מדריך לוגיקה

תוכן עניינים

מבוא

במדריך זה, ישנם סרטוני וידאו מתוך ערוץ היוטיוב 'עולם ההייטק' וכן גם קטעי קריאה עם הסברים על מושגים בעולם התכנות. המושגים הינם אוניברסאליים , כלומר - מתאימים לכל שפות התכנות ולאו דווקא רק לשפה אחת ספציפית.

כלומר, כל הרוצה להבין נושא מסוים (קרי מודולו) יכול להבין זאת מהמדריך הזה, ללא כל קשר באיזו שפה הוא מתכנת או כותב. יכול להיות מפתח java ויכול להיות מפתח c ששניהם יבינו מה זה מודולו דרך המדריך.

מבחינתי, הדבר האידיאלי הוא כן להיאחז באיזושהי שפת קוד כלשהי. נניח Java, או java scriptשהן יחסית קלילות להתחלה ויש עליהן מדריכים מלאים כאן באתר. אולם, יש סטודנטים שלומדים במוסדות אקדמאיים וצריכים להכיר את המושגים כפי שהם בהתאמה לכל שפה שהיא. המדריך הזה הוא המקום התיאורטי בעיקר.

כבר בסרטון הראשון שנמצא כאן, אסביר לכם על מה זה בכלל 'תכנות' ואיך המחשב חושב? האם הוא חושב בכלל?

מזמין אתכם לצפות בסרטונים ולקרוא את הכיתוב הנלווה לכל סרטון. שיהיה בהצלחה.

קלט ופלט

בשיעור הזה נלמד איך כותבים אלגוריתם על דף בצורה מסודרת בטכניקה שפיתחנו כאן בעולם ההייטק.

בנוסף לכך, מזמין אתכם לצפות בסרטון הקצר ולהבין כיצד מבצעים קלט, פלט ומניפולציה. זאת בהמשך לשיעור הקודם בו הסברתי על מודל התופרת.

נקודה חשובה לקראת סיום:

תמיד נשמור כל ערך, כל פעולה מתמטית וכל הוראה בתוך משתנה. זאת מהסיבה הפשוטה - המחשב הוא רובוט והוא לא זוכר כלום ושום דבר (ראה ערך שיעור 1). אנו חייבים להאכיל אותו בכפית כל הזמן ולודא שהוא אכן זוכר את הערכים שנקלטו. אולי בכלים הראשונים אין כל כך חשיבות ועולה השאלה 'למה אני צריך לשמור בכלל?' אולם בכלים ארוכים, באלגוריתמים מורכבים מאוד רצוי ואף חובה שהמחשב יזכור כל פרט וכל משתנה להמשך הדרך. לכן, כאשר

נאמר לכם 'קלוט מספר' יש ליצור ממש מגירה חדשה בתוך הארון הוירטואלי ולאחסן בו את המספר. לכל שפת תכנות יש את השיטה שלה לשמור.

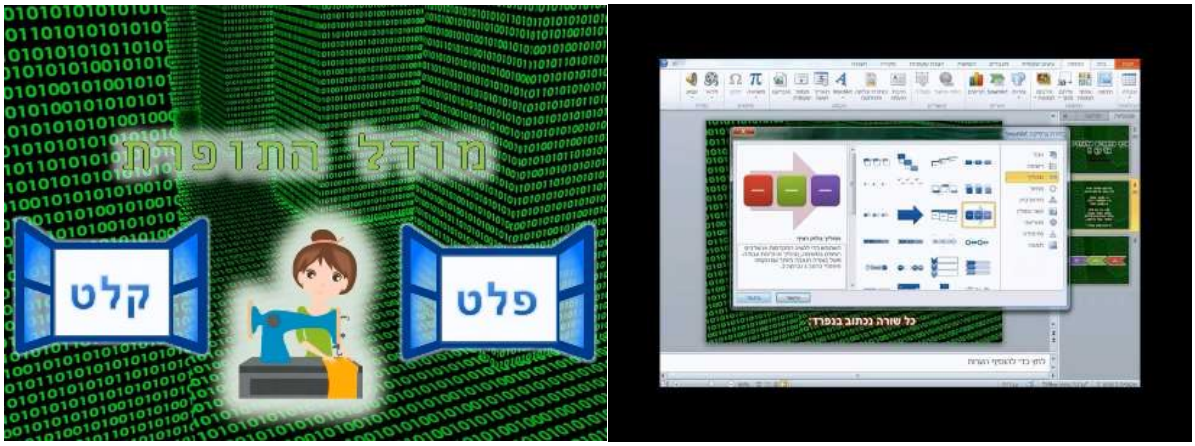
למגירה הזו קוראים גם 'משתנה.'

