**Full stack day 1 - 27/7/2022**

מבחנים עושים רק בכיתה

להגשת פרוייקט הסיום יש חודשיים

תאריך סיום הקורס עלול להדחות במקרים שבהם יצטרכו לבטל מפגש בגלל אילוצים

מפגשי go online - כל תלמיד מקבל רבע שעה. כל הקודם זוכה. תרגול עם מתרגל.

זכאים לתעודת סטודנט 100 שח כנראה

דור היא רכזת הקורס

מורן

גיא ניסים - מנוי לחניה. פרטים באתר, לשונית ברוכים הבאים יש שם טלפון ומייל שלו.

תנאים לתעודה: 85% נוכחות.

Ipv 4 - ip address

שולחים בקשה

Internet backbone - הכבל שמחוברים אליו כל שרתי האינטרנט שמספקים את התשתית, בזק וכו׳. הם יודעים לשלוח בקשה דרכם ל-backbone וחזרה לשרת. השרת קיבל את הבקשה, מעבד אותה ושולח תשובה

כלומר יש request ו-response.

ב-dev tool אפשר לראות ב-remote address את ה-ip של האתר ואם נזין את ה-ip ישירות ככתובת זה אמור להביא לאותו אתר. אם לא עבד, אפשר ב-cmd לחפש בפקודה של tracert ynet.co.il או ב-nslookup.

Http - hypertext transfer protocol - פרוטוקול שמדברים איתו שני מחשבים, צד השרת וצד הדפדפן, והוא אומר שעכשיו ישלח מידע בפורטוקול http. המרה של כל המידע לטקסט. איך הוא עושה את זה? באמצעות unicode שבו לכל תו מורכב מאותיות, מספרים וסימנים.

Https - פרוטוקול עם שכבת אבטחה נוספת

Www - world wide web

הדפדפן ממיר את המידע שהוא מקבל לטקסט.

UDP - דגש על מהירות, פרוטוקול מהיר, מעביר מידע בלי לוודא שהגיע כמו שצריך, שהכל עבר, בלי לעשות retry שבודק שוב.

TCP - דגש על אמינות, פרוטוקול איטי יותר, אבל יותר מוודא את הגעת הדאטה בשלמותו

לרוב המכשירים יש IP מתחלף. לשרת יש IP קבוע כי אחרת לא נדע לאן לשלוח את הבקשות.

דומיין הוא השם שמפנה לכתובת אינטרנט (ip).

סימן שאלה (?) אומר ל-url איפה מתחילים הפרמטרים וזה כבר לא חלק מה-path. באמצעות encoder/decoder אפשר להפוך את הפרמטרים. אם היינו משתמשים רק בסלאשים, עדיין היה אפשר לחשוב שזה חלק מה-path.

URI (uniform resource identifier) - מזהה מתי מתחיל/נגמר פרמטר.

XML - סוג של HTML, העברת מידע באמצעות תגים. כל שפה שמשתמשת בתגיות נקראת markup language

**Full stack 2 - 31/07/2022 - HTML part 1**

תחילת שיעור - חזרה על אינטרנט, ממה מורכב url - לא מוקלט

Decoding url - המרה מג׳יבריש למשהו שהמשתמש יכול להבין בחלק של הפרמטרים של ה-url.

Intro to HTML

HTML הוא השכבה הראשונה של הבית, הבנייה עצמה. זו לא שפת תכנות אלא markup, תגים. המבנה עצמו, בלי עיצוב בלי פונקציונליות.

ברגע שמכניסים שם אתר example.com אין צורך להכניס פורט ואינדקס כי זה הדיפולט.

כדי לפתוח dev tool לוחצים כפתור ימני ו-inspect (cmd+i). נגיד ב-dev tool של גוגל ב-network אפשר לקחת את התמונה היומית.

אפשר בכפתור ימני לעשות view page source. מקבלים את קוד המקור של JS כדי לקחת פרמטרים מהבראוזר, מהיוזר, שפה.

מדגים איך מסמן את כפתור ״כניסה״ ומשנה אותו לשם שלו. אם עושה ריפרש, הוא מוציא בקשה חדשה ולכן זה לא נשמר.

אם נפתח קובץ ב-text editor, נקרא לו html בסיומת הוא ייטען בתור כזה.

Html התחיל כי אנשים רצו לשלוח אחד לשני טקסטים, קצת קבצים וכו׳. לא הרבה אנימציה וכו׳.

18:19 - לבדוק מה זה מחולל HTML.

<!DOCTYPE html> - מכריז על סוג הקובץ

בתוך ה-head יהיה לנו title שאחראי בעיקר על שם העמוד.

כל תג שמכיל דברים חייב פתיחה וסגירה. תג שלא מכיל דברים, לא חייב סגירה.

18:32 איך עשה רווח מצא ב-stack overflow

<hr/> horizontal row - סוגר את עצמו, לא מכיל כלום.

<body> מכיל את כל הטקסט של האתר. אחריו לא אמור להיות שום דבר.

<body style> איזה סוג סטייל רוצים

<body style=”color: red; background-color:yellow”;text-align:”center”; title= “body”

הטייטל במקרה הזה יתן tooltip כשעומדים על הטקסט שבו כתוב "body״.

<!-- noooote--> כתיבת הערה

<h1> </h1> - נותן גודל פונט של כותרת

<h2> ייתן כמו כותרת משנית

בתוך ה-inspect יש כפתור inspect element ועומדים על האלמנט והוא מראה את התכונות שלו. שם אפשר לראות את התכונות הדיפולטיביות של h1, h2 וכו׳.

תגית <p> היא של פסקה paragraph.

תגית <span> דברים שקצרים יותר מפסקה. אחד ההבדלים בין p לבין span הוא שכשסוגרים p הוא נותן שורה חדשה באתר. ב-span הוא מצמיד גם אם נתנו כמה.

ישנן כמה צורות להתייחס לצבעים css color - או שמות הצבע או hexdecimal או RGB.

לבדוק התחלה אחרי הפסקה

Div - מאגד כמה אובייקטים לרוב קשורים אחד לשני. אם למשל יש body בצבע צהוב, div בצבע כחול ו-p באדום, אז החלק הכי ספציפי יקבע. נגיד ה-body יהיה צהוב חוץ מה-div שיהיה כחול וחוץ מה-p שיהיה אדום.

Br - break line

אם רוצים לתת לינק משתמשים ב-<a>. משתמשים ב-attribute שונים.

<a href="<https://www.google.com/>" target=”\_blank”>

Google

</a>

הוספת טקסט רנדומלי - lorem ipsum.

<p lorem50>

הוספת תמונה:

<img src=״blabla.jpg” alt=”error” style=”width:100px; height: 100px”>

ה-alt נותן טקסט למקרה שאי אפשר לטעון את התמונה. ה-width יתאים לגודל הרצוי.

Img היא תגית שלא צריך לסגור.

פונטים - שימוש במשפחת פונטים למקרה שהדפדפן לא מזהה את הפונט/אין לו אותו. ממליץ לכתוב בסוף את sans-serif כי תמיד יש אותו בכל דפדפן.

<body style=”font-family: Arial, Helvetica, sans-serif;”>

**Visual code**

Shortcut shift+! - נותן את כל הטמפלט

**Full stack 3 - 03/08/2022 - HTML part 2**

אם רוצים למרכז משהו אפשר להקיף אותו (את ה-span או ה-footer וכו׳) בתגיות <center>.

Navigation bar

אם רוצים ליצור לינקים. הוא יצר כמה עמודים של html ושמר בתיקייה. בקובץ שעובד עליו, יצר קישור של לינקים באמצעות <a> נפרד לכל טאב שרוצה ליצור.

אם רוצים לינק למקום אחר **באותו עמוד**, להקפיץ למטה בעמוד למשל, אז מייצרים div שונים עבור כל מקום שרוצים להקפיץ אליו. יוצרים ל-div הרלוונטי id ומשתמשים ב# כדי להקפיץ אליו.

<a href=”#blogs”>Blogs</a>&nbsp

<div id=”blogs”>

אם רוצים להקפיץ לאיזור מסוים בתוך עמוד אחר:

<a href=”sections.html#blogs”>Blogs</a>&nbsp

אם רוצים שבמקום טקסט Blogs יהיה אייקון, נכניס במקום טקסט תגית של תמונה.

אם רוצים לתת מיקום של תמונה/קובץ לוקאלי על המחשב, אפשר לתת או את שם הקובץ בתוך התיקיה בה נמצא גם ה-html או לתת את ה-path המלא או להשתמש ב/.. כדי לעלות תיקיה למעלה.

אם ההיררכיה היא

Folder a  
 Img1

Folder B

Img2

Folder C

Img3

my.html

קובץ ה-html נמצא בתיקיה C. אם רוצים לקחת את img 2 נשתמש כך

../Img2

אם נרצה להשתמש ב-img1 נעלה שתי תיקיות למעלה

../../Img1

<ul> - unorder list

<li> Milk </li>

<li> Bread </li>

<li> Tomato </li>

</ul>

נקבל רשימה עם bullets. אם נשתמש ב-<ol> שזה ordered list, נקבל אותה רשימה עם מספרים.

כדי להכניס להערה, מסמנים ועושים ?+ctrl

**טבלאות**

Tr - table row

Th - table header - מדגיש טקסט בטבלה

Td - table data

Caption - title for table

Bgcolor - צבע לתא מסוים

Colgroup - מאפשר לתת הגדרת קבוצה לטורים שמתחתיו

<table width=”600” border=”1” align=”center” cellpadding=”10”>

<caption>title for table</caption>

<colgroup>

<col width=”50”>

<col width=”100”>

<col width=”50”>

<tr>

<th bgcolor=”yellow”>Name</th>

<th>Last name</th>

</tr>

<tr>

<td>Ann</td>

<td>Cohen</td>

</tr>

</table>

**Full stack 4 - 10/08/2022 - HTML part 3**

**Nested table (from home work 3)**

<table>

<tr>

<td>

<table>

<tr>

<td> css </td>

<td> bla </td>

<td>

</td>

</td>

</tr>

</table>

Input is a way to get input from the user.

<h1>Contact Me</hr>

<form action=”index.html” method=”post”>

<label for=”name”>First Name:<label><br/>

<input type=”text” name=”fname” id=”name”/>

<label for=”last\_name”>Last Name:<label><br/>

<input type=”text” name=”lname” id=”last\_name”/>

<input type=”submit”>

</form>

For - is same as id in the input.

Name - is the key that is being sent for this input

צריך להגיד לו לאן לשלוח את הדאטה שמתקבל מהמשתמש. Action בתוך תגית form אומר לו מה הפעולה שצריך לעשות לדאטה שהתקבל. אפשר לשלוח ב-action גם לעשות <mailto:ann@gmail.com> ואז הוא הולך לתוכנת האימייל שמוגדרת כברירת מחדל של הדפדפן או המחשב.

ב-method: שתי מתודות של שליחת נתונים. מתודת get ומתודת post. מתודת get שולחת את הנתונים על גבי ה-url. ה-post שולחת בתוך ההודעה.

ולידציה - אם רוצים שהמשתמש לא יוכל לשלוח פחות משני תווים בכל תא, נשתמש בattribute שהוא אורך מינימלי minlength. אותו דבר יש maxlength וגם לכתוב required בלי תיאור שלו וזה מכריח שהשדה ימולא לפני שליחת הטופס.

<input type=”text” name=”fname” id=”name” minlength=”2” maxlength=”20” required/>

<label for=”phone”>Phone number::<label><br/>

<input type=”tel” name=”phone” id=”phone” required/>

יש attribute שהוא autocomplete="off". אם מוסיפים אותו בתגית של formֿ, כל הטופס יהפוך להיות בלי השלמה אוטומטית. אם נוסיף את זה רק לinput מסוים, אז רק אותו input לא יהיה יותר בהשלמה אוטומטית.

אם ה-type של האינפוט הוא email, אוטומטית חייב שיכיל @.

אם רוצים להכניס password בתור type זה גורם לזה שהטקסט שמכניסים הוא בנקודות ולא רואים מה המשתמש כתב. כדי לראות את זה בעמוד שהתקבל, מחליפים את ה-type מ-password לסוג text. זה למעשה מה שעושה העין שיש כשרוצים לראות מה הסיסמה שלנו ששמורה מראש.

סוג של checkbox יקבל attribute של value.

<input type=”checkbox” name=”practice1” value=”work”/><label>Work</label>

אפשר להוסיף ל-checkbox גם attribute שהוא disabled ולא מאפשר ללחוץ על ה-checkbox המסוים.

אפשרות לבחור כמו multichoice נקרא radio.

