// BEISPIEL UND AUFGABE:

// Dieses Skript soll als Beispiel dazu dienen, wie Interfaces und Arrays genutzt werden können.

// Hier wird ein ungefährer Aufbau eines simplen Klick-Spiels gezeigt. Der Nutzer kann dabei durch Button ein neues Monster erstellen.

// Zu beginn werden hier zuerst Interfaces, danach Variablen deklariert.

// Weiter unten kommen dann die Funktionen.

// EINGEBAUTE FEHLER: Innerhalb jedes Programmteiles wurden ein paar fiese Fehler eingebaut!

// Diese werden vermutlich erst in der Browser-Konsole angezeigt.

// Testet also alle Funktionen, jeden Button welchen ihr finden könnt!

// Hilfe: Benutzt auf Verdacht ein Konsolen-Log oder ruft die Variable in der Konsole des Browsers auf.

// Hilfe2: Betrachtet den umliegenden Code. Trackt die Werte von Variablen, falls euch etwas komisch vorkommt!

// ------- interfaces --------- //

// INSGESAMT EINGEBAUTE FEHLER beu den interfaces: Keine. (0 / null)

// Monster sind vielfältig und können sehr unterschiedlich sein. Dennoch werden sie durch allgemeine Attribute, wie Name und Lebenspunkte, vereint.

// Deshalb wird hier ein interface genutzt!

// Ein interface erlaubt das erstellen von einem ungefährem Haupt-Objekt.

// Object = Komplexer Datentyp auf Grundlage primitiver Datentypen

interface Monster {

monsterName : string; // Name des Monsters

monsterHealthPoints : number; // Lebenspunkte

monsterExperience : number; // Erfahrungspunkte bei besiegen des Monsters

monsterModifier : string []; // Monster-Verstärker. Diese sind in diesem Fall nur Text! (Da hier einfacher Zufall für die Auswahl genutzt wird, kann der gleiche Eintrag auch doppelt vorkommen)

}

// ------- Variablen -------- //

// INSGESAMT EINGEBAUTE FEHLER bei den Variablen: I (1 / einer)

let monsterHolder : string = "monsterHoldingCell"; // ID für das Haupt-Element, in welchem die Monster sich befinden werden. Wird vielleicht mehrfach in dem Skript gebraucht, deshalb einmalig definitiert.

let playerName : string = "Spielername"; // Ein paar globale Variablen, welche den Spieler darstellen.

let playerXP : number = 0; // Stellt die gesammelte Erfahrung des Spielers dar.

let playerXPperLevel : number = 500; // Da es nur einen Spieler gibt, ergibt sich noch nicht viel Sinn darin, für den Spieler ein interface (im Sinne der Programmierung) zu erstellen.

// Mehrere Arrays, welche jeweils Bauteile für Namen oder Eigenschaften der Monster beinhalten.

let prefix : string[] = ["Wald-", "Seuchen-", "Uralte(s) ", "Gift-", "Brennende(s) ", "Kniescheibenzertrümmernde(s) ", "Müde(s)", "Hungrige(s)", "Fett(s)", "Kugelrundes", "Wasser"]; // length = 6, da 6 Einträge. Von 0-5.

let monsterName : string[] = ["Ratte", "Nagetier", "Ungeziefer", "Spinne", "Mondgesicht", "Gummibär", "Osterhase", "Ungeheuer"]; // length = 3, da 3 Einträge. Von 0-2.

let suffix : string[] = [" des Verderbens", " aus der Hölle", " der Lethalität", " mit Rheuma", " der Redundanz", " der Zerberstung", "aus dem Meer", "aus der Tiefe", "aus Fuwa", "mit Spliss", "mit gebrochener Nase"]; // length = 6, da hier 6 Einträge sind. Von 0-5.

let monsterModifers : string[] = ["Ist nervig", "Linkshänder", "Bier-Connoisseur", "Verfehlt häufig", "Prokrastiniert", "Müde", "Verwirrt", "Wasserscheu", "Bipolar", "Hat Schnupfen", "Verläuft sich oft"]; // Eine Reihe von zufälligen "Verstärkern" für das Monster.