בסופו של דבר - הקלט שלנו יהיה מספר.

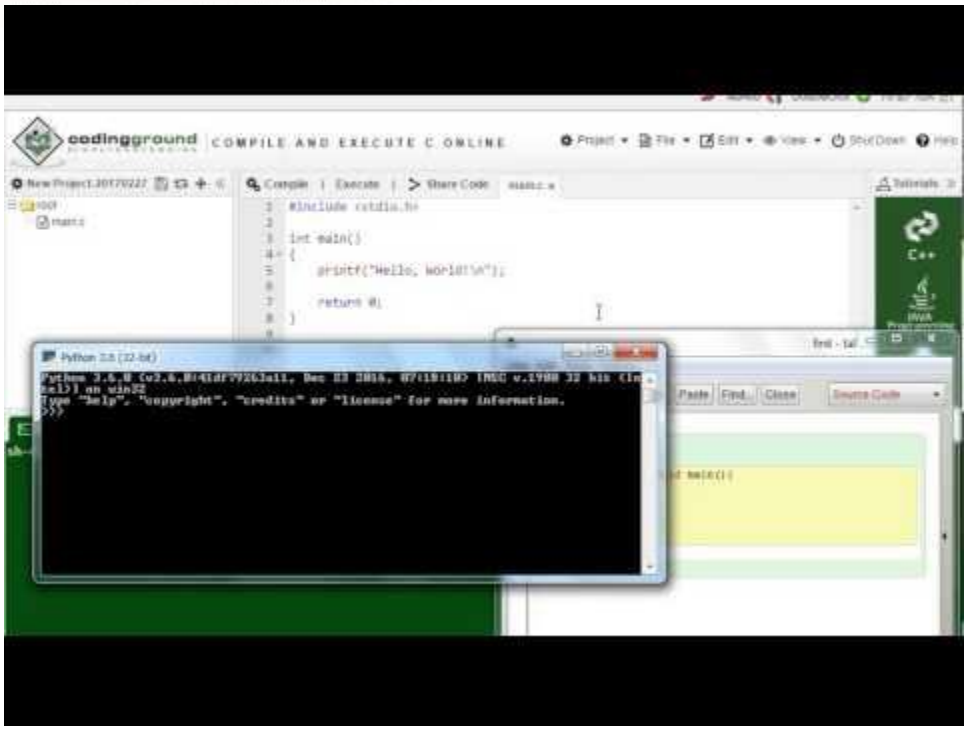
המניפולציה תהיה למשל : המספר כפול 2.

הפלט יהיה : 'המספר שיצא לך הוא: ' והמספר כפול 2.

כך למעשה יוצא מצב בו אנחנו יוצרים תוכנה, לא משנה איזה מספר המשתמש מזין בתוכה (גדול או קטן, שלילי או חיובי, לא משנה מה) תמיד זה יציג לנו בפלט את המספר כפול 2.



קודים



השמה

שיעור זה יתמקד ויתרכז במונח שנקרא 'השמה.'

למעשה, כאשר אנו רוצים לקרוא למשתנה מסוים שכבר קיים בזיכרון ולשנות את הערך שיש בתוכו נצטרך לקרוא לשם המשתנה ולאחר מכן לשים את הסימן שווה.

