תרגיל 7:
High Order Functions

תאריך פרסום: 18/12/2022

תאריך הגשה: 28/12/2022 בשעה 23:59

מתרגל אחראי: תום משיח

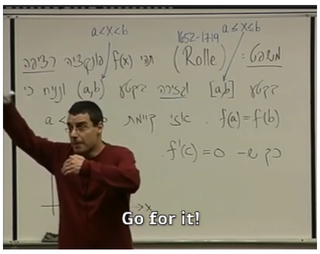
משקל תרגיל: 2 נקודות

מטרות העבודה: שימוש ומימוש פונקציות מסדר גבוה ושימוש בספריות חיצוניות.

<u>דגשים לעבודה 7:</u>

1. עבודה זו מורכבת משאלה אחת מרובת סעיפים.
2. בתרגיל זה יש להשתמש בספריות החיצוניות matplotlib ו-sympy. מומלץ לעיין בדוקומנטציה של כל ספריה בלחיצה על שמה. ניתן להעזר בכל אתר שתרצו על מנת להבין כיצד להשתמש בספריות הנ"ל.
3. בתרגיל נשתמש בפונקציות מתמטיות. נניח כי כל הפונקציות בהן נשתמש הינן פונקציות שתחום ההצבה שלהן הינו כל x.
4. ניתן להניח בכל סעיפי השאלה שהקלט תקין.

שאלה 1:



אתם כל כך נהנים בקורס חדו"א 1 שהחלטתם להגדיל ראש ולממש חלק מחומר הקורס בפייתון!

סעיף א':

אתם מעוניינים לממש פונקציה המחזירה טווח של מספרים, בדומה ל-range, אך ללא האילוץ של מספרים שלמים. לדוגמא:

[0, 0.2, 0.4, 0.6, 0.8, 1, 1.2]

בסעיף זה יש לממש את הפונקציה הבאה **<u>בשורת קוד אחת</u>** (חתימת הפונקציה אינה נחשבת לשורה):

def float_range(begin=0, end=10, step=0.1):

הפונקציה מקבלת שלושה מספרים (לא בהכרח שלמים) ומחזירה רשימה המתחילה במספר begin, ומתקדמת בקפיצות של step עד ל-end (בדומה ל-range). בנוסף יש לעגל כל איבר ברשימה ל-2 ספרות אחרי הנקודה.

<u>דוגמאות:</u>

.1

```
print(float_range(0, 10, 0.5))

[0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0,
6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0]
```

.2

```
print(float_range(-1, 1, 0.11))

[-1.0, -0.89, -0.78, -0.67, -0.56, -0.45, -0.34, -0.23, -0.12,
-0.01, 0.1, 0.21, 0.32, 0.43, 0.54, 0.65, 0.76, 0.87, 0.98]
```

.3

```
print(float_range(-2, -1.7, 0.03))

[-2.0, -1.97, -1.94, -1.91, -1.88, -1.85, -1.82, -1.79, -1.76,
-1.73, -1.7]
```

סעיף ב':

בסעיף זה תממשו 4 פונקציות המשתמשות בספרייה sympy, כאשר כל אחת תמומש **<u>בשורת קוד אחת</u>** (חתימת הפונקציה אינה נחשבת לשורה) המשתמשות בספרייה sympy.

ב' 1:

ממשו את הפונקציה:

def create_quadratic_equation(symbol, a,b,c):

אשר מקבלת משתנה סימבולי (symbol) ושלושה מספרים (a, b, c) ומחזירה ביטוי סימבולי שהוא משוואה ריבועית כאשר המשתנה הינו symbol והמקדמים הינם שלושת המספרים בהתאם (a- המקדם של x^2, b- המקדם של x , ו-c האיבר החופשי.

<u>דוגמאות:</u>

```
>>> x = symbols('x')
>>> create_quadratic_equation(x, 1,2,3)
x**2 + 2*x + 3
>>> create_quadratic_equation(x , 0 ,1,1)
x + 1
```

ב' 2:

בסעיף זה תממשו את הפונקציה:

def concatenating_expressions (exprs):

אשר מקבלת רשימה של ביטויים סימבולים (רשימה לא ריקה) ומחזירה את סכומם כביטוי סימבולי.

