תרגיל 7:

High Order Functions – Built-ins and Custom HOFs

תאריך פרסום:	23/12/2021
תאריך הגשה:	2/1/2022 בשעה 23:59
מתרגל אחראי:	ישעיה צברי
משקל תרגיל:	2 נקודות
מטרות העבודה:	שימוש ומימוש פונקציות מסדר גבוה.

הנחיות כלליות

קראו בעיון את ההנחיות והעבודה. לא יתקבלו ערעורים על טעויות שחרגו מההנחיות.

1. העבודה תבוצע ותוגש ביחידים.
2. מומלץ לקרוא את העבודה כולה לפני שאתם ניגשים לפתרון.
3. עליכם להוריד את קובץ הקוד שמסופק עם התרגיל ולהשלים את הקוד החסר. יש לממש את הפתרון לתרגיל אך ורק באזורים שהוגדרו לשם כך בקובץ!
4. <u>כתיבת קוד קריא</u>:
   - 4.1. השתמשו בשמות משתנים משמעותיים. שימוש בשמות לא משמעותיים עשוי לגרור לפגיעה בציון.
   - 4.2. כתבו תיעוד (הערות) שמסביר את הקוד שלכם. יש לכתוב תיעוד docstring בכל פונקציה כפי שנלמד בכיתה.
   - 4.3. אין לכתוב הערות בעברית! עבודה שתכיל טקסט בשפה שאינה אנגלית (או פייתון) תקבל <u>ציון אפס</u> ללא אפשרות ערעור.
5. אין להשתמש בחבילות או במודולים חיצוניים <u>מלבד</u> מה שהוגדר בתרגיל! אם יש ספק ניתן לשאול בפורום המתאים (ראו סעיף 10).
6. יש לכתוב קוד אך ורק באזורים שהוגדרו לשם כך!
7. <u>הנחות על הקלט:</u>
   - 7.1. בכל שאלה יוגדר מה הקלט שהקוד מקבל וניתן להניח כי הקלט שנבדוק מקיים את התנאים הללו. אין להניח הנחות נוספות על הקלט מעבר למה שהוגדר.
   - 7.2. בכל שאלה סיפקנו עבורכם דוגמאות לקלט והפלט הרצוי עבורו. עליכם לערוך בדיקות נוספות לקוד שמימשתם ולא להסתמך על דוגמאות אלו בלבד.
8. <u>בדיקת העבודה:</u>
   - 8.1. העבודה תיבדק באופן אוטומטי ולכן על הפלטים להיות <u>זהים</u> לפלטים שמוגדר בתרגיל.
   - 8.2. טרם ההגשה יש לעבור על המסמך assignments checklist שנמצא במודל.
   - 8.3. מערכת הבדיקות קוראת לפונקציות שהוגדרו בתרגיל בצורה אוטומטית. אין לשנות את חתימות הפונקציות. חריגה מההנחיות תגרור ציון אפס.
9. <u>העתקות</u>:
   - 9.1. <u>אל תעתיקו!</u>
   - 9.2. העתקת קוד (משנים קודמות, מחברים או מהאינטרנט) אסורה בהחלט. בפרט אין להעביר קוד בין סטודנטים. צוות הקורס ישתמש בכלים אוטומטיים וידניים כדי לזהות העתקות. תלמיד שייתפס בהעתקה יועמד בפני ועדת משמעת (העונש המינימלי לפי תקנון האוניברסיטה הוא כישלון בקורס).
   - 9.3. אנא קראו בעיון את המסמך שהכנו בנושא: https://moodle2.bgu.ac.il/moodle/mod/resource/view.php?id=1922556

