מג'אווה לפייתון-בקרת זרימה

כברירת מחדל מחשב מבצע פקודות לפי סדר כתיבתן, אך לפעמים נרצה לשנות את אופן הפעלת הפקודות שיהיו יותר מבוקרות, למשל נרצה לבצע פעולה מסוימת מספר פעמים, או לבצע פעולה בתנאי שגם פעולה אחת התבצעה כראוי.
בג'אווה יש שני סוגים של בקרות זרימה: תנאים ולולאות, בפייתון בנוסף לשני הסוגים האלה יש גם פונקציות callback, אבל על כך בנושא נפרד.

תנאים-

בשפות תכנות פקודת תנאי שלרוב נקראת פקודת if היא פקודה שמתבצעת אם ורק אם מתקיים(או לא) תנאי מסוים.
בג'אווה הפקודה if היא כמו פונקציה שמקבלת ערך בוליאני ומבצעת את הקטע קוד המגיע לאחר מכן שסגור בבלוק ('{}'),או השורה שמופיע בדיוק לאחר הפקודה.
את הפקודה הרבה פעמים ניתן לסייג לכמה תנאים שמתחלקים לפונקציה else ולסוג של פונקציה מקוננת else if, else מתבצעת במקרה בו לא מתבצע התנאי ונרצה שיתבצע משהו אחר כברירת מחדל, ו-else if היא הוספת תנאי נוסף:

```
if (condition1)
{
  Statement1;
}
else if(condition2)
{
 statment2
}
else
{
  StatementDefault;
}
```

בפייתון הסינטקס דיי דומה.
if נכתב באופן דומה רק שהביטוי שלפיו אנחנו מבצעים את הבלוק מתקבל ישר(ולא כמו פרמטר לפונקציה), אחריו המבנה יהיה זהה למבנה של בלוק בפייתון- נקודתיים, ואז פקודה חדשה ברווח מתחילת ההתניה:

```
>> if 3 > 2:
. . print("3 > 2")
'3 > 2'
```

גם else של פייתון זהה לשל ג'אווה רק בסינטקס של פייתון, וה-else if נכתב כ- elif ומשם כמו ב-if רגיל:

```
>>> if a > 10:
...  print("a>10")
... elif a < 10:
...  print("a<10")
... else:
...  print("a==10")
```

יש גם syntactic sugar לכתיבת התניה מקוצרת של if ו- else בלבד:

```
>> a = 10 if a>10 else 0:
```

a יקבל את הערך 10 אם הוא גדול מעשר אחרת הוא יקבל 0.
בג'אווה יש גם אפשרות לביצוע התניה עם switch ו-cases , בפייתון **אין** אפשרות כזאת.
שאלה למחשבה: האם ניתן לממש התנית switch ו- cases בפייתון או לפחות למצוא לה תחליף ראוי?

לולאות-

לפעמים נרצה לבצע בלוק קוד מסוים כמה פעמים, וכמתכנתים טובים תמיד נשאף לבצע את הדרך הקלה והפשוטה ביותר שנוכל.
לולאה מאפשרת לבצע את אותו הקטע קוד בכמה איטרציות.
כמו כל דבר בחיים, או לפחות במסמכים האחרונים, גם לולאות מתחלקות לכמה סוגים או יותר נכון לשניים:
לולאות עם מספר איטרציות לא מוגדר- הלולאה תבצעה כל זמן שלא הוגדר אחרת.
ולולאות עם מספר סופי של איטרציות.

לולאות עם מספר איטרציות לא מוגדר מוכרות גם בשם לולאות while – כל זמן שלא שתקיים התנאי המסוים תבצע את קטע הקוד שמופיע מתחת.
גם לג'אווה וגם לפייתון האיטרציות פועלות בצורה דומה, והשוני הוא כמו השינוי בין if של השפות, כלומר בג'אווה נכתוב את שם הפקודה while ואח"כ בסוגרים את התנאי של הלולאה ולמטה בלוק עם הפעולה שאמורה להתבצע.
ובפייתון נכתוב while, התנאי (בלי סוגריים) ובלוק שיתבצע באיטרציות:

```
//while loop in java
while(x>2)
{
   x++;
}

#while loop in python
>>> while x>2:
...  x+=1
```

לולאת while תמיד מתקדמת לפי תנאי בוליאני ולכן נוכל לבצע ביטויים כגון:

```
>>> a = [1,2,3]
>>> while a:
...  print(a.pop(-1))
3
2
1
```

pop() היא פונקציה של רשימות, והיא מוציאה את האיבר האחרון מהרשימה, וכפי שכבר ראינו רשימה ריקה נחשבת כ-False.
בפייתון ובג'אווה ללולאות יש שתי מילות מפתח שמסיימות את האיטרציה של הלולאה(או את כל הלולאה) לפני הזמן, והן continue ו- break .
* continue- מסיימת את האיטרציה הנוכחית של הלולאה ,ומקפיצה יש לראש הלולאה מבלי לבצע את הפקודות הבאות לאחר פקודת continue.
* break- מסיימת את הלולאה לחלוטין.

```
>>> n = 5
>>> while n>0:
...  n-=1
...  if n==3:
...   continue
...  if n==1:
...   break
...  print(n)
4
2
```

