

# ماجراجویی با پایتون

سفری پروژه محور به دنیای برنامه نویسی

نویسنده و مؤلف: محمد مهدی قائمی



نام و لوگوی انتشارات

: سرشناسه
: عنوان قراردادی
: عنوان و نام پدیدآور
: مشخصات نشر
: مشخصات ظاهري
: شابک
: وضعیت فهرستنویسی
: یادداشت
: موضوع
: موضوع
: ردبهندی کنگره
: ردبهندی دیوی
: شماره کتابشناسی ملی

# ماجراجویی با پایتون

نویسنده: محمد مهدی قائمی

ناشر: انتشارات

شمارگان: نسخه

نوبت چاپ: اول، ۱۴۰۴

قیمت: تومان

شابک: - - - ۶۰۰- ۹۷۸-

شناسنامه کتاب در این صفحه می‌آید.



# سخن ناشر

این کتاب با هدف کمک به علاقهمندان به یادگیری برنامهنویسی پایتون تهیه شده است.  
امیدواریم مطالعه‌ی آن گامی هرچند کوچک در مسیر رشد علمی و فردی شما باشد.  
این اثر را با عشق تقدیم می‌کنیم به همه‌ی کسانی که در راه علم، یادگیری و پیشرفت  
تلاش می‌کنند.



# مقدمه

سلام (: من محمد مهدی قائمی هستم و خیلی خوشحالم که تصمیم گرفتید وارد دنیای برنامه‌نویسی شوید. در این کتاب قرار است با هم از صفر شروع کنیم، پایتون را یاد بگیریم و در طول مسیر با پروژه‌های جذاب و واقعی، یادگیری را ساده‌تر و لذت‌بخش‌تر کنیم. از آغاز تمدن، انسان همواره به دنبال ابزاری بوده است که کارهای سخت و تکراری را ساده‌تر کند. از چرتکه و ماشین حساب‌های ابتدایی گرفته تا اختراع کامپیوتر، همه در خدمت همین نیاز بوده‌اند: محاسبه سریع‌تر، دقیق‌تر و صرفه‌جویی در زمان. با پیدایش کامپیوترها، بشر توانست مسائلی را حل کند که پیش‌تر غیرممکن به نظر می‌رسید، اما کار کردن مستقیم با زبان‌های سخت‌فهم ماشین برای بسیاری دشوار بود. همین چالش باعث شد زبان‌های برنامه‌نویسی به وجود بیایند تا پلی باشند میان انسان و ماشین.

در میان صدھا زبان برنامه‌نویسی که در طول تاریخ ساخته شده‌اند، پایتون جایگاه ویژه‌ای پیدا کرد. پایتون زبانی ساده، روان و قدرتمند است که یادگیری آن حتی برای مبتدیان هم آسان است. در عین حال، آنقدر توانمند است که بزرگ‌ترین شرکت‌های فناوری دنیا از آن برای هوش مصنوعی، وب، تحلیل داده و حتی فضانوردی استفاده می‌کنند. به همین دلیل، امروز پایتون به یکی از محبوب‌ترین زبان‌های برنامه‌نویسی دنیا تبدیل شده و میلیون‌ها نفر در مسیر یادگیری و ساخت آینده‌ای هوشمند، آن را انتخاب کرده‌اند.



# فهرست مطالب

ط	فهرست مطالب
ف	بخش اول: شروع ماجراجویی!
۱۹	فصل ۱ : اولین قدم: آماده‌سازی محیط توسعه
۲۰	چرا پایتون؟
۲۱	آماده کردن کوله‌پشتی: نصب پایتون یا استفاده از آزمایشگاه آنلайн
۲۲	مسیر اول: ساخت کارگاه شخصی (نصب پایتون روی کامپیوتر و لپتاپ)
۲۲	قدم اول: نصب مترجم پایتون
۲۲	۱. برای کاربران ویندوز Windows
۲۳	۲. برای کاربران مک (The Mac Guild)
۲۳	۳. برای کاربران لینوکس (The Linux Clan)
۲۴	قدم دوم: آماده کردن کتاب جادو (محیط توسعه VS Code)
۲۵	قدم سوم : قدرتمند کردن کتاب جادو با افزونه پایتون
۲۵	مسیر دوم: استفاده از آزمایشگاه جادویی آنلайн (Google Colab)
۲۶	اولین ورد جادویی: اجرای print("Hello, World!") و دیدن نتیجه!
۳۰	پروژه: ساخت کارت ویزیت دیجیتال
۳۳	تبديل شدن به کارآگاه کد
۳۴	ذهنیت یک ماجراجو: حل معما، نه حفظ کردن ورد!
۳۵	جعبه ابزار کارآگاه

۳۵	۱. Stack Overflow: کتابخانه بزرگ دانش تمام جادوگران دنیا
۳۵	۲. دستیارهای هوشمند (AI): غول‌های چراغ جادوی جدید
۳۷	پروژه: شکار باگ در کد طلسمنشده
۴۹	<b>فصل ۳: ساخت اولین ابزار: ماشین حساب جادویی</b>
۴۰	۱. جعبه‌های جادویی (متغیرها)
۴۱	۲. آشنایی با غنائم: انواع داده‌های پایه (Primitive Types)
۴۲	۴۲ سکه‌های طلا (Integer) 
۴۲	۴۲ گرد جادویی (Float) 
۴۳	۴۳ طومارهای باستانی (String) 
۴۳	۴۳ کلیدهای جادویی (Boolean) 
۴۳	۴۳ جادوی ریاضی: کار با عملگرها
۴۵	۴۵ پروژه: ساخت ماشین حساب جادویی
۴۷	<b>فصل ۴: رام کردن منطق: ساخت اولین بازی حدس عدد</b>
۴۸	۱. صحبت با ماجراجو: ورد جادویی <code>input()</code>
۴۸	۲. تله‌ی کیمیاگری: تبدیل «String» به «Integer» با <code>int()</code>
۴۹	۳. طلسمنهای مقایسه و دوراهی جادویی <code>if</code> و <code>else</code>
۵۰	۴. مغز متفکر بازی: مسیرهای چندگانه با <code>elif</code>
۵۱	۵. تکرار تا پیروزی: حلقه <code>while</code>
۵۲	۶. پروژه: پیاده‌سازی کامل بازی حدس عدد
۵۵	<b>بخش دوم: مدیریت کوله پشتی ماجراجو</b>
۵۷	<b>فصل ۵: مدیریت هوشمند کوله پشتی: لیست خرید</b>
۵۸	۱. کوله پشتی جادویی (Lists)
۵۸	۲. پر کردن کوله پشتی (اضافه کردن، دیدن و حذف کردن)
۵۸	ورد (): اضافه کردن آیتم <code>append()</code>

۵۹	خواندن آیتم‌ها با «ایندکس» (Index)
۶۰	ورد () : حذف کردن آیتم remove()
۶۰	۳. بازرسی کوله پشتی (طلسم‌های in و len)
۶۰	ورد () len : طلسم شمارش‌گر
۶۱	ورد in : طلسم جستجوگر
۶۲	۴. سرشماری غنائم (حلقه for)
۶۳	۵. پروژه: ساخت برنامه مدیریت لیست خرید

## فصل ۶: تکنیک‌های پیشرفته کوله پشتی: تابلوی امتیازات

۶۸	۱. برش زدن کوله پشتی (Slicing)
۶۸	۲. کیسه‌ی مُهر و موم شده (Tuples)
۶۹	۳. جادوی باز کردن تاپل (Tuple Unpacking)
۷۰	۴. ترکیب جادوها: لیستی از تاپل‌ها
۷۱	۵. پروژه: ساخت تابلوی امتیازات "Top 3"

## فصل ۷: کتاب جادویی: ساخت مترجم کلمات

۷۶	۱. دیکشنری‌ها (Dictionaries): کتاب جادویی شما
۷۷	۲. جادوی واژه‌نامه: افزودن، تغییر و حذف
۷۷	اضافه کردن یا تغییر دادن یک «ورد»
۷۷	حذف کردن یک «ورد»
۷۸	۳. تله‌ی کلید گمشده (The KeyError)
۷۸	۴. طلسم‌های اینمی (get) و in
۷۸	طلسم اول: بررسی با in
۷۹	طلسم دوم (حرفه‌ای): متدهای get()
۷۹	۵. سرشماری در کتاب جادویی (Looping)
۸۱	۶. هشدار ماجراجو: طلسم نمایش حروف فارسی 
۸۲	۷. پروژه: ساخت مترجم کلمات

۸۷	<b>فصل ۸: ساخت جادوهای شخصی: هنر ساخت توابع</b>
۸۸	۱. "یک بار بنویس، صد بار استفاده کن": معرفی توابع.....
۸۹	۲. «تعریف» (Define) و «صدا زدن» (Call) یک طلسما.....
۹۰	۳. اولین طلسما: خداحافظی با تکرار.....
۹۱	۴. ورودی و خروجی توابع (return Arguments و.....
۹۳	۵. پروژه: رفع طلسما «طومار تکراری».....
۹۷	<b>فصل ۹: طلسما محافظتی: ساخت رمز عبور امن</b>
۹۸	۱. استفاده از قدرت‌های مخفی پایتون: کتابخانه‌ها و import.....
۹۸	۲. کتابخانه random: بهترین دوست ما برای کارهای تصادفی.....
۱۰۰	۳. جادوی شمارش: تکرار با for i in range().....
۱۰۱	۴. پروژه: انتخاب رمز برای صندوقچه گنج خودمان.....
۱۰۲	<b>بخش سوم: ورود به بعد چهارم</b>
۱۰۳	<b>فصل ۱۰: رمزگشایی از طومارهای باستانی: تحلیلگر متن</b>
۱۰۴	۱. کار حرفه‌ای با متن‌ها (طلسم‌های String).....
۱۰۵	۲. طلسما اول: جادوی باستانی (قالب‌بندی با %).....
۱۰۶	۳. طلسما دوم: جادوی مدرن (متدهای format() و .....)
۱۰۶	۴. طلسما سوم: جادوی استاد (f-strings).....
۱۰۷	۳. پروژه: ساخت تحلیلگر متن.....
۱۱۱	<b>فصل ۱۱: طلسما ضد فراموشی: کار با فایل‌ها</b>
۱۱۲	۱. چطور از شر "کرش" کردن خلاص شویم؟ (try و except).....
۱۱۳	۲. خواندن و نوشتمن در فایل‌های متنه (txt).....
۱۱۴	۳. زیان جادویی جهانی (JSON).....
۱۱۶	۴. طلسما زبان‌های باستانی: جادوی encoding (مخصوص فارسی).....
۱۱۷	۵. پروژه: ارتقاء «مدیر لیست خرید».....

۱۲۱	فصل ۱۲: دمیدن روح در کد: خلق موجودات با شیء‌گرایی
۱۲۲	۱. ترکیب داده و رفتار: کلاس (Class) و شیء (Object)
۱۲۳	۲. نقشه ساخت (کلاس) و طلسما خلقت ( <code>__init__</code> )
۱۲۴	۳. ویژگی‌ها (Attributes): ذخیره داده‌ها روی <code>self</code>
۱۲۵	۴. رفتارها (Methods): افروزن جادو به موجودات
۱۲۶	۵. پروژه: ساخت مغازه جادوگری
۱۳۰	بخش چهارم: آیین‌های جادوگر ارشد
۱۳۱	فصل ۱۳: سازماندهی کتابخانه جادویی (ساخت مأذول‌های شخصی)
۱۳۲	۱. چرا یک فایل کافی نیست؟ (اصل تفکیک مسئولیت‌ها)
۱۳۳	۲. طلسما <code>import</code> شخصی (ساخت اولین مأذول)
۱۳۵	۳. پروژه: معماری «غازه جادوگری»
۱۴۱	فصل ۱۴: کوله پشتی جادویی زمان
۱۴۲	۱. حضار <code>datetime</code>
۱۴۲	۲. طلسما <code>now()</code> (گرفتن لحظه حال)
۱۴۳	۳. قالب‌بندی زمان ( <code>strftime</code> طلسما)
۱۴۴	۴. سفر در زمان ( <code>timedelta</code> طلسما)
۱۴۵	۵. پروژه: اعلان‌گر مهلت (The Deadline Notifier)
۱۴۷	فصل ۱۵: آیین قبیله پایتون هنر کدنویسی پایتونیک
۱۴۸	۱. ذن پایتون (The Zen of Python)
۱۴۸	۲. طلسما <code>enumerate</code> (شمارش پایتونیک)
۱۴۹	۳. طلسما <code>zip</code> (جفت کردن طومارها)
۱۵۰	۴. طلسما تک خطی (List Comprehensions)
۱۵۱	۵. جادوی ناشناس (Lambda)
۱۵۲	۶. پروژه: کیمیاگری لیست (List Alchemy)

۱۵۴	بخش پنجم: نبرد نهایی (پروژه بزرگ ماجراجو)	
فصل ۱۶: نبرد نهایی: ساخت دفترچه مأموریت ماجراجو		
۱۵۵	۱. نگاهی به گذشته (Gathering Your Spells)	
۱۵۶	۲. گزارش شناسایی (The Adventurer's Challenge)	
۱۵۷	۳. میز معمار (Building Step-by-Step)	
بخش ششم: ورود به تالار پیشگویان (مسیر دانشمند داده)		
فصل ۱۷: دروازه احضار و آزمایشگاه جادویی (Pip & Venv)		
۱۶۵	۱. آزمایشگاه ایزوله (venv)	
۱۶۶	۲. فعالسازی آزمایشگاه (ورود به حباب)	
۱۶۷	۳. دروازه احضار (pip)	
۱۶۸	۴. طومار نیازمندی‌ها (requirements.txt)	
۱۶۹	۵. پروژه: تابلوی اعلانات جادویی	
فصل ۱۸: طومارهای NumPy (جادوی محاسبات نورآسا)		
۱۷۱	۱. احضار NumPy	
۱۷۲	۲. چیست؟ (ارتش منظم) ndarray	
۱۷۳	۳. جادوی Vectorization (حذف حلقه‌ها)	
۱۷۴	۴. جادوی ابعاد: ماتریس‌ها	
۱۷۵	۵. جعبه ابزار جادویی: ۱۰ طلسیم پرکاربرد	
۱۷۶	۶. مسابقه بزرگ: مفهوم زمان‌سننجی	
۱۷۷	۷. پروژه: چالش ۱,۰۰۰,۰۰۰ ماجراجو	
فصل ۱۹: کتابخانه بزرگ پانداس (رام کردن داده‌های جدولی)		
۱۸۱	۱. احضار Pandas	
۱۸۲	۲. چیست؟ (جدول جادویی) DataFrame	
۱۸۳	۳. خواندن طومارهای باستانی (فایل‌های CSV)	

۱۸۳.....	۴. ذرهبین جادویی: فیلتر کردن داده‌ها
۱۸۴.....	۵. جعبه ابزار پانداس: ۱۰ طلسم پرکاربرد
۱۸۵.....	۶. پروژه: تحلیل تابلوی امتیازات
۱۸۹	<b>فصل ۲۰: نقشه‌ی مسحور (روایت داستان با <b>Matplotlib</b>)</b>
۱۹۰.....	۱. احضار نقاش سلطنتی ( <b>Matplotlib</b> )
۱۹۰.....	۲. اولین قلم مو ( <b>plot</b> ) - نمودار خطی
۱۹۱.....	۳. ستون‌های مقایسه ( <b>bar</b> ) - نمودار میله‌ای
۱۹۱.....	۴. زیبایی‌شناسی جادویی (شخصی‌سازی)
۱۹۱.....	۵. جعبه ابزار نقاش: ۱۰ طلسم پرکاربرد
۱۹۳.....	۶. پروژه: گزارش تصویری انجمن
۱۹۶	<b>بخش هفتم: ماجراجویی ادامه دارد...</b> 
۱۹۷	<b>فصل ۲۱: افق‌های بی‌پایان (نقشه‌های گنج آینده)</b>
۱۹۸.....	۱. قدم‌های بعدی ( <b>The Essential Next Steps</b> )
۱۹۸.....	۲. مسیر معماران دیجیتال ( <b>Programming &amp; Web</b> )
۱۹۹.....	۳. مسیر پیشگویان داده ( <b>AI &amp; Data</b> )
۲۰۰.....	۴. مسیر محافظان و مدیران سیستم ( <b>SysAdmin &amp; Security</b> )
۲۰۲	<b>سخن پایانی: پایان آغاز</b>
۲۰۳	<b>منابع و مأخذ</b>



# بخش اول: شروع ماجرایی!

در این بخش، مشعل را روشن می‌کنیم، با دنیای کد آشنا می‌شویم و یاد می‌گیریم چطور مثل یک ماجرایی حرفه‌ای، اولین معماها را حل کنیم.



# فصل ۱ : اولین قدم: آماده‌سازی محیط توسعه

مأموریت فصل : کارت معرفی برای ورود به انجمان ماجراجویان

نقشه راه :

- چرا پایتون؟

آماده کردن کوله‌پشتی : نصب پایتون یا استفاده از آزمایشگاه آنلاین (Google

Colab)

- اولین ورد جادویی : اجرای print("Hello, World!") و دیدن نتیجه!

- پروژه : ساخت کارت ویزیت دیجیتال با نام، عنوان و اطلاعات تماس خودتان.

## چرا پایتون؟

صدها زبان برنامه‌نویسی وجود داره، پس چرا همه دارن از پایتون حرف می‌زنن و چرا ما این ماجراجویی رو با پایتون شروع می‌کنیم؟

### ۱. یادگیریش مثل چت کردن، خیلی راحت!

بزرگ‌ترین فرق پایتون با زبان‌هایی مثل Java یا C++ یا سیتکس (قواعد نوشتاری) خیلی ساده و روانی داره. انگار داری به زبان انگلیسی دستور می‌دی. به جای اینکه درگیر کلی علامت عجیب و غریب و قوانین سخت بشی، مستقیم میری سر اصل مطلب و خلاقیت رو پیاده می‌کنی.

### ۲. با یک تیر، سه تا نشونه رو هدف می‌گیری!

پایتون فقط برای شروع کردن عالی نیست؛ اونقدر قدرتمنده که بزرگ‌ترین شرکت‌های دنیا دارن ازش برای ساختن آینده استفاده می‌کنن. با یادگیری پایتون، در واقع کلید ورود به سه تا از خفن‌ترین دنیاهای تکنولوژی رو به دست میاری:

- **هوش مصنوعی (AI)** : فکر کردی جادوهایی مثل ChatGPT چطور یاد

می‌گیرن باهات حرف بزنن، یا ماشین‌های خودران چطور رانندگی رو یاد می‌گیرن؟ ورد اصلی برای آموزش دادن به این هوش‌های مصنوعی، پایتونه. یعنی جادوگرهای دنیای تکنولوژی، با استفاده از پایتون به این ماشین‌ها یاد میدن که چطور فکر کنن، الگوها رو تشخیص بدن و تصمیم بگیرن. در واقع، پایتون زبان اصلی برای ساختن و تربیت کردن مغز متغیریه که پشت این تکنولوژی‌های شگفت‌انگیز قرار داره.

- **توسعه وب (Web Development)** : اون بخش‌های پنهان و مغز متغیر

سایتها و اپلیکیشن‌های معروفی مثل اینستاگرام، یوتیوب و اسپاتیفای با پایتون

ساخته شده. با پایتون می‌توانی قلعه‌های دیجیتالی خودت رو بسازی و بهشون قدرت بدی. (البته سایت‌های بزرگ از زبان‌های دیگه هم استفاده کردن!)

- علم داده (Data Science) : امروز، داده‌ها مثل گنج‌های پنهان هستن. پایتون بهترین بیل و کلنگ برای کشف این گنج‌هاست. باهش می‌توان حجم عظیمی از اطلاعات رو تحلیل کنی، الگوهای عجیب و غریب پیدا کنی و آینده رو پیش‌بینی کنی!

### حرف آخر:

انتخاب پایتون مثل انتخاب یک چاقوی سوئیسی برای شروع ماجراجوییه. ، تو با یک ابزار ساده، قدرتمند و همه‌کاره شروع می‌کنی که هم راه رو برات باز می‌کنه و هم تو رو به هر مقصدی که بخوای می‌رسونه.

پس کوله‌پشتیت رو آماده کن، چون قراره با این زبان جادویی، اولین قدم‌ها رو برای تبدیل شدن به یک ماجراجوی واقعی در دنیای تکنولوژی برداریم! 

## آماده کردن کوله‌پشتی: نصب پایتون یا استفاده از آزمایشگاه آنلاین

هر ماجراجویی به ابزار نیاز داره. ابزار اصلی ما، خود زبان پایتونه. برای استفاده از قدرت پایتون، دو راه اصلی پیش روی ماست: یکی اینکه کارگاه شخصی خودمون رو بسازیم (نصب پایتون روی کامپیوتر و یا لپ تاپ خودمون) و دومی اینکه از یک آزمایشگاه آنلاین جادویی و آماده استفاده کنیم (پایتونی که به صورت آنلاین ارائه می‌شه مناسب بچه‌هایی هست که سیستم قوی ندارن و یا از گوشی و تبلت استفاده می‌کنن).

انتخاب با توجه، ماجراجو!

### مسیر اول: ساخت کارگاه شخصی (نصب پایتون روی کامپیوتر و لپتاپ)

این مسیر برای کساییه که دوست دارن همه‌چیز تحت کنترل خودشون باشه و ابزارهاشون رو روی سیستم شخصی‌شون داشته باشن.

### قدم اول: نصب مترجم پایتون

#### ۱. برای کاربران ویندوز Windows

به وب‌سایت رسمی پایتون python.org برو. اونجا آخرین نسخه پایتون منتظرته. روی دکمه "Downloads" کلیک کن و فایل نصب (installer) رو برای ویندوز دانلود کن.

مهم‌ترین طلسمن نصب: فایل نصبی رو که اجرا کردی، به یک صفحه نصب می‌رسی. خیلی خیلی مهمه که در همون صفحه اول، تیک گزینه‌ی **Add Python to PATH** رو بزنی. این کار مثل اینه که به سیستم عاملت بگی "هی، من یه ابزار جدید به اسم پایتون دارم، حواست بهش باشه!". اگه این تیک رو نزنی، بعداً برای اجرای کدهات به مشکل می‌خوری پس خیلی مهمه. (اگه این کار رو انجام ندی زمانی که کدهای پایتون رو اجرا کنی بہت ارور اینکه پایتون رو نمی‌شناسم میده)

نصب نهایی: حالا روی Install Now کلیک کن و منتظر بمون تا جادو کامل بشه. تمام! کارگاه شخصی تو آماده‌ست.

## ۲. برای کاربران مک (The Mac Guild)

کاربران مک هم می‌توانند به سادگی از نصب‌کننده رسمی پایتون استفاده کنند تا جدیدترین ابزارها را در اختیار داشته باشند.

سفر به منبع اصلی: به وبسایت رسمی پایتون [python.org](https://python.org) برو و وارد بخش "Downloads" شو. وبسایت به طور هوشمند سیستم‌عامل مک تو را تشخیص می‌دهد.

دربیافت ابزار: روی دکمه دانلود آخرین نسخه برای macOS کلیک کن. یک فایل با پسوند `dmg` یا `pkg`. دانلود خواهد شد.

اجرای جادوی نصب: فایل نصب کننده را باز کن. این کار یک پنجره نصب استاندارد مک را باز می‌کند. مراحل بسیار ساده است؛ فقط کافیست روی دکمه‌های Continue, Agree و در نهایت Install کلیک کنی. در این مرحله، مک از تو رمز عبور سیستمت را می‌خواهد تا اجازه نصب را صادر کند.

پایان نصب: پس از چند لحظه، نصب‌کننده کارش را تمام می‌کند و پایتون به صورت کامل روی سیستم تو نصب می‌شود. حالا کارگاه شخصی تو روی مک هم آماده است!

## ۳. برای کاربران لینوکس (The Linux Clan)

لینوکس، سرزمین مورد علاقه کدنویس‌هاست و پایتون معمولاً جزئی از خون این سیستم‌عامله. اما باید مطمئن بشیم که جدیدترین نسخه رو داریم.

چک کردن نسخه: ترمینال رو باز کن و این دستور رو تایپ کن: `python3 --version`. اگه نسخه‌ای که نشون می‌ده جدیده (مثلاً ۳.۶ به بالا)، کارت راحته!

آپدیت کردن ابزار: اگه نسخهٔ قدیمیه یا اصلاً نصب نیست، با توجه به توزیع لینوکست (مثل اوبونتو، فدورا و...)، با یک دستور ساده آپدیتیش کن. برای اکثر سیستم‌های مبتنی بر دبيان (مثل اوبونتو)، این دستور کافیه:

```
sudo apt update
sudo apt install python3
```

تمام! تو آماده‌ای.

## قدم دوم: آماده کردن کتاب جادو (محیط توسعه (VS Code

حالا که مترجم جادویی (پایتون) را نصب کردیم، به یک "کتاب جادوی هوشمند" برای نوشتن وردها و طلسم‌هایمان (کدها) نیاز داریم. بهترین انتخاب برای این کار، Visual Studio Code (یا به اختصار VS Code) است.

فکر کن VS Code یک دفترچه خفنه که کلمات جادویی رو برات رنگی می‌کنه، اگه وردی رو اشتباه بنویسی زیرش خط می‌کشه و کمک می‌کنه کدهات رو مثل یک ماجراجوی حرفه‌ای، تمیز و مرتب نگه داری.

دانلود کتاب جادو: به وبسایت رسمی [code.visualstudio.com](https://code.visualstudio.com) برو. سایت به طور خودکار سیستم‌عامل تو (ویندوز، مک یا لینوکس) را تشخیص می‌دهد. روی دکمه بزرگ دانلود کلیک کن.

نصب ساده: فایل دانلود شده را اجرا کن و مراحل نصب را که بسیار ساده است، دنبال کن.

### قدم سوم : قدرتمند کردن کتاب جادو با افزونه پایتون

برای اینکه کتاب جادوی ما زبان پایتون را به بهترین شکل بفهمد، باید یک افزونه (Extension) به آن اضافه کنیم:

VS Code را باز کن. از نوار کناری سمت چپ، روی آیکونی که شبیه چند مربع است کلیک کن (بخش Extensions). در کادر جستجو، کلمه Python را تایپ کن. اولین گزینه‌ای که توسط Microsoft ارائه شده را پیدا کن و روی دکمه آبی رنگ Install کلیک کن.

تبیریک! حالا کارگاه شخصی تو با بهترین ابزارها آماده‌ی ماجراجویی است.

### مسیر دوم: استفاده از آزمایشگاه جادویی آنلайн (Google Colab)

اگه حوصله نصب و درگیری با سیستم عامل رو نداری، یا با گوشی و تبلت این ماجراجویی رو دنبال می‌کنی، یا سیستم خیلی قوی نیست، این مسیر برای تو ساخته شده!

گوگل کولب (Google Colab) یک دفترچه یادداشت آنلайн و هوشمند که پایتون از قبل روش نصب شده و فقط منتظر دستورات جادویی توانه.

چرا خفنه؟

بدون نیاز به نصب: فقط با یک مرورگر و اکانت گوگل (رایگان) کارت راه می‌افته. همیشه در دسترس: از هرجایی (کافه، کتابخونه، اتوبوس) و با هر وسیله‌ای (لپتاپ، تبلت، گوشی) به کدهات دسترسی داری.

قدرت ابری: انگار داری با کامپیوترهای قدرتمند گوگل کار می‌کنی، پس نگران ضعیف بودن سیستم نباش. (عملاً کدهات روی سرورهای گوگل اجرا می‌شوند)

چطور شروع کنیم؟

فقط کافیه به وب‌سایت [colab.research.google.com](https://colab.research.google.com) بروی، با اکانت گوگلت وارد بشی و روی New notebook کلیک کنی. یک صفحه جدید باز می‌شه که می‌تونی اولین کدهای پایتونت رو همونجا بنویسی و اجرا کنی. به همین سادگی!

خب ماجراجو، کوله‌پشتیت رو انتخاب کردی؟ ابزارت رو آماده کن که می‌خوایم اولین ورد جادویی رو اجرا کنیم!

### اولین ورد جادویی: اجرای print("Hello, World!") و دیدن نتیجه!

تبریک ماجراجو! کوله‌پشتیات آماده است و ابزارهایت را انتخاب کرده‌ای. اکنون زمان آن رسیده که اولین ورد جادویی خود را یاد بگیری و اولین طلسما را در دنیای کدنویسی اجرا کنی. این لحظه، لحظه‌ای فراموش‌نشدنی است؛ لحظه‌ای که برای اولین بار به ماشین دستور می‌دهی و او به تو پاسخ می‌دهد.

این ورد جادویی، `print` نام دارد. کارش خیلی ساده است: هر چیزی که تو داخل پرانتز و بین دو علامت نقل قول (" ") به او بدهی، روی صفحه نمایش حک می‌کند. (`print()` مثل

بایاید با هم این طلسما را اجرا کنیم!

اگر از مسیر اول (کارگاه شخصی و **VS Code**) می‌آیی: (برای بچه‌های **colab** بخش جدایی رو در ادامه توضیح میدم)

برای اینکه ماجراجویی‌مان مرتب باشد، اول یک قلمرو برای پروژه‌هایمان می‌سازیم.

ساخت قلمرو پروژه: روی دسکتاپ یا هر جای دیگری که دوست داری، یک پوشه (Folder) جدید بساز و اسم آن را پروژه بگذار. تمام گنجها و نقشه‌های ما در این قلمرو نگهداری خواهد شد.

باز کردن کتاب جادو (VS Code): برنامه VS Code را که قبلاً نصب کردی، باز کن.

فرآخوانی قلمرو: از منوی بالا، روی File کلیک کرده و گزینه‌ی Open Folder را انتخاب کن. حالا به همان پوشه‌ی پروژه که ساختی برو و آن را انتخاب کن. با این کار، VS Code تمام تمرکزش را روی قلمرو ماجراجویی تو می‌گذارد.

ساخت اولین طومار (فایل پایتون): در سمت چپ VS Code، جایی که اسم پوشه‌ی پروژه را می‌بینی، یک آیکون کوچک شبیه به یک برگه با علامت + وجود دارد (New File). روی آن کلیک کن و اسم فایل جدید را project.py بگذار. پسوند .py به کتاب جادوی ما می‌گوید که این یک طومار پایتون است! (حتماً پسوند فایل را .py، بزاراین خیلی مهمه! و اگه نزاری دیگه code vs فایلت رو درست نمیشناسه)

نوشتن ورد جادویی: حالا در صفحه‌ی سفیدی که باز شده، اولین ورد جادویی خود را با دقت تایپ کن (خیلی به علامت‌های " و فاصله‌ها و ... دقت کن چون توی برنامه نویسی اگه حتی یک فاصله رو اشتباه بزاری کدت کار نمیکته):

```
1. print("Hello, World!")
```

اجرای طلسم! وقت هیجان‌انگیز ماجرا فرا رسیده. باید "ترمینال" یا همان اتاق فرمان را باز کنیم تا ورد خود را در آن بخوانیم.

از منوی بالای VS Code، روی Terminal کلیک کن و New Terminal را انتخاب کن.

راه میانبر و سریع‌تر: می‌توانی دکمه‌های ترکیبی Ctrl + ~ (کلید ~ معمولاً کنار عدد ۱ روی کیبورد است) را فشار دهی تا ترمینال فوراً باز شود. در مک، این میانبر Cmd + ~ است.

خواندن دستور نهایی: در ترمینال که پایین صفحه باز شده، این دستور را تایپ کن و کلید Enter را بزن (به فاصله و املای کلمات دقت کن و همچنین اگه اسم فایل پایتونت رو چیز دیگه‌ای گذاشتی اون رو اینجا بنویس و به صورت کامل با پسوند .py):

```
python project.py
```

(نکته برای ماجراجویان مک و لینوکس: بجای دستور بالا آن python3 project.py را امتحان کنید).

و ناگهان... جادو اتفاق می‌افتد!

درست در خط بعدی ترمینال، این پیام ظاهر می‌شود:

```
Hello, World!
```

تبریک! تو همین الان اولین کد خود را اجرا کردی. تو به کامپیوتر دستور دادی و او از تو اطاعت کرد. این اولین قدم تو برای تبدیل شدن به یک جادوگر قدرتمند در دنیای کدنویسی است. این لحظه را جشن بگیر!

اگر از مسیر دوم (آزمایشگاه جادویی Google Colab) می‌آیی: کار تو بسیار ساده‌تر است! آزمایشگاه آنلاین همه چیز را برایت آماده کرده.

ورود به آزمایشگاه: در همان دفترچه یادداشت جدیدی که در Colab باز کردی، یک سلول کد خاکستری رنگ با یک دکمه "پلی" (▶) در کنارش می‌بینی.