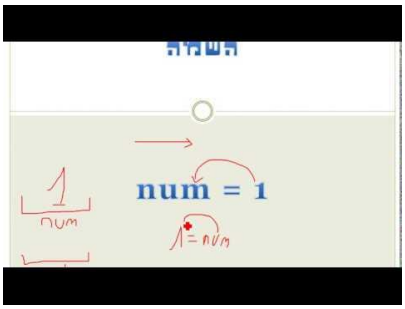
```
num = 5;
```

כאשר אנו רק קוראים למשתנה, שכבר קיים והמחשב מכיר אותו, אין צורך לרשום לפני המשתנה שום ערך. למשל, ב js זה ידוע שאנו רושמים var לפני יצירה של משתנה חדש או ב javaרושמים את סוג (טיפוס) המשתנה... למשל אם זה מספר אז נרשום לפני המילה num את המילה...int כמו שראינו, בדוגמא לעיל, לא מעניין את המפתח מה היה לפני כן ב .numהוא כעת משנה אותו למספר אחר לחלוטין. כביכול, מה שהיה בתוך המגירה הוא זורק ומשמיד ומכניס משהו חדש ואחר. לשיטה הזו קוראים גם 'דריסה'. לקראתי משהו ישן ודרסתי עליו משהו חדש.

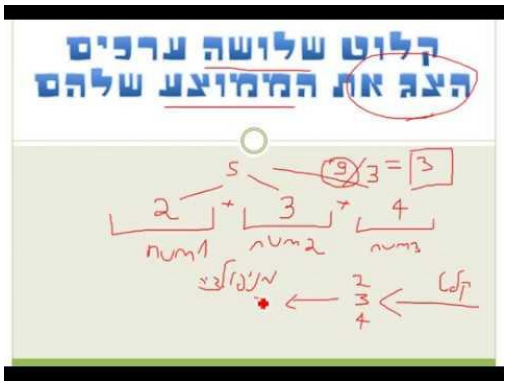
על מנת גם להתייחס לתוכן הישן של המספר ולבצע פעולה על התוכן הישן (למשל המספר הישן ועוד 1). ניתן לכתוב זאת כך: `num = num + 1;`

כלומר , קח את המספר הישן ותכניס אליו את המספר הישן ועוד 1. וזה עובד.

כל אלה נקראים 'השמה'. אני מזין ערכים לתוך משתנה.



כלים בנושאים בסיסיים



מודולו

מודולו זוהי שיטה מתמטית (ולאו דווקא בתכנות) אשר עונה על השאלה 'כמה ספרות נשארו לי עד המספר אחרי החלוקה'?

אסביר זאת דרך שני מספרים לדוגמא.

החישוב של 15/5 יוצא בדיוק 3. ללא כל שארית. לכן, התשובה לשאלה 'כמה ספרות נשארו לי עד 15 אחרי החלוקה?' היא 0. שהרי 3+3+3 בול 15.

הנה למשל תרגיל שכן יש פה שארית. למשל 16/5 נותן לנו שארית. התוצאה היא 3.2.

בחישוב מהיר נעשה 5+5+5 נגיע ל15. אם נוסיף עוד 5 זה כבר 20 ורצינו להגיע ל16.

לכן, מה שנשאר מ15 ועד 16 זה בדיוק מספר 1. והתשובה לשאלה 'כמה ספרות נשארו לי עד 16 אחרי החלוקה? (15)' זה 1.

נסו בעצמיכם, צפו בסרטון, הבינו את הרעיון ולכו לגוגל.

בגוגל אם תרשמו 16%5 (סימן ההיכר של המודולו זה אחוז) ייצא לכם 1. נסו בעצמיכם ואתגרו את עצמיכם בכלים שונים. ישנם כלים גם בעולם ההייטק ערוץ היוטיוב.

אתגר: איך נצליח לאתר האם המספר שהמשתמש הזין הוא זוגי או אי זוגי? תחשבו על זה...

