ללמוד ג׳אווהסקריפט בעברית

רן בר-זיק

WiXEngineering

Really Good

Outbrain Engineering

Chegg

Ono

הקריה האקדמית אונו
Ono Academic College
החוג למדעי המחשב

ללמוד ג'אווהסקריפט בעברית

רן בר-זיק

מהדורה: 2.2.1

כל הזכויות שמורות © רן בר-זיק, 2023.

ספר זה הוא יצירה המוגנת בזכויות יוצרים. אתה קיבלת רשיון לא-בלעדי, לא-ייחודי, אישי, בלתי ניתן להעברה (למעט על פי דין), ובלתי ניתן להסבה לעשות שימוש אישי בספר זה לצרכים לימודיים בלבד.

אסור לך להעתיק את הספר, לשכפל אותו, לצור יצירות נגזרות ממנו או לפרסם אותו בכל צורה אחרת.

מותר לך לצטט קטעים קצרים מהספר במסגרת הגנת שימוש הוגן, כלומר פסקה או שתיים, כאשר אתה מפנה למקור ומזכיר את רן בר-זיק כמחבר הספר.

הדוגמאות המובאות בספר זה הן בבעלות של רן בר-זיק, ואסור לך להשתמש בהן בתוך תוכנות שתפתח. אם אתה רוצה להכניס אותן לפרויקט שלך, שלח מייל ונדבר על זה.

עריכה לשונית: יעל ניר

הגהה: חנן קפלן

עיצוב הספר והכריכה: טל סולומון ורדי (tsv.co.il)

הפקה: כריכה – סוכנות לסופרים

www.kricha.co.il

תוכן העניינים

על הספר *12*
על המחבר *13*
על העורכים הטכניים *14*
דניאל שטרנליכט 14
גיל פינק 14
יגאל סטקלוב 14
תום ביגלאייזן 15
צחי נמני 15
על החברות התומכות *16*
Really Good 16
אאוטבריין 16
Chegg 17
Wix.com 18
על ג'אווהסקריפט *19*
איך לומדים *22*
ארגנו לעצמכם סביבת עבודה מסודרת 22
קראו את הפרקים לפי הסדר 22
קראו כל פרק פעמיים וכתבו לעצמכם את כל הדוגמאות 23
פתרו את התרגילים לדוגמה בסוף כל פרק 23
התייעצו עם אחרים 23
הגיעו לסדנאות ולמיטאפים 24
תרגלו, תרגלו, תרגלו 24
התקנת סביבת העבודה ודרך הלימוד *25*
משתנים *35*
טקסט *40*
מספרים *46*
מציאת השארית 49

אופרטורים מקוצרים 50

סוגי מידע פרימיטיביים נוספים 56

בוליאני..... 56

משתנה לא מוגדר..... 57

ריק..... 57

Symbol..... 58

מציאת הסוג של המשתנה..... 58

הערות 62

בקרת זרימה – משפטי תנאי..... 65

אופרטורים השוואתיים נוספים..... 68

אופרטורים לוגיים..... 72

אופרטור שלילי 75

אופרטור תנאי..... 77

אופרטורים המשווים ערכים..... 78

switch case 90

קבועים 100

בקרת זרימה – פונקציות..... 104

פונקציה עם ארגומנטים 109

ארגומנטים עם ערכים דיפולטיביים 111

הפונקציה כאובייקט 113

Hoisting..... 113

closure 115

פונקציה אנונימית ופונקציית חץ..... 118

פונקציה אנונימית במשתנה..... 119

פונקציה אנונימית שמבודדת מהסקופ הגלובלי..... 121

אובייקטים 129

מחיקת מפתח..... 135

הכנסת פונקציה בערך..... 135

קינון ובדיקת מפתח..... 136

אובייקט כקבוע..... 137

מערכים..... 146

מערכים ומחרוזות טקסט 151
new ו-this *154*
תבנית טקסט *162*
לולאות *167*
לולאת for 167
לולאה אינסופית 178
לולאת while 178
לולאת do while 180
לולאת forEach 180
לולאת for of 184
לולאת map 189
לולאת filter 192
לולאת sort 196
לולאות על אובייקטים 196
לולאות for in 197
Object.keys 200
ג'אווהסקריפט בסביבת דפדפן *209*
הסבר כללי על HTML 209
מזהים של תגיות HTML 212
גישה אל תגיות באמצעות ג'אווהסקריפט 213
אלמנטים של HTML מתורגמים לאובייקטים של ג'אווהסקריפט 213
אירועים עם HTML וג'אווהסקריפט 215
הצמדת אלמנטים של HTML לאלמנטים אחרים של HTML באמצעות ג'אווהסקריפט 217
שינוי עיצוב 219
סלקטורים של DOM 220
אירועים נוספים 225
פעפוע של אירועים 227
הצמדת אירועי ג'אווהסקריפט לאלמנטים ב-HTML 231
דיבאגינג *240*
אובייקטים גלובליים ואובייקטים מובנים *247*
parseInt 251

eval **252**
Math **253**
Date **254**
JSON **257**
setTimeout **259**
ביטויים רגולריים ***265***
טיפול בשגיאות **273**
finally **276**
מבני נתונים מסוג Map ו-Set ***280***
Map **280**
Set **283**
תכנות אסינכרוני – קולבקים ***288***
Promises ***298***
שרשור הבטחות **304**
קיבוץ הבטחות **307**
קיבוץ הבטחות מתקדם **308**
פונקציית async **313**
AJAX ***320***
מתודות של HTTP וארגומנטים נוספים **322**
ES6 Classes ***327***
מתודות ותכונות פרטיות 332
ומה עכשיו? ***336***
נספח: ללמוד טייפסקריפט בעברית ***338***
קביעת סוג הקלט של פונקציה 342
הגדרת סוג מידע של משתנה 345
הגדרת סוג מידע של פלט של פונקציה 345
מערכים 347
אובייקט עם סוגי מידע פרימיטיביים 349
סוג מידע: פונקציה 349
כמה סוגי מידע אפשריים 352
Type\ interface 354
קאסטינג 356
גנריות 357
טייפסקריפט עם קלאסים 364
מתודות פרטיות, מוגנות ופומביות 364
קריאה בלבד 365

קלאס אבסטרקטי 367
סביבת עבודה אמיתית של טייפסקריפט 369
סיכום 379
תרגילים 380

נספח: Best Practices 383

מה זה Best Practices ולמה כדאי ליישם אותם? 383

Best Practices שכל מפתח מקצועי צריך להכיר 385

בחירת שמות 385
KISS 387
DRY 388
לא להמציא את הגלגל 388
Make it work, make it right, make it fast 389
תיעוד 390
ניהול גרסאות 390
לבקש עזרה 391
ביקורת עמיתים – Code Review 392
Tech Design 393
Best Practices ואוטומציה 393
מה זה linting? 395
ESLint 396

רשימה של Best Practices והחוק הרלוונטי של ESLint 397

בלי מספרי קסם 397
השוואה קפדנית: === 398
קוד לא נגיש 398
חלוקה לחלקים קטנים 399
אורך מינימלי ומקסימלי לשמות משתנים 400
בלי eval 400
בלי משתנים שלא הוצהרו 401

נספח: בדיקות, יציבות ואיכות קוד 402

קצת רקע 402
מבנה בדיקות 403
בדיקות יחידה 404
בדיקות קצה לקצה (End-to-End) 405
בדיקות ממשקי משתמש (UI Tests) 406
ספריות ופריימוורקים מומלצים 407
סיכום 407

נספח: Velo by Wix 409

הקדמה 409

מה בונים? 409

איך מתחילים? 411

הצגת נתונים ממסד הנתונים באתר 414

Dataset 415

חיבור רכיבים למידע מהטבלה ... 416

הוספת משימה חדשה ... 419

Events ... 419

$w ... 421

wix-data ... 422

wixData.insert() – הוספת רשומה לטבלה ... 423

שינוי סטטוס המשימה ... 426

wixData.get() – שליפת רשומה מהטבלה ... 427

wixData.update() – עדכון רשומה בטבלה ... 428

מספר המשימות שלא הושלמו ... 430

wixDataQuery ... 430

$w.onReady() ... 434

סינון המשימות לפי סטטוס המשימה ... 434

wixDataFilter ... 436

ניקוי המשימות שהושלמו ... 439

wix-window ... 440

wixWindow.openLightBox() ... 441

wixWindow.lightbox.close() ... 442

wixData.bulkRemove() – מחיקת רשומה מהטבלה ... 443

תוספת – יצירת מסד נתונים ... 446

Sandbox | Live ... 449

Permissions ... 450

סיכום ... 451

נספח: ג'אווהסקריפט מונחה עצמים ... 452

מה זה תכנות מונחה עצמים? ... 452

Prototype based ... 460

נספח: שינויים מהמהדורה הקודמת (מהדורה 2.1.0) ... 467

על הספר

הספר "ללמוד ג'אווהסקריפט בעברית" מלמד את השפה ג'אווהסקריפט (JavaScript), שפת סקריפט קלה ופשוטה ללימוד שהפכה לפופולרית מאוד עם השנים. הספר מיועד ללא-מתכנתים שרוצים להתנסות בשפת תכנות ולמתכנתים בג'אווהסקריפט שרוצים להעמיק את הידע התיאורטי שלהם.

הספר מתחיל בלימוד בסיסי של משתנים ומגיע עד לימוד תכנות אסינכרוני ו-AJAX. בכל פרק בספר יש הסברים תיאורטיים לצד דוגמאות רבות הממחישות את העקרונות השונים שהוסברו בו, ובסופו תמצאו שאלות דוגמה לתרגול עם הסברים מקיפים על הפתרונות.

נוסף על הספר קיים אתר המכיל מאות תרגילים ופתרונות אשר יסייעו לכל הלומדים להשיג שליטה בעקרונות השפה ובתחביר שלה. הספר מיועד להביא אדם שלא מכיר את ג'אווהסקריפט לנקודה שבה הוא מכיר היטב את השפה ואת התחביר שלה ויודע איך להשתמש בה כדי לפתור בעיות בסיסיות.

הספר יסייע גם למתכנתים מנוסים יותר שמבקשים לחזק את הידע התיאורטי שלהם. יש בו הסברים על יכולות מתקדמות מאוד של השפה שהוחלו בשנת 2017, כמו מימוש טוב יותר של תכנות אסינכרוני, AJAX באמצעות fetch ותכונות נוספות.

בספר יש נספח מפורט עם תרגילים המלמד על טייפסקריפט, שבנויה על ג'אווהסקריפט ונעשה בה שימוש בסביבות מודרניות רבות.

על המחבר

רן בר-זיק הוא מפתח תוכנה משנת 1996 במגוון שפות ופלטפורמות ועובד כמפתח בכיר במרכזי פיתוח של חברות רב-לאומיות, מ-HPE ועד Verizon, שם הוא מפתח בטכניקות מתקדמות הן בצד הלקוח, הן בצד השרת, ושם דגש על בניית תשתית פיתוח נכונה, על שימוש ב-CI\CD וכמובן על אבטחת מידע.

נוסף על עבודתו כמפתח במשרה מלאה, רן הוא עיתונאי ב"דה מרקר" במדור המחשבים, שם הוא מסקר נושאים הקשורים לטכנולוגיה ולאבטחת מידע וכותב על אינטרנט ורשתות.

משנת 2008 מפעיל רן את האתר "אינטרנט ישראל" (internet-israel.com), שהוא אתר טכני המכיל מדריכים, מאמרים והסברים על תכנות בעברית ומתעדכן לפחות פעם בשבוע.

רן נשוי ליעל ואב לארבעה ילדים: עומרי, כפיר, דניאל ומיכל. רץ למרחקים ארוכים וחובב טולקין מושבע.

על העורכים הטכניים

דניאל שטרנליכט

דניאל שטרנליכט הוא מפתח Frontend מאז גיל 14, המייסד של המיזמים Common Ninja ו-There is a bot for that, צרכן כבד של קוד Open Source (ומשתדל גם לתרום בחזרה) ונכון לזמן כתיבת הספר מוביל את גילדת ה-Frontend בחברת אאוטבריין.
נוסף על כך, דניאל הקים ומוביל את קבוצת הווטסאפ FEDs Community שמאגדת מפתחי Frontend מחברות מובילות בארץ ובעולם, ובה מעלים דיונים, מתייעצים ומשתפים לינקים, חדשות ועדכונים מעולם ה-Frontend.
בעבר כתב בבלוג "עיצוב גרפי וטכנולוגיה", וכיום הוא כותב בלוג טכני על טכנולוגיות Frontend ועל NodeJS.
בשאר הזמן הוא נשוי לרונה ואבא להדר ולאייל המתוקים, מתופף, גולש סקי, משחק כדורסל, צופה בסדרות, קורא ספרי פנטזיה ומתח ומדקלם את כל סרטי דיסני בעל פה.

גיל פינק

גיל פינק הוא מומחה לפיתוח מערכות ווב, Google Developer Web Technologies Expert, Microsoft Developer Technologies MVP והמייסד של חברת sparXys.
כיום הוא מייעץ לחברות ולארגונים שונים, שם הוא מסייע בפיתוח פתרונות מבוססי אינטרנט ו-SPAs. הוא עורך הרצאות וסדנאות ליחידים ולחברות המעוניינים להתמחות בתשתיות, בארכיטקטורה ובפיתוח מערכות ווב. הוא גם מחבר של כמה קורסים רשמיים של מיקרוסופט (Microsoft Official Course - MOC), מחבר משותף של הספר "Pro Single Page Application Development" (Apress) ושותף בארגון הכנס הבינלאומי AngularUP. לפרטים נוספים על גיל: http://www.gilfink.net

יגאל סטקלוב

יגאל סטקלוב הוא מפתח Frontend ו-Full Stack, מוביל טכנולוגי ומנהל פיתוח מנוסה בעל ותק של יותר מעשור וחצי בתעשייה. כיום משמש מנכ"ל חברת Webiya.

בעבר היה ממובילי תחום ה-Frontend בחברות Wix ו-Netcraft והוביל פרויקטי פיתוח רבים.
יגאל פעיל מאוד למען קהילת ה-Frontend בארץ והוא אחד מהיזמים והמארגנים של כנס ה-Frontend הבינלאומי הראשון בישראל, כנס You Gotta Love Frontend (YGLF).

תום ביגלאייזן

מפתח Frontend ומעצב, נשוי, אב לשתיים וגר בתל אביב.
תום התחיל את דרכו בתחום ה-Frontend בתחילת שנות האלפיים מכיוון פחות צפוי – בפיתוח אפליקציות בג'אווהסקריפט לממירים של יס בסטארט-אפ קטן. אחרי שזה נסגר, עבד כעצמאי ובכמה סטארט-אפים, הפך לאחד המומחים הראשונים בארץ ל-HTML ו-CSS ולא היה בקיא ממנו בבאגים של CSS עם rtl באקספלורר 6. תרם לפרויקטים שונים בקוד פתוח, היה שותף בגיור פלטפורמת ניהול התוכן דרופל ובנה את אחד הטמפלטים העבריים הפופולריים באותה תקופה. תום גם עיצב את האייקונים שעדיין נמצאים בשימוש עד היום בנגן הווידיאו הפופולרי בקוד פתוח VLC. לפני קצת יותר משבע שנים נחת ב-Wix, ובחמש השנים האחרונות עומד בראש צוות קטן ומוכשר שמוביל את תחום ה-Frontend במדיה – וידיאו, תמונות, וקטורים ואנימציות בפלטפורמת בניית האתרים של Wix.

צחי נמני

צחי נמני הוא מומחה בפיתוח, סרבר, אוטומציה ודב אופס
מתמחה בטכנולוגיות הבאות: Java, Kotlin, Node.js, C#, Docker, Kubernetes, Git, Selenium, Appium, Cypress ו־Terraform.
כיום הוא עובד בחברת הייטק בתחום התיירות. בנוסף, צחי מייעץ ומרצה לחברות וליחידים בנושאים של CI/CD, Automation Development, Docker, Kubernetes ועוד.
כרגע הוא עובד על סדרה של הרצאות בנושא Docker and Kubernetes
צחי מאמין גדול בקוד פתוח ותורם לפרוייקטים שהוא משתמש בהם.

על החברות התומכות

Really Good

Really Good היא בוטיק פיתוח Front End שעובדת עם סטארטאפים וחברות טכנולוגיה מאז הקמתה ב-2012 על ידי שחר טל ורוני אורבך. אנחנו נהנים לבנות אפליקציות מורכבות עם UX מוקפד במגוון טכנולוגיות ללקוחות מעניינים שחוויית המשתמש חשובה להם, ושומרים על איזון בריא בין עבודה לחיים.

אנחנו מגייסים מפתחי Front End מנוסים וממש טובים עם תשומת לב לפרטים הקטנים.

https://reallygood.co.il

אאוטבריין

אאוטבריין נוסדה בשנת 2006, על ידי זוג היזמים ירון גלאי (כיום CEO) ואורי להב (CTO הנוכחי ומנהל כללי- אאוטבריין ישראל). החזון המרכזי של החברה הוא לייצר את Feed התוכן האמין והמשמעותי ביותר. מוצר ההמלצות המרכזי שאאוטבריין פיתחה, מבוסס טכנולוגיית machine learning ומודלים שחוזים את הרגלי צריכת התוכן של הגולש על בסיס התנהגויות חוזרות. באמצעות כך, האלגוריתם לומד את הגולש ושולח לו המלצות תוכן בזמן אמת. בין היתר, אאוטבריין פיתחה מוצרים טכנולוגיים נוספים, ה Outbrain Amplify וה Outbrain Smartfeed. טכנולוגיית פרסום הנייטיב של אאוטבריין מציגה את החדשות והידיעות של אתרי התוכן והערוצים המובילים בעולם לרבות MSN, CNN, BBC, The Washington Post, The Guardien, Spiegel online, El Pais וכן Sky News.

מטה חברת אאוטבריין נמצא בניו יורק, משרדי החברה ממוקמים ב-19 ערים בעולם, מרכז המחקר והפיתוח ברובו יושב בישראל, במשרדים בנתניה.

קישורים:

https://twitter.com/OutbrainEng

https://www.facebook.com/Outbraineng

https://www.youtube.com/channel/UCJLORR2uJglrKm-JlKV-rJA

https://www.outbrain.com/techblog

https://medium.com/outbrain-engineering

Chegg

סטודנטים בארה"ב, כמו בכל מקום בעולם, רוצים לרכוש השכלה גבוהה כדי למצוא עבודה טובה, מעניינת וכמובן רווחית. רק שלימודים גבוהים עולים כסף – והרבה. אז איך משיגים עבודה טובה כשמלכתחילה אין מספיק כסף לממן תואר ראשון או שני ולהמשיך לחיות תוך כדי? כאן בדיוק Chegg נכנסת לתמונה.

Chegg היא חברה אמריקאית שחרתה על דגלה לעזור לסטודנטים במהלך לימודיהם ומכוונת להפוך את הצורך בהשכלה לאפשרי ומשתלם לכל אחד. את השם היא בחרה בגלל אותה דילמה, והוא למעשה הלחמה של שתי המילים -Chicken ו-Egg- שמסמלות את השאלה העתיקה ביותר בעולם הפילוסופיה – מה בא קודם: הביצה או התרנגולת? הלימודים האקדמאיים או הניסיון בשוק העבודה?

החברה שמה את הסטודנט ואת הצרכים שלו בראש, ובהתאמה- "Students First" הוא אחד הערכים המרכזיים לאורם פועלת החברה. Chegg עוזרת לסטודנטים לא רק להיות מצטיינים במהלך הלימודים, אלא גם אחריהם, בתהליך חיפוש העבודה. היא משפרת את התוצרים של הסטודנט האמריקאי במגוון שירותים: מהוזלת קניית ספרי לימוד בצורה משמעותית, דרך הצעת שירותים ויישומים לשיפור הזיכרון בלמידה למבחנים ומומחי תוכן שמסייעים מרחוק, ועד עזרה במימון הלימודים, מציאת המקום להתמחות לצבירת ניסיון ואפילו תכנון של קריירה והשלמת מיומנויות עסקיות שחסרות באקדמיה.

החברה האמריקאית הוקמה ב-2007, היא חברה ציבורית בינלאומית ועובדים בה למעלה מ-1500 אנשים. המשרד הישראלי, שממוקם ברחובות, נוסד בתחילת 2011 והתחיל

כפרויקט חיצוני עבור Chegg והיום- המשרד הקטן ברחובות נחשב לחוד החנית של כל תחום הפיתוח של החברה.

Wix.com

Wix.com הינה פלטפורמה גלובלית מובילה לפיתוח נוכחות מקצועית באינטרנט, עם למעלה מ-160 מיליון משתמשים רשומים ברחבי העולם כיום. Wix נוסדה על התפיסה כי רשת האינטרנט צריכה להיות נגישה לכל לפתח, לייצור ולשתף תוכן. באמצעות שירותים חינמיים, כמו גם שירותים נוספים למנויי פרימיום, מאפשרת Wix למיליוני עסקים, ארגונים, אנשי מקצוע ואנשים פרטיים ליצור ולנהל נוכחות מקצועית ודינמית ברשת. Wix ADI, ה-אדיטור של Wix, שוק האפליקציות Ascend by Wix ו-Corvid by Wix מאפשרים למשתמשים לבנות ולנהל נוכחות דינאמית דיגיטלית. מטה חברת Wix ממוקם בתל-אביב ולחברה סניפים נוספים בבאר שבע, חיפה, ברלין, סן פרנסיסקו, ניו יורק, מיאמי, לוס אנג'לס, סאו פאולו, וילנה, קייב, ניפרו, דבלין וטוקיו.
במחלקת הפיתוח שלנו, Wix Engineering, שותפים מפתחים מובילים אשר מעצבים את הארכיטקטורה של השירותים והמוצר שלנו. הם קובעים את הטון ואת סטנדרט הפיתוח, תוך שילוב של היבטי הנדסה, ניהול מוצר, DevOps, ניתוחים וניהול טכני. הם גם מנטורים, ועוזרים למהנדסים פחות מנוסים לשפר את כישוריהם, כך שגם הם בסופו של דבר יהיו מסוגלים להיות מנהלים טכניים בעצמם. אנו כותבים בסקאלה, ג'אווה, ג'אווהסקריפט, ריאקט, פייתון, גו ועוד. נוסף על כך, אנו מאמינים מאוד בהעצמת קהילת המפתחים בארץ ובעולם, ולהעניק בחזרה על ידי הצגה בכנסים, כמו גם תרומה לקוד פתוח. המהנדסים שלנו גם תומכים ומשתתפים בכל מיני אירועים טכנולוגיים ללא מטרות רווח.
מוזמנים לנסות בעצמכם לבנות אתר ב-Wix.com ולראות עד כמה הפעולה הזאת פשוטה - מה שבעצם מעיד על המורכבות ההנדסית העמוקה מאחורי הקלעים.

על ג'אווהסקריפט

ג'אווהסקריפט החלה את צעדיה הראשונים ב-1993 כשפה שפועלת בסביבת דפדפן ונועדה להעשיר דפי HTML. בג'אווהסקריפט יכולנו ליצור אנימציות ופידבק למשתמשים – כל דבר שהיה קשור לאתרי אינטרנט דינמיים, למשל דברים שנראים היום מאוד טריוויאליים ופשוטים כמו לחיצה על כפתור שעושה פעולה כלשהי בדף. אף על פי שההתחלה שלה הייתה צנועה, ברנדון אייך, ממציא השפה, יצר אותה מלכתחילה כשפה גמישה מאוד. הגמישות הזו, וגם חוסר ההבנה של רבים מהמתכנתים שהשתמשו בה בנוגע לעקרונות הבסיסיים שלה, גרמו ללא מעט מתכנתים בשפות אחרות לזלזל בה. גם השם שלה לא סייע לתדמית. השם ג'אווהסקריפט נקבע מסיבות שיווקיות בלבד – JAVA היא שפת תכנות פופולרית, ואנשי נטסקייפ חשבו שכך יוסיפו לתדמיתה. בפועל השם הזה לא ממש עזר, ואין כמובן שום קשר בין ג'אווה לג'אווהסקריפט.

על אף ההתחלה הקשה, ג'אווהסקריפט הפכה לפופולרית מאוד. בשנת 1996, חברת נטסקייפ העבירה את השליטה בסטנדרטים של השפה אל ארגון ECMA, ארגון אירופי (היום בינלאומי) המתמחה בתקינה. המהלך הוביל לשחרור הספסיפיקציה של השפה, שידוע בשם ECMAScript, וג'אווהסקריפט "התיישרה" לפי התקינה של ECMAScript. משנת 1997, שנת שחרור ECMAScript, ג'אווהסקריפט, כפי שהיא מיושמת בדפדפנים שונים, עוקבת אחר התקינה של ECMAScript, שהיא בעצם "תוכנית המתאר", וג'אווהסקריפט עצמה היא היישום. לכל גרסה יש מספר משלה בצמוד למילים ES (ראשי תיבות של ECMAScript).

מיקרוסופט התנגדה בתחילה ליישום השפה ויישמה שפה משלה בשם Jscript בדפדפן אינטרנט אקספלורר, שהייתה בנויה בדומה לג'אווהסקריפט. למרות היריבות הגדולה בין אנשי מיקרוסופט לאנשי ECMA, שנוצרה כתוצאה מפיתוח שתי שפות שנשענות על שני תקנים מתחרים, העקרונות של ג'אווהסקריפט שולבו גם בגרסה של מיקרוסופט. הפופולריות של השפה עלתה כאשר מקרומדיה (יוצרת פלאש) שיתפה פעולה עם ארגון ECMA ושילבה את עקרונות השפה בשפת Actionscript, ששימשה את תוכנת פלאש שהייתה פופולרית מאוד אז.

בשנת 2008 נפגשו אנשי מיקרוסופט ו-ECMA באוסלו והחלו בשיחות שלום. בניגוד לשיחות שלום אחרות שהתקיימו באוסלו, שיחות השלום האלו הסתיימו בהצלחה. תקן ES5, הגרסה הרביעית של ג'אווהסקריפט, שוחרר ויושם בכל הדפדפנים שהיו קיימים אז. מאז, התפתחות השפה והתפוצה שלה הואצו דרמטית. דפדפן כרום, שמריץ ג'אווהסקריפט באופן יוצא דופן, נכנס אל השוק בסערה ואפשר למפתחי ג'אווהסקריפט לכתוב סקריפטים שפועלים על מנוע V8 העוצמתי של כרום ולהריץ ג'אווהסקריפט במהירות מסחררת. השימוש ב-AJAX תקשורת אסינכרונית עם השרת – נכנס לפעולה, החליף שיטות מיושנות כגון Long polling ואפשר לאתרים לספק חוויות שימושיות מדהימות למשתמשים. בשנים האחרונות, פריימוורקים וספריות ג'אווהסקריפט אפשרו פונקציונליות מורכבת מאוד וספריות אחרות אפשרו כתיבה של ג'אווהסקריפט גם לטלפונים ניידים ואפילו בקלות. הראשונות שבספריות האלו נקראו MooTools ו-jQuery והן אפשרו לכל מתכנת לכתוב אפליקציות בקלות. הספריות האחרונות נקראות ריאקט, אנגולר ו-vue והן מאפשרות לבנות תוכנות מורכבות מאוד על גבי הדפדפן (צד הלקוח). ג'אווהסקריפט לא נותרה מוגבלת רק לצד הלקוח, כלומר לדפדפנים ולמכשירי קצה אחרים; המימוש של ג'אווהסקריפט לצד השרת, הידוע בכינויו node.js, הפך לפופולרי גם בשרתים. ג'אווהסקריפט מריצה כיום אפליקציות מורכבות גם בצד השרת, במיוחד אפליקציות שצריכות לבצע קריאות ולשרת מיליוני משתמשים. כיום אפשר למצוא ג'אווהסקריפט בכל מקום: באתרי אינטרנט, באפליקציות של טלפונים ניידים, באפליקציות המיועדות למחשבים רגילים וכמובן בשרתים. הביקוש למתכנתי ג'אווהסקריפט נמצא בשיאו ואין זה פלא – אפשר לעשות בשפה הזו המון דברים יישומיים כמעט מאפס. יש כל כך הרבה ספריות וכלי עזר, עד שכמעט בכל שבוע יוצאת ספרייה שימושית חדשה. בעזרת ידע מועט אפשר לעשות הרבה מאוד. מה שחשוב הוא ידע בסיסי בשפה.

בשנים האחרונות, תקן ES מתעדכן בכל שנה ומתווספים אליו תכונות ושימושים חדשים. ספר זה מעודכן לגרסה האחרונה של ECMAScript. חשוב לזכור שאם התקן מתעדכן, אין פירוש הדבר שהעדכון החדש מופיע מייד בדפדפנים שמריצים ג'אווהסקריפט או בשרתים שמריצים ג'אווהסקריפט, אלא לוקח זמן עד שהעדכונים החדשים ביותר עושים את דרכם אל הדפדפנים/שרתים שכולנו משתמשים בהם. אם שמעתם מפתחי אינטרנט "מקטרים" על דפדפנים ישנים – זו בדיוק הסיבה.

זה המניע לכתיבת הספר. הבן שלי, כיום מתכנת בזכות עצמו, ניסה ללמוד ג'אווהסקריפט מאפס ולא הצליח למצוא ספרים בעברית. החומר שיש כיום בעברית בנוגע

לג'אווהסקריפט הוא דל ומיושן. חלק מחוברות העזר הנמצאות בבתי הספר מתייחסות לתקנים שהפסיקו להיות בשימוש בשנת 2007! התייחסות אמיתית לתקנים החדשים ביותר של השפה, שיצאו בשנת 2017, אין בנמצא בעברית. התחלתי לכתוב הסברים עבור בני, ומפה לשם הבנתי שאני חייב לקחת את זה הלאה.

ג'אווהסקריפט היא שפה שקל ללמוד. בניגוד לשפות אחרות, לא נדרשים בה סביבת שרת מורכבת או כלי פיתוח שעולים כסף. לא נדרש ידע מקיף במדעי המחשב. כל שצריך הוא לפתוח Notepad במחשב, לפתוח דפדפן ולהתחיל ללמוד ולפתח. צריך גם הדרכה נכונה ומשמעת עצמית. אני מקווה שבספר הזה תמצאו לפחות הדרכה נכונה. המשמעת העצמית – עליכם. אני מאמין שככל שתתקדמו בספר תראו את פירות הלימודים, הניצוץ בעיניים יתחזק ולא תצטרכו עוד משמעת עצמית – אתם פשוט תתאהבו בג'אווהסקריפט בכלל ובפיתוח לווב בפרט. אל תדלגו על הפרק שבו אני מסביר איך ללמוד; זהו הפרק החשוב ביותר בספר.
אני מאחל לכם הצלחה רבה, בין שאתם מתכנתים בתחילת דרככם בין שאתם מתכנתים ותיקים שמשתמשים בספר כדי לשפר את הידע שלכם.

— רן בר-זיק

איך לומדים

לא למדתי מדעי המחשב באוניברסיטה או במכללה. למען האמת, עד גיל מאוחר מאוד לא למדתי תכנות בעזרת מדריך. רוב מה שאני יודע למדתי ללא הדרכה, ומכמה סיבות: הראשונה היא שכאשר עשיתי את צעדיי הראשונים בתכנות, בשנת 1996, החומר שהיה זמין באינטרנט היה דל מאוד. על חומר בעברית לא היה מה לדבר, והחומר באנגלית סיפק בדרך כלל רק את הדוקומנטציה. אני לא ממש מתגאה בכך; למידה לבד ללא הדרכה היא למידה מאוד לא יעילה. הדרכה משמעה לא רק מרצה מנוסה, אלא גם אתר אינטרנט שבו יש הסבר מקיף, פורום או קבוצה ברשת חברתית זו או אחרת (לא רק פייסבוק), שיש בהם אנשים מנוסים שיכולים לסייע או להפנות לחומרי עזר. היא גם מקום שבו אפשר לתרגל ולהתנסות. למרבה המזל, בימים אלו קיימים שלל חומרים, עזרה וסיוע. גם הספר הזה הוא הדרכה. לא תידרשו לצלול לתוך הדוקומנטציה של ECMAScript על מנת להבין את השפה, אבל גם בעזרת הספר תמצאו דרך טובה ללמידה.
כיוון שרוב הזמן למדתי ללא הדרכה, גיבשתי כמה עקרונות למידה שהכנסתי לספר הזה. אני ממליץ לכם לעקוב אחריהם.

ארגנו לעצמכם סביבת עבודה מסודרת

הפרק הראשון עוסק בבניית סביבת העבודה והוא הפרק החשוב בספר. ארגנו את המחשב שלכם וסדרו לעצמכם תיקייה מאורגנת שבה תשמרו את כל התרגולים. אם המחשב שלכם מקפיץ התראות של עדכוני ג'אווה או משהו בסגנון דומה, טפלו בכך.
ודאו שאתם יכולים להיכנס לאתר הלימודי שמלווה את הספר ושהסיסמה שלכם תקינה ופועלת.

קראו את הפרקים לפי הסדר

הפרקים לא פוזרו באקראי אלא תוכננו בסדר מסוים. אין טעם ללמוד AJAX לפני שמבינים איך תכנות אסינכרוני עובד. אין טעם ללמוד תכנות אסינכרוני לפני שמבינים איך קולבקים עובדים. ואם גם זה נשמע לכם ג'יבריש, סימן שאתם צריכים להתחיל מההתחלה. קראו כל פרק לפי הסדר.

קראו כל פרק פעמיים וכתבו לעצמכם את כל הדוגמאות

קראו את הפרק פעם אחת קריאה שוטפת. לאחר מכן קראו אותו שוב. הפעם קחו את כל דוגמאות הקוד, העתיקו אותן לסביבת העבודה שלכם ושחקו בהן! נסו לשנות את הערכים, לגרום לשגיאות, לכתוב קוד דומה.

אני מאמין שקוד לומדים דרך הידיים ולא רק דרך הראש. חשוב להבין את התיאוריה ואת הרעיונות מאחורי מה שמנסים לעשות, אך ללא מימוש, הידע הזה יתנוון וייעלם, בדיוק כמו בשפת דיבור. אם לא תשתמשו ותתרגלו, גם הלימוד התיאורטי המעמיק ביותר לא יהיה שווה הרבה. הקריאה הראשונה נועדה ללימוד התיאוריה, הקריאה השנייה נועדה לתרגול. לימוד שפה הוא לא מרוץ! קחו את הזמן, כתבו את כל דוגמאות הקוד ונסו לכתוב כאלו משלכם.

פתרו את התרגילים לדוגמה בסוף כל פרק

בסוף כל פרק יש תרגילים לדוגמה. נסו לפתור אותם. אל תתייאשו מהר ואל תרוצו אל הפתרון אלא שברו קצת את הראש. הצלחתם לפתור? נהדר. קראו את הפתרון המוצע ואת ההסבר וראו אם הם דומים לשלכם או שונים במקצת. יש יותר מפתרון אפשרי אחד לכל בעיה...

התייעצו עם אחרים

יש לא מעט קבוצות בעברית (בפייסבוק, אבל לא רק) המיועדות ללימוד ג'אווהסקריפט ולדיון בה. מצאו את זו שהכי נוח לכם בה. אל תהססו לשאול שם שאלות. לא הבנתם משהו? דוגמה כלשהי לא הייתה מובנת? הפתרון לתרגיל לא היה ברור מספיק? שאלו שם, ובעברית.

אל תהססו לקרוא, למשל דיון על משמעות השפה, להשתתף בדיונים או לסייע למישהו שהידע שלו דל משלכם. אל תשכחו להבין את רוח הדברים בקבוצה ולא להציק או להעיק. רוב המשתתפים בקבוצה הם אנשים עובדים שלאו דווקא זמינים להודעות מיידיות.

הגיעו לסדנאות ולמיטאפים

לא מעט חברות מארגנות בחינם סדנאות, מיטאפים ומפגשים שבהם מתכנתים מציגים את ג'אווהסקריפט ומרצים עליה. כדאי מאוד להגיע למפגשים האלו, לא רק על מנת לשמוע את התכנים ולהשתתף בסדנאות אלא גם להכיר אנשים אחרים בתעשייה. עם חלקם אתם תתכתבו בקבוצות הפייסבוק לג'אווהסקריפט. קהילת מפתחי הג'אווהסקריפט בארץ היא קהילה מאוד שיתופית וקרובה, ורבים בה מכירים זה את זה.

תרגלו, תרגלו, תרגלו

הספר לא שווה הרבה בלי תרגול מקיף ושימוש בו. אל תעברו לפרק הבא לפני שתסיימו את כל התרגולים שקשורים לפרק שקראתם. זה לא קריטי, אלא סופר-קריטי.
באתר hebdevbook.com, המלווה ספר זה, ישנו אתר תרגילים המיועד לקוראי הספר.

העזרו ב-Chat GPT וכלים דומים

מומלץ להעזר בבינה מלאכותית בכל הקשור ללימודים. הבינה המלאכותית הנפוצה כיום היא Chat GPT וניתן להעזר בשרותיה באמצעות כניסה אל האתר https://chat.openai.com ויצירת חשבון משתמש חינמי. ניתן להזין אל הצ׳אט שאלות ובקשות שונות הקשורות לקוד. למשל: Explain about this error ואז את השגיאה המתקבלת. ניתן גם לפנות לבינה המלאכותית בבקשות לדוגמאות שונות והסברים על נושאים בספר. למרות שרמת העברית של הבינה המלאכותית משתפרת, מומלץ לשאול את השאלות באנגלית.
אני ממליץ להמנע מהפיתוי להשתמש בבינה מלאכותית לבצע תרגילים עבורכם או לתכנת עבורכם. מעבר לכך שתהליך הלימוד לא יהיה אפקטיבי, בסופו של יום הבינה המלאכותית לא יכולה להחליף מתכנתים והכרות עם הקוד נדרשת על מנת לפקח עליה או להנחות אותה באופן אופטימלי. אם לא תכירו היטב קוד, לא תוכלו לעבוד איתה ביעילות תוך יצירת קוד איכותי מספיק לשימושים בעולם האמיתי.

התקנת סביבת העבודה ודרך הלימוד

ג'אווהסקריפט יכולה לרוץ בסביבת שרת או דרך הדפדפן. איך זה בדיוק עובד? ג'אווהסקריפט נמצאת בקובץ טקסט. כן, בדיוק כמו זה שאפשר ליצור באמצעות כתבן הטקסט (Notepad) שיש בכל מערכת חלונות. בדפדפן מותקן כלי שלוקח את קובץ הטקסט הזה ומריץ אותו ואת הפקודות שנמצאות בתוכו. אם הדפדפן היה אדם, הוא היה פותח את קובץ הטקסט וקורא את מה שיש בתוכו, למשל: "לך ימינה ופתח את הדלת", ועושה בדיוק מה שכתוב. הפעולה הזו נקראת בלשון הפופולרית "רינדור", מלשון render בלעז. הדפדפן לוקח את קובץ הג'אווהסקריפט ומריץ אותו. קובץ הג'אווהסקריפט יכול לעשות כל מיני דברים ולהציג או לא להציג אותם.

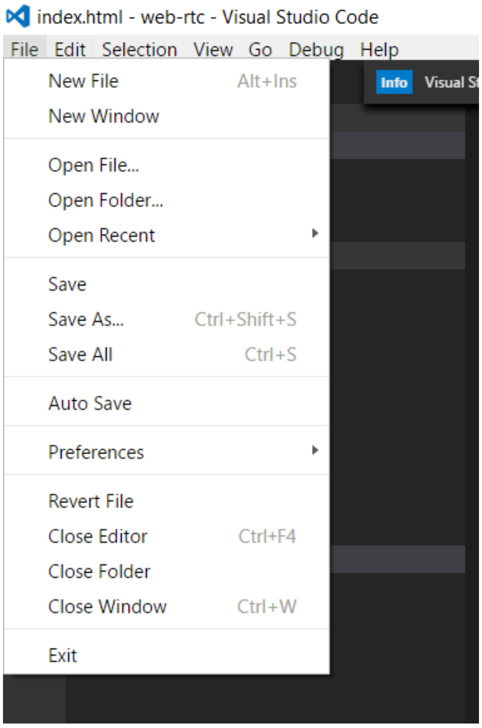
איך הדפדפן טוען את קובץ הג'אווהסקריפט? יש כמה דרכים לעשות זאת, אבל כרגע ארצה ללמד אתכם איך לכתוב קובץ ג'אווהסקריפט ולראות אותו פועל על מנת להבין את כללי השפה ולכתוב משהו באופן ראשוני ביותר. החלק החשוב והקשה ביותר הוא יצירת סביבת העבודה, כלומר סביבה ממוחשבת שבה אפשר להקליד ג'אווהסקריפט ולראות אותה עובדת. סביבה זו היא חשובה מאוד כאשר לומדים, כיוון שלימוד של שפת תכנות נעשה ראשית כול "דרך הידיים" וחשוב מאוד לא רק לקרוא אלא גם לתרגל. וכדי לתרגל צריך סביבה שמאפשרת להקליד פקודות שפה, לשמור ולראות את הפלט. אני שב ומדגיש: התקנת הסביבה היא החלק הקשה ביותר בתחילת לימוד שפה חדשה וגם החשוב ביותר. לפיכך כדאי להיאזר בסבלנות, לקחת נשימה ארוכה ולזכור שדווקא עכשיו מתמודדים עם החלק הקשה ביותר.

הבה נתחיל בעורך טקסט טוב. אמנם אפשר להשתמש בנוטפד, אבל הוא לא מציע צביעת קוד לצורך עזרה בקריאות ולא יצירת הזחות בקלות. יש כמה עורכי טקסט המותאמים במיוחד לכתיבת ג'אווהסקריפט, שאציין כמה מהם. בחרו בעורך טקסט אחד! רובם זהים למדי ומכילים יכולת עריכה בסיסית של HTML, CSS וג'אווהסקריפט.

שימו לב: מתכנתים מנוסים לא משתמשים בעורך הטקסט הפשוט אלא בעורך טקסט משוכלל יותר שנקרא IDE או Integrated Development Environment – סביבת פיתוח משולבת. הסביבה הזו מאפשרת השלמת קוד, מציינת שגיאות בכתיבה וגם יכולה להריץ את הקוד עצמו.

IDE מעולה הוא Visual Studio Code. הוא חינמי, מבוסס קוד פתוח, כתוב בג'אווהסקריפט (כן, כן), נתמך על ידי מיקרוסופט וניתן להורדה פה: https://code.visualstudio.com/.

אחרי ההתקנה תוכלו לפתוח את התוכנה, לבחור ב-File ואז לפתוח את התיקייה שבחרתם ולפתוח או ליצור בה קבצים. אני ממליץ לכם בחום להוריד את עורך הקוד הזה: הוא חינמי לחלוטין, אינו מכביד על המחשב וכן נפוץ בתעשייה היום.



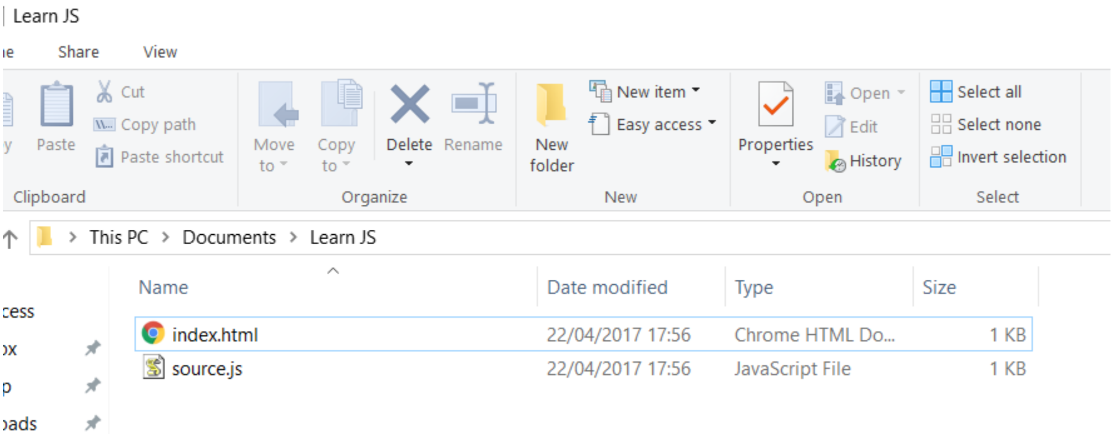
IDE מוצלח אחר הוא Atom. גם הוא חינמי ומבוסס קוד פתוח וגם הוא... כתוב בג'אווהסקריפט. הוא נתמך על ידי גיטהאב וניתן להורדה פה: https://atom.io/. הוא די דומה ל-Visual Studio Code ואחרי ההורדה וההתקנה שלו אפשר להפעיל אותו בקלות. עורך טקסט נוסף שנחשב לאמין וטוב הוא תוכנה חינמית בקוד פתוח שנקראת Notepad++. הוא ניתן להורדה בקישור הבא: https://notepad-plus-plus.org/download והוא בסיסי יותר משני קודמיו.

שימוש ב-Visual Studio Code או ב-Atom הוא מומלץ יותר כיוון שיש בו "השלמה אוטומטית" של פקודות נפוצות בג'אווהסקריפט, דבר המקל מאוד את הלמידה. כמו כן הוא מציג שגיאות בקוד כבר במהלך הכתיבה, עוד לפני ההרצה.

הסביבה הטובה ביותר ללימוד היא יצירת קובץ HTML שטוען קובץ ג'אווהסקריפט. קובץ HTML הוא קובץ שהדפדפן יודע לפרש ולהציג, והוא יכול לקרוא לקובץ ג'אווהסקריפט באופן כזה שהדפדפן ירנדר אותו. כאמור, רינדור הוא הרצת הפקודות שנכתבות בקוד

ג'אווהסקריפט והצגתן על המסך או במקום אחר. רינדור, מלשון render, הוא מונח מתחום מדעי המחשב ופירושו הוא "הרצה". כשאני כותב "הקוד מרונדר" הפירוש הוא שהקוד רץ ומציג את התוצאות. יש ליצור במחשב תיקייה – זה יכול להיות ב"המסמכים שלי" או על שולחן העבודה – ובתוכה ליצור קובץ ששמו source.js וקובץ ששמו index.html.

בחלונות יצירת תיקיה נעשית באמצעות: כניסה אל סייר הקבצים, בחירת המקום שבו רוצים למקם את התיקיה. לחיצה על המקש הימני של העכבר ואז בחירה ב"חדש" וב"תיקיה". את הקובץ עצמו כדאי ליצור באמצעות התוכנות Visual Studio Code או Atom או אפילו ++Notepad. פיתחו את אחת התוכנות האלו (אני ממליץ על Visual Studio Code) נווטו אל התיקיה וביחרו ב-File ואז New.



ודאו שמערכת החלונות או המק שלכם תומכת בתצוגת שם הקובץ המלא, כולל הסיומת (Extension) של הקובץ. אחרת, קובץ ה-index.html יהיה בעצם index.html.txt.

בקובץ ה-HTML כתבו את הקוד הבא:

```
<!doctype html>
<html>

<head>
  <meta charset="utf-8">
</head>

<body>
  <script src="./source.js"></script>
</body>

</html>
```

בתוך קובץ ה-source.js כתבו את הטקסט הבא:

```
document.write('Hello World!');
```

אחרי ששמרתם את תוכן שני הקבצים, פתחו את הקובץ index.html בדפדפן כרום או בפיירפוקס (לא באדג'). אם הכול תקין, תראו שכתוב על המסך "Hello World!" – כתבתם את הג'אווהסקריפט הראשון שלכם!

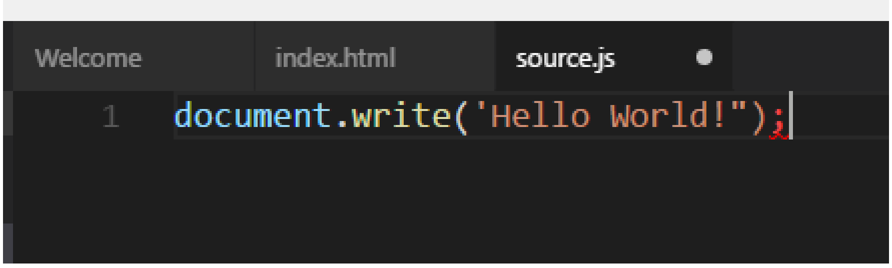
שימו לב: זה השלב המועד ביותר לפורענות, שעלול להיות מתסכל מאוד, אבל הוא שלב חשוב ביותר ואסור להרים ידיים ולהתייאש. אם פתחתם את הקובץ ודבר לא הופיע, נסו את הדברים הבאים:

בדקו שאכן קראתם לקבצים index.html ו-source.js. הבדיקה צריכה להתבצע באמצעות הכפתור הימני של העכבר בחלונות, כדי למנוע מצב שבו מערכת ההפעלה הוסיפה תוספות לשמות ו-index.html נשמר בשם index.html.txt.

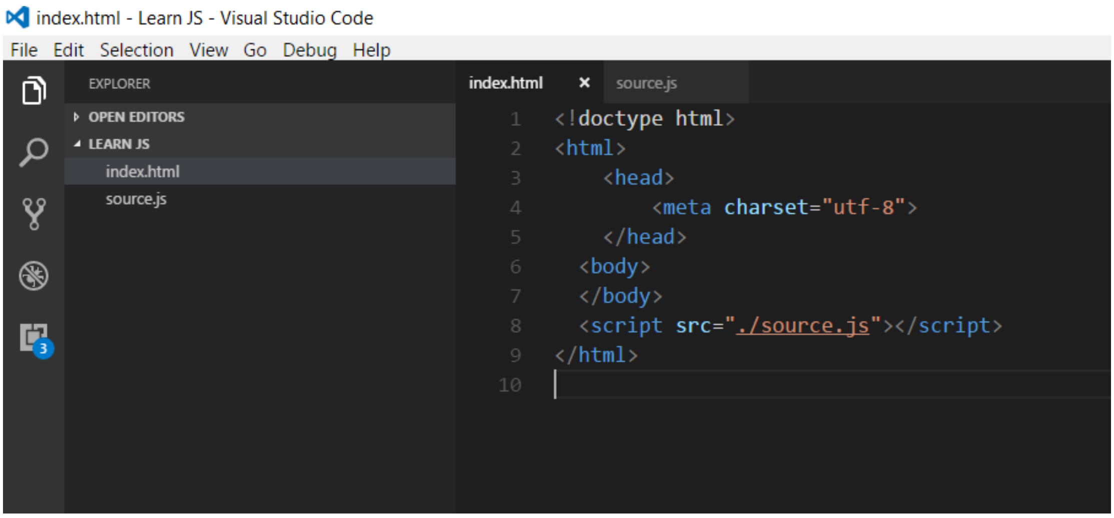
בדקו שאתם פותחים את הקבצים בדפדפן כרום או בפיירפוקס. בדקו שאין תוספים מיוחדים לדפדפנים שעלולים לחסום את הרצת הקובץ על ידי הרצה של מצב פרטיות.

בדקו שהקלדתם את הטקסט כשורה בקובץ source.js ללא רווחים או תווים מיוחדים.

נסו להשתמש ב-Atom או ב-Visual Studio Code. שימו לב שאין התראות על שגיאות הקלדה. כאן למשל מובאת דוגמה של שגיאת הקלדה שביצעתי. אם קיבלתם התראה כזו, בדקו שוב שלא טעיתם בגרש ושלא הכנסתם רווחים מיותרים כמו בדוגמה הזו שבה יש שימוש שגוי בגרש יחיד ואז בגרשיים. עדיף להשתמש תמיד בגרש יחיד:



אם אתם משתמשים ב-Visual Studio Code או ב-Atom, סביבת כתיבת הקוד שלכם אמורה להיראות כך:

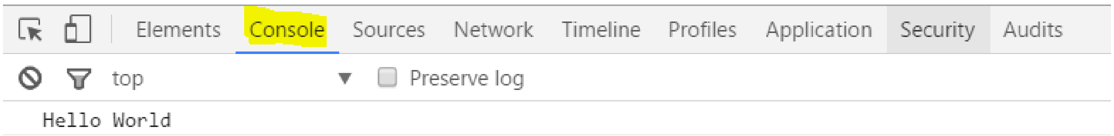


מצד שמאל תוכלו לראות את כל הקבצים בתיקייה. כדאי ליצור תיקייה מיוחדת בשם learnJavaScript או בשם דומה ולא לשים את כל הקבצים בתיקייה משותפת כמו "המסמכים שלי".

מצד ימין תוכלו לצפות בתוכן הקבצים. בצילום המסך רואים את תוכן הקובץ index.html. אם תקלידו דבר מה, תוכלו לראות שיש השלמה אוטומטית של קוד HTML, ואם תקלידו בקובץ source.js שמכיל את הג'אווהסקריפט תראו שיש השלמה אוטומטית של פקודות ג'אווהסקריפט. נוסף על כך, תקבלו גם התראה על קוד לא תקין.

כיוון שאתם כבר מפתחי ג'אווהסקריפט מקצועיים, כדאי שתלמדו להשתמש בקונסולה של הדפדפן. מדובר בממשק מיוחד שמאפשר "לדבג" את ג'אווהסקריפט. הפועל "לדבג" כוונתו להסתכל על צפונות הרינדור ולראות ממש את הפלט של השפה או את תוצאות ההרצה שלה. כלומר מה שכתבנו. נשמע מסובך? יש להבין איך הדפדפן מנסה להריץ את הקוד שלו (כאמור מה שנקרא "רינדור", מלשון הרצה, באנגלית) ולקבל מידע נוסף במקרה שהוא נכשל, כלומר "זורק" שגיאה צבועה באדום בקונסולה ואף מפנה לשורה הבעייתית.

על מנת לראות את הקונסולה, פתחו את כלי המפתחים של הדפדפן. בכרום ובפיירפוקס לחצו Ctrl + Shift + i אם יש לכם חלונות או Cmd + Option + i אם יש לכם מק. אפשר לעשות את זה גם דרך התפריט העליון בשני הדפדפנים. לאחר פתיחת כלי המפתחים, לוחצים על לשונית Console.

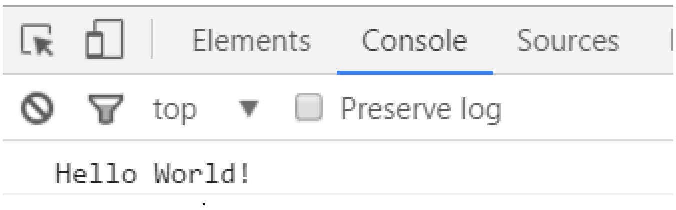


הבה נבדוק מה אפשר לעשות בעזרת הקונסולה. היכנסו אל source.js והחליפו את הטקסט ל:

```
console.log('Hello World!');
```

טענו מחדש את הדף באמצעות Ctrl + F5 בחלונות או Cmd + R במק. הטעינה מחדש חשובה. הדפדפן לא יודע שהוכנסו שינויים בקובץ הג'אווהסקריפט, ויש לגרום לו לטעון מחדש את קובץ הג'אווהסקריפט על מנת להריץ את הפקודות החדשות. אחרי הטעינה מחדש הסתכלו על לשונית ה-Console.

אם הכול תקין, המסך יהיה ריק, אך בקונסולה תראו Hello World! יש!



חשוב: אל תדלגו על השלב הזה. בכל שלבי הלימוד כדאי להשתמש בקונסולה, שהיא הרבה יותר נוחה להצגה. אם משהו לא עובד, אנא בדקו את הדברים הבאים:

1. האם השלמתם את שלב הקוד? הצלחתם להציג Hello World על המסך?
2. האם שמרתם את הקובץ לאחר השינויים?
3. האם טענתם מחדש באמצעות Ctrl + F5 את הדפדפן?

מדובר בסביבה העבודה הטובה ביותר ללימוד ג'אווהסקריפט, אך יש סביבות עבודה נוספות. ברשת יש אתרים המאפשרים לכתוב ג'אווהסקריפט ישירות, להריץ את הקוד דרכם ולראות את התוצאות. אתר מפורסם וחשוב כזה הוא https://codepen.io/ ואפשר להקליד בו פקודות של ג'אווהסקריפט. פתחו שם חשבון וצרו "pen" חדש. בדקו שיש אפשרות להכניס קודי HTML, CSS ו-JS, שהוא בעצם קיצור של ג'אווהסקריפט. אפשר להכניס את הקוד ולראות את התוצאות על גבי דף מדומה או על גבי הקונסולה של הדפדפן.

מכאן, דרך הלימוד תהיה פשוטה למדי. אני אסביר על תכונות מסוימות של השפה ואתן דוגמאות. מומלץ בחום רב להעתיק את הדוגמאות אל קובץ ה-source.js ולהריץ את הג'אווהסקריפט כדי לראות איך זה עובד באמת.

בסוף כל פרק יש תרגילים לתרגול עצמאי ומומלץ מאוד לנסות אותם בסביבת העבודה. אי-אפשר ללמוד שפה על ידי קריאה תיאורטית בלבד ורצוי לתרגל, לתרגל, לתרגל. את התרגול עושים רק בסביבת עבודה יציבה. לפיכך, אנא אל תדלגו אל הפרק הבא לפני שיש לכם סביבת עבודה יציבה. מומלץ מאוד לעבוד ב-Visual studio code החינמית.