نوشتن ورد جادویی: داخل این سلول، ورد جادویی را تایپ کن:

```
1. print("Hello, World!")
```

اجرای طلسما: حالا کافیست روی همان دکمه "پلی" (▶) کلیک کنی یا کلیدهای Shift + را با هم فشار دهی.

بالا فاصله زیر سلول کد، نتیجه‌ی جادوی خود را خواهی دید:

Hello, World

فوق العاده بود، نه؟ تو اولین پیام خود را از دل ابرها به دنیا فرستادی! این قدرت آزمایشگاه جادویی است. (منظور از ابرها سرورهای ابری گوگل هست)

ماجراجویی رسماً آغاز شد! تو اولین طلسما را با موفقیت اجرا کردی. حالا بریم یک پروژه باحال باهم بزنیم!

## پروژه: ساخت کارت ویزیت دیجیتال

ماجراجوی عزیز، تو اولین ورد جادویی را با موفقیت اجرا کردی و به دنیا سلام دادی! حالا وقت آن است که با استفاده از همین طلسم قدرتمند، هویت خودت را به "انجمان ماجراجویان پایتون" اعلام کنی. ما یک کارت ویزیت دیجیتال خواهیم ساخت تا همه بدانند با چه قهرمان جدیدی آشنا شده‌اند.

قبل از ساختن کارت، بیا کمی عمیق‌تر به جادویی که استفاده کردیم نگاه کنیم.

ورد (`print`) و علامت‌های نقل قول ("") برای چیست؟

`()`: فرمان نمایش

فکر کن `()` یک فرمان مستقیم به غول چراغ جادوی تو (یعنی کامپیوتر) است. وقتی می‌گویی `print`, در واقع داری به او دستور می‌دهی: "هی، چیزی که بہت می‌گم رو بگیر و روی صفحه نمایش بده!". این یک طلسم برای نمایش دادن است.

"": مرز دنیای جادو و دنیای واقعی

غول چراغ جادو باید بداند که حرف‌های تو دقیقاً چیست. علامت نقل قول ("") یک مرز جادویی ایجاد می‌کند. هر چیزی که داخل این علامت‌ها باشد، یک "رشته" (String) نامیده می‌شود. پایتون به محتوای داخل رشته‌ها دست نمی‌زند و آن را دقیقاً همانطور که هست، نمایش می‌دهد. اگر این علامت‌ها را نگذاریم، پایتون فکر می‌کند که کلمات داخل پرانتز، ورد‌های جادویی دیگری هستند و گیج می‌شود!

پایتون چطور کدهای ما را می‌خواند؟ مثل یک آبشار!

یک نکته خیلی مهم در مورد پایتون این است که کدها را مثل یک آبشار، از بالا به پایین می‌خواند و اجرا می‌کند. او از خط اول شروع می‌کند، دستورش را اجرا می‌کند، سپس به

خط دوم می‌رود، آن را اجرا می‌کند و همینطور تا آخر ادامه می‌دهد. این ترتیب خیلی مهم است، چون به ما اجازه می‌دهد داستان ماجراجویی خود را مرحله به مرحله تعریف کنیم.

آماده برای ساخت کارت ویزیت؟

حالا که رازهای print() و ترتیب اجرای کدها را می‌دانی، بیا کارت ویزیت خودت را بسازیم.

کد زیر را در فایل project.py خود (یا در یک سلوول جدید در Google Colab) بنویس.  
یادت باشد که اطلاعات داخل علامت‌های نقل قول را با مشخصات خودت عوض کنی!

اجرا کن و نتیجه را ببین!

```
1. # --- Adventurer's ID Card ---
2.
3. # Line 1: Displays the adventurer's name
4. print("Name: [Enter your name here]")
5.
6. # Line 2: Your chosen title
7. print("Title: Python World Explorer")
8.
9. # Line 3: A way for others to contact you
10. print("Contact: adventurer@example.com")
11.
12. # Line 4: Your personal motto for this journey
13. print("Motto: Ready to solve the first puzzle!")
14.
15. print("=====")
```

در هر ماجراجویی، یادداشت‌برداری برای به خاطر سپردن نکات مهم ضروری است. در دنیای کدنویسی، ما این یادداشت‌ها را با استفاده از علامت # (هشتگ) می‌نویسیم. هر چیزی که بعد از علامت # در یک خط نوشته شود، "کامنت" نام دارد. پایتون به طور کامل کامنت‌ها را نادیده می‌گیرد و آن‌ها را اجرا نمی‌کند. کامنتم‌ها فقط برای ما ماجراجویان هستند تا به خودمان (یا به همتیمی‌هایمان) یادآوری کنیم که یک طلسم (کد) خاص چه کاری انجام می‌دهد، یا چرا آن را به این شکل نوشته‌ایم. فکر کن داری در حاشیه کتاب جادوی خودت، نکات مهم را یادداشت می‌کنی تا بعداً فراموششان نکنی!

وقتی این کد را اجرا کنی (با دستور `python project.py` در ترمینال یا دکمه در Colab)، می‌بینی که پایتون دقیقاً به همان ترتیبی که نوشته‌ای، خط به خط اطلاعات تو را چاپ می‌کند و کارت ویزیت دیجیتال تو را می‌سازد.

آفرین! تو نه تنها اولین کد خود را اجرا کردی، بلکه اولین پروژه عملی خود را هم به پایان رساندی. این کارت ویزیت، مدرک ورود تو به دنیای شگفت‌انگیز کدنویسی است.  
ماجراجویی تازه شروع شده!

کارتت رو توی شبکه‌های اجتماعی با هشتگ `#codewithmahdi` با ما به اشتراک بزار.  
توی فصل بعد باهم یاد می‌گیریم کارهای باحال تری انجام بدیم!

# تبديل شدن به کارآگاه کد

مأموریت فصل: شکاراولین "باغ".

نقشه راه:

- ذهنیت یک ماجراجو: برنامهنویسی یعنی حل معما، نه حفظ کردن ورد!
- جعبه ابزار کارآگاه:
- Overflow Stack: کتابخانه بزرگ دانش تمام جادوگران دنیا.
- دستیارهای هوشمند(AI): چطور از غول های چراغ جادو Gemini, ChatGPT
- کمک بگیریم؟
- هنر پرسیدن سوال خوب.
- پروژه: یک کد طلسمند شده به شما داده می شود . با ابزارهای جدیدتان، خطای را پیدا و آن را فیکس کنید.

## ذهنیت یک ماجراجو: حل معما، نه حفظ کردن ورد!

اولین و مهم‌ترین قانونی که هر ماجراجوی کهنه‌کاری می‌داند این است:

برنامه‌نویسی یعنی حل معما، نه حفظ کردن ورد.

بسیاری از تازه‌کارها فکر می‌کنند باید صدها دستور و طلسم جادویی مانند `print` را «حفظ» کنند. اما جادوی واقعی در این نیست. جادوی واقعی در «ذهنیت» توست.

وقتی کد تو کار نمی‌کند یا با یک پیام خطای ترسناک قرمز رنگ (که به زودی با آن‌ها آشنا می‌شویم) مواجه می‌شوی، نترس! این یک معما است. کامپیوتر، برخلاف انسان‌ها، موجودی کاملاً منطقی است. او دقیقاً کاری را می‌کند که به او می‌گویی. اگر نتیجه اشتباه است، یعنی ما دستور را به شکلی نامفهوم یا اشتباه به او داده‌ایم.

به این دستورات اشتباه، «باغ» یا «طلسم معیوب» می‌گویند. و مأموریت ما در این فصل، پیدا کردن (Debugging) یا «رفع طلسم» کردن آن‌هاست.

پس ذهنیت خود را آماده کن: تو یک کارآگاه هستی. هر خطای (Error) یک سرنخ است. هر باغ، یک چالش جذاب است که منتظر توست تا آن را حل کنی.

## جمعه ابزار کارآگاه

هر کارآگاهی به ابزار نیاز دارد. برای شکار باگ‌ها، لازم نیست همه‌چیز را از صفر اختراع کنیم. ما دو ابزار فوق العاده قدرتمند در «کوله پشتی» خود داریم که باید طرز استفاده از آن‌ها را یاد بگیریم.

### ۱. Stack Overflow: کتابخانه بزرگ دانش تمام جادوگران دنیا

تصور کن یک کتابخانه جادویی بی‌انتها وجود دارد که در آن، هر جادوگر و ماجراجویی که تا به حال با یک طلس معیوب روبرو شده، مشکل و راه حلش را در آن ثبت کرده است.

آن کتابخانه، Stack Overflow است.

این یک وب‌سایت پرسش و پاسخ برای برنامه‌نویسان است. به احتمال ۹۹٪، هر مشکلی که تو در این ماجراجویی با آن روبرو می‌شوی، قبل از صدھا نفر دیگر هم با آن روبرو شده‌اند و بهترین جادوگران دنیا به آن پاسخ داده‌اند. یادگیری «جستجو کردن» در این کتابخانه، اغلب مهم‌تر از حفظ کردن هر وردی است.

### ۲. دستیارهای هوشمند (AI): غول‌های چراغ جادوی جدید

در فصل اول با «غول چراغ جادوی» خودمان یعنی کامپیوتر آشنا شدیم که دستور print ما را اجرا می‌کرد. حالا تصور کن غول‌هایی وجود دارند که نه تنها دستورات ما را اجرا می‌کنند، بلکه می‌توانند به ما در نوشتن طلس‌های بهتر کمک کنند، کدهای ما را توضیح دهند و حتی باگ‌های آن را پیدا کنند!

ابزارهایی مثل Gemini (که من هستم!) و ChatGPT دقیقاً همین کار را می‌کنند. آن‌ها دستیارهای هوشمند تو هستند. می‌توانی کد معیوب خود را به آن‌ها نشان دهی و پرسی:

«کجای این طلسم اشتباه است؟» آن‌ها با صبر و حوصله به تو کمک خواهند کرد تا سرنخ را پیدا کنی.

## هنر پرسیدن سوال خوب

داشتن جعبه ابزار کافی نیست؛ باید بلد باشی چطور از آن استفاده کنی.

چه در Stack Overflow و چه از یک دستیار هوشمند، «خوب پرسیدن» یک هنر است. یک ماجراجوی تازه‌کار فریاد می‌زند: «کمک! طلسم کار نمی‌کند!» و هیچ‌کس نمی‌تواند به او کمک کند.

اما یک کارآگاه حرفه‌ای می‌گوید: «من در حال اجرای یک طلسم print در فایل project.py هستم. انتظار داشتم کارت ویزیت دوستم را چاپ کنم، اما در عوض با این پیام خطای قرمز مواجه شدم: SyntaxError: EOL while scanning string literal. این هم کدی که نوشته‌ام. به نظر تان مشکل از کجاست؟»

همیشه سه چیز را در سوالات بیاور:

۱. چه کار می‌خواستی بکنی؟ (هدف تو چه بود؟)
۲. چه مشکلی پیش آمد؟ (بیام خطای دقیقاً چه بود؟ آن را کپی کن.)
۳. کدی که نوشته‌ی چه بود؟ (کد خودت را نشان بده.)

با این سه سرنخ، هر کسی می‌تواند به تو در شکار باگ کمک کند.

## پروژه: شکار باگ در کد طلسمنشده

آماده‌ای اولین شکارت را انجام دهی، کارآگاه؟

در زیر، یک «طومار پایتون» به تو داده شده که قرار بود «کارت ویزیت» ماجراجوی دیگری را چاپ کند، اما طلس آن توسط یک باگ معیوب شده است. آن را در فایل project.py خودت (یا در یک سلول جدید در Google Colab) کپی کن و سعی کن آن را اجرا کنی.

کد طلسمنشده:

```
1. # The Cursed ID Card
2. print("Name: Mahdi ghaemi")
3.
4. print("Title: World Ender"
5.
6. print("Motto: "Brave and Bold!")
7.
8. print(=====)
```

حالا، کلاه کارآگاهیات را بگذار و شروع کن:

۱. اجرای طلس: کد را اجرا کن. به پیام خطای قرمزی که پایتون به تو نشان می‌دهد در «ترمینال» (یا زیر سلول Colab) به دقت نگاه کن. این اولین سرنخ توست!

۲. پیدا کردن سرنخ: پایتون معمولاً به تو می‌گوید که در کدام خط (line) به مشکل خورده است. به آن خط برو.

۳. بررسی مظنون‌ها: با استفاده از دانشی که از فصل ۱ داری (به خصوص در مورد ورد print و «مرزهای جادویی» ""), سعی کن مشکل را پیدا کنی. \* آیا همه‌ی پرانتزها باز و بسته شده‌اند؟ \* آیا همه‌ی «مرزهای جادویی» (علامت‌های نقل قول) به درستی جفت

شده‌اند؟ \* آیا پایتون چیزی را که تو فکر می‌کنی متن است، به عنوان یک ورد جادویی دیگر می‌شناسد؟

۴. استفاده از جعبه ابزار: اگر گیر کردی، از دستیار هوشمند (ChatGPT) بپرس یا کلمات کلیدی پیام خطا (مثلاً SyntaxError) را در «کتابخانه» Stack Overflow جستجو کن.

وقتی توانستی کد را طوری «رفع طلسماً» کنی که کارت ویزیت زیر به درستی و بدون هیچ پیام خطای قرمزی چاپ شود، تو رسماً اولین باگ خود را شکار کرده‌ای!

نتیجه نهایی (کد رفع طلسماً شده):

```
1. # The Fixed ID Card
2. print("Name: Mahdi ghaemi")
3. print("Title: World Ender")
4. print("Motto: Brave and Bold!")
5. print("=====")
```

تبیریک می‌گوییم، کارآگاه! تو با موفقیت از ذهنیت حل مسئله و ابزارهای استفاده کردی. «کوله پشتی» تو برای ماجراجویی بعدی سنگین‌تر و ارزشمندتر شد.

امیدوارم قدر این جعبه ابزار رو بدونی چون در گذشته به این خوبی وجود نداشتند!

در فصل بعد، ابزارهای جادویی جدیدی برای « تقسیم غنائم » (کار با اعداد) خواهیم ساخت!

# فصل ۳: ساخت اولین ابزار: ماشین حساب جادویی

مأموریت فصل: ساخت یک ماشین حساب جادویی برای محاسبه و تقسیم غنائم بین اعضای گروه (با استفاده از اعدادی که خودمان مشخص می‌کنیم).

## نقشه راه

- جعبه‌های جادویی (متغیرها): چطور غنائم و اطلاعات را در حافظه نگه داریم؟
- آشنایی با غنائم: انواع داده‌های پایه (Integers, Floats, Strings, Booleans).
- جادوی ریاضی: کار با عملگرهای  $+$ ,  $-$ ,  $*$  و  $/$  روی اعداد.
- جادوی متن: چطور طلس + برای متن‌ها (Strings) متفاوت عمل می‌کند؟
- پروژه: ساخت یک ماشین حساب که غنائم (اعداد ثابت) را محاسبه و نتایج را چاپ می‌کند.

## ۱. جعبه‌های جادویی (متغیرها)

وقتی در ماجراجویی خود یک «سکه طلا» یا یک «معجون سلامتی» پیدا می‌کنی، آن را روی زمین رها نمی‌کنی تا گم شود! تو آن را در «کوله پشتی» یا یک «جعبه» امن می‌گذاری تا بعداً از آن استفاده کنی.

در برنامه‌نویسی، به این جعبه‌های جادویی «متغیر» (Variable) می‌گوییم.

متغیرها، مکان‌هایی در **حافظه (RAM)** کامپیوتر هستند که ما داده‌هایمان را در آن‌ها ذخیره می‌کنیم. این کار باعث می‌شود بتوانیم بعداً به آن داده‌ها دسترسی داشته باشیم. این داده‌ها موقتی هستند و وقتی برنامه تمام شود (یا کامپیوتر خاموش شود)، از بین می‌روند.

ساختن یک متغیر مثل برچسب زدن به یک جعبه است. ما از علامت = (که به آن عملگر تخصیص می‌گوییم) برای ریختن چیزی در جعبه استفاده می‌کنیم:

```
1. # "Mahdi Ghaemi" is the content inside the box
2. # adventurer_name is a label
3. adventurer_name = "Mahdi Ghaemi"
4.
5. # gold_coins is a label
6. # 50 is the content inside the box
7. gold_coins = 50
```

حالا، هر وقت که بخواهیم، می‌توانیم محتوای داخل جعبه را ببینیم:

```
1. # Let's make some magic boxes
2. adventurer_name = " Mahdi Ghaemi"
3. gold_coins = 50
4. health_points = 100
5.
6. # Now let's print the content of the boxes
7. print("Adventurer's Name:")
8. print(adventurer_name) # Python prints the box content, not its name
```

```
9.  
10. print("Gold Coins:")  
11. print(gold_coins)  
12.  
13. print("Health Points:")  
14. print(health_points)
```

اجرا کن و نتیجه را ببین! وقتی این کد را اجرا کنی، کامپیوتر به جای چاپ کلمه "gold\_coins"، می‌رود و محتوای داخل آن جعبه (یعنی ۵۰) را چاپ می‌کند.

### قدرت متغیرها: تغییرپذیری

جادوی واقعی متغیرها این است که محتوای آن‌ها می‌تواند تغییر کند.

```
1. # At the start of the adventure, we have 10 coins  
2. gold_coins = 10  
3. print("Current gold coins:")  
4. print(gold_coins)  
5.  
6. # We defeat a giant and get 20 new coins  
7. # We replace the old content of the box with the new value  
8. gold_coins = 20  
9. print("Gold coins after defeating the giant:")  
10. print(gold_coins)  
11.  
12. # We can even do calculations on the box itself  
13. # We spend 5 coins  
14. gold_coins = gold_coins - 5  
15. print("Remaining gold coins:")  
16. print(gold_coins) # Output will be 15
```

این قابلیت برای ماجراجویی ما حیاتی است!

## ۲. آشنایی با غنائم: انواع داده‌های پایه (Primitive Types)

«جعبه‌های» جادویی تو (متغیرها) می‌توانند انواع مختلفی از «غنائم» (داده‌ها) را در خود نگه دارند. اما یک قانون بزرگ در این ماجراجویی وجود دارد: نوع غنیمت، کاربرد آن را مشخص می‌کند.

فکرش را بکن: در یک ماجراجویی واقعی، تو نمی‌توانی یک «شمشیر» (که مثل متن است) را به «سکه‌های طلا» (که مثل عدد است) «اضافه» کنی. تو نمی‌توانی یک «طومار جادویی» (متن) را بر «تعداد اعضای گروه» (عدد) تقسیم کنی. هر غنیمتی، جادوی خاص خودش را دارد.

به این غنائم اصلی، «داده‌های پایه» (Primitive Types) می‌گویند. این‌ها بنیادی‌ترین مقادیری هستند که برای نمایش داده‌های اساسی به کار می‌روند و در خود زبان پایتون «ساخته» شده‌اند. غول چراغ جادوی تو ذاتاً می‌فهمد که «سکه» (عدد) چیست و «طومار» (متن) چیست و لازم نیست تو این‌ها را به او یاد بدھی.

در ماجراجویی پایتون، ما با چهار نوع غنیمت اصلی سروکار داریم:

## سکه‌های طلا (Integer)

این‌ها اعداد کامل و صحیح هستند. می‌توانند مثبت، منفی یا صفر باشند. هر وقت چیزی را می‌شماری (تعداد تیر، تعداد دشمن، تعداد قدم) از این نوع استفاده می‌کنی.

- 1. age = 21
- 2. score = -20
- 3. count = 0

## گُرد جادویی (Float)

این‌ها اعداد اعشاری هستند؛ مثل معجون‌هایی که می‌توانند خُرد شوند. هر وقت به «دقت» نیاز داری (مثل محاسبه میزان سم یا درصد سلامتی) از این نوع استفاده می‌کنی.

- 1. pi = 3.14
- 2. temp = -273.0
- 3. percent = 99.9

## طومارهای باستانی (String)

این‌ها «متن» هستند. هر چیزی که داخل علامت نقل قول (" یا ') قرار بگیرد، یک متن یا است. نام‌ها، پیام‌ها، وردها و آدرس‌ها همه از این نوع هستند. String

```
1. name = "mahdi"  
2. message = "Oppsi"
```

## کلیدهای جادویی (Boolean)

این‌ها فقط دو حالت دارند: «روشن» یا «خاموش». True (درست) یا False (نادرست). این کلیدها برای تصمیم‌گیری استفاده می‌شوند. (آیا در باز است؟ True. آیا غول زنده است؟ False). ما در فصل بعد خیلی بیشتر از این کلیدها استفاده خواهیم کرد.

```
1. is_up = True  
2. is_down = False
```

## ۳. جادوی ریاضی: کار با عملگرها

حالا که «سکه‌های طلا» (Integers) و «گرد جادویی» (Floats) را می‌شناسیم، بیا با آن‌ها محاسبات انجام دهیم. پایتون چهار عملگر اصلی ریاضی را بلد است:

+ (جمع)

- (تفریق)

\* (ضرب)

/ (تقسیم)

بیا این‌ها را با متغیرهایمان امتحان کنیم:

```

1. # We found the loot
2. gold_coins_chest1 = 50
3. gold_coins_chest2 = 75
4. adventurers_count = 4
5.
6. # --- Performing the magic of math ---
7.
8. # 1. Add the loot from both chests
9. total_gold = gold_coins_chest1 + gold_coins_chest2
10.
11. # 2. Divide the loot among the party members
12. gold_per_adventurer = total_gold / adventurers_count
13.
14. # --- Displaying the results ---
15. print("Total gold found:")
16. print(total_gold) # Output: 125
17.
18. print("Share per adventurer:")
19. print(gold_per_adventurer) # Output: 31.25

```

نکته مهم: دیدی که حاصل تقسیم ۱۲۵ (Integer) بر ۴ (Float) شد ۳۱.۲۵؟ در پایتون، عملگر / همیشه نتیجه را به صورت «گرد جادویی» (Float) برمی‌گرداند تا هیچ بخشی از غنائم در تقسیم از بین نرود!

#### ۴. جادوی متن: چسباندن طومارها

یک نکته بسیار مهم! عملگر + یک جادوی دوگانه دارد. وقتی با اعداد (Integers/Floats) استفاده شود، کار «جمع ریاضی» را انجام می‌دهد. وقتی با متن‌ها (Strings) استفاده شود، کار «چسباندن» (Concatenation) را انجام می‌دهد.

این دو مثال را مقایسه کن:

```

1. # The magic of math (adding numbers)
2. num1 = 20
3. num2 = 5
4. result_math = num1 + num2
5. print("Result of math magic:")

```

```

6. print(result_math) # Output: 25
7.
8. # The magic of text (concatenating strings)
9. text1 = "20" # This is a string, not a number
10. text2 = "5" # This is also a string
11. result_text = text1 + text2
12. print("Result of text magic:")
13. print(result_text) # Output: "205"

```

این تفاوت بسیار مهم است و در فصل‌های آینده که می‌خواهیم ورودی بگیریم، دوباره با آن روبرو خواهیم شد. فعلًاً فقط یادت باشد: پایتون بین «سکه» ۲۰ و «طومار» ۲۰ تفاوت قائل می‌شود!

## ۵. پروژه: ساخت ماشین حساب جادویی

وقت آن است که تمام وردهای جادویی که امروز یاد گرفتیم (variables, primitive types) را با هم ترکیب کنیم تا ابزار نهایی این فصل را بسازیم.

هدف: برنامه‌ای بنویس که دو عدد ثابت (که خودمان در کد می‌نویسیم) را بگیرد، هر چهار عمل اصلی (جمع، تفریق، ضرب، تقسیم) را روی آنها انجام دهد و نتیجه را به زیبایی چاپ کند.

کد پروژه در project.py بنویس (یا در یک سلول در Colab):

```

1. # --- Magic Calculator ---
2.
3. print("Welcome to the Magic Calculator!")
4. print("Today's numbers for calculation...")
5.
6. # --- Step 1: Define the loot (variables) ---
7. num1 = 150.5 # First treasure
8. num2 = 25.0 # Second treasure
9.
10. print("First number:")
11. print(num1)
12. print("Second number:")
13. print(num2)
14.
15. # --- Step 2: Perform math magic ---

```

```

16. sum_result = num1 + num2
17. subtract_result = num1 - num2
18. multiply_result = num1 * num2
19. divide_result = num1 / num2
20.
21. # --- Step 3: Display the results (Using + and str() casting) ---
22. print("====")
23. print("Your magical calculation results:")
24.
25. print("Sum: ")
26. print(sum_result)
27.
28. print("Subtraction: ")
29. print(subtract_result)
30.
31. print("Multiplication: ")
32. print(multiply_result)
33.
34. print("Division: ")
35. print(divide_result)
36.
37. print("====")
38. print("====")

```

اجرا کن و نتیجه را بیین! خروجی تو باید یک ماشین حساب کامل و زیبا باشد:

### نتایج جادویی محاسبات تو:

```

1. Welcome to the Magic Calculator!
2. Today's numbers for calculation...
3. First number:
4. 150.5
5. Second number:
6. 25.0
7. =====
8. Your magical calculation results:
9. Sum:
10. 175.5
11. Subtraction:
12. 125.5
13. Multiplication:
14. 3762.5
15. Division:
16. 6.02
17. =====

```

# فصل ۴: رام کردن منطق: ساخت اولین بازی حدس عدد

مأموریت فصل: ساخت بازی "حدس عدد" و به چالش کشیدن کامپیوتر.

نقشه راه

- صحبت با ماجراجو: گرفتن ورودی از کاربر با ورد جادویی (`input()`)
- تله‌ی کیمیاگری: تبدیل «طومار» ورودی به «سکه» با (`.int()`)
- طلسه‌های مقایسه: آشنایی با `==`, `>` و `<` برای گرفتن `True` یا `False`
- دوراهی جادویی: چطور برای کامپیوتر شرط بگذاریم؟ (آشنایی با `if` و `else`)
- مغز متفکر بازی: ساخت مسیرهای چندگانه با `elif` ("بزرگتره", "کوچکتره", "درسته").

- تکرار تا پیروزی: معرفی حلقه while برای دادن شанс مجدد.
- پروژه: پیاده‌سازی کامل بازی حدس عدد.

## ۱. صحبت با ماجراجو: ورد جادویی input()

در فصل ۳، ماشین حساب ما فقط با اعدادی کار می‌کرد که ما در کد نوشته بودیم. اما یک ابزار جادویی واقعی باید بتواند از کاربرش «سوال» پرسید.

ورد جادویی برای این کار input() است.

وقتی پایتون به این ورد می‌رسد، ماجراجویی را متوقف می‌کند و منتظر می‌ماند تا کاربر چیزی تایپ کند و Enter را بزند. (تا وقتی Enter وارد نشود منتظر می‌ماند)

```

1. # The spell to ask for a name
2. print("Greetings, adventurer! What is your name?")
3.
4. # 1. Python stops and waits...
5. # 2. The user types their name (e.g., "Aria") and hits Enter
6. # 3. The string "Aria" is stored in the 'adventurer_name' box
7. adventurer_name = input()
8.
9. print("Welcome, " + adventurer_name + "! Let's begin.")

```

اجرا کن و نتیجه را ببین! تو همین الان اولین برنامه تعاملی خودت را ساختی!

## ۲. تله‌ی کیمیاگری: تبدیل «String» به «Integer» با int()

حالا بیا ماشین حساب فصل ۳ را «تعاملی» کنیم.

```

1. # --- Interactive Calculator (The Trap!) ---
2. print("Enter the first number:")
3. num1_text = input() # User types: 20
4.
5. print("Enter the second number:")
6. num2_text = input() # User types: 5
7.
8. result = num1_text + num2_text
9. print("The result is:")
10. print(result)

```

اجرا کن و شوکه شو ! به جای 25، پایتون به تو 205 را نشان می‌دهد!  
این همان «تلهی جادوی متن» از فصل قبل است. ورد `input()` یک ورد لجوح است؛ او همه‌چیز را به شکل متن (`String`) برمی‌گرداند، نه عدد (`Number`). پایتون در واقع متن "20" را به متن "5" چسبانده است.

ما برای محاسبات ریاضی، به کیمیاگری (`Casting`) نیاز داریم. ما باید این طومارها را به سکه‌های طلا (`Integers`) تبدیل کنیم. ورد جادویی برای این کار `int()` است.

```
1. # --- Interactive Calculator (Fixed with int()) ---
2. print("Enter the first number:")
3. num1_text = input() # num1_text is "20"
4.
5. print("Enter the second number:")
6. num2_text = input() # num2_text is "5"
7.
8. # --- The Alchemy Step ---
9. num1_int = int(num1_text) # num1_int is 20 (Integer)
10. num2_int = int(num2_text) # num2_int is 5 (Integer)
11.
12. result = num1_int + num2_int
13. print("The result is:")
14. print(result) # Output is now 25. Hooray!
```

### ۳. طلسه‌های مقایسه و دوراهی جادویی if و else

حالا که می‌دانیم چطور از کاربر عدد بگیریم، چطور به کامپیوتر بگوییم با آن «تصمیم» بگیرد؟  
اول، باید کلیدهای جادویی (`Booleans`) را که در فصل قبل دیدیم، به یاد بیاوریم `True` و `False`. ورد `if` (اگر) به کامپیوتر می‌گوید: اگر این شرط `True` (درست) بود، این کار را بکن.

اما چطور `True` یا `False` بسازیم؟ با طلسه‌ای مقایسه:

• `==`: آیا دو چیز برابرند؟ (مراقب باش = برای ریختن در جعبه بود، `==` برای مقایسه است)

• `!=`: آیا دو چیز نابرابرند؟

- > آیا اولی بزرگتر از دومی است؟
- < آیا اولی کوچکتر از دومی است؟
- >= آیا بزرگتر یا مساوی است؟
- <= آیا کوچکتر یا مساوی است؟

بیا یک «نگهبان غار» بسازیم که فقط عدد جادویی را قبول می‌کند:

```

1. secret_number = 7
2.
3. print("Speak the magic number to enter the cave:")
4. guess = int(input()) # We cast to int immediately
5.
6. # The decision point
7. # Notice the colon (:) at the end and the indentation (tab)
8. if guess == secret_number:
9.     print("Correct! The cave opens.")
10.
11. # What if they are wrong?
12. else:
13.     print("Wrong! The cave remains sealed.")
14.
15. print("The adventure continues...")

```

اگر 7 را وارد کنی، نگهبان در را باز می‌کند. اگر هر عدد دیگری وارد کنی، ورد else (در غیر این صورت) اجرا می‌شود.

#### ۴. مغز متفسر بازی: مسیرهای چندگانه با elif

نگهبان ما کمی خنگ است. او فقط می‌گوید «اشتباه است». او به ماجراجویی کمک نمی‌کند. ما چطور می‌توانیم راهنمایی کنیم که حدهش «بزرگتره» یا «کوچکتره»؟ ما به بیش از دو مسیر (درست/غلط) نیاز داریم. ما به یک سهراهی نیاز داریم. برای این کار، از elif (مخفف Else If) استفاده می‌کنیم.

درست است : if ... (اگر قبلی درست نبود) در غیر این صورت، اگر/ین یکی درست است : else ... (اگر هیچ کدام از بالایی ها درست نبودند) در نهایت این کار را بکن.

```
1. secret_number = 7
2.
3. print("Speak the magic number:")
4. guess = int(input())
5.
6. if guess == secret_number:
7.     print("Correct! The cave opens.")
8. elif guess < secret_number:
9.     print("Your guess is too small!")
10. else:
11.     # If it's not equal, and not smaller, it MUST be bigger
12.     print("Your guess is too big!")
```

حالا نگهبان ما بسیار باهوش تر شده و به ماجراجو راهنمایی می دهد!

## 5. تکرار تا پیروزی: حلقه while

یک مشکل بزرگ: بازی ما فقط یک شانس به ماجراجو می دهد و تمام می شود! این اصلاً عادلانه نیست. ما نیاز به وردی داریم که بگویید: تا زمانی که ماجراجو برنده نشده، به پرسیدن ادامه بده.

این ورد جادویی، while (تا زمانی که) است.  
حلقه while یک بلوک کد را بارها و بارها تکرار می کند، تا زمانی که شرط جلوی آن True باشد.

یا یک «کلید جادویی» (Boolean) «بسازیم تا وضعیت بازی را کنترل کند:

```
1. secret_number = 7
2. is_game_over = False # The game is not over yet (False)
3.
4. while is_game_over == False:
```

```

5.     # This whole block will repeat
6.     print("-----")
7.     print("Speak the magic number:")
8.     guess = int(input())
9.
10.    if guess == secret_number:
11.        print("Correct! The cave opens! You win!")
12.        is_game_over = True # We flip the switch to True to stop the
loop!
13.    elif guess < secret_number:
14.        print("Your guess is too small! Try again.")
15.    else:
16.        print("Your guess is too big! Try again.")
17.
18. # When the loop finally stops (is_game_over becomes True), this line
runs
19. print("Game Over. Thanks for playing!")

