בס"ד

HTML + CSS

ספר קורס

בס"ד

©

כל הזכויות שמורות למחברת

אין לשכפל, להעתיק, לצלם, להקליט, לתרגם, לאחסן במאגר מידע, לשדו
או לקלוט בכל דרך אחרת כל חלק שהוא מהחומר בספר זה.

שימוש מסחרי מכל סוג שהוא בחומר הכלול בספר זה אסור בהחלט אלא
ברשות מפורשת בכתב מהמחברת.

מהדורה שלישית. תשפ"ה

פרק מספר 1

מבוא

מבוא לבניית אתרים

best practice

עבודה עם visual studio code

מבוא

איך עובד אתר?

האינטרנט היא רשת מחשבים המקושרים ביניהם בכל העולם (www = world wide web). הרשת מכילה שרתים ולקוחות.

שרתים - הם מחשבים מרכזיים המאחסנים את האתרים השונים.

לקוחות - אלו משתמשי הקצה. המחשב ממנו אני גולשת לדוגמה. התוכנה שבאמצעותה פותחים אתרי אינטרנט מכונה דפדפן. ישנם מספר סוגי דפדפנים. הנפוצים על מחשבים שולחניים הם chrome, firefox, edge ועל מחשבי Mac הדפדפן הנפוץ הוא Safari.

כאשר גולש נכנס לאתר, הדפדפן שלו (הלקוח) שולח קריאה לשרת שיציג את העמוד המבוקש. השרת עובד עם קוד בשפת צד שרת, לדוגמה c#, JAVA, PHP וכדו'. הקוד שולף נתונים מהDB (באמצעות SQL), ומבצע עליו חישובים. בסופו של דבר השרת מייצר קוד בשפת HTML ומחזיר אותו ללקוח.

כלומר - אתר אינטרנט מורכב מ2 חלקים:

- **צד שרת** - נקרא בשפה המקצועית back end או server side - שכאן יש מגוון שפות שתפקידן לייצר קוד HTML.
- **צד לקוח** - front end או client side - שכאן יש 3 שפות קבועות בלבד: HTML, CSS, JS.

בקורס שלנו אנחנו לומדות את שלושת השפות של צד לקוח.

שפות צד לקוח

HTML - HyperText Markup Language

ובתרגום חופשי - שפת סימנים וקישורים.

HTML היא שפה המורכבת מתגיות, ותפקידה להגדיר **מה מכיל העמוד**. לדוגמה כותרות, טקסטים, תמונות, קישורים, טבלאות וכדו'.

HTML5 היא הגרסה החדשה ביותר של שפת HTML, והוסיפה אפשרויות חדשות בבניית עמוד, כמו הכנסת וידאו/אודיו, תגיות ומאפיינים נוספים ועוד.

הקורס כולל את התוספות שמביאה HTML5, וזו צורת העבודה המקובלת בשנים האחרונות.

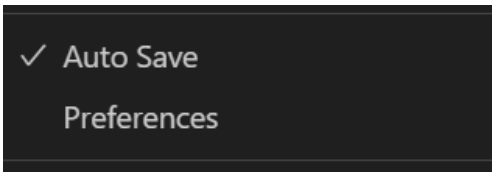
CSS - Cascading Style Sheets

ובתרגום חופשי - גליון סגנונות מדורג.

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

כאשר יש קבצים שלא נשמרו - את תראי בסרגל השמאלי, על האייקון של הקבצים, עיגול כחול קטן ובו מספר הקבצים שעדיין לא נשמרו. משהו כזה:

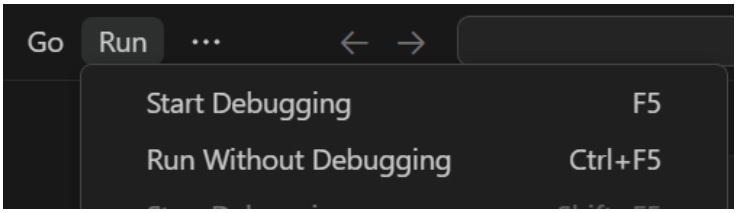
מומלץ מאד להגדיר **שמירה אוטומטית**. עליך להכנס ל"File" ושם "Auto Save". כאשר שמירה אוטומטית מופעלת - יופיע סימון V באותה שורה:



הרצת הפרויקט בדפדפן

יש כמה דרכים להריץ פרויקט בVSC. להלן הדרך הפשוטה ביותר, שגם לא דורשת התקנת תוסף.

כדי להריץ את הקוד, נעמוד על קובץ הHTML שלנו, ואז בחלונית נכנס לRun ומתחתיו נלחץ על Start Debugging.



במידה ואת חוזרת לקוד כדי לערוך שינויים - אין צורך לסגור את הדפדפן ולפתוח מחדש בכל פעם. הדפדפן נשאר פתוח, ולאחר שינוי בקוד את פשוט מרעננת באמצעות לחיצה על F5 או על כפתור הריענון שמופיע בשורה העליונה בדפדפן.

שמירה על קוד מסודר

חשוב לשמור על קוד מסודר, הן בקוד HTML, הן בקוד CSS וגם בהמשך - בקוד JS.

כדי לסדר את הקוד אפשר ללחוץ לחיצה ימנית של העכבר (כשאת נמצאת על החלון של הקוד), ואז Format document. לחיצה עליו תארגן את הקוד בצורה מסודרת, כולל הזחות ומרווחים.

בחלק מהגרסאות של VSC לא מותקן formatter לקוד CSS, ויש להתקין אותו דרך ההרחבות. חפשי css formatter.

פרק מספר 2

HTML

תחביר - תגיות ומאפיינים

תגיות שימושיות לבניית עמוד

תגיות HTML5

מאפיינים שימושיים

טפסים

טבלאות

HTML - תחילת עבודה ותחביר

מסמך HTML נשמר בסיומת html. לדוגמה homepage.html.

כל מסמך HTML מורכב מאלמנטים. האלמנטים מיוצגים באמצעות תגיות (tags) ומאפיינים (attributes).

התגיות הן הדרך שלנו לומר לדפדפן מה יש בעמוד. הדפדפן לא מדפיס לעמוד את התגית עצמה אלא את התוכן או המשמעות שלה. התגית אומרת לדפדפן מה סוג הרכיב שעליו להציג בעמוד. האם מדובר בכותרת, בקישור או בתמונה וכדו'.

שמות התגיות הן מילים שמורות שהדפדפן יודע לקרוא ולהבין, ולכן לא ממציאים שמות של תגיות. (באנגולר וכדו' יש אפשרות לתגיות מותאמות אישית אבל זה לא כלול בחומר שלנו).

המאפיינים נותנים מידע נוסף על התגית. לדוגמה בתגית קישור - נגדיר מאפיין של לאן להפנות בעת לחיצה, בתמונה נגדיר מה הכתובת של התמונה וכדו'. ישנם מאפיינים שלא מעניינים את הדפדפן אלא עוזרים לנו כמתכנתות.

סוגי תגיות

ישנם שני סוגים של תגיות:

- תגיות מכולה - תגיות בעלת חלק פותח וחלק סוגר. בין שני החלקים יופיע התוכן של התגית. לדוגמה - תגית כותרת, תגית פסקה ועוד.
- תגית בודדת - תגית ללא תוכן, ואין לה חלק סוגר.

רוב התגיות בHTML הן תגיות מכולה.

כל תגית HTML תמיד תתחיל ותיסגר עם הסימנים < >.

מבנה תגיות מכולה

לדוגמה:

```
<h1>My Title</h1>
```

החלק הפותח - מתחיל ב < ואז שם התגית. אחר כך יבואו מאפיינים במידה וקיימים וסגירת החלק הפותח באמצעות >.

בין החלק הפותח והסוגר יופיע תוכן התגית. זה יכול להיות טקסט כמו בדוגמה, וזה יכול להיות תגיות נוספות. לדוגמה:

```
<div>
  <h1>My Title</h1>
```

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

```
</div>
```

בדוגמה יש תגית מסוג div, שמכילה בתוכה תגית h1. ה1h מכילה את הטקסט של הכותרת. המבנה הזה נקרה "מבנה מקונן" או באנגלית nested, כלומר - פריט אחד בתוך השני. זה מה שמאפשר לנו גמישות רבה במבנה העמוד וביצירת רכיבים. לדוגמה - תגית של טופס ובתוכה מספר תגיות של פקדים להזנת נתונים.

כאשר נדרשים מאפיינים לתגית מכולה - הם לעולם יופיעו רק בחלק הפותח של ה תגית.

התגית הסוגרת מאופיינת ב/ שמופיע לפני שם התגית. כך הדפדפן יודע שהגיע הזמן לסגור את התגית שנפתחה.

מבנה תגית בודדת

```
<br />
```

תגית בודדת לא מכילה תוכן, ובנויה באופן הבא:

קודם כל הסימן <, אחריו שם התגית, אחר כך מאפיינים במידה וקיימים, ובסוף - הסימן / ואחריו סגירת התגית באמצעות >.

