1. Exercism

1.1. Bob

My Solution:

```ts
export function hey(message: string): string {
    // Remove leading or trailing whitespace/s.
    message = message.trim();

    // Check if it is a 'silence.'
    if (message.length === 0) {
      return "Fine. Be that way!";
    }

    // RegEx for any letters (alphabetical characters), while
    // excluding any numbers (digits) and underscores.
    let regex: RegExp = /[^\W\d_]+/g;


    // Check if it is a 'yell.'
    let isEndQuestionMark = message.endsWith('?');
    if (regex.test(message) && message === message.toUpperCase()) {
        return isEndQuestionMark ? "Calm down, I know what I'm doing!" : "Whoa, chill out!";
    }

    // Check if it is a 'question.'
    if (isEndQuestionMark) {
        return "Sure.";
    }

    // Otherwise, return 'whatever.'
    return "Whatever.";
  }
```

Community Solution:

```ts
const answers: string[] = ["Whatever.", "Sure.", "Whoa, chill out!", "Calm down, I know what I'm doing!"];

export const hey = (input: string): string => {
  const speech = input.trim()
  if (speech == "") return "Fine. Be that way!"

  const isQuestion = speech.endsWith("?") ? 1 : 0
  const isShout = /[A-Z]/.test(speech) && !/[a-z]/.test(speech) ? 2 : 0

  return answers[isQuestion + isShout]
}
```

This is somewhat interesting since the usage of too many if statements was mitigated by utilizing an array of string answers and their index position (calculated through the witty implementation of the ternary operators - conditionals) Moreover, it is true that the lines of code are lower compared to my solution, but I think this solution is not that as readable when compared to mine.

1.2. Isogram

My Solution:

```ts
export function isIsogram(input: string): boolean {
    input = input.trim().toLowerCase();


    const knownChars: string[] = [];
    for (const char of input) {
      console.log(char);

      if (knownChars.includes(char)) {
        return false;
      }

      if (!(char ===  ' ' || char === '-')) {
        knownChars.push(char);
      }
    }

    return true;
  }
```

Community Solution:

```ts
export function isIsogram(word: string): boolean {
  word = word.replace(/\s+|\-+/g, '')
  word = word.toLowerCase()
  return Array.from(new Set(word.split(''))).length === word.length
}
```

This is interesting since instead of using an if statement, he used ReGex to remove the spaces and hyphens of the input string. The author of this code also cleverly used split and the property of set (unique elements) to determine if the input string is an isogram or not.

1.3. Flatten Array

My Solution:

```ts
type MyArray = number | undefined | MyArray[];

export function flatten(input: MyArray[]): number[] {
    if (input.length === 0) return [];

    const temp: MyArray = input.shift();
    if (Array.isArray(temp)) {
        return [...flatten([...temp, ...input])];
    } else if (temp === undefined) {
        return [...flatten(input)];
    } else {
        return [temp, ...flatten(input)]
    };
}
```

Community Solution:

```ts
type ArrayOfNumbers = number | undefined | ArrayOfNumbers[];

export function flatten(nums: ArrayOfNumbers[]) {
  return nums.flat(Infinity).filter((v) => typeof v === "number");
}

```

It was clever of him to use filter and typeof to only include numbers on the final output array. Also, he used the built-in flat method which is I think somewhat cheating lol. That is why his solution is so short.

2. Array Object

2.1. Negative indexing

From MDN:

Array.prototype.at()

The `at()` method of `Array` instances takes an integer value and returns the item at that index, allowing for positive and negative integers. Negative integers count back from the last item in the array.

Sample code from MDN:

JavaScript Demo: Array.at()

```
1  const array1 = [5, 12, 8, 130, 44];
2
3  let index = 2;
4
5  console.log(`Using an index of ${index} the item returned is ${array1.at(index)
6  // Expected output: "Using an index of 2 the item returned is 8"
7
8  index = -2;
9
10 console.log(`Using an index of ${index} item returned is ${array1.at(index)}`);
11 // Expected output: "Using an index of -2 item returned is 130"
12
```

2.2. Sort and toSorted

To summarize, the key semantic difference between [.sort()] and [.toSorted()] is that [.sort()] sorts the original array in place and returns the reference to the same array, now sorted. While [.toSorted()] creates a new sorted array, leaving the original array unchanged. Both methods sort in ascending order by default.

