1 Fundamentals

The jQuery library defines a single global function named jQuery(). This function is so frequently used that the library also defines the global symbol $ as a shortcut for it. These are the only two symbols that jQuery defines in the global namespace. This single global function is the central query function in jQuery :

```
var divs = $("div");
```

The value returned by this function represents a set of zero or more DOM elements and is known as a jQuery object. jQuery objects define several methods for operating on the set of elements they represent, as the following example illustrates :

```
$("p.more").css("background-color", "gray").show("fast");
$(".hide").click(function() { $(this).slideUp("slow"); } );
```

To use the jQuery library you should download it and include it in the the usual way :

```
<script src="jqueryxxx.js"> </script>
```

There are several different ways in which you can use the jQuery function :

- You can pass it a CSS selector. When called in this way, it returns the set of elements from the current document that match the selector. If you pass an element or a jQuery object as a second argument to $(), it only returns the matching descendants of the specified element. This optional second argument defines the starting point (or points) for the query and is often called **context**.
- The second way to invoke $() is to pass it an Element, Document, or Window object. Called like this, it simply wraps the element, document or window in a jQuery object and returns that object, allowing you to use the jQuery methods to manipulate the element rather than using raw DOM methods.
- The third way to invoke $() is to pass a string of HTML text. When you do this, jQuery creates the HTML element described by that text and then returns a jQuery object representing those elements. jQuery does not automatically insert the newly created elements into the document. When invoked in this way, $() accepts an optional second argument which specifies the element with which the created elements are to be associated. Or, you can pass a second argument that specifies the names and values of the attributes to set on the newly created elements as an object.
  ```
  var img = $("<img/>", {scr:url, alt:desc});
  ```

– The fourth way to invoke $() is to pass it a function. If you do this, the function you pass will be invoked when the document has been loaded and the DOM is ready to be manipulated.

```
jQuery(function() {
      // The code goes here
});
```

The function you pass to jQuery() will be invoked with the document object as its **this** value and the jQuery function as its single argument. jQuery triggers functions registered through $() when the 'DOMContentLoaded' event is fired, or, in browsers that don't support that event, when the "load" event is fired.

The jQuery library also uses the jQuery() function as its namespace, and defines a number of utility functions and properties under it.

Important to terminolagy to retain :

– The jQuery function : the jQuery function is the value of jQuery or $. This is the function that creates jQuery objects and registers handlers to be invoked when the DOM is ready.
– A jQuery object : an object returned by the jQuery function. A jQuery object represents a set of document elements and can also be called a 'jQuery result', a 'jQuery set', or a 'wrapped set'.
– The selected elements : when you pass a CSS selector to the the jQuery function, it returns a jQuery object that represents the set of document elements matching that selector. Those elements will be called the selected elements.
– A jQuery function : a function that is defined in the namespace of the jQuery function.
– A jQuery method : a jQuery method is a method of a jQuery object returned by the jQuery function.

jQuery objects are array-like, that is, they have a **length** property and numeric properites from 0 to length-1. This means that you can access the contents of the jQuery object using standard square-bracket array notation as follows :

```
$("body")[0]
```

The previous snippet of code is equivalent to : *document.body*.

If you prefer not to use array notation with jQuery objects, you can use the size method instead of the length property, and the get() method instead of indexing with square brackets. If you need to convert a jQuery object to a true array, call the toArray() method.

Finally, all jQuery objects have a property named jquery, and testing for the existence of this property is a simple way to distinguish jQuery objects from other array-like objects. Example :

```
var bodyscripts = $("script", document.body);
```

```
bodyscripts.selector  // => "script"
bodyscripts.context   // => document.body
bodyscripts.jquery    // => "1.4.2"
```

If you want to loop over all elements in a jQuery object, call the each() method instead of writing a for loop. It expects a callback function as its sole argument, and invokes that callback once for each element in the jQuery object (in document order). The callback is invoked as a method of the matched element, so within the callback the this keyword refers to an Element object. If the callback returns false for any element, iteration is terminated after that element. each() returns the jQuery object on which it is called so that it can be used in method chains.

```
$("div").each(function(idx) {
  $(this).prepend(idx + ": ");
  if (this.id === "last")
     return false;
  });
```

The jQuery function map() accepts a callback function as its argument and invokes that function once for each element of the jQuery object.

