Übungsblatt 4

16 Komplexe Zahlen in JavaScript*

Sie sollen eine Klasse zur Rechnung mit komplexen Zahlen erstellen. Nennen Sie diese Klasse `cmplx`. Komplexe Zahlen bestehen aus dem sog. Realteil und dem sog. Imaginärteil. Die Trennung erfolgt durch die Konstante i, welche durch $i^2 \equiv -1$ definiert ist. Wenn a, b, c und d reelle Zahlen sind (d.h. Datentyp *Number* in JavaScript), so ergibt sich für Multiplikation und Addition, dass

$$(a + bi) + (c + di) = (a + c) + (b + d)i, \quad (1)$$
$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i. \quad (2)$$

Ihre Klasse sollte daher folgende Eigenschaften besitzen:

- Realteil (genannt `re`)
- Imaginärteil (genannt `im`)

Fügen Sie folgende Methoden hinzu:

- `add()` zum Addieren von Zahlen mit der aktuellen Instanz, wobei über die Variable `arguments.length` die Anzahl der übergebenen Argumente überprüft werden soll. Bei zwei Argumenten sollen die Zahlen als Real- und Imaginärteil interpretiert werden. Bei einem Argument soll das Argument als komplexe Zahl interpretiert werden. Führen Sie die Addition wie oben gezeigt aus.
- `multiply()` zum Multiplizieren von Zahlen mit der aktuellen Instanz. Gehen Sie hierbei analog zur Implementierung von `add()` vor, wobei Sie anstell der Addition die Multiplikation wie oben gezeigt implementieren.

Der Konstruktor der Klasse soll bis zu zwei Elemente (Real- und Imaginärteil) erlauben. Denken Sie daran, dass in JavaScript Argumente optional sind und geben Sie den Argumenten daher als Standardwert 0.
Testen Sie Ihre Klasse über einige Rechnungen in der JavaScript Konsole Ihres Webbrowsers.

17 Multimedia mit `<audio>`*

Erstellen Sie eine Webseite die ein `<audio>`-Element, sowie vier Buttons (Stopp, Abspielen / Pause, Langsamer, Schneller) enthält. Für das Element sollen folgende Angaben gelten:

- Erstellen Sie das Element ohne Controls.

- Geben Sie die Quelle nicht über das `src`-Attribut an, sondern über `<source>`-Elemente.
- Folgende Quellen sollen verwendet werden:

  MP3-Format:`https://raw.githubusercontent.com/WebAppLecture/Exercises.Day4/master/Resources/sample.mp3`
  OGG Vorbis-Codec: `https://raw.githubusercontent.com/WebAppLecture/Exercises.Day4/master/Resources/sample.ogg`
- Das Element soll über die Buttons durch JavaScript gesteuert werden. Um den letzten Punkt zu verwirklichen müssen Sie das `onclick`-Ereignis der Buttons setzen. Um die entsprechenden Funktionalitäten zu implementieren werden Sie folgende Methoden des Audioelements benötigen:
- Die Eigenschaften paused und ended gibt Ihnen Auskunft über den aktuellen Zustand.
- Mit den Methoden `play()` und `pause()` steuern Sie den aktuellen Zustand. Die aktuelle Zeit setzen Sie über die Eigenschaft `currentTime`[1].
- Mit der Eigenschaft playbackRate können Sie die Geschwindigkeit einstellen (1 entspricht hier 100%, d.h. normale Geschwindigkeit).

18 Noch mehr Multimedia*

Für diese Aufgabe können Sie zum Beispiel den Trailer zu Sintel[2] verwenden. Sie finden das Video unter
`https://download.blender.org/durian/trailer/sintel_trailer-480p.mp4`
(MP4-Format) bzw.
`https://download.blender.org/durian/trailer/sintel_trailer-480p.ogv`
(OGG-Vorbis). Verwenden Sie auch in dieser Aufgabe beide Quellen!
Ändern Sie nun Ihre Webseite aus Aufgabe 17 so, dass anstelle eine Soundausgabe eine Videoausgabe platziert wird. Sie müssen im Prinzip nur den `<audio>`-Tag durch den `<video>`-Tag ersetzen. Fügen Sie nun allerdings die Standardkontrollelemente über das Attribut controls hinzu. Des Weiteren sollen Sie ein (beliebiges) Vorschaubild über das `poster`-Attribut angeben.
Entfernen Sie die Buttons für Stopp und Abspielen / Pause. Diese sind nun über die Standardkontrollelemente eingebaut. Die Buttons für die Abspielgeschwindigkeit dagegen sollen weiterhin verwendbar sein. Fügen Sie zum Abschluss noch Behandlung für einige Ereignisse hinzu: Beim Auslösen des `ontimeupdate` Ereignisses sollen Sie die aktuelle Videozeit in einem `<div>` auf der Seite anzeigen. Beim Auslösen des `onended` Ereignisses soll eine Meldung, z.B.

