Adult and Higher Education (2025)
Clausius Scientific Press, Canada

DOI: 10.23977/aduhe.2025.070316
ISSN 2523-5826 Vol. 7 Num. 3

Practice of "Multiple Solutions to One Problem" in Python Programming Teaching

Weigang Guo[*]

School of Computer Science and Artificial Intelligence, Foshan University, Foshan, Guangdong, China
awgguo@qq.com
[*]Corresponding author

Keywords: Python Language, Programming, Multiple Solutions to One Problem, Computational Thinking, Teaching Method

Abstract: By Utilizing the flexibility and powerful function of Python language, this paper proposes to use different ideas and perspectives in the classroom and experiment to solve problems and write programs, so as to achieve the goal of "multiple solutions to one problem". This paper gives three examples, including calculating the sum of all integers from 1 to N, finding the Narcissistic number and sorting. Three years' teaching practice shows that the teaching method of "multiple solutions to one problem" in Python programming teaching has the advantages of cultivating divergent thinking, improving problem-solving ability, stimulating innovation potential, enhancing knowledge transfer ability, meeting the requirements of engineering thinking training, improving learning interest, and promoting cooperation and communication, which can cultivate students' computational thinking.

1. Introduction

With the rapid development and in-depth application of artificial intelligence technology, the vast majority of colleges and universities across the country have set up Python programming courses, and a large number of non computer majors have begun to learn Python programming. Python has a simple syntax and a wide range of application scenarios, which meets the needs of interdisciplinary research. Non professionals can also get started quickly. Since 2022, Python programming has been included in the basic courses of artificial intelligence offered by our university among all students. In the process of learning, although students can quickly learn to write some simple programs, they are helpless for slightly complex programs and lack the ability to use computational thinking to solve problems. In order to cultivate students' computational thinking, the author uses the method of "discovering problems - analyzing problems - seeking a variety of solutions - comparing and selecting various solutions - realizing solutions" to explain in teaching, and focuses on the element of "seeking a variety of solutions". The teacher emphasizes "multiple solutions to one problem" in class explanation and programming training to expand students' