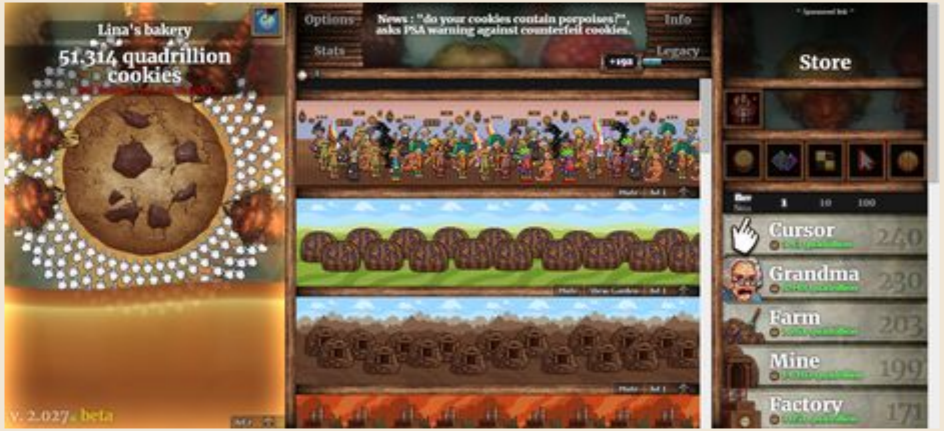
Cookie Clicker

Event-driven programming in JavaScript

Chris Dalziel and Adam Chapman-Dodd,
with design help from Moth Dalziel

What is Cookie Clicker?

Cookie Clicker is a web-based incremental video game written in JavaScript by French developer Julien “Orteil” Thiennot.


Lina's bakery
51.314 quadrillion cookies
Options
Stats
News : "do your cookies contain porpoises?", asks PSA warning against counterfeit cookies.
Info
Legacy
Store
Cursor
240
Grandma
230
Farm
203
Mine
199
Factory
171
v. 2.027 beta

Screenshot from https://en.wikipedia.org/wiki/Cookie_Clicker

What is Cookie Clicker?

The game has a simple progression loop revolving around the collection and spending of cookies, which serve both as points and as a kind of currency.

Cookies can be spent to purchase upgrades and auto-clickers, which in turn allows for the collection of more cookies.

Try Cookie Clicker!

Try it at https://orteil.dashnet.org/cookieclicker/ for five-ish minutes.

What are the most important parts of the game? How might you implement these?

Event-based Programming

An **event** is an interaction between the user and the webpage.

Event-based programming is a kind of reactive “When x do y” approach, which allows web-pages to react to the actions of the user dynamically.

Event	**Description**
click	User clicks an element
mouseover	User moves the mouse over an element
onmouseout	User moves the mouse off an element
keydown	User presses a key

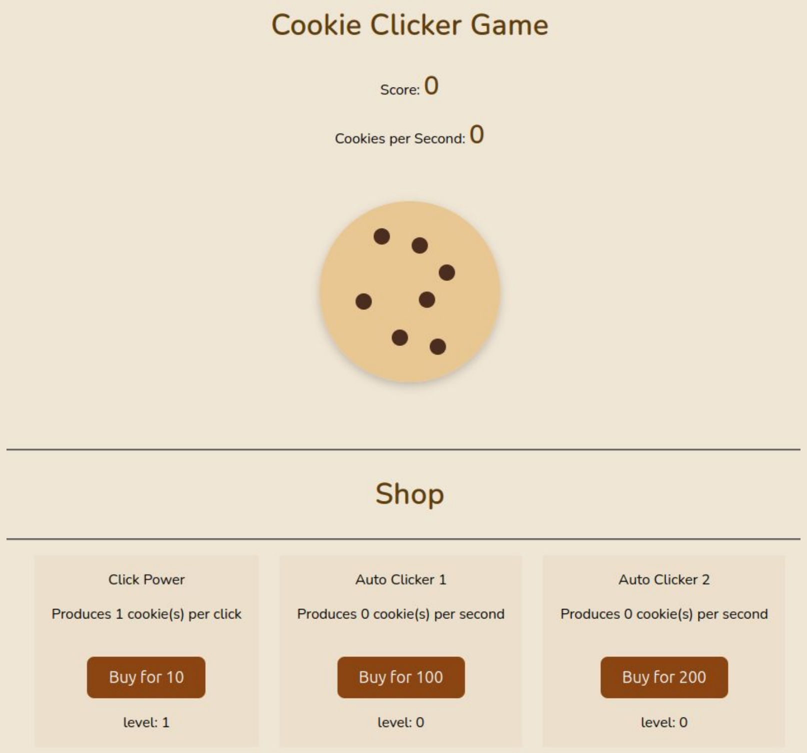
There are loads of events, not just these: https://www.w3schools.com/jsref/dom_obj_event.asp

Event-based Programming

Event-based approaches are also used in designing video games, where user interactions are of central importance.

