זהות אובייקט, שינוי אובייקט וגיבוב

מבוא למדעי המחשב - 67101

מרצה: אריה שלזינגר

מתרגלים: יפעת חדד ואורי מאיר

סוכם ע"י: שריה אנסבכר

סמסטר ב' תשפ"ג, האונ' העברית

אשמח לקבל הערות והארות על הסיכומים על מנת לשפרם בעתיד,
כל הערה ולו הפעוטה ביותר)אפילו פסיק שאינו במקום או רווח מיותר(תתקבל בברכה;
אתם מוזמנים לכתוב לי לתיבת הדוא"ל: sraya.ansbacher@mail.huji.ac.il.
לסיכומים נוספים היכנסו לאתר:
אקסיומת השלמות - סיכומי הרצאות במתמטיקה
https://srayaa.wixsite.com/math

זהות אובייקט

הפונקציה 'id' מקבלת אובייקט יחיד (מכל טיפוס אפשרי) ומחזירה מספר ייחודי עבור אובייקט זה ("מס' הת"ז שלו"), מובטח לנו שהמספר יישאר קבוע לאורך הריצה הנוכחית של פייתון[1] ושלא יהיה אובייקט נוסף בעל אותו מספר id בריצה זו[2]; בד"כ מספר זה יהיה פשוט הכתובת של האובייקט בזיכרון המחשב.

יצרנו אובייקט חדש בפייתון, ביצענו השמה למשתנה על האובייקט ואז ביצענו על האובייקט פעולות שונות ומשונות, אולי גם נתנו לו שמות נוספים על הראשון ולא זו אף זו: יצרנו אובייקטים נוספים שאולי זהים בערכם לראשון, אבל...

- האם ה-id של המשתנה[3] נשאר כשהיה?
- האם ה-id של השמות האחרים של אותו אובייקט זהים לזה של הראשון?
- האם כתובות ה-id של שני אובייקטים בעלי ערך זהה שוות זו לזו?

מתי ה-id משתנה[4]?

ראשית נזכור שלמעט אובייקטים מסוימים[5] ה-id של אובייקט נשאר קבוע ויחיד רק במשך הריצה הנוכחית של פייתון ורק עבור אובייקטים אלו העובדה שהערך של שני משתנים זהה גוררת שגם ה-id שלהם זהה. שנית, השמת משתנה אחד ברעהו[6] (ראה דוגמה להלן) גורמת לכך ששניהם יצביעו על אותו אובייקט ולכן כל עוד לא קרה אחד מן המקרים המפורטים לעיל (אלו שמשנים את ה-id) יהיה להם את אותו id:

```
>>> a = "something"
>>> b = a
>>> id(a) == id(b)
True
>>> id(a)
2437717015280
>>> a = "x"
>>> id(a) == id(b)
False
>>> id(a)
140733696350024
>>> id(b)
2437717015280
```

♣ כמובן שלא ייתכן שלשני משתנים המחזיקים אובייקטים בעלי ערכים *שונים* באותו זמן יהיה את אותו id, למרות זאת, עבור אובייקטים הניתנים לשינוי ה-id אינו משתנה בהכרח עם שינוי הערך אולם אז כל המשתנים המצביעים אליהם יתעדכנו עם כל שינוי של האובייקט (כל עוד אכן מדובר באותו אובייקט) ולכן יחזיקו את אותו ערך בכל זמן נתון.

---

[1] ישנם אובייקטים מסוימים שה-id שלהם קבוע בכל הריצות של פייתון, נראה זאת בהמשך.
[2] למעשה ה-id הוא מה שקובע אם שני אובייקטים הם שונים או שהם אותו אובייקט.
[3] כשאנו אומרים "id של משתנה" כוונתנו למספר שתחזיר הפונקציה 'id' אם נפעיל אותה על המשתנה, כלומר ל-id של האובייקט עליו מצביע המשתנה ברגע זה, הרי משתנה אינו אובייקט בפני עצמו ואין לו id...
[4] נדגיש שוב: ה-id של אובייקט אינו משתנה לעולם (זו ההגדרה של שני אובייקטים: ה-id שלהם שונה), כשאנו אומרים שה-id השתנה אנחנו מתכוונים שכך זה נראה לנו ובעצם מה שקרה הוא שנוצר אובייקט חדש ואנחנו מתעסקים בו במקום בישן.
[5] בינתיים ראינו שה-id של המספרים מ-5- ועד 256 (כולל הקצוות) קבועים בכל ריצה שהיא.
[6] נקרא גם "aliasing".

