

GeoAD

Advanced Geometric Descriptor

By Alon Dayan



תוכן עניינים

2.....	❖ תוכן עניינים
4.....	❖ מבוא
5.....	❖ התוצר הסופי ומדריך למשתמש
5.....	◆ תהליך השימוש בתוצר
6.....	◆ פעולות שימושיות למשתמש במחלקה Helper
7.....	❖ מבוא תיאורטי
7.....	◆ מבוא לשפת פירטון
8.....	◆ מבוא לפתרון בעיות גיאומטריה בשיטת "טענה-nimוק"
11.....	❖ החלק המעשִׁי
11.....	◆ דרישות המודל
11.....	◆ המודל הישן
12.....	• בעיות במודל הישן
12.....	- כיוון ישרים
14.....	- זווית נשאה
15.....	- שרטוטים בלתי אפשריים
16.....	◆ המודל החדש והסופי
17.....	• מבנה המודל
18.....	- המחלקה Point
20.....	- המחלקה AbsSegment
21.....	- המחלקה AbsAngle
22.....	- המחלקה Expression
24.....	• המחלקות Length-1 Degree
24.....	- המחלקה RealAngle
25.....	- המחלקה RealSegment
25.....	- המחלקה Handler
25.....	• פעולות בסיסיות על AbsAngle
29.....	• פעולות בסיסיות על AbsSegment
29.....	• המרת מאובייקטי Abs לאובייקטי Real
29.....	- המחלקה Convertor
30.....	- המשך המחלקה Handler
30.....	- המחלקה Polygon

30	המחלקות Quadrilateral ו Triangle	.
31	המשר המחלקה Handler	-
31	פעולות מצולעים	.
37	פעולת הוכחה	.
37	פעולות השוויון / מקבילים	.
37	פעולות המשפטים	.
38	המחלקה Helper	-
38	פעולות יצרה או חזרה	.
39	פעולות אתחול והוכחה (חישוב)	.
39	פעולות Set/Get/Equal	.
39	פעולות קטעים	.
40	המחלקה ProblemCollection	-
41	נספחים	.
41	הסבר על קבועי ggb	-
44	הסביר על UnionFind	-
45	הוכחות למשפטים באמצעות האקסיומות	-
45	משפט 1: סכום הזווית על ישר הוא 180°	.
46	משפט 2: זוג זוויות קודקודיות הן שוות	.
48	משפט 3: סכום הזווית סביב נקודה הוא 360°	.
49	משפט 4: זוג זווית מתאימות וחד צדיות (בין שני ישרים מקבילים וחותך)	.
52	משפט 5: משפט הפוך לזוגיות בין ישרים מקבילים	.
57	משפט 6: סכום הזווית במשולש הוא 180°	.
60	משפט 7: סכום הזווית במרובע הוא 360°	.
62	משפט 8: זווית חיצונית למשולש	.
65	הצעות לשיפור	.
66	הקדם הסופי	❖

מבוא

מאז שגיליתי את עולם התכונות ומדעי המחשב, העסיקה אותי השאלה "כיצד מחשבים יכולים להחליף את בני האדם?". בהקשר של כמעט כל מילה או תחביב שאני עוסק בהם, אני חשב לעצמי כיצד אוכל ליצור תוכנה שתבצע מה שאני עושה ברגע זה.

ובכן, כאשר למדתי לבגרות למתמטיקה ונטקلت בבעיות גיאומטריה, עלתה בראשי המחשבה הזו. התחלתי לחשב על בניה תוכנה שתפתרו בעיות גיאומטריה בשיטת "טענה-ניסיוק". תחום הוכחות הגיאומטריה ירוויח במיוחד מתוכנה שכזו, משום שאין תשובות בספרים לתרגול וקשה למצוא פתרונות מלאים לרוב התרגילים. בנוסף, שלבי הפתרון לבעיות גיאומטריה הם קבועים למדי – הפעלת משפטי רשות המשפטים לבגרות על הנתונים, אחד אחר השני, עד הגיעו לטענה שצריך להוכיח. חשבתי לעצמי שתוכנת מחשב תוכל לפעול באופן דומה על מנת להגיע לפתרון הרצוי.

בחודשים שלאחר הגיאת הרעיון, חשבתי עליו לשירוגין והتلכדתי כיצד לגשת לבעה. לבסוף, הגיעתי למסקנה שהדרך הכי הגיונית שעלה פיה התוכנה תוכל לפתור את בעיות הגיאומטריה, תהיה אם היא תחשוב כמו בני אדם על הפתרון.

כאשר אנו מנסים לפתור בעיות גיאומטריה, אנו חושבים על הנתונים כאוסף של נקודות, קווים וחווות ומנסים לחשב על משפטיים שיכולים לפעול נתונים הקיימים. כך אנו יוצרים מעין רשימה של משפטיים, שכל אחד מהם תורם מוכיח נתון שנדרש במשפט הבא, עד שנוצרת רשימה המובילה מנתוני הבעה לטענה שצריך להוכיח. באופן זהה בדיקת הבنتי של התוכנה לפעול.

התוכנה תציג את הנתונים באמצעות אובייקטים של נקודות, קווים וחווות ותנסה להפעיל משפטיים שונים על הנתונים. אנו כבני אדם, בוחרים את המשפטים שנשתמש בהם על פי אינטואיציה או ניסיון עבר, ואילו התוכנה תוכל לנצל את כוח החישוב הרב שלה ולנסות את כולם, עד שתגיע לאופציה הטובה ביותר.

כאשר החלטתי להתחיל בתהליך מחקר ולכתוב עבודות גמר במדעי המחשב, ידעתי שזו תהיה ההזדמנות שלי לבנות את התוכנה שחשבתי עליה. רציתי להוציא את הרעיון שלי לעולם ולהוכיח לעצמי שהואאמת יכול לעבוד. בספר זהATAR בפניכם, הקוראים, את תהליך ייצור התוכנה שככתבתי ואת התוצר הסופי שהגעתי אליו.

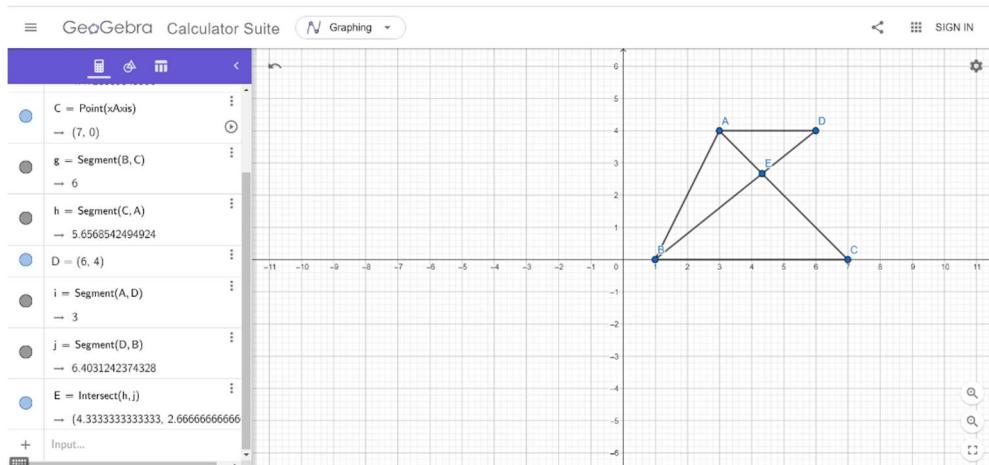
התוצר הסופי ומדריך למשתמש

התוצר הסופי שלי הוא מuin API שבעזרתו המשתמש מזין את הנתונים לתוכנה וմבקש ממנו ליצור הוכחה לבית הגיאומטריה שלו. התוכנה דורשת קוואורדינטות של נקודות בשילוב שתכל לדעת בערך היכן נמצאות הנקודות ביחס אחת לשניה במישור.

כלי שימושי ביותר לשרטוטים במחשב הוא האתר "GeoGebra". השימוש בו רבות לשרטוטים של בעיות ולבסוף הוספה לתוכנה אפשרות ליבא אוסף נקודות משפטו מהאתר.

תהליך השימוש בתוצר

ראשית, על המשתמש לשרטט את בעית הגיאומטריה באתר "GeoGebra". אין חובה לשרטט את הקווים או נתוניים נוספים, מה שחשוב והכרחי הם הנקודות מהן בנוי השרטוט.



שנית, על המשתמש להוריד את הקובץ המתאר את השרטוט באופן המואר בתמונות הבאות:

The screenshots show the GeoGebra interface. The first screenshot shows the main workspace with a red box around the top-left menu bar. The second screenshot shows a context menu with 'Download as' highlighted. The third screenshot shows the 'Download as' submenu with 'GeoGebra file (.ggb)' selected, also with a red box around it.

לאחר מכן, נעבור לחלק של הזנת הנתונים באמצעות פיתון.

על המשתמש להגדיר אובייקט מסוג **Helper** שיעזר להזנת הנתונים בצורה נוחה. הנה דוגמה להזנת הנתונים לשרטוט שהציגי קודם לכן:

```
def p353_e27(print_proof=True):
    h = Helper()
    h.ps_from_file(r"ggb_test\p353_e27.ggb")
    # can also be written as:
    # h.ps("ABCDE", [0, 0, 6, 4, 2.4], [4, 0, 0, 4, 2.4])
    h.s("AB", "BC", "CEA", "AD", "DEB")
    h.paras("AD", "BC")
    h.seta("DAC", 42)
    h.seta("CEB", 94)
    h.calc(print_proof)
    return (h.get("EBC"),), (44,),
```

בשורה השנייה של הפונקציה, נקראת הפעולה **ps_from_file** שמייבאת את אוסף הנקודות מקובץ **ggb**. דרך נוספת ליבא את הנקודות היא להכניס את הקואורדינטות שלהן באופן יידי, באמצעות הפעולה **ps**.

בשורה החמישית, התוכנה קולעת את הקטעים השונים בשרטוט (**s**). אחר כר נתונים שני ישרים מקבילים (**parallel segments**) (**paras**) ושני ערכים של זוויות (**seta** = **set** (**angle**)). לבסוף נקראת פעולה **calc** שיוצרת את הוכחה ונקראת הפעולה **geta** (**get angle**) שמחזירה את ערך **EBC**.

פעולות שימוש למשתמש במחלקה **Helper**

- **ps / ps_from_file** – ייצרת אוסף נקודות.
- **s** – ייצרת קטע או מספר קטעים.
- **conts** – המשכת קטע קיים ו/או הוספת נקודות בינוים.
- **given** – ייצרת נעלם של מעלות או אורך.
- **tri / poly** – ייצרת משולש או מצלע.
- **tri_med, tri_angbi, tri_alt, tri_segbi** – ייצרת קטעים מיוחדים במשולש (תיקון, חוצה זוויות, גובה ואנך אמצע).
- **seta / sets** – הגדרת ערך של קטע או זוויות במספר או ביטוי.
- **equala / equals** – הגדרת שוויון בין שני קטעים או שתי זוויות.
- **perps, paras** – הגדרת שני קטעים מאונכים או מקבילים.
- **calc** – קריאה לחישוב והוכחה על פי הנתונים הקיימים.
- **geta / gets** – קבלת ערך של קטע או זוות לאחר הוכחה וחישוב.
- **isparas** – בדיקה האם שני קטעים הם מקבילים לאחר הוכחה וחישוב.

מבוא תיאורטי

מבוא לשפת פיתון

הפרויקט שלי נכתב בשפת פיתון גרסה 3.8. שפת פיתון היא שפת תכנות דינמית שasma דגש על קריאות הקוד וקלות כתיבת הקוד. השפה פותחה מtower רצון להציג לשפה פשוטה ומובנת, נוחה לקרוא וקלה לתחזקה. לצד עיניהם של מפתחי השפה, עדמה המטרה לאפשר קוד "יפה", "מפורש" ו"פשוט".

השפה נחשבת ל"שפה עילית" - היא מושתתת על יסודות משפט C אך מפשטת המונע עקרונות ומכליה מנוגנים אוטומטיים שהופכים אותה להרבה יותר קריאה וחוסכים כתיבה רבה של קוד.

מאפיין בולט בשפה הוא השימוש בהזחות (tabs) להגדרת בלוקים של קוד, בניגוד לשפות אחרות בהן יש שימוש בסוגרים מסולסלות או במיללים שמורות.

מבוא לפתרון בעיות גיאומטריה בשיטת "טענה-נימוק"

גיאומטריה היא ענף בתחום המתמטיקה העוסק בצורות וקשרים שבין קווים, נקודות חוויות. היוונים הקדמונים עסקו בענף זה, וביניהם אוקלידס, מתמטיקאי יווני הנחשב לאבי הגיאומטריה. בספרו "יסודות" הוא ערך באופן שיטתי את מרבית הידע הgeomטריה והמתמטי שנוצרו עד לתקופתו.

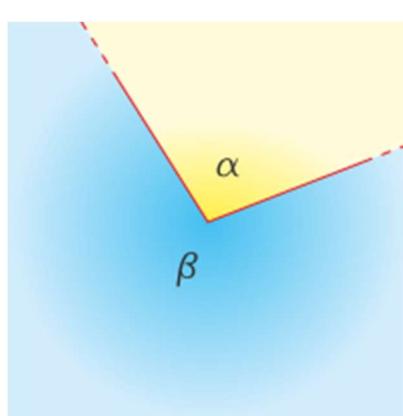
כמו מושגים שאתיחס אליהם בהמשך העבודה:

גיאומטריה אוקלידית – הגיאומטריה כפי שהוא מכיר אותה כיום, המכונה כך על שמו של אוקלידס. זו בעצם התורה המתמטית של נקודות, ישרים ומעגלים במשור.

אקסומה – הנחת בסיס אשר מתייחסים אליה כנכונה ולבונת מלאו. לא ניתן להוכיח את הנחה זו ולפיה ניתן להסיק מסקנות שונות על המבנה הלוגית שבה היא נמצאת. השילוב בין מספר אקסומות נקרא "מערכת אקסומות". מקור המילה "אקסומה" היא מיוונית עתיקה ומשמעותה "עקרון מובן מלאו", שאינו מצריך הוכחה.

משפט – בתחום הגיאומטריה, משפט הינו מסקנה לוגית שנייה להסיק כתוצאה מתנאים מסוימים. המשפטים השונים בגיאומטריה מוכחים על בסיס האקסומות והמשפטים הקיימים (שגם הם מוכחים על בסיס האקסומות). משפטיים בגיאומטריה בדרך כלל מוכבבים במבנה של "אם אז...". כמובן, אם מתקיים מצב מסוים, אז ניתן להסיק עליו את המסקנות הבאות. לדוגמה, המשפט הידוע "סכום הזויות של משולש הוא 180° ", טוען שאם מתקיים מצב בו קיים משולש, אז סכום הזויות שלו יהיה שווה -180° (בהמשך העבודה אוכיח את המשפט ואנו תומכה שליל להוכיח אותו בעצמה).

геומטריה האוקלידית מבוססת על כמה אקסומות (הנקודות) בסיסיות המתארות את היחס בין כמה מושגים יסודים בתחום הגיאומטריה. המושגים הללו אינם מוגדרים באופן פורמלי, אלא מאופיינים באמצעות האקסומות העוסקות בהם.icut את אובייקט הגדירה לא פורמלית למושגים אלו, על מנת ליישר קו (intended) בין הקוראים השונים:



נקודה – הנקודה מייצגת מקום מדויק במרחב, היא חסרת אורך, רוחב ועומק.

ישר – אוסף נקודות אינסופי בעלות מידת יחיד, אורך. חלק של הישר המצווי בין שתי נקודות עליו הוא קטע. קרן היא חלק של הישר שמתחיל בנקודה מסוימת ונמשך עד אינסוף.

זווית – חלק מהמשור המוגבל על ידי שתי קרניים בעלות נקודת קצה משותפת. הנקודה המשותפת נקראת "נקודת הזווית" (ובאנגלית vertex). ניתן לראות בתמונה המuschת החלוקה של המשור על ידי שתי הקרניים. החלק הצהוב של המשור הוא הזווית המסומנת באות אלפא, והחלק הכהול הוא הזווית המסומנת באות בטא.

כפי שצייתי קודם לכן, על פי האקסיומות הבסיסיות ניתן להוכיח את כל המשפטים הידועים לנו בגיאומטריה. אציג לפניכן את שלושת האקסיומות המרכזיות שבהן:

1. כל זווית הישר (או השטוחות) שוות בינהן.

לקורא המומצע, זו תראה כמו אמת ברורה מליו שאין צורך לציין אותה, אך משום שאנחנו נשענים על נכונות הטענה זו בהוכחות שלנו ואנו לא מוכחים אותה עצמה, היא נחשבת אקסיומה.

2. השלם שווה לסכום חלקי.

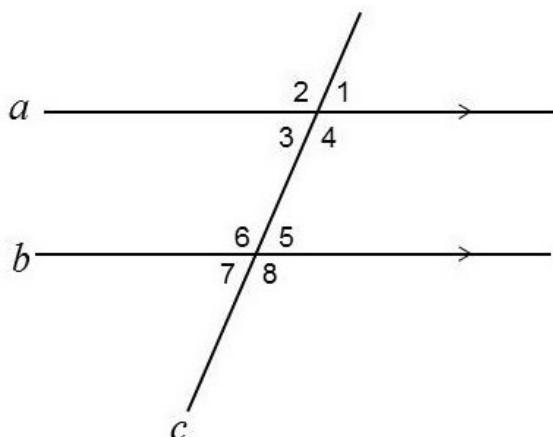
ישנן גרסאות רבות לאקסיומות מהאוף הזה, וכולן ברורות והגיניות להפליא.

3. זוג זוויות מתחלפות בין שני ישרים מקבילים וחותך הן שווות.

משפט הכרחי ביותר, אך הרבה פחות אינטואיטיבי מאשר האקסיומות שהציגתי. גם למשפט זה יש גרסאות רבות (רובן שקולות) אך בחרתי לציין את גרסה זו משום שכאשר למדתי לראשונה את תחום הגיאומטריה בכיתה ז', זהה האקסיומה שהוצאה לפני.

על מנת להבין את האקסיומה הבאה, ראשית עליינו להבין מהם "שני ישרים מקבילים". שני ישרים מקבילים הם ישרים הנמצאים באותו מישור אך אין נפגשים לעולם (גם אם נמשיר אותם עד אינסוף). שנית, עליינו להבין מהן "זוג זוויות מתחלפות".

בין שני ישרים מקבילים וחותך יש שלושה סוגי זוויות:

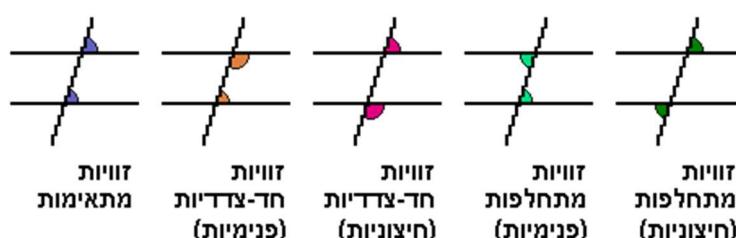


a. זוויות מתחלפות – זוויות מתחלפות הן שווות אחת לשנייה. לדוגמה: 4,6 או 2,8. הן מתחלפות מצד שלhn עם הישר וחותך. בדוגמה זו, זווית 4 נמצאת מתחת לישר ומימין לחותך וזוית 6 נמצאת מעל הישר ומשמאלו לחותך. זוג זוויות מתחלפות שנמצאות בין השרים נקראות "זווית מתחלפות פנימיות".

b. זוויות מתאימות – זוויות מתאימות הן שווות אחת לשנייה. לדוגמה: 1,5 או 3,7. הן מתאימות מצד שלhn עם הישר וחותך.

בדוגמה זו, זווית 1 נמצאת מעל הישר ומימין לחותך וכן גם זווית 5.

ג. זוויות חד צדדיות – זוויות שנמצאות מצד אחד של החותך ומצדדים שונים של הישר. לדוגמה: 4,5 או 1,8. בדוגמה זו, זווית 4 נמצאת מתחת לישר ובצד ימין של החותך ואילו זווית 5 נמצאת מעל הישר ובצד הימין של החותך. זוג זוויות חד צדדיות שנמצאות בין השרים נקראות "זווית חד צדדיות פנימיות".



כעת נחזר לאקסיומה שעסוקנו בה, "זוג זויות מתחלפות בין שני ישרים מקבילים וחותך הן שות". אם נתבונן בכמה דוגמאות למערכת של ישרים מקבילים וחותך, נראה שהאקסיומה הזו מתקינה, אך לא ניתן להוכיח אותה באמצעות מושגיה האקסיומות שהתייחסנו אליה ("הוכחה" נפוצה לאקסיומה זו משתמשת במשפט של סכום הזויות במשולש, אך משפט זה מtabס על האקסיומה ולא להפר ולכנן לא ניתן להשתמש בו בשביב להוכחה).

הידע שכיסינו בעמודים האחרונים מספיק על מנת להבין את עבודה זו, ולמעוניינים בחומר גיאומטרי נוספת, בהמשך העבודה אovich כמה משפטיים בסיסיים באמצעות האקסיומות שהציגנו וכמוון גם באמצעות התוכנה שפיתחתи.

החלק המעני

כאשר ניגשתי לכתוב את GeoAD בפעם הראשונה, היה לי רעיון כללי למודל שעלה פיו התוכנה תוכל לפתרו ולהוכיח משפטי בגיאומטריה בשיטת "טענה-נימוק". אך בסופו של דבר, הבנתי שהמודל הנ"ל פגום ביסודו ונאלצתי לחשב על מודל חדש (דומה בעיקרו אך שונה), שיכל לפתור את הבעיה שעלו מהמודל המקורי. כתעת אציג בפניכם את המודל המקורי, שרוב ההגיוון שבו תקף גם לגבי המודל הנוכחי והסופי של התוכנה.

דרישות המודל

- קליטה של נתוני הבעיה מהמשתמש.
- קליטה של המשפט או הטענה שיש להוכיח מהמשתמש.
- פלט של הוכחת הטענה שהיא נדרש להוכיח.

המודל המקורי

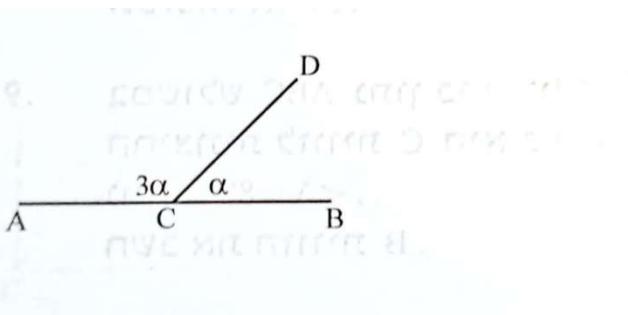
כאשר דיברתי עם אנשים שונים על הרעיון לעבודה שלי בשנה האחרונה, והסבירתי להם את הפתרון שחשבתי עליו, רובם התקשו בהבנת הדרך שבה התוכנה תקלוט את נתונים הבעיה. דרך קליות הנתונים היא ליבת הפתרון שלי וכאשר תבינו אותה, כל שאר מרכיבי הפתרון יהיו ברורים לחלוטין.

על פי המודל, המשמש יזין לתוכנה את שמות הנקודות המופיעות בבעיה, את הקווים העוברים דרך הנקודות ונתונים נוספים כגון: ערך זוויות מסוימות, אם חלק מהישרים מקבילים זה לזה וכו'. המודל לא מקבל נקודות במערכת צירים או שרטוט באופן כלשהו, אלא שומר בזיכרון את נתונים הבעיה כפי שאנו מבינים אותם בראשנו, וכך שHAM מוגדרים באופן בסיסי: נקודות, קווים וזוויות.

אמחיש את כוונתי באמצעות דוגמה:

הנקודה C נמצאת על הקטע AB .
 נתון : $\alpha = \angle BCD$, $\angle ACD = 3\alpha$.
 א. חשב את α .
 ב. מהו גודל הזווית ACD ?

תשובה: א. 45° . ב. 135° .



בעיה זו נתון לנו הקטע AB שעליו נמצאת הנקודה C. בנוסף, יש קטע CD היוצא מנקודה C. ישנן שני סימונים - $\angle EBC$ שווה $-\alpha$ ו- $\angle ACD = 3\alpha$. אנו נדרשים לחשב את גודלה של אלפא ואת גודלה של $\angle ACD$.

את ההסבר המילולי שהציגתי כאן, ניתן להמיר בקלהות לפקודות במשק התוכנה שלי. גם מבי להבין אותו באופן מלא, תוכלו לראות התאמה בין ההסבר המילולי שלי לבין הקוד:

```
def p349_e2():
    h = Helper()
    h.s("ACB")
    h.s("CD")
    x = Degree.given("x")
    h.seta("DCB", x)
    h.seta("ACD", 3 * x)
    h.calc()
    return h.get("DCB"), h.get("ACD")
```

בשורה השנייה של הפעולה אני מכניס כקלט את הקטע ACB ובשורה שלאחר מכן את הקטע CD.

לאחר מכן, אני יוצר אובייקט בשם x שיחליף את אלף במקורה שלנו ואני מגדר שווית DCB \angle שווה לאיקס שווית ACD \angle שווה לא- $3x$. לבסוף, אני מבקש מהתוכנה את הערך של זווית DCB שווית ACD \angle .

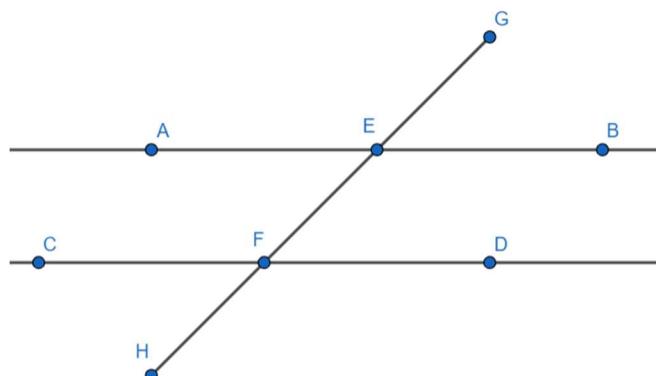
בשומם שלב במהלך התוכנה לא הצagi לתוכנה את השרטוט, או "גיליתי" לה היכן הנקודות נמצאות במרחב. התוכנה שומרת את הנתונים שקיבלה באמצעות אובייקטים של נקודות, ישרים חוויות וכן מצילה להכיל עליהם את המשפטים השונים.

בעיות במודל הישן

המודל הישן מצילח להתמודד עם כמות נכבדת של בעיות אך במהלך הפיתוח שלו נתקלתי בכמה תקלות. בחלק הבא אציג את התקנות האלו, את הפתרונות החלקיים שהשתמשתי בהם על מנת "לפטור" את התקנות ואסביר מדוע, לדעתי, מודל כפי שתיארתי קודם لكن לא יכול להתקיים ויש חובה לשנות דבר מהותי בו.

כיוון ישרים

בואו נראה כיצד התוכנה קיבל את הקטע של שני ישרים מקבילים וחותך:



```

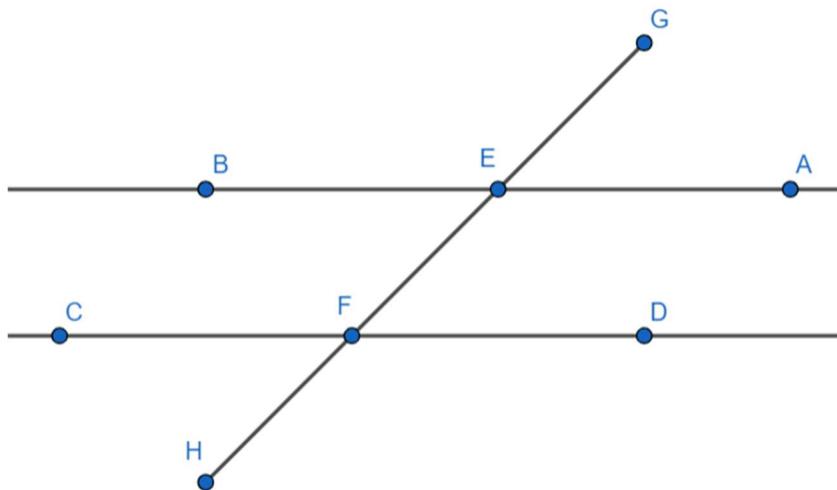
h = Helper()
h.s("AEB")
h.s("CFD")
h.s("GEFH")
h.paras("AB", "CD") # set AB || CD

```

נגידר את שלושת היסרים: AB, CD ו-GH עם נקודות הביניים שיש עליהם ואחר כך נגידר שישר AB מקביל ל-CD.

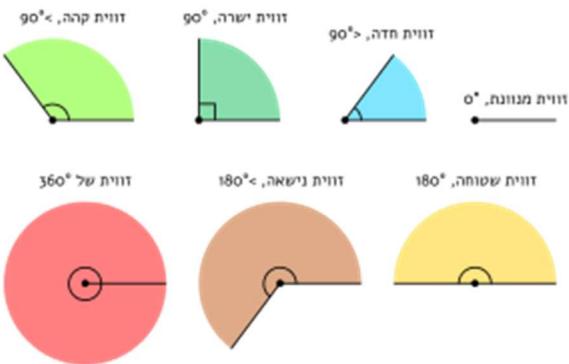
המודל מצליח לקבל את הקטל זהה ועובד בצורה מעולגה, אבל כשבדקתי מספר בעיות נוספות הבנתי את הבעיה הנוצרת – המודל אינו מודע לכיווני היסרים. ניתן להמחיש זאת באמצעות דוגמה פשוטה.

אם נחליף את נקודות A ו-B בשרטוט, הקטל ישאר זהה אבל השרטוט ישתנה ויוצר שינוי בין הנתונים בשרטוט לבין הנתונים בתוכנה.



כך למשל, בשרטוט האחרון מופיעות זוג חזויות המתאימות: $\angle ADF \cong \angle AEH$, אך על פי הנתונים של התוכנה (שהם לשרטוט הקודם), זוג חזויות המתאימות הוא בעצם: $\angle AEH \cong \angle DFE$.

על מנת לפתור את הבעיה זו במודל הישן, הוסיףתי כלל שני ישרים מקבילים צריכים להיות מוכנסים לתוכנה באותו הכיוון. ככלומר בשרטוט החדש, התוכנה צריכה לקבל את היסרים BEA ו-CFD על מנת שתתדע את את הכיוון הנוכחי של היסרים. נראה בהמשך מדוע "כלל" זה אינו מספיק על מנת לפתור את בעיה זו.