We can use event programming and the “click” event to build our own Cookie Clicker!

The provided project file contains an incomplete implementation of a Cookie Clicker game. Some of the code is missing, and there are bugs which need to be addressed.



Materials adapted from https://www.geeksforgeeks.org/how-to-make-a-cookie-clicker-using-html-and-javascript/

`id` and `class` in HTML

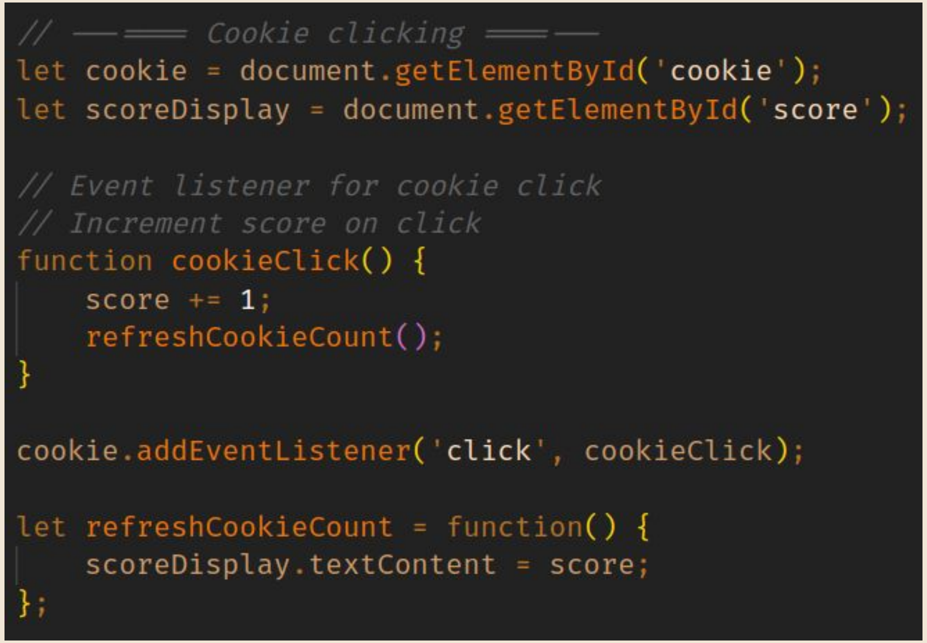
`id` and `class` are HTML attributes which can be set for any given element tag, and allow us to set properties for those elements. A `class` can be given to many elements, where an `id` is unique (and therefore identifying).

At National 5 we use these purely to set the styles of elements with CSS, but they can also be used with JavaScript to create “event handlers” for an element.

Event Listeners - Learn

An **event listener** is a piece of code which waits for an event to happen, and runs function whenever it does.

In this example, the code which updates the user's score runs whenever the cookie is clicked.



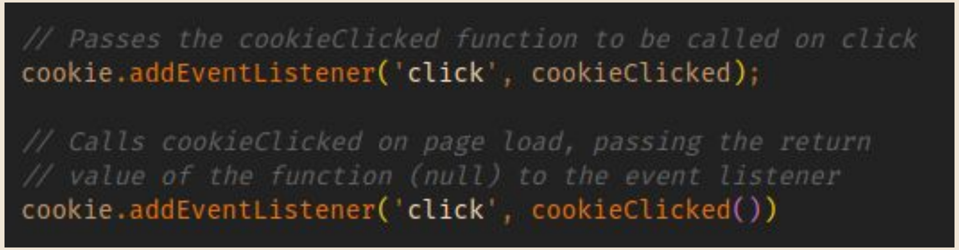
```
// ——═══ Cookie clicking ═══——
let cookie = document.getElementById('cookie');
let scoreDisplay = document.getElementById('score');

// Event listener for cookie click
// Increment score on click
function cookieClick() {
    score += 1;
    refreshCookieCount();
}

cookie.addEventListener('click', cookieClick);

let refreshCookieCount = function() {
    scoreDisplay.textContent = score;
};
```

Aside — Functions as Arguments

JavaScript allows for functions themselves to be passed as an argument to other functions.