<input type=”radio” name=”gender1” id=”gender1” value=”male”/><label for=”gender1”>Male</label>

<input type=”radio” name=”gender2” id=”gender2” value=”female”/><label for=”gender2”>Female</label>

\*\*עורכת אחרי שיעורי בית. עדיף לדעתי להשתמש באותו name כי זה הולך לאותו שדה בדאטה. בנוסף, אם רוצים להכריח לבחור אופציה אחת - חייבים שהname יהיה זהה ולא name1, name2.

**לבדוק מה זה title לinput**

Placeholder - מאפשר לתת דוגמה לשדה שרוצים למשל באימייל אפשר שיהיה באפור [example@gmail.com](mailto:example@gmail.com)

אפשר לתת גם שדה דיפולטיבי שימולא אוטומטית שהוא באמצעות value.

אפשר לאפס את ה-form ע״י input type="reset".

<select name=”city” id=”city >

<optgroup>Center</optgroup> \*option group

<option value=”tlv”>Tel Aviv<option/>

<option value=”lod”>Lod<option/>

</select>

<div>

<h3>Dream city</h3>

<select name=”city” id=”city multiple> שימוש ברבים מאפשר כמה בחירות

<optgroup label=”Center”> \*option group

<option value=”tlv”>Tel Aviv<option/>

<option value=”lod”>Lod<option/>

</optgroup>

<optgroup label=”North”> \*option group

<option value=”hifa”>Hifa<option/>

<option value=”akko”>Akko<option/>

</optgroup>

</select>

</div>

אם רוצים לנסות ולא ב-VS אפשר לפתוח playground כמו codepen.io/pen

סוג אינפוט date שהוא רק תאריך לעומת datetime שהוא תאריך ושעה

<label>Date:</label>

<input type=”datetime” id=”date” name=”dateTime” value=”2022-08-10 19:53” hidden>

אם רוצים שהמשתמש לא ימלא בעצמו אבל כן לקבל את התאריך והשעה של מילוי הטופס, מוסיפים value שהוא הדיפולטיבי ועושים לו hidden ואז המשתמש שולח את זה בלי לדעת ששלח.

ברגע שמוסיפים type="file" הוא מאפשר הוספת קובץ.

text area - אזור ייעודי לכתיבת טקסט ארוך (למשל הודעה, תלונה, חוות דעת וכו׳)

<textarea name="message" cols="40" rows="10"></textarea>

**Iframe** - למעשה פותח חלון בתוך החלון הקיים.

<h1>iframe</h1>

<iframe src=”index.html” width=”600” height=”600”></iframe>

אם נוסיף frameborder=”none” אז המסגרת הדיפולטית תעלם וכביכול לא רואים שזה חלון בתוך חלון.

אם רוצים להכניס סרטון מיוטיוב, צריך לשנות ב-url את ה-watch ל-embed.

אפשר להוסיף ל-iframe גם name="frame\_1" ואז להוסיף בעמוד המקורי שלנו

<iframe src=”iframe.html” name=”frame\_1” width=”600” height=”600”></iframe>

<a href=”<http://www.youtube.com/embed/XXXXX>” target=”frame\_1”>Click here</a>

**להשלים emulation מאפשר לנו לראות כמו מראה של מובייל, scale**

אתר wayback machine הקליט איך אתרים היו נראים במשך השנים.

**Full stack 5 - 14/08/2022 - CSS part 1**

דרך inspec → network → payload אנחנו רואים את כל מה שנשלח דרך ה-http של הטופס.

<sub> תגית שמאפשר לכתוב בכתב קטן יותר.

\*\*כתובת ip של 127.1.1.0 היא כמו local host.

***Cascading Style Sheets***

יש שתי דרכים להוסיף css. ראשונה היא inline.

איפה בודקים?

בתוך dev tool הולכים ל-elements ושם ל-styles.

הוא מכניס תגית **בתוך ה-head** של CSS שתחול על כל המסמך.

<style>

H1 {

Font-size: 12px;

}

</style>

זה יוצר חוק rule שיפעל מבלי שיהיה צורך לשנות את ה-style של כל תגית ב-body.

אפשרות אחרת היא בתוך ה-head להכניס קובץ חיצוני:

<link rel=”stylesheet” href=”styles.css”>

אנחנו מכריזים שהקשר בין העמוד שלנו ל-href הוא שזה הקשר שלהם (rel=relation).

כמו שהייתה לנו תיקיית images, אפשר להוסיף תיקיית styles. את שתי התיקיות עדיף להכניס לתיקיה בשם assets.

הוספה של קובץ חיצוני נקרא external css לעומת לכתוב ב-head שזה אומר internal css.

מכל ה-rules מי שמגיע אחרון הוא האחרון שישאר וידרוס את השאר. Inline דורס יותר חזק מכולם. אם רוצים להוסיף משהו בכל זאת שהוא לא inline, אפשר לכתוב

!important

בתוך התגית של ה-css ואז הוא דורס את הרולים האחרים.

לרוב הקובץ link יבוא ראשון וה-style (בתוך head) יבוא אחריו. כי לינק יהיה להרבה עמודים לעומת ה-style שיכול להיות יותר ספציפי לעמוד.

איך נראה קובץ חיצוני? למשל

h1 {

font-family: Arial, Helvetica, sans-serif;

font-size: 30px;

}

דרך נוספת אך פחות שימושית היא import. מייבאים מתוך קובץ אבל ל-internal:

<style>

@import url(“assets/styles/styles.coo”);

אפשר לטרגט אלמנט ספציפי. למשל

<style>

H1 {

Background-color: blue;

Fone-weight: 100;

}

Label {

Color: red;

}

</style>

הבעיה היא שככה אוטומטית קבענו חוקים לכל לייבלים. איך אפשר להיות יותר ספציפיים? Id.

בתוך ה-label נוסיף לו id ואז

<label id=”labelFname”...>

וב-head נוסיף

#labelFname{

Color: aqua;

}

Label {

Color: red;

}

במקרה הזה, אמנם ה-label הרגיל הוא האחרון, אבל מאחר והלייבל המטורגט הוא **יותר ספציפי**, הוא זה שייקבע ולכן הצבע של הלייבל יהיה aqua ולא אדום.

מה זה class? הסימן המוסכם שלו הוא נקודה (.).

בתוך ה-head כותבים:

.bg-input{

Background-color: chocolate;

Color: aqua;

}

.color-input{

color: white;

}

<body>

<input class=”bg-input color-input”...>

<input class=”bg-input”...>

האינפוט הראשון יהיה ברקע שוקולד ובצבע לבן כי שרשר את שני הקלאסים! האינפוט השני יהיה ברקע שוקולד ובצבע אקווה כי ירש רק את הקלאס הראשון.

מה יקרה אם לאינפוט הראשון נהפוך את הסדר?

<input class=”color-input bg-input”...>

התשובה היא שזה לא משנה הסדר שבו כותבים את זה בתגית אלא הסדר שבו נכתבו הקלאסים ב-head.

אם רוצים לטרגט קלאס באמצעות קובץ אנחנו מכניסים link בתוך ה-head (לא ב-style אלא ישירות בתוך ה-head כמו כל קובץ css). הוא מוריד את הקובץ באופן לוקאלי לדפדפן. לכן אם בקובץ כתוב

.color{

Background-color:”green”

}

ואז טענו אותו, אפשר יהיה ב-body לטרגט בתוך האלמנטים למשל

<label for=”lname” class=”color”>Last Name</label>

הערות ב-css כותבים

/\* comment \*/

זה שונה בכוונה מ-html כדי להבדיל בין השפות ושם זה

<!--comment -->

האם אפשר לטרגט פעם עם קלאס פעם עם id ופעם עם אלמנט? כן. אבל מי שקובע הוא תמיד הספציפי יותר. Id הוא יותר ספציפי מ-class.

Head

#lastName {

Margin: 10px;

}

Body

<div class=”bg-input” id=”lastName”



מה זה display: block? זה אומר שהגדרת css תופסת את כל רוחב העמוד. הבלוק הוא כל מה שבסגול, ה-input מוגדר בירוק.

אם נשנה ונגדיר display: inline החוקים יחולו רק על מה שיש לו בתוך ה-p או ה-div וכו׳.

#lastName {

Margin: 10px;

Display: inline;

}

אפשר להשתמש ב-display: block בתוך קובץ css כדי להגיד שאלמנט מסוים רוצה להיות תמיד בשורה (בלוק) משל עצמו.

Input{

Display: block;}

זה אומר שכל אינפוט יהיה בשורה משל עצמו גם בלי להגדיר <br>.

איך ליצור קופסאות

Div {

width:300px;

height: 300px;

Font-size:1em;

Text-align: center;

Padding: 10px;

Border-width: 20px;

Border-style: dotted;

Margin: 10px;

}

#div1 {

Background-color: green

}

#div2 {

Background-color: blue

}

הרווח הפנימי בין הקופסה עם הצבע לבין ה-border נקרא padding.

הרווח החיצוני בין ה-border לאלמנטים שסביבו נקרא margin. אם לא נגיד של איזה כיוון ה-margin למשל margin-top, אז יתקבל margi בכל הצדדים.

אם ניתן margin: auto - זה ימרכז את האלמנט **רק אם האלמנט תופס את כל השורה**, כלומר כ-blcok.

**Full stack 6 - 17/08/2022 - CSS part 2**

שינוי כיוון ימין לשמאל direction:"rtl".

Favicon - האייקון שמופיע בכותרת של העמוד בטאב, לא בעמוד עצמו.

Favicon.cc - אתר שאפשר להכין בעצמך

שומרים כ-favicon.ico

משתמשים בקובץ html בתוך ה-head באופן הבא:

Link rel=”icon” href=”assets/images/favicon.ico”

שימוש ב-opacity=0.3 הופך את הצבע ליותר שקוף עבור ה-background-color או background-image.

מה זה overflow? מגדיר איך הטקסט מתנהג בתוך קופסה מסוימת. למשל אפשר overflow: scroll ואז גוללים בתוך הגודל של האייטם שהגדרנו.

כמו שדיברנו בשיעור שעבר, padding הוא שכבה שקופה שמפרידה בין האייטם (שמכיל את הטקסט) לבין ה-border.

יש כמה אלמנטים שמקבלים הגדרה של display:block מוגדרת מראש. למשל: div,p, h1. אם רוצים ש-box שהכנסנו באמצעות div למשל, יהיו אחד ליד השני, צריך להגדיר display: inline. כשניסינו גילינו שיש בעיה. מסתבר ש-inline אז האלמנט לא יכול לקבל height, width אלא מקבל רק את הפרמטרים של הטקסט עצמו.

כשניסינו להוסיף צבע לרקע כשיש עליו div שונים בצבעים שונים, גילינו שמסתבר שבהגדרה הדיפולטית, ל-body יש margin מסוים. לכן ה-div לא צמודים לשמאל. אם רוצים להצמיד אותם, צריך להגדיר ל-body שיהיה לו margin=0.

מה זה line height? גובה של האלמנט מעבר לגובה הטקסט. הוא הולך ביחידות של 1.5 או 2

**Position**

ישנם שני מושגים רלוונטים position relative, position absolute.

אם רוצים שאלמנט יזוז באופן **יחסי** לאיפה שהוא היה, אנחנו נותנים לו.

Position: relative

Top: 600px

Z-index: 1 - זה המדד שקובע את הסידור על ציר העומק, כלומר מי קדימה ומי אחורה.

היחס הוא ביחס לתגית בה נמצא. האם ב-body או בתוך div או אחרים. כלומר תלוי מי ה-parent שלו. במצב כזה, המיקום המקורי של האלמנט נשמר בזכרון ולכן אם נכניס עוד אלמנט אחריו, הוא יבוא אחרי המקום המקורי של האלמנט שהזזנו באופן רלטיבי.

אם משתמשים ב-position: absolute, המקום המקורי של האלמנט לא נשמר. לכן אם נוסיף אלמנט נוסף אחריו, הוא ימוקם אחרי האלמנט שהיה לפני האלמנט שהוזז אבסולוטית.

<div class=”green”>

<div class=”blue” </div>

</div>

נותנים ל-blue מיקום אבסולוטי והוא יהיה בתוך אבסולוטי לריבוע הירוק ולא ביחס לכל העמוד.

אם עושים position: fixed אז האלמנט יוצא מסדר האלמנטים מי בא אחרי מי.

אם רוצים שאלמנט יהיה קבוע באותו מיקום גם כשגוללים בעמוד נשתמש ב-position: sticky.

\*דיבר על מרכוז אלמנטים רבע שעה לסוף השיעור.