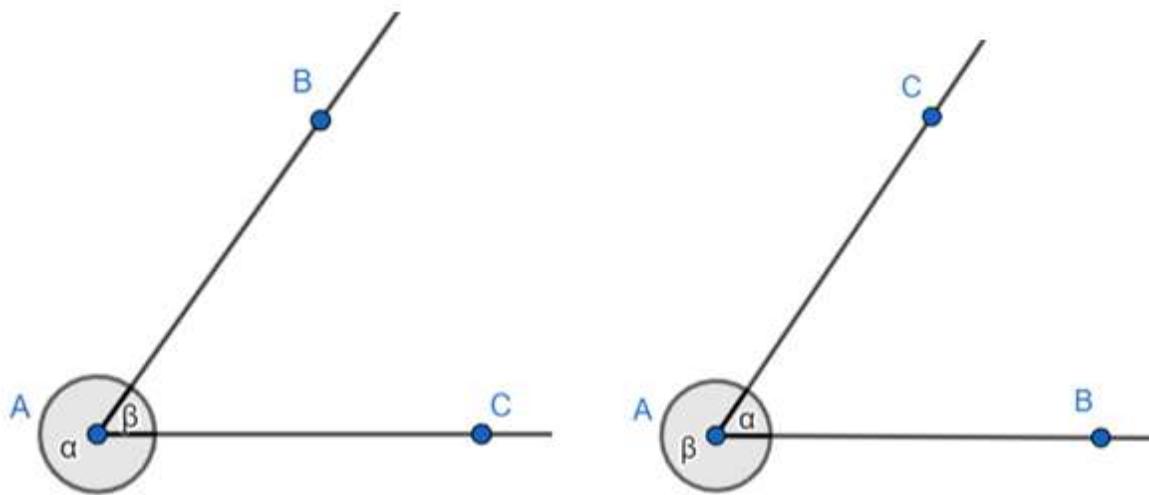


זווית נישאה

בגיאומטריה, נהוג להגדיר את סוג הזווית על פי הגודל שלה. הסוגים המוכרים הם זווית חדה, ישרה, קלה ושטוחה. אך סוג פחות מוכר, שהינו גם פחות שימושי, היא הזווית הנישאה (באנגלית reflex angle). זווית נישאה היא זווית שגודלה מעל 180° ומתחת -360° .

לרוב בבעיות גיאומטריה, אין התייחסות לزواיות נישאות, אלא מדובר בעיקר על זוויות הקטנות מ- 180° . אך לצורך התוכנה, חובה התייחסות לزواיות כאלה, משום שהתוכנה אינה יודעת אילו מהزواיות היןן זוויות נישאות ואילו לא.

לדוגמה, בשרטוט השמאלי, ברור לנו שזוית אלף היא הזווית הנישאה, הזווית בטא היא אינה זווית נישאה. אך בהינתן הקלט לתוכנה, היא לא תוכל להבדיל בין השרטוט הימני לשמאלי:



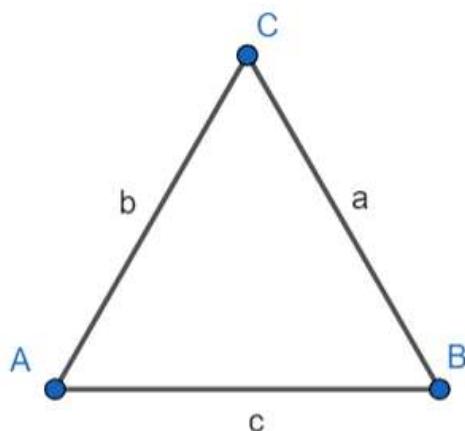
```
h = Helper()
h.s("BA")
h.s("CA")
```

על מנת שנוכל להתייחס בקלות לשתי הזווית אלף ובטא, הגדרתי סימון זווית מוגדרת עם כיוון השעון, מהק焉 הראשונה אל השניה. כלומר, זווית BAC בשרטוט השמאלי מתיחס לזוית בטא, זווית מקטע AB אל קטע AC עם כיוון השעון. הגדרה זו תשאר עד סוף העבודה, וגם במודול החדש.

בכדי לפטור את הבעיה של זיהוי הזרויות הנישאות במודל הישן, יצרתי כל נוסף: על המשמש להכנס את היסרים לפי הסידור שלהם בהתייחסות לכל נקודה, על פי כיוון השעון, כאשר במקרה שיש רק שני ישרים העוברם בנקודה, הזרות הנישאה תוכנס אחרונה.

על פי כלל זה, הקלט שהציגתי מתאר רק את הشرطוט השמאלי ואילו הקלט לשרטוט הימני יגדיר את AC קודם ואחריו את AB . כלל זה מסביר את הכנסת הקלט של המשמש לתוכנה, משום שהוא מחייב את המשמש לחשב רבודות באיזה סדר יכנס את הקלט. אך יתרה מזאת, הכלל הזה יוצר מצבים "בלתי אפשריים" להכנסת קלט.

شرطוטים בלתי אפשריים

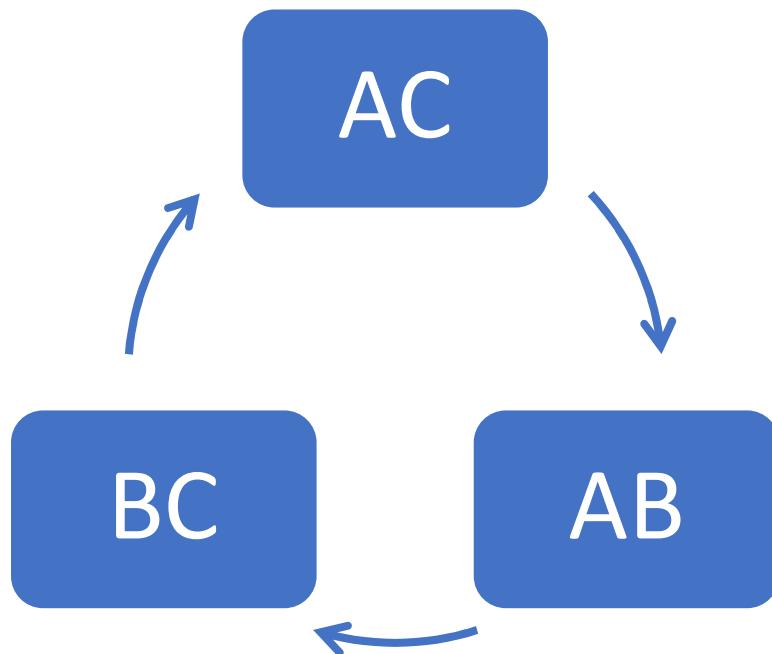


דוגמה לשרטוטים בעיתים היא דוקא פשוטה
לmdi: $\triangle ABC$.

אם ננסה לקבוע את סדר הכנסת היסרים לתוכנה,
נקבל פרדוקס.

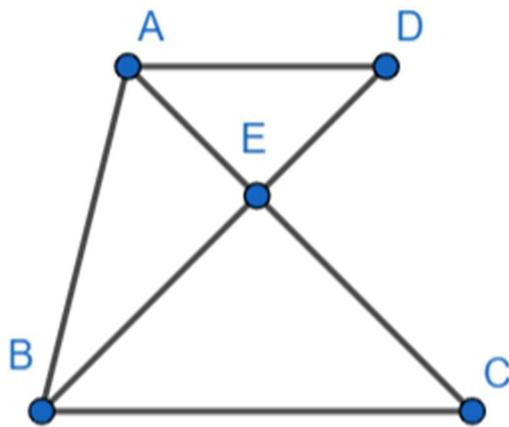
- לפי נקודה A, יש להכנס את קטע AC ואחריו את קטע AB .
- לפי נקודה B, יש להכנס את קטע AB ואחריו את קטע BC .
- לפי נקודה C, יש להכנס את קטע BC ואחריו את קטע AC .

נסכם את המסקנות שקיבלנו בדיאגרמה, כאשר חוץ מקטע אחד לקטע שני אומר שיש להכנס את הקטע הראשון לפני השני:



לכן, אין לנו אפשרות להכנס את הקלטים בצורה הרגילה על פי הכללים שציינתי עד כה.

כאשר נתקלתי לראשונה בבעיה זו, לא ויתרתי בקלות על המודל הישן ויצרתי פונקציות עזר שמשנות את הסדר של היסרים בנתוני התוכנה, כך שהמשתמש יוכל להכנס קלט של "משולש ABC" והתוכנה תדע כיצד לסדר את היסרים בצורה הנכונה, אך ככל שהشرطוטים הסתבכו, הבנתי כי גישה זו לא יכולה להחזיק במשך זמן רב.



זו היא דוגמה לשרטוט שמאוד בעייתי להכניס אותו באמצעות המודל הישן.

המודל החדש והסופי

לאחר חישבה מרובה בנושא, הבנתי שאין אפשרות נוחה להכניס את הנתונים מבלי שרטוט כלשהו לתוכנה, על פי הסיבות שהציגו עד כה. לכן, במודל החדש הוסףתי קלט אט הקואורדינטות של הנקודות במערכת צירים. יש לציין שהקואורדינטות לא חייבות להיות מדויקות אלא משמשות לצורך הערכה גסה לסידור הנקודות במישור (למשל על מנת להבדיל בין שני שרטוטים "זהים" למודל הישן, כפי שהציגי מוקדם). בעזרת הקואורדינטות האלו הצלחתי להכניס לתוכנה את הנתונים הדרושים לה על מנת להוכיח את הטענות הרצויות, בקלות יחסית למודל הקודם.

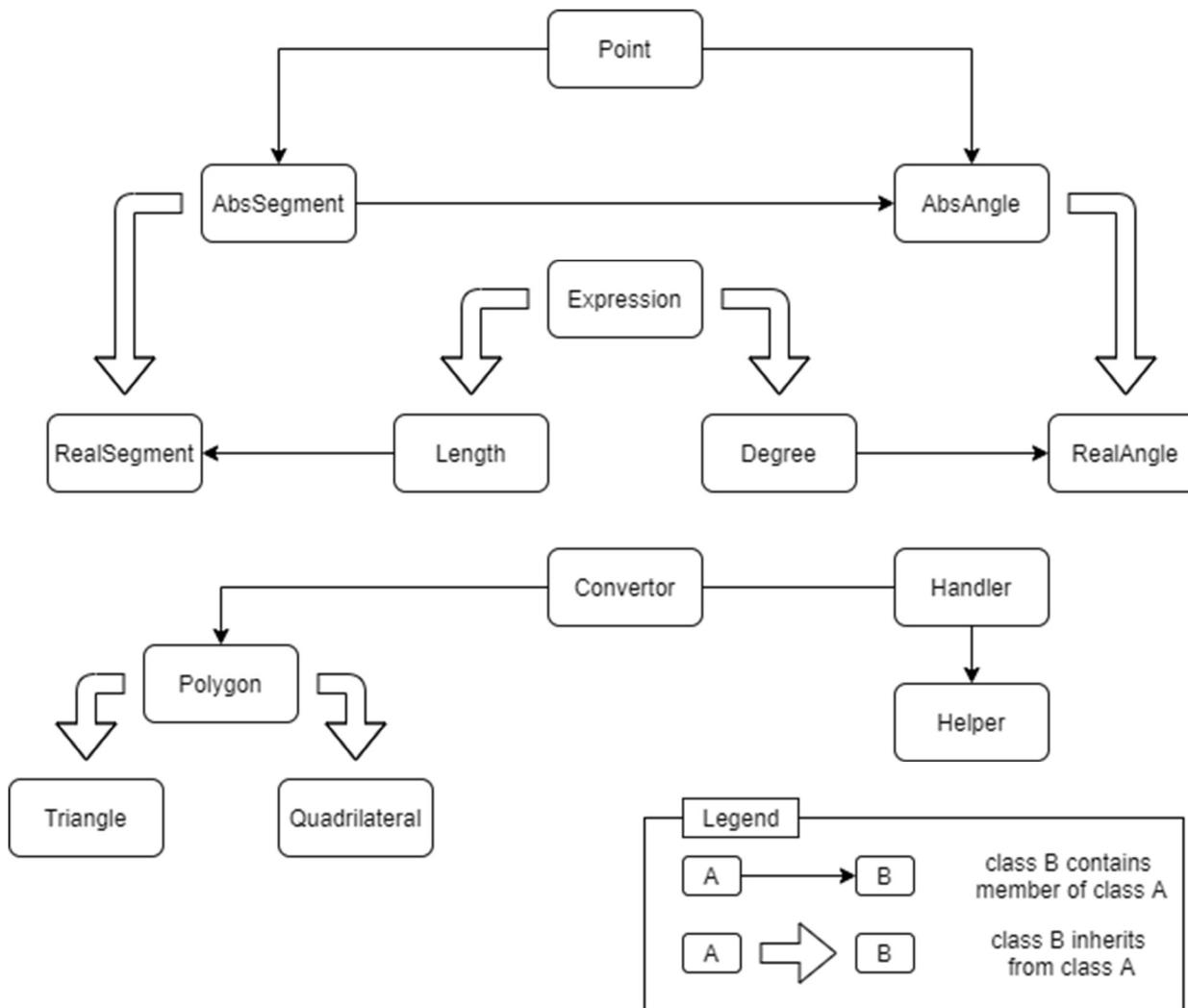
ברמת המשמש, השינוי הוא קטן. לפני הכנסת הקטעים והנתונים, על המשתמש להכניס את הנקודות שנמצאות בשרטוט ואת הקואורדינטות המשוערות שלهن במערכת צירים. לאחר מכן, המשתמש מכניס את הנתונים באותו האופן כמו קודם, רק שבעת אין שום כללים עליהם הוא צריך לחשב במהלך הכנסת הנתונים. דוגמה להכנסת הנקודות ושיפור בנוחות שביצעת:

```
# long version
h.p("A", -4, 0)
h.p("B", 4, 0)
h.p("C", 0, 0)
h.p("D", 3, 3)

# short version
h.ps("ABCD", [-4, 4, 0, 3], [0, 0, 0, 3])
```

לאחר כמה בדיקות של תרגילים, גם השיטה הקצרה יותר נראית לי כמיותרת למשתמש, ולכן הופטי אופצייתנו יותר, שמשתמשת בשרטוט באתר "GeoGebra". ארכיון על אופצייה זו בהמשך.

מבנה המודל



(geo/abs/point.py) Point

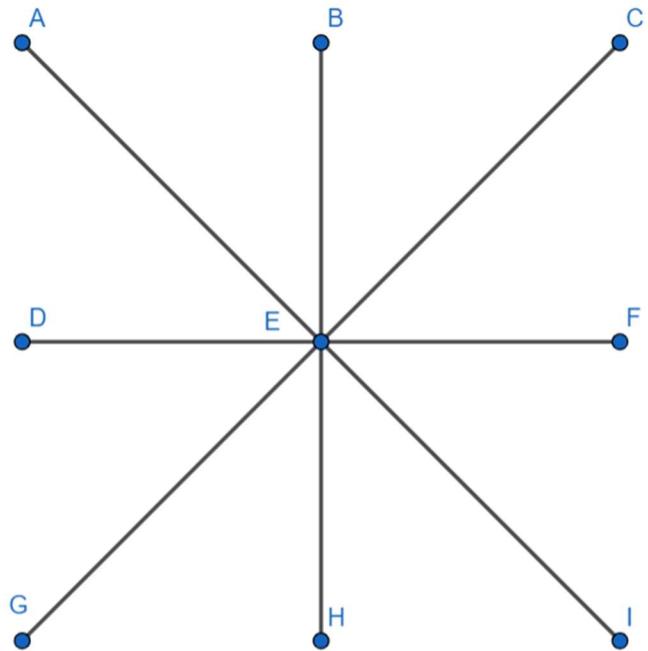
המחלקה מוגדרת באמצעות `y`, `x` כאשר `y`, `x` הן הקואורדינטות של הנקודה במערכת הצירים. כל פעם שנוצר `Point` חדש, הוא מקבל מספר `id` חדש, והמשתנה `הסטטי Point.next_id` גדל באחד.

בנוסף, לכל אובייקט במחלקה יש תכונה בשם `lines` המתארת את הקווים העוברים דרך הנקודה. כאשר נוצר קו חדש שעובר דרך הנקודה, מתרחשת קריאה לפעולה `add_line` שמושיפה את הקו החדש למערך `lines`.

```
def add_line(self, *lines):
    """Add line or lines that the point is on"""
    for line in lines:
        if self in line.midpoints:
            # split the segment into 2 segments (self is an endpoint to both)
            self.lines.append(
                (line.get_subsegment(self.name + line.start.name), line)
            )
            self.lines.append(
                (line.get_subsegment(self.name + line.end.name), line)
            )
        else:
            self.lines.append((line, line))

    # sort self.lines by slope_angle
    self.lines.sort(key=lambda t: -t[0].get_slope_angle(self))
```

הפעולה בודקת האם הנקודה נמצאת במרכז קטע חדש או באחת הקצוות שלו. במקרה שהוא נמצאת באחת הקצוות, הקטע נוסף למערך `lines`. אחרת, הקטע מחולק לשני קטעים שהנקודה שלנו נמצאת בקצויותיהם, ושתיהם מוספים למערך `lines`, כאשר לכל איבר במערך מוסף גם הקטע המקורי שמננו הוא נוצר. לבסוף, המערך ממון לפי זווית הקטע כך שלבסוף הקטעים במערך ממוקמים לפי הזווית שלהם עם ציר ה-`x`.



כך למשל אם נכניס את הנתונים לשרטוט הבא, ונדפיס את המערך `E.lines` ואת הזווית שיצרת כל איבר במערך עם ציר ה-`x` נקבל את הפלט הבא:

```
(EI, AI) 315.0
(EH, BH) 270
(EG, CG) 225.0
(DE, DF) 180
(AE, AI) 135.0
(BE, BH) 90
(CE, CG) 45.0
(EF, DF) 0
```

(geo/abs/abssegment.py) AbsSegment

המחלקה מוגדרת באמצעות שתי נקודות קצה ומערך של נקודות ביןיהם. אם הקטע מוגדר בקטע חדש (`isnew = True`) אז כל פעם שנוספת נקודה חדשה, מתבצעת קריאה לפעולה `add_line` שלה.

לפעמים אנו צריכים ליצור תת-קטע שמאכל בתחום קטע מסוים. על מנת לעשות זאת בצורה נוחה, יצרתי את הפעולה `:get_subsegment`

```
def get_subsegment(self, name):
    """Get subsegment based on name"""
    furthest, closest = [
        p
        for p in self.get_all_points()
        if p.name == name[0] or p.name == name[-1]
    ]

    if self.get_all_points().index(furthest) < self.get_all_points().index(
        closest
    ):
        furthest, closest = closest, furthest
    return self.get_subsegment_to(furthest).get_subsegment_from(closest)
```

הפעולה מקבלת השם של תת הקטע שצרכן להחזיר ומחזירה אובייקט מסווג שמתאר אותו. ראשית, הפעולה מושגאת את שתי נקודות הקצה של תת הקטע ע"י השוואת השם שלhn לשמות של הנקודות על הקטע. לאחר מכן, הפעולה בודקת אילו משתי הנקודות היא הרחוקה ואילו היא הקרובה, ע"י בדיקה של האינדקס שלhn במערך הנקודות על הקטע.

לבסוף, הפעולה מוחזירה את תת הקטע באמצעות הפעולות `get_subsegment_to` ו-`get_subsegment_from`. הפעולה `get_subsegment_to` מקבלת נקודה שעל הקטע ומוחזירה תת קטע מתחילה הקטע עד הנקודה. הפעולה `get_subsegment_from` מקבלת נקודה שעל הקטע ומוחזירה תת קטע מהנקודה עד סוף הקטע.



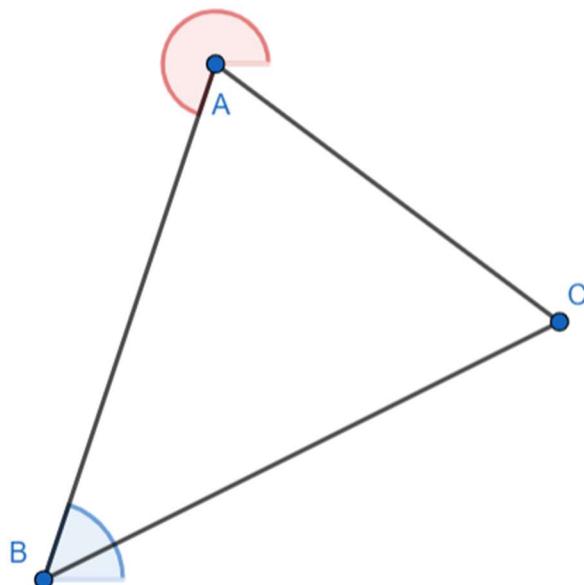
`self.get_subsegment_to(furthest)`



`self.get_subsegment_to(furthest).get_subsegment_from(closest)`



דבר נוסף שיש לדעת לבצע בין קטעים הוא לקבוע האם הם מקבילים או לא. על מנת לבצע זו בצורה נוחה, יצרתי מבנה של **UnionFind** המאפשר לזכור קבוצות של אובייקטים ולבדק בסיבוכיות גבוהה האם הם נמצאים באותה קבוצה (כלומר האם הם מקבילים) ולאחד קבוצות (לפי כלל המעבר). מצורף הסבר על המבנה שמייחsti בהמשך העבודה.



פעולה נוספת הראיתי לצוון היא הפעולה [get_slope_angle](#) שמחזירה את זווית השיפוע של הקטע לפי הקואורדינטות של נקודות הקצה שלו. החישוב הוא באמצעות הפונקציה הטריגונומטרית `arctan`. זהו בעצם השימוש היחידלקואורדינטות של הנקודות ולכן אין צורך בדיקן מוחלט של הנקודות על מנת לקבל את גודל זווית השיפוע, משום שיש צורך רק בסדר גודל. הפעולה יכולה לקבל נקודה ולהחזיר את זווית השיפוע בלבד לנקודה.

בכחול מופיעה זווית השיפוע של קטע AB ביחס לנקודה B ובאדום מופיעה זווית השיפוע של קטע AB ביחס לנקודה A.

[\(geo/abs/absangle.py\) AbsAngle](#)

המחלקה מוגדרת באמצעות שתי קרנויים (`massog` ו`קוודקוד` (`massog`)).

פעולה חשובה של המחלקה היא [get_angle_size_from_coordinates](#), המחזירה את גודל הזווית על פי הקואורדינטות של הנקודות. זהו אינו גודל מדויק ויש בו רק שימוש על מנת לקבוע האם זווית היא נישאה או לא. החישוב של הגודל המשוער זהה מתבצע באמצעות הפעולה [get_slope_angle](#) על שתי הקרנויים:

```
def get_angle_size_from_coordinates(self):
    """Return angle size based on points' coordinates"""
    ray1_slope_angle = self.ray1.get_slope_angle(self.vertex)
    ray2_slope_angle = self.ray2.get_slope_angle(self.vertex)
    return (ray1_slope_angle - ray2_slope_angle + 360) % 360
```

(geo/real/expression.py) Expression

המחלקה מציגת ערך מסוים (למשל גודל זווית או אורך קטע), בעזרת קבועים ונעלמים. המחלקה משתמשת בתכונה `value` מסוג `dict` שמאגדה את המקדים של הנעלמים השונים בערך האובייקט. כך למשל אובייקט המציג את הערך $(3\alpha - 4\beta + 42)$ יכיל את התכונה `value` עם הערך $\{0: 42, 1: 40, 2: -3, 3: 35\}$. כאשר האיבר `self.value[0]` מיצג את המקדם החופשי, והאיברים `self.value[1]` ו- `self.value[2]` מיצגים בהתאם את המקדים של אלף ובטא.

בעזרת operator overloading (העמסת אופרטורים) הגדרתי את פעולות החיבור, חיסור, כפל (בקבוע) וחילוק (בקבוע) על המחלקה. הנה דוגמה לימוש של פעולות החיבור:

```
def __add__(self, other):
    """
    Add the values for the same key and add the
    missing keys with dict/Expression/int/float
    """

    res = self.new_copy()
    if isinstance(other, type(self)):
        for idx, val in other.value.items():
            if idx in res.value:
                res.value[idx] += val
            else:
                res.value[idx] = val
    elif isinstance(other, (dict, int, float)):
        return self + type(self)(newvar=False, d=other)
    else:
        return NotImplemented

    res.clean()
    return res
```

הפעולה לחברת את הערכים להם יש מפתחות משותפים ב-self ובי-other ושם את הערכים כפי שהם במקומות שבהם אין מפתחות משותפים. דוגמה לחיבור שני ביטויים ומילונים שלهما:

$$(3\alpha - 5\beta) + (-3\alpha + 40\beta + 42) = (35\beta + 42) \\
 \{1: 3, 2: -5\} + \{1: -3, 2: 40, 0: 42\} = \{2: 35, 0: 42\}$$

כאשר התוכנה מצליחה ליצור משווה על בסיס טענה כלשהי בגיאומטריה, לבסוף מתקבלת משווהה שבה נעלם כלשהו שווה לתרוכות של נעלמים אחרים. במקרה זה, אפשר להחילף את כל המופעים של הנעלם באובייקטים אחרים מסוג `Expression` לתרוכות. זו בדיקת מה שעושה הפעולה `:switch`

```

def switch(self, key, switch_exp):
    """switch every instance of this key to switch_exp"""
    type(self).switch_watched(key, switch_exp)
    if key not in self.value:
        return
    else:
        times = self.value[key]
        del self.value[key]
        self += switch_exp * times
        self.clean()

```

הפעולה מחליפה את המופיע של `key` הנוכי ל-`switch_exp`. נשמרות כמהות הפעמים ש-`key` מופיע במשתנה `times`, נמחוק את המופיע שלו מהמילון ונוסף לערך של ה-`switch_exp` הנוכי את `times`, `switch_exp`, כמוות של `times` פעמים.

אם נרצה לשמר אובייקטים מסווג `Expression` שגם להם תקרא פעולה `switch`, נוסיף אותו למערך `watched` של המחלקה וכך בכל פעם שתקרא פעולה `switch` תקרא גם פעולה `switch_watched` שקוראת לפעולה `switch` על האובייקטים שבמערך `switch_watched`. על מנת למנוע קיראה אינסופית לפעולה `switch_watched`, נשמר גם מערך `switched` של המחלקה ונבדוק האם כבר נקרה הפעולה `switch_watched` לזוג `key, switch_exp` הנוכי.

```

@classmethod
def switch_watched(cls, key, exp):
    """
    Switch key for exp in every Expression in watched
    if it hasn't been done before
    """

    if (key, exp) in cls.switched:
        return
    cls.switched.append((key, exp.new_copy()))
    for w in cls.watched:
        w.switch(key, exp)

```

במקרה שקיבלנו משווהה שמננה ניתן לצמצם געלם כלשהו, עליינו לבחור איזה געלם "למצוא" בעזרת שאר הגעלמים וכך לצמצם אותו בכל הביטויים שיש לנו. בחרתי לצמצם את הגעלם עם המפתח הגבוה ביותר בכל פעם, ועל מנת למצאו אותו בצורה נוחה ואלגנטית בחרתי למש את פעולות ההשוואה בין שני אובייקטים מסווג `Expression`, כך שיישו ההשוואה לקסיקוגרפית על המפתחות המומוחים של שני האובייקטים. כך, האובייקט עם המפתח הגדל ביותר שאינו לאובייקט השני יהיה גדול יותר.

על מנת ליצור נעלם עם סימון על פי רצון המשתמש, התוכנה משתמש בפעולת `given`. הפעולה תקרא לפעולת `watch` על האובייקט החדש על מנת שייעודכו ערכו. ערך המפתח של הנעלם החדש יהיה $1/3$, וכן הלאה, זאת על מנת שהתוכנה תשאף להחליף את שאר הנעלמים שערכם המפתח שלהם הוא $1, 2, 3$, וכן הלאה) לפני שתחליף את הנעלם החדש. מה שקרה בפועל הוא שהנעלם החדש יקבל ערך מסוים או שכל שאר הנעלמים ייצגו באמצעותו.

```
@classmethod
def given(cls, *symbols):
    """Returns Expression(s) with the given symbol(s)"""
    res = []
    for symbol in symbols:
        d = cls(False, {})
        d.value[1 / cls.next_given_idx] = 1
        cls.given_symbols[1 / cls.next_given_idx] = symbol
        cls.next_given_idx += 1
        d.watch()
        res.append(d)
    return tuple(res) if len(res) > 1 else res[0]
```

(geo/real/expression.py) Length- α Degree- β

מחלקות ירושות מ-`Degree` ואין להן פונקציונליות נוספת. המחלקה `Degree` משתמש באלפבית היווני לסימון הזווית במקום האלבבית האנגלית כמו `Length`-`Expression`.

יצרתי את הפרדה בין שתי המחלקות כדי להציג את ההבדל בין גדי זווית לאורכים ולמנוע השוואה בין אחד לשני (למשל כאשר קוראים לפעולת `switch_watched` על אחד מהם, אין צורך לבדוק את המשתנים של השני).

(geo/real/realangle.py) RealAngle

מחלקה ירושת מ-`AbsAngle` ומוסיפה לה את התכונה `deg`, ערך חזיות מסוג `Degree`. הפעולה תומכת בפעולות חיבור וחיסור כמו `Degree` ובנוסף, יש אפשרות ליצור אובייקט מהמחלקה מתוך אובייקט `AbsAngle` באמצעות הפונקציה הבאה:

```
@classmethod
def fromAbsAngle(cls, absang, deg=None):
    """Create a RealAngle based on an AbsAngle"""
    return RealAngle(absang.ray1, absang.vertex, absang.ray2, deg)
```

בנוסף, המחלקה ממשת את פעולות `abs` כרשותן לקרוא עליה לאובייקט מהמחלקה ולקבל את האובייקט ה-`Abs` שלו.

(geo/real/realsegment.py) RealSegment

המחלקה ירושת מ-**AbsSegment** ומוסיפה לה את התכונה `length`, ערך הקטע מסוג **Length**. הפעולה תומכת בפעולות חיבור וחיסור כמו **Length**. בנוסף, יש אפשרות ליצור אובייקט מהמחלקה מתוך אובייקט **AbsSegment**. יתר על כן, המחלקה מימושת את פעולות `abs` כרשותה לארון **RealAngle** בדומה לפונקציונליות במחלקה **AbsSegment** ל-**RealSegment**.

(geo/handler.py) Handler

זו היא המחלקה המרכזית בפרויקט. המחלקה מקבל רשימה של נקודות וממנה יוצרת את רשימת הנקודות המופיעים בנתונים. הפעולות במחלקה מחולקות לכמה קבוצות, אסביר בקצרה על הפעולות המשמעותיות בכל חלק.