תרגיל:

במקום "Hello World!" גרמו לקונסולה להדפיס את המילים "Ahla Bahla".

פתרון:

היכנסו לקובץ source.js, מחקו את הטקסט שיש שם והדביקו במקומו את:

```
console.log('Ahla Bahla');
```

שמרו את הקובץ, פתחו את הקובץ index.html, שטוען את הקובץ source.js, ורעננו את הדף. לחצו על Ctrl + Shift + i, לחצו על הלשונית Console וראו את התוצאות.

תרגיל:

גרמו לקובץ ה-HTML לטעון קובץ ג'אווהסקריפט ששמו targil.js ושמדפיס בקונסולה: I am new file.

פתרון:

צרו קובץ targil.js באותה תיקייה של הקובץ index.html. יש לוודא שזה שמו האמיתי של הקובץ ושמערכת ההפעלה לא מצמידה לקובץ עם סיומת txt. את זה עושים על ידי בדיקה בהגדרות התצוגה של מערכת ההפעלה. פתחו את קובץ ה-HTML בעזרת עורך טקסט (מומלץ להשתמש ב-Visual Studio Code) ושנו את שם קובץ הג'אווהסקריפט ל-targil.js.

```
<!doctype html>
<html>

<head>
  <meta charset="utf-8">
</head>

<body>
  <script src="./targil.js"></script>
</body>

</html>
```

בקובץ targil.js כתבו את השורה הזו:

```
console.log('I am new file');
```

פתחו את הקובץ index.html בדפדפן ולחצו i + Shift + Ctrl על מנת להציג את כלי המפתחים. בחרו את הלשונית Console ובחנו את התוצאה.

פרק 1

משתנים

משתנים

החלק הבסיסי והחשוב ביותר הוא המשתנה. מדובר ברכיב שיכול להכיל בתוכו מידע, ואפשר לשנותו – זו הסיבה שהוא נקרא "משתנה". משתנה בג'אווהסקריפט מוגדר באופן הבא:

```
let variant;
variant = 'Hello World';
```

מה קורה פה? יש כאן הגדרת משתנה ששמו variant. ההגדרה נעשית באמצעות המילה השמורה let. מילה שמורה היא מילה מיוחדת בשפה שהשימוש בה שמור למקרה מסוים. במקרה הזה, let שמורה אך ורק להגדרת משתנים.

לאחר מכן מכניסים ערך למשתנה. הפעולה הזו נקראת "הצבת ערך" או "השמה", והיא נקראת כך מכיוון שהערך מוצב בתוך המשתנה. במקרה הזה מדובר בטקסט הכולל את המילים Hello World.

שימו לב: יש סימן ";" בסוף כל שורה. בג'אווהסקריפט מוטב להציב סימן ";" בסוף כל שורה כדי למנוע בעיות וכדי שהסקריפט יעבוד. הוא מסמן סוף שורה עבור מנוע הג'אווהסקריפט שמרנדר את הסקריפט, ממש כמו נקודה בסוף משפט. על אף שלא כל מנוע מקפיד על כך, אני ממליץ לכם: הקפידו תמיד להקליד ; בסוף כל שורה.

אפשר לקצר ולהציב את הערך מייד בהגדרת המשתנה:

```
let variant = 'Hello World';
```

הבה נבדוק את המשתנה ואת הערך שלו. אפשר להציב את המשתנה הזה בתוך console.log כפי שלמדנו בפרק הקודם:

```
let variant = 'Hello World';
console.log(variant);
```

אם תציצו בקונסולה, תראו שמודפס המשפט "Hello World". מדוע? כיוון שזה מה שיש בתוך המשתנה. מן הסתם, משתנה ניתן לשינוי. נסו את הקוד הזה:

```
let variant = 'Hello World';
variant = 'I am a new version';
console.log(variant);
```

מה לפי דעתכם יוצא בקונסולה? יוצא "I am a new version".
למה? כיוון שהערך הקודם "נדרס" על ידי הערך החדש. הכנסתם (כלומר הצבתם) ערך חדש לתוך המשתנה, ועכשיו מה שיש בתוכו השתנה. כשמציגים את המשתנה רואים את הערך החדש.
יש הבדל מהותי בין הגדרת משתנה לבין הכנסת ערך לתוכו. הגדרת משתנה היא כמו בניית ארון או קופסה ואפשר בכל פעם להכניס לתוכו ערך אחר.

שימו לב: אחרי שמשתנה מסוים הוגדר, אי-אפשר להגדיר אותו מחדש. הקוד הבא:

```
let variant = 'Hello World';
let variant = 'I am a new version';
console.log(variant);
```

יגרום לשגיאה הבאה: **Identifier 'variant' has already been declared**
שמות המשתנים יכולים להיות מגוונים אך יש להם כמה כללים מחייבים. אפשר להשתמש בכל אות שהיא ובסימנים _ או **$** בתחילת השם, ובכל האותיות והמספרים ובסימנים _ או **$** בהמשך השם.

שמות תקינים	**שמות לא תקינים**
myVar3	3myVar מתחיל במספר
my_var	my-var מכיל את התו - שאינו תקין
$myVar	#myVar מכיל את התו # שאינו תקין
myVar	my var מכיל רווח

שימו לב: אפשר להשתמש בעברית בהגדרת המשתנים, אבל מומלץ שלא לעשות את זה גם כיוון שהקוד שלכם יהיה פחות קריא וגם כיוון שזה עלול להוביל להתנהגות מוזרה ולצרות.

אתן לכם טיפ: בעולם התכנות אל תחפשו צרות כי יש מספיק מהן גם כך.

שימו לב 2: בגרסאות קודמות של ג'אווהסקריפט השתמשו במילה השמורה var להגדרת משתנה. בתקנים החדשים כבר לא מקובל להשתמש ב-var כיוון שלשימוש בו יש השלכות שנדון בהן בהמשך הספר. הוא נשאר איתנו בעיקר בשביל תאימות לאחור עבור קוד שנכתב בשנים עברו.

תרגיל:

צרו משתנה בשם המורכב לפחות משתי מילים, הדפיסו אותו בקונסולה וודאו שבהדפסה בקונסולה יופיעו המילים "I know JavaScript".

פתרון:

```
let myVar;
myVar = 'I know JavaScript';
console.log(myVar);
```

הסבר:

יצירת המשתנה נעשית באמצעות המילה השמורה let. השמת הטקסט נעשית באמצעות הסימן =. שימו לב שהטקסט מוקף בגרשיים. פקודת ה-console.log שלמדנו בפרק הקודם משמשת להדפסת הטקסט - ובמקרה הזה המשתנה שהיא מקבלת.

תרגיל:

צרו משתנה בשם myVar והכניסו לתוכו את טקסט "me not know JavaScript". דרסו את הטקסט הזה בטקסט "I know JavaScript" והדפיסו אותו באמצעות console.log

פתרון:

```
let myVar = 'me not know JavaScript';
myVar = 'I know JavaScript';
console.log(myVar);
```

הסבר:

יוצרים את המשתנה myVar ומכניסים לתוכו את הטקסט "me not know JavaScript" ממש ברגע היצירה. לאחר מכן דורסים את הערך הזה באמצעות השמה נוספת. ההדפסה של מה שיש במשתנה נעשית באמצעות console.log.

תרגיל:

צרו משתנה בשם myVar והכניסו לתוכו את הטקסט "I am myVar". צרו משתנה נוסף בשם myVar2 והכניסו לתוכו את הטקסט "I am myVar 2".

הדפיסו את שניהם באמצעות console.log:

(בונוס: הדפיסו את שני המשתנים בקונסולה באמצעות פקודת console.log אחת)

פתרון:

```
let myVar = 'I am myVar';
let myVar2 = 'I am myVar 2';
console.log(myVar);
console.log(myVar2);
```

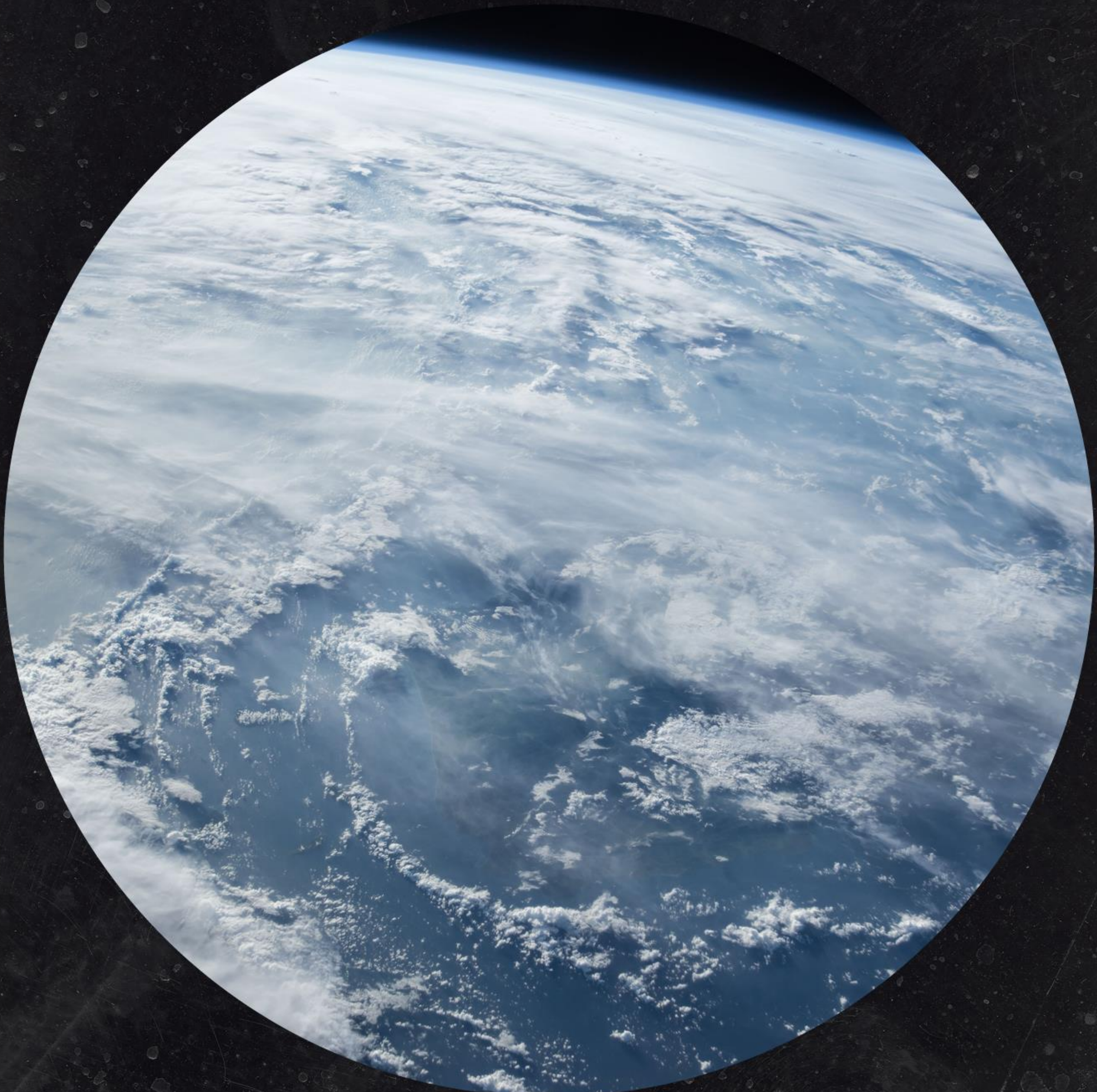
הסבר:

אין מניעה להגדיר כמה משתנים באותו סקריפט. הגדרתי את המשתנה באמצעות המילה השמורה let, שם המשתנה (הכולל מספר בסוף) והדפסה שלו ל-console.log.