**Full stack 7 - 21/08/2022 - CSS part 3**

שיעורי בית

Body {

Text-align: center;

Margin:0;

}

A {

Text-decoration: none;

Color: black;

}

.navbar {

Background-color: #5585b5;

Height: 60px;

}

.logo {

Float: left;

Height: 60px;

Width: 200px;

{

.logo img {

Width: 50px;

Height: 50px;

Border-radius: 50%;

Float: left; → כדי שהוא יהיה יותר שמאל מהכותרת

Padding: 5px;

}

.logo h1 {

Margin: 0:

Display: inline-block; → בגלל שהפלואט משפיע על מה שניתן בדיב אב שמכיל אותם

Float: right; → כדי שיהיה הימני יותר מבין האלמנטים שבתוך הלוגו?

}

.nav-bar ul {

Float: right;

Margin-right: 10px;

}

.navbar li {

Display: inline-block;

Margin: 5px 10px;

}

דרך אחרת הייתה להשתמש בתוך ה-h1 בכזה:

Background-inage: url(“../images/profile\_yos.jpeg”);

Background-size: 50px;

.top-container {

Background-color:#bbe4e9;

Height: calc(100vh - 60px); → vh gives height according to viewport (what i see on the screen according to screen size). Decreasing 60px is for the navbar.

}

אם עושים לאלמנט position: fixed אז הוא לא חלק מהספירה של של ה-viewport.

.top-container h1 {

Width: 30%;

Margin: auto;

Padding: 100px;

}

#flyingBird {

Width: 15%;

Position: absolute; → אבסולוט אומר איפה למקם באופן אבסולוטי ביחס לאבא שלו. רלטיבי ביחס למיקום שהיה בו

Right: 150px;

}

.bird-tree-container {

Position: relative;

Width: 60%;

Margin: auto;

}

#doveBird {

width: 15%;

Position: relative;

Bottom: 250px;

}

אפשר להגדיר media query - להגדיר לכל גודל מסך, יהיו חוקים שונים של תצוגה.

מה שקרה היה שכשמקטינים את המסך, הציפור נעלמה.

מפעילים את האימולטור ורואים באיזה גודל האתר מתחיל להישבר.

#tree {

width : 60%;

}

@media only screen and (max-width 1027px)

#doveBird {

Position: absolute;;

Left: -57px;

Bottom: 150px;

}

.bird-tree-container {

Position: relative;

Width: 60%;

Margin: auto;

}

יחידת rem - הוא root. מסתכל על כל המסמך ולא על האבא כמו ה-em שהוא גודל פונט רלטיבי לאבא של האלמנט.

לפי מה מחליטים אם להשתמש ב-margin או ב-padding? צבע רקע מגיע עד ה-border וה-margin הוא אחרי ה-border. לכן הצבע לא יכול על margin ולכן זה מסובך יותר מאשר padding.

מה זה nth-child?

מאפשר לבנות רולים לפי מספר ילד זוגי או אי זוגי. למשל יש לנו פסקאות ורוצים שאחת תהיה מימין והבאה משמאל וחוזר כך. נעשה

#skills img:nth-child(odd) {

Float: left

}

#skills img:nth-child(even) {

Float: right

}

אפשר לייצר כפתור ע״ button generator ואז להעתיק את הקוד אלינו.

Background-image: linear-gradient(to right red,yellow)

בשביל רספונסיביות נרצה להגדיר דברים למובייל (עד 720). לאחר מכן מוסיפים media query למשל מ-720 עד 1024.

@media only screen and (min-width: 720px) and (max-width: 1023px) {

.menu {

Width: 20%;

}

.right-rail {

Width: 20%;

}

@media only screen and (min-width: 1024px) {

.menu {

Width: 30%;

}

שימוש ב-display: flex או ב-display: inline flex. מאפשר לאלמנטים להיות אחד ליד השני ויכולים להגיד לו איך רוצים שהם יהיו, באיזה כיוון, האם כל אחד תופס שורה או שהם אחד ליד השני.

אפשר להשתמש ב-flex-grow שמגדיר לנו את היחס של כל אלמנט ליד האלמנטים האחרים ומגדיר מה הגודל של כל אחד

.one {

Flex-grow: 4;

}

.two {

Flex-grow: 2;

}

את three כבר לא צריך להגדיר כי הוא ישלים לרוחב שהגדרנו לעמוד או הרוחב הדיפולטיבי.

אפשר לתת סדר לאלמנטים ע״י order: number. פתר בעיה של float שגרם לדברים מסביבו לזוז.

זה נועד כדי לסדר מיקום לילדים של container מסוים.

צורה קלה לסדר navbar היא ע״י Justify-content: space-between.

כל flex יכול להיות אבא של flex אחר.

**Full stack 8 - Bootstrap - 24/8/2022**

איך עובדים עם flex?

<https://savvy.co.il/blog/wordpress-design/complete-guide-to-flexbox/>

**Bootstrap**

סוג של מערכת עיצובים מוכנים מראש.

באתר שלהם יש quickstart.

כדי לייב נכניס ב-head את

link rel="stylesheet" href:bootstrap.css"

Button class=”btn btn\_1”

אפשר לשחק עם הבוטסטראפ ב-codeply.

בבוטסטראפ יש הרבה דוגמאות שאפשר להשתמש בהן ב-examples.

בוטסטראפ מפחית את הצורך בשימוש ב-media queries ומאפשר שימוש ב-grid, כלומר כמה אייטמים בשורה. נגיד במובייל 3 אבל במחשב 5.

אתר בשם ui patterns מראה דוגמאות איך לעשות גלריה.

Mock up - איך יראה מה שנבנה כשזה רק מצויר, שום דבר לא פעיל.

באתר dribbble אפשר למצוא כל מיני דוגמאות להציג מוקאפים.

אתר נוסף sneakpeekit שמאפשר להציג כל אתר במובייל/טאבלט/מחשב וכו׳.

עוד כלי בתשלום פרט לניסיון, הוא balsamiq cloud.

חסר שעתיים!

**Full stack 9 - intro to programming (flowcharts) 1 - 28/8/2022**

תוסף לכרום שיעזור לנו הוא gliffy. מאפשר לבנות תרשימי זרימה.

start / end - eliptical

Input - diamond/parallelogram

If - rotated square

compute/calc - rectangle

Output - rectangle with rounded line

ישנם שני סוגי לולאות - for, while

מודולו modulu שנותן לנו שארית של חילוק, עוזר לנו להבדיל בין מספר זוגי לאי זוגי.

**Full stack 10 - intro to programming (flowcharts) 2 - 31/8/2022**

תנאי מקונן - תנאי בתוך תנאי.

**Full stack 11 - intro to programming (flowcharts) 3 - 4/9/2022**

מערכים הן שורות שמכילות ערכים מסוימים. יכול להיות שורה אחת, יכול להיות כמה עמודות שמכילות כמה שורות.

מערך מתחיל תמיד באפס.

**JavaScript 1**

שימוש בפעם הראשונה, באחת הגרסאות של נטסקייפ, הוציאו גרסה שלא היה צריך לעשות קומפילציה כלומר לפרשן מטקסט ל-0,1 כי הדפדפן כבר ידע לקרוא את זה.

אין קשר בין javascript ל-java. Java היא שפת סקריפטים.

Object oriented programming - JS היא פחות שפה כזו, כלומר לא צריך לבנות אובייקט אחד שהוא ישייך אליו אובייקטים.

אפשר להשתמש גם ב-codepen בנוסף ל-codeply שדיברנו עליו לכתיבה של js.

בתוך ה-body ניתן תגית script שבה נוכל לכתוב.

console.log(“Hello world”); - יכתוב בקונסול

document.write(“Hello world”); - יכתוב בדף html עצמו

השפה מדברת באובייקטים - אפשר להגדיר סוג של משהו מסוים, לתת לו תכונות ולתת לו לעשות פעולות. יכול גם להזהיר אותי מדברים.

נניח יש לנו מכונית. מה מאפיין אותה?

* תכונות - היא בצבע ירוק, יש לה 3 דלתות, מנוע 1200, דלק 95. אפשר לשנות את התכונות ממכונית למכונית אבל סוג התכונות יהיה קיים בכולן.
* פעולות - היא יכולה להדליק אורות, לנסוע קדימה ואחורה
* התרעות (אינדיקציות) - כל פעם שהדלק נגמר, כל פעם שמתקרבים לרכב אחר.

יצירת משתנה חדש var.

נניח הגדרנו פונקציה שיצירה של סטודנט מצריכה שם פרטי, שם משפחה וגיל. בכל פעם שניצור משתנה חדש של שם סטודנט, הוא יצפה לקבל את שלושת הערכים האלו.

Function Student(name, lastName, age) {

This.name = name;

this.lastName = lastName;

This.age = age;

};

Var oz = new Student (“Oz”, “B”, 32)

בכל פעם שנבקש את oz נקבל את כל הפרטים האלו. אפשר להשתמש גם רק בחלק מהתכונות

Oz.name + “ “ + oz.lastName + “ his age: “ + oz.age

Oz B his age: 32

ב-JS יש סוגי data types (שלושת הראשונים הם הכי בסיסיים ופרימיטיביים):

* מספרים
* אותיות
* בוליאן
* מערך שיכול להכיל הרבה תאים - יכול להכיל אותיות, מספרים, אובייקטים, מערכים נוספים
* אובייקט - לדוגמה מסוג string שהוא מערך אותיות

JS כבר נתנה דברים לחסוך עבודה. למשל יש אובייקט שנותן שעה, תאריך וכו׳.

אובייקט מסוג סטרינג - כל פעם שרוצים לכתוב מילה ולחבר כמה אותיות, נצטרך להשתמש בו. אם למשל

String {‘welcome’} - יכיל במקום ה-0 את האות w ובמקום השני את האות 2 ובמרום השלישי את האות l.

למעשה בתוך סטרינג, JS אחסן מערך של אותיות ["w", "e", "l", "c", "o", "m"}.

welcome.indexOf(“e”} - ייתן את המקום במערך בו אוחסנה האות e בפעם הראשונה

פירוק כזה יאפשר הרבה פעולות למשל replace

example.replace(“a”, “e”)

Welcome

example.replaceAll(“a”, “e”)

אם רוצים לאתחל אובייקט צריך להשתמש במילה new ולתת לו מספר פרמטרים דרושים.

Var welcome = new String(“welcome”);

פונקציות שאפשר להפעיל על סטרינג למשל

“Hello world”.toUpperCase()

* .toUpperCase - הופך את כל האותיות לגדולות
* .toLowerCase - הופך את כל האותיות לקטנות
* .substring(0,1) - נותן את הטווח של התוים בין האינדקס הראשון עד אחד לפני האינדקס השני
* .charAt(8) - נותן את התו במקום של האינדקס
* .concat(var, string…) - name.concat(lastname) - AnnNahir
* .replace(“h”,”H”)
* .split(“”) - מחלק את הסטרינג למערך של תאים. אם ניקח מילה, בכל תא תהיה אות אחת. החלוקה היא לפי התו שמופיע בסוגריים. במקרה שבדוגמה החלוקה תהיה בין כל אחד מהתווים. אפשר לרצות לחלק לפי פסיק או נקודה או אות מסוימת או מספר וכו.  
  Arr = input.split(“”)

Arr[0] = arr[0].toUpperCase()

* arr.join(“”) - יחבר תאים שונים במערך לתא אחד
* .trim - מוריד רווחים לפני ואחרי.

‘ Yos Israel ‘.trim  
‘Yos Israel’

* .length - נותן אורך של מה שובדקים עבורו

Concatenation - חיבור משתנים, סטרינגים,

מערכים array

Var arr = new Array(100) **OR** var arr = []

arr.push(1)

arr.push(‘Y’)

arr.push({}) - מסוג אובייקט

arr.push(new Student(“Yos”, “Isreal”, 37))

Arr[4] = 123

Arr

- (5) [1, ‘Y’, {}, student, 123]

for(i=0, i<arr.length, i++){

console.log(arr[i]);

}

Var classSits = []

classSits[0] = “Alex”

classSits[1] = “Dima”

classSits.push(“Katya”)

classSits.push(“Or”)

classSits

- (4) [‘Alex’, ‘Dima’, ‘Katya’, ‘Or’]

classSits.includes(‘Yos’) - בודק האם נמצא במערך

- false

classSits.indexOf(‘Yos’) - מחפש באיזה מקום במערך נמצא האובייקט. אם לא נמצא מחזיר מינוס 1

- -1

classSits.indexOf(‘Alex)

- 0

classSits.concat([“Yehuda”])

- (5) [‘Alex’, ‘Dima’, ‘Katya’, ‘Or’, ‘Yehuda’]

classSits = classSits.concat([“Yehuda”, “Amir”])

- (6) [‘Alex’, ‘Dima’, ‘Katya’, ‘Or’, ‘Yehuda’, ‘Amir’]

classSits.forEach(name => console.log(name)) - תעבור על כל אחד מהאיברים ותעשה איתו משהו

- Alex  
- Dima

- Katya  
- Or  
- Yehuda  
- Amir

אובייקט מסוג date

Var time = new Date()

Time

- Sun Sep 04 2022 21:19:37 GMT+0300 (Israel Daylight Time)

בכל פעם שנכתוב כרגע time נקבל את אותה השעה שבה יצרנו את המשתנה time.

time.getDate - מחזיר את מספר היום בחודש

- 4

time.toLocaleString()

- ‘9/4/2022, 9:20:29 PM’

time.getFullYear()

- 2022

אובייקט מסוג math (אם נכתוב math ונעשה enter נקבל את המתודות שאפשר לעשות איתו).

