📌 مثال‌های کاربردی جامع از کار با فایل‌ها در پایتون

در این بخش، چندین **مثال کاربردی و جامع** از خواندن، پردازش و نوشتن داده‌ها در فایل‌ها ارائه می‌شود. همچنین، به **کار با فایل‌های متنی و باینری** می‌پردازیم.

✅ ۱. خواندن یک فایل متنی، پردازش داده‌ها و ذخیره نتایج در فایل دیگر

🎯 سناریو:

یک فایل متنی داریم که شامل **لیستی از نمرات دانش‌آموزان** است. می‌خواهیم:

1. این فایل را **خط به خط بخوانیم**.
2. **میانگین نمرات** را محاسبه کنیم.
3. نتایج را **در یک فایل خروجی ذخیره کنیم**.

🔹 📂 **فایل ورودی (`grades.txt`):**

```
Ali, 18, 20, 19
Sara, 15, 14, 16
Reza, 20, 20, 19
```

🔹 📂 **کد پردازش و ذخیره در فایل خروجی (`results.txt`)**

```
with open("grades.txt", "r") as infile, open("results.txt", "w") as outfile:
    for line in infile:
        parts = line.strip().split(", ")  # جداسازی نام و نمرات
        name = parts[0]
        scores = list(map(int, parts[1:]))  # تبدیل رشته‌ها به اعداد
        avg_score = sum(scores) / len(scores)  # محاسبه میانگین

        outfile.write(f"{name}: میانگین نمرات = {avg_score:.2f}\n")

print("✅ پردازش انجام شد و نتایج در فایل `results.txt` ذخیره شد.")
```

🔹 📂 **فایل خروجی (`results.txt`):**

```
Ali: 19.00 = میانگین نمرات
Sara: 15.00 = میانگین نمرات
Reza: 19.67 = میانگین نمرات
```

✅ **نکات مهم:**

- **خواندن خط به خط** باعث **کاهش مصرف حافظه** در فایل‌های حجیم می‌شود.
- از `strip()` برای حذف فاصله‌ها و `split(", ")` برای جداسازی داده‌ها استفاده شده است.

✅ ۲. جستجوی یک کلمه در یک فایل متنی و نمایش تعداد تکرار آن

🎯 سناریو:

می‌خواهیم تعداد تکرار **یک کلمه مشخص** را در یک فایل متنی پیدا کنیم.

🔹 📂 **فایل ورودی (`document.txt`):**

```
Python is a great programming language.
Many developers use Python for web development, data science, and AI.
Python is easy to learn and powerful.
```

🔹 📂 **کد جستجوی کلمه:**

```
word_to_find = "Python"
count = 0

with open("document.txt", "r") as file:
    for line in file:
        count += line.lower().count(word_to_find.lower())  # حساس نبودن به بزرگی و کوچکی حروف

print(f"✅ کلمه '{word_to_find}' {count} بار در فایل یافت شد.")
```

✅ **خروجی:**

```
✅ کلمه 'Python' 3 بار در فایل یافت شد.
```

✅ **نکات:**

- با استفاده از `lower()` حساسیت به حروف بزرگ و کوچک را حذف کرده‌ایم.
- می‌توان از `re.findall()` برای جستجوی دقیق‌تر استفاده کرد.

✅ ۳. کار با فایل‌های باینری (تصویر)

🎯 سناریو:

می‌خواهیم یک **فایل تصویری (باینری) را بخوانیم و کپی کنیم**.

🔹 📂 **کد کپی کردن تصویر (`image_copy.jpg`)**

```
with open("image.jpg", "rb") as infile, open("image_copy.jpg", "wb") as outfile:
    outfile.write(infile.read())

print("✅ تصویر با موفقیت کپی شد!")
```

✅ **نکات:**

- **(read binary)** `rb` برای خواندن فایل به صورت باینری.
- **(write binary)** `wb` برای نوشتن فایل باینری.

- این روش برای **کپی کردن تصاویر، ویدئوها، فایل‌های PDF و ...** مفید است.

✅ ۴. خواندن فایل CSV و ذخیره داده‌های پردازش‌شده در یک فایل جدید

🎯 سناریو:

یک فایل CSV حاوی **اطلاعات فروش** داریم. می‌خواهیم:

1. آن را بخوانیم.
2. **جمع فروش هر محصول را محاسبه کنیم.**
3. نتایج را در یک فایل جدید ذخیره کنیم.

🔹 📂 **فایل ورودی (`sales.csv`):**

```
Product, Price, Quantity
Laptop, 1500, 2
Phone, 800, 3
Tablet, 500, 5
```

🔹 📂 **کد پردازش CSV:**

```python
import csv

with open("sales.csv", "r") as infile, open("sales_summary.csv", "w", newline="") as outfile:
    reader = csv.reader(infile)
    writer = csv.writer(outfile)

    header = next(reader)  # خواندن هدر
    writer.writerow(["Product", "Total Sales"])  # نوشتن هدر جدید

    for row in reader:
        product = row[0]
        total_sales = int(row[1]) * int(row[2])  # قیمت * تعداد
        writer.writerow([product, total_sales])

print("✅ پردازش انجام شد و نتایج در `sales_summary.csv` ذخیره شد.")
```

🔹 📂 **فایل خروجی (`sales_summary.csv`):**

```
Product, Total Sales
Laptop, 3000
Phone, 2400
Tablet, 2500
```

✅ **نکات:**

- از **ماژول** `csv` برای کار با فایل‌های CSV استفاده کردیم.
- `next(reader)` برای خواندن و رد کردن هدر استفاده شد.

✅ ۵. ایجاد فایل `log` برای ذخیره رویدادهای برنامه

🎯 سناریو:

می‌خواهیم یک **فایل گزارش (log file)** برای ذخیره خطاها و رویدادهای برنامه ایجاد کنیم.

🔹 📂 **کد ذخیره پیام‌های لاگ در فایل (`app.log`)**

```
import datetime

def log_message(message, level="INFO"):
    with open("app.log", "a") as file:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        file.write(f"[{timestamp}] {level}: {message}\n")

log_message("برنامه شروع شد.")
log_message("اتصال به دیتابیس موفق بود.", "SUCCESS")
log_message("خطا در خواندن فایل تنظیمات!", "ERROR")

print("✅ پیام‌های لاگ در `app.log` ذخیره شدند.")
```

🔹 📂 **فایل `app.log`:**

```
[2025-02-16 12:00:00] INFO: برنامه شروع شد.
[2025-02-16 12:00:05] SUCCESS: اتصال به دیتابیس موفق بود.
[2025-02-16 12:00:10] ERROR: خطا در خواندن فایل تنظیمات!
```

✅ **نکات:**

- **تاریخ و زمان** هر لاگ ثبت شده است.
- سطح لاگ (`INFO`, `ERROR`, `SUCCESS`) قابل تنظیم است.

🎯 جمع‌بندی نهایی

✅ **خواندن و پردازش فایل‌های متنی** (مثل محاسبه میانگین نمرات).
✅ **جستجو و شمارش کلمات در یک فایل**.
✅ **کار با فایل‌های باینری** (کپی کردن تصویر).
✅ **کار با CSV و پردازش اطلاعات**.
✅ **ایجاد فایل لاگ برای ذخیره پیام‌ها و خطاهای برنامه**.

📌 💡 **تمرین:**
یک برنامه بنویسید که اطلاعات دانشجویان را از یک فایل CSV بخواند، **نمرات آنها را پردازش کند** و نتایج را در یک فایل جدید ذخیره کند.