דוגמה נוספת:

```
<img src="mysrc.jpg" />
```

בתגית הזו אפשר לראות את המבנה כאשר יש גם מאפיינים. מתחילים בשם התגית, ממשיכים במאפיינים, וסוגרים תמיד עם />.

מבנה בסיסי של דף HTML

```
<!DOCTYPE html>
<html>
  <head>
   <meta charset="utf-8" />
   <title>Page Title</title>
  </head>
  <body>
   Hello World!!
  </body>
</html>
```

<u>הסבר:</u>

```
</a>
```

מאפיינים חשובים לקישורים

- **href** - כאן נגדיר לאן עוברים בעת לחיצה. מאפיין זה הוא חובה. האפשרויות להגדרת href:
  - **קישור אבסולוטי** - קישור מלא כולל http, ישמש אותנו בעיקר כשאנחנו מפנים לאתר חיצוני. בעולם האמיתי באתרים בדרך כלל משתמשים בקישור אבסולוטי גם כשמדובר על ניווט בתוך האתר עצמו.
  - **קישור יחסי/רלטיבי** - משמש אותנו כשאנחנו מפנים לעמודים אחרים בתוך אותו אתר או פרויקט. כאן נגדיר קישור ביחס לHTML שבו אני נמצאת עכשיו.
  - **קישור עוגן/Anchor** - אלו קישורים שמפנים לאיזורים אחרים בתוך אותו עמוד. לקומה שאליה רוצים להגיע ניתן מאפיין id, לדוגמה contact-floor, ואז בקישור העוגן נגדיר `href="#contact-floor"`, והקישור יפנה לאותה קומה. תוכלי לגשת לנספח "מבנה קישורים" לפירוט מלא של ההתנהגות.
- **target** - האם לפתוח את הקישור באותו טאב, או בטאב חדש.
  - **_self** - הקישור ייפתח באותו טאב. זה ברירת המחדל ובדרך כלל אין צורך לכתוב אותו.
  - **_blank** - הקישור ייפתח בטאב חדש.

הגדרת תמונה כקישור

כדי שתמונה תהיה לחיצה - נעטוף אותה בתגית a. לדוגמה:

```
<a href="home-page.html">
  <img src="images/logo.svg" alt="לוגו האתר" />
</a>
```

רשימות - ol/ul + li

רשימות בHTML עוזרות לנו לארגן פריטים, גם כאשר הפריטים לא נראים כמו רשימה, אבל לוגית הם רשימה. שימי לב שקורא מסך קורא את פריטי הרשימה בצורה מסודרת (פריט 1 מתוך 8), כך שזה עוזר לעיוורים בהבנה של המבנה הלוגי (הם הרי לא רואים את העיצוב).

לכן נשתמש ברשימה כשאנחנו מגדירים תפריטים, גלרית תמונות או רשימת כפתורי סינון וכדו', למרות שהם לא נראים כמו רשימת מכולת.

ישנן שתי סוגי רשימות

- רשימה מסודרת/ממוספרת - Ordered list - התגית נקראת
- רשימה לא מסודרת - Unordered list - התגית נקראת

טפסים

טופס באתר הוא אזור שמסוגל לשלוח מידע ונתונים ממקום אחד למקום אחר. טופס יכול לשלוח נתונים לשרת בכמה אופנים שנפרט בהמשך, אבל ישנם טפסים שהנתונים שלהם משמשים לפרוצדורות ברמת JS, ואינם נשלחים לשרת, לדוגמה טפסי סינון שונים.

דוגמאות לטפסים נפוצים באתרים:

- טופס השארת פרטים - הנתונים יישלחו לשרת לצורך שליחת מייל למנהל האתר וכדו'.
- טופס הרשמה לניוזלטר - הנתונים יישלחו לשרת של חברת דיוור.
- טופס התחברות/הרשמה - הנתונים יישלחו לשרת, ושם ייבדקו מול הטבלאות הרלוונטיות בDB, כדי לאמת את הפרטים האישיים של הגולש.
- טופס חיפוש - מופיע כמעט בכל אתר. בשליחת הטופס הגולש מועבר לעמוד תוצאות חיפוש, שבו מופיעות התוצאות הרלוונטיות מתוך הDB, בהתאם למילת החיפוש שהזין.
- טופס סינון - יופיע לדוגמה בעמודי מוצרים או פוסטים באתר. הטופס מאפשר לגולש לסנן את הפריטים שמוצגים לו בעמוד, כדי לקבל את המידע הרלוונטי ביותר. טפסי סינון לפעמים נשלחים לשרת ולפעמים לא, תלוי באופן שבו נבנה כל העמוד.

ועוד.

ממה מורכב טופס?

כל טופס באתר מורכב מ3 חלקים:

1. הגדרות הקשורות בטופס עצמו - יגדירו לאן הטופס נשלח, באיזו מתודה, האם לבצע בדיקת תקינות לשדות, והגדרות חשובות נוספות.
2. מידע וקלט - חלק זה מורכב מפקדים מסוגים שונים. פקדים (inputs) הם שדות להזנת תוכן על ידי הגולש, ושדות לשמירת מידע מוסתר, שיישלח יחד עם שאר השדות.
3. פעולות - כפתור שליחה, כפתור איפוס הטופס, וכפתורים לביצוע פעולות נוספות (לדוגמה בשדה של סיסמה הקלט מוצג ככוכביות/עיגולים. לפעמים יש שם כפתור שבלחיצה מציג את הסיסמה).

חשוב לציין שכל טופס אמור להכיל את החלק של מידע ואת החלק של מה קורה בעת שליחה, אבל לא בכל טופס יש כפתורים לביצוע פעולות.

הגדרות הקשורות בטופס

את המאפיינים שנפרט מיד יש לתת לתגית הטופס - form.

מאפיינים אלו מאפשרים לנו להגדיר לדוגמה לאן יישלח הטופס, בלי לערב JS כדי שזה יקרה.

שם המאפיין	דוגמה	הערות
action	<form action="thanks.html">	מגדיר לאן הטופס יישלח. יש להגדיר קישור באותו אופן שבו מגדירים קישור רגיל.
method	<form action="..." method="post">	מגדיר את המתודה של השליחה. האפשרויות הן get או post.
novalidate	<form novalidate>	מבטל בדיקות תקינות בטופס.

עוד קצת על בדיקות תקינות מפורט בהמשך.

ישנם מאפיינים נוספים שטופס יכול לקבל, את הרשימה מלאה אפשר למצוא בw3schools. כאן פירטתי את המאפיינים השימושיים ביותר.

כמה כללים חשובים

לפני שאנחנו צוללות לרשימת סוגי השדות, ישנם כמה כללים חשובים שחובה להכיר כשכותבים טופס:

שימוש בname

רק שדות שיש להם את המאפיין name עם ערך כלשהו - ייחשבו כנתונים של הטופס. לכן חובה להגדיר לכל שדה את המאפיין name. תוכלי לראות בכל הדוגמאות שבהמשך איך הוגדר המאפיין name לשדות השונים.

שימי ❤️! המאפיין name מגדיר את שם השדה בטופס, ולכן אסור לתת לשני שדות שונים את אותו name. יוצאי דופן הם כפתורי רדיו וצ'קבוקסים, שעליהם נפרט בעז"ה בהמשך.

במידה ויש באותו עמוד כמה טפסים - אין שום בעיה שיהיה name שמופיע בשני הטפסים, אבל באותו טופס - אסור. אחרת - יישלח רק שדה אחד מהם.

label

תגית label היא תגית HTML שמיועדת לתת את השם/תיאור של השדות השונים. בשביל נגישות - חובה שלכל שדה בכל טופס יהיה label ייעודי. הלייבל מכיל את השם של השדה, לדוגמה "שם פרטי", או הערה שקשורה בשדה.

כדי לקשר בין label לבין השדה שאליו הוא שייך - ניתן לשדה id, ואז לlabel ניתן מאפיין for שמכיל את אותו id של הטופס.

אין שום בעיה שלשדה אחד יהיו כמה label'ים שונים, שכל אחד נותן מידע אחר.

להלן דוגמה:

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

פרק מספר 3
מתחילים CSS

סלקטורים שונים

Pseudo class

הגדרות עיצוב בסיסי לטקסט ובלוקים

display ו-flex

CSS - תחילת עבודה ותחביר בסיסי

תחילת עבודה

כמו שנכתב למעלה, CSS מגדירה את העיצוב של העמוד והרכיבים שלו, כלומר - היא מתכתבת ישירות עם תגי הHTML שבעמוד.

זה עובד בצורה הבאה:

הדפדפן קיבל מהשרת את קוד הHTML ומתחיל לקרוא אותו ולצייר את הפריטים על המסך. במקביל הוא מחפש האם הוגדר לעמוד קוד CSS, ואז מכיל את העיצוב שנכתב בCSS על הפריטים שהוגדרו בHTML.

קוד CSS יכול להיות כתוב בתוך הHTML, בתגית <style>, אבל נכון יותר לכתוב קוד CSS בקובץ ייעודי.

כדי שהדפדפן יידע שעליו לקרוא את הקוד שיושב בקובץ CSS נפרד - עלינו להגדיר בHTML קישור לקובץ הCSS.

