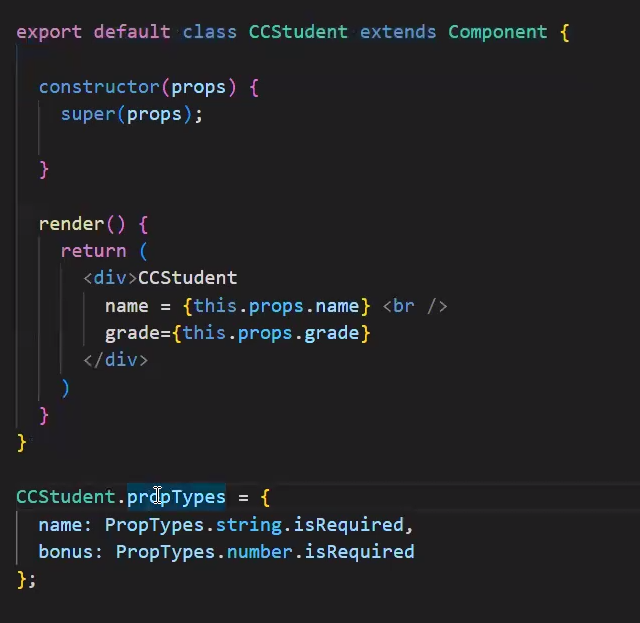
11.02.2024

שיעור 7

חזרה על שיעור קודם:

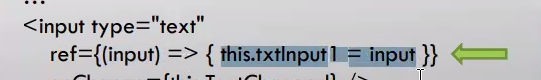
Prop types – צריך להתקין ולכתוב npm I props-types (כמו במצגת)  
וצריך לעשות import

כותבים את זה מתחת לקלאס, ועושים שם של הקומפוננטה נקודה prop types ובתוך JSON לכתוב מה אנחנו מצפים מכל שדה, למשל name: ואז לכתוב string.IsRequired



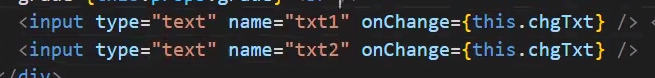
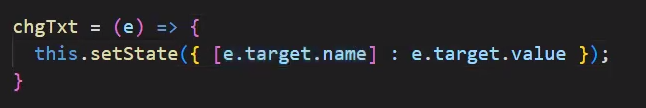
Ctrl c c - זה להפסיק את ההרצה

**Refs**

Ref זה מצביע לקונטרול. כמו שתופסים בDOM בגאווה סקריפט. כדאי לא להשתמש בזה.  
כותבים את זה בשדות של אינפוט מסוים, למשל יש לי אינפוט text ובתוכו כותבים ref={}.  
\* השם שכותבים בתוך הסוגריים הוא בעצם המצביע שנוצר. Ref={input}  
  
אם יצרתי שדה בקלאס ואני רוצה לשמור את האינפוט שלי בתור השדה אז רושמים:  
 => {nameOfField = input}  


\*השימושים לרפס זה למשל כשיש מכשיר צד שלישי כמו למשל חיבור למצלמה.

אפשר לגשת לvalue של האינפוט ששמנו לו רפס כזה, דרך השם של השדה נקודה value  
  
  
**name**

לגשת לאינפוט דרך name:  
  
לגשת לכל אינפוט:  
[e.target.name] = e.target.value  


e.target.name ישמש בעצם כמו מפתח. שנקרא לפי הname.

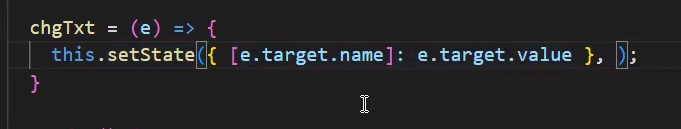
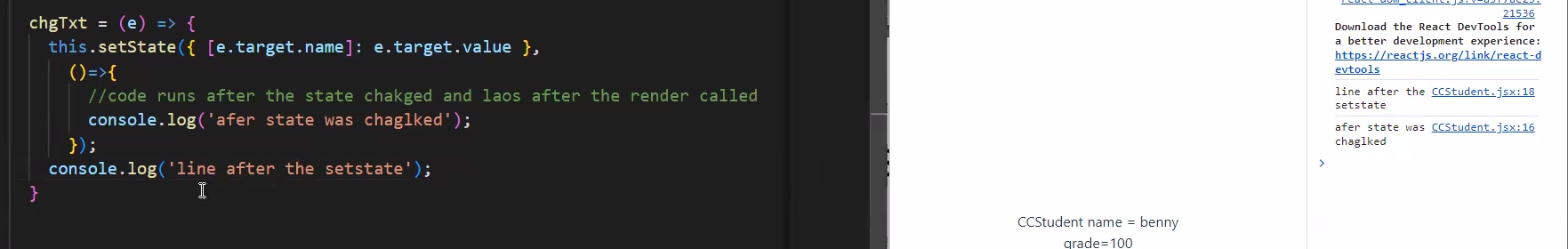
---  
  