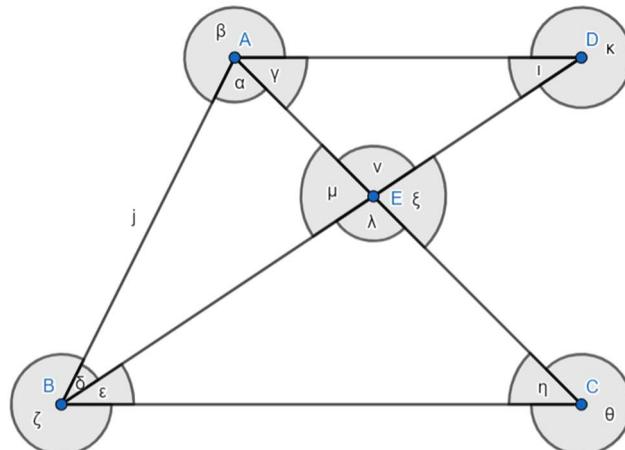
פעולות בסיסיות על AbsAngle

במהלך תהליך ההוכחה, התוכנה משתמשת באובייקטים מסוג **AbsAngle** על מנת לבדוק האם ניתן להפעיל משפטיים שונים על הנתונים הקיימים. לאחר שנמצא שווין מסוים על פי אחד המשפטים, התוכנה מבצעת המרה של הזויות מסוג **AbsAngle** לזרויות מסוג **RealAngle**.

כלشرطוט ניתן לפרק לאוסף של זויות "יסודות" (elementry). על מנת למצוא את אוסף הזויות האלו, התוכנה עוברת על כל הזויות מסביב לנקודות, ומתקבלת רשימת זויות יסודות מכל נקודה.

```
def get_angles(self):
    """Return a list of all the elementary AbsAngles"""
    res = []
    for p in self.points:
        res += self.get_angles_around_point(p)
    return res
```

הנה דוגמה לכל הזויות היסודות בشرطוט מסוים:



לכל נקודה, התוכנה יוצרת את הזרויות היסודיות מכל זוג קוים סמוכים שעוברים בנקודה
באמצעות הפעולה `:get_angles_around_point`

```
def get_angles_around_point(self, p):
    """Return a list of all the elementary AbsAngles around a point"""
    if len(p.lines) == 0:
        return []
    elif len(p.lines) == 1 and p in (
        p.lines[0][0].start,
        p.lines[0][0].end,
    ):
        return []
    rays = [line for line, _ in p.lines]

    return [
        AbsAngle(r1, p, r2) for r1, r2 in zip(rays, rays[1:] + rays[:1])
    ]
```

פעולה חשובה נוספת היא הפעולה `get_non_reflex_angle` המקבלת שלוש נקודות ומחזירה את הזווית שאינה נישאה בין הנקודות. הפעולה בודקת האם ידוע לנו שאחת מהזוויות קטנה מ-180, אחרת היא בודקת האם ידוע לנו על תת סכום שמסוגל בתור את הזווית שווה או גדול מ-180. רק אם כל התנאים הללו לא מתקיים, התוכנה בודקת את הגודל המשוער של הזווית עפ"י הקואודינטות המשוערות של הנקודות וכך מחליטה איזו זווית היא הנישאה ואיזו זווית היא אינה נישאה.

```
def get_non_reflex_angle(self, pfrom, vertex, pto):
    """Return non reflex angle based on 3 points"""
    ans = [
        AbsAngle(
            self.get_full_seg(pfrom, vertex),
            vertex,
            self.get_full_seg(vertex, pto),
        )
    ]
    ans.append(AbsAngle(ans[0].ray2, vertex, ans[0].ray1))
    # ans[0] and ans[1] are the two options
    # both close a circle around the vertex

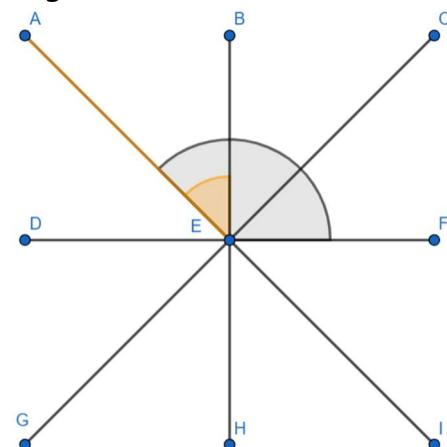
    for i in range(2):
        a = ans[i]
        arr = self.disassemble_angle(a)

        a_sum = sum(self.aconv[a])
        # if a is known to be less than 180, return a
        if a_sum.isknown() and a_sum < 180:
            return a
        for ii in range(len(arr)):
            for j in range(ii, len(arr)):
                # if a contains a subsum that is >= 180, return the other ans
                a_sum = sum([self.rangles[aa] for aa in arr[ii : (j + 1)]])
                if a_sum == 180 or all([
                    val > 0 for key, val in (a_sum - 180).value.items()])
                ):
                    return ans[0] if i == 1 else ans[1]

    # use x,y to understand which is acute
    return min(ans, key=lambda aa: aa.get_angle_size_from_coordinates())
```

פעולה שיש בה שימוש רב במהלך הפרויקט היא הפעולה `disassemble_angle` שמקבלת זווית מסווג `AbsAngle` ומחזירה רשימה של הזווות היסודות שמרכיבות אותה.

הפעולה עוברת עם כיוון השעון על הזרויות הבסיסיות שמסבב לקודקוד הזרית שיש לפרק (ang).



בדוגמה שלפנינו, הפעולה מנסה למצוא את זווית AEF. הפעולה עוברת על הקטעים העוברים בקודקוד הזווית (נקודה E) עד שהגיע לזרית BEF. הראשתונה של זזה לkrן הראשונה של זווית המוקו. לאחר מכן הפעולה תעבור על שאר הזרויות שמסבב לנקודה, עד שתגיע לזרית BEF, במקורה של זווית ECF. התוכנה תשמור את הזרויות ותחזיר את המערך של הזרויות שעברה עליה בין שני זווית הקצה. בדוגמה זו, הפלט יהיה זה: [∠AEB, ∠BEC, ∠CEF].

```
def disassemble_angle(self, ang):
    """
    Return a list of all the elementary AbsAngles
    that are included in ang
    """
    ang = self.get_better_name_angle(ang)
    sub_angles = self.get_angles_around_point(ang.vertex)
    i = 0
    found = False
    while i < len(sub_angles):
        if sub_angles[i].ray1 == ang.ray1:
            found = True
            break
        i += 1

    if not found:
        raise Exception(
            f"didn't find the ray of the given angle ({ang})"
            f" around the vertex ({ang.vertex.name})"
        )

    res = []
    while sub_angles[i].ray1 != ang.ray2:
        res.append(sub_angles[i])
        i = (i + 1) % len(sub_angles)

    return res
```

פעולות בסיסיות על *AbsSegment*

בדומה לפעולות הבסיסיות על *AbsAngle*, יש צורך בפעולות של פירוק ומציאת קטעי הבסיס. אפרט על הפעולות הללו בקצרה:

- *get_full_seg* – פעולה המקבלת שתי נקודות ומחזירה את ה-*AbsSegment* שמכיל אותן, עם כל נקודות הביניים שנמצאות עליו.
- *is_elementry_seg* – פעולה המקבלת *AbsSegment* ובודקת האם הוא תת-קטע יסודי בשרטוט (כלומר אין לו נקודות ביןיהם).
- *disassemble_seg* – פעולה המקבלת *AbsSegment* ומחזירה רשימה של תתי הקטעים היסודיים המוכלים בו.

המלה מאובייקטי *Abs* לאובייקטי *Real*

לפני שאמוך להסביר על שאר הפעולות, עלי להסביר כיצד אובייקט ה-*Handler* מתיחס לערכימ של אורכי הקטעים וגדלי הזווית השוניים בשרטוט.

כאשר משתמש מבקש מהתוכנה להתחיל בתהליך ההוכחה, נקראות שתי פעולות חשובות באובייקט ה-*Handler*: *init_segments* ו-*init_angles*. הפעולות הללו יוצרות שני מיליוןים בשם *AbsSegments* ו-*RealAngles* בהתאם. התוכנות מייצגת מילון שמקבל אובייקט יסודית מסוג *Abs* ומוחזיר את הייצוג ה-*Real* שלו עם הערך הנוכחי שלו.

במהלך תהליך ההוכחה, התוכנה יוצרת פעמים רבים של אובייקטי *Abs* (למשל זוויות *AbsAngle*), אך בהתייחסות לערכימ קוונטריטים של זוויות או קטעים, התוכנה משתמשת באותו אובייקטי ה-*Real* היסודיים על מנת לשמור על ערכם לאורך ההוכחה.

בשביל לקבל בנוחות את אובייקטים אמייתיים (*RealAngle/RealSegment*) שמרכיבים אובייקטים *Abs* (*AbsAngle/AbsSegment*) כלשהם, שאינם חיבת להיות יסודים, יצרתי מחלקה בשם *Convertor*.

המחלקה *Convertor* (geo/comp/convertor.py)

המחלקה מקבלת הפניה לשתי פעולה:

- *to_list* – פעולה המקבלת אובייקט *Abs* ומוחזירה רשימה של אובייקטים *Abs* יסודים המרכיבים אותו. במקרה שהאובייקט המקורי היה יסודי, הפעולה תחזיר רשימה המכילה איבר יחיד, האובייקט המקורי.
- *get* – פעולה המקבלת אובייקט *Abs* יסודי ומוחזירה את הייצוג שלו כאובייקט *Real*.

בעזרת שתי הפעולות הללו, המחלקה יכולה למש את פעולה `[-]` ולקבל אובייקט *RealAngle/RealSegment* *AbsAngle/AbsSegment* כלשהו ולהחזיר רשימה של ה-*Abs* המיצגים אותו.

```
def __getitem__(self, key):  
    return [self.get(i) for i in self.tolist(key)]
```

(geo/handler.py) Handler

בסוף פועלות האתחול של הזריות/הקטעים באובייקט ה-**Handler**, מאותחלים ה-**Convertor** של הזריות והקטעים כך שנוכל להמיר בנוחות מאובייקט **Abs** לאובייקט **Real**.

לשם כך יצרתי את הפעולות **get** ו-**to_list** לקטעים הזריות במחלקה **Handler**. הנה פירוט קצר על הפעולות הנ"ל:

- **AbsAngle** – פעולה המקבלת **AbsAngle** ומחזירה רשימה של **disassemble_angle** יסודים המרכיבים את הזריות המקוריים.
- **RealAngle** – פעולה המקבל **AbsAngle** יסודי ומחזירה את הייצוג שלה כ-**rangles** באמצעות שימוש שימוש בתוכנה.
- **AbsSegment** – פעולה המפרקת **AbsSegment** לחת הקטעים היסודיים המרכיבים אותו.
- **get_rseg** – פעולה המקבלת **AbsSegment** יסודי ומחזירה את הייצוג שלו כ-**RealSegment** באמצעות שימוש שימוש בתוכנה **rsegments**

אתחול תכונות ה-**Handler** במחלקה **Convertor** נעשה באופן הבא:

```
self.aconv = Convertor(self.disassemble_angle, self.get_rang)
self.sconv = Convertor(self.disassemble_segment, self.get_rseg)
```

לפניהם שאמшиיך לקבוצה נוספת של פעולות במחלקה ה-**Handler**, אציג בפניכם מחלוקת נוספת שיצירתי על מנת להעביר בצורה נוחה נתונים עם מצולעים.

(geo/comp/polygon.py) Polygon

מחלקה מקבלת את רשיימת הקודקודים, הצלעות (**AbsSegment**) והזריות (**AbsAngle**) שמכיל המצלול. בנוסף, המחלוקת מקבלת הפניה ל-**Convertor** שבעזרתו ניתן להמיר את זרויות וצלעות המצלול לאובייקטים מסווג **Real**. המחלוקת מסוגלת להחזיר את הזרית ששיכת לנקודה מסוימת במצלול.

(geo/comp/polygon.py) Quadrilateral ו- Triangle

שתי מחלוקות המייצגות משולשים ומרובעים, הירושות מהמחלקה **Polygon**. הן מממשות שתי פעולות שונות מחלוקת האב שלהן:

- **repr** – כאשר מדפיסים את המשולש או המרובע, נוסף סימן Δ או \square לצד השם שלהם.
- **from_polygon** – ממיר אובייקט **Polygon** לאובייקט **Triangle** או **Quadrilateral**

המשך המחלקה Handler (geo/handler.py)

פעולות מצולעים

הפעולה המרכזית שקשורה למצולעים במחלקה **Handler** היא הפעולה **find_all_polygons**.
הפעולה מקבלת מספר צלעות ומחזירה רשימה של המצולעים הקיימים בשרטוט בעלי מספר
הצלעות הנדרש. הפעולה עצמה מעט ריקה מתוכן והוא מפוקת למספר תת-פעולות מרכזיות.

```
def find_all_polygons(self, num_of_sides):
    """Return list of all polygons with num_of_sides sides"""
    res = []
    for point_list in itertools.combinations(self.points, num_of_sides):
        for perm in self.circle_perm(point_list):
            if self.is_pointlist_poly(list(perm)):
                res.append(self.create_poly_from_pointlist(list(perm)))
    return res
```

הפעולה עוברת על כל הקומבינציות האפשרות של מספר הקודקודים הדרושים למצולע ועל כל סידור אפשרי שלהם. שימו לב שאין משמעות לאיזה נקודה נמצאת ראשונה, אלא רק מה הסדר הפנימי שלהם. הסידורים הללו פשוטים לשינויים של הקבוצה על מעגל, שבו אין אף נקודה שהיא "ראשונה" משום שהמעגל הוא לולאה סגורה. לכן קראתי לפעולה **שמחזירה את הפרמוטציות השונות הללו לקבוצה נתונה**.

כל פרמוטציה, הפעולה בודקת האם היא מצולע לפי הנתונים ואם כן, יוצרת ממנה אובייקט **Polygon** ומיצרת אותה למערך **res** שהיא מחזירה בסוף הפעולה.

כיצד נבדוק האם רשימה של נקודות היא מצולע? בפעולת **is_pointlist_poly** נבדקים בדיקות התנאים הבאים, 3 תנאים ליתר דיוק:

- אם לא כל צלעות המצולע קיימות, המצולע לא יכול להתקיים.
- אם יש 3 נקודות סמוכות שנמצאות על אותו הקטע, המצולע לא יכול להתקיים (כי הוא בעצם עם פחות קודקודים ממש שחיפשנו).
- אם יש שתי צלעות שאינן סמוכות בעלות נקודות חיתוך, המצולע לא יכול להתקיים.

בדיקות תנאים אלו באמצעות קוד:

```
def is_pointlist_poly(self, pointlist):
    """Checks if pointlist is a polygon"""
    sides = [
        AbsSegment(pfrom, pto)
        for pfrom, pto in zip(pointlist, pointlist[1:] + pointlist[:1])
    ]

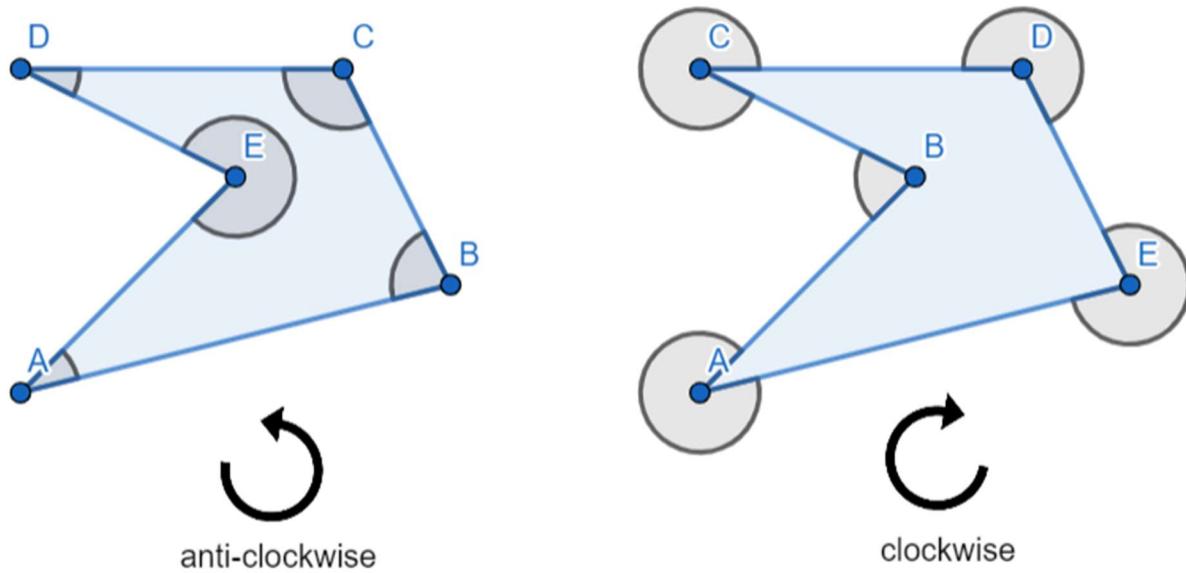
    # if not (all segments exist) -> no poly
    if not all(
        [
            any([seg.is_subsegment(side) for seg in self.segments])
            for side in sides
        ]
    ):
        return False

    # if (3 points on same line) -> no poly

    sides = [self.get_full_seg(seg.start, seg.end) for seg in sides]
    if any(
        [
            [p in seg.get_all_points() for p in pointlist].count(True) > 2
            for seg in self.segments
        ]
    ):
        return False
    # if (non-neighbor sides intersect) -> no poly
    if any(
        [
            side.get_intersection_point(non_adj_side) is not None
            for side_idx, side in enumerate(sides)
            for non_adj_idx, non_adj_side in enumerate(sides)
            if non_adj_idx != side_idx
            and non_adj_idx != (side_idx + 1) % len(sides)
            and non_adj_idx != (side_idx - 1 + len(sides)) % len(sides)
        ]
    ):
        return False
    return True
```

יצירת אובייקט **Polygon** מרשימת נקודות נראהית בתחילת שימושה פשוטה. מוצאים את הزواיות והצלעות ויוצרים את אובייקט המצלול. ובכן, אלו אכן השלבים לייצור אובייקט **Polygon** אבל לצורך מציאת הزواיות, נדרש להגיע למסקנה מעניינת לפני כן.

בהתאם רשימת נקודות, קל מאוד למצוא צלעות המחברות אותן. פשט מוחרים כל שתי נקודות סמוכות. אך על מנת למצוא את צלעות המצלול, علينا להבין אם רשימת הנקודות היא עם כיוון השעון או נגדו. אסביר זאת באמצעות הממחשה ויזואלית:



בציור מסומנות הزواיות: $\angle ABC$, $\angle BCD$, $\angle CDE$, $\angle DEA$, $\angle EAB$.

אם נתיחס לزواיות עם כיוון הרשימה, נקבל בחלוקת מהמרקם (כאשר הנקודות מסודרת עם כיוון השעון) את הزواיות המשילומות ל- -360° ולא את זווית המצלול.

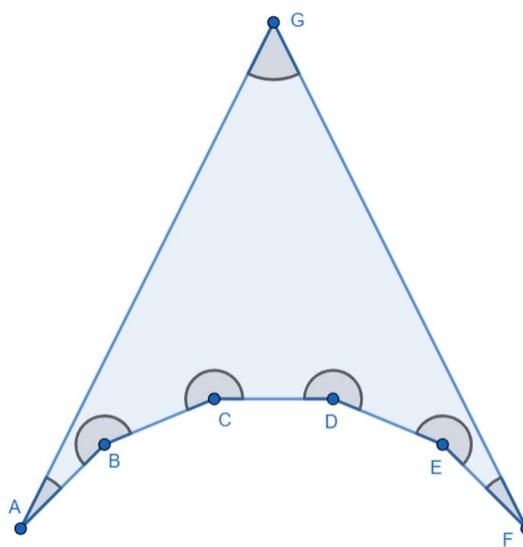
על מנת לפטור את בעיה זו, מצאתי שיטה לכלייה למציאת הزواיות הפנימיות של המצלול. בעת כתיבת ספר פרויקט זה, חשבתי לעומק על שיטה זו והבנתי שהיא עובדת תמיד כאשר מדובר על מצלול בעל 5 קודקודים או פחות, אך לא עובדת תמיד במצלולים בעלי 6 קודקודים ומעלה. לכן, מצאתי שיטה נוספת שעובדת לכל המצלולים. אציג בפניכם את שתי השיטות:

1. השיטה הראשונה: ספירת זווית נישאות

נזכיר את שתי הרשימות האפשרות לزواיות המצלול, אלו שמתיחסות לרשימת הנקודות עם כיוון השעון ואלו שנגד כיוון השעון. לאחר מכן, נספר כמה זווית נישאות יש בכל רשימה ונבחר את הרשימה עם מספר הزواיות הנישאות הקטן יותר, בהנחה שבמצלול יש יותר זווית שאין נישאות מאשר זווית נישאות.

הנחה זו נכונה תמיד עבור מצלעים קמורים ומצלעים בעלי 5 קודקודים ומטה. זאת משומש שמספר הזויות הנישאות המקסימלי שיכולה להיות במלול בעל n קודקודים הוא ($3 - n$). כאשר $6 = n$, יכול להיווצר מצב בו כמה הזויות הנישאות במלול שווה לכמות הזויות שאינן נישאות ובאר $6 > n$ יכול להיווצר מצב בו כמה הזויות הנישאות גדולה מכמות הזויות שאינן נישאות.

ניתן להוכיח זאת באמצעות העובדה שסכום הזויות במלול הוא $(2 - n) \cdot 180$ ולכן יכולות להיות לכל היותר ($3 - n$) זויות שגודלה גדול מ- 180 . על מנת להוכיח שמדובר במקרה באמת יכול להתקיים, אציג דוגמה למלול בעל 7 צלעות ו-4 זויות נישאות. בשיטה דומה ניתן ליצור מצלעים עם מספר צלעות רב יותר עם עד ($3 - n$) זויות נישאות.



2. השיטה השנייה: סכום זויות משוער

למרות שברוב השרטוטים בגיאומטריה אין מצלעים המכילים את השיטה הראשונה, רציתי למצוא שיטה שעבדת תמיד, לכל סוג המצלעים הקמורים והקעורים. לכן, לאחר זמן מה של מחשבה, הגעת לשליטה השנייה, שימושת בהזויות המשוערות, כפי שהן נמצאות בשרטוט. השיטה יוצאת מנקודת הנחה שהמלול אכן מתקיים בשרטוט שהתקבל ולכן נוכל לחשב את סכום הזויות כפי שהוא קיים בשרטוט, ולראות האם הוא אכן מתאים לסכום זויות שאנו אמרוים לקבל.

בשיטת זו, נסuum את סכום הזויות המשוערות של כל קבוצה זויות (קבוצה עם כיוון השעון וקבוצה נגד כיוון השעון). לאחר מכן, נבחר את הקבוצה עם הסכום הקטן ביותר. כאשר חשבתי על כך בתחילת, הייתה לי אינטואיציה חזקה שישיטה זו תעבוד, אך רציתי להוכיח שזה אכן המקרה. ההוכחה פשוטה וمستמכת על העובדה שסכום הזויות הפנימיות במלול בעל n צלעות הוא $(2 - n) \cdot 180$. נסמן את זויות המצלול בעל n הקודקודים באופן הבא:
 a_1, a_2, \dots, a_n

צריך להוכיח:

$$\sum_{i=1}^n a_i < \sum_{i=1}^n (360 - a_i)$$

הוכחה:

$$\sum_{i=1}^n a_i < \sum_{i=1}^n 360 - \sum_{i=1}^n a_i$$

$$2 \cdot ((n-2) \cdot 180) < 360 \cdot n$$
$$360 \cdot n - 360 < 360 \cdot n$$

$$0 < 360 \Rightarrow \sum_{i=1}^n a_i < \sum_{i=1}^n (360 - a_i)$$

```

def create_poly_from_pointlist(self, pointlist):
    """Create Polygon from pointlist (pointlist is a polygon)"""
    sides = [
        self.get_full_seg(pfrom, pto)
        for pfrom, pto in zip(pointlist, pointlist[1:] + pointlist[:1])
    ]
    # choose aangs based on most non-reflex angles
    aangs = [[], []]
    counter = [0, 0]
    for pfrom, vertex, pto in zip(
        pointlist[-1:] + pointlist[:-1],
        pointlist,
        pointlist[1:] + pointlist[:1],
    ):
        from_seg = self.get_full_seg(pfrom, vertex)
        to_seg = self.get_full_seg(vertex, pto)
        aangs[0].append(AbsAngle(from_seg, vertex, to_seg))
        aangs[1].append(AbsAngle(to_seg, vertex, from_seg))

    if len(pointlist) < 6:
        for aang in aangs[0]:
            pfrom = aang.get_start_point()
            vertex = aang.vertex
            pto = aang.get_end_point()
            non_reflex_aang = self.get_non_reflex_angle(pfrom, vertex, pto)
            if non_reflex_aang == aang:
                counter[0] += 1
            else:
                counter[1] += 1
        # polygon with n sides has a maximum of (n-3) reflex angles
        aangs = aangs[argmax(counter)]
    else:
        slope_angle_sums = [0, 0]
        for i in range(2):
            slope_angle_sums[i] = sum(
                [
                    aang.get_angle_size_from_coordinates()
                    for aang in aangs[i]
                ]
            )
        aangs = aangs[argmin(slope_angle_sums)]

    return Polygon(pointlist, sides, aangs, self.aconv, self.sconv)

```

פעולות ההוכחה

משמעות "פעולות ה-main" של המחלקה. נקרא לפעולה זו כאשר נרצה שיבוצעו חישובים והוכחות על פי הנתונים שהזנו למחלקה. הפעולה קוראת לפעולות אתחול אם יש בכך צורך ומאותחלת את מערכ ההוכחות לריק. הפעולה מקבלת פונקציה שתקרה לאחר האתחול (אסביר בהמשך מדוע יש בכך צורך). אחר כך, הפעולה עוברת על המשפטים הידועים לה ו"מפעילה" אותם על הנתונים. לאחר שנערך סיבוב של מעבר על המשפטים בו לא התקבלו נתונים חדשים, הפעולה עוצרת. בנוסף, יש גבול לכמות הסיבובים המקסימלית שהפעולה תבצע. לבסוף, הפעולה מדפיסה את ההוכחה.

פעולות השוויון / מקבילים

כל פעם שהתוכנה מוצאת שוויון כלשהו, נקראות אחת מהפעולות האלו, שמוצאות את השוויון ופעולות על פי במידת הצורך:

- הפעולות `abs_equal_abs` – פעולה שמקבלת שני פרמטרים: `abs1`, `abs2` שווים אחד לשני. ערכי הפרמטרים הם אובייקט `Abs` או רשימה של אובייקטי `Abs`.
- הפעולות `abs_equal_exp` – פעולה שמקבלת שני פרמטרים: `aabs`, `exp` שווים אחד לשני. ערכי הפרמטר `aabs` הוא אובייקט `Abs` או רשימה של אובייקטי `Abs`. ערך הפרמטר `exp` הוא אובייקט מסווג `Length` או `Degree` בהתאם ל-`aabs`.
- הפעולות `real_equal_exp` – פעולה שמקבל שני פרמטרים: `real`, `exp` שווים אחד לשני.
- הפעולה `set_parallel` – פעולה המקבלת שני ישרים וקובעת שהם מקבילים.

פעולות המשפטים

התוכנה תומכת בכמה משפטים, ולכל משפט יש פעולה מיוחד. בפועלה, התוכנה מ Chapman היכן תוכן להפעיל את המשפטים ולגלות נתונים חדשים בעזרתם.

אפרט בקורס על חלק מהמשפטים שהתוכנה יודעת:

- סכום הזווית על ישר הוא 180° . – `angle_sum_on_line`
- שתי זוויות קודקודיות הן שוות. – `vertical_angles`
- סכום הזווית סביב נקודה הוא 360° . – `angle_sum_around_point`
- זווית מתאימות בין שני ישרים מקבילים וחוטף הן שוות, זוג זווית חד צדדיות בין שני ישרים וחוטף הוא 180° . – `angles_on_parallel_lines`
- המשפט הפוך למשפט הקודם. אם מתקיים מצב של זווית מתאימות שווה, מתחלפות שות או זוג זווית חד צדדיות שסכום 180° , בין שני ישרים וחוטף, אז הישרים מקבילים. – `converse_angles_on_parallel_lines`
- סכום הזווית במשולש שווה ל- 180° . – `angle_sum_in_triangle`
- סכום הזווית במרובע שווה ל- 360° . – `angle_sum_in_quadrilateral`
- זווית חיצונית למשולש שווה לסכום שתי הזווית האחרות במשולש. – `exterior_angle_in_triangle`

(geo/helper.py) Helper

המחלקה שבערצה המשמש מכניס את הנתונים לתוכנה, מעין מחלקה מעטפת למחלקה **Handler**. אפרט על הפעולות השונות במחלקה.

פעולות יצילה או החזרה

פעולות שבهن יוצרים אובייקט חדש ומחזירים אותו במידת הצורך. אם האובייקט כבר קיים, הפעולה מחזירה אותו במידת הצורך.

- **d** – מקבלת שם של נקודה וקואורדינטות במידת הצורך, יוצרת את הנקודה אם אינה קיימת ומחזירה אותה.
- **ds** – מקבלת מחרחת של שמות של נקודות ורשימת קואורדינטות במידת הצורך ומחזירה את אוסף הנקודות המתאימות.
- **ps** – מקבלת path לקובץ **ggb** ומחזירה את אוסף הנקודות המוגדר בקובץ (הסביר על כך בהמשך).
- **s** – מקבלת שם או אוסף של שמות של קטעים, יוצרת את הקטע אם אין קיים ומחזירה אותו.
- **a** – פעולה זהה עבור **AbsAngle** יחיד או אוסף.
- **g** – יוצרת אובייקט **Handler** zusätzlich נקודות במידת הצורך שצריך ומחזירה אותו.
- **given** – מקבלת **ds** (מחלקה שירשת מ-**Expression**) ושם של ביטוי, יוצרת אובייקט של ביטוי עם השם הנתון. בנוסף, הפעולה שומרת את האובייקט במערך כך שייתעדכן ערכו במידת הצורך.
- **tri** – מקבלת שם של משולש ויצירת את שלוש הצלעות שלו.
- **poly** – מקבלת שם של מצולע ויצירת את הצלעות שלו.