לתגית ה<head> נוסיף את ההגדרה הבאה:

```
<link rel="stylesheet" href="path/to/style.css" />
```

זו **link** - תגית בודדת, ללא חלק פותח וחלק סוגר. ובתגית קיימים המאפיינים הבאים:

rel - מגדיר שהלינק הבא משמש עבור עיצוב העמוד. תמיד נכתוב כאן stylesheet.

href - הנתיב והשם של קובץ הCSS. הנתיב יוגדר כמו שמפורט בפרק "מבנה קישורים".

בעמוד HTML אחד יכולים להיות מוטמעים כמה קבצי CSS. יש להקפיד להטמיע את כל הCSS'ים בתוך תגית הhead.

תחביר בסיסי

התחביר של קוד CSS שונה מHTML לחלוטין:

```
selector{
  property: value;
}
```

בתחילה יש לנו את הסלקטור - על מי בעמוד אני רוצה להגדיר את העיצוב.

אחר כך פותחים בלוק הגדרות באמצעות סוגריים מסולסלים.

בתוך הבלוק נכתוב את כל ההגדרות העיצוביות שנדרשות לאותו סלקטור.

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

בס"ד

בכל שורה נכתוב את שם ההגדרה (לדוגמה צבע, רקע, מסגרת), אחר כך נקודותיים, אחר כך הערך (לדוגמה סגול, עבה, שקוף). כל משפט יסתיים בסימן ; (נקודה פסיק).

ובסוף יש לזכור לסגור את הבלוק.

רק לאחר סגירת בלוק - נוכל לתת את הסלקטור הבא ולפתוח את הבלוק שלו.

דוגמה:

```
h1{
  color: red;
  font-size: 20px;
}
```

הסבר:

במקרה הזה, הסלקטור שלנו הוא תגית H1. כלומר - כל תגית מסוג H1 תקבל את כל סט ההגדרות שנכתבו בתוך הבלוק.

ההגדרות שנתנו לH1 - צבע אדום וגודל פונט - 20 פיקסלים.

סלקטורים של CSS

כאשר אנחנו כותבות הגדרות עיצוביות - הן תמיד מגיעות ביחס לסלקטורים מסויימים. הדפדפן מאתר את כל תגיות הHTML שעונות על אותו סלקטור ומחיל עליהן את הגדרות הCSS.

שם התגית

אפשר לכתוב פשוט את שם התגית. לדוגמה

```
p{
  color: red;
}
```

במקרה הזה - כל תגיות הפסקה - p יקבלו צבע אדום.

קלאס

כדי לומר בCSS את המילה "קלאס" - כותבים נקודה. לכן כדי להחיל הגדרות עיצוביות על תגיות בעלות קלאס מסויים - נכתוב בצורה הבאה:

```
.nav-link{
  color: blue;
  text-decoration: none;
}
```

במקרה הזה - כל הקישורים שיש להם את הקלאס nav-link יקבלו צבע כחול ויהיו ללא קו תחתי.

מזהה - id

כדי לומר בCSS את המילה "id" - כותבים סולמית - #. לכן כדי להחיל הגדרות עיצוביות על האלמנט שיש לו את מזהה כלשהו - נכתוב בצורה הבאה:

```
#readmore-button{
  color: white;
  background-color: yellow;
}
```

במקרה הזה - הקישור שנקרא readmore-button יקבל צבע לבן ורקע צהוב.

הכל - *

יש אפשרות להגדיר סלקטור שיכלול את כל סוגי התגיות, בלי להיות ספציפיים. לדוגמה:

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

הגדרות CSS בסיסיות

direction - כיוון כתיבה

כברירת מחדל דפי אינטרנט נקראים משמאל לימין. הכוונה לא לאיפה ממוקם הטקסט בעמוד אלא לכיוון הכתיבה. בעברית נידרש להחליף לrtl.

האפשרויות הן:

- rtl - מימין לשמאל (right to left)
- ltr - משמאל לימין (left to right)

עיצוב טקסט

כל ההגדרות שלהלן עוזרות לנו לעצב טקסטים, פסקות וכדו'. לא כל ההגדרות נמצאות כאן עדיין. את אלו שחסרות - נלמד בעז"ה בהמשך ותקבלי בנפרד.

text-align - יישור טקסט

זו ההגדרה שתיישר את הטקסט ימינה, שמאלה או למרכז בתוך המכולה שלה.

האפשרויות הן:

- right
- left
- center
- justify - מיישר את הטקסט בפסקה לשתי הכיוונים, כמו שזה מיושר בטור בעיתון
- start - מיישר לתחילת הפסקה.
- end - מיישר לסוף הפסקה

ההגדרות start/end תלויות בdirection, בכיוון הכתיבה. אם הפסקה מוגדרת כמיושרת RTL - הרי שstart יתן יישור ימינה, וend יתן לנו יישור שמאלה. אם הפסקה מוגדרת עם ltr - גם start + end יתנהגו בהתאם.

הגדרות אלו מאד שימושיות כשבונים אתר שמיועד להיות כאתר עם עמודים בשתי שפות. אם ניישר עם text-align: right - הרי שבאנגלית נצטרך לדרוס את זה עם text-align: left.

אבל כשמשתמשים בstart או end - הצד שאליו הטקסט מיושר תלוי ישירות בשפה של האתר. אתר שמוגדר rtl יהיה מיושר לימין, וכשנחליף לltr - היישור יתהפך באופן אוטומטי.

color - צבע טקסט

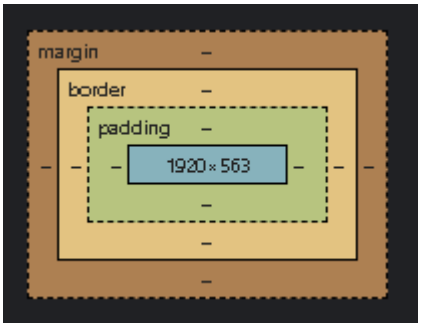
אפשר להגדיר שם צבע - red, ואפשר להגדיר מספר צבע במבנה הקסה דצימלי עם סולמית בהתחלה - #ffffff. תוכלי לראות בנספח פרק שעוסק בנושא הגדרת צבעים בCSS.

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

box-model

הbox-model הוא לא הגדרת CSS אלא הבנה של מבנה האלמנטים.

כל אלמנט בנוי מכמה חלקים:



- **אזור התוכן** - החלק הפנימי - התכול. כאן נכנסים כל התכנים שבתוך האלמנט. זה יכול להיות טקסטים ויכול להיות תגיות HTML אחרות.
  גודל אזור התוכן יכול להקבע על ידי הגדרות width, height, או להיות בהתאם לאורך/רוחב התוכן.
- **אזור הpadding** - החלק הפנימי מרופד על ידי האיזור של הpadding - החלק הירוק בתמונה. אזור הpadding כבר מקבל את הרקע שהוגדר לפריט. העובי נקבע לפי הpadding שהוגדר לאלמנט.
- **border** - המסגרת עוטפת את התוכן והשוליים הפנימיים.
- **אזור השוליים - margin** - החלק של השוליים החיצוניים. כבר מחוץ למסגרת של האלמנט, מחוץ לרקע שלו. כאן נוכל לשלוט על המרחק בינו לבין אלמנטים אחרים בעמוד.

ההתנהגות של אורך/רוחב/שוליים - תלוי בdisplay של האלמנט.

margin - שוליים חיצוניים

יש אפשרות להגדיר שוליים חיצוניים לאלמנטים. כמה יהיה המרחק בין האלמנט לבין אלמנטים אחרים בעמוד. margin נמצא מחוץ למסגרת של האלמנט.

כמו בעובי של מסגרת, גם כאן אפשר לתת הגדרות בכל יחידת מידה שהיא, ואפשר להגדיר בין 1-4 ערכים. ראי בנספח על הגדרת 1-4 כיוונים.

```
<div id="first-div">Hello</div>
<div id="second-div">Bye</div>
```

```
#first-div{
  margin-bottom: 50px;
```

CSS Flex

קצת רקע

flexbox layout, או בקיצור - flex, הוא מודול שמאפשר לנו לעמד, ליישר ולשלוט על מבנה של רכיבים בצורה נוחה ופשוטה מאין כמוה. פלקס מאפשר לנו להגדיר התנהגות לפריטים, גם כאשר לא ידוע לנו מספר או גודל הפריטים.

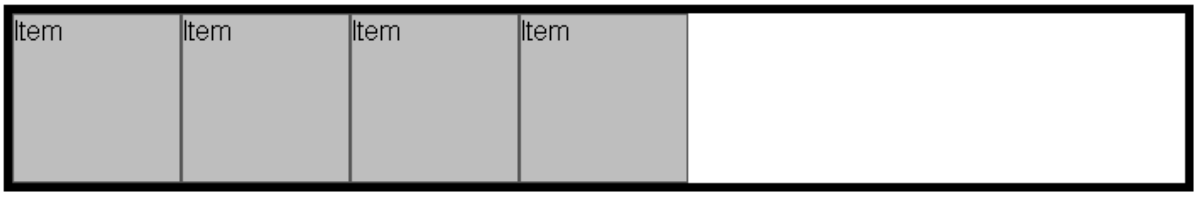
המילה flex היא קיצור של flexibility - גמישות. ואכן, כפי שנלמד מיד, הוא מאפשר לנו גמישות מדהימה באמצעות 2-3 שורות CSS בלבד.

flex הוא אחד הערכים שאפשר לתת עבור display. אבל בשונה מהערכים האפשריים האחרים שלמדנו עד עכשיו - block, inline, inline-block - הוא מגדיר התנהגות לא רק של האלמנט שעליו הוגדר הdisplay, אלא בעיקר את ההתנהגות של **הבנים** של אותו אלמנט.