צריך אם יוצרים סטייט, ליצור סטייט בהתחלה. This.state={} ככה ריק.

או לשאול this.state ? this.state.txt1 : “…” – אם קיים משהו בסטייט תציג לי אותו ואם לא תציג לי שלוש נקודות.

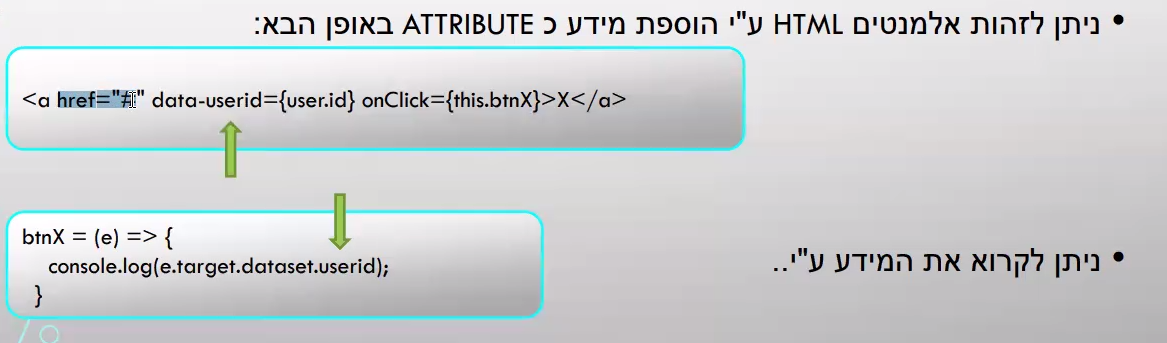


איך לעשות פעולות ברגע שסטייט כלשהו השתנה?  
אם משתמשים ברנדר ובname צריך לעשות המון איפים בחלק של הרנדר, וזה יהיה עמוס מדי.  
  
נעשה: **sync setstate**

במקום זה, כדאי לעשות בthis.setState לעשות **פסיק**, ואז היא תריץ את הקוד לאחר כל השינויים – אפשר לעשות שם פונקציה אנונימית. בצורה אסינכרונית.

  
  
מבחינת ריצה – מה שבחוץ ירוץ לפני מה שבפנים ^

Data attribute:

להכניס פרטי מידע בתוך הHTML: רושמים **data-** ואחריו שם מסוים שאני רוצה לשמור שאף אחד לא ייראה. למשל data-userid – ואני יכולה להשתמש בזה עם **e.target.dataset.**userid  


