

- כיתבו פונקציה בשם beep שמקבלת מחזרות ומחזירה את המחזרות ועוד beep בסופה.
- כיתבו פונקציה בשם mul_2nums שמקבלת שני מספרים ומחזירה את המכפלה שלהם, או 0 אם התוצאה שלילית. שימו לב, אפשר לכתוב return יותר מפעם אחת בפונקציה.

פייתון מתחת למכסה המנוע (הרחבה)



בחלק זה נפרט שני נושאים הקשורים לפרקי הלימוד עד כה:

- נמחיש את עקרון התרגום של פייתון לשפת מכונה בזמן ריצה
- נכיר את הפונקציה ID ואת האופרטור is
- נחקור כיצד פרמטרים מועברים לפונקציות

כזכור פייתון היא שפת סקריפטים, ולכן לא מתבצעת קומפילציה לקוד. במילים אחרות, כל שורת קוד שאנחנו כותבים מומרת לשפת מכונה רק כאשר מגיע תורה של שורת הקוד להיות מורצת. נמחיש את העיקרון הזה בעזרת תוכנית קטנה.

```
def check(num1):
    if True:
        print("OK")
    else:
        bla(blalba)
```

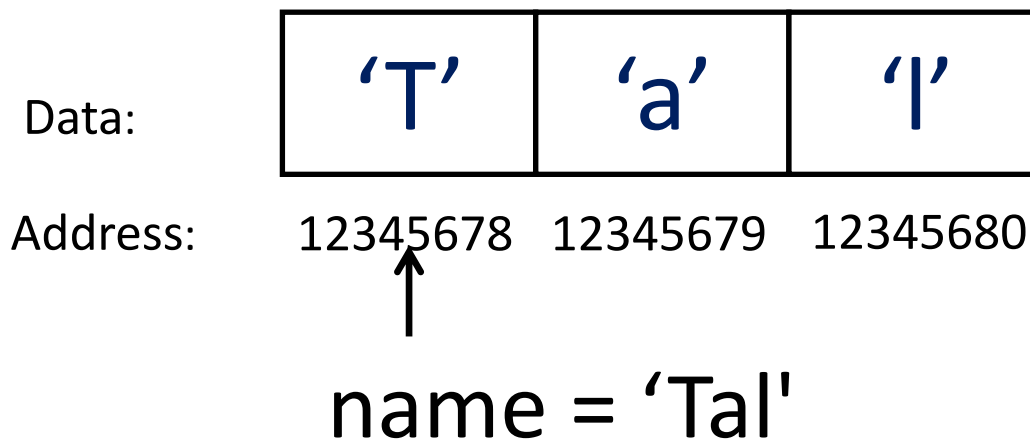
```
check(1)
```

כפי שאפשר לראות, הקוד מכיל קריאה לפונקציה בשם bla עם ארגומנט blabla. הן הפונקציה והן הארגומנט אינם מוגדרים. למרות זאת, אם נריץ את התוכנית נקבל תמיד 'OK'. הסיבה לכך היא שתנאי ה-if מתקיים תמיד, ולכן הקוד שבתוך תנאי ה-else לעולם אינו מגיע להיות מורץ. זוהי המחשה של העובדה שבפיתוח כל שורת קוד מפורשת ומתורגמת לשפת מכונה רק כאשר מגיע זמנה להיות מורצת.

בכך טמון גם סיכון: שורת קוד שנמצאת בתוך בלוק של תנאי שמתקיים לעתים נדירות עלולה להכיל שגיאות, שיובילו גם לקריסת התוכנית, והדבר לא יתגלה עד שהתנאי יתקיים.

id, is

זיכרון המחשב מכיל את כל המשתנים שמוגדרים בתוכנית שלנו. כל משתנה נמצא בכתובת מוגדרת, כלומר כאשר אנחנו מגדירים משתנה הוא מקבל כתובת מתוך טווח הכתובות שמוקצה לריצת התוכנית שלנו. עבור כל משתנה שמוגדר בתוכנית פיתוח, שם המשתנה משמש כדי להצביע על כתובת בזיכרון. לדוגמה, אנחנו יכולים להגדיר משתנה בשם name והוא יצביע על כתובת 12345678. בכתובת 12345678 יאוחסן הבית הראשון של המשתנה, אם המשתנה שלנו הוא בגודל של יותר מבית אחד אז יתר הבתים שלו מאוחסנים בכתובות העוקבות, לדוגמה 12345679, 12345680 וכך הלאה.



