## Visitors

מטרתה של מערכת האימות היא הערכת הקוד וזיהוי שגיאות העלולות לעלות בזמן ריצה (runtime). על מנת לעשות זאת הוגדר עבורה אוסף חוקים עבור הפעולות המותרות והאסורות בקוד. כדי לבדוק האם קוד נתון עומד באוסף החוקים המותר יש צורך בעיבוד הקוד לכדי מבנה נתונים המאפשר לבדוק אותם. בהינתן אוסף זה יש צורך במנוע אשר יעבד את הקוד ויוודא אותו. המנוע אשר נכתב מורכב מתתי מנועים אשר כולם מתבססים על גבי תשתית ה-ast visitor של פייתון.

### אוסף החוקים – מותר ואסור

החוקים אשר הגדרנו מהווים תת-אוסף (subset) של שפת פייתון. הם נבחרו כך שנוכל לעבד נאמנה קוד פייתון בעל תחביר (syntax) סטנדרטי ובעל שימוש סטנדרטי במאפייני השפה. כחלק מהרצון לתמוך גם בתוכניות מורכבות יותר נכתב גם רכיב ה-Simpler אשר מתואר בפרק משל עצמו.

חוקי השפה:

* אסור לבצע השמת attributes למשתנים פשוטים (int, str, tuple, list, …)
* אסור לבצע השמה שמקורה במשתנה לא מוגדר
* אסור לקרוא ערך של משתנה לא מוגדר
* אסורה ירושה מרובה
* מה עוד?

### מנוע העיבוד

על מנת לעבד קוד נתון המערכת מניחה הנחות מקדימות:

* כלל הקוד כתוב בקובץ אחד ( תפקידו של ה-simpler)
* כלל הקוד עובד ל-Abstract Syntax Tree (תפקידו של מנוע ה-ast)

בהינתן ההנחות המקדימות המנוע עובר בצורה רקורסיבית מעל גבי ה-AST ומנסה לזהות פעולות בקוד אשר עלולות לעבור על האיסורים שהוגדרו. המנוע משתמש בתתי מנועים הנקראים Visitors ותפקידם הוא לעבד אזור מסוים בעץ (ה-AST) ולזהות בו דפוסים (כל Visitor מקבל אחריות אחרת).

### ProgramVisitor

מנוע ה-Visitor הראשי. תפקידו הוא לעבור על רצף השמות, הגדרות משתנים, הגדרות פונקציות ועוד מבני בקרת זרימה, לעבד אותם ולהפעיל את ה-Visitors האחרים בהתאם לתפקידם.

בתחילת ריצתו המנוע מאתחל את מבני הנתונים איתם נעבוד לאורך הדרך:

* Stack – מבנה נתונים המתאר את מחסנית זמן הריצה ומורכב ממסגרות (frames) המכילות מידע על המשתנים שמוגדרים ברמה המתאימה להן במחסנית.
* AbstractState – מבנה הנתונים (המתואר בפרק משל עצמו) המכיל את כל הידע על הקשרים בין משתנים.
* Functions – מידע על פונקציות שהוגדרו לאורך הדרך.
* Classes – מידע על פונקציות שהוגדרו לאורך הדרך.

המנוע מסוגל להתמודד עם הצמתים הבאים בעץ:

* הגדרת מחלקה

בדיקת ירושה מרובה  
הפעלת ה-ClassDefVisitor

* הגדרת פונקציה

הפעלת ה-ClassDefVisitor

* הפעלת ביטוי

הפעלת FunctionDefVisitor

* ביצוע השמה

הפעלת AssignVisitor

* לולאת For

מחושב lub מעל גבי איברי הרשימה ולאחר מכן גוף הלולאה מחושב פעם אחת עבור ה-lub.

* בלוק if-else

עבור כל תת בלוק משוכפל מבנה הנתונים AbstractState ומורץ גוף הבלוק על גבי המבנה המשוכפל. לאחר מכן אנו מבצעים פעולת lub בין מבנה הנתונים המקורי (מלפני הכניסה לשרשרת ה-if-else) ולבין המבנה הנוכחי. זאת מכיוון שאין אנו יודעים אילו מתתי הבלוקים ירוצו (אם בכלל).

* בלוק try-except-finally

מכיוון שאיננו יודעים איזה חלק מבלוק ה-try ירוץ עד לזריקת שגיאה (אם בכלל) אנו משכפלים את מבנה הנתונים עובר כל רצף הרצות מתחילת הפונקציה ומבצעים lub שלו עם ה-AbstractState המקורי מלפני הכניסה לבלוק.

* הפעלת return

חישוב ערך החזרה אשר צריך לחזור מהפונקציה.

### ClassDefVisitor

מעבד הגדרת מחלקה בעץ ה-AST. מכיוון שהגדרת מחלקה אינה כוללת "הרצת" קוד המידע היבש נשמר במבני נתונים מתאימים ומחושב (קונסטרקטור, פונקציות) רק כאשר אחד מהמנועים מזהה הפעלה שלהם.

* חישוב ירושה

המנוע מזהה את עץ הירושה של האובייקט ושומר אותו במבנה נתונים מתאים על מנת שנוכל בהמשך לבצע פעולות בהתאם.

* זיהוי פונקציות

המנוע מפריד בין פונקציות סטטיות לפונקציות של האובייקט ומאחסן גם אותן במבנה נתונים מתאים.

### FunctionDefVisitor

Visitor בסיסי אשר מטרתו היא רק שמירת הגדרת פונקציה ושיוכה לשם המתאים לה לשם שימוש ע"י שאר המנועים.

### AssignVisitor

אחראי על ביצוע ההשמות לתוך משתנים/attributes. מנוע זה מקבך צומת העץ ה-AST ומחשב את ערכי ההשמה הרלוונטיים אליה בהתאם למידע אשר מזוהה בצומת. בהתאם לכך השמה יכולה להיות:

* משנה אחר (x=y)
* ערך attribute (x = y.a)
* string (x="hello")
* מספר (x=2)
* רשימה (x=[1,2,3])
* Tuple (x=(1,2,3))
* מילון (x={1:2,3:4})

### CallVisitor

אחראי על קריאות לפונקציות (כולל קונסטרקטורים). כאשר מתבצעת קריאה תפקידו הוא לזהות את סוג הקריאה (קונסטרקטור, פונקציה, פונקציה של אובייקט (מחלקה)) ולהפעיל את הלוגיקה המתאימה לביצוע המשך הפירסור. כאשר הפונקציה המופעלת שייכת לאובייקט (כלומר זוהי פונקציה של מחלקת אב שלו) תפקידו של מנוע זה הוא לאתחל את אובייקט ה-self לפני הקריאה לפונקציה על מנת שקריאת ורישום ערכים אליו תישאר עקבית גם בתוך גוף הפונקציה.

מקרה מיוחד הנו כאשר לאובייקט יכולים להיות מספר מימושים עבור אותה הפונקציה. מקרה זה מתרחש כאשר אובייקטים שונים עברו lub והייתה להם פונקציה בעלת שם זהה. במקרה זה נחשב הרצה של כל אחד מהמימושים ולאחר מכן נבצע lub על התוצאות.