Abs - נותן מספר אבסוטלוטי. נניח יש לנו מספרים, חלקם חיוביים וחלקם שליליים ואנחנו רוצים לדעת את ההפרשים בין כל המספרים האבסולוטיים ללא סימן. אם כל מספר יהיה בabs נוכל לחשב הפרש.

Floor - מעגל מספר למטה תמיד

Ceil - מעגל מספר למעלה תמיד

max/min - נותן מינימום או מקסימום מבין מספרים

Pow - מעלה בחזקה

Math.pow(2,4)

- 16

Random - מגריל מספר בין 0 ל-1. אם נרצה מספר בין 0 ל-100, גם אם נכפיל ב-100 אף פעם לא נקבל 100 כי זה לא מגריל 1. רק גדול שווה מ-0 וקטן מ-1. אם רוצים לקבל נגיד לו

math.floor(math.random() \*100) +1 OR math.ceil(math.random() \*100)

- 85

1. קלוט מחרוזת שמייצגת אימייל וודא שיש רק סימן @ אחד.

Var email = “[yos@israel@gmail.com](mailto:yos.israel@gmail.com)”

Email. split(“@”)

- (3) [‘yos’, ‘israel’, ‘gmail.com’]

email.split(“@”).length == 2

- false

פיצלנו למערך לפי התו שמחפשים לדעת כמה יש ממנו ואז בדקנו האם אורך המערך (מספר התאים) שווה ל-2. קיבלנו false.

אפשרות אחרת:

Var last = email.lastIndexOf(“@”)

Var firsl = email.IndexOf(“@”)

First == last

False

1. קלוט מילה שמכילה בתוכה מילה אחרת. קלוט גם את המיקום של האות ההתחלתית של המילה הפנימית ואת אורך המילה. הצג את המילה החדשה למסך.

var firstWord = "background”.substring(0,5)

var SecondWord = "background”.substring(5)

OR

Var word = “background”

word.substring(word.indexOf(“ground”)) - שימוש במתודה בתוך מתודה. האינדקס אומר מאיזה מיקום המילה שאנחנו מחפשים מתחילה והסאבסטרינג נותן לנו את החיתוך.

**Full stack 12 - JavaScript 1 - 7/9/2022**

window.alert

alert()

alert(message)

Console.log - דרך לתקשר עם הקונסול, תשתמש במתודה לוג. זה בן של window.

Alert נותן לנו התראה בקצה העליון של המסך.

אפשר לכתוב קובץ שלם בתוך טאב sources ובתוכו משמאל יש snippets וליצור סניפט חדש.

Variable - אני רוצה שהמחשב יקצה תא בזכרון וישתמש בו כל עוד צריך. Var הוא keyword שמור ולכן הוא מופיע בצבע שונה ״אני הולך להכריז על משתנה״.

Var firstName = “Yos”;

עכשיו אם נכתוב בקונסול למטה firstName הוא יחזיר Yos. הexecution הוא עם חץ play או עם ctrl+enter.

עד לפני כמה שנים היה רק את var ב-JS.

יש הבדל בין משתנה שהוא קבוע סטטי שמגדירים פעם אחת בתחילת התוכנית ולא רוצים שהוא ישתנה בהמשך ולא ידרסו אותו, נגדיר אותו כ-const. אם ננסה לשנות משתנה שהוא const נקבל הודעת שגיאה רלוונטית.

משתנה let יהיה משתנה שאנחנו מצפים שייתכן שישתנה במהלך התוכנית. אמור לעבוד בסקופ מסוים.

Var הוא יותר גלובליף יותר יכול לגרום לבעיות ולכן לא אמורים לעבוד איתו יותר.

אפשר להגדיר let בלי לתת שום ערך. למה? כי אנחנו יכולים לתת לו ערך בהמשך.

let example

לעומת זאת, אם ננסה לתת const exampleConst נקבל error. צריך לעשות השמה כי המשתנה הזה לא צפוי להשתנות בהמשך.

אם בהשמה של משתנה מכניסים משתנה אחר, המשתנה הראשון לא מתעדכן אוטומטית אלא רק עם נעשה השמה נוספת לדוגמה:

Let example = “yos”

Let message = example

Example = moshe

console.log(message)

- yos

כלומר למרות ששינינו את המשתנה example למשה,

prompt (“What is your name ?”)

prompt (“What is your age ?”)

פותח חלון עם השאלה הזו ומחכה לאינפוט. אבל אין לנו איך לגשת למידע. לכן נשמור אותו בתוך משתנה.

Const yourName = prompt (“What is your name ?”)

Const yourAge = prompt (“What is your age ?”)

const first = prompt("first number:");

console.log(first);

const second = prompt("second number:");

console.log(second);

const third = prompt("third number:");

console.log(third);

const cn1 = Number(first);

number הופך סטרינג למספר

const cn2 = Number(second);

const cn3 = Number(third);

const sum = cn1 + cn2 + cn3;

const avg = sum / 3;

console.log("your sum is: " + sum + " and the average is: " + avg);

למה להשתמש ב-number כדי להפוך סטרינג למספר ולא ב-typeof או ב-parseint?

console.log(typeof 8);

- number

console.log(typeof “bla”);

- string

console.log(typeof true);

- boolean

cn1=1, cn2=7, cn3=10

console.log(“sum is: “ +cn1+cn2+cn3)

- sum is: 1710

Concat - חיבור של אלמנטים (סטרינגים, מספרים וכו). כשמחברים סטרינג ל-number הוא הופך את ה-number לחלק מהסטרינג ולא מתייחס אליו כמספר.

Const meaningfulNames = 12;

Const userInput1 = prompt(“Insert number 1”);

Const userInputList = prompt(“Please insert list of numbers, (comma separated)”);

userInputList.split(“,”)

אסור לתת שמות של משתנים:

* מתחילים במספר 1stNum
* שיש מקפים ביניהם
* להימנע מסימנים כמו דולר וכו׳

//get user comment

Const userInput = prompt(“please insert comment (not above 100 chars)”);

console.log(“User input is: “ + userInput);

//check number of chars

constinputLength = userInput.length;

//show only 100 chars

Const message = userInput.slice(0,100);

ייתן את התווים במקומות 0 עד 100 לא כולל האחרון ואז זה ייתן לנו באמת 100 תווים כמו שרצינו.

alert(“Your comment is: \n” + message);

console.log(message.length)

שימוש ב-slice אפשרי נניח אם נותנים מספר אחד רק - יציג מאותה נקודה או אם נותנים מספר שלילי אז יספור מהסוף.

שימוש ב-backslash \ מאפשר לנו escaping כלומר אומר להתעלם מהתו הבא אחריו בתוך סטרינג ולהתייחס כחלק מהסטרינג. ככה למשל הוא

תרגיל - לקבל זמן סרט בדקות ולהציג בשעות ודקות

Const inputMovieInMinutes = prompt(“Please insert movie length in minutes”);

console.log(“inputMovieInMinutes: “ + inputMovieInMinutes);

**OR**

console.log(`inputMovieInMinutes: ${inputMovieInMinutes}`); - מזהה את המשתנה בלי לסגור את הסטרינג.

Const convertedMinutes = number(inputMovieInMinutes);

console.log(`convertedMinutes ${convertedMinutes}`);

Const hours = math.floor(convertedMinutes / 60);

console.log(`hours ${hours}`);

Const minutes = convertedMinutes % 60;

console.log(`minutes ${minutes}`);

Const message = `${convertedMinutes} are: ${hours} hour(s) and ${minutes} minute(s).

הבדל בין floor ל-round - round יעגל לפי הקצה אליו הוא קרוב. Floor או ceil יעגל תמיד למטה או למעלה בהתאמה.

תרגיל -

//get at least 4 digits number

const atleastFourDigits = prompt("Please insert at least 4 digit number and get the right digit");

console.log(atleastFourDigits);

// get the last char

const lastDigit = atleastFourDigits.substring(atleastFourDigits.length -1);

console.log(lastDigit);

//another way

const convertedNumber = number(atleastFourDigits);

const lastDigit2 = convertedNumber % 10;

console.log("second way result is: " + lastDigit2);

const twoDig = 1111174 % 100;

console.log(Math.floor(twoDig / 10));

**Full stack 13 - JavaScript 2 - 11/9/2022**

פונקציה - הגדרה של פעולות קבועות מראש שנרצה לבצע שוב ושוב.

אנו נרצה שפונקציה תהיה לה פעולה אחת. למשל בדוגמה של הזמנת פיצה בעזרת בוט, יש שלבים שונים: ללכת לחנות, להכניס כסף, לחשב כמה פיצות ואת העודף ובסוף לשלוח הבית. נרצה שלכל פעולה תהיה פונקציה משלה ואז אפשר לעשות פונקצית-על שתכלול את כל הפונקציות הקטנות.

פונקציה יכולה לקבל פרמטרים אבל יכולה גם להחזיר משהו return. כל מה שנכתוב אחריה, לא יבוצע כי אנחנו יוצאים מהפונקציה.

אם הגדרנו const ועכשיו נכנסנו לפונקציה, אפשר להגדיר מחדש את ה-const כי זה scope אחר.

**Full stack 14 - JavaScript 3 - 14/9/2022**

רקורסיה - לקרוא לפונקציה מתוך הפונקציה באופן חזרתי.

Stack - מחסנית. דוחפים עוד אחד ועוד אחד. הראשון שנכנס הוא הראשון שיוצא. כל פעם שנתחיל להוציא לפועל, אוגרים לפי הסדר.

לולאת while

Let number = 0;

While (number < 10) {

console.log(number);

Number++

}

console.log(`Number is ${number}`);

חייבים כל פעם לשנות את המשתנה אחרת ניכנס ללולאה אינסופית.

Let didntCome = true;

Let years = 0;

While (didntCome) {

Years++

if(years == 100) {

didntCome = false;

}

}

לולאת תנאי (for אינדקס) - כדי להימנע מלולאת while אינסופית בגלל ששוכחים להוסיף את התנאי שגורם לסיום הלולאה, עשו לולאת תנאי המורכבת משלושה חלקים:

* החלק הראשון מגדיר את האינדקס ומאתחל אותו. זה קורה רק פעם אחת ולא שוב ושוב בלולאה.
* תנאי מפסיק
* מתודה שגורמת לאינדקס להגיע ליעד (ל-bound) שלו.

אם היינו כותבים בלולאת while, זה היה נראה כך:

Let index = 0;

While (index < 100) {

Index++

}

את כל זה אפשר להכניס ללולאת תנאי באופן הבא:

For (let index = 0; index <= bound; index++) {

}

בתנאי המפסיק חשוב לעשות קטן/שווה ולא רק קטן כדי שהוא ירוץ גם כשהוא שווה וגם כשקטן. אם לא נעשה קטן אלא רק שווה, אז הוא לא ירוץ בכלל כי הוא שואל ״האם האינדקס שווה ל-bound?״ אם התשובה היא לא, אז הוא לא ירוץ.

בתוך הלולאה כותבים קוד לדוגמה עם if.

דוגמה:

Function getSumOfNumbers(iterations){

Let sum=0;

Let number;

for(let i = 1; i<= iterations, i++){

// get random number

Number = math.ceil(math.random() \* iterations);

//add random number to sum

console.log(`old sum = ${sum}, number = ${number}`);

Sum = sum + number

console.log(`new sum = ${sum}`);

}

// return sum

Return sum;

תרגיל 3 - קלוט 100 נתונים והדפס את הגדול ביניהם (במקום 100 נתונים הוא מעדיף random).

