تمرین ۱: جمع کننده لیست

**مفاهیم اصلی:** Lists, Loops, Functions, Numbers

**شرح تمرین:** تابعی بنویسید که یک لیست از اعداد را به عنوان ورودی دریافت کرده و مجموع تمام اعداد داخل آن را محاسبه و برگرداند.

**داده ورودی (Input):**

|  |  |
| --- | --- |
| 1 | number\_list = [10, 20, 30, 40] |

**وظایف شما:**

۱. یک تابع بنویسید که یک لیست به عنوان آرگومان بگیرد.

۲. یک متغیر برای نگهداری مجموع اعداد (مثلاً total) با مقدار اولیه صفر تعریف کنید.

۳. روی هر عدد در لیست ورودی حلقه بزنید.

۴. در هر مرحله از حلقه، عدد فعلی را به متغیر total اضافه کنید.

۵. پس از پایان حلقه، مقدار نهایی total را برگردانید.

**خروجی مورد انتظار (Output):**

|  |  |
| --- | --- |
| 1 | 100 |

**مثال ساده برای درک بهتر:**

**ورودی:**

|  |  |
| --- | --- |
| 1 | simple\_data = [1, 2, 3] |

**خروجی:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 6  تمرین ۲: پیدا کردن بزرگترین عدد  **مفاهیم اصلی:** Lists, Loops, if-else, Functions  **شرح تمرین:** تابعی بنویسید که یک لیست از اعداد را دریافت کرده و **بدون استفاده از تابع max()**،  بزرگترین عدد موجود در لیست را پیدا و برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | number\_list = [150, 80, 277, 444, 274] |   **وظایف شما:**  ۱. یک تابع بنویسید که یک لیست به عنوان آرگومان بگیرد.  ۲. اولین عدد لیست را به عنوان بزرگترین عدد موقت در نظر بگیرید و در یک متغیر (مثلاً largest) ذخیره کنید.  ۳. روی بقیه اعداد لیست حلقه بزنید.  ۴. در هر مرحله، عدد فعلی را با largest مقایسه کنید. اگر عدد فعلی بزرگتر بود، مقدار largest را با آن جایگزین کنید.  ۵. پس از پایان حلقه، مقدار largest را برگردانید.  **خروجی مورد انتظار (Output)** :   |  |  | | --- | --- | | 1 | 277 |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_data = [100, 500, 200] |   **خروجی:**   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 500  تمرین ۳: تکرار کننده رشته  **مفاهیم اصلی:** Strings, Basic Operators, Functions  **شرح تمرین:** تابعی بنویسید که یک رشته (مثلاً Hi) و یک عدد صحیح (مثلاً 3) را به عنوان ورودی دریافت کرده و  رشته‌ای جدید که حاصل تکرار رشته ورودی به تعداد آن عدد است را برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | text = "Python" repeat\_count = 4 |   **وظایف شما:**  ۱. یک تابع با دو آرگومان (یک رشته و یک عدد) بنویسید.  ۲. از عملگر \* برای تکرار رشته استفاده کنید.  ۳. نتیجه را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | 'Python Python Python Python' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | text = "Ho" repeat\_count = 3 |   **خروجی:** |   تمرین ۴: تشخیص زوج یا فرد بودن عدد  **مفاهیم اصلی:** Numbers, Basic Operators (%), if-else  **شرح تمرین:** تابعی بنویسید که یک عدد صحیح را به عنوان ورودی دریافت کرده و در صورتی که عدد زوج باشد  رشته "Even" و در صورتی که فرد باشد رشته "Odd" را برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | number = 77 |   **وظایف شما:**  ۱. یک تابع بنویسید که یک عدد به عنوان آرگومان بگیرد.  ۲. از عملگر باقیمانده (%) برای بررسی بخش‌پذیری عدد بر ۲ استفاده کنید.  ۳. اگر باقیمانده تقسیم عدد بر ۲ برابر با صفر بود، رشته "Even" را برگردانید.  ۴. در غیر این صورت، رشته "Odd" را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | 'Odd' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | number = 10 |   **خروجی:**   |  |  | | --- | --- | | 1 | 'Even' |   تمرین ۵: دسترسی به مقدار در دیکشنری  **مفاهیم اصلی:** Dictionary, Functions  **شرح تمرین:** تابعی بنویسید که یک دیکشنری و یک کلید (key) را به عنوان ورودی دریافت کند و  مقدار (value) متناظر با آن کلید را برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6 | user\_profile = {      "username": "ali123",      "email": "ali.reza@example.com",      "level": 15  }  key\_to\_find = "email" |   **وظایف شما:**  ۱. یک تابع با دو آرگومان (یک دیکشنری و یک کلید) بنویسید.  ۲. با استفاده از براکت []، به مقدار متناظر با کلید ورودی در دیکشنری دسترسی پیدا کنید.  ۳. مقدار پیدا شده را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | 'ali.reza@example.com' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_dict = {"color": "blue"} key = "color" |   **خروجی:**   |  |  | | --- | --- | | 1 | 'blue' |   تمرین ۶: شمارش حروف صدادار  **مفاهیم اصلی:** Strings, Loops, if-else  **شرح تمرین:** تابعی بنویسید که یک رشته را به عنوان ورودی دریافت کرده و تعداد حروف صدادار (a, e, i, o, u)  موجود در آن را بشمارد. این شمارش باید به حروف بزرگ و کوچک حساس نباشد.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | input\_string = "Programming is Fun" |   **وظایف شما:**  ۱. یک تابع با یک آرگومان رشته‌ای بنویسید.  ۲. یک متغیر شمارنده با مقدار اولیه صفر تعریف کنید.  ۳. لیستی از حروف صدادار (['a', 'e', 'i', 'o', 'u']) تعریف کنید.  ۴. روی هر کاراکتر از رشته ورودی (که به حروف کوچک تبدیل شده) حلقه بزنید.  ۵. بررسی کنید که آیا کاراکتر فعلی در لیست حروف صدادار شما وجود دارد یا خیر.  ۶. اگر وجود داشت، شمارنده را یکی افزایش دهید.  ۷. در نهایت، مقدار شمارنده را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | output = 5 |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_data = "Hello" |   **خروجی:**   |  |  | | --- | --- | | 1 | output = 2 |   تمرین ۷: ترکیب دو لیست  **مفاهیم اصلی:** Lists, Basic Operators, Functions  **شرح تمرین:** تابعی بنویسید که دو لیست را به عنوان ورودی دریافت کرده و یک لیست جدید که شامل تمام اعضای هر دو لیست  است را برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | list\_a = [1, 2, 3]  list\_b = ['a', 'b', 'c'] |   **وظایف شما:**  ۱. یک تابع با دو آرگومان (دو لیست) بنویسید.  ۲. از عملگر + برای الحاق دو لیست به یکدیگر استفاده کنید.  ۳. لیست جدید و ترکیب شده را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | [1, 2, 3, 'a', 'b', 'c'] |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | l1 = [10] l2 = [20] |   **خروجی:**   |  |  | | --- | --- | | 1 | [10, 20] |   تمرین ۸: تابع خوشامدگویی با Type Hints  **مفاهیم اصلی:** Functions, Strings, Function Type Hints  **شرح تمرین:** یک تابع ساده بنویسید که یک نام (رشته) را دریافت کرده و یک پیام خوشامدگویی به صورت  "Hello, [name]!" برگرداند. در تعریف تابع حتماً از Type Hints استفاده کنید.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | name = "Sara" |   **وظایف شما:**  ۱. تابعی به نام greet تعریف کنید.  ۲. برای آرگومان ورودی name، تایپ str را مشخص کنید.  ۳. مشخص کنید که مقدار بازگشتی تابع نیز از نوع str است.  ۴. با استفاده از f-string، پیام خوشامدگویی را بسازید و آن را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | 'Hello, Sara!' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | name = "World" |   **خروجی:**   |  |  | | --- | --- | | 1 | 'Hello, World!' |   تمرین ۹: ساخت یک تاپل ساده  **مفاهیم اصلی:** Tuples, Functions  **شرح تمرین:** تابعی بنویسید که دو آرگومان با هر نوعی (مثلاً یک عدد و یک رشته) دریافت کرده و  یک tuple که شامل آن دو آرگومان است را برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | item1 = "Laptop" item2 = 2500 |   **وظایف شما:**  ۱. یک تابع با دو آرگومان بنویسید.  ۲. در دستور return، دو آرگومان را با یک ویرگول بین آن‌ها قرار دهید تا یک تاپل ساخته شود.  ۳. تاپل ساخته شده را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | ('Laptop', 2500) |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | i1 = 1 i2 = 2 |   **خروجی:**   |  |  | | --- | --- | | 1 | (1, 2) |   تمرین ۱۰: بررسی عضویت در یک Set  **مفاهیم اصلی:** Sets, Functions, if-else  **شرح تمرین:** تابعی بنویسید که یک set از اعداد و یک عدد را به عنوان ورودی دریافت کند.  اگر عدد مورد نظر در set وجود داشت، True و در غیر این صورت False برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | id\_set = {101, 105, 203, 404}  item\_to\_check = 203 |   **وظایف شما:**  ۱. یک تابع با دو آرگومان (یک set و یک عدد) بنویسید.  ۲. از عملگر in برای بررسی وجود عدد در set استفاده کنید.  ۳. نتیجه این بررسی (که یک مقدار بولین True یا False است) را مستقیماً برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | True |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2 | s = {10, 20}  item = 30 |   **خروجی:**   |  |  | | --- | --- | | 1 | False |   تمرین 11: محاسبه موجودی انبار  **مفاهیم اصلی:** Dictionary, Lists, Loops, Functions  **شرح تمرین:** شما مدیر انبار یک فروشگاه هستید. تابعی بنویسید که لیست تراکنش‌ها را دریافت کرده و  موجودی نهایی هر کالا را محاسبه کند. هر تراکنش یک تاپل است که شامل نام کالا، نوع تراکنش ('add' یا 'remove') و  تعداد است.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | transactions = [      ("Laptop", "add", 5),      ("Mouse", "add", 10),      ("Laptop", "remove", 2),      ("Keyboard", "add", 7),      ("Mouse", "remove", 3),      ("Laptop", "add", 1),  ] |   **وظایف شما:**  ۱. یک تابع بنویسید که لیست transactions را بگیرد.  ۲. یک دیکشنری خالی به نام inventory برای نگهداری موجودی نهایی کالاها ایجاد کنید.  ۳. روی هر تراکنش در لیست حلقه بزنید.  ۴. اگر تراکنش از نوع 'add' بود، تعداد را به موجودی آن کالا در inventory اضافه کنید.  (اگر کالا از قبل وجود نداشت، آن را با مقدار فعلی ایجاد کنید).  ۵. اگر تراکنش از نوع 'remove' بود، تعداد را از موجودی آن کالا کم کنید.  ۶. در نهایت، دیکشنری inventory را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | {'Laptop': 4, 'Mouse': 7, 'Keyboard': 7} |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_data = [("Pen", "add", 10), ("Pen", "remove", 3)] |   **خروجی:**   |  |  | | --- | --- | | 1 | {'Pen': 7} |   تمرین 12: پیدا کردن موارد مشترک در دو لیست  **مفاهیم اصلی:** Lists, Sets, Functions  **شرح تمرین:** تابعی بنویسید که دو لیست را به عنوان ورودی دریافت کرده و لیستی از آیتم‌هایی که در **هر دو** لیست  وجود دارند را برگرداند. ترتیب آیتم‌ها در خروجی مهم نیست و نباید آیتم تکراری وجود داشته باشد.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | list1 = [1, 2, 3, 4, 5]  list2 = [4, 5, 6, 7, 8] |   **وظایف شما:**  ۱. یک تابع با دو آرگومان (دو لیست) بنویسید.  ۲. برای حل بهینه این مسئله، هر دو لیست را به set تبدیل کنید.  ۳. از عملگر اشتراک (&) برای پیدا کردن آیتم‌های مشترک بین دو set استفاده کنید.  ۴. نتیجه را که یک set است، دوباره به list تبدیل کرده و برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | [4, 5] |   (توجه: ترتیب ممکن است متفاوت باشد، مثلا [5, 4] هم صحیح است)  **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | l1 = ['a', 'b', 'c'] l2 = ['x', 'y', 'b'] |   **خروجی:**   |  |  | | --- | --- | | 1 | ['b'] |   تمرین 13: چرخاندن یک لیست  **مفاهیم اصلی:** Lists, Functions  **شرح تمرین:** تابعی بنویسید که یک لیست و یک عدد صحیح k را دریافت کند و لیست را به  اندازه k خانه به سمت راست بچرخاند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | my\_list = [1, 2, 3, 4, 5, 6, 7]  k = 3 |   **وظایف شما:**  ۱. یک تابع با دو آرگومان (لیست و عدد k) بنویسید.  ۲. با استفاده از اسلایسینگ (slicing)، k آیتم آخر لیست را جدا کنید.  ۳. بقیه آیتم‌های لیست (از ابتدا تا k خانه مانده به آخر) را نیز جدا کنید.  ۴. لیست اول (که شامل k آیتم آخر بود) را به ابتدای لیست دوم بچسبانید.  ۵. لیست جدید و چرخانده شده را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | [5, 6, 7, 1, 2, 3, 4] |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2 | simple\_list = ['a', 'b', 'c']  k = 1 |   **خروجی:**   |  |  | | --- | --- | | 1 | ['c', 'a', 'b'] |   تمرین 14: رمز سزار (Caesar Cipher)  مفاهیم اصلی: Strings, Loops, Functions, Basic Operators  توضیح برای کسانی که نمیدونن رمز سزار چی هست؟ : “رمز سزار” یکی از قدیمی‌ترین و ساده‌ترین روش‌های رمزنگاریه. فکر کنید حروف الفبا  روی یک چرخ نوشته شدن. کاری که شما می‌کنید اینه که برای هر حرف از متن اصلی، چرخ رو به اندازه یک  عدد مشخص (مثلاً ۳ تا) می‌چرخونید و حرف جدید رو جایگزین می‌کنید.  مثلاً اگه عدد ما ۳ باشه:  حرف a میشه d.  حرف b میشه e.  …  و وقتی به آخر الفبا می‌رسیم، دوباره از اول شروع می‌کنیم. مثلا حرف y میشه b.  شرح تمرین: تابعی برای پیاده‌سازی “رمز سزار” بنویسید. این تابع یک رشته و یک عدد برای شیفت (مثلاً 3) دریافت می‌کند و  هر حرف از رشته را به اندازه عدد شیفت در الفبا به جلو می‌برد. این تابع باید حروف z را دوباره  به a برگرداند (wrapping).  داده ورودی (Input):   |  |  | | --- | --- | | 1  2 | text = "hello"  shift = 3 |   وظایف شما: ۱. یک تابع با دو آرگومان (رشته و عدد شیفت) بنویسید. ۲. یک رشته شامل حروف الفبا (‘abcdefghijklmnopqrstuvwxyz’) تعریف کنید. ۳. روی هر کاراکتر از رشته ورودی حلقه بزنید. ۴. جایگاه (ایندکس) هر کاراکتر را در رشته الفبا پیدا کنید. ۵. ایندکس جدید را با اضافه کردن عدد شیفت به ایندکس فعلی محاسبه کنید. ۶. برای مدیریت wrapping، از عملگر باقیمانده (26%) استفاده کنید. ( توجه !! = در این تمرین، ما  با ۲۶ حرف الفبای انگلیسی کار داریم. می‌توانیم این حروف را از ۰ تا ۲۵ شماره‌گذاری کنیم (A=0, B=1, …, Z=25). حالا  فرض کنید می‌خواهیم حرف ‘Y’ (که شماره‌اش ۲۴ است) را با شیفت ۳ رمز کنیم:  جایگاه فعلی: 24 شیفت: 3 جایگاه جدید: 27 = 24 + 3 ما حرفی در جایگاه ۲۷ نداریم! اینجا همان “wrapping” یا برگشتن  به ابتدای الفبا اتفاق می‌افتد. ما می‌خواهیم بعد از Z دوباره به A برگردیم. اینجاست که 26 % به کمک ما  می‌آید: 1 = 27 % 26. باقیمانده تقسیم ۲۷ بر ۲۶ می‌شود ۱. حرفی که در جایگاه ۱ قرار دارد، حرف ‘B’ است.  این دقیقاً همان چیزی است که می‌خواهیم! (Y -> Z -> A -> B). پس به طور خلاصه، استفاده از 26 % تضمین می‌کند  که هرگاه شماره حرف جدید از ۲۵ بیشتر شد، به طور خودکار به ابتدای الفبا برگردد و محاسبات ما همیشه در  محدوده ۰ تا ۲۵ باقی بماند.) ۷. حرف متناظر با ایندکس جدید را به رشته خروجی اضافه کنید. ۸. رشته رمزنگاری شده را برگردانید.  خروجی مورد انتظار (Output):   |  |  | | --- | --- | | 1 | 'khoor' |   مثال ساده برای درک بهتر:  ورودی:   |  |  | | --- | --- | | 1  2 | text = "xyz"  shift = 2 |   خروجی:   |  |  | | --- | --- | | 1 | 'zab' |   تمرین 15: محاسبه فاکتور سبد خرید  **مفاهیم اصلی:** Dictionary, Loops, Functions  **شرح تمرین:** تابعی بنویسید که قیمت کل یک سبد خرید را محاسبه کند. شما دو دیکشنری به عنوان ورودی  دارید: یکی سبد خرید که شامل نام کالا و تعداد آن است، و دیگری منوی قیمت‌ها که شامل نام کالا و قیمت واحد آن است.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | cart = {"Apple": 3, "Banana": 5, "Orange": 2}  prices = {"Apple": 1.5, "Banana": 0.5, "Orange": 1.25} |   **وظایف شما:**  ۱. یک تابع با دو آرگومان (دیکشنری cart و دیکشنری prices) بنویسید.  ۲. یک متغیر برای نگهداری قیمت کل (total\_price) با مقدار اولیه صفر ایجاد کنید.  ۳. روی آیتم‌ها (کالا و تعداد) در دیکشنری cart حلقه بزنید.  ۴. برای هر کالا، قیمت واحد آن را از دیکشنری prices پیدا کنید.  ۵. قیمت کل آن کالا را با ضرب تعداد در قیمت واحد محاسبه کنید.  ۶. مبلغ محاسبه شده را به total\_price اضافه کنید.  ۷. در نهایت، total\_price را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | Output = 9.5 |   (محاسبه: (3 \* 1.5) + (5 \* 0.5) + (2 \* 1.25) = 4.5 + 2.5 + 2.5)  **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2 | simple\_cart = {"Book": 2}  simple\_prices = {"Book": 10} |   **خروجی:**   |  |  | | --- | --- | | 1 | Output = 20 |   تمرین 16: تحلیلگر فایل لاگ (Log)  **مفاهیم اصلی:** Strings, Dictionary, Lists, Functions  **سطح:** سخت  **شرح تمرین:** شما یک رشته چندخطی دارید که محتوای یک فایل لاگ وب‌سرور را شبیه‌سازی می‌کند.  تابعی بنویسید که این لاگ را تجزیه کرده و یک گزارش آماری تولید کند. گزارش باید شامل تعداد کل درخواست‌ها و  تعداد درخواست‌های موفق (کد وضعیت 200) و ناموفق (کدهای دیگر) برای هر آدرس IP باشد.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7 | log\_data = """  192.168.1.1 - [10/Jul/2025:22:14:15] "GET /api/user HTTP/1.1" 200  172.16.0.5 - [10/Jul/2025:22:14:16] "POST /api/login HTTP/1.1" 401  192.168.1.1 - [10/Jul/2025:22:14:18] "GET /api/products HTTP/1.1" 200  212.5.25.10 - [10/Jul/2025:22:14:20] "GET /static/style.css HTTP/1.1" 304  192.168.1.1 - [10/Jul/2025:22:14:22] "DELETE /api/user/1 HTTP/1.1" 500  """ |   **وظایف شما:**  ۱. تابعی بنویسید که رشته log\_data را بگیرد.  ۲. با استفاده از متدهای رشته‌ای، هر خط را به اجزای آن (IP، تاریخ، درخواست، کد وضعیت) تقسیم کنید.  ۳. یک دیکشنری برای ذخیره گزارش بسازید. کلیدهای این دیکشنری، آدرس‌های IP خواهند بود.  ۴. مقدار هر کلید، یک دیکشنری دیگر شامل کلیدهای total\_requests, successful\_requests و failed\_requests است.  ۵. با پیمایش خطوط لاگ، این دیکشنری گزارش را تکمیل کنید.  ۶. دیکشنری گزارش نهایی را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5 | {      '192.168.1.1': {'total\_requests': 3, 'successful\_requests': 2, 'failed\_requests': 1},      '172.16.0.5': {'total\_requests': 1, 'successful\_requests': 0, 'failed\_requests': 1},      '212.5.25.10': {'total\_requests': 1, 'successful\_requests': 0, 'failed\_requests': 1}  } |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_log = '1.1.1.1 - "GET /" 200' |   **خروجی:**   |  |  | | --- | --- | | 1 | {'1.1.1.1': {'total\_requests': 1, 'successful\_requests': 1, 'failed\_requests': 0}} |   تمرین17: مرتب‌سازی وظایف وابسته  **مفاهیم اصلی:** Dictionary, Lists, Sets, Functions, Recursion  **سطح:** بسیار سخت  **شرح تمرین:** در مدیریت پروژه‌ها، برخی وظایف باید قبل از وظایف دیگر انجام شوند. تابعی بنویسید که یک دیکشنری  از وظایف و پیش‌نیازهایشان را دریافت کرده و یک لیست از ترتیب صحیح انجام وظایف را برگرداند. این مسئله به  عنوان مرتب‌سازی توپولوژیک (Topological Sort) شناخته می‌شود.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7 | tasks = {      'A': ['B', 'C'],      'B': ['D'],      'C': ['D'],      'D': [],      'E': ['C']  } |   (برای انجام A، باید B و C انجام شوند)  **وظایف شما:**  ۱. یک تابع بنویسید که دیکشنری tasks را بگیرد.  ۲. از دو set برای نگهداری وظایف بازدید شده (visited) و مسیر فعلی بازگشت (recursion\_stack) برای تشخیص  دور (dependency cycle) استفاده کنید.  ۳. یک تابع داخلی بازگشتی (recursive helper) تعریف کنید.  ۴. این تابع بازگشتی باید برای هر وظیفه، ابتدا تمام پیش‌نیازهای آن را به صورت بازگشتی پیمایش کند.  ۵. پس از پیمایش پیش‌نیازها، خود وظیفه را به لیست نتیجه نهایی اضافه کنید.  ۶. اگر در حین پیمایش به یک وظیفه که در recursion\_stack است برخورد کردید، یعنی یک دور وجود دارد و باید خطا برگردانید.  ۷. لیست مرتب شده وظایف را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | ['D', 'B', 'C', 'E', 'A'] |   (توجه: خروجی‌های معتبر دیگری نیز ممکن است وجود داشته باشد)  **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_tasks = {'X': ['Y'], 'Y': []} |   **خروجی:**   |  |  | | --- | --- | | 1 | ['Y', 'X'] |   تمرین 18: مقایسه عمیق دو دیکشنری  **مفاهیم اصلی:** Dictionary, Recursion, Sets, Functions  **سطح:** بسیار سخت  **شرح تمرین:** تابعی بنویسید که دو دیکشنری تو در تو را مقایسه کرده و تفاوت‌های آن‌ها را در قالب یک دیکشنری  گزارش دهد. گزارش باید شامل کلیدهای اضافه‌شده، حذف‌شده و مقادیر تغییرکرده باشد.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | dict\_before = {      "user": {"name": "Ali", "email": "ali@a.com", "active": True},      "settings": {"theme": "dark", "notifications": True}  }  dict\_after = {      "user": {"name": "Ali", "email": "ali.r@a.com", "active": False},      "settings": {"theme": "light"},      "permissions": ["admin"]  } |   **وظایف شما:**  ۱. یک تابع بازگشتی بنویسید که دو دیکشنری و یک مسیر پیشوند (برای ردیابی کلیدهای تو در تو) را بگیرد.  ۲. تمام کلیدهای هر دو دیکشنری را با استفاده از set پیدا کنید.  ۳. کلیدهای مشترک، کلیدهای اضافه‌شده در dict\_after و کلیدهای حذف‌شده از dict\_before را مشخص کنید.  ۴. برای کلیدهای مشترک، مقادیر را مقایسه کنید. اگر مقدارها خودشان دیکشنری بودند، تابع را به صورت بازگشتی  فراخوانی کنید. اگر متفاوت بودند، تغییر را ثبت کنید.  ۵. تمام تفاوت‌ها را در یک دیکشنری ساختاریافته جمع‌آوری و برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | {      'added': ['permissions'],      'removed': ['settings.notifications'],      'changed': {          'user.email': ('ali@a.com', 'ali.r@a.com'),          'user.active': (True, False),          'settings.theme': ('dark', 'light')      }  } |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2 | d1 = {"a": 1}  d2 = {"a": 2, "b": 3} |   **خروجی:**   |  |  | | --- | --- | | 1 | {'added': ['b'], 'removed': [], 'changed': {'a': (1, 2)}} |   تمرین 19: شبیه‌ساز دیتابیس با قابلیت تراکنش  **مفاهیم اصلی:** Dictionary, Functions, Nonlocal, Sets  **سطح:** بسیار سخت  **شرح تمرین:** یک سیستم پایگاه داده ساده در حافظه (in-memory) با قابلیت تراکنش پیاده‌سازی کنید.  این سیستم باید از دستورات begin (شروع تراکنش)، put (ذخیره داده)، get (خواندن داده)، commit (ثبت نهایی تغییرات)  و rollback   (لغو تغییرات) پشتیبانی کند.  **وظایف شما:**  ۱. یک تابع کارخانه‌ای (factory function) به نام create\_database بنویسید.  ۲. در داخل آن، یک دیکشنری برای داده‌های اصلی و یک لیست برای نگهداری وضعیت تراکنش‌های تو در تو ایجاد کنید.  ۳. توابع داخلی begin, put, get, commit, و rollback را تعریف کنید.  ۴. تابع put باید داده‌ها را در آخرین تراکنش باز بنویسد، نه در پایگاه داده اصلی.  ۵. تابع get باید ابتدا در تراکنش‌ها (از جدیدترین به قدیمی‌ترین) و سپس در پایگاه داده اصلی جستجو کند.  ۶. تابع commit باید تمام تغییرات تمام تراکنش‌های باز را در پایگاه داده اصلی ادغام کند.  ۷. تابع rollback باید آخرین تراکنش باز را لغو کند.  ۸. از nonlocal برای دسترسی و تغییر وضعیت‌ها (دیکشنری اصلی و لیست تراکنش‌ها) استفاده کنید.  ۹. در نهایت، create\_database باید دیکشنری‌ای از این توابع را برگرداند.  **مثال استفاده:**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | db = create\_database()  db['put']('a', 1)  print(db['get']('a'))  # --> 1    db['begin']()  db['put']('a', 2)  print(db['get']('a'))  # --> 2  db['rollback']()  print(db['get']('a'))  # --> 1    db['begin']()  db['put']('b', 3)  db['commit']()  print(db['get']('b'))  # --> 3 |   تابع کارخانه‌ای (Factory Function) چیست؟  بچه‌ها نگران این اسم عجیب نباشید! **تابع کارخانه‌ای** یک نوع جدید از تابع در پایتون نیست، بلکه فقط یک **الگو**  **یا روش استفاده از توابع** است که شما همین الان بلدید. فکر کنید یک کارخانه دارید. کارخانه چه کار می‌کند؟  یک سری مواد اولیه می‌گیرد و یک محصول کامل تحویل می‌دهد. **تابع کارخانه‌ای** هم دقیقا همین کار را می‌کند:  یک تابع است که وظیفه‌اش **ساختن و تحویل دادن توابع دیگر** است.  چرا این کار مفید است؟  چون به ما اجازه می‌دهد یک **“محیط خصوصی”** یا یک **“حافظه اختصاصی”** برای مجموعه‌ای از توابع بسازیم.  در تمرین پایگاه داده ما، تابع create\_database همان **کارخانه** است. وقتی شما آن را فراخوانی  می‌کنید (db = create\_database())، اتفاقات زیر می‌افتد:  یک دیکشنری خصوصی برای نگهداری داده‌ها در حافظه ساخته می‌شود. چند تابع  داخلی (مثل put, get, commit و rollback) تعریف می‌شوند. این توابع به آن دیکشنری خصوصی دسترسی دارند و  می‌توانند آن را تغییر دهند (با کمک nonlocal). کارخانه، این توابع داخلی را به عنوان محصول نهایی به شما برمی‌گرداند.  نتیجه این است که شما یک موجودیت مستقل به نام db دارید که داده‌هایش از بقیه کد جداست و شما فقط از  طریق “دکمه‌هایی” که کارخانه بهتان داده (db['put'], db['get'] و…) می‌توانید با آن کار کنید.  تمرین 20: موتور قالب‌بندی ساده  **مفاهیم اصلی:** Strings, Dictionary, Loops, Functions  **سطح:** سخت  **شرح تمرین:** یک تابع بنویسید که به عنوان یک موتور قالب‌بندی ساده عمل کند. این تابع یک  رشته قالب (template) که شامل متغیرهایی به شکل {{variable\_name}} است و یک دیکشنری از داده‌ها  را دریافت کرده و رشته نهایی را تولید کند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5 | template\_string = "Hello {{user.name}}, your last login was on {{login\_date}}."  data = {      "user": {"name": "Reza"},      "login\_date": "2025-07-15"  } |   **وظایف شما:**  ۱. تابعی با دو آرگومان (رشته قالب و دیکشنری داده) بنویسید.  ۲. با استفاده از حلقه و متدهای رشته‌ای، تمام متغیرهای داخل {{...}} را پیدا کنید.  ۳. برای هر متغیر پیدا شده، مقدار متناظر با آن را از دیکشنری داده استخراج کنید.  ۴. از متد .replace() برای جایگزین کردن {{variable\_name}} با مقدار واقعی آن در رشته قالب استفاده کنید.  ۵. رشته نهایی تولید شده را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | "Hello Reza, your last login was on 2025-07-15." |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2 | template = "Color: {{c}}"  d = {"c": "blue"} |   **خروجی:**   |  |  | | --- | --- | | 1 | "Color: blue" |   تمرین 21: محدود کننده نرخ فراخوانی  **مفاهیم اصلی:** Functions, Nonlocal, Lists, Numbers (ماژول time)  **سطح:** بسیار سخت  **شرح تمرین:** در دنیای واقعی، برای جلوگیری از سوءاستفاده از APIها، نرخ دسترسی را محدود می‌کنند.  شما باید یک تابع کارخانه‌ای (factory function) به نام create\_rate\_limiter بنویسید که یک تابع،  یک حداکثر تعداد فراخوانی (مثلاً 3 بار) و یک بازه زمانی (مثلاً 10 ثانیه) را بگیرد. این کارخانه باید یک تابع جدید  برگرداند که فقط به تعداد مشخصی در بازه زمانی تعیین شده قابل اجرا باشد.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7 | import time    def my\_function():      print("Function executed!")    # ساخت یک تابع محدود شده: حداکثر 2 بار در 5 ثانیه  limited\_func = create\_rate\_limiter(my\_function, 2, 5) |   **وظایف شما:**  ۱. یک تابع کارخانه‌ای create\_rate\_limiter با سه آرگومان (تابع هدف، حداکثر فراخوانی، بازه زمانی) بنویسید.  ۲. در داخل کارخانه، از یک لیست برای نگهداری زمان فراخوانی‌های اخیر (timestamps) استفاده کنید.  این لیست باید با استفاده از nonlocal در دسترس تابع داخلی باشد.  ۳. یک تابع داخلی (wrapper) تعریف کنید که منطق اصلی را پیاده‌سازی کند.  ۴. قبل از اجرای تابع هدف، این wrapper باید لیست timestamps را بررسی کند و زمان‌های فراخوانی  که قدیمی‌تر از بازه زمانی مجاز هستند را حذف کند.  ۵. سپس بررسی کند که آیا تعداد فراخوانی‌های باقی‌مانده در لیست کمتر از حد مجاز است یا خیر.  ۶. اگر مجاز بود، زمان فعلی (time.time()) را به لیست اضافه کرده، تابع هدف اصلی را اجرا و نتیجه آن را برگرداند.  ۷. اگر مجاز نبود، پیامی مبنی بر اینکه “Rate limit exceeded” چاپ کرده و None برگرداند.  ۸. کارخانه باید تابع wrapper را به عنوان خروجی برگرداند.  **خروجی مورد انتظار (Output):** با اجرای کد زیر:   |  |  | | --- | --- | | 1  2  3  4  5 | limited\_func() # بار اول  limited\_func() # بار دوم  limited\_func() # بار سوم  time.sleep(5)  limited\_func() # بار چهارم (بعد از 5 ثانیه) |   خروجی باید چیزی شبیه این باشد:   |  |  | | --- | --- | | 1  2  3  4 | Function executed!  Function executed!  Rate limit exceeded  Function executed! |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2  3  4 | # محدودیت: 1 بار در 10 ثانیه  simple\_limited = create\_rate\_limiter(lambda: print("OK"), 1, 10)  simple\_limited()  simple\_limited() |   **خروجی:**   |  |  | | --- | --- | | 1  2 | OK  Rate limit exceeded |   تمرین 22: شبیه‌ساز سیستم فایل  **مفاهیم اصلی:** Dictionary, Functions, Nonlocal, Strings  **سطح:** سخت  **شرح تمرین:** یک شبیه‌ساز خط فرمان برای ناوبری در یک ساختار فایل که با دیکشنری تو در تو نمایش داده شده،  بنویسید. سیستم شما باید یک “وضعیت فعلی” (مسیر فعلی) داشته باشد و از دستورات pwd (نمایش مسیر  فعلی)، ls (نمایش محتویات مسیر فعلی) و cd (تغییر مسیر) پشتیبانی کند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | file\_system = {      'home': {          'user': {              'documents': {                  'report.txt': 'This is a report.'              },              'downloads': {}          },          'guest': {}      }  } |   **وظایف شما:**  ۱. یک تابع کارخانه‌ای create\_file\_navigator بنویسید که دیکشنری file\_system را بگیرد.  ۲. در داخل کارخانه، یک متغیر برای نگهداری مسیر فعلی (current\_path) به صورت یک لیست  (مثلاً ['home', 'user']) تعریف کنید.  ۳. توابع داخلی pwd, ls, و cd را پیاده‌سازی کنید که با استفاده از nonlocal به current\_path دسترسی دارند.  ۴. pwd: باید مسیر فعلی را به صورت رشته‌ای مانند '/home/user' برگرداند.  ۵. ls: باید لیستی از نام فایل‌ها و پوشه‌های موجود در مسیر فعلی را برگرداند.  ۶. cd: باید یک مسیر جدید بگیرد. اگر مسیر .. بود، یک پله به عقب برود. اگر نام یک پوشه بود، وارد  آن شود. باید مسیرهای نامعتبر را مدیریت کند.  ۷. کارخانه باید دیکشنری‌ای از این توابع را برگرداند.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | nav = create\_file\_navigator(file\_system)  print(nav['pwd']())         # --> '/'  print(nav['ls']())          # --> ['home']  nav['cd']('home')  nav['cd']('user')  print(nav['pwd']())         # --> '/home/user'  print(nav['ls']())          # --> ['documents', 'downloads']  nav['cd']('..')  