Coding Best Practices

Programmers employ numerous tactics to ensure readable and organized code. These include 1) using naming conventions for variables; 2) placing whitespaces, indentations and tabs within code; and 3) adding comments throughout to aid in interpretation. In this tutorial we will examine these concepts.

Variable Naming Conventions

Variable naming is an important aspect in making your code readable. Naming variables follow a simple idea: Create variables that describe their function and which follow a consistent theme throughout your code. Let’s take a look at some naming conventions.

Multiword Delimited

This convention is to separate words in a variable name without the use of white space. White space within variables is usually difficult for programming languages to interpret. Because of this variables must be delimited in some way. Here are several delimiting conventions commonly used in code:

Snakecase: Words are delimited by an underscore.
Examples: variable_one, variable_two

Pascalcase: Words are delimited by capital letters.
Examples: VariableOne, VariableTwo

Camelcase: Words are delimited by capital letters, except the initial word.
Examples: variableOne, variableTwo

These conventions are by no means binding, but instead examples of how many programmers format their code. Consistency and readability are key ideas that should be utilized in the naming of variables.

Hungarian Notation

This notation describes the variable type or purpose at the start of the variable name, followed by a descriptor that indicates the variable’s function. The Camelcase notation is used to delimit words. Here are a few examples of Hungarian Notation:

arrDistrubuteGroup	Array called “Distribute Group”
sUserName	Sting called “User Name”
iRandomSeed	Integer called “Random Seed”

Regardless of how you choose to name your variables, always ensure that your naming conventions are consistent throughout the code. Consistency allows others to more easily understand your code.

Function and Class Naming conventions

Much like variable naming conventions, functions and classes should also follow a similar structure of descriptive titles delimited with the conventions described above. An important aspect of naming is to ensure your classes, functions, and variables can be distinguished from each other. For example, one could use Camelcase and Pascalcase for functions and classes respectively, while reserving Snakecase or Hungarian notation for variable names. Distinguishing functions, classes, and variables with different naming conventions can greatly aid other users of your code, and can eliminate the need for large sections of comments that would otherwise be needed.

Whitespace and Tabbing

Whitespace and tabbing are critical for organizing code. Whitespace is essentially any bit of space in your code that is not taken up by physical characters. Tabbing is one way to create whitespace in consistent units using the ‘tab’ key. Many languages ignore whitespace and tabbing all together, so it is important to ensure your code is correctly tabbed and you utilize whitespace to segment your code into neat blocks. Whitespace and tabbing are often used to clarify nested loops and logical statements. Let’s look at some examples of C code that demonstrate the effect of whitespace.

Minimal Whitespace:

```
#include <stdio.h>
int main(int argc, char const *argv[]) { int loop_Sum = 0; for(int i =
0; i < 50; i++){ loop_Sum += 1;} printf("%d\n", loop_Sum); return 0; }
```

Liberal use of Whitespace:

```
#include <stdio.h>
```

```
int main(int argc, char** argv){
	int loop_Sum = 0;

	for(int i = 0; i < 50; i++){
		loop_Sum += 1;
	}

	printf("%d\n", loop_Sum);
	return 0;
}
```

It is important to note that like variable naming, whitespace can be utilized in various different styles and approaches. Just remember to use whitespace and tabbing in a consistent and intuitive manner throughout your code.

Comments Comments Comments

Commenting may be the most important way to organize and segment code. Comments are sections of code that the compiler ignores, but humans can read, and thus can be used to label code and divide it into logical segments to aid users. For example, one can label loops, scopes, functions, and other code snippets. Lines with comments in code are preceded by a symbol that tells the compiler to ignore that line when compiling, for example “#”, “!” or “;” (the symbol depends on the programming language). Let’s look at some C++ code that uses no comments vs. comments.

No Comments:

```
#include <stdio.h>
#include <vector>

using namespace std;

int main(int argc, char** argv){

	vector<int> multiples;

	for(int i = 0; i < 50; i++){
		if(i % 5 == 0){
			multiples.push_back(i);
		}
	}

	for(int i; i < multiples.size(); i++){
		printf("%d is a multiple of 5\n", multiples[i]);
	}

	return 0;
}
```

Comments:

```
#include <stdio.h>
#include <vector>

using namespace std;

int main(int argc, char** argv){

	//Declare a vector to store
	vector<int> multiples;

	//Iterate from 0 to 50
	for(int i = 0; i <= 50; i++){

		//If iterator is a multiple of 5 add it to the vector
		if(i % 5 == 0){
			multiples.push_back(i);
		}
	}

	//Print all items that are a multiple of 5
	for(int i; i < multiples.size(); i++){
		printf("%d is a multiple of 5\n", multiples[i]);
	}

	return 0;
}
```

Note that it is entirely possible to “over comment” code. Code should be designed in an efficient, consistent and intuitive manner such that comments enhance user understanding but are not needed to describe the entire code.