הנחיות לפתירת המבדק

- **קראו היטב** את שלד התוכנית (skeleton).
- **מלאו** את החלקים החסרים עם פקודות נכונות בהתאם להוראות (השוואה, פעולה מתמטית, המרה לטיפוס, הכרזה על משתנים גלובליים וכו').
- **הריצו ובדקו** מספר פעמים עם קלט שונה.
- **שתפו** את הגרסה הסופית שלכם להגשה או במהלך השיעור.

1. **הורדת (או העתקת) קובץ התבנית** - התבנית מכילה שלד קוד עם מקומות שסומנו כ-`<FILL IN>`.
- שם תצטרכו להשלים את הקוד החסר (השוואות, חישובים, המרות לטיפוס נתונים, ועוד).

2. **מילוי החוסרים (Placeholders)** - בכל מקום בו כתוב `<FILL IN>`, השלימו בפייתון קוד אמיתי ולוגי. למשל:

```
credits = 100  # כמות קרדיטים מוגדרת מראש
if credits >= cost:  # השוואה בסיסית לבדיקה של האם יש לי מספיק קרדיטים?
```

- שימו לב היכן צריך להמיר קלט (`int(input_value)` וכדומה).

3. **הרצת הקוד** - לאחר שמילאתם את כל המקומות החסרים, שמרו את הקובץ (למשל בשם `space_travel.py`) והריצו אותו בעזרת הפקודה `python space_travel.py` (או מתוכנה כמו VS Code/PyCharm).
- בדקו האם ההרצה זורמת כמצופה, והאם אתם מקבלים פלט הגיוני לפי תרחיש הסיפור (בחירת דלק, פתרון חידות, בחירת שער A או B).

4. **בדיקות** - נסו להזין ערכים שונים: לדוגמה, מה קורה אם אתם בוחרים לקנות הרבה דלק או מעט מדי, מה קורה אם אתם עונים תשובות שגויות לחידות וכדומה.
- ראו אם התוכנה פועלת כמו שתוכננה (למשל, נפסלת אם נכשלתם בשתי החידות).

5. **שאלות / סיוע** - אם לא ברור לכם איך להשתמש ב-פונקציות או ב-If, חזרו לדוגמאות הבסיסיות מהספר.
- זכרו להשתמש בפקודות הדפסה (`print`) על מנת לעקוב אחר משתנים באמצע התוכנית (לצורכי דיבוג).

להלן הסבר כללי על מה עושה הקוד הסופי (ה"משחק" המורחב) אשר כולל שני פאזלים (חידות) ועלילה מתפצלת:

1. **משתנה גלובלי (Global Variable)**
   - בתחילת הקוד מוגדר משתנה גלובלי `credits` (כמות קרדיטים שיש לשחקן). פונקציות שונות מעדכנות ערך זה כאשר השחקן קונה דברים.

2. **פונקציות עזר (Utility Functions)**
   - **`purchase_item(cost)`**: מקבלת מספר שלם ובודקת אם יש מספיק קרדיטים לרכוש פריט (או דלק). אם כן, מורידה את המחיר מהקרדיטים ומחזירה `True`. אחרת מחזירה `False`.
   - **`check_fuel_choice(fuel_amount)`**: מקבלת כמות דלק שהשחקן רוצה לקנות, בודקת אם אפשר לבצע את הפעולה ומחזירה הודעה האם הפעולה `safe` (בטוחה),

risky (מסוכנת אם מעל 100 יחידות דלק, יכול להיות ולא תוכלו להמשיך), או impossible (מקבלים קרידיטים שליליים ולכן לא יהיה ניתן להמשיך).
- **solve_riddle()**: החידה הראשונה, מבוססת על אופרטור ה־Modulo. השחקן צריך לחשב את 2 * (6 % 57). אם פותר נכון מחזירה True, ואם לא — False.
- **solve_logic_puzzle()**: החידה השנייה, מבוססת על אופרטורים לוגיים (and, or, not). השחקן צריך להחליט אם הביטוי (True and not False) or (False and not True) יוצא True או False.