print(nav['pwd']())         # --> '/home' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2  3 | simple\_fs = {'root': {'file.txt': ''}}  nav = create\_file\_navigator(simple\_fs)  nav['cd']('root') |   **خروجی**:   |  |  | | --- | --- | | 1 | nav['ls'](): ['file.txt'] |   تمرین 23: کنترل دسترسی مبتنی بر نقش  **مفاهیم اصلی:** Dictionary, Sets, Functions  **سطح:** سخت  **شرح تمرین:** یک سیستم ساده برای کنترل دسترسی مبتنی بر نقش (Role-Based Access Control) پیاده‌سازی کنید.  این سیستم باید بررسی کند که آیا یک کاربر با نقش‌های مشخص، اجازه انجام یک عمل خاص را دارد یا خیر.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13 | # نقش‌ها و مجوزهایشان  roles\_permissions = {      'guest': {'read'},      'editor': {'read', 'write'},      'admin': {'read', 'write', 'delete'}  }    # کاربران و نقش‌هایشان  users\_roles = {      'ali': {'guest'},      'reza': {'editor'},      'sara': {'admin', 'editor'} # سارا هم ادمین است و هم ویرایشگر  } |   **وظایف شما:**   1. تابعی به نام has\_permission با سه آرگومان بنویسید: نام کاربر (username)، عمل 2. مورد نظر (action)، و دو دیکشنری users\_roles و roles\_permissions. ۲. ابتدا نقش‌های کاربر مورد نظر را از دیکشنری users\_roles پیدا کنید. اگر کاربر 3. وجود نداشت، باید False برگرداند. ۳. یک set خالی برای نگهداری تمام مجوزهای آن کاربر ایجاد کنید. ۴. روی تمام نقش‌های کاربر حلقه بزنید. در هر حلقه، مجوزهای مربوط به آن نقش را 4. از roles\_permissions پیدا کرده و به set مجوزهای کاربر اضافه کنید (از متد .update() برای set استفاده کنید). ۵. در نهایت، بررسی کنید که آیا action درخواستی در set مجوزهای جمع‌آوری شده کاربر 5. وجود دارد یا خیر و نتیجه (True یا False) را برگردانید.   **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4 | print(has\_permission('ali', 'read', users\_roles, roles\_permissions))    # --> True  print(has\_permission('ali', 'write', users\_roles, roles\_permissions))   # --> False  print(has\_permission('reza', 'delete', users\_roles, roles\_permissions)) # --> False  print(has\_permission('sara', 'delete', users\_roles, roles\_permissions)) # --> True |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | has\_permission('reza', 'read', users\_roles, roles\_permissions) |   **خروجی:**   |  |  | | --- | --- | | 1 | True |   (چون ‘reza’ یک ‘editor’ است و ‘editor’ مجوز ‘read’ را دارد)  تمرین 24: احراز هویت دو مرحله‌ای  **مفاهیم اصلی:** Dictionary, Functions, Numbers (ماژول time), Strings  **سطح:** سخت  **شرح تمرین:** یک سیستم ساده برای تولید و اعتبارسنجی کدهای یکبار مصرف  (مانند Google Authenticator) شبیه‌سازی کنید. این سیستم باید بتواند برای یک کاربر یک کد ۶ رقمی تولید کند  و سپس بررسی کند که آیا کد وارد شده توسط کاربر صحیح و در بازه زمانی مجاز (مثلاً ۶۰ ثانیه) است یا خیر.  **وظایف شما:**  ۱. یک تابع کارخانه‌ای create\_tfa\_system بنویسید.  ۲. در داخل آن، یک دیکشنری برای نگهداری کدهای فعال (active\_codes) به شکل {'username': (code, generation\_time)} ایجاد کنید.  ۳. تابع داخلی generate\_code(username) را پیاده‌سازی کنید. این تابع باید یک کد ۶ رقمی تصادفی  (به صورت رشته) تولید کرده، آن را به همراه زمان فعلی (time.time()) در active\_codes ذخیره کند و خود کد را برگرداند.  ۴. تابع داخلی validate\_code(username, code) را پیاده‌سازی کنید.  ۵. این تابع باید بررسی کند آیا کاربری با آن نام در active\_codes وجود دارد یا خیر.  ۶. سپس بررسی کند آیا کد ذخیره شده با کد وارد شده برابر است یا خیر.  ۷. در نهایت، بررسی کند آیا زمان فعلی کمتر از ۶۰ ثانیه از زمان تولید کد گذشته است یا خیر.  ۸. اگر تمام شرایط برقرار بود، True برگردانده و کد را از active\_codes حذف کند. در غیر این صورت False برگرداند.  ۹. کارخانه باید دیکشنری‌ای از این دو تابع را برگرداند.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | tfa = create\_tfa\_system()  user\_code = tfa['generate\_code']('ali')  print(f"Generated code for ali: {user\_code}")    print(tfa['validate\_code']('ali', user\_code)) # --> True (بلافاصله)  print(tfa['validate\_code']('ali', user\_code)) # --> False (چون کد پس از استفاده حذف شد)    # سناریوی کد منقضی شده  user\_code\_2 = tfa['generate\_code']('reza')  # time.sleep(61) # فرض کنید 61 ثانیه گذشته  # print(tfa['validate\_code']('reza', user\_code\_2)) # --> False (چون منقضی شده) |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2  3 | tfa = create\_tfa\_system()  code = tfa['generate\_code']('test')  result = tfa['validate\_code']('test', 'wrong\_code') |   **خروجی**:   |  |  | | --- | --- | | 1 | result: False |   (چون کد اشتباه است)  تمرین 25: زمان‌بندی کنفرانس  **مفاهیم اصلی:** Lists, Dictionary, Functions, Loops  **سطح:** بسیار سخت  **شرح تمرین:** شما لیستی از سخنرانی‌ها با مدت زمانشان را در اختیار دارید. تابعی بنویسید که این سخنرانی‌ها  را در دو نوبت صبح و بعد از ظهر برنامه‌ریزی کند. هر نوبت یک ظرفیت زمانی مشخص دارد  (مثلاً نوبت صبح ۱۸۰ دقیقه و نوبت بعد از ظهر ۲۴۰ دقیقه). هدف، جا دادن بیشترین تعداد سخنرانی ممکن است.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | talks = [      {'title': 'Intro to Python', 'duration': 60},      {'title': 'Advanced Recursion', 'duration': 45},      {'title': 'Data Science Basics', 'duration': 120},      {'title': 'Web Scraping', 'duration': 30},      {'title': 'Lambda Functions', 'duration': 30},      {'title': 'Working with APIs', 'duration': 90},      {'title': 'Final Project Ideas', 'duration': 45},  ] |   **وظایف شما:**   1. تابعی بنویسید که لیست talks را بگیرد. ۲. یک دیکشنری برای نگهداری برنامه‌ نهایی با کلیدهای ‘morning\_session’ و ‘afternoon\_session’ ایجاد کنید. 2. هر کدام از اینها یک لیست از سخنرانی‌های برنامه‌ریزی شده هستند. ۳. دو متغیر برای نگهداری زمان باقی‌مانده هر نوبت تعریف کنید 3. (morning\_remaining = 180, afternoon\_remaining = 240) 4. . ۴. برای یک راه حل ساده (Greedy Approach)، ابتدا سخنرانی‌ها را بر اساس مدت 5. زمانشان به صورت نزولی مرتب کنید. ۵. روی لیست مرتب شده سخنرانی‌ها حلقه بزنید. برای هر سخنرانی، ابتدا سعی کنید آن را در نوبت صبح جا دهید. 6. اگر زمان کافی وجود داشت، آن را اضافه کرده و زمان باقی‌مانده را کم کنید. ۶. اگر در نوبت صبح جا نشد، همین کار را برای نوبت بعد از ظهر امتحان کنید. ۷. در نهایت، دیکشنری برنامه‌ریزی شده را برگردانید.   **خروجی مورد انتظار (Output):** (توجه: خروجی دقیق ممکن است بسته به پیاده‌سازی متفاوت باشد، اما ساختار  کلی باید مشابه زیر باشد)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13 | {      'morning\_session': [          {'title': 'Data Science Basics', 'duration': 120},          {'title': 'Intro to Python', 'duration': 60}      ],      'afternoon\_session': [          {'title': 'Working with APIs', 'duration': 90},          {'title': 'Advanced Recursion', 'duration': 45},          {'title': 'Final Project Ideas', 'duration': 45},          {'title': 'Web Scraping', 'duration': 30},          {'title': 'Lambda Functions', 'duration': 30}      ]  } |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2 | simple\_talks = [{'title': 'A', 'duration': 20}, {'title': 'B', 'duration': 10}]  # ظرفیت صبح: 25 دقیقه |   **خروجی (با ظرفیت صبح 25):**   |  |  | | --- | --- | | 1  2 | {'morning\_session': [{'title': 'A', 'duration': 20}],  'afternoon\_session': [{'title': 'B', 'duration': 10}]} |   تمرین 26: پیاده‌سازی قابلیت Undo/Redo  **مفاهیم اصلی:** Functions, Lists, Nonlocal  **سطح:** بسیار سخت  **شرح تمرین:** یک ویرایشگر متن ساده با قابلیت undo و redo پیاده‌سازی کنید.  