נכיר את הקשר הלוגי is המשווה את כתובות ה-id של שני אובייקטים ומחזיר True או False בהתאמה:

```
>>> a is b
False
>>> a = b
>>> a is b
True
```

- ♣ נשים לב להבדל בין 'is' לבין '==': האופרטור '==' משווה את *הערך* של שני אובייקטים ואילו האופרטור 'is' משווה את *כתובות ה-id* שלהם
- ♣ להלן נעסוק במשתנים[7] אולם כל מה שנאמר עבור שני משתנים שנוצרו בהשמות שאינן קשורות זו לזו (ללא aliasing) נכון גם עבור סתם שתי "קריאות" לאובייקטים בעלי ערך זהה מפני שכל "קריאה" כזו יוצרת אובייקט חדש, למשל:

```
>>> id(257)
1406674587408
>>> id(257)
1406674591536
```

כמה כללי אצבע[8] לפני שנתחיל לעבור על הטיפוסים השונים:

- כמעט כל הופעה של אובייקט בקוד (הופעה שלו עצמו ולא של משתנה המצביע אליו) יוצרת אובייקט חדש שיש לו id משל עצמו (וזאת גם אם כבר קיים אובייקט בעל ערך זהה); לפיכך כל הופעה של גרשיים[9] או סוגריים (מרובעים/עגולים/מסולסלים) המגדירים מחרוזת או container אחר (בהתאמה) הן יצירה של אובייקט חדש בעל id שונה, כמו כן כל הופעה של מספר בקוד[10] (הופעה שלו עצמו[11] ולא של משתנה המצביע אליו) היא יצירה של אובייקט חדש ולו id שונה.
- כל הפעלת אופרטור על אובייקטים קיימים יוצרת אובייקט חדש בעל id משלו, כך למשל שרשור רצף אחד לאחר (מאותו הסוג כמובן) או חיסור קבוצות יוצר אובייקט חדש בעל id משלו.
- slicing יוצר תמיד אובייקט חדש גם אם בעל ערך זהה.
- הפעלת פונקציה יוצרת תמיד אובייקט חדש[12] (זה שהיא מחזירה).
- בד"כ[13] שיטות (methods) של אובייקטים הניתנים לשינוי (mutable) אינן יוצרות אובייקט חדש אלא שומרות על ה-id של האובייקט עליו פועלת השיטה ומשנות רק את ערכו; כמובן ששיטות של אובייקטים שאינם ניתנים לשינוי (immutable) יוצרות אובייקט חדש בעל id משלו.
- השמה במשתנה באמצעות הסימן '=' לעולם אינה יוצרת אובייקט חדש אלא יוצרת מצביע חדש על האובייקט המופיע בצד ימין, יש לזכור שייתכן שהכתיבה בצד ימין יצרה אובייקט חדש (ע"פ כל הכללים המפורטים כאן).
- שני משתנים שהאחד הושם בתוך רעהו (aliasing) יצביעו על אותו אובייקט (אותו id) כל עוד לא בוצעה השמה חדשה באחד מהם, לכן גם אם נשנה את האובייקט[14] דרך אחד מהם או באופן ישיר עדיין שניהם יצביעו על אותו משתנה ויהיו בעלי אותו id.
- מה שקורה בתוך סקופ מקומי אינו משפיע על מה שקורה בכללי, כך למשל פונקציה המקבלת משתנה ומבצעת עליו מניפולציות אינה משנה את ה-id שלו כל עוד אינה מחזירה אותו בסיום ובנוסף בוצעה השמה עבור הערך המוחזר באותו משתנה.