במילים אחרות, ברגע שנתתי לאלמנט את ההגדרה display: flex, הוא הופך למעין מכולה של פריטים, שיכולה לשלוט על סדר, מיקום, גודל ופריסה של הפריטים שבתוכה.

ההגדרות שנלמד מיד, מתחלקות לשני חלקים:

1. **הגדרות עבור המכולה** - כגון מה יהיה הכיוון של הפריטים והפריסה שלהם על פני המכולה.
2. **הגדרות ברמת פריט** - יש אפשרות לשנות את ההתנהגות לפריט ספציפי מתוך המכולה.



המכולה היא כל מה שעטוף במסגרת השחורה. זה האלמנט שעליו הגדרנו display: flex.

הפריטים - הם הקוביות האפורות שבתוכו.

שימי ❤️, שההגדרות של פלקס משפיעות אך ורק על הבנים הישירים של המכולה, ולא על התוכן הפנימי שלהם.
ועוד נקודה לתשומת ❤️, כי זה יצוץ בהמשך:

כשמדברים על המכולה, יש לה ציר ראשי וציר משני. בתמונה שלמעלה - הפריטים עומדים בשורה, ולכן ציר הX (אופקי) הוא הציר הראשי, ואילו ציר הY (אנכי) - הוא הציר המשני.

כאשר פריטי המכולה יעמדו בטור - זה בדיוק הפוך. הציר הראשי הוא ציר הY (האנכי), וציר הX (האופקי) הוא הציר המשני.

אז נתחיל? קדימה.

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

הגדרות flex עבור המכולה

כל ההגדרות שיופיעו להלן אלו הגדרות שיש לתת למכולה ולא לפריטים.

הגדרות אלו יעזרו לנו לתת התנהגות לפריטים שבתוך המכולה, ועדיין - יש לתת אותן למכולה עצמה. בהמשך יופיעו הגדרות שנותנים לפריטים עצמם כדי להגדיר להם התנהגות שונה ממה שהמכולה הגדירה.

display

קודם כל עלינו להגדיר את הdisplay. אפשר לבחור אחת משתי ההגדרות שלהלן:

1. **flex** - המכולה תתנהג כמו block, תתפוס את הרוחב של אבא שלה, תתפוס שורה שלמה. כמו כל בלוק רגיל.
2. **inline-flex** - המכולה עצמה תתנהג כמו inline. תתפוס את רוחב התוכן שלה, תעמוד בשורה אחת יחד עם inline'ים נוספים וכו'.

על פי רוב נשתמש ב-display: flex בלבד. נדירים מאד הפעמים שבהם נצטרך inline-flex.

flex-direction - כיוון הפריטים

ההגדרה flex-direction מאפשרת לנו לקבוע את הסדר שבו יופיעו הפריטים בתוך המכולה.

האפשרויות הן:

- **row** - ברירת המחדל. הפריטים יעמדו בשורה מההתחלה לסוף.
- **row-reverse** - הפריטים יעמדו בשורה מהסוף להתחלה
- **column** - הפריטים יעמדו אחד מתחת לשני.
- **column-reverse** - הפריטים יעמדו אחד מתחת לשני, אבל מהסוף להתחלה

עבור ההגדרות row, row-reverse, כשאני כותבת "התחלה" או "סוף" הכוונה לפי כיוון הכתיבה הכללי של המסמך.

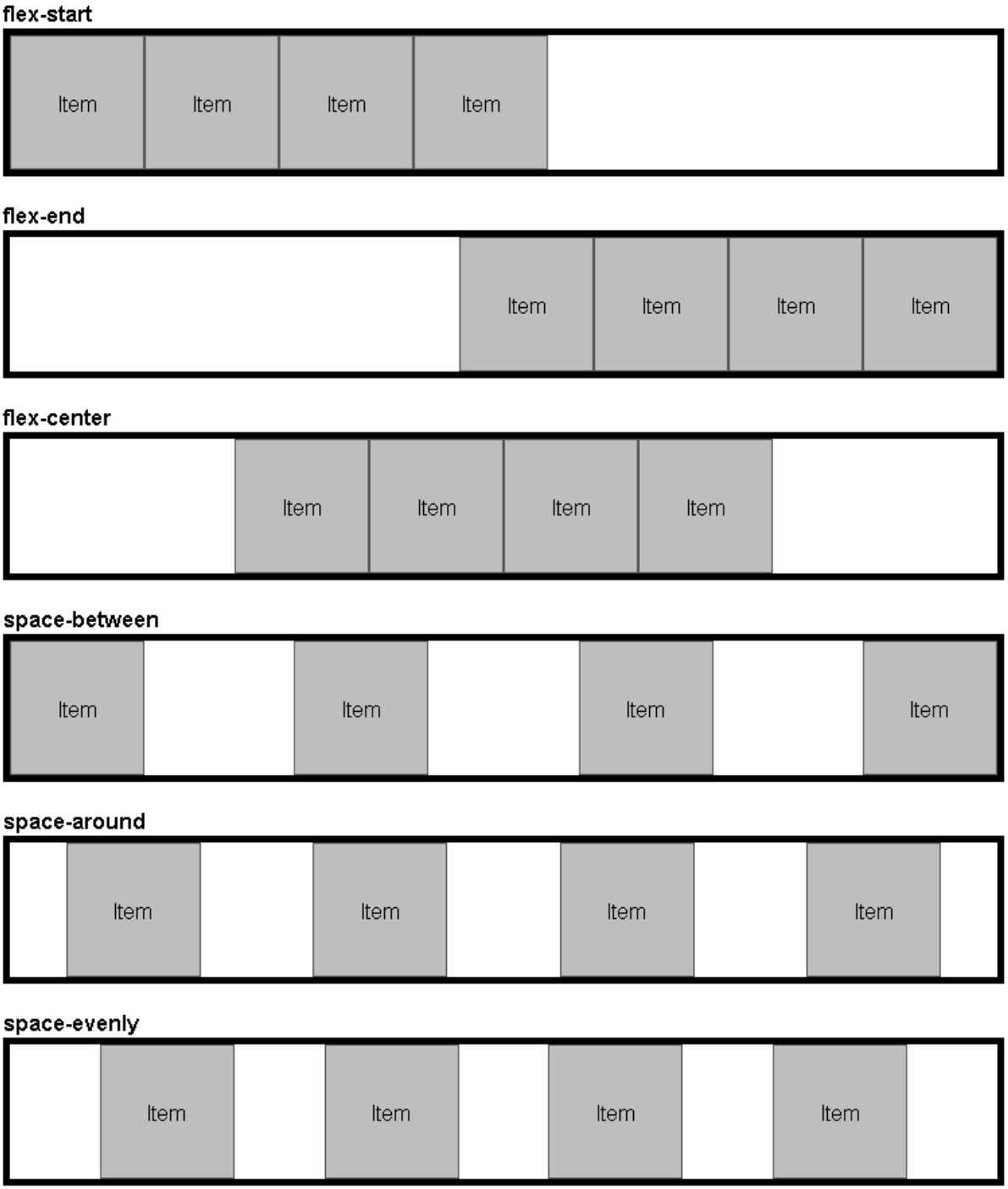
כלומר, אם למסמך הוגדר כיוון כתיבה מימין לשמאל - direction: rtl - ההתחלה היא צד ימין, והסוף הוא צד שמאל.

אבל בעמודים שהוגדרו עם כיוון כתיבה משמאל לימין - direction: ltr - ההתחלה היא צד שמאל והסוף הוא צד ימין.

שימי ❤️ שבאתרים שצריכים תמיכה בשתי השפות - נחסכה לך כאן עבודת התאמה, כי הפריטים תמיד יעמדו לפי כיוון הכתיבה של העמוד.

flex-wrap

כברירת מחדל, כל הפריטים יעמדו בשורה אחת, גם אם יצטרכו לקטון מהרוחב שהוגדר להם. ובמידה ונגמר המקום בתוך המכולה - הם יזלגו החוצה כדי להמשיך שורה ישרה.



align-content

ההגדרה align-content מאפשרת לנו להגדיר את מיקום האלמנטים ביחס לציר המשני, אבל לא אחד ביחס לשני (ברמת שורה/עמודה) אלא כל קבוצת הפריטים ביחס למכולה.

ההגדרה הזו דומה ל-justify-content, אבל בעוד justify מדברת על היישור בציר הראשי, ההגדרה align מדברת על היישור בציר המשני.

