



**FULL SAIL**  
UNIVERSITY<sup>TM</sup>

programming for web applications 2



jQuery selectors and manipulators

# Course Review

# PWA-2 DUE Dates

Item	Due Dates
Branding / Logo	08/05/13 - After Lab on the First Day
Project Pitch	08/09/13 - Before Lecture on 3
Creative Brief - Finished Document	08/12/13 - Before Lecture 4
Site Prototype ( <i>html/css</i> )	08/16/13 - After Last Lab of the 2nd Week
Development Milestone ( <i>javascript</i> )	08/19/13 - Due End of Lab 7
Inclusion of 5 media center items	08/28/13 - Last Day of Class After Lab
Aesthetics & Usability ( <i>finished site</i> )	08/28/13 - Last Day of Class After Lab
Functionality ( <i>finished site</i> )	08/28/13 - Last Day of Class After Lab
Professionalism	The duration of the course
Class Participation	The duration of the course

# jQuery Setup (REVIEW)

## ▶ Setup

- ▶ 1. Create and save a jQuery minified file (.js) (slide 5)
- ▶ 2. Create a .js file for your jQuery program
  - ▶ Insert DOM Ready statement (slide 6)
- ▶ 3. Setup the jQuery minified file within the .HTML file
  - ▶ For Development: use a local copy of jQuery
  - ▶ For Production: use the Google CDN version



FULL SAIL  
UNIVERSITY

go to: [jquery.com](http://jquery.com) (REVIEW)

The screenshot shows the official jQuery website. At the top, there's a blue header bar with navigation links: Download, API Documentation, Blog, Plugins, and Browser Support. To the right is a search bar. Below the header, there's a dark grey section with three main features: "Lightweight Footprint" (represented by a cube icon), "CSS3 Compliant" (represented by a stylized '3' icon), and "Cross-Browser" (represented by a globe with arrows icon). Each feature has a brief description below it. To the right of these features is a large orange button with a download icon and the text "Download jQuery v1.10.2 or v2.0.3". Below this button are two links: "View Source on GitHub →" and "How jQuery Works →".

download both (*minified (compressed) and uncompressed*)



FULL SAIL  
UNIVERSITY

# jQuery Setup (REVIEW)

- ▶ The “dom ready” callback

```
$(document).ready(function() {
    // all site code
});
```

```
$(function() {
    // all site code
});
```

- ▶ This argument is a nameless function literal. It should already look familiar, as this is how our **DOMReady** method worked last month.
- ▶ In the same manner, any code inside this function is run when the DOM is loaded.



FULL SAIL  
UNIVERSITY

# jQuery Setup (REVIEW)

## › jQuery Setup

### In the .HTML file

**Use for LIVE sites : Google's CDN**

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.3/
jquery.min.js"></script>
```

Or

**Use during the DEVELOPMENT of a jQuery website**

```
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
```

- › Place one of the script tags above the </body> tag.



FULL SAIL  
UNIVERSITY

# jQuery selectors



- Selecting Items in the DOM

- download this program to try out using jQuery SELECTORS and see them applied:

in FSO's Reference Tab:  
**Course Material Follow Along Files**  
**'Goal3.zip'**



## ▶ Targeting an HTML Element

- ▶ As we've learned, jQuery's methods (*such as css or attr*) **almost always need a target. Most uses in jQuery revolve around manipulating html elements inside the DOM, but we have to select them first.**
- ▶ Essentially, we'll be passing a ***string*** to the jQuery factory that "selects" which elements to perform all of the following methods on.
- ▶ The ***selector set*** creates an array of items.



## ▶ Targeting with Selectors

- ▶ We've already looked at some of the basic selector options, and we've seen how jQuery utilizes a CSS syntax to make this even easier.
- ▶ Our basic selectors will use: **ids**, **classes**, or **element tags**, such as:

```
$("#myid")      $(".myclass")      $("p")
```

- ▶ Most advanced selectors still begin with these, for example

```
$("#myid > .topnav:first a[href][rel]")
```



## ▶ Targeting with Selectors

- ▶ jQuery supports normal CSS syntax for sub-selections, combining the basic selector types... such as:

```
$("#myid ul li")    $("div.hilite a")
```

- ▶ In the first example, using a space in this way is known as the **ancestor-descendant** relationship, as opposed to the **parent-child** alternative.
- ▶ It's important to remember how these CSS selectors work. Keep in mind that these will make *nested* selections. For example .. \$("div.hilite a") .. will target *all* anchor tags inside the div.hilite elements, no matter how deeply nested the links are.



## Selectors

```
<div class="hilite">
  <a href=""></a>
  <a href=""></a>
  <ul>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
  </ul>
  <a href=""></a>
  <a href=""></a>
</div>
```



## Selectors

```
$(“div.hilite a”)
```

```
<div class="hilite">
  <a href=""></a>
  <a href=""></a>
  <ul>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
  </ul>
  <a href=""></a>
  <a href=""></a>
</div>
```



## Selectors

```
$(“div.hilite a”)
```

```
<div class="hilite">
  <a href=""></a>
  <a href=""></a>
  <ul>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
  </ul>
  <a href=""></a>
  <a href=""></a>
</div>
```



## ▶ Targeting with Selectors

- ▶ Several of the more advanced jQuery selectors utilize syntax from CSS2 and CSS3. By using jQuery however, we have a cross-browser compatible format for selecting more detailed targets.
- ▶ For example, we can use `>` to select only direct children, instead of descendants, such as:

```
$("div.hilite > a")
```

- ▶ In this example now, only `<a>` tags that are direct children of `div.hilite` will be selected.



## Selectors

```
<div class="hilite">
  <a href=""></a>
  <a href=""></a>
  <ul>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
  </ul>
  <a href=""></a>
  <a href=""></a>
</div>
```



## Selectors

```
$(“div.hilite > a”)
```

```
<div class="hilite">
  <a href=""></a>
  <a href=""></a>
  <ul>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
  </ul>
  <a href=""></a>
  <a href=""></a>
</div>
```



## Selectors

```
$(“div.hilite > a”)
```

```
<div class="hilite">
  <a href=""></a>
  <a href=""></a>
  <ul>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
  </ul>
  <a href=""></a>
  <a href=""></a>
</div>
```



## ▶ Targeting with Selectors

- ▶ Multiple selector expressions can also be combined using commas.
- ▶ As a basic example, if I wanted to target all divs and paragraphs:

```
$("div, p")
```

- ▶ The comma acts as an *OR* operator, allowing results of either matched set. Either set can be as complex as needed:

```
$("div > a:first, p:even")
```



## ▶ Targeting with Selectors

- ▶ Our next selector type **targets by element attributes**. Consider the following elements:

```
<a href="http://fullsail.com">Full Sail</a>

```



## ▶ Targeting with Selectors

- ▶ Notice the attribute selector syntax follows an element with **[attr]**. In this selector, **spacing is important**, [attr] selector must directly follow a selector (to indicate connection). Some more examples:
- ▶ You can choose an element that simply has an attribute, no matter the value:

```
$("img[title]")
```

- ▶ ^= selects text at the *beginning* of an attribute's value:

```
$("a[href^='http://'])")
```



```

```

## Selectors

### ‣ Targeting with Selectors

- To select text at the *end* of the value, use `$=`

```
$(“img[src$=.jpg]”)
```

- To match text *anywhere* in the value, use `*=`

```
$(“img[src*=puppy]”)
```

- We can also match where the text is *not* inside the value, with `!=`

```
$(“img[src!=kitten]”)
```

```
$(“img[src=puppy1.jpg]”)
```



## ‣ Targeting with Selectors

- Multiple attribute selectors can be applied to the same element by simply repeating the **[attr]** syntax.
- For example, If I wanted to match any external link that pointed google.com, inside my content area:

```
$ (“#content a[href^=http://][href*=google.com]”)
```



## ▶ Selecting with Filters

- ▶ Our next selector syntax are **filters**. Filters take a set of matches and reduces or refines the set. Filters, like [attr], must directly follow a selector without space. The syntax looks like...

```
$("selector:filter")
```

- ▶ For example...

