آچار فرانسه برنامه‌نویسی (تمرین کدنویسی)

موضوع: برنامه نویسی پایتون (۳)

این جزوه برای یادگیری بهتر و عملی تهیه شده است. ابتدا دستورات کاربردی را مرور شده و سپس مثال‌های کاربردی حل شده است

هدف: یادگیری دستورات اصلی همراه با کدنویسی عملی

ما در این پودمان که شامل مباحث بسیار مهم و پایه‌ای است، قرار است به صورت مفهومی، عمیق و با مثال‌های دنیای واقعی کالبدشکافی کنیم. این سبک یادگیری باعث می‌شود منطق پشت کدها را درک کنید، نه اینکه فقط آن‌ها را حفظ کنید.

در اینجا ۴ رکن اصلی این پودمان را با هم تحلیل می‌کنیم:

۱. توابع انعطاف‌پذیر (*args و **kwargs)

چرا به این نیاز داریم؟

در توابع معمولی، شما مجبورید دقیقاً بگویید چند ورودی می‌خواهید (مثلاً ۲ عدد برای جمع). اما در دنیای واقعی، شاید بخواهید ۱۰ عدد را جمع کنید یا شاید ۱۰۰ عدد.

- **مفهوم *args (آرگومان‌های ستاره‌دار):**

تصور کنید یک کیسه جادویی به تابع می‌دهید. شما نمی‌دانید چند توپ داخل کیسه است (۱۰ تا یا یکی)، اما تابع شما طوری طراحی شده که زیپ کیسه را باز می‌کند و دانه‌دانه روی آن‌ها عملیات انجام می‌دهد!.

  - در کد: وقتی *args می‌نویسید، پایتون تمام ورودی‌های اضافی را در یک بسته (Tuple) می‌ریزد. شما با یک حلقه for داخل آن می‌چرخید.

- **مفهوم **kwargs (آرگومان‌های کلیدواژه‌ای):**

این مثل سفارش دادن پیتزا با جزئیات دلخواه است. شما فقط نمی‌گویید "پیتزا"، بلکه می‌گویید: نان=نازک، سس = سیر، پنیر = زیاد.

- در کد: پایتون این‌ها را به صورت یک دیکشنری (لغت‌نامه) ذخیره می‌کند که هر ورودی یک نام (Key) و یک مقدار (Value) دارد.

---

۲. مفهوم فضای نام (Scope) و متغیر Global

چرا به این نیاز داریم؟

برای اینکه متغیرها با هم قاطی نشوند. فرض کنید شما در اتاق خودتان یک متغیر به نام "دما" دارید و در شهر هم یک متغیر "دما" داریم. این دو نباید روی هم اثر بگذارند.

- **محلی (Local):** متغیری که داخل تابع ساخته می‌شود، مثل "دمای اتاق خواب" است. بیرون از اتاق کسی آن را نمی‌شناسد.
- **سراسری (Global):** متغیری که در بدنه اصلی برنامه است، مثل "دمای شهر" است.
- چالش: اگر داخل تابع بخواهید "دمای شهر" را تغییر دهید، پایتون به طور پیش‌فرض اجازه نمی‌دهد (برای امنیت).
- **راه حل:** با دستور global x به پایتون می‌گویید: «منظورم از x، همان x اصلی و سراسری است، نه یک متغیر جدید داخلی»

---

۳. شی‌گرایی (OOP)؛ قلب تپنده برنامه‌نویسی مدرن

این مهم‌ترین بخش این پودمان است.

مفهوم داستان:

تا قبل از این، برنامه‌نویسی ما مثل نوشتن یک "دستور آشپزی" خط به خط بود (اول این کار را بکن، بعد آن کار). اما شی‌گرایی یعنی مدل‌سازی دنیای واقعی.

- **کلاس (Class) چیست؟**

کلاس مثل نقشه ساخت یا قالب برش شیرینی است. نقشه ساخت خودرو، خودِ خودرو نیست؛ فقط می‌گوید خودرو چه ویژگی‌هایی دارد (رنگ، مدل) و چه کارهایی می‌کند (توقیف، حرکت).

- **شیء (Object) چیست؟**

شیء، همان خودرویی است که از روی نقشه ساخته شده و در خیابان حرکت می‌کند. شما می‌توانید از روی یک کلاس (Car)، هزاران شیء (car۱, car۲, ...) بسازید که هر کدام رنگ و مدل متفاوتی دارند.

- **جادوی __init__ (تابع سازنده):**

وقتی در کارخانه یک ماشین تولید می‌شود، نمی‌شود آن را بدون رنگ و موتور بیرون داد. تابع __init__ همان لحظه اول تولدِ شیء اجرا می‌شود و ویژگی‌های اولیه (مثل رنگ و مدل) را به آن می‌چسباند.

- **داستان self:**

این کلمه گیج‌کننده است اما مفهومش ساده است: «مالِ خودم».

وقتی می‌گوییم self.color = red، یعنی رنگِ همین ماشینی که الان داریم می‌سازیم قرمز شود، نه رنگ ماشین کناری. بدون self، برنامه نمی‌فهمد این ویژگی متعلق به کدام ماشین است.

---

۴. رابط گرافیکی (Tkinter)؛ خداحافظی با صفحه سیاه

مفهوم:

ویندوز بر اساس "رویداد" (Event) کار می‌کند. یعنی برنامه منتظر می‌ماند تا شما کلیک کنید.

- **حلقه بی‌پایان (mainloop):**

وقتی پنجره ساخته می‌شود، اگر برنامه تمام شود، پنجره فوراً بسته می‌شود. دستور mainloop مثل یک نگهبان است که می‌گوید: «پنجره را باز نگه دار و گوش‌به‌زنگ باش تا کاربر کاری انجام دهد (مثل کلیک موس)»۱۰.