```

## ۶. پروژه: پیاده‌سازی کامل بازی حدس عدد

ماجراجو، تو آماده‌ای! ما تمام قطعات پازل را داریم `input`, `int`, `if`, `elif`, `else`, `while`: و `Boolean`.

بیا همه را کنار هم بگذاریم. برای این مأموریت، ما خودمان یک «عدد جادویی» ثابت را در کد مخفی می‌کنیم و از ماجراجوی دیگر می‌خواهیم آن را حدس بزنند.

کد پروژه نهایی در `project.py` بنویس:

```

1. # --- The Great Guessing Game ---
2.
3. # 1. Setup the Game
4. # YOU, the author, set the secret number here.
5. # Try changing it later!
6. secret_number = 14
7. is_game_over = False
8. guess_count = 0
9.
10. print("Welcome, Adventurer, to the Guessing Game!")
11. print("I have chosen a magic number between 1 and 20.")
12.
13. # 2. Start the Game Loop
14. while is_game_over == False:
15.     print("-----")
16.     print("What is your guess?")
17.
18.     guess_text = input()
19.

```

```

20.     # --- Alchemy and Safety Check ---
21.     # We must be careful! What if the user types "hello"?
22.     # The int() spell will crash!
23.     # (We learn to fix this in later chapters)
24.     # For now, we trust the adventurer to enter a number.
25.     guess_int = int(guess_text)
26.     guess_count = guess_count + 1 # Add 1 to the guess count
27.
28.     # 3. The Decision Logic
29.     if guess_int == secret_number:
30.         # We need str() to join text and numbers with '+'
31.         print("You got it! The magic number was " + str(secret_number))
32.         print("It took you " + str(guess_count) + " guesses.")
33.         is_game_over = True # Flip the switch to end the game
34.
35.     elif guess_int < secret_number:
36.         print("Your guess is too small! Try again.")
37.
38.     else: # It must be too big
39.         print("Your guess is too big! Try again.")
40.
41. # 4. End of Game
42. print("====")
43. print("Game Over. Well played!")

```

اجرا کن و نتیجه را بین ! کد را اجرا کن و سعی کن عدد جادویی را که خودت مخفی کرده‌ای پیدا کنی. بعداً می‌توانی عدد 14 را به هر عدد دیگری تغییر دهی و از دوستانت بخواهی آن را حدس بزنند.

آفرین! تو همین الان به غول چراغ جادو «منطق» آموختی. تو از یک آبشار ساده ، به یک سیستم تصمیم‌گیری هوشمند رسیدی و اولین بازی کامل خودت را ساختی. «کوله پشتی» تو سنگین‌تر از همیشه است!



# بخش دوم: مدیریت کوله

## پشتی ماجراجو

عالی! ماجراجویی ما تا اینجا فوق العاده بوده . 

تو در فصل ۴ «منطق» را رام کردی، به غول چراغ جادو قدرت «تصمیم‌گیری» دادی و اولین بازی خودت را ساختی . تو حالا با `if`, `while`, `input` و «کیمیاگری (int)» «کاملاً آشنا هستی .

اما تا به حال، «جعبه‌های جادویی» (متغیرهای) ما فقط می‌توانستند یک غنیمت را در خود نگه دارند (مثل `secret_number = 14`).  
اگر در یک ماجراجویی، ناگهان یک صندوقچه پر از آیتم‌های مختلف پیدا کنی چه؟ یک شمشیر، یک سپر، ۱۰ معجون، و یک نقشه... آیا می‌خواهی برای هر کدام یک متغیر جدا بسازی؟ `item2 = "shield"`... `item1 = "sword"`... این که خیلی نامرتب است!

ما به یک «کوله پشتی» جادویی نیاز داریم. یک جعبه‌ی بزرگ که بتواند همه‌ی غنائم دیگر را به صورت مرتب در خود نگه دارد.

وقت آن است که وارد بخش دوم: مدیریت کوله پشتی ماجراجو شویم .



# فصل ۵: مدیریت هوشمند

## کوله پشتی: لیست خرید

مأموریت فصل: ساخت یک برنامه برای مدیریت لیست خرید.

نقشه راه

- کوله پشتی جادویی (Lists): قدرتمندترین ابزار برای نگهداری آیتم‌ها.
- پر کردن کوله پشتی: اضافه کردن، دیدن، و حذف کردن آیتم‌ها.
- بازرسی کوله پشتی (طلسم‌های (in و len()
- سرشماری غنائم (حلقه for): ابزاری برای خواندن و مدیریت کوله پشتی.
- پروژه: ساخت برنامه مدیریت لیست خرید با قابلیت افروden و حذف آیتم.

## ۱. کوله پشتی جادویی (Lists)

تا به حال ما با «داده‌های پایه» (Primitive Types) مثل int و string کار می‌کردیم. اما حالا با اولین «داده‌ی غیرپایه» (Non-Primitive Type) آشنا می‌شویم: لیست (List).

لیست، دقیقاً مثل یک «کوله پشتی» است:

این یک مجموعه‌ی مرتب از آیتم‌هاست. (ترتیب آیتم‌ها مهم است). می‌تواند شامل انواع مختلف داده باشد (عدد، متن، یا حتی لیست‌های دیگر!). ساختن یک کوله پشتی (لیست) جدید، با استفاده از براکت [] انجام می‌شود:

```
1. # Our backpack is empty at the start of the adventure
2. # We can create an empty list like this:
3. backpack = []
4. print(backpack)
```

خروجی:

```
1. []
```

## ۲. پر کردن کوله پشتی (اضافه کردن، دیدن و حذف کردن)

یک کوله پشتی خالی به دردی نمی‌خورد. ما باید وردهای جادویی مدیریت آن را یاد بگیریم.

**وردها: append() : اضافه کردن آیتم**

برای اضافه کردن یک آیتم به انتهای کوله پشتی، از طلسنم .append() استفاده می‌کنیم:

```
1. # Let's create our shopping list for the journey
2. shopping_list = []
3. print("List at start:")
4. print(shopping_list)
5.
6. # Now, let's add items using the .append() spell
7. shopping_list.append("sword") #
8. shopping_list.append("health_potion")
9. shopping_list.append("bread")
10.
```

```
11. print("List after adding items:")
12. print(shopping_list)
```

خروجی:

```
1. List at start:
2. []
3. List after adding items:
4. ['sword', 'health_potion', 'bread']
```

## خواندن آیتم‌ها با «ایندکس» (Index)

چطور آیتم اول را ببینیم؟ ما از «ایندکس» (شماره موقعیت) استفاده می‌کنیم.

هشدار ماجراجو! در دنیای جادویی پایتون، شمارش از ۰ (صفر) شروع می‌شود، نه از ۱.

اولین آیتم در ایندکس [۰] است.

دومین آیتم در ایندکس [۱] است.

و الی آخر...

```
1. shopping_list = ["sword", "health_potion", "bread"]
2.
3. # Access items by their index number
4. first_item = shopping_list[0] #
5. second_item = shopping_list[1]
6.
7. print("First item needed:")
8. print(first_item) # Output: sword
9. print("Second item needed:")
10. print(second_item) # Output: health_potion
11.
12. # What about the LAST item?
13. # You can use -1 to get the last item, -2 for the second-to-last, etc.
14. last_item = shopping_list[-1] #
15. print("Last item:")
16. print(last_item) # Output: bread
```

## ورده (remove) : حذف کردن آیتم

اگر یک آیتم را خریدیم و بخواهیم آن را از لیست حذف کنیم چه؟ اگر نام آیتم را بدانیم، از طلسه (remove) استفاده می‌کنیم.

```

1. shopping_list = ["sword", "health_potion", "bread"]
2. print("Original list:")
3. print(shopping_list)
4.
5. # We found a health potion! Let's remove it from the list
6. shopping_list.remove("health_potion")
7.
8. print("List after removing potion:")
9. print(shopping_list)

```

خروجی:

```

1. Original list:
2. ['sword', 'health_potion', 'bread']
3. List after removing potion:
4. ['sword', 'bread']

```

## ۳. بازرسی کوله پشتی (طلسم‌های len و in)

گاهی اوقات ما نمی‌خواهیم آیتمی را اضافه یا حذف کنیم، بلکه فقط به «اطلاعات» در مورد کوله پشتی مان نیاز داریم. مثلاً:

«کوله پشتی من چقدر سنگین شده؟» (چند آیتم داخل آن است؟)

«آیا من قبلاً "شمشیر" را برداشته‌ام؟» (آیا آیتمی در لیست هست؟)

برای این دو سوال حیاتی، ما دو ورد جادویی جدید داریم:

## ورده (len) : طلسه شمارش گر

اگر بخواهی بدانی دقتاً چند آیتم در کوله پشتی‌ات (لیست) وجود دارد، از طلسه (len) (مخفف Length به معنی طول) استفاده می‌کنی.

```

1. shopping_list = ["sword", "health_potion", "bread"]
2. empty_list = []
3.
4. # Get the number of items in the list
5. item_count = len(shopping_list) #
6. empty_count = len(empty_list)
7.
8. print("Items in shopping list:")
9. print(item_count) # Output: 3
10.
11. print("Items in empty list:")
12. print(empty_count) # Output: 0

```

کاربرد در پروژه: این طلسم برای «چالش ۲» پروژه حیاتی است! ما می‌توانیم قبل از نمایش آیتم‌ها، با یک if چک کنیم: if `len(shopping_list) == 0`: تا اگر لیست خالی بود، پیام «لیست شما خالی است» را نمایش دهیم.

## ورد in: طلسم جستجوگر

این یکی از قدرتمندترین طلسم‌های پایتون است. اگر بخواهی چک کنی که آیا یک آیتم خاص «داخل» (in) کوله پشتیات هست یا نه، از این ورد استفاده می‌کنی. نتیجه این طلسم یک «کلید جادویی» (Boolean) یعنی True یا False است.

```

1. shopping_list = ["sword", "health_potion", "bread"]
2.
3. # Check if 'sword' is in the list
4. has_sword = "sword" in shopping_list
5. print("Do I have a sword?")
6. print(has_sword) # Output: True
7.
8. # Check if 'shield' is in the list
9. has_shield = "shield" in shopping_list
10. print("Do I have a shield?")
11. print(has_shield) # Output: False

```

کاربرد در پروژه: این طلسم، «طلسم ایمنی» ما برای «چالش ۳» (حذف آیتم) است. اگر ماجراجو بخواهد آیتمی را حذف کند که در لیست نیست، برنامه «کرش» می‌کند. ما قبل از

True، اول با if چک می‌کنیم: و فقط اگر item\_to\_remove in shopping\_list بود، آن را remove می‌کنیم.

#### ۴. سرشماری غنائم (حلقه for)

در فصل ۴، ما از حلقه while برای تکرار کد استفاده کردیم. while عالی بود چون تا زمانی که یک شرط True بود ادامه می‌داد.

اما برای «کوله پشتی» (لیست)، ما اغلب می‌خواهیم یک کار ساده انجام دهیم: «به ازای هر آیتم در کوله پشتی، آن را به من نشان بده.»

انجام این کار با while سخت است. اما ما یک ورد جادویی جدید و بسیار ساده‌تر برای این کار داریم: حلقه for.

حلقه for برای گشتن (iterate) روی یک «دبالة» (sequence) مثل لیست ساخته شده است.

```

1. shopping_list = ["sword", "health_potion", "bread"]
2.
3. print("--- Shopping List (one by one) ---")
4.
5. # The 'for' loop:
6. # "item" is a temporary variable we create.
7. # Python will put 'sword' in 'item', run the code.
8. # Then put 'health_potion' in 'item', run the code.
9. # Then put 'bread' in 'item', run the code.
10. for item in shopping_list: #
11.     print("I need to buy a " + item)
12.
13. print("-----")
```

خروجی:

```

1. --- Shopping List (one by one) ---
2. I need to buy a sword
3. I need to buy a health_potion
4. I need to buy a bread
5. -----
```

این حلقه بسیار تمیزتر از while است!

## ۵. پروژه: ساخت برنامه مدیریت لیست خرید

بیا یک برنامه مدیریت لیست خرید کامل بسازیم که از کاربر دستور می‌گیرد. ماجراجو، تو آماده‌ای. این پروژه، «آزمون بزرگ» توئه. در این مأموریت، ما قرار نیست طلسماً جدیدی یاد بگیریم؛ در عوض، می‌خواهیم تمام وردهای جادویی که در چهار فصل گذشته یاد گرفتیم رو مثل یک جادوگر کهنه‌کار با هم ترکیب کنیم و یک ابزار قدرتمند بسازیم.

این ابزار، «مدیر کوله پشتی» می‌خواهد بود.

نقشه و طرح اولیه (قبل از کدنویسی)

بیا قبل از نوشتن هر کدی، مثل یک معمار، «نقشه» ابزارمان را بکشیم. ما می‌خواهیم برنامه‌ای بسازیم که:

همیشه روشن باشد: برنامه نباید بعد از اجرای یک دستور تمام شود. باید مدام روشن بماند و منتظر دستور بعدی ما باشد. (کدام ورد جادویی از فصل ۴ این کار را برای ما انجام می‌داد؟)

دستور بگیرد: برنامه باید از ما بپرسد که چه کاری می‌خواهیم انجام دهیم. (کدام ورد جادویی از فصل ۴ از کاربر «سوال» می‌پرسید؟)

تصمیم بگیرد: برنامه باید بتواند بر اساس دستور ما (مثلًا "add", "show", "quit") تصمیم بگیرد که کدام بخش از جادو را اجرا کند. (کدام طلسماهی منطقی از فصل ۴ برای ایجاد «چندراهی» بودند؟)

یک حافظه داشته باشد: ما به یک «کوله پشتی» نیاز داریم که آیتم‌ها در آن ذخیره شوند. این کوله پشتی باید قبل از شروع حلقه بی‌نهایت ما ساخته شود تا با هر بار تکرار حلقه، حالی نشود. (کدام ابزار از فصل ۵ برای این کار عالی است؟)

### چالش‌های فکری (تمرین تو)

حالا بیا با هم فکر کنیم چطور هر بخش را می‌سازیم:

#### چالش ۱: طلسم "add" (اضافه کردن)

وقتی کاربر دستور "add" را وارد کرد، برنامه باید چه کار دومی انجام دهد؟ (راهنمایی: باید پرسد «چه آیتمی؟»)

بعد از اینکه نام آیتم را گرفت، با کدام ورد جادویی از این فصل (فصل ۵) آن را به «کوله پشتی» (لیست) اضافه می‌کند؟

#### چالش ۲: طلسم "show" (نمایش دادن)

وقتی کاربر دستور "show" را زد، ما باید تمام آیتم‌های داخل کوله پشتی را یک‌یکی نشان دهیم.

کدام حلقه جادویی از این فصل (فصل ۵) برای «سرشماری» و گشتن روی تمام آیتم‌های یک لیست عالی بود؟

#### چالش ۳: طلسم "remove" (حذف کردن)

این یکی کمی تله دارد! وقتی کاربر "remove" را می‌زند و نام آیتم را می‌گوید، ما از ورد استفاده می‌کنیم.

#### چالش ۴: طلس "quit" (خروج)

برنامه ما در یک حلقه بی‌نهایت while گیر کرده است. وقتی کاربر "quit" را تایپ می‌کند، ما چطور می‌توانیم آن «کلید جادویی» (متغیر Boolean) که حلقه را کنترل می‌کند، روی False تنظیم کنیم تا حلقه متوقف شود و برنامه به پایان برسد؟

وقت تمرین! قبل از اینکه به کد نهایی در صفحه بعد نگاه کنی، سعی کن خودت در فایل project.py، با استفاده از این چهار چالش و ابزارهایی که یاد گرفتی (input, list, append, remove, for).

حتی اگر کامل نشد نگران نباش! نفسِ تلاش برای حل این معما، تو را قوی‌تر می‌کند.

#### کد پروژه نهایی (در project.py بنویس):

```
1. # --- The Adventurer's Shopping List Manager ---
2.
3. # 1. Setup the backpack
4. shopping_list = []
5. is_running = True
6.
7. print("Welcome to the Shopping List Manager!")
8. print("You can 'add', 'remove', 'show', or 'quit'.")
9.
10. # 2. Start the main game loop (from Chapter 4)
11. while is_running == True:
12.     print("-----")
13.     # 3. Get command from the user (from Chapter 4)
14.     command = input("What do you want to do? ")
15.
16.     # 4. Use logic (from Chapter 4) to decide
17.     if command == "add":
18.         item_to_add = input("What item to add? ")
```

```

19.     shopping_list.append(item_to_add) # Use our new spell!
20.     print(item_to_add + " was added to the list!")
21.
22. elif command == "remove":
23.     item_to_remove = input("What item to remove? ")
24.     # We need to make sure the item is in the list first
25.     if item_to_remove in shopping_list:
26.         shopping_list.remove(item_to_remove) # Use our remove
spell!
27.         print(item_to_remove + " was removed.")
28.     else:
29.         print(item_to_remove + " is not in the list!")
30.
31. elif command == "show":
32.     print("--- Your Current List ---")
33.     # 5. Use our 'for' loop to show the items!
34.     if len(shopping_list) == 0: # len() gets the length of a list
35.         print("Your list is empty.")
36.     else:
37.         for item in shopping_list:
38.             print("- " + item)
39.     print("-----")
40.
41. elif command == "quit":
42.     print("Goodbye, adventurer!")
43.     is_running = False # This will stop the 'while' loop
44.
45. else:
46.     print("Invalid command. Please use 'add', 'remove', 'show', or
'quit'.")
47.
48. # 6. End of program
49. print("Manager shutting down.")

```

اجرا کن و نتیجه را ببین! برنامه را اجرا کن و با آن تعامل کن. آیتم‌ها را اضافه کن، لیست را ببین، آیتم‌ها را حذف کن و در نهایت خارج شو.

آفرین! تو همین الان یک «کوله پشتی» هوشمند ساختی. تو یاد گرفتی چطور مجموعه‌ای از غنائم را مدیریت کنی و با استفاده از حلقه for آن‌ها را سرشماری کنی.

# فصل ۶: تکنیک‌های پیشرفته

## کوله پشتی: تابلوی امتیازات

مأموریت فصل: ساخت یک تابلوی امتیازات (High Score Board) برای بازی حدس عددمان.

نقشه راه

- برش زدن کوله پشتی (Slicing): چطور فقط چند آیتم خاص را برداریم؟
- کیسه‌ی مُهر و موم شده (Tuples): آیتم‌های جادویی که تغییر نمی‌کنند.
- ترکیب جادوها: گشتن روی لیستی از تاپل‌ها.
- پروژه: ساخت تابلوی امتیازات و نمایش "Top 3".

## ۱. برش زدن کوله پشتی (Slicing)

گاهی ما نمی‌خواهیم فقط یک آیتم را برداریم ، بلکه می‌خواهیم بخشی از کوله پشتی را ببینیم (مثلاً سه آیتم اول برای تابلوی امتیازات). به این کار «برش زدن» (Slicing) می‌گویند.

ما از طلس [start:end] استفاده می‌کنیم.

نکته مهم: ایندکس end (پایان) در برش حساب نمی‌شود.

```

1. # Our list of adventurers
2. adventurers = ["Aria", "Kael", "Lyra", "Zane"]
3.
4. # Get the first two adventurers (index 0 and 1)
5. # We use [0:2] because it goes up to (but not including) index 2
6. first_two = adventurers[0:2] #
7. print("The first two adventurers:")
8. print(first_two) # Output: ['Aria', 'Kael']
9.
10. # Get the middle adventurers (from index 1 up to 3)
11. middle_group = adventurers[1:3]
12. print("The middle group:")
13. print(middle_group) # Output: ['Kael', 'Lyra']
14.
15. # A magic trick: leave start or end blank
16. # Get everything FROM index 2 TO THE END
17. all_except_first_two = adventurers[2:]
18. print("Everyone else:")
19. print(all_except_first_two) # Output: ['Lyra', 'Zane']

```

این طلس برای انتخاب «Top 3» یا «۵ نفر آخر» عالی است.

## ۲. کیسه‌ی مهر و موم شده (Tuples)

لیست‌ها (کوله پشتی) عالی هستند چون می‌توانیم آیتم‌ها را به آن‌ها append یا remove کنیم. آن‌ها «تغیرپذیر» (Mutable) هستند.

اما گاهی غنائمی داریم که هرگز نباید تغییر کنند، مثل مختصات جادویی یک گنج (X=10, Y=20)، یا یک رکورد امتیاز (که نام بازیکن و امتیازش به هم قفل شده‌اند).

برای این کار، ما از «تاپل» (Tuple) استفاده می‌کنیم. تاپل‌ها دقیقاً مثل لیست‌ها هستند، اما با پرانتز () ساخته می‌شوند و تغییرناپذیر (Immutable) هستند.

```

1. # A list is mutable (can change)
2. mutable_list = [10, 20]
3. mutable_list.append(30) # This works!
4. print("Mutable list:")
5. print(mutable_list)
6.
7. # A tuple is immutable (cannot change)
8. # Tuples are defined using parentheses ()
9. immutable_tuple = (10, 20)
10. print("Immutable tuple:")
11. print(immutable_tuple)
12.
13. # If you try to change a tuple, you get an ERROR!
14. # immutable_tuple.append(30) # This will crash the code!
    (AttributeError)

```

ما از تاپل‌ها برای داده‌هایی استفاده می‌کنیم که می‌خواهیم مطمئن باشیم به صورت تصادفی تغییر نمی‌کنند.

### ۳. جادوی باز کردن تاپل (Tuple Unpacking)

حالا که «تاپل» داریم، چطور به محتویات آن دسترسی پیدا کنیم؟ راه عادی (که از لیست‌ها بدلیم) استفاده از ایندکس است:

```

1. # A single high score record
2. score_record = ("Aria", 100)
3.
4. # The "old" way using indexes
5. name = score_record[0]
6. score = score_record[1]
7. print(name + " scored " + str(score))

```

این روش کار می‌کند، اما یک راه جادویی‌تر و تمیزتر هم وجود دارد.

پایتون به ما اجازه می‌دهد تا یک تاپل را در یک خط «باز» کنیم و هر آیتم آن را مستقیماً داخل یک متغیر برویزیم. به این کار Tuple Unpacking می‌گویند.

```

1. # The "magic" way: Tuple Unpacking
2. score_record = ("Aria", 100)
3.
4. # Python smartly puts "Aria" into 'name' and 100 into 'score'
5. name, score = score_record
6.
7. print(name + " scored " + str(score))

```

هر دو کد بالا یک نتیجه می‌دهند، اما روش دوم بسیار خواناتر و «پایتونیک»‌تر (Pythonic) است. ما به جای دو خط کد برای دسترسی به آیتم‌ها، در یک خط آن‌ها را «باز» کردیم.

#### ۴. ترکیب جادوها: لیستی از تاپل‌ها

خب، جادوی واقعی زمانی اتفاق می‌افتد که این دو را ترکیب کنیم. یک "تاپل" برای قفل کردن داده‌ها (مثل نام و امتیاز) عالی است. یک "لیست" برای نگهداری مجموعه‌ای از این تاپل‌ها عالی است!

این ساختار، ستون فقرات تابلوی امتیازات ما خواهد بود:

```

1. # Our High Score board
2. # It's a LIST (backpack) containing Tuples (sealed records)
3. high_scores = [
4.     ("Aria", 100),
5.     ("Kael", 80),
6.     ("Lyra", 75),
7.     ("Zane", 60),
8.     ("Finn", 30)
9. ]

```

حالا چطور روی این لیست سرشماری کنیم؟ درست مثل فصل ۵، با حلقه for

اما این بار، ما می‌توانیم از جادوی «باز کردن تاپل» (که در بخش قبل یاد گرفتیم) مستقیماً در خود حلقه for استفاده کنیم:

```
1. print("--- All Scores (unpacked) ---")
2.
3. # Instead of: for entry in high_scores:
4. # We write:   for name, score in high_scores:
5. # Python unpacks each tuple (like ('Aria', 100))
6. # into 'name' and 'score' in each loop.
7. for name, score in high_scores:
8.     # Now we use str() to join text and numbers (from Chapter 4)
9.     print(name + " scored " + str(score) + " points")
```

خروجی این کد بسیار زیباتر است:

```
1. --- All Scores (unpacked) ---
2. Aria scored 100 points
3. Kael scored 80 points
4. ...
```

## ۵. پروژه: ساخت تابلوی امتیازات "Top 3"

ماجراجو، تو آماده‌ای! ما می‌خواهیم با استفاده از لیستی از تاپل‌ها، جادوی «باز کردن تاپل» و طلسما «برش زدن»، تابلوی امتیازات بازی‌مان را بسازیم و سه نفر اول را نمایش دهیم.

چالش‌های فکری (تمرین تو):

ساخت لیست: چطور یک «لیست» به نام `high_scores` می‌سازی که شامل ۵ «تاپل» باشد؟ هر تاپل باید شامل (نام بازیکن، امتیاز) باشد. (لیست را طوری بساز که امتیازها از زیاد به کم مرتب باشند).

برش زدن: چطور با استفاده از طلسما «برش زدن» (Slicing)، فقط سه آیتم اول را از `high_scores` جدا می‌کنی و در یک لیست جدید به نام `top_3` می‌ریزی؟

نمایش نهایی: چطور با استفاده از یک حلقه `for` و جادوی «باز کردن تاپل» (Unpacking)، لیست `top_3` را چاپ می‌کنی تا نتیجه‌ای زیبا داشته باشد؟

وقت تمرین! قبل از اینکه به کد نهایی نگاه کنی، سعی کن خودت در فایل project.py این معما را حل کنی.

کد پروژه نهایی (در project.py بنویس):

```

1. # --- The High Score Board ---
2.
3. # 1. Setup the data
4. # A list of immutable (name, score) tuples
5. high_scores = [
6.     ("Aria", 100),
7.     ("Kael", 80),
8.     ("Lyra", 75),
9.     ("Zane", 60),
10.    ("Finn", 30)
11. ]
12.
13. print("--- All Scores ---")
14. # 2. Display all scores using tuple unpacking
15. for name, score in high_scores:
16.     print(name + " scored " + str(score) + " points")
17.
18. print("") # An empty print() adds a blank line
19. print("--- TOP 3 ADVENTURERS ---")
20.
21. # 3. Get the Top 3 using Slicing
22. # We want items from index 0 up to (but not including) 3
23. top_3_list = high_scores[0:3] #
24.
25. # 4. Display the Top 3 with ranking
26. rank = 1 # Our counter variable for the extra challenge
27. for name, score in top_3_list:
28.     # We cast 'rank' and 'score' to str() to join them
29.     print(str(rank) + ". " + name + " scored " + str(score) + " points")
30.     rank = rank + 1 # Increment the rank for the next loop

```

اجرا کن و نتیجه را بیین! خروجی نهایی تو باید این باشد:

```

1. --- All Scores ---
2. Aria scored 100 points
3. Kael scored 80 points
4. Lyra scored 75 points
5. Zane scored 60 points
6. Finn scored 30 points
7.
8. --- TOP 3 ADVENTURERS ---
9. 1. Aria scored 100 points

```

- |     |    |      |        |    |        |
|-----|----|------|--------|----|--------|
| 10. | 2. | Kael | scored | 80 | points |
| 11. | 3. | Lyra | scored | 75 | points |

آفرین! تو همین الان دو مورد از قدرتمندترین تکنیک‌های مدیریت داده در پایتون را یاد گرفتی. تو یاد گرفتی چطور با `Tuples` داده‌هایت را ایمن کنی و با `Slicing` به سرعت بخش‌هایی از آن‌ها را استخراج کنی. کوله پشتی تو حالا واقعاً حرفه‌ای شده است!



# فصل ۷: کتاب جادویی:

## ساخت مترجم کلمات

مأموریت فصل: ساخت یک دیکشنری ساده برای ترجمه کلمات در سفرهای خارجی.

نقشه راه

- دیکشنری‌ها (Dictionaries): کتاب جادویی که داده‌ها را به صورت کلید : مقدار ذخیره می‌کند.
- جادوی واژه‌نامه: دسترسی، اضافه کردن، تغییر دادن و حذف کردن «ورد»‌ها.
- تله‌ی کلید گمشده (The KeyError): چه اتفاقی می‌افتد اگر کلمه‌ای در کتاب نباشد؟
- طلسه‌های ایندیکس (in و get): دو راه برای جلوگیری از «کرش» کردن برنامه.
- سرشماری در کتاب جادویی: چطور روی تمام کلمات و معانی آن‌ها حلقه بزنیم؟
- پروژه: ساخت مترجمی که چند کلمه فارسی را به انگلیسی برمی‌گرداند.

## ۱. دیکشنری‌ها (Dictionaries): کتاب جادویی شما

تا به حال، ما از «لیست» ([]) برای نگهداری مجموعه‌ای از آیتم‌ها پشت سر هم استفاده می‌کردیم.

اما نوع جدیدی از ساختار داده وجود دارد به نام دیکشنری (Dictionary) که با آکولاد {} ساخته می‌شود. دیکشنری، آیتم‌ها را بر اساس «ایندکس عددی» (مثل ۰، ۱، ۲) ذخیره نمی‌کند.

در عوض، دیکشنری آیتم‌ها را به صورت جفت‌های «کلید : مقدار» (Key : Value) ذخیره می‌کند.

کلید (Key): «کلمه‌ای» است که ما جستجو می‌کنیم (باید یکتا باشد، مثل نام ورد).  
مقدار (Value): «معنی» یا «داده‌ای» است که به آن کلید متصل است.

درست مثل یک واژه‌نامه واقعی! «کلمه» کلید شماست و «معنی» آن مقدار شماست. یا مثل اطلاعات یک ماجراجو:

```
1. # We use curly braces {} for dictionaries
2. # We use colons (:) to link a key to a value
3. adventurer_stats = {
4.     "name": "Kael",
5.     "level": 5,
6.     "class": "Mage"
7. }
```

حالا جادوی واقعی: برای دسترسی به اطلاعات، ما دیگر از ایندکس [۰] استفاده نمی‌کنیم. ما مستقیماً «کلید» را صدا می‌زنیم:

```
1. # Accessing data using the 'key'
2. print("Adventurer's Name:")
3. print(adventurer_stats["name"]) # Output: Kael
4.
```

```
5. print("Adventurer's Level:")
6. print(adventurer_stats["level"]) # Output: 5
```

این کار بسیار خواناتر و قدرتمندتر از حدس زدن ایندکس `adventurer_list[۰]` است!

## ۲. جادوی واژه‌نامه: افزودن، تغییر و حذف

قدرت دیکشنری‌ها در اضافه کردن، تغییر دادن و جستجوی سریع است.

### اضافه کردن یا تغییر دادن یک «ورد»

برای اضافه کردن یک جفت «کلید: مقدار» جدید، یا تغییر دادن یک مقدار قدیمی، به سادگی از این طلسم استفاده می‌کنیم:

```
1. # Our translator starts empty
2. translator = {}
3.
4. # Add a new word (key-value pair)
5. translator["hello"] = "salam"
6. translator["goodbye"] = "khodahafez"
7.
8. print("Translator after adding words:")
9. print(translator) # Output: {'hello': 'salam', 'goodbye': 'khodahafez'}
10.
11. # Change an existing word (update the value)
12. translator["hello"] = "dorood"
13. print("Translator after update:")
14. print(translator) # Output: {'hello': 'dorood', 'goodbye': 'khodahafez'}
```

### حذف کردن یک «ورد»

اگر بخواهیم یک کلمه و معنی آن را کاملاً از کتاب جادویی پاک کنیم، از طلسم `del` (مخفف `delete`) استفاده می‌کنیم:

```
1. translator = {"hello": "salam", "goodbye": "khodahafez"}
2. print("Translator before delete:")
3. print(translator)
```

```

4.
5. # Delete the word 'hello'
6. del translator["hello"]
7.
8. print("Translator after delete:")
9. print(translator) # Output: {'goodbye': 'khodahafez'}
10.

```

### ۳. تلهی کلید گمشده (The KeyError)

هشدار ماجراجو! یک تلهی بزرگ در دیکشنری‌ها پنهان شده است. چه اتفاقی می‌افتد اگر سعی کنی به کلمه‌ای دسترسی پیدا کنی که در کتاب جادویی تو وجود ندارد؟

```

1. translator = {"hello": "dorood"}
2. # Try to access a key that DOES NOT exist
3. print(translator["cat"])

```

برنامه «کرش» می‌کند! غول چراغ جادو فریاد می‌زند: KeyError: 'cat'. این یک «باغ» جدید است. پایتون می‌گوید: «من کلیدی به نام 'cat' در این کتاب پیدا نمی‌کنم!»