```
$("li:first")
```

```
$("li:even")
```



## ▶ Selecting with Filters

- ▶ Let's look at some of the basic filter options:

Filter	Operation
:first	First item in the current selector set ( <i>set is reduced to 1 item</i> )
:last	Last item in the current selector set ( <i>set is reduced to 1 item</i> )
:first-child	Selects matches that are the first child of their parent
:last-child	Selects matches that are the last child of their parent
:only-child	Selects matches that are the only-child of their parent
:even	Even instances of selector, by array (0th, 2nd, 4th, 6th, etc)
:odd	Odd instances of selector, by array (1st, 3rd, 5th, etc)
:header	Matches instances of header tags (h1, h2, h3, etc)



## ▶ Selecting with Filters

- ▶ Notice how the *even* and *odd* basic filters use instances like array indexes, beginning at 0 as the first *even* instance.
- ▶ Below are some more filters that use this array-like syntax for further filtering:

Filter	Operation
:eq( <i>n</i> )	Matches the <i>n</i> th selector match :eq(0) is same as :first
:gt( <i>n</i> )	Matches elements after (excluding) the <i>n</i> th selector match
:lt( <i>n</i> )	Matches elements before (excluding) the <i>n</i> th selector match



## ▶ Selecting with Filters

- ▶ While the *odd*, *even*, *eq*, *gt*, and *lt* are direct filters that use an array-like index, the **:nth-child( )** filter only selects children, and **begins at 1 instead of 0**.
- ▶ For example, *1n<sup>th</sup>* is the *1st* child.

Filter	Operation
<code>:nth-child(n)</code>	Matches only the <i>n<sup>th</sup></i> child ie- <code>\$(“ div:nth-child(3) ”)</code>
<code>:nth-child(formula)</code>	...



## ▶ Selecting with Filters

- ▶ A formula of repeatable selects can also be used in **:nth-child()**. The syntax is:

`:nth-child(Xn+Y)`

Match the Yth element after every Xth element

- ▶ For example: `$(“div:nth-child(3n+1)”)`

- ▶ This example would match: 4, 7, 10, 13, etc.

- ▶ Without a Y: `$(“div:nth-child(3)”)`

- ▶ Matches every 3rd element: 3, 6, 9, 12



## Selectors

Filter	Operation
:parent	Matches elements that have at least 1 child ( <i>which can include text nodes</i> )
:empty	Matches elements with no children at all ( <i>including text</i> )
:has(selector)	Matches the element if they contain an instance of the provided selector  <code>\$("li:has(div)")</code>
:not(selector)	Matches elements of only if they are not instances of the provided selector  <code>\$("li:not(.navs)")</code>  <b>not</b> is the opposite of a normal selector  <code>\$("li.navs")</code>



## ❖ Selecting with Filters

Filter	Operation
:hidden	Matches elements currently hidden ( <i>hidden is defined as the element OR its parent consuming no visible space</i> )
:visible	Opposite of hidden, using the same conditions. jQuery animations <i>do</i> affect these filters (such as <code>fadeOut</code> or <code>slideUp</code> )
:animated	Matches elements currently being animated by a jQuery method (such as <code>animate</code> , <code>fadeOut</code> , <code>slideUp</code> , etc)



## ▶ Selecting with Filters

- ▶ jQuery also provides filters for specifically targeting elements of *forms*. A few key examples are below (*not a complete list*)

Filter	Operation
:input	(i.e. input, select, textarea, button)
:button	(i.e. button, submit)
:password	
:submit	
:reset	
:checkbox	:checked
:radio	:selected
:text	



## ▶ Selecting with Filters

- ▶ Filters can also be **stacked**, again by simply repeating the syntax
  - ▶ :filter1:filter2:filter3...
- ▶ For example, if I wanted to select to the last div on the page that has something in it, but does not have a class of “mine”:

```
$("div:parent:not(.mine):last")
```



## ▶ Selecting with Filters

- ▶ Filters can also be **stacked**, again by simply repeating the syntax
  - ▶ :filter1:filter2:filter3...
- ▶ For example, if I wanted to select to the last div on the page that has something in it, but does not have a class of “mine”:

```
$("div:parent:not(.mine):last")
```

```
$("div:not(.mine):last:parent")
```

# the common filters

Filter	Filter	Filter
:hidden	:has(s)	:eq(n)
:visible	:not(s)	:first
:animated	:parent	:last
	:empty	[attribute='value']

# jQuery traversing

## ▶ Modifying Selector Sets

- ▶ Once we've used the `$()` factory to target a ***DOM set***, we can use **traversing methods** to **modify the set**.
- ▶ It's important to note, once one of these methods is used, **further methods are being applied to the *new match set*, NOT the original**.