אתגר: איך נצליח להציג את הספרה הימנית של כל מספר באשר הוא? תחשבו על זה...

בדיקות

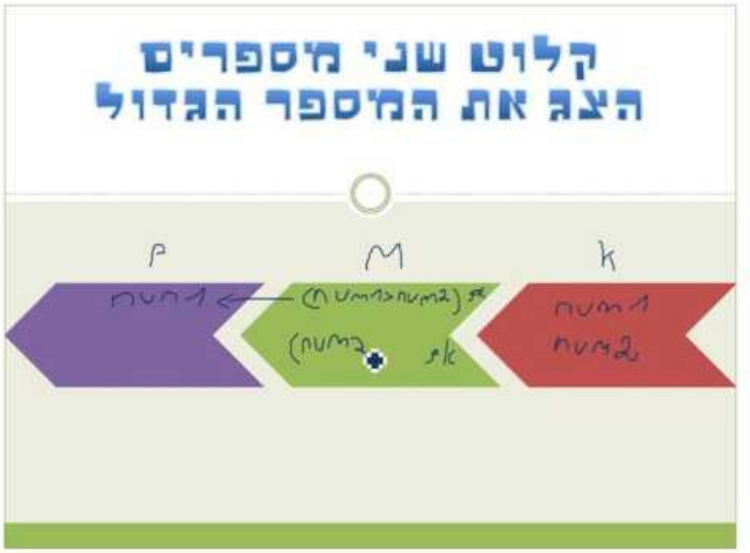
שיטת תכנות לביצוע בדיקה נקראת if .
למעשה, אנו רוצים לחצץ את הקוד שלנו ל2 חלקים. במידה והמספר חיובי - לך ימינה. במידה והמספר שלילי - לך שמאלה.

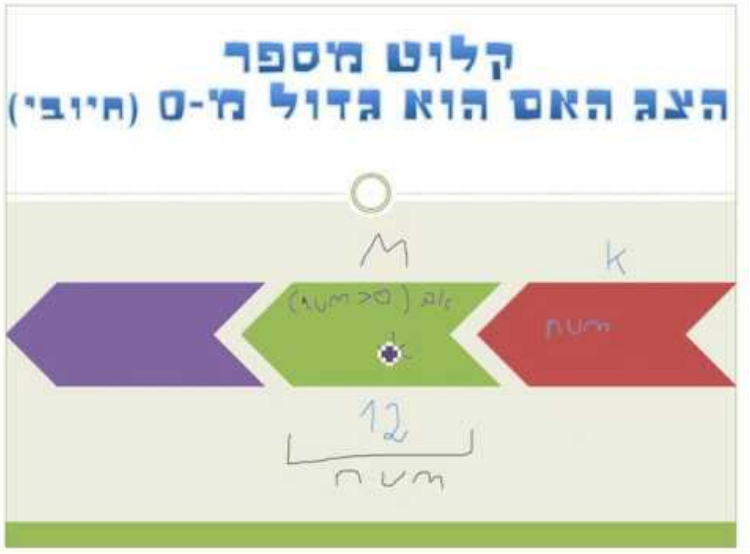
נוכל לעשות זאת על ידי שימוש באופרטורים של גדול מ - , קטן מ - או שווה ל-
גדול מ - נכתוב כך: `num > 5`
קטן מ - נכתוב כך: `num < 5`
שווה ל - הרי השווה הפשוט נותן לנו השמה בתכנות. לכן, נכתוב כך: `num == 5`

את הבדיקה עצמה תמיד נכתוב כך:

```
if (num > 0){
!');חיובי'alert(
}
else{
!');שלילי'alert(
}
```

פתרון תרגילי בדיקות





תנאי מורכב

על מנת ליצור אפשרות של בדיקה כפולה , למשל אם הגיל הוא גדול מ18 אך קטן מ99 , נוכל להשתמש בטכניקה של תנאי מורכב.
ניקח את התנאי הרגיל ונוסיף אליו את האופרטורים הבאים, בהתאמה לתרגיל שלנו:
כאשר נרצה לבצע את התנאי 'וגם&& '
כאשר נרצה לבצע את התנאי 'או|| '