### ۴. طلسه‌های ایمنی (get و in)

ما باید مثل یک کارآگاه کد حرفه‌ای، قبل از دسترسی به کلید، از وجود آن مطمئن شویم. ما دو طلسه ایمنی برای این کار داریم:

#### طلسم اول: بردسی با in

ما می‌توانیم از همان طلسه in (که در فصل ۵ برای لیست‌ها یاد گرفتیم) استفاده کنیم تا بینیم آیا «کلید» (کلمه) در واژهنامه ما وجود دارد یا نه.

```

1. translator = {"hello": "dorood"}
2. word_to_find = "cat"
3.
4. # Check if the 'key' is in the dictionary

```

```
5. if word_to_find in translator:  
6.     print("Translation is:")  
7.     print(translator[word_to_find])  
8. else:  
9.     print("Sorry, that word was not found.") # This will run!
```

این امن‌ترین و رایج‌ترین راه است.

### طلسم دوم (حروف‌ای): متدهای get()

دیکشنری‌ها یک ورد جادویی داخلی به نام `get()` دارند. این طلسم دو ورودی می‌گیرد: ۱. کلیدی که دنبالش می‌گردد. ۲. یک «مقدار پیش‌فرض» که اگر کلید پیدا نشد، آن را برمی‌گرداند.

این طلسم هرگز کرش نمی‌کند!

```
1. translator = {"hello": "dorood"}  
2.  
3. # Get the translation for 'hello'  
4. translation_1 = translator.get("hello", "Not Found")  
5. print(translation_1) # Output: dorood  
6.  
7. # Get the translation for 'cat'  
8. translation_2 = translator.get("cat", "Word not found.")  
9. print(translation_2) # Output: Word not found. (No crash!)
```

## ۵. سرشماری در کتاب جادویی (Looping)

در فصل ۵، ما با حلقه `for` روی «لیست‌ها» گشتمیم. چطور می‌توانیم روی «دیکشنری» حلقه `for` بزنیم؟

حلقه زدن روی «کلیدها» (Keys)

اگر به سادگی روی دیکشنری `for` بزنی، پایتون به تو «کلیدها» را می‌دهد:

```
1. translator = {"hello": "dorood", "goodbye": "khodāhāfez"}  
2.  
3. print("--- Dictionary Keys ---")  
4. for word in translator:  
5.     print(word)
```

خروجی:

```
1. --- Dictionary Keys ---
2. hello
3. goodbye
```

حلقه زدن روی «مقدارها» (Values)

اگر فقط به «معانی» (مقدارها) نیاز داری، از طلسم values استفاده کن:

```
1. print("--- Dictionary Values ---")
2. for meaning in translator.values():
3.     print(meaning)
```

خروجی:

```
1. --- Dictionary Values ---
2. dorood
3. khodāhāfez
```

حلقه زدن روی هر دو (Key & Value) - بهترین روش!

و اما قدرتمندترین جادو! اگر بخواهی هم «کلید» و هم «مقدار» را با هم داشته باشی، از طلسم items استفاده کن. این طلسم لیستی از «تاپل‌ها» (که در فصل ۶ یاد گرفتیم) برمی‌گرداند!

ما می‌توانیم از همان جادوی «باز کردن تاپل» (Tuple Unpacking) در حلقه for استفاده کنیم:

```
1. print("--- Full Translator ---")
2. # .items() gives us (key, value) pairs
3. for word, meaning in translator.items():
4.     print(word + " means " + meaning)
```

خروجی:

```
1. --- Full Translator ---
```

- |                             |
|-----------------------------|
| 2. hello means dorood       |
| 3. goodbye means khodāhāfez |
| 4.                          |

## ۶. هشدار ماجراجو: طلسی نمایش حروف فارسی ✎

یک تله‌ی کوچک ممکن است در مسیر ماجراجویی تو وجود داشته باشد! گاهی اوقات، «اتفاق فرمان» (ترمینال) در برخی سیستم‌ها (به خصوص نسخه‌های قدیمی‌تر ویندوز) نمی‌داند چطور با طومارهای فارسی کار کند.

اگر بعد از وارد کردن یک کلمه فارسی، در خروجی علامت سوال (?????) یا حروف عجیب و غریب دیدی، نترس! این یعنی ترمینال تو به یک ورد جادویی نیاز دارد تا زبان فارسی را بفهمد.

اگر در ترمینال VS Code یا ویندوز به مشکل خورده، قبل از اجرای برنامه، این دستور را یک بار در ترمینالت تایپ کن و Enter را بزن:

```
chcp 65001
```

این طلسی به اتفاق فرمان می‌گوید که از «کتاب رمزگشایی» استاندارد جهانی (UTF-8) برای خواندن حروف استفاده کند. در اکثر سیستم‌های مدرن (مثل Windows، macOS، لینوکس و Terminal جدید) این مشکل وجود ندارد، اما دانستن این طلسی، تو را برای هر چالشی آماده می‌کند!

## ۷. پروژه: ساخت مترجم کلمات

ماجراجو، تو آمده‌ای! ما می‌خواهیم با استفاده از ابزار جدید و قدرتمندمان، «دیکشنری»، یک مترجم ساده بسازیم.

نقشه و طرح اولیه (قبل از کدنویسی)

بیا «نقشه» ابزارمان را بکشیم:

حافظه (واژه‌نامه): ما به یک «دیکشنری» نیاز داریم که کلمات فارسی (به عنوان کلید) و ترجمه انگلیسی آن‌ها (به عنوان مقدار) را نگه دارد.

تعامل: برنامه باید به طور مداوم از ما بپرسد که کدام کلمه را می‌خواهیم ترجمه کنیم. (کدام ورد جادویی از فصل ۴ برای حلقه بینهایت خوب بود؟)

شرط خروج: باید یک کلمه جادویی (مثل "quit") برای خروج از حلقه داشته باشیم.

منطق: برنامه باید چک کند آیا کلمه‌ای که ما گفتیم، در واژه‌نامه «موجود» است یا نه. (کدام طلسمنی از این فصل برای این کار عالی است؟)

پاسخ:

اگر کلمه موجود بود، «مقدار» (Value) آن (یعنی ترجمه‌اش) را چاپ کند.

اگر موجود نبود، پیام «کلمه یافت نشد» را چاپ کند.

چالش‌های فکری (تمرین تو)

حالا بیا با هم فکر کنیم چطور هر بخش را می‌سازیم:

چالش ۱: ساخت واژه‌نامه چطور یک متغیر دیکشنری به نام farsi\_to\_english می‌سازی که از قبل شامل این سه جفت باشد؟

"Hello" به "سلام"  
"Goodbye" به "خداحافظ"  
"Cat" به "گربه"

چالش ۲: حلقه بی‌نهایت و ورودی چطور یک حلقه while True می‌سازی که داخل آن، با input() از کاربر بپرسد: Enter a Farsi word to translate: و جوابش را در متغیری به نام word ذخیره کند؟

چالش ۳: شرط خروج چطور درست بعد از گرفتن ورودی، چک می‌کنی که == quit، و اگر درست بود، با طلسما break از حلقه بیرون بپرسی؟

چالش ۴: بررسی ایمن و نمایش چطور یک بلوك if/else می‌نویسی که:

(طلسم ایمنی) بررسی کند آیا word in farsi\_to\_english (داخل) هست یا نه؟  
اگر بود، با استفاده از farsi\_to\_english[word] ترجمه‌اش را چاپ کند.  
اگر نبود (else)، پیام "Word not found in dictionary." را چاپ کند؟

وقت تمرین! قبل از اینکه به کد نهایی نگاه کنی، سعی کن خودت در فایل project.py، با استفاده از این چالش‌ها و ابزارهای جدید (in, {}, ,key:value) و ابزارهای قدیمی (while, True, input, if / elif / else, break) این برنامه را بسازی.

کد پروژه نهایی (در project.py بنویس):

```

1. # --- The Farsi to English Translator ---
2.
3. # 1. Setup the Dictionary (our magic glossary)
4. # Keys are Farsi, Values are English
5. farsi_to_english = {
6.     "سلام": "Hello",
7.     "خد احافظ": "Goodbye",
8.     "گربه": "Cat",
9.     "سگ": "Dog",
10.    "خانه": "House"
11. }
12.
13. print("Welcome to the Farsi-English Translator.")
14. print("--- Type 'quit' to exit ---")
15.
16. # 2. Start the main loop (from Chapter 4)
17. while True:
18.     print("-----")
19.     # Get input from the user (from Chapter 4)
20.     word = input("Enter a Farsi word to translate: ")
21.
22.     # 3. Create an exit command
23.     if word == "quit":
24.         print("Goodbye, adventurer!")
25.         break # 'break' is a spell that jumps out of a loop
26.
27.     # 4. Check if the word is in our dictionary (Safety Spell!)
28.     if word in farsi_to_english:
29.         # If YES, get the 'value' using the 'key'
30.         translation = farsi_to_english[word]
31.         print("The English translation is: " + translation)
32.     else:
33.         # If NO, tell the user
34.         print("Sorry, that word was not found in the dictionary.")
35.
```

اجرا کن و نتیجه را ببین! برنامه را اجرا کن و کلماتی مثل «سلام» یا «خانه» را امتحان کن. سپس کلمه‌ای مثل «کتاب» را امتحان کن تا پیام «یافت نشد» را ببینی. در نهایت با تایپ quit از برنامه خارج شو.

آفرین! تو همین الان سریع‌ترین ابزار جستجوی خود را ساختی. تو یاد گرفتی که «لیست‌ها» برای ترتیب و «دیکشنری‌ها» برای جستجوی سریع عالی هستند. کوله پشتی تو حالا یک «کتاب جادویی» هم در کنارش دارد!



# فصل ۸: ساخت جادوهای

## شخصی: هنر ساخت توابع

مأموریت فصل: بازسازی یک «کدی که هی تکرار شده» (Repetitive Code) و ساخت یک «کارت معرفی» جادویی و قابل استفاده مجدد با استفاده از توابع.

### نقشه راه

- "یک بار بنویس، صد بار استفاده کن": معرفی توابع (Functions) و اصل DRY.
- چطور یک بلوک جادویی را «تعریف» و آن را «صدای» بزنیم؟ طلسم‌های def و () .
- اولین طلسم ما: ساخت یک تابع ساده برای جلوگیری از تکرار.
- ورودی تابع (Arguments): چطور به طلسم خود «داده» پاس بدهیم؟
- خروجی تابع (Return): چطور از طلسم خود «غنیمت» پس بگیریم؟
- پروژه: «رفع طلسم» یک کد تکراری برای نمایش مشخصات ماجراجویان.

## ۱. "یک بار بنویس، صد بار استفاده کن": معرفی توابع

در ماجراجویی‌هایمان تا به حال، ما از وردهای جادویی داخلی پایتون مثل `print()`, `input()`, `len()` و `()` استفاده کرده‌ایم. این‌ها «توابع» (Functions) از پیش ساخته شده هستند.

اما قدرت واقعی زمانی آشکار می‌شود که تو «تابع» خودت را بسازی.

یک تابع (Function)، یک بلوک کد نام‌گذاری شده و قابل استفاده مجدد است که برای انجام یک کار خاص طراحی شده.

یک مثال ساده: فکر کن می‌خواهی یک ساندویچ درست کنی.

کدنویسی بدون تابع (کاری که تا الان می‌کردیم):

نان را بردار.

کره بمال.

پنیر بگذار.

نان دوم را بگذار.

(حالا می‌خواهی ساندویچ دوم را درست کنی...)

نان را بردار.

کره بمال.

پنیر بگذار.

نان دوم را بگذار.

کدنویسی با تابع (کاری که یاد می‌گیریم):

دستورالعمل (تابع) می‌سازیم: اسمش را می‌گذاریم (`make_sandwich()`)

داخل این دستورالعمل، ۴ مرحله‌ی بالا را می‌نویسیم.

(حالا هر وقت گرسنه شدیم...)

`make_sandwich()` را صدا می‌زنیم.

را دوباره صدا می‌زنیم `make_sandwich()`.

شعار ماجراجویان حرفه‌ای این است: Don't Repeat Yourself (DRY) یا «خودت را تکرار نکن». اگر کدی را دوبار کپی-پیست کردی، احتمالاً باید آن را به یک تابع تبدیل کنی.

## ۲. «تعريف» (Define) و «صدا زدن» (Call) یک طلسما

ساختن یک جادوی شخصی دو مرحله دارد:

مرحله ۱: «تعريف» طلسما با `def` اول، باید جادو را به غول چراغ جادو «یاد بدھی». ما این کار را با ورد `def` (مخفف Define) انجام می‌دهیم. این کار مثل نوشتمن دستورالعمل طلسما روی یک «طومار» جدید است.

```
1. # --- 1. Defining the Spell (Writing the Scroll) ---
2. # We create a new function called 'greet_adventurer'
3. # The code INSIDE (indented) is the spell's logic.
4. def greet_adventurer():
5.     print("Greetings, mighty adventurer!")
6.     print("Welcome to the Python Guild.")
```

اگر این کد را به تهابی اجرا کنی، هیچ اتفاقی نمی‌افتد! چرا؟ چون ما فقط «دستورالعمل» ساندویچ را نوشته‌ایم، هنوز ساندویچی درست نکرده‌ایم.

مرحله ۲: «صدا زدن» طلسم با () نوشتن طلسم (تعریف تابع) آن را اجرا نمی‌کند. تو باید آن را «صدا بزنی» (Call) تا اجرا شود. صدا زدن یعنی استفاده از نام تابع همراه با پرانتز () .

```

1. # --- 1. Defining the Spell ---
2. def greet_adventurer():
3.     print("Greetings, mighty adventurer!")
4.     print("Welcome to the Python Guild.")
5.
6. # --- 2. Calling the Spell (Making the sandwich) ---
7. print("The guild master steps forward...")
8. greet_adventurer() # <-- The magic happens here!
9.
10. print("The quest begins...")
11. greet_adventurer() # <-- We can call it again, easily!
```

اجرا کن و نتیجه را بین!

```

1. The guild master steps forward...
2. Greetings, mighty adventurer!
3. Welcome to the Python Guild.
4. The quest begins...
5. Greetings, mighty adventurer!
6. Welcome to the Python Guild.
```

ما کد را یک بار نوشتیم و دو بار (یا صد بار!) از آن استفاده کردیم.

### ۳. اولین طلسم ما: خداحافظی با تکرار

یادت هست در پروژه‌های قبلی مدام از "print("=====")" برای جداسازی استفاده می‌کردیم؟ بباید این کد تکراری را «رفع طلسم» کنیم.

```

1. # --- The Old, Repetitive Way ---
2. print("--- Adventurer 1 ---")
3. print("Name: Aria")
4. print("=====")
5. print("--- Adventurer 2 ---")
6. print("Name: Kael")
7. print("=====")
8.
9. # --- The New, Clean Way (with a Function) ---
```

```

10.
11. # 1. Define the spell once
12. def print_divider():
13.     print("====")
14.
15. # 2. Call it whenever we need it
16. print("--- Adventurer 1 ---")
17. print("Name: Aria")
18. print_divider()
19.
20. print("--- Adventurer 2 ---")
21. print("Name: Kael")
22. print_divider()

```

کد جدید تمیزتر، خواناتر و حرفه‌ای‌تر است!

## ۴. ورودی و خروجی توابع (return و Arguments)

طلسم‌های ما خوب بودند، اما کمی خنگ بودند. greet\_adventurer به همه‌ی ماجراجویان یک پیام تکراری می‌دهد و print\_divider فقط یک کار ثابت انجام می‌دهد. چطور طلسم‌هایی بسازیم که بتوانند «داده» بگیرند و «نتیجه» (غنیمت) برگردانند؟

### ورودی: آرگومان‌ها (Arguments)

فکر کن تابع تو یک «دستگاه جادویی» است. ما می‌توانیم برای این دستگاه «ورودی» تعریف کنیم. به این ورودی‌ها پارامتر (Parameter) می‌گوییم.

```

1. # 'name' is a PARAMETER (a magic box inside the function)
2. def greet_by_name(name):
3.     # 'name' acts like a variable that exists ONLY inside this function
4.     print("Greetings, " + name + "!")
5.     print("Welcome to the guild.")
6.
7. # Now, when we call it, we provide an ARGUMENT (the loot)
8. # An argument is the *actual data* we pass in.
9. greet_by_name("Aria") # "Aria" is put into the 'name' box
10. greet_by_name("Kael") # "Kael" is put into the 'name' box
11.

```

خروجی:

- ```

1. Greetings, Aria!
2. Welcome to the guild.
3. Greetings, Kael!
4. Welcome to the guild.

```

«پارامتر» (name) مثل جای خالی روی طومار است. «آرگومان» ("Aria") جوهری است که آن جای خالی را پر می‌کند.

خروجی: طلسما return

تا به حال، توابع ما فقط کارها را print می‌کردند (به این کار «اثر جانبی» یا Side Effect می‌گویند). اما یک طلسما واقعی باید بتواند یک «غニمت» (نتیجه) به ما «برگرداند».

ورد return جادو را متوقف می‌کند و یک مقدار را به بیرون «پاس می‌دهد».

تفاوت print و return (بسیار مهم):

فقط چیزی را روی صفحه نشان می‌دهد.  
return یک مقدار را به کدی که آن را صدا زده پس می‌دهد.

بیا «ماشین حساب جادویی» فصل ۳ را به یک تابع واقعی تبدیل کنیم:

- ```

1. # --- Version 1: Using print (Not reusable) ---
2. def print_sum(num1, num2):
3.     total = num1 + num2
4.     print("The sum is: " + str(total))
5.
6. # --- Version 2: Using return (Powerful!) ---
7. def add_numbers(num1, num2):
8.     total = num1 + num2
9.     return total # Give this value back!
10.
11. # --- Let's see the difference ---
12.

```

```
13. # Using print_sum:  
14. print_sum(5, 10) # This prints "The sum is: 15"  
15. # result = print_sum(5, 10) # This does NOT work! 'result' will be  
empty.  
16.  
17. # Using add_numbers:  
18. # We must "catch" the returned value in a variable  
19. sum_of_gold = add_numbers(50, 75)  
20. print("The total gold is:")  
21. print(sum_of_gold) # Output: 125  
22.  
23. # Now we can USE the result  
24. if sum_of_gold > 100:  
25.     print("We are rich!")  
26.
```

تابع add\_numbers بسیار مفید‌تر است، چون به ما «داده» بر می‌گرداند که می‌توانیم با آن کار کنیم (مثلًاً در یک if از آن استفاده کنیم).

## ۵. پروژه: رفع طلس «طومار تکراری»

ماجراجو، تو آماده‌ای! «ماموریت» ما این است که یک کد «بد» و «تکراری» را با جادوی جدیدمان (توابع) پاکسازی کنیم.

کد طلس شده (The Repetitive Scroll): تصور کن می‌خواهیم مشخصات دو ماجراجوی تیممان را چاپ کنیم. کد ما در حال حاضر این شکلی است:

```
1. # --- The Bad, Repetitive Code ---  
2.  
3. print("--- Adventurer ID Card ---")  
4. print("Name: Aria")  
5. print("Class: Ranger")  
6. print("HP: 100")  
7. print("====")  
8.  
9. print("--- Adventurer ID Card ---")  
10. print("Name: Kael")  
11. print("Class: Mage")  
12. print("HP: 80")  
13. print("====")
```

این کد کار می‌کند، اما افتضاح است! ما تمام بلوک print را دوبار کپی-پیست کردیم . اگر بخواهیم یک خط جدید (مثلاً "Guild: Python") اضافه کنیم، باید آن را در هر دو بلوک تغییر دهیم. این یعنی اتلاف وقت و دعوت از «باغ»ها!

نقشه و طرح اولیه (قبل از کدنویسی)

شناسایی تکرار: بلوک print مانند ۵ خط تکراری دارد.

شناسایی تفاوت‌ها: تنها چیزهایی که تغییر می‌کنند عبارتند از: «نام» (Name)، «کلاس» (Class)، و «میزان سلامتی» (HP).

نقشه تابع: ما باید یک تابع بسازیم (مثلاً def show\_stats (...)) که این سه چیز (نام، کلاس، HP) را به عنوان «ورودی» (پارامتر) بگیرد و آن ۵ خط print را اجرا کند.

چالش‌های فکری (تمرین تو)

چالش ۱: تعریف تابع چطور یک تابع جدید به نام show\_adventurer\_stats تعریف می‌کنی؟

چالش ۲: تعریف پارامترها (ورودی‌ها) این تابع به چند «جعبه جادویی» (پارامتر) در داخل پرانتزش نیاز دارد تا بتواند اطلاعات هر ماجراجو را بگیرد؟ (راهنمایی: ۳ تا لازم داریم). اسم آنها را چه می‌گذاری؟ (مثلاً name, job\_class, hp)

چالش ۳: نوشتن بدنه طلسم چطور ۵ خط print تکراری را داخل تابع جدیدت کپی می‌کنی؟

چالش ۴: استفاده از متغیرها چطور خطوط print داخل تابع را تغییر می‌دهی تا به جای مقادیر ثابت مثل "Aria"، از متغیرهای ورودی (پارامترها) مثل name استفاده کنند؟ (راهنمایی: hp یک عدد (Integer) است، برای .print("Name: " + name) مراقب باش! چسباندن آن به متن باید از کیمیاگری str() استفاده کنی!)

چالش ۵: صدا زدن طلسم پاکسازی شده حالا که تابع جادویی show\_adventurer\_stats را ساختی، چطور آن کد «بد» و ۱۰ خطی بالا را پاک می‌کنی و آن را با دو خط «صدا زدن» تمیز جایگزین می‌کنی؟ (یکی برای "Aria" و یکی برای "Kael").

وقت تمرین! قبل از دیدن کد نهایی، سعی کن خودت این بازسازی را در project.py انجام دهی.

کد پروژه نهایی (در project.py بنویس):

```

1. # --- The Cleaned-Up Spellbook (Using Functions) ---
2.
3. # 1. --- Define the Reusable Spell ---
4. # We create ONE function that takes 3 arguments
5. def show_adventurer_stats(name, job_class, hp):
6.     print("--- Adventurer ID Card ---")
7.     print("Name: " + name)
8.     print("Class: " + job_class)
9.     # We must cast the integer 'hp' to a string 'str()' to join it
10.    print("HP: " + str(hp))
11.    print("=====")
12.
13.
14. # 2. --- Call the Spell as many times as we want ---
15. # Our main code is now simple, clean, and easy to read!
16.
17. show_adventurer_stats("Aria", "Ranger", 100)
18. show_adventurer_stats("Kael", "Mage", 80)
19. show_adventurer_stats("Gimli", "Warrior", 150) # Adding a new one is
easy!
```

اجرا کن و نتیجه را بیین! خروجی این کد تمیز، دقیقاً مشابه آن کد کثیف و تکراری است، اما حالا کد ما «حروفهای» است. اگر بخواهیم یک خط جدید به کارت اضافه کنیم، فقط کافیست تابع را یک بار ویرایش کنیم.

آفرین! تو همین الان از یک «نوآموز» که کدها را پشت سر هم می‌نویسد، به یک «معمار کد» تبدیل شدی که ابزارهای تمیز، سازمان یافته و قابل استفاده مجدد می‌سازد. «کتاب جادوی» تو حالا به جای یک طومار درهم، شبیه یک کتابخانه‌ی واقعی از جادوهای شخصی شده است!

# فصل ۹: طلسمندی محفوظ

## ساخت رمز عبور امن

مأموریت فصل: تولیدکننده رمز برای یک صندوقچه گنج.

نقشه راه

- استفاده از قدرت‌های مخفی پایتون: کتابخانه‌ها (Modules) و طلسمند (Imports)
- کتابخانه random: بهترین دوست ما برای کارهای تصادفی.
- جادوی شمارش: تکرار یک طلسمند با for i in range( )
- پروژه: ساخت برنامه‌ای برای تولید رمزهای عبور قوی و تصادفی.

## ۱. استفاده از قدرت‌های مخفی پایتون: کتابخانه‌ها و `import`

تا به حال، تمام کدهای ما در فایل `project.py` (کتاب جادویی شخصی ما) نوشته شده‌اند. اما پایتون یک «کتابخانه بزرگ» (Python Standard Library) از طلسم‌ها و «کتاب‌های جادویی» آماده دارد که به آن‌ها «ماژول» (Module) می‌گویند.

وقتی تو پایتون را نصب کردی، این کتابخانه عظیم هم همراه آن نصب شد.

برای استفاده از این طلسم‌های آماده، ما باید آن‌ها را به کتاب جادویی خودمان «وارد» کنیم. ورد جادویی برای این کار `import` است.

```
1. # Import the 'random' spellbook from the Great Library
2. import random
```

با این کار، ما به پایتون می‌گوییم: «برو به کتابخانه بزرگ، کتاب جادویی `random` را پیدا کن و بیاور. من می‌خواهم از طلسم‌های داخل آن استفاده کنم.»

## ۲. کتابخانه `random`: بهترین دوست ما برای کارهای تصادفی

غول چراغ جادوی ما (کامپیوتر) ذاتاً یک موجود «منطقی» و «قابل پیش‌بینی» است. او عاشق «تصادف» نیست. ماژول `random` به ما این قدرت جادویی را می‌دهد که او را مجبور به کارهای تصادفی کنیم.

وقتی ماژولی را `import` می‌کنیم، برای استفاده از طلسم‌های داخل آن، از «طلسم نقطه» (.) استفاده می‌کنیم. این یعنی: «از کتاب `random`، طلسم `choice` را اجرا کن».

در اینجا سه طلسما بسیار پرکاربرد از این کتابخانه را می‌بینیم:

random.randint(a, b): طلسما تاس ریختن این طلسما یک عدد تصادفی (Integer) بین دو عددی که به او می‌دهی (شامل خود آن دو عدد) انتخاب می‌کند. این دقیقاً مثل ریختن یک تاس در بازی است.

```
1. import random
2.
3. # Roll a 20-sided die
4. die_roll = random.randint(1, 20)
5. print("You rolled the die:")
6. print(die_roll)
7.
8. # Roll a simple 6-sided die
9. die_roll_6 = random.randint(1, 6)
10. print("You rolled a 6-sided die:")
11. print(die_roll_6)
```

random.choice(sequence): طلسما انتخاب از کیسه این مهم‌ترین طلسما ما برای این فصل است. تو یک «کوله پشتی» (List) یا «طومار» (String) به او می‌دهی، و او یک آیتم تصادفی از داخل آن برای تو انتخاب می‌کند.

```
1. import random
2.
3. # A backpack of possible loot
4. loot_options = ["Gold Coin", "Health Potion", "Empty Bottle", "Ancient
Sword"]
5.
6. # Call the 'choice' spell from the 'random' spellbook
7. chosen_loot = random.choice(loot_options)
8.
9. print("You opened the chest and found a...")
10. print(chosen_loot) # Try running this multiple times!
```

random.shuffle(list) طلسما بُر زدن این طلسما یک «کوله پشتی» (List) را می‌گیرد و آیتم‌های آن را درجا بُر می‌زند (ترتیب‌شان را به هم می‌ریزد). این طلسما هیچ چیزی return نمی‌کند، بلکه خود لیست اصلی را تغییر می‌دهد.

```

1. import random
2.
3. # A list of adventurers
4. adventurers = ["Aria", "Kael", "Lyra", "Zane"]
5. print("Original party order:")
6. print(adventurers)
7.
8. # Shuffle the list in-place
9. random.shuffle(adventurers)
10.
11. print("Shuffled party order:")
12. print(adventurers)
```

اجرا کن و نتیجه را بیین! هر بار که کد shuffle را اجرا کنی، ترتیب جدیدی می‌بینی.

### ۳. جادوی شمارش: تکرار با for i in range()

ما در فصل ۵ یاد گرفتیم که چطور با for روی آیتم‌های یک «کوله پشتی» (List) حلقه بزنیم . (for item in shopping\_list)

اما اگر «کوله پشتی» نداشته باشیم چه؟ اگر فقط بخواهیم یک کار را «۱۰ بار» تکرار کنیم؟

برای این کار، ما از یک ورد جادویی داخلی به نام range استفاده می‌کنیم.

range(10) یک دنباله‌ی جادویی و نامرئی از اعداد ۰ تا ۹ (دقیقاً ۱۰ عدد) می‌سازد. ما می‌توانیم روی این دنباله حلقه بزنیم:

```
1. # The range(stop) spell
2. # This will loop 5 times (0, 1, 2, 3, 4)
3. for i in range(5):
4.     print("This is loop number: " + str(i))
```

متغیر `i` فقط یک نام موقتی است که شماره‌ی مرحله‌ی فعلی حلقه را نگه می‌دارد. (ما حتی می‌توانستیم اسمش را `for number in range(5)` بگذاریم).

این طلسما برای ساختن رمز عبوری که مثلاً «۱۲ حرف» دارد، عالی است!

#### ۴. پروژه: انتخاب رمز برای صندوقچه گنج خودمان

ماجراجو، تو آماده‌ای! ما یک «صندوقچه گنج» پیدا کرده‌ایم و می‌خواهیم غنائم خود را در آن پنهان کنیم. اما این صندوقچه به یک قفل جادویی نیاز دارد.

ما نمی‌توانیم یک رمز ساده مثل "۱۲۳۴" یا "password" انتخاب کنیم، چون دزدها به راحتی آن را حدس می‌زنند. ما به یک طلسما نیاز داریم که یک رمز عبور ۱۲ حرفی، قوی و کاملاً تصادفی برای ما بسازد تا بتوانیم با خیال راحت از غنائم خود محافظت کنیم.

نقشه و طرح اولیه (قبل از کدنویسی)  
وارد کردن جادو: ما به جادوی «تصادف» نیاز داریم. کدام کتابخانه را باید `import` کنیم؟

مواد اولیه (Ingredients): یک رمز قوی از چه چیزهایی ساخته شده؟ حروف کوچک، حروف بزرگ، اعداد و نمادها. ما باید «طومارهایی» (Strings) بسازیم که همه‌ی این‌ها را در خود داشته باشند.

# بخش سوم: ورود به بعد چهارم

آفرین ماجراجو! تو بخش دوم را با موفقیت کامل پشت سر گذاشتی.

تو حالا یک استاد مدیریت «کوله پشتی» (List)، «کیسه‌ی مهر و موم شده» (Tuple)، و «کتاب جادویی» (Dictionary) هستی. تو فقط یک کاربر طلسمنیستی، بلکه با یادگیری «تابع» (Functions) به یک «خالق طلسمن» تبدیل شدی و با import یاد گرفتی که از جادوی دیگران هم استفاده کنی.

اما یک مشکل بزرگ باقی مانده است. یک تله‌ی مرگبار به نام «فراموشی»!  
وقتی ۲۰ آیتم به «مدیر لیست خرید» خود اضافه می‌کنی و برنامه را می‌بندی... چه اتفاقی می‌افتد؟ همه‌ی آیتم‌ها پاک می‌شوند! وقتی «متترجم» تو کلمه‌ی "quit" را می‌شنود، تمام کلماتی که به آن یاد داده بودی را فراموش می‌کند. جادوی ما حافظه‌ی دائمی ندارد. تمام غنائم ما تا امروز در حافظه‌ی موقت (RAM) زندگی می‌کردند. به محض اینکه غول چراغ جادو به خواب می‌رود (برنامه بسته می‌شود)، همه چیز از بین می‌رود.

در این بخش، ما یاد می‌گیریم که چطور غنائم خود را روی «طومارهای باستانی» واقعی (فایل‌های متنه‌ی روی هارد دیسک) حک کنیم. ما یاد می‌گیریم چطور داده‌ها را ذخیره (Save) و بازیابی (Load) کنیم تا ابزارهای ما واقعاً ماندگار شوند. ما یاد می‌گیریم که مثل یک مهندس نرم‌افزار فکر کنیم و کدهایمان را به موجوداتی زنده و دارای «روح» تبدیل کنیم.

وقت آن است که وارد بخش سوم: ورود به بعد چهارم شویم.

# فصل ۱۰: رمزگشایی از طومارهای باستانی: تحلیلگر متن

مأموریت فصل: ساخت ابزاری که یک متن را تحلیل کرده و آمار آن را استخراج می‌کند.

نقشه راه

- کار حرفه‌ای با متن‌ها: طلسه‌های جدید برای «طومارها» (Strings).
- هنر نمایش: سه طلسه جادویی برای قالب‌بندی (Formatting).
- طلسه باستانی: قالب‌بندی با %.
- طلسه مدرن: متدهای format()
- طلسه استاد: قالب‌بندی با f-strings
- پروژه: ساخت برنامه‌ای که تعداد کلمات و کاراکترهای یک متن را می‌شمارد.