באמצעות הפונקציה `id` אנחנו יכולים לחשוף את הכתובת בזיכרון שהוקצתה למשתנה מסויים, או לפונקציה כלשהי. כן, גם לפונקציה יש כתובת בזיכרון – המעבד צריך לדעת לאיזה מקום בזיכרון לקפוץ כדי להריץ את הפונקציה.

דוגמה לשימוש ב-`id`:

```
>>> name = 'Tal'
>>> id(name)
50183144L
```

האות `L` בסוף הכתובת בזיכרון, מסמנת שזהו מספר מטיפוס `Long`, כלומר כזה המיוצג על ידי 64 ביט.

חישוב: האם לשני משתנים יכול להיות אותו `id`, ואם כן – מה הדבר אומר?

לשני משתנים יכול להיות אותו `id`, אבל רק אם הם מצביעים על אותה כתובת בזיכרון. נראה דוגמה:

```
>>> name_copy = name
>>> name_copy
'Tal'
>>> id(name_copy)
50183144L
```

הגדרנו משתנה בשם `name_copy` וקבענו שהוא שווה למשתנה `name`. בכך, גרמנו לו למעשה להצביע לאותה הכתובת בזיכרון שאליה מצביע המשתנה `name`. לאחר מכן, בדקנו שהערך של `name_copy` הוא גם כן 'Tal', כפי שהיה הערך של `name`. בשלב האחרון בדקנו מה ה-`id` של `name_copy`. כפי שניתן לראות, הוא זהה ל-`id` של `name`.

את האופרטור `is` הכרנו כאשר למדנו לכתוב תנאי `if` שונים. ראינו שאפשר לבדוק בעזרת `is` אם משתנה הוא `True` או לא. לדוגמה:

if result is True:

...

if results is not True:

...

כעת אנחנו יכולים להבין טוב יותר מהו `is`. האופרטור `is` בודק האם `id` של שני משתנים הוא זהה. אם כן – נקבל `True`, אחרת – `False`. שימו לב שגם ל-`True`, גם ל-`False` וגם ל-`None` יש `id`.

```
>>> id(True)
1516068552L
>>> id(False)
1516068136L
>>> id(None)
1516022872L
>>> a = True
>>> b = False
>>> c = None
>>> a is True
True
>>> b is False
True
>>> c is None
True
```

נסכם בכך שנפעיל את `is` על שני המשתנים שהגדרנו – `name` ו-`copy_name` – התוצאה היא `True` מכיוון שהם מצביעים על אותו מקום בזיכרון:

```
>>> name_copy is name
True
```

כדי להבין את ההבדל בין `is` לבין פעולת השוואה `==`, אתם מוזמנים לצפות בסרטון שבלינק הבא:

https://www.youtube.com/watch?v=0_dQpUtcubM

העברת פרמטרים לפונקציה

האם שאלתם את עצמכם איך פרמטרים מועברים לפונקציה? אם חשוב לכם לדעת איך הדברים עובדים, ומה זה `stack`, מומלץ לקרוא את ספר האסמבלי של גבהים. בקצרה, ישנן שני שיטות להעברת פרמטרים לפונקציה.

- Pass by value

- Pass by reference

בשיטת pass by value מועבר לפונקציה העתק של הפרמטר. ההעתק נמצא על אזור בזיכרון שנקרא מחסנית, או stack, ומשמש פונקציות. דמיינו שהמורה מחזיק דף נייר שכתוב עליו המספר 10. המורה ניגש למכונת הצילום ומכין לכל אחד מתלמידי הכיתה העתק של דף הנייר עם הספרה 10 עליו. לכל תלמיד יש העתק של המספר. אם אחד התלמידים יכתוב על הדף שלו 11 במקום המספר 10, הדף של המורה לא ישתנה. באופן זה, אם הפונקציה משנה את ערכו של משתנה שהועבר אליה בשיטת pass by value, היא למעשה לא משנה את ערך המשתנה עצמו, אלא העתק שלו. אי לכך, ביציאה מהפונקציה ערכו של המשתנה יהיה כפי שהיה לפני כן.

