Coding Standards

written by Alec Sherman
of Programming Labs

Overview

Some of these coding standards may seem obvious but others may not. One of the most important factors when multiple programmers are involved is **consistency** from one programmer to another and from one application to another when applicable. Having written guidelines and standards is a great way to make certain everyone is consistent. As a way to explain why standards are critically important consider the following scenario…

After only 3 hours of sleep you are called at 2am in the morning with a frightened client and told you needed to fix some code and it has to be done before 5am so he can prepare his presentation for a 6am big sales presentation (he's in a different time zone than you). This can double his business if it goes well but some developer broke his website late last night.

You have never seen the program or code before. He called you because he is desperate and you have done incredible work for him in the past.

It is a certainty that if the coder(s) that wrote the code did not use good coding standards it would be an impossible task. It still might be an impossible task… but if standards are followed you have a chance.

That's an extreme case but *it actually happened to me*. Even when a programmer is not under extreme pressure, it just makes life so much easier if the code he has to understand and modify is standardized and consistent.

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live." — John Woods

Primary Goals

Readability Text Searching Consistency Reusability

One hour of forethought and planning can prevent literally hundreds of hours of unnecessary upkeep, straining for interpretation, and possible future bugs.

On average programmers spend 10 hours reading code for every 1 hour of actual coding. If Coding Standards are used properly this can be reduced significantly. Imagine having to work 11 hours and knowing 10 of those hours will be reading and deciphering code so you can do one hour of coding and inserting it into the proper locations. If Coding Standards are strictly adhered to it is completely possible to change that so in an 8 hour day you spend only 4 hours reading code and the other 4 hours actually coding. **Work 3 hours less and be 4 times more productive!**

Case Types

readability, text searching

snake_case	words separated by underscore
camelCase	first letter and word lowercase; subsequent words WordCapped
WordCaps / PascalCase	first letter of each word Uppercase
kebab-case	generally kebab-case is only used for CSS styles
_UPPERCASE	often Constants are all UPPERCASE and preceded by an underscore

There is a never-ending debate as to which is better between camelCase versus snake_case. Which you choose is subjective but have your company choose and then stick with it for life.

Note there could even be a valid argument to use one versus the other based on language or object type. For example:

- all CSS variables use snake_case_names
- variables in other languages, pageNames and functionNames use camelCase
- column names in SQL tables use WordCaps

Naming Conventions

Reserved Words

readability, text searching

Do not use any reserved word as a variable or object name - instead find a more descriptive name. Abbreviating “longitude” to be “long” not only makes it more difficult to understand the meaning but “long” is a reserved word in SQL and will fail on some SQL calls unless additional syntax is used (for example back tics for MySQL). Here are examples of reserved or “overly common” words that should never be used as variable/ object names:

type	status	data	back
xml	thing	long	int or any data type
name	item	user	any reserved words

Be Descriptive

readability, text searching

Make the variable and object names long enough to be searchable and descriptive enough so others know what their purpose is. **name** is a horrible variable name. Make it more descriptive so if you need to do a text search you can find it. For example **clientName**, **userName**, or **staffName** are all better. Then when you are searching for **userName** you do not also get all the client name references.

"A long descriptive name is better than a short enigmatic name. A long descriptive name is better than a long descriptive comment." —Robert C. Martin

Variable Names - include scope

readability, text searching

It is good to include the scope within the variable naming convention. This way deep within a large function it is blatantly obvious whether the variable you are looking at is a constant, a global, page or function variable.

Decide what works best based for your programming language and for readability. For example you might want to have all variables start with first letter of scope, then 'v', then underscore, then descriptor. Something like that is super easy to read, you know what scope it has, what it means and can find it on a text search easily.

Scope	Example 1	Example 2	Example 3
function	fv_my_var_name	fnc_my_var_name	fncMyVarName
page	pv_my_var_name	pg_my_var_name	pgMyVarName
global	gv_my_var_name	glo_my_var_name	gloMyVarName

Temporary Variables - the exception

readability

If you are coding a short for-next loop or some code that uses a local variable within a very small code-space that is clearly visible then it is fine to use a single or short variable name. For example these are all obviously used solely in their loops so a descriptive variable name is not necessary. You'll never be searching for **`$i`** variable.

```
for ($i = 0; $i < $numfields; $i++) {
    // do something
}
```

Page and File Naming

readability, text searching

If you are working with web pages, have naming standards for your HTML, CSS and JS file names. If you are compiling DLLs, have naming conventions for your DLL and executable file names. Always keep in mind: readability, consistency and search-ability.