Function getHeighest(iterations){

Let number = 0;

Let highestNumber = number

For (let i =1; i <= iterations; i++){

Number = math.ceil(math.random() \* iterations);

if(heighestNumber < number){

heighestNumber = number;

}

}

Return heighestNumber;

}

אופציה אחרת

Function getHighest(iterations){

Let number = 0;

Let highestNumber = number

For (let i =1; i <= iterations; i++){

Number = math.ceil(math.random() \* iterations);

higestNumber = math.max(highestNumber, number);

}

Return highestNumber;

}

**מערכים array**

Const cars1 = new array(“Saab”, “Volvo”, “BMW”);

Const cars2 = [Saab”, “Volvo”, “BMW”];

מתודת push מוסיפה לסוף המערך עוד איבר

cars2.push(“Suzuki”);

Cars1[3] = “Suzuki”;

Reverse - אפשר לבקש את המערך בסדר הפוך. לשים לב **שדורס את המערך הקיים!!**

cars1.reverse()

מספר האיברים לא שווה למספר האינדקס כי אינדקס מתחיל ב-0 ומספר איברים גדול באחד מהאינדקס הכי גדול.

Sort - ממיין את המערך וגם הוא **דורס** את הסדר הקיים.

cars1.sort()

אם מכניסים מערך מספרים ועושים לו sort הוא לא ימיין לפי גודל 1,3,5,12,56,1000 אלא ימיין כמו string כלומר 1,1000,12,3,5,56. כדי להתגבר, צריך להשתמש בפונקציה.

Function numbersSort(a,b){

if (a < b){

Return -1;

}else if (b < a){

Return 1;

}else{

Return 0;

}

}

numbers.sort(numbersSort);

כל פעם ששואל מי גדול משני מספרים, תחזיר את הראשון.

למה צריך למיין?

Const randomArr = [];

For (let i = 1; i < 100; i++) {

randomArr[i] = math.ceil(math.random() \* 100);

randomArr.indexOf(12)

מאחר והכל רנדומלי, הולכים לאינדקס באמצע ושואלים איזה מספר זה. לפי זה בוחרים איזה חצי צריכים ואז בוחרים שוב את האמצע שלו. שוב מחלקים לחצי וכו׳.

רוצים למצוא את מספר 9 בתוך מערך.

Function getIndexOfNumber(arr,number){

For (let i = 0; i < arr.length; i++) {

if(arr[i] == number) {

Return i;

}

console.log(i)

}

console.log(i);

אם רוצים להחליף איבר במערך

Const fruits = [“Banana”, “Orange”, “Apple”, “Mango”];

fruits[fruits.indexOf(“Banana”)] = “Date”

Fruits

(4) [‘Date’, ‘Orange’, ‘Apple’, ‘Mango’]

fruits.join(“ “).replace(“Banana”, “Date”).split(“ “) - מאחדים את המערך לסטרינג ואז מחלקים שוב לפי הרווחים.

Const newFruits = fruits.join(“ “).replace(“Apple”, “Watermelon”)

**מערך של מערכים**

לדוגמה לוח הכפל

Const board = [];

for(let i=1; i <=10, i++){

Const arr = [];

for(let j=1; j <=10; j++){

Arr[j -1] = j\*i;

}

board.push(arr);

}

תרגיל 1

Const arr = []

Let number;

Let sum = 0;

Let avg;

// get 50 numbers and set inside the array

For (let index = 1; index <= 50; index++){

Number = math.ceil(math.random() \* 100);

arr.push(number);

Sum = sum+number

}

// get the sum and the avg. avg = sum / length

Avg = sum / arr.length;

console.log(“The avg is: “ + avg);

// go over the array and print the numbers that are higher than the avg

console.log()

Const higherThanAvg = [];

for(let i = 0; i < arr.length; i ++){

If (Arr[i] > avg){

higerThanAvg.push(arr[i]);

}

}

console.log(“The following numbers are higher than

**Full stack 15 - JavaScript 4 - 18/9/2022**

לעבור על 45 דקות - sort

**Iterators**

***forEach***

Const numbers = [45, 4, 9, 16, 25];

numbers.forEach(function (value, index, array){

console.log(value, index, array);

})

יש פונקציה אנונימית שאז כותבים function אם למשל לא קוראים לה בתוך ה-forEach. אבל אם רוצים לההשתמש ולקרוא לה בחוץ, צריך לתת לה שם.

Const printValues = (vale, index, array) ⇒ {

console.log(value, index, array);

}

numbers.forEach(printValues);

***Map***

בניגוד ל-forEach שלא משנה את ה-array, שימוש ב-map מייצר מערך חדש. נותנים array קיים ורוצים לקבל array חדש לפי תנאים מסוימים. למשל להכניס מערך קיים ולהחזיר רק את המספרים הזוגיים שבו.

numbers.map(function(value){

Return value % 2 == 0;

})

Const newNumber = numbers.map(function(val){

if(val % 2 == 0){

Return val

}

});

Const newNumber = numbers.map(function(val){

Return val \* vals;

}

});

console.log(newNumber);

console.log(numbers);

Filter מצפה ל-true או false. אם יצא true - זה אומר שתנאי הפילטור מתקיים ואז הוא מכניס את האיבר לתוך מערך חדש.

תרגיל - צרו מערך עם 10 מספרים.

1. עבור כל אחד מהמספרים הוסיפו לו 2 והדפיסו אותו.
2. קבלו מערך חדש המורכב מהמערך הראשון ובו כל מספר הוא מחולק ב-2.
3. קבלו מערך חדש שיש בו רק את האיברים הזוגיים.

Const arr = [1,55,4,2,68,543,98,43,221,46];

Function printAddTwo(value){

console.log(value + 2);

}

Function divValue(value){

Return value / 2;

}

Function oddValues(value){

Return value % 2 != 0;

}

numbers.forEach(printAddTwo);

Const divNumbers = numbers.map(divValue);

Const oddNumbers = numbers.filter(oddValues)

Const Car = {

Door: 4,

Model: “Ford”,

Color: “red

Break: function(){ זו מתודה בתוך המשתנה

console.log(‘breaked’);

}

}

Car.color

* ‘Red’

Car.break()

* Undefined

**Dom - Document Object Model**

לקחו את כל ה-DOM שזה האלמנטים ואמרו שלכל אלמנט, למשל h1, יש סטייל, הרבה attributes והם שווי ערך לתכונות של אובייקטים. אז כל אחת מהתגיות יש גם תכונות וגם מתודות ופונקציות שיכול לעשות. כל attribute זו תכונה (href, style, color וכו׳). מצד שני, לכל תג יש גם event ואז למשל אפשר לקרוא on hover - תעשה משהו.

עברנו ל-visualstudio.

JS אפשר לכתוב **inline** (כמו שהיה ב-CSS).

למשל אפשר לכתוב onload שזה קוד לכל דבר.

<body onload=”document.querySelector(‘title’).innerHTML(‘Hello world’)”> - לא עבד לו

<h1>Hello</h1>

הכוונה ב-document היא לכל עמוד ה-html שאנחנו עומדים עליו.

<body onload=”console.log(‘Hello world’)”>

אופציה אחרת היא להשתמש בתגית script בצורה שהיא **internal**.

<script>

document.querySelector(‘h1’).innerHTML = “Goodbye”;

</script>

בשימוש ב-querySelector אפשר לבחור את כל סוגי הסלקטורים - class, id, tag וכו׳. אנחנו אומרים לו לחפש את סוג התגית הראשונה שמוצא (למשל אם מחפשים p או h1), תיגגש ל-attribute שנקרא innerHTML ותעשה לו השמה.

אפשר ליצור תגית script גם בתוך ה-head. אם נוסיף

<script>

alert(“Hello world”);

</script>

בתוך ה-head, הוא יריץ alert כבר בטעינת העמוד.

יוס נותן קונסול לוג כדי לראות מה רץ קודם. הוא מראה שקודם רץ הסקריפט ב-head אח״כ תגית script שהיא internal ורק בסוף כשסיים לטעון את ה-body אז הוא יודיע שסיים onload למרות שמופיע לפני התגית שכתוב ב-internal. כדי לעצור טעינה לפני שורה מסוימת אפשר לכתוב debugger; ואז בטעינה הוא מחכה שנלחץ לו פליי כדי להמשיך. השימוש ב-onload נקרא eventListener. למעשה ״נרשמים״ לאירוע מסוים ורוצים שמשהו יקרה רק אחרי שקרה האירוע המסוים. בדוגמה שלנו, מבקשים שיהיה console log מסוים רק אחרי שכל העמוד מסיים להטען. דוגמה נוספת היא hover שאפשר להירשם onHover ואז כשמישהו עושה hover יקרה משהו.

בד״כ לא שמים קוד JS ב-head אלא אם הוא הגיע מבחוץ. בד״כ קוד JS יגן על עצמו כך שייכתב בתוך ה-body בתור onload כלומר, רק אחרי שהעמוד סיים להטען - אז תתחיל להריץ את הקוד.

אנחנו לא חייבים לכתוב את זה inline בתוך ה-body. אפשר בקובץ js נפרד לכתוב:

Document.body.onload = (‘console.log()’)

אחר כך שינה ל-

**// external**

Document.onload = function(){

console.log(‘From index js’);

}

איך מחברים קובץ חיצוני?

<script src=”index.js”></script>

DOM - Document Object Model

כשרצו להוסיף js לדפדפן, היו צריכים למדל ועשו את זה בצורה של עץ. למשל יש את ה-document כולו. יש לו שני ילדים head, body שהם אחים ולכל אחד יש ילדים משלו. למשל ל-body יש ילדים משלו h1, script וכו׳. הפכו כל תגית לאייטם ולכל תגיד יש attribute (סטייל, צבע, וכו׳) יש לו תכונה.

משתמש ב-extension HTML tree generator

document.lastChild.lastChild - מראה מקום בעץ

document.querySelector(“h1”) - ניגש לראשון שמוצא.

// targeting querySelector

Const header = document.querySelector(“h1”) - נשמור את הגישה אליו בתוך משתנה אם נרצה לעשות את זה הרבה פעמים.

Header

– <h1 id=”title”>Hello world</h1>

Header.style.color = “red”

Header.style.backgroundColor = “yellow”

// targeting getElementsByTagName

Const checkbox = document.getElementsByTagName(‘input’)[10];

checkbox.checked = true;

ברגע שהוא טוען קובץ ב-head עם אלמנטים שבעצם עוד לא נטענו ב-body, צריך להשתמש ב-event listener.

השימוש ב-querySelector הוא כמו טירגוט ב-css. למשל

// querySelector of inner item

Const firstLi = document.querySelector(“ul li”);

firstLi.style.backgroundColor = “yellow”;

// targeting by class names

Const itemList = document.getElementsByClassName(“item”);

console.log(itemList);

אופציה אחרת היא להשתמש ב-querySelectorAll ע״י שימוש בטירגוט ממש כמו כתיבה של css, נכתוב את ה-class הרצוי. הוא תמיד מחזיר מערך גם אם הוא ריק או שיש בו רק איבר אחד.

// targeting class names

Const listItems = document.querySelectorAll(“.item”);

listItems.forEach(value => value.style.color = “blue”);

listItems[2].innerHYML = “Michael”;

ישנם מערכים קבועים שאפשר להשתמש בהם של document:

Document.links - מכניס למערך את כל הלינקים שמופיעים בקובץ

Document.anchors - נותן את כל האלמנטים של a

Const link = document.getElementById(“list”).firstElementChild.firstElementChild.href;

במקום כל הסיבוך הזה, אפשר להגיד לו ב-querySelector.

Const link = document.querySelector(“#list a”).href;

Link = “[www.example.com](http://www.example.com)”

אח״כ שינה והוריד את ה-href מהגדרת המשתנה אבל הוסיף אותו כ-link.href. אם היינו שומרים בתור משתנה את ה-href זה לא מספיק לנו כי זה היה שומר לנו טקסט ולא אובייקט. ברגע ששומרים אובייקט בתור משתנה, זה מאפשר לשנות אותו.

**Full stack 16 - JavaScript 5 - 21/9/2022**

כשמתחילים לעבוד עם js על html, נפתח בתוך ה-body:

<script src=”index.js”></script>

כדי לראות שאכן מחוברים, ב-JS נדפיס:

console.log("object connected")

אובייקט הוא אלמנט של js שיש לו:

* properties (תכונות)
* methods (פונקציות, מתודות)
* events (אירועים) - במידה ויקרה אירוע מסוים, נרצה שהוא יעשה פעולה מסוימת.

יצירה של אובייקט - את השם ניתן עם אות גדולה ראשונה. הכרזה על אובייקט באמצעות סוגריים מסולסלים. לכל אובייקט נותנים תכונות והוא מסוגל להפעיל פונקציות מסוימות. יש אירועים שאפשר להירשם אליהם ולקבוע מה יקרה כשהאירוע יקרה.

