תרגיל בניית Shell

בתרגיל זה נממש Shell פשוט (בדומה ל CMD) משל עצמנו ב-Python. ה-Shell שלנו יכיל את רוב הפקודות Shell הסטנדרטיות של Windows ויאפשר לנו להוסיף בצורה קלה ובטוחה פעולות Shell משל עצמנו.

אם נזכר, אז ראינו בשיעור כי חלק מהפקודות של ה-CMD של Windows הן הרי בעצם תהליכים שרצים כ-Process נפרד. באופן דומה, כאשר ה-Shell שלנו יקבל פקודה הוא יעבור על מספר תיקיות שונות ויבדוק האם הפקודה שקיבלנו הינה בעצם שם של תוכנה באחת התיקיות האלה, ואם כן הוא יפעיל אותה עם הפרמטרים שקיבלנו, ויחזיר את הפלט שלה. את רשימה זו הגדירו כמשתנה

בנוסף, נוסיף ל-Shell שלנו תמיכה ב-Pipe-ים ו- Redirection:

- **Pipe**: יהיה ניתן להעביר פלט של פקודה אחת לקלט של הפקודה הבאה (*cmd1 | cmd2* – במקרה זה הפלט של cmd1 עובר ל- cmd2).
- **Redirection**: כרגע נתמוך רק באפשרות של העברת הפלט של הפקודה האחרונה לקובץ (cmd > output_file – הפלט של הפקודה cmd יכתוב לתוך הקובץ output_file).

מעבר לכך, נשלב ב-Shell שלנו תמיכה גם בהרצה של script-ים של Python אותם אנחנו נכתוב ונשמור בתיקייה מסוימת. על מנת להראות שה-script-ים שלנו באמת נטענים ועובדים כפי שצריך, נכתוב מספר סקריפטים לדוגמא:

- Hello.py – סקריפט שידפיס למסך "Hello" ואת שם המשתמש שהריץ אותו. סקריפט זה ישמש בעיקר לצורכי "דיבאגינג" ובדיקות.
- HexDump – סקריפט המקבל נתיב לקובץ בינארי ומדפיס אותו בצורה הקסדצימלית, כאשר כל <u>בית</u> בקובץ מומר למספר הקסדצימלי (עבור מחרוזת בפייתון - בית = תו בודד) .
- **<u>בונוס</u>**: grep – סקריפט המקבל 2 פרמטרים – (1) נתיב לקובץ ו- (2) Regex ומדפיס את כל השורות בקובץ אשר יש בהן התאמה ל-Regex. על מנת לממש את הפעולה בצורה הטבעית ביותר - נוסיף תמיכה לקבלת תוכן קובץ (או פלט של פקודה אחרת) דרך ה-stdin שעליו יבוצע החיפוש הנ"ל (להסבר נוסף - חפשו מידע על הפקודה grep באינטרנט).

התיקייה שבה ה-Shell יחפש את סקריפטי הפייתון תשמר כמשתנה סביבה בשם **SCRIPTS_PATH** (כך נוכל לקבוע את ערכה טרם הרצת הסקריפט), אך דאגו שאם לא קיים משתנה סביבה בשם הזה - ה-Shell יחפש בתיקייה דיפולטית כלשהי שהגדרתם בקוד. **<u>להרחבה</u>**, ניתן לשמור במשתנה הסביבה מספר תיקיות ולחפש בכל אחת מהן לפי סדר מסוים (באופן דומה לאיך שמשתנה ה-Path עובד ב-Windows). השתמשו בתו מפריד בין התיקיות בדומה לצורת ההפרדה במשתנה Path.

לבסוף, נזכר כי ל-CMD של Windows יש גם מספר פעולות שהן מובנות לתוכו ולא ניתן להריץ אותן בחוץ (או שאין משמעות להרצה שלהן ב-Process נפרד) כמו set (האחראי על משתני סביבה) או cd (המשנה את התיקייה הנוכחית). תחילה, הבינו למה אין משמעות להרצת פקודות אלו ב-Process נפרד, ולאחר מכן מצאו דרך לתמוך בפקודות אלו ב-Shell שלכם (אין צורך לתמוך בשילוב פקודות אלה עם Pipe או Redirection. הן תמיד תורצנה אחת בכל פעם, ולאחר מכן הפלט שלהן יודפס למסך).

<u>הנחיות חשובות לתהליך העבודה על התרגיל - בעמוד הבא</u>

<u>הנחייה לתחילת הפיתוח</u>:

1. התחילו מכתיבת סקריפט בסיסי אשר באופן מחזורי מקבל קלט מהמשתמש ומחזיר פלט, עד אשר הוא מקבל את הפקודה exit (ואז הוא יוצא).
2. הוסיפו פונקציה שתורץ כל פעם עם הקלט שהתקבל מהמשתמש והיא תדאג לבצע את הלוגיקה ולהחזיר פלט בסוף הביצוע. בתור התחלה - נסו לפונקציה להריץ את הפקודה המתקבלת ולהחזיר פלט. מודול שימושי שיכול לעזור כאן הינו *subprocess* ובו הפונקציה *check_output* (קיים רק בפייתון 2.7 ומעלה).
3. נרצה להוסיף זיהוי והרצה של קבצי פייתון. ההרצה שלהם בקוד שלנו תתבצע בדומה להרצה של תוכנה חיצונית, כאשר אנחנו מעבירים אותם כפרמטר ל python.exe. הוסיפו תמיכה זו. (שימו לב להנחיות - יש לחפש את הסקריפטים בתוך התיקיה המוגדרת במשתנה הסביבה SCRIPTS_PATH, ואם הוא לא קיים, אז בתיקיה דיפולטית שהגדרתם. כמובן שגם את השם של משתנה הסביבה וגם את התיקיה הדיפולטית תכננתם לשים כקבועים בתחילת הקוד).
4. כעת נוסיף תמיכה ב PIPE-ים. מומלץ לחפש קצת באינטרנט איך אפשר לעשות את זה. (במקרה זה נרצה להשתמש במחלקה Popen במודול *subprocess* שמאפשרת לנו לשלוט בצורה יותר מלאה ב stdin ו-stdout).

לאחר השלבים הנ"ל נותר להמשיך ולפתח את ה Shell - להוסיף לו אפשרות ל Redirection, פקודות מובנות וכו'.

<u>הערה חשובה</u>: שימו לב לא להשתמש בפרמטר shell=True בפונקציות השונות במודול *subprocess*. כפי שתוכלו לקרוא בתיעוד, הפרמטר מאפשר להריץ פקודות דרך ה-Shell של Windows, מה שלמעשה מאפשר "לעקוף" את כל המימוש של Pipes ו-Redirections שהפעם תצטרכו לממשו בעצמכם בתרגיל.

בהצלחה!