מבוא לרספונסיביות

בניית אתרים רספונסיבית, או Responsive web design או פשוט - רספונסיביות, משמעותה בניית האתר באופן שיותאם לכל גדלי המסך. החל ממסכי מחשב רחבים, עבור דרך לפטופים וטאבלטים, וכלה בניידים.

כלומר - אותו קוד HTML נראה באופן מצוין בכל מכשיר שבו גולשים באתר.

כמובן ישנן התאמות, ודברים נראים קצת אחרת, אבל עדיין - אנחנו לא נכתוב את הקוד שלנו כמה פעמים כדי להתאים לכמה גדלים, אלא נמקם את אותם האלמנטים בצורה שונה באמצעות CSS בכל רזולוציה.

רזולוצית מסך מול גודל פיזי. פיקסל מול אינצ'.

כשאנחנו מדברות בשפת CSS, חובה להבין קודם כל שגודל מסך נקבע לפי מספר הפיקסלים ולא לפי הגודל הפיזי.

פיקסל הוא היחידה הקטנה ביותר בתצוגה במסך. כשאני מדברת על פיקסלים אני מדברת על רזולוציה. לא אכפת לי עם המחשב הוא בגודל 3 או 6 אינצ'ים, אלא משנה לי מה הרזולוציה שלו. כמה פיקסלים יש לי ברוחב וכמה בגובה.

שימי ❤️ שאותו מסך מחשב יכול להיות פעם אחת ברוחב של 1920 פיקסלים ופעם אחרת ברוחב של 1500. תלוי ברזולוציה שהוגדרה לו.

במחשבים של ווידוס תוכלי ללחוץ לחצן ימני כשאת בשולחן העבודה, ואז הגדרות תצוגה. יש לך שם אפשרות לקבוע את מספר הפיקסלים שיופיעו לך במסך. ואפשר לראות שבלי להחליף למסך אחר - את יכולה לעבוד על המחשב ברזולוציות שונות.

כמובן שככל שנגדיל את מספר הפיקסלים במסך - נראה את הכל בגודל קטן יותר, כי כשאני מגדילה את הרזולוציה ומוסיפה עוד פיקסלים למסך - אני בעצם מקטינה את הגודל של כל פיקסל, הרי גודל המסך הפיזי לא השתנה לי. לכן דברים שאמורים להיות בגובה 40 פיקסלים - אני אראה פעם בגובה של סנטימטר ופעם בגובה של שלושת רבעי סנטימטר. תלוי ברזולוציה שהגדרתי.

מרגע זה הלאה, שכשתראי את המילים "גודל מסך" - הכוונה לרזולוציה, מספר הפיקסלים שיש ברוחב או גובה המסך, ולא הגודל הפיזי, שכאמור, לא מעניין אותנו בCSS.

רוחבי מסך מקובלים

רוחבי המסך הבאים הם רוחבים סטנדרטיים שעליך להתאים את האתר שלך אליהם.

דסקטופ/לפטופ

- 1920 פיקסלים

- 1360 פיקסלים
- 1280 פיקסלים

טאבלט

- 1024 פיקסלים - רוחב מסך של טאבלט שוכב (landscape)
- 768 פיקסלים - רוחב מסך של טאבלט עומד (portrait)

היום יש מגוון רחב של טאבלטים ברזולוציות גדולות או קטנות יותר מ1024*768. הגדלים שאליהם אנחנו מתייחסות הם הגדלים הסטנדרטיים של טאבלטים.

ניידים

כאן יש לנו מנעד רחב מאד, ויש להתאים ולבדוק כמעט כל רזולוציה החל מ500 פיקסלים ומטה, כי הרזולוציות של הניידים הולכות וגדלות ככל שעובר הזמן. אם לפני 7-9 שנים דברנו על רוחב של 320 פיקסלים, היו מדברים על 360-414 פיקסלים ויותר.

שימי ❤️ שישנם מכשירים נוספים בגדלי ביניים כמו טאבלט מיני וכדו', ולכן האתר צריך להיות מותאם לכל הרזולוציות, ולא רק לגדלים שנכתבו למעלה.

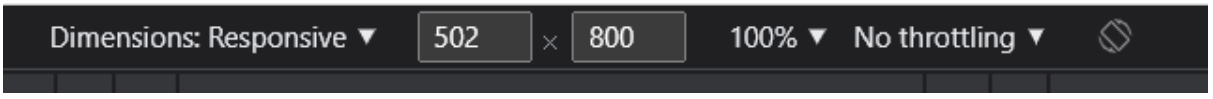
בדיקת רזולוציות שונות בכרום

כדי שתוכלי לעבור על הרזולוציות השונות ולבדוק את הנראות - תוכלי להשתמש בפיצ'ר נהדר שיש לכרום להציע ועונה על 90 ומשהו אחוז מהמקרים.

תפתחי את כלי הפיתוח (F12), ושם תלחצי על הכפתור הקטן עם אייקון של 2 מסכים:

בלחיצה על הכפתור האתר המוצג ייכנס לך לתוך מסגרת שאת יכולה להגדיר את הגודל שלה בשני אופנים.

האפשרויות השונות נמצאות לך בראש העמוד, מעל המסגרת שתוחמת את העמוד:



- גודל משתנה/Responsive - כאן באפשרותך לקבוע את הגודל לתצוגה. שימי לב שהכלי הזה יאפשר לך לבדוק את האתר ברזולוציות גדולות יותר מהרזולוציה של המחשב שלך. אמנם תראי הכל די קטן, אבל תוכלי להתרשם איך העמוד נראה לגולשים האחרים מבחינת מבנה והתנהגות.
  השדה הראשון (משמאל) מגדיר את הרוחב, והשדה השני מגדיר את הגובה.
- בדיקה על רזולוציה של מכשיר ספציפי - אם תלחצי על המילה Responsive, ייפתח לך דרופדאון עם רשימת מכשירים לבחירה. כאשר את בוחרת מכשיר - שטח החלון יותאם

לרזולוציה של אותו מכשיר, כך שתוכלי לראות איך האתר נראה באותו מכשיר ספציפי. זה שימושי מאד אם לקוחה אומרת לך שאצלה בנייד רכיב מסויים לא נראה טוב - תוכלי לבדוק את האתר על אותו סוג מכשיר ספציפי דרך הכלי הזה של כרום.

כלים לפיתוח רספונסיבי

media query

זהו הכלי הבסיסי ביותר כשמדברים על פיתוח אתרים רספונסיבי. בכל פעם שאנחנו מדברות על "הגדרות CSS שונות בהתאם לרזולוציה" בעצם דיברנו על שימוש ב-media query.

media query, או @media - הכוונה לטכניקה בCSS, שמאפשרת לנו להגדיר תנאי אחד או יותר, ואחריו סט הגדרות CSS. כל הגדרות הCSS יתקיימו אך ורק אם וכאשר המסך עונה על התנאים שהוגדרו לפני אותן הגדרות.

התנאים יכולים להיות קשורים לרספונסיביות, אבל לא חובה. אפשר גם להגדיר תנאי של הדפסה. רק כאשר נמצאים במצב הדפסה (רוצים להדפיס את האתר) - הגדרות הCSS יתקיימו (לדוגמה כשמדפיסים - נרצה להסתיר את התמונות שבעמוד כדי לחסוך דיו לגולש).

התחביר של @media הוא פשוט למדי, ושומר על התחביר הרגיל של CSS. לדוגמה:

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

תחילה מופיעה שורת התנאים.

```
@media only screen and (max-width: 600px)
```

במקרה הזה יש לנו 2 תנאים:

1. כאשר צופים במסך (ולא בהדפסה למשל).
2. כאשר רוחב המסך הוא מקסימום 600px. או במילים אחרות - כאשר גודל המסך קטן מ601 פיקסלים.

אחרי שורת התנאים נפתח בלוק באמצעות סוגריים מסולסלים, באותו אופן בדיוק שפותחים בלוק רגיל של הגדרות CSS.

בתוך הבלוק נוכל לכתוב רשימת הגדרות CSS כמו שאנחנו מכירות ורגילות.

ולבסוף נסגור את הבלוק באמצעות סוגריים מסולסלים.

כל ההגדרות שנמצאות בתוך הבלוק יקרו רק כאשר גולשים במסך ברוחב של מקסימום 600 פיקסלים. כאשר גולשים ממסך רחב יותר - ההגדרות הללו לא יחולו על העמוד.

פרק מספר 5
CSS מתקדם

---

position

before & after

transform

transition

משתנים של CSS

transform

אחת ההגדרות המורכבות והמקסימות שCSS3 מביא לנו היא ההגדרה transform.

באופן כללי, ההגדרה מסוגלת לשנות דברים בנראות של סלקטור/ים, אך השינוי הוא רק **בנראות** של האלמנט ולא **במהות** שלו. שינוי חיצוני בלבד.

לדוגמא הגדרת שינוי גודל (scale) – מגדילה את האלמנט כמו בזכוכית מגדלת. כלומר, האלמנט + כל האלמנטים שבתוכו גדלים באחוז מסוים.