דוגמאות:

```
>>> x = symbols('x')
>>> concatenating_expressions([x+2, x**2, 6*x, 1/x])
x**2 + 7*x + 2 + 1/x
>>> concatenating_expressions([x, x, x, x, x])
5*x
```

ב' 3:

בסעיף זה תממשו את הפונקציה:

def string_to_expressions(str_exprs):

אשר מקבלת מחרוזת (str_exprs) המכילה ביטויים סימבוליים המופרדים אחד מהשני בתו פסיק (',') ומחזירה את הביטויים כרשימה של מופעי sympy.

דוגמאות:

```
>>> string_to_expressions('x + 3, cos(x), x**10, x*x')
[x + 3, cos(x), x**10, x**2]
```

ב' 4:

בסעיף זה תממשו את הפונקציה:

def sub_in_expr(expr, symbol):

אשר מקבלת ביטוי סימבולי ואת המשתנה הסימבולי (symbol) ומחזירה **פונקציה** אשר מקבלת כקלט ארגומנט אחד ומחזירה את תוצאת החישוב של הביטוי הסימבולי עם הקלט.

דוגמאות:

```
>>> x, y = symbols('x y')
>>> sub_x_squared = sub_in_expr(x**2,x)
>>> sub_x_squared(2)
4
>>> sub_x_squared(y)
y**2
>>> sub_x_squared(x+2)
(x + 2)**2
>>> sub_x_squared(y**2)
y**4
```

```
>>> sub_cox_x = sub_in_expr(cos(x),x)
>>> sub_cox_x(0)
1
>>> sub_cox_x(sin(x))
cos(sin(x))
```

סעיף ג':

בסעיף זה נשתמש בספריה matplotlib ובאופן פרטני ב- matplotlib.pyplot.

ממשו את הפונקציה:

def plot_expr(expr, symbol, start, end, step = 1):

הפונקציה מקבלת ביטוי סימבולי, משתנה סימבולי וטווח של מספרים (התחלה, סוף וגודל צעד) ומשרטטת את הגרף של הביטוי בטווח המספרים.

* רמז: זכרו את ששכפול קוד אינו דבר נכון תכנותית.

<u>דגשים לגרף:</u>

- לגרף תהיה כותרת באופן הבא:

A GRAPH OF THE FUNCTION: Y = <expr>

- כאשר החלק האדום הוא קבוע ובחלק השחור יהיה הביטוי עבורו צויר הגרף.
- ציר ה-x יקרא 'x', ציר ה-y יקרא 'y'
- יופיע גריד (grid)
- על הגרף להיות קווי (צבע הגרף נתון להחלטתכם)

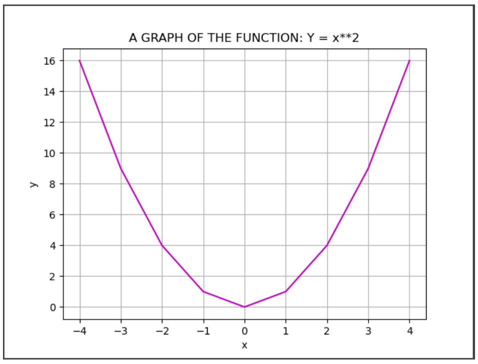
<u>דוגמאות:</u>

1.

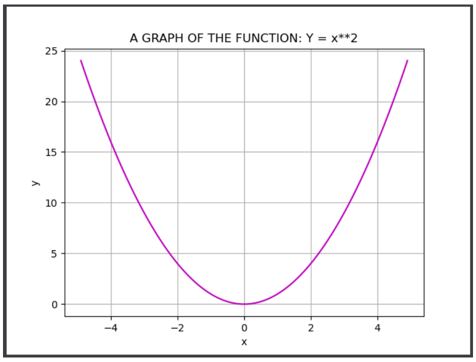
```
x = symbols('x')
plot_expr(x ** 2, x, -5, 5, 1)
plot_expr(x ** 2, x, -5, 5, 0.1)
```