Encapsulation - אנחנו לרוב לא נרצה לשנות משהו באובייקט באופן ישיר אלא נשתמש בפונקציות/מתודות שיעשו את זה עבורנו כדי לא להתערב ביצירה של האובייקט. יוצרים גטרים getters וסטרים setters. למשל getName תהיה פונקציה שמחזירה לנו את השם. setName הוא פונקציה שמקבלת שם חדש והתפקיד שלה לשים במקום השם שנמצא. השימוש הוא כדי לא לגשת ישירות ולשנות את התכונה בעצמה.

כששמים סוגריים מפעילים פונקציה math.random למשל. בלי סוגרים מבקשים לקבל ערך של תכונה למשל length.

//object

console.log(“object connected”);

Const student = {

Name: “Yos”,

lastName: “Israel”,

getName: function(){

Return this.name;

},

setName: function(newName){

This.name = newName;

},

getLastName: function(){

Return this.lastName;

},

setLastName: function(newLastName){

This.lastName = newLastName;

}

אנחנו רואים שלאובייקט student יש שתי תכונות name, lastName וארבע מתודות get/set name/lastName.

student.setLastName(“Oliva”); - לשנות את שם המשפחה

המשתנה הבא הוא פונקציה שהתפקיד שלה היא לייצר אובייקטים באמצעות פונקציות אחרות.

//object with constructors

Const StudentObject = function(name, lastName) { **השמה**

This.name = name;

this.lastName = lastName;

this.getName = function(){ **קבלה, read only**

Return this.name;

};

this.getLastName =() => {

Return this.lastName;

};

this.setName = function(newName){ **דריסה**

This.name - newName;;

};

this.setLastName = function(newLastName){

This.lastName = newLastName;

};

}

Const student1 = new StudentObject(“Aram”, “Aroian”) - מוסיפים סטודנט חדש לכיתה ויוצרים אותו.

student1.getName()

* ‘Aram’

student1.setLastName(“Aroian Cohen”) - עכשיו הוא התחתן ושינה את שם המשפחה, נשתמש בסטר כדי לשנות לו את שם המשפחה.

student1.getLastName()

* ‘Aroian Cohen’

שימוש במילה this אומר ״תתייחס ספציפית למקום שבו הגדירו אותך״. אם יש name במקום אחר - אל תתייחס אליו.

למה נרצה שאי אפשר יהיה לשנות דברים אחרי שכתבנו אותם? נניח שאנחנו הגדרנו אובייקט שיש לו שם, גיל ומקצוע. אנחנו נרצה שתמיד יצירת האובייקט תתבצע באמצעות הפונקציה שתכלול set כי אז נהיה מחויבים למלא את שלושת התכונות שהגדרנו. אם אפשר לבצע שינוי ולדרוס משתנים, אז יכול לבוא מישהו אחר שלא מכיר את הקוד שלנו, והוא פשוט יגדיר name= x מבלי לתת גיל ומקצוע.

אם נגדיר משהו ב-let ולא ב-const זה יאפשר לשנות את הסוג נניח מאובייקט לסטרינג או למערך. אנחנו רוצים שהוא ישאר אובייקט ולכן נגדיר כ-const.

איך עושים השמה בלי להשתמש בגטרים או סטרים או קונסטרקטורים (לדוגמה שהיה מקודם this.name = name).

Const.person = {}; יצירת אובייקט

Person.name = “Alex” יצירת תכונה לאובייקט שלא הוגדרה בהתחלה. לא מתנגש עם קונסט שהוא קבוע.

Person

* {name: ‘Alex’}

Person = []

* Error!!

person.lastName = “Samih”

person.lastName

* ‘Samih’

person.name

* ‘Alex’

Const.person = {};

Person.name = “Example name”;

person.lastName = “Example Last Name”

person.getFullName = function(){

Return ‘The name is ${this.name} and this is the last name ${this.lastName}’

};

**Event listeners**

אם נריץ את הscript ב-head ולא בסוף ה-body יווצר לנו race condition בין הקובץ js שמנסה לרוץ לבין מה שמופיע ב-body שחלק מהדברים שם אנחנו רוצים לטרגט בקובץ js.

אמרנו שלכל אובייקט יש תכונות, מתודות ואירועים. אחד האירועים שאפשר להירשם אליהם הוא onload - כשאתה מסיים לטעון את העמוד loadedֿ, רק אז תפעיל את מה שאני מגדיר לך עכשיו.

לדוגמה נגדיר בתוך ה-body את ה-h1

<h1> Example on load issue</h1>

בקובץ JS נכניס

document.getElementsByTagName(“title”)[0].innerHTML = “Hello world”;

document.getElementsByTagName(“h1”)[0].innerHTML = “Hello world”;

את ה-title הוא שינה אבל את ה-h1 לא שינה. למה? כי ה-script מופיע אחרי ה-title אבל לפני ה-body וה-h1 נכנס בתוך ה-body.

איך נרשמים ל-event? אופציה ראשונה היא inline

<body onload = “runJs()”>

את הפונקציה שאנחנו קוראים לה מתוך ה-HTML אנחנו נלך להגדיר בתוך קובץ ה-JS מאחר והם מחוברים באותו שרת.

Function runJs(){

document.getElementsByTagName(“h1”)[0].innerHTML = “Hello world”;

}

document.querySelector(‘body’).addEventListener(‘load’, runScript());

document.getElementsByTagName(“h1”)[0].innerHTML = “Hello world”;

אנחנו רוצים עכשיו שכל פעם שנקליק על h1 יקרה משהו

document.getElementsByTagName(“h1”)[0].addEventListener(‘click’, runScript)

אנחנו מעבירים לו את הפונקציה באירוע מסוים (קליק) שזה למעשה נקרא callback.

אפשר כמובן להירשם לאירוע גם inline בתוך קובץ ה-html

<h1 ondblclick=”handledClick()”>Example on load issue</h1>

ואז קוראים לפונקציה או מתוך קובץ ה-JS

Function handledClick(){

console.log(‘yos’);

}

או ישירות בתוך קובץ ה-html כשמייצרים את הפונקציה בתוך תגית script.

אחרי הפסקה - תרגיל

In HTML

<title>Add event listener button</title>

<body>

<button>Click me</button>

<script src=”index.js”></script>

In JS

Const btn = document.querySelector(‘button’);

btn.addEventListener(‘click’, handleClick);

Function handleClick(){

alert(‘Clicked’);

}

\*\*לבדוק - בערך 1.50-2.00

מה ההבדל בין querySelector ל-getElementsByTagName - בעיקרון get מקבל רק לפי tag names והסלקטור מקבל לפי סוג. למשל במקרה זה רק נגיד button. עדיין לא ברור.

נוסיף ל-html כל מיני h1

In HTML

<h1></h1>

In JS

Const headers = document.getElementsByTagName(‘h1’);

Const headers2 = document.querySelectorAll(‘h1’);

כשהשתמשנו ב-getElements קיבלנו HTMLCollection שהוא כמו מערך אבל לא בדיוק.. כשהשתמשנו ב-querySelector קיבלנו NodeList שמתפקד כמו מערך.

headers.forEach(item => console.log(item.innerHTML));

For(i=0; i<headers.length; i++){

console.log(headers[i])

}

ה-forEach לא עבד אבל ה-for כן. למה? כי forEach עובד רק על מערך וה-collection הוא לא list. לכן על headers אי אפשר להשתמש במתודה של array.

אם נריץ את ה-forEach על headers2 - זה כן יעבוד מהסיבה שהסברנו.

תרגיל - בכל פעם שמכניסים משהו בתוך חלון input, ה-header משתנה.

In HTML

<div class=”container”>

<h1 class=”header”>Input example</h1>

<input type=”text” name=”inputText” id=”headerChanger”>

</div>

צריך לטרגט את האינפוט ולקבל את ה-innerHTML שלו. נשתמש באירוע on change.

In JS

//get input object and h1 object

Const input = document.querySelector(‘#headerChanger’);

Const header = document.querySelector(‘.header’);

//set eventListener

input.addEventListener(‘change’, handleInputChange)

למה השתמשנו ב-change בתור event ולא onChange? אם היינו כותבים את זה כ-inline בתוך ה-HTML אז באמת היה צורך לציין שזה onChange. אבל מאחר ואנחנו מגדירים את זה בתוך ה-js, אז כבר ידוע שזה eventListener ולכן לא צריך לציין on אלא מספיק change.

//create handleChange and change h1

Function handleInputChange(){

header.innerHTML = input.value;

}

מה ההבדל בין inner text ל-innerHTML? עקרונית שתיהן מכניסות טקסט לתגיות מכולה. אבל אם נרצה להכניס טקסט שהוא מודגש לדוגמה, אז נעביר ל-h1 תגית strong שבתוכה יש טקסט. אם נשתמש ב-inner text הוא יעביר רק את הטקסט. אבל אם רוצים שהתגית של html תעבוד - צריך להשתמש ב-innerHTML.

מכיוון של-input אין טקסט בתוכו אלא רק attribute מסוג value שנקבע כל פעם שכותבים משהו, אנחנו רוצים את הערך שבתוך ה-value. כי למעשה ה-innerHTML של input הוא ריק.

עכשיו ננסה להשתמש בפונקציה

Function handleInputChange(event){

console.log(event);

}

מה זה event? מתוך ה-eventListener אנחנו יכולים לבקש לקבל מידע על פרמטר שנקרא event. ל-event יש הרבה תכונות שאחת מהן היא target. בתוך target הוא נותן לנו את כל המידע על האירוע שקרה. ה-type=change, name, id הכל.

Function handleInputChange(event){

Const value = event.target.value;

משכנו את ה-value שקיבלנו ב-event ושמרנו אותו.

Const name = event.target.name;

שמרנו גם את שם האובייקט

Const example = {

[name]: value

}

console.log(example);

}

נשמור לקבוע את ה-key עם ה-value שלו. שימוש של name בתוך סוגריים מרובעים יתן לנו את הערך שמתקבל מהקבוע name (שם כלשהו), אחרת הוא יציב שם של key כזה (המילה name). הסוגריים אומרים לו ״אל תתייחס למילה name בתור key אלא תשתול את הערך של ה-name ששמור לך שהוא למעשה נקרא inputText

}

header.innerHTML = value;

console.log(example.inputText);

מאחר והגדרנו את האובייקט של name: value כאשר ה-name למעשה מקבל את הערך של inputText, אז הגדרנו כבר מה זה inputText. לכן כשנדפיס את example.inputText למעשה נקבל את ה-value של inputText שהוא הערך שהכנסנו בעמוד.

דוגמה אחרת forms2:

<div class=”container”>

<h1>Form</h1>

<form id=”login-form”>

משתמש בדוגמה מהאתר של בוטסטראפ. מוחק הרבה דברים, מוסיף שמות וכו׳.

אנחנו רוצים לשלוח לקבל מהמשתמש אימייל וסיסמה ולעשות ולידציה, בצורת שליחה לשרת ולוודא האם תקין. לשם כך, נרצה להירשם לאירוע מסוג submit. נזכיר שכשעושים submit ל-form זה שולח את הערכים שהתקבלו כ-input ל-url ומרפרש את העמוד.

// get user email, get user password

Const loginForm = document.querySelector(“#loginForm”);

// add eventListener for form values

loginForm.addEventListener(‘submit’, handleSubmit);

// add function to handle submit

Function handleSubmit(event) {

event.preventDefault()

אמרנו שאחרי submit העמוד מתרפרש. preventDefault אומר לו תעצור כל מה שאתה אמור לעשות באופן אוטומטי (למשל ריפרש) כי אני לא רוצה לאבד את האינפוט שמופיע בדף.

console.log(event.target[0].value);

}

מחזיר לנו את האימייל שהכניס ב-form.

למעשה טרגט מחזיר לנו מערך. לכן אפשר לרוץ עליו כמו מערך.

Const loginData = {};

Const target = event.target;

for(let index = 0; index <target.length; index++) {

Const element = target[index];

loginData[element.name] = element.value

}

console.log(loginData);

}

ה-preventDefault הוא כדי למנוע מהעמוד להתרפרש כי זה חלק ממה שקורה כשעושים submit ואין הגדרה לשלוח את הדאטה ל-url, אז הדאטה חוזר לעמוד ומתרפרש. אבל אז אנחנו לא מצליחים לתפוס את הדאטה ולכן אומרים לעמוד לא לעשות את הפעולות המוגדרות כ-default.

קובץ חדש index2.js

רוצים לקחת כל אחד מהאינפוטים שהם לא מסוג type=submit, ליצור eventListener על ה-change וכל פעם שמשתנה ניצור לו אובייקט מהתחלה.

