10 useful tips for new programmers

Do you know where the new programmers should start from? Let's TipsMake.com refer to 10 useful tips for new programmers in this article!

1. Top 6 outsourcing trends will change IT industry by 2018
2. 12 extremely useful tricks for JavaScript programmers
3. Top 10 basic network troubleshooting tools that IT people need to know

Do you know where the new programmers should start from? Let's TipsMake.com refer to 10 useful tips for new programmers in this article!

I - the author of the article heard some of the new programmers are managing to know where to start.

You understand the problem, understand the logic, understand the basics of the syntax . When you look at the code of other people you can understand and follow but when you make yourself feel uncertain about transferring your thoughts into code, although understanding syntax or logic.

In this article, I will discuss the process of problem solving steps. Hope to be helpful during your programming.

1. Read the problem at least three times

You can't solve the problem if you don't really understand. There is a difference between the real problem and the problem that you think you are dealing with. It's easy to start reading the first few lines of a problem and putting the assumption to the rest because it's like something you've seen in the past.

If you're creating a popular game like Hangman, make sure you've read through all the rules even if you've played it before. Once, I was asked to create a Hangman-like game and realized that I was "Evil Hangman" because I only read instructions without reading all the rules.

1. According to Wikipedia, the hangman is a game that uses pen and paper to guess words based on the number of characters of the word. This game consists of 2 players with simple means of pen and white paper. The first player will think of a word in the head and give the number of characters of that word in the form of a horizontal row and the second person in turn will guess all the letters they think are present in the word. Each correct guess of the corresponding dash is replaced by a predictable letter, while every wrong guess, the "gallows" will be drawn with a stroke of the hangman. The game ends when a word is correctly guessed or a hanged figure consists of 6 strokes, corresponding to 6 wrong guesses completed.

Sometimes, I try to explain the problem to my friends and see if they understand what I'm talking about and ask them to assess whether it suits my problem. Surely you don't want to find out that you misunderstood the problem when you got halfway through, right? Therefore, spending more time at the beginning to really understand is essential. The more you understand the problem, the easier it will be solved.

Imagine that we are creating a simple `selectEvenNumbers` function that places an array of numbers and returns an array of numbers `selectEvenNumbers` only even numbers. If there is no even number, it will return the empty array `evenNumbers` .

```
function selectEvenNumbers() {
  // your code here
}
```

Here are some questions that come to mind:

1. How can the computer tell if it is an even number? Divide that number by 2 and check if the balance is zero.
2. What would I assign to this function? An array.
3. What will that array contain? One or more numbers.
4. What is the data type of the elements in the array? Number type.
5. What is the purpose of this function? What will I return when ending this function? The goal is to flip through the elements, pick out even numbers and return them in an array. If there is no even number, the function will return an empty array.

See also: With these 5 tips will help you improve the logic programming ability

2. Try at least 3 sample data sets manually when solving problems

Take out a piece of paper and handle the problem manually. Think of at least three sample data sets you can use. Take a look at the **Corner** and **Edge cases** :

1. **Case Corner** : A problem or situation occurring outside the normal operating parameters, especially when multiple environmental variables or conditions occur simultaneously at extremely high levels , although each parameter is within the range specified for that parameter.
2. **Where Edge** : A problem or situation occurs only in an extreme activity parameter (largest or smallest).

Here are some sample data sets that can be used:

```
[1]
[1, 2]
[1, 2, 3, 4, 5, 6]
[-200.25]
[-800.1, 2000, 3.1, -1000.25, 42, 600]
```

When you first start, you will easily skip the steps. Because your brain may already be familiar with even numbers, you can just look at a set of sample data, pick up the numbers `2` , `4` , and `6` and from there, no more careful steps are taken. to solve problems. If this is a challenge, try using very large data sets to train your brain with a step-by-step detailed handling routine. That helps you work through real algorithms.

Let's look at the first array `[1]`

1. Look at the only element in the array `[1]` .
2. Determine if it is an even number. It's not.
3. Notice there's no element in this array.
4. Determine that there are no even numbers in the array provided.
5. Returns an empty array.