A
B

A



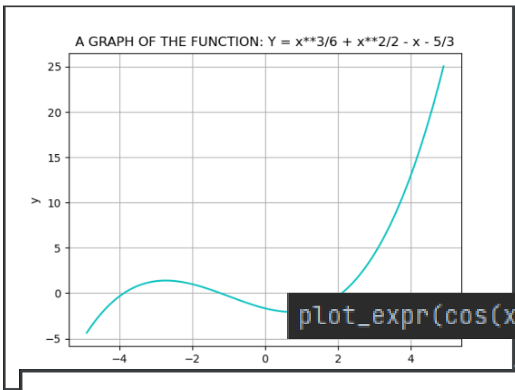
B



** שימו לב מה משתנה בגרף כאשר מקטינים/ מגדילים את step.

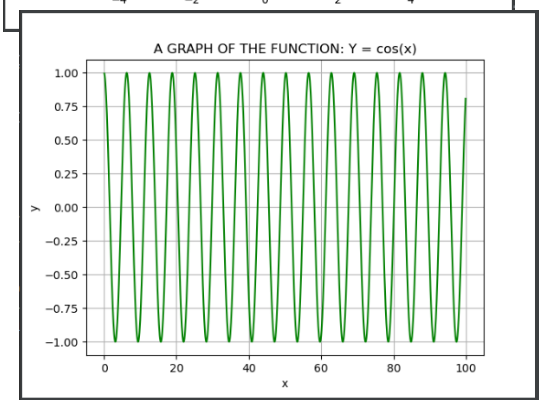
2.

```
plot_expr((x**3 + 3*x**2 - 6 * x - 10) / 6, x, -5, 5, 0.1)
```



3.

```
plot_expr(cos(x), x, 0, 100, 0.1)
```



סעיף ד':

בסעיף זה תממשו פונקציה אשר מקבלת ביטוי סימבולי ומשרטטת את הגרף המתאים ואת 2 הנגזרות הראשונות שלו.

ד' 1:

על מנת לממש פונקציה זו ראשית עליכם לממש את פונקציית העזר:

def derivative_func_recursive(expr, symbol, num):

פונקציה זו הינה פונקציה רקורסיבית (תודו שהתגעגעתם) אשר מקבלת ביטוי סימבולי, את המשתנה הסימבולי לפיו נרצה לגזור ומספר המייצג את מעלת הנגזרת שנרצה (1 - נגזרת ראשונה, 2 - נגזרת שנייה וכן הלאה) ומחזירה ביטוי סימבולי שהינו הנגזרת ה-num של הביטוי המקורי.

שימו לב, השתמשו בהגדרת הנגזרת שראיתם בהרצאה:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

<u>דגשים:</u>

- **המימוש רקורסיבי. אין** להשתמש בלולאה.
- **אין** להשתמש בפונקציית הנגזרת של sympy.
- ניתן להשתמש בפונקציית חישוב הגבול של sympy.

<u>דוגמאות:</u>

```
>>> x, y = symbols('x, y')
>>> derivative_func_recursive(x**2 + 3*x + 2, x, 1)
2*x + 3
>>> derivative_func_recursive(x**2 + 3*x + 2, x, 2)
2
>>> derivative_func_recursive(x**2 + 3*x + 2, x, 3)
0
```

```
>>> derivative_func_recursive(cos(y) + y*sin(x), y, 1)
sin(x) - sin(y)
>>> derivative_func_recursive(cos(y) + y*sin(x), y, 2)
-cos(y)
>>> derivative_func_recursive(cos(y) + y*sin(x), y, 3)
sin(y)
>>> derivative_func_recursive(cos(y) + y*sin(x), y, 4)
cos(y)
```

ד' 2:

כעת, לאחר שמימשתם את פונקציית העזר עליכם לממש את הפונקציה המרכזית

def graph_of_2_derivatives(expr, symbol, start, end, step=1):

הפונקציה מקבלת ביטוי סימבולי, משתנה סימבולי וטווח של מספרים (התחלה, סוף וגודל צעד) ומשרטטת את הגרף של הביטוי, הנגזרת הראשונה והשנייה בטווח הנתון. ראו דוגמא.