בפייתון אפשר להשתמש ב-else אחרי לולאות while ,מה שמאפשר ביצוע של סדר פקודות שיבואו בתום הלולאה:

```
>>> num1 = 1
>>> num 2 = 0
>>> n = 5
>>> while n>5:
...  temp = num1
...  num1+=num2
...  num2=temp
...  #print(num2)
...  n-=1
... else:
...  num1=1
...  num2=0
>>> print(f"num1={num1},num2={num2}")
'num1=1,num2=0'
```

שאלה: מה היה מודפס לו לא היינו מסמנים את `print(num2)`?

מה בעצם מוסיף לנו ה"פיצ'ר" הזה? הרי בכל מקרה אחרי הלולאה היו מתבצעים הפקודות בשורות הבאות אחריה.
אז זהו שהצהרת ה`else` מאפשרת לבצע את הפעולות המוגדרות בבלוק שלה רק אם הלולאה נגמרה בצורה טבעית,
כלומר אם התבצעה `break` במהלך הלולאה, המפרש לא יבצע את השורות של הפקודה `else`:

```
>>> n = 5
>>> while n>5:
...  n-=1
...  if n==2:
...   break
... else:
...  n=10
>>> print(n)
2
```

שאלה למחשבה: בפייתון אין לולאת `do while` , האם ניתן לממש בצורה יעילה לולאה כזאת או לפחות שיטה לסמלץ אותה?

לולאות עם מספר סופי של איטרציות- מוכרות גם בשם לולאות `for`.
בג'אווה יש שני סוגים של לולאות `for`, הראשונה היא לולאה שבנויה משלושה חלקים: חלק אחד נקודות התחלה ,חלק שני תנאי להתקדמות, חלק שלישי התקדמות בכל איטרציה:

```
for(int i=0; i!=arr.length; ++i)
{
//code..
}
```

סוג שני של לולאות הוא כמין `syntactic sugar` של הסוג הראשון, והוא לולאת `for each` , שעוברת באיטרציה על כל משתנה מתוך אובייקט שבנוי איטרציות (ומוכר גם כמשתנה `iterable`), בדר"כ זה אוספים כמו מערך, סט וכו'.
הלולאה בנויה משני חלקים: הגדרת שם לכל משתנה מתוך האוסף, נקודותיים ושם האוסף:

```
for(int elem: arr){//...
```

פייתון שונה מג'אווה במקרה זה.
בפייתון כל לולאת `for` היא לולאת `for each` בעצם,
מה הכוונה? כל פעם שאנחנו מבצעים איטרציה המוגבלת במספר מסוים, צריך להגדיר מתוך איזה אוסף, או יותר מדויק

משתנה iterable, אנחנו מבצעים את הלולאה, למשל כשרוצים לעבור על כל איבר ברשימה אנחנו משתמשים באופרטור in כדי להכריז על הגדרה של הלולאה:

```
>>> arr = [1,'2',(3,4)]
>>> for elem in arr:
...  print(elem)
1
'2'
(3,4)
```

אם כך איך ניתן לבצע לולאה קלאסית כמו בג'אווה?
אז זהו, כדי ליצור לולאה של טווח מסויים נצטרך 'ליצור' אובייקט iterable של אותו טווח.
עכשיו לכאורה יכולנו לחשוב, רגע אז כל פעם שנרצה ליצור אובייקט של טווח מסוים נצטרך להגדיר רשימה שאיבריה הם המספרים של הטווח, זה לא הגיוני! עדיף כבר להשתמש בלולאת while עם משתנה חיצוני שעולה באחד כל פעם!
ישנה דרך נוחה, אפילו יותר מלולאות בג'אווה: לפייתון יש פונקציה שנקראת range() שמחזירה אובייקט שניתן לעבור עליו באיטרציה (לא אוסף), של מספרים מסוים.
כברירת מחדל חייב לשלוח לפונקציה ארגומנט אחד של הטווח(נקודת עצירה), אך ניתן להוסיף לו גם ארגומנט של נקודת התחלה, וגם משתנה של קפיצות בין איטרציה לאיטרציה, למשל נניח שאנחנו רוצים לקפוץ שלושה מספרים כל איטרציה במקום קפיצות של מספר אחד.
סדר הפרמטרים בפונקציה הוא קודם כל נקודת ההתחלה, הטווח וקפיצות:

```
>>> for n in range(10):      #n=0 -> n=9
...  #...
>>> for n in range(2,10):    #n=2 -> n=9
...  #...
>>> for n in range(2,10,2): #n=2-> n=8 iff n%2==0
...  #...
```

ניתן גם להציג טווח הפוך:

```
>>> for n in range(10,1,-1):
...  # n=10 -> n=1
```

גם בלולאת for ניתן להשתמש ב-break,continue וגם לה יש else בדיוק כמו בלולאת while.