- **چیدمان (Grid):**

صفحه را مثل یک جدول اکسل (سطر و ستون) می‌بینند. شما می‌گویید دکمه در سطر ۱ و ستون ۲ باشد. این روش، چیدن اجزا را منظم می‌کند.

تحلیل پروژه‌ی ماشین‌حساب (صفحه ۵۳)

این پروژه تمام درس را جمع‌بندی کرده است:

۱. **GUI:** دکمه‌ها و کادر نمایش دارد.

۲. **تابع lambda:** (نکته طلایی)

در خطوط ۲۹ تا ... می‌بینید نوشته command=lambda: press(۱)

- **چرا؟** چون اگر پرانتز بگذاریم press(۱) ، تابع همان لحظه اجرا می‌شود. اما ما می‌خواهیم وقتی کلیک شد اجرا شود. lambda یک ترفند است که اجرای تابع را به تعویق می‌اندازد تا زمانی که دکمه فشار داده شود.

۳. مدیریت خطا (try/except):

در محاسبه (خط ۶۳۳)، از try استفاده کرده تا اگر کاربر چیز بی‌معنی وارد کرد (مثلاً تقسیم بر صفر)، برنامه کرش نکند و مودبانه بنویسد "Error ".

در ادامه، برای هر کدام از آن ۴ رکن اصلی، **مثال‌های کُدنویسی شده** را آورده‌ام تا دقیقاً ببینید آن "داستان‌ها" در محیط VS Code چه شکلی می‌شوند:

۱. مثال برای توابع انعطاف‌پذیر (*args و **kwargs)

یادآوری مفهوم: همان کیسه جادویی و سفارش پیتزا.

مثال کد (کیسه جادویی اعداد):

در اینجا ما نمی‌دانیم کاربر چند عدد می‌فرستد، اما همه را جمع می‌زنیم.

```
def jam_adad(*args):            # اون ستاره یعنی هر چی عدد اومد بریز تو کیسه args
    total = 0
    for number in args:         # حالا دونه دونه از کیسه دربیار
        total += number
    return total

print(jam_adad(10, 20))         # خروجی: 30
print(jam_adad(1, 2, 3, 4, 5)) # خروجی: 15 (می‌بینی؟ تعداد مهم نیست)
```

مثال کد (سفارش پیتزا با **kwargs):

در اینجا ورودی‌ها برچسب (اسم) دارند.

```python
def sefares_pizza(**kwargs):    # باش "کلید:مقدار" یعنی منتظر اون دو ستاره
    for item, value in kwargs.items():
        print(f"{item} : {value}")

sefares_pizza(nan="Italian", panir="Mozarella", sos="Sir")
# خروجی:
# nan : Italian
# panir : Mozarella
# sos : Sir
```

---

۲. مثال برای فضای نام (Scope) و global

یادآوری مفهوم: دمای شهر (سراسری) در مقابل دمای اتاق (محلی).

مثال کد (تغییر دمای شهر از داخل اتاق):

```python
dama_shahr = 30  # متغیر سراسری (بیرون تابع)

def taghir_dama():
    global dama_shahr     # اجازه بگیر که میخوای متغیر اصلی رو عوض کنی
    dama_shahr = 40       # حالا تغییر اعمال میشه
    print("Dama dar tabe:", dama_shahr)

taghir_dama()
print("Dama dar shahr:", dama_shahr) # 40 :خروجی (اگر global بود 30 بود، این هنوز نمیذاشتیم)
```

۳. مثال برای شی‌گرایی (OOP)

یادآوری مفهوم: نقشه (کلاس) و ماشین واقعی (شیء).

مثال کد (ساخت کارخانه ماشین‌سازی):

این همان مثال استاندارد کتاب شماست که کمی ساده‌تر شده است.

```
# 1. کلاس (نقشه ساخت)
class Car:
    # تابع سازنده (لحظه تولد ماشین)
    def __init__(self, name, color):
        self.name = name       # اسم این ماشین یکه چیزی که ورودی اومده
        self.color = color     # رنگ این ماشین یکه رنگی که ورودی اومده

# 2. شیء (ساخت ماشین واقعی از روی نقشه)
mashin1 = Car("Pride", "White")  # ماشین اول: پراید سفید
mashin2 = Car("Peugeot", "Black") # ماشین دوم: پژو مشکی

# 3. دسترسی به ویژگی‌ها
print(mashin1.color)  # خروجی: White (رنگ ماشین خودمه، کاری به پژو نداره)
print(mashin2.color)  # خروجی: Black
```

۴. مثال برای رابط گرافیکی و Lambda

یادآوری مفهوم: mainloop (نگهبان پنجره) و lambda (جلوگیری از اجرای خودسرانه).

مثال کد (یک دکمه که تا کلیک نکنی، سلام نمی‌کند):

این کد نشان می‌دهد چرا در ماشین‌حساب از lambda استفاده شده است.

```
from tkinter import *

window = Tk()
window.geometry("300x200")

def say_hello():
    print("Salam!")

# روش غلط (بدون lambda):
# Button(window, command=say_hello).pack()
# بلافاصله بعد از اجرای برنامه، خودش چاپ میکنه "Salam" (بدون کلیک شما!)

# روش درست (با lambda):
btn = Button(window, text="Click Me", command=lambda: say_hello())
# حالا "say_hello" اجرا نمیشه، مگر اینکه دکمه فشار داده بشه.

btn.pack()
window.mainloop() # پنجره باز می‌مونه
```

این مثال‌ها را می‌توانید در **VS Code** کپی کنید و اجرا بگیرید تا نتیجه را به چشم ببینید.