<u>דגשים לגרף:</u>

- לגרף תהיה כותרת באופן הבא:

TWO DERIVATIVES OF THE FUNCTION: Y = <expr>

- כאשר החלק האדום הוא קבוע ובחלק השחור יהיה הביטוי עבורו צויר הגרף.
- ציר ה-x יקרא 'x', ציר ה-y יקרא 'y'
- יופיע גריד (grid)
- על כל אחד מהגרפים להיות גרף קווי ובצבע שונה אחד מהשני (לבחירתכם)

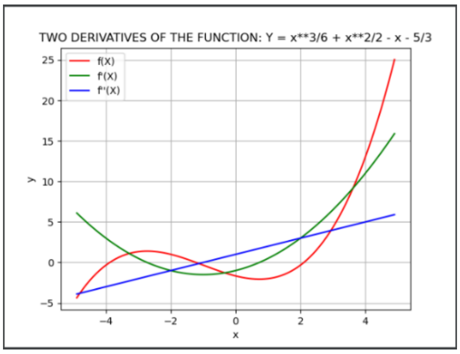
- יופיע מקרא כאשר הפונקציה המקורית תיקרא f(x), הנגזרת הראשונה תיקרא f'(x) והשלישית f''(x).

<u>דוגמאות:</u>

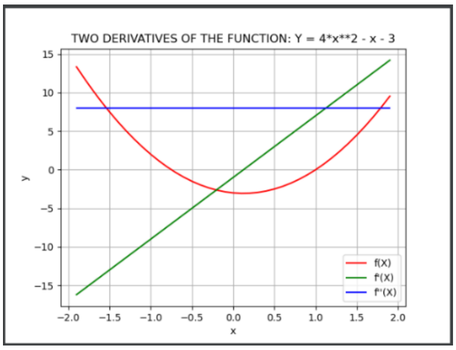
A
B
C

```
x = symbols('x')
graph_of_2_derivatives((x ** 3 + 3 * x ** 2 - 6 * x - 10) / 6, x, -5, 5, 0.1)
graph_of_2_derivatives(4 * x ** 2 - x - 3, x, -2, 2, 0.1)
graph_of_2_derivatives(cos(x), x, -10, 10, 0.1)
```

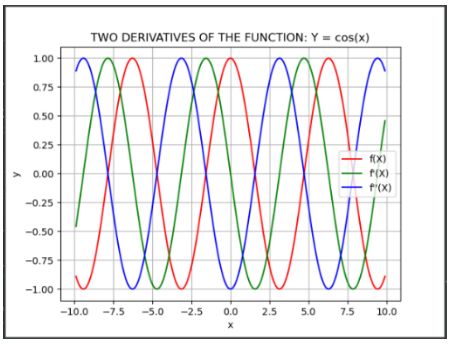
A



B



C



סעיף ה':

בהרצאה ראיתם כיצד ניתן לחשב קירוב לאינטגרל על ידי שימוש בנוסחא:

$$hf(a)+hf(a+h)+\ldots+hf(a+ih)+\ldots+hf\left(a+\left\lfloor\frac{b-a}{h}\right\rfloor h\right)$$
$$=h\sum_{i=0}^{\lfloor b-a/h\rfloor} f(a+ih)$$

כעת, עליכם לממש את הפונקציה:

def integral_func(expr, symbol, h=0.001):

אשר מקבלת ביטוי סימבולי, משתנה סימבולי וערך של h ומחזירה **פונקציה** המקבלת 2 נעלמים (a,b) ומחזירה את האינטגרל המסויים של הביטוי בטווח שבין a ל-b עם רוחב h.

<u>דגשים:</u>

- אין להשתמש בפונקציה המובנת של sympy המחשבת אינטגרל מסויים.
- על המימוש להיות לכל היותר בעל 2 שורות קוד (חתימת הפונקציה אינה נספרת כשורת קוד).
- אין להשתמש בפונקציה המובנת sum.

<u>דוגמאות:</u>

```
>>> integral_x_squared = integral_func(x**2, x)
>>> integral_x_squared(0,1)
0.332833500000000
>>> integral_x_squared(0,2)
2.66466700000000
>>> integral_polynomial = integral_func((x ** 3 + 3 * x ** 2 - 6 * x - 10) / 6, x)
>>> integral_polynomial(0,5)
26.0291681250000
>>> integral_polynomial(-5,5)
24.9891675000000
```

בהצלחה! ★