's your name?" )
    names.append(name)
```

در این کد از کاربر 3 بار درخواست می‌شود که نام خود را وارد کند. و سپس مقدارهای وارد شده در هر بار اجرا به لیست names اضافه می‌شود.

- این کد را میتوانیم به شکل زیر ساده کنیم. با اجرا کردن این کد متوجه می‌شوید که نتیجه‌ای مشابه با کد قبلی دارد.

```
names = []

for _ in range(3):
    names.append(input("What's your name?" ))
```

- حالا قصد داریم لیستی که شامل اسم‌های ذخیره شده است را بصورت مرتب شده نمایش دهیم.

```
names = []

for _ in range(3):
    names.append(input("What's your name?" ))

for name in sorted(names):
    print(f"hello, {name}")
```

این برنامه درست به شکلی که انتظار داریم اجرا می‌شود، اما پس از اجرای کد، تمام اطلاعات ذخیره شده از بین می رود و دیگر قابل دسترس نخواهد بود.

File I/O این امکان را به ما میدهد که از این اطلاعات ذخیره شده در ادامه نیز استفاده کنیم.

- در لینک رو به رو می توانید درباره دستور sorted بیشتر مطالعه کنید: sorted

- Open یک تابع پیش فرض پایتون می‌باشد که به ما این امکان را میدهد که فایلی را در برنامه خود باز کرده و فرایندهای خواندن و نوشتن را روی آن انجام بدهیم.
- برای اینکه نحوه استفاده از File I/O را بهتر درک کنیم، کدمون رو مرور میکنیم.

```
name = input("What's your name? ")

file = open("names.txt", "w")
file.write(name)
file.close()
```

در این کد، تابع open فایلی به اسم names.txt را در حالت نوشتن دیتا بر روی فایل باز میکند که توسط w (Write) تعیین شده است. پس از باز کردن فایل مدنظر، باید آن را به یک متغیر نسبت دهیم تا برای نوشتن دیتا بر روی آن از این متغیر استفاده کنیم. در این کد اسم متغیر file انتخاب شده است و در خط بعد با دستور file.write(name)

نامی که کاربر وارد می‌کند بر روی فایل نوشته می‌شود و در خط بعد این فایل بسته شده و دیگر به آن دسترسی نخواهیم داشت.

- برای تست برنامه می‌توانید با وارد کردن دستور python names.py در ترمینال خود، و وارد کردن نام خود خروجی آن را ببینید که چطور نام وارد شده بر روی فایلی با نام names.txt ذخیره شده است. اما با چند بار اجرا کردن این کد متوجه خواهید شد که تنها آخرین اسم وارد کرده در فایل وجود دارد و عملا هر بار اجرای برنامه محتوای فایل را پاک کرده و از اول دیتا را می نویسد.

- در حقیقت چیزی که ما نیاز داریم این است که با هر بار اجرای برنامه و وارد کردن اسم جدید، این مقدار به فایل مدنظر اضافه شده و مقدارهای قبل نیز حفظ شود. فایل فعلی را با وارد کردن دستور rm names.txt در ترمینال حذف کنید و کد خود را به شکل زیر تغییر دهید.

```
name = input("What's your name? ")

file = open("names.txt", "a")
file.write(name)
file.close()
```

دقت کنید که تنها تفاوت این دو کد تغییر w به a (append) میباشد. حالا با چندین بار اجرای کد متوجه می‌شوید که تمام نام ها بر روی فایل ذخیره می‌شوند. اما با مشکلی جدید روبرو هستیم!

- با باز کردن فایل names.txt متوجه می‌شوید که تمام اسم‌ها بدون هیچ فاصله‌ای بصورت پشت سر هم در فایل ذخیره شده‌اند. اما به راحتی این مشکل را نیز میتوانیم حل کنیم. فایل فعلی را با دستور rm names.txt حذف کنید و کد خود را به شکل زیر تغییر دهید.

```
name = input("What's your name? ")

file = open("names.txt", "a")
file.write(f"{name}\n")
file.close()
```

در کد نهایی ما با اضافه کردن کارکتر \n (به عنوان شکننده خط مورد استفاده قرار میگیرد) به انتهای هر اسم، مشکل را حل کردیم .

- الان این برنامه بدون مشکل اجرا میشود ولی همیشه راه هایی برای بهتر کردن و توسعه برنامه وجود دارد. یکی از اشتباهات رایجی که در استفاده از فایل‌ها صورت میگیرد، نبستن فایلی است که در برنامه مورد استفاده قرارگرفته است.
- در لینک رو به رو نیز می توانید بیشتر درباره open بخوانید:

- استفاده از with به شما این امکان را میدهد تا نیازی به بستن فایل نداشته باشید و این کار بصورت خودکار صورت گیرد.
- حالا کد خود را به شکل زیر تغییر دهید.

```
name = input("What's your name? ")

with open("names.txt", "a") as file:
    file.write(f"{name}\n")
```

به این نکته توجه کنید که چطور خط زیر with دارای تو رفتگی می‌باشد.

- تا به الان نوشتن بر روی یک فایل را تمرین کردیم. اما برای خواندن مقدار موجود در یک فایل باید از چه روشی استفاده کنیم؟برای انجام این کار کد خود را به شکل زیر تغییر دهید.

```
with open("names.txt", "r") as file:
    lines = file.readlines()

for line in lines:
    print("hello,", line)
```

در این برنامه دستور readlines این امکان را میدهد که تمام خطهای یک فایل را بخوانیم و سپس آن را بر روی متغیری دیگر ذخیره کنیم. با اجرای برنامه متوجه خواهید شد که خروجی آن خیلی غیر منتظره می‌باشد. دلیلش این هست که ، به جای تنهای یک \n به عنوان شکننده خط، چندین شکننده خط استفاده شده است.

- برای حل این مشکل چندین راه حل وجود دارد اما یکی از ساده ترین آنها روش زیر می‌باشد.

```
with open("names.txt", "r") as file:
    lines = file.readlines()

for line in lines:
    print("hello,", line.rstrip())
```

تابع rstrip باعث حذف کارکترهای \n اضافی در انتهای هر خط می‌شود.

- این کد را می توان بیشتر هم ساده کرد:

```
with open("names.txt", "r") as file:
    for line in file:
        print("hello,", line.rstrip())
```

این کد درست اجرا می شود، با این حال، توجه داشته باشید که ما نام ها را مرتب نمی کنیم.

- می توانیم این کد را بهتر کنیم تا اسامی خوانده شده از فایل را بصورت مرتب شده نمایش دهیم:

```
names = []

with open("names.txt") as file:
    for line in file:
        names.append(line.rstrip())

for name in sorted(names):
    print(f"hello, {name}"
```

در ابتدای برنامه names یک لیست خالی می‌باشد که در ادامه‌ی برنامه نام های خوانده شده از فایل به آن اضافه می‌شود. و سپس نام های موجود در این لیست بصورت مرتب شده نمایش داده می‌شوند. با اجرا برنامه خود میبینید که خروجی متناسب با انتظار را نمایش می‌دهد.

- اما اگر بخواهیم اطلاعات بیشتری از اسم دانشجو را ذخیره کنیم چطور؟چطور میتوانیم علاوه بر اسم دانشجو، نام گروه آنها را نیز ذخیره کنیم؟

- CSV مخفف (Camma separated value) یا مقادیر جدا شده توسط ویرگول می‌باشد.
- در ترمینال خود دستور code students.csv را وارد کنید و دقیقا مقدار زیر را وارد کنید.

```
Hermoine,Gryffindor
Harry,Gryffindor
Ron,Gryffindor
Draco,Slytherin
```

- بیایید با دستور code students.py فایلی برای نوشتن کد ایجاد کنیم و کد زیر را بنویسیم :

```
with open("students.csv") as file:
    for line in file:
        row = line.rstrip().split(",")
        print(f"{row[0]} is in {row[1]}")
```

تابع rstrip کارکترهای ناخواسته انتهای هر خط از فایل csv را حذف می‌کند. تابع split نیز هر خط را از کارکتر ویرگول می‌شکند و بدین صورت مقدارهای هر خط را از هم جدا می‌کند. بنابراین row[0] اولین مقدار هر خط و row[1] دومین مقدار هر خط از فایل CSV می‌باشد.

- این کد برای جدا کردن مقادیر یک فایل CSV مناسب است. اما ممکن است از نظر ظاهر کمی گیج کننده به نظر برسد. برای ساده تر کردن کد می‌توانیم از یکی از ویژگی‌های پایتون استفاده کنیم. کد خود را به شکل زیر ویرایش کنید.

```
with open("students.csv") as file:
    for line in file:
        name, house =
line.rstrip().split(",")
        print(f"{name} is in
{house}")
```

تابع split در واقع دو مقدار را به عنوان خروجی برمیگرداند، یکی مقدار قبل از ویرگول و دیگری، مقدار بعد از ویرگول. به همین دلیل می‌توانیم این دو مقدار را به جای یک متغیر ، به دو متغیر جدا نسبت دهیم.

- حالا اگر بخواهیم مقدار این فایل را بصورت مرتب شده در خروجی نمایش دهیم چطور؟
  برای دستیابی به این هدف کد خود را به شکل زیر تغییر دهید.

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        students.append(f"{name} is in {house}")

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

در ابتدا یک لیست ایجاد کرده، سپس هر رشته را به آن اضافه می‌کنیم و در نهایت نسخه مرتب شده آن را در خروجی چاپ می‌کنیم.

- برای توسعه بیشتر کد می‌توانیم از داده ساختار dictionary که امکان ذخیره اطلاعات بصورت key value را به ما می‌دهد استفاده کنیم.

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        student = {}
        student["name"] = name
        student["house"] = house
        students.append(student)

for student in students:
    print(f"{student['name']} is in {student['house']}")
```

در این کد، ابتدا اسم و گروه هر دانشجو را در دیکشنری student ذخیره می‌کنیم و سپس آن را به لیست students اضافه میکنیم.

- برای بهتر شدن عملکرد کد می‌توانیم آن را به شکل زیر تغییر دهیم.

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        student = {"name": name, "house": house}
        students.append(student)

for student in students:
    print(f"{student['name']} is in {student['house']}")
```

توجه داشته باشید که این کد نتیجه مطلوب را ایجاد میکند به جز مرتب سازی دانش آموزان.

- متأسفانه ما نمی‌توانیم دانش‌آموزان را همانطور که قبلاً داشتیم مرتب کنیم چون حالا هر دانش آموز از لیست یک دیکشنری بوده و شامل مقدارهای اسم و گروه دانش آموز می‌باشد. برای مرتب کردن این لیست باید آیتم های آن را بر اساس اسم ذخیره شده در هر دیکشنری مرتب کنیم.

- از کد زیر برای رسیدن به این هدف استفاده کنید.

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        students.append({"name": name, "house": house})


def get_name(student):
    return student["name"]


for student in sorted(students, key=get_name):
    print(f"{student['name']} is in {student['house']}")
```

تابع sorted نیاز دارد تا بداند بر چه اساس باید لیست مدنظر را مرتب کند. پایتون این امکان را به ما می‌دهد تا با پارامتری به نام key مشخص کنیم لیست مدنظر بر چه اساسی مرتب شود. تابع get_name مقدار اسم هر دانشجو را برمیگرداند و بنابراین لیست ما بر اساس نام مرتب می‌شود. برنامه را اجرا کرده و نتیجه آن را مشاهده کنید.

- اما بازهم می‌توانیم کد خود را ارتقا دهیم، اگر از تابع get_name تنها برای مرتب کردن این لیست استفاده میکنید و کاربرد دیگری در برنامه شما ندارد، آن را به شکل زیر تغییر دهید.

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        students.append({"name": name, "house": house})

for student in sorted(students, key=lambda student:
student["name"]):
    print(f"{student['name']} is in {student['house']}")
```

در این کد از تابع lambda استفاده شده است که در پایتون به توابع ناشناس (anonymous) شناخته می‌شوند زیرا هیچ اسمی ندارند. به عنوان ورودی به آن student را پاس می‌دهیم و مقدار خروجی آن کلید name از آن دانشجو خواهد بود.

- متاسفانه کد فعلی ما کمی ضعیف است و با ایجاد تغییر کوچک در فایل csv عملکرد صحیح خود را از دست می‌دهد. به عنوان مثال بیایید محل زندگی هر دانشجو را نیز در فایل csv قراردهیم. با اجرای این کد چه چیزی رخ خواهد داد؟ ابتدا فایل csv خود را به شکل زیر تغییر دهید.

```
Harry,"Number Four, Privet Drive"
Ron,The Burrow
Draco,Malfoy Manor
```

به ارورهای ایجاد شده توسط برنامه دقت کنید.

- ابتدا چون با محل زندگی هر دانشجو سر و کار داریم اسم متغیر houses را به homes تغییر دهید.