[1] In einigen Browser ist das Setzen von `currentTime` und `playbackRate` noch nicht implementiert.
[2] © copyright Blender Foundation | `www.sintel.org`

```
alert('Video abgespielt.');
```

ausgegeben werden.

19 Zeichnen dank <canvas>*

Es soll nun eine WebApplikation zum Zeichnen von primitiven Grafiken entwickelt werden. Sie benötigen hierzu ein <canvas>-Element, sowie mehrere HTML-Forms Elemente.
Über ein <select>-Element soll der Benutzer entsprechende Aktionen (z.B. Rechteck (rect()), Linie (offener Pfad zu einem Punkt), Dreieck (geschlossener Pfad zu zwei Punkten), Kreis (arc()), ...) mit Koordinaten und Längenangaben (hier wären numerische Eingabeboxen sehr passend) auswählen.
Außerdem werden zwei Color-Boxen (Füllfarbe und Rahmenfarbe), sowie ein Range-Slider für die Rahmendicke benötigt. Beim Klick auf einen Button soll die entsprechende Zeichnung eingefügt werden.
Fügen Sie außerdem noch zwei zusätzliche Buttons oder Hyperlinks mit folgenden Funktionalitäten hinzu:

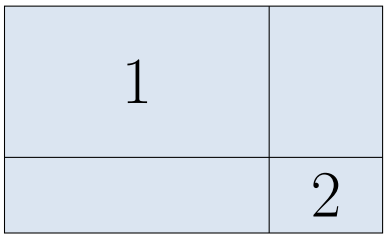
- Das Canvas zurücksetzen (als Füllfarbe soll die zurzeit ausgewählte Füllfarbe verwendet werden).
- Das derzeitige Bild als PNG speichern.

Tipp zum Speichern des Bildes als PNG.
Öffnen Sie ein neues Fenster mit window.open(...) und verwenden die mit [CanvasObject].toDataURL('image/png') generierte Adresse. Vorsicht: Verwenden Sie keine DOM Manipulationen wie z.B. createElement()!

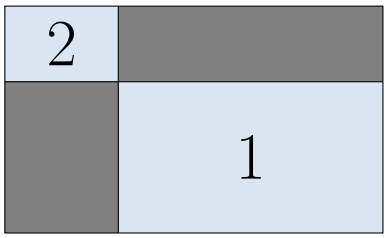
20 Video und Canvas – Gemeinsam mehr erreichen

Canvas an sich wäre schon fast langweilig wenn da nicht noch das neue Videoelement hinzukäme. In dieser Aufgabe verbinden Sie die beiden Elemente um einen sehr einfachen dynamischen Effekt zu erzeugen. Was Sie dazu brauchen ist eine Webseite mit einem <canvas>-Element, einem nicht angezeigten , welches ein <video> sowie ein weiteres <canvas> enthält und ein wenig JavaScript. Verwenden Sie als Videoquellen wieder das bereits bekannte Synergy Video aus Aufgabe 18.
Folgender Aufbau ist hilfreich:
Setzen Sie die Attribute des <video>-Elements autoplay=true und loop=true. Sobald das Video geladen hat sollten Sie dann in entsprechenden Abständen (z.B. in 40ms Pulsen) in das (versteckte, d.h. im -Container enhaltene) Canvas zeichnen. Übergeben Sie hierzu einfach das Videoelement der drawImage() Methode des Canvas Contexts:

```
mycontext.drawImage(bild, sx, sy, sw, sh, dx, dy, dw, dh);
```



Original (versteckter Canvas)



Modifiziert (sichtbarer Canvas)

Abschließend zeichnen Sie auf das (sichtbare) Canvas. Hierzu sollen Sie folgendermaßen vorgehen:
Dies bedeutet, dass Sie zwei Ausschnitte aus dem Videobild versetzt zeichnen und den Rest mit einer entsprechenden Füllung versehen. Diese muss nicht einheitlich sein, sondern kann von Ihnen frei entschieden werden.