בשיטת pass by reference מועברת לפונקציה – גם כן דרך המחסנית – הכתובת בזיכרון שבו נמצא המשתנה. ערך המשתנה לא מועבר לפונקציה, ואם ברצונה לקרוא את הערך עליה לגשת לזיכרון בכתובת שנמסרה. דמיינו שכעת המורה שלנו מניח את הדף עם המספר 10 בתא שלו בחדר המורים, ובמקום לגשת למכונת הצילום הוא ניגש למנעולן ומשכפל לכל אחד מתלמידי הכיתה מפתח לתא שלו. המורה מחלק את המפתחות לתלמידים שלו. לאף תלמיד אין את הדף עם המספר 10, אבל הפעם התלמידים יכולים לגשת אל תא המורה, לקרוא את המספר 10 ואם הם רוצים – גם לשנות אותו. שינוי שיבצעו התלמידים ישפיע על הדף המקורי שבידי המורה. שימו לב שהפעם לא נוצרו לדף עותקים, יש רק עותק מקור.

אז מה קורה בשפת פייתון? האם פרמטרים מועברים בשיטת pass by value או בשיטת pass by reference? הבה ניצור פונקציה ונעביר לה פרמטרים:

```
def func(x):
    print("id(x): {} x: {}".format(id(x), x))
    x = "Bye"
    print("id(x): {} x: {}".format(id(x), x))
```

```
x = "Hi"
print(id(x))
func(x)
print(id(x))
```

אנחנו קובעים מחרוזת בשם x. מדפיסים את id(x) רק כדי שנוכל להשוות אותו לפני ואחרי הכניסה לפונקציה. לאחר מכן אנחנו קוראים לפונקציה ומעבירים לה את x כפרמטר. בתוך הפונקציה אנחנו בודקים את id(x), לאחר מכן משנים את ערכו של x ואז בודקים שוב את id(x). לאחר היציאה מהפונקציה, שוב בודקים את id(x) כדי לראות אם ה-id שהיה בתוך הפונקציה נשמר.

והנה מה שקיבלנו:

```

37058640
id(x) : 37058640  x: Hi
id(x) : 37058000  x: Bye
37058640

```

בשורת ההדפסה השניה אנחנו רואים שלפונקציה הועבר ה-id של x, שהרי ה-id מחוץ לפונקציה ובתוך הפונקציה הם זהים. האם זה אומר שהפרמטר הועבר by reference?

נראה כך, אבל אז מגיעה שורת ההדפסה השלישית, ואנו רואים שהעובדה שהפונקציה שינתה את x שינתה גם את id(x). במילים אחרות, נוצר בתוך הפונקציה משתנה חדש בשם x, שיש לו id שונה מאשר ה-id שנמסר לפונקציה. ואכן, בשורת ההדפסה הרביעית, שמתרחשת מחוץ לפונקציה, אנחנו רואים שערכו של id(x) חזר להיות כפי שהיה לפני הקריאה לפונקציה.

לסיכום: פייתון מעבירה לפונקציות את הכתובת של הפרמטרים, אך ברגע שפונקציה מנסה לשנות אותם נוצר העתק, כך שהערך של המשתנה המקורי לא ישתנה.

רגע לפני שנסכם, נציין שכל מה שכתבנו כעת נכון למשתנים מסוג מסויים, שנקרא immutable, שעד עכשיו עסקנו בהם בלי לקרוא להם כך. מהם משתנים מסוג immutable? ומהם משתנים מסוג mutable? על כך נרחיב כשנלמד על משתנים מסוג רשימה, list.

סיכום

בפרק זה למדנו אודות פונקציות בפייתון. למדנו להגדיר פונקציה, להעביר לה פרמטרים וגם לקבל ממנה ערכים. לאחר מכן ראינו שלמשתנים יש scope שבו הם מוגדרים, כלומר משתנה מוכר רק בפונקציה שבה הוא מוגדר. סקרנו אפשרויות שונות של שימוש במשתנים גלובליים בתוך פונקציה והגענו למסקנה שתמיד כדאי להעביר משתנים כפרמטרים לפונקציה, ובכל מקרה עדיף להמנע משינוי של משתנים בתוך פונקציה בלי שהקוד שקורא לפונקציה מודע לכך.

לאחר מכן, במסגרת ההרחבה, ראינו איך פייתון עובד "מתחת למכסה המנוע", כיצד מועברים פרמטרים לפונקציה. כעת אנחנו יכולים לכתוב פונקציות ולהשתמש בהם בצורה חופשית.