בארבעת הפעולות הבאות, של ייצור קטעים מיוחדים במשולש, נתקلت בבעיה קטנה. האובייקט **Helper** יוצר אובייקט **Handler** רק כאשר יש קריאה לפעלת **calc** ולכון, אין אפשרות לקבוע נתונים נוספים של שווין או ערכים בשלב ייצור הנתונים. למשל, כאשר המשתמש רוצה ליצור תיקון במשולש, התוכנה תיזור את התיכון אך צריכה להגדיר שהוא חוצה את הצלע שאליו הוא מגיע. אין אפשרות לעשות זאת בשלב הקריאה לפונקציה בגל שאם יתווסף נקודות נוספות לנ נתונים לאחר הקריאה לפונקציה, נדרש לבנות אובייקט **Handler** חדש.

בכדי לפתור את בעיה זו, השתמשתי בפונקציה **partial** מספרית **functools**. פונקציה זו מקבלת הפניה לפונקציה ומשתנים, "מקפיאה" את הקריאה לפונקציה ומחזירה אובייקט **partial**. כאשר נקרא לאובייקט, הקריאה לפונקציה "טופש" והוא תקרה עם הפרמטרים שננתנו לה במקור. בפעולות הבאות הוסיףתי למערך את אובייקטי ה-**partial** שמתפלים בתוכנות המיעודות של הקטעים במשולש וקרأتي להם לאחר אתחול אובייקט ה-**Handler**.

- **tri_med** – יוצר תיקון במשולש.
- **tri_angbi** – יוצר חוצה זווית במשולש.
- **tri_alt** – יוצר גובה במשולש.
- **tri_segbi** – יוצר אנך אמצעי במשולש.

פעולות אתחול והוכחה (חישוב)

פעולות אתחול הזריות והקטעים – `init_angles`, `inita`, `inits`, `init_segments` ב奧ビיקט `Handler` במידת הצורך. שתי הפעולות הללו לא נקראות עלי ידי המשמש אלא נקראות ע"י הפונקציה `.calc`.

הfonקציה `init_to` שקוראת לכל אובייקטי ה-`partial` ש"הוקפאו" במהלך תהליך חישוב הנטונים. גם פעולה זו נקראת ע"י הפונקציה `calc`.

הfonקציה `calc` מתחילה את הקטעים והזריות, וקוראת בתורה לפונקציה `calc` באובייקט `Handler`. תוך העברת הפניה לפונקציה `to_init` שתקרה לאחר מכן בתוך האובייקט `Handler`. המשמש יקרה לפונקציה `calc` כאשר ירצה ליצור את ההוכחה שלו וליצור את חישובי הזריות.

```
def calc(self, print_proof=True, use_theorems=None):
    """Call self.geo.calc() & perform init"""
    self.inita()
    self.inits()
    self.g().calc(False, False, print_proof, self.to_init, use_theorems)
    self.did_calc = True
```

פעולות Set/Get/Equal

הfonקציה `seta` מקבלת שם של זווית וערך של ביטוי מסוג `Degree` וקוראת לפונקציה `abs_equal_exp` במחלקה `Handler` (באמצעות `partial`, כלומר הפעולה "מקפיה" את הקריאה עד לאחר אתחול הזריות והקטעים של אובייקט-`Handler`). הפעולה `sets` עשויה דבר דומה עבור קטע ואורך.

הפעולה `geta` מקבלת שם או שמות של זווית ומחזירה את הערך שלהן לאחר ההוכחה (וקוראת לפעולה `calc` במידת הצורך). הפעולה `gets` מבצעת דבר דומה עבור שם או שמות של קטעים.

הפעולות `equals` ו-`equala` מקבלות שתי שמות ומשוואות בין אובייקטי ה-`Abs` אותם הם מייצגים, זוויות וקטעים בהתאם. זאת באמצעות אובייקט `partial` שיקרא לאחר אתחול הזריות והקטעים ב-`Handler.calc()`.

פעולות קטעים

- מקבלת שם של קטע מקורי וקטע חדש ו"המשך" את הקטע המקורי לקטע החדש ו/או מוסיפה נקודות ביניים לפי הצורך.
- מקבלת שתי שמות של קטעים וקובעת שהם מאונכים (פעולת באופן דומה ל-`perps` עם אובייקט `partial`).
- מקבלת מספר שמות של קטעים וקובעת שהקטעים מקיימים זה זה. פעולה זו פועלת באופן מיידי ולא באמצעות אובייקט `partial` מכיוון שהיא יכולה לפעול על קטעים מסוג `AbsSegment`.
- מקבלת שתי שמות של קטעים, קוראת לפעולה `calc` במידת הצורך ומחזירה האם הקטעים מקבילים.

(geo/problemcollection.py) ProblemCollection

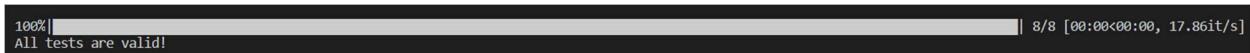
בשביל לAGED בצורה נוחה אוסף של תרגילים ובעיות, יצרתי מחלקה שניית לרשות ממנה על מנת למשוך אוסף תרגילים. המחלקה בנוייה כך שכל תרגיל הינו פועלה סטטית במחלקה, המחזירה `tuple` של שני `iterables`. הראשון מכיל את התשובות שהתוכנה נתנה והשני מכיל את התשובה הנכונה.

המחלקה פועלת באמצעות שלושת הפעולות הבאות:

- `check_prob` – פעולה סטטית (`@staticmethod`) המקבלת הפניה לפונקציה ובודקת אם היא מחזירה תשובה נכונה, כלומר האם האיבר הראשון בערך ההצהרה (רשימה של התשובות שהחזרה הפונקציה) שווה לאיבר השני (רשימת התשובות הנכונה).
- `all` – פעולה המחלקה (`@classmethod`) שמחזירה רשימה של הפניות לפונקציות של הביעות. הפעולה בודקת את כל הפעולות שמוגדרות במחלקה ומחזירה את כל אלו שאינם מתחילה ב-`_` ואינם שלושת הפעולות שנמצאות במחלקה המקורית.
- `check_all` – פעולה המחלקה (`@classmethod`) הבודקת את כל הביעות שנמצאות באוסף הביעות. הפעולה מדפסה progress bar של הביעות שנבדקו.

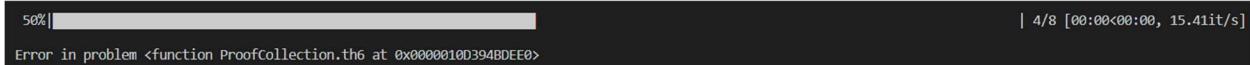
בפרויקט שלי יצרתי שני אוסףים של תרגילים: **YoelGevaProblems** (בעיות מספר יואל גבע) ו-**ProofCollection** (אוסף של הוכחות המשפטים).

הנה דוגמה לפולט מבדיקת `check_all` על **ProofCollection**:



```
100%|██████████| 8/8 [00:00<00:00, 17.86it/s]
All tests are valid!
```

במידה ויש תקליה באחת הביעות:



```
50%|██████████| 4/8 [00:00<00:00, 15.41it/s]
Error in problem <function ProofCollection.th6 at 0x0000010D394BDEE0>
```

נספחים

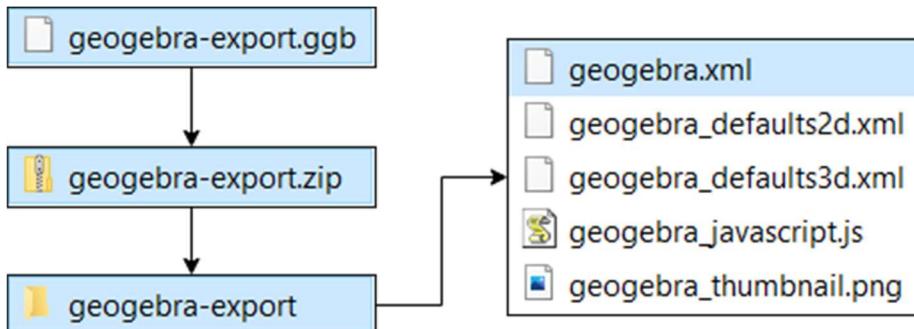
הסבר על קבצי ggb

במודל החדש, המשמש ציריך להזין את אוסף הנקודות והקואורדינטות שלהן. בשביל למצוא את הנקודות במערכת הצירים העתקתי את השרטוטים אל אתר "GeoGebra" שמקיל ממשק נוח לבניית שרטוטים בגיאומטריה. מכמה הפעמים הראשונות עשית זאת באופן יידי, אך הבנתי מהר מאד ששיטה זו לא תהיה נוחה למשתמש כלל וכלל.

חיפשתי דרך ליצא את רשימת הנקודות מהאתר אל קובץ במחשב או אל ה-console בדרכים שונות, אך ללא הצלחה. המשמש ציריך להעביר רק את רשימת הנקודות, משום שאת שאר הנתונים הוא יכנס באמצעות ה-API של התוכנה (כלומר, הפעולות במחalkerת Helper).

לאחר כמה חיפושים בגוגל וניסיונות רבים באתר, הבנתי שאין דרך ליצא רק את רשימת הנקודות מהאתר וכן עברתי לחפש דרך ליצא את השרטוט כולו מהאתר. הדרך הייתה נראה לי הגיונית ביותר הייתה ".GeoGebra file (.ggb)".

בשלב הבא, ניסיתי "לרוורס" את הקובץ ולהבין כיצד הנתונים נשמרים בקובץ. ראשית, גיליתי שהקובץ הוא בעצם קובץ zip בעל סימות שונה, אך שנייתן החלץ ממנו כמה קבצים. לאחר שעשית זאת, קיבלתי תקיה מלאה בכמה קבצים וביניהם הקובץ "geogebra.xml".



קובץ xml הם קבצים שמייצגים נתונים באמצעות תגיוט וקובצי טקסט (באופן דומה לשפת התכנות HTML). לאחר עיון קצר בקובץ xml הבנתי כיצד "GeoGebra" שומר נתונים על נקודות בקובץ: באמצעות תגיות <element type="point">.

```

<element type="point" label="A">
  <show object="true" label="true"/>
  <objColor r="21" g="101" b="192" alpha="0"/>
  <layer val="0"/>
  <labelMode val="0"/>
  <animation step="0.1" speed="1" type="1" playing="false"/>
  <coords x="0" y="4" z="1"/>
  <pointSize val="5"/>
  <pointStyle val="0"/>
</element>
  
```

הקוואורדינטות של הנקודות נשמרות בין שתי תגיות ה-`element` בtaggit `<coords>` שמכילה מידע על ה-`x` וה-`y` של הנקודות.

באמצעות הידע זהה, כתבתי את הפונקציה שמקבלת קובץ **ggb** ומחזירה את רשימת הנקודות הקיימות בשרטוט.

```

def get_points_from_file(path, temp_folder_path="temp_ggb"):
    """Return a list of (name,x,y) from a .ggb file"""
    # check file extension
    if not path.endswith(".ggb"):
        raise Exception(
            f"file type for file {path} is not supported."
            f" use .ggb instead"
        )
    # extract files from ggb file (actually zip)
    with zipfile.ZipFile(path, "r") as zip_ref:
        zip_ref.extractall(temp_folder_path)
    # read main file from unzipped folder
    with open(os.path.join(temp_folder_path, "geogebra.xml"), "r") as f:
        lines = f.readlines()
    # create a list of point tags
    point_tags = []
    is_inside_point_tag = False
    for line in lines:
        line = " ".join(line.strip().split())
        if line.startswith('<element type="point">'):
            is_inside_point_tag = True
            point_tags.append("")
        if is_inside_point_tag:
            point_tags[-1] += line + "\n"
        if line.startswith("</element>"):
            is_inside_point_tag = False
    # create list of points from tags
    points = []
    # points[i] = (name,x,y)
    for point_tag in point_tags:
        name = point_tag.split("\n")[0].split('label="')[1].split('"')[0]
        x, y = None, None
        for line in point_tag.split("\n"):
            if line.startswith("<coords"):
                x = float(line.split('x="')[1].split('"')[0])
                y = float(line.split('y="')[1].split('"')[0])
                break
        points.append((name, x, y))
    # remove temp_folder
    for filename in os.listdir(temp_folder_path):
        os.remove(os.path.join(temp_folder_path, filename))
    os.rmdir(temp_folder_path)
    return sorted(points) # return the sorted points' list

```

הסבר על UnionFind

הינו מבנה נתונים (אלגוריתם) שמבצע מעקב על קבוצה של עצמים המחלקים למספר של תת-קבוצות זרות. האלגוריתם תומך בשתי פעולות (כפי שרמז שמו):

- חיפוש (find) – קביעה איזו קבוצה מכילה איבר מסוים. ע"י קריאה לפעולה זו על שני איברים שונים ניתן לבדוק אם שניהם נמצאים באותו הקבוצה.
- איחוד (union) – איחוד שתי קבוצות לכדי קבוצה אחת.

על מנת להגדיר את פעולות אלו, ניתן לכל איבר שתי תכונות:

- הורה (self.par) – נציג הקבוצה שבו נמצאת האיבר. על מנת לקבל את הנציג הכלול של הקבוצה, נמierz בחיפוש רקורסיבי להורה של ההורה וכן הלאה עד שנגיע לאיבר שהו הורה של עצמו.
- דרגה (self.rank) – חסם עליון על גובה ה"ע"ץ" של שרשרת ההורים של הקבוצה. כאשר מתחלים איבר, הדרגה שלו היא 0.

```
class UnionFind:  
    def __init__(self):  
        """Init self to a set by itself"""  
        self.par = self  
        self.rank = 0  
  
    def find(self):  
        """Return the root of self and do path compression"""  
        if self.par != self:  
            self.par = self.par.find()  
        return self.par  
  
    def union(self, other):  
        """Union the sets of self and other (by rank)"""  
        x, y = self.find(), other.find()  
        if x == y:  
            return  
        if x.rank < y.rank:  
            x, y = y, x  
        # now x.rank >= y.rank  
        y.par = x  
        if x.rank == y.rank:  
            x.rank += 1  
  
    def is_same(self, other):  
        """Check if self and other are in the same set"""  
        return self.find() == other.find()
```

הסיבוכיות של אלגוריתם כזה עם n פעולות ייצור סט ו- m פעולות [union/find](#) היא $(n\alpha \cdot m)O$ כך שהסיבוכיות לשיעורין (הסיבוכיות המשווקלתת חלקית כמויות הפעולות) של כל פעולה היא $(n\alpha)O$ כאשר α היא פונקציית אקרמן ההופכית, פונקציה שגדלה מאד מאוד לאט וסקולה (במיוחד במקרה של תרגילי הגיאומטריה שלנו) ל- $O(1)$.

* הפונקציה $(n\alpha)$ תחזיר 4 כאשר n הוא בסדר גודל של $2^{2^{2^{16}}}$.

אלגוריתם [UnionFind](#) הינו חזק הרבה יותר מהדרוש במקרה שלנו, אך רציתי למשמש אותו על מנת ליצור פתרון אלגנטי ויפה לבדיקת ישרים מקבילים.

הוכחות למשפטים באמצעות האקסיומות

משפט

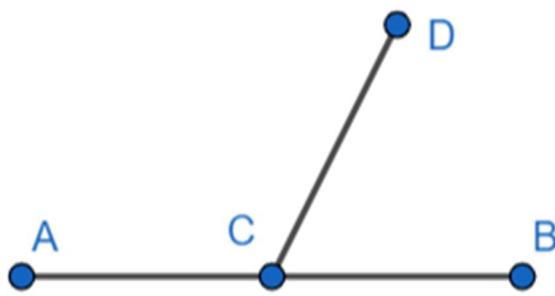
בhocחות אלו אתייחס לאקסיומות הבאות (לפי המספר שלהן):

1. כל חזויות השטוחות הן זרות.
2. השלם שווה לסכום חלקיו.
3. זוג חזויות מתחלפות בין שני ישרים מקבילים וחוטן הן שוות.

משפט 1: סכום חזויות על ישר הוא 180°

נתון: ישר AB , נקודה C על השר. ישר CD .

הוכחה: סכום חזויות על השר ACB שווה 180° .



הוכחה:

$$\angle ACB = 180^\circ \text{ (אקסיממה 1)}$$

$$\angle ACB = \angle ACD + \angle DCB \text{ (אקסיממה 2)}$$

$$\angle ACD + \angle DCB = 180^\circ \text{ (כלל המעבר)}$$

מ.ש.ל

לעט:

```
def th1(print_proof=False):
    """_ax1 & _ax2 -> _th1"""
    h = Helper()
    h.ps_from_file(r"ggb_files\th1.ggb")
    h.s("ACB", "CD")
    h.calc(print_proof, use_theorems=[ ])
    # by ax1, ∠ACB = 180°
    ACB = h.a("ACB")
    # by ax2, ∠ACB is the sum of its parts
    parts = h.g().disassemble_angle(ACB)
    # therefore, we know sum(parts) = 180
    if print_proof:
        print(f"∠ACB = 180, {ACB} = {' + '.join(map(str,parts))}")
        print(f"\n")
        print(f"{' + '.join(map(str,parts))} = 180")

    return None, None
```

פלט:

```
∠ACB = 180, ∠ACB = ∠ACD + ∠DCB
↓
∠ACD + ∠DCB = 180
```

משפט 2: זוג זוויות קודקודיות הן שוות

נתון: ישרים AB ו-CD. נקודות החיתוך שלהם היא E.

הוכחה: $\angle AEC = \angle BED$

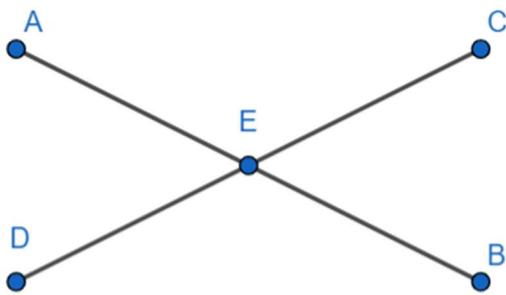
הוכחה:

$$(1) \angle CEB = 180 - \angle AEC$$

$$(1) \angle BED = 180 - \angle CEB$$

לכן, $\angle AEC = \angle BED$ (ככל המעביר)

מ.ש.



```

def th2(print_proof=False):
    """_th1 -> _th2"""
    h = Helper()
    h.ps_from_file(r"ggb_files\th2.ggb")
    h.s("AEB", "CED")
    h.calc(print_proof, use_theorems=[1])
    if print_proof:
        print(
            f"\{h.a('AEC')\} = \{h.get_a('AEC')\}, \{h.a('BED')\} = \{h.get_a('BED')\}"
        )
        print(f"\Downarrow")
        print(f"\{h.a('AEC')\} = \{h.a('BED')\}")
    return (h.get_a("AEC") == h.get_a("BED"),), (True,)

```

```

180 = ∠AEB = ∠AEC + ∠CEB
(angle upon line AB is 180, the whole is the sum of its parts)
∠CEB = 180 - ∠AEC (same)
∠CEB = 180 -γ (eval) -> ∠CEB = -γ +180 (calc)
---

180 = ∠BEA = ∠BED + ∠DEA
(angle upon line AB is 180, the whole is the sum of its parts)
∠DEA = 180 - ∠BED (same)
∠DEA = 180 -α (eval) -> ∠DEA = -α +180 (calc)
---

180 = ∠CED = ∠CEB + ∠BED
(angle upon line CD is 180, the whole is the sum of its parts)
∠CEB = 180 - ∠BED (same)
-γ +180 = 180 -α (eval)
-γ +180 = -α +180 (same)
γ = α (found var γ)
∠CEB = -γ +180 = -(α) +180 -> ∠CEB = -α +180
∠AEC = γ = (α) -> ∠AEC = α

∠AEC = α, ∠BED = α
↓
∠AEC = ∠BED
((True,), (True,))

```

משפט 3: סכום הזווית סביב נקודה הוא 360°

נתון: ישרים AB ו-CD. נקודת החיתוך שלהם היא E (זהה לנוטונים של משפט 2).

הוכחה: סכום הזווית סביב נקודה E שווה ל- 360° .

הוכחה:

$$(1) \angle CEB = 180 - \angle AEC$$

$$(1) \angle DEA = 180 - \angle BED$$

$$\begin{aligned} \angle AEC + \angle CEB + \angle BED + \angle DEA &= \angle AEC + (180 - \angle AEC) + \angle BED + (180 - \angle BED) \\ &= 360^\circ \end{aligned}$$

לכן, סכום הזווית סביב נקודה E שווה ל- 360° .

מ.ש.ל

:
קוד:

```
def th3(print_proof=False):
    """_th1 -> _th3"""
    h = Helper()
    h.ps_from_file(r"ggb_files\th3.ggb")
    h.s("AEB", "CED")
    h.calc(print_proof, use_theorems=[1])
    E = h.p("E")
    aangs_around_E = h.g().get_angles_around_point(E)
    sum_aangs_around_E = sum(h.get_a(*aangs_around_E))
    if print_proof:
        print(
            " + ".join(map(str, aangs_around_E)),
            "=",
            f"({'} + ('.join(map(str, h.get_a(*aangs_around_E))))})",
            "=",
            sum_aangs_around_E,
        )
        print("↓")
        print(" + ".join(map(str, aangs_around_E)), "=", sum_aangs_around_E)

    return (sum_aangs_around_E,), (360,)
```

פלט:

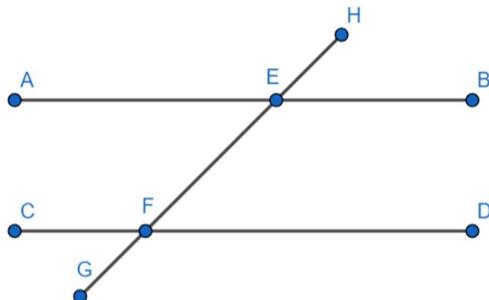
$180 = \angle AEB = \angle AEC + \angle CEB$
 (angle upon line AB is 180, the whole is the sum of its parts)
 $\angle CEB = 180 - \angle AEC$ (same)
 $\angle CEB = 180 - \gamma$ (eval) $\rightarrow \angle CEB = -\gamma + 180$ (calc)

 $180 = \angle BEA = \angle BED + \angle DEA$
 (angle upon line AB is 180, the whole is the sum of its parts)
 $\angle DEA = 180 - \angle BED$ (same)
 $\angle DEA = 180 - \alpha$ (eval) $\rightarrow \angle DEA = -\alpha + 180$ (calc)

 $180 = \angle CED = \angle CEB + \angle BED$
 (angle upon line CD is 180, the whole is the sum of its parts)
 $\angle CEB = 180 - \angle BED$ (same)
 $-\gamma + 180 = 180 - \alpha$ (eval)
 $-\gamma + 180 = -\alpha + 180$ (same)
 $\gamma = \alpha$ (found var γ)
 $\angle CEB = -\gamma + 180 = -(\alpha) + 180 \rightarrow \angle CEB = -\alpha + 180$
 $\angle AEC = \gamma = (\alpha) \rightarrow \angle AEC = \alpha$

 $\angle BED + \angle DEA + \angle AEC + \angle CEB = (\alpha) + (-\alpha + 180) + (\alpha) + (-\alpha + 180) = 360$
 \Downarrow
 $\angle BED + \angle DEA + \angle AEC + \angle CEB = 360$
 $((<360>,), (360,))$

משפט 4. זוג זוויות מתאימות הן שווות,זוג זוויות חד צדדיות סכומן הוא 180° (בין שני ישרים מקבילים וחותך)



נתון: AB מקביל ל-CD, HG חותך. נקודות E,F,G נקודות החיתוך של HG עם הישרים CD,AB בהתאם.

הוכחה: $\angle GEA = \angle GFC$ (זוג זוויות מתאימות) וגם $\angle BEG + \angle HFD = 180^\circ$ (זוג זוויות חד צדדיות).

הוכחה:

(אקסימוה 3 מקבילים, HG חותך) $\angle GEA = \angle HFD$

(חוויות קודקודיות) $\angle HFD = \angle GFC$

לכן, $\angle GEA = \angle GFC$ (כלל המעבר)

(אקסימוה 3 מקבילים, HG חותך) $\angle BEG = \angle CFH$

(CD סכום זוויות על ישר) $180^\circ = \angle CFD + \angle HFD$

לכן, $180^\circ = \angle BEG + \angle HFD$

מ.ש.ל

קיד:

```
def th4(print_proof=False):
    """_th1 & _ax3 -> _th4"""
    h = Helper()
    h.ps_from_file(r"ggb_files\th4.ggb")
    h.s("AEB", "CFD", "HEFG")
    h.paras("AB", "CD")
    # ax3 - set  $\angle GEA = \angle HFD$ 
    # (alternate interior angles, AB || CD, HEFG traversal)
    h.equals("GEA", "HFD")
    h.calc(print_proof, use_theorems=[1, 2, 3])
    # Corresponding angles example (GEA & GFC)
    corresponding = h.get_angle("GEA") == h.get_angle("GFC")
    # Consecutive interior angles example (BEG & HFD)
    consecutive = h.get_angle("BEG") + h.get_angle("HFD") == 180
    if print_proof:
        print(
            f"\{h.angle('GEA')\} = \{h.get_angle('GEA')\}, \{h.angle('GFC')\} = \{h.get_angle('GFC')\}"
        )
        print("↓")
        print(f"\{h.angle('GEA')\} = \{h.angle('GFC')\} (corresponding)")
        print()
        print(
            f"\{h.angle('BEG')\} + \{h.angle('HFD')\} = (\{h.get_angle('BEG')\})"
            f" + (\{h.get_angle('HFD')\})"
            f" = \{sum(h.get_angle('BEG'), 'HFD'))\}"
        )
        print("↓")
        print(f"\{h.angle('BEG')\} + \{h.angle('HFD')\} = 180 (consecutive)")

    return (corresponding, consecutive), (True, True)
```

```

 $\angle GEA = \angle HFD$  (given)
 $\angle HFD = \alpha$  (eval)
 $\angle HFD = \alpha$  (given)
---
 $180 = \angle AEB = \angle AEH + \angle HEB$ 
(angle upon line AB is 180, the whole is the sum of its parts)
 $\angle HEB = 180 - \angle AEH$  (same)
 $\angle HEB = 180 - \beta$  (eval) ->  $\angle HEB = -\beta + 180$  (calc)
---
 $180 = \angle BEA = \angle BEG + \angle GEA$ 
(angle upon line AB is 180, the whole is the sum of its parts)
 $\angle BEG = 180 - \angle GEA$  (same)
 $\angle BEG = 180 - \alpha$  (eval) ->  $\angle BEG = -\alpha + 180$  (calc)
---
 $180 = \angle CFD = \angle CFH + \angle HFD$ 
(angle upon line CD is 180, the whole is the sum of its parts)
 $\angle CFH = 180 - \angle HFD$  (same)
 $\angle CFH = 180 - \alpha$  (eval) ->  $\angle CFH = -\alpha + 180$  (calc)
---
 $180 = \angle DFC = \angle DFG + \angle GFC$ 
(angle upon line CD is 180, the whole is the sum of its parts)
 $\angle DFG = 180 - \angle GFC$  (same)
 $\angle DFG = 180 - \varepsilon$  (eval) ->  $\angle DFG = -\varepsilon + 180$  (calc)
---
 $180 = \angle HEG = \angle HEB + \angle BEG$ 
(angle upon line HG is 180, the whole is the sum of its parts)
 $\angle HEB = 180 - \angle BEG$  (same)
 $-\beta + 180 = 180 - (-\alpha + 180)$  (eval)
 $-\beta + 180 = \alpha$  (same)
 $\beta = -\alpha + 180$  (found var  $\beta$ )
 $\angle HEB = -\beta + 180 = -(-\alpha + 180) + 180$  ->  $\angle HEB = \alpha$ 
 $\angle AEH = \beta = (-\alpha + 180)$  ->  $\angle AEH = -\alpha + 180$ 
---
 $180 = \angle HFG = \angle HFD + \angle DFG$ 
(angle upon line HG is 180, the whole is the sum of its parts)
 $\angle DFG = 180 - \angle HFD$  (same)
 $-\varepsilon + 180 = 180 - \alpha$  (eval)
 $-\varepsilon + 180 = -\alpha + 180$  (same)
 $\varepsilon = \alpha$  (found var  $\varepsilon$ )
 $\angle DFG = -\varepsilon + 180 = -(\alpha) + 180$  ->  $\angle DFG = -\alpha + 180$ 
 $\angle GFC = \varepsilon = (\alpha)$  ->  $\angle GFC = \alpha$ 

```

$$\angle GEA = \alpha, \angle GFC = \alpha$$

↓

$$\angle GEA = \angle GFC \text{ (corresponding)}$$

$$\angle BEG + \angle HFD = (-\alpha + 180) + (\alpha) = 180$$

↓

$$\angle BEG + \angle HFD = 180 \text{ (consecutive)}$$

((True, True), (True, True))

משפט 5: משפט הfork לזוויות בין ישרים מקבילים (זוויות מתחלפות, מתאימות וחד צדיות)

נתון: שני ישרים AB ו-CD. נקודות E,F הן נקודות החיתוך של ישר HG עם השרים AB, CD, בהסתמך.

א. **נתון:** $\angle ABG = \angle CFE$ (זוויות מתחלפות שוות). **הוכחה:**

ב. **נתון:** $\angle HEB = \angle HFD$ (זוויות מתאימות שוות). **הוכחה:**

ג. **נתון:** $\angle BEG + \angle HFD = 180^\circ$ (זוג זוויות חד צדיות שסכוםן הוא 180°). **הוכחה:**

הוכחה:

א. נניח בשלילה ש-AB אינו מקביל ל-CD. לכן, חייב להיות ישר אחר המקביל ל-CD וועבר דרך נקודה E. נקרא לישר זה I.

$$\angle IEG = \angle CFH \text{ (זוויות מתאימות בין IJ ו-CD, חותך HG)}$$

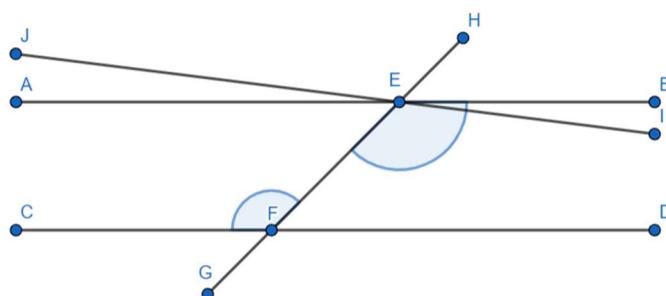
$$\angle ABG = \angle CFH \text{ (כל המעבר, נתון)} \quad \angle ABG = \angle IEG$$

$$\angle BEI + \angle IEG = \angle BEG = \angle BEI + \angle IEG = IEG \text{ (השלם שווה לסכום חלקיו)}$$

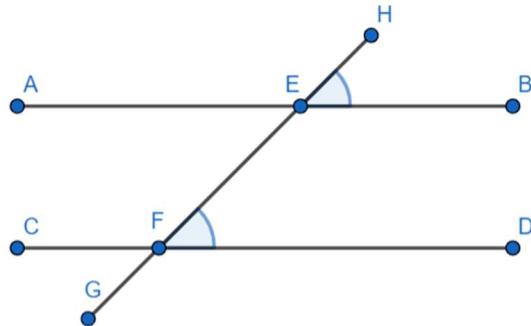
לכן, $0 = IEG$ ולכן הוא בעצם AB. לפיכך ההנחה שלנו היא שגויה.

$$AB \parallel CD$$

מ.ש.ל (א)



.ב.