```
$("div").find("p").addClass("nav");
```

# Traversing Methods

`target.find(selector)`

This method adds a new descendant to the *target* selector set.

`$(“div p”)`

could be:

`$(“div”).find(“p”)`

`target.children(selector)`

This method selects direct children that match the provided selector inside the *target*.

`$(“div, p”)`

could be:

`$(“div”).add(“p”)`

# Traversing Methods

`target.filter(selector)`

This method refines a list, removing elements that do not match the *selector* expression.

`$(“div.myclass”)`

could be:

`$(“div”).filter(“.myclass”);`

`target.not(selector)`

This method refines a list, removing elements from the list that match the given *selector* expression.

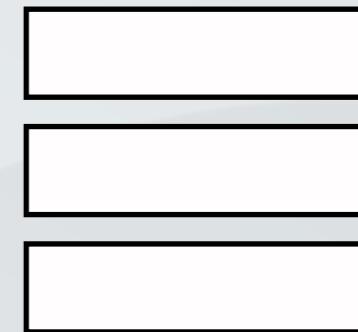
opposite of filter

`$(“div”).not(“.myclass”);`

## ❖ Modifying Selector Sets

- ❖ We can also chain these traversing methods to powerful effect.

```
<td></td>
<td>I'm not blank</td>
<td></td>
```

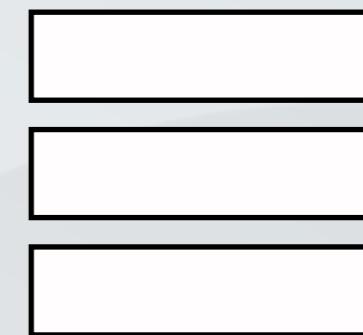
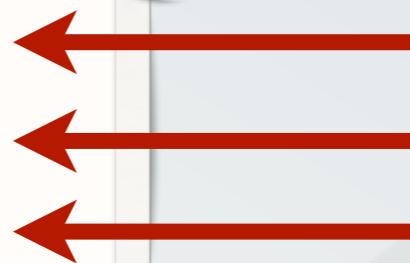


```
$("td")
  .css({'background-color': 'black'}
  .filter(":parent")
    .css({'border': 'red solid 4px'})
;
```

## ❖ Modifying Selector Sets

- ❖ We can also chain these traversing methods to powerful effect.

```
<td></td>  
<td>I'm not blank</td>  
<td></td>
```



```
$("td")  
  .css({ 'background-color': 'black' })  
  .filter(":parent")  
    .css({ 'border': 'red solid 4px' })  
;
```

## ❖ Modifying Selector Sets

- ❖ We can also chain these traversing methods to powerful effect.

```
<td></td>  
<td>I'm not blank</td>  
<td></td>
```



```
$("td")  
  .css({ 'background-color': 'black' })  
  .filter(":parent")  
    .css({ 'border': 'red solid 4px' })  
;
```

## ❖ Modifying Selector Sets

- ❖ We can also chain these traversing methods to powerful effect.

```
<td></td>  
<td>I'm not blank</td>  
<td></td>
```

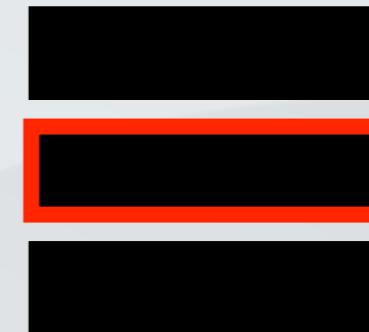


```
$("td")  
  .css({ 'background-color': 'black' })  
  .filter(":parent")  
    .css({ 'border': 'red solid 4px' })  
;
```

## ❖ Modifying Selector Sets

- ❖ We can also chain these traversing methods to powerful effect.