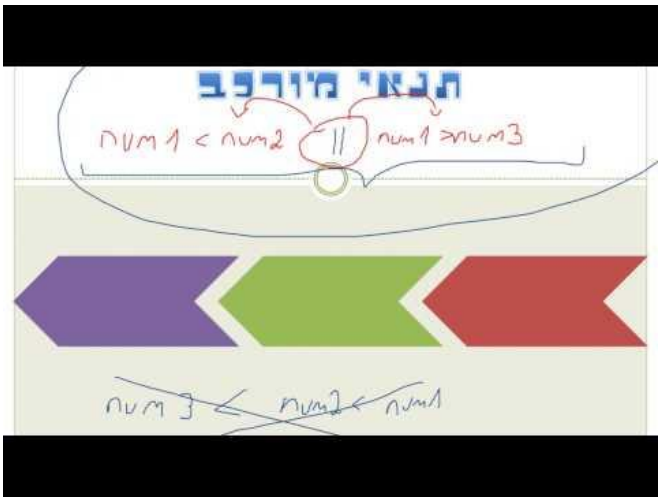
דוגמא לפתרון תרגיל: הגיל מעל 18 וגם קטן מ99:

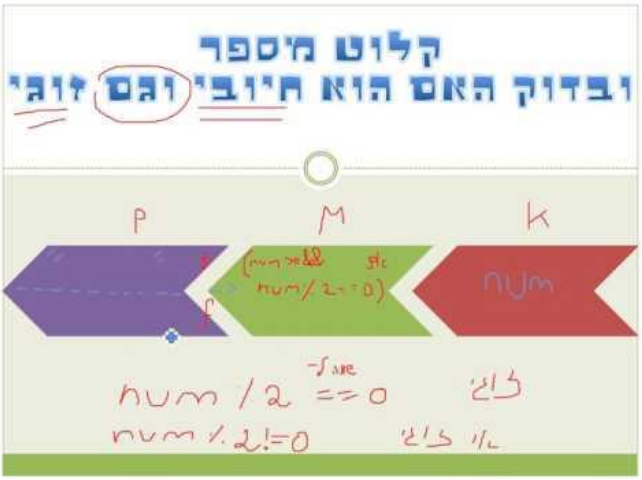
```
if (num > 18 && num < 99){
!');תקיוalert('
}
else{
!');לא תקיוalert('
}
```

דוגמא לפתרון תרגיל: הגיל מעל 18 או קטן מ99:

```
if (num > 18 || num < 99){
!');תקיוalert('
}
else{
!');לא תקיוalert('
}
```

מזמין אתכם להביט גם על סרטוני הפרק הזה.
תרגיל אחד עוסק בתנאי מורכב כשלעצמו הסבר ולמידה והשני בפתרון תרגיל.





לולאת while

בשיעור זה אסביר על טכניקת התכנות בשם לולאה. אסביר תחילה על לולאת while.

כאשר אנו רוצים לקרוא לקטע קוד מסוים שוב ושוב, נוכל להשתמש בלולאה, במקום לכתוב את אותה השורה 20 פעם.
למשל, נרצה להציג את המילה 'welcome!' לצורך הדוגמא 5 פעמים. אז או שנרשום `alert('welcome!');` חמש פעמים או שנשתמש בלולאה. לולאה עדיפה כמובן, אין מה לכתוב סתם שורות קוד ועדיף ליעל ולחסוך. זאת כדי להקל על הזכרון של התוכנה שלנו ולהגיע למהירות מירבית בלי באגים ותקלות בהמשך הדרך.