```
students = []

with open("students.csv") as file:
    for line in file:
        name, home = line.rstrip().split(",")
        students.append({"name": name, "home": home})

for student in sorted(students, key=lambda student:
student["name"]):
    print(f"{student['name']} is in {student['home']}")
```

با اجرای برنامه متوجه میشوید که برنامه ما همچنان به درستی کار نمیکند. میتوانید دلیل آن را حدس بزنید؟

- ارور ValueError: too many values to unpack زمانی رخ میدهد که تعداد مقدارهای نسبت داده شده با تعداد متغیرها برابر نباشد. ما در کد خود میخواستیم فایل CSV را با استفاده از , (کاما) تقسیم کنیم. اما در این مثال اینطور نیست. ما می‌توانیم زمان بیشتری را برای پرداختن به این موضوع صرف کنیم، اما در واقع شخص دیگری قبلاً راهی برای "تجزیه" (یعنی خواندن) فایل‌های CSV ایجاد کرده است!
- کتابخانه csv بطور پیشفرض در پایتون وجود دارد و دارای ابجکتی به نام reader می‌باشد. از این ابجکت می‌توانیم برای خواندن اطلاعات فایل خود بدون مواجه شدن با مشکل خاصی استفاده کنیم. reader در یک حلقه قابل استفاده می‌باشد. به این صورت که در هر بار اجرای حلقه این ابجکت یک خط از فایل csv را برمیگرداند که آن خط نیز خود یک لیست می‌باشد که هر آیتم آن برابر با مقدارهای مجزا در فایل csv ما هستند.

```
import csv

students = []

with open("students.csv") as file:
    reader = csv.reader(file)
    for row in reader:
        students.append({"name": row[0], "home": row[1]})

for student in sorted(students, key=lambda student:
student["name"]):
    print(f"{student['name']} is from {student['home']}")
```

و بلاخره برنامه ما بدون مشکل اجرا شد.

- تا این مرحله، ما اینطور درنظر داشتیم که اولین مقدار از هر خط برابر با اسم دانشجو و مقدار دوم محل زندگی او می‌باشد. اما بهتر است این اطمینان به مقادیر را کنار گذاشته و این فرآیند را بصورت خودکار پیش ببریم. برای این کار مقدار هر ستون را در اولین سطر از فایل مشخص می‌کنیم.

```
name,home
Harry,"Number Four, Privet Drive"
Ron,The Burrow
Draco,Malfoy Manor
```

در این فایل مشخص می‌کنیم که هر ستون به چه معناست.

- برای تعامل با این فایل می‌توانیم از آبجکت DictReader از کتابخانه csv استفاده کنیم تا مقیاس پذیری برنامه خود را افزایش دهیم.

```
import csv

students = []

with open("students.csv") as file:
    reader = csv.DictReader(file)
    for row in reader:
        students.append({"name": row["name"], "home": row["home"]})

for student in sorted(students, key=lambda student:
student["name"]):
    print(f"{student['name']} is in {student['home']}")
```

در این برنامه ما از DictReader به جای reader استفاده کردیم تا در هر با اجرای حلقه یک دیکشنری به جای لیست داشته باشیم که نام ستون را بطور خودکار از فایل دریافت کرده باشد. همانطور که میبینید توسط دیکشنری row به دو کلید name و home دسترسی خواهیم داشت.

- خب تا به اینجای کار خواندن از یک فایل csv را تمرین کردیم. اما چطور می‌توانیم بر روی آن بنویسیم؟
- برای شروع، کمی محیط کار خود را تمیزتر می‌کنیم. ابتدا فایل students.csv را با دستور rm students.csv حذف می‌کنیم.
- و سپس فایل students.py را به شکل زیر تغییر می‌دهیم.

