**מאיפה ומה לומדים**

Js זאת שפה הנחשבת לhigh level משמע אפשר לעשות בעזרת הכל ויש לה המון פיצ'רים שייחודיים רק לשפה וכמובן את הפונקציונליות הבסיסית שיש בכל שפה. מטיפוסי נתונים לכלים לאיטרציה טובה יותר לפעולות וקונספטים שיש בשפה – על הרוב אנחנו הולכים לדבר בשיעורי ההמשך אבל חלק מזה יהיה גם אקסטרה ולא נתמקד בו במהלך הקורס.

שני אתרים ממש מעולים שיעזרו לנו להתקדם בחומר, יש בהם המון חומר ללמידה דוגמאות והסברים ממש פשוטים

הראשון **JavaScript info**– אתר מאוד ממקוד מחלק את הלמידה לבלוקים מאוד מובנים ויעבור אתכם שלב אחרי שלב

השני **JavaScript The right way** – אתר ממש מעולה שמפנה אתכם להמון מקורות חיצוניים שכתבו דברים על js פרדיגמות עיצוב הקוד וכו'

**טיפוסי נתונים**

* Number – מספר כלשהו גם int גם double גם float פשוט מספר...
* String - מחרוזות
* Boolean - בוליאני
* Array - מערך
* Null – לא הגדרתי לו ערך
* Undefined – לא אתחלתי את האובייקט, יש הבדל דק בין null לבין Undefined
* Function – שגרה, יכולת מדהימה של שפות סקריפטית ללא טיפוסים זה האפשרות לשמור פונקציות שלמות בתור משתנה
* Object – {} אובייקטים מורכבים נדבר עליהם בהמשך

**מחרוזות:**

בjs ניתן לייצג במספר דרכים - בעזרת גרש יחיד ('), גרשיים (") , ותו חדש שכנראה לא הכרתם עד היום backtick-quoted Strings (`) תו זה מייצג לנו טמפלייט כלומר נוכל להשתמש באופרטורים חדשים שכבר נראה דוגמה להם וליצור מחרוזות רגילות

"I'm a String" - גרשיים

'I'm a String' - גרש

` I'm a String ` - backtick quoted Strings

דוגמה מגניבה לשימוש בתו המיוחד שהרגע ראינו זה

`half of 100 is ${100 / 2}`

הביטוי המתמטי בתוך הדולר והסוגריים יחושב והמחרוזת שתוצר תהיה :

half of 100 is 50

מעבר לשימוש באופרטורים $ {} ניתן ממש לכתוב פסקאות עם ריווח ירידת שורות והמחרוזת תיווצר בדיוק איך שהיא ישבה בtemplate

` Look there is a

New Line`

והפלט אכן יהיה :

Look there is a

New Line

וממש כמו שראינו בjava ניתן לבצע פעולות חיבור בין מחרוזות "b"+"a" והתוצאה תהיה "ab"

**משתנים בשפה**

מגניב הכרנו את סוגי הטיפוסים אז בואו נדבר רגע על כל המשתנים שניתן להגדיר בשפה על מנת לשמור ערכים

יש לנו שלושה סוגים של משתנים

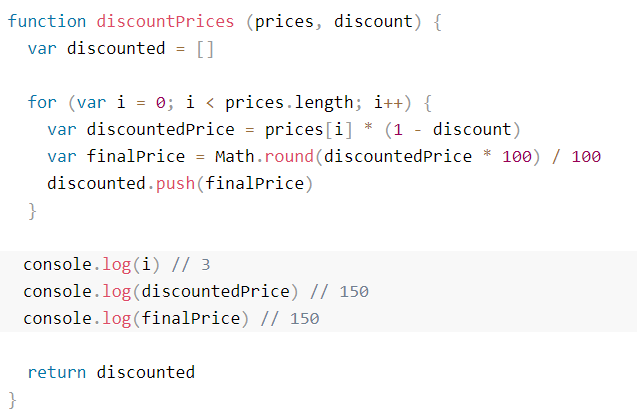
Const – סוג המשתנה קבוע, הטיפוס הנפוץ ביותר ברגע שאתחלתי את המשתנה בערך כלשהוא הוא יישאר קבוע (נקודה חשובה בנוגע לאובייקטים מורכבים – שניתן לשנות את הערכים הפנימיים, שווה להראות דוגמה לזה)

Let – משתנה בשפה מכיל ערך כלשהו מטיפוס כל שהו, אני לא חייב לציין לו את הטיפוס זה יכול להיות כל דבר .