את הלולאה נכתוב כך:

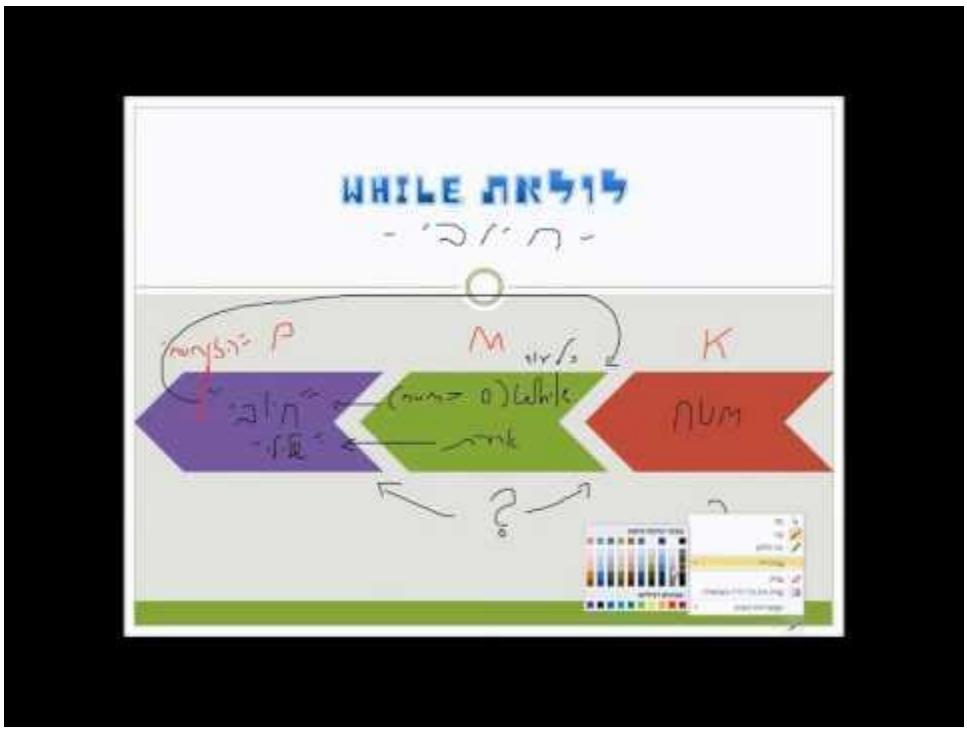
```
while (num>0){
!');תקיlalert('
}
```

1. בניגוד ל if, הפעם אין else. פשוט כל הקוד שיבוא בהמשך יתקיים וירוץ ברגע שהלולאה תסתיים לרוץ.

2. כרגע, אנו תקועים בלולאה אינסופית. הכוונה היא - שאנחנו לא יוצאים ממנה... תביטו בעצמכם, הרובוט נכנס פנימה) בהנחה ש num גדול מ0. למשל (num=5 ולא יוצא... הוא תמיד שווה ל5. הוא יציג alert של 'תקין!' אינסוף פעמים.
על מנת להתמודד עם לולאות אינסופיות עלינו להוסיף משתנה בתוך הלולאה תמיד שישנה את התנאי. למשל בואו נגדיר ש num=num-1 ואז גם אם המספר גדול מ0 הוא מתישהו יגיע למינוס.

```
while (num>0){
!');תקיlalert('
num = num - 1;
}
```

ב java למשל במקום לכתוב num=num-1 ניתן לקלוט מחדש משתנה לתוך num.
בנוסף, במקום num=num-1 או num=num+1 ניתן לכתוב --num או ++num בסירוגין.



לולאת for

טכניקת לולאה נוספת בתכנות היא גם לולאת הfor.

אם ב while ראינו כי אפשר להכניס ממש בדיקה שחוזרת על עצמה שוב ושוב, כאן דרך לולאת הfor נבדוק טווח מסוים. לצורך הדוגמא, אני מעוניין לייצר 4 גלגלי רכב דרך תוכנת מחשב. לכן, ידוע לי שאותו קטע קוד (ליצור הגלגל) ירוץ על עצמו 4 פעמים בצורה מדויקת אחד לאחד.