شما باید یک تابع کارخانه‌ای بسازید که یک موجودیت ویرایشگر با حالت (state) داخلی ایجاد کند.  این ویرایشگر باید بتواند متن را تغییر دهد، تغییرات را به عقب برگرداند (undo) و عملیات عقب‌گرد را لغو کند (redo).  **وظایف شما:**  ۱. یک تابع کارخانه‌ای به نام create\_editor با یک مقدار اولیه برای متن (مثلاً initial\_text) بنویسید.  ۲. در داخل کارخانه، سه متغیر با قابلیت دسترسی nonlocal تعریف کنید: current\_text، پشته undo\_stack (برای نگهداری حالت‌های قبلی متن) و پشته redo\_stack (برای نگهداری حالت‌های undo شده).  ۳. تابع داخلی edit(new\_text) را پیاده‌سازی کنید. این تابع باید متن فعلی را در   undo\_stack پوش کند، redo\_stack را خالی کند (چون یک تغییر جدید تمام تاریخچه redo را بی‌اعتبار می‌کند)  و متن فعلی را آپدیت کند.  ۴. تابع داخلی undo() را پیاده‌سازی کنید. این تابع باید متن فعلی را در redo\_stack پوش کند و  آخرین حالت را از undo\_stack برداشته و به عنوان متن فعلی قرار دهد.   1. تابع داخلی redo() را پیاده‌سازی کنید. این تابع باید متن فعلی را در undo\_stack پوش کرده و 2. آخرین حالت را از redo\_stack بازیابی کند.   ۶. یک تابع get\_text() برای نمایش متن فعلی نیز بنویسید.  ۷. کارخانه باید دیکشنری‌ای از این توابع را برگرداند.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10 | editor = create\_editor("Hello")  editor['edit']("Hello World")  editor['edit']("Hello Python")  print(editor['get\_text']())  # --> "Hello Python"  editor['undo']()  print(editor['get\_text']())  # --> "Hello World"  editor['undo']()  print(editor['get\_text']())  # --> "Hello"  editor['redo']()  print(editor['get\_text']())  # --> "Hello World" |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2  3 | e = create\_editor("A")  e['edit']("B")  e['undo']() |   **خروجی ()e[‘get\_text’] :**   |  |  | | --- | --- | | 1 | "A" |   تمرین 27: اعتبارسنج JSON Schema  **مفاهیم اصلی:** Dictionary, Recursion, Functions, Type Hints  **سطح:** بسیار سخت  **شرح تمرین:** تابعی بنویسید که بررسی کند آیا یک آبجکت JSON (که در پایتون به صورت دیکشنری است) با  یک اسکیمای (schema) تعریف شده مطابقت دارد یا خیر. اسکیما ساختار، نوع داده‌ها و کلیدهای ضروری را مشخص می‌کند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | schema = {      "type": "object",      "properties": {          "name": {"type": "string"},          "email": {"type": "string"},          "age": {"type": "number"},          "address": {              "type": "object",              "properties": {                  "city": {"type": "string"},                  "zip\_code": {"type": "string"},              },              "required": ["city"],          },      },      "required": ["name", "email"],  }    valid\_json = {"name": "Ali", "email": "a@b.com", "age": 30}  invalid\_json = {"name": "Reza", "age": "twenty"}  # age باید number باشد |   **وظایف شما:**  ۱. یک تابع بازگشتی به نام is\_valid(data, schema) با Type Hints دقیق بنویسید.  ۲. تابع باید ابتدا نوع داده data را با فیلد type در schema مقایسه کند.  ۳. اگر نوع object بود، بررسی کند که تمام کلیدهای موجود در لیست required اسکیما، در data نیز وجود دارند.  ۴. سپس روی تمام کلیدهای data حلقه زده و برای هر کدام، به صورت بازگشتی تابع is\_valid را با مقدار آن کلید  و اسکیمای متناظرش در properties فراخوانی کند.  ۵. تابع باید برای انواع دیگر (string, number) نیز بررسی صحت نوع داده را انجام دهد.  ۶. در صورت عدم تطابق در هر مرحله، تابع باید False برگرداند. اگر کل اعتبارسنجی موفق بود، True برگرداند.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2 | print(is\_valid(valid\_json, schema))   # --> True  print(is\_valid(invalid\_json, schema)) # --> False |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | s = {      "type": "object",      "properties": {"a": {"type": "string"}},      "required": ["a"],  }  d = {      "a": 123,  } |   **خروجی:**   |  |  | | --- | --- | | 1 | False |   (چون مقدار a باید رشته باشد نه عدد)  تمرین 28: تجزیه‌گر Query String  **مفاهیم اصلی:** Strings, Dictionary, Functions  **سطح:** سخت  **شرح تمرین:** دو تابع بنویسید. یکی برای تجزیه (parse) یک رشته کوئری URL به یک دیکشنری،  و دیگری برای انجام عمل برعکس، یعنی تبدیل یک دیکشنری به رشته کوئری.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | query\_string = "user=ali&amp;page=2&amp;filter=active&amp;sort=name" |   **وظایف شما:**   1. تابع parse\_query(query\_str) را بنویسید. این تابع باید رشته را بر اساس & به زوج‌ها تقسیم کند. 2. سپس هر زوج را بر اساس = به کلید و مقدار تقسیم کرده و در یک دیکشنری ذخیره کند. 3. تابع build\_query(data\_dict) را بنویسید. این تابع باید روی آیتم‌های دیکشنری حلقه زده، هر زوج کلید-مقدار 4. را با = به هم چسبانده و در نهایت تمام زوج‌ها را با & به یکدیگر متصل کند تا رشته کوئری نهایی ساخته شود.   **خروجی مورد انتظار (Output)** **برای parse\_query:**   |  |  | | --- | --- | | 1 | {'user': 'ali', 'page': '2', 'filter': 'active', 'sort': 'name'} |   **مثال ساده برای درک بهتر:**  **ورودی**:   |  |  | | --- | --- | | 1 | parse\_query: s = "a=1&amp;b=2" |   **خروجی:**   |  |  | | --- | --- | | 1 | {'a': '1', 'b': '2'} |   تمرین 29: پاک‌سازی داده‌های استخراج شده از وب  **مفاهیم اصلی:** Strings, Lists, Dictionary, Functions  **سطح:** سخت  **شرح تمرین:** شما لیستی از رشته‌های نامرتب دارید که از یک وبسایت استخراج شده‌اند. تابعی بنویسید  که این لیست را پاک‌سازی کرده و به یک ساختار داده تمیز و یکدست (لیستی از دیکشنری‌ها) تبدیل کند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | scraped\_data = [      '\n    Price: $49.99\t',      '  Stock: Available (15 items) ',      'Name: "Super Widget"    ',      'Price: $19.95', # محصول دوم      'Stock: OUT OF STOCK',      'Name: "Basic Gadget"'  ] |   **وظایف شما:**  ۱. یک تابع بنویسید که لیست scraped\_data را بگیرد.  ۲. روی رشته‌های ورودی حلقه بزنید. هر رشته را با استفاده از متد .strip() از فضاهای خالی اضافی در ابتدا و انتها پاک کنید.  ۳. هر رشته پاک‌شده را بر اساس کالن (:) به دو بخش کلید و مقدار تقسیم کنید.  ۴. کلیدها را به یک فرمت استاندارد تبدیل کنید (مثلاً همه را به حروف کوچک و بدون فاصله).  ۵. مقادیر را نیز پاک‌سازی کنید: علامت $ را حذف کنید، مقادیر عددی را به float تبدیل کنید، و وضعیت موجودی را  به یک مقدار بولین (True/False) تبدیل کنید.   1. هر سه خط ورودی مربوط به یک محصول است. داده‌های پاک‌سازی شده هر محصول را در یک دیکشنری جمع‌آوری 2. کرده و به لیست نهایی اضافه کنید.   **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4 | [      {'price': 49.99, 'stock': True, 'name': 'Super Widget'},      {'price': 19.95, 'stock': False, 'name': 'Basic Gadget'}  ] |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_data = [' Price: $9.99 '] |   **خروجی:**   |  |  | | --- | --- | | 1 | [{'price': 9.99}] |   تمرین 30: مرتب‌سازی سفارشی با چندین کلید  **مفاهیم اصلی:** Lists, Dictionary, Lambda Function, Functions  **سطح:** سخت  **شرح تمرین:** تابعی بنویسید که لیستی از دیکشنری‌های دانشجویان را بر اساس دو معیار مرتب کند: ابتدا بر  اساس **نمره (score) به صورت نزولی** (بیشترین به کمترین)، و اگر نمرات برابر بودند، سپس بر اساس  **نام (name) به صورت الفبایی صعودی**.