2 Element Getters and Setters

General considerations with respect to jQuery getters and setters :

- Rather than defining a pari of methods, jQuery uses a single method as both getter and setter. If you pass a new value to the method, it sets the value ; if you don't specify a value, it returns the current value.
- When used as setters, these values on every element in the jQuery object and then return the jQuery object itself to allow method chaining.
- When used as a getter, these methods query only the first element of the set of elements and return a single value.
- When used as setters, these methods often accept object arguments. In this case, each property of the object specifies a name and a value to be set.
- When used as setters, these methods often accept functions as values. In this case, the function is invoked to be set for each element.

2.1 Getting and setting HTML attributes

The attr() method is the jQuery getter/setter for HTML attributes. The following example illustrates its use :

```
// Query the action attr of the 1st form
$("form").attr("action");
```

```
// Set the src attribute of the element whose id is icon
$("#id").attr("src", "icon.gif");

// Set 4 attributes at once
$("#banner").attr({src: "banner.gif",
                   alt: "Advertisement",
                   width:720, height:64});

// Make all links load in new windows
$("a").attr("target", "_blank");

// Load local links locally and off-site links in a new window
$("a").attr("target", function(){
     if(this.host == location.host) return "_self";
     else return "_blank";
});
```

2.2 Getting and setting CSS attributes

In order to query or set CSS styles one has to use the css() function. When querying style values using this function, it return the current style or computed style of the element. That is, the returned value may come from the style attribute or from a stylesheet. When querying values, css() returns numeric values as strings, with the units suffix included. When setting, however, it converts numbers to strings and adds a ”px” (pixels) suffix to them when necessary.

```
// Get the font weight of 1st <h1>
$("h1").css("font-weight");

// Set style on all <h1> tags
$("h1").css("font-variant", "smallcaps");

// You cannot query compound style attributes
$("h1").css("font") => this yields an error

// You can set compound styles, however
$("div.note").css("border", "solid black 2px");

// You can also set multiple styles at once
$("h1").css({ backgroundColor: "black",
              textColor: "white",
              fontVariant: "smallcaps",
              padding: "10px 2px 4px 20px",
              border: "dotted black 4px"
             });

// Increase all <h1> font sizes by 25%
$("h1").css("font-size", function(i, curval){
               return Math.round(1.25*parseInt(curval);
            });
```

2.3 Getting and setting CSS classes

addClass() and removeClass() add and remove classes from the selected elements. toggleClass() adds classes to elements that don't already have them, and removes classes from those that do. hasClass() tests the presence of a specified class. Consider the example :

```
// Add a CSS class to all <h1> tags
$("h1").addClass("hilite");

// Add 2 classes to <p> tags after <h1>
$("h1+p").addClass("hilite firstpara");

// Pass a function to add a computed class
$("section").addClass(function(n){
           return "section"+n;
          });

// Remove a class from all <p> tags
$("p").removeClass("hilite");

// Remove all classes from all <div>s
$("div").removeClass();
```

2.4 Getting and setting HTML Form Values

val() is a method for setting and querying the value attribute of HTML form elements.

```
// Get the value of the surname text field
$("#surname").val();

// Get single value form <select>
$("#usstate").val();

// Get array of values from <select multiple>
```

```
$("select#extras").val();

// Get val of checked radio button
$("input:radio[name=ship]:checked").val();

// Set the value of a textfield
$("#email").val("Invalid email address");
```

2.5 Getting and setting element content

The text() and html() mehtods query and set the plain-text or HTML content of an element. When invoked with no arguments, text() returns the plain-text content of all descendant text nodes of all matched elements. If you invoke the html() method with no arguments, it returns the HTML content of just the first matched element. jQuery uses the innerHTML property to do this : x.html() is effectively the same as x[0].innerHTML. If you pass a string to text() or html(), that string will be used for the plain-text or HTML-formatted text content of the element, and it will replace all existing content.