---

[7] הסיבה לכך שנעסוק במשתנים היא מפני שלמעשה ה-id של אובייקט אינו יכול להשתנות, הדבר היחיד שיכול לקרות הוא שניצור אובייקט *חדש* ונעביר את ההצבעה של המשתנה אליו.

[8] אין מקרים חריגים שעליהם לא חלים כללים אלו מלבד אלו שנציין בהערות הבאות.

[9] ראה את החריגה בנושא זה אצל מחרוזות.

[10] ראה את פירוט החריגים בפסקה אודות מספרים.

[11] כמו בדוגמה שלעיל עבור המספר 257.

[12] אלא אם היא מחזירה אובייקט שכבר מובנה בפייתון.

[13] אולי תמיד? אני לא מכיר את כל השיטות הקיימות...

[14] אם הוא ניתן לשינוי.

תזכורת: טבלה המפרטת את הטיפוסים העיקריים מבין אלו שלמדנו עד כה:

		types	הסבר קצרצר
numbers		int	מספרים שלמים
		float	מספרים רציונליים
functions		builtin_function_or_method	פונקציות מובנות של פייתון
		function	פונקציות חדשות שאנחנו מגדירים
None		NoneType	כלום, שום דבר
containers	sequences	str	מחרוזות של תווים
		list	רשימה (סדרה של אובייקטים)
		tuple	סדרה שאינה ניתנת לשינוי
	unordered containers	set	קבוצה (ממש כמו במתמטיקה)
		frozenset	קבוצה שאינה ניתנת לשינוי
		dict	מילון (התאמה בין מפתחות לערכים)
boolean expressions		bool	ביטויים בוליאניים
types		type	הטיפוסים השונים בפייתון
iterators		range	מחזיר את כל המספרים השלמים בטווח נתון ובקפיצה נתונה בזה אחר זה
		list_reverseiterator	מחזיר את כל האיברים ברצף (sequence) נתון בסדר הפוך

מספרים

ה-id של המספרים השלמים שבין 5- לבין 256 (כולל הקצוות) אינו משתנה מריצה לריצה (הם שמורים מראש אצל פייתון כדי לייעל את הקוד) ולכן כל שני משתנים המחזיקים מספר מטיפוס 'int' בטווח הנ"ל יצביעו על אותו אובייקט ולכן יהיה להם את אותו id. לעומת זאת כל שתי הופעות בקוד של מספר רציונלי או של מספר שלם שאינו בטווח הנ"ל יוצרות שני אובייקטים נפרדים ולכן ה-id שלהם יהיה זהה.

פונקציות

- א"א לשנות פונקציות לאחר שנוצרו אולם ניתן לבצע השמה על השם שלהן[15] ואז הוא יצביע לעבר אובייקט אחר בעל id שונה. גם עבור פונקציות מובנות בפייתון ה-id משתנה מריצה לריצה.
- הגדרת פונקציה באמצעות הפקודה 'def' יוצרת פונקציה חדשה תמיד (כמו השמה שאינה aliasing) ולכן אם פונקציה הוגדרה מחדש באמצעות פקודה זו ה-id שלה "ישתנה"[16].
- ניתן לבצע aliasing על פונקציות ובמקרה כזה ה-id יהיה זהה.
- הפעלת פונקציה שיש בה משתנה דיפולטיבי יוצרת את אותו אובייקט אם הוא עדיין אינו קיים, גם אם נכניס במשתנה הדיפולטיבי אובייקט אחר ייוצר האובייקט הדיפולטיבי ורק השם של המשתנה יעבור לאובייקט החדש.

None

קיים רק אובייקט אחד מטיפוס None, כל שני משתנים המחזיקים את הערך None בעת ובעונה אחת הם בעלי id זהה.

רצפים (sequences)

slicing, שרשור רצף אחד למשנהו (ע"י האופרטור "+") והכפלת רצף במספר שלם (ע"י האופרטור "*") יוצרים רצף חדש (id שונה), למרות זאת ברצפים שאינם מחרוזות האיברים ברצף זהים לאלו שברצף המקורי (id זהה)[17]; כדי להעתיק רצף כך שגם איבריו יהיו אובייקטים חדשים יש לייבא את הפונקציה 'deepcopy' מהספרייה 'copy' ולהפעיל אותה על הרצף הרצוי.