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6 | students = [      {'name': 'Reza', 'score': 85},      {'name': 'Ali', 'score': 92},      {'name': 'Sara', 'score': 85},      {'name': 'Maryam', 'score': 92}  ] |   **وظایف شما:**  ۱. یک تابع بنویسید که لیست students را بگیرد.  ۲. از تابع sorted() برای مرتب‌سازی استفاده کنید.  ۳. برای آرگومان key، یک تابع lambda بنویسید که یک **تاپل** را برگرداند.  ۴. عضو اول تاپل باید معیاری برای مرتب‌سازی نزولی باشد. برای این کار، از منفی نمره (-student['score']) استفاده کنید.  ۵. عضو دوم تاپل باید معیار مرتب‌سازی صعودی باشد که همان نام دانشجو (student['name']) است. پایتون  به طور خودکار ابتدا بر اساس عضو اول تاپل و سپس برای موارد مساوی، بر اساس عضو دوم مرتب می‌کند.  ۶. لیست مرتب شده نهایی را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5  6 | [      {'name': 'Ali', 'score': 92},      {'name': 'Maryam', 'score': 92}, # علی قبل از مریم آمده چون A قبل از M است      {'name': 'Reza', 'score': 85},      {'name': 'Sara', 'score': 85}   # رضا قبل از سارا آمده چون R قبل از S است  ] |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | d = [{'name': 'B', 'score': 10}, {'name': 'A', 'score': 10}] |   **خروجی:**   |  |  | | --- | --- | | 1 | [{'name': 'A', 'score': 10}, {'name': 'B', 'score': 10}] |   تمرین 31: پردازشگر داده‌های CSV  **مفاهیم اصلی:** Strings, Lists, Dictionary, Functions, Lambda  **سطح:** سخت  **شرح تمرین:** تابعی بنویسید که یک رشته چندخطی با فرمت CSV را دریافت کرده، آن را به لیستی از  دیکشنری‌ها تبدیل کند و سپس قابلیتی برای فیلتر کردن این داده‌ها بر اساس یک شرط فراهم کند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6 | csv\_data = """id,name,age,city  1,Ali,34,Tehran  2,Reza,25,Shiraz  3,Sara,34,Tehran  4,Maryam,41,Isfahan  """ |   **وظایف شما:**  ۱. تابع parse\_csv(csv\_str) را بنویسید. این تابع باید رشته را به خطوط تقسیم کند. خط  اول header است و بقیه خطوط، ردیف‌های داده هستند. خروجی باید لیستی از دیکشنری‌ها باشد  که کلیدهایشان از header گرفته شده است.  ۲. تابع filter\_data(data\_list, condition\_func) را بنویسید. این تابع لیستی از دیکشنری‌ها  (خروجی تابع قبل) و یک تابع شرطی (یک lambda) را می‌گیرد و فقط ردیف‌هایی را برمی‌گرداند  که در آن شرط صدق کنند.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | parsed = parse\_csv(csv\_data)  # خروجی parsed:  # [  #   {'id': '1', 'name': 'Ali', 'age': '34', 'city': 'Tehran'},  #   ...  # ]    # فیلتر کردن افرادی که سنشان بالای 30 است  filtered = filter\_data(parsed, lambda row: int(row['age']) > 30)  # خروجی filtered:  # [  #   {'id': '1', 'name': 'Ali', 'age': '34', 'city': 'Tehran'},  #   {'id': '3', 'name': 'Sara', 'age': '34', 'city': 'Tehran'},  #   {'id': '4', 'name': 'Maryam', 'age': '41', 'city': 'Isfahan'}  # ] |   تمرین 32: بررسی برنده در بازی دوز  **مفاهیم اصلی:** Lists (تو در تو), Functions, if-elif-else, Sets  **سطح:** متوسط  **شرح تمرین:** تابعی بنویسید که یک جدول بازی دوز ۳x۳ را به صورت یک لیست تو در تو دریافت کرده  و مشخص کند که آیا بازیکنی برنده شده است یا خیر.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | board\_x\_wins = [      ['X', 'O', 'O'],      [' ', 'X', ' '],      [' ', ' ', 'X']  ]    board\_no\_winner = [      ['X', 'O', 'X'],      ['O', 'X', 'O'],      ['O', 'X', 'O']  ] |   **وظایف شما:**  ۱. یک تابع check\_winner(board) بنویسید.  ۲. تمام ۸ حالت ممکن برای برنده شدن را بررسی کنید: ۳ سطر، ۳ ستون و ۲ قطر.  ۳. برای هر سطر، ستون یا قطر، بررسی کنید که آیا هر سه خانه آن یکسان (و غیرخالی) هستند یا  خیر. \* **نکته هوشمندانه:** می‌توانید هر سطر/ستون/قطر را به یک set تبدیل کنید. اگر طول set برابر  با ۱ بود (و عضو آن ‘ ‘ نبود)، یعنی یک نفر برنده شده.  ۴. اگر ‘X’ برنده بود، رشته "X" را برگردانید.  ۵. اگر ‘O’ برنده بود، رشته "O" را برگردانید.  ۶. اگر هیچکس برنده نشده بود، رشته "No winner" را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2 | print(check\_winner(board\_x\_wins))      # --> 'X'  print(check\_winner(board\_no\_winner)) # --> 'No winner' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | simple\_board = [['O', 'O', 'O'], ['X', ' ', 'X'], ['X', ' ', ' ']] |   **خروجی:**   |  |  | | --- | --- | | 1 | 'O' |   تمرین 33: منطق بازی سنگ، کاغذ، قیچی  **مفاهیم اصلی:** Functions, Dictionary, if-elif-else  **سطح:** متوسط  **شرح تمرین:** تابعی بنویسید که انتخاب بازیکن و انتخاب کامپیوتر را در بازی سنگ، کاغذ، قیچی گرفته و  نتیجه بازی را مشخص کند (برد، باخت یا مساوی).  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | player\_choice = "rock"  computer\_choice = "scissors" |   **وظایف شما:**  ۱. یک تابع determine\_winner(player, computer) بنویسید.  ۲. یک دیکشنری برای تعریف قوانین بازی ایجاد کنید. کلید، انتخاب برنده و مقدار، انتخاب بازنده است.  rules = {'rock': 'scissors', 'scissors': 'paper', 'paper': 'rock'}. ۳. ابتدا حالت مساوی  را بررسی کنید (اگر انتخاب بازیکن و کامپیوتر یکسان بود).  ۴. سپس با استفاده از دیکشنری rules بررسی کنید که آیا انتخاب بازیکن، انتخاب کامپیوتر را شکست می‌دهد یا خیر.  ۵. اگر هیچکدام از حالات بالا نبود، یعنی بازیکن باخته است.  ۶. نتیجه را به صورت یک رشته ('Player wins!', 'Computer wins!' یا 'It\'s a tie!') برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | 'Player wins!' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | determine\_winner('paper', 'paper') |   **خروجی:**   |  |  | | --- | --- | | 1 | 'It\'s a tie!' |   تمرین 34: مدیریت کوله‌پشتی بازیکن  **مفاهیم اصلی:** Dictionary, Functions, if-elif-else  **سطح:** متوسط  **شرح تمرین:** سیستمی برای مدیریت کوله‌پشتی (inventory) یک بازیکن در یک بازی نقش‌آفرینی  پیاده‌سازی کنید. کوله‌پشتی یک دیکشنری است که نام آیتم کلید و تعداد آن مقدار است.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | inventory = {'potion': 3, 'sword': 1} |   **وظایف شما:**   1. تابع add\_item(inv, item, quantity=1) را بنویسید. این تابع باید یک آیتم را به کوله‌پشتی اضافه کند. 2. اگر آیتم از قبل وجود داشت، تعدادش را افزایش دهد. 3. تابع remove\_item(inv, item, quantity=1) را بنویسید. این تابع باید یک آیتم را از کوله‌پشتی کم کند. 4. باید بررسی کنید که آیا به تعداد کافی از آن آیتم وجود دارد یا خیر. اگر تعداد به صفر رسید، آیتم 5. را از دیکشنری حذف کنید.   ۳. تابع display\_inventory(inv) را بنویسید که محتویات کوله‌پشتی را به شکل خوانا چاپ کند.