הדגמת 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 https://docs.microsoft.com/he-il/windows/win32/apiindex/windows-api-list 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 שמיוצאת על ידי מערכת ההפעלה, ובחלק אחר – נעשו עטיפות נוספות המקלות את השימוש בפונקציות אלה.

1 תרגיל 1

בתרגיל זה נתנסה בפונקציה בסיסית ב-winAPI בשם *MessageBox*. אתם יכולים לנחש מה היא עושה? תחילה, חפשו את הפונקציה בתיעוד באינטרנט. שימו לב לאילו פרמטרים היא מקבלת ומה היא מחזירה:

```
int MessageBox(
  HWND    hWnd,
  LPCTSTR lpText,
  LPCTSTR lpCaption,
  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 בתיעוד ☺).

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

הרחבה: מתוך התיעוד 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 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?

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

4. ייבאו את החבילה win32api ונסו להריץ מתוכה את MessageBox עם כמה פרמטרים שהיא דורשת.
5. 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)?
6. 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 יכולה לעטוף את האובייקטים כך שנוכל לקרוא להם פשוט מפייתון.

למה זה מעניין אותנו? הפונקציות ומבני הנתונים של WinAPI הם בפורמט C ולמעשה מיוצאים על ידיDLL-ים של מערכת ההפעלה. את ה DLL הרלוונטי לפונקציה מסוימת ניתן למצוא בתיעוד של מיקרוסופט.

1. 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 – לדוגמה my_library.Blah():

2. 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'.

3. 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()).

2 תרגיל 2

עכשיו כשכולנו יודעים-WinAPI הגיע הזמן להשתמש בו!
כתבו תוכנית העוברת באופן חד פעמי על כל החלונות הפתוחים במחשב ומציגה את הכותרת של כל חלון. (היעזרו בפונקציה EnumWindows ושימו לב שהיא מקבלת callback).
שימו לב: אם תספרו את כל החלונות תגיעו למספרים גבוהים, נסו לספור רק את החלונות הנראים.

3 תרגיל 3

כתבו תוכנית שדוגמת כל הזמן את לוח העריכה (העתק-הדבק) של מעה"פ ובכל פעם שמעתיקים טקסט חדש הוא יופיע על המסך בתוכנית (חבילת פייתון שיכולה לעזור win32clipboard).
הנחייה: שימו לב שתהליך קבלת המידע הנוכחי מלוח העריכה כולל כמה שלבים:

- תחילה יש "לפתוח" את לוח העריכה באמצעות OpenClipboard. בצורה זו אנו מקבלים "בעלות" על לוח העריכה – ומונעים מתוכנות אחרות לשנות את התוכן בו.
- לאחר מכן ניתן לקבל את המידע שבתוכו (שימו לב לפרמטרים השונים).
- בסוף יש לסגור את לוח העריכה, כדי לאפשר לתוכנות אחרות להשתמש בו שוב.

הזמן שבו לוח העריכה פתוח צריך להיות מינימאלי – כדי שלא נפריע לתוכנות אחרות להשתמש בו.
ודאו שהתוכנית שלכם עובדת גם על העתקת טקסט באנגלית וגם בעברית!

שימו לב: ה-CMD לא תומך בעברית, בידרו את התוכנית בסביבה שתומכת בעברית כגון ה-pycharm.

היזהרו לא להיכנס ללולאה אין סופית.

קרדיט

תומר גלון

עומר ברק