Var –  כמו Let, אך יש שוני קטן

אז מה הקטע שלו? – ההבדל בין const ו-let לעומת var זה שטווח ההכרה שלהם זה הבלוק שבו הם מוגדרים לעומת var שטווח ההגדרה שלו זה בייחוס לפונקציה שבה הוא מוגדר

מה הכוונה שימו לב לדוגמה הבאה:

לפנינו שגרה שמקבלת שני פרמטרים ועושה עליהם איזה שהוא עיבוד שהעיקר שלו לא חשוב בכלל

מה כן חשוב בדוגמה שאתם רואים ניתן להשתמש בi, discountedPrice וfinalPrice למרות שהם מוגדרים בscope של הfor. איך זה קורה? בעזרת מנגנון הhoisting, js לפני הפירוש מגדירה את המשתנים האלה בscope של הפונקציה הראשונה שהם פוגשים (כלומר ממש מתחת לfunction discountPrices)

**מבני בקרה**

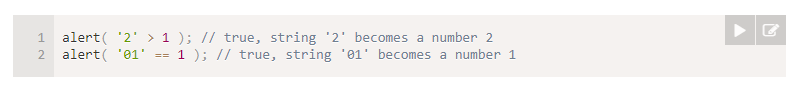
בjs יש לנו מספר מבני בקרה - לולאת while, for , do while מוזמנים לקרוא עוד בלינק הבא

* <https://javascript.info/while-for>
* <https://javascript.info/switch>

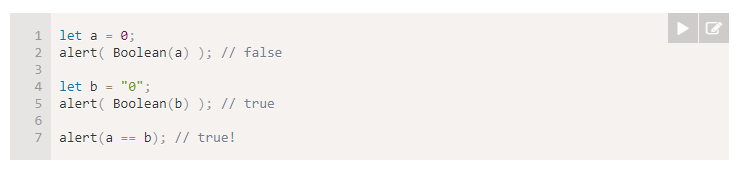
**פעולות השוואה בשפה + המרות**

אם עד עכשיו היינו מנסים להשוות בין מחרוזת למשתנה בוליאני כנראה שהייתה קופצת הערה שים לב ההשוואה לא אמורה לעבוד או שים לב אתה משווה בין טיפוסים שונים. אבל בjs זה לא קורה, למנוע יש מנגנון המרות שקורה באופן אוטומטי.

לפניכם דוגמה של המרה בין שני טיפוסים, המחרוזת מומרת למספר ולכן בשני המקרים מוחזר לי true.



אוקי אבל המפקד זה מרגיש לי מאוד אינטואיטיבי ההמרה פה – אז בואו נראה מה קורה בדוגמה הבאה



בשורה תחתונה ההמרות בjs מאוד מוזרות וקשה להכיר את כל החוקים ולכן ישבו וחשבו על פתרון, הפתרון שהוצא זה להשוות את הערך מבלי לעשות המרות

ולכן מצאו לטובת העניין אופרטור חדש '===' שלוש סימני שווה. בעצם מה שקורה מאחורי הקלעים זה שמתבצעת השוואה של הערכים מבלי שJs עושה המרות.

מוזמנים לקרוא [כאן](https://codeburst.io/javascript-double-equals-vs-triple-equals-61d4ce5a121a)

**פונקציות**

בjs יש לנו מספר דרכים ליצור פונקציות בקוד.

Declared Functions

הדרך הראשונה והפשוטה ביותר זה להשתמש במילה השמורה בשפה function, משהו שונה משפות סטטיות זה שאנחנו לא צריכים לציין אם הפונקציה מחזירה ערך או לא כל מה שעלינו לעשות זה לתת שם לפונקציה ואנחנו יכולים להשתמש בה.

רגע ומה קורה אם אנחנו כן רוצים להחזיר ערך?