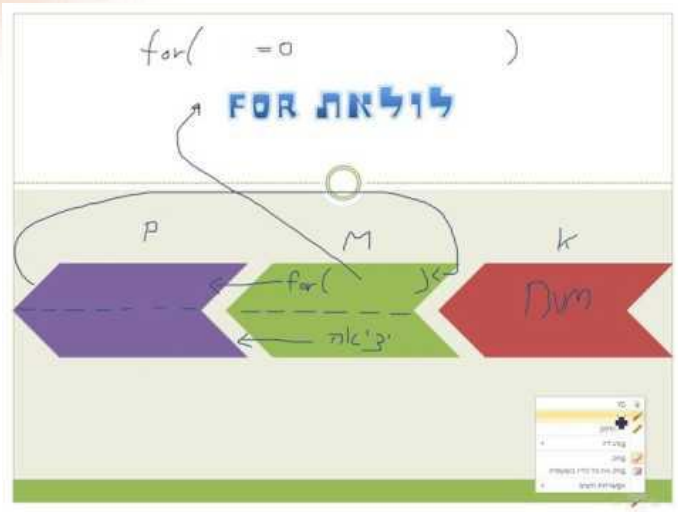
ניתן לחסוך לנו זמן ושורות קוד ולכתוב את הטווח בתוך לולאת ה for בדרך הבאה:

1. בתוך לולאת הfor ליצור משתנה שמציין את נקודת ההתחלה. (למשל, מתחילים לרוץ מהמספר 1).
2. בתוך הלולאה, ניצור גם תנאי. אם המספר שיצרנו, קטן מ10 למשל.
3. בתוך הלולאה, ניצור גם הוספה למספר כל פעם מחדש. למשל, המספר פלוס 1. כך נדע שבפעם הראשונה המספר אכן היה 1 , אך בפעם השניה הוא כבר 2 ואז 3 ... עד 10 ואז הוא ייצא מהלולאה. לפי התנאי.

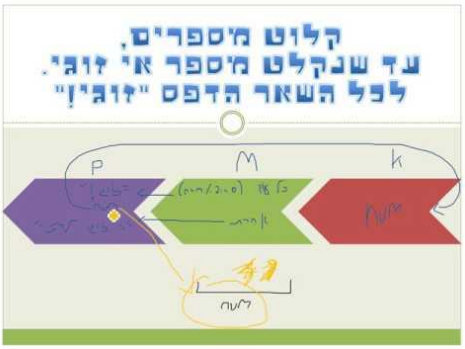
את כל שלושת הנקודות הללו ניתן לכתוב בתוך לולאת ה forבתוך הסוגריים בצורה מסודרת. שימו לב, כך נראית לולאת הfor:

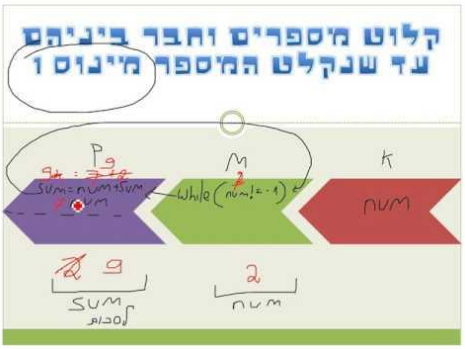
```
for (i = 1; i<10; i++){
alert('אני בלולאה');!
}
```

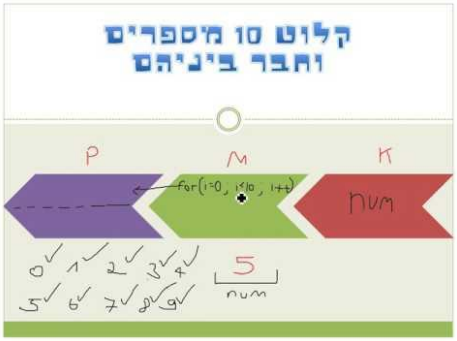
מדריך לוגיקה מאתר "עולם ההייטק" המקורי ©



פיתרון תרגילי לולאות







פונקציות

טכניקת תכנות נוספת שעוזרת לנו לייעל את הקוד שלנו ולגרום לו להיות נוח ונגיש יותר, היא פונקציות.

