10 סדר בתים (Endianness) ומספרים שלמים

עשינו קצת עבודה בהעברת טקסט דרך הרשת. אבל עכשיו אנחנו רוצים לעשות משהו אחר: אנחנו רוצים להעביר נתוני מספרים שלמים בינאריים.

בוודאי שהיינו יכולים פשוט להמיר את המספרים למחרוזות, אבל זה בזבזני יותר ממה שצריך. ייצוג בינארי הוא יותר קומפקטי וחוסך רוחב פס.

אבל הרשת יכולה רק לשלוח ולקבל בתים! איך נוכל להמיר מספרים שרירותיים לבתים בודדים?

על זה בדיוק מדבר הפרק הזה.

אנחנו רוצים:
* להמיר מספרים שלמים לרצפי בתים
* להמיר רצפי בתים בחזרה למספרים שלמים

ובפרק זה נסתכל על:
* איך מספרים מיוצגים על ידי רצפי בתים
* באיזה סדר הבתים האלה הולכים
* איך להמיר מספר לרצף בתים בפייתון
* איך להמיר רצף בתים למספר בפייתון

נקודות מפתח לשים לב אליהן:
* מספרים שלמים יכולים להיות מיוצגים על ידי רצפי בתים.
* נמיר מספרים שלמים לרצפי בתים לפני שנשדר אותם ברשת.
* נמיר רצפי בתים בחזרה למספרים שלמים כשנקבל אותם מהרשת.
* Big-Endian ו-Little-Endian הן שתי דרכים שונות לסידור רצפי הבתים האלה.
* פייתון מציעה פונקציונליות מובנית להמרת מספרים שלמים לרצפי בתים ובחזרה.

10.1 ייצוגי מספרים שלמים

בחלק זה נצלול עמוק לתוך איך מספר שלם יכול להיות מיוצג על ידי רצף של בתים בודדים.

10.1.1 ייצוג בתים עשרוני

בוא נסתכל איך מספרים שלמים מיוצגים כרצפים של בתים. רצפי הבתים האלה הם מה שנשלח דרך הרשת כדי לשלוח ערכי מספרים שלמים למערכות אחרות.

בית בודד (בהקשר זה נגדיר בית כ-8 סיביות כרגיל) יכול לקודד ערכים בינאריים מ-00000000 עד 11111111. בעשרוני, המספרים האלה הולכים מ-0 עד 255.

אז מה קורה אם אתה רוצה לאחסן מספר גדול מ-255? כמו 256? במקרה כזה, אתה צריך להשתמש בבית שני כדי לאחסן את הערך הנוסף.

ככל שאתה משתמש ביותר בתים כדי לייצג מספר שלם, כך גדול יותר טווח המספרים השלמים שאתה יכול לייצג. בית אחד יכול לאחסן מ-0 עד 255. שני בתים יכולים לאחסן מ-0 עד 65535.

בחשיבה אחרת על זה, 65536 הוא מספר הצירופים של 1 ו-0 שאתה יכול להיות במספר 16-סיביות.

חלק זה מדבר על מספרים שלמים לא-שליליים בלבד. מספרים בנקודה צפה משתמשים בקידוד שונה.
מספרים שלמים שליליים משתמשים בטכניקה דומה לחיוביים, אבל נשמור על זה פשוט לעכשיו ונתעלם מהם.

בוא נסתכל מה קורה כשאנחנו סופרים למעלה מ-253 עד 259 במספר 16-סיביות. מכיוון ש-259 גדול ממה שבית בודד יכול להכיל, נשתמש בשני בתים (המכילים מספרים מ-0 עד 255), עם הערך העשרוני המתאים מיוצג בצד ימין:

0 253 מייצג 253
0 254 מייצג 254
0 255 מייצג 255
1 0 מייצג 256
1 1 מייצג 257
1 2 מייצג 258
1 3 מייצג 259

שים לב שהבית בצד ימין "התגלגל" מ-255 ל-0 כמו מד מרחק. זה כמעט כאילו שהבית הזה הוא "מקום האחדות" והבית משמאל הוא "מקום ה-256ים"... כמו להסתכל על מערכת מספרים בבסיס 256, כמעט.

אנחנו יכולים לחשב את הערך העשרוני של המספר על ידי לקיחת הבית הראשון והכפלתו ב-256, ואז הוספת הערך של הבית השני:

1 * 256 + 3 = 259

או בדוגמה הזו, שבה שני בתים עם ערכים 17 ו-178 מייצגים את הערך 4530:

17 * 256 + 178 = 4530

אף אחד מהמספרים 17 ו-178 אינו גדול מ-255, אז שניהם מתאימים בבית בודד כל אחד.