## ۱. کار حرفه‌ای با متن‌ها (طلسم‌های String)

«طومار» (String) در پایتون، فقط یک متن ساده نیست؛ بلکه یک «شیء» (Object) قدرتمند با مجموعه‌ای از طلسه‌های داخلی (متدها) است. بیا چند مورد از پرکاربردترین آن‌ها را یاد بگیریم.

### طلسم len() (بازخوانی)

تو این طلسه را از فصل ۵ برای «کوله پشتی» (List) به یاد داری. خبر خوب این است که () len روی طومارها هم دقیقاً همان‌طور کار می‌کند و «تعداد کل کاراکترها» (شامل فاصله‌ها و علائم) را برمی‌گرداند.

```
1. scroll = "Hello World"
2. char_count = len(scroll)
3. print(char_count) # Output: 11
4. های طلسه .lower() و .upper()
```

این طلسه‌ها متن تو را به حروف کوچک یا بزرگ تبدیل می‌کنند. این برای «یکسانسازی» داده‌ها عالی است (مثلاً تا "Hello" و "hello" یکسان در نظر گرفته شوند).

```
1. scroll = "Aria the Ranger"
2.
3. print(scroll.lower()) # Output: "aria the ranger"
4. print(scroll.upper()) # Output: "ARIA THE RANGER"
```

### طلسم split(): قدرتمندترین طلسه این فصل

این طلسه، جادوی اصلی ما برای مأموریت این فصل است. split() یک «طومار» (String) را می‌گیرد و آن را بر اساس «فاصله‌ها» (space) می‌شکند و نتیجه را در قالب یک «کوله پشتی» (List) از کلمات به ما برمی‌گرداند! (اگه مقداری به آن ندهیم بر اساس فاصله کلمه به کلمه با ما خروجی لیست میدهد)

```
1. sentence = "Welcome to the Python Guild"
2.
3. # Split the string by spaces
4. word_list = sentence.split()
5.
```

```
6. print(word_list)
7. # Output: ['Welcome', 'to', 'the', 'Python', 'Guild']
```

حالا که ما یک «لیست» از کلمات داریم، چطور می‌توانیم تعداد کلمات را بشماریم؟ دقیقاً با

$$\text{! len(word\_list)}$$

۲. هنر نمایش: سه طلسما جادویی برای قالب‌بندی (Formatting) تا به حال، اگر می‌خواستیم متن و متغیر را با هم چاپ کنیم، کارمان کمی زشت و سخت بود. ما مجبور بودیم از + برای چسباندن استفاده کنیم و اعداد را با () به متن تبدیل کنیم:

```
1. # The "Ugly Way" (from Chapter 8)
2. name = "Aria"
3. score = 100
4. print("Adventurer: " + name + " - Score: " + str(score))
```

این کد کار می‌کند، اما خوانا نیست و به راحتی باعث خطأ می‌شود. برای حل این مشکل، جادوگران پایتون سه طلسما قدرتمند اختراع کرده‌اند:

### طلسم اول: جادوی باستانی (قالب‌بندی با %)

این قدیمی‌ترین طلسما است. در این روش، تو در طومار خودت «جانگهدار» (Placeholder) می‌گذاری (مثل %s برای متن‌ها و %d برای اعداد صحیح) و سپس متغیرها را با استفاده از علامت % به آن پاس می‌دهی.

```
1. # The "Ancient Way" (%)
2. name = "Aria"
3. score = 100
4.
5. # %s is a placeholder for a String
6. # %d is a placeholder for a Decimal (Integer)
7. print("Adventurer: %s - Score: %d" % (name, score))
```

نتیجه: Adventurer: Aria - Score: 100 این طلسما هنوز هم در برخی طومارهای قدیمی دیده می‌شود، اما استفاده از آن به خاطر خوانایی پایین دیگر توصیه نمی‌شود.

## طلسم دوم: جادوی مدرن (متده `format()`)

این طلسما، نسخه‌ی بسیار قدرتمندتر و انعطاف‌پذیرتر است. در این روش، تو از آکولا德 {} به عنوان «جانگهدار» استفاده می‌کنی و سپس طلسما `format()`. را در انتهای طومار صدا می‌زنی.

```
1. # The "Modern Way" (.format)
2. name = "Aria"
3. score = 100
4.
5. # The {} are placeholders
6. print("Adventurer: {} - Score: {}".format(name, score))
```

نتیجه: Adventurer: Aria - Score: 100 این طلسما بسیار خواناتر است و حتی به تو اجازه

## طلسم سوم: جادوی استاد (f-strings)

این جدیدترین، تمیزترین و خواناترین طلسما است که در پایتون مدرن استفاده می‌شود. کافیست قبل از علامت نقل قول "، یک حرف f بگذاری.

حالا، می‌توانی متغیرهایت را مستقیماً داخل متن و درون آکولا德 {} قرار دهی!

```
1. # The "Wizard's Way" (f-string)
2. name = "Aria"
3. score = 100
4.
5. # Notice the 'f' at the start
6. print(f"Adventurer: {name} - Score: {score}")
```

نتیجه: Adventurer: Aria - Score: 100

کدام را استفاده کنیم؟ از این به بعد در ماجراجویی‌مان، ما همیشه از f-strings استفاده خواهیم کرد، چون بهترین طلسما موجود است. اما دانستن دو طلسما دیگر به تو کمک می‌کند تا طومارهای نوشته شده توسط جادوگران دیگر را هم بفهمی.

### ۳. پروژه: ساخت تحلیلگر متن

ماجراجو، تو آماده‌ای! ما می‌خواهیم با استفاده از طلسماهی جدیدمان (()) و (f-strings split) و طلس قدمی‌مان (len)، یک طومار باستانی را تحلیل کنیم.

کد طلس شده (The Ancient Scroll): ما یک طومار باستانی داریم که در یک متغیر چندخطی ذخیره شده است. (ما با استفاده از سه علامت نقل قول """ می‌توانیم متنی بسازیم که چندین خط داشته باشد).

```
1. # This is our input text
2. scroll_text = """
3. The dragon sleeps in the mountain's heart.
4. We must retrieve the magic sword.
5. Only the brave will succeed.
6. """
```

مأموریت: برنامه‌ای بنویس که این متن را بگیرد و دو چیز را بشمارد:

تعداد کل کاراکترها (با احتساب فاصله‌ها و خطوط جدید).  
تعداد کل کلمات.

نقشه و طرح اولیه (قبل از کدنویسی)  
ورودی: ما متغیر scroll\_text را به عنوان ورودی داریم.

شمارش کاراکترها: این آسان است. ما فقط به طلس (len) روی scroll\_text نیاز داریم.

شمارش کلمات (تله!): ما نمی‌توانیم (len) را مستقیماً روی متن اجرا کنیم. اول باید طلس (split) را روی scroll\_text اجرا کنیم تا یک «کوله پشتی» (List) از کلمات به دست آوریم.

شمارش نهایی کلمات: سپس، طلسه `(len)` را روی آن «کوله پشتی» جدید اجرا می‌کنیم.

نمایش: نتایج را با f-strings (بهترین طلسه قالب‌بندی) به زیبایی چاپ می‌کنیم.

چالش‌های فکری (تمرین تو)

چالش ۱: طومار متغیر `scroll_text` بالا در فایل `project.py` خود کپی کن.

چالش ۲: شمارش کاراکترها چطور از `(len)` برای پیدا کردن تعداد کل کاراکترهای استفاده می‌کنی؟ نتیجه را در متغیری به نام `char_count` ذخیره کن.

چالش ۳: شکستن طومار به کلمات چطور از طلسه `.split()` روی `scroll_text` استفاده می‌کنی تا یک «لیست» از کلمات به دست آوری؟ نتیجه را در متغیری به نام `word_list` ذخیره کن. (می‌توانی `print(word_list)` را اجرا کنی تا بینی چه شکلی شده!)

چالش ۴: شمارش کلمات چطور از `(len)` روی `word_list` (که در چالش ۳ ساخته) استفاده می‌کنی تا تعداد کل کلمات را پیدا کنی؟ نتیجه را در متغیر `word_count` ذخیره کن.

چالش ۵: نمایش نتایج با f-strings چطور با استفاده از f-strings (که در بخش ۲ یاد گرفتیم) نتایج را به این شکل زیبا چاپ می‌کنی؟

```
1. --- Text Analysis Results ---
2. Total Characters: 96
3. Total Words: 17
```

(نکته: عدد دقیق شما ممکن است بسته به فاصله‌هایی که کپی کرده‌اید کمی متفاوت باشد، این اصلاً مهم نیست).

وقت تمرین! قبل از دیدن کد نهایی، سعی کن خودت این معما را حل کنی.

### کد پروژه نهایی (در project.py بنویس):

```
1. # --- The Ancient Scroll Analyzer ---
2.
3. # 1. The Input Scroll
4. scroll_text = """
5. The dragon sleeps in the mountain's heart.
6. We must retrieve the magic sword.
7. Only the brave will succeed.
8.
9.
10. print("--- Analyzing Ancient Scroll ---")
11. print(scroll_text)
12. print("=====")
13.
14. # 2. Challenge 2: Count Characters
15. # len() on a string counts all characters
16. char_count = len(scroll_text)
17.
18. # 3. Challenge 3: Split into words
19. # .split() turns the string into a list of words
20. word_list = scroll_text.split()
21.
22. # 4. Challenge 4: Count Words
23. # len() on a list counts all items (words) in it
24. word_count = len(word_list)
25.
26. # 5. Challenge 5: Display results with f-strings
27. print("--- Text Analysis Results ---")
28. print(f"Total Characters: {char_count}")
29. print(f"Total Words: {word_count}")
30. print("=====")
```

اجرا کن و نتیجه را ببین!

آفرین! تو همین الان یک طومار باستانی را «رمزگشایی» کردی. تو یاد گرفتی که «طومارها» (Strings) فقط یک متن ساده نیستند، بلکه ابزارهای قدرتمندی با طلسهای خاص خودشان (.split()) هستند. تو همچنین سه طلس مختلف قالب‌بندی را یاد گرفتی و می‌دانی که چرا بهترین انتخاب یک جادوگر مدرن است. f-strings

در فصل بعد، ما بزرگترین چالش خود را تا به امروز انجام خواهیم داد: به ابزارهای خود «حافظه‌ی دائمی» خواهیم داد!



# فصل ۱۱: طلسم ضد فراموشی: کار با فایل‌ها

**مأموریت فصل:** ارتقاء برنامه «مدیر لیست خرید» تا با بستن برنامه، اطلاعات از بین نرود!

## نقشه راه

- چطور از شر "کوش" کردن خلاص شویم؟ مدیریت خطا با try و except.
- خواندنی از طومار ابدی: خواندن و نوشتן در فایل‌های متنی (txt.).
- زبان جادویی جهانی (JSON): یک روش استاندارد برای ذخیره داده‌های ساختاریافته.
- پروژه: ارتقاء لیست خرید فصل ۵ برای ذخیره و بازیابی اطلاعات در یک فایل JSON

## ۱. چطور از شر "کرش" کردن خلاص شویم؟ (try و except)

قبل از اینکه به «طومارهای دائمی» (فایل‌ها) دست بزنیم، باید یک «طلسم محافظتی» حیاتی یاد بگیریم.

کار کردن با فایل‌ها خطرناک است. چرا؟

چه اتفاقی می‌افتد اگر بخواهیم فایلی را بخوانیم که هنوز وجود ندارد؟

چه اتفاقی می‌افتد اگر بخواهیم در فایلی بنویسیم که اجازه‌ی این کار را نداریم؟

در هر دو حالت، برنامه «کرش» می‌کند و با یک خطای قرمز ترسناک متوقف می‌شود!

برای جلوگیری از این فاجعه، ما از یک «سپر جادویی» به نام try...except استفاده می‌کنیم.

try (امتحان کن): ما کد خطرناک را در این بلوک قرار می‌دهیم.  
except (مگر اینکه): ما به پایتون می‌گوییم «اگر در بلوک try مشکلی پیش آمد، به جای کرش کردن، این بلوک را اجرا کن.»

بیا با مثالی که از فصل ۴ می‌شناسیم، این طلسمن را امتحان کنیم. اگر کاربر به جای عدد، کلمه "hello" را وارد کند، int() کرش می‌کند:

```

1. # --- The "Unsafe" Way (Crashes!) ---
2. # print("Enter a number:")
3. # user_input = input()
4. # number = int(user_input) # This line will CRASH if user types "hello"
5. # print("Your number is: " + str(number))
6.
7. # --- The "Safe Way" (with try...except) ---
8. print("Enter a magic number:")
9. user_input = input()
10.
11. try:
12.     # 1. We "try" to run the dangerous code
13.     number = int(user_input)

```

```

14.     print(f"Your magic number is: {number}")
15. except:
16.     # 2. If the 'try' block fails, this code runs instead
17.     print("That wasn't a number, adventurer! Spell failed.")

```

اجرا کن و نتیجه را ببین! اگر "hello" را تایپ کنی، برنامه دیگر کرش نمی‌کند، بلکه به زیبایی پیام خطا را نمایش می‌دهد. ما برای خواندن فایل‌ها دقیقاً به همین سیر نیاز داریم.

## ۲. خواندن و نوشتن در فایل‌های متنه (txt.)

ساده‌ترین راه برای ذخیره اطلاعات، نوشتن آن‌ها روی یک «طومار متنه» (txt.) است.

ورد (with open(...))

برای باز کردن یک فایل، ما از طلسمند (open) استفاده می‌کنیم. این ورد جادویی یک «حلقه‌ی امن» ایجاد می‌کند که فایل را باز می‌کند، به ما اجازه می‌دهد با آن کار کنیم، و به‌طور خودکار در انتها آن را می‌بندد (حتی اگر کد ما کرش کند).

ما به این طلسمند دو چیز می‌دهیم:

- نام فایل (مثالاً "scroll.txt").

- (Mode) (حالت):

"w": حالت Write (نوشتن). هشدار: این حالت فایل قبلی را پاک می‌کند و از نو می‌نویسد!

"r": حالت Read (خواندن).

"a": حالت Append (اضافه کردن). این حالت، متن جدید را به انتهای فایل اضافه می‌کند

بدون اینکه آن را پاک کند.

```

1. # --- 1. Write (Overwrites the file) ---
2. # 'f' is just a variable name for our open file
3. with open("scroll.txt", "w") as f:
4.     f.write("The first line of the ancient scroll.\n")
5.     f.write("The dragon sleeps here.\n")
6.

```

```

7. # --- 2. Append (Adds to the end) ---
8. with open("scroll.txt", "a") as f:
9.     f.write("We must be quiet!")
10.
11. # --- 3. Read (Gets all content) ---
12. print("Reading the scroll:")
13. with open("scroll.txt", "r") as f:
14.     content = f.read()
15.     print(content)

```

اجرا کن و نتیجه را ببین! حالا به پوشه‌ی پروژه‌ات نگاه کن. یک فایل جدید به نام scroll.txt آنجاست! جادوی تو اکنون دائمی شده است.

### ۳. زبان جادویی جهانی (JSON)

طلسم بالا یک مشکل بزرگ دارد. اگر ما «کوله پشتی» (List) خود را در آن بنویسیم، چه اتفاقی می‌افتد؟

طلسم my\_list = ["sword", "shield"] f.write(my\_list) بالا می‌کند! (TypeError)

طلسم write() فقط «طومار» (String) قبول می‌کند، نه «کوله پشتی» (List) یا «کتاب جادویی» (Dictionary).

ما به یک «زبان جادویی جهانی» نیاز داریم تا بتوانیم داده‌های ساختاریافته‌ی خود (مثل لیست‌ها و دیکشنری‌ها) را به شکلی ذخیره کنیم که بعداً بتوانیم دقیقاً به همان شکل آن‌ها پس بگیریم.

این زبان JSON (JavaScript Object Notation) نام دارد. این یک فرمت متنی استاندارد است که شبیه دیکشنری‌ها و لیست‌های پایتون است.

پایتون یک کتابخانه جادویی داخلی به نام json دارد که این کار را برای ما انجام می‌دهد.

```
1. import json
```

ما فقط به دو طلسم جدید از این کتابخانه نیاز داریم:  
`json.dump(data, file)`: داده‌های پایتون (مثل لیست) را می‌گیرد و آن‌ها را به صورت متن JSON در فایل «تخلیه» (Dump) می‌کند.

`json.load(file)`: متن JSON را از فایل می‌خواند و آن را دوباره به داده‌های پایتون (لیست یا دیکشنری) تبدیل می‌کند.

```
1. import json # Don't forget to import!
2.
3. # Our data (a Python List)
4. shopping_list = ["sword", "shield", "health_potion"]
5.
6. # --- 1. Save (Dump) the list to a JSON file ---
7. with open("backpack.json", "w") as f:
8.     # Dump our 'shopping_list' data INTO the file 'f'
9.     json.dump(shopping_list, f)
10.
11. print("Backpack saved!")
12.
13. # --- 2. Load (Load) the list FROM the JSON file ---
14. with open("backpack.json", "r") as f:
15.     # Load the JSON data FROM the file 'f' and turn it back into Python
16.     loaded_list = json.load(f)
17.
18. print("Backpack loaded:")
19. print(loaded_list) # Output: ['sword', 'shield', 'health_potion']
```

ما موفق شدیم! ما یک «کوله پشتی» کامل را ذخیره و بازیابی کردیم!

#### ۴. طلسی زبان‌های باستانی: جادوی encoding (خصوصی فارسی)

طلسی بالا یک مشکل پنهان دارد. کد ما برای متن انگلیسی عالی کار کرد. اما اگر بخواهیم کلمات فارسی (مثل کلمات «متترجم» فصل ۷) را ذخیره کنیم چه؟

```
1. # --- Potential Trap! ---
2. with open("farsi_scroll.txt", "w") as f:
3.     f.write("سلام ماجراجو!")
```

در بسیاری از سیستم‌ها (به خصوص ویندوز)، این کد یا با خطای UnicodeEncodeError کرش می‌کند یا فایلی با متن ??????? می‌سازد!

چرا؟ چون غول چراغ جادو (کامپیوتر) به طور پیش‌فرض از یک «کتاب زبان» قدیمی استفاده می‌کند که فقط حروف انگلیسی را می‌فهمد. ما باید به او دستور دهیم که از «کتاب زبان جهانی» (UTF-8) استفاده کند.

این کار با اضافه کردن یک آرگومان سوم به نام "encoding="utf-8" انجام می‌شود.

```
1. # --- The "Safe Way" for all languages ---
2.
3. # Writing Farsi safely
4. with open("farsi_scroll.txt", "w", encoding="utf-8") as f:
5.     f.write("سلام ماجراجو!")
6.
7. # Reading Farsi safely
8. with open("farsi_scroll.txt", "r", encoding="utf-8") as f:
9.     content = f.read()
10.    print(content) # Output: !سلام ماجراجو!
```

قانون ماجراجویی: از این به بعد، همیشه هنگام کار با فایل‌های متنی از "encoding="utf-8" استفاده کن تا مطمئن شوی طلسی تو برای همه‌ی زبان‌ها کار می‌کند.

## ۵. پروژه: ارتفاع «مدیر لیست خرید»

ماجراجو، تو آماده‌ای! ما می‌خواهیم «طلسم ضد فراموشی» را روی پروژه فصل ۵ خود اجرا کنیم.

مأموریت: کاری کن که «مدیر لیست خرید» ما:

هنگام شروع برنامه، لیست قبلی را از یک فایل به نام shopping.json بخواند (Load).

هنگام خروج از برنامه (وقتی کاربر quit را می‌زند)، لیست فعلی را در همان فایل ذخیره (Save) کند.

نقشه و طرح اولیه (قبل از کدنویسی)  
کتابخانه: ما به جادوی json نیاز داریم. پس باید آن را import کنیم.

طلسم بارگذاری (Loading): این بخش خطرناک است! در اولین باری که برنامه را اجرا می‌کنیم، فایل shopping.json هنوز وجود ندارد. اگر سعی کنیم آن را باز کنیم، برنامه «کرش» می‌کند (خطای FileNotFoundError).

راه حل: ما باید کد «بارگذاری» (load) را در یک سپر try...except قرار دهیم.

کجا؟ قبل از اینکه حلقه while اصلی برنامه شروع شود.

try: سعی کن فایل shopping\_list را با json.load() بخوانی و در متغیر shopping\_list بثبیزی.

except: اگر فایل پیدا نشد (یا هر خطای دیگری داد)، یعنی این اولین اجرای ماست. پس یک لیست خالی بساز: `[:] = shopping_list`

طلسم ذخیره (Saving):

کجا؟ داخل حلقه while، در بخشی که کاربر دستور quit را وارد می‌کند.

کی؟ درست قبل از اینکه با break یا is\_running = False از حلقه خارج شویم.

چطور؟ فایل shopping.json را در حالت "w" (نوشتن) باز کن و با json.dump()، متغیر shopping\_list را در آن ذخیره کن.

چالش‌های فکری (تمرین تو)

چالش ۱: کپی کردن کد نهایی «مدیر لیست خرید» از فصل ۵ را در فایل project.py جدیدت کپی کن.

چالش ۲: وارد کردن جادو در خط اول فایل، طلسمن import json را اضافه کن.

چالش ۳: طلسمن بارگذاری (Loading) با سپر کد shopping\_list = [] را که در بالای حلقه while بود، پیدا کن. آن را با این بلوک جادویی جایگزین کن:

```
1. # --- 1. Load the backpack (with safety shield) ---
2. try:
3.     with open("shopping.json", "r") as f:
4.         shopping_list = json.load(f) # Try to load the old list
5.         print("Backpack loaded from save file!")
```

```

6. except:
7.     print("No save file found. Starting with an empty backpack.")
8.     shopping_list = [] # Start fresh

```

چالش ۴: طلسی ذخیره (Saving) داخل حلقه while، بلوک را پیدا کن. قبل از خط `is_running = False`، این کد را اضافه کن:

```

1. # --- Save the list before quitting ---
2. with open("shopping.json", "w") as f:
3.     json.dump(shopping_list, f)
4. print("Backpack saved. Goodbye!")
5. # --- Now we can quit ---
6. is_running = False

```

وقت تمرین! سعی کن این ۴ چالش را خودت انجام دهی. کد نهایی تو باید شبیه کد زیر باشد:

کد پروژه نهایی (در `project.py` بنویس):

```

1. # --- The PERMANENT Shopping List Manager ---
2. import json # Step 1: Import the magic
3.
4. # --- Step 2: Load the list (with try/except shield) ---
5. try:
6.     with open("shopping.json", "r") as f:
7.         shopping_list = json.load(f) # Try to load the old list
8.         print("Backpack loaded from save file!")
9. except:
10.     print("No save file found. Starting with an empty backpack.")
11.     shopping_list = [] # Start fresh
12.
13. # (The rest of the code is from Chapter 5)
14. is_running = True
15. print("Welcome to the Shopping List Manager!")
16. print("You can 'add', 'remove', 'show', or 'quit'.")
17.
18. while is_running == True:
19.     print("-----")
20.     command = input("What do you want to do? ")
21.
22.     if command == "add":
23.         item_to_add = input("What item to add? ")
24.         shopping_list.append(item_to_add)
25.         print(item_to_add + " was added to the list!")
26.

```

```

27.     elif command == "remove":
28.         item_to_remove = input("What item to remove? ")
29.         if item_to_remove in shopping_list:
30.             shopping_list.remove(item_to_remove)
31.             print(item_to_remove + " was removed.")
32.         else:
33.             print(item_to_remove + " is not in the list!")
34.
35.     elif command == "show":
36.         print("--- Your Current List ---")
37.         if len(shopping_list) == 0:
38.             print("Your list is empty.")
39.         else:
40.             for item in shopping_list:
41.                 print("- " + item)
42.             print("-----")
43.
44.     elif command == "quit":
45.         # --- Step 3: Save the list before quitting ---
46.         with open("shopping.json", "w") as f:
47.             json.dump(shopping_list, f)
48.         print("Backpack saved. Goodbye, adventurer!")
49.
50.         is_running = False # This will stop the 'while' loop
51.
52.     else:
53.         print("Invalid command. Please use 'add', 'remove', 'show', or
54. 'quit'.")
55.     print("Manager shutting down.")
56.

```

اجرا کن و نتیجه را ببین! برنامه را اجرا کن. چند آیتم (sword, shield) اضافه کن. سپس quit را تایپ کن. حالا دوباره برنامه را اجرا کن! خواهی دید که برنامه با پیام "Backpack ...loaded" شروع می‌شود و اگر show را تایپ کنی، sword و shield هنوز آنجا هستند!

آفرین! تو همین الان بزرگترین طلسم فراموشی را شکستی. تو به ابزارهایت «حافظه‌ی دائمی» دادی و آن‌ها را از اشیاء موقت به ابزارهای جادویی و ماندگار تبدیل کردی!

# فصل ۱۲: دمیدن روح در کد: خلق موجودات با شیء‌گرایی

مأموریت فصل: طراحی یک سیستم ساده برای مدیریت محصولات یک مغازه جادوگری.

## نقشه راه

- ترکیب داده و رفتار: معرفی کلاس (Class) و شیء (Object).
- "نقشه ساخت" (کلاس): طراحی یک «بلوپرینت» برای موجودی به نام «معجون».(Potion)
- طلسخ خلقت (`__init__`): چطور یک معجون واقعی خلق کنیم؟
- ویژگی‌ها (Attributes): ذخیره داده‌ها (مثل نام و قیمت) روی خود موجود.
- رفتارها (Methods): اضافه کردن «توابع» (جادوها) به خود موجود.

بروژه: ساخت کلاس Potion و خلق چند معجون واقعی از روی آن برای مغازه.

## ۱. ترکیب داده و رفتار: کلاس (Class) و شیء (Object)

تا به حال، ما با «انواع داده» (Data Types) مثل int (سکه)، str (طومار)، list (کوله پشتی) و dict (کتاب دیکشنری) کار می‌کردیم. اینها «نقشه‌های ساختی» بودند که خود پایتون به ما داده بود.

شیء‌گرایی (Object-Oriented Programming) به ما این قدرت را می‌دهد که «انواع داده‌ی» سفارشی خودمان را بسازیم!

ما این کار را با دو مفهوم کلیدی انجام می‌دهیم:

کلاس (Class): این «نقشه ساخت» (Blueprint) یا «دستورالعمل» ماست. کلاس به پایتون می‌گوید: «من می‌خواهم یک نوع داده‌ی جدید به نام "Potion" بسازم. هر Potion یک "نام" و یک "قیمت" خواهد داشت.» این فقط یک نقشه است؛ هنوز هیچ معجون واقعی ساخته نشده.

شیء (Object): این «موجود» واقعی است که از روی آن نقشه ساخته شده. وقتی ما یک معجون واقعی با نام "Health Potion" و قیمت ۵۰ می‌سازیم، آن یک «شیء» یا «نمونه» (Instance) از کلاس Potion است.

می‌توانیم صدها «شیء» (معجون) مختلف از روی یک «کلاس» (نقشه ساخت) بسازیم.

## ۲. "نقشه ساخت" (کلاس) و طلسمن خلقت (`__init__`)

بیایید «نقشه ساخت» (Class) خود را برای «معجون» (Potion) تعریف کنیم. ما از ورد جادویی `class` استفاده می‌کنیم (که با حرف بزرگ شروع می‌شود):

```
1. # --- 1. Defining the Blueprint (The Class) ---
2. # We're telling Python we are creating a new "type" of thing.
3. class Potion:
4.     pass # 'pass' just means "nothing here yet"
```

حالا چطور یک «شیء» واقعی از روی این نقشه بسازیم؟ ما آن را مثل یک تابع «صدای» می‌زنیم:

```
1. # --- 2. Creating Actual Beings (Objects) ---
2. # We "call" the class to create an "instance"
3. potion1 = Potion()
4. potion2 = Potion()
5.
6. print(potion1)
7. print(potion2)
```

اجرا کن و نتیجه را ببین! خروجی شبیه این خواهد بود:>  
<at 0x10a2f3c10><\_\_main\_\_.Potion object at 0x10a2f3c40>

پایتون به ما می‌گوید: «تو دو شیء "Potion" واقعی ساخته‌ای که در دو مکان مختلف در حافظه زندگی می‌کنند.»

طلسم خلقت: `__init__` اما چطور در لحظه‌ی خلقت، به این معجون‌ها نام و قیمت بدهیم؟ ما از یک طلسمن جادویی داخلی به نام `__init__` (مخلف `initialize` به معنی «آماده‌سازی اولیه») استفاده می‌کنیم.

یک «متده» (Method) ویژه است (یک تابع در داخل کلاس) که به طور خودکار هر بار که یک شیء جدید می‌سازیم، اجرا می‌شود.

```
1. class Potion:  
2.     # --- The Constructor Method ---  
3.     # This spell runs automatically when 'Potion()' is called  
4.     def __init__(self):  
5.         print("A new potion was just created!")  
6.  
7. # Let's create two potions  
8. p1 = Potion() # Output: A new potion was just created!  
9. p2 = Potion() # Output: A new potion was just created!
```

### ۳. ویژگی‌ها (Attributes): ذخیره داده‌ها روی self

جادوی \_\_init\_\_ یک ورد کلیدی به نام self دارد.

self چیست؟ یک کلمه جادویی است که به «شیء»‌ای اشاره دارد که همین الان در حال ساخته شدن است.

وقتی ما به تابع \_\_init\_\_ ورودی (پارامتر) می‌دهیم، می‌توانیم از self استفاده کنیم تا آن داده‌ها را روی خود شیء «ذخیره» کنیم. به این داده‌های ذخیره شده «ویژگی» (Attribute) می‌گویند.

```
1. class Potion:  
2.     # Give 'name' and 'price' as arguments at creation  
3.     def __init__(self, name, price):  
4.  
5.         # --- Attributes ---  
6.         # Store the 'name' argument ON 'self' (the object)  
7.         self.name = name  
8.         # Store the 'price' argument ON 'self'  
9.         self.price = price  
10.  
11.        # --- Create (instantiate) objects WITH arguments ---  
12.        # "Health Potion" goes into the 'name' parameter  
13.        # 50 goes into the 'price' parameter  
14.        health_potion = Potion("Health Potion", 50)
```

```

15. mana_potion = Potion("Mana Potion", 70)
16.
17. # --- Access the attributes (data) using the dot ---
18. print(health_potion.name) # Output: Health Potion
19. print(health_potion.price) # Output: 50
20.
21. print(mana_potion.name) # Output: Mana Potion

```

ما موفق شدیم! ما دو موجود زنده خلق کردیم و هر کدام نام و قیمت خودشان را می‌دانند.

## ۴. رفتارها (Methods): افزودن جادو به موجودات

حالا که موجود ما «داده» دارد، بیا به آن «رفتار» هم بدهیم. متدهای (Method) یک «تابع» (Function) است که داخل کلاس زندگی می‌کند.

متدهای همیشه self را به عنوان اولین پارامتر می‌گیرند تا بتوانند به ویژگی‌های (Attributes) خودشان (مثل self.name) دسترسی داشته باشند.

```

1. class Potion:
2.     def __init__(self, name, price, effect):
3.         self.name = name
4.         self.price = price
5.         self.effect = effect # Added a new attribute
6.
7.     # --- This is a METHOD (a behavior) ---
8.     # It MUST have 'self' to access its own attributes
9.     def display_info(self):
10.        print(f"--- {self.name} ---")
11.        print(f"Effect: {self.effect}")
12.        print(f"Price: {self.price} gold")
13.
14. # Create an object
15. health_potion = Potion("Greater Health Potion", 100, "Heals 50 HP")
16.
17. # Call the method (behavior) on the object
18. health_potion.display_info()

```

اجرا کن و نتیجه را ببین!

```
1. --- Greater Health Potion ---
2. Effect: Heals 50 HP
3. Price: 100 gold
```

ما دیگر داده و رفتار را جدا نداریم. خود شیء `health_potion` می‌داند که چطور اطلاعات خودش را نمایش دهد!

## ۵. پروژه: ساخت مغازه جادوگری

ماجراجو، تو آماده‌ای! ما می‌خواهیم با استفاده از جادوی جدید «کلاس» (Class)، موجوداتی به نام «معجون» (Potion) خلق کنیم و آن‌ها را برای فروش در مغازه جادویی خود لیست کنیم.

نقشه و طرح اولیه (قبل از کدنویسی)  
"نقشه ساخت" (Class): ما به یک کلاس به نام `Potion` نیاز داریم.

"طلسم خلقت" (Constructor): این کلاس به یک متدهای `__init__` نیاز دارد که ۳ ورودی بگیرد: `name` و `effect` و `price`.

"ویژگی‌ها" (Attributes): در داخل `__init__`، باید این سه ورودی را روی `self` ذخیره کنیم (مثل `.self.name = name`).