הכוונה היא, לעטוף כל אלגוריתם שאנו כותבים בתוך איזשהי פונקציה צדדית ומתוך הפונקציה הראשית שלנו לקרוא לאותן פונקציות צדדיות בעת הצורך. הדבר יגרום לכך שיש לנו פונקציה ראשית אחת שרצה כל הזמן וכל פעם היא קוראת לקטעי קוד אחרים.

הדבר יעיל יותר ונוח הרבה יותר. הקוד שלהם ייראה נקי יותר ובמידה ואתם תעבדו בעתיד בצוות של מתכנתים יהיה קל יותר ונוח יותר להבין זה את זה.

לצורך העניין בקטע קוד מסוים יש לנו פונקציה ראשית:

```
function main(){
var num1 = 3; var num2 = 4; alert(3*4);
}
```

אנו רוצים מתוך הפונקציה הזו לקרוא לפונקציה אחרת שתבצע פעולת חישוב.

שימו לב, אנו יכולים לשלוח פרמטרים בתוך הסוגריים של הפונקציות. כמו בדוגמא הבאה:

```
function cal(x,y){
var multi = x*y;
alert(multi);
}
```

על מנת לשדרג עוד קצת את הקוד, לא חובה להציג alert בתוך פונקציות צדדיות אלא רק בתוך הראשית. זה יגרום לקוד להיראות נקי ונוח יותר. כמו בדוגמא הבאה בעזרת return, למעשה הפונקציה רק תחזיר מספר ולא תציג אותו.

```
function cal(x,y){
var multi = x*y;
return multi;
}
```

לסיום, כך ייראה הקוד המלא עם הקריאה בין הפונקציות:

אני שולח מתוך main קריאה ל cal עם שני המספרים שלנו. ב cal יש שני משתנים x,y שמוכנים לקבל את כל מי שמגיע (במקרה זה מגיעים num1,num2) ועושים עליהם את המניפולציה. בסוף, דרך return מחזירים את המספר שקיבלנו ובכך var sum מקבל את התוצאה.

```
function main(){
var num1 = 3;
var num2 = 4;
var sum = cal(num1,num2);
alert(sum);
}

function cal(x,y){
var multi = x*y;
return multi;
}
```

מערכים

כאשר יהיה לנו מצב של רשימה כלשהי, נוכל ליצור את הפריטים ממש כמו רשימה דינאמית...

למשל, רשימה של פירות...
במקום לכתוב כך:

```
var fruit1 = “banana”;
var fruit2= “apple”
var fruit3= “watermelon”
```

נוכל ליצור אותם ברשימה אחת...
כאשר בזיכרון יהיה לנו רק משתנה אחד בשם fruit ונחלק אותו לתאים...

את המספר של התא נשים בתוך סוגריים מרובעים מיד אחרי השם של המשתנה.
תמיד נתחיל את התא הראשון מ0 ולא מ1...
כלומר–

```
var fruit[0] = “banana”;
var fruit[1] = “apple”;
var fruit[2] = “watermelon”;
```

או לחלופין גם כך:

```
var fruit = [“banana” , “apple” , “watermelon]
```

לסיום, מערכים זו איזשהי טכניקת תכנות אשר בעזרתה ניתן לחלק את המגירה שלנו בזיכרון לתאים ולשלוף בכל פעם מחדש את התא הרלוונטי.
זה גורם ליעילות במשתנים של הזיכרון שלנו...אנו לא חייבים להגדיר מיליון משתנים לכל האלגוריתם. אפשר גם מספר מצומצם יותר ולהיעזר במערך המחלק את המשתנה לתאים. זיכרו שתמיד הספירה מתחילה מ0 במערכים.