```
<td></td>  
<td>I'm not blank</td>  
<td></td>
```



```
$("td")  
  .css({ 'background-color': 'black' })  
  .filter(":parent")  
    .css({ 'border': 'red solid 4px' })  
;
```

# Traversing Methods

Method	Operation
.contents( )	Returns a set of the children of matched elements ( <i>including text nodes</i> ). <i>Also works on Iframes</i>
.add(s)	Adds all matching selector into the current set. (s = selector)
.siblings(s)	Returns a set of all the siblings of the previous set
.closest(s)	Searches up the Ancestor tree for the selector. Reduces the set to 1.
.parent(s)	Matches the <i>direct</i> parent of the matches, with optional selector
.parents(s)	Matches <i>all</i> ancestors of the matches, with optional selector
.offsetParent( )	Finds the parents of the matches that are not position static
.parentsUntil(s)	Matches <i>all</i> ancestors of the matches, <i>up to but not including</i> the given selector.

# Traversing Methods

Method	Operation
.next(s) .nextAll(s)	Returns siblings ( <i>either the first sibling, or all siblings</i> )
.prev(s) .prevAll(s)	...
.nextUntil(s) .prevUntil(s)	Returns all siblings of matches <i>up to but not including</i> the selector
.slice(start, end)	Returns a set containing a sliced portion. Uses index-basis beginning at 0, using <i>start</i> and <i>end</i> to slice out the return. <i>end</i> is optional.
.eq(i)	Reduces the set to the <i>index</i> ( <i>to 1 item</i> )
.has(s)	Reduces set to those that contain the <i>selector</i> as a descendant
.children(s)	Selects one level deep. If no selectors used it will get all selector items.
.find(s)	This will find all selectors, and will go down to all levels.

## ▶ Lecture Activity

- ▶ jQuery Challenge: selectors
- ▶ download from the FSO references link in the **Lecture Activities** link the file **Goal3-1.zip** file

# jquery content manipulation

## ‣ Get/Set Contents

.html()	Gets all the html contents of the set. (getter)
.html( <i>text</i> )	Replaces the html content of the set with new code. (setter)

- Note that **these methods interpret html** (which includes tags).
- The **get** method will return a string of all the html content (**cannot be chained**). It is the equivalent of the **.innerHTML** property, from Javascript.
- The **set** method returns the original matched set (including any changes), and **can be chained**. *Just like innerHTML, this replaces all of the html inside the element.*

## ❖ Get/Set Contents

.text()

Gets all the text contents of the set (*concatenates all of the text inside any elements and returns a single string as a result*). (**getter**)

.text(text)

Replaces the content of the set with text. (**setter**)

- One of the benefits of the text set-method is that it will automatically encode HTML entities correctly. For example:

```
$(“selector”).text(“Miyagi > Karate Kid”)
```

becomes...

“Miyagi &gt; Karate Kid”

s

## ‣ Creating New Elements

- These next methods we'll discuss **create new elements in the DOM**. The key difference between all of them is **where they add the new content**.
- These are extremely **useful for adding content to a target set**, using parent-child relationships to designate the point of entry.

`target.append(content)`

Appends the contents (can be html, text, or element sets) as the last child of the *target* set.

- ▶ If the *content* is a string of html or text, it is **created** and **then appended** inside.
- ▶ With **append**, the original *target* is the returned object that can be chained on.
  - ▶ *This code would animate the paragraphs, not just the anchors.*

```
$( '#desc' ).append( '<a href="#">link</a>' ).animate();
```

`content.appendTo(selector)`

Moves all the elements specified by `content` and appends them to the elements of the selector

```
$( '<a href="">Link</a>' ).appendTo( '#nav' ).animate();
```

- **appendTo** is similar to the reverse of **append**, with the *target* in the factory.
- The benefit of this is in chaining. Previous, the target of further chains was the element being appended *on*. Now, the targets are the elements *being* appended.
- In the above example, the link is being made into `#nav`, and then animated.

## ▶ Creating Elements

- ▶ Keep in mind, with **appendTo**, the new html is the factory target.. so all chained methods will occur on the newly created markup. Use **append** if you want the jquery target to be the insertion point instead.

```
$('<a href="">Go Home</a>')
  .appendTo("#nav")
  .slideDown()
;
```

```
$('#nav')
  .append("<a href="">Go Home</a>")
  .slideDown()
;
```

## ▶ Creating Elements

- ▶ Similar to the append method are **prepend**, which **adds the content to the beginning of the target elements**.

`target.prepend(content)`

Moves all elements of *content* into the beginning of the *target* set. Returns the *target* for chaining.

`content.prependTo(target)`

Moves all elements of *content* into the beginning of the *target* set. Returns the *content* for chaining.

## ‣ Creating Elements

- Similar again, these **before** and **after** methods **insert the content as siblings of the target, instead of as a child of the target.**

`target.before(content)`

Moves all elements of `content` into positions of siblings of `target`, in front of `target`. Returns **target** for chaining.

`content.insertBefore(target)`

Moves all elements of `content` into positions of siblings of `target`, in front of `target`. Returns **content** for chaining.

`target.after(content)`

Moves all elements of `content` into positions of siblings of `target`, after the `target`. Returns **target** for chaining.

`content.insertAfter(target)`

Moves all elements of `content` into positions of siblings of `target`, after the `target`. Returns **content** for chaining.

## ▶ Wrapping Elements

- ▶ The methods we've looked at so far (*before, after, append, etc..*), these all move or insert content **into** something.
- ▶ These next few methods will instead take a target of content, and **wrap it with some html markup**.
- ▶ For example, if I wanted to target all links on a page and wrap them each individually in a li with a class:

```
$("a").wrap('<li class="anchor"></li>');
```

## ‣ Replacing Elements

- These next 2 methods instead take a *target* and replace all of its contents with the new specified *content*.
- As usual, the key difference between these is *what* is being returned...

`target.replaceWith(content)`

Replaces all of the contents of *target* with *content*. This method returns the *target* for chaining, even though it has been removed from the DOM.

`content.replaceAll(target)`

Replaces all of the contents of *target* with *content*. Returns *content* as the target of further chaining.

```
$( '<p>New Paragraph</p>' ).replaceAll( 'span' ).animate();
```

```
$( 'span' ).replaceWith( '<p></p>' ).appendTo( '#blah' ).animate();
```

## ‣ Wrapping Elements

`target.wrap(wrapper)`

Takes each element in the *target* set and wraps each individually with the html of *wrapper*.

`target.wrapAll(wrapper)`

Takes the entire *target* set as a whole and wraps the entire blocks with the html of *wrapper*.

`target.wrapInner(wrapper)`

Takes each element in the *target* set and wraps the **contents** of each individually with the html of *wrapper*.

- Each of these returns the *target*. The *wrapper* can be any set of complex html elements, but must be validly closed:

```
$(“p”).wrapInner(“<span><strong><em></em></strong></span>”);
```

*parent*

wrap

*sibling*

before

target

after

*child*

prepend

append

## ▶ Removing Elements

<code>target.empty()</code>	Removes all elements inside of the matched set.
<code>target.remove()</code>	Removes all elements of the matched set.
<code>target.detach()</code>	Same as <code>remove</code> , except it saves jQuery data on the element instead of discarding.

- ▶ While **empty** leaves the original set selectable (*can be chained*), **remove** destroys the set and its descendants.

```
$(“p”).empty();  
$(“p”).remove();
```

## ‣ Copying Elements

- The clone method is a unique manipulation method with only 1 argument:

**`target.clone(boolean)`**

Creates a clone of the *target* element(s), including all children and text. If *boolean* is true, it also copies all events throughout. (*default: false*).

- With this method, the returned element is the *clone*, not the original. In addition, the clone is not yet inserted into the DOM, until one of our other methods is used (such as *appendTo*)

```
$("p").clone().appendTo("#mydiv").animate();
```

```
$("a#mine").clone(true).appendTo("#mydiv").animate();
```



## ▶ Lecture Activity

- ▶ jQuery Challenge: Manipulation
- ▶ download from the FSO references link in the **Lecture Activities** link the file **Goal3-2.zip** file



## Goal 3

---

- **Today:** Work on Finalizing your Creative Brief
- **Next Milestone:** Creative Brief
  - see the schedule for the due take of this task.
  - 1 pdf file: **lastname\_firstname\_CB.pdf**
  - *(refer to the Day 1 Requirements pdf for guidance)*