מבחינת נראות – נראה שהאלמנט אכן גדל. אך מכל בחינה אחרת האלמנט נותר בגודלו המקורי; האלמנטים שסביבו קולטים אותו בגודל המקורי, וכן פונקציות JS שנועדו למצוא גודל – קולטות את הגודל המקורי שלו ולא לוקחות בחשבון את הטרנספורמציה שעבר האלמנט.

פונקציות הtransform מתחלקות לפונקציות דו מימדיות ופונקציות תלת מימד. נסקור כאן חלק מהפונקציות של transform, הנפוצות שבהן.

את הרשימה המלאה אפשר לקרוא ב w3schools.

רק לפני שאנחנו צוללות פנימה, שימי ❤️שכשנותנים הגדרה באחוזים או בלי יחידת מידה - השינוי יהיה ביחס לפריט עצמו. 1 = 100%. לדוגמה אם ניתן 2 - הכוונה לפי 2 מהגודל המקורי של האלמנט.

למה להשתמש בtransform?

את רוב הטרנספורמציות אפשר לבצע גם באמצעות הגדרות CSS אחרות, אז למה להשתמש דווקא בtransform?

קודם כל - זה כל כך נוח!

אבל בעיקר - כאשר משתמשים באנימציות (transition / animations) - הדפדפן מרנדר את השינויים של transform בקלות רבה יותר מאשר שינויים אחרים.

לכן נשתדל מאד שכשאפשר - נממש את האנימציות באמצעות transform דווקא.

שינוי גודל – scaleX, scaleY, scale

ההגדרה scale מסוגלת לשנות לנו את הגודל של האלמנט.

scaleX - ישנה את הרוחב של של האלמנט.

scaleY - ישנה את הגובה של האלמנט.

scale - ישנה את שתיהם.

<u>תחביר:</u>

```
transform: scaleX(num);
transform: scaleY(num);
transform: scale(num); /*ישנה את הגובה וגם את הרוחב באותו יחס*/
transform: scale(x,y);
```

באפשרות האחרונה - המשתנה הראשון בסוגריים מגדיר בכמה אחוזים להגדיל את הרוחב, והשני – בכמה אחוזים להגדיל את האורך.

אם נגדיר מספר שלילי - הפריט יתהפך על אותו ציר.

<u>דוגמאות:</u>

```
transform: scaleX(1.5); /* יכפיל את הרוחב פי 1.5 */
transform: scaleY(0.5); /* יקטין את הגובה בחצי */
transform: scale(50px); /* יוסיף 50 פיקסלים לגובה ולרוחב */
transform: scale(2,4); /* יכפיל את הרוחב פי 2 ואת הגובה פי 4 */
```

הזזה – translateX, translateY, translate

ההגדרה translate מסוגלת להזיז לנו את האלמנט על המסך. נדגיש שוב, שאר האלמנטים על המסך לא מודעים לתזוזה של האלמנט ומבחינתם הוא נשאר במקום המקורי, לכן נוח להשתמש בהגדרה הזו עבור אפקטים שונים של תזוזה, לדוגמה כניסה של קומה עם מעט תזוזה, תפריט שנפתח ובפתיחה הוא מעט עולה, פופאפ שעולה בכניסה שלו ועוד. כך האלמנט זז אך לא משפיע על הסביבה והאפקט עובד נפלא.

<u>תחביר:</u>

```
transform: translateX(num); /* תזוזה על ציר האיקס */
transform: translateY(num); /* תזוזה על ציר הוואי */
transform: translate(x,y); /* תזוזה שונה לכל ציר */
```

נקודת המוצא היא תמיד הנקודה השמאלית העליונה, ולכן אם נרצה להזיז ימינה – נגדיר translateX עם מספר חיובי, ואם נרצה להזיז שמאלה נגדיר translateX עם מספר שלילי. כנ"ל לגבי למעלה ולמטה. למטה נגדיר מספר חיובי וללמעלה – מספר שלילי.

שימי ❤️ שאם להגדיר translate תגדירי רק ערך אחד בסוגריים - האלמנט יזוז רק על ציר הX ולא על ציר הY.

<u>דוגמאות</u>

```
transform: translateX(0.5); /* יזיז את האלמנט ימינה בחצי מגודלו */
```

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

משתנים בCSS

שפת CSS מאפשרת עבודה עם משתנים למגוון שימושים.

אפשר להגדיר למשתנים ערך שונה ברזולוציות שונות או לסלקטורים שונים, וכך לייצר גמישות גבוהה מצד אחד, ואחידות וקוד קצר מצד שני.

שימושים לדוגמה:

- **צבעים שונים לרכיבים זהים שמופיעים במקומות שונים.**

לדוגמה יש קישור בעיצוב זהה שמופיע בעמודים שונים באתר, אבל בכל עמוד הוא אמור להופיע בצבע שונה.
במקרה הזה נגדיר את המשתנה "color" עם ערך שונה בכל עמוד.
ובעיצוב של הכפתור נשתמש במשתנה כדי להגדיר את הצבע של הכפתור.
באופן הזה הCSS של הכפתור - אחיד בכל העמודים, ועדיין כל עמוד יקבל את הכפתורים בצבעוניות משלו.

- **מרחק קבוע בין קומות/רכיבים בעמוד.**

נדמיין סיטואציה של עמוד ארוך עם הרבה קומות. בין קומה לקומה יש רווח של 180 פיקסל בדסקטופ שהופכים ל90 פיקסלים בלבד במובייל.
בלי משתנים - אצטרך לכתוב בכל קומה בנפרד את המרחק במובייל ובדסקטופ.
עם משתנים - אוכל להגדיר לכל הקומות שהmargin-bottom שלהם הוא לפי הערך של המשתנה, ואז במקום אחד בלבד אגדיר את הערכים של המשתנה בדסקטופ ובמובייל, וכולם יקבלו את הערכים הללו.

תחביר

שמות משתנים בCSS לעולם יתחילו ב-- (פעמיים מינוס). הכנסת הערך נראית כמו כל הגדרת עיצוב בCSS.

כדי להשתמש במשתנה - נכתוב את המילה var ואז בתוך סוגריים את שם המשתנה.

לדוגמה:

```
.my-btn{
  --color: red;
  background: var(--color);
  border: 2px solid var(--color);
  color: white;
}
.my-btn:hover{
```

```
  background: white;
  color: var(--color);
}
```

בדוגמה הזו יש לנו כפתור עם רקע אדום וטקסט לבן. במעבר עכבר הרקע הופך ללבן והטקסט לאדום.

כמו שאפשר לראות, אני מגדירה את הצבע פעם אחת במשתנה color–, ובכל המקומות השתמשתי בו.

עכשיו נוסיף הגדרה, שכאשר הכפתור נמצא בתוך הדיב "hello" - הצבעוניות שלו משתנה לכחול. כל מה שעלינו לעשות זה לשנות את הערך של המשתנה. לדוגמה:

```
#hello .my-btn{
  --color: blue;
}
```

החלפתי את הערך של המשתנה. זה דורס את הערך המקורי (כי כאן אני משתמשת בID וזה שוקל יותר מקלאס, ראי עוד על זה בפרק על משקל של סלקטורים). וכך מתקבל לי כפתור עם אותה התנהגות של הכפתורים האדומים, אבל במקום אדום - הוא כחול.

ערכי ברירת מחדל

ישנה אפשרות, כאשר קוראים למשתנה, להגדיר ערך ברירת מחדל, כך שאם המשתנה לא הוצהר - הדפדפן ישתמש בערך ברירת המחדל. לדוגמה:

```
.my-btn{
  color: var(--color, blue);
}
```

בדוגמה הזו, הצבע של הטקסט בכפתור יהיה מה שמכיל המשתנה color, אבל אם לא הוצהר המשתנה - הצבע יהיה כחול.

דוגמה נוספת:

```
.some-class{
  font-size: var(--size, 16px);
}
```

גודל הפונט באלמנטים עם הקלאס יהיה מה שיש בתוך המשתנה size, אבל אם המשתנה לא הוגדר - גודל הפונט יהיה 16 פיקסלים.

scope של משתנים

כמו שראינו בדוגמאות שלמעלה, יש להגדיר את המשתנים כמו הגדרות CSS, תחת סלקטור מסויים. לדוגמה תחת הסלקטור my-btn.

המשתנה חי ופעיל ומוכר אך ורק בתוך אותו אלמנט HTMLי (אחד או יותר) שעליו הוגדר המשתנה. האלמנט הHTMLי הוא הscope של המשתנה.

בשפות תכנות אחרות מקובל לקרוא לזה גם "אורך חיי משתנה". זה זהה גם כאן. בשפות תכנות אחרות כדוגמת C# או JS, אם יש לי פונקציה ובתוכה הצהרתי על משתנה - ברגע שהסתיימה הפונקציה מת המשתנה ואי אפשר להשתמש בו יותר.

הפונקציה היא בעצם הscope, מרחב המחיה של המשתנה.

אם אנחנו רוצים משתנה שיהיה מוכר בכל הפונקציות - הוא צריך להיות מוצהר כמשתנה גלובלי, הוא להיות מוצהר מחוץ לכל פונקציה אחרת. באופן הזה מרחב המחיה שלו הוא כל התוכנית וכל הפונקציות יכולות להשתמש בו, לערוך אותו ולעדכן לו את הערכים.

