היי ??

הבנתי שלשניכם כבר יש את מסמך התרגיל של RPyC אבל רציתי לשלוח לכם גם טעויות נפוצות שמצאתי באחד ה-OneNote-ים של התרגיל:

שורת הרצה אחת ולא הרבה בקשות

הרבה חניכים חושבים שיש עקרון של "שורת הרצה". נגיד בשורה הבאה:

c.modules.os.x.y

הם מתארים שהפקודה נבנית כולה אצל ה-client, ורק כשהשורה נגמרת זה נשלח לשרת. כלומר, c.modules.os מחזיר איזשהו אובייקט שה-getattr שלו נדרס. אז .x עליו מחזיר אובייקט חדש ששומר את השם x, וכשעושים עליו .y מוחזר אובייקט חדש שעוטף את האובייקט הקודם, וזוכר שהתווסף y. רק כש"נגמרת השורה", נעשית פנייה לשרת. הדיזיין הזה לא עובד, כי בשום שלב המשתמש לא יודע ש"נגמרה השורה".

בעיה דומה שחוזרת על עצמה היא שחניכים חושבים שהם יכולים לקבל את הרצף של "כל השורה" (עוד לפני שהם מגיעים לדיזיין שכולל עטיפה של אובייקטים בכל פעם שעושים getattr). זאת אומרת, במקרה של c.modules.os.x.y, הם חושבים אפשר לקבל את הסטרינג os.x.y ואז לעשות לו exec או eval על השרת. כשהם מבינים שאין להם דרך לקבל את כל הסטרינג הזה, הם מציעים את הדיזיין שהזכרתי קודם, של אובייקטים שעוטפים אחד את השני (ואז מהאובייקטים האלה אפשר לשחזר את הסטרינג).

דוגמה טובה לגרום להם להבין את ההבדל היא c.modules.sys לעומת c.modules.sys.executable. לפי הדיזיין השגוי, לא ברור איך במקרה הראשון אחרי sys יוצאת בקשה לשרת, ובמקרה השני יוצאת בקשה לשרת רק אחרי executable.

טסטים זה קשה

יש כמה דרכים לעשות טסטים לתרגיל הזה. דרך אחת היא להריץ את השרת ב-subprocess, להריץ קליינט שפונה לשרת בפרוסס הראשי, ולהרוג את השרת בסוף כל טסט. החסרון הוא שמריצים פרוסס בכל טסט, וצריך לדאוג להרוג אותו. פתרון אלטרנטיבי הוא להפריד בין החלק של התקשורת לבין הלוגיקה של rpyc במימוש, ככה שיש קלאס שאחראי לתקשורת (או לחילופין, שני קלאסים – אחד בשרת ואחד בלקוח). במקרה הזה, בהרצה רגילה הקלאס שאחראי לתקשורת ישתמש ב-requests ו-flask, אבל לצורך unittests אפשר להחליף אותו בקלאס שפועל ישירות בזכרון. זה דיזיין שקצת קשה לעשות נכון, והבדיקה שלו בסוף פחות יסודית (כי הוא לא בודק את השימוש ב-flask וב-requests), אבל אפשר לעשות את שני סוגי הטסטים. כברירת מחדל, עדיף שיעשו רק את המימוש עם subprocess כי הוא פשוט יותר למימוש. בתור הרחבה אפשר לעשות את סוג הטסט השני, ובכל מקרה כדאי לדבר עליו עם החניכים כי הוא מלמד על שיקולי דיזיין לצורכי טסטים. צריך לזכור שהם לא בהכרח למדו mockים בשלב הזה בקורס, מה שהיה יכול להיות עוד אסטרטגיה לכתוב טסטים לקלאס.

איך עושים ID?

חניכים מתלבטים לפעמים איך לממש את המזהה שיישלח בין הלקוח לשרת. הם לא בהכרח מכירים את הפונקציה id, ולא בהכרח יודעים שכבר יש איזשהו מזהה של אובייקט שהם יכולים להשתמש בו. במקרה הזה, הם יכולים להציע שימוש ב-counter שנשמר בשרת והם מעלים אותו בעצמם.

מה זה \_\_call\_\_ בעצם?

גם לא כל החניכים מכירים את \_\_call\_\_, אז גם אם הם יודעים שהם צריכים לדרוס "קריאה לפונקציה", הם לא יודעים מה הפונקציה שהם צריכים לממש.

לא בדיוק הבנתי מה זה modules.. מי צריך לעשות import ולמה?

ההתייחסות ל-modules היא מבלבלת. בסוף, modules צריך להיות instance של class ששונה מהפרוקסי, וכשעושים עליו getattr הם גורם ל-import בשרת, ומחזיר אובייקט פרוקסי בקליינט. אבל לוקח זמן לחניכים להבין את זה. צריך להבהיר לחניכים שאפשר לעשות import לכל מודול, ולא לקבוצת מודולים מוסכמת מראש. כלומר, getattr של modules צריכה כבר לעשות גישה לשרת, ואי אפשר לשמור מראש על modules את קבוצת המודולים שאפשר לעשות להם אימפורט.

יש דיזיין אלטרנטיבי לטיפול ב-modules, שאני חושב שיכול לעבוד. אפשר לדרוס את getattr של ה-connection, ולממש קלאס module loader שנעשה בו שימוש בשרת. במימוש של הקלאס הזה דורסים את getattr, ככה שיעשו אימפורט למודול לפי השם שהוא קבל. אז, getattr של ה-connection כבר הוא עושה בקשה לשרת, והשרת דואג ש- c.modules יחזיר כבר בעצמו proxy עבור module loader. הטיפוס של הפרוקסי הזה הוא אותו טיפוס של c.modules.os. היתרון של המימוש הוא שהוא הופך את ה-client ליותר טפש אפילו – יש בו רק אובייקט connection ואובייקט proxy.

יותר מדי תשתיתיות

לפעמים חניכים מודאגים מכמות האובייקטים שיישמרו ב-dict בצד השרת. צריך להבהיר שזו לא דאגה במסגרת התרגיל הזה. פתרון אפשרי שהם מציעים הוא לעשות caching לתוצאות של ביטויים שלמים (למשל – לזכור מה הערך של "x.y.z", וככה לא צריך לשמור מיפוי עבור x ועבור x.y). זה לא מאוד מסתדר עם הדיזיין "הנכון" בתרגיל, אבל זה כן יכול להסתדר עקרונית עם דיזיין בו x.y.z מוציא בקשה אחת (אם כי בסוף כל הדיזיין הזה נופל). בכל מקרה, אם חניכים מציעים לעשות caching אפשר להראות להם למה זה יכול להביא להתנהגויות לא נכונות. למשל, אם איכשהו נעשה caching ל-x.y.z ואז משתמש עשה x.y.z = 3, אז התשובה ב-cache כבר לא נכונה. (שימו לב שהדוגמה הזו מניחה שאפשר לשנות ערכים, ולא רק לקרוא אותם, וזה לא חלק מהדרישות הבסיסיות של התרגיל).

נחמד לדעת

• במימוש הבסיסי list ו-tuple הם גם proxy.