[1]

An array of integer numbers is given:

```
const a = [ 11, 41, 2, 18, 99, 559, 1, 8, 53, 16, 11 ];
```

Write code that sorts this array into a descending order.

[2]

An array of country descriptors is given:

```
const countries = [
  { name: 'Serbia', flag_colors: [ 'red', 'blue', 'white' ] },
  { name: 'Germany', flag_colors: [ 'red', 'yellow', 'black' ] },
  { name: 'Austria', flag_colors: [ 'red', 'white' ] },
  { name: 'Italy', flag_colors: [ 'green', 'white', 'red' ] },
  { name: 'Hungary', flag_colors: [ 'red', 'white', 'green' ] }
];
```

Write code that tests if any of country flags contains *yellow* color.
Write code that finds a list of unique colors across all the country flags.

[3]
Write a function that can calculate the sum of an arbitrary number of arguments.

[4]
The following code is given:

```
const f = (n) => new Promise((resolve) => setTimeout(() => resolve(n), 1000));
const a = [ { x: 5, y: 3 }, { x: 2, y: 6 }, { x: 1, y: 8 } ];
```

Write down code that calculates the following sum where $N = 2$:

```
f(a[0].x) * f(a[0].y) + f(a[1].x) * f(a[2].y) + ... + f(a[N].x) * f(a[N].y)
```

[5]

Write code that mimics a simple event publisher/subscriber use case:

- an event is represented as a simple JSON object
- event content/structure is defined by a *schema* – we define N different schemas (or in other words *event types*) where each schema has its own unique *name* (or, in other words, each event type has its own unique name)
- at a minimum, each event has a property called *name* – this property contains the name of a schema that defines content of the event
- we have a number of event publishers, one event publisher per event type
- each event publisher exports 2 methods: *add_listener* method is used to subscribe an event handler; *remove_listener* method is used to remove previously subscribed event handler
- event handler is a function that gets *event* parameter on its input – this parameter represents an event object that is created and published by event publisher
- both *add_listener* as well as *remove_listener* methods take event handler function on their input

Suppose that at least 2 event types are defined – we don't care for event properties other than *name*:

- "event_a" - schema for event type "event_a" doesn't define any property other than *name* which is mandatory
- "event_b" - schema for event type "event_b" doesn't define any property other than *name* which is mandatory

Code must create/initialize 2 event publishers, one that will publish events of type "event_a" and the other that will publish events of type "event_b". On every N seconds, an event publisher should create an event (with a proper value for *name* property) and pass that event to all its registered event handlers - N is an integer number provided at the time when the event publisher is created. An example event that is generated and published by "event_a" event publisher could look like:

```
{
  name: 'event_a'
}
```

Each event handler should have a unique name that is set at the time the event handler is created. Each event handler handles an incoming event in a way that it logs the following line to console:

```
`event handler ${name} handles event ${event.name}`
```

where *name* variable represents the name of the given event handler.

Code must create/initialize 3 event handlers:

- "event_a" event publisher must have 2 registered handlers:
  - one is represented by an anonymous function (handler A1)
  - the other one is used to handle "event_b" events too (shared handler S1)

- "event_b" event publisher must have 2 registered handlers:
  - one is used to handle "event_a" events too (shared handler S1)
  - the other is used to handle "event_b" events only (handler B2)

Upon 60 seconds from the beginning of the program handler B2 must be removed as a listener of "event_b" event publisher.

NOTE: use no javascript classes, only functions and objects.
NOTE: code that is used to create/initialize event publisher should be generic enough so that it could be used to create any number of event publishers
NOTE: code that is used to create/initialize event handlers should be generic enough so that it could be used to create any number of event handlers