גם במשתנים של CSS קיים מרחב המחיה הזה. אבל במקום שמרחב המחיה הוא פונקציה/תוכנית, בCSS מרחב המחיה הוא האלמנט שבתוכו הוצהר המשתנה.

אם הגדרתי את המשתנה בתוך האלמנט my-btn. - רק האלמנט my-btn. וכל האלמנטים שנמצאים בתוכו בHTML יכולים להשתמש בפונקציה.

יותר מזה, אם הצהרתי על המשתנה בתוך אבא, ואז שוב בתוך בן - הרי שבתוך הבן יש למשתנה ערך אחד, ומחוצה לו - ערך אחר כפי שהוגדר באבא. לדוגמה:

```
<div id="parent">
  <div id="child"></div>
  <div id="another_child"></div>
</div>
```

```
#parent{
  --size: 45px;
  height: var(--size); /* height = 45px */
}
#child{
  --size: 30px;
  height: var(--size); /* height = 30px */;
}
#another_child{
  height: var(--size); /* height = 45px */
```

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

פרק מספר 6

נספחים ותוספות

משקל של סלקטורים

עבודה עם צבעים בCSS

הגדרת כיוונים בCSS

שימוש ביחידות מידה שונות

בניית קישורים

המדריך המלא ליישור לאמצע

פתרון בעיות נפוצות

יחידות מידה בCSS

בCSS קיימות כמה יחידות מידה, ששונות זו מזו במשמעות וממילא - בשימוש. עלינו להבין איך עובדת כל יחידת מידה, ומתי נכון להשתמש בכל אחת.

הבנה טובה של המאפיינים של כל אחת מיחידות המידה תעזור לנו לבנות את העמודים בצורה טובה ונכונה יותר, ובעיקר תסייע לאורך זמן - בבנייה דינאמית (מותאמת לתכנים באורך משתנה) ורספונסיבית.

יחידת המידה פיקסל – px

נראה לי ההכי שימושית ומוכרת. כותבים בדיוק כמה פיקסלים רוצים לגודל פונט, שוליים, מרחק, גודל וכדו'. זהו גודל קבוע ולא משתנה בהתאם לגודל המסך, וההגדרה לא מושפעת מהגדרות הדפדפן של הגולש או מאלמנטים אחרים בסביבה. מה שכתבנו זה מה שיופיע בדפדפן.

שימי ❤️ שכמובן ניתן להשתמש בmedia query של CSS על מנת להגדיר גדלים שונים לרוחבי מסך שונים, כך ש**שימוש בפיקסלים אינו סותר בניה רספונסיבית**.

<u>מתי להשתמש בפיקסלים</u>

בכל מצב שרוצים גודל קבוע שאינו מושפע מהסביבה ומרוחב המסך. הדוגמה הכי נפוצה ושימושית היא רוחב הקונטיינר. (לא יודעת מה הוא קונטיינר? חזרי לפרק העוסק בטיפים לפיתוח רספונסיבי).

זו דוגמה מצוינת לשימוש בפיקסלים, כי כפי שנראה בהמשך, שאר יחידות המידה הן תלויות באלמנטים אחרים או ברוחב המסך, ולעולם לא נצא מזה לא בפיתוח ולא בבדיקות.

גם מסגרת דקה (בעובי פיקסל או 2) בדרך כלל כדאי להגדיר בפיקסלים, כדי לא להגיע למצב של מסגרת בעובי חצי פיקסל או פיקסל וחצי, מה שיוצר חוסר אחידות במסך.

ככלל, פיקסלים זו יחידת המידה הנוחה ביותר. היא לא דורשת הבנה מורכבת של איך היא עובדת, ולכן תלמידות בתחילת הדרך תעדפנה להשתמש בה. אבל כאשר מתחילים לדבר על רספונסיביות - שם נכנסות יחידות מידה נוספות.

<u>מתי לא להשתמש בפיקסלים</u>

קודם כל - כאשר עובדים עם rem+vw עבור רזולוציות של דסקטופ. ראי בפרק העוסק ברספונסיביות על צורת העבודה הזו.

מצב אחר - כאשר יש להשתמש באחוזים - לעולם לא נשתמש בפיקסלים. (עוד על אחוזים בהמשך).

יחידת המידה rem – אחוזים מהroot

מהו הroot? – זהו הרכיב הHTMLי שמחזיק לנו את כל שאר האלמנטים. במילים אחרות – התגית html.

כאשר משתמשים בהגדרה rem כיחידת מידה – התוצאה הסופית תהיה שקלול של המספר שנתנו בהגדרה כפול הfont-size של תגית הhtml שלנו.

אני יודעת שזה נשמע מסובך, אבל הנה דוגמה:

```
html{ font-size: 16px; }
#header{ height: 8rem; }
```

בדוגמה הזו הגדרנו לHEADER גובה של 8 פעמים גודל הפונט של הhtml. גובה הHEADER יהיה 16*8=128 פיקסלים.

כלומר - לא משנה את מה אני מגדירה כרגע, האם מדובר בגובה, עובי מסגרת, עיגול פינות וכו'. כאשר הגדרתי באמצעות rem במקום בפיקסלים - החישוב יהיה המספר שנתתי כאן כפול גודל הפונט של הHTML.

שימי ❤️ שכברירת מחדל, גודל הפונט של הHTML הוא 16 פיקסלים, כך שאם את רוצה להשתמש בREM כמו שמודגם בפרק על רספונסיביות - עליך לשנות את ההגדרה הזו.

יחידת המידה %

הנה הגענו לאחוזים. אחת ההגדרות השימושיות מאד.

כאשר אנחנו משתמשות באחוזים, אנחנו קודם כל מדברות על **אחוזים מהאלמנט האב**. לגבי מה באלמנט האב – כאן זה שונה קצת מהגדרה להגדרה. להלן פירוט קצר על ההגדרות הנפוצות:

שימוש באחוזים לקביעת גודל - רוחב וגובה

בכל ההגדרות הללו – אנחנו הולכות על אחוזים של אותה הגדרה מתוך האלמנט האב. לדוגמה – אם אני נותנת לאלמנט width: 80% – הכוונה לקחת 80% מרוחב האבא.

שימו לב שבהגדרות שקשורות לגובה – במידה ולאבא לא הוגדר גובה – אין אפשרות להגדיר לבן שלו גובה באחוזים. כנ"ל לגבי גובה מינימום וגובה מקסימום. גם אם נגדיר – הדפדפן לא יתחשב בהגדרה.

יוצא דופן של יוצא הדופן הזה הוא FLEX. כאשר יש לי אלמנט עם FLEX, גם אם לא הגדרתי לו עצמו גובה - אני יכולה להגדיר לבנים שלו גובה באחוזים.

נשתמש באחוזים כמעט בכל פעם שנדרשת חלוקה לעמודות (חצי-חצי, שלישים, רבע ושלושת רבעי וכו'). בכל המקרים שבהם **לא מעניין אותי הגודל המדוייק אלא החלוקה**.

באופן הזה גם ברספונסיביות היחס בין העמודות יישמר בצורה מדוייקת.

אפשר לומר שגודל באחוזים משמש בעיקר לחלוקה ועימוד, ואילו גודל בפיקסלים/rem משמש לכל מצב אחר.

המדריך המלא ליישור לאמצע

כמו שאנחנו רואות לאורך הקורס, יישור לאמצע ויישור בכלל - תלוי במספר גורמים. ביניהם: הdisplay שמוגדר לאלמנט, האם הוא בן של FLEX או לא, האם יש לו position מסויים או לא.

הפרק מחולק לפי סיטואציות שונות שבהן אנחנו נתקלות בעבודה, ואיך ליישר לאמצע בכל אחת מהן.

במצבים שבהם אפשר ליישר רק אופקית (ימין/שמאל) - יש אייקון כזה ↔ , וכאשר אפשר ליישר לגם אופקית וגם אנכית - יופיע האייקון הבא:

↔ display: block

כאשר לאלמנט עם display:block הוספנו רוחב - הוא עדיין תופס את כל השורה. לדוגמה הוא יתפוס 500 פיקסלים רוחב, אבל עדיין נשארים 500 פיקסלים כשוליים חיצוניים שלו, בצד שמאל או ימין (תלוי בכיוון של המסמך.

כדי ליישר אותו לאמצע אנחנו צריכות לבקש מהשוליים להתחלק חצי חצי בין ימין ושמאל של האלמנט. ולכן נשתמש בauto. לדוגמה:

```
.centeredDiv{
  margin-right: auto;
  margin-left: auto;
}
```

ברגע שהשוליים מתחלקים שווה בשווה - האלמנט עצמו נמצא במרכז השורה.

↔ display: inline/inline-block

אלמנט שמוגדר עם display: inline; או inline-block מתנהג כמו טקסט ולכן ניישר אותו לאמצע כמו שמיישרים טקסט.