Look at the array `[1, 2]`

1. Look at the first element in the array `[1, 2]` .
2. That is `1` .
3. Determine whether it is an even number. It's not.
4. Look at the next element of the array.
5. That is `2` .
6. Determine whether it is an even number. It is an even number.
7. Create an array of `evenNumbers` and add `2` to this array.
8. Notice there's no element in this array.
9. Returns the array of `evenNumbers` with number `2` .

Watch a few more times. Note how I write about `[1]` changes a bit compared to `[1, 2]` . That's why I try with many different sample data sets. I have several sets that have only one element, some sets with decimal numbers instead of integers, some multi-digit sets in one element and some sets of negative numbers.

See also: 16 programming languages ??will change your luck in 2017

3. Simple and optimized steps

Find common patterns and see if anything can be generalized. From there you can reduce the steps taken or the number of repetitions:

1. Create a `selectEvenNumbers` function.
2. Create a new empty array `evenNumbers` where you can store even numbers if available.
3. Browse each element in the array `[1, 2]` .
4. Find the first element.

5. Decide whether it is even or not by divisibility by 2. If it is an even number, add it to `evenNumbers` .
6. Find the next element.
7. Repeat step # 4.
8. Repeat steps # 5 and # 4 until there are no elements in this array.
9. Returns the array `evenNumbers` , whether the array has any element or not.

This approach may remind you of mathematical induction, by you:

1. Let the thing prove true with `n = 1` , `n = 2` , .
2. Suppose it is true for `n = k`
3. Prove it true with `n = k + 1`

4. Write fake code (pseudocode)

Even after completing the general steps, write a fake code to transform the idea to help identify the structure of the code and make writing code much easier. Write fake code by hand one by one. You can do it on paper or through comments in the code editor. You should write on paper to focus better.

The pseudocode doesn't really have a specific rule but sometimes, I still apply the syntax from the programming language because I'm familiar with the language. However, don't pay too much attention to the syntax, but focus on the logic and the code steps above.

For the above example, there are many different ways. For example, you can use filter `filter` but for simplicity and ease of tracking, we use a basic `for` loop (but will use the `filter` after refactoring the code).

Here is an example of fake code written in a long way:

```
function selectEvenNumbers

create an array evenNumbers and set that equal to an empty array

for each element in that array
  see if that element is even
    if element is even (if there is a remainder when divided by 2)
      add to that to the array evenNumbers

return evenNumbers
```

This is an example of a fake code written in a more concise way:

```
function selectEvenNumbers

evenNumbers = []

for i = 0 to i = length of evenNumbers
  if (element % 2 === 0)
    add to that to the array evenNumbers

return evenNumbers
```

Either way, as long as you write each line and understand the logic in each line.

Please review the issue to make sure you are on the right track.

See also: What should beginners learn about computer programming?

5. Convert fake code into code and debug

After the fake code is available, convert each line to real code in the language you are working. I use Javascript for this example.

If you have written it down, retype the code editor and replace each line.

I then call the function and pass the sample data we used earlier. I use them to see if the return is exactly what I want. You can also write test cases to check if the returned result is correct.

```
selectEvenNumbers([1])
selectEvenNumbers([1, 2])
selectEvenNumbers([1, 2, 3, 4, 5, 6])
selectEvenNumbers([-200.25])
selectEvenNumbers([-800.1, 2000, 3.1, -1000.25, 42, 600])
```

I usually use `console.log ()` after every variable or line. This helps me check if the values ??and code work as expected before continuing to the next step. In this way, I will discover problems before being too serious. Here is an example of what I will check before starting. I do this through the code I type out.

```
function selectEvenNumbers(arrayofNumbers) {

let evenNumbers = []
 console.log(evenNumbers) // I remove this after checking output
 console.log(arrayofNumbers) // I remove this after checking output

}
```

After stepping through each of the fake lines, I added `//` first. The text in bold is the real code in JavaScript.

```
// function selectEvenNumbers
function selectEvenNumbers(arrayofNumbers) {

// evenNumbers = []
  let evenNumbers = []

// for i = 0 to i = length of evenNumbers
  for (var i = 0; i < arrayofNumbers.length; i++) {

// if (element % 2 === 0)
    if (arrayofNumbers[i] % 2 === 0) {

// add to that to the array evenNumbers
      evenNumbers.push(arrayofNumbers[i])
    }
  }

// return evenNumbers
  return evenNumbers
}
```

Then remove the fake code to avoid confusion.

```
function selectEvenNumbers(arrayofNumbers) {
  let evenNumbers = []

for (var i = 0; i < arrayofNumbers.length; i++) {
    if (arrayofNumbers[i] % 2 === 0) {
      evenNumbers.push(arrayofNumbers[i])
    }
  }

return evenNumbers
}
```

Sometimes, new developers will get stuck with the syntax to prevent them from moving forward. Remember that the syntax only becomes natural and familiar after working for a long time and it is no shame to re-examine the documents to complete the syntax when the code is actually.

