

## היכרות עם WinAPI

בשיעור למדנו שמערכת ההפעלה מספקת לנו ממשק (API) לביצוע פעולות שונות אותן היא מממשת. ממשק זה נותן למפתח התוכנה להשתמש בשירותי מערכת ההפעלה בתחומים שונים כגון ניהול תהליכים, גרפיקה, ניהול חלונות, אבטחה ועוד.

הממשק הינו למעשה שכבת אבסטרקציה מעל מערכת ההפעלה, כך שאיננו תלויים בצורת המימוש של הפעולות השונות וכן איננו תלויים בגרסת מערכת ההפעלה (שכן ממשק זה שומר על מבנה דומה ותאימות לאחור) בקישור הבא (MSDN): <https://msdn.microsoft.com/en-us/library/ff818516.aspx> תוכלו למצוא אינדקס של פקודות ה API השונות אותן מספקת Windows.

**הערה:** בתרגילים הבאים נשתמש בחבילות של פייתון העוטפות את פונקציות ה-API השונות. עבור חלק מהפונקציות – השימוש יהיה זהה לפונקציה שמוצאת על ידי מערכת ההפעלה, ובחלק אחר – נעשו עטיפות נוספות (של פייתון) המקלות את השימוש בפונקציות אלה.

בתרגיל זה נתנסה בפונקציה בסיסית ב API בשם `MessageBox`. אתם יכולים לנחש מה היא עושה? **תעדו כל פקודה שהרצתם**, והגישו בסוף את רשימת הפקודות העונות לסעיפים השונים. תחילה, חפשו את הפונקציה ב MSDN. שימו לב לאילו פרמטרים היא מקבלת ומה היא מחזירה:

C++

```
int WINAPI MessageBox(  
    _In_opt_ HWND    hWnd,  
    _In_opt_ LPCTSTR lpText,  
    _In_opt_ LPCTSTR lpCaption,  
    _In_     UINT     uType  
);
```

הפונקציה מקבלת למעשה 4 פרמטרים ומחזירה מספר. קראו קצת על כל אחד מהפרמטרים. כעת נכיר כמה חבילות שונות בפייתון העוטפות לנו את הפונקציה: `win32api`, `win32gui`, `win32ui` פעלו לפי ההוראות הבאות:

1. נסו להריץ את הפונקציה `MessageBox` מתוך החבילה `win32gui`. נסו להבין כמה פרמטרים נדרשים לפונקציה. הדרך הכי פשוטה, בתור התחלה (ובמידה שאנחנו יודע שהפונקציה לא יכולה לעשות שום דבר "רע" למחשב) היא פשוט להריץ את הפונקציה בלי פרמטרים, ולראות מה קורה.

```
>>> import win32gui  
>>> win32gui.MessageBox()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: MessageBox() takes exactly 4 arguments (0 given)
```

ניתן לראות שהפונקציה דורשת 4 פרמטרים (במקרה.. דומה למה שמופיע ב- MSDN ©).

2. הכניסו פרמטרים מתאימים לפי ה MSDN והריצו את הפונקציה. שימו לב שבפייתון – None הוא לצורך העניין המקביל של NULL. כעת הקפיצו חלון שבתוכו מופיע השם המלא שלכם, ויש בו כפתורים של Yes ו- No בלבד.

**הרחבה:** מתוך התייעוד ב MSDN ניתן לשים לב כי הפרמטר uType המסמל את סוג החלון מיוצג על ידי מספר. ליד כל מספר – מופיע קבוע המייצג את אותו מספר בצורה נוחה יותר (למשל MB\_OK הוא קבוע המייצג את הערך 0 המסמל חלון בו מופיע כפתור OK בלבד). כדי להגיע לקבועים האלה מפייתון – קיימת החבילה win32con ובה מוגדרים הקבועים השונים.