Capitalize SQL Special Words

readability, text searching

Database interaction is a big part of most web applications. If you are writing SQL queries, it is a good idea to keep them readable as well.

Even though SQL special words and function names are case insensitive, it is common practice to capitalize them to distinguish them from your table and column names.

```
SELECT id, username FROM user ORDER BY username ASC;

UPDATE user SET last_login = NOW() WHERE id = 123;
```

Sample Naming Standards

searching-ability

These do not need to be followed exactly, but determine some "standard" and stick to it.

Object	Example	Notes
page name, variables	profilePage	camelCase
REST API call	restGetNotesList	Make camelCase but always start with "rest". Make naming rules like "rest" then "action" then "descriptor". This way all "POST" type calls start with "restPost" then descriptor.
SQL column names	FirstName	makes SQL queries column headers nicer

"You should name a variable using the same care with which you name a first-born child."
— Robert C. Martin

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
— Martin Fowler

Comments - good and bad

readability

If you use good descriptive names for your variables and functions, you should not need comments. Adding comments and then later the code changes but the comments are not updated… is worse than no comments at all. Which is another reason to have good descriptive names for your variables and functions. Try to make your naming so good that comments are not necessary. However sometimes, especially when calling third-party API's, you may need to add comments to explain what is happening.

"Comments are often lies waiting to happen. Code should speak for itself whenever possible."
— Michael Toppa

Developer Consistency

text searching

Often someone will need to do a text search to find code a particular developer has done. That becomes pointless if the developer is not consistent with how they comment their code. Many developers just use their initials but if you do that then make sure they are not likely to be too common for text searches. For example if your initials are "ELS" then searching for that will also find every place "else" is used. Choose a different three-character initial set that is not common in codebases.

Your company should choose a standard and stick with it. For example, all comments include the developer initials plus one space plus the date of the change in the format of MM/DD/YY. Exactly like that - no variance. Of course choose whatever date format you like, but then stick with it religiously.

```
// ABS 01/06/21  You should know whether this is Jan 6th or June 1st
```

That way if all you have for a clue is an email from a client you can do a text search on the email date. Only by having that standard will text searches work reliably.

Note: almost all programming tools have the ability to make macros. Making a "me" macro to auto-populate your initials and today's date is a one-time investment that will pay off hours the first year.

"Any code of your own that you haven't looked at for six or more months might as well have been written by someone else."
— Eagleson's Law