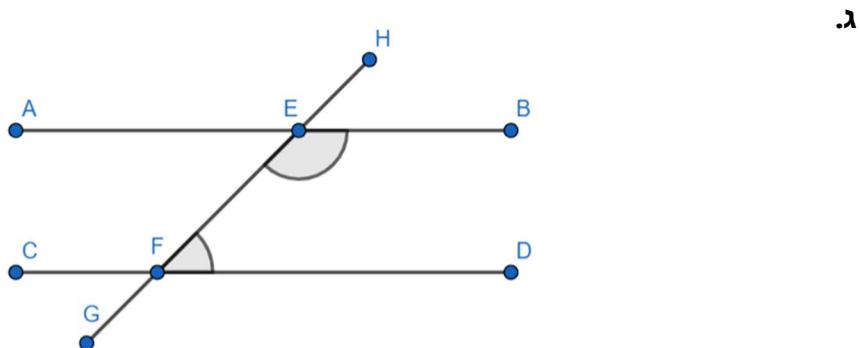


$$\angle GEA = \angle HEB \text{ (זווית קודקודיות)}$$

$$(\angle HEB = \angle HFD = \angle GEA = \angle GFC \text{ (כל המעבר, נתון})$$

לכן, $CD \parallel AB$ (לפי משפט 5א, אם קיימ זוג זוויות מתחלפות שוות בין שני ישרים וחותם, אז הישרים מקבילים).

מ.ש.ל (ב)



$$\angle GEA = \angle BEG \text{ (סכום זווית על ישר } AB = 180 - \angle BEG)$$

$$(\angle HFD = 180 - \angle BEG = \angle HED = \angle GEA = \angle HFD \text{ (כל המעבר, נתון})$$

לכן, $CD \parallel AB$ (לפי משפט 5א, אם קיימ זוג זוויות מתחלפות שוות בין שני ישרים וחותם, אז הישרים מקבילים).

מ.ש.ל (ג)

```

def th5(print_proof=False):
    """_ax3 + _th4 -> _th5 (proof by contradiction)
    3 part Proof:
    1. Converse alternate interior angles (proof by contradiction)
    2. Converse corresponding angles (proof based on 1)
    3. Converse consecutive interior angles (proof based on 1)
    """
    ans = [[], []]

    """ 1. Prove Converse Alternate Interior angles """
    # We Need to prove that AB || CD
    # based on equal alternate interior angles
    h = Helper()
    h.ps_from_file(r"ggb_files\th5_1.ggb")
    h.s("AEB", "CFD", "HEFG")
    # Assume CD is not parallel to AB
    # Therefore, there must be a different
    # parallel to CD that goes through E
    # That parallel is IJ
    h.s("IEJ")
    h.paras("IJ", "CD")
    # It is given that BEG == CFH
    h.equala("BEG", "CFH")
    # Let's calc
    h.calc(print_proof, use_theorems=[1, 2, 3, 4])
    # We get that BEJ is zero
    ans[0].append(h.get("BEJ") == 0)
    ans[1].append(True)
    # Therefore IJ is the same as AB
    # And therefore our assumption that AB is not parallel to CD was wrong
    # Thus, AB || CD
    if print_proof:
        print(f"\{h.a('BEJ')\} = {h.get('BEJ')}"))
        print(f"\\")
        print(f"(IJ is the same as CD) -> AB || CD")

    print("\n")

```

```

""" 2. Prove Converse equal corresponding angles """
# We need to prove that AB || CD, based on equal corresponding angles
h = Helper()
h.ps_from_file(r"ggb_files\th5_2.ggb")
h.s("AEB", "CFD", "HEFG")
# it is given that HEB == HFD
h.equala("HEB", "HFD")
# lets see if GEA == HFD
h.calc(print_proof, use_theorems=[1, 2, 3, 4])
ans[0].append(h.get("GEA") == h.get("HFD"))
ans[1].append(True)
# because they are a pair of equal
# alternate interior angles, AB || CD (as was proven in 1)
if print_proof:
    print(
        f"\n{h.a('GEA')} = {h.get('GEA')} = {h.get('HFD')}"
        f" = {h.a('HFD')}"
    )
    print(f"\n↓")
    print(
        f"AB || CD (Converse alternate interior angles"
        f" - proven before)"
    )
    print("\n")

```

```

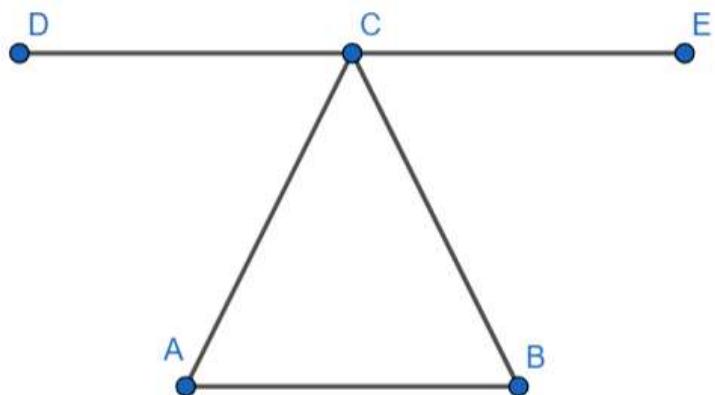
""" 3. Prove converse consecutive interior angles """
# We need to prove that AB || CD, based on consecutive interior angles
h = Helper()
h.ps_from_file(r"ggb_files\th5_3.ggb")
h.s("AEB", "CFD", "HEFG")
# it is given that BEG + HFD == 180
x = h.given(Degree, "x")
h.seta("BEG", x)
h.seta("HFD", 180 - x)
# lets see if GEA == HFD
h.calc(print_proof, use_theorems=[1, 2, 3, 4])
ans[0].append(h.get("GEA") == h.get("HFD"))
ans[1].append(True)
# because they are a pair of equal
# alternate interior angles, AB || CD (as was proven in 1)
if print_proof:
    print(
        f"\n{h.a('GEA')} = {h.get('GEA')} = {h.get('HFD')}"
        f"\n= {h.a('HFD')}"
    )
    print(f"\n")
    print(
        f"\nAB || CD (Converse alternate interior angles"
        f"\n- proven before)"
    )
return tuple(ans[0]), tuple(ans[1])

```

משפט 6: סכום הזווית במשולש הוא 180°

נתון: משולש ABC, ישר DE, מקביל ל-AB העובר דרך נקודה C.

הוכחה: סכום הזווית במשולש ABC שווה ל- 180° .



הוכחה:

$$(\text{זווית מתחלפות בין } AC\text{-}DE \text{ ו-} AB, \text{ חותך } AC = \angle CAB)$$

$$(\text{זווית מתחלפות בין } BC\text{-}DE \text{ ו-} AB, \text{ חותך } BC = \angle ABC)$$

$$(DE \text{ זווית על ישר } DE = \angle ECB + \angle BCA + \angle ACD = 180^\circ)$$

$$(\text{כלל המעבר } \angle ABC + \angle BCA + \angle CAB = 180^\circ)$$

לכן, סכום הזווית במשולש ABC שווה ל- 180° .

מ.ש.

```

def th6(print_proof=False):
    """_ax3 + _th1 -> _th6"""
    h = Helper()
    h.ps_from_file(r"ggb_files\th6.ggb")
    h.tri("ABC")
    h.s("DCE")
    h.paras("AB", "DE")
    h.calc(print_proof, use_theorems=[1, 2, 3, 4, 5])
    if print_proof:
        print(
            f"\{h.a('CAB')\} + \{h.a('ABC')\} + \{h.a('BCA')\} = "
            f"\{h.get_a('CAB')\} + (\{h.get_a('ABC')\}) + (\{h.get_a('BCA')\})",
            f"\sum(h.get_a('CAB', 'ABC', 'BCA'))",
        )
        print(f"\Downarrow")
        print(f"sum of angles in \triangle ABC is 180°")

    return (sum(h.get_a("CAB", "ABC", "BCA")),), (180,)

```

$$180 = \angle ECD = \angle ECB + \angle BCA + \angle ACD$$

(angle upon line DE is 180, the whole is the sum of its parts)

$$\angle ECB = 180 - (\angle BCA + \angle ACD) \text{ (same)}$$

$$\angle ECB = 180 - (\varepsilon + \zeta) \text{ (eval)} \rightarrow \angle ECB = -\zeta - \varepsilon + 180 \text{ (calc)}$$

$$360 = \angle CAB + \angle BAC = \angle CAB + \angle BAC$$

(sum of angles around point A, the whole is the sum of its parts)

$$\angle BAC = 360 - \angle CAB \text{ (same)}$$

$$\angle BAC = 360 - \alpha \text{ (eval)} \rightarrow \angle BAC = -\alpha + 360 \text{ (calc)}$$

$$360 = \angle ABC + \angle CBA = \angle ABC + \angle CBA$$

(sum of angles around point B, the whole is the sum of its parts)

$$\angle CBA = 360 - \angle ABC \text{ (same)}$$

$$\angle CBA = 360 - \gamma \text{ (eval)} \rightarrow \angle CBA = -\gamma + 360 \text{ (calc)}$$

$$\angle ACD = \angle CAB \text{ (alternating angles are equal between } DE \parallel AB \text{ and } CA)$$

$$\angle ACD = \alpha \text{ (eval)}$$

$$\angle ACD = \alpha \text{ (given)}$$

$$\zeta = \alpha \text{ (eval)}$$

$$\zeta = \alpha \text{ (same)}$$

$$\zeta = \alpha \text{ (found var } \zeta)$$

$$\angle ACD = \zeta = \alpha \rightarrow \angle ACD = \alpha$$

$$\angle ECB = -\zeta - \varepsilon + 180 = -(\alpha) - \varepsilon + 180 \rightarrow \angle ECB = -\varepsilon - \alpha + 180$$

$$\angle ABC = \angle ECB$$

(alternating angles are equal between $AB \parallel DE$ and BC)

$$\angle ECB = \gamma \text{ (eval)}$$

$$\angle ECB = \gamma \text{ (given)}$$

$$-\varepsilon - \alpha + 180 = \gamma \text{ (eval)}$$

$$-\varepsilon - \alpha + 180 = \gamma \text{ (same)}$$

$$\varepsilon = -\gamma - \alpha + 180 \text{ (found var } \varepsilon)$$

$$\angle ECB = -\varepsilon - \alpha + 180 = -(-\gamma - \alpha + 180) - \alpha + 180 \rightarrow \angle ECB = \gamma$$

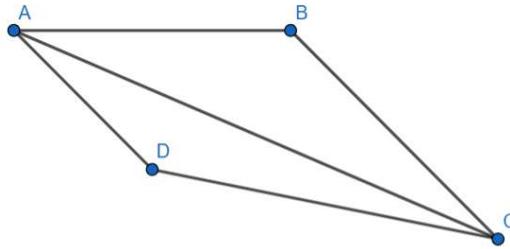
$$\angle BCA = \varepsilon = (-\gamma - \alpha + 180) \rightarrow \angle BCA = -\gamma - \alpha + 180$$

$$\angle CAB + \angle ABC + \angle BCA = (\alpha) + (\gamma) + (-\gamma - \alpha + 180) = 180$$

↓

sum of angles in $\triangle ABC$ is 180°

((<180>,), (180,))



משפט 7: סכום הזווית במלובע הוא 360°

נתון: מרובע ABCD, אלכסון AC.

הוכחה: סכום הזווית במלובע ABCD שווה ל- 360° .

הוכחה:

$$\text{סכום הזווית במשולש } (ABC) \angle BAC + \angle CBA + \angle ACB = 180^\circ$$

$$\text{סכום הזווית במשולש } (ACD) \angle CAD + \angle DCA + \angle ADC = 180^\circ$$

$$(\text{השלם שווה לסכום חלקי}) \angle BAD = \angle BAC + \angle CAD, \angle DCB = \angle DCA + \angle ACB$$

$$\angle BAD + \angle CBA + \angle DCB + \angle ADC$$

$$= (\angle BAC + \angle CAD) + \angle CBA + (\angle DCA + \angle ACB) + \angle ADC \quad (\text{כלל המעבר})$$

$$= (\angle BAC + \angle CBA + \angle ACB) + (\angle CAD + \angle DCA + \angle ADC) = 180 + 180 = 360$$

לכן, סכום הזווית במלובע ABCD שווה ל- 360°

מ.ש.ל

קוד:

```
def th7(print_proof=False):
    """_ax2 + _th6 -> _th7"""
    h = Helper()
    h.ps_from_file(r"ggp_files\th7.ggb")
    h.poly("ABCD")
    h.s("AC")
    h.calc(print_proof, use_theorems=[1, 2, 3, 4, 5, 6])
    if print_proof:
        print(
            " + ".join(map(str, h.a("BAD", "ADC", "DCB", "CBA"))),
            "=",
            "(",
            ") + ".join(map(str, h.get_a("BAD", "ADC", "DCB", "CBA"))),
            ")",
            "=",
            sum(h.get_a("BAD", "ADC", "DCB", "CBA")),
        )
        print(f"\n")
        print(f"sum of angles in □ ABCD is {360}°")

    return (sum(h.get_a("BAD", "ADC", "DCB", "CBA")), (360,))
```

$$360 = \angle CAD + \angle DAB + \angle BAC = \angle CAD + \angle DAB + \angle BAC$$

(sum of angles around point A, the whole is the sum of its parts)

$$\angle BAC = 360 - (\angle CAD + \angle DAB) \text{ (same)}$$

$$\angle BAC = 360 - (\alpha + \beta) \text{ (eval)} \rightarrow \angle BAC = -\beta - \alpha + 360 \text{ (calc)}$$

$$360 = \angle CBA + \angle ABC = \angle CBA + \angle ABC$$

(sum of angles around point B, the whole is the sum of its parts)

$$\angle ABC = 360 - \angle CBA \text{ (same)}$$

$$\angle ABC = 360 - \delta \text{ (eval)} \rightarrow \angle ABC = -\delta + 360 \text{ (calc)}$$

$$360 = \angle DCA + \angle ACB + \angle BCD = \angle DCA + \angle ACB + \angle BCD$$

(sum of angles around point C, the whole is the sum of its parts)

$$\angle BCD = 360 - (\angle DCA + \angle ACB) \text{ (same)}$$

$$\angle BCD = 360 - (\zeta + \eta) \text{ (eval)} \rightarrow \angle BCD = -\eta - \zeta + 360 \text{ (calc)}$$

$$360 = \angle ADC + \angle CDA = \angle ADC + \angle CDA$$

(sum of angles around point D, the whole is the sum of its parts)

$$\angle CDA = 360 - \angle ADC \text{ (same)}$$

$$\angle CDA = 360 - \iota \text{ (eval)} \rightarrow \angle CDA = -\iota + 360 \text{ (calc)}$$

$$180 = \angle BAC + \angle CBA + \angle ACB = \angle BAC + \angle CBA + \angle ACB$$

(the sum of the interior angles of $\triangle ABC$ is 180° ,

the whole is the sum of its parts)

$$\angle ACB = 180 - (\angle BAC + \angle CBA) \text{ (same)}$$

$$\eta = 180 - (-\beta - \alpha + 360 + \delta) \text{ (eval)}$$

$$\eta = -\delta + \beta + \alpha - 180 \text{ (same)}$$

$$\eta = -\delta + \beta + \alpha - 180 \text{ (found var } \eta)$$

$$\angle ACB = \eta = (-\delta + \beta + \alpha - 180) \rightarrow \angle ACB = -\delta + \beta + \alpha - 180$$

$$\angle BCD = -\eta - \zeta + 360 = -(-\delta + \beta + \alpha - 180) - \zeta + 360 \rightarrow \angle BCD = -\zeta + \delta - \beta - \alpha + 540$$

$$180 = \angle CAD + \angle DCA + \angle ADC = \angle CAD + \angle DCA + \angle ADC$$

(the sum of the interior angles of $\triangle ACD$ is 180° ,

the whole is the sum of its parts)

$$\angle ADC = 180 - (\angle CAD + \angle DCA) \text{ (same)}$$

$$\iota = 180 - (\alpha + \zeta) \text{ (eval)}$$

$$\iota = -\zeta - \alpha + 180 \text{ (same)}$$

$$\iota = -\zeta - \alpha + 180 \text{ (found var } \iota)$$

$$\angle ADC = \iota = (-\zeta - \alpha + 180) \rightarrow \angle ADC = -\zeta - \alpha + 180$$

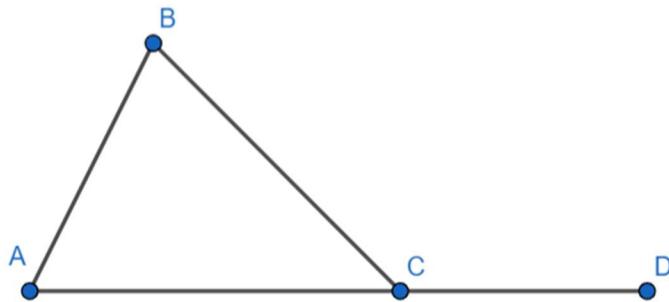
$$\angle CDA = -\iota + 360 = -(-\zeta - \alpha + 180) + 360 \rightarrow \angle CDA = \zeta + \alpha + 180$$

$$\begin{aligned}
 & \angle BAD + \angle ADC + \angle DCB + \angle CBA \\
 &= (-\beta + 360) + (-\zeta - \alpha + 180) + (\zeta - \delta + \beta + \alpha - 180) + (\delta) = 360 \\
 &\Downarrow \\
 & \text{sum of angles in } \square ABCD \text{ is } 360^\circ \\
 & ((360), (360))
 \end{aligned}$$

משפט 8: זווית חיצונית למשולש שווה לסכום שתי הזוויות הפנימיות שאינן צמודות לה

נתון: משולש ABC, D היא נקודה בה המשך הישר AC.

$$\text{הוכחה: } \angle BCD = \angle BAC + \angle CBA$$



הוכחה:

$$\angle ACB = 180 - (\angle BAC + \angle CBA) \quad (\text{סכום זוויות במשולש ABC})$$

$$\angle BCD = 180 - \angle ACB \quad (\text{סכום זוויות על ישר AD})$$

$$\angle BCD = 180 - (180 - (\angle BAC + \angle CBA)) = \angle BAC + \angle CBA \quad (\text{כלל המעבר})$$

$$\therefore \angle BCD = \angle BAC + \angle CBA$$

מ.ש.

```

def th8(print_proof=False):
    """_th1 + _th6 -> _th8"""
    h = Helper()
    h.ps_from_file(r"ggb_files\th8.ggb")
    h.tri("ACB")
    h.conts("AC", "ACD")
    h.calc(print_proof, use_theorems=range(1, 8))
    if print_proof:
        print(
            f"{{h.a('BCD')}} = {{h.get_a('BCD')}},",
            f"{{h.a('BAC')}} + {{h.a('CBA')}} = ({ {h.get_a('BAC')} })",
            f" + ({ {h.get_a('CBA')} })",
            f"= {{sum(h.get_a('BAC'), 'CBA'))}}",
        )
        print(f"\u21d2")
        print(f"{{h.a('BCD')}} = {{h.a('BAC')}} + {{h.a('CBA')}}"
    )
    return (h.get_a("BCD") == sum(h.get_a("BAC", "CBA")),), (True,)

```

$$180 = \angle ACD = \angle ACB + \angle BCD$$

(angle upon line AD is 180, the whole is the sum of its parts)

$$\angle BCD = 180 - \angle ACB \text{ (same)}$$

$$\angle BCD = 180 - \varepsilon \text{ (eval)} \rightarrow \angle BCD = -\varepsilon + 180 \text{ (calc)}$$

$$360 = \angle BAD + \angle DAB = \angle BAD + \angle DAB$$

(sum of angles around point A, the whole is the sum of its parts)

$$\angle DAB = 360 - \angle BAD \text{ (same)}$$

$$\angle DAB = 360 - \alpha \text{ (eval)} \rightarrow \angle DAB = -\alpha + 360 \text{ (calc)}$$

$$360 = \angle CBA + \angle ABC = \angle CBA + \angle ABC$$

(sum of angles around point B, the whole is the sum of its parts)

$$\angle ABC = 360 - \angle CBA \text{ (same)}$$

$$\angle ABC = 360 - \gamma \text{ (eval)} \rightarrow \angle ABC = -\gamma + 360 \text{ (calc)}$$

$$180 = \angle BAC + \angle CBA + \angle ACB = \angle BAD + \angle CBA + \angle ACB$$

(the sum of the interior angles of $\triangle ABC$ is 180° ,

the whole is the sum of its parts)

$$\angle ACB = 180 - (\angle BAD + \angle CBA) \text{ (same)}$$

$$\varepsilon = 180 - (\alpha + \gamma) \text{ (eval)}$$

$$\varepsilon = -\gamma - \alpha + 180 \text{ (same)}$$

$$\varepsilon = -\gamma - \alpha + 180 \text{ (found var } \varepsilon)$$

$$\angle ACB = \varepsilon = (-\gamma - \alpha + 180) \rightarrow \angle ACB = -\gamma - \alpha + 180$$

$$\angle BCD = -\varepsilon + 180 = -(-\gamma - \alpha + 180) + 180 \rightarrow \angle BCD = \gamma + \alpha$$

$$\angle BCD = \gamma + \alpha, \angle BAD + \angle CBA = (\alpha) + (\gamma) = \gamma + \alpha$$

↓

$$\angle BCD = \angle BAD + \angle CBA$$

((True,), (True,))

הצעות לשיפור

הוסףת משפטים נוספים

התוכנה שלי עדין לא תומכת בכל המשפטים שניתן להשתמש בהם לצורך פתרון בעיות גיאומטריה, והוסףת משפטיים תגדיל את כמות הבעיה שהתוכנה תוכל לפתור, ואולי תפשט את הדרך שתפתרו בעיות שהוא כבר מסוגל לפתור.

צמצום הפלט ו שימוש במשפטים הנדרשים בלבד

כרגע, התוכנה מספקת פلت שחלקיו יכול להיות לא רלוונטיים למה שצריך להוכיח. שיפור מעניין יהיה לנסוט לצמצם את הפלט הזה, ולהשתמש במינימום משפטיים על מנת להגיע לטענה שצריך להוכיח.

הוסףת תמיכה במבנה עז

כרגע התוכנה מנסה להתאים את המשפטים לנתונים אותם היא קיבלה, מבליל להוסיף ישרים נוספים או נקודות נוספות. בהמשך, אש mach אם התוכנה תוכל להוסיף על דעת עצמה בניית עזר רלוונטיות שיקלו את פתרון הבעיה או יגרמו לה להיות אפשרית מלכתחילה.

```
1  GeoAD/
2  |   STRUCTURE.txt
3  |
4  |   geo
5  |       __init__.py
6  |       abs
7  |           __init__.py
8  |           point.py
9  |           abssegment.py
10 |           absangle.py
11 |       alg
12 |           __init__.py
13 |           unionfind.py
14 |       real
15 |           __init__.py
16 |           expression.py
17 |           realsegment.py
18 |           realangle.py
19 |       comp
20 |           __init__.py
21 |           convertor.py
22 |           polygon.py
23 |       filehandler.py
24 |       handler.py
25 |       helper.py
26 |       problemcollection.py
27 |
28 |   ggb_files
29 |       th1.ggb
30 |       th2.ggb
31 |       th3.ggb
32 |       th4.ggb
33 |       th5_1.ggb
34 |       th5_2.ggb
35 |       th5_3.ggb
36 |       th6.ggb
37 |       th7.ggb
38 |       th8.ggb
39 |       proofs.py
40 |       yoel_geva.py
41 |       main.py
```

```
1     """geo/__init__.py"""
```



```

1  """geo/abs/point.py"""
2
3
4  class Point:
5
6      next_id = 0
7
8      def __init__(self, name, x=None, y=None):
9          """Create a new point"""
10         self.name = name
11         self.id = Point.next_id
12         Point.next_id += 1
13         self.lines = []
14
15         self.x = x
16         self.y = y
17
18     def add_line(self, *lines):
19         """Add line or lines that the point is on"""
20         for line in lines:
21             if self in line.midpoints:
22                 # split the segment into 2 segments (self is an endpoint to both)
23                 self.lines.append(
24                     (line.get_subsegment(self.name + line.start.name), line)
25                 )
26                 self.lines.append(
27                     (line.get_subsegment(self.name + line.end.name), line)
28                 )
29             else:
30                 self.lines.append((line, line))
31
32         # sort self.lines by slope_angle
33         self.lines.sort(key=lambda t: -t[0].get_slope_angle(self))
34
35     def remove_line(self, *lines):
36         """Remove line or lines from self.lines"""
37         rm_idxs = []
38         for idx, (_, line) in enumerate(self.lines):
39             if line in lines:
40                 rm_idxs.append(idx)
41         for idx in rm_idxs[::-1]:
42             del self.lines[idx]
43
44     def __str__(self):
45         """Return the point's name"""
46         return self.name
47
48     def __repr__(self):
49         """Return the point's name, id and coordinates"""
50         return f"<{self.name} ({self.id}) ({self.x}, {self.y})>"
51
52     def __hash__(self):
53         """Used for hashing Segment and set(Point)"""
54         return hash(self.id)
55
56     def __eq__(self, other):
57         """Compare Points' id"""
58         return other.id == self.id
59
60     @staticmethod
61     def createPoints(names, xs, ys):
62         """Return a tuple of Points according to points' names, xs and ys"""
63         return tuple([Point(name, x, y) for name, x, y in zip(names, xs, ys)])
64

```

```

1  """geo/abs/abssegment.py"""
2
3  import math
4
5  from geo.alg.unionfind import UnionFind
6  from geo.abs.point import Point
7
8
9  class AbsSegment:
10     def __init__(self, pointstart, pointend, isnew=False):
11         """ if isnew: add this line to the start & end points"""
12         self.start = pointstart
13         self.end = pointend
14         self.midpoints = []
15         self.parallel = UnionFind()
16         self.isnew = isnew
17         if self.isnew:
18             self.start.add_line(self)
19             self.end.add_line(self)
20
21     def set_midpoints(self, *x):
22         """Set line's midpoints"""
23         self.midpoints = list(x)
24         if self.isnew:
25             for p in x:
26                 p.add_line(self)
27
28     def update_midpoints(self, *x):
29         """Add midpoints to segment"""
30         if self.isnew:
31             # remove all instances of this segment in Point.lines
32             for p in self.midpoints:
33                 rm = []
34                 for i in range(p.lines):
35                     if p.lines[i][1] == self:
36                         rm.append(i)
37                 for idx in rm[::-1]:
38                     del p.lines[idx]
39             # set midpoints to x
40             self.set_midpoints(*x)
41
42     def get_all_points(self):
43         """Return a list containing startpoint, midpoints and endpoint"""
44         return [self.start] + self.midpoints + [self.end]
45
46     def get_subsegment_to(self, point):
47         """Return subsegment to point (containing all the midpoints in between)"""
48         if point in self.midpoints:
49             res = AbsSegment(self.start, point)
50             res.set_midpoints(*self.midpoints[0:self.midpoints.index(point)])
51             res.parallel = self.parallel
52             return res
53         elif point == self.end:
54             return self
55         elif point == self.start:
56             return AbsSegment(point, point)
57         return None
58
59     def get_subsegment_from(self, point):
60         """Return subsegment from point (containing all the midpoints in between)"""
61         if point in self.midpoints:
62             res = AbsSegment(point, self.end)
63             res.set_midpoints(*self.midpoints[self.midpoints.index(point)+1:])
64             res.parallel = self.parallel
65             return res
66         elif point == self.start:
67             return self

```

```

68     .....
69     .....
70     .....
71
72     .....
73     .....
74     .....
75     .....
76     .....
77     .....
78     .....
79     .....
80     .....
81
82     .....
83     .....
84     .....
85     .....
86
87     .....
88     .....
89     .....
90     .....
91     .....
92
93     .....
94     .....
95     .....
96
97     .....
98     .....
99     .....
100
101    .....
102    .....
103    .....
104
105    .....
106    .....
107    .....
108
109    .....
110    .....
111    .....
112    .....
113    .....
114    .....
115    .....
116    .....
117
118    .....
119    .....
120    .....
121    .....
122
123    .....
124    .....
125    .....
126    .....
127    .....
128    .....
129    .....
130    .....
131    .....
132    .....