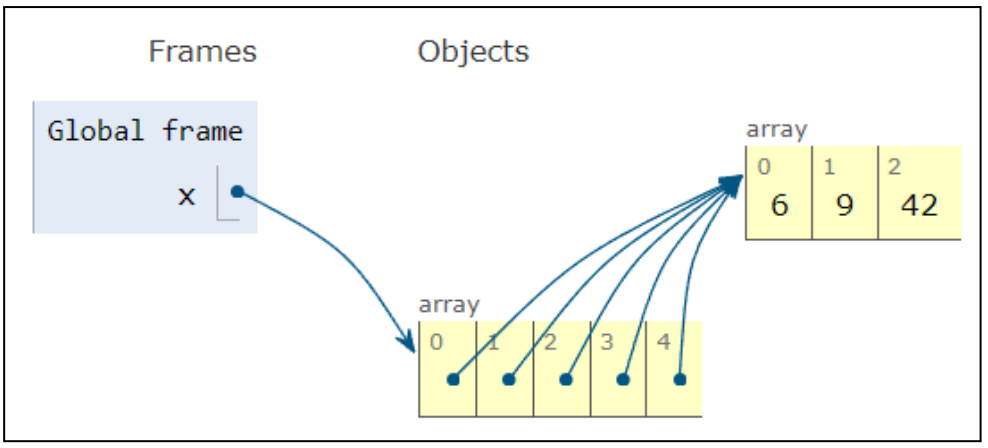
2.3. Frame

The code in consideration. I mutated some of the elements of [x] in order to analyze its behavior:

JavaScript (ES6)
known limitations

```
1  let x = Array(5).fill([]);
2
3  x[0].push(6);
4  x[1].push(9);
5  x[4].push(42);
```

The resulting frame:



As it turns out, all of the five elements of [x] points/refer to the same array object. That is, for instance, the modification made on the array pointed by x[4] will also take effect on the array pointed by x[1].

2.4. For and/or for-of

From:

```
console.log([...Array(3)].map((_, r) => [...Array(5)].map((_, c) => r * 5 + c)));
```

```
[ [ 0, 1, 2, 3, 4 ], [ 5, 6, 7, 8, 9 ], [ 10, 11, 12, 13, 14 ] ]
```

To:

```
console.log(
    (() => {
        let output: number[][] = [];
        for(let r = 0; r < 3; r++) {
            let temp: number[] = [];
            for (let c = 0; c < 5; c++)
                temp.push(r * 5 + c);
            }

            output.push(temp);
        }

        return output;
    })()
);
```

```
[ [ 0, 1, 2, 3, 4 ], [ 5, 6, 7, 8, 9 ], [ 10, 11, 12, 13, 14 ] ]
```

2.5. Python to JS/TS

From:

```python
# Code is in Python
names = ['a', 'b', 'c']

for idx, name in enumerate(names):
    print(f'Index {idx}: {name}')
```

```
Index 0: a
Index 1: b
Index 2: c
```

To:

```typescript
let names: Array<string> = ['a', 'b', 'c'];
let entries: IterableIterator<[number, string]> = names.entries();

for (const entry of entries) {
    console.log(`Index ${entry[0]}: ${entry[1]}`);
}
```

```
Index 0: a
Index 1: b
Index 2: c
```

2.6. For-of to join

From:

```javascript
const strs = ['a', 'b', 'c'];
let answer = strs[0];

for (const str of strs.slice(1)) {
  answer += `, ${str}`;
}
```

```
a, b, c
```

To:

```typescript
const strs: string[] = ['a', 'b', 'c'];
let answer: string = strs[0];

answer = strs.join(', ');
console.log(answer);
```

```
a, b, c
```

