Better Programming

Chuncheng Zhang

March 24, 2021

Abstract

Every one has *computer*. It computes at a very high speed. Basically, you put your data in *objects*, and generate *functions* to do the computation. The *languages* are the communication between you and the computer. And the *algorithms* are the tools of how to do the computation precisely and quickly. A good *programming* is about all of the above, you have to manage them, like the arm controls the hand.

Contents

1 Computer

The computer is PC or laptop in real-world. But in programming, the computer can be a very *abstraction* concept. Four components are necessary:

- Input
  Used for user input
- Output
  Used for output to screen or speaker
- Memory
  Where the variables stay during computing
- Disk
  Where the files stay forever

1.1 Input & Output

The input and output are the interfaces between the computer and users. Input means message from user to computer, and output means the reverse. They are applied in different syntax in different language. Take *Python* and *JavaScript* for examples.

Listing 1: InputOutput.py

```python
'''
File: InputOutput.py
Aim: Example of input and output in Python
'''

# Input
inp = input('Input:')

# Output
print(f'You just inputted: {inp}')
```

Listing 2: InputOutput.js

```javascript
/*
File: InputOutput.js
Aim:  Example of input and output in JavaScript
*/

// Input
const inp = prompt("Input:");

// Output
console.log("You just inputted:", inp);
```

1.2 Memory

When you practice programming, all the variables, functions and objects in your code is in the memory. In another word, the computation equals to the operation to the memory. More about memory can be found in the section of Objects. In current stage, all you need to know is everything you program is in the memory.

1.3 Disk

After computing, users may have their stuff to be stored forever. The disk is where to put them.

It should be noticed that it can be very different between the things in memory and their storage in disk. For example, an article is structured as a characters array in the memory. However, it is stored as a highly compressed binary series in the disk. Although the difference between the two formats, they are the same article in fact.

At the viewpoint of the memory, there are fine programs to save the data to and read the data from the disk. In ideal condition, The two-way process is *transparent* to the user, which it frees the users to think about the conversion, thus the users can focus on the object in memory during computation.

2 Objects

The object is an overall calling to the things of interest. It can be a number, character, string, list or set. Moreover, it can even be a collection of them.

When you are thinking about a object, you are actually summarizing its features in the mind. But the computer works in the real world. That is a large separation. *In the abstraction level*, the object is the summary of features. *And in the concrete level*, the object is an instance of features. The gap causes several problems.

2.1 Effect of precision

Basically, a number has two features, the value and the precision. In abstraction level, 100/3 is an existing number. However, in the real world, the computer can not represent it with infinite precision. As a result, every time you put your hand to it in the program, it shows as the given precision, and the output value can be different.

Listing 3: Precision.js

```
/*
File: Precision.js
Aim: Example of a number
*/

// The number of 100 / 3
const a = 100 / 3;

// The int precision
// 33
console.log(parseInt(a));

// The float precision
// 33.333333333333336
console.log(parseFloat(a));
```

Fortunately, the *float* precision is far beyond to meet the standard of daily usage. Evenly, in modern programming language, like JavaScript and Python, the precision can be intelligent assigned to the variables without causing problems in most cases, which is largely convenience to the users.

2.2 Array & Dict

The array and dict are both ordered series in our mind. Let's take an scoring list for example.

Listing 4: Score.js

```
/*
File: Score.js
Aim: Example of the scoring object
*/

// We think the obj is the collection of scores.
// It records the raw scores and knows how to convert them
    into discrete scores.

// The raw scores
let obj = [79, 54, 80, 90];

// The scoring thresholds
obj.thresholds = {
    A: 90,
    B: 80,
    C: 70,
    D: 60,
    E: 0,
};

// Scoring the raw scores
obj.score = (e, i, t) => {
    for (let s in t.thresholds) {
        if (e >= t.thresholds[s]) {
            return [i, s];
        }
    }
};

// See what we got
// [ [ 0, 'C' ], [ 1, 'E' ], [ 2, 'B' ], [ 3, 'A' ] ]
console.log(obj.map(obj.score));
```

The obj is a array of scores, and its threshold member is a dict. As it writes, one can tell the discrete score based on the raw score value using the threshold table. Are they the same in a computer? absolutely NO. The problem is how they are different.

An array is a list of something, like numbers, that linked one-by-one. One can think of it as a chain, the nodes are the numbers. And a dict is somehow like the array, but each element is a pair of key and value. We can think the array is the special version of dict, the array's key is natural numbers, from 0 to its length.

It is almost the simplest data structure in the computer. The advantage is memory saving and easy to access to certain position, and the pitfall is it can be slow for query the certain value.

Listing 5: Array.js

```javascript
/*
File: Array.js
Aim: Example of array.
     Explain why it is good or bad.
*/

// Generate array
// Assume it is very long, like 1,000,000 elements.
let arr = [4, 5, 70, 29, ...., ];

// Example of Good
// Access to the 7,364th element,
// it only requires ONE operation.
console.log("The 7,364th element is", arr[7363]);

// Example of Bad
// Tell if 365 is in the array, and where it first appears,
// it may require 1,000,000 operations to find an answer.
let found = false;
for (let i = 0; i <arr.length; i++) {
    if (arr[i] === 365) {
        console.log("Found 365 in the array at", i)
        found = true;
        return
    }
}
if (!found) {
    console.log("The 365 is not in the array")
}
```

The code describes the thing. On an array, users can easily access to certain position with one operation; However, it can be time consuming to find out if certain value exists. No one wants to wait almost forever to just find out some value using computer, then the question is how to speed up the query?

2.3 Stack & Heap

3 Functions

4 Languages

5 Algorithms