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5 | inventory = {'potion': 3, 'sword': 1}  add\_item(inventory, 'gold', 50)  add\_item(inventory, 'potion', 2)  remove\_item(inventory, 'sword', 1)  display\_inventory(inventory) |   خروجی display\_inventory باید چیزی شبیه این باشد:   |  |  | | --- | --- | | 1  2  3 | Inventory:  - potion: 5  - gold: 50 |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | inv = {'arrow': 10} remove\_item(inv, 'arrow', 15) |   **خروجی تابع remove\_item:**   |  |  | | --- | --- | | 1 | (یک پیام خطا) Not enough 'arrow' in inventory. |   تمرین 35: شبیه‌ساز تاس‌ریختن  **مفاهیم اصلی:** Functions, Loops, Strings, Numbers (ماژول random)  **سطح:** متوسط  **شرح تمرین:** تابعی بنویسید که یک رشته با فرمت تاس‌های بازی‌های  رومیزی (مانند "3d6" به معنی ریختن ۳ تاس ۶ وجهی) را دریافت کرده و مجموع نتیجه تاس‌ها را برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | dice\_notation = "3d6" |   **وظایف شما:**  ۱. یک تابع roll\_dice(notation) بنویسید.  ۲. رشته ورودی را بر اساس حرف 'd' به دو بخش تعداد تاس (number\_of\_dice) و تعداد وجه‌های  تاس (dice\_sides) تقسیم کنید.  ۳. این دو بخش را که به صورت رشته هستند، به عدد صحیح (int) تبدیل کنید.  ۴. یک متغیر برای نگهداری مجموع نتایج (total\_result) با مقدار اولیه صفر تعریف کنید.  ۵. یک حلقه به تعداد number\_of\_dice اجرا کنید.  ۶. در هر بار اجرای حلقه، یک عدد تصادفی بین ۱ و dice\_sides (با استفاده از random.randint) تولید  کرده و به total\_result اضافه کنید.  ۷. مقدار نهایی total\_result را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | یک عدد صحیح بین ۳ تا ۱۸ (چون نتیجه تصادفی است). |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | roll\_dice("1d20") |   **خروجی:**   |  |  | | --- | --- | | 1 | یک عدد صحیح بین ۱ تا ۲۰. |   تمرین 36: تولیدکننده مشخصات شخصیت بازی  **مفاهیم اصلی:** Dictionary, Lists, Loops, Functions, Numbers (ماژول random)  **سطح:** متوسط  **شرح تمرین:** در بسیاری از بازی‌های نقش‌آفرینی، مشخصات (stats) شخصیت با ریختن تاس مشخص می‌شود.  تابعی بنویسید که برای یک شخصیت، مشخصاتی مانند قدرت، چابکی و هوش را با یک روش استاندارد  تولید کند: “برای هر مشخصه، ۴ تاس ۶ وجهی بریز، کمترین نتیجه را حذف کن و ۳ تاس باقی‌مانده را با هم جمع کن”.  **وظایف شما:**  ۱. یک تابع generate\_stats() بنویسید.  ۲. لیستی از نام مشخصات تعریف کنید:  stats\_names = ['Strength', 'Dexterity', 'Intelligence']. ۳. یک دیکشنری خالی برای نگهداری  نتیجه نهایی شخصیت ایجاد کنید.  ۴. روی هر نام در لیست stats\_names حلقه بزنید.  ۵. در هر حلقه، یک لیست برای نگهداری نتیجه ۴ تاس ایجاد کنید و آن را پر کنید.  ۶. لیست نتایج تاس را مرتب (sort) کرده و اولین عضو آن (کمترین) را حذف کنید.  ۷. مجموع سه عضو باقی‌مانده را محاسبه کرده و در دیکشنری نهایی با کلید نام همان مشخصه ذخیره کنید.  ۸. دیکشنری مشخصات شخصیت را برگردانید.  **خروجی مورد انتظار (Output):** یک دیکشنری با مقادیر تصادفی، مانند:   |  |  | | --- | --- | | 1 | {'Strength': 14, 'Dexterity': 9, 'Intelligence': 16} |   تمرین 37: منطق حدس کلمه در بازی Hangman  **مفاهیم اصلی:** Strings, Lists, Sets, Functions  **سطح:** متوسط  **شرح تمرین:** هسته اصلی بازی هنگ‌من را پیاده‌سازی کنید. شما باید تابعی بنویسید که کلمه مخفی،  حروفی که تا الان حدس زده شده، و حرف جدیدی که بازیکن حدس زده را دریافت کرده و وضعیت  جدید نمایش کلمه را برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2  3 | secret\_word = "python"  guessed\_letters = {"p", "o", "n"}  new\_guess = "t" |   **وظایف شما:**  ۱. یک تابع update\_display(secret, guessed) بنویسید.  ۲. این تابع باید یک رشته برای نمایش کلمه (مثلاً \_ \_ \_ \_ \_ \_) بسازد.  ۳. روی هر حرف از secret\_word حلقه بزنید.  ۴. اگر حرف در مجموعه guessed\_letters وجود داشت، خود حرف را به رشته نمایش اضافه کنید.  ۵. در غیر این صورت، یک آندرلاین (\_) اضافه کنید.  ۶. رشته نمایش آپدیت شده را برگردانید.  **خروجی مورد انتظار (Output):**  با فراخوانی update\_display("python", {'p', 'o', 'n', 't'}):   |  |  | | --- | --- | | 1 | 'p y t \_ o n' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | update\_display("apple", {'p', 'l'}) |   **خروجی:**   |  |  | | --- | --- | | 1 | '\_ p p l \_' |   تمرین 38: سیستم امتیازدهی ساده  **مفاهیم اصلی:** Numbers, Functions, if-else  **سطح:** ساده  **شرح تمرین:** تابعی بنویسید که امتیاز فعلی یک بازیکن و مقدار امتیازی که کسب کرده (یا از دست داده) را  دریافت کند و امتیاز جدید را محاسبه کند. امتیاز بازیکن هرگز نباید کمتر از صفر شود.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | current\_score = 150  points\_change = -70  # از دست دادن امتیاز |   **وظایف شما:**  ۱. یک تابع update\_score(score, points) بنویسید.  ۲. امتیاز جدید را با جمع کردن score و points محاسبه کنید.  ۳. با استفاده از یک دستور if، بررسی کنید که آیا امتیاز جدید کمتر از صفر شده است یا خیر.  ۴. اگر کمتر از صفر بود، مقدار 0 را برگردانید.  ۵. در غیر این صورت، امتیاز جدید محاسبه شده را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2 | new\_score = update\_score(150, -70)  # --> 80  final\_score = update\_score(new\_score, -100)  # --> 0 |   خروجی final\_score باید 0 باشد.  **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | update\_score(20, 50) |   **خروجی:**   |  |  | | --- | --- | | 1 |  |   تمرین 39: استخراج پسوند فایل  **مفاهیم اصلی:** Strings, Functions  **سطح:** ساده  **شرح تمرین:** تابعی بنویسید که نام یک فایل را به صورت رشته دریافت کرده و پسوند آن فایل را برگرداند.  اگر فایل پسوندی نداشت، باید یک رشته خالی برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1 | filename = "document.report.pdf" |   **وظایف شما:**  ۱. یک تابع get\_extension(filename\_str) بنویسید.  ۲. بررسی کنید که آیا کاراکتر نقطه (.) در نام فایل وجود دارد یا خیر.  ۳. اگر وجود داشت، با استفاده از متد .split('.') رشته را تقسیم کرده و آخرین عضو لیست حاصل را که  همان پسوند است، برگردانید.  ۴. اگر نقطه وجود نداشت، یک رشته خالی ('') برگردانید.  **نکته:** استفاده از rsplit('.', 1) راه بهینه‌تری برای این کار است.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1 | 'pdf' |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | get\_extension("archive.zip") |   **خروجی:**   |  |  | | --- | --- | | 1 | 'zip' |   تمرین 40: افزودن کار به لیست انجام کارها  **مفاهیم اصلی:** Lists, if-else, Functions  **سطح:** ساده  **شرح تمرین:** تابعی بنویسید که یک لیست از کارها (To-Do List) و یک کار جدید را دریافت کند. اگر  کار جدید از قبل در لیست وجود نداشت، آن را به لیست اضافه کند و True برگرداند. در غیر این  صورت، هیچ تغییری ایجاد نکند و False برگرداند.  **داده ورودی (Input):**   |  |  | | --- | --- | | 1  2 | todo\_list = ["خرید شیر", "ورزش کردن"]  new\_task = "تماس با دوست" |   **وظایف شما:**  ۱. یک تابع add\_task(tasks, task) بنویسید.  ۲. با استفاده از عملگر in، بررسی کنید که آیا task جدید در لیست tasks وجود دارد یا خیر.  ۳. اگر وجود نداشت، آن را با متد .append() به لیست اضافه کرده و True را برگردانید.  ۴. اگر وجود داشت، False را برگردانید.  **خروجی مورد انتظار (Output):**   |  |  | | --- | --- | | 1  2  3  4  5 | result1 = add\_task(todo\_list, "تماس با دوست")  # --> True  # todo\_list حالا: ['خرید شیر', 'ورزش کردن', 'تماس با دوست']    result2 = add\_task(todo\_list, "ورزش کردن")  # --> False  # todo\_list بدون تغییر باقی می‌ماند |   **مثال ساده برای درک بهتر:**  **ورودی:**   |  |  | | --- | --- | | 1 | my\_list = ['a'] add\_task(my\_list, 'b') |   **خروجی:**   |  |  | | --- | --- | | 1 | True و my\_list به ['a', 'b'] تبدیل می‌شود. | |