---

[15] כדי לבצע השמה לאובייקט קיים יש להשתמש בסימון "=" וכדי לבצע השמה לפונקציה שעוד לא קיימת נשתמש במילת הקוד 'def'.

[16] כמובן שהוא לא באמת משתנה אלא שנוצר אובייקט *חדש* ואנו מעבירים אליו את ההצבעה של שם הפונקציה.

[17] העתקה כזו נקראת 'shallow copy' (העתקה רדודה) בניגוד ל-'deep copy' (העתקה עמוקה) שתוצג להלן.

מחרוזות

- בניגוד לשאר הרצפים גישה לאינדקס במחרוזת יוצרת מחרוזת חדשה המכילה את התו שבאינדקס המבוקש.
- לפעמים, כדי לחסוך בזיכרון, פייתון מתייחס לשתי פעמים שבהם יצרנו מחרוזת קצרה יחסית בתור אובייקט אחד:

```
>>> a = "abc"
>>> b = "abc"
>>> a is b
True
```

רשימות

- רשימה היא בסך הכל סדרה של משתנים המצביעים לעבר אובייקטים קיימים[18] ולכן כל פעולה על גישה לאינדקס ברשימה שקולה להפעלת אותה פעולה על משתנה המצביע על האובייקט המתאים ברשימה (אם ברשימה מופיע משתנה נוסף המצביע בעצמו על אובייקט אז מדובר ב-aliasing).
- השמה מהצורה הבאה אינה יוצרת רשימה חדשה (בניגוד שלמה שקורה בד"כ ב-slicing) אלא מעדכנת את הרשימה הקיימת:

```
>>> lst = ["1", 2, [3], {4}]
>>> id(lst)
2413706824832
>>> lst[1:3] = ["a", ["b"], 3]
>>> lst
['1', 'a', ['b'], 3, {4}]
>>> id(lst)
2413706824832
```

- לא מצאתי שיטות (methods) של רשימות היוצרות אובייקט חדש, כל אלו שראיתי מעדכנות את האובייקט הקיים ושומרות על ה-id.
- בגלל שרשימות ניתנות לשינוי השמות מהצורה "+="[19] ו-"*=" אינם יוצרים אובייקט חדש אלא משנים את הרשימה הקיימת (ה-id נשאר זהה):

```
>>> lst = []
>>> id(lst)
1357875956480
>>> lst1 += [1, 2, 3]
>>> id(lst)
1357875956480
```

וזאת בניגוד להשמה מהצורה:

```
>>> lst = lst + [4,5,6]
>>> id(lst)
1357875988736
```

---

[18] אמנם ייתכן שיצירתם היתה יחד עם יצירת הרשימה אך הרשימה עצמה היא סדרה של מצביעים לעבר אובייקטים אלו.
[19] נזכור שהסימן "+=" מאפשר לשרשר לרשימה כל רצף אחר.

tuples

גם tuple הוא סדרה של משתנים אלא שבגלל ש-tuples אינם ניתנים לשינוי ה-id של כל משתנה בסדרה נשאר קבוע במשך כל הריצה ורק אם הוא מצביע על אובייקט הניתן לשינוי יהיה ניתן לשנותו וזאת בכפוף לכך שה-id לא ישתנה.
בגלל ש-tuples אינם ניתנים לשינוי כל שיטה (methods) או אופרטור הפועל על tuples יוצר/ת אובייקט חדש.

קבוצות (sets)

כמו ברשימות הפעלת שיטות (methods) על קבוצות אינה משנה את ה-id, אופרטורים יוצרים קבוצה חדשה (בעלת id שונה) והשמות מהצורות "=|", "=&", "=^" או "=-" מעדכנים את הקבוצה הקיימת אך אינם משנים את ה-id.
בגלל שהאיברים בקבוצות וכל האיברים של איבריהן וכו' אינם ניתנים לשינוי (כי הם hashable) נדע שערכם לא ישתנה לעולם וכל שיטה או אופרטור הפועלים על איבר בקבוצה יוצרים אובייקט חדש.

frozensets

בגלל ש-frozensets אינן ניתנות לשינוי ולא רק שהן כאלה אלא גם כל האיברים שלהן וכל האיברים של איבריהן וכו' אינם ניתנים לשינוי (כי הם hashable) נדע שכל שיטה או אופרטור הפועלים על frozenset או על איבר שלה יוצרים אובייקט חדש.