3. **מהלך המשחק (בפונקציה main())**
- **הודעת תחילת המשחק**: שואל את שם המשתמש ומדפיס כמה קרדיטים יש בהתחלה.
- **רכישת פריטים**: השחקן בוחר לקנות דלק או אוכל (או לא לקנות כלל).
  - אם השחקן בוחר דלק, בודק בכמה יחידות מדובר, בודק אם זה מעל 100 או פחות מ־0 (באמצעות check_fuel_choice), ומחשב את עלות הדלק (למשל 2 קרדיטים ליחידה). קורא ל־purchase_item כדי להוריד את הקרדיטים ובודקת אם זה אפשרי.
  - אם השחקן בוחר אוכל, יש לכך עלות קבועה (למשל 30 קרדיטים). השחקן מצליח לקנות אך ורק אם יש מספיק קרדיטים.
- **פאזל ראשון (חידת מודולו)**: אם השחקן פותר נכון את החידה (פונקציה solve_riddle), ממשיך במסלול הראשי. אם לא — ננעל הדלת והמשתמש נאלץ לברוח ל"מסלול צדדי" (חדר סודי).
- **פאזל שני (חידת לוגיקה)**: אם השחקן נכשל בחידה הראשונה, יש הזדמנות שנייה לפתור חידה מבוססת לוגיקה (solve_logic_puzzle). כישלון בשני הפאזלים רצוף מביא ל־Game Over. הצלחה בכל אחד מהפאזלים מאפשרת להמשיך הלאה.
- **נקודת הסיום (Gate A או Gate B)**:
  - Gate A עולה 20 קרדיטים לפתיחה. אם יש מספיק קרדיטים, אפשר לעבור בהצלחה.
  - Gate B עולה 50 קרדיטים אך מקנה בונוס (למשל דלק נוסף). אם אין מספיק קרדיטים, נתקעים.
- **סיכום קרדיטים**: בסוף המשחק מדפיסים כמה קרדיטים נשארו ומעירים הערה לגבי ההתנהלות של השחקן (אם נשארו הרבה, מעט או בכלל לא).

4.מדוע קיימת הבדיקה

```
if __name__ == "__main__":
  main()
```

- כך פייתון יודע להריץ את main() **רק** אם הקובץ הזה הוא הקובץ הראשי שמפעילים
- אם מייבאים את הקובץ לסקריפט אחר (למשל import game), ולא מעוניינים שהמשחק יתחיל אוטומטית.

סיכום

הקוד הסופי הוא משחק טקסטואלי קצר שמעביר אתכם דרך **תחנות קנייה**, **חידות מתמטיות/לוגיות**, ו**החלטות סופיות** לגבי שערים, תוך כדי ניהול של משתנה קרדיטים גלובלי. כל אלו נועדו לבחון אתכם על מושגי הבסיס בפייתון (משתנים, הצבות, המרות טיפוסים, תנאים, אופרטורים לוגיים, פונקציות וטווחי משתנים).

כיצד עובד הקוד?

1. **משתנה גלובלי: `credits` מייצג כמה קרדיטים יש לשחקן לאורך משחק.**

2. **פונקציות העזר:**
   - **`purchase_item(cost)`: מורידה את הסכום מהמשתנה קרדיט הגלובלי, או מחזירה False אם אין מספיק.**
   - **`solve_riddle()`: בודקת אם המשתמש עונה נכון לחידת המודולו.**
   - **`solve_logic_puzzle()`: בודקת אם המשתמש עונה נכון לחידת האופרטורים הלוגיים.**
   - **`check_fuel_choice(fuel_amount)`: בודקת האם יש כמות דלק מספקת ומחזירה הודעה האם היא בטוחה, מסוכנת או בלתי אפשרית.**

3. **תסריט המשחק:**
   - **קנייה בתחנה (דלק או אוכל).**
   - **חידה ראשונה (מודולו). כישלון מפעיל דרך עוקפת (חידה שנייה, לוגיקה).**
   - **בחירת שער (A או B) בסוף, בתנאי שיש מספיק קרדיטים.**
   - **סיכום קרדיטים והערה על התנהלות השחקן.**

רקע כללי

תזכורת על הנושאים שצריך לדעת בפייתון כדי לעבור את המבדק

1. **משתנים (Variables) והצבה (Assignment)**
- משתנה מייצג מקום בזיכרון המחשב לאחסון ערכים.
- הצבת ערך נעשית באופרציית `=`, לדוגמה: `x = 5`.

2. **טיפוסי נתונים (Data Types)**
- לדוגמה: מספר שלם `**int**`, מספר ממשי `**float**`, מחרוזת `**str**`, ועוד.
- שימו לב שיש לבצע המרה מתאימה כשאתם משתמשים בפונקציות כמו `input()` (שהערך המוחזר מהן הוא תמיד מחרוזת) ורוצים להפוך אותו למספר באמצעות `int()` או `float()`.

3. **פלט (print) וקלט (input)**
- `print()` מציג מידע על המסך.
- `input()` קורא קלט מהמשתמש ומחזיר מחרוזת.

4. **אופרטורים מתמטיים (Math Operators)**
- `+` (חיבור), `-` (חיסור), `*` (כפל), `/` (חילוק), `//` (חילוק שלם), `%` (מודולו), `**` (חזקה).
- במבחן נעשה שימוש מיוחד ב-`%` (מודולו) לבדיקת שארית.

5. **אופרטורים להשוואה (Comparison) ולאופרטורים לוגיים**
- השוואה: `==`, `!=`, `<`, `>`, `=<`, `=>`.
- אופרטורים לוגיים: `not`, `or`, `and`.
- לדוגמה: `if x > 5 and y < 10: ...`

6. **היקף משתנים (Scope) ומשתנים גלובליים (Global Variables)**
- משתנה גלובלי מוגדר מחוץ לפונקציות, אך כדי לשנות אותו בתוך פונקציה יש להצהיר `global var_name`.
- דוגמה:

```
credits = 100

def purchase_item(cost):
    global credits
    # ...
```

7. **תנאים (if/elif/else)**
- מבנה בסיסי:

```
if תנאי1:
    ...
elif תנאי2:
    ...
else:
    ...
```

נקודות עיקריות (דברים שצריך לזכור ולדעת)

1. **עבודה עם קלט (input) ויציאה מהתוכנית**
- שימו לב שכל קריאה ל-`input()` מחזירה טקסט (מחרוזת). כדי לבצע פעולות חשבון יש להמיר למספר באמצעות `int()` או `float()`.
- לדוגמה:

```
amount_str = input("Enter amount: ")
amount = int(amount_str)
```

- זכרו שאם תנסו לעשות `int("hello")`, תקבלו שגיאה. לכן לעיתים כדאי לבדוק שהקלט אכן מספר לפני ההמרה (אך זה כבר מעבר לידע בסיסי ונלמד את זה בהמשך).

2. **מודולו (Modulo) בשאלות מתמטיות**
- `x % y` מחזיר את השארית של `x` בחלוקה ל-`y`.
- אם למשל שאלו אתכם: `57 % 6 = ?`, התוצאה היא שארית החלוקה של 57 ב-6, כלומר 3.

3. **שימוש באופרטורים לוגיים**
- `True and False) -> False)`,
- `True or False) -> True)`,
- `not True -> False`.

4. **גלובל/לוקל**
- אם רוצים לשנות משתנה (כמו `credits`) בתוך פונקציה, צריך להצהיר עליו:

```
global credits
```

- בלי ההצהרה, כל שינוי על משתנה בשם זהה בתוך פונקציה ייחשב כיצירה של משתנה חדש ולא ישפיע על הערך הגלובלי מחוץ לפונקציה.

5. **מבנה התוכנית**
- הקוד בנוי כך שכל פעולה גדולה (כמו קניית פריט, פתרון חידה) תהיה בפונקציה נפרדת עם שם ברור.
- לקרוא לפונקציות אלו מתוך `main()`, שהוא בעצם הבקרה המרכזית של התוכנית, משם אני מנהל את כל הקודים שלי במקום אחד.

בהצלחה!