הסבר לתרגיל הבונוס: נסו לחפש באינטרנט איך להדפיס ל-console שני משתנים. הרבה פעמים יהיו דברים שלא תבינו בתרגול. במקום לקרוא שוב ושוב את ההסבר, נסו לחפש תשובה ברשת. כך עושים מתכנתים מקצוענים: הם מחפשים תשובות ברשת.

פרק 2

טקסט



טקסט

טקסט (באנגלית text string) הוא סוג מידע חשוב מאוד ובסיסי מאוד במערכות מידע. משתמשים בטקסט בכל מקום ובכל מערכת, ומדובר בחלק חשוב מאוד בלמידת קוד – יותר ממתמטיקה ומפעולות חשבוניות. בעבר היה קשה לעבוד עם טקסט שאינו באנגלית. אם אתם מבוגרים מספיק אתם ודאי זוכרים כל מיני אותיות מעוותות שהופיעו באתרים או בתוכנות. היום קל לעבוד במגוון שפות בתוכנות, באתרים וכמובן בג'אווהסקריפט, בזכות תקן חדש שנקרא "**יוניקוד**". יוניקוד מאפשר לכתוב בעברית, ביפנית, ברוסית ובכל מערכת כתב אחרת בקלות ובלי חשש להופעת ג'יבריש והוא מוטמע במחשב שלכם באופן אוטומטי.

בפרק הקודם למדנו איך ליצור משתנים ולהכניס לתוכם טקסט. בפרק הזה תרחיבו את הידע שלכם בנושא הטקסט. כאמור, אפשר להכניס טקסט למשתנים בג'אווהסקריפט באופן הבא:

```
let myVar = 'This is text';
```

הטקסט מוקף בגרש בודד, אבל אפשר להשתמש גם בגרשיים כפולים:

```
let myVar = "This is text";
```

הבחירה ביניהם היא בדרך כלל עניין של טעם. רוב המתכנתים נוהגים נכון להיום להשתמש בגרש בודד להגדרת טקסט או בגרש מסוג ` (backtick), שעליו נלמד מאוחר יותר. ג'אווהסקריפט מאפשרת לחבר בין מחרוזות טקסט די בקלות בעזרת הסימן **+** (כמו בפעולת חיבור):

```
let part1 = 'foo';
let part2 = 'bar';
let myVar = part1 + part2;
console.log(myVar);
```

מה שיודפס על הקונסולה הוא **foobar**. אף על פי שסימן ה"**+**" מוכר מחיבור מספרים, בג'אווהסקריפט אפשר כאמור להשתמש בו לחיבור מחרוזות.

שימו לב: הרווח בין שמות המשתנים לבין הסימן **+** לא הכרחי להרצה תקינה אבל נוח מאוד לקריאה. חשוב לציין שהשם התקני של טקסט הוא "מחרוזת טקסט".

שימו לב: השימוש ב-foo וב-bar הוא מאוד נפוץ בדוגמאות בשפות תכנות ונדיר למצוא מדריך לשפת תכנות שלא משתמש במילים האלה ובמילה baz למשתנים (או לחלקים אחרים בקוד שעוד תלמדו עליהם). לדוגמאות האלו יש שם מפחיד: משתנים מטה-סינטקטיים – אבל לא צריך לפחד משמות מסובכים.
נשאלת השאלה מה קורה אם רוצים להכניס מחרוזת טקסט עם גרש, משהו בסגנון הזה: I don't know JavaScript.

לדוגמה:

```
let myVar = 'I don't know JavaScript';
console.log(myVar);
```

יחזיר שגיאה בקונסולה. כיוון שכל מחרוזת טקסט חייבת להיות מוקפת בגרשיים, המנוע של ג'אווהסקריפט חושב שכל מה שמגיע אחרי הגרש הוא משתנה ולא יודע מה לעשות איתו. בדוגמה הבאה, החלק המודגש הוא מחרוזת הטקסט, והחלק שאינו מודגש הוא מה שהמנוע של ג'אווהסקריפט לא יודע מה לעשות איתו:

```
let myVar = 'I don't know JavaScript';
```

יש כמה דרכים לפתור את הבעיה הזו. הדרך הטובה ביותר היא לציין בפני המנוע של ג'אווהסקריפט שהגרש שיש במילה don't הוא גרש שמהווה חלק ממחרוזת הטקסט. איך עושים את זה? באמצעות הסימן **** :

```
let myVar = 'I don\'t know JavaScript';
console.log(myVar);
```

ההדפסה תציג את הטקסט כמו שצריך, עם הגרש. הפעולה הזו נקראת **escaping** ובה לוקחים טקסט שהיה שובר את שפת התכנות והופכים אותו לבטוח. לחלופין, אפשר ליצור איתו תווים שלא ניתן לכתוב במקלדת, למשל ירידת שורה. איך כותבים ירידת שורה? בעזרת **\n**. העתיקו את הדוגמה הזו:

```
let myVar = 'I don\'t \nknow\n JavaScript';
console.log(myVar);
```

ותראו מה קורה. בקונסולה יופיעו שלוש שורות.

ייתכן שתיתקלו ב-escaping במקומות נוספים. אם רוצים לכתוב רק ****, צריך לעשות לו escaping ולכתוב ****.

כעת, כשאתם מרגישים בנוח בכל מה שקשור לטקסט או, נכון יותר, למחרוזת טקסט בג'אווהסקריפט, הבה נסבך מעט את העניינים. בג'אווהסקריפט מחרוזת טקסט נחשבת לסוג מידע. באנגלית זה נקרא **Data Type**. יש כמה סוגי מידע, כמו מספרים למשל, אבל סוג המידע של מחרוזת טקסט נקרא **string** והוא עומד ברשות עצמו. כאשר יוצרים משתנה ומכניסים לתוכו טקסט, בעצם אומרים שהמשתנה הוא מסוג מחרוזת טקסט. ברגע שמשתנה מסוים מקבל לעצמו מידע והופך למשתנה מסוג מסוים, הוא מקבל גם סט של יכולות מיוחדות, ממש כמו קלארק קנט שהופך לסופרמן. מהשנייה שנכנס למשתנה טקסט, אפשר לעשות בו כמה פעולות שאפשריות אך ורק לסוג המידע של הטקסט.

כך למשל אפשר לבצע על המשתנה המכיל טקסט את הפעולה שתגרום לכל האותיות שלו להיות ראשיות (כלומר קפיטלס). לדוגמה:

```
let myVar = 'Hello World';
let myVarUpper;
myVarUpper = myVar.toUpperCase();
console.log(myVarUpper);
```

יוצרים משתנה עם טקסט. על המשתנה הזה עושים פעולה שאפשרית אך ורק עם משתנה מסוג טקסט. שם הפעולה הוא **toUpperCase()**. התוצאות של הפעולה הזו מושמות למשתנה נוסף בשם **myVarUpper** ומודפסות. אם תעתיקו את הטקסט הזה ותריצו בקונסולה, תראו שהתוצאה היא 'HELLO WORLD'.

השימוש בהפעלת פעולות על משתנה עם **Data Type** מסוים אינו בלעדי רק למחרוזות ובהמשך הספר תראו שאפשר לבצע פעולות גם על Data Types אחרים. בינתיים צריך להבין שיש סט רחב מאוד של פעולות שאפשר להפעיל על מחרוזות, ו-**toUpperCase** היא רק אופציה אחת. על מנת להפעיל פעולות שונות משתמשים ב- . (נקודה) על המשתנה. אם משתמשים ב-IDE (עורך טקסט) כמו VS Code, כאשר תרשמו "**.**" אחרי משתנה, עורך הקוד יראה לכם את שלל הפעולות האפשריות על משתנה זה.

שימו לב שהפעולה **toUpperCase** לא משנה את המשתנה עצמו! שינוי המשתנה עצמו נקרא "מוטציה", וזה לא מה שעושים פה. כדי לקבל את הערך שהשתנה צריך להגדיר משתנה נוסף ולהכניס את הערך למשתנה.

מבלבל? הבה נדגים זאת שוב בפעולה נוספת. הפעם צרו משתנה והכניסו לתוכו טקסט. המשתנה הראשון, מהרגע שיש בתוכו טקסט, יכול לבצע פעולות של טקסט ולהחזיר את התוצאה אל משתנה אחר שאותו תדפיסו:

```
let firstVar = 'HELLO WORLD';
let secondVar = firstVar.toLowerCase();
console.log(secondVar);
```

יש פעולות נוספות שאפשר לעשות על מחרוזות טקסט, אבל כרגע התמקדו בשתי הפעולות האלו גם בתרגול.

תרגיל:

צרו שני משתנים, שאחד מהם מכיל את המילה Hello, והאחר מכיל את המילה World. חברו ביניהם, הכניסו את התוצאה אל משתנה שלישי והדפיסו אותה.

פתרון:

```
let myVar1 = 'Hello';
let myVar2 = 'World';
let answer = myVar1 + myVar2;
console.log(answer);
```

הסבר:

יוצרים ומציבים ערכים במשתנים בשם myVar1 ו-myVar2 (יש להקפיד שמחרוזות הטקסט יהיו מוקפות בגרשיים בודדים). מגדירים משתנה שלישי בשם answer ושמים בתוכו את הסכום של myVar1 ו-myVar2. ההדפסה נעשית כרגיל. שימו לב שרווח גם נחשב לאות. אם לא מכניסים רווח באחת ממחרוזות הטקסט בתוך המשתנים, לא יהיה רווח.

תרגיל:

צרו משתנה שיכיל את מחרוזת הטקסט:

"The student's and the teacher's motivations were in conflict".

פתרון:

```
let myVar = 'The student\'s and the teacher\'s motivations were in
conflict.';
console.log(myVar);
```

הסבר:

שימו לב לשימוש ב-**escaping**! מה זה escaping? פשוט שימוש בסימן **** על מנת להודיע למנוע של ג'אווהסקריפט שהגרש הוא חלק ממחרוזת הטקסט. ההשמה וההדפסה של המשתנה נעשות כרגיל.

תרגיל:

הכניסו את הערך JavaScript. המירו אותו לאותיות גדולות והדפיסו את התוצאה. המירו אותו לאותיות קטנות והדפיסו את התוצאה.

פתרון:

```
let myVar;
let answer;
myVar = 'JavaScript';
answer = myVar.toUpperCase();
console.log(answer);
answer = myVar.toLowerCase();
console.log(answer);
```

הסבר:

יוצרים שני משתנים ריקים בשם myVar ו-answer. הראשון מכיל את מחרוזת הטקסט JavaScript. אחרי שמכניסים לתוכו את הערך הזה, הוא מסוג טקסט ואפשר להשתמש בפעולה toUpperCase. את תוצאות הפעולה הזו מכניסים ל-answer ומדפיסים. אחרי ההדפסה משתמשים בפעולה toLowerCase על מה שיש ב-myVar, מכניסים את התוצאה אל answer ומדפיסים.

פרק 3

מספרים

מספרים

בדיוק כמו שאפשר להכניס טקסט לתוך משתנים, אפשר להכניס אליהם מספרים. למשל:

```
let myVar = 12;
console.log(myVar);
```

המספר 12 הוא ללא גרשיים. הוא ממש נכנס כמו שהוא, כיוון שהוא מספר. מספרים טהורים יכולים להיכנס ללא גרשיים. להזכירכם, מחרוזת טקסט, שעליה למדנו קודם, מוקפת בגרש או בגרשיים. מספרים לא מוקפים בגרש או בגרשיים.

