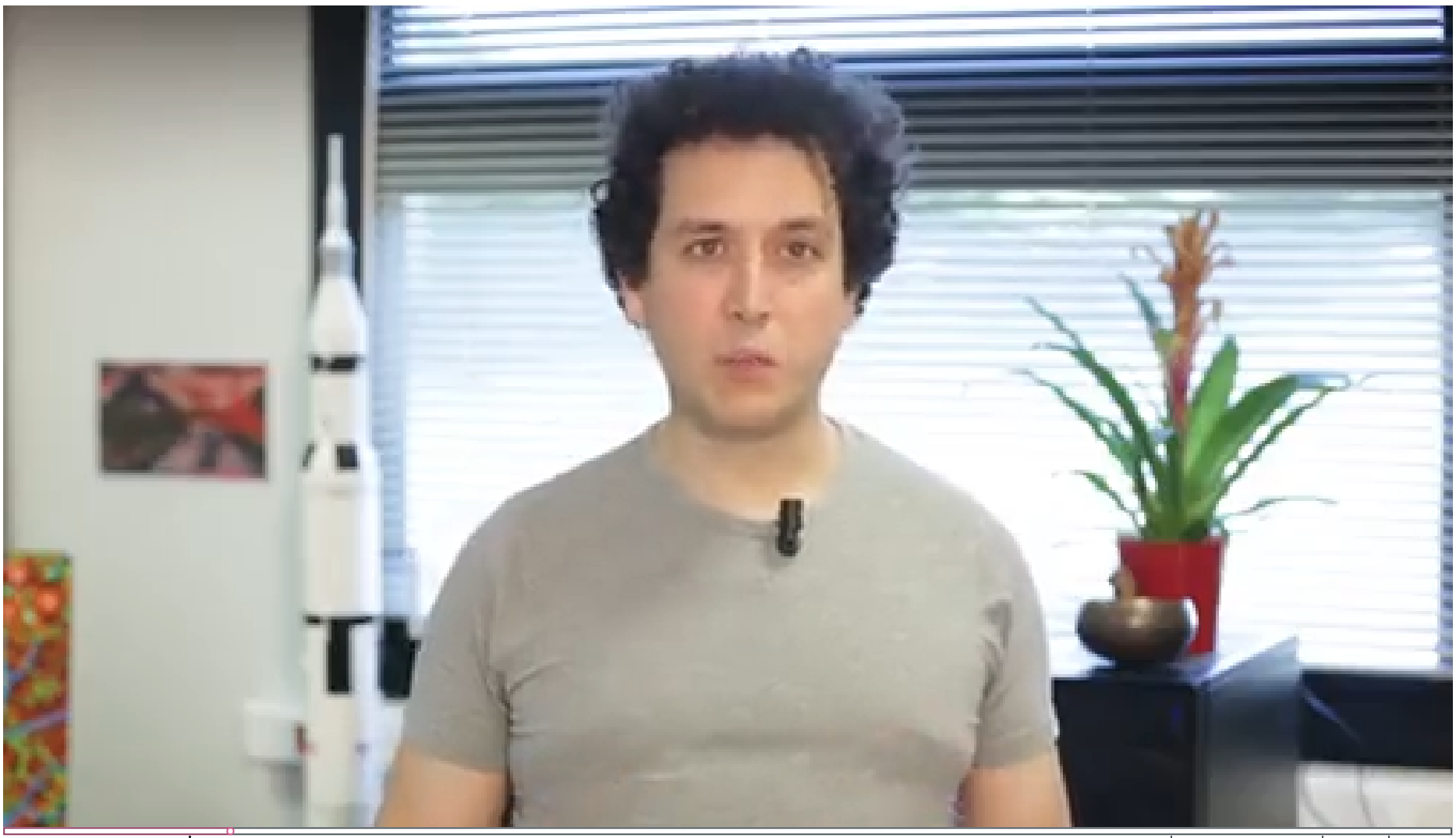
Lesson: Good programming practices

Bookmark this page

By the end of this lesson, you will learn how to adopt good programming practices.

Video - Good Programming Practices



Good programming practices

Hi and welcome to this lesson. Today we're going to look at good programming practices.

Why adopt good programming practices?

Coding is not just about writing lines of codes, one after the other. It's complex and challenging, and requires a certain degree of skill.

I'm going to introduce a few important good practices that you should always keep in mind while coding.

Do not optimize first

First, do not optimize first!

As we'll see in the coming weeks, coding is often a case of finding the best compromise between correctness of the code and speed. The problem is that achieving a good speed boils down to exploiting adapted data structures, using elaborate algorithms, and sometimes even using low-level tricks to accelerate computations.

In many cases, these optimizations have an impact on the readability of the code, which can make it more difficult to see what it does.

You should always have a correct program before optimizing, and there's no need to consider speed at this point. Optimizations can be implemented later.

It's always much simpler to debug a code that was built to be correct rather than a code that was optimized in terms of speed.

Structure the code

It's no secret that coding is difficult. And why's this? Because programming languages are not meant to be easily understood by humans, but to be easily processed by machines.

Writing code means that you can explicitly describe the details of a given method in a way that is unambiguous. In other

Writing code means that you can explicitly describe the details of a given method in a way that is unambiguous. In other words, writing code involves describing a method using elementary notions available in a programming language.

Most of the time, a programmer doesn't think about the low-level programming language itself, but uses abstract language layers.

Let's take the example of finding the shortest path in a graph. There's no such thing as a function *find_the_shortest_path(maze)* in Python. We're going to code this function so that we can use it to perform more complex challenges.

Coding is all about stacking up more and more abstract layers, until your program can be written with the help of just a few functions. In this course, we will use graph theory to play a game in a maze. For this, we're start off by finding paths, then finding the shortest path, then multiple shortest paths, then we'll take the opponent into account.

This process is called structuring the code. And a code should always be structured.

As a rule of thumb, you should always try to code functions that are no more than 15 lines long. If it contains more than that, then you should probably try to split your function into multiple subfunctions.

Factorize the code

A programmer should never -- EVER -- use the copy/paste shortcuts when developing a code.

However, if you SHOULD ever wish to do this, then you should create a subfunction containing the lines you want to duplicate.

This process is very important, because one day you might want to modify your code, update the code to take into account new inputs, or optimize part of the code. It's much simpler when parts of the code to be updated are at one location.

A common source of errors is when duplicated lines are modified but you forget to change other occurrences in your code.

No mysterious constants

Anything that appears to be arbitrary in a code should be avoided. For example, you might want to use the number "infinity" in your code, but such a thing doesn't exist in computer memory. One trick would be to use a large number instead, say 100,000, which should be defined once at the beginning of your code using an explicit name, like *infinity_value*.

When you next look at your code, perhaps years from now, it will be much easier to understand what *infinity_value* is than 100,000.

Explicit inputs/outputs and comments

Finally, you should always write your code thinking that it will be read by someone else. This is obviously the case if you're working in a team, but even if you're not, adding comments is a good way to be sure that what you're doing is correct.

A good way of using comments is to write them first, and then add code sparingly between comments.

If you don't know what comments to write, begin with an explicit description of a function before you write the corresponding code. This should contain a short description of how it works, details about the parameters and outputs, and remarks about correctness conditions.

Coding is not and should never be a question of the number lines of codes written per minute. If it takes one hour to write one line, then so be it. Maybe that single line will be used by hundreds or thousands of programmers, so it can be worth every moment.

Also name your variables and functions so that it is explicit for anyone reading the code. Naming a variable "temp" is often a poor choice.

If you follow all these principles, then you have all the tools to become a great programmer.

‹ Previous

Next ›

edX

Legal

Connect

深圳市恒宇博科技有限公司 粤ICP备17044299号-2