let monsterWeapon = ["Spritzpistole", "Wasserbombe", "Lasso", "Melone", "Axt"]

// -- Initialisierung für viele/variable Anzahl an Monster --

let monsterArray : Monster[] = []; // Das Haupt-Array wurde erstellt und initialisiert!

console.log(monsterArray ); // Gebe das Monster-Array einmal zu beginn aus. Es sollte leer sein.

let Monsterbilder = ["osterhase.jpg", "Gummibär.jpg", "mondgesicht.jpg"];

// ----------- Funktionen ----------- //

// INSGESAMT EINGEBAUTE FEHLER bei den Funktionen: IIIII (5 / fünf)

// Generelle onload-funktion um Event-Listener zum Dokument hinzuzufügen

window.onload = function () {

document.getElementById("monsterSpawner").addEventListener("click", generateMonster, false);

updatePlayerLevel(); // Zu Anfang wird durch eine Funktion ein HTML-Element mit Inhalt befüllt.

}

console.log(document.getElementById("monsterSpawner").innerHTML);

// Die Hauptfunktion, um ein Monster zu erstellen. Wird von einem Button ausgerufen.

// Generiert ein neues Monster. Dieses wird zu dem Monster-Array hinzugefügt.

// Ruft eine Funktion auf, welche dann das entsprechende HTML erzeugt.

function generateMonster()

{

let newMonsterName : string = generateMonsterName(); // Eigens-gebaute Funktion, welche einen string zurück gibt.

let newMonsterHP : number = generateMonsterHitPoints(); // Eigens-gebaute Funktion, welche eine Zahl zurück gibt.

let newMonsterXP : number = generateMonsterXP(); // Eigens-gebaute Funktion, welche eine Zahl zurück gibt.

let newMonsterModifier : string[] = generateMonsterModifer();

let newMonsterWeapon : string [] = generateMonsterWeapon (); // Eigens-gebaute Funktion, welche ein string-Array zurück gibt.

let newMonster : Monster = { // Monster wird erstellt.

monsterName : newMonsterName,

monsterHealthPoints : newMonsterHP,

monsterExperience : newMonsterXP,

monsterModifier : newMonsterModifier,

monsterWeapon : newMonsterWeapon,

};

monsterArray.push(newMonster); // Monster wird erst in diesem Schritt zu dem Array hinzugefügt

console.log(monsterArray[monsterArray.length -1]); // Man kann nur auf Array-Teile zugreifen, welche definiert sind. -1 ist nicht definitiert (und wird es auch nie sein).

monsterGenerateHTML(); // Triggere die Generierung von HTML

}

// Generiert HTML-Elemente, welche dann einem Element untergeordnet werden. Erzeugt ebenfalls einen Event-Listener auf dem Button.

function monsterGenerateHTML()

{

let holdingDiv : HTMLElement = document.createElement("div"); // Erstelle ein neues HTML-Element vom typ <div>. Es ist jedoch noch nicht zu sehen!

holdingDiv.setAttribute("id", "monster" + monsterArray.length); // Die ID jedes neu-erstellten Monsters entspricht der aktuellen Array-Länge.

holdingDiv.setAttribute("class", "monster"); // Klasse für Visuals.

document.getElementById(monsterHolder).appendChild(holdingDiv); // Das HTML-Element muss erst noch zu einem Objekt hinzugefügt werden, in diesem Fall mit der id "monsterHoldingCell"

let monsterName : HTMLElement = document.createElement("p"); // Generiere einen <p>

monsterName.innerHTML = monsterArray[monsterArray.length - 1].monsterName; // Inhalt des <p>: Monster-Name des letzten Monsters im Array.

holdingDiv.appendChild(monsterName); // Füge das <p> zum HTML-Dokument hinzu, indem es dem holding-Div angefügt wird.