2.6 Getting and setting element geometry

To query or set the position of an element, use the offset() method. This method measures positions relative to the document, and returns them in the form of an object with left and top properties that hold the X and Y coordinates.

```
var elt = $("#sprite");
var pos = elt.offset();
pos.top += 100;
elt.offset(pos);

$("h1").offset(function(index,curpos){
  return {
      left: curpos.left + 25*index,
      top: curpos.top
  };
});
```

The position() method is like offset() except that is a getter only, and it returns element positions relative to their offset parent, rather than to the document as a whole. In the DOM, every element has an offsetParent property to which its position is relative. Positioned elements always serve as the offset parents for their descendants. The offsetParent() method of a jQuery object maps each element to the nearest positioned ancestor element or to the ¡body¿ element. Note the unfortunate naming mismatch for these methods : offset() returns the absolute position of an element, in document coordinates ; position() returns the offset of an element relative to its offsetParent.

The width() and height() methods return the basic width and height and do not include padding, borders, or margins. innerWidth() and innerHeight() return the width and height of an element plus the width and height of its padding. outerWidth() and outerHeight() return the element's dimensions plus its padding and border. If you pass the value true to either of these methods, they also add in the size of the element's margins.

```
var body = $("body");
var contentWidth = body.width();
var paddingWidth = body.innerWidth();
var borderWidth = body.outerWidth();
var marginWidth = body.outerWidth(true);

var padding = paddingWidth - contentWidth;
var border = borderWidth - paddingWidth;
var margin = marginWidth - borderWidth;
```

As before, if you pass to these methods a number it is taken as a dimension in pixels. If you pass a string value, it is used as the value of the CSS width or heigth attribute and can therefore use any CSS unit.

The final pair of geometry-related jQuery methods are scrollTop() and scrollLeft(), which query the scrollbar positions for an element or set the scrollbar positions for all elements.

```
function page(n) {
  var w = $(window);
  var pagesize = w.height();
  var current = w.scrollTop();
  w.scrollTop(current + n*pagesize);
}
```

3 Altering document structure

The following table summarizes the main methods available for altering document structure :

Operation	\$(target).*method*(content)	\$(content).*method*(target)
Insert content at end of target	append()	appendTo()
Insert content at start of target	prepend()	prependTo()
Insert content after target	after()	insertAfter()
Insert content before target	before()	insertBefore()
Replace target with content	replaceWith()	replaceAll()

Some remarks :

– If you pass a string to one of the methods in column two, it is taken as a string of HTML to insert. If you pass a string to one of the methods in column three, it is taken as a selector.

– The methods in column two return the jQuery objects on which they were invoked while the methods in column three return the new content after its insertion. If the content is inserted in multiple location the returned jQuery object will include one element for each location.

Example of column two methods :

```
// Add content at the end of the #log element
$("#log").append("<br/>" + message);

// Add Z to the beginning of each heading <h1>
$("h1").prepend("Z");

// Insert a rule before and after each <h1>
$("h1").before("<hr/>");
$("h1").after("<hr/>");

// Replace <hr/> tags with <br/> tags
$("hr").replaceWith("<br/>");

// Replace <h2> with <h1> keeping the content
$("h2").each(function(){
         var h2 = $(this);
         h2.replaceWith("<h1>"+h2.html() + "</h1>");
     });

// Add the letter Z to the beggining of each <h1>
$("h1").map(function(){
       return this.firstchild;
     }).before($Z$);
```

Example of column three methods :

```
// Append html to #log
$("<br/>"+ message).appendTo("#log");
```

```
// Append text node to <h1>s
$(document.createTextNode(message)).appendTo("h1");

// Insert rule before and after <h1>s
$(<hr/>).insertBefore("h1");
$(<hr/>).insertAfter("h1");

// Replace <hr/> with <br/>
$(<br/>).replaceAll("hr");
```

If you insert elements that are already part of the document, those elements will simply be moved, not copied to their new location. If you want to copy elements to a new location instead of moving them, you must first make a copy with the clone() method. clone() makes and returns a copy of each selected item. However, by default, clone() does not copy event handlers, pass true if you want to clone that additional data as well.

```
$(document.body).append("<div id="x"><h1>Links</h1></div>");
$("a").clone().appendTo("#x");
```

jQuery defines three wrapping functions : wrap() wraps each of the selected items, wrapInner() wraps the contents of each selected element and wrapAll() wraps the selected elements as a group. These methods are usually passed a newly created wrapper element or a string of HTML used to create a wrapper. Check the example below.

```
// Wrap all <h1> tags with <i> tags
$("h1").wrap(document.createElement("i"));

// Wrap the content of all <h1> tags with <i> tags
$("h1").wrapInner("<i/>");
```

The method empty() removes all children (including text nodes) of each of the selected elements without removing the elements themselves. The remove() method, by contrast, removes the selected elements (and all of their content) from the document. remove() is normally invoked with no arguments, however, if you pass an argument it is treated as a selector. In that case, it only removes the elements that match the selector. The unwrap() method performs element removal in a way that is opposite to the wrap() or wrapAll() methods : it removes the parent of each selected element without affecting the selecting elements or their siblings. That is, for each selected element, it replaces the parent of that element with its children.