"رفتار" (Method): کلاس ما به یک متدهای `display_info` (یا هر اسم دلخواه) نیاز دارد که `self` را بگیرد و اطلاعات معجون (نام، اثر، قیمت) را با `f-strings` به زیبایی چاپ کند.

магазاه (The Shop): ما یک «کوله پشتی» (List) خالی به نام `magic_shop_stock` می‌سازیم.

خلق موجودات (Objects): ما حداقل ۳ معجون مختلف (شیء) از روی کلاس Potion می‌سازیم (مثلًا p1 = Potion (...)).

پر کردن قفسه‌ها: هر شیء معجون را که می‌سازیم، به لیست magic\_shop\_stock اضافه می‌کنیم. (append)

باز کردن مغازه: ما یک حلقه for می‌نویسیم که روی magic\_shop\_stock می‌گردد و متدهای display\_info را روی هر معجون صدا می‌زنند.

چالش‌های فکری (تمرین تو)

چالش ۱: "نقشه ساخت" چطور Potion class را تعریف می‌کنی؟

چالش ۲: "طلسم خلقت" چطور متدهای \_\_init\_\_ را با چهار پارامتر name, price, effect و self می‌نویسی؟

چالش ۳: "ویژگی‌ها" چطور در \_\_init\_\_، مقادیر name, price و effect را به self.name, self.price و self.effect متصل می‌کنی؟

چالش ۴: "رفتار" چطور متدهای display\_info(self) را می‌نویسی که با f-strings مشخصات معجون را (از روی self.name و...) چاپ کند؟ (یادت باشد یک جداکننده مثل --- هم چاپ کنی).

چالش ۵: ساخت مغازه و خلق معجون‌ها

چطور یک لیست خالی به نام magic\_shop\_stock می‌سازی؟

چطور ۳ شیء Potion واقعی می‌سازی (مثلاً health\_potion, mana\_potion, invisibility\_potion) و آن‌ها را در متغیرهای جداگانه ذخیره می‌کنی؟

چطور با append() این سه شیء را به magic\_shop\_stock اضافه می‌کنی؟

چالش ۶: نمایش محصولات چطور یک حلقه for potion in magic\_shop\_stock می‌نویسی، و داخل حلقه، متده است potion.display\_info() را صدا می‌زنی؟

وقت تمرین! قبل از دیدن کد نهایی، سعی کن خودت این معما را در project.py حل کنی.

کد پروژه نهایی (در project.py بنویس):

```
1. # --- The Adventurer's Magic Shop ---
2.
3. # 1. --- The Blueprint (Class) ---
4. class Potion:
5.
6.     # 2. --- The Constructor (runs on Potion()) ---
7.     def __init__(self, name, price, effect):
8.         # 3. --- The Attributes (Data) ---
9.         self.name = name
10.        self.price = price
11.        self.effect = effect
12.
13.    # 4. --- The Behavior (Method) ---
14.    def display_info(self):
15.        print(f"--- {self.name} ---")
16.        print(f"Effect: {self.effect}")
17.        print(f"Price: {self.price} gold")
18.        print("=====")
19.
20. # 5. --- Create the Shop's Stock (List) ---
21. magic_shop_stock = []
22.
23. print("Creating new potions for the shop...")
24.
25. # 6. --- Create Beings (Objects) ---
```

```
26. health_potion = Potion("Health Potion", 50, "Heals 25 HP")
27. mana_potion = Potion("Mana Potion", 70, "Restores 50 MP")
28. strength_potion = Potion("Potion of Strength", 150, "Grants +5
Strength")
29. invisibility_potion = Potion("Invisibility Elixir", 300, "Grants 30s
invisibility")
30.
31. # 7. --- Fill the Shelves (Append objects to list) ---
32. magic_shop_stock.append(health_potion)
33. magic_shop_stock.append(mana_potion)
34. magic_shop_stock.append(strength_potion)
35. magic_shop_stock.append(invisibility_potion)
36.
37. print("") # Add a blank line
38. print("WELCOME TO THE MAGIC SHOP!")
39.
40. # 8. --- Open the Shop (Loop and call methods) ---
41. for potion in magic_shop_stock:
42.     # Call the display_info() method on EACH potion object
43.     potion.display_info()
```

اجرا کن و نتیجه را ببین! تو همین الان یک سیستم کامل ساختی. تو دیگر داده‌ها و رفتارها را جداگانه مدیریت نمی‌کنی. تو «موجودات» واقعی خلق کردای که هم «داده» (ویژگی) دارند و هم «رفتار» (متدها).

آفرین! تو همین الان به کد خود «روح» دمیدی و وارد دنیای قدرتمند «شیءگرایی» شدی.

# بخش چهارم: آینه‌های جادوگر ارشد

آفرین ماجراجوا! تو در سه بخش اول، طلسم‌های اصلی را یاد گرفتی. حالا زمان آن رسیده که هنر «معماری» جادو را بیاموزی. در این بخش، یاد می‌گیریم چطور کتابخانه طلسم‌های خود را سازماندهی کنیم، زمان را رام کنیم و کدهای خود را مانند یک استاد «پایتونیک» بنویسیم.

# فصل ۱۳: سازماندهی

## کتابخانه جادویی (ساخت مازول‌های شخصی)

مأموریت فصل: فایل project.py ما به یک طومار درهم پیچیده تبدیل شده است! باید یاد بگیریم چطور طلسها و نقشه‌های ساخت (کلاس‌ها) خود را در فایل‌های جداگانه سازماندهی کنیم.

- نقشه راه:
- چرا یک فایل کافی نیست؟ (اصل تفکیک مسئولیت‌ها).
- طلس import شخصی: چطور فایل spells.py خودمان را بسازیم و آن را در main.py وارد (import) کنیم.
- سازماندهی پروژه: معرفی ساختار پوشه‌بندی ساده برای پروژه‌های بزرگ.
- پروژه: معماری «معازه جادوگری»

## ۱. چرا یک فایل کافی نیست؟ (اصل تفکیک مسئولیت‌ها)

ماجراجوی عزیز، تا به امروز ما تمام طلسم‌ها، نقشه‌های ساخت و پروژه‌هایمان را در یک طومار واحد به نام `project.py` نوشته‌ایم. این کار برای شروع عالی بود، اما حالا که به یک جادوگر با تجربه تبدیل شده‌اید، با یک مشکل جدید روبرو می‌شوید: در هم پیچیدگی.

تصور کنید یک جادوگر تمام طلسم‌های خود را از جادوی درمان گرفته تا گلوله‌های آتشین و طلسم‌های پرواز را روی یک طومار بی‌انتهای هزارمتی بنویسد. چه اتفاقی می‌افتد؟

پیدا کردن سخت می‌شود؛ اگر فقط بخواهید طلسم «درمان زخم ساده» را پیدا کنید، باید ساعت‌ها در طومار جستجو کنید.

خطرناک است: هنگام ویرایش طلسم آتش، ممکن است تصادفاً جوهر روی طلسم پرواز بریزید و آن را خراب کنید.

قابل اشتراک‌گذاری نیست: اگر جادوگر دیگری فقط به طلسم‌های درمان شما نیاز داشته باشد، شما نمی‌توانید آن بخشن از طومار را به او بدهید؛ باید کل طومار آشفته را به او بدهید.

در برنامه‌نویسی، این اصل را «تفکیک مسئولیت‌ها» (*Separation of Concerns*) می‌نامیم. هر فایل (که ما آن را مأژول می‌نامیم) باید یک مسئولیت واحد و شفاف داشته باشد.

یک فایل برای نقشه‌های ساخت (کلاس‌ها).

یک فایل برای طلس‌های کمکی (توابع).

و یک فایل اصلی (main.py) که جادو را اجرا می‌کند.

در این فصل، ما یاد می‌گیریم که چطور مانند یک کتابدار، کتابخانه جادویی خود را سازماندهی کنیم.

## ۲. طلس import شخصی (ساخت اولین مazzoل)

ما قبلاً از طلس import برای احضار کتابخانه‌های قدرتمند پایتون مانند random و json استفاده کردیم. خبر خوب این است: هر فایل .py که خودتان می‌سازید، یک «مazzoل» است و شما می‌توانید آن را import کنید!

بیایید امتحان کنیم:

فرض کنید ما یک طلس کمکی داریم که بارها از آن استفاده می‌کنیم (مانند جداکننده فصل ۸). بیایید آن را به کتاب طلس خود منتقل کنیم.

۱. کتاب طلس (spells.py): یک فایل جدید به نام spells.py بسازید و این کد را در آن بنویسید:

```
1. # spells.py
2. # This is our library for helper spells
3.
4. def print_divider(character=="="):
5.     """
6.     Prints a neat divider.
7.     """
8.     print(character * 30)
9.
10. def greet_adventurer(name):
```

```

11. """
12.     Welcomes a new adventurer.
13. """
14.     print(f"Greetings, {name}! Welcome to the guild.")

```

۲. طومار اصلی (main.py): حالا، یک فایل جدید دیگر به نام main.py در همان پوشه بسازید. برای استفاده از آن طلسماها، ما آنها را import می‌کنیم:

```

1. # main.py
2. # Our main program
3.
4. # We import the spells.py module (file) that we just created
5. import spells
6.
7. print("Main program started...")
8.
9. # To use the spells, we must reference the book (module) name
10. # Just like we did with random.choice()
11. spells.greet_adventurer("Aria")
12. spells.print_divider()
13. spells.greet_adventurer("Gandalf")
14. spells.print_divider("-")

```

اجرا کن و نتیجه را ببین! وقتی main.py را اجرا می‌کنید، خروجی این خواهد بود:

```

1. Main program started...
2. Greetings, Aria! Welcome to the guild.
3. =====
4. Greetings, Gandalf! Welcome to the guild.
5. -----

```

ما با موفقیت جادوها یمان را «تفکیک» کردیم! main.py ما اکنون تمیز و خواناست و منطق اصلی را نشان می‌دهد، در حالی که spells.py تمام جزئیات پیاده‌سازی را پنهان می‌کند.

### ۳. پروژه: معماری «مغازه جادوگری»

مأموریت: ما می‌خواهیم «مغازه جادوگری» خود را از صفر بسازیم (مانند فصل ۱۲)، اما این بار مانند یک معمار کد حرفه‌ای. ما «نقشه ساخت معجون» را از «ویترین مغازه» جدا خواهیم کرد.

نقشه و طرح اولیه (قبل از کدنویسی):

فایل نقشه (potions.py): ما به یک فایل جداگانه نیاز داریم که فقط «نقشه ساخت» کلاس Potion را در خود نگه دارد.

فایل مغازه (shop.py): ما به یک فایل اصلی نیاز داریم که «ویترین» ما باشد. این فایل نقشه import Potion را می‌کند، چند معجون واقعی (شیء) از روی آن می‌سازد و آن‌ها را برای فروش نمایش می‌دهد.

چالش‌های فکری (تمرین تو):

چالش ۱: نقشه ساخت (potions.py)

یک فایل جدید به نام potions.py بساز.

درون آن، کلاس Potion را تعریف کن.

«طلسم خلقت» (`__init__`) آن باید سه ورودی بگیرد: `.name, price, effect`:

این ورودی‌ها را به عنوان «ویژگی» (Attributes) روی self ذخیره کن (مثل self.name (= name).

یک «رفتار» (Method) به نام display\_info بساز که مشخصات معجون را به زیبایی چاپ کند.

## چالش ۲: ویترین مغازه (shop.py)

یک فایل جدید به نام shop.py بساز.

چطور می‌توانی «نقشه» (Potion) را از فایل potions وارد کنی؟ (راهنمایی: from .(potions import Potion

یک لیست خالی به نام magic\_shop\_stock بساز.

چطور سه «شیء» معجون واقعی (مثل معجون سلامتی، معجون مانا) با قیمت‌ها و اثرهای مختلف خلق می‌کنی؟

چطور آن‌ها را به لیست magic\_shop\_stock اضافه (append) می‌کنی؟

چطور یک حلقه for می‌نویسی که روی تمام معجون‌های داخل انبار بگرد و طلسماً display\_info() را روی هر کدام صدا بزند؟

وقت تمرین! قبل از دیدن کد نهایی، سعی کن خودت این دو فایل را بسازی.

کد پروژه نهایی:

فایل ۱: potions.py (کتابچه نقشه ساخت ما)

```
1. # potions.py
2. # This module only contains the blueprints (classes)
3. # for magical items.
4.
5. class Potion:
6.     """
7.         A blueprint for a magical potion in the shop.
8.     """
9.
10.    # 1. The Constructor Spell
11.    def __init__(self, name, price, effect):
12.        # 2. The Attributes
13.        self.name = name
14.        self.price = price
15.        self.effect = effect
16.
17.    # 3. The Behavior (Method)
18.    def display_info(self):
19.        """Prints the potion's stats neatly."""
20.        print(f"--- {self.name} ---")
21.        print(f"  Effect: {self.effect}")
22.        print(f"  Price: {self.price} Gold")
23.        print("=====")
24.
```

فایل ۲: shop.py (طلسم اصلی و ویترین مغازه ما)

```
1. # shop.py
2. # The main application for our magic shop.
3.
4. # 1. Summon the blueprint from our other file
5. # We import the Potion class directly from the potions.py file
6. from potions import Potion
7.
8. print("--- Welcome to the Adventurer's Magic Shop! ---")
9. print("Stocking the shelves...")
10.
11. # 2. Create the shop's inventory (a list)
12. magic_shop_stock = []
13.
```

```

14. # 3. Create the beings (Objects) from the blueprint
15. health_potion = Potion(name="Health Potion", price=50, effect="+25 HP")
16. mana_potion = Potion(name="Mana Potion", price=70, effect="+50 MP")
17. strength_potion = Potion(name="Potion of Strength", price=150,
effect="+5 Strength for 3 min")
18.
19. # 4. Fill the shelves
20. magic_shop_stock.append(health_potion)
21. magic_shop_stock.append(mana_potion)
22. magic_shop_stock.append(strength_potion)
23.
24. print("--- Today's Wares ---")
25.
26. # 5. Open the shop and display the items
27. for item in magic_shop_stock:
28.     # We call the display_info() method on EACH potion object
29.     item.display_info()
30.

```

اجرا کن و نتیجه را بین!

فایل `shop.py` را اجرا کن (نه `potions.py`). خواهی دید که برنامه اصلی ما به نگاه می‌کند، کلاس `Potion` را قرض می‌گیرد و مغازه را با موفقیت باز می‌کند.

خروچی:

```

1. --- Welcome to the Adventurer's Magic Shop! ---
2. Stocking the shelves...
3. --- Today's Wares ---
4. --- Health Potion ---
5.   Effect: +25 HP
6.   Price: 50 Gold
7. =====
8. --- Mana Potion ---
9.   Effect: +50 MP
10.  Price: 70 Gold
11. =====
12. --- Potion of Strength ---
13.   Effect: +5 Strength for 3 min
14.   Price: 150 Gold
15. =====

```

آفرین! تو همین الان از یک جادوگر که همه چیز را روی یک طومار می‌نویسد، به یک «معمار» تبدیل شدی که یک کتابخانه جادویی سازمان یافته می‌سازد. کدهای تو اکنون تمیزتر، خواناتر و بی‌نهایت قدر تمندتر شده‌اند. تو «نقشه‌ها» (کلاس‌ها) را از «مناطق اجرایی» (برنامه اصلی) جدا کردی.



# فصل ۱۴: کوله پشتی

## جادویی زمان

**مأموریت فصل:** تا به حال، ابزارهای ما هیچ درکی از «زمان» نداشته‌اند. در این فصل، یاد می‌گیریم چطور زمان حال را بخوانیم، تاریخ‌ها را محاسبه کنیم و برای مأموریت‌هایمان «مهلت» تعیین کنیم.

### نقشه راه:

- احضار `datetime.datetime`: وارد کردن مازول استاندارد
- طلسه `now()`: گرفتن تاریخ و زمان دقیق همین لحظه.
- قالب‌بندی زمان (`strftime`): تبدیل تاریخ به یک «طومار» خوانا.
- سفر در زمان (`timedelta`): محاسبه زمان‌های آینده یا گذشته.
- پروژه: ساخت «اعلان گر مهلت».

## ۱. احضار datetime

تا به حال، طومارهای جادویی ما بی‌زمان بوده‌اند. دفترچه مأموریت ما، مغازه ما... هیچ‌کدام نمی‌دانند الان صبح است یا شب. برای حل این مشکل، باید یکی از قدرتمندترین کتاب‌های جادو در کتابخانه استاندارد پایتون را احضار کنیم: `datetime`.

مانند `random` و `json`، این کتاب طلسماً از قبل بخشی از کتابخانه جادویی شماست. فقط کافیست آن را `import` کنید. این مژول «اشیاء» قدرتمندی (مانند آن‌هایی که در فصل ۱۲ ساختیم) برای مدیریت تاریخ و زمان به ما می‌دهد.

```
1. # We summon the entire datetime spellbook
2. import datetime
```

حالا، اسکریپت ما از جریان زمان آگاه است!

## ۲. طلسماً now() (گرفتن لحظه حال)

اولین طلسماً که هر جادوگر زمان یاد می‌گیرد این است: «الآن ساعت چند است؟»

مژول `datetime` (فایل) شامل یک کلاس قدرتمند همانام `datetime` (نقشه ساخت) است. این نقشه ساخت می‌داند که چگونه یک نقطه زمانی واحد (سال، ماه، روز، ساعت، دقیقه، ثانیه) را ذخیره کند.

برای گرفتن لحظه دقیق، از متدهای `now()` آن استفاده می‌کنیم:

```
1. import datetime
2.
3. # Ask the datetime class for the current moment
4. current_moment = datetime.datetime.now()
5.
6. print("The magical clock reads:")
7. print(current_moment)
```

خروجی (برای شما متفاوت خواهد بود):

1. 2066-11-16 06:24:08.123456

این... زشت است. این یک شیء داده جادویی است، نه یک طومار خوانا. دقیق است، اما برای یک ماجراجوی انسان مفید نیست. ما به طلسم دیگری برای ترجمه این نیاز داریم.

### ۳. قالب‌بندی زمان (طلسم (strftime

آن برچسب زمانی زشت ۲۰۶۶-۱۱-۱۶... یک شیء زمان است. ما باید آن را به یک رشته استفاده (String Format Time) (String) قالب‌بندی کنیم. برای این کار، از طلسم (strftime) می‌کنیم.

این طلسم یک «طومار قالب‌بندی» به عنوان آرگومان می‌گیرد که به آن می‌گوید چگونه زمان را بنویسد، با استفاده از رون‌های جادویی خاص (کدهایی که با٪ شروع می‌شوند):

1. %Y: سال کامل (۲۰۶۶ مثلاً)
2. %m: شماره ماه (۰۱-۱۲)
3. %d: روز ماه (۰۱-۳۱)
4. %A: نام کامل روز هفته (Sunday مثلاً)
5. %B: نام کامل ماه (November مثلاً)

```
1. import datetime
2.
3. current_moment = datetime.datetime.now()
4.
5. # Let's format this object into a readable string
6.
7. # Simple format: YYYY-MM-DD
8. simple_date = current_moment.strftime("%Y-%m-%d")
9. print(f"Simple Date Scroll: {simple_date}")
10.
11. # A more beautiful, adventurous format
12. fancy_date = current_moment.strftime("Today is %A, the %dth of %B, %Y")
13. print(f"Fancy Date Scroll: {fancy_date}")
```

خروجی:

- ```
1. Simple Date Scroll: 2026-11-16
2. Fancy Date Scroll: Today is Sunday, the 16th of November, 2026
```

حالا این طوماری در خور یک پادشاه است!

#### ۴. سفر در زمان (timedelta)

استاد مأموریت ما یک مأموریت جدید به ما می‌دهد: «۷ روز دیگر گزارش بده!» چطور مهلت را محاسبه کنیم؟ ما نمی‌توانیم فقط `current_moment + 7` را به شیء خود اضافه کنیم. این مانند اضافه کردن «شمیر» به «معجون» است — آن‌ها انواع جادویی متفاوتی هستند!

برای اضافه کردن زمان، ما به یک طلس «مدت زمان» خاص نیاز داریم. این `timedelta` نامیده می‌شود. یک شیء `timedelta` یک بازه زمانی را نشان می‌دهد (مثلاً «۷ روز» یا «۳ ساعت»).

```
1. # We need both datetime and timedelta for this
2. import datetime
3.
4. # Get the current moment
5. now = datetime.datetime.now()
6. print(f"Quest accepted on: {now.strftime('%Y-%m-%d')}")  

7.
8. # 1. Create a "duration" spell
9. quest_duration = datetime.timedelta(days=7)
10.
11. # 2. Add the duration to the current date
12. deadline = now + quest_duration
13.
14. print(f"Quest deadline is: {deadline.strftime('%Y-%m-%d')}")  

15.
16. # You can also travel to the past!
17. three_hours_ago = now - datetime.timedelta(hours=3)
18. print(f"We started 3 hours ago, at:  

{three_hours_ago.strftime('%H:%M')}"")
```

خروجی:

- ```
1. Quest accepted on: 2025-11-16 Quest deadline is: 2025-11-23
2. We started 3 hours ago, at: 03:24
```

## ۵. پروژه: اعلان‌گر مهلت (The Deadline Notifier)

مأموریت: استاد مأموریت ما فراموشکار است. ما باید یک ابزار ساده بسازیم که تاریخ امروز و تاریخ هر مأموریت «۷ روزه» را به او بگوید. این طلسم برای مأموریت نهایی ما (فصل ۱۶) حیاتی خواهد بود!

چالش‌ها (نوبت شما):

چالش ۱: چگونه ابزارهای مورد نیاز خود را از کتابخانه `datetime` وارد (`import`) می‌کنید؟

چالش ۲: چگونه تاریخ و زمان فعلی را دریافت می‌کنید و آن را در متغیری به نام `today` ذخیره می‌کنید؟

چالش ۳: چگونه یک طلسم مدت زمان برای دقیقاً ۷ روز ایجاد می‌کنید و آن را در `one_week` ذخیره می‌کنید؟

چالش ۴: چگونه `today` و `one_week` را با هم جمع می‌کنید تا `deadline` را پیدا کنید؟

چالش ۵: چگونه `today` و `deadline` را هر دو به روشی زیبا (مثلًا: "Today is... The deadline is..." ) با استفاده از `strftime` چاپ (`print`) می‌کنید؟

زمان تمرین! به فایل project.py خود بروید و سعی کنید این را خودتان قبل از دیدن طومار نهایی بسازید.

### طومار نهایی پروژه (project.py)

```

1. # --- The Quest Deadline Notifier ---
2.
3. # 1. Summon the magic
4. # We import the main class and the duration class
5. import datetime
6.
7. # 2. Get the current date (our "now" spell)
8. today = datetime.datetime.now()
9.
10. # 3. Create the duration (our "timedelta" spell)
11. one_week = datetime.timedelta(days=7)
12.
13. # 4. Calculate the future (our "time travel" spell)
14. deadline = today + one_week
15.
16. # 5. Format the scrolls (our "strftime" spell)
17. today_formatted = today.strftime("%A, %B %d")
18. deadline_formatted = deadline.strftime("%A, %B %d")
19.
20. # Display the results
21. print("--- Quest Master's Log ---")
22. print(f"Quest assigned on: {today_formatted}")
23. print(f"Quest deadline is: {deadline_formatted}")
24. print("=====")
```

خروجی:

```

1. --- Quest Master's Log ---
2. Quest assigned on: Sunday, November 16
3. Quest deadline is: Sunday, November 23
4. =====
```

فوق العاده است! تو نه تنها زمان را خواندی، بلکه آن را خم کردی. تو با موفقیت یک لحظه را ثبت کردی، در زمان سفر کردی و آن را برای خواندن همه ترجمه کردی. کوله پشتی جادویی تو اکنون بی‌نهایت قدرتمندتر شده است.

# فصل ۱۵: آینین قبیله پایتون

## هنر کدنویسی پایتونیک

مأموریت فصل: بازگشت به کدهای قبلی. حالا که با تجربه‌ایم، طلسم‌های قدیمی خود را بازنویسی می‌کنیم تا سریع‌تر، خواناتر و «پایتونیک» (Pythonic) شوند.

نقشه راه:

- ذن پایتون (The Zen of Python): آشنایی با اصول و آینینی که جادوگران ارشد پایتون به آن متعهد هستند.
- طلسم enumerate: شمارش پایتونیک (جایگزین حلقه‌های range(len())).
- طلسم zip: جفت کردن طومارها (پیمایش همزمان دو لیست).
- طلسم تک خطی (List Comprehensions): جادوی ساختن لیست‌ها در یک خط نفس‌گیر.
- جادوی ناشناس (Lambda): ساخت توابع یکبار مصرف و بی‌نام.
- پروژه: کیمیاگری لیست (ترکیب جادوهای پایتونیک).

## ۱. ذن پایتون (The Zen of Python)

ماجراجوی عزیز، تا به حال یاد گرفته‌ایم که کدی بنویسیم که «کار کند». اما در قبیله پایتون، این کافی نیست. کد ما باید «زیبا»، «خوانا» و «ساده» باشد.

جادوگران ارشد پایتون، فلسفه‌ای دارند که به آن «ذن پایتون» می‌گویند. این فلسفه در خود پایتون پنهان شده است. در یک اسکریپت پایتون (یا حتی در حالت تعاملی)، این طلس را اجرا کن:

```
1. import this
```

طوماری ظاهر می‌شود که با این کلمات آغاز می‌شود:

- 1. [The Zen of Python, by Tim Peters](#)
- 2. Beautiful is better than ugly.
- (آشکار بهتر از پنهان است)
- 3. Explicit is better than implicit.
- (ساده بهتر از پیچیده است)
- 4. Simple is better than complex.
- (خوانایی اهمیت دارد)
- 5. Readability counts.

در این فصل، ما یاد می‌گیریم که چطور با رعایت این آیین، از یک «شاگرد» به یک «استاد پایتونیک» تبدیل شویم.

## ۲. طلس (شمارش پایتونیک) enumerate

مشکل: فرض کنید می‌خواهیم در کوله‌پشتی (لیست) خود بگردیم و هر آیتم را همراه با شماره ردیف آن چاپ کنیم (مثلاً "۱. شمشیر").

راه شاگرد (راه غیر پایتونیک): یک شاگرد تازه‌کار (که شاید از سرزمه‌های دیگر مثل C++ آمده باشد) ممکن است چنین طلسی بنویسد. این کد کار می‌کند، اما زشت و غیر پایتونیک است:

```

1. # The "Clunky" Way
2. backpack = ["Sword", "Shield", "Health Potion"]
3. print("Backpack Inventory (Clunky):")
4.
5. i = 0
6. for item in backpack:
7.     print(f"{i}: {item}")
8.     i = i + 1

```

راه استاد (طلسم enumerate): یک استاد پایتونیک می‌داند که برای این کار یک طلسم داخلی و زیبا وجود دارد: enumerate. این طلسم، لیست شما را می‌گیرد و آن را به لیستی از «جفت‌ها» (ایندکس، آیتم) تبدیل می‌کند.

```

1. # The "Pythonic" Way
2. backpack = ["Sword", "Shield", "Health Potion"]
3. print("Backpack Inventory (Pythonic):")
4.
5. # enumerate gives us (index, item) pairs
6. # We use tuple unpacking to catch them! [cite: 1248-1250]
7. for index, item in enumerate(backpack):
8.     print(f"{index}: {item}")

```

Sword 1: Shield 2: Health Potion ::

کد دوم خواناتر، کوتاه‌تر و «ساده‌تر» است. (Simple is better than complex).

### ۳. طلسم zip (جفت کردن طومارها)

مشکل: فرض کنید دو طومار (لیست) جداگانه داریم: یکی برای نام ماجراجویان و دیگری برای امتیازات آنها. چطور می‌توانیم آنها را با هم چاپ کنیم؟

```

1. adventurers = ["Aria", "Kael", "Lyra"]
2. scores = [100, 80, 75]

```

راه شاگرد: باز هم، شاگرد به سراغ ایندکس‌های عددی می‌رود. این کد شکننده است (اگر طول لیست‌ها متفاوت باشد چه؟) و خوانایی ندارد.

```
1. # The "Clunky" Way
2. print("Leaderboard (Clunky):")
3. for i in range(len(adventurers)):
4.     print(f"{adventurers[i]}: {scores[i]}")
```

راه استاد (طلسم zip): استاد از طلسم zip استفاده می‌کند. zip مانند یک زیپ لباس، دو لیست را می‌گیرد و آیتم‌های متناظر آنها را به صورت «جفت» به هم می‌دوزد.

```
1. # The "Pythonic" Way
2. print("Leaderboard (Pythonic):")
3.
4. # zip creates pairs: ("Aria", 100), ("Kael", 80), ...
5. for name, score in zip(adventurers, scores):
6.     print(f"{name}: {score}")
```

این کد «آشکار» (Explicit) و «زیبا» (Beautiful) است.

#### ۴. طلسم تک خطی (List Comprehensions)

این یکی از قدرتمندترین جادوهای پایتونیک است.

مشکل: فرض کنید لیستی از قیمت غنائم داریم و می‌خواهیم لیستی جدید فقط از آیتم‌های گران‌قیمت (بالای ۶۰ سکه) بسازیم.

راه شاگرد (راه فصل ۵): این روشی است که ما در فصل ۵ یاد گرفتیم. کاملاً درست است، اما ۴ خط طول می‌کشد.

```
1. # The "Old" Way (from Chapter 5)
2. loot_prices = [100, 50, 20, 400, 30]
3. expensive_loot = []
4.
5. for price in loot_prices:
6.     if price > 60:
7.         expensive_loot.append(price)
```

```
8.  
9. print(f"Old Way: {expensive_loot}")
```

راه استاد (طلسم تک خطی): استاد این ۴ خط را در یک خط جادویی خلاصه می‌کند.

```
[ (شرط تو) if (لیست) in (آیتم) (خواهی انجام دهی کاری که می) ]
```

```
1. # The "Pythonic" Way (List Comprehension)  
2. loot_prices = [100, 50, 20, 400, 30]  
3.  
4. expensive_loot = [price for price in loot_prices if price > 60]  
5.  
6. print(f"Pythonic Way: {expensive_loot}")
```

این طلسم حتی می‌تواند داده‌ها را «تغییر» دهد. اگر بخواهیم قیمت همه آیتم‌ها را دو برابر کنیم:

```
1. # Transformation with List Comprehension  
2. doubled_prices = [price * 2 for price in loot_prices]  
3. print(f"Doubled Prices: {doubled_prices}")
```

## ۵. جادوی ناشناس (Lambda)

مشکل: گاهی ما به یک تابع ساده (مانند فصل ۸) فقط برای یک بار استفاده نیاز داریم (مثلاً به عنوان «کلید» برای مرتب‌سازی).

راه شاگرد: فرض کنید می‌خواهیم لیست امتیازات (که در فصل ۶ دیدیم) را بر اساس امتیاز (عدد دوم تاپل) مرتب کنیم.

```
1. # The "Old" Way (using a full function)  
2. leaderboard = [("Kael", 80), ("Aria", 100), ("Lyra", 75)]  
3.  
4. # We need a helper function just for sorting  
5. def get_score(entry):
```

```

6.     return entry[1] # Returns the score
7.
8. leaderboard.sort(key=get_score)
9. print(f"Sorted (Old): {leaderboard}")

```

راه استاد (طلسم lambda): استاد برای چنین تابع ساده‌ای، یک تابع کامل def نمی‌سازد. او از lambda (یک تابع ناشناس و یکخطی) استفاده می‌کند.

(آرگومان‌ها): (کاری که انجام می‌دهد) lambda

```

1. # The "Pythonic" Way (using lambda)
2. leaderboard = [("Kael", 80), ("Aria", 100), ("Lyra", 75)]
3.
4. # We create an anonymous function "on the fly"
5. leaderboard.sort(key=lambda entry: entry[1]) # Sort by the 2nd item
(score)
6.
7. print(f"Sorted (Pythonic): {leaderboard}")

```

## ۶. پروژه: کیمیاگری لیست (List Alchemy)

مأموریت: ما لیستی از معجون‌ها در مغازه جادویی خود داریم (به شکل دیکشنری). ما می‌خواهیم فقط نام معجون‌های «درمانی» (Heal) را استخراج کرده و نام آن‌ها را با حروف بزرگ چاپ کنیم.

