FULL SAIL UNIVERSITY

scripting for web applications 2

jQuery events and animation

target.Review

```
$("#nav > li")...
```

```
$("#nav a[data-id='001']")...
```

```
$("#nav li:first)...
```

```
$("#nav li:not(.active)")...
```

use IDs for parent-level items (singular)
uses classes for repeatable elements (like lists/collections)

target.**Review**

```
<ul id=”nav”>
   <li></li> <li></li> <li></li> <li></li>
</ul>
```

```
$(“#nav li”).css().filter(“:odd”).css().parent().css();
```

```
[
<li/>,
<li/>,
<li/>,
<li/>
]
```

```
[
<li/>,
<li/>,
]
```

```
[
<ul id=”nav”/>
]
```

manipulation.Review

```
var html = '<a href="">Link</a>';
```

```
$(html)
    .appendTo('#nav')
    .animate()
;
```

```
$('#nav')
    .append(html)
    .animate()
;
```

```
append
appendTo
prepend
prependTo
```

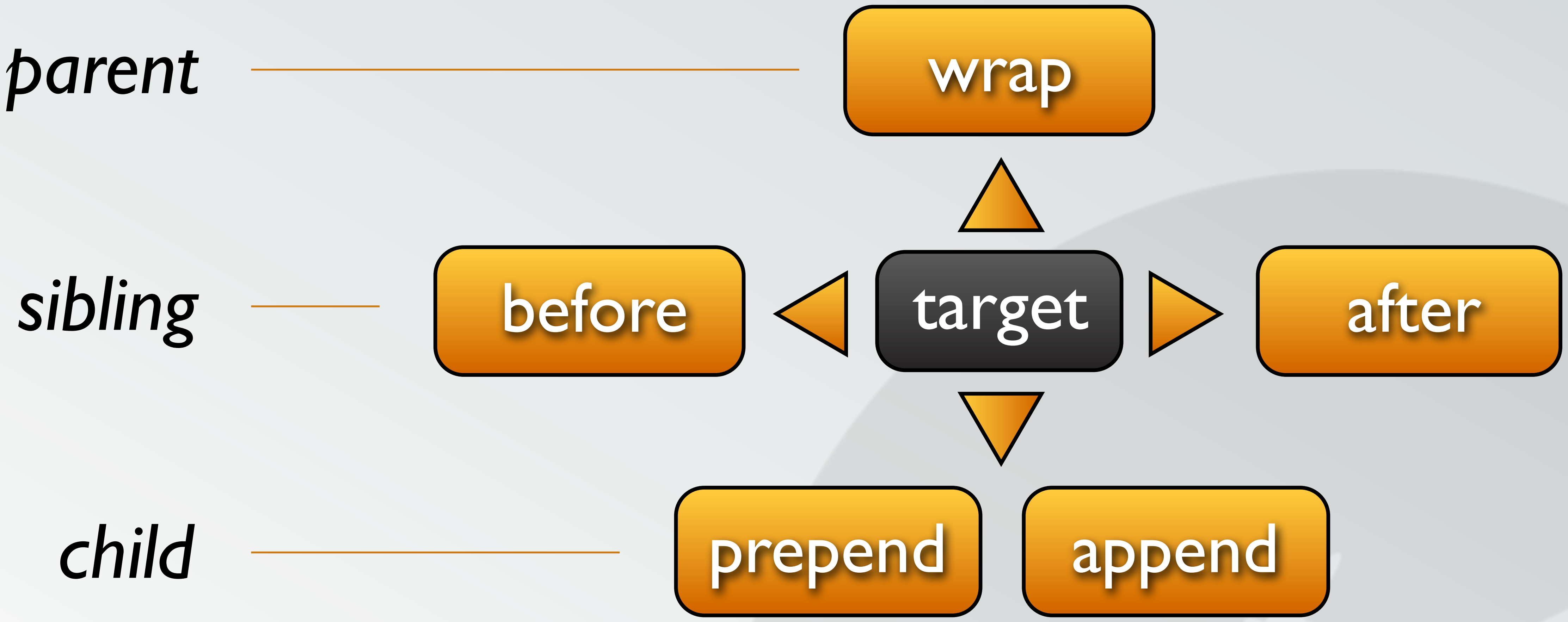
```
after
insertAfter
before
insertBefore
```

```
replaceWith
replaceAll
clone
```

```
wrap
wrapAll
wrapInner
```

```
remove
empty
```

manipulation.**Review**



jquery.**Events**

DOM Event Model

- In PWA1 you explored the basic **Event Model** of the DOM.
- There are several key parts to how the browser interprets and handles events, and you used JavaScript to assign and control those event actions.
- Let’s re-examine the Event Model:

jQuery.**Events**

DOM Event Model

- When a DOM element triggers an event, an “**event**” is always created
- We can create functions called “**handlers**” that are called to do something

Anatomy of an Event

- Events have 2 components:
  - The DOM element we listen on
  - The function we assign to that listener

jQuery Events

- So what does jQuery provide us?
  - Chainable methods for binding event handlers,
  - Allows multiple handlers to be bound to each event type,
  - Delegated event model,
  - Provides a cross-browser-compatible **event object**,
  - Provides cross-browser methods for canceling bubbling and browser-defaults

jQuery.Events

jQuery Events (OLD School)

- There are a few different methods for event bindings in jQuery. The most basic is a method *type* where the *event name* is the name of the method itself.
- The argument is what function to use as the handler *(can be a reference, or a literal)*

Event Method	Example
click(*fn*)	$(“a”).click(*function(){}*);
mousemove(*fn*)	$(“a”).mousemove(*function(){}*);
mouseup(*fn*)	$(“a”).mouseup(*function(){}*);
mousedown(*fn*)	$(“a”).mousedown(*function(){}*);
keyup(*fn*)	$(“input”).keyup(*function(){}*);
etc...	