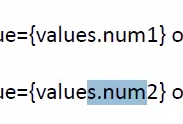
**Hooks**

I got no hooks but my home was never on the ground…

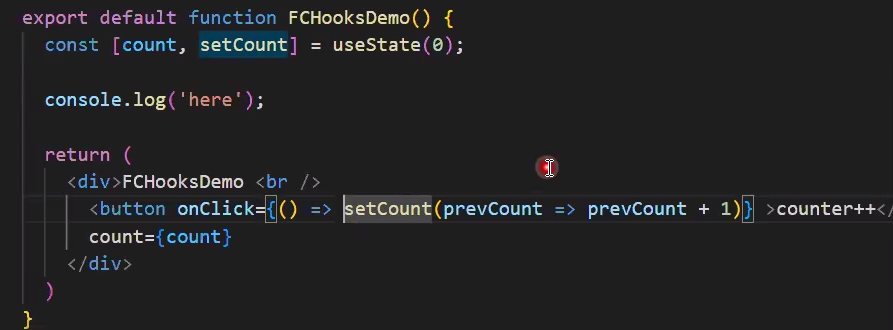
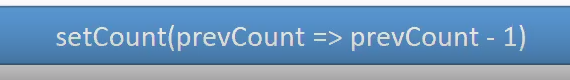
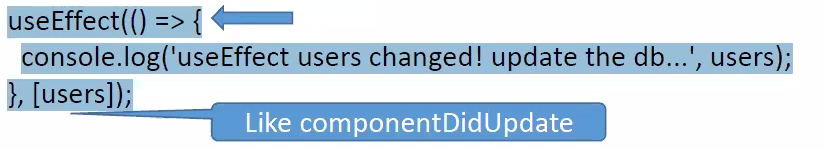
האופציה לתכנת בלי קלאס. באמצעות אלמנט או באמצעות FC.  
משתמשים ב**useState** בתוך FC, ככה אפשר סטייט בלי קלאס.  
*יש סניפט יש useStateSnippet שאפשר להשתמש בשביל קיצור כתיבה מהיר*

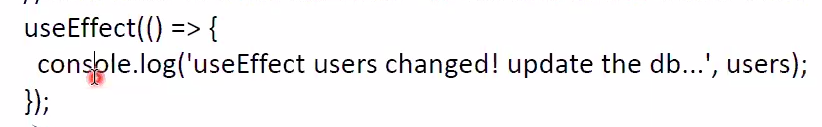
עושים אימפורט ל useState, ונותנים לה כפרמטר את הערך ההתחלתי.   
מקבלת 2 אינדנטיפריים בתוך const – אחד יהיה השם של הסטייט והערך השני יהיה הפונקציה שאיתה אני משנה את הסטייט.  
  
  
כשרוצים לשנות ערך (וזה גם קורה אסינכרונית), כותבים בפונקציה את שם הפונקציה של הסטייט, ובסוגריים () את מה שאני רוצה שהיא תהיה. ואחרי זה כל **הFC ירוץ מחדש עם הרנדר**+ערך החדש.

 ככה להציג את הערך החדש:

כשעושים עם JSON צריך להחליף את כל הסטייטים.   
  
כדי לגשת לערכים ולהציג אותם עושים: values נקודה ושם הערך:

ואם רוצים לשנות לפי הערך הקודם עם תלות עושים prevCount => prevCount +1

  
  
**useContext**  
useContext – יהיה כמו משהו ששומר סטייטים באופן גלובלי שאפשר יהיה לראות מכל המקומות, במקום לעשות uplifting נגיד  
  
  
  
**useEffect**  
useEffect – כשאני רוצה לשנות סטייט ולשים שם קודים. משתמשים עם פונקציה אנונימית בפנים, והיא תרוץ כל פעם **שיש שינוי של סטייט כלשהו,** וגם אחרי רנדר ראשוני. ואם עושים לו פסיק ואז בסוגריים מרובעים שם של סטייט – זה בעצם אומר שהקוד הזה ישתנה רק אחרי שהסטייט שכתבתי לו השתנה. ואם אני אעשה פסיק בסוגריים המרובעים – אז זה יעבוד ברגע שאחד מהם או שניהם ישתנו.   
אני לא אכתוב setState בתוך זה, כי זה יקרא לעצמו כל פעם ויתקע בלולאה אינסופית.



componentDidMount – רצה פעם אחת אחרי **הרנדר הראשון**. למשל להכניס דברים מהדאטא בייס או משהו או לעשות משהו אחרי שכבר קיים לי דברים, אחרי שכבר יש לי DOM ראשוני. זה דומה לקונסטרקטור, אבל בשונה מהבנאי זה ירוץ אחרי הרנדר הראשון ולא לפני הרנדר הראשון.  
כותבים את זה אחרי הuseEffect עם פסיק וסוגריים ריקים.   
בUseEffect היא תרוץ פעם אחת בלבד.

componentwillUnmount – לנקות DOM. למשל כשסוגרים אפליקציה או אתר, הקומפוננטה נמחקת מהזיכרון, ואם אני רוצה להריץ משהו שנייה לפני שהקומפוננטה נמחקת מהזיכרון. כותבים את זה באמצעות פסיק וסוגריים מרובעים.