```

```

133     ..... return 0
134     ..... dy = self.end.y - self.start.y
135     ..... dx = self.end.x - self.start.x
136
137     ..... other_p = [i for i in (self.end, self.start) if i != p][0]
138     ..... if dx == 0:
139     .....     if other_p.y > p.y:
140     .....         return 90
141     .....     else:
142     .....         return 270
143     ..... if dy == 0:
144     .....     if other_p.x > p.x:
145     .....         return 0
146     .....     else:
147     .....         return 180
148
149     ..... arctan = math.degrees(math.atan(dy / dx))
150
151     ..... if other_p.x < p.x:
152     .....     return (arctan + 180 + 360) % 360
153     ..... return (arctan + 360) % 360
154
155     ..... def better_direction(self, base_angle_direction=0):
156     .....     """Reverse segment if needed (closest direction to base_angle)"""
157     .....     slope_ang = self.get_slope_angle()
158     .....     self.reverse()
159     .....     new_slope_ang = self.get_slope_angle()
160     .....     new_ang_dist = min(
161     .....         (base_angle_direction - new_slope_ang + 360) % 360,
162     .....         (new_slope_ang - base_angle_direction + 360) % 360,
163     .....     )
164     .....     old_ang_dist = min(
165     .....         (base_angle_direction - slope_ang + 360) % 360,
166     .....         (slope_ang - base_angle_direction + 360) % 360,
167     .....     )
168     .....     if new_ang_dist > old_ang_dist:
169     .....         self.reverse()
170
171     ..... def is_valid(self):
172     .....     """Checks if self is a valid segment (has startpoint and endpoint)"""
173     .....     return (
174     .....         self.start is not None
175     .....         and self.end is not None
176     .....         and self.start != self.end
177     .....         and isinstance(self.start, Point)
178     .....         and isinstance(self.end, Point)
179     .....     )
180
181     ..... def __repr__(self):
182     .....     """Return Segment's name and midpoints"""
183     .....     return (
184     .....         "<" +
185     .....         + self.start.name
186     .....         + self.end.name
187     .....         + ":" +
188     .....         + str([str(i) for i in self.midpoints])
189     .....         + ">"
190     .....     )
191
192     ..... def __str__(self):
193     .....     """Return Segment's name"""
194     .....     return self.start.name + self.end.name
195
196     ..... def __hash__(self):
197     .....     """Used for set(Segment)"""
198     .....     # added reverse so that seg, seg.reverse() will have the same hash
199     .....     return hash(tuple(self.get_all_points())) + hash(

```

```
200     .....tuple(self.get_all_points() [::-1])
201     ....)
202
203     def __eq__(self, other):
204         """Check Segment == Segment"""
205         return (
206             .....other.get_all_points() == self.get_all_points()
207             .....or other.get_all_points() [::-1] == self.get_all_points()
208         )
209
```

```

1  """geo/abs/absangle.py"""
2
3  from geo.abs.abssegment import AbsSegment
4
5
6  class AbsAngle:
7      def __init__(self, ray1, vertex, ray2):
8          """Create abstract angle"""
9          self.ray1 = ray1
10         self.vertex = vertex
11         self.ray2 = ray2
12
13     def get_start_point(self):
14         """Return the startpoint of ray1"""
15         return self.ray1.start if self.ray1.end == self.vertex else self.ray1.end
16
17     def get_end_point(self):
18         """Return the endpoint of ray2"""
19         return self.ray2.start if self.ray2.end == self.vertex else self.ray2.end
20
21     def get_minimized_absangle(self):
22         """Return an AbsAngle with the smallest rays possible (common for all same angles)"""
23         mr = []
24         for r in [self.ray1, self.ray2]:
25             if r.start == self.vertex:
26                 mr.append(r.get_subsegment_to(r.get_all_points()[1]))
27             else:
28                 mr.append(r.get_subsegment_from(r.get_all_points()[-2]))
29         return AbsAngle(mr[0], self.vertex, mr[1])
30
31     def reverse(self):
32         """Switch ray1 & ray2"""
33         self.ray1, self.ray2 = self.ray2, self.ray1
34
35     def get_angle_size_from_coordinates(self):
36         """Return angle size based on points' coordinates"""
37         ray1_slope_angle = self.ray1.get_slope_angle(self.vertex)
38         ray2_slope_angle = self.ray2.get_slope_angle(self.vertex)
39         return (ray1_slope_angle - ray2_slope_angle + 360) % 360
40
41     def __hash__(self):
42         """custom hash function such that every same angle gets the same hash"""
43         ang = self.get_minimized_absangle()
44         return hash((ang.get_start_point(), ang.vertex, ang.get_end_point()))
45
46     def __eq__(self, other):
47         """Check if the angles are the same abstract angle"""
48         return (
49             self.ray1.are_inclusive(other.ray1)
50             and self.vertex == other.vertex
51             and self.ray2.are_inclusive(other.ray2)
52         )
53
54     def __str__(self):
55         """Return Angle's name"""
56         return (
57             "□"
58             + self.get_start_point().name
59             + self.vertex.name
60             + self.get_end_point().name
61         )
62
63     def __repr__(self):
64         """Return str(self)"""
65         return str(self)
66

```



```
1  """geo/alg/unionfind.py"""
2
3
4  class UnionFind:
5      def __init__(self):
6          """Init self to a set by itself"""
7          self.par = self
8          self.rank = 0
9
10     def find(self):
11         """Return the root of self and do path compression"""
12         if self.par != self:
13             self.par = self.par.find()
14         return self.par
15
16     def union(self, other):
17         """Union the sets of self and other (by rank)"""
18         x, y = self.find(), other.find()
19         if x == y:
20             return
21         if x.rank < y.rank:
22             x, y = y, x
23             # now x.rank >= y.rank
24             y.par = x
25         if x.rank == y.rank:
26             x.rank += 1
27
28     def is_same(self, other):
29         """Check if self and other are in the same set"""
30         return self.find() == other.find()
31
```



```

1  """geo/real/expression.py"""
2
3  from functools import total_ordering
4
5
6  @total_ordering
7  class Expression:
8
9      next_given_idx = 2
10     next_var_idx = 1
11     watched = []
12     switched = []
13     given_symbols = {}
14     alphabet = "abcdefghijklmnopqrstuvwxyz"
15
16     def __init__(self, newvar=True, d={}):
17         """Create Expression object based on dict,int,float or empty (w/ or w/o
18             newvar)"""
19         if isinstance(d, dict):
20             self.value = dict(d)
21         elif isinstance(d, (int, float)):
22             self.value = {0: d}
23         else:
24             raise TypeError(f"d as a {type(d)} is not supported")
25         if newvar:
26             # create new variable
27             self.value[type(self).next_var_idx] = 1
28             type(self).next_var_idx += 1
29             self.clean()
30
31     def new_copy(self):
32         """Return a new_copy of this object"""
33         res = type(self)(False)
34         for i, j in self.value.items():
35             res.value[i] = j
36         return res
37
38     def copy(self, other):
39         """Copy other dict to this object"""
40         del self.value
41         self.value = {}
42         for i, j in other.value.items():
43             self.value[i] = j
44         self.clean()
45
46     def switch(self, key, switch_exp):
47         """switch every instance of this key to switch_exp"""
48         type(self).switch_watched(key, switch_exp)
49         if key not in self.value:
50             return
51         else:
52             times = self.value[key]
53             del self.value[key]
54             self += switch_exp * times
55             self.clean()
56
57     def clean(self):
58         """clean keys that have a zero value or float to int"""
59         # remove zero value keys
60         for key in [key for key in self.value if self.value[key] == 0]:
61             del self.value[key]
62         # convert float keys to int when needed
63         for key in self.value:
64             if isinstance(self.value[key], float) and self.value[key].is_integer():
65                 self.value[key] = int(self.value[key])
66
67     def isknown(self):

```

```

67     .... """Check if the expression is known or has variables in it"""
68     .... self.clean()
69     .... for i in self.value:
70     ....     if i != 0:
71     ....         return False
72     .... return True
73
74 def watch(self):
75     .... """Set watch on this expression (update its value when can)"""
76     .... type(self).watched.append(self)
77
78 def __add__(self, other):
79     .... """Add the values for the same key and add the missing keys with
80     .... dict/Expression/int/float"""
81     .... res = self.new_copy()
82     .... if isinstance(other, type(self)):
83     ....     for idx, val in other.value.items():
84     ....         if idx in res.value:
85     ....             res.value[idx] += val
86     ....         else:
87     ....             res.value[idx] = val
88     .... elif isinstance(other, (dict, int, float)):
89     ....     return self + type(self)(newvar=False, d=other)
90     .... else:
91     ....     return NotImplemented
92
93     .... res.clean()
94     .... return res
95
96 def __radd__(self, other):
97     .... return self + other
98
99 def __iadd__(self, other):
100    .... self.copy(self + other)
101    .... return self
102
103 def __sub__(self, other):
104    .... if isinstance(other, (type(self), int, float)):
105    ....     return self + (-other)
106    .... elif isinstance(other, dict):
107    ....     return self + (-(type(self)(newvar=False, d=other)))
108    .... else:
109    ....     return NotImplemented
110
111 def __rsub__(self, other):
112    .... return other + (-self)
113
114 def __isub__(self, other):
115    .... self.copy(self - other)
116    .... return self
117
118 def __mul__(self, other):
119    .... if isinstance(other, (int, float)):
120    ....     res = self.new_copy()
121    ....     for i in res.value.keys():
122    ....         res.value[i] *= other
123    ....     res.clean()
124    ....     return res
125    .... return NotImplemented
126
127 def __rmul__(self, other):
128    .... return self * other
129
130 def __imul__(self, other):
131    .... self.copy(self * other)
132    .... return self

```

```

133     ...def __truediv__(self, other):
134         ...if isinstance(other, (int, float)):
135             ...    return self * (1 / other)
136         ...return NotImplemented
137
138     ...def __itruediv__(self, other):
139         ...self.copy(self / other)
140         ...return self
141
142     ...def __neg__(self):
143         ...return self * (-1)
144
145     ...@classmethod
146     ...def str_term(cls, tup, custom):
147         ...idx, val = tup
148         ...if val == 0:
149             ...    return ""
150         ...if idx == 0: # const
151             ...    if val > 0:
152                 ...        return "+" + str(val)
153             ...        return str(val)
154
155         ...alp = cls.alphabet
156         ...if isinstance(idx, int):
157             ...    str_var = alp[idx - 1] if 0 < idx < len(alp) + 1 else f"A{idx-1}"
158             ...else:
159                 ...    str_var = cls.given_symbols[idx]
160             ...if idx in custom:
161                 ...    str_var = custom[idx]
162             ...if val == 1 or val == -1:
163                 ...    sgn = "-" if val < 0 else "+"
164                 ...    return sgn + str_var
165             ...sgn = "" if val < 0 else "+"
166             ...return sgn + str(val) + str_var
167
168     ...def __str__(self, *custom):
169         """Returns a alphabet letter or A{idx} polynomial"""
170         ...if self == 0:
171             ...    return "0"
172         ...if len(custom) % 2 != 0:
173             ...    raise "Custom alphabet size should be even!"
174         ...else:
175             ...    d = {}
176             ...    for i in range(0, len(custom), 2):
177                 ...        d[custom[i]] = custom[i + 1]
178             ...    custom = d
179
180         ...res = ""
181         ...for i in sorted(self.value.items())[::-1]:
182             ...if res == "":
183                 ...    if type(self).str_term(i, custom)[0] == "+":
184                     ...        res = type(self).str_term(i, custom)[1:]
185                 ...else:
186                     ...        res = type(self).str_term(i, custom)
187             ...else:
188                 ...        res += " " + type(self).str_term(i, custom)
189         ...return res
190
191     ...def __repr__(self):
192         """Return <str(self)>"""
193         ...return f"<{self}>"
194
195     ...def __int__(self):
196         """Return the value of the expression"""
197         ...if self.isknown():
198             ...    return self.value[0]
199             ...raise ValueError(f"Cannot convert {self} to type int")

```

```

200
201     ...def __lt__(self, other):
202         """Do lexicographic compare between the objects' sorted keys"""
203         if other is None:
204             raise TypeError("NoneType compared to Degree")
205             if isinstance(other, (int, float, dict)):
206                 return self < type(self)(False, other)
207             if isinstance(other, type(self)):
208                 return (
209                     sorted(list(self.value.keys()))[::-1]
210                     < sorted(list(other.value.keys()))[::-1]
211                 )
212             return NotImplemented
213
214     ...def __eq__(self, other):
215         """Compare two Expressions"""
216         if isinstance(other, (int, float)):
217             if other == 0 and len(self.value) == 0:
218                 return True
219             return len(self.value) == 1 and 0 in self.value and self.value[0] == other
220         elif isinstance(other, (type(self), dict)):
221             return self - other == 0
222         else:
223             return NotImplemented
224
225     @classmethod
226     def reset(cls):
227         """Resets the variable count to start over from alpha"""
228         cls.nextVarIdx = 1
229
230     @classmethod
231     def variable_reduction(cls, *all_exp):
232         """Replaces variables' names to fill the alphabet from the beginning"""
233         existing_keys = set(sum([list(i.value.keys()) for i in all_exp], []))
234         if 0 in existing_keys:
235             existing_keys.remove(0)
236             xchg = 1
237             for i in list(existing_keys)[::-1]:
238                 if i > len(existing_keys):
239                     while xchg in existing_keys:
240                         xchg += 1
241                     for exp in all_exp:
242                         exp.switch(i, cls(False, {xchg: 1}))
243
244     @classmethod
245     def switch_watched(cls, key, exp):
246         """Switch key for exp in every Expression in watched if it hasn't been done
247         before"""
248         if (key, exp) in cls.switched:
249             return
250             cls.switched.append((key, exp.new_copy()))
251             for w in cls.watched:
252                 w.switch(key, exp)
253
254     @classmethod
255     def given(cls, *symbols):
256         """Returns Expression(s) with the given symbol(s)"""
257         res = []
258         for symbol in symbols:
259             d = cls(False, {})
260             d.value[1 / cls.next_given_idx] = 1
261             cls.given_symbols[1 / cls.next_given_idx] = symbol
262             cls.next_given_idx += 1
263             d.watch()
264             res.append(d)
265         return tuple(res) if len(res) > 1 else res[0]

```



```

1  """geo/real/realsegment.py"""
2
3  from geo.abs.abssegment import AbsSegment
4  from geo.real.expression import Length
5  from functools import total_ordering
6
7
8  @total_ordering
9  class RealSegment(AbsSegment):
10     def __init__(self, *points, leng=None):
11         """Create an Segment with length"""
12         self.leng = leng.new_copy() if leng is not None else None
13         AbsSegment.__init__(self, points[0], points[-1])
14         self.set_midpoints(*points[1:-1])
15
16     def set_value(self, val=None):
17         """Set segmnt's length based on val [int,float,dict,Length] (default is newvar)"""
18         if isinstance(val, (int, float, dict)):
19             self.leng = Length(False, val)
20         elif isinstance(val, Length):
21             self.leng = val.new_copy()
22         elif val is None:
23             self.leng = Length()
24         else:
25             raise TypeError(f"cannot set segment value to {type(val)}")
26
27     def get_expression(self):
28         """Return self.leng"""
29         return self.leng
30
31     def new_copy(self):
32         """Return a new copy of this object"""
33         return RealSegment(
34             *self.get_all_points(),
35             leng=self.leng.new_copy() if self.leng is not None else None,
36         )
37
38     def isknown(self):
39         """Check if the segment's length is known or has variables in it"""
40         return (self.leng is not None) and self.leng.isknown()
41
42     def get_all_subsegments(self):
43         """Return a list of all Realsubsegments"""
44         points = self.get_all_points()
45         return [RealSegment(*points[i:i+2]) for i in range(len(points)-1)]
46
47     def __str__(self):
48         """Return Segment's name and length"""
49         return (
50             AbsSegment.__str__(self)
51             + "("
52             + (str(self.leng) if self.leng is not None else "")
53             + ")"
54         )
55
56     def __repr__(self):
57         """Same as __str__"""
58         return str(self)
59
60     def __lt__(self, other):
61         if isinstance(other, RealSegment):
62             return self.leng < other.leng
63         else:
64             return self.leng < other
65
66     def __eq__(self, other):

```

```
67     ..... if isinstance(other, RealSegment):
68     .....     return self.leng == other.leng
69     ..... elif isinstance(other, AbsSegment):
70     .....     return other == abs(self)
71     ..... else:
72     .....     return self.leng == other
73
74     ..... def __add__(self, other):
75     .....     """Return Length of sum"""
76     .....     if isinstance(other, RealSegment):
77     .....         return self.leng + other.leng
78     .....     else:
79     .....         return self.leng + other
80
81     ..... def __radd__(self, other):
82     .....     return self + other
83
84     ..... def __sub__(self, other):
85     .....     if isinstance(other, RealSegment):
86     .....         return self.leng - other.leng
87     .....     else:
88     .....         return self.leng - other
89
90     ..... def __rsub__(self, other):
91     .....     return other - self.leng
92
93     ..... def __abs__(self):
94     .....     """Return Segment from this RealSegemnt"""
95     .....     res = AbsSegment(self.start, self.end)
96     .....     res.set_midpoints(*self.midpoints)
97     .....     return res
98
99     ..... def __int__(self):
100     .....     """Return the value of the length"""
101     .....     return int(self.leng)
102
103     ..... @classmethod
104     ..... def from_abssegment(cls, seg, leng=None):
105     .....     """Create a RealSegment based on Segment"""
106     .....     return RealSegment(*seg.get_all_points(), leng=leng)
107
```

```

1  """geo/real/realangle.py"""
2
3  from geo.abs.abssegment import AbsSegment
4  from geo.abs.absangle import AbsAngle
5  from geo.real.expression import Degree
6
7  from functools import total_ordering
8
9
10 @total_ordering
11 class RealAngle(AbsAngle):
12     def __init__(self, ray1, vertex, ray2, deg=None):
13         """Create an Angle with value"""
14         self.deg = deg.new_copy() if deg is not None else None
15         AbsAngle.__init__(self, ray1, vertex, ray2)
16
17     def set_value(self, val=None):
18         """Set angle's degree based on val [int, float, dict, Degree] (default is newvar)"""
19         if isinstance(val, (int, float, dict)):
20             self.deg = Degree(False, val)
21         elif isinstance(val, Degree):
22             self.deg = val.new_copy()
23         elif val is None:
24             self.deg = Degree()
25         else:
26             raise TypeError(f"cannot set angle value to {type(val)}")
27
28     def get_expression(self):
29         """Return self.deg"""
30         return self.deg
31
32     def new_copy(self):
33         """Return a new copy of this object"""
34         return RealAngle(
35             self.ray1,
36             self.vertex,
37             self.ray2,
38             self.deg.new_copy() if self.deg is not None else None,
39         )
40
41     def isknown(self):
42         """Check if the angles's degree is known or has variables in it"""
43         return (self.deg is not None) and self.deg.isknown()
44
45     def __str__(self):
46         """Return Angle's name and degree"""
47         return (
48             AbsAngle.__str__(self)
49             + "("
50             + (str(self.deg) if self.deg is not None else ""))
51             + ")"
52         )
53
54     def __lt__(self, other):
55         """Compare self to Degree or RealAngle"""
56         if isinstance(other, RealAngle):
57             return self.deg < other.deg
58         else:
59             return self.deg < other
60
61     def __eq__(self, other):
62         """Compare self to Degree/RealAngle or AbsAngle"""
63         if isinstance(other, RealAngle):
64             return self.deg == other.deg
65         elif isinstance(other, AbsAngle):
66             return (
67                 self.get_end_point() == other.get_end_point())

```

```
68     .....and self.vertex == other.vertex
69     .....and self.get_start_point() == other.get_start_point()
70     ....)
71     else:
72     .....return self.deg == other
73
74 def __add__(self, other):
75     """Return Degree of sum"""
76     if isinstance(other, RealAngle):
77         return self.deg + other.deg
78     else:
79         return self.deg + other
80
81 def __radd__(self, other):
82     return self + other
83
84 def __sub__(self, other):
85     if isinstance(other, RealAngle):
86         return self.deg - other.deg
87     else:
88         return self.deg - other
89
90 def __rsub__(self, other):
91     return other - self.deg
92
93 def __abs__(self):
94     """Return AbsAngle from this RealAngle"""
95     return AbsAngle(self.ray1, self.vertex, self.ray2)
96
97 def __int__(self):
98     """Return the value of the degree"""
99     return int(self.deg)
100
101 @classmethod
102 def from_absangle(cls, absang, deg=None):
103     """Create a RealAngle based on an AbsAngle"""
104     return RealAngle(absang.ray1, absang.vertex, absang.ray2, deg)
```



```
1 """geo/comp/convertor.py"""
2
3
4 class Convertor:
5     def __init__(self, tolist, get):
6         self.tolist = tolist
7         self.get = get
8
9     def __getitem__(self, key):
10        return [self.get(i) for i in self.tolist(key)]
11
```

```
1 """geo/comp/polygon.py"""
2
3
4 class Polygon:
5     def __init__(self, points, sides, aangs, aconv, sconv):
6         """Init points, sides, and angles"""
7         self.points = points
8         self.sides = sides
9         self.aangs = aangs
10        self.aconv = aconv
11        self.sconv = sconv
12
13    def get_angle_from_point(self, p):
14        """Return the interior angle of point p"""
15        for a in self.aangs:
16            if a.vertex == p:
17                return a
18        return None
19
20    def __str__(self):
21        """Return name of Polygon"""
22        return ".join([i.name for i in self.points])"
23
24    def __repr__(self):
25        return str(self)
26
27
28 class Quadrilateral(Polygon):
29     def __repr__(self):
30         return "□" + str(self)
31
32     @classmethod
33     def from_polygon(cls, poly):
34         return Quadrilateral(
35             poly.points, poly.sides, poly.aangs, poly.aconv, poly.sconv
36         )
37
38
39 class Triangle(Polygon):
40     def __repr__(self):
41         return "△" + str(self)
42
43     @classmethod
44     def from_polygon(cls, poly):
45         return Triangle(poly.points, poly.sides, poly.aangs, poly.aconv, poly.sconv)
46
```

```

1  """geo/filehandler.py"""
2
3  import zipfile
4  import os
5
6
7  def get_points_from_file(path, temp_folder_path="temp_ggb"):
8      """Return a list of (name,x,y) from a .ggb file"""
9
10     # check file extension
11     if not path.endswith(".ggb"):
12         raise Exception(f"file type for file {path} is not supported. use .ggb instead")
13
14     # extract files from ggb file (actually zip)
15     with zipfile.ZipFile(path, "r") as zip_ref:
16         zip_ref.extractall(temp_folder_path)
17
18     # read main file from unziped folder
19     with open(os.path.join(temp_folder_path, "geogebra.xml"), "r") as f:
20         lines = f.readlines()
21
22     # create a list of point tags
23     point_tags = []
24     is_inside_point_tag = False
25     for line in lines:
26         line = "\n".join(line.strip().split())
27         if line.startswith('<element type="point">'):
28             is_inside_point_tag = True
29             point_tags.append("")
30         if is_inside_point_tag:
31             point_tags[-1] += line + "\n"
32         if line.startswith("</element>"):
33             is_inside_point_tag = False
34
35     # create list of points from tags
36     points = []
37     # points[i] = (name,x,y)
38
39     for point_tag in point_tags:
40         name = point_tag.split("\n")[0].split('label="')[1].split('"')[0]
41         x, y = None, None
42         for line in point_tag.split("\n"):
43             if line.startswith('<coords'):
44                 x = float(line.split('x="')[1].split('"')[0])
45                 y = float(line.split('y="')[1].split('"')[0])
46                 break
47         points.append((name, x, y))
48
49     # remove temp folder
50     for filename in os.listdir(temp_folder_path):
51         os.remove(os.path.join(temp_folder_path, filename))
52     os.rmdir(temp_folder_path)
53
54     # return the sorted points' list
55     return sorted(points)
56
57
58 def print_points_from_file(
59     path=r"ggb_test\test.ggb",
60     temp_folder_path="temp_ggb",
61 ):
62     """Print Point names, xs and ys"""
63     res = get_points_from_file(path, temp_folder_path)
64
65     names = [name for name, *_ in res]
66     xs = [round(x, 2) for _, x, _ in res]
67     ys = [round(y, 2) for _, y in res]

```

```
68     ...     xs = [int(x) if x.is_integer() else x for x in xs]
69     ...     ys = [int(y) if y.is_integer() else y for y in ys]
70     ...     print(f"h.ps(\"{\".join(names)}\", {xs}, {ys})")
71
```

```

1  """geo/handler.py"""
2
3  from numpy import argmax, argmin
4  import itertools
5
6  from geo.abs.point import Point
7  from geo.abs.abssegment import AbsSegment
8  from geo.abs.absangle import AbsAngle
9  from geo.real.realsegment import RealSegment
10 from geo.real.realangle import RealAngle
11 from geo.real.expression import Degree
12 from geo.real.expression import Length
13 from geo.comp.polygon import Polygon, Triangle, Quadrilateral
14 from geo.comp.converter import Convertor
15
16
17 class Handler:
18     MAX_CALC_CYCLES = 10
19
20     # Theorem dictionary that maps theorem number to function
21     THEOREM_DICT = {}
22
23     def __init__(self, *points):
24         """Create a Handler object, keep all Points and collect all Segments"""
25         Handler.initialize_therem_dict()
26
27         self.points = list(points)
28         self.segments = list(set([l for p in self.points for _ , l in p.lines]))
29
30         # a list of proof blocks, initialized every time calc is called
31         self.proof = None
32
33         # angle & segment convertors
34         # initialized in self.init_angles & self.init_segments respectively
35         self.aconv, self.sconv = None, None
36
37     """THEOREMS"""
38
39     def angle_sum_on_line(self):
40         """_th1: the sum of 2 angles on a line is 180°"""
41         for ang180, seg in self.find_all_180_angles():
42             self.abs_equal_exp(
43                 ang180, Degree(False, 180), f"angle upon line {seg} is 180"
44             )
45
46     def vertical_angles(self):
47         """_th2: 2 vertical angles are equal"""
48         for p in self.points:
49             for a1, a2 in itertools.combinations(self.get_angles_around_point(p), 2):
50                 if (
51                     self.is_180_angle(AbsAngle(a1.ray1, p, a2.ray1))
52                     or self.is_180_angle(AbsAngle(a2.ray1, p, a1.ray1))
53                 ) and (
54                     self.is_180_angle(AbsAngle(a1.ray2, p, a2.ray2))
55                     or self.is_180_angle(AbsAngle(a2.ray2, p, a1.ray2))
56                 ):
57                     self.abs_equal_abs(a1, a2, "vertical angles")
58
59     def angle_sum_around_point(self):
60         """_th3: all angles around a point sum up to 360°"""
61         for p in self.points:
62             parts = self.get_angles_around_point(p)
63             if len(parts) != 0:
64                 self.abs_equal_exp(
65                     parts, Degree(False, 360), f"sum of angles around point {p}"
66                 )

```

```

68
69     ...def angles_on_parallel_lines(self):
70     """_ax3 + _th4: corresponding, alternating and consecutive angles between 2
71     parallel lines and a transversal are equal"""
72     if len(self.segments) < 3:
73         ...
74     parallels_transversal = [
75         (p1, p2, t)
76         for p1, p2 in itertools.combinations(self.segments, 2)
77         if p1.is_parallel(p2)
78         for t in self.segments
79         if t.get_intersection_point(p1) is not None
80         and t.get_intersection_point(p2) is not None
81     ]
82
83     for *p, t in parallels_transversal:
84         tp = [t.get_intersection_point(p[0]), t.get_intersection_point(p[1])]
85         if t.get_all_points().index(tp[0]) > t.get_all_points().index(tp[1]):
86             p[0], p[1] = p[1], p[0]
87             tp[0], tp[1] = tp[1], tp[0]
88             # make sure both p are the same direction
89             p[0].better_direction()
90             p[1].better_direction(p[0].get_slope_angle())
91
92     aangs = [[], []]
93     for i in range(2):
94         pe = p[i].get_subsegment_from(tp[i])
95         te = t.get_subsegment_from(tp[i])
96         ps = p[i].get_subsegment_to(tp[i])
97         ts = t.get_subsegment_to(tp[i])
98         order = [pe, te, ps, ts]
99         for j in range(4):
100            if (
101                order[j] is not None
102                and order[j].is_valid()
103                and order[(j + 1) % 4] is not None
104                and order[(j + 1) % 4].is_valid()
105            ):
106                aangs[i].append(
107                    self.get_non_reflex_angle(
108                        order[j].end
109                        if order[j].end != tp[i]
110                        else order[j].start,
111                        tp[i],
112                        order[(j + 1) % 4].end
113                        if order[(j + 1) % 4].end != tp[i]
114                        else order[(j + 1) % 4].start,
115                    )
116                )
117            else:
118                aangs[i].append(None)
119
120    # Corresponding angles
121    for i in range(4):
122        if aangs[0][i] is None or aangs[1][i] is None:
123            continue
124        self.abs_equal_abs(
125            aangs[0][i],
126            aangs[1][i],
127            f"Corresponding angles are equal between {p[0]} || {p[1]} and {t}",
128        )
129
130    # alternate angles
131    for i in range(4):
132        coridx = (i + 2) % 4
133        if aangs[0][i] is None or aangs[1][coridx] is None:

```

```

134     .....continue
135     .....self.abs_equal_abs(
136     .....    aangs[0][i],
137     .....    aangs[1][coridx],
138     .....    f"alternating angles are equal between {p[0]} || {p[1]} and {t}",
139     ..)
140
141     .....# consecutive angles
142     .....for i in range(4):
143     .....    considx = 3 - i
144     .....    if aangs[0][i] is None or aangs[1][considx] is None:
145     .....        continue
146     .....    self.abs_equal_exp(
147     .....        [aangs[0][i], aangs[1][considx]],
148     .....        Degree(False, 180),
149     .....        f"sum of consecutive angles between {p[0]} || {p[1]} and {t}",
150     ..)
151
152 def converse_angles_on_parallel_lines(self):
153     """_th5: Converse of angles between parallel lines (alternate interior /
154     .....corresponding / consecutive)"""
155     if len(self.segments) < 3:
156         return
157
158     .....pos_parallel_transversal = [
159     .....    (p1, p2, t)
160     .....    for p1, p2 in itertools.combinations(self.segments, 2)
161     .....    if not p1.is_parallel(p2) and p1.get_intersection_point(p2) is None
162     .....    for t in self.segments
163     .....    if t.get_intersection_point(p1) is not None
164     .....    and t.get_intersection_point(p2) is not None
165     ..]
166
167     .....for *p, t in pos_parallel_transversal:
168     .....    if p[0].is_parallel(p[1]):
169     .....        continue
170     .....    tp = [t.get_intersection_point(p[0]), t.get_intersection_point(p[1])]
171     .....    if t.get_all_points().index(tp[0]) > t.get_all_points().index(tp[1]):
172     .....        p[0], p[1] = p[1], p[0]
173     .....        tp[0], tp[1] = tp[1], tp[0]
174     .....    # make sure both p are the same direction
175     .....    for s in p:
176     .....        s.better_direction()
177     .....    aangs = [[], []]
178     .....    for i in range(2):
179     .....        pe = p[i].get_subsegment_from(tp[i])
180     .....        te = t.get_subsegment_from(tp[i])
181     .....        ps = p[i].get_subsegment_to(tp[i])
182     .....        ts = t.get_subsegment_to(tp[i])
183     .....        order = [pe, te, ps, ts]
184     .....        for j in range(4):
185     .....            if (
186     .....                order[j] is not None
187     .....                and order[j].is_valid()
188     .....                and order[(j + 1) % 4] is not None
189     .....                and order[(j + 1) % 4].is_valid()
190     ..):
191     .....            aangs[i].append(
192     .....                self.get_non_reflex_angle(
193     .....                    order[j].end
194     .....                    if order[j].end != tp[i]
195     .....                    else order[j].start,
196     .....                    tp[i],
197     .....                    order[(j + 1) % 4].end
198     .....                    if order[(j + 1) % 4].end != tp[i]
199     .....                    else order[(j + 1) % 4].start,
199     ..)

