שיעור 9

מתחילים...

exception

try / catch / throw

טיפול בשגיאות - ככל שנדע "לטפל" בשגיאות מראש, כך נוכל להימנע מקריסות עתידיות.

```
try{
    …. Code ….
}catch(e){
   alert(e.name);
}
```

exception

try / catch / throw

דוגמא לאופציה בשימוש במשתנה שלא קיים / חלוקה ב 0:

```
let num1 = document.getElementById(“..1”).value;
let num2 = document.getElementById(“..2”).value;
try{
    alert(num1/num2);
}catch(e){
    alert(e.name);
}
```

Closure

- נהוג לחשוב שיצירת משתנים מחוץ לסקופ של הפונקציה אינה מאפשרת לפונקציה להכיר בהם .
- אנו יכולים לשמור על המשתנים שנוצרים בתוך הפונקציה כ" closure בעזרת פרטיים" – נשמרים בתוך הפונקציה בלבד.
- לפונקציה היכולת להכיר במשתנים שנוצרו מבחוץ ואף לייצר את אותו המשתנה בתוכה , אך כאשר המשתנה נוצר בתוך הפונקציה אינו משפיע על אותו המשתנה שנוצר מחוצה לה.

Closure

```
4
5   // בדוגמה זו ניתן לראות שפונקציה היכולת להכיר במשתנה שנוצר מחוצה לה
6   let num = 10;
7
8   const func = () => {
9
10      return num + num;
11
12  }
13
14  console.log(func()); // 20
15
16  // 10 -  ויתן לראות שלמרות שהפונקציה מכירה ומשתמשת מחוצה לה, איזה משנה את ערכו
17  console.log(num);
18
```

Closure

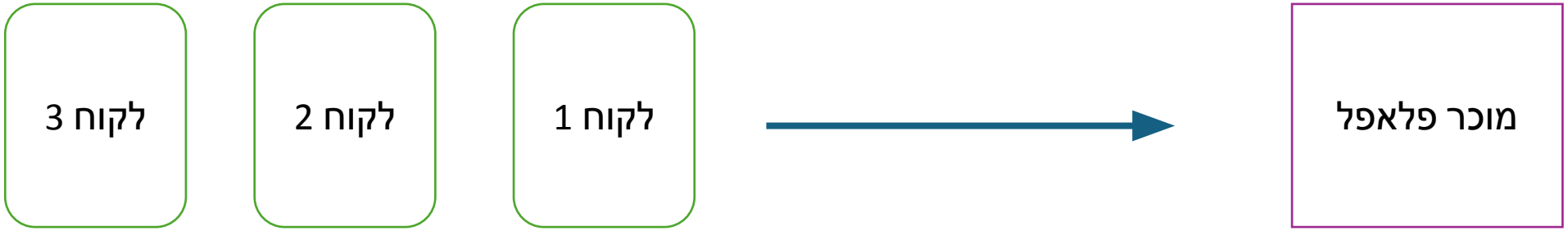
```
const myFunf = () => {
    let myName = 'aviv';
    return myName;

}
console.log(myFunf());// aviv
let myName = 'ohad';// ohad -  הערך שנשמר מחוץ לפונקציה אינו נדרס

console.log(myName);
```

סינכרוני לעומת אסינכרוני

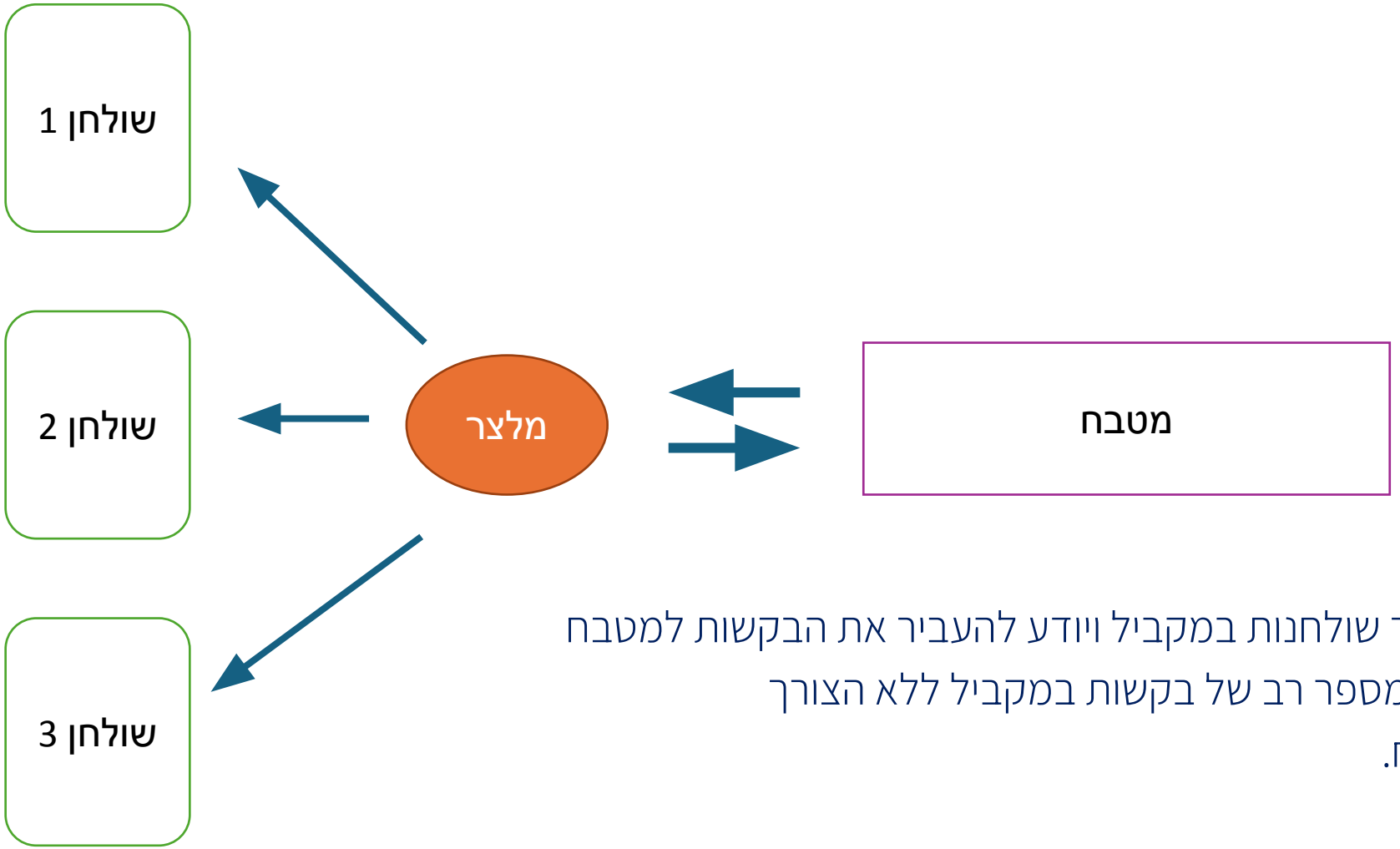
סינכרוני קיים צורך בסנכרון – עבודה בצורה סינכרונית



דוגמא: מוכר הפלאפל יכול לקבל כל פעם בקשה אחת בלבד מהלקוח, רק לאחר שיסיים עם לקוח 1, יוכל להתפנה ולטפל בבקשות של לקוח 2, וכך הלאה.

אסינכרוני

אסינכרוני - ללא הצורך בסנכרון.



למשל, מלצר במסעדה עובד על מספר שולחנות במקביל ויודע להעביר את הבקשות למטבח המטבח והמלצר יכולים להתמודד על מספר רב של בקשות במקביל ללא הצורך להיות בסנכרון (קשר) שוטף עם הלקוח.

setTimeout

- הינה פונקציה שמאפשרת לנו לעכב/לתזמן את הקוד setTimeOut .
- מטרתה העיקרית של הפונקציה לעכב בזמן (שנגדיר) את ביצוע הפעולה .
- למעשה היא מסייעת לנו לבצע פעולה מקדימה או לחילופין לקבל נתונים ממקום אחר ורק אז להפעיל את הקוד שניצור כשיחלוף הזמן שהוגדר בהמתנה, setTimeout עובד בצורה אסינכרונית, כך שהקוד לא נעצר.

- פונקציה זו מקבלת 2 ארגומנטים:

1. פונקציה אנונימית <=() .
2. זמן העיכוב במילישניות ms .

setTimeOut

בדוגמא זו הגדרנו מספר הדפסות חיצוניות לפונקציה ולפונקציה הגדרנו 2 ארגומנטים:

1. פונקציה אנונימית.
2. זמן העיכוב של 3 שניות

```
// timeout

console.log('start');
setTimeout (()=>{
    console.log('the function work after 3 seconds');
},3000);
console.log('finish');

// start
// finish
// the function work after 3 seconds
```

<u>output:</u>

ראשית הודפס start .

שנית הודפס finish שלמעשה נמצא אחרי הפונקציה.

רק לאחר 3 שניות הופעלה הפונקציה והדפיסה את קטע הקוד שבתוכה.

Promise
הבטחה לקבלת תשובה

- Promise אשר מחזיר לנו הצלחה או כישלון javascript הינו אובייקט של.
- Promise מקבל פונקציה שבתוכה 2 פרמטרים : resolve , reject .
- resolve - יפעל במידה והקוד הצליח .
- reject - יפעל במידה והקוד נכשל.

- המתנה לסיום פעולת API , נפוץ עבור קבלת נתונים מ Promise שימוש ב פונקציה על מנת שפונקציה אחר תפעל לאחריה או קבלת מידע/תמונה ממקור חיצוני.

Promise

```
1
2   //                          פרמטרים : כישלון , הצלחה
3   let pro = new Promise((resolve, reject) => {
4       let a = 3
5       if (a == 2) {
6           resolve('success')
7       } else {
8           reject('failed')
9       }
10  })
11  //במידה והקוד הקודם הצליח
12  pro.then((res) => {
13      console.log(`this is in the then :${res}`);
14  })
15      // במידה וישנה שגיאה יחדור לנו כישלון והוא יתפס
16      .catch((res) => {
17          console.log(`this is in the catch :${res}`);
18      })
19
```

Fetch

- Fetch הינו promise שמחזיר לנו הצלחה או כישלון אם הפניה לשרת/שורות הקוד נכשלו.
- ככל וחזרה שגיאה, ניתן לתפוס אותה בעזרת catch.
- Fetch נותן לנו למשוך נתונים ממקורות חיצוניים , שרתים , וכן הלאה.

```
fetch('<הזנת הכתובת לשרת>')
    .then((res) => {
        return res.json()//חילוץ המידע שהגיע מהשרת. המידע יחולץ ע"י json
    })
    .then((data) => {
        // מחזיק את הנתונים שחזרו מהשרת ולאחר שחולצו מ data
    })
    .catch((err) => {
        if (err) throw err; //אם קיימת שגיאה היא תיתפס
    })
```

סוף שיעור 9

האם יש שאלות?