// get all input elements and submit

Const inputList = document.querySelectorAll(‘#loginForm input’);

Const loginForm = document.querySelector(‘#loginForm);

Const loginData = {}; - יוצר את המשתנה הזה מחוץ לפונקציה כי ככה לכולם יש גישה אליו ולא רק בתוך פונקציה.

// add event listener for all of them

inputList.forEach(input => input.addEventListener(‘change’, handleChange); - תופס כל שינוי שנעשה באינפוט ולא מחכה שיבצעו סבמיט.

loginForm.addEventListener(‘submit’, handleSubmit);

// create function for handleChange and for submit

Function handleChange(event) {

Const value = event.target.value;

Const name = event.target.name;

Object.assign(loginData, {[name]: value}); - לוקח שני אובייקטים ומחבר אותם

}

Function handleSubmit(event){

event.preventDefault();

console.log(loginData);

}

מתודה object.assign לוקחת שני אובייקטים ומחברת אותם בלי למחוק את שאר הדברים האחרים שהיו בהם. לדוגמה

const returnedTarget = Object.assign(target, source)

**Full stack 17 - JavaScript 6 - 28/9/2022**

מתכננים עמוד של חנות ספרים. לכל ספר את הפרטים שלו (מחבר, תיאור קצר, תמונה מחיר וכו׳). רוצים שגם יאפשר להוסיף ספר חדש לרשימה עם כל הפרטים שהזכרנו.

מתחת, אנחנו רוצים להציג את כל הספרים אחד ליד השני.

כמובן רוצים שכל פעם שנרפרש את העמוד, כל הספרים ישמרו.

הוא לוקח מהאתר של בוטסטראפ את הדוגמה ל-form ומתחיל לנקות מה שלא צריך. משום מה בבוטסטראפ הם לא מוסיפים name וחייבים להוסיף אותו בעצמנו כי אח״כ יהיה לנו קשה לטרגט.

הוא מחפש קו מפריד אז פשוט עושה חיפוש בבוטסטראפ ובוחר באיזה סגנון שרוצה.

בנוסף, בבוטסטראפ יש משהו שנקרא card. הוא בוחר בזה כדי להציג כל ספר בנפרד עם תמונה והפרטים שרצינו. הוא מוחק מה שלא צריך מהתבנית של bootstrap עבור ה-card.

בשביל תמונה הוא הולך ל-flaticon ומחפש placeholder.

איך יודעים כמה cards נכניס בשורה? אפשר להוסיף ב-class יחד עם card לציין card col ואז נכנסים עד 12 כרטיסים כי שימוש ב-col מחלק את השורה ל-12 טורים. הוא רוצה 5 ספרים בשורה אז צריך לעשות 12:5. בפועל כתב card col-2. משהו לא מסתדר אז הוא בודק את מימדי התמונה שהכניס. בסופו של דבר הוא משנה את התבנית של בוטסטראפ מרוחב 18 ל-15.

<div class="card" style="width: 18rem;">

בסוף שינה שוב ל-width: 20% דרך קובץ css.

.card{

Width: 20%;

}

חוזר להסבר על בוטסטראפ - הם מחלקים את ה-layout, grid, container ל-12 יחידות. אנחנו אומרים כמה כל איבר יכיל בשורה. 12 לחלק ל-x נותן את מספר האייטמים בשורה.

ה-form כרגע תופס את כל רוחב העמוד. הוא מכניס div class=row ול-form הוא נותן col-6

<div class=”row”>

<form class=”col-6”>

עכשיו הוא עושה משהו שחייבים להוריד בסוף ומשמש רק לבדיקות וחוסך את ההזנה של ערכים בכל פעם.

עבור ה-author או תמונה הוא מכניס באינפוט הרלוונטי (שם מחבר, מחיר, לינק לתמונה וכו׳):

value= "Brian Bilson" OR value = “https://images-eu.bookshop.org/image…”

עכשיו עוד לפני שעוברים ל-JS, הוא עושה בדיקה. הוא מעתיק את ה-url שבו מוזנים כבר הרבה אינפוטים. הוא מעתיק לקובץ טקסט (planning). כדי להפריד את הפרמטרים שקיבלנו ב-url, הוא עושה ctrl+f לסימן & ולוחץ על החץ הקטן כדי לעשות replace. ה-seperator הוא & והוא מחליף אותו ב-shift+enter.

אח״כ הוא הולך ל-encoder decoder ומשתמש בו בשביל להוריד את כל החלקים המיותרים. עדיף לעשות את זה לפני ה-replace.

כדי לנסות לראות איך נראה כרטיס, הוא מכניס בכל מקום ב-card את הטקסט או url שצריך. למשל בחלק בכרטיס שמיועד לשם הספר, הוא מזין בין תגית הפתיחה לתגית הסוגרת את שם הספר.

עכשיו עוברים ל-JS.

אנחנו רוצים:

* ליצור books object
* לקבל את הערכים מהטופס לאחר סיבמוט
* בעת הוספת ספר נרצה שיתווסף ל-books object
* לרנדר - לקחת קוד ולהציג את התוצאה שלו. נרצה לרנדר את הגלריה של הספרים.
* אפשרות להסרת כל הספרים
* שמירת הספרים גם אם האתר עובר ריפרוש

דבר ראשון נרצה לתפוס את ה-form ואז להוסיף לו eventListener בסבמיט.

Const form = document.querySelector(‘form’);

form.addEventListener(‘submit’, handleSubmit);

Function handleSubmit(event){

event.preventDefault();

}

ה-event שבשימוש

נרצה לחלק את הקוד לשלושה חלקים: בחלק הראשון נגדיר משתנים, בחלק השני, פעולות על המשתנים (addEventListener וכו׳) ובחלק השלישי פונקציות.

\*לא לשכוח 1: לחבר את הקובץ JS ל-html שלנו ולבדוק שמחובר.

\*לא לשכוח 2: אם משתמשים ב-handleSubmit, לעשות event.preventDefault.

בתוך ה-hanldeSubmit נרצה לקבל כל אחד מהאינפוטים ולשמור בתור אובייקט שמכיל key:value.

formDataAPI - הופך להיות API לכל הדאטה שנמצא ב-form. ה-API הוא סוג של תוכנית/אובייקט עם המון פונקציות שמקל את העבודה עם הנתונים שיש בתוך הטופס. איך מאתחלים?

בתוך הפונקציה של handleSubmit נכניס

Const formData = new FormData(form);

אנחנו מקבלים למעשה אובייקט שמכיל key, value. אם רוצים לראות מה מתקבל מזה, אפשר לעשות בפונקציה גם console.log(formData)

איך עובדים עם זה?

formData.get(name)

במקרה שלנו:

formData.get(‘title’)

הוא מנסה להריץ את זה בקונסול עצמו ומחזיר לו not defined. למה? כי הגדרנו את ה-formData רק בתוך הפונקציה. במקום זה, הוא מנסה לעשות console.log(formData.get('title') ואז מקבל את זה בקונסול ורואה שזה עובד. גם כשמנסה להכניס description במקום title. זו אחת הסיבות שהציע לשים value דיפולטיבי ב-html.

בגלל שלא הגדרנו את form בתוך הפונקציה אלא הוא פרמטר גלובלי, יותר נכון יהיה להשתמש ב-event הגדרנו בתוך הפונקציה.

Const formData = new FormData(event.target);

למעשה formData מייצר לנו רשימה שאפשר לעבור עליה עם forEach. ה-key הוא מה שמתקבל מה-name שמוגדר עבור כל אינפוט ב-html!! לא מה-label שלו או משהו אחר.

formData.forEach(value,key)=>{}

הכתיבה המקוצרת שקולה לכתיבה המלאה:

formData.forEach(function(value,key){})

מה שנקבל זה אובייקט עם זוגות ואם נוסיף קונסול.לוג, נקבל את כל הזוגות מהפונקציה של forEach שעברנו עליהם.

console.log(key, value);

כדי לגשת לאיבר מסוים באובייקט, צריך לדעת את השם שלו שהוא ה-key שלו. עבור כל אחד מה-items (הצמדים של key, value) תחזיר לנו אותם. אנחנו רוצים להרכיב אייטם ממה שמתקבל ב-formData. עבור כל book אנחנו רוצים ליצור רשימת books. אחרי ה-preventDefault הוא מוסיף

Const book = {};

ובתוך ה-forEach הוא מוסיף (במקום ה-console.log):

Book[key] = value;

נדפיס את book לראות מה התקבל ונראה את כל האייטמים שמורכבים מזוגות key, value.

אם היה לנו מערך books, היינו רוצים להוסיף בכל פעם עוד book למערך.

Const books = [];

books.push(book);

עכשיו ניצור פונקציה חיצונית שתייצר את רשימת הספרים. בסוף ה-handleSubmit, נוסיף

addBookToList(book);

את const books = [] ניצור לפני הפונקציות כמשתנה גלובלי.

מחוץ לhandleSubmit, נוסיף פונקציה אחרת שתוסיף כל book אלמנט לתוך המערך books:

Function addBookToList(book){

books.push(book);

}

עכשיו השאלה היא איך אפשר ליצור card של כל ספר באופן דינמי? ב-html יש לנו container שנקרא booksContaoner שמכיל את כל כרטיסי הספרים וכותרת most popular books. בכל פעם שנרצה להוסיף לקונטיינר ילד (card) נרצה לעשות append.

\*הערה - בשונה מ-append שמוסיף בסוף, prepend מוסיף בהתחלה.

עבור כל ספר אנחנו רוצים להזריק את המידע הנכון של לינק לתמונה, כותרת, מחבר וכו׳.

מעתיקים את האובייקט של ה-card מתוך ה-HTML ל-JS ונשים ב-comment.

Function createNewBook(book){

Const card = document.createElement(‘div’); - יצרנו אלמנט

card.classList.add(‘card’, ‘col-3’);

מוסיף את הקלאסים של הכרטיס. כרגע הוא באוויר ואנחנו רוצים לשייך אותו למשהו על ה-dom.

card.classList - מחזיר את רשימת הקלאסים על האובייקט שביקשנו לראות

כדי ליצור עוד

document.querySelector(‘#booksContainer’).appendChild(card)

כדי לא להוסיף כל ספר עם כל התכונות שלו באמצעות createElement לכל דבר, נצטרך ליצור string שאותו יודע לקבל ה-innerHTML ולפרסר אותו. מכניסים לתוך הגרש הקטן את הקוד שמוכל בתוך ה-div שאנחנו רוצים להכניס כאובייקט של book כולל ה-div הפותח.

card.innerHTML = `*code from html*`

אנחנו רואים שקיבלנו בעמוד של most popular books עוד ספר. ממש הזריק באמצעות innerHTML את הקוד. למה לא הכנסנו את הפונקציה של createElement עבור ספר חדש בתוך ה-innerHTML?

booksContainer.innerHTML = `*code from html*`

דורס את מה שהיה קיים וכל פעם שנוסיף ספר חדש הוא ידרוס את הישן. כדי שלא יעשה את זה, צריך להגיד לו append.

\*\*רק תגים מסוג container שיש להם גם פתיחה וגם סגירה אפשר לעשות להם appendChild. ל-img למשל אין ילדים אז אי אפשר לעשות.

אז מה עשינו בעצם? יצרנו card שמכיל את ה-innerHTML של כל ספר ולו עושים כל פעם append לתוך ה-booksContainer.

עכשיו נוסיף כל מה שעשינו לתוך הפונקציה:

Function createNewBook(book){

Const card = document.createElement(‘div’); - יצרנו אלמנט

card.classList.add(‘card’, ‘col-3’);

card.innerHTML = `

<img src=${book.imageUrl} class=”card-img-top” alt=${book.title}/>

<div class=”card-body”>

<h5 class=”card-title”>${book.title}</h5>

<p clas=”cad-text”>${book.description}</p>

</div>

<ul class=”list-group list-group-flush”>

<li class=”list-group-item”>${book.author}<li>

<li class=”list-group-item”>${book.price} &pound;<li>

</ul>

`

Return card;

}

איך גרמנו לזה להיות דינמי לפי הפרמטרים שבחנות הכניסו כל פעם בטופס לספר חדש? מחקנו את ה-values הקבועים ששמנו ואנחנו רוצים שבכל card שהוא מייצר, הוא מכניס את החלקים הרלוונטים מתוך האלמנט book וכך כל פעם נוצר לנו card שמתאים לפרטים של הספר אותם הקלדנו. ב-price השארנו את האייקון של סוג המטבע בכוונה.

אנחנו רוצים כל פעם להציג את books. עשינו books.push לכל ספר חדש שנכנס ואנחנו רוצים להגיד לו שבכל פעם שהוא מכניס ספר חדש, שירנדר מחדש את כל הרשימה.