let monsterMod : HTMLElement = document.createElement("p"); // Generiere einen <p>

monsterMod.innerHTML = monsterArray[monsterArray.length - 1].monsterModifier[0] + ", " + monsterArray[monsterArray.length -1].monsterModifier[1]; // Inhalt des <p>: Monster-Modifizierer null und eins

holdingDiv.appendChild(monsterMod); // Füge das <p> zum HTML-Dokument hinzu, indem es dem holding-Div angefügt wird.

let monsterWeapon = document.createElement ("p");

monsterMod.innerHTML = monsterArray[monsterArray.length - 1].monsterModifier[0] + ", " + monsterArray[monsterArray.length - 1].monsterModifier[1];

holdingDiv.appendChild(monsterMod);

let monsterImg : HTMLElement = document.createElement("img"); // Erstelle ein <img>-Element

monsterImg.setAttribute("src", "imgs/pinguin.png"); // Der Pfad für das Bild muss über setAttribute festgelegt werden. Der Bildpfad kann natürlich auch anders aussehen.

monsterImg.setAttribute("alt", "Schreckliches Monster"); // Das alt für das Bild wird hier festgelegt.

holdingDiv.appendChild(monsterImg); // Füge das Bild zu dem holding-div hinzu (<div>, welche ein paar Zeilen zuvor erstellt worden ist)

let monsterBtn : HTMLElement = document.createElement("BUTTON"); // Erstelle ein <button>-Element

monsterBtn.innerHTML = "Monster bekämpfen!"; // Verändere den Inhalt des HTML-Elementes. Der genaue Text ist dabei euch überlassen.

holdingDiv.appendChild(monsterBtn); // Füge den Button zu dem holding-div hinzu.

let monsterCount : number = monsterArray.length; // Die aktuelle Anzahl vorhandener Monster, zudem auch die neue Zahl für das Monster-Array.

console.log("Aktuelle Anzahl an Monstern: " + monsterCount);

monsterBtn.addEventListener( // Füge dem Monster eine Funktion hinzu.

'click', function() { // Wird bei Maus-Click ausgelöst.

fightMonster(monsterCount); // Wenn das Monster erstellt wird erhält die Funktion einen Parameter, welcher der aktuellen Anzahl entspricht.

}, false); // Ignoriert das false.

}

// Wird für den Zugriff auf eine zufällige Stelle in einem Array aufgerufen.

// [ ] Optionale Aufgabe: verkleinere diesen Code auf eine Zeile mit nur einem Semikolon!

// Muss mit einer Zahl aufgerufen werden: getRNGNumber(5); // Liefert eine ganze Zahl zwischen 0 bis 4 zurück.

function getRNGNumber(\_maxNumber : number) : number

{

let rngNumber : number = Math.random(); // Macht folgendes: Generiere eine zufällige Komma-Zahl zwischen 0 - 1.

rngNumber = rngNumber \* \_maxNumber; // Multipliziere diese Zahl mit der Länge des entsprechenden Array (hier: \_maxNumber, ein Parameter, siehe in der runden Klammer der Funktion).

rngNumber = Math.floor(rngNumber);

// Fehler // Floore diese Zahl, damit diese nun Ganzzahlig ist // Diese Zeile ist einer der drei Fehler in den Funktionen. Ich bin mal so frei und vermerke das hier. Einfach löschen und alles wird besser.

return rngNumber; // Gebe diese Zahl zurück, Funktion kann ähnlich einer Variable in Rechnungen genutzt werden.

}

// Diese Funktion gibt einen zusammengewürfelten Namen zurück.

// Wird für die Monster-generierung verwendet!

// Liefert einen zusammengesetzten String zurück.

function generateMonsterName() : string

{

let generatedMonsterName : string = ""; // Erstelle einen leeren String für das Monster

// Monster-Vorname

// Mathematik! Hier wird eine zufällig-generierte Zahl benötigt.

let rngNumber : number = getRNGNumber(prefix.length); // Der Rückgabewert der Funktion wird hier verwendet um den entsprechenden Teil des Namens (hier: Anfang) zu generieren.

generatedMonsterName = prefix[rngNumber]; // Füge den Monsternamen zusammen: nimm aus dem entsprechenden Array mit der zufallsgenerierten Zahl den entsprechenden Eintrag.