This allows the function to be called from within these other functions, rather than just passing the result of these functions.

This is useful for event listeners because we don't necessarily want to call the function immediately when the page loads.



```
// Passes the cookieClicked function to be called on click
cookie.addEventListener('click', cookieClicked);

// Calls cookieClicked on page load, passing the return
// value of the function (null) to the event listener
cookie.addEventListener('click', cookieClicked())
```

Event Listeners - Do

Your first task is to use an event listener to implement the most important feature in the game — a clickable cookie!

Event Listeners - Discuss

- Why do we separate logic (JS), styling (CSS) and structure (HTML) rather than keep it all in the same file?
- How might you use different types of events (like `mouseenter`, `dblclick` or `keydown`) to change how players interact with this game?
- Why is `refreshCookieCount()` called after increasing the score? What would happen if we forgot to call it?
- How could you make this click interaction more engaging or apparent for a user?

Event Listeners - Summary

You can use different types of ***events***, like `keydown`, to make a game more interactive and accessible (e.g. making the spacebar act like cookie click allows the webpage to be usable even by those who might struggle with mouse.

Event listeners allow us to separate the behaviour of the game from the design and keep ***functions*** running **if and only if a specific *event* occurs** - like clicking the cookie.

This first task should have shown you the basics of developing interactive webpages: ***event listeners*** are attached to elements and the necessary functions are called when the ***event*** occurs.

Periodic Functions - Learn

One “event” we might want to react to is the passage of a set amount of time.

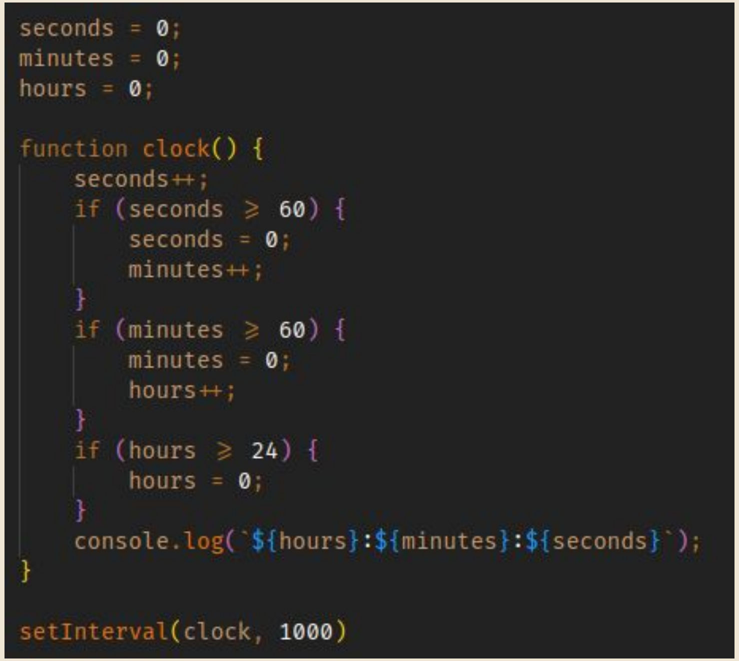
The `setInterval` function lets us run a function every time the provided interval has passed, allowing for functions which run repeatedly with a delay.

Periodic Functions - Learn

This `clock()` function increments the number of seconds that have passed and then adjusts the minutes and hours.

For this to work correctly, we must call the `clock()` function exactly every second, which we do by passing the function and 1000 milliseconds to `setInterval`.

Similarly to passing functions earlier we *don't* put brackets on the function name in the `setInterval` call.



```
seconds = 0;
minutes = 0;
hours = 0;

function clock() {
    seconds++;
    if (seconds >= 60) {
        seconds = 0;
        minutes++;
    }
    if (minutes >= 60) {
        minutes = 0;
        hours++;
    }
    if (hours >= 24) {
        hours = 0;
    }
    console.log(`${hours}:${minutes}:${seconds}`);
}

setInterval(clock, 1000)
```

Periodic Functions - Do

Now you should look at fixing the cookies per second counter, using the `setInterval` function to update it every second.

Periodic Functions - Discuss

- Why use `setInterval()` instead of updating CPS on each click? What problems could occur if CPS was updated on every frame?

- Why do we need to reset the counter each second? What happens if you don’t do this?

Periodic Functions - Summary

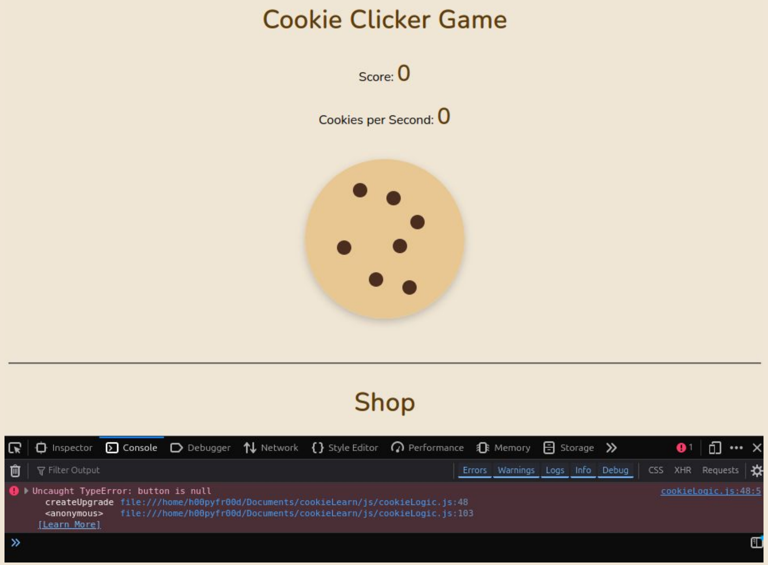
Periodic functions, like `setInterval()`, allow us to run tasks at regular intervals (such as updating the CookiesPerSecond value and updating the display every second). This is especially useful when we want to **perform actions purely based on time instead of any user interactions**, which allows the game to update its elements without waiting for a click to be detected with event listeners.

This task should have helped you understand that **some elements need to be updated periodically**: if we were waiting for the cookie to be clicked to update, the CPS meter may not change when it should, and if we updated it every frame, the display may not be smooth or even legible.

Console debugging and Functions - Learn

The console tab of the debugger (accessible via Ctrl+Shift+K) can be very useful for finding bugs, as it shows any errors or warnings produced by your code.

Additionally you can print your own messages to the console using `console.log()`.

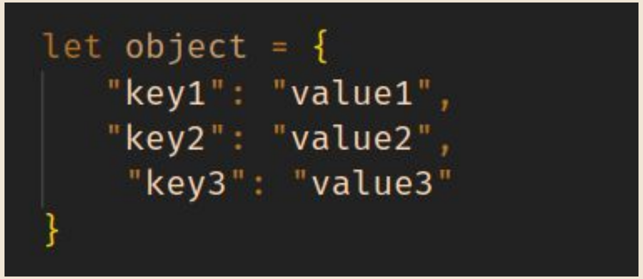


```
Uncaught TypeError: button is null                                          cookieLogic.js:48:5
    createUpgrade file:///home/h00pyfr00d/Documents/cookieLearn/js/cookieLogic.js:48
    <anonymous>   file:///home/h00pyfr00d/Documents/cookieLearn/js/cookieLogic.js:103
  [Learn More]
```

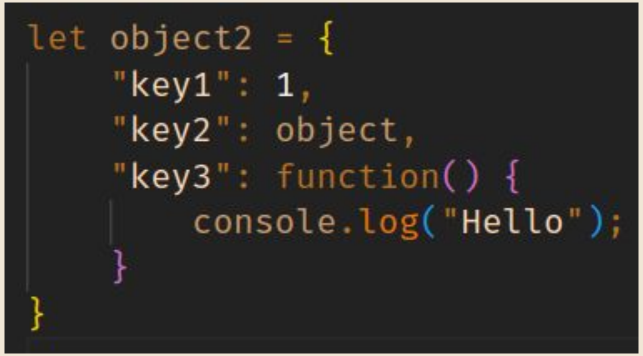
Aside — JavaScript Object Notation (JSON)

JavaScript's `Object` type is very versatile, as it allows us to collect many values into one object.

JavaScript Object Notation (more commonly abbreviated to JSON) lets us define the properties and values of an object using key-value syntax. The keys must be strings (quotes may be omitted), but the value may be any type — even functions and other objects.