2.7. For-of to forEach

From:

```
const names = ['a', 'b', 'c'];

for (const name of names) {
  console.log(name);
}
```

```
a
b
c
```

To:

```
const names: Array<string> = ['a', 'b', 'c'];

names.forEach((name: string) => console.log(name));
```

```
a
b
c
```

2.8. For-of to map

From:

```
const chars = [...Array(3)].map(_ => String.fromCharCode(Math.floor(Math.random() * 26) + 'a'.charCodeAt(0)));
const answer: Array<string> = [];

for (const ch of chars) {
  answer.push(ch.repeat(3));
}

console.log(answer);
```

```
[ 'rrr', 'iii', 'sss' ]
```

To:

```
const chars: string[] = [...Array(3)].map(_ => String.fromCharCode(Math.floor(Math.random() * 26) + 'a'.charCodeAt(0)));
const answer: string[] = [];

chars.map((char: String) => answer.push(char.repeat(3)));

console.log(answer);
```

```
[ 'qqq', 'sss', 'ddd' ]
```

// The ‘form’ is the same.

2.9. For-of to filter

From:

```
const strs = ['123', 'abc', '!@#', '123abc', '123!@#', 'abc!@#', '123abc!@#'];
const answer: string[] = [];

for (const str of strs) {
  if (str.toUpperCase() === str.toLowerCase()) {
    answer.push(str);
  }
}

console.log(answer);
```

```
[ '123', '!@#', '123!@#' ]
```

To:

```
const strs: string[] = ['123', 'abc', '!@#', '123abc', '123!@#', 'abc!@#', '123abc!@#'];
const answer: string[] = [];

strs.filter((str: string) => {
    if (str.toUpperCase() === str.toLowerCase()) {
        answer.push(str);
    }
});

console.log(answer);
```

```
[ '123', '!@#', '123!@#' ]
```

2.10. For-of to findIndex

From:

```
const nums = Array(5).fill().map(_ => Math.floor(Math.random() * 100));
let isAllEven = true;

for (const num of nums) {
  if (num % 2 !== 0) {
    isAllEven = false;
  }
}

console.log(nums);
console.log(isAllEven);
```

```
[ 6, 12, 49, 45, 27 ]
false
```

To:

```
const nums = Array(5).fill().map(_ => Math.floor(Math.random() * 100));
let isAllEven = true;

const isNotEven = (num) => num % 2 !== 0;
isAllEven = (nums.findIndex(isNotEven) === -1) ? true : false;

console.log(nums);
console.log(isAllEven);
```

```
[ 49, 91, 12, 15, 7 ]
false
```

2.11. For-of to every

From:

```
const nums = Array(5).fill().map(_ => Math.floor(Math.random() * 100));
let isAllEven = true;

for (const num of nums) {
  if (num % 2 !== 0) {
    isAllEven = false;
  }
}

console.log(nums);
console.log(isAllEven);
```

```
[ 90, 7, 2, 60, 57 ]
false
```

To:

```
const nums = Array(5).fill().map(_ => Math.floor(Math.random() * 100));
let isAllEven = true;

isAllEven = (nums.every(num => num % 2 === 0)) ? true : false;

console.log(nums);
console.log(isAllEven);
```

```
[ 89, 1, 66, 35, 15 ]
false
```

2.12. For-of to some

From:

```
const nums = Array(5).fill().map(_ => Math.floor(Math.random() * 100));
let isAllEven = true;

for (const num of nums) {
  if (num % 2 !== 0) {
    isAllEven = false;
  }
}

console.log(nums);
console.log(isAllEven);
```

```
[ 95, 15, 10, 48, 92 ]
false
```

To:

```
const nums = Array(5).fill().map(_ => Math.floor(Math.random() * 100));
let isAllEven = true;

isAllEven = nums.some(num => num % 2 !== 0) ? false : true;

console.log(nums);
console.log(isAllEven);
```

```
[ 21, 61, 48, 16, 79 ]
false
```