מילונים

בכל הנוגע לנושא שלנו מילונים מתנהגים בדיוק כמו רשימות, הרי ההבדל היחיד הוא שהמילון אינו מסודר בסדר כלשהו ושהמפתחות שלו אינם מוכרחים להיות דווקא מספרים שלמים.

containers

כדי להעתיק container כך שגם איבריו יהיו אובייקטים חדשים יש לייבא את הפונקציה 'deepcopy' מהספרייה 'copy' ולהפעיל אותה על הרצף הרצוי; כל העתקה אחרת[20], גם אם היא יוצרת container חדש, בסך הכל מכניסה ל-container החדש את כתובות ה-id של אותם אובייקטים.

ביטויים בוליאניים

ה-id של True ו-False אינו משתנה מריצה לריצה וה-id של כל ביטוי בוליאני הוא ה-id של הערך שלו[21] (True או False) ולכן ה-id של שני ביטויים בוליאניים ששניהם נכונים או ששניהם אינם נכונים -זהה.

טיפוסים (types)

הטיפוסים בפייתון הם כמובן אובייקטים מובנים וככאלה א"א לשנות את ה-id שלהם.

---

[20] אריה רמז בהרצאה על כך שיש דרך נוספת לעשות זאת אך עוד לא למדנו אותה.
[21] כלומר כל הביטויים הבוליאניים הם בעצם "תחפושת" ל-True או ל-False ממש כפי ש-1+2 הוא "תחפושת" של 3, קיימים בדיוק שני ביטויים בוליאניים.

iterators

בחלק מן התוכנות[22] שבהם כל עוד פייתון מסוגל לעשות זאת יהיה רק אובייקט אחד מטיפוס range:

```
>>> print(id(range(10)))
2470524285792
>>> print(id(range(5)))
2470524285792
>>> print(id(range(7)))
2470524285792
```

ואילו באחרות[23] לכל אובייקט מטיפוס range יהיה id משלו:

```
>>> print(id(range(10)))
2758443135408
>>> print(id(range(5)))
2758443128832
```

בכל המפרשים ניתן "להכריח" את פייתון להקצות שני מקומות בזיכרון עבור אובייקטים מטיפוס range ע"י שימוש במשתנים:

```
>>> a = range(3)
>>> b = range(3)
>>> a is b
False
```

בכל המפרשים iterators הנוצרים ע"י הפונקציה 'reversed' מקבלים id שונה בכל הפעלה של הפונקציה:

```
>>> a = [1, 2, 3]
>>> id(reversed(a))
1739076261904
>>> id(reversed(a))
1739076268480
```

[22] כגון המפרש של PyCharm ושל ה-IDLE Shell (התוכנה הלבנה שבאה עם ההתקנה הבסיסית של פייתון).
[23] כגון הפעלת פייתון בשורץ הפקודה.

שינוי אובייקטים (mutable & immutable)

רשימות (list), קבוצות (set ולא frozenset) ומילונים[24] (dict) הם אובייקטים הניתנים לשינוי (mutable), וכל אובייקט שאינו מאחד מהטיפוסים הנ"ל אינו ניתן לשינוי (immutable) מרגע שנוצר.
כשאנו אומרים שניתן לשנות אובייקט כוונתנו שניתן לשנות את *הערך* שלו[25], כשמדובר ב-container (כגון רשימות, קבוצות ומילונים) הערך שלו הוא האובייקטים שהוא מכיל[26]. לכן כשאמרנו לעיל שרשימות קבוצות ומילונים הם אובייקטים הניתנים לשינוי אמרנו בעצם שניתן להסיר אובייקטים מן ה-container, להחליף אובייקט באובייקט אחר (לאו דווקא כזה שכבר נמצא בו), להוסיף אובייקטים ושמדובר ברשימה גם לשנות את הסדר של האובייקטים בתוך הרשימה.