```

```

200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258

```

-)
-else:
-aangs[i].append(**None**)
-# Corresponding angles
-for i in range(4):
-if aangs[0][i] is **None** or aangs[1][i] is **None**:
-continue
-if sum(self.aconv[aangs[0][i]]) == sum(self.aconv[aangs[1][i]]):
-self.set_parallel(
-*p,
-f"**{aangs[0][i]}** = {sum(self.aconv[aangs[0][i]])} = **{aangs[1][i]}**, converse corresponding angles between {p[0]}, {p[1]} and traverse {t}",
-)
-# alternate angles
-for i in range(4):
-altidx = (i + 2) % 4
-if aangs[0][i] is **None** or aangs[1][altidx] is **None**:
-continue
-if sum(self.aconv[aangs[0][i]]) == sum(self.aconv[aangs[1][altidx]]):
-self.set_parallel(
-*p,
-f"**{aangs[0][i]}** = {sum(self.aconv[aangs[0][i]])} = **{aangs[1][altidx]}**, converse alternating angles between {p[0]}, {p[1]} and traverse {t}",
-)
-# consecutive angles
-for i in range(4):
-considx = 3 - i
-if aangs[0][i] is **None** or aangs[1][considx] is **None**:
-continue
-if sum(self.aconv[aangs[0][i]]) == sum(self.aconv[aangs[1][considx]]):
-self.set_parallel(
-*p,
-f"**{aangs[0][i]}** = {sum(self.aconv[aangs[0][i]])} = **{aangs[1][considx]}**, converse consecutive angles between {p[0]}, {p[1]} and traverse {t}",
-)
-
-def angle_sum_in_triangle(self):
-""" th6: The sum of the measures of the interior angles of a triangle is 180° """
-for t in self.find_all_triangles():
-self.abs_equal_exp(
-t.aangs,
-Degree(**False**, 180),
-f"the sum of the interior angles of Δ{t} is 180°",
-)
-
-def angle_sum_in_quadrilateral(self):
-""" th7: The sum of the measures of the interior angles of a Quadrilateral is 360° """
-for q in self.find_all_quadrilateral():
-self.abs_equal_exp(
-q.aangs,
-Degree(**False**, 360),
-f"the sum of the interior angles in quadrilateral {q} is 360°",
-)
-
-def exterior_angle_in_triangle(self):
-""" th8: the size of an exterior angle at a vertex of a triangle equals the sum of the sizes of the interior angles at the other two vertices of the triangle """
-tri_side_seg = [
-(tri, side, seg)
-for tri in self.find_all_triangles()

```

259     .....  

260     .....  

261     .....  

262     .....  

263     .....  

264     .....  

265     .....  

266     .....  

267     .....  

268     .....  

269     .....  

270     .....  

271     .....  

272     .....  

273     .....  

274     .....  

275     .....  

276     .....  

277     .....  

278     .....  

279     .....  

280     .....  

281     .....  

282     .....  

283     .....  

284     .....  

285     .....  

286     .....  

287     .....  

288     .....  

289     .....  

290     .....  

291     .....  

292     .....  

293     .....  

294     .....  

295     .....  

296     .... "CALC"  

297  

298     .....  

299     .....  

300     .....  

301     .....  

302     .....  

303     .....  

304     .....  

305     .....  

306     .....  

307     .....  

308     .....  

309  

310     .....  

311     .....  

312     .....  

313     .....  

314     .....  

315     .....  

316     .....  

317     .....  

318     .....  

319     .....  

320     .....  

321     .....  

322     .....  

323  

324     .....  

325

```

```

326     ...def calc(
327     ...     self,
328     ...     inita=True,
329     ...     inits=True,
330     ...     print_proof=True,
331     ...     after_init=lambda: None,
332     ...     use_theorems=None,
333     ...):
334         """Get information through theroms and given data"""
335         self.proof = []
336         if inita:
337             self.init_angles()
338         if inits:
339             self.init_segments()
340
341         after_init()
342
343         if print_proof:
344             print(self.rangles)
345
346         last_len = -1
347         for _ in range(self.MAX_CALC_CYCLES):
348             last_len = len(self.proof)
349
350             if use_theorems is None:
351                 for th in self.THEOREM_DICT.keys():
352                     self.apply_theorem(th)
353             else:
354                 for th in use_theorems:
355                     self.apply_theorem(th)
356
357             if len(self.proof) == last_len:
358                 break
359
360             if print_proof:
361                 self.print_proof()
362
363     def apply_theorem(self, th):
364         """Apply Theorem th"""
365         theorem_func = self.THEOREM_DICT[th]
366         theorem_func(self)
367
368     def print_proof(self):
369         """Prints self.proof"""
370         print("".join(self.proof))
371
372         """SET EQUAL/PARALLEL"""
373
374     def abs_equal_abs(self, abs1, abs2, reason="given"):
375         """Assume Abs1=Abs2 and act apon it"""
376         message = ""
377         abs_arr = [abs1, abs2]
378         abs_sum_strs = [None, None]
379         for i in range(2):
380             if not isinstance(abs_arr[i], list):
381                 abs_arr[i] = [abs_arr[i]]
382                 abs_sum_strs[i] = " ".join(map(str, abs_arr[i]))
383
384             message += f"{abs_sum_strs[0]} = {abs_sum_strs[1]} ({reason})\n"
385
386             reals = [sum([self.conv(a) for a in abs_lst], []) for abs_lst in abs_arr]
387             # if some angles were not elementry
388             if sum([len(lst) for lst in reals]) > sum([len(lst) for lst in abs_arr]):
389                 message += " ".join([
390                     " ".join([str(abs(r)) for r in reals[i]]) for i in range(2)])
391
392             message += "\n"

```

```

393
394     max_real = max(*reals[0], *reals[1])
395     if abs(max_real) in reals[0]:
396         minus_group, plus_group = reals
397     else: # elif abs(max_real) in reals[1]:
398         plus_group, minus_group = reals
399         minus_group = [r for r in minus_group if abs(r) != max_real]
400     if len(minus_group) > 0:
401         message += (
402             f"\n{abs(max_real)} = {'+'.join([str(abs(r)) for r in plus_group])} "
403             f"- ({'+'.join([str(abs(r)) for r in minus_group])})\n"
404         )
405
406     message += (
407         f"\n{abs(max_real)} = "
408         f"\n{'+'.join([str(r.get_expression()) for r in plus_group])}"
409     )
410
411     should_add_parentheses = len(minus_group) > 1 or (
412         len(minus_group) == 1
413         and len(minus_group[0].get_expression().value.keys()) > 1
414     )
415     should_not_add_parentheses = len(minus_group) == 1
416     if should_add_parentheses:
417         message += (
418             f"\n{'+'.join([str(r.get_expression()) for r in minus_group])}) "
419         )
420     elif should_not_add_parentheses:
421         message += f"\n{minus_group[0].get_expression()}"
422
423     message += "\n(eval)\n"
424     if self.abs_equal_exp(abs(max_real), sum(plus_group) - sum(minus_group)):
425         self.proof[-1] = message + self.proof[-1]
426
427     def abs_equal_exp(self, aabs, exp, reason="given"):
428         """Assume (Abs = Exp) and act upon it. Return if new data was found"""
429         message = ""
430         reals = []
431
432         # if aabs is elementry
433         if not isinstance(aabs, list) and self.is_elementry(aabs):
434             reals = self.conv(aabs)
435             message += f"\n{aabs} = {exp} ({reason})\n"
436         else:
437             if not isinstance(aabs, list):
438                 aabs = [aabs]
439             reals = sum([self.conv(a) for a in aabs], [])
440
441             message += (
442                 f"\n{exp} = {'+'.join(map(str, aabs))} = "
443                 f"\n{'+'.join(map(str, map(abs, reals)))} ({reason})"
444                 f", the whole is the sum of its parts)\n"
445             )
446
447         # if we already know, there is no point to continue
448         if all([r.isknown() for r in reals]):
449             return False
450
451         # the real that will be evaluated
452         eval_real = max(reals)
453         # a copy of eval_real
454         eval_real_copy = eval_real.new_copy()
455         # all reals accept eval_real
456         other_reals = [r for r in reals if abs(r) != abs(eval_real)]
457         # copy of other_reals
458         other_reals_copy = [r.new_copy() for r in other_reals]
459

```

```

460     .....# reduce to (eval_real = exp - sum(other_reals))
461     .....if len(other_reals) > 0:
462     .....    if len(other_reals) == 1:
463     .....        other_reals_sum = str(abs(other_reals[0]))
464     .....    else:
465     .....        other_reals_sum = f"({'} + '.join(map(str, map(abs, other_reals))))"
466     .....    message += f"\n{abs(eval_real)} = {exp} - {other_reals_sum} (same)\n"
467
468     .....res = self.real_equal_exp(eval_real, exp - sum(other_reals))
469
470     .....if len(res) <= 1:
471     .....    return False # nothing has been discovered
472     .....if len(res) == 2 and len(other_reals) == 0:
473     .....    pass # skip unnecessary printing
474     .....else:
475     .....    # - (sum(other_reals))
476     .....    minus_other_reals_copy_sum = ""
477     .....    should_add_parentheses = len(other_reals_copy) > 1 or (
478     .....        len(other_reals_copy) == 1
479     .....        and len(other_reals_copy[0].get_expression().value.keys()) > 1
480     .....    )
481     .....    should_not_add_parentheses = len(other_reals) == 1
482     .....    if should_add_parentheses:
483     .....        # add parentheses
484     .....        minus_other_reals_copy_sum = (
485     .....            f" - (" +
486     .....            f"{' + '.join([str(r.get_expression()) for r in other_reals_copy])}" +
487     .....            f")"
488     .....        )
489
490     .....elif should_not_add_parentheses:
491     .....    # no parentheses
492     .....    minus_other_reals_copy_sum = f" - {other_reals_copy[0].get_expression()}"
493
494     .....# check for other repercussions
495     .....if len(res) == 2 and eval_real == exp - sum(other_reals_copy):
496     .....    # no other repercussions from switching
497
498     .....    # message += evaluation -> final value
499     .....    message += (
500     .....        f"\n{abs(eval_real)} = {exp} {minus_other_reals_copy_sum} (eval)"
501     .....        f"\n-> {abs(eval_real)} = {eval_real.get_expression()} (calc)\n"
502     .....    )
503     .....else:
504     .....    # there are other repercussions from switching
505
506     .....    # message += evaluation (eval)
507     .....    message += (
508     .....        f"\n{eval_real_copy.get_expression()} = {exp} {"
509     .....            minus_other_reals_copy_sum +
510     .....            f"\n (eval)\n"
511     .....        )
512     .....    # message += reduction (same)
513     .....    message += (
514     .....        f"\n{eval_real_copy.get_expression()} = "
515     .....        f"\n{exp - sum(other_reals_copy)} (same)\n"
516     .....    )
517     .....    # message += (var = found) (found var)
518     .....    var_found = Degree(False, {res[0][0]: 1})
519     .....    message += f"\n{var_found} = {res[0][1]} (found var {var_found})\n"
520     .....    # for all reals affected
521     .....    for pre_exp, r in res[1:]:
522     .....        # message += (abs = proven = eval -> abs = final)
523     .....        message += (
524     .....            f"\n{r} = {pre_exp} = "
525     .....            f"\n{pre_exp.__str__(res[0][0], f'({res[0][1]})')}"
526     .....            f"\n-> {r} = {sum(self.conv(r))}\n"

```

```

526     .....
527     .... self.proof.append(message)
528     .... return True
529
530     def real_equal_exp(self, real, exp):
531         """Set real to be exp, Return list of (preExp, affected Reals), list[0] =
532             (varsswitched.key,switchval)"""
533         # if real has no value, set it to be exp
534         if real.get_expression() is None:
535             real.set_value(exp)
536         return []
537         # if real's value is the same as exp, we cannot continue
538         switch_val = exp - real.get_expression()
539         if switch_val == 0:
540             return []
541
542         # choose key,value to switch in all reals
543         max_key = max(switch_val.value.keys())
544         switch_val = switch_val / (-switch_val.value[max_key])
545         del switch_val.value[max_key]
546         """example:
547             exp = real
548             3x = 7y
549             switch_val = exp - real = 3x-7y
550             switch_val = (3x-7y)/7, remove y -> y = 3/7x
551             """
552         # that means every (1 max_key = switch_val)
553         res = [(max_key, switch_val)]
554         if isinstance(real, RealAngle):
555             reals = self.rangles.values()
556         elif isinstance(real, RealSegment): # else:
557             reals = self.rsegments.values()
558
559         for new_real in reals:
560             pre_val = new_real.get_expression().new_copy()
561             # make the switch for new_real
562             new_real.get_expression().switch(max_key, switch_val)
563             # if the value has changed as a result of the switch
564             if pre_val != new_real.get_expression():
565                 # if abs(new_real) is real, insert it at the front
566                 if abs(new_real) == abs(real):
567                     res.insert(1, (pre_val, abs(real)))
568                 else:
569                     res.append((pre_val, abs(new_real)))
570
571     def set_parallel(self, p1, p2, reason="given"):
572         """Set p1 || p2 based on resaon"""
573         for s in self.segments:
574             if s.is_subsegment(p1):
575                 p1 = s
576             if s.is_subsegment(p2):
577                 p2 = s
578
579             p1.set_parallel(p2)
580             self.proof.append(f"{str(p1)} || {str(p2)} ({reason})\n")
581
582     """ GENERIC REAL/ABS METHODS"""
583
584     def conv(self, aabs):
585         """Convert abs to real using aconv/sconv"""
586         if isinstance(aabs, AbsAngle):
587             return self.aconv[aabs]
588         if isinstance(aabs, AbsSegment):
589             return self.sconv[aabs]
590
591     def is_elementry(self, aabs):

```

```

592     """Return if aabs is elementry angle/segment"""
593     return len(self.conv(aabs)) == 1
594
595     """ BASIC CONVERTOR METHODS"""
596
597     def disassemble_angle(self, ang):
598         """Return a list of all the elementary AbsAngles that are included in ang"""
599         ang = self.get_better_name_angle(ang)
600         sub_angles = self.get_angles_around_point(ang.vertex)
601         i = 0
602         found = False
603         while i < len(sub_angles):
604             if sub_angles[i].ray1 == ang.ray1:
605                 found = True
606                 break
607             i += 1
608
609         if not found:
610             raise Exception(
611                 f"didn't find the ray of the given angle ({ang}) around the vertex ({ang.vertex.name})"
612             )
613
614         res = []
615         while sub_angles[i].ray1 != ang.ray2:
616             res.append(sub_angles[i])
617             i = (i + 1) % len(sub_angles)
618
619         return res
620
621     def disassemble_segment(self, seg):
622         """Return list of all SubSegments (all elementary AbsSegments that are included in seg)"""
623         seg = self.get_full_seg(seg.start, seg.end)
624         points = seg.get_all_points()
625         return [AbsSegment(*points[i:i+2]) for i in range(len(points)-1)]
626
627     def get_rang(self, aang):
628         """Return RealAngle from elementry AbsAngle"""
629         return self.rangles[aang]
630
631     def get_rseg(self, seg):
632         """Return RealSegment from elementry AbsSegment"""
633         return self.rsegments[seg]
634
635     """ POLYGONS """
636
637     def find_all_polygons(self, num_of_sides):
638         """Return list of all polygons with num_of_sides sides"""
639         res = []
640         for point_list in itertools.combinations(self.points, num_of_sides):
641             for perm in self.circle_perm(point_list):
642                 if self.is_pointlist_poly(list(perm)):
643                     res.append(self.create_poly_from_pointlist(list(perm)))
644
645         return res
646
647     def is_pointlist_poly(self, pointlist):
648         """Checks if pointlist is a polygon"""
649         sides = [
650             AbsSegment(pfrom, pto)
651             for pfrom, pto in zip(pointlist, pointlist[1:] + pointlist[:1])
652         ]
653
654         # if not (all segments exist) -> no poly
655         if not all(
656             [any([seg.is_subsegment(side) for seg in self.segments]) for side in sides]
657         ):

```

```

657     ..... return False
658
659     ..... # if (3 points on same line) -> no poly
660
661     ..... sides = [self.get_full_seg(seg.start, seg.end) for seg in sides]
662     ..... if any(
663     ..... [
664     .....     [p in seg.get_all_points() for p in pointlist].count(True) > 2
665     .....     for seg in self.segments
666     ..... ]
667     ..):
668     .....     return False
669     ..... # if (non-neighbor sides intersect) -> no poly
670     ..... if any(
671     ..... [
672     .....     side.get_intersection_point(non_adj_side) is not None
673     .....     for side_idx, side in enumerate(sides)
674     .....     for non_adj_idx, non_adj_side in enumerate(sides)
675     .....     if non_adj_idx != side_idx
676     .....     and non_adj_idx != (side_idx + 1) % len(sides)
677     .....     and non_adj_idx != (side_idx - 1 + len(sides)) % len(sides)
678     ..... ]
679     ..):
680     .....     return False
681     ..... return True
682
683 def create_poly_from_pointlist(self, pointlist):
684     """Create Polygon from pointlist (pointlist is a polygon)"""
685     sides = [
686         ..... self.get_full_seg(pfrom, pto)
687         ..... for pfrom, pto in zip(pointlist[1:], pointlist[:1])
688     ]
689     ..... # choose aangs based on most non-reflex angles
690     aangs = [[], []]
691     counter = [0, 0]
692     for pfrom, vertex, pto in zip(
693         ..... pointlist[-1:] + pointlist[:-1],
694         ..... pointlist,
695         ..... pointlist[1:] + pointlist[:1],
696     ):
697         ..... from_seg = self.get_full_seg(pfrom, vertex)
698         ..... to_seg = self.get_full_seg(vertex, pto)
699
700         ..... aangs[0].append(AbsAngle(from_seg, vertex, to_seg))
701         ..... aangs[1].append(AbsAngle(to_seg, vertex, from_seg))
702
703     ..... if len(pointlist) < 6:
704     .....     for aang in aangs[0]:
705     .....         pfrom = aang.get_start_point()
706     .....         vertex = aang.vertex
707     .....         pto = aang.get_end_point()
708     .....         non_reflex_aang = self.get_non_reflex_angle(pfrom, vertex, pto)
709     .....         if non_reflex_aang == aang:
710     .....             counter[0] += 1
711         ..... else:
712             counter[1] += 1
713
714     ..... # polygon with n sides has a maximum of (n-3) reflex angles
715     aangs = aangs[argmax(counter)]
716     ..... else:
717         ..... slope_angle_sums = [0, 0]
718         ..... for i in range(2):
719         .....     slope_angle_sums[i] = sum(
720         .....         [aang.get_angle_size_from_coordinates() for aang in aangs[i]]
721         .....     )
722         ..... aangs = aangs[argmin(slope_angle_sums)]
723

```

```

724     ..... return Polygon(pointlist, sides, aangs, self.aconv, self.sconv)
725
726     def find_all_triangles(self):
727         ..... [Triangle.from_polygon(p) for p in self.find_all_polygons(3)]
728
729     def find_all_quadrilateral(self):
730         ..... [Quadrilateral.from_polygon(p) for p in self.find_all_polygons(4)]
731
732     """ BASIC ABS_SEGMENT METHODS"""
733
734     def get_full_seg(self, startpoint, endpoint):
735         """Return AbsSegment from 2 points"""
736         maybeline = AbsSegment(startpoint, endpoint)
737         for i in self.segments:
738             if i.is_subsegment(maybeline):
739                 ..... return i.get_subsegment(startpoint.name + endpoint.name)
740
741     return None
742
743     def is_elementry_seg(self, seg):
744         ..... return len(self.disassemble_segment(seg)) == 1
745
746     """ BASIC ABS_ANGLE METOHDS """
747
748     def get_non_reflex_angle(self, pfrom, vertex, pto):
749         """Return non reflex angle based on 3 points"""
750         ans = [
751             AbsAngle(
752                 ..... self.get_full_seg(pfrom, vertex),
753                 ..... vertex,
754                 ..... self.get_full_seg(vertex, pto),
755                 ..... )
756             ]
757         ..... ans.append(AbsAngle(ans[0].ray2, vertex, ans[0].ray1))
758         # ans[0] and ans[1] are the two options, both close a circle around the vertex
759
760         for i in range(2):
761             a = ans[i]
762             arr = self.disassemble_angle(a)
763
764             a_sum = sum(self.aconv[a])
765             # if a is known to be less than 180, return a
766             if a_sum.ishknown() and a_sum < 180:
767                 ..... return a
768             for ii in range(len(arr)):
769                 for j in range(ii, len(arr)):
770                     # if a contains a subsum that is >= 180, return the other ans
771                     a_sum = sum([self.rangles[aa] for aa in arr[ii:(j+1)]])
772                     if a_sum == 180 or all(
773                         [val > 0 for key, val in (a_sum - 180).value.items()])
774                     ..... return ans[0] if i == 1 else ans[1]
775
776             # use x,y to understand which is acute
777             return min(ans, key=lambda aa: aa.get_angle_size_from_coordinates())
778
779     def get_angles_around_point(self, p):
780         """Return a list of all the elementary AbsAngles around a point"""
781         if len(p.lines) == 0:
782             ..... []
783             elif len(p.lines) == 1 and p in (
784                 ..... p.lines[0][0].start,
785                 ..... p.lines[0][0].end,
786                 ..):
787                 ..... return []
788
789             rays = [line for line, _ in p.lines]
790

```

```

791     .....     return [AbsAngle(r1, p, r2) for r1, r2 in zip(rays, rays[1:] + rays[:1])]
792
793     def get_angles(self):
794         """Return a list of all the elementary AbsAngles"""
795         res = []
796         for p in self.points:
797             res += self.get_angles_around_point(p)
798         return res
799
800     def get_better_name_angle(self, ang):
801         """Find a better representation for the AbsAngle and return it"""
802         true_rays = []
803         for ray in [ang.ray1, ang.ray2]:
804             # find the biggest segment from vertex instead of ray
805             if ray in self.segments:
806                 true_rays.append(ray)
807             else:
808                 for seg in self.segments:
809                     if seg.is_subsegment(ray):
810                         l = seg
811                         break
812                     otherpoint = ray.end if ang.vertex == ray.start else ray.start
813                     opt1 = l.get_subsegment_from(ang.vertex)
814                     opt2 = l.get_subsegment_to(ang.vertex)
815                     if otherpoint in opt1.get_all_points():
816                         true_rays.append(opt1)
817                     else:
818                         true_rays.append(opt2)
819
820         return AbsAngle(true_rays[0], ang.vertex, true_rays[1])
821
822     def find_all_180_angles(self):
823         """Return a list of (180° AbsAngles, segment on)"""
824         res = []
825         for l in self.segments:
826             for p in l.midpoints:
827                 res.append(
828                     (
829                         AbsAngle(l.get_subsegment_to(p), p, l.get_subsegment_from(p)),
830                         l,
831                     )
832                 )
833                 res.append(
834                     (
835                         AbsAngle(l.get_subsegment_from(p), p, l.get_subsegment_to(p)),
836                         l,
837                     )
838                 )
839         return res
840
841     def is_180_angle(self, ang):
842         """Check if an angle is 180°"""
843         if isinstance(ang, RealAngle):
844             if ang == 180:
845                 return True
846             maybeline = AbsSegment(ang.get_start_point(), ang.get_end_point())
847             maybeline.set_midpoints(ang.vertex)
848
849             return any(i.is_subsegment(maybeline) for i in self.segments)
850
851     @classmethod
852     def initialize_them_dict(cls):
853         """Initialize THEOREM_DICT if needed"""
854         if cls.THEOREM_DICT != {}:
855             return
856         cls.THEOREM_DICT = {
857             1: cls.angle_sum_on_line,

```

```
858     ..... 2: cls.vertical_angles,
859     ..... 3: cls.angle_sum_around_point,
860     ..... 4: cls.angles_on_parallel_lines,
861     ..... 5: cls.converse_angles_on_parallel_lines,
862     ..... 6: cls.angle_sum_in_triangle,
863     ..... 7: cls.angle_sum_in_quadrilateral,
864     ..... 8: cls.exterior_angle_in_triangle,
865     .....
866
867     ..... @staticmethod
868     def circle_perm(lst):
869         """Return all possible permutation to put lst on a circle"""
870         res = [i for i in itertools.permutations(lst[1:])]
871         i = 0
872         while i < len(res):
873             res.remove(res[i][::-1])
874             i += 1
875         res = [(lst[0],) + i for i in res]
876
877         return res
```



```

68             res.append(seg.get_subsegment(name))
69             break
70         else: # only executed if the inner loop did NOT break
71             self.segments.append(
72                 AbsSegment(self.p(name[0]), self.p(name[-1]), True)
73             )
74             self.segments[-1].set_midpoints(*[self.p(i) for i in name[1:-1]])
75             res.append(self.segments[-1])
76     return tuple(res) if len(res) > 1 else res[0]
77
78     def a(self, *names):
79         """Create new AbsAngle(s) (if needed) and return it (them)"""
80         if len(names) == 0:
81             return None
82         res = []
83         for name in names:
84             if isinstance(name, AbsAngle):
85                 res.append(name)
86             else:
87                 res.append(
88                     self.g().get_better_name_angle(
89                         AbsAngle(self.s(name[:-1]), self.p(name[1]), self.s(name[1:])))
90                 )
91         return tuple(res) if len(res) > 1 else res[0]
92
93     def g(self):
94         """Create new Handler (if needed) and return it"""
95         if self.geo is None or len(self.geo.points) < len(self.points):
96             self.geo = Handler(*self.points)
97         return self.geo
98
99
100    def given(self, cls, symbol):
101        """Create new given Expression (if needed) and return it"""
102        if not issubclass(cls, Expression):
103            raise TypeError(f"{cls} cannot be given, must be Expression")
104        try:
105            return self.given_dict[(cls, symbol)]
106        except KeyError:
107            self.given_dict[(cls, symbol)] = cls.given(symbol)
108        return self.given_dict[(cls, symbol)]
109
110    def tri(self, *names):
111        """Create new triangle(s) by name"""
112        for name in names:
113            self.poly(name)
114
115    def poly(self, *names):
116        """Create Polygon based on name"""
117        for name in names:
118            p_arr = [self.p(p_name) for p_name in name]
119            for pfrom, pto in zip(p_arr, p_arr[1:] + p_arr[:1]):
120                self.s(f"{pfrom}{pto}")
121
122    """
123
124    def tri_inner_line(self, tri, name):
125        """Create inner line inside triangle, Return useful objects:
126        pfrom -- one of the triangle's vertex
127        pto -- a point from inner line that is on one of the triangle's sides
128        across_side -- the side of the triangle that is across pfrom
129        other_points -- the other triangle points that are not pfrom
130        """
131        # Create inner line
132        self.s(name)
133        # Get return values
134        pfrom = self.p(self.get_common_point(tri, name))

```

```

135     .....other_points = [self.p(pname) for pname in tri if pname != str(pfrom)]
136     .....across_side = self.s("".join([str(p) for p in other_points]))
137     .....pto = self.get_common_point(name, across_side.get_all_points())
138     .....return pfrom, pto, across_side, other_points
139
140     .....def tri_med(self, tri, name):
141     .....    """Build median in existing triangle (end point should already exists)"""
142     .....    # Create median segment
143     .....    pfrom, pto, across_side, *_ = self.tri_inner_line(tri, name)
144
145     .....    # Set median to be the middle of the triangle side
146     .....    def func(h, across_side, pto, pfrom, name, tri):
147     .....        h.g().abs_equal_abs(
148     .....            across_side.get_subsegment_to(pto),
149     .....            across_side.get_subsegment_from(pto),
150     .....            f"given that {name} is median to {across_side} in Δ{tri}",
151     .....        )
152
153     .....    self.to_inits.append(partial(func, self, across_side, pto, pfrom, name, tri))
154
155     .....def tri_angbi(self, tri, name):
156     .....    """Build Angle bisector in existing triangle (end point already exists)"""
157     .....    # Create angle bisector
158     .....    pfrom, pto, _, other_points = self.tri_inner_line(tri, name)
159
160     .....    # Set the two angles as equals
161     .....    def func(h, other_points, pfrom, pto, name, tri):
162     .....        h.g().abs_equal_abs(
163     .....            h.g().get_non_reflex_angle(other_points[0], pfrom, pto),
164     .....            h.g().get_non_reflex_angle(other_points[1], pfrom, pto),
165     .....            f"given that {name} is angle bisector of angle {h.g()}."
166     .....            get_non_reflex_angle(other_points[0], pfrom, other_points[1]) in Δ{tri}",
167     .....        )
168
169     .....    self.to_inita.append(partial(func, self, other_points, pfrom, pto, name, tri))
170
171     .....def tri_alt(self, tri, name):
172     .....    """Build Altitude in existing triangle (end point already exists)"""
173     .....    # Create altitude
174     .....    *_ , across_side, _ = self.tri_inner_line(tri, name)
175
176     .....    # Set altitude to be perpendicular to across_side
177     .....    self.perps(
178     .....        across_side,
179     .....        name,
180     .....        f"given that {name} is an altitude to side {across_side} in Δ{tri}",
181     .....    )
182
183     .....def tri_segbis(self, tri, name, side):
184     .....    """Build Segment bisector in existing triangle (end point already exists)"""
185     .....    # create segment bisector
186     .....    self.s(name)
187
188     .....    # the midpoint in segment
189     .....    pmid = self.p(self.get_common_point(name, self.s(side).midpoints))
190     .....    # append Equality of halves
191     .....    def eq_of_halves(h, side, pmid, name, tri):
192     .....        h.g().abs_equal_abs(
193     .....            h.s(side).get_subsegment_to(pmid),
194     .....            h.s(side).get_subsegment_from(pmid),
195     .....            f"given that {name} is segment bisector to side {side} in Δ{tri}",
196     .....        )
197
198     .....    self.to_inits.append(partial(eq_of_halves, self, side, pmid, name, tri))
199
200     .....    # set perpendicularity of side and segbi
201     .....    self.perps(