10. <u>שאלות על העבודה</u>:
    - 10.1. שאלות בנוגע לעבודה ישאלו בפורום שאלות לתרגיל במודל או בשעות הקבלה של המתרגל/ת האחראית <u>בלבד</u> .
    - 10.2. אין לפנות במייל לבודקת התרגילים או למתרגלים אחרים בנוגע לעבודות הגשה. מיילים בנושאים אלו לא יקבלו מענה.
    - 10.3. לפני ששואלים שאלה בפורום יש לוודא שהשאלה לא נשאלה קודם!
    - 10.4. אנו מעודדים סטודנטים לענות על שאלות של סטודנטים אחרים. המתרגל/ת האחראית תאשר שתשובה כזו נכונה.
11. <u>הגשת העבודה</u>:
    - 11.1. עליכם להוריד את הקבצים מתיקיית "תרגיל בית 7" מהמודל. התיקייה תכיל תיקייה נוספת ובה קבצי העבודה וקובץ הוראות. עליכם למלא את הפתרון במקום המתאים ובהתאם להוראות התרגיל.
    - 11.2. שימו לב: בנוסף לקבצי העבודה מצורף קובץ בשם get_id.py. עליכם למלא במקום המתאים בקובץ את תעודת הזהות שלכם. הגשה שלא תכיל את הקובץ הנ"ל עם תעודת הזהות הנכונה לא תיבדק ותקבל ציון אפס!
    - 11.3. את העבודה יש להגיש באמצעות תיבת ההגשה הייעודית במודל.
    - 11.4. פורמט ההגשה הוא להלן:
        - 11.4.1. את תוכן התיקייה בשם hw7 שהורדתם מתיקיית העבודה במודל, ובה נמצאים קבצי הקוד שלכם, יש לכווץ לפורמט zip (קבצים אחרים, כגון קבצי rar, יקבלו ציון אפס).
        - 11.4.2. השם של התיקייה המכווצת יהיה תעודת הזהות שלכם.
        - 11.4.3. העלו את התיקייה המכווצת לתיבת ההגשה של העבודה.

בהצלחה!

שאלה 1 – Built-in High Order Functions

בוב ואליס הם סטודנטים ושותפים הגרים בדירת גן בשכונת ד'. ע"מ לנהל את ההוצאות החודשיות שלהם בצורה מיטבית, החליטו בוב ואליס לנצל את היותה של אליס סטודנטית להנדסת מערכות מידע ולנהל מעקב אחר הוצאות הקניות החודשיות שלהם בצורה ממחושבת. לאחר כל רכישה, מזינה אליס את העלות לרשימה המרכזת את ההוצאות החודשיות. את הרשימות אליס שומרת במילון, כאשר המפתח הוא שם החודש באנגלית והערך המתאים הוא רשימת העלויות החודשית.
דוגמה למילון:

```
>>> monthly_shopping_costs = {'January': [150, 22],
'Februray': [45, 200, 14]}
```

נשים לב שהרשימות שומרות על סדר הכנסה. בדוגמא תוכלו לראות שבחודש ינואר הוזנה קניה ראשונה בסכום של 150 ₪ ואז קניה נוספת בסכום של 22 ₪. בסעיפים הבאים תעזרו לאליס לממש פונקציות שיאפשרו מעקב נוח יותר אחר הוצאותיהם של השותפים.

שימו לב:

1. מטרת התרגיל היא תרגול שימוש בפונקציות מסדר גבוה, ולכן **אין להשתמש בלולאות** (**for** או **while**) **בשאלה 1** ואין לממש פונקציות עזר נוספות. תיעוד לפונקציות Python built-in High Order Functions תוכלו למצוא פה: filter, map, reduce, sorted.
2. בכל הסעיפים בשאלה 1 עליכם לממש **פתרון בשורה אחת בלבד** (חתימת הפונקציה אינה נחשבת כשורה).
3. בכלל הסעיפים של שאלה 1 **אין להשתמש** בפתרונות של סעיפים קודמים.

סעיף א

אליס ובוב עשויים לחרוג מעבר לתקציב. כדי לאתר את החודשים בהם עמדו השותפים במסגרת התקציב, עליכם לממש את הפונקציה:

```
get_months_under_limit(monthly_shopping_costs, monthly_limit)
```

המקבלת מילון המכיל את ההוצאות החודשיות (monthly_shopping_costs) ואת מסגרת התקציב החודשי (monthly_limit) ומחזירה רשימה של שמות החודשים שעומדים במסגרת התקציב.

קלט:

- **monthly_shopping_costs [Dict[str, List[int]]]** – מילון, כאשר המפתחות הם מחרוזות של שמות החודשים, והערכים הם רשימת ההוצאות (כל הוצאה נשמרת כ-int) בהתאם לחודש המתאים.
- **monthly_limit [int]** – מספר שלם לא שלילי, תקרת התקציב החודשית הקבועה.

פלט:

- **list** - רשימה של שמות החודשים (לפי סדר הופעתם במילון) בהם סך ההוצאות הוא **קטן או שווה** ממסגרת התקציב החודשי.

הנחות קלט:

- ערכי הפרמטרים monthly_shopping_costs, monthly_limit אינם None.

- המפתחות במילון monthly_shopping_costs הם מחרוזות של שמות של חודשים בלוח השנה הלועזי בלבד.
- המפתחות והערכים במילון monthly_shopping_costs אינם None.
- כל ערכי המילון monthly_shopping_costs הינם רשימות.
- ערך הפרמטר monthly_limit הוא מספר שלם לא שלילי.

דוגמאות:

1. החודשים January ו-March עומדים בתקציב בגובה 500 ₪ ולכן יוחזרו:

```
>> monthly_shopping_costs = {'January': [199, 100, 200],
'February': [200, 100, 201], 'March': [150, 87, 35, 100]}
>> get_months_under_limit(monthly_shopping_costs, 500)
['January', 'March']
```

סעיף ב

חבריהם של אליס ובוב נתנו להם טיפ לצריכה נבונה יותר (אין להתייחס לאמור כעצה כלכלית): לבצע מס' רכישות קטן בחודש (לדוגמה, **עד** 5 רכישות בחודש).
ממשו את הפונקציה:

`sum_under_limit_number_of_purchases(monthly_shopping_costs, max_number_of_purchases)`

המקבלת מילון המכיל את ההוצאות החודשיות (monthly_shopping_costs) ומספר רכישות מקסימלי לחודש (max_number_of_purchases) ומחזירה את סכום הרכישות אם כמות הרכישות עומדת באילוץ של מספר רכישות מקסימלית או אינסוף שלילי (-math.inf), אם לא.

קלט:

- **`monthly_shopping_costs`** **[Dict[str, List[int]]]**– מילון, כאשר המפתחות הם מחרוזות של שמות החודשים, והערכים הם רשימת ההוצאות (כל הוצאה נשמרת כ-int) בהתאם לחודש המתאים.
- **`max_number_of_purchases`** **[int]**– מספר שלם לא שלילי, כמות הרכישות המקסימלית בחודש.

פלט:

- **`int/float`** - אם כמות הרכישות בכל חודש (בנפרד) **קטנה או שווה** ל-max_number_of_purchases הפונקציה מחזירה את סכום העלויות של כל הרכישות בכל החודשים יחדיו, אחרת, הפונקציה מחזירה את הערך אינסוף שלילי (float('-inf')).

הנחות קלט:

- ערכי הפרמטרים monthly_shopping_costs, max_number_of_purchases אינם None.
- monthly_shopping_costs הוא מטיפוס מילון.
- כל ערכיו ומפתחותיו של monthly_shopping_costs אינם None.

- המפתחות במילון monthly_shopping_costs הם מחרוזות של שמות של חודשים בלוח השנה הלועזי בלבד.
- כל ערכי המילון monthly_shopping_costs הינם רשימות שלא מכילות None.
- ערך הפרמטר max_number_of_purchases הוא מספר שלם לא שלילי.

דוגמאות:

1. כל החודשים עומדים בתקרת כמות הרכישות החודשית (5) ולכן הפונקציה תחזיר את סכום כל הרכישות:

```
>> monthly_shopping_costs = {'January': [199, 100, 200],
'February': [200, 100, 201], 'March': [150, 87, 35, 100]}
>> print(sum_under_limit_number_of_purchases
(monthly_shopping, 5))
'1372'
```

2. פברואר אינו עומד בתקרת כמות הרכישות החודשית (2) ולכן הפונקציה תחזיר אינסוף שלילי:

```
>>>> monthly_shopping_costs = {'January': [1, 2],
'February': [2, 3, 2}
>> print(sum_under_limit_number_of_purchases
(monthly_shopping, 2) == float('-inf'))
'True'
```

סעיף ג

חבריהם של אליס ובוב נתנו להם עוד טיפ לצריכה נבונה יותר (שגם הוא ככל הנראה לא מוצלח): מומלץ לבצע רכישות בעלויות הולכות וגדלות (או שוות) במהלך כל חודש. כלומר מומלץ שסדרת הרכישות תהיה מונוטונית עולה. דוגמה לעמידה בהמלצה בחודש מסוים: הרכישה הראשונה היתה בסכום של 15 ₪, הרכישה השנייה 20 ₪, השלישית ב-20 ₪, והרביעית בסך 30 ₪.