// Monster-Mittelname

rngNumber = getRNGNumber(monsterName.length); // Der Rückgabewert der Funktion wird hier verwendet um den entsprechenden Teil des Namens (hier: Mitte) zu generieren.

generatedMonsterName += monsterName[rngNumber]; // Füge den Monsternamen zusammen: nimm aus dem entsprechenden Array mit der zufallsgenerierten Zahl den entsprechenden Eintrag.

// Monster-Titel

rngNumber = getRNGNumber(suffix.length); // Der Rückgabewert der Funktion wird hier verwendet um den entsprechenden Teil des Namens (hier: Ende) zu generieren.

generatedMonsterName += suffix[rngNumber]; // Füge den Monsternamen zusammen: nimm aus dem entsprechenden Array mit der zufallsgenerierten Zahl den entsprechenden Eintrag.

return generatedMonsterName;

}

// Wird für die Monster-Lebenspunkte aufgerufen.

// Liefert eine variierende Zahl zurück.

function generateMonsterHitPoints() : number

{

// Diese Funktion gibt eine zufällige ganze Zahl (zwischen 0 und 10) + 1 zurück.

let tempMonsterHP : number = 1 + getRNGNumber(10);

return tempMonsterHP;

}

// Wird für die Erstellung der Monster-Lebenspunkte aufgerufen.

// Liefert eine variierende Zahl zurück.

function generateMonsterXP() : number

{

// Diese Funktion gibt eine zufällige ganze Zahl (zwischen 0 und 350) + 100 zurück.

let tempMonsterXP : number = 100 + getRNGNumber(350);

return tempMonsterXP;

}

// Wird für die Erstellung der Monster-Modifizierer aufgerufen.

// Liefert ein Array mit zwei Einträgen zurück.

function generateMonsterModifer() : string[]

{

let tempMonsterMod : string[] = []; // Initialisiere ein leeres Array (verhindert Folge-Fehler)

tempMonsterMod[0] = monsterModifers[getRNGNumber(monsterModifers.length)]; // Setze Schublade 0 des Arrays auf einen Wert.

tempMonsterMod[1] = monsterModifers[getRNGNumber(monsterModifers.length)]; // Setze Schublade 1 des Arrays auf einen Wert.

return tempMonsterMod; // Gebe das hier zusammengesetzte Array wieder zurück.

}

function generateMonsterWeapon () {

let tempMonsterWeapon = [];

tempMonsterWeapon [0] = monsterWeapon [getRNGNumber(monsterWeapon.length)];

return tempMonsterWeapon;

}

// Aufgerufen, wenn man auf den Button klickt.

// Der Spieler kämpft gegen das entsprechende Monster. Er erhält dann Erfahrungspunkte.

function fightMonster(\_index : number)

{

console.log("Spieler kämpft gegen Monster und gewinnt!"); // Ohne Logik mit if/else ist so etwas wie ein Kampf nicht leicht umzusetzen.

console.log("Das Monster weigert sich zu verschwinden."); // Wird nächste Stunde erweitert.

playerXP += monsterArray[\_index - 1].monsterExperience;      // \_index ist in diesem Fall die Länge des Arrays - allerdings zählt der Computer beginnend von null, nicht eins! Deshalb \_index-1.

updatePlayerLevel();

}

// Aufgerufen, um das HTML-Element, welches das Spieler-Level darstellt, zu erneuern.

function updatePlayerLevel()

{

let tempLevel : number = Math.floor(playerXP / playerXPperLevel); // Spieler-Level = XP / XPproLevel

document.getElementById("xpCounter").innerHTML = "Player-Level: " + tempLevel + " (XP: " + playerXP + " / " + playerXPperLevel + ")"; // Baue den String für die Spieler-Info zusammen

console.log("Spieler " + playerName + " hat nun Level " + tempLevel + " mit " + playerXP + " (" + playerXPperLevel + " pro Level)"); // Spieler-Level in der Konsole.

}