4 Event registration

jQuery defines simple event registration methods for each of the commonly used and unversally implemented browser events. These are the simple event handler registration methods jQuery defines :

blur()	**focusin()**	**mousedown()**	**mouseup()**	**change()**	**focusout()**
mouseenter()	**resize()**	**click()**	**keydown()**	**mouseleave()**	**scroll()**
dblclick()	**keypress()**	**mousemove()**	**select()**	**error()**	**keyup()**
mouseout()	**submit()**	**focus()**	**load()**	**mouseover()**	**unload()**

A very simple example :

```
$("p").click(function(){
        $(this).css("background-color", "gray");
   });
```

Recall that you can pass a string of HTML to **$()**to create the elements described by that string and you can also pass as a second argument an object of attributes to be set on the newly created elements. This second argument can be any object that you would pass to **attr()** method. But, if any of the properties have the same name of the event registration methods listed above, the property value is taken as a handler function for the named event type. For example :

```
$("<img/>", {
       src: image_url,
       alt: image_description,
       className: "translucent_image",
       click: function() { $(this).css("opacity", "50%"); }
   });
```

Every event handler is passed a jQuery event object as its first argument. The fields of this object provide details about the event. jQuery does not define a hierarchy of Event object types, for example, there are not separate Event, MouseEvent and KeyEvent types. jQuery copies all of the following fields from the native Event object into every jQuery Event object (though some of them will be undefined for certain event types) :

altkey	attrChange	attrName	bubbles	button	cancelable
charCode	clientX	clientY	ctrlKey	currentTarget	detail
eventPhase	fromElement	keyCode	layerX	layerY	metaKey
newValue	offsetX	offsetY	originalTarget	pageX	pageY
prevValue	relatedNode	relatedTarget	screenX	screenY	shiftKey
srcElement	target	toElement	view	wheelDelta	which

In addition to these properties the Event object also defines the following methods :

preventDefault()	**stopPropagation()**	**stopImmediatePropagation()**
isDefaultPrevented()	**isPropagationStopped()**	**isImmediatePropagationStopped()**

Let us carify the meaning os some of these properties of the Event object :

– **pageX**, **pageY** : document coordinates of the mouse.
– **target**, **currentTarget**, **relatedTarget** : the **target** property corresponds to the document element in which the Event occurred. **currentTarget** is the element in which the current executing event handler was registered (this should be the same as this). The related target is the other element involved in transition events such as mouseover or mouseout.
– **timeStamp** : the time at which the event occurred.
– **which** : speciefies which mouse button or keyboard key was pressed. For mouse buttons : 0 means no buttons are pressed, 1 means the left button is pressed, 2 means the middle button is pressed and 3 means the right button is pressed.

If an event handler returns false, both the default action associated with the event and any future propagation of the event are canceled. That is, returning false is the same as calling the **preventDefaultPropagation()** and **stopPropagation()** methods of the Event object. Also, when an event handler return a value (other than **undefined**), jQuery stores this value in the **result** property of the Event object, where it can be accessed by subsequently invoked event handlers.

Instead of using one of the Event handler registration methods introduced above one can use a more general and complex method : **bind()**, which expects an event type string as its first argument and an event handler as its second one.

```
$("p").bind("click", f);
```

bind() can also be invoked with three arguments. In this case, the first argument must be a string corresponding to the type of the event, the third argument must be the event handler and the second argument can be any value. This value is then stored in the **data** property of every generated event object. You can specify several event types in the first argument of bind (they must be separated by a space) :

```
$("a").bind("mouseenter mouseleave", f);
```

To bind an event handler in a namespace, add a period and the namespace name to the event type string :

```
${"a").bind{"mosueover.myMod", f);
```

The advantage of doing this is that you can distinguished between several different kinds of events. You can also assign a handler to multiple namespaces :

```
$("a").bind("mouseout.myMod.yourMod", f);
```

jQuery has another event handler registration method : **one()**. This method is invoked and works just like **bind()**, except that the event handler you register will automatically deregister itself after it is invoked.