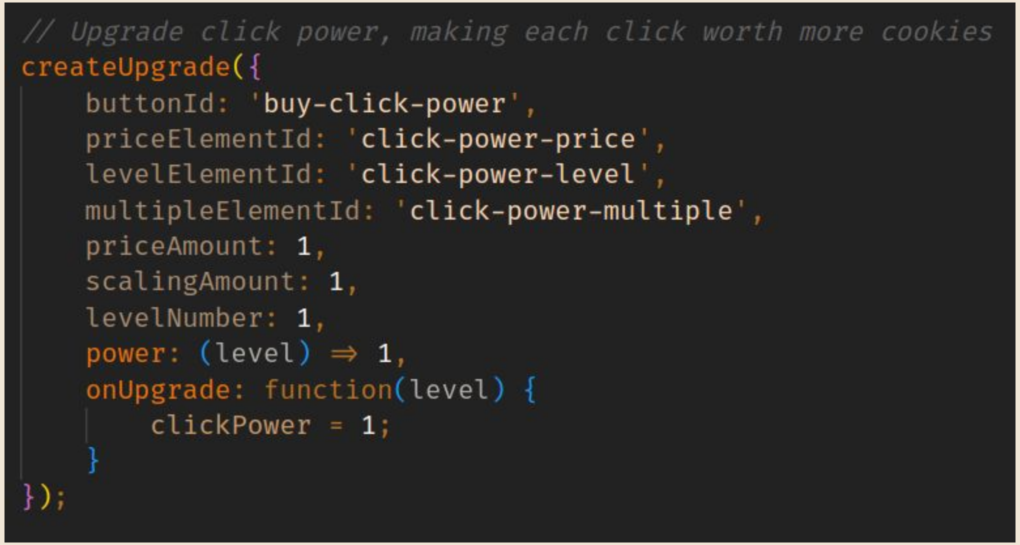
```
let object = {
   "key1": "value1",
   "key2": "value2",
    "key3": "value3"
}
```



```
let object2 = {
    "key1": 1,
    "key2": object,
    "key3": function() {
        console.log("Hello");
    }
}
```

Aside — JavaScript Object Notation (JSON)

We use JSON in the `createUpgrade` function to make it clearer what each argument is meant to be when calling the function.



```
// Upgrade click power, making each click worth more cookies
createUpgrade({
    buttonId: 'buy-click-power',
    priceElementId: 'click-power-price',
    levelElementId: 'click-power-level',
    multipleElementId: 'click-power-multiple',
    priceAmount: 1,
    scalingAmount: 1,
    levelNumber: 1,
    power: (level) ⇒ 1,
    onUpgrade: function(level) {
        clickPower = 1;
    }
});
```

Console debugging and Functions - Do

Now you should look at fixing the first upgrade to scale the correct values and, using console debugging, find and correct the error in the second upgrade making sure that the correct values are changed and the necessary elements updated every second.

Then use functions to create your own auto-clicker upgrade, making sure to add it as a HTML element.

Console debugging and Functions - Discuss

- Why is it a good idea to have one function like `createUpgrade()` handle upgrade setup?

- What is the issue with upgrade prices not scaling with level?

- How do HTML element IDs help JavaScript interact with the correct parts of a web page? What would happen if IDs are reused?

Console debugging and Functions - Summary

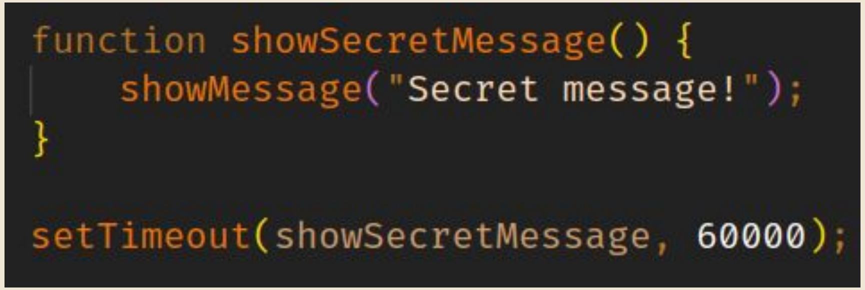
Using ***functions*** **helps to manage code and keep it organised**: in this case, the `createUpgrade()` function makes it much more efficient to create, update and manage upgrades in the shop. Reusing code in this way **makes it much easier to scale** as the game gets bigger and more complex: this is a large part of a game of this nature.