ממשו את הפונקציה:

verify_monthly_monotonic_growing_expenses(monthly_shopping_costs)

המקבלת מילון המכיל את ההוצאות החודשיות (monthly_shopping_costs) ומחזירה משתנה בוליאני (bool) האם **כל** סדרת רכישות חודשית היא מונוטונית עולה.

קלט:

- **monthly_shopping_costs [Dict[str, List[int]]]**– מילון, כאשר המפתחות הם מחרוזות של שמות החודשים, והערכים הם רשימת ההוצאות (כל הוצאה נשמרת כ-int) בהתאם לחודש המתאים.

פלט:

- **bool** - הפונקציה מחזירה משתנה בוליאני בערך True אם עבור **כל** חודש מתקיים שסדרת הרכישות שלו היא סדרה מונוטונית עולה, False אחרת.

הנחות קלט:

- ערך הפרמטר monthly_shopping_costs הוא מטיפוס מילון וכל ערכיו ומפתחותיו אינם None.
- כל ערכי המילון monthly_shopping_costs הינם רשימות שלא מכילות None.

דוגמאות:

1. כל רכישה עוקבת בחודשים ינואר ופברואר אכן גדולה יותר או שווה לקודמתה:

```
>> monthly_shopping_costs = {'January': [1, 2, 2],
'February': [4, 5, 6]}
>>
print(verify_monthly_monotonic_growing_expenses(monthly_s
hopping_costs))
'True'
```

2. מרץ אינו עומד בתקרת כמות הרכישות החודשית (3) ולכן הפונקציה תחזיר את הערך הבוליאני False:

```
>> monthly_shopping_costs = {'January': [0, 2, 2], 'February':
[4, 5, 4]}
>>
print(verify_monthly_monotonic_growing_expenses(monthly_s
hopping_costs))
'False'
```

סעיף ד

אליס עובדת בהייטק ומרוויחה משכורת חודשית נאה, לכן היא משלמת את רוב ההוצאות השוטפות של הדירה. בוב, שאינו עובד אלא בעבודות מזדמנות, מגיע להכנסה שבועית בסיסית.
כאשר אליס ובוב גילו כי לא שילמו חודשים רבים חלק מחשבונות הדירה על שירותים (מים, חשמל וכו'), החליטו שבוב ישלם את כל החשבונות של השירותים אשר סכומם נמוך או שווה להכנסה השבועית שלו. את כל שאר החשבונות, אליס תשלם (ראו דוגמאות בהמשך). עליכם לעזור להם לחלק את תשלום החשבונות.

ממשו את הפונקציה:

divide_monthly_bills(apartment_bills, bob_weekly_income)

המקבלת מילון המכיל את חשבונות הדירה (apartment_bills) ואת ההכנסה השבועית של בוב (bob_weekly_income) ומחזירה מחרוזת המפרטת איזה חשבונות על אליס ועל בוב לשלם.

קלט:

- **apartment_bills [Dict[str, List[int]]]**– מילון, כאשר המפתחות הם מחרוזות של שמות השירותים ('Water', 'Electricity'), והערכים הם רשימת סכומי החשבונות של השירות שעל אליס ובוב לשלם.
- **bob_weekly_income [int]-** מספר שלם לא שלילי, הסכום אותו בוב מרוויח כל שבוע מעבודות מזדמנות.

פלט:

- **str** - הפונקציה מחזירה מחרוזת במבנה המתואר (שימו לב לרווחים המסומן בקו תחתון אדום):
"<NAME>: <SERVICENAME>, <NAME>: <SERVICENAME>,.....<NAME>: <SERVICENAME>"
לדוגמה, הפונקציה תחזיר את המחרוזת "Alice: Electricity, Bob: Water, Alice: Rent" בהינתן הקלט:

```
apartment_bills = {'Electricity': [2, 7, 1], 'Water': [1,1,3], 'Rent': [100, 2, 3]}
bob_weekly_income = 5
```

הנחות קלט:

- ערך הפרמטר apartment_bills הוא מסוג מילון וכל ערכיו ומפתחותיו אינם None.
- כל ערכי המילון apartment_bills הינם רשימות שלא מכילות None וכלל האיברים ברשימות הן מטיפוס מספרי (int/float) בעל ערך לא שלילי.
- כל מפתחות המילון apartment_bills הם מטיפוס מחרוזת וערכם מייצג שירות של דירה (כמו בדוגמאות).
- ערך הפרמטר bob_weekly_income אינו None והוא מטיפוס int עם ערך לא שלילי.