3. כעת הקפיצו שוב את אותו חלון עם שם הזמר האהוב עליכם, כשיש לו כפתורים של Yes, No ו- HELP. שימו לב שניתן לחבר יחד חלק מהפרמטרים של ה uType - הדבר נעשה באמצעות האופרטור | (bitwise or). קראו על האופרטור קצת והבינו איך זה בדיוק עובד. אילו מבין הערכים האפשריים ל uType ניתן לחבר יחד?

**הרחבה:** שימו לב כי כפתור ה Help מתנהג שונה מהכפתורים האחרים. ניתן לקרוא על כך עוד בתייעוד באינטרנט.

4. כעת, הוסיפו לפקודה האחרונה הודעה שתודפס למסך בהתאם לכפתור שנלחץ – במידה ונלחץ Yes יודפס למסך שם שיר של אותו זמר, במידה ונלחץ No הדפיסו טקסט לבחירתכם. (הכוונה – מה הפונקציה מחזירה? מה משמעות מספר זה?)

עד כאן ראינו שימוש בחבילה אחת (win32gui). כעת נתנסה בעוד 2 חבילות שלהן עטיפה ל MessageBox:

5. ייבאו את החבילה win32api ונסו להריץ מתוכה את MessageBox עם כמה פרמטרים שהיא דורשת.  
6. כעת, נסה להריץ את הפונקציה עם אותם פרמטרים שהרצת בסעיף 2. האם זה עבד? למה? מה ההבדל בין העטיפה שבחבילה זו (win32api) לזו שבקודמת (win32gui)?

7. כעת ננסה את החבילה win32ui – **אזהרה:** בסעיף זה אתם עלולים להקריס את תהליך הפייתון שרץ, במידה ותכניסו פרמטרים שגויים – נסו למצוא את הפרמטרים המתאימים עם מינימום הקרסות של פייתון. ☺

ראינו כי לפייתון נוצרו מספר עטיפות שונות לפונקציה המהוללת MessageBox, כאשר שיטת הקריאה לפונקציה שונה מאחת לשנייה. הדבר נוח ויפה כאשר נוצרה באמת עטיפה לפונקציה – השאלה מה עם פונקציות אשר קיימות ב WinAPI ולא בחבילות הנחמדות של פייתון?

כעת, נכיר חבילה חדשה בשם `ctypes` :

`ctypes` הינה חבילה שמאפשרת לנו להשתמש בטיפוסים ובפונקציות שהוגדרו במקור לשפת C. החבילה יכול לעטוף את האובייקטים כך שנוכל לקרוא להם פשוט מפייתון. למה זה מעניין אותנו? הפונקציות ומבני הנתונים של WinAPI הם בפורמט של C ולמעשה מיוצאים על ידי DLL-ים של מערכת ההפעלה. את ה DLL הרלוונטי לפונקציה מסויימת ניתן למצוא בתיעוד ב MSDN.

8. חפשו מה ה DLL המייצא את הפונקציה `MessageBox`. כעת נניח שה- DLL הרלוונטי הינו `Kernel32.dll` (רמז: זה לא!) – כדי לייצר את העטיפה נשתמש בפקודה הבאה:

```
>>> import ctypes
>>> my_library = ctypes.WinDLL("kernel32.dll")
```

כעת ניתן לקרוא לפקודות שונות בצורה מאוד פשוטה – לדוגמא: `my_library.Blah()`

9. כעת נסו להריץ את הפונקציה `MessageBox` מתוך ה DLL שמצאתם קודם לכן – האם הפונקציה נמצאה? למה לא? נסו להיזכר במה שלמדנו בשיעור לגבי ANSI / Unicode . למרבית הפונקציות יש מימוש לשני שיטות קידוד הטקסט הללו – כאשר עבור ANSI מוסיפים 'A' בסוף שם הפונקציה, ועבור Unicode משתמשים ב 'W'.

10. חפשו את הפונקציה המתאימה, קראו לה וצרו חלון קופץ.

**הרחבה:** ב- `ctypes` ניתן להשתמש ב `syntax` הבא עבור גישה לספריות: `ctypes.windll.kernel32.Blah()` (במקרה זה יטען ה- DLL של `Kernel32.dll` ומתוכו תקרא הפונקציה `Blah`)