طومارداده‌ها:

```

1. # Our inventory of potions
2. potions = [
3.     {"name": "Lesser Healing", "type": "Heal", "price": 50},
4.     {"name": "Mana Potion", "type": "Mana", "price": 70},
5.     {"name": "Greater Healing", "type": "Heal", "price": 150},
6.     {"name": "Invisibility", "type": "Stealth", "price": 200}
7. ]

```

راه شاگرد (استفاده از حلقه for ساده):

```

1. # --- The Apprentice's Way ---
2. print("Apprentice's Result:")
3. healing_potion_names = []
4. for potion in potions:
5.     if potion["type"] == "Heal":
6.         name_upper = potion["name"].upper()
7.         healing_potion_names.append(name_upper)
8.
9. print(healing_potion_names)

```

راه استاد (استفاده از طلسم تکخطی): حالا نوبت شماست، ماجراجو. سعی کنید طلسم بالا را با استفاده از یک List Comprehension بازنویسی کنید.

راهنما: شما هم به «تغییر» (upper()) و هم به «فیلتر» (if) در یک خط نیاز دارید.

طومار نهایی پروژه:

```

1. # --- The Master's Way (List Comprehension) ---
2. print("Master's Result:")
3.
4. healing_potion_names = [
5.     potion["name"].upper()           # The Transformation (SELECT)
6.     for potion in potions          # The Iteration (FROM)
7.     if potion["type"] == "Heal"    # The Filter (WHERE)
8. ]
9.
10. print(healing_potion_names)

```

خروجی (هر دو): ['LESSER HEALING', 'GREATER HEALING']

آفرین! تو همین الان ۴ خط کد را به یک خط کد خوانا، قدرتمند و کاملاً «پایتونیک» تبدیل کردی. تو دیگر فقط کد نمینویسی؛ تو در حال «کیمیاگری» هستی.

# نبرد بخش پنجم: نبرد نهایی (پروژه بزرگ ماجراجو)

تو به انتهای آموزش نزدیک می‌شوی و زمان «آزمون نهایی» فرا رسیده است. در این بخش، ما تمام جادوهایی که از فصل ۱ تا ۱۵ یاد گرفته‌ایم را در یک پروژه بزرگ و کاربردی ترکیب می‌کنیم.

# فصل ۱۶: نبرد نهايی: ساخت دفترچه مأموریت ماجراجو

مأموریت بزرگ : ساخت ابزار نهايی یک ماجراجوی فراموشکار: یک «دفترچه مأموریت» (Quest Log) کامل. اين پروژه، مدرک فارغ التحصیلی شما از این ماجراجویی است!

آفرین ماجراجو! تو به پایان این کتاب و «نبرد نهايی» رسیده‌ای. این فصل، آزمون توست. ما قرار است تمام جادوهایی که در این ماجراجویی یاد گرفته‌ایم را با هم ترکیب کنیم تا قدرتمندترین ابزار خود را بسازیم.

## ۱. نگاهی به گذشته (Gathering Your Spells)

قبل از ورود به نبرد، یک ماجراجوی خردمند ابزارهایش را بررسی می‌کند. برای ساختن «دفترچه مأموریت»، ما به تمام جادوهای قدرتمندی که آموخته‌ایم نیاز داریم:

- **کلاس‌ها (فصل ۱۲)**: برای طراحی «نقشه ساخت» (Blueprint) «یک مأموریت.
- **ماژول‌ها (فصل ۱۳)**: برای سازماندهی کد و جدا کردن «نقشه ساخت» از «منطق اصلی».
- **Datetime فصل ۱۴**: برای ثبت خودکار «زمان ایجاد» هر مأموریت.
- **JSON فصل ۱۱**: برای دادن «حافظه ابدی» به مأموریت‌هایمان تا با بستن برنامه پاک نشوند.
- **توابع (فصل ۸)**: برای نوشتن طلسم‌های تمیز و قابل استفاده مجدد (مثل add\_quest, show\_quests).
- **حلقه‌ها (فصل ۴)**: برای ساختن «منوی فرمان» تعاملی برنامه.

## ۲. گزارش شناسایی (The Adventurer's Challenge)

اینجا جایی است که تو را به چالش می‌کشم، ماجراجو!

قبل از اینکه نقشه قلعه را با هم بکشیم، سعی کن خودت آن را شناسایی کنی. این بهترین راه برای سنجیدن مهارت‌های توست.

مأموریت تو (اگر پذیری): به فایل project.py خود برو و سعی کن برنامه‌ای با این مشخصات بسازی:

۱. **نقشه ساخت**: به یک کلاس Quest نیاز داری که title, description, status (وضعیت) و created\_at (تاریخ ایجاد) را نگه دارد.
۲. **منوی فرمان**: برنامه باید یک منوی while True داشته باشد که ۴ گزینه ارائه دهد: «افزودن»، «نمایش همه»، «تکمیل کردن» و «خروج».

۳. حافظه ابدی : وقتی کاربر «خروج» را می‌زند، کل لیست مأموریت‌ها باید در یک فایل quest\_log.json ذخیره شود.

۴. بارگذاری حافظه : وقتی برنامه دوباره اجرا می‌شود، باید فایل quest\_log.json را بررسی کند و تمام مأموریت‌های قبلی را بارگذاری کند.

سعی کن! بین چقدر می‌توانی پیش بروی. اگر در هر بخشی گیر کردی، نگران نباش. در بخش بعدی، ما این قلعه را آجر به آجر با هم خواهیم ساخت.

### ۳. میز معمار (Building Step-by-Step)

آماده‌ای؟ بیا این شاهکار را با هم بسازیم. ما پروژه را به دو فایل تقسیم می‌کنیم (طبق جادوی فصل ۱۳) :

۱. quest\_model.py: نقشه ساخت ما.

۲. main.py: مرکز فرماندهی ما.

#### مرحله ۱: نقشه ساخت (quest\_model.py)

ابتدا، «نقشه ساخت» مأموریت را طراحی می‌کنیم. این کلاس مسئول نگهداری داده‌های یک مأموریت است.

یک فایل جدید به نام quest\_model.py بسازید:

#### quest\_model.py

```
1. # quest_model.py
2. # This file contains the "blueprint" for our quests.
3. # We import the datetime magic (from Chapter 14).
4. import datetime
5.
6. class Quest:
7.     """
8.     Blueprint for a single quest in our Quest Log.
9.     Uses magic from Chapter 12 (Classes).
10.    """
11.
```

```
12.     def __init__(self, title, description):
13.         # 1. Attributes from the user
14.         self.title = title
15.         self.description = description
16.
17.         # 2. Automatic attributes (Chapter 14 magic)
18.         self.created_at = datetime.datetime.now()
19.
20.         # 3. Default status
21.         self.status = "Pending" # Other status: "Complete"
22.
23.     def display_info(self):
24.         """
25.             Prints the details of this quest in a neat format.
26.             Uses magic from Chapter 14 (strftime) and Chapter 10 (f-
strings).
27.         """
28.         # Format the date into a readable string
29.         date_str = self.created_at.strftime("%Y-%m-%d %H:%M")
30.
31.         print(f"    Title: {self.title}")
32.         print(f"    Status: {self.status}")
33.         print(f"    Created: {date_str}")
34.         print(f"    Description: {self.description}")
35.
36.     def complete_quest(self):
37.         """
38.             Changes the status of the quest to "Complete".
39.         """
40.         self.status = "Complete"
41.         print(f"\n[Quest '{self.title}' marked as complete!]")
```

عالی! نقشه ساخت ما کاملاً شد.

میں حلہ ۲: میں کنے فرماندھی (main.py)

حالا، فایل اصلی برنامه را می‌سازیم. این فایل منطق اصلی، منو و جادوهای ذخیره‌سازی را مدبی پیت می‌کند.

یک فایل جدید به نام main.py بسازید. ما آن را تکه به تکه خواهیم ساخت.

**main.py** تکه ۱: احضار جادوها ابتدا، تمام ابزارهای مورد نیازمان را import می کنیم.

```

6. import datetime           # For time magic (Chapter 14)
7. from quest_model import Quest # Our own blueprint! (Chapter 13)
8.
9. # Global variables (our magic scrolls)
10. quest_list = [] # This is our main "backpack" for quests
11. SAVE_FILE = "quest_log.json" # The name of our eternal memory file

```

تکه ۲: طلسم‌های حافظه ابدی این مهم‌ترین بخش است json. نمی‌تواند اشیاء Quest را مستقیماً ذخیره کند. ما باید اشیاء خود را به «دیکشنری» («که JSON می‌فهمد») تبدیل کنیم و هنگام بارگذاری، دیکشنری‌ها را دوباره به اشیاء Quest تبدیل کنیم.

```

1. # --- 2. ETERNAL MEMORY SPELLS (Chapter 11 Magic) ---
2.
3. def load_quests():
4.     """
5.         Loads the quest list from the JSON file at the start.
6.         Uses try...except to handle the first run.
7.     """
8.     global quest_list
9.     try:
10.         with open(SAVE_FILE, "r") as f:
11.             # 1. Load the simple data (list of dictionaries)
12.             data_list = json.load(f)
13.
14.             # 2. Re-create the Quest objects
15.             quest_list = [] # Clear the list before loading
16.             for item in data_list:
17.                 # Create a new quest object
18.                 quest = Quest(item['title'], item['description'])
19.                 # Restore its saved data
20.                 quest.status = item['status']
21.                 quest.created_at =
22.                     datetime.datetime.fromisoformat(item['created_at'])
23.                     quest_list.append(quest)
24.
25.             print(f"Quest Log loaded. {len(quest_list)} quests found.")
26.         except FileNotFoundError:
27.             print("No save file found. Starting a new Quest Log.")
28.         except Exception as e:
29.             print(f"An error occurred while loading: {e}")
30.     def save_quests():
31.         """
32.             Saves the current quest_list to the JSON file before quitting.
33.         """
34.             # 1. We must convert our list of Objects into a list of
Dictionaries

```

```

35.     data_list = []
36.     for quest in quest_list:
37.         # Convert datetime object to a string to save in JSON
38.         date_str = quest.created_at.isoformat()
39.
40.         data_list.append({
41.             "title": quest.title,
42.             "description": quest.description,
43.             "created_at": date_str,
44.             "status": quest.status
45.         })
46.
47.     # 2. Now, save the simple list of dictionaries
48.     try:
49.         with open(SAVE_FILE, "w") as f:
50.             json.dump(data_list, f, indent=4)
51.             print("Quests saved successfully!")
52.     except Exception as e:
53.         print(f"An error occurred while saving: {e}")
54.

```

تکه ۳: طلسم‌های تعاملی حالا، توابع تمیزی می‌نویسیم که هر کدام یک کار در

منوی ما انجام می‌دهند (جادوی فصل ۸).

```

1. # --- 3. INTERACTIVE SPELLS (Chapter 8 Magic) ---
2.
3. def add_quest():
4.     """
5.     Asks the user for quest details and adds a new Quest to the list.
6.     """
7.     print("\n--- Add New Quest ---")
8.     title = input("Enter quest title: ")
9.     description = input("Enter quest description: ")
10.
11.    # Create a new Quest object (using our blueprint)
12.    new_quest = Quest(title, description)
13.
14.    # Add it to our main backpack
15.    quest_list.append(new_quest)
16.    print("\n[Quest added successfully!]")
17.
18. def show_all_quests():
19.     """
20.     Displays all quests in the quest_list.
21.     Uses magic from Chapter 15 (enumerate) and Chapter 12 (methods).
22.     """
23.     print("\n--- Your Current Quest Log ---")
24.     if not quest_list:
25.         print("Your quest log is empty. Time to find adventure!")
26.         return
27.

```

```

28.     # Use 'enumerate' to get a clean index (Chapter 15 magic)
29.     for index, quest in enumerate(quest_list):
30.         print(f"\n--- Quest {index + 1} ---")
31.         quest.display_info() # Call the method from our class!
32.         print("-----")
33.
34. def complete_quest():
35.     """
36.     Asks the user which quest to mark as complete.
37.     """
38.     # Show them the list first
39.     show_all_quests()
40.     if not quest_list:
41.         return # Nothing to complete
42.
43.     try:
44.         choice = input("Enter the number of the quest to complete: ")
45.         # Convert choice to a zero-based index
46.         index = int(choice) - 1
47.
48.         # Check if the index is valid
49.         if 0 <= index < len(quest_list):
50.             quest_list[index].complete_quest()
51.         else:
52.             print("Invalid quest number! No quest was completed.")
53.
54.     except ValueError:
55.         print("That's not a number! Spell failed.")

```

تکه ۴: حلقه اصلی برنامه در نهایت، حلقه while (جادوی فصل ۴) را می‌نویسیم  
تا همه چیز را به هم متصل کند.

```

1. # --- 4. THE MAIN SPELL LOOP (Chapter 4 Magic) ---
2.
3. def main():
4.     """
5.     The main entry point of our program.
6.     """
7.     # Load quests from our eternal memory first!
8.     load_quests()
9.
10.    while True:
11.        print("\n===== Adventurer's Quest Log =====")
12.        print("1. Add a new quest")
13.        print("2. Show all quests")
14.        print("3. Complete a quest")
15.        print("4. Save and Quit")
16.        print("=====")
17.        choice = input("Cast your spell (1-4): ")
18.
19.        if choice == '1':

```

```

20.         add_quest()
21.     elif choice == '2':
22.         show_all_quests()
23.     elif choice == '3':
24.         complete_quest()
25.     elif choice == '4':
26.         # Save before quitting!
27.         save_quests()
28.         print("\nQuest Log saved. Goodbye, adventurer!")
29.         break # Exit the while loop
30.     else:
31.         print("\nInvalid spell! Try again.")
32.
33. # --- 5. RUN THE PROGRAM ---
34. # This standard Python line ensures main() only runs
35. # when the script is executed directly.
36. if __name__ == "__main__":
37.     main()

```

#### ۴. پیروزی نهایی!

ماجراجو، تو موفق شدی! حالا دو فایل در پوشش خود داری:

۱. quest\_model.py نقشه ساخت

۲. main.py مرکز فرماندهی

فایل main.py را اجرا کن:

1. python main.py

#### خروجی اولین اجرا

```

1. : No save file found. Starting a new Quest Log.
2. ===== Adventurer's Quest Log ===== 1. Add a new quest ... 4. Save and
   Quit Cast your spell (1-4):

```

حالا می توانی:

۱. مأموریت ها را **Add** کنی (گزینه ۱).

۲. آنها را **Show** کنی (گزینه ۲).

۳. یک مأموریت را **Complete** کنی (گزینه ۳).

۴. و در نهایت، **Quit** کنی (گرینه ۴).

وقتی خارج شدی، یک فایل جدید به نام quest\_log.json در کنار برنامه‌ات ظاهر می‌شود.  
اگر main.py را دوباره اجرا کنی، خواهی دید:  
**خروجی دومین اجرا**

```
1. : Quest Log loaded. 3 quests found. ===== Adventurer's Quest Log =====  
...
```

تمام مأموریت‌های تو به صورت جادویی بارگذاری شده‌اند!  
آفرین! تو همین الان یک برنامه کامل، سازمان‌یافته، با حافظه دائمی و مبتنی بر کلاس ساختی.  
تو تمام جادوهای اصلی را با هم ترکیب کردی و در «نبرد نهایی» پیروز شدی. تو رسماً یک  
جادوگر پایتون هستی.

# بخش ششم: ورود به تالار پیشگویان (مسیر دانشمند داده)

تبریک می‌گوییم! تو حالا یک «معمار کد» پایتون هستی. اما جادوی واقعی پایتون در «پیش‌بینی آینده» از روی «داده‌ها» نهفته است. در این بخش، ما چهار کتاب جادویی قدرتمند را احضار می‌کنیم تا به یک «دانشمند داده» تبدیل شویم.

# فصل ۱۷: دروازه احضار و آزمایشگاه جادویی (Pip & venv)

مأموریت فصل: یاد می‌گیریم چطور آزمایشگاه جادویی خود را ایزوله کنیم (venv) و سپس طلسم‌های جدیدی که دیگران نوشته‌اند را با pip احضار کنیم.

نقشه راه:

- آزمایشگاه ایزوله (venv): چرا نباید طلسم‌ها را در کتابخانه اصلی نصب کرد؟
- فعال‌سازی آزمایشگاه: طلسم ورود به حباب جادویی.
- دروازه احضار (pip): نصب اولین کتابخانه خارجی.
- طومار نیازمندی‌ها: ساخت requirements.txt برای به اشتراک گذاشتن جادوها.
- پروژه: ساخت «تабلوی اعلانات جادویی» با یک کتابخانه خارجی.

## ۱. آزمایشگاه ایزوله (venv)

ماجراجوی عزیز، به تالار پیشگویان خوش آمدی. در اینجا ما با کتابخانه‌های عظیمی مثل pandas و numpy کار خواهیم کرد. اما قبل از دست زدن به این قدرت‌ها، باید یک قانون مهم امنیتی را یاد بگیریم.

**خطر الودگی جادویی:** تصور کن تمام معجون‌های دنیا را در یک دیگ بزرگ بربیزی. چه می‌شود؟ انفجار! پایتون نصب شده روی کامپیوتر تو (System Python)، همان «دیگ بزرگ» است. اگر برای هر پروژه، کتابخانه‌های مختلف را مستقیم روی آن نصب کنی، به زودی تداخل ایجاد می‌شود و همه چیز از کار می‌افتد.

**راه حل: حباب جادویی (Virtual Environment):** ما برای هر پروژه جدید، یک «آزمایشگاه ایزوله» می‌سازیم. این یک نسخه کپی شده از پایتون است که مخصوص همان پروژه است. اگر این آزمایشگاه منفجر شود، هیچ آسیبی به بقیه سیستم نمی‌رسد. طلسم ساخت آزمایشگاه: ترمینال PowerShell یا Terminal CMD را باز کن، به پوشه پروژه‌ات برو و این دستور را بنویس:

```
# Windows / Mac / Linux
python -m venv my_magic_lab
```

نکته: اگر در مک یا لینوکس هستید و دستور بالا کار نکرد، *python3* را امتحان کنید. با این دستور، یک پوشه جدید به نام *my\_magic\_lab* ساخته می‌شود. این آزمایشگاه شخصی توست.

## ۲. فعال‌سازی آزمایشگاه (ورود به حباب)

ساختن آزمایشگاه کافی نیست؛ باید وارد آن شویم. وقتی وارد شوی، نام آزمایشگاه را در کنار خط فرمان خود خواهی دید (مثلاً *(my\_magic\_lab)*). طلسم ورود (بسته به سیستم عامل):

• برای ویندوز: (Windows)

```
my_magic_lab\Scripts\activate
```

• برای مک و لینوکس: (Mac/Linux)

```
source my_magic_lab/bin/activate
```

اگر موفق شده باشی، ابتدای خط فرمان تو تغییر می‌کند و چیزی شبیه به این می‌بینی

```
(my_magic_lab) C:\Users\Adventurer\Project>
```

حالا تو در اینمی کامل هستی! هر چه نصب کنی، فقط همینجا می‌ماند.

نکته مهم

یادت باشه وقتی ترمینال رو بیندی و دوباره باز کنی نیاز هست دوباره این دستور فعال سازی رو بزنی.

### ۳. دروازه احضار (pip)

حالا که در آزمایشگاه هستیم، زمان احضار جادوهای دیگران است. در دنیای پایتون، میلیون‌ها جادوگر کدهای خود را در مخزنی به نام **PyPI** (Python Package Index) گذاشته‌اند. ابزار ما برای آوردن این بسته‌ها، **pip** نام دارد.

بیایید اولین کتابخانه خارجی را نصب کنیم :ما می‌خواهیم کتابخانه‌ای به نام **art** را نصب کنیم که برای چاپ متن‌های هنری و زیبا (ASCII Art) در ترمینال استفاده می‌شود. در همان ترمینال (که آزمایشگاهش فعال است) بنویس:

```
pip install art
```

ترمینال شروع به دانلود و نصب می‌کند. وقتی تمام شد، می‌گوید

Successfully installed art....  
تبریک! تو اولین جادوی خارجی خود را احضار کردی.

#### ۴. طومار نیازمندی‌ها (requirements.txt)

فرض کن می‌خواهی کد خود را به یک دوست بدهی. او از کجا بداند که باید کتابخانه art را نصب کند تا کد تو کار کند؟

ما لیست تمام مواد اولیه (کتابخانه‌ها) را در یک فایل متنه به نام requirements.txt ذخیره می‌کنیم.

طلسم ثبت مواد اولیه :

```
pip freeze > requirements.txt
```

این دستور، لیست تمام چیزهایی که در آزمایشگاهت نصب کرده‌ای را در فایل می‌نویسد.  
طلسم نصب مواد اولیه (برای دوست) : وقتی دوست کد تو را گرفت، فقط کافیست بنویسد:

```
pip install -r requirements.txt
```

و pip تمام آن‌ها را به صورت خودکار برای او نصب می‌کند!

نکته مهم

پیشنهاد می‌شود که از pip freeze استفاده نکنی و به جاش هر کتابخونه که پروژت نیاز دارد رو خودت بنویسی چرا چون اگه به این خروجی دقت کرده باشی تعداد خیلی زیادی از کتابخونه هارو که با خود پایتون نصب شدن هم داخلش هست و زمان انتقال روی کامپیوتر دوست احتمال تداخل این کتابخونه‌ها زیاده پس پیشنهاد می‌کنم فقط اسم کتابخونه‌هایی که پروژت نیاز داره رو در هر خط جداگانه داخل فایل requirements.txt بنویسی.

## ۵. پروژه: تابلوی اعلانات جادویی

مأموریت: حالا که کتابخانه art را نصب کردیم، بباید از آن استفاده کنیم تا یک تابلوی خوش‌آمدگویی حماسی برای شروع بخش «علم داده» بسازیم.

کد پروژه: (welcome\_sign.py):

```
1. # welcome_sign.py
2. # Using an external library summoned by pip!
3.
4. # 1. Import the external magic
5. # We installed this with: pip install art
6. from art import tprint
7.
8. # 2. Use the library's spell
9. print("-----")
10. print("Welcome to the Hall of Data Science:")
11. print("-----")
12.
13. # tprint converts text to ASCII art
14. tprint("WIZARD")
15.
16. print("-----")
17. print("Your magic lab is ready.")
18. print("-----")
```

اجرا کن و نتیجه را ببین!

```
1. -----
2. Welcome to the Hall of Data Science:
3. -----
4.
5. \ \ / / \ / | \ / | / \ / | / \ / | \ / | \ / | \ / |
6. \ \ / / \ / | \ / | / \ / | / \ / | \ / | \ / | \ / |
7. \ V V / | | / / | / \ | \ ( ) | \ [ ] | \ [ ] |
8. \ / \ / | | / | / \ | / \ | \ \ | \ \ | \ \ | \ \ |
9.
10. -----
11. Your magic lab is ready.
12. -----
```

آفرین! تو حالا نه تنها کد می‌نویسی، بلکه می‌دانی چطور محیط کار حرفه‌ای بسازی و از قدرت هزاران برنامه‌نویس دیگر در کدهای خودت استفاده کنی.

آزمایشگاه تو آماده است. در فصل بعد، از همین روش (pip install numpy) استفاده می‌کنیم تا قدرتمندترین ابزار ریاضیات جهان را احضار کنیم.

# فصل ۱۸: طومارهای NumPy (جادوی محاسبات نورآسا)

مأموریت فصل: لیست‌های پایتون برای نگهداری چند صد آیتم عالی هستند، اما وقتی با میلیون‌ها داده طرف هستیم، آن‌ها کُند و تبلیل می‌شوند. ما به طلسمنی نیاز داریم که محاسبات را با سرعت نور انجام دهد. به دنیای NumPy خوش آمدید.

نقشه راه:

- احضار NumPy: نصب و وارد کردن کتابخانه.
- چیست؟ ndarray (تفاوت ارتش منظم با گروه ماجراجویان).
- جادوی Vectorization: حذف حلقه‌های for.
- جادوی ابعاد: ساخت ماتریس‌ها و ضرب آن‌ها.
- جعبه ابزار جادویی: ۱۰ طلسم پرکاربرد NumPy با مثال.
- مسابقه بزرگ: مفهوم زمان‌سنجی و تفاوت سرعت.
- پروژه: چالش ۱,۰۰۰,۰۰۰ ماجراجو (برگزاری مسابقه سرعت).

## ۱. احضار NumPy

قبل از هر کاری، باید مطمئن شویم که در «آزمایشگاه جادویی» خود (که در فصل ۱۷ ساختیم) هستیم. ترمینال را باز کن و این کتابخانه را احضار کن:

```
pip install numpy
```

حالا در فایل پایتون خود، آن را وارد می‌کنیم. جادوگران معمولاً برای NumPy یک نام مستعار کوتاه (np) انتخاب می‌کنند تا نوشتن کد سریع‌تر شود.

```
1. import numpy as np
```

## ۲. ndarray چیست؟ (ارتاش منظم)

در فصل‌های قبل، ما از لیست‌ها (List) استفاده می‌کردیم. لیست‌ها مثل یک «کوله پشتی» هستند که می‌توانند هر چیزی را در خود نگه دارند: یک عدد، یک رشته، یک شیء Potion و... این انعطاف‌پذیری عالی است، اما هزینه دارد: سرعت پایین.

آرایه جادویی NumPy (یا ndarray): این آرایه مثل یک «ارتاش منظم» است.

قانون سخت‌گیرانه: تمام سربازان باید یک نوع باشند (مثلاً فقط عدد صحیح).

پاداش: چون همه یک‌شکل هستند، پایتون می‌تواند محاسبات را روی آن‌ها هزاران برابر سریع‌تر انجام دهد.

طلسم ساخت آرایه:

```
1. import numpy as np
2.
3. # 1. Python List (The Backpack)
4. my_list = [1, 2, 3, 4, 5]
```

```
5.  
6. # 2. NumPy Array (The Army)  
7. my_array = np.array([1, 2, 3, 4, 5])  
8.  
9. print(f"List: {my_list}")  
10. print(f"Array: {my_array}")  
11. print(type(my_array)) # <class 'numpy.ndarray'>
```

### ۳. جادوی Vectorization (حذف حلقه‌ها)

اینجا جایی است که جادو اتفاق می‌افتد. فرض کنید می‌خواهیم تمام اعداد لیست را در ۲ ضرب کنیم.

روش قدیمی (پایتون خالص): باید یک حلقه for بنویسیم و دانه دانه با اعداد حرف بزنیم:

```
1. # Old way  
2. numbers = [1, 2, 3]  
3. doubled = []  
4. for x in numbers:  
5.     doubled.append(x * 2)
```

روش NumPy (Vectorization): ما به کل ارتش یک فرمان واحد می‌دهیم و همه همزمان اطاعت می‌کنند! نیازی به حلقه نیست.

```
1. # NumPy way  
2. arr = np.array([1, 2, 3])  
3. doubled = arr * 2 # Magic! No loop needed.  
4.  
5. print(doubled) # Output: [2 4 6]
```

شما می‌توانید جمع، تفریق، ضرب، تقسیم و توان را روی کل آرایه با یک دستور انجام دهید.

## ۴. جادوی ابعاد: ماتریس‌ها

در علم داده و هوش مصنوعی، ما اغلب با جداول اعداد (ردیف و ستون) کار می‌کنیم. به این‌ها ماتریس (Matrix) می‌گویند. NumPy استاد ساخت ماتریس است.

ساخت ماتریس (آرایه ۲ بعدی):

```

1. # Creating a 2x3 Matrix (2 rows, 3 columns)
2. matrix_a = np.array([
3.     [1, 2, 3],
4.     [4, 5, 6]
5. ])
6.
7. print("--- Matrix A ---")
8. print(matrix_a)
9. print(f"Shape: {matrix_a.shape}") # Output: (2, 3)
```

ضرب ماتریسی (The Dot Product): این یکی از مهم‌ترین طلسم‌ها در هوش مصنوعی است. ضرب ماتریسی با ضرب معمولی فرق دارد. در NumPy، ما از طلسم dot یا عملگر @ استفاده می‌کنیم.

```

1. matrix_b = np.array([
2.     [10, 20],
3.     [30, 40],
4.     [50, 60]
5. ])
6.
7. # Multiplying Matrix A (2x3) by Matrix B (3x2)
8. # Result will be a 2x2 Matrix
9. result = np.dot(matrix_a, matrix_b)
10. # Or simply: result = matrix_a @ matrix_b
11.
12. print("--- Matrix Multiplication Result ---")
13. print(result)
14.
```

## ۵. جعبه ابزار جادویی: ۱۰ طلسمند پرکاربرد

یک دانشمند داده همیشه به ابزارهای سریع نیاز دارد. در اینجا ۱۰ مورد از پرکاربردترین وردهای NumPy را برایت لیست کرده‌ایم که در هر مأموریتی به کارت می‌آیند.

۱. np.arange (ساخت دنباله اعداد): مثل range در پایتون است، اما آرایه می‌سازد.

```
1. # Create numbers from 0 to 9
2. seq = np.arange(10)
3. print(seq) # [0 1 2 3 4 5 6 7 8 9]
```

۲. np.ones و np.zeros (ساخت بوم خالی): ساخت آرایه‌ای پر از صفر یا یک (برای شروع محاسبات).

```
1. # Create a 3x3 matrix of zeros
2. zeros = np.zeros((3, 3))
3. print(zeros)
4.
```

۳. np.random.randint (تاس جادویی): تولید اعداد تصادفی (بسیار مهم برای شبیه‌سازی).

```
1. # 5 random numbers between 1 and 100
2. lucky_numbers = np.random.randint(1, 100, 5)
3. print(lucky_numbers)
```

۴. np.reshape (تغییر آرایش ارتش): تغییر شکل آرایه (مثلاً تبدیل یک صف طولانی به یک مستطیل).

```
1. arr = np.arange(12) # 0 to 11 (12 items)
2. # Change to 3 rows, 4 columns
3. matrix = arr.reshape(3, 4)
```

۵. np.max, np.min, np.sum (گزارش میدان): گرفتن آمار سریع از کل آرایه.

```
1. scores = np.array([10, 50, 90, 20])
2. print(f"Max Score: {np.max(scores)}") # 90
3. print(f"Total Score: {np.sum(scores)}") # 170
4.
```

۶. np.mean (میانگین): مهم‌ترین طلسمن آماری.

```
1. print(f"Average Score: {np.mean(scores)}")
```

۷. np.sort (مرتب‌سازی): مرتب کردن سربازان از کم به زیاد.

```
1. sorted_scores = np.sort(scores)
2. print(sorted_scores) # [10 20 50 90]
```

۸. np.unique (حذف تکراری‌ها): پیدا کردن آیتم‌های یکتا.

```
1. items = np.array(["Sword", "Shield", "Sword", "Potion"])
2. print(np.unique(items)) # ['Potion' 'Shield' 'Sword']
```

۹. فیلتر کردن شرطی (Boolean Masking): پیدا کردن اعدادی که شرط خاصی دارند (جادوی خالص!).

```
1. # Find all scores greater than 40
2. high_scores = scores[scores > 40]
3. print(high_scores) # [50 90]
```

۱۰. np.concatenate (اتحاد ارتش‌ها): چسباندن دو آرایه به هم.

```
1. a = np.array([1, 2])
2. b = np.array([3, 4])
3. c = np.concatenate((a, b))
4. print(c) # [1 2 3 4]
5.
```

## ۶. مسابقه بزرگ: مفهوم زمان‌سنجی

ما ادعا کردیم که NumPy سریع است. اما یک دانشمند داده به «ادعا» اعتماد نمی‌کند؛ او به «داده» نیاز دارد. ما باید راهی برای اندازه‌گیری زمان اجرای کد پیدا کنیم.

برای این کار، مازول استاندارد time را احضار می‌کنیم. منطق ساده است:

زمان شروع را ثبت کن (start).

طلسم را اجرا کن.

زمان پایان را ثبت کن (end).

مدت زمان = پایان - شروع.

```

1. import time
2.
3. # Record the start time
4. start_time = time.time()
5.
6. # ... The spell runs here ...
7. # (Example: Count to 1,000,000)
8. x = 0
9. for i in range(1000000):
10.     x += 1
11.
12. # Record the end time
13. end_time = time.time()
14.
15. # Calculate duration
16. duration = end_time - start_time
17. print(f"The spell took {duration} seconds.")

```

حالا که ابزار زمان‌سنجی را داریم، آمده‌ی برگزاری بزرگترین مسابقه تاریخ انجمن هستیم.

## ۷. پروژه: چالش ۱,۰۰۰,۰۰۰ ماجراجو

مأموریت: انجمن ماجراجویان به تازگی ۱,۰۰۰,۰۰۰ عضو جدید جذب کرده است! ما لیستی از امتیازات آن‌ها (اعداد بین ۰ تا ۱۰۰) داریم. شورای عالی جادوگران می‌خواهد «میانگین» امتیازات کل اعضا را بداند.

ما دو داوطلب برای انجام این محاسبه داریم:

حلقه پایتون (The For Loop): روش سنتی و قدیمی.

نامپای (The NumPy Array): جادوی مدرن و برداری.