נוסיף ל**אלמנט שעוטף את האלמנט שאותו רוצים ליישר** את ההגדרה של יישור טקסט לאמצע. לדוגמה:

```
.items_parent{
  text-align: center;
}
```

position: absolute/fixed

אלמנטים עם ערכי positionהללו הם בעצם אלמנטים שנעוצים לנקודה מסויימת ביחס למסך או ביחס לאלמנט אחר. אי אפשר ליישר אותם עם text-align או עם margin.

ניישר אותם עם transform באופן הבא:

```
.myElement{
  left: 50%;
  top: 50%;
  transform: translateX(-50%, 50%);
}
```

אמרנו לאלמנט שיזוז שמאלה 50% מרוחב האבא שלו. עכשיו הנקודה השמאלית של האלמנט נצמדה לאמצע האבא, ואנחנו רק צריכות להחזיר אותו שמאלה בחצי מהרוחב שלו. לכן השתמשנו בtranslate שמאפשר תזוזה ביחס לרוחב של עצמך.

ובדיוק אותו הדבר גם בציר הY - זוז 50% מגובה האבא שלך, ואז תחזור אחורה 50% מהגובה של עצמך.

עוד על יישור של אלמנטים אלו ואיך זה עובד - ראי בפרק על translate בחוברת מספר 3.

flex

אלמנט שיש לו display: flex מתנהג כמו כל אלמנט עם display: block. החלק הזה מתייחס לאלמנטים שהם **בנים של אלמנט עם flex**. מה שנהוג לקרוא גם flex-items.

כאשר אני רוצה ליישר למרכז את הבנים של הFLEX - נשתמש בהגדרות justify-content: center או align-items: center, או בשילוב של שניהם, תלוי בכיוון של הFLEX עצמו. להלן דוגמה:

```
#flex-container{
  justify-content: center; /* יישור על פני הציר הראשי */
  align-items: center; /* יישור על פני הציר המשני */
}
```

הוספת שני ההגדרות הללו לאלמנט הFLEX עצמו (כלומר, אבא של מי שאנחנו מיישרים לאמצע) - תיישר את התוכן לאמצע גם בגובה וגם ברוחב.

המדריך המהיר לפתרון בעיות

כאן תוכלי למצוא פתרונות אפשריים למגוון בעיות שעשויות לצוץ לך במהלך העבודה.

לכל בעיה יש מספר פתרונות אפשריים, אחרי וידוא כל סעיף תרעננני את העמוד ותבדקי האם הבעיה נפתרה. אם לא - המשיכי לפתרון הבא.

בהצלחה!

הדפדפן לא מציג לי את השינויים שערכתי בHTML

- וודאי שאת עובדת על שולחן העבודה או דיסק-און-קי ולא על השרת המשותף.
- וודאי ששמרת את הקובץ (ctrl+S)
- רעננני את העמוד בדפדפן
- פתחי את הDeveloper tools (באמצעות לחיצה על F12), ואז לחיצה ימנית של העכבר על כפתור הריענון בדפדפן, ובחרי את האופציה השלישית.
- וודאי שאת אכן מריצה את העמוד הנכון. לפעמים מריצים גרסה ישנה ולא את זו שאת עובדת עליה. סגרי את הדפדפן והריצי את הקוד מחדש.

העמוד לא קורא כלל את קוד הCSS שכתבתי

- וודאי שקבצי הCSS + HTML שלך שמורים (ctrl+K S).
- וודאי שבHEAD בקובץ הHTML צירפת את הCSS באמצעות התגית <link>
- וודאי שהסיומת של הקובץ אכן CSS ולא SCC, ואכן כתבת את השם בצורה מדוייקת בתגית link.
- וודאי שהתגית מוגדרת בצורה נכונה וללא שגיאות כתיב:
  - "rel=”stylesheet - קבוע
  - "href=”link/to/style.css - כאן יש לשנות לנתיב והשם של הקובץ שלך

  מומלץ להשתמש בקיצור של VSC כדי ליצור את התגית לינק בלי שגיאות כתיב (כותבים link ואז אנטר, לך נותר רק להזין את שם/נתיב הקובץ).
- פתחי את כלי הפיתוח (לחיצה על F12), ופתחי שם את הטאב "console". בדקי האם מופיעה שגיאה אדומה שקשורה לקובץ הCSS שלך. אולי הנתיב שכתבת לא מדוייק.

הוספתי/שיניתי הגדרת CSS והיא לא מופיעה בדפדפן

סעיף זה רלוונטי כאשר הסעיף הקודם כן תקין. כלומר - הגדרות CSS אחרות כן מופיעות בעמוד.

- וודאי שקבצי הCSS + HTML שלך שמורים (ctrl+K S).

תוכן עניינים

פרק מספר 1 - מבוא.....3
מבוא.....4
איך עובד אתר?.....4
שפות צד לקוח.....4
מבנה הפרויקט, סגנון ותחביר.....5
סביבת עבודה - Visual Studio Code.....7
פתיחת פרויקט חדש.....7
פתיחת פרויקט קיים.....7
שמירת הקוד.....7
הרצת הפרויקט בדפדפן.....8
שמירה על קוד מסודר.....8
קיצורי מקשים שימושיים.....8
עוד קצת על עבודה עם VSC.....10
מה לעשות אם לא עובד לי?.....10
פרק מספר 2 - HTML.....11
HTML - תחילת עבודה ותחביר.....12
סוגי תגיות.....12
מבנה תגיות מכולה.....12
מבנה תגית בודדת.....13
מבנה בסיסי של דף HTML.....13
מאפיינים - Attributes.....14
הערות בקוד HTML.....14
תגיות HTML.....15
כותרות - H1-H6.....15
פסקה - p.....15
שבירת שורה - br.....15
תמונה - img.....15
קישור - a.....17
רשימות - ol/ul + li.....18
div.....20
span.....20
header, footer, main, aside, nav.....21

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה

מאפייני HTML.....22
src - מקור.....22
href - קישור.....22
alt - טקסט אלטרנטיבי.....22
title.....23
id - מזהה.....23
class - קלאס.....23
טפסים.....25
דוגמאות לטפסים נפוצים באתרים.....25
ממה מורכב טופס?.....25
הגדרות הקשורות בטופס.....25
כמה כללים חשובים.....26
סוגי שדות לקלט ושמירת מידע.....27
ואלידציה - בדיקת תקינות.....34
כפתורים.....35
עיצוב שדות בטופס.....36
טבלאות.....38
דוגמה לטבלה בHTML.....38
פירוט התגיות בטבלה.....39
עוד קצת על עבודה עם טבלאות.....40
פרק מספר 3 - מתחילים CSS.....45
CSS - תחילת עבודה ותחביר בסיסי.....46
תחילת עבודה.....46
תחביר בסיסי.....46
סלקטורים של CSS.....48
שם התגית.....48
קלאס.....48
id - מזהה.....48
* - הכל.....48
שימוש במספר סלקטורים לאותן הגדרות.....49
קינון סלקטורים - בניית סלקטור עם היררכיה.....49
כמה הגדרות לאותה תגית.....51
Pseudo Class - מחלקות דמה.....53
מחלקות דמה לקישורים וכפתורים.....53

מחלקות דמה תלויות DOM 54
הסבר על אופן ההגדרה של nth 55
הגדרות CSS בסיסיות 60
direction - כיוון כתיבה 60
עיצוב טקסט 60
פונטים 61
height - גובה 63
width - רוחב 64
min-height | max-height | min-width | max-width 64
object-fit - התאמת גודל תמונה 65
border - מסגרת 65
border-radius - עיגול פינות 67
background - רקע 67
opacity - שקיפות 71
display 73
box-model 75
margin - שוליים חיצוניים 75
padding - שוליים פנימיים 76
box-sizing 76
CSS Flex 78
פרק מספר 4 - רספונסיביות 98
מבוא לרספונסיביות 99
רזולוצית מסך מול גודל פיזי. פיקסל מול אינצ' 99
רוחבי מסך מקובלים 99
בדיקת רזולוציות שונות בכרום 100
כלים לפיתוח רספונסיבי 102
media query 102
התנהגות רספונסיבית מקובלת לעמודים 103
התנהגות רכיבים ברזולוציות השונות 104
טיפים וקיצורי דרך 107
טיפ מספר 1: נקודות שבירה - breakpoints 107
טיפ מספר 2: שימוש בcontainer'ים 108
טיפ מספר 3: הבנה של יחידות מידה בCSS 110
טיפ מספר 4: שילוב של אחוזים, REM ו-VW 110

פרק מספר 5 - CSS מתקדם.....113
position.....113
before + after.....120
transform.....124
scaleX, scaleY, scale – שינוי גודל.....124
translateX, translateY, translate – הזזה.....125
rotate – סיבוב.....126
יותר מטרנספורמציה אחת לאלמנט.....127
transition.....128
משתנים בCSS.....132
פרק מספר 6 - נספחים ותוספות 137
מבנה קישורים.....138
משקל של סלקטורים.....141
יחידות מידה בCSS.....145
הגדרת צבעים בCSS.....148
הגדרת כיוונים בCSS.....149
המדריך המלא ליישור לאמצע.....150
המדריך המהיר לפתרון בעיות.....152

 כל הזכויות שמורות // ח. קוסטליץ // תשפ"ה