Question 1.1: Explain the key points in your implementation (4pt).

יצרתי מערך של N נקודות X בהן נדגום את הפומקציה ומערך Y בגודל זהה אשר ישמור את ערכי הY המצאימים לכל X.

יצרתי פונקציה שלוקחת ערך ומחזירה את ערך הY שלו בעזרת שיטת לגראנז'

סיבוכיות זמן הריצה של שיטת לגרנז' היא O(N^2)

דגמתי N נקודות על מנת למקסם את הדיוק של האינטרפולצה

האלגוריתם לא רץ באO(N) אלא בO(N^2) אך מביא תוצאות ברמת דיוק גבוהה.

הקוד משתמש בN נקודות על מנת ליצור אינטרפולציה ומוצא פולינום מדרגה N-1

Question 2.1: Explain the key points in your implementation (4pt).

יצרתי פונקציה חדשה שהיא ההפרש של 2 הפונקציות שקיבלתי

השתמשתי בשיטת בייסקשן על מנת למצוא את נקודות החיתוך של הפונקציה החדשה עם ציר X כלומר שורשי הפונקציה החדשה. עקרון השיטה מסתמך על חציית הקטע ל2 ומציאת השורשים בו ע"י החלפת בימן של הפונקציה. כל פעם הבאתי לשיטה חלק של הפונקציה בעזרת לולאת FOR

היתרון בשיטה זו היא שניתן תמיד להגיע לשורשים של הפונקציה אך לא תמיד זמן הריצה יהיה אופטילי. לכן על מנת להגיע לדיוק מקסימלי העדפתי להשתמש בשיטה זו. בנוסף שיטה זו תמיד תמצא את השורשים בניגוד לשיטות אחרות.

Question 3.1: Explain the key points in your implementation of Assignment3.integrate(…). (4pt)

אם יש לנו רק נקודה אחת, נשתמש ב mid-point אחרת נשתמש ב Simpson's

בשביל להשתמש בסמפסון נצטרך שיהיה לנו מספר זוגי של קטעים(מספר אי זוגי של נקודות) לכן אם יש לנו מספר זוגי של נקודות, נוריד אחת כדי לשמור על דרישות התרגיל.

נחלק את הקטע לN-1 חלקים שווים(מספר זוגי של חלקים). ולאחר מכן מדגום נקודה מכל קטע ונמצא את ערך הY שלה. נחלק את הנקודות לנקודות זוגיות ואי זוגיות. ונכפיל ב4 ו2 בהתאם ולבסוף נסכום יחד עם נקודת ההתחלה והסיום.

כל המשתנים המספריים הם מסוג np.flaot32 כנדרש בתרגיל.

שיטת סימפסון נותנת לנו דיוק טוב יחסית וזמני ריצה טובים O(N) כתלות במספר הנקודות המותרות.

Question 3.2: Explain the key points in your implementation of Assignment3. areabetween (…) (4pt).

ניצור פונקציה חדשה להיות ההפרש של הפונקציות הנתונות(בערך מוחלפט)

נמצא את נקודות החיתוך של הפונקציות כדי לפצל את האינטגרציה לתחומים שונים. נשמור את כל הנקודות חיתוך במערך.

בעזרת הפונקציה החדשה שבינו, נחשב את האינטגרל שלה בין כל 2 נקודות חיתוך ונצבור לסכום.

עיקרון האלגוריתם מתבסס על הפרדה לקטעים שיוצרות נקודות החיתוך(יוצר לנו את גבולות האינטגרציה) ופתרון כל חלק בנפרד על סמך חישובים של פונקציות קודמות. סיבוכיות זמן הריצה היא תלויה בפונקציה שממישנו למציאת נקודות חיתוך O(N^2)

Question 3.3: Explain why is the function $2^{\frac{1}{x^2}} * \sin\left(\frac{1}{x}\right)$ is difficult for numeric integration with equally spaced points? (4pt)

הבעיה בפונקציה זו היא שכאאשר אנחנו מתקרבים ל0 הפונקציה משתנה בקצב גבוה מאוד. כלומר שינוי קטן בX גורם לשינויים גדולים בY. אם נבחר מרווח אחיד בין הנקודות, לא נוכל לתפוס את השינויים האלה, לכן כאשר נדגום את הפונקציה בתחום הזה יהיה לנו קשה להעריך מה ערך הפונקציה באיזור, מה שמשפיע על שיטות לחישוב של אינטגרל, אשר מסתמכות על ערכי הפונקציה בנקודות כלשהן.