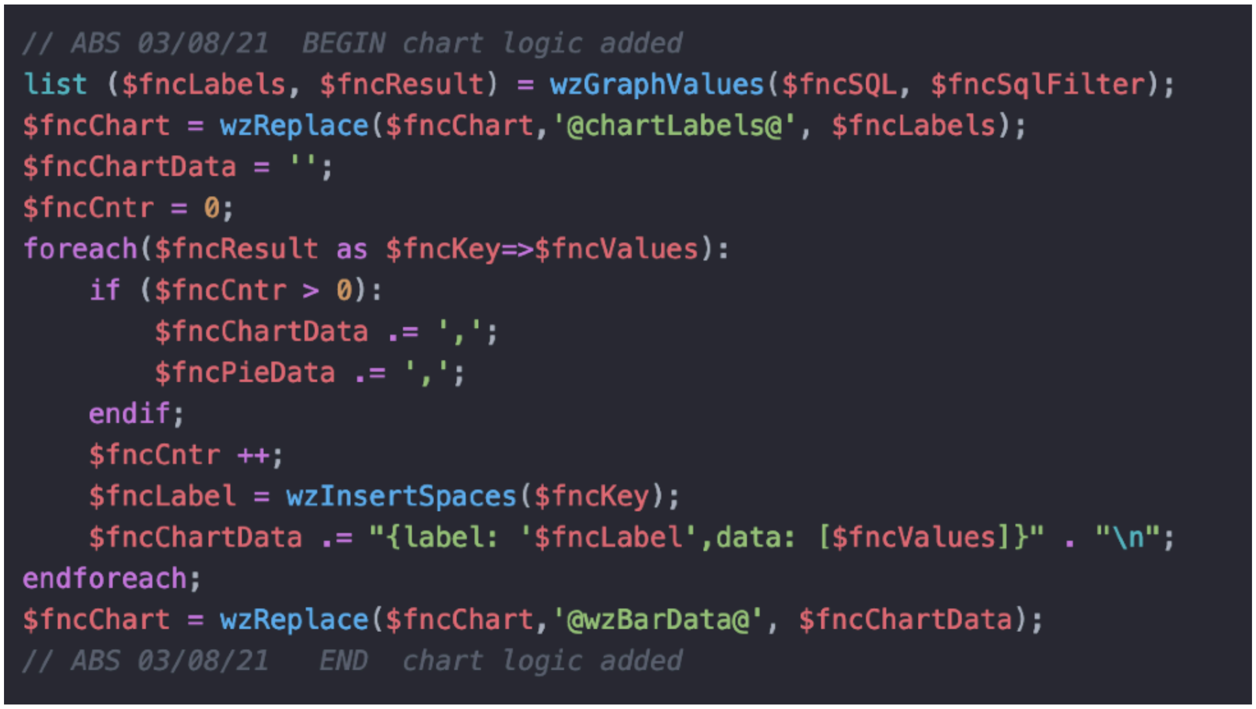
Change Requests

readability

If you are modifying code that is currently already in-use to remove functionality or make other radical changes, you should probably add a comment noting why the change was done. Do this partly because someone at the client location may ask you later “Why on earth was that changed, it used to work perfectly!” This makes it easy to revert or see who requested the change and what is happening with a file.

Comment Blocks

Regarding complicated logic being used, it is a good practice to leave a comment "block" so that another programmer can understand what section of code is involved with a particular piece of logic or functionality. You should be able to also do this with a macro in your code editor.



```
// ABS 03/08/21  BEGIN chart logic added
list ($fncLabels, $fncResult) = wzGraphValues($fncSQL, $fncSqlFilter);
$fncChart = wzReplace($fncChart,'@chartLabels@', $fncLabels);
$fncChartData = '';
$fncCntr = 0;
foreach($fncResult as $fncKey=>$fncValues):
    if ($fncCntr > 0):
        $fncChartData .= ',';
        $fncPieData .= ',';
    endif;
    $fncCntr ++;
    $fncLabel = wzInsertSpaces($fncKey);
    $fncChartData .= "{label: '$fncLabel',data: [$fncValues]}" . "\n";
endforeach;
$fncChart = wzReplace($fncChart,'@wzBarData@', $fncChartData);
// ABS 03/08/21   END   chart logic added
```

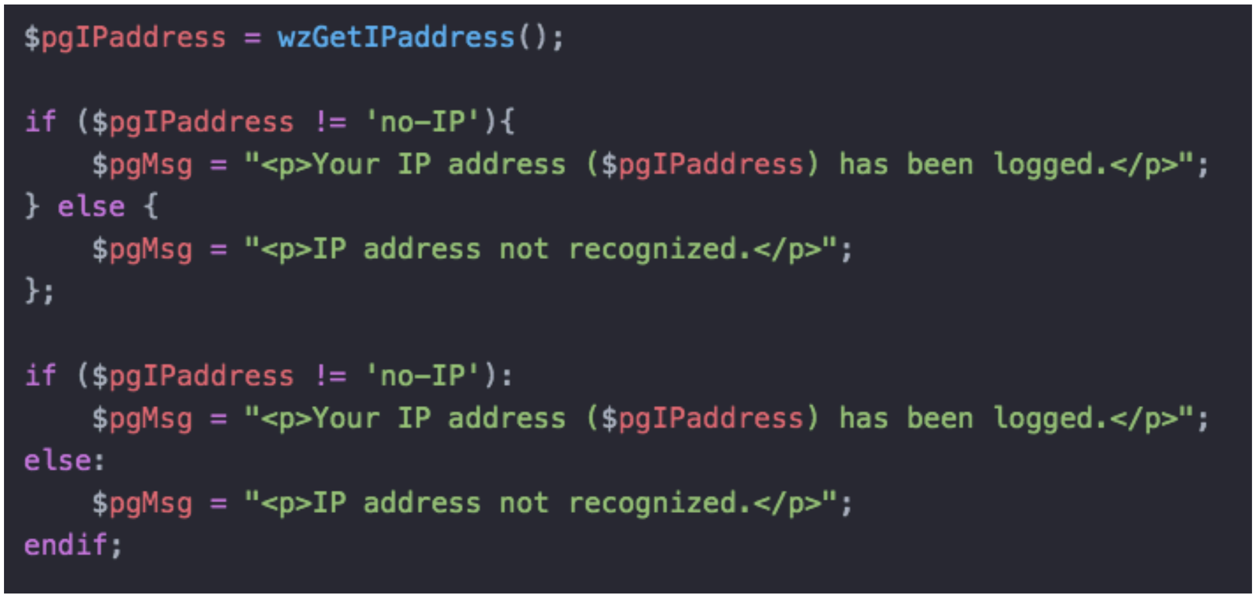
Text Searching

Usually the fastest way to find an issue is to do a case-sensitive text search on some known aspect. Often that is a column name, variable name, developer initials, or change request date. Even a “small” project is likely to grow over time to become hundreds or thousands of pages and hundreds of thousands of lines of code. With good naming conventions and rules regarding date-formatting and comments, anything can be found quickly even years later.

Syntax Standardization **reusability**

Many languages have multiple syntax options to write the same exact code. Have your coding team choose a single syntax style and stick with it. This makes copy/pasting of code from one page to another easier, makes reading the code more standardized, etc.

When there are different options choose the one that is most readable to be the company standard. For example in PHP an if/else statement can be written either of these ways and both work the same:



```
$pgIPaddress = wzGetIPaddress();

if ($pgIPaddress != 'no-IP'){
    $pgMsg = "<p>Your IP address ($pgIPaddress) has been logged.</p>";
} else {
    $pgMsg = "<p>IP address not recognized.</p>";
};

if ($pgIPaddress != 'no-IP'):
    $pgMsg = "<p>Your IP address ($pgIPaddress) has been logged.</p>";
else:
    $pgMsg = "<p>IP address not recognized.</p>";
endif;
```

For a short IF/ELSE statement either are equally easy to read. However if there are lots of lines of code before the end **};** it can be uncertain if you are looking at the end of an **if** statement or the end of something else.

Instead of thinking you'll use **{curly-bracket}** syntax for short IF statements and **endif;** syntax for long IF statements, just standardize on the **endif;**. There are two advantages to this.

1) later more code may be added which makes your "short" IF statement "long"

2) you can copy/paste the **if** line to somewhere else that may need a long IF statement

"It's OK to figure out murder mysteries, but you shouldn't need to figure out code. You should be able to read it." — Steve McConnell

Readability

Alignment and End Of Statements

readability

This really should not need to be stated but based on past code I have seen this apparently does need to be mentioned to some people.

For multi-line statements the first line of a statement should left-align with the last line of that statement. In the case of an `if/else` statement the '`else`' should also left-align with the `'if'` and the last line. If you are using curly bracket notation then the curly brackets should align instead.

Code Indentation

readability

For all languages having good indentation is a necessity. Many script editors now assist with this or can be configured to do so. Whether your company standard is 3 spaces, 4 spaces or based on tabs… make a decision and then make it a standard and stick to it 100% consistently. If you copy/paste code from one page to another then your indentation should not need to be tweaked beyond a “section-at-a-time” indentation.

Spacing

readability

Code should line up visually. This is true for PHP, Javascript, SQL and any other language you may work on.

- Spaces should be used liberally to make sure equal signs line up if there is a series of values being set.
- Spaces should be used around variables and periods to make lines easier to read.
- In general, if a space does not cause a problem it should be inserted so developers can use the Control key with the Arrow keys to navigate. If spaces are skipped then that makes keyboard navigation not work for some text editors… as well as making the code difficult to read.

When looking at your code and trying to decide if it looks like the spacing is correct assume your project manager will be called at 2am after only 3 hours of sleep and has to read your code with blurry eyes. If you cannot see the general structure of your code with blurry eyes then you do not have your spacing correct.

Examples

Horrible SQL table and column names

Because one developer made some bad decisions 5 years before I arrived it makes everyone's job 10 times harder… forever. This is from a project I inherited.

```
CREATE TABLE `xml` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `wo` int(10) NOT NULL,
  `xml` text NOT NULL,
  `type` enum('I','O') DEFAULT NULL,
  PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO_INCREMENT=2385476 DEFAULT
CHARSET=latin1;
```

When a text search needs to be done an *any* of these column names (or the table name), the results are excessive. For example 'xml' was found 3,677 times in 155 files of the 731 PHP files of the website. Search for 'wo' and it was found 15,897 times in 536 files. Search for 'type' and it is found 6,894 times in 443 files. If the developer had chosen more descriptive names the code support and enhancements would be so much easier today.

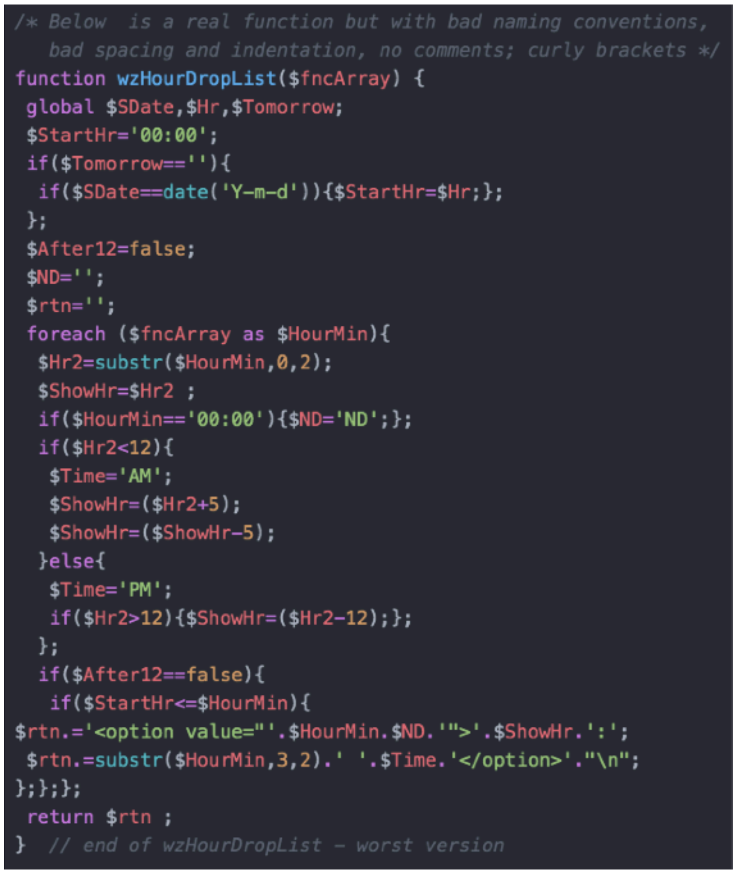
That developer has long since disappeared into obscurity and nobody wants to change the table and column names now then search the 536 of 731 pages to determine which instances need to be corrected. So the inefficiency of that bad decision is stuck with the system for life.

"Of course bad code can be cleaned up. But it's very expensive."
— Robert C. Martin

"There's nothing more permanent than a temporary hack."
— Kyle Simpson

"Your obligation is that of active participation. You should not act as knowledge-absorbing sponges, but as whetstones on which we can all sharpen our wits."
— Edsger W. Dijkstra