```
import csv

name = input("What's your name? ")
home = input("Where's your home? ")

with open("students.csv", "a") as file:
    writer = csv.DictWriter(file, fieldnames=["name", "home"])
    writer.writerow({"name": name, "home": home})
```

در این برنامه از آبجکت DictWriter به نحوی استفاده می‌کنیم که فایل مدنظر و نام ستون های فایل را به آن پاس می‌دهیم تا در ادامه بتوانیم در ستون ها اطلاعات را وارد کنیم. و سپس تابع writerow یک دیکشنری را به عنوان ورودی گرفته که شامل اسم ستون و مقداریست که قصد داریم در فایل ذخیره کنیم.

- انواع مختلفی از فایل وجود دارد که می‌توانیم بر روی آنها بنویسیم و یا از آنها بخوانیم.
- می توانید در داکیومنت رسمی پایتون اطلاعات بیشتری از کتابخانه CSV بدست بیاورید.

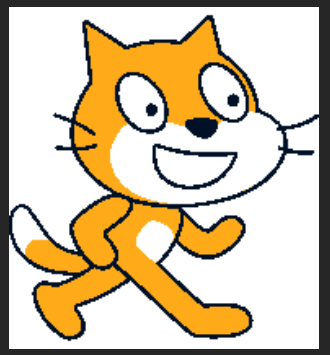
- یکی دیگر از انواع فایلی که امروز درباره آنها بحث خواهیم کرد، فایل‌های باینری هستند. یک فایل باینری تنها شامل مجموعه‌ای از صفر و یک می‌باشد. این نوع از فایل می تواند هر چیزی، از جمله موسیقی و تصویر را ذخیره کند.

- پایتون شامل یک کتابخانه به نام PIL می‌باشد که به خوبی امکان تعامل با تصاویر را به ما میدهد.

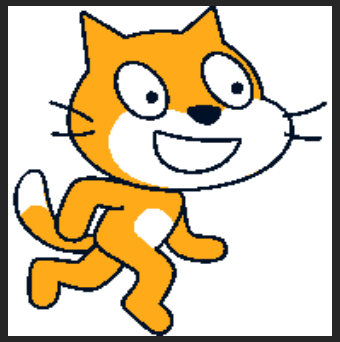
- GIFهای متحرک یک نوع محبوب از فایل های تصویری هستند که در حقیقت شامل چندین تصویر می-باشند که بصورت پشت سرهم و متوالی تکرار می‌شوند و یک انیمیشن یا افکت ویدیویی ساده ایجاد می کنند.

- تصور کنید که ما چندین تصویر به شکل زیر داریم:

- در ابتدا تصویری به نام costume1.gif داریم:



- و یک تصویر دیگر به نام costume2.gif که نسبت به تصویر قبل موقعیت پاها کمی متفاوت می‌باشد.



قبل از اجرای کد تصاویر بالا را در دایرکتوری پروژه خود ذخیره کنید تا برای برنامه قابل دسترسی باشد.

- سپس با دستور code customes.py فایلی ایجاد کرده و کدهای زیر را درون آنها قرار دهید.

```
import sys

from PIL import Image

images = []

for arg in sys.argv[1:]:
    image = Image.open(arg)
    images.append(image)

images[0].save(
    "costumes.gif", save_all=True, append_images=[images[1]],
duration=200, loop=0
)
```

در ابتدا ما کلاس Image از ماژول PIL را import کردیم. سپس برروی تصاویری که از طریق ورودی‌های command-line وارد شده است حلقه می‌زنیم و آنها را در یک لیست به نام images ذخیره می‌کنیم.

در قسمت sys.argv[1:] به این نکته اشاره داریم که پارامترهای ورودی از طریق command-line از آیتم دوم برای حلقه درنظر گرفته شود. چرا که اولین آیتم اسم فایلی است که قصد اجرای آن را داریم (costumes.py) . در آخرین خط از برنامه، اولین عکس دریافت شده را ذخیره کرده و سپس دومین تصویر را به آن اضافه می‌کنیم تا یک GIF ایجاد شود. سپس برنامه خود را با دستور python costumes.py costume1.gif costume1.gif اجرا کنید. سپس در ترمینال خود دستور code costumes.gif را وارد کنید تا فایل خروجی برنامه را ببینید.

- برای اطلاعات بیشتر درباره ماژول PIL میتوانید به این لینک مراجعه کنید.

در این جلسه به خواندن و نوشتن متن و البته صفر و یک در یک فایل پرداختیم. بسیار مشتاق دیدن خلاقیت‌های شما در استفاده از این توانایی کسب شده هستیم.

- File I/O ✓
- Open ✓
- With ✓
- CSV ✓
- PIL ✓

CS50x Iran

Harvard's Computer Science 50x Iran