Lesson 03.02 START

This page consists of several UI elements which call functions and produce output.

Get the various DOM elements:

1. Get the output box:

```
const outBox = document.getElementById('out-box');
```

2. Get the text input fields, where the user enters their name:

```
const firstName = document.getElementById('first-name');
const lastName = document.getElementById('last-name');
```

3. Get the Greet World and Greet User buttons:

```
const greetWorldBtn = document.getElementById('greet-world-btn');
const greetUserBtn = document.getElementById('greet-user-btn');
```

4. Instruct the buttons to call their respective functions:

```
greetWorldBtn.addEventListener('click', greetWorld);
greetUserBtn.addEventListener('click', greetUser);
```

5. Get the "Pick a Fruit" menu:

```
const fruitMenu = document.getElementById('fruit-menu');
```

6. Whenever the fruit menu is changed, call a function:

```
fruitMenu.addEventListener('change', pickFruit);
```

7. Get the numeric input fields:

```
const num1 = document.getElementById('num1');
const num2 = document.getElementById('num2');
```

8. Get the menu for choosing the math operator:

```
const mathMenu = document.getElementById('math-menu');
```

9. Whenever a number changes or a math operator is chosen, call the calculateAnswer function:

```
mathMenu.addEventListener('change', calculateAnswer);
num1.addEventListener('change', calculateAnswer);
num2.addEventListener('change', calculateAnswer);
```

Extra functions for running alternate code scenarios

We want to try out some cool, new moves, so we'll need alternate versions of the pickFruit and calculateAnswer functions.

10. Make calls to these alternate functions, but keep them commented out for the time being; these functions won't exist for a while, but we'll come back to them:

```
fruitMenu.addEventListener('change', getFruitFood);
mathMenu.addEventListener('change', calculate);
num1.addEventListener('change', calculate);
num2.addEventListener('change', calculate);
```

Define the functions: Whenever you mention a function in a listener, the function must actually exist or else you get an error. Comment out all listeners except for the one that calls greetWorld.

11. Define the **greetWorld()** function:

```
function greetWorld() {
  // 12. Output the greeting to the web console:
  console.log("Hello World!");
  // 13. Now output greeting to the DOM--the web page:
  outBox.textContent = "Hello World!";
}
```

14. Define the **greetUser()** function:

```
function greetUser() {

  // 15. Get the values from the name fields and save them to local
variables:
  let fName = firstName.value;
  let lName = lastName.value
  // 16. Concatenate a greeting from the inputted names:
  let greeting = `Greetings, ${fName} ${lName}!`;
  outBox.textContent = greeting;

  // 17. If no names were inputted, we get a wonky-looking "Greetings, !".
Let's refactor with if-else logic: if both names are blank, output "Hey,
you!":
  if(!fName && !lName) { // if both names are falsey empty strings
      outBox.textContent = "Hey, you!";
  } else { // if at least one name is NOT an empty string, greet by
name(s):
      outBox.textContent = `Hi, ${fName} ${lName}!`;
  }
}
```

CHALLENGE:

A. Make it a Timely Greeting, so that instead of "Hey, you!", we get "Good morning!" or other appropriate greeting for the time of day. If at least one name field is NOT blank, output the timely greeing along with the name(s):

B. Refactor the if-else as a ternary

this and event.target

The this keyword, fruitMenu and event.target all refer to the same thing: the object that called the function; in this case, that object is the fruit menu:

18. Define the **pickAFruit()** function:

```
function pickFruit(event) {

  // 19. Log the value of the fruit menu in each of the three ways:
  console.log('this.value:', this.value);
  console.log('fruitMenu.value:', fruitMenu.value);
  console.log('event.target.value:', event.target.value);
  // 20. Get rid of the cross out line, by passing event to the function
  // 21. Output the chosen fruit in UPPERCASE:
  outBox.textContent = `You picked ${fruitMenu.value.toUpperCase()}`;
}
```

"data-" attribute

Additional data may be attached to any html element by means of the "data -" attribute, where "data-myVar" means that a myVar variable is associated with this tag and will be available to JS as the dataset.myVar property.

menu.options

The select object has an options property, which stores all the options as an array. We will get deeper into arrays later on, but we already know that an array is a variable which stores multiple items.

index

The items in an array are stored by numeric position, called index, with the first item at index 0.

menu.selectedIndex property

The select object has a selectedIndex property, which is the index of the selected item. So, if the 10th menu item is selected, the selectedIndex is 9.

menu.options[menu.selectedIndex]

The selected option can be gotten from the options array by looking it up by its index.

menu.option.dataset

Once we have the selected option, we can get any data contained in the "data-" attribute via the option's dataset property. Most tags won't have a "data-" attribute, but the fruit menu options all have a "data-food" attribute, the value of which is accessible as menu.option.dataset.food:

menu.option.dataset.food

`<option value="mango" data-food="mango salsa">Mango</option>`

- **menu** : the select menu as a whole
- **option** : the selected menu option
- **dataset** : property of an element option that has a "data-" attribute
- **food** : the property of dataset via the "data-food" attribute