jQuery.Events

.on( )

- This is the preferable, source method for event binding
- Pre version 1.7 the command was .bind()

$(target).on(*type, data, function*)	***type*** (string) eg - “click” ***data*** (object) *optional* custom event data ***fn*** (function) event handler

```
$("#link").on("click", {myvar:"test"}, function(e){
  alert(e.data.myvar);
  return false;
});
```

jQuery.**Events**

event information

- Any function bound to an event will only receive 1 argument, **the event object**
- The **event object** contains information about what happened in the event

```
$("#mylink").on("click", function(e){
  alert(e.type);
  return false;
});
```

jQuery.**Events**

e.Property	*Description*
type	*string:* The name of the event type *(ie- "click" or "mouseleave")*
target	*object:* DOM reference to the element that triggered the event. *(if a child element is the source, it will be the trigger)*
currentTarget	*object:* DOM reference to the current element in the bubbling chain. Note: *currentTarget is equal to the value of* ***this***
relatedTarget	*object:* DOM reference for mouse event issues
timeStamp	*number:* Date timestamp of when the event was triggered *(in ms)*
which	*number:* Normalized key code to use instead of keyCode or charCode
pageX / pageY	*number:* The x/y event position, relative to the *document page*.
screenX / screenY	*number:* The x/y event position, relative to the *client's screen*.
data	*object:* The custom event object, if used.
namespace	*string:* The custom namespace, if used.

jQuery.**Events**

Event Context

- You learned that all functions have a ***context*****,** which is the object that the function was assigned to.
- In events, the context of our function is the ***element that fires the event***.
- Context is an object, called **this**

```
$(“#box”).on(“click”, function(){
    console.log( this );
    return false;
});
```

Saving the Context

- A common trick is to create a variable called **that**, with the **this** jQuery object
- Reduces factory calls and creates a localized store

```
$("a:first").on('click', function(){
    var that = $(this);
    that.css({background: 'red'});
    return false;
});
```

jQuery.**Events**

- Let’s look at a few other notable event types:

Event Method	Description
mouseover(*fn*)	Triggers when the cursor enters the element’s area *or* enters the area of a child element *(this event may trigger* ***multiple times****)*.
mouseout(*fn*)	Triggers when the cursor leaves the element’s area *or* the cursor leaves a child element *(this event may trigger* ***multiple times****)*.
mouseenter(*fn*)	Triggers only ***once*** when the cursor enters the element’s area, not including any children elements. Only exist in jQuery. Replaces mouseover(fn)
mouseleave(*fn*)	Triggers ***once*** only when the cursor leaves the element’s area. Only exist in jQuery. Replaces mouseleave(fn)

jQuery.Events

- Let's look at a few other notable event types:

Event Method	Description
focusin(*fn*)	This is a fixed version of *focus*, to include bubbling and child detection. This method is a shortcut for .on('focusin', handler).
focusout(*fn*)	The *blur* version of focusin.
load(*fn*)	Can be used on any element to detect when that element has been rendered to the page *(useful for images, scripts, iframes)*

jQuery.Events

.off( )

$(target).off()	No arguments, this will remove *all* events from *target*
$(target).off(*type*)	Removes the specified event *type* from *target*
$(target).off(*type, handler*)	If a named function was used, you can unbind just that handler by passing its name

```
var hn = function(e){
  return false;
};
$(“a”).on(‘click’, hn);
$(“a”).on(‘click’, function(){});
```

```
$(“a”).off();
```

```
$(“a”).off(“click”);
```

```
$(“a”).off(“click”, hn);
```

jQuery.**Events**

Binding Multiple Events w/ one Handler

$(target).on(*type, data, function*)

- You can bind multiple events to the *same function* by using **space(s) in the type string**

```
$("#link").on("mouseenter mouseleave", function(e){
    return false;
});
```

jQuery.Events

Binding Multiple Events w/ Multiple Handler

$(target).on(*object*)	Object with events as keys, paired with function handlers

- Using an object, you can bind multiple individual events at the same time.

```
$("#box").on({
    click: function(e){},
    mouseenter: function(e){},
    mouseleave: function(e){}
});
```

Custom Event Namespaces

- Add a class name to turn on and off a bind, by name

$(target).on(*type***.***namespace, data, function*)

```
$("#box").on("click.topmenu", function(e){
    return false;
});
```

```
$("#box").off("click.topmenu");
```

jQuery.Events

.one( )

- Exact same as .on, except this handler will self-destruct after 1 use

$(target).one(*type, data, function*)	Binds an event hander *function* to *event* as normal, except the handler is automatically unbound after the event is triggered once.

```
$(“#link”).one(“click”, function(e){
  return false;
});
```

jQuery.Events

.toggle( )

- A specialized “click” listener, alternates between multiple functions automatically

$(target).toggle(*oddFn, evenFn*)	*oddFn:* function fires for odd *n*th clicks (1st, 3rd, etc) *evenFn:* function fires for even *n*th clicks (2nd, 4th, etc)

```
$(“#link”).toggle(
    function(e){         // odd function handler
    },
    function(e){         // even function handler
    }
);
```

delegated.**Events**

event.Delegation

$(window).on(*target, type, function*)	Binds the event listener to the global *window* object, and delegates to the *target*

```
$(window).on('#nav a', 'click', function(e){});
```

event.**Delegation**

- Additionally, the delegated **on** events cannot be removed normally, will need use **.off**

$(window).off(target, type)	Unbinds all instances of the specified delegated *".on" event type* for the *target selector.*

```
$(window).off('#nav a', 'click');
```