מג'אווה לפייתון- אופרטורים

אופרטורים הם פונקציות מיוחדות של שפות תכנות שמטרתן לשפר את קריאות התוכנית, את הדמיון בינה לבין טקסט מתמטי, לוקי או שפה טבעית, או כדי להדגיש משמעות של פעולה כלשהי.
את האופרטורים ניתן לחלק לארבעה קבוצות עיקריות - אריתמטיים, השמה, השוואה, ולוגים.
כמו כן יש קבוצות של אופרטורים מיוחדים כגון: אופרטורים של זהות או שייכות, אופרטורים של בסיסי נתונים ו-bitwise .

אופרטורים אריתמטיים-

אופרטורים אריתמטיים הם כל אותם אופרטורים שאנחנו משתמשים בהם להגדרת פעולות מתמטיות יסודיות.
בג'אווה יש את הפעולות המתמטיות הבסיסיות:
חיבור שמסומן ב- '+' , חיסור '-' , כפל '*' , חילוק '/' , ושארית חלוקה המוכרת בשם modulus '%'.
המתכנתים של ג'אווה גם הוסיפו כמה syntactic sugar (תחביר קצר ומובן יותר לפעולות שדורשות הרבה תווים)-
הוספת אחד לסכום '++', והפחתת אחד מהסכום '- -'.
בפייתון לעומת זאת הפעולות האריתמטיות היסודיות של השפה הן יותר מתקדמות, למשל ניתן למצוא חזקה שמסומנת ב-'**' , או ערך תחתון של חלוקה (מה שהיינו עושים לו קאסטינג ל-int בג'אווה) המסומן ב-'//', אך הsyntactic sugar של ג'אווה הוא לא חלק מהשפה.
סיכום הפעולות האריתמטיות של פייתון:

+	חיבור	x+y
-	חיסור	x-y
*	כפל	x*y
/	חילוק	x/y
%	Modulus	x%y
**	חזקה	x**y
//	ערך תחתון של חלוקה	x//y

אופרטורי השמה-

אופרטורים של השמה הם כל אותם האופרטורים שמכניסים ערך לתוך משתנה.
בתיאוריה קיים רק אופרטור אחד של השמה לג'אווה ופייתון והוא האופרטור '=',
אך עם הזמן פותחו עוד כמה syntactic sugars לשפות רבות שעוזרות לקצר תהליכים כגון += שהוא מקצר תחבירים של חיבור והשמה , למשל במקום לכתוב x= x+y , נוכל לכתוב x+=y , וכנ"ל לגבי כל אחד מהאופרטורים האריתמטיים(וה-bitwise).
גם בג'אווה וגם פייתון במקרה זה ניתן להשתמש ב-syntactic sugars הזה.

אופרטורי השוואה-

כל אותם האופרטורים שנועדו כדי לתת לנו אינדיקציה של גודל או סוג לערכים של המשתנים.
אנחנו משתמשים באופרטורי השוואה כדי למדוד האם אובייקט מסוים הוא גדול, קטן ,שווה ערך, או לא שווה ערך מאובייקט אחר, ומקבלים ערך "אמת" או "שקר" במידה והביטוי נכון.
בפייתון ובג'אווה האופרטורים זהים: '>' -הערך הימני גדול יותר, '<' -הערך השמאלי גדול יותר, '==' -שני הערכים שווים, ו- '=!' –הערכים אינם שווים.
בנוסף יש כמה syntactic sugars לשפה שהם שילוב של שני אופרטורים גדול/קטן ו-שווה: '=>' הערך הימני גדול **או** שווה, ועל אותה הדרך רק עם הערך השמאלי ב- '=<'.

אופרטורים לוגיים-

אופרטורים לוגיים הם כל אותם אופרטורים שמגדירים לנו נכונות בין ביטויים, כלומר הם מחזירים "אמת" אם ביטוי מסוים או כמה ביטויים נכונים, ו"שקר" אחרת.
שלא כמו האופרטורים הקודמים הסינטקס של האופרטור שונה, אך התוכן שלו זהה, למשל
אם יש לנו שני ביטויים(או יותר) ואנחנו רוצים לבדוק ששני הביטויים עם ערך אמת, בג'אווה נעשה את זה עם האופרטור "וגם" && ובפייתון ממש נכתוב and ,
אם נרצה לוודא שלפחות ביטוי אחד נכון, נשתמש באופרטור 'או' שבג'אווה מצוין כ-'||' ובפייתון ממש כותבים "or" ,
ואם נרצה לוודא שההפך של ביטוי הוא מה שקורה נשתמש באופרטור not שבג'אווה אנחנו מציינים אותו ב-'!' ובפייתון ממש כותבים not:

האופרטור	ג'אווה	פייתון
and	x<5 && y<7	x<5 and y<7
or	x<5 || y<7	x<5 or y<7
not	!(x<5 && y<7)	not(x<5 and y<7)