דוגמאות:

1.

```
>>apartment_bills = {"Electricity": [2], "Water": [1, 1,
3], "Rent": [100, 2, 3]}
>>bob_weekly_income = 5
>>divide_monthly_bills(apartment_bills,
bob_weekly_income)
"Bob: Electricity, Bob: Water, Alice: Rent"
```

2.

```
>> {"Electricity": [0.1], "Water": [0.5], "Rent": [1]}
>> bob_weekly_income = 1
>> divide_monthly_bills(apartment_bills,
bob_weekly_income)
"Bob: Electricity, Bob: Water, Bob: Rent"
```

שאלה 3 -

Code line #1: Numeric Value	***0***	0	1
Code line #2: Command	***1***	1	1
Code line #3: Numeric Value	***0***	0	1

בשפת מכונה (קוד מחשב בינארי) כל שורת קוד מקודדת ע"י רצף של ביטים ומייצגת פקודה או ערך מספרי. ע"מ להבדיל בין פקודה לערך מספרי, התווים הראשונים בכל שורת קוד מוקדשים ל"דגל" (בדוגמה מסומן ב-bold) שערכו מציין האם שורת הקוד מייצגת *פקודה* (1) או *ערך מספרי* (0), שאר התווים מייצגים את תוכן הפקודה (לדוגמה חיבור) או את הערך המספרי בהתאמה. פקודה היא פעולת חישוב, כפל או חיבור, **הנתונה כפונקציה** המקבלת שני ערכים מספריים ומחזירה את תוצאת החישוב (ערך מספרי). ערך מספרי מיוצג כמספר בינארי (באורך 2 ביטים). בשאלה זו, אורך שורת קוד הוא שלושה תווים (כולל הדגל, ראו בדוגמא).

משימתכם היא לממש גרסה מופשטת של "מפרש גנרי" (generic interpreter), המקבל רשימה של שורות קוד ומבצע פעולות כפל או חיבור בין שני ערכים מספריים בהתאם לשורות הקוד שניתנו.

נדגיש, בשאלה 3 מותר לממש פונקציות עזר.

סעיף א

ממשו את הפונקציה:

```
operations_parser_function(code)
```

אשר מקבלת מחרוזת באורך שני תווים (code) המייצגת פקודה (ללא הדגל), ומחזירה פקודה בהתאם לקלט. שימו לב שמחרוזת הקלט פה מהווה **חלק** משורת קוד משום שהיא חסרה את הדגל.

<u>קלט:</u>

- **`[str]code`** – מחרוזת.

<u>פלט:</u>

- **`function`** - אם סכום הספרות ב-code הוא 1 (01 או 10) אז הפונקציה תחזיר פקודת חיבור, אחרת הפונקציה תחזיר פקודת כפל.

<u>הנחות קלט:</u>

- מחרוזת הקלט code מכילה את התווים "0" ו/או "1" בלבד והינה באורך 2.

<u>דוגמאות:</u>

.1

```
>> operation = operations_parser_function('01')
>> print(operation(1, 1))
"2"
```

.2

```
>> operation = operations_parser_function('11')
>> print(operation(1, 1))
"1"
```

סעיף ב

נגדיר *פעולה* - שלוש שורות קוד המייצגות: ערך מספרי, אופרציה, ערך מספרי. הדוגמא בראשית השאלה מייצגת את הפעולה של פקודת הכפל על הערכים המספריים 1, ו-1 והתוצאה של הפעולה תהיה 1 = 1*1.

ממשו את הפונקציה:

`binary_code_compiler(lines_of_code_triplets, value_parser, operations_parser)`

אשר מקבלת רשימה של tuples (lines_of_code_triplets), כאשר כל tuple מקודד פעולה, ומחזירה את סכום תוצאות **פעולות החישוב**. כאשר אחד (או יותר) מה-tuples בקלט אינו תואם למבנה שהוגדר עבור פעולה או שהפקודה הנתונה כפונקציה אינה תקינה, הפונקציה תחזיר את מספר הפעולה **הראשונה** בה ארעה התקלה **כמחרוזת**.