הוא מוחק מקובץ ה-html את כל חלק הקוד שהזרקנו בתוך ה-innerHTML של הצגת כרטיס לספר. לתוך הפונקציה של הוספת ספר הוא מוסיף רינדור:

Function addBookToList(book){

books.push(book);

renderBooks();

}

נוסיף למשתנים הגלובלים את:

Const booksContainer = document.querySelector(‘#booksContainer’)

עכשיו נכתוב את הפונקציה לרינדור:

Function renderBook(){

booksContainer.innerHTML = “”;

books.forEach(book => {

Const newBook = createNewBook(book);

booksContainer.appendChild(newBook);

});

}

כל המטרה של פונקציית הרינדור היא לאפס את הקונטיינר, ליצור card עבור כל האובייקטים book שנמצאים בתוך מערך books ולהכניס אותו לתוך הקונטיינר. ראינו שזה עובד.

עכשיו נעיף מה-HTML את כל ה-values הדיפולטיבים ששמנו בהתחלה.

בשלב הבא, הוא עושה שינוי בפונקציה של handleSubmit:

Function handleSubmit(event){

event.preventDefault();

Const book = {};

Const form = event.target;

Const formData = new FormData(form);

formData.forEach(function (value, key){

Book[key] = value;

})

addBookToList(book);

form.reset();

}

כשמעתיקים url של תמונה, מוחקים בד״כ את הפרמטרים שמגדירים גודל וכו׳.

כשניסה להכניס מחיר 18.03, קיבל שגיאה כי השדה כנראה יודע לקבל רק מספרים שלמים. הוא הולך ל-input של מחיר ומוסיף אחרי ה-name גם attribute של step="0.01" שמגדיר לאינפוט בכמה יכול להתקדם בכל פעם.

עכשיו הוא מוסיף עבור כל האינפוטים attribute של required בסוף התגית.

הוספה של כפתור remove all books:

בקובץ HTML מוסיף תחת ה-div של booksContainer

<button type=”button” class=”btn btn-danger col-2” id=”removeBooks”>Remove all</button>

עוברים לקובץ JS:

Const removeBooksBtn = document.querySelector(‘#removeBooks’); - מגדירים משתנה גלובלי

removeBooksBtn.addEventListener(‘click’, handleRemoveAll); - מוסיפים קריאה לפונקציה בלחיצה על הכפתור

עושים שינוי של const books ל-let books כדי שיהיה אפשר לאפס אותו

Function handleRemoveAll(){

Books = [];

renderBooks(); - מאפס את הקונטיינר ואז המערך ריק ולכן לא יציג כלום

}

הריץ וגילה שאחרי שלחץ על remove all זה מחק גם את הכפתור. הסיבה היא שכנראה רוקן את מה שהיה בתוך התגית של booksContainer שכלל גם את הכפתור. הוא מוציא את הכותרת ואת הכפתור מתוך ה-div הזה ומכניס את כולם באותה רמה בעץ:

<div>

<h2>Most popular books</h2>

<button type=”button” class=”btn btn-danger col-2” id=”removeBooks”>Remove all</button>

<div class=”row” id=”booksContainer”></div>

</div>

בשלב הבא רוצים לשמור את כל הספרים גם אם האתר מרופרש. כרגע אין לנו בקאנד והכל קורה בצד הלקוח. Local storage הוא אובייקט שיכול לשמור סטרינגים בצורת key, value. ברגע שנשמור כל פעם את האובייקט של books, נעדכן גם בדפדפן, גם נעדכן את האלמנט וגם את ה-gallery מחדש.

Local storage נשמר בדפדפן של הלקוח אלא אם הוא ניקה את ה-cache. יש sessionStorage שמתנקה בכל פעם שה-session נסגר.

הוא בן של window (הדפדפן). נכתוב localStorage בקונסול לראות אם מכיל משהו.

localStorage.getItem(“lastname”, “Smith”);

אפשר להשתמש ב-getItem לקבל מה שקיים או ב-setItem כדי להוסיף לו דאטה.

הוא מנסה להכניס את books שהוא מערך לתוך localStorage אבל לא מצליח. כי זה אובייקט שבתוך האובייקט יש אובייקט (הוא מחזיר object object). אם לא הפכנו את הדאטה לסטרינג, הוא לא יכול לקבל אותו. צריך להשתמש ב-stringify שזו אופציה של ספריית JSON - Java Script Object Notation.

JSON.stringify(books) -

מקיף הכל בגרש חיצוני בודד בהתחלה ובסוף וגם כל key, value שם בגרשיים.

עכשיו נרצה לשמור את זה ב-localStorage:

localStorage.setItem(‘books’, JSON.sringify(books))

אם רוצים להחזיר חזרה לסטרינג, נשתמש ב:

JSON.parse(localStorage.getItem(‘books’))

אנחנו רוצים בכל פעם שמאתחלים את הקוד, לבדוק האם יש משהו ב-localStorage, אם יש משהו, תכניס אותו ל-books, ואם יוצרים חדש, להוסיף אותו ל-localStorage.

נשנה את המשתנה הגלובלי books ממערך לפונקציה:

Let books = initBooks()

בנוסף אנחנו רוצים לתת const BOOKS\_LIST = 'books'; כדי שתמיד נשתמש בשם קבוע.

הפונקציה

Function initBooks(){

Const localBooks = localStorage.getItem(BOOKS\_LIST);

if(localBooks){

Return localBooks;

}

}

עכשיו החליט לשנות את books המקורי ל-booksList ואת BOOKS\_LIST ל-BOOKS. הוא עושה את זה באמצעות rename symbol או refactor.

Function initBooks(){

Const localBooks = localStorage.getItem(BOOKS);

if(localBooks){

Return localBooks;

}else {

Return [];

}

}

עכשיו צריך לוודא שכשמעדכנים את booksList אז מעדכנים גם את ה-localStorage. רוצים ליצור פונקציה:

Function updateStorage(booksList){

Const booksString = JSON.stringify(booksList); תקבל אובייקט שאני רוצה להפוך לסטרינג

localStorage.setItem(BOOKS, booksString);

}

נעדכן גם את addBookToList

Function addBookToList(book){

booksList.push(book);

updateStorage();

renderBooks();

}

ונעדכן גם את handleRemoveAll

Function handleRemoveAll() {

boosList = [];

updateStorage();

renderBooks();

}

הוא מנסה להריץ ויש באג ש-booksList הוא לא אובייקט ואי אפשר לעשות לו push. אחרי שבדק גילה שהבעיה היא שב-initBook לא עשה parse. עדיף לעשות את ה-parse רק אחרי שאנחנו יודעים שיש משהו ששמור ב-localStorage ואז localBooks הוא לא משתנה ריק. כי אם קודם ננסה לעשות parse כשאין שם כלום, יכול לעשות לנו בעיה ולהיות undefined.

Function initBooks(){

Const localBooks = localStorage.getItem(BOOKS);

if(localBooks){

Return JSON.parse(localBooks);

}else {

Return [];

}

}

כשמרפרש את העמוד אחרי שהכניס כמה ספרים, אנחנו רואים בקונסול שאכן ב-booksList יש ספרים אבל לא מוצגים בעמוד וזה כי אף אחד לא אמר לו לרנדר. נקרא ל-renderBooks(); אחרי ההגדרה של כל המשתנים הגלובלים.

**Full stack 18 - Git - 2/10/2022**

הכנסה של value מראש יכול לחסוך לנו הרבה בדיקות.

בפונקציה של updateSotrage יש

History = [...localHistory, item];

סינטקס כזה הוא כמו push שמבקש לקחת את המערך הקיים ורוצה להוסיף לו את item.

בפונקציה של renderItem הוא לא השתמש ב-append אלא ב-innerHTML כי רצה לדרוס את מה שהיה קודם ולהזריק card חדש.

מה זה GIT?

יוס עובד יחד עם מישהו נוסף על אותו form. צריך למשל שכל פעם שמישהו מחפש תז של אדם, זה מחזיר הרבה פרטים שלו, תמונה וכו׳. נניח שהמשימה מחולקת בין אנשים, מישהו אחד עושה את ה-frontend והשני עושה את ה-backend.

כמו גוגל דוקס, אפשר לעשות version control.

בדור ראשון עשו ניהול בתיקיות. בדור שני אמרו שבמקום לשמור הכל בצורה לוקאלית, ניקח DB ונשמור את כל הגרסאות שם. הדור השלישי כבר יצר מצב שכל הגרסאות נמצאות על שרת אחד וכל אחד יכול לעבוד על גרסה ולדחוף לשרת. הדור הרביעי אמר שלא רק שיש שרת שמנהל את כל הגרסאות, אלא כל מחשב שומר את כל הגרסאות אצלו ואז אפשר לעבור בין הגרסאות ולדחוף את הכל חזרה לשרת ולהחליט איזו גרסה תהיה.  
צריך תמיד לסנכרן את כל הקבצים שיש על המחשב כי אולי מישהו אחר עבד על הגרסה שהייתה בשרת.

ב-GIT יש עץ של גרסאות, יש גזע מרכזי. אם רוצים לשנות את הכפתור השלישי, את תצא מהגזע המרכזי, תקבל עותק של הגרסה המרכזית ותעשה שינויים בענף branch. בסוף חייבים לחזור לענף המרכזי.

מפעילים את GIT באמצעות פקודות: תתחיל לעקוב, תעשה שינוי וכו׳.

הוא פותח תיקיה בשם git ומראה את הסימן של git ב-vs code. צריך לעשות אתחול לתיקיה, אפשר ע״י file → open folder → git ואז עושים initialize repository.

Dir (win) OR ls (mac)

Cd \_\_\_\_\_

הוסיף תת תיקיה example\_init וכתב git init.

אנחנו יוצרים קובץ html ורוצים שה-git יעקוב אחריו.

Git status

הוא אומר לו שיש לו בעיה כי לא הגדרת מי אתה, מה היוזר, מה המייל וכו. למה צריך את זה? כדי להדביק לכל שינוי, מי הבנאדם שעשה אותו.

אם רוצים לשנות קונפיגורציה באופן כללי, צריך לתת דגל global.

Git config --global

בתוך כונן C

Mkdir example\_init

Cd example\_init

Git init

Git status

- On branch master

- No commits yet

- nothing to commit

Git config - נותן את כל האפשרויות שיש לקנפג

Git config -l OR git config --list

Git config --global user.name yosisrael

Git config --global user.email [yosisrael@gmail.com](mailto:yosisrael@gmail.com)

Git config -l יציג לנו את הקונפיגורציות שהגדרנו

שימוש ב-git הוא למעשה local environment. לעומתו gitHub הוא השרת המרוחק.

Dir /a:d

Nul >index.html יוצר קובץ חדש

Git comit -m “Initial state” OR git commit -m “Created new html files and gitignore”

אם רוצים לראות את השינויים משלב ה-staging לשלב ה-comit אפשר להשתמש ב-git log.

הפעולה הזו של comit הכניסה (מתחייבת) על השינויים בקובץ הלוקאלי.

ירוק - הוספתי קובץ חדש.

כתום - קובץ שהיה תחת מעקב, עבר שינוי.

Git status -

בודק את השלב בין ה-working directory לבין ה-staging כלומר הקבצים ששונו באופן לוקאלי אבל עדיין לא הועברו לשינוי אצל השרת ב-git.

Git diff

שלב ה-directory הוא השינויים שעושים כל הזמן. שלב ה-staging אומר שמסמנים את השינויים המועמדים ל-comit ושלב ה-comit הוא השלב שבו מבוצעים השינויים.

אם עושים commit ואז עושים שוב status נקבל תשובה nothing to commit, working tree clean וזה בגלל שכבר נעשו השינויים ע״י commit ומאז לא היה שום שינוי.

אם רוצים לראות מה השתנה ב-commit אפשר להשתמש ב-git log. כל שינוי מצוין עם תאריך, שעה, יוזר מבצע.

אחרי הפסקה של 20.30

חיבור לשרת

C:\> Git remote add origin <https://github.com/yosisrael/example_init.git>

Git push -u origin master

הוא מקבל permission denied. למה? כי אנחנו לא יכולים לבוא ולדחוף מה שרוצים לכל תיקיה שרוצים.

20:47 התחבר מהמחשב הפרטי שלו ומדגים את כל השלבים.

אם רוצים לשכפל תיקיה מ-gitHub

C:\users\Jbt> mkdir clone\_example

Cd clone\_example

Git clone <https://github.com/yosisrael/example_init.git>

???

Clone\_example

Cd example\_init

Dir

Example\_init

Git pull

Git\_init - המטרה שלו היא להצהיר שמעכשיו התיקיה הזו תהיה במעקב של git