אם תדפיסו את המספר כמו בדוגמה, תראו שיודפס 12. חדי העין שביניכם ישימו לב שההדפסה בקונסולה נראית מעט שונה כשמדובר במספר ולא במחרוזת טקסט. ההבדל נוצר בגלל השוני בין סוג המידע של מחרוזת טקסט לסוג המידע של מספר.

אם מחברים בין מספרים אזי החיבור ייעשה בדיוק כפי שהייתם מצפים:

```
let foo = 12;
let bar = 10;
let answer = foo + bar;
console.log(answer);
```

אפשר לבצע פעולות אפילו בשלב ההשמה. למשל:

```
let foo = 2 + 4 + 5;
console.log(foo);
```

התשובה שתודפס תהיה 11. 2 ועוד 4 ועוד 5.

בדומה לחיבור, אפשר לעשות פעולת חיסור בעזרת הסימן "-". למשל:

```
let foo = 2;
let bar = 8;
let answer = foo - bar;
console.log(answer);
```

שימו לב שהתוצאה כאן היא שלילית. אין בעיה עם מספרים שליליים גם בהצבה. למשל:

```
let foo = -12;
let bar = -10;
let answer = foo + bar;
console.log(answer);
```

כאן התוצאה תהיה 22-. 10- ועוד 12-.

ואפשר גם לבצע כפל בעזרת הסימן "*". למשל:

```
let foo = 2;
let bar = 8;
let answer = foo * bar;
console.log(answer);
```

התוצאה כאן תהיה 16. 2 כפול 8.

אפשר גם לבצע חילוק בעזרת הסימן "/".

```
let foo = 1;
let bar = 2;
let answer = foo / bar;
console.log(answer);
```

התוצאה כאן תהיה 0.5. ג'אווהסקריפט לא מפחד משברים עשרוניים, כמובן, ואפשר להגדיר לו גם שברים עשרוניים.

אפשר לעבוד גם עם חזקות. חזקות מסמנים בג'אווהסקריפט באמצעות "**":

```
let foo = 10;
let bar = 2;
let answer = foo ** bar;
console.log(answer);
```

התוצאה כאן תהיה 100 כמובן. 10 בחזקת 2.

בדומה לפעולות המיוחדות של מחרוזת טקסט שראיתם שאפשר להפעיל על משתנים מסוג מחרוזת טקסט, גם למשתנים מסוג מספר יש פעולות מיוחדות. למשל, פעולה המגבילה את המספר לאחר הנקודה העשרונית. אם תחלקו 10 ב-3, תקבלו 3.3333333 עד דלא ידע. אפשר להעיף את הנקודות העשרוניות. איך? באמצעות פעולה מיוחדת בשם Math.round():

```
let foo = 10;
let bar = 3;
let answer = foo / bar;
let finalAnswer = Math.round(answer);
console.log(finalAnswer);
```

הבה ננתח את קטע הקוד הזה.
בשורה הראשונה יוצרים משתנה בשם foo ומכניסים לתוכו את המספר 10.
בשורה השנייה יוצרים משתנה בשם bar ומכניסים לתוכו את המספר 3.
בשורה השלישית יוצרים משתנה בשם answer. מה הוא מכיל? foo חלקי bar, שזה בעצם 10 חלקי 3. מה התוצאה של התרגיל הזה? 3.3333333, שנמצאת כרגע במשתנה answer.
בשורה הרביעית מפעילים את הפעולה Math.round על answer ומכניסים את התוצאה למשתנה finalAnswer. הפונקציה round מעגלת את המספר. לְמה? ל-3. אם תדפיסו את finalAnswer תקבלו 3.

הינה טבלה קצרה שמסכמת את כל הפעולות הבסיסיות שאפשר לעשות:

סוג פעולה	התו שמשתמשים בו כדי לעשות את הפעולה
חיבור	+
חיסור	-
כפל	*
חילוק	/
חזקה	**

מציאת השארית

אם אתם זוכרים מתקופת בית הספר היסודי, יש דבר כזה שנקרא שארית. כמה זה שש חלקי ארבע? אחת ושארית שתיים. כלומר, החלק מהמספר השלם שהוא קטן מהמחלק. תשע חלקי שמונה זה אחת עם שארית אחת. תאמינו או לא, בתכנות יש עדנה לדברים שלומדים בכיתה ד'. אפשר לחלק ולמצוא את השארית בעזרת סימן מיוחד – **%**. הסימן הזה נקרא על ידי המתכנתים "**מודולוס**" (באנגלית modulus). אפשר לקרוא לו שארית:

```
let foo = 6;
let bar = 4;
let answer = foo % bar;
console.log(answer);
```

לוקחים משתנה foo ומציבים בתוכו את הערך 6. במשתנה bar מציבים את הערך המספרי 4. שואלים מה השארית של 6 חלקי 4. התשובה כאן היא המספר 2.

הפעולות המתמטיות שנלמדו עד כה – חיסור, חיבור, כפל, חילוק והעלאה בחזקה – נקראות אופרטורים (באנגלית operators).
אפשר להציב את תוצאות האופרטורים ישירות בתוך המשתנה. למשל, אם רוצים להציב בתוך משתנה את התוצאה של 4 כפול 2, אין צורך ליצור משתנה שיכיל 4, משתנה שיכיל 2 ומשתנה שיכיל את המכפלה של שניהם, כלומר משהו כזה:

```
let foo = 2;
let bar = 4;
let answer = foo * bar;
console.log(answer);
```

במקומו, אפשר לכתוב משהו כזה:

```
let answer = 2 * 4;
console.log(answer);
```

זה די מובן אם זוכרים שבסופו של דבר, מאחורי כל משתנה עומד מידע, בין שמדובר במספר ובין שבטקסט.

אופרטורים מקוצרים

נניח שיש משתנה שרוצים להוסיף לו רק ספרה אחת. אפשר להשתמש באופרטור מקוצר להוספה או להחסרה. האופרטור להוספה הוא **++** והוא נראה כך:

```
let answer = 4;
answer++;
console.log(answer);
```

מה יש פה? מגדירים משתנה ששמו הוא answer והערך שלו הוא 4. עם האופרטור **++** מוסיפים לו 1. ערך המשתנה החדש הוא 5. באותו אופן כמו האופרטור ++ קיים גם האופרטור – שמבצע את הפעולה ההפוכה של חיסור:

```
let answer = 10;
answer--;
answer--;
answer--;
console.log(answer);
```

כאן למשל יוצרים משתנה ומציבים בתוכו את המספר 10. באמצעות האופרטור **--** מורידים
ממנו 1. כיוון שחוזרים על התוצאה שלוש פעמים, המשתנה שווה ל-7.
הבעיה היא שזה מכוער... יש גם אופרטור קיצור שיכול להחסיר או להוסיף איזה מספר שרוצים. כך, למשל, גם הקוד הזה ידפיס 7.

```
let answer = 10;
answer -= 3;
console.log(answer);
```

גם כאן יוצרים משתנה ומציבים בו 10. בעזרת האופרטור של החיסור המקוצר מורידים ממנו 3. התשובה היא 7.

הינה טבלה חלקית של האופרטורים המקוצרים הנפוצים ביותר:

אופרטור מקוצר	משמעות
foo++	foo = foo + 1;
foo–	foo = foo - 1;
foo+=10;	foo = foo + 10;
foo-=10;	foo = foo - 10;

לסיום הפרק יש לציין תכונה חשובה של השפה – ג'אווהסקריפט היא שפה סלחנית מאוד. אם מנסים לחבר בין משתנה מסוג טקסט למשתנה מסוג מספר היא לא תעיף שגיאה אלא תמיר באופן אוטומטי את המספר לטקסט ותבצע חיבור. כך למשל:

```
let foo = 1;
let bar = '1';
let baz = foo + bar;
console.log(baz);
```

תיתן את התשובה המדהימה מחרוזת טקסט של 11. למה? כי bar הוא מסוג טקסט. אם אתם כבר מכירים שפת תכנות, זה השלב שבו אתם מתמלאים בזעם או בתדהמה. אבל כאמור ג'אווהסקריפט היא שפה סלחנית ותנסה להגיע לפשרה אם מדובר בחיבור בין טקסט למספר. בחיבור בין מספר לבין חלק מסוגי המידע האחרים, תתקבל שגיאה של **NaN**, שהיא ראשי תיבות של Not a Number.

זה נראה משונה לעין בלתי מיומנת, אבל ג'אווהסקריפט היא שפה שסוגי המשתנים בה הם **implicit** – כלומר המנוע של השפה מנסה לנחש אותם ולא ממהר לזרוק שגיאה כמו בשפות אחרות. כאמור, משתמש לא מיומן עלול להתבלבל, ולמתכנתים בשפות אחרות זה נראה כאוטי, אבל ברגע שמתרגלים - מדובר בתכונה נוחה מאוד.

תרגיל:

צרו שלושה משתנים שיכילו את המספרים 2, 4 ו-6 והציגו את תוצאת החיבור שלהם.

פתרון:

```
let foo = 2;
let bar = 4;
let baz = 6;
let answer = foo + bar + baz;
console.log(answer);
```

הסבר:

יוצרים שלושה משתנים ושמים בתוכם מספרים באופן מקוצר כפי שלמדנו. שלושת המספרים מחוברים והתוצאה נכנסת למשתנה answer, שבתורו מודפס.

תרגיל:

צרו משתנה, הציבו בתוכו את המספר 12 והדפיסו אותו. באותו משתנה הציבו מחרוזת טקסט של 12 והדפיסו אותה.

פתרון:

```
let foo = 12;
console.log(foo);
foo = '12';
console.log(foo);
```

הסבר:

כאשר מציבים מספר במשתנה, לא משתמשים בגרשיים. כאשר מציבים מחרוזת טקסט משתמשים בגרשיים. ההדפסה נעשית באותו אופן, אך כאמור, ברוב הקונסולות תראו הבדל בין שתי ההדפסות שנועד לסמן שהדפסה אחת היא מספר והאחרת היא טקסט.

חשוב: צריך לשים לב שסוג המידע של המשתנה השתנה וכדאי להימנע מלהציב באותו משתנה סוגי מידע שונים בשלבים שונים של ריצת תוכנית. זה מתכון לטעויות בהמשך.

תרגיל:

צרו משתנה והציבו בתוכו את המספר 100. הדפיסו את השורש של המספר.

פתרון:

```
let foo = 100;
let bar = 0.5;
let answer = foo ** bar;
console.log(answer);
```

הסבר:

מציבים 100 ו-0.5 בתוך משתנים. שורש, למי שלא זוכר מתמטיקה, הוא חזקת חצי. התשובה היא 100 בחזקת 0.5, ואני מתבייש בעצמי על זה שהכנסתי תחכום במתמטיקה לתרגיל כאן.

תרגיל:

צרו משתנה שיכיל את השארית של 10 חלקי 3.

פתרון:

```
let foo = 10 % 3;
console.log(foo);
```

הסבר:

על מנת לחסון במשתנים מציבים בתוך המשתנה foo את תוצאות הביטוי 10 % 3.

התוצאה היא 1.

תרגיל:

צרו משתנה, הכניסו אליו מספר ובאמצעות אופרטור מקוצר הגדילו את המספר בעוד 1.

פתרון:

```
let foo = 10;
foo++;
console.log(foo);
```

או

```
let foo = 10;
foo += 1;
console.log(foo);
```

הסבר:

האופרטורים המקוצרים מאפשרים להוסיף 1 בקלות. יוצרים משתנה ומציבים בו את המספר 10. השתמשתם באופרטור המקוצר ++ להוספת 1 בדיוק למשתנה הזה. אפשר להשתמש באופרטור המקוצר 1=+ כדי להוסיף 1.