אז כל מספר שלם יכול להיות מיוצג באופן מושלם על ידי רצף של בתים. אתה פשוט צריך יותר בתים ברצף כדי לייצג מספרים גדולים יותר.

10.1.2 ייצוגי בתים בינאריים
בינארי, הקסדצימלי ועשרוני הם פשוט כולם "שפות" שונות לכתיבת ערכים.

אז היינו יכולים לכתוב מחדש את כל החלק הקודם של המסמך על ידי תרגום פשוט של כל המספרים העשרוניים לבינאריים, וזה עדיין היה נכון באותה מידה.

למעשה, בוא נעשה את זה לדוגמה מהחלק הקודם. זכור: זה שווה ערך מספרית - פשוט שינינו את המספרים מעשרוני לבינארי. כל המושגים האחרים זהים.

11111101 00000000 מייצג 11111101 (253 עשרוני)
11111110 00000000 מייצג 11111110 (254 עשרוני)
11111111 00000000 מייצג 11111111 (255 עשרוני)
00000000 00000001 מייצג 100000000 (256 עשרוני)
00000001 00000001 מייצג 100000001 (257 עשרוני)
00000010 00000001 מייצג 100000010 (258 עשרוני)
00000011 00000001 מייצג 100000011 (259 עשרוני)

אבל רגע - רואה את התבנית? אם אתה פשוט מדביק את שני הבתים יחד אתה מקבל בדיוק את אותו מספר כמו בייצוג הבינארי! (בהתעלמות מאפסים מובילים.)

באמת כל מה שעשינו הוא לקחת את הייצוג הבינארי של מספר ולפצל אותו לחתיכות של 8 סיביות.

10.1.3 ייצוגי בתים הקסדצימליים
שוב, זה לא משנה באיזה בסיס מספרים אנחנו משתמשים - הם פשוט כולם "שפות" שונות לייצוג ערך מספרי.

מתכנתים אוהבים הקס כי זה מאוד תואם לבתים (כל בית הוא 2 ספרות הקס). בוא נעשה את אותה טבלה שוב, הפעם בהקס:

00 fd	מייצג 00fd (253 עשרוני)
00 fe	מייצג 00fe (254 עשרוני)
00 ff	מייצג 00ff (255 עשרוני)
01 00	מייצג 0100 (256 עשרוני)
01 01	מייצג 0101 (257 עשרוני)
01 02	מייצג 0102 (258 עשרוני)
01 03	מייצג 0103 (259 עשרוני)

תסתכל על זה שוב! הייצוג ההקסדצימלי של המספר הוא אותו דבר כמו שני הבתים פשוט דחוסים יחד! נוח במיוחד.

10.2 סדר בתים (Endianness)
מוכן לקבל מפתיע בעבודות?

הרגע סיימתי להגיד לך שמספר כמו (בהקס):

45f2

יכול להיות מיוצג על ידי שני הבתים האלה:

f2 45

אבל נחש מה! חלק מהמערכות ייצגו את 0x45f2 כ:

f2 45

זה הפוך! זה בדומה לכך שאני אומר "אני רוצה 123 פרוסות לחם" כשבעצם רציתי 321!

יש שם לשים את הבתים הפוך ככה. אנחנו אומרים שייצוגים כאלה הם little endian.

זה אומר שה"קצה הקטן" של המספר (בית ה"אחדות", אם אני יכול לקרוא לו ככה) מגיע בקצה הקדמי.

הדרך היותר נורמלית, יותר קדימה לכתוב את זה (כמו שעשינו בהתחלה, שבו המספר 0x45f2 היה מיוצג באופן סביר בסדר f2 45) נקראת big endian. הבית במקום הערך הגדול ביותר (נקרא גם הבית המשמעותי ביותר) נמצא בקצה הקדמי.

החדשות הרעות הן שכמעט כל דגמי המעבד של אינטל הם little-endian.

החדשות הטובות הן שמחשבי Mac M1 הם big-endian.

החדשות הטובות עוד יותר הן שכל המספרים ברשת משודרים כ-big-endian, בדרך ההגיונית.