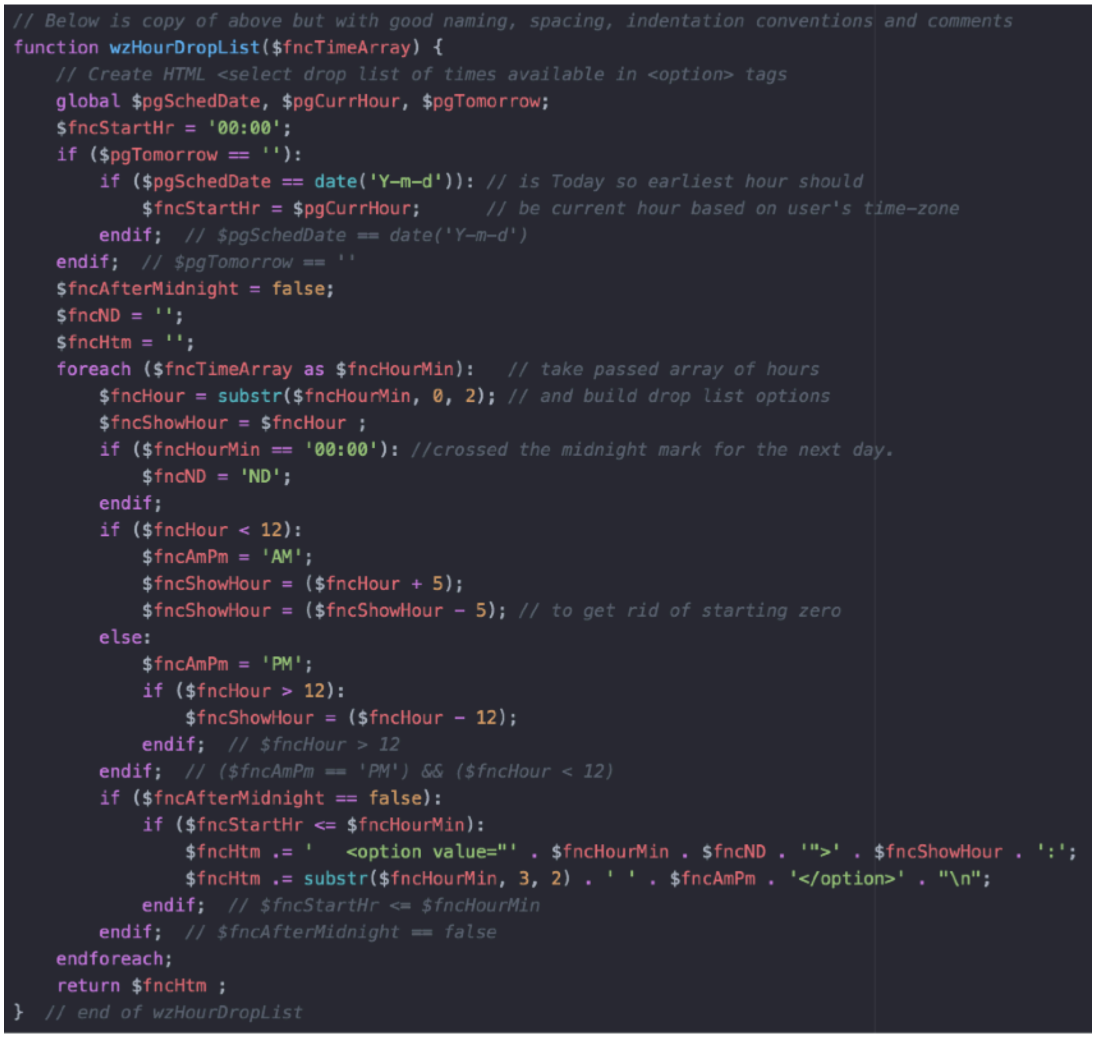
PHP Specific Examples



```php
/* Below  is a real function but with bad naming conventions,
   bad spacing and indentation, no comments; curly brackets */
function wzHourDropList($fncArray) {
 global $SDate,$Hr,$Tomorrow;
 $StartHr='00:00';
 if($Tomorrow==''){
  if($SDate==date('Y-m-d')){$StartHr=$Hr;};
 };
 $After12=false;
 $ND='';
 $rtn='';
 foreach ($fncArray as $HourMin){
  $Hr2=substr($HourMin,0,2);
  $ShowHr=$Hr2 ;
  if($HourMin=='00:00'){$ND='ND';};
  if($Hr2<12){
   $Time='AM';
   $ShowHr=($Hr2+5);
   $ShowHr=($ShowHr-5);
  }else{
   $Time='PM';
   if($Hr2>12){$ShowHr=($Hr2-12);};
  };
  if($After12==false){
   if($StartHr<=$HourMin){
$rtn.='<option value="'.$HourMin.$ND.'">'.$ShowHr.':';
 $rtn.=substr($HourMin,3,2).' '.$Time.'</option>'."\n";
};};};
 return $rtn ;
}  // end of wzHourDropList - worst version
```

Here is an example of terrible spacing, indentation, syntax and naming in PHP but the same concepts apply for all text languages.