See also: Did you know the 15 hottest programming languages ??on this GitHub?

6. Simple and optimal your code

You may find simplification and optimization as repetitive topics.

In this example, a way to optimize the function is to filter out items from an array by returning an array using the `filter` . In this way, we do not need to define a new variable `evenNumbers` because the `filter` will return a new array with copies of `filter` matching elements. This will not alter the original array. We also do not need to use a `for` loop with this approach. `filter` will browse through each item and return `true` (including that element in the array) or `false` (ignore the element in that array).

```
function selectEvenNumbers(arrayofNumbers) {
  let evenNumbers = arrayofNumbers.filter(n => n % 2 === 0)
  return evenNumbers
}
```

Simple and optimized code may require you to repeat a few times, determine how to simplify and optimize the code further.

Here are some questions to keep in mind:

1. What is your goal of simplification and optimization? The goal will depend on the style of the group or your personal preferences. Are you trying to shorten the code as much as possible? If your goal is to make the code more readable, you may want to use more lines to define variables or calculate something rather than trying to define and calculate all in one line.
2. How can you make the code more readable?
3. Can you take any steps?
4. Are there any variables or functions that you have ended, even without or using?
5. Do you repeat some steps several times? See if you can define it in another function?
6. Is there a better way to handle edge cases?

See more: Top 20 free programming learning websites need to bookmark immediately!

7. Debug

This step really needs to be applied throughout the implementation process. The entire debugging will help you detect any syntax errors or logic flaws earlier. Take advantage of Integrated Development Environment (IDE) and debugger. When I get an error, I look at each line of code to see if anything doesn't work out as expected. Here are some techniques I use:

1. Check the control panel to see the error message. Sometimes, it will show some other lines. This gives me a rough idea of ??where to start - although the error is sometimes not in this line but in another line.
2. Note next to blocks or lines of code and output to quickly see if the code works as expected. From there, it is easy to edit the code when needed.
3. Use other sample data if there are situations that I don't think about and see if the code is working properly.
4. Save as different versions and try each one separately, if you don't want to take a lot of effort every time you have to go back to check the code.

8. Write helpful comments

You may not remember the great function of each code line in a month. And others don't even know. That's why it is important to write helpful reviews to avoid problems and save time later if you need to go back to work.

Stay away from the following comments:

```
// This is an array. Iterate through it.
```

(roughly translated: `// ?ây là m?ng nhá. Duy?t qua nó`.)

```
// This is a variable
```

(roughly translated: `// ?ây là bi?n`)

I try to write a summary of the comments so readers can understand what is happening if it is not really clear. This is especially useful when you're in complex code. It clearly explains what each function does and why. Through the use of variable names, function names and clear comments, you (and others) will be able to understand:

1. What is this code for?
2. What is this code doing?

See also: Forming a way of thinking like a programmer

9. Receive feedback through code reviews

Get feedback from colleagues, experts or other developers. Check back in Stack Overflow. See how others ask about it and learn from them. Sometimes, a problem has many approaches. Find out what they are, you will be better and faster in finding solutions yourself.

10. Practice, practice, practice

Even experienced developers, they always practice and learn every day. If you get useful feedback, integrate it again. Recreate the problem or solve similar problems, push yourself forward. With every problem you solve is a step further for you on the road to programming. Celebrate every success you have and remember how far you've come. Remember that programming, like anything, is becoming easier and more developed.

Thanks, author Gavin Stark.

Refer to some more articles:

1. 10 harsh facts about success in IT
2. 27 things I wish I knew before programming
3. If you want a successful career, find out about the five 2018 technology trends!

Having fun!