אופרטורי זהות ושייכות-

אופרטורי זהות-

אופרטורי זיהוי הם אופרטורים לבדיקה האם שני אובייקטים מצביעים לאותו מקום.
בג'אווה כשאנו בונים מחלקה חדשה ומגדירים אובייקט שמושם לו ערך המחלקה, לדוג' Person p = new Person() , אז p במקרה זה הוא לא אובייקט מסוג Person אלא מצביע לאובייקט מסוג Person.
יש לזה הרבה יתרונות, למשל במקום לשלוח לפונקציה פרמטר מטיפוס אובייקט ואז היא תעתיק אותו, כפי שהיא עושה במשתנים פרימיטיביים כמו int וכו' ,שמעתיקה אותם ומחזירה ערך אך לא משנה את הפרמטר שנשלח , נשלח לה מצביע למשתנה ואז השינוי יהיה בזיכרון מה שיחסוך מקום(העתקה של אובייקט כבד לוקחת זמן ומקום נוסף בזיכרון), והשינוי יהיה ניכר.
אבל יש לכך גם חסרונות למשל בשימוש באופרטור '==' על אובייקט מורכב תתבצע בדיקה על המצביע ולא על הערך שהוא מחזיק, מה שאומר שהבדיקה תהיה לפי המיקום בזיכרון של המצביע.
בפייתון לעומת זאת כל המשתנים הם מצביעים, וניתן להגדיר למחלקות אופרטורים כמו ב ++c, כפי שנראה בהמשך, לכן השימוש ב-'==' יכול להיות ממש לפי ערך ולא לפי מיקום בזיכרון, אבל כדי שלא תישלל האפשרות לבדוק מצביעים גם לפי המיקום שלהם בזיכרון יש את האופרטור is או האופרטור is not , כך למשל נוכל לבדוק את הדבר הבא:

```
>> x=3
>> z=x
>> y=z
>> y is x
      True
```

וכנ"ל נוכל לבדוק חוסר התאמה עם is not.
הערה: בבניית מחלקה חדשה, במידה ולא הגדרנו את האופרטור של המחלקה: '__eq__', אז האופרטור '==' יתפרש באותו אופן כמו האופרטור 'is'.

אופרטורי שייכות-

הם אופרטורים בלעדיים לפייתון שבודקים האם ערך מסוים נכלל בקבוצה כלשהי.
כדי לבצע את הבדיקה משתמשים במילה השמורה in או ב- not in כדי לבדוק חוסר שייכות
למשל:

```
>> primes_num_under_ten = [2 ,3 ,5 ,7]
>> 8 not in primes_num_under_ten
       True
```

אופרטורי BITWISE-

אופרטורי bitwise הם אופרטורים שפועלים על מספרים בינריים.
לפעמים נצטרך לבצע חישוב ברמת הביטים על משתנים, למשל בפרוטוקולי תקשורת לחישוב checksums וכדו', או באלגוריתמי דחיסה והצפנה.
האופרטורים בג'אווה ובפייתון כמעט זהים במקרה זה. הbitwise מבצעים פעולות לוגיות על ביטים- מחשבים, שמתקשרים בשפה בינארית, מחשיבים ערך כ"שקר" אם ערכו הוא 0 אחרת (1) ערכו אמת, או יותר מדויק אם יש זרם הערך 1, ואם אין זרם הערך הוא 0.
מכאן שפעולות שאנחנו מגדירים על ביטויים לוגים ניתן לבצע גם בצורה בינארית.
לפייתון וג'אווה יש שישה אופרטורי bitwise משותפים:
and – שדומה לסימון של 'וגם' בביטויים בוליאניים בג'אווה –'&' ,
ובדומה גם כאן ערכו אמת אמ"מ לשני הביטויים יש ערך 1, אחרת הערך שחוזר הוא 0, ואם מדברים על מספר המורכב מכמה ביטים, אז רק אם שני הביטים שבאותו המקום(באותו החזקה של 2) עם ערך 1, אז התוצאה תקבל ערך אחד באותו המקום, למשל: $5\&1 = 0101 \ \& \ 0001 = 0001 = 1$, כי רק במקום 2^0 לשני הביטים יש ערך 1.

or- שמסומן ב- ' | ', והוא פועל בצורה דו לאופרטור 'או' בביטויים בוליאניים, כלומר מקבל ערך 1 אם לפחות אחד משני הביטים באותו המקום עם הערך אחד: $5 \mid 1 = 0101 \mid 0001 = 0101 = 5$.

not- מסומן ב- '~' והוא מסמן שלילת הביטוי, כלומר כל מה שערכו אחד יהפוך להיות אפס וכנ"ל אפס יהפוך לאחד, למשל: $\sim 5 = \sim 0101 = 1010 = 10$.

xor- פעולה לוגית שמסומנת ב-^ ,והיא מגדירה שהערך הוא אחד רק אם **באחד** מהביטויים יש ערך אחד באותו המקום, כלומר בעוד ש-*or* מגדיר ערך גם אם שני הערכים הם 1 , *xor* מגדיר ערך אחד אם רק אחד משני הביטויים הוא אחד, למשל: $5 \ ^\wedge \ 1 = 0101 \ ^\wedge \ 0001 = 0100 = 4$.

פעולות נוספות ברמת ה-bitwise הן:
shift left- שמסומן ב-'<<', והוא משמש לדחוף את הביטוי ביט אחד שמאלה ע"י הוספה של המספר אפס לביט הכי ימני והוצאה של כל הביט השמאלי ביותר מהביטוי, למשל: $9 << 1 = 1001 << 1 = 0010 = 2$.
shift right- שמסומן ב-'>>' ,והוא מעתיק פעמיים את הביט השמאלי ביותר ומוריד את הביט הימני ביותר מהביטוי, למשל: $9 \gg 1 = 1001 \gg 1 = 1100 = 12$.

בכל הדוגמאות לעיל השתמשנו בדוגמא ב-*shift* לאחד אבל ניתן לעשות *shift* גם ל2 ואז המספרים ינועו שתיים ימינה או שמאלה וכו'.