If your task was to make a “minor” change to this code, the time it would take you to decipher and interpret would be considerable.



```php
// Below is copy of above but with good naming, spacing, indentation conventions and comments
function wzHourDropList($fncTimeArray) {
    // Create HTML <select drop list of times available in <option> tags
    global $pgSchedDate, $pgCurrHour, $pgTomorrow;
    $fncStartHr = '00:00';
    if ($pgTomorrow == ''):
        if ($pgSchedDate == date('Y-m-d')): // is Today so earliest hour should
            $fncStartHr = $pgCurrHour;      // be current hour based on user's time-zone
        endif;  // $pgSchedDate == date('Y-m-d')
    endif;  // $pgTomorrow == ''
    $fncAfterMidnight = false;
    $fncND = '';
    $fncHtm = '';
    foreach ($fncTimeArray as $fncHourMin):   // take passed array of hours
        $fncHour = substr($fncHourMin, 0, 2); // and build drop list options
        $fncShowHour = $fncHour ;
        if ($fncHourMin == '00:00'): //crossed the midnight mark for the next day.
            $fncND = 'ND';
        endif;
        if ($fncHour < 12):
            $fncAmPm = 'AM';
            $fncShowHour = ($fncHour + 5);
            $fncShowHour = ($fncShowHour - 5); // to get rid of starting zero
        else:
            $fncAmPm = 'PM';
            if ($fncHour > 12):
                $fncShowHour = ($fncHour - 12);
            endif;  // $fncHour > 12
        endif;  // ($fncAmPm == 'PM') && ($fncHour < 12)
        if ($fncAfterMidnight == false):
            if ($fncStartHr <= $fncHourMin):
                $fncHtm .= '   <option value="' . $fncHourMin . $fncND . '">' . $fncShowHour . ':';
                $fncHtm .= substr($fncHourMin, 3, 2) . ' ' . $fncAmPm . '</option>' . "\n";
            endif;  // $fncStartHr <= $fncHourMin
        endif;  // $fncAfterMidnight == false
    endforeach;
    return $fncHtm ;
}  // end of wzHourDropList
```

Here is the same function but with coding standards applied. Notice how much more readable it is!

Even at 2am with 3 hours of sleep this could be understood.

Be Consistent!

In addition to above naming conventions, if one page pulls a value from a REST service and stores it in a local variable, then all other pages that pull that same value should store it in a variable of the same name. Not only is this more intuitive to future programmers viewing the code, but then the code can be copy/pasted from one page to the next.

In your code and data you have flags to determine one way or another. Standardize on that as well. For example for "Yes" and "No" if it is referenced as "Y" and "N" in one place then it should be declared and referenced like that in ***all*** other places as well.

When you are confident that you and others on your team all use the same strict naming conventions and coding standards you save time because you don't have to re-check for every variable name as you are building new pages to make sure you matched a prior page's variable name. Based on the convention and logic the variable names become obvious.

When guidelines are adhered to it makes copying and pasting code much safer thus increase code reuse and decreases bugs.

"You are responsible for the quality of your code. Not your client. Not your boss. […] You don't go to the doctor and say: You know what doc? I'm in a hurry, why won't you skip washing your hands?"
— Michael Toppa

From Here Onward

Define your coding standards that you will use for your own projects. If you work in a company, make sure they define their coding standards. Feel free to copy them from my Programming Labs website and modify them for your specific needs.

Then for all new code you write, use the coding standards. Any time you have to modify or enhance previously written code, refactor it and make it use the new coding standards. Over time more and more of your current code base will become more readable, searchable and better for code reuse.

Read up on Code Refactoring at Wikipedia.

"Code that communicates its purpose is very important. I often refactor just when I'm reading some code. That way, as I gain understanding about the program, I embed that understanding into the code for later so I don't forget what I learned."
— Martin Fowler

Summary

"Weeks of coding can save you hours of planning."

In summary some forethought before starting can make all development go faster.

It makes more code reusable. It makes searching for specific code ***much*** easier and faster. Well named variables and functions makes understanding the code easy instead of laborious and time consuming.

You won't have to memorize as much because rules determine all the basics. For example with rules in place if someone tells you check the code regarding the function variable "client ID", you would automatically know which of these to search for:

client_id	fv_client_id	fnc_client_id	f_client_id
ClientID	fClientID	fClientId	fncClientId
cli_id	cid	c_id	CliID

Choose what works best for your environment. Write down your Coding Standards and post them in your company Wiki or some other public place where all your developers have access. Then make sure everyone on the team consistently uses the chosen standards.

Programming Pioneers

Robert C. Martin	https://en.wikipedia.org/wiki/Robert_C._Martin
Martin Fowler	https://en.wikipedia.org/wiki/Martin_Fowler_%28software_engineer%29

Programming Labs

If you need help with a software development project or system administration support, contact Alec@ProgrammingLabs.com or visit https://ProgrammingLabs.com

To see the Programming Labs Coding Standards go to:
https://ProgrammingLabs.com/standards.php