After registering an event, you can deregister it with method **unbind()**. Note that **unbind()** only deresgiters event handlers registered with **bind()** and related jQuery methods. It does not deregister handlers passed to **addEventListenner()** and it does not remove handlers defined by element attributes such as **onclick** and **onmouseover**. With no argunments **unbind()** deregisters all handlers for all elements in the jQuery object :

```
$("*").unbind();
```

With one string argument, all handlers for the name event type (or types, if the string names more than one) are unbound from all elements in the jQuery object :

```
$("a").unbind("mouseover mouseout");
```

When using namespaces to register event handlers, these namespaces can be specified to **unbind()** :

```
$("a").unbind("mouseover.myMod mouseout.myMod");
// Unbind handlers for any event that is in the myMod namespace
$("a").unbind(".myMod");
// Unbind click handlers that are in both namespaces
$("a").unbind("click.ns1.ns2");
```

Another way to invoke the **unbind()** method is to retain a reference to the event handler and to pass it as the second argument :

```
$("#mybutton").unbind("click", myClickHandler);
```

Sometimes it is useful to be able to trigger events manually. The simple way to do this is to invoke one of the event registration methods (like **click()**) with no argument :

```
$("#my_form").submit();
```

Another way to trigger an event manually is to use method **trigger()**. You invoke trigger with an event type string as the first argument and it triggers the handlers registered for events of that type on all elements of in the jQuery object :

```
$("#my_form").trigger("submit");
```

You can specify event namespaces to trigger only handlers defined in that namespace. If you want to trigger only event handlers that have no namespace, append an exclamation mark to the event type.

```
$("button").trigger("click.ns1");
// Trigger button click handlers in no namespace
$("button").trigger("click!");
```

Instead of passing an event type string as the first argument to **trigger()**, you can also pass an Event object (or any object that has a **type** property). The **type** property will be used to determine what kind of handlers to trigger. This is an easy way to pass additional data to event handlers :

```
$("#button1").click(function(e) {
       $("#button2").trigger(e);
});

$("#button1").trigger({type: "click",
                        synthetic: true});
```

There is another way to pass additional data to event handlers when you trigger them manually. The value you pass as the second argument to **trigger()** will become the second argument to each of the event handlers that is triggered.

```
$("#button1").trigger("click", true);
```

Sometimes, you may want to trigger all handlers for a given event type. You could select all elements with $('*') and then call **trigger()**. For efficiency reasons, you can instead call **jQuery.event.trigger()**.

If you want to invoke event handlers without performing the default action, use **triggerHandler()**, which works just like **trigger()** except that it first calls the **preventDefault()** and **cancelBubble()** methods of the Event object.

5 Animated Effects

For every animation, you need to specify a duration of time - in milliseconds or by using a string - for how long the effect should last. The string 'fast' means 200ms. The string 'slow' means 600ms. You can define new duration names by adding new string-to-number mappings to **jQuery.fx.speeds**.

jQuery makes it easy to disable all effects globally : simply by setting **jQuery.fx.off** to **true**.

jQuery's effects are asynchronous. When you call an animation method like **fadeIn()**, it returns right away and the animation is performed in background. jQuery's effect methods usually take as an optional first argument the duration of the effect and as a second argument a function that will be invoked when the effect is complete. The function is not passed any arguments but the **this** value is set to the document element that was animated.

```
$("#message").fadeIn("fast", function(){
      $(this).text("Hello World");
   });
```

If you call an animation mehtod on an element that is already being animated, the new animation does not begin right away but is deferred until the current animation ends.

```
$("#blinker").fadeIn(100).fadeOut(100)
             .fadeIn(100).fadeOut(100)
             .fadeIn(100);
```

6 Ajax

All jQuery Ajax utilities are based on the function **jQuery.ajax()()**. The **load()** method is the simplest of all jQuery Ajax utilities. You must pass it an URL, which it will asynchronously load the content of, and then insert that content in each of the selected elements, replacing any content that is already there.

```
  // Load and displays a status report for every 60s
  setInterval(function() {
      $("#stats").load("status_report.html");
  }, 60000);
```

If the first argument passed to **load()** is a function, then this function will be called when the **onload** event is triggered. Therefore, one can use the **load()** method to register handlers for the **onload** event.