גיבוב (hashing)

הרעיון בפונקציות גיבוב (כגון 'hash' של פייתון) היא לאפשר השוואה מהירה של אובייקטים בגדלים חופשיים (כלומר ארוכים ככל שנרצה[27]) וזאת ע"י הקצאת פלט באורך קבוע לכל קלט (שלרוב ארוך בהרבה מהפלט), מסיבה זו ישנה הסתברות נמוכה ששני אובייקטים בעלי ערך שונה יקבלו מספר זהה ולכן פונקציות גיבוב[28] נמדדות בהסתברות לתת את אותו פלט לשני קלטים שונים (להרחבה ראו את הערך פונקציית גיבוב בויקיפדיה).
בפייתון קיימת פונקציית גיבוב מובנת בשם 'hash' המקבלת אובייקט יחיד ומחזירה עבורו מספר באורך קבוע[29] התלוי *בערך* של האובייקט (ולא ב-id שלו) כאשר ישנה סבירות גבוהה מאד שלא יהיו שני אובייקטים בעלי ערך שונה[30] שמספר ה-hash שלהם יהיה זהה. לא כל אובייקט בפייתון ניתן לגיבוב ע"י הפונקציה 'hash', על אובייקיטים הניתנים לגיבוב נאמר שהם hashable ובפסקה הבאה נראה באלו אובייקטים מדובר.

מה בין hashable ל-immutable

עבור אובייקטים שאינם containers מתקיים:

אובייקט הוא hashable אם"ם הוא immutable.

עבור containers השאלה אם אובייקט הוא hashable אם לאו תלויה בשאלה אם האובייקט עצמו ניתן לשינוי ואם כל האובייקטים שהוא מכיל הם ניתנים לשינוי[31], לפיכך:

- רשימות (list), קבוצות (set) ומילונים (dict) אינם hashable.
- מחרוזות הן hashable (גישה לאינדקס במחרוזת יוצרת מחרוזת חדשה ואינה משנה את הקיימת).
- frozensets הן hashable מפני שכל האיברים ב-frozenset מוכרחים להיות hashable מהגדרה.
- וכאן הגענו לחלק המעניין: כדי ש-tuple יהיה hashable על כל האיברים שהוא מכיל להיות hashable בעצמם[32], לכן אם למשל tuple מכיל רשימה כאחד מאיבריו אז הוא unhahsable.

לסיכום: אובייקט הוא hashable אם"ם א"א לשנותו בשום צורה שהיא, אפילו אם מדובר בצורה שאינה משנה את ה-id של איבר בו (אם הוא container) אך משנה את ה-id של איבר של איבר... של איבר שלו.

---

[24] אובייקטים מהטיפוסים dict_items, dict_keys ו-dict_values הם אובייקטים דינמיים: הם משתנים יחד עם שינוי המילון אליו הם משוייכים אך איננו יכולים לשנות אותם בעצמנו ולכן הם נחשבים immutable ולא זו אף זו אלא שהם גם hashable בניגוד למילון עצמו (בהמשך).
[25] א"א לשנות את ה-id של אובייקט או את ה-type שלו.
[26] ואם מדובר ברצף (sequence) אז הערך כולל גם את הסדר שבו האובייקטים מופיעים וכמות הופעותיהם.
[27] ניתן למשל להכניס את כל התנ"ך כמחרוזת יחידה.
[28] יש כמובן הרבה שיטות ליצור פונקציית גיבוב וגם פונקציית ה-'hash' של פייתון מתעדכנת מגרסה לגרסה.
[29] אצלי במחשב האורך הקבוע הוא 18.
[30] לעניין זה אין הבדל אם מספר הוא מטיפוס 'int' או מ-'float', הדבר היחיד שמשנה הוא הערך המספרי שלהם (מספר ה-hash של 1 וזה של 1.0 זהה).
[31] ואם חלק מהם גם הם containers אז השאלה מתגלגלת הלאה.
[32] אם זהו tuple ריק אז הוא hashable.