וכשאני אומר "כל", אני מתכוון "כמות מסוימת". אם שני הצדדים מסכימים לשדר ב-little endian, אין חוק נגד זה. זה היה הגיוני אם השולח והמקבל היו שניהם ארכיטקטורות little-endian - למה לבזבז זמן בהיפוך בתים רק כדי להפוך אותם בחזרה? אבל רוב הפרוטוקולים מציינים big-endian.

סדר בתים big-endian נקרא network byte order בהקשרי רשת מסיבה זו.

10.3 פייתון וסדר בתים
מה אם יש לך איזשהו מספר בפייתון, איך אתה ממיר אותו לרצף בתים?

למרבה המזל, יש פונקציה מובנית שעוזרת עם זה: .to_bytes().

ויש אחת שהולכת לכיוון השני: .from_bytes()

זה אפילו מאפשר לך לציין את סדר הבתים! מכיוון שנשתמש בזה כדי לשדר בתים דרך הרשת, תמיד נשתמש ב-"big" endian.

10.3.1 המרת מספר לבתים
הנה הדגמה שבה אנחנו לוקחים את המספר 3490 ומאחסנים אותו כמחרוזת בתים של 2 בתים בסדר big-endian.

שים לב שאנחנו מעבירים שני דברים לשיטה .to_bytes(): מספר הבתים לתוצאה, ו-"big" אם זה צריך להיות big-endian, או "little" אם זה צריך להיות little endian.

Continuing the translation:

סדר בתים מברירת מחדל ל-"big". בגרסאות ישנות יותר, אתה עדיין צריך להיות מפורש.

```python
n = 3490

bytes = n.to_bytes(2, "big")
```

אם נדפיס אותם נראה את ערכי הבתים:

```python
for b in bytes:
    print(b)
13
162
```

אלה ערכי הבתים ב-big-endian שמרכיבים את המספר 3490. אנחנו יכולים לאמת בקלות ש- 13 * 256 + 162 == 3490.

אם תנסה לאחסן את המספר 70,000 בשני בתים, תקבל OverflowError. שני בתים אינם מספיק גדולים לאחסן ערכים מעל 65535 - תצטרך להוסיף עוד בית.

בוא נעשה עוד דוגמה בהקס:

```python
n = 0xABCD
bytes = n.to_bytes(2, "big")

for b in bytes:
    print(f"{b:02X}")  # הדפס בהקס
```

מדפיס:

```
AB
CD
```

זה אותן ספרות כמו הערך המקורי שמאוחסן ב-n!

10.3.2 המרת בתים בחזרה למספר
בוא נעשה את הסיבוב המלא. אנחנו הולכים ליצור מספר הקס ולהמיר אותו לבתים, כמו שעשינו בחלק הקודם. אז אפילו נדפיס את מחרוזת הבתים כדי לראות איך היא נראית.

אז נמיר את מחרוזת הבתים הזו בחזרה למספר ונדפיס אותה כדי לוודא שהיא תואמת למקור.

```python
n = 0x0102
bytes = n.to_bytes(2, "big")

print(bytes)
```

נותן את הפלט:

```
b'\x01\x02'
```

ה-b בהתחלה אומר שזו מחרוזת בתים (בניגוד למחרוזת רגילה) וה-\x הוא רצף בריחה שמופיע לפני מספר הקס בן 2 ספרות.

מכיוון שהמספר המקורי שלנו היה 0x0102, הגיוני שלשני הבתים במחרוזת הבתים יש ערכים \x01 ו-\x02.

עכשיו בוא נמיר את המחרוזת הזו בחזרה ונדפיס בהקס:

```python
v = int.from_bytes(bytes, "big")

print(f"{v:04x}")
```

וזה מדפיס:

```
0102
```

בדיוק כמו הערך המקורי שלנו!

10.4 שאלות לחשיבה

1. איך אפשר להחליף את סדר הבתים במספר בן 2 בתים בעזרת שימוש רק בשיטות .to_bytes() ו-.from_bytes()? (כלומר להפוך את הבתים.) איך אפשר לעשות זאת בלי להשתמש בלולאות או בשיטות אחרות? (רמז: "big" ו-"little"!)

2. תאר במילים שלך את ההבדל בין Big-Endian ל-Little-Endian.

3. מהו Network Byte Order?

4. למה לא פשוט לשלוח מספר שלם בבת אחת במקום לפרק אותו לבתים?

5. Little-endian פשוט נראה הפוך. למה הוא בכלל קיים? עשה קצת חיפוש באינטרנט כדי לענות על השאלה הזו.