התשובה לכך פשוטה - נשתמש במילה השמורה return, אם הפעולה הינה "void" כלומר לא השתמשנו בreturn בjs מוחזר ערך דיפולטי undefined

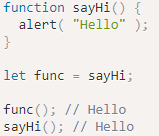
נסו להריץ את הקטע קוד הבא והוכיחו את הנקודה שלכם

function hey() {}

console.log(hey());

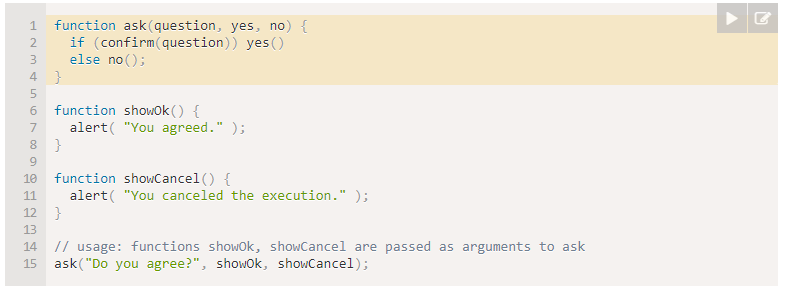
Function Expressions

כפי שכבר ראיתם משהו אופייני לשפות דינאמיות זה שפונקציה יכולה להיות פרמטר. כלומר אני יכול לשמור בתוך משתנה פעולה מסוימת



תחשבו על האפשרויות שניתן לעשות עם זה, בואו ניקח דוגמה:

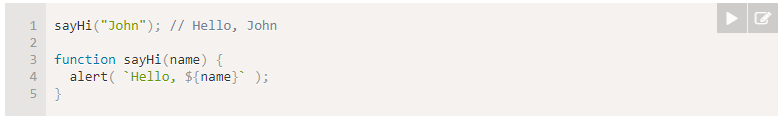
נניח אני רוצה לכתוב שגרה שמקבלת תשובה של חניך – אם התשובה היא נכונה אני ארצה להודיע לחניך שהוא צדק אחרת הוא טעה. נכון להיום אני משתמש בממשק קונסולי כדי להדפיס הודעות אבל אני יודע שבעתיד אני ארצה לכתוב מימוש אחר נכון לעכשיו רק מדפיס.



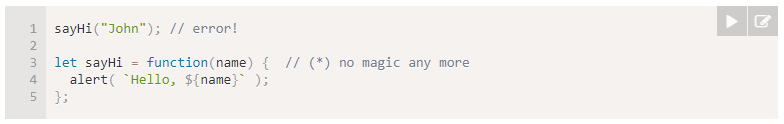
בדוגמה כאן אנחנו בעצם שולחים לשגרה ask שני מימושים של פעולות – פעולת ok וcancel במידה כזאת או אחרת אני אוכל בהמשך לרשום מימוש שונה של הפעולות. העברה של פעולה בתור משתנה שימושית בטירוף ואנחנו נראה לזה המון שימושים.

עובדה מעיינת לזכור לגבי ההבדל בין שני הסוגים, זוכרים את עקרון הhoisting – כאשר היינו מגדירים var לא משנה איפה בקוד מאחורי הקלעים המפרש של השפה יגדיר מחדש את אותו המשתנה ברמת הפונקציה הגובה ביותר?

אז על אותו עקרון  - קטע קוד כזה יעבוד



אך כזה לא!



זאת בעקבות אותו העיקרון הפונקציה מוגדרת מחדש ברמה הגבוהה ביותר בעוד שבדוגמה השניה היא מוגדרת במשתנה ואז יש כאן סתירה – יש קריאה למתודה לפני שהמשתנה הוגדר

**Arrow function – קיים החל מגרסת es6**

בjs קיים לנו סוג נוסף של פונקציות מאוד דומות בשימוש שלהן ל Function Expressions, מכיוון שהם לא מוצהרות ואני חייב לשמור אותן במשתנה על מנת להשתמש בהן. לסוג הזה יש שם נוסף בשם arrow functions בגלל הצורה שבה הן נכתבות

let func = (arg1, arg2, ...argN) => expression

הסינטקס מאוד דומה לlambda expression יש לי סוגריים עגולים שמציינים את הפרמטרים שאני רוצה לשלוח, לאחר מכן יבוא החץ (שימו לב משתמשים ב= ולא במקף) ולאחר מכן יבוא המימוש, במידה והפעולה קצרה אני לא חייב להשתמש בסוגריים מסולסלות והביטוי שנמצא לאחר החץ יחושב ויחזור למשתנה ברגע שאני אפעיל את המתודה.