```

```

201     ..... side,
202     ..... name,
203     ..... f"given that {name} is segment bisector to side {side} in  $\triangle$ {tri}",
204     ....)
205
206     .... """Init / Calc"""
207
208     .... def inita(self):
209     ..... """Call self.geo.init_angles() if needed"""
210     ..... if not self.did_calc:
211     .....     self.g().init_angles()
212
213     .... def inits(self):
214     ..... """Call self.geo.init_segments()"""
215     ..... if not self.did_calc:
216     .....     self.g().init_segments()
217
218     .... def to_init(self):
219     ..... """Perform all self.to_inita/to_inits functions"""
220     ..... for f in self.to_inita:
221     .....     f()
222     ..... self.to_inita = []
223     ..... for f in self.to_inits:
224     .....     f()
225     ..... self.to_inits = []
226
227     .... def calc(self, print_proof=True, use_theorems=None):
228     ..... """Call self.geo.calc() & perform init"""
229     ..... self.inita()
230     ..... self.inits()
231     ..... self.g().calc(False, False, print_proof, self.to_init, use_theorems)
232     ..... self.did_calc = True
233
234     .... """Set/Get/Equal Expressions"""
235
236     .... def seta(self, name, deg, reason="given"):
237     ..... """Set value of angle in geo to deg"""
238
239     ..... def func(h, name, deg, reason):
240     .....     h.g().abs_equal_exp(h.a(name), deg, reason)
241
242     ..... self.to_inita.append(partial(func, self, name, deg, reason))
243
244     .... def sets(self, name, leng, reason="given"):
245     ..... """Set value of segment in geo to leng"""
246
247     ..... def func(h, name, leng, reason):
248     .....     h.g().abs_equal_exp(h.s(name), leng, reason)
249
250     ..... self.to_inits.append(partial(func, self, name, leng, reason))
251
252     .... def geta(self, *names):
253     ..... """Get angle(s) value(s) from geo by name"""
254     ..... if len(names) == 0:
255     .....     return None
256
257     ..... if not self.did_calc:
258     .....     self.calc()
259
260     ..... res = []
261     ..... for name in names:
262     .....     res.append(sum(self.g().aconv[self.a(name)]))
263     ..... return tuple(res) if len(res) > 1 else res[0]
264
265     .... def gets(self, *names):
266     ..... """Get segment(s) value(s) from geo by name"""
267     ..... if len(names) == 0:

```

```

268     .....     return None
269
270     ..... if not self.did_calc:
271     .....     self.calc()
272
273     ..... res = []
274     ..... for name in names:
275     .....     res.append(sum(self.g().sconv[self.s(name)]))
276     ..... return tuple(res) if len(res) > 1 else res[0]
277
278     ..... def equala(self, name1, name2, reason="given"):
279     .....     """Set angle name1 to be equal to name2"""
280
281     .....     def func(h, name, deg, reason):
282     .....         h.g().abs_equal_abs(h.a(name1), h.a(name2), reason)
283
284     .....     self.to_inita.append(partial(func, self, name1, name2, reason))
285
286     ..... def equals(self, name1, name2, reason="given"):
287     .....     """Set seg name1 to be equal to name2"""
288
289     .....     def func(h, name, deg, reason):
290     .....         h.g().abs_equal_abs(h.s(name1), h.s(name2), reason)
291
292     .....     self.to_inita.append(partial(func, self, name1, name2, reason))
293
294     ..... """Segments"""
295
296     ..... def conts(self, og, new):
297     .....     """Continue AbsSegment og to new"""
298     .....     og = self.s(og)
299     .....     if new.index(str(og.start)) > new.index(str(og.end)):
300     .....         new = new[::-1]
301     .....     if og.start.name == new[0] and og.end.name == new[-1]:
302     .....         new_p = [self.p(i) for i in new]
303     .....         og.update_midpoints(*new_p[1:-1])
304     .....     else:
305     .....         for point in og.get_all_points():
306     .....             point.remove_line(og)
307     .....         self.s(new)
308
309     ..... def perps(self, seg1, seg2, reason="given"):
310     .....     """Set segments to be perpendicular"""
311     .....     seg1 = self.s(seg1)
312     .....     seg2 = self.s(seg2)
313     .....     # intersection point
314     .....     interp = seg1.get_intersection_point(seg2)
315
316     .....     def func(h, seg1, interp, seg2, reason):
317     .....         h.seta(
318     .....             h.g().get_non_reflex_angle(
319     .....                 seg1.start if seg1.start != interp else seg1.end,
320     .....                 interp,
321     .....                 seg2.start if seg2.start != interp else seg2.end,
322     .....                 ),
323     .....                 90,
324     .....                 reason=reason,
325     .....             )
326
327     .....     self.to_inita.append(partial(func, self, seg1, interp, seg2, reason))
328
329     ..... def paras(self, *names):
330     .....     """Set segments to be parallel by their names"""
331     .....     if len(names) < 2:
332     .....         return
333     .....     first = self.s(names[0])
334     .....     for name in names[1:]:

```

```
335     .....first.set_parallel(self.s(name))
336
337     def isparas(self, name1, name2):
338         if not self.did_calc:
339             self.calc()
340         return self.s(name1).is_parallel(self.s(name2))
341
342     """Other"""
343
344     def get_common_point(self, iter1, iter2):
345         lst = list(
346             set([self.p(i) for i in iter1]).intersection(
347                 set([self.p(i) for i in iter2]))
348         )
349
350         return lst[0] if len(lst) > 0 else None
351
```

```
1  """geo/problemcollection.py"""
2
3  from tqdm import tqdm
4
5
6  class ProblemCollection:
7      @classmethod
8      def all(cls):
9          """Return all problem functions from class"""
10         ans = []
11         for p in dir(cls):
12             attr = getattr(cls, p)
13             if not p.startswith("__") and callable(attr):
14                 if p not in ["all", "check_all", "check_prob"]:
15                     ans.append(attr)
16
17         return ans
18
19     @classmethod
20     def check_all(cls, print_proof=False):
21         """Check all problem functions from class"""
22         problems = tqdm(cls.all())
23         for prob in problems:
24             try:
25                 is_solved = ProblemCollection.check_prob(prob, print_proof)
26                 if not is_solved:
27                     problems.close()
28                     print("Error in prob", prob)
29                     break
30             except Exception as e:
31                 problems.close()
32                 print(e)
33                 print("Error in problem", prob)
34                 break
35             else:
36                 print("All tests are valid!")
37
38     @staticmethod
39     def check_prob(func, print_proof=False):
40         """Check a single problem function"""
41         ans = func(print_proof)
42         return ans[0] == ans[1]
```

```

1  """proofs.py"""
2
3  from geo.problemcollection import ProblemCollection
4  from geo.filehandler import print_points_from_file as _pggb
5  from geo.helper import Helper
6  from geo.real.expression import Degree
7
8  # http://mathbitsnotebook.com/JuniorMath/Geometry/GEORules.html
9
10 """
11 Axioms:
12
13 _ax1: All straight angles are congruent ( $180^\circ$ ).
14 _ax2: The whole is equal to the sum of its parts.
15 _ax3: A pair of alternate interior angles between 2 parallel lines and a transversal are equal
16 """
17
18 """
19 Theorems:
20 _th1: The sum of 2 angles on a line is  $180^\circ$  (adjacent supplementary angles / linear pair)
21 _th2: Two vertical angles are equal
22 _th3: All angles around a point sum up to  $360^\circ$ 
23 _th4: Corresponding angles are equal and the sum of two consecutive interior angles is  $180^\circ$  (2pl&t)
24 _th5: Converse of angles between parallel lines (alternate interior / corresponding / consecutive)
25 _th6: The sum of the measures of the interior angles of a triangle is  $180^\circ$ 
26 _th7: The sum of the measures of the interior angles of a Quadrilateral is  $360^\circ$ 
27 _th8: The size of an exterior angle at a vertex of a triangle equals the sum of the sizes of the interior angles at the other two vertices of the triangle
28 """
29
30
31 class ProofCollection(ProblemCollection):
32     @classmethod
33     def all(cls):
34         ans = []
35         for p in dir(cls):
36             attr = getattr(cls, p)
37             if p.startswith("t") and callable(attr):
38                 if p not in ["all", "check_all", "check_prob"]:
39                     ans.append(attr)
40         return ans
41
42     @staticmethod
43     def th1(print_proof=False):
44         """_ax1 & _ax2 -> _th1"""
45         h = Helper()
46         h.ps_from_file(r"ggb_files\th1.ggb")
47         h.s("ACB", "CD")
48         h.calc(print_proof, use_theorems=[])
49         # by ax1,  $\angle ACB = 180^\circ$ 
50         ACB = h.a("ACB")
51         # by ax2,  $\angle ACB$  is the sum of its parts
52         parts = h.g().disassemble_angle(ACB)
53         # therefore, we know  $\text{sum}(\text{parts}) = 180$ 
54         if print_proof:
55             print(f"\{ACB}\} = 180, \{ACB\} = \{ ' + '.join(map(str, parts)) \}")
56             print(f"\square")
57             print(f"\{ ' + '.join(map(str, parts)) \} = 180")
58
59         return None, None
60
61     @staticmethod
62     def th2(print_proof=False):
63         """_th1 -> _th2"""

```

```

64     h = Helper()
65     h.ps_from_file(r"ggb_files\th2.ggb")
66     h.s("AEB", "CED")
67     h.calc(print_proof, use_theorems=[1])
68     if print_proof:
69         print(f"{h.a('AEC')} = {h.get_a('AEC')}, {h.a('BED')} = {h.get_a('BED')}")
70         print(f"\u2225")
71         print(f"{h.a('AEC')} = {h.a('BED')}")
72     return (h.get_a("AEC") == h.get_a("BED"),), (True,)
73
74     @staticmethod
75     def th3(print_proof=False):
76         """_th1 -> _th3"""
77         h = Helper()
78         h.ps_from_file(r"ggb_files\th3.ggb")
79         h.s("AEB", "CED")
80         h.calc(print_proof, use_theorems=[1])
81         E = h.p("E")
82         aangs_around_E = h.g().get_angles_around_point(E)
83         sum_aangs_around_E = sum(h.get_a(*aangs_around_E))
84         if print_proof:
85             print(
86                 "\n" + ".join(map(str, aangs_around_E)),
87                 "=",
88                 f"({'}" + ('.join(map(str, h.get_a(*aangs_around_E))))})",
89                 "=",
90                 sum_aangs_around_E,
91             )
92             print("\u2225")
93             print("\n" + ".join(map(str, aangs_around_E)), =", sum_aangs_around_E)
94
95         return (sum_aangs_around_E,), (360,)
96
97     @staticmethod
98     def th4(print_proof=False):
99         """_th1 & _ax3 -> _th4"""
100
101         """example of types of angles:
102             1 / 2
103             1 / 3
104             8 / 7
105             -----/-----
106             5 / 6
107             1 / 4
108             1 / 5
109             4 / 3
110             -----/-----
111             1 / 2
112             1 / 3
113             1 / 4
114             5 = 3 (alternate interior angles)
115             1 = 5 (Corresponding angles)
116             3+6 = 180° (consecutive interior angles)
117 """
118
119     h = Helper()
120     h.ps_from_file(r"ggb_files\th4.ggb")
121     h.s("AEB", "CFD", "HEFG")
122     h.paras("AB", "CD")
123     # ax3 - set \GEA = \HFD (alternate interior angles, AB || CD, HEFG traversal)
124     h.equala("GEA", "HFD")
125     h.calc(print_proof, use_theorems=[1, 2, 3])
126     # Corresponding angles example (GEA & GFC)
127     corresponding = h.get_a("GEA") == h.get_a("GFC")
128     # Consecutive interior angles example (BEG & HFD)
129     consecutive = h.get_a("BEG") + h.get_a("HFD") == 180
130     if print_proof:

```

```

131     ..... print(f"\{h.a('GEA')\} = {h.get_a('GEA')} \{h.a('GFC')\} = {h.get_a('GFC')} )")
132     ..... print("□")
133     ..... print(f"\{h.a('GEA')\} = {h.a('GFC')} (corresponding)")
134     ..... print()
135     ..... print(
136     ..... f"\{h.a('BEG')\} + {h.a('HFD')} = ({h.get_a('BEG')} ) + ({h.get_a('HFD')} ) = "
137     ..... f"\n{sum(h.get_a('BEG', 'HFD')) }"
138     ..... )
139     ..... print("□")
140     ..... print(f"\{h.a('BEG')\} + {h.a('HFD')} = 180 (consecutive)")
141
142     ..... return (corresponding, consecutive), (True, True)
143
144     ..... @staticmethod
145     ..... def th5(print_proof=False):
146     ..... """_ax3 + _th4 -> _th5 (proof by contradiction)
147     ..... 3 part Proof:
148     ..... 1. Converse alternate interior angles (proof by contradiction)
149     ..... 2. Converse corresponding angles (proof based on 1)
150     ..... 3. Converse consecutive interior angles (proof based on 1)
151     ..... """
152
153     ..... ans = [[], []]
154
155     ..... """ 1. Prove Converse Alternate Interior angles """
156     ..... # We Need to prove that AB || CD, based on equal alternate interior angles
157     ..... h = Helper()
158     ..... h.ps_from_file(r"ggb_files\th5_1.ggb")
159     ..... h.s("AEB", "CFD", "HEFG")
160     ..... # Assume CD is not parallel to AB
161     ..... # Therefore, there must be a different parallel to CD that goes through E
162     ..... # That parallel is IJ
163     ..... h.s("IEJ")
164     ..... h.paras("IJ", "CD")
165     ..... # It is given that BEG == CFH
166     ..... h.equala("BEG", "CFH")
167     ..... # Let's calc
168     ..... h.calc(print_proof, use_theorems=[1, 2, 3, 4])
169     ..... # We get that BEJ is zero
170     ..... ans[0].append(h.get_a("BEJ") == 0)
171     ..... ans[1].append(True)
172     ..... # Therefore IJ is the same as AB
173     ..... # And therefore our assumption that AB is not parallel to CD was wrong
174     ..... # Thus, AB || CD
175     ..... if print_proof:
176     .....     print(f"\{h.a('BEJ')\} = {h.get_a('BEJ')} )")
177     .....     print(f"□")
178     .....     print(f"(IJ is the same as CD) -> AB || CD")
179
180
181     ..... """ 2. Prove Converse equal corresponding angles """
182     ..... # We need to prove that AB || CD, based on equal corresponding angles
183     ..... h = Helper()
184     ..... h.ps_from_file(r"ggb_files\th5_2.ggb")
185     ..... h.s("AEB", "CFD", "HEFG")
186     ..... # it is given that HEB == HFD
187     ..... h.equala("HEB", "HFD")
188     ..... # lets see if GEA == HFD
189     ..... h.calc(print_proof, use_theorems=[1, 2, 3, 4])
190     ..... ans[0].append(h.get_a("GEA") == h.get_a("HFD"))
191     ..... ans[1].append(True)
192     ..... # because they are a pair of equal alternate interior angles, AB || CD (as was
193     ..... # proven in 1)
194     ..... if print_proof:
195     .....     print(f"\{h.a('GEA')\} = {h.get_a('GEA')} = {h.get_a('HFD')} = {h.a('HFD')} )")
196     .....     print(f"□")

```

```

196     ..... print(f"AB || CD (Converse alternate interior angles - proven before)")  

197  

198     ..... print("\n")  

199  

200     """ 3. Prove converse consecutive interior angles """
201     # We need to prove that AB || CD, based on consecutive interior angles  

202     h = Helper()  

203     h.ps_from_file(r"ggb_files\th5_3.ggb")  

204     h.s("AEB", "CFD", "HEFG")  

205     # it is given that BEG + HFD == 180  

206     x = h.given(Degree, "x")  

207     h.seta("BEG", x)  

208     h.seta("HFD", 180 - x)  

209     # lets see if GEA == HFD  

210     h.calc(print_proof, use_theorems=[1, 2, 3, 4])  

211     ans[0].append(h.get("GEA") == h.get("HFD"))  

212     ans[1].append(True)  

213     # because they are a pair of equal alternate interior angles, AB || CD (as was  

214     # proven in 1)  

215     if print_proof:  

216         print(f"\{h.a('GEA')\} = \{h.get('GEA')\} = \{h.get('HFD')\} = \{h.a('HFD')\}")  

217         print(f"\u2225")  

218         print(f"AB || CD (Converse alternate interior angles - proven before)")  

219  

220     return tuple(ans[0]), tuple(ans[1])  

221  

222     @staticmethod  

223     def th6(print_proof=False):  

224         """ _ax3 + _th1 -> _th6 """  

225  

226         h = Helper()  

227         h.ps_from_file(r"ggb_files\th6.ggb")  

228         h.tri("ABC")  

229         h.s("DCE")  

230         h.paras("AB", "DE")  

231         h.calc(print_proof, use_theorems=[1, 2, 3, 4, 5])  

232         if print_proof:  

233             print(  

234                 f"\{h.a('CAB')\} + \{h.a('ABC')\} + \{h.a('BCA')\} = (\{h.get('CAB')\}) + (\{h.  

235                 get('ABC')\}) + (\{h.get('BCA')\})",  

236                 f"\u2212 \{sum(h.get('CAB'), 'ABC', 'BCA')\}\)",  

237             )  

238             print(f"\u2225")  

239             print(f"sum of angles in \u0394ABC is 180\u00b0")  

240  

241     return (sum(h.get("CAB", "ABC", "BCA")),), (180,)  

242  

243     @staticmethod  

244     def th7(print_proof=False):  

245         """ _ax2 + _th6 -> _th7 """  

246         h = Helper()  

247         h.ps_from_file(r"ggb_files\th7.ggb")  

248         h.poly("ABCD")  

249         h.s("AC")  

250         h.calc(print_proof, use_theorems=[1, 2, 3, 4, 5, 6])  

251         if print_proof:  

252             print(  

253                 "\u2212 ".join(map(str, h.a("BAD", "ADC", "DCB", "CBA"))),  

254                 "=",  

255                 f"\u2212 \{(''.join(map(str, h.get("BAD", "ADC", "DCB", "CBA"))))\}\)",  

256                 "=",  

257                 sum(h.get("BAD", "ADC", "DCB", "CBA")),  

258             )  

259             print(f"\u2225")  

260             print(f"sum of angles in \u2212 ABCD is 360\u00b0")  

261  

262     return (sum(h.get("BAD", "ADC", "DCB", "CBA")),), (360,)
```

```
261
262     ....@staticmethod
263     ....def th8(print_proof=False):
264         ...."\"_th1 + _th6 -> _th8\""
265         ....h = Helper()
266         ....h.ps_from_file(r"ggb_files\th8.ggb")
267         ....h.tri("ACB")
268         ....h.conts("AC", "ACD")
269         ....h.calc(print_proof, use_theorems=range(1, 8))
270         ....if print_proof:
271             ....print(
272                 ....f"{h.a('BCD')} == {h.get_a('BCD')}", "
273                 ....f"{h.a('BAC')} == {h.a('CBA')} = ({h.get_a('BAC')} ) + ({h.get_a('CBA')} )",
274                 ....f"= {sum(h.get_a('BAC'), 'CBA')} )",
275             )
276             ....print(f" $\square$ ")
277             ....print(f"{h.a('BCD')} == {h.a('BAC')} + {h.a('CBA')}")
278
279     ....return (h.get_a("BCD") == sum(h.get_a("BAC", "CBA")), (True,)
280
```

```

1  """yoel_geva.py"""
2
3  from geo.problemcollection import ProblemCollection
4  from geo.helper import Helper
5  from geo.real.expression import Degree
6  from geo.filehandler import print_points_from_file as _pggb
7
8
9  class YoelGevaProblems(ProblemCollection):
10     """Problems from Yoel Geva, Part A, 10th grade class"""
11
12     @classmethod
13     def all(cls):
14         ans = []
15         for p in dir(cls):
16             attr = getattr(cls, p)
17             if p.startswith("p") and callable(attr):
18                 ans.append(attr)
19         return ans
20
21     @staticmethod
22     def p349_e1(print_proof=True):
23         h = Helper()
24         x = h.given(Degree, "x")
25         h.p("A", -4, 0)
26         h.p("B", 6, 0)
27         h.p("C", 4, 4)
28         h.p("D", -4, -2)
29         h.p("E", -1.33, 0)
30         h.s("AEB", "CED")
31         h.seta("AEC", 2 * x + 40)
32         h.seta("BED", x + 90)
33         h.calc(print_proof)
34         return (x,), (50,)
35
36     @staticmethod
37     def p349_e2(print_proof=True):
38         h = Helper()
39         h.ps("ABCD", [-4, 4, 0, 3], [0, 0, 0, 3])
40         h.s("ACB", "CD")
41         x = h.given(Degree, "x")
42         h.seta("ACD", 3 * x)
43         h.seta("DCB", x)
44         h.calc(print_proof)
45         return (x, h.geteta("ACD")), (45, 135)
46
47     @staticmethod
48     def p349_e3(print_proof=True):
49         h = Helper()
50         h.ps("ABCD", [-4, 4, 0, 3], [0, 0, 0, 3])
51         h.s("ACB", "CD")
52         x = h.given(Degree, "x")
53         h.seta("ACD", x + 48)
54         h.seta("DCB", x)
55         h.calc(print_proof)
56         return (h.geteta("DCB"), h.geteta("ACD")), (66, 114)
57
58     @staticmethod
59     def p349_e4(print_proof=True):
60         h = Helper()
61         h.ps("ABCDE", [-4, 0, 4, 3, -3], [0, 0, 0, 3, -3])
62         h.s("ABC", "DBE")
63         h.equala("EBA", "DBC")
64         h.calc(print_proof)
65         return (h.geteta("DBE"),), (180,)
66
67     @staticmethod

```

```

68     ... def p349_e5(print_proof=True):
69         ...     h = Helper()
70         ...     h.ps("ABCDEFG", [-4, 4, 3, -3, 0, -2, 2], [0, 0, 3, -3, 0, 4, -4])
71         ...     h.s("AEB", "CED", "FEG")
72         ...     h.equala("AEF", "FEC")
73         ...     h.calc(print_proof)
74         ...     return (h.get("BEG") == h.get("GED"),), (True,)
75
76     ... @staticmethod
77     ... def p349_e6(print_proof=True):
78         ...     h = Helper()
79         ...     h.ps("ABC", [4, 0, 8], [6, 0, 0])
80         ...     h.tri("ABC")
81         ...     x = h.given(Degree, "x")
82         ...     h.seta("CAB", 7 * x)
83         ...     h.seta("ABC", 5 * x)
84         ...     h.seta("BCA", 3 * x)
85         ...     h.calc(print_proof)
86         ...     return (h.get("CAB"), h.get("ABC"), h.get("BCA")), (84, 60, 36)
87
88     ... @staticmethod
89     ... def p349_e7(print_proof=True):
90         ...     h = Helper()
91         ...     h.ps("ABC", [0, 0, 8], [6, 0, 0])
92         ...     h.tri("ABC")
93         ...     h.seta("ABC", 90)
94         ...     x = h.given(Degree, "x")
95         ...     h.seta("CAB", x)
96         ...     h.seta("BCA", x + 40)
97         ...     h.calc(print_proof)
98         ...     return (h.get("CAB"), h.get("BCA")), (25, 65)
99
100    ... @staticmethod
101    ... def p350_e11(print_proof=True):
102        ...     h = Helper()
103        ...     h.ps("ABCDEF", [1, 0, 7, 3, 4, 2.4], [4, 0, 0, 0, 2, 1.2])
104        ...     h.tri("ABC")
105        ...     h.conts("BC", "BDC")
106        ...     h.tri_med("ABC", "AFD")
107        ...     h.conts("AC", "AEC")
108        ...     h.tri_angbi("ABC", "BFE")
109        ...     h.sets("BC", 12)
110        ...     h.seta("ABE", Degree(False, {0: 36}))
111        ...     h.calc(print_proof)
112        ...     return (h.get("DC"), h.get("ABC")), (6, 72)
113
114    ... @staticmethod
115    ... def p350_e12(print_proof=True):
116        ...     h = Helper()
117        ...     h.ps("ABCD", [0, 2, 6, 4], [0, 4, 0, 2])
118        ...     h.tri("ABC")
119        ...     h.conts("BC", "BDC")
120        ...     h.tri_angbi("ABC", "AD")
121        ...     h.seta("ACB", 44)
122        ...     h.seta("CBA", 80)
123        ...     h.calc(print_proof)
124        ...     return (h.get("ADB"), h.get("CDA")), (72, 108)
125
126    ... @staticmethod
127    ... def p351_e15(print_proof=True):
128        ...     h = Helper()
129        ...     h.ps("ABCDE", [0, 8, 0, 2, 4], [4, 0, 0, 3, 2])
130        ...     h.tri("ABC")
131        ...     h.conts("AB", "ADEB")
132        ...     h.tri_alt("ABC", "CD")
133        ...     h.tri_angbi("ABC", "CE")
134        ...     h.seta("ACB", 90)

```

```

135     ..... h.seta("CBA", 26)
136     ..... h.calc(print_proof)
137     ..... return (h.get("DCE"),), (19,)
138
139     ..... @staticmethod
140     ..... def p351_e17(print_proof=True):
141     .....     h = Helper()
142     .....     h.ps("ABCDE", [0.0, 6.0, 0.0, 3.0, 3.0, 2.1], [4.0, 0.0, 0.0, 0.0, 2.0, 1.4])
143     .....     h.tri("ABC")
144     .....     h.conts("BC", "BDC")
145     .....     h.conts("AB", "AEB")
146     .....     h.tri_alt("ABC", "CNE")
147     .....     h.tri_angbi("ABC", "AND")
148     .....     h.seta("ACB", 90)
149     .....     h.seta("BAC", 34)
150     .....     h.calc(print_proof)
151     .....     return (h.get("DNC"),), (73,)
152
153     ..... @staticmethod
154     ..... def p352_e19(print_proof=True):
155     .....     h = Helper()
156     .....     h.ps(
157     .....         "ABCDEO", [3.0, 0.0, 6.0, 3.0, 4.488, 3.0], [4.0, 0.0, 0.0, 0.0, 2.016, 2.0]
158     .....     )
159     .....     h.tri("ABC")
160     .....     h.conts("BC", "BDC")
161     .....     h.conts("AC", "AEC")
162     .....     h.tri_segb("ABC", "DO", "BC")
163     .....     h.tri_segb("ABC", "EO", "AC")
164     .....     h.sets("BD", 6)
165     .....     h.sets("AC", 8)
166     .....     h.calc(print_proof)
167     .....     return (h.get("CE"), h.gets("BC"), h.get("BCA") + h.get("EOD")), (4, 12, 180)
168
169     ..... @staticmethod
170     ..... def p352_e23(print_proof=True):
171     .....     h = Helper()
172     .....     h.ps("ABCDEFGH", [0, 9, 0, 9, 6, 0, 4, 2], [4, 4, 2, 2, 6, 0, 4, 2])
173     .....     h.paras("AGB", "CHD")
174     .....     h.s("EGHF")
175     .....     x, y = h.given(Degree, "x"), h.given(Degree, "y")
176     .....     h.seta("FGA", x + y)
177     .....     h.seta("BGF", 4 * x - 2 * y)
178     .....     h.seta("EHD", 60)
179     .....     h.calc(print_proof)
180     .....     return (x, y), (40, 20)
181
182     ..... @staticmethod
183     ..... def p352_e24(print_proof=True):
184     .....     h = Helper()
185     .....     h.ps("ABCDE", [4, 0, 8, 2, 6], [6, 0, 0, 3, 3])
186     .....     h.tri("ABC")
187     .....     h.conts("AB", "ADB")
188     .....     h.conts("AC", "AEC")
189     .....     h.s("DE")
190     .....     h.paras("DE", "BC")
191     .....     h.seta("ABC", 50)
192     .....     h.seta("BCA", 60)
193     .....     h.calc(print_proof)
194     .....     return (h.get("ADE"), h.get("CED")), (50, 120)
195
196     ..... @staticmethod
197     ..... def p353_e25(print_proof=True):
198     .....     h = Helper()
199     .....     h.ps("ABCDE", [4, 0, 8, 2, 6], [6, 0, 0, 3, 3])
200     .....     h.tri("ABC")
201     .....     h.conts("AB", "ADB")

```

```

202     ..... h.conts("AC", "AEC")
203     ..... h.s("DE")
204     ..... h.seta("CAB", 72)
205     ..... h.seta("ABC", 48)
206     ..... h.seta("CED", 120)
207     ..... h.calc(print_proof)
208     ..... return (h.isparas("DE", "BC"),), (True,)
209
210     ..... @staticmethod
211     ..... def p353_e26(print_proof=True):
212         ..... # pggb()
213         ..... h = Helper()
214         ..... h.ps("ABCDE", [7, 6, 0, 2, 4], [0, -2, 0, 2, 0])
215         ..... h.s("AEC")
216         ..... h.s("BED")
217         ..... h.s("AB")
218         ..... h.s("CD")
219         ..... h.seta("BAC", 39)
220         ..... h.seta("BDC", 98)
221         ..... h.seta("AEB", 43)
222         ..... h.calc(print_proof)
223         ..... return (h.isparas("AB", "DC"),), (True,)
224
225     ..... @staticmethod
226     ..... def p353_e27(print_proof=True):
227         ..... h = Helper()
228         ..... h.ps("ABCDE", [0, 0, 6, 4, 2.4], [4, 0, 0, 4, 2.4])
229         ..... h.s("AB", "BC", "CEA", "AD", "DEB")
230         ..... h.paras("AD", "BC")
231         ..... h.seta("DAC", 42)
232         ..... h.seta("CEB", 94)
233         ..... h.calc(print_proof)
234         ..... return (h.get("EBC"),), (44,)
235
236     ..... @staticmethod
237     ..... def p353_e28(print_proof=True):
238         ..... h = Helper()
239         ..... h.ps("ABCD", [2, 6, 8, 0], [4, 4, 0, 0])
240         ..... h.poly("ABCD")
241         ..... h.s("AC")
242         ..... h.paras("AB", "DC")
243         ..... h.equala("DCA", "ACB")
244         ..... h.calc(print_proof)
245         ..... A = h.get("ACB") == h.get("BAC")
246         ..... if print_proof:
247             ..... print("\n" * 10)
248             ..... h.seta("DCB", 80)
249             ..... h.calc(print_proof)
250             ..... B = h.get("BAC")
251             ..... return (A, B), (True, 40)
252
253     ..... @staticmethod
254     ..... def p353_e29(print_proof=True):
255         ..... h = Helper()
256         ..... h.ps("ABCDE", [2, 8, 10, 0, 6], [4, 4, 0, 0, 2])
257         ..... h.poly("ABCD")
258         ..... h.s("BE", "EC")
259         ..... h.paras("AB", "DC")
260         ..... h.equala("CBE", "EBA")
261         ..... h.equala("DCE", "ECB")
262         ..... h.calc(print_proof)
263         ..... return (h.get("BEC"),), (90,)
264

```

```
1 """main.py"""
2
3 from yoel_geva import YoelGevaProblems
4 from proofs import ProofCollection
5
6 YoelGevaProblems.check_all()
7 ProofCollection.check_all()
```