<u>קלט:</u>

- **`lines_of_code_triplets [List[Tuple[str, str, str]]]`** – רשימה של tuples, כל tuple היא שלשה של מחרוזות המייצגות שלוש שורות קוד מלאות (**עם הדגל כתו באינדקס 0**).
- **`value_parser [function]`** – פונקציה המקבלת מחרוזת באורך שני תווים כארגומנט בודד, ומחזירה ערך מספרי (float), במקרה של שגיאה כלשהי בהפעלתה, הפונקציה מעלה שגיאה מסוג ValueError.
- **`operations_parser [function]`** – פונקציה המקבלת מחרוזת באורך שני תווים כארגומנט בודד, ומחזירה פונקציה המקבלת שני ארגומנטים ומחזירה ערך מספרי (float), במקרה של שגיאה כלשהי בהפעלתה, הפונקציה מעלה שגיאה מסוג TypeError.

<u>פלט:</u>

- **Union[float,int,str]**- הפונקציה מחזירה את תוצאת הפעולה כפי שהוגדרה ע"י הפעולות או את מספר הפעולה כמחרוזת (מספור המתחיל ב0) אשר בה התרחשה שגיאה.

הנחות קלט:

- ערכי הפרמטרים lines_of_code_triplets, value_parser, operations_parser אינם None.
- ערך כל tuple ברשימה lines_of_code_triplets אינו None.
- אורך כל מחרוזת ("שורת קוד") בכל tuple היא באורך שלושה תווים בדיוק.

שימו לב:

1. על הפונקציה "להגן" מפני שגיאות של הפונקציות (שניתנו כארגומנטים) ולמנוע את קריסת התוכנית, נדגיש שהשגיאות שעליכן "להגן" מפניהם הן מסוג **ValueError** **או** **TypeError** **בלבד**.
2. שורות הקוד אותן הפונקציה מקבלת מכילות גם את דגל שורת הקוד (המציין האם מדובר בפקודה או בערך מספרי).
3. עליכם לתרגם כל שורת קוד בעזרת הפונקציה המתאימה לה לפני ביצוע הפעולה כולה (value_parser לערך מספרי, operations_parser לאופרציה).

דוגמאות:

1. בדוגמה זו, הפעולה הראשונה מייצגת את הקוד 1, addition, 1 (1+1) ולכן יבוצע חיבור בין 1 ל1. הפעולה השנייה מייצגת את הקוד 1, addition, 2 (1+2) ולכן יבוצע חיבור בין 1 ל2. סכום הפעולות הינו 5 ולכן הפונקציה תחזיר את הערך המספרי 5 (float/integer).

```
>> lines_of_code_triplets = [
('001', '101', '001'),
('001', '101', '011')]
>> ans=binary_code_compiler(lines_of_code_triplets,
value_parser1, operations_parser_function)
>> print(ans==5)
"True"
```

```
def value_parser1(code:
str):
    ans = 0
    for char in code:
        ans += int(char)
    return ans
```

2. בדוגמה זו, הפעולה הראשונה מייצגת את הקוד 1, multiplication, 1 (1*1) ולכן תבוצע פעולת כפל בין 1 ל1. הפעולה השנייה מייצגת את הקוד 1, multiplication, 2 (2*1) ולכן יבוצע כפל בין 1 ל2.

סכום הפעולות הינו 3 ולכן הפונקציה תחזיר את הערך המספרי 3 (float/integer).

```
>> lines_of_code_triplets = [
('001', '111', '001'),
('001', '100', '011')]
>> ans=binary_code_compiler(lines_of_code_triplets,
value_parser1, operations_parser_function)
>> print(ans==3)
"True"
```

3. בדוגמה זו, בפעולה השנייה, שורת הקוד השלישית היא של פקודה ולא ערך מספרי (מתחילה ב'1' במקום ב'0'), לכן הפונקציה תחזיר מחרוזת '1' כאינדקס השלשה בה ארעה השגיאה.

```
>> lines_of_code_triplets = [
('001', '101', '001'),
('001', '101', '111')]
>> ans=binary_code_compiler(lines_of_code_triplets,
value_parser1, operations_parser_function)
>> print(ans==’1’)
"True"
```

4. בדוגמה זו, מכיוון שהפונקציה value_parser תחזיר שגיאה תמיד, הפונקציה תחזיר מחרוזת '0' כאינדקס הפעולה בה ארעה השגיאה.

```
>> lines_of_code_triplets = [
('001', '101', '001'),
('001', '101', '111')]
>> ans=binary_code_compiler(lines_of_code_triplets,
Lambda x: 1/0, operations_parser_function)
>> print(ans==’0’)
"True"
```

בהצלחה!