Let's define a new function for tapping into the "data-food" attribute:

22. Comment out the listener that calls the pickFruit function, and activate the listener that calls getFruitFood.

23. Define the **getFruitFood()** function, which is an alternate function to the "Pick a Fruit" menu.

```
function getFruitFood(event) {

  // 24. Save the value of the fruit menu to a variable:
  let fruit = fruitMenu.value;

  // 25. Save the selectedIndex to a variable. This is the index of
selected option:
  let indx = fruitMenu.selectedIndex;

  // 26. Look up the selected option in the options array and save it to
an object:
  let optn = fruitMenu.options[indx];
  // 27. Save the food property of the option's dataset property to a
variable:
  let food = optn.dataset.food;

  // 28. Output the chosen food:
  outBox.textContent = `Try the ${food}!`;
}
```

The Calculator

The calculator works as follows:

- **calculateAnswer()** does the math
- the function is called by a **change** event
- the change event occurs whenever:
  - the user enters numbers(s) and hits Enter
  - the user uses the arrows to increase-decrease numbers
  - the user chooses a math operator from the menu
- **calculateAnswer()** function does the following:
  - gets the values from the number input boxes
  - converts these "number like strings" to actual numbers
  - gets the operator from the math menu
  - does the math in accordance with the math operator We have two ways of doing the math, which is why we have two calculate functions (the second one is MUCH shorter).

switch-case-break revisited

We need to run a big chunk of logic that does the math based on that operator. We could use a long if-else if-else, but instead, let's get take the opportunity to get in a little more practice with switch-case-break:

29. Define the **calculateAnswer()** function:

```
function calculateAnswer() {

  // 30. Get the values from the number boxes and convert them to actual
numbers
  let n1 = Number(num1.value);
  let n2 = Number(num2.value);

  // 31. Get the math operator, which is the value of the math menu:
  let mathOperation = mathMenu.value;

  // 32. Declare a variable to hold the answer:
  let ans = 0;

  // 33. Start with switch, which takes what is to be compared to the
case:
  switch(mathOperation) {

      // 34. Write the case to compare to mathOperation:
      case 'add':
          // 35. If the switch-case comparision is true, run the case
code:
          ans = n1 + n2;
      // 36. After the case code, break before going on to the next case:
      break;

      // 37. Repeat the above for all the other operations:
      case 'subtract':
          ans = n1 - n2;
      break;

      // 38. Since each case executes only one line, roll it up onto the
same line:
      case 'multiply': ans = n1 * n2;
      break;
      case 'divide': ans = n1 / n2;
      break;
      case 'modulo': ans = n1 % n2;
      break;
      case 'exponent': ans = n1 ** n2;
      break;
  }

  outBox.textContent = ans;

}
```

eval()

The eval() method takes a a string as its argument and considers it as a math operation. If it can detect numbers with a math operator in between, it will perform the calculation.

- We will get the operator directly the text of the selected math menu option
- Then, we will concatenate a string from the numbers and the operator
- Then, we will pass the string to eval(), which will perform the calculation.

39. Start by defining a new **calculate()** function, and switch over to calling this function, instead of calculateAnswer:

```
function calculate() {

    // 40. Get the number:
    let n1 = Number(num1.value);
    let n2 = Number(num2.value);

    // 41. Get the index of selected option and save it to a variable:
    let i = mathMenu.selectedIndex;
    console.log('i:', i);

    // 42. Get the text property of the selected option, which we look up
by index in the math menu's options array, and save it to a variable:
    let op = mathMenu.options[i].text;
    console.log('op:', op);

    // 43. Do the math as a concatenated string using the op variable:
    let ans = `${n1} ${op} ${n2}`;

    // 44. Output the answer; all we get is the literal string, but at
least it looks like a math problem:
    outBox.textContent = ans;

    // 45. Try again this time wrapping the string in eval():
    ans = eval(`${n1} ${op} ${n2}`);

    // 46. Output the answer again. This time it should work as expected:
    outBox.textContent = ans;
}
```

*** CHALLENGE SOLUTION START: *** ### ***

A. Make it a Timely Greeting, so that instead of "Hey, you!", we get "Good morning!" or other appropriate greeting for the time of day. If at least one name field is NOT blank, output the timely greeing along with the name(s):

```
let dt = new Date();
let hr = dt.getHours();
// hr = 19; // testing hr
let g = "";
if(hr < 12) {
    g = `Good Morning`;
} else if(hr < 18) {
    g = `Good Afternoon`;
} else {
    g = `Good Evening`;
}
```

B. Refactor the if-else as a ternary

```
outBox.textContent = !fName && !lName ? `${g}!` : `${g}, ${fName}
${lName}!`;

// *** ### *** CHALLENGE SOLUTION END *** ### ***
```