Question 3.4: What is the maximal integration error of the $2^{\frac{1}{x^2}} * \sin\left(\frac{1}{x}\right)$ in the range [0.1, 10]? Explain. (4pt)

לפי שיטת סימפסון השגיאה היא ביחס לנגזרת הרביעית של הפונקציה

$$error \leq \frac{(b-a)^5}{180 * n^4} * \max\left(f^4(x)\right)$$

כדי למקסם את השגיאה נרצה *n=1*

$$f^4(x) = 4.269 * 10^{38}, a = 0.1, b = 10$$

לכן:

$$error \leq \frac{(10-0.1)^5}{180} * 4.269 * 10^{38} = 2.2556 * 10^{38}$$

Question 4.1: Explain the key points in your implementation. (4pt)

נדגום שליש מהזמן הנתון את הפונקציה בשביל להשאיר זמן לחישובים.

נחלק את הטווח ל150 מקטעים שווים ונבצע דגימות בכל קטע

בכל קטע נבצע ממוצע של ערך הפונקציה על מנט להקטין את הרעש ונשמור את שיעורי הנקודות המייצגות כל קטע. הדגימות מתבצעות באופן רנדומלי כדי לקבל תמונה אמיתית יותר של הדאטה לעטמת דגימות בקטע קבוע.

ניצור משוואת ישר בין כל 2 נקודות ונחבר אותם לפונקציה אחת שנוסיף לרשימה של פונקציות.

ניצור פונקציה g שמחזירה את הערך המקורב של המספר ע"י מציאת הפונקציה שלו מציאת הפונקציה המתאימה. נחזיר את ערך הפונקציה המקורבת במקום שהתבקשנו.

למעשה יצרנו רשימה של פונקציות ממעלה ראשונה ובעזרתן קירבנו את ערך הפונקציה. ככל שנגדיל את מספר החלקים שחילקנו את הקטע, נוכל להגדיל את הדיוק שלנו אך הדבר יפגע בזמן הריצה. מספר החלוקות וזמן הדגימה נבחר באופן שרירותי לאחר ניסוי ותהייה. יתרון האלגוריתם הוא שהוא יחסית פשוט וזמן הריצה שלו תלוי במספר החלוקות שנבצע

Question 4B.1: Explain the key points in your implementation. (4pt)

נבצע 500 דגימות מהפונקציה ונשמור אותם במערך של נקודות (מספר הנקודות נבחר באופן שרירותי)

נפעיל אלגוריתם shoelace כדי למצוא את שטח הצורה על סמך הנקודות.

האלגוריתם מחשב שטח של צורה כלשהי ע"י מכפלת הX של כל נקודה בY של הנקודה הבאה פחות מכפלת הY של כל נקודה בX של הנקודה הבאה ולבסוף מחלק ב2.

יתרון בשיטה זו שהיא פשוטה למימוש וזמן הריצה שלה הוא נמוך(O(N כתלות במספר הדגימות

Question 4B.2: Explain the key points in your implementation. (4pt)

נדגום נקודות מהצורה ונשמור אותם במערך שביעית מהזמן הנתון כדי לעמוד בדרישות הזמנים ולהשאיר מקום לחישובים.

בעזרת אלגוריתם kmeans נסווג את הנקודות לקבוצות ולאחר מכן נמיין אותם לפי סטיית הזיוות של כל מרכז של קלאסטר.

נחשב את המחרחקים בין כל מרכז ונמצא את הסידור האופטימאלי שלהם כדי לסגור את הצורה.

בעזרת הסידור שיצרנו ניצור פוליגון שיוגדר על הסידור ונגדיר את שטח הפוליגון להיות שטח הצורה שלנו.

הרעיון הכללי של האלגוריתם מתבסס על מיון הנקודות לקלאסטרים ויצירת פוליגון ע"י סידור מרכזי הקלאסטרים. היתרון בשיטה זו היא שבעזרת חבילת פוליגון קל לחשב את השטח של הצורה לאחר שהגדרנו אותה. האלגוריתם כולל מספר יחסית גדול של פעולות ומיונים אך מביא לתוצאות מדוייקות בזכות השימוש בפוליגון. בחירת K=12 נעשתה באופן שרירותי לאחר ניסוי ותהיה, בחירה אחרת של K הייתה עשויה להביא לתוצאות יותר טובות עבור חלק מהמקרים. כמו כן בחירת זמן הדגימה להיות שביעית מהזמן נעשה באותו עקרון. בחירת זמן גדול יותר הייתה יכולה לעלות את רמת הדיוק אך לא לעמוד בדרישות זמן הריצה.

MyShape:

בנאי: הבנאי מקבל שטח ומערך ממויין של נקודות

```
def area(self):
```

מחזריר את השטח של הצורה שהוגדר בבנאי

```
def contour(self, n):
```

מחזיר נקודות ממוינות עד הנקודה הN לפי הנקודות שהבאנו ביצירה של הבנאי