Using the ***console*** to check things are working as intended is an incredibly important skill. Being able to use ***error messages*** to find where the mistakes are in code means you can create a robust site without worry of things not working as intended.

Timed Functions - Learn

Sometimes we'd like to time a function to happen some interval in the future, but not necessarily periodically.

This example shows a secret message a minute after the page loads. We only want to show the message once, so we use a timed function instead with `setTimeout`.



```
function showSecretMessage() {
    showMessage("Secret message!");
}

setTimeout(showSecretMessage, 60000);
```

Timed Functions - Do

Fill in the missing code in the `spawnGoldenCookie()` function definition to assign random values to the golden cookies being spawned.

Read through the function and start the cycle of spawning these bonus cookies.

The golden cookies doesn’t have any chips. Use CSS and HTML to add some!

Timed Functions - Discuss

- Why do we use `Math.random()` for the values of bonus cookies? How could you skew that to be weighted towards lower numbers?

- Why do we use `setTimeout()` instead of `setInterval()` for controlling how long a golden cookie stays on screen?

- How could we make golden cookies rarer or more powerful? How would you start with implementing that?

Timed Functions - Summary

Timed functions, like `setTimeout()`, allow us to run tasks once after a delay (such as despawning the golden cookie after a certain duration has passed). This is useful when we want to **perform an action based on time instead of any user interactions** but only want to do it once rather than periodically.

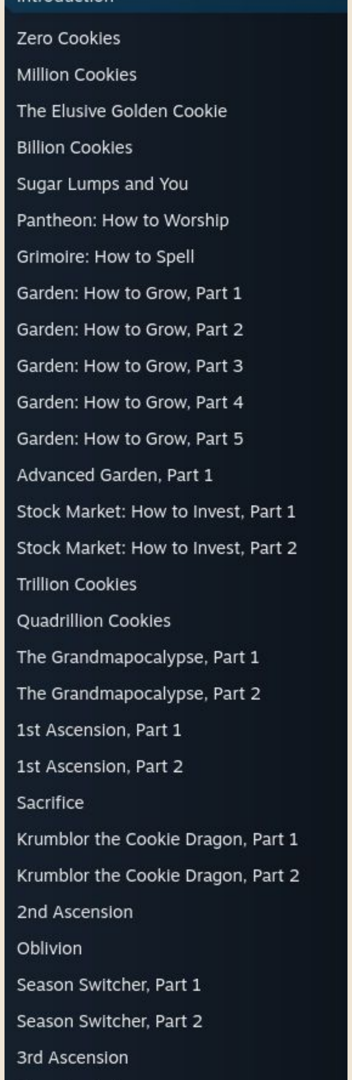
This task should have helped you understand that **some elements need to be updated only once after a certain amount of time**: in this instance, if we tried to instead use a periodic function, `setInterval()`, the code would keep trying to remove the golden cookie over and over again rather than only once, which doesn’t make sense.

Where would you take it next?

Cookie Clicker has years of content — just look at this portion of a beginners Steam community guide!

Some of these mechanics are super out there, the potential for new ones is limitless: what would you add?

https://steamcommunity.com/sharedfiles/filedetails/?id=2591407770



Zero Cookies
Million Cookies
The Elusive Golden Cookie
Billion Cookies
Sugar Lumps and You
Pantheon: How to Worship
Grimoire: How to Spell
Garden: How to Grow, Part 1
Garden: How to Grow, Part 2
Garden: How to Grow, Part 3
Garden: How to Grow, Part 4
Garden: How to Grow, Part 5
Advanced Garden, Part 1
Stock Market: How to Invest, Part 1
Stock Market: How to Invest, Part 2
Trillion Cookies
Quadrillion Cookies
The Grandmapocalypse, Part 1
The Grandmapocalypse, Part 2
1st Ascension, Part 1
1st Ascension, Part 2
Sacrifice
Krumblor the Cookie Dragon, Part 1
Krumblor the Cookie Dragon, Part 2
2nd Ascension
Oblivion
Season Switcher, Part 1
Season Switcher, Part 2
3rd Ascension

Summary

- **Events** are user interactions with the page.
- We can react to events dynamically using **event listeners.**
- If we want to **delay** the call of a function we can use `setInterval` and `setTimeout`, where an interval function will repeat periodically and a timeout function will run once (unless called within itself!).