شما باید کدی بنویسید که این دو را در یک مسابقه عادلانه شرکت دهد و برنده را اعلام کند.

نقشه و طرح اولیه (قبل از کدنویسی):

تولید داده: ما نمی‌توانیم ۱ میلیون عدد را دستی تایپ کنیم! باید از `np.random.randint` (که در بخش ۵ یاد گرفتیم) برای تولید داده‌های تصادفی استفاده کنیم.

تبديل: داده‌ها را هم به صورت «لیست پایتون» و هم «آرایه نامپای» ذخیره کنید تا شرایط برابر باشد.

مسابقه اول: محاسبه میانگین با حلقه `for` و اندازه‌گیری زمان.

مسابقه دوم: محاسبه میانگین با `np.mean()` و اندازه‌گیری زمان.

داوری: تقسیم زمان پایتون بر زمان نامپای برای فهمیدن اینکه نامپای چند برابر سریع‌تر است.

چالش‌های فکری (تمرین تو):

چالش ۱: چطور با استفاده از `np.random.randint` یک میلیون عدد بین ۰ تا ۱۰۰ می‌سازی؟

چالش ۲: خروجی نامپای یک آرایه است. چطور آن را با `list()` به یک لیست معمولی پایتون تبدیل می‌کنی تا داوطلب اول (حلقه `for`) بتواند با آن کار کند؟

چالش ۳ (پایتون): چطور با یک حلقه `for` مجموع اعداد را حساب می‌کنی و بر تعداد تقسیم می‌کنی؟ یادت نرود قبل و بعدش از `time.time()` استفاده کنی.

چالش ۴ (نامپای): چطور با `np.mean()` میانگین را در یک خط حساب می‌کنی؟ زمان‌سنجی یادت نرود.

چالش ۵ (نتیجه): چطور محاسبه می‌کنی که نامپای چند برابر سریع‌تر بوده؟ (زمان پایتون تقسیم بر زمان نامپای).

وقت تمرین! سعی کن فایل `speed_test.py` را خودت بسازی.

#### کد پروژه نهایی (`speed_test.py`)

```
1. # speed_test.py
2. # The Great Race: Python Lists vs. NumPy Arrays
3.
4. import numpy as np
5. import time
6.
7. print("--- Generating Data for 1,000,000 Adventurers ---")
8.
9. # 1. Create the NumPy Array (The Army)
10. # Generates 1 million random numbers between 0 and 100
11. numpy_array = np.random.randint(0, 100, 1_000_000)
12.
13. # 2. Create the Python List (The Backpack)
14. # We convert the array to a list for a fair comparison
15. python_list = list(numpy_array)
16.
17. print("Data ready! Starting the race...\n")
18.
19. # --- RACE 1: The Old Way (Python Loop) ---
20. start_time = time.time()
```

```

21.
22. total_sum = 0
23. for score in python_list:
24.     total_sum += score
25. average_python = total_sum / len(python_list)
26.
27. end_time = time.time()
28. python_duration = end_time - start_time
29.
30. print(f"Python Loop Result: {average_python}")
31. print(f"Python Time: {python_duration:.5f} seconds\n")
32.
33.
34. # --- RACE 2: The Modern Way (NumPy) ---
35. start_time = time.time()
36.
37. # NumPy calculates mean in ONE ultra-fast step
38. average_numpy = np.mean(numpy_array)
39.
40. end_time = time.time()
41. numpy_duration = end_time - start_time
42.
43. print(f"NumPy Result: {average_numpy}")
44. print(f"NumPy Time: {numpy_duration:.5f} seconds\n")
45.
46.
47. # --- The Verdict ---
48. # Calculate how many times faster NumPy was
49. speedup = python_duration / numpy_duration
50.
51. print("-----")
52. print(f"CONCLUSION: NumPy was {speedup:.1f} times faster!")
53. print("-----")

```

اجرا کن و شگفتزده شو!

وقتی این کد را اجرا کنی، خواهی دید که پایتون معمولی ممکن است حدود ۱.۰ ثانیه طول بکشد، اما نامپای آنقدر سریع است (مثلاً ۰.۰۰۱ ثانیه) که انگار زمان متوقف شده است.

ممکن است بینی که نامپای ۵۰، ۱۰۰ یا حتی ۲۰۰ برابر سریع‌تر است!

این قدرت «علم داده» است. ابزارهای مناسب، کارهای غیرممکن را ممکن می‌کنند.

# فصل ۱۹: کتابخانه بزرگ

## پandas (رایم کردن داده‌های جدولی)

مأموریت فصل: آرایه‌های NumPy برای محاسبات عددی عالی بودند، اما اگر داده‌های ما شامل «نام»، «تاریخ» و «عدد» در کنار هم باشند چه؟ (مثل یک فایل اکسل). ما به ابزاری نیاز داریم که بتواند جداول عظیم داده را مدیریت کند. به کتابخانه بزرگ Pandas خوش آمدید.

نقشه راه:

- احضار Pandas: نصب و وارد کردن کتابخانه.
- چیست؟ DataFrame (جدول جادویی).
- خواندن طومارهای باستانی: بارگذاری فایل‌های CSV.
- ذره‌بین جادویی: بازرسی و آمارگیری سریع داده‌ها.
- جعبه ابزار پandas: ۱۰ طلسمن پرکاربرد با مثال.
- پروژه: تحلیل تابلوی امتیازات (خواندن، فیلتر کردن و ذخیره گزارش).

## ۱. احضار Pandas

مانند همیشه، ابتدا باید مطمئن شویم که در «آزمایشگاه جادویی» (venv) هستیم. سپس این کتابخانه عظیم را احضار می‌کنیم:

```
pip install pandas
```

در کد پایتون، جادوگران همیشه برای Pandas از نام مستعار pd استفاده می‌کنند.

```
1. import pandas as pd
```

## ۲. DataFrame چیست؟ (جدول جادویی)

در NumPy ما با «اعداد» سروکار داشتیم. در Pandas، ما با جداول‌ها کار می‌کنیم. Series (سری): یک ستون از داده‌ها (مثل یک لیست). DataFrame (دیتاframes): یک جدول کامل شامل سطرها و ستون‌ها. هر ستون نام دارد و هر سطر یک شماره (Index) دارد.

طلسم ساخت یک جدول دستی: فرض کنید می‌خواهیم لیست موجودی مغازه جادوگری را به یک جدول تبدیل کنیم.

```
1. import pandas as pd
2.
3. # 1. Prepare data (Dictionary of Lists)
4. data = {
5.     "Item": ["Health Potion", "Mana Potion", "Iron Sword", "Wooden
Shield"],
6.     "Price": [50, 70, 150, 30],
7.     "Stock": [10, 5, 2, 15]
8. }
9.
10. # 2. Create the DataFrame
11. df = pd.DataFrame(data)
12.
13. print("--- My Magic Shop Inventory ---")
14. print(df)
```

خروجی:

	Item	Price	Stock
2.	0 Health Potion	50	10
3.	1 Mana Potion	70	5
4.	2 Iron Sword	150	2
5.	3 Wooden Shield	30	15

می‌بینید؟ پایتون حالا داده‌ها را به صورت یک جدول مرتب و زیبا می‌فهمد!

### ۳. خواندن طومارهای باستانی (فایل‌های CSV)

در دنیای واقعی، ما داده‌ها را دستی تایپ نمی‌کنیم. ما آن‌ها را از فایل‌های Comma (CSV) (Comma Separated Values) می‌خوانیم. این فایل‌ها شبیه فایل اکسل هستند اما ساده‌تر.

طلسم `read_csv`: این طلسمند فایلی با میلیون‌ها سطر را در یک چشم به هم زدن بخواند و به DataFrame تبدیل کند.

```
1. # Reading a scroll named 'adventurers.csv'
2. # df = pd.read_csv("adventurers.csv")
```

(نکته: در بخش پروژه، ما ابتدا این فایل را می‌سازیم و سپس می‌خوانیم).

### ۴. ذره‌بین جادویی: فیلتر کردن داده‌ها

قدرت واقعی Pandas در «پرس و جو» (Query) است. مثلاً: «فقط آیتم‌هایی را به من نشان بده که قیمت‌شان زیر ۶۰ سکه است».

در پایتون معمولی، این کار نیاز به حلقه `for` و `if` داشت. در Pandas، این کار با یک خط انجام می‌شود:

```
1. # Filter: Find items cheaper than 60 gold
2. cheap_items = df[df["Price"] < 60]
3.
4. print("\n--- Bargain Bin (< 60 Gold) ---")
5. print(cheap_items)
```

خروجی:

	Item	Price	Stock
2.	0 Health Potion	50	10
3.	3 Wooden Shield	30	15

## ۵. جعبه ابزار پانداس: ۱۰ طلسیم پر کاربرد

یک دانشمند داده بدون این ۱۰ طلسیم نمی‌تواند زندگی کند. فرض کنید متغیر df همان جدول مغازه ماست.

۱. df.head(n) (نگاه دزدکی): ۵ سطر اول جدول را نشان می‌دهد. عالی برای وقتی که فایل خیلی بزرگ است.

```
1. print(df.head(2)) # Show first 2 rows
```

۲. df.info() (شناسنامه): اطلاعات فنی جدول (جنس ستون‌ها، تعداد داده‌های خالی و...) را می‌گوید.

```
1. df.info()
```

۳. df.describe() (آمار سریع): مهم‌ترین طلسیم آماری! میانگین، ماکریتم، مینیمم و انحراف معیار ستون‌های عددی را یکجا می‌دهد.

```
1. print(df.describe())
```

۴. df.columns (لیست ستون‌ها): اسمی تمام ستون‌های جدول را برمی‌گرداند.

```
1. print(df.columns) # Index(['Item', 'Price', 'Stock'], dtype='object')
```

۵. df.shape (ابعاد): می‌گوید جدول چند سطر و چند ستون دارد. (مثل NumPy).

```
1. print(df.shape) # (4, 3) -> 4 rows, 3 columns
```

۶. df.sort\_values(): جدول را بر اساس یک ستون خاص مرتب می‌کند.

```
1. # Sort by Price (High to Low)
2. sorted_df = df.sort_values(by="Price", ascending=False)
```

۷. df["Column"].mean(): میانگین ستونی را حساب می‌کند.

```
1. avg_price = df["Price"].mean()
```

۸. df.to\_csv(): جدول تغییر یافته را در یک فایل جدید ذخیره می‌کند.

```
1. df.to_csv("new_inventory.csv", index=False)
```

۹. df.iloc[آدرس]: دسترسی به یک سلوی خاص با شماره سطر و ستون.

```
1. # Row 0, Column 0
2. item_name = df.iloc[0, 0] # "Health Potion"
```

۱۰. df["NewCol"] = ... (خلق ستون جدید): ساخت یک ستون جدید بر اساس محاسبات ستون‌های قبلی.

```
1. # Calculate total value (Price * Stock)
2. df["Total Value"] = df["Price"] * df["Stock"]
3.
```

## ۶. پروژه: تحلیل تابلوی امتیازات

مأموریت: ما یک فایل CSV داریم که حاوی اطلاعات ۱۰۰ ماجراجو (نام، کلاس، امتیاز) است. ما باید:

فایل را بخوانیم.

میانگین امتیازات کل را پیدا کنیم.

فقط ماجراجویان «سطح بالا» (امتیاز بالای ۸۰) را پیدا کنیم.

لیست نخبگان را در یک فایل جدید ذخیره کنیم.

گام اول: ساخت داده‌های تمرینی (چون شما فایل adventurers.csv را ندارید، ابتدا یک کد کوچک می‌نویسیم تا آن را برایتان بسازد).

یک فایل به نام create\_data.py بسازید و اجرا کنید:

```

1. # create_data.py
2. import pandas as pd
3. import numpy as np
4.
5. # Create dummy data using NumPy magic
6. names = [f"Adventurer_{i}" for i in range(1, 101)]
7. classes = np.random.choice(["Mage", "Warrior", "Rogue", "Healer"], 100)
8. scores = np.random.randint(50, 100, 100) # Scores between 50 and 100
9.
10. # Create DataFrame
11. df = pd.DataFrame({
12.     "Name": names,
13.     "Class": classes,
14.     "Score": scores
15. })
16.
17. # Save to CSV
18. df.to_csv("adventurers.csv", index=False)
19. print("File 'adventurers.csv' created successfully!")
20.

```

گام دوم: تحلیل داده (پروژه اصلی) حالا فایل اصلی پروژه analyze\_scores.py را می‌نویسیم:

```

1. # analyze_scores.py
2. # The Pandas Analysis Tool
3.
4. import pandas as pd
5.
6. print("--- Loading the Ancient Scroll (CSV) ---")
7. # 1. Read the file
8. df = pd.read_csv("adventurers.csv")
9.

```

```

10. # Show the first few rows to check
11. print("First 5 adventurers:")
12. print(df.head())
13. print("-----")
14.
15. # 2. Statistical Magic
16. # Calculate the average score
17. average_score = df["Score"].mean()
18. print(f"Average Guild Score: {average_score:.2f}")
19.
20. # Count adventurers by Class
21. print("\nClass Distribution:")
22. print(df["Class"].value_counts())
23.
24. # 3. Filtering the Elites
25. # We want adventurers with Score > 85
26. print("\n--- Searching for Elites (Score > 85) ---")
27. elites = df[df["Score"] > 85]
28.
29. # Sort them by Score (Highest first)
30. elites_sorted = elites.sort_values(by="Score", ascending=False)
31.
32. print(f"Found {len(elites_sorted)} elite adventurers.")
33. print(elites_sorted.head())
34.
35. # 4. Saving the Result
36. print("\n--- Saving the Elite List ---")
37. elites_sorted.to_csv("elites.csv", index=False)
38. print("Saved to 'elites.csv'. Mission Complete!")
39.

```

اجرا کن و لذت ببر! حالا فایل analyze\_scores.py را اجرا کن. تو خواهی دید که پایتون چگونه فایل را می‌خواند، آمار را محاسبه می‌کند و یک فایل اکسل جدید (elites.csv) برایت می‌سازد که فقط شامل قهرمانان است.

این قدرت علم داده است. تو اکنون می‌توانی داده‌ها را رام کنی! در فصل بعد، یاد می‌گیریم چطور این اعداد را به «تصویر» (نمودار) تبدیل کنیم.



# فصل ۲۰: نقشه‌ی مسحور

## (روایت داستان با Matplotlib)

مأموریت فصل: یک پیشگو (Data Scientist) می‌داند که پادشاهان و فرماندهان وقت ندارند هزاران سطر عدد را در جدول‌ها بخوانند. آن‌ها «تصویر» می‌خواهند. در این فصل، یاد می‌گیریم چطور داده‌ها را به نقشه‌های بصری زیبا تبدیل کنیم تا داستان پنهان پشت اعداد آشکار شود. به دنیای Matplotlib خوش آمدید.

نقشه راه:

- احضار نقاش سلطنتی (Matplotlib): نصب و راهاندازی.
- اولین قلم مو (plot): رسم نمودار خطی برای دیدن «رونده».
- ستون‌های مقایسه (bar): رسم نمودار میله‌ای.
- زیبایی‌شناسی جادویی: اضافه کردن عنوان، برچسب و رنگ.
- جعبه ابزار نقاش: ۱۰ طلسما پرکاربرد با مثال.
- پروژه: گزارش تصویری انجمن (ترکیب Pandas و Matplotlib).

## ۱. احضار نقاش سلطنتی (matplotlib)

ما برای رسم نمودار به یک کتابخانه قدرتمند نیاز داریم. مشهورترین آن‌ها matplotlib است. ابتدا در آزمایشگاه جادویی خود (venv) آن را نصب می‌کنیم:

```
pip install matplotlib
```

در کد پایتون، ما زیرمجموعه pyplot را احضار می‌کنیم و طبق سنت قدیمی جادوگران، به آن نام مستعار plt می‌دهیم.

```
1. import matplotlib.pyplot as plt
```

## ۲. اولین قلمرو (plot) – نمودار خطی

نمودار خطی (Line Plot) برای نشان دادن تغییرات در طول زمان (Time Series) عالی است. مثلاً: «قدرت جادویی شاگرد ما در ۱۰ روز گذشته چقدر رشد کرده است؟»

```
1. import matplotlib.pyplot as plt
2.
3. # 1. Prepare the data
4. days = [1, 2, 3, 4, 5]
5. power_levels = [10, 15, 18, 25, 40]
6.
7. # 2. Cast the plot spell
8. # plt.plot(x_axis, y_axis)
9. plt.plot(days, power_levels)
10.
11. # 3. Reveal the canvas
12. print("Opening the magical canvas...")
13. plt.show()
```

وقتی این کد را اجرا کنید، پنجره‌ای باز می‌شود و خط رشد قدرت را به شما نشان می‌دهد.

جادو مرئی شد!

### ۳. ستون‌های مقایسه (bar) - نمودار میله‌ای

اگر بخواهیم چیزهای مختلف را با هم مقایسه کنیم (مثلاً قدرت شمشیر در برابر قدرت تبر)، از نمودار میله‌ای (Bar Chart) استفاده می‌کنیم.

```
1. # Data
2. weapons = ["Sword", "Axe", "Bow", "Staff"]
3. damage = [50, 65, 40, 80]
4.
5. # Cast the bar spell
6. plt.bar(weapons, damage)
7.
8. plt.show()
9.
```

### ۴. زیبایی‌شناسی جادویی (شخصی‌سازی)

یک نقشه گنج بدون راهنمای اسم، بی‌ارزش است. ما باید به نمودارمان «عنوان»، «برچسب محورها» و «رنگ» اضافه کنیم.

```
1. days = [1, 2, 3, 4, 5]
2. gold = [100, 80, 50, 120, 200]
3.
4. # Customizing the magic
5. plt.plot(days, gold, color='green', marker='o', linestyle='--')
6.
7. # Adding labels (The Legend)
8. plt.title("My Gold Over Time")           # Title of the spell
9. plt.xlabel("Day of Adventure")            # X-axis label
10. plt.ylabel("Gold Coins")                 # Y-axis label
11. plt.grid(True)                         # Add a grid for better reading
12.
13. plt.show()
```

### ۵. جعبه ابزار نقاش: ۱۰ طلسم پرکاربرد

یک مصورساز داده (Data Visualizer) باید این ۱۰ ابزار را همیشه در کمربند خود داشته باشد.

۱. plt.plot(x, y) (خط زمان): رسم نقاط به هم پيوسته. عالي برای روندها.

```
1. plt.plot([1, 2, 3], [2, 4, 6])
```

۲. plt.bar(x, height) (ستونها): رسم ميله‌های عمودی. عالي برای مقایسه دسته‌ها.

```
1. plt.bar(['A', 'B'], [10, 20])
```

۳. plt.scatter(x, y) (پراكندگی): فقط نقاط را رسم می‌کند (بدون خط). عالي برای دیدن رابطه بین دو چيز (مثلًاً رابطه «سن» و «قدرت»).

```
1. plt.scatter([20, 30, 40], [50, 45, 30])
```

۴. plt.hist(x) (توزيع): هيستوگرام. نشان می‌دهد که داده‌ها چقدر تکرار شده‌اند (مثلًاً چند نفر نمره ۲۰ گرفتند، چند نفر ۱۵ و...).

```
1. scores = [10, 10, 20, 20, 20, 30]
2. plt.hist(scores, bins=3)
```

۵. plt.pie(x) (کیک جادویی): نمودار دایره‌ای. برای نشان دادن سهم هر بخش از کل (مثلًاً چند درصد جادوگرند، چند درصد جنگجو).

```
1. plt.pie([40, 60], labels=['Mages', 'Warriors'])
```

۶. plt.title("Name") (نام‌گذاري): انتخاب نام برای کل نمودار.

```
1. plt.title("Name")
```

۷. plt.xlabel() و plt.ylabel() (راهنماي محورها): نوشتن توضیح برای محور افقی و عمودی.

```
1. plt.xlabel('Titles label')
2. plt.ylabel('Numbers label')
```

۸. plt.figure(figsize=(10, 5)) (اندازه بوم): تغيير اندازه تصوير قبل از رسم. (۱۰ اينچ عرض، ۵ اينچ ارتفاع).

```
1. plt.figure(figsize=(8, 6))
```

۹. plt.savefig("filename.png"): به جای نشان دادن روی صفحه، نمودار را به صورت یک عکس ذخیره می‌کند.

```
1. plt.savefig("my_chart.png")
```

۱۰. plt.legend(): اگر چند خط در یک نمودار دارید، نشان می‌دهد کدام رنگ مربوط به کدام داده است.

```
1. plt.plot(x, y1, label="Gold")
2. plt.plot(x, y2, label="Silver")
3. plt.legend() # Shows the legend box
```

## ۶. پروژه: گزارش تصویری انجمان

مأموریت: ما در فصل قبل (۱۹) فایل adventurers.csv را تحلیل کردیم و نخبگان را پیدا کردیم. حالا رئیس انجمان یک گزارش تصویری می‌خواهد. او می‌خواهد بداند:

توزیع کلاس‌ها: چند درصد از اعضا Warrior و Mage و... هستند؟ (نمودار دایره‌ای).

مقایسه نخبگان: ۵ ماجراجوی برتر چه کسانی هستند و امتیازشان چقدر است؟ (نمودار میله‌ای).

نکته: ما از pandas برای خواندن داده و از matplotlib برای رسم آن استفاده می‌کنیم.

### : (visualize\_guild.py)

```
1. # visualize_guild.py
2. # The Visual Report Generator
3.
4. import pandas as pd
5. import matplotlib.pyplot as plt
6.
```

```

7. print("--- Summoning Data & Painter ---")
8.
9. # 1. Read the data (Chapter 19 Magic)
10. # We assume 'adventurers.csv' exists from the previous chapter
11. df = pd.read_csv("adventurers.csv")
12.
13. # --- MISSION 1: The Class Distribution (Pie Chart) ---
14. # Count how many adventurers are in each class
15. class_counts = df["Class"].value_counts()
16.
17. # Create a new canvas
18. plt.figure(figsize=(8, 8))
19.
20. # Draw the Pie
21. plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%',
startangle=140)
22. plt.title("Guild Class Distribution")
23.
24. # Save the spell result
25. plt.savefig("guild_classes.png")
26. print("Saved 'guild_classes.png'")
27. plt.show() # Show it to us
28.
29.
30. # --- MISSION 2: The Top 5 Elites (Bar Chart) ---
31. # Sort by score and take top 5
32. top_5 = df.sort_values(by="Score", ascending=False).head(5)
33.
34. # Create a new canvas
35. plt.figure(figsize=(10, 6))
36.
37. # Draw the Bars
38. # X axis = Names, Y axis = Scores
39. colors = ['gold', 'silver', 'brown', 'blue', 'green']
40. plt.bar(top_5["Name"], top_5["Score"], color=colors)
41.
42. # Decorate the map
43. plt.title("Top 5 Guild Members")
44. plt.xlabel("Adventurer Name")
45. plt.ylabel("Score (0-100)")
46. plt.grid(axis='y', linestyle='--', alpha=0.7)
47.
48. # Save the spell result
49. plt.savefig("top_5_elites.png")
50. print("Saved 'top_5_elites.png'")
51. plt.show()
52. print("Visual Report Complete!")

```

اجرا کن و تماشا کن! کد را اجرا کن. دو پنجره جادویی باز می‌شوند (یکی بعد از بستن دیگری) و دو تصویر png در پوشه پروژهات ذخیره می‌شوند.

یک نمودار دایره‌ای رنگارنگ که نشان می‌دهد ارتش ما از چه کسانی تشکیل شده.

یک نمودار میله‌ای که ۵ قهرمان برتر را با ستون‌های رنگی (طلایی، نقره‌ای و...) نشان می‌دهد.

تبریک می‌گوییم! تو اکنون قدرت کامل «علم داده» را در اختیار داری: جمع‌آوری (NumPy)، تحلیل (Matplotlib) و نمایش (Pandas).

تو آماده‌ای تا در فصل بعد (فصل ۲۱)، مسیر نهایی خود را در دنیای بی‌کران برنامه‌نویسی انتخاب کنی.

## بخش هفتم:



# ماجراجویی ادامه دارد...

# فصل ۲۱: افق‌های بی‌پایان (نقشه‌های گنج آینده)

مأموریت فصل: ماجراجویی با پایتون در اینجا تمام نمی‌شود؛ بلکه تازه شروع شده است! حالا که کیف ابزارت پر از طلسم‌های قدرتمند است، باید تصمیم بگیری که چه نوع جادوگری می‌خواهی باشی؟ در این فصل، بر اساس نقشه‌های گنج باستانی، مسیرهای تخصصی دنیای تکنولوژی را بررسی می‌کنیم.

نقشه راه:

- قدم‌های بعدی (توصیه‌های ضروری برای همه).
- مسیر معماران دیجیتال (Web & App).
- مسیر پیشگویان داده (AI & Data Science).
- مسیر محافظان و مدیران (Security & SysAdmin).
- مسیر خالقان دنیاهای (Game & AR/VR).
- مسیر صنعتگران جادویی (IoT & Robotics).

## ۱. قدم‌های بعدی (The Essential Next Steps)

«قبل از انتخاب تخصص، باید شمشیر خود را تیزتر کنید.»

فرقی نمی‌کند کدام مسیر تخصصی را انتخاب کنید؛ بر اساس تجربیات بزرگان سه مهارت وجود دارد که همین الان باید یاد بگیرید:

پایتون پیشرفته (Advanced Python):

- یادگیری عمیق‌تر OOP (شیء‌گرایی که در فصل ۱۲ شروع کردیم).
- یادگیری Async (انجام چند طلسمن همزمان).
- مدیریت خطای Testing (تست کردن کدها برای جلوگیری از انفجار).
- جادوی زمان و همکاری (Git & GitHub):
- یادگیری Git برای کنترل نسخه (مثل Save Point در بازی‌ها).
- آشنایی با GitHub برای اشتراک‌گذاری کد با دیگران.

استقرار (Deployment):

کد شما نباید فقط روی لپ‌تاپان بماند. یاد بگیرید چطور آن را آنلاین کنید (با استفاده از Docker یا سرورهای Linux).

## ۲. مسیر معماران دیجیتال (Programming & Web)

«برای کسانی که می‌خواهند زیرساخت‌های دنیای دیجیتال را بسازند.»  
اگر دوست دارید وب‌سایتها، اپلیکیشن‌ها و نرم‌افزارهایی بسازید که میلیون‌ها نفر از آن استفاده کنند.

توسعه وب (Web Development):

- يادگیری فریم‌ورک‌های قدرتمند پایتونی مثل FastAPI یا جنگو.
- آشنایی با ظاهر ماجرا (Frontend...HTML, CSS, JavaScript...React).

**پروژه پیشنهادی:** ساخت یک پروژه وبلاگ کامل (با قابلیت ثبت‌نام کاربر، پست گذاشتن و دیتابیس).

#### برنامه‌نویسی موبایل و دسکتاپ:

ساخت اپلیکیشن برای گوشی‌ها یا کامپیوترها (برای موبایل فلاتر و زبان دارت رو پیشنهاد میکنم و برای ویندوز زبان #c).

#### ساختمان داده و الگوریتم:

یادگیری اصول مهندسی برای نوشتن کدهای بهینه.

### ۳. مسیر پیشگویان داده (AI & Data)

«برای کسانی که می‌خواهند آینده را از روی داده‌ها بخوانند و به ماشین‌ها یادگیری بیاموزند.» این مسیر برای کسانی است که عاشق ریاضیات، آمار و هوش مصنوعی هستند.

#### هوش مصنوعی (AI & Machine Learning)

- یادگیری مبانی یادگیری ماشین (Machine Learning) و یادگیری عمیق (Deep Learning).
- بینایی ماشین (Computer Vision): یاد دادن دیدن به کامپیوتر.
- NLP: پردازش زبان طبیعی (مثل کاری که ChatGPT می‌کند).

#### ۴. مسیر محافظان و مدیران سیستم (SysAdmin & Security)

«برای کسانی که می‌خواهند نظم را برقرار کنند و از قلعه‌ها محافظت کنند.»  
اگر به شبکه، سرورها و امنیت علاقه دارید، این مسیر هیجان‌انگیز شماست.

##### :Sys Admin (مدیریت سیستم)

- تسلط کامل بر Linux Administration
- تسلط کامل بر شبکه

##### :DevOps

خودکارسازی فرآیندهای نرم‌افزاری.  
کسی که از سرور‌ها نگهداری می‌کنند و مسئول این هستیش کد‌های شما رو روی سرور قرار بدله.

- خدمات ابری (Cloud Services) و شبکه (Networking).
- امنیت سایبری (Cybersecurity).
- هک اخلاقی (Ethical Hacking) و تست نفوذ (Penetration Testing).
- امنیت شبکه و جرم‌شناسی دیجیتال.

##### :Blockchain ( بلاکچین )

- برای بلاکچین زبان سالیدیتی رو پیشنهاد می‌کنم.
- نوشتمن قراردادهای هوشمند (Smart Contracts).
  - توسعه برنامه‌های غیرمت مرکز (DApps).

#### ۵. مسیر خالقان دنیاهای (Game)

برای کسانی که رویاهایشان را به تصویر می‌کشند.

### بازی‌سازی (Game Dev)

- کار با موتورهای بازی‌سازی (Game Engines) مثل Unity یا Godot یا Unreal

### واقعیت افزوده و مجازی (AR/VR)

دنیای جذاب واقعیت مجازی و افزوده.

### ۶. مسیر صنعتگران جادویی (IoT & Robotics)

برای اینکار زبان C و آردنو رو پیشنهاد میکنم.

- (برای کسانی که می‌خواهند به اشیاء بی‌جان، جان بیخشند).»

- اتصال کدهای پایتون به دنیای فیزیکی و سخت‌افزارها.

### ۷. مسیر طراحان دنیاهای (Design & Graphic)

برای بچه‌هایی که به کار طراحی رابط کاربر و ظاهر اپلیکیشن و بازی‌ها علاوه مندن

#### گرافیک و طراحی:

- طراحی رابط کاربری (UI/UX).

- مدل‌سازی سه‌بعدی و انیمیشن (3D Modeling).

- طراحی لوگو

# سخن پایانی: پایان آغاز

ماجراجوی عزیز، کتاب «ماجراجویی با پایتون» در اینجا به پایان می‌رسد، اما داستان تو تازه شروع شده است. کدنویسی مثل جادوگری است؛ هرچه بیشتر تمرین کنی، طلسم‌هایت قادر تمندتر می‌شوند.

به یاد داشته باش:

- از اشتباه کردن نترس (ارورها بهترین معلمان تو هستند).
- همیشه کنجکاو باش و یادگیری را متوقف نکن.
- و مهم‌تر از همه... به کد زدن ادامه بده!

# منابع و مأخذ

سفر ما در اینجا به پایان می‌رسد، اما اقیانوس دانش بی‌انتهای است. برای نگارش این کتاب و همچنین برای ادامه مسیر یادگیری شما، از منابع ارزشمند زیر استفاده شده است.

۱. طومارهای حقیقت (مستندات رسمی) این‌ها معتبرترین منابعی هستند که توسط خود خالقان این ابزارها نوشته شده‌اند:

Python Official Documentation ([docs.python.org](https://docs.python.org)) مرجع اصلی زبان پایتون.  
NumPy Documentation ([numpy.org/doc](https://numpy.org/doc)) برای محاسبات عددی.  
Pandas Documentation ([pandas.pydata.org/docs](https://pandas.pydata.org/docs)) برای تحلیل داده‌ها.  
Matplotlib Documentation ([matplotlib.org/stable](https://matplotlib.org/stable)) برای رسم نمودارها.

۲. کتاب‌های جادویی (منابع آموزشی پیشنهادی) اگر دوست دارید با خواندن کتاب یاد بگیرید، این‌ها بهترین همراهان شما خواهند بود:

Eric Matthes اثر Python Crash Course (مناسب برای شروع سریع).  
Al Sweigart اثر Automate the Boring Stuff with Python (عالی برای کاربردهای عملی و خودکارسازی).

اثر Luciano Ramalho **Fluent Python** (برای کسانی که می‌خواهند به سطح استادی برستند).

۳. اوراکل‌های آنلاین (جوامع و وبسایت‌ها) زمانی که در طلسماً گیر کردید، این‌ها مکان‌هایی هستند که پاسخ را در آن‌ها خواهید یافت:

Stack Overflow بزرگترین انجمن پرسش و پاسخ برنامه‌نویسان.  
Real Python (realpython.com) مقالات و آموزش‌های عالی و عمیق.  
W3Schools & GeeksforGeeks برای دسترسی سریع به مثال‌ها و مفاهیم پایه.

۴. تقدیر و تشکر با تشکر از جامعه متن‌باز (Open Source) پایتون که این ابزارهای قدرتمند را به رایگان در اختیار تمام ماجراجویان جهان قرار داده اند.