If you only want to display a portion of the loaded document, add a space to the URL and follow it with a jQuery selector.

```
$("#temp").load("weather_report.html #temperature");
```

The **load()** method accepts two optional arguments in addition to the required URL. The first is data to append to the URL or to send along with the request. If you pass a string, it is appended to the URL (after a ? or & as needed). If you pass an object, it is converted to a string of ampersand-separated name=value pairs and sent along with the request. The **load()** method normally makes an HTTP GET request, but if you pass a data object, it makes a POST request instead.

```
$("#temp").load("us_weather.html", "zipcode=02134");


$("#temp").load("us_weather.html",
               {zipcode:02134, units:"F"});
```

Another optional argument to **load()** is a callback function that will be invoked when the Ajax request completes. If it ;s successful, it will be invoked after the URL has been loaded and inserted into the selected elements. If you don't specify any data, you can pass this callback function as the second argument. Otherwise, it should be the third argument.

The other high-level jQuery Ajax utilities are functions, not methods, and they are invoked directly through jQuery not on a jQuery object. **jQuery.getScript()** loads and executes files of JavaScript code. **jQuery.getJSON()** loads a URL, parses it as JSON, and passes the resulting object to the specified callback. Both of these functions call **jQuery.get()**, which is a more general purpose URL-fetching function. Finally, **jQuery.post()** works like **jQuery.get()** but performs an HTTP POST request instead of a GET. Like the **load()** method all of these functions are asynchronous. Below you can find small examples :

```
jQuery.getScript("http://example.com/js/widget.js");

jQuery.getJSON("data.json", function(data) {
           // Now data is the object {x:1, y:2}
    });
```

The methods **jQuery.get()** and **jQuery.post()** expect an URL, an optional data string or object, a callback function and the type of response expected :

– text
– html : works like the previous one
– xml : the value passed to the callback is a Document object representing the XML document instead of a string holding the text.
– script : the script is executed and then the control is passed to the callback.
– json : the value passed to the callback is the object obtained by parsing the URL contents with **jQuery.getJSON()**.
– jsonp

All of jQuery's Ajax utilities end up invoking **jQuery.ajax()** - the most complicated function in the entire library.

```
jQuery.ajax({
   type: "GET",   // The HTTP request method
   url: url,      // the url of the data to fetch
   data: null,    // ...
   dataType: "script", // what to do with the response
   success: callback   // call this function when done
});
```

Finally, to submit an HTML form using the **load()** method :

```
$("#submit_button").click(function(e){
         var f = this.form;
         $(f).load(
             f.action,
             $(f).serialize());
         e.preventDefault();
         this.disabled = "disabled";
});
```

7 Selectors

A simple selector begins with a tag type specification. If you are only interested in $< p >$ tags, your simple selector would be "p". If you are interested all elements regardless of their tag type just use "*". If the selector does not begin either with a tag name or a wildcard, the the later is implicit.

The grammar of the selectors (only a very small part, check the book) :

- #id : matches the element with an **id** attribute of id.
- .class : matches any elements whose **class** attribute includes the word class.

attr : matches any elements that have a **attr** attribute.

attr = val : matches any elements that have an **attr** attribute whose value is **val**.

- and so on

Simple selectors can be combined :

- $A\ \ B$: Selects the document elements that match selector B and that are descendants of elements that match selector A.
- $A > B$: Selects document elements that match selector B and that are direct children of elements that match selector A.
- $A + B$: Selects document elements that match selector B and that immediately follow elements that match selector A (ignoring textnodes and comments).
- $A \sim B$: Selects document elements that match selector B and that are siblings that come after elements that match selector A.

```
"blockquote i"   // matches an <i> within a blockquote
"ol > li"        // matches all <li> that are direct children of an <ol>
"#output + *"    // matches the elements that comes after #output elt
"div.note > h1 + p"  // <p> after <h1> inside a div.note
```

A selector group is simply a comma separated list of selectors :

```
"h1, h2, h3"
```

You cannot put a selector group in parenthesis and treat it as a simple selector in the selector grammar :

```
(h1, h2, h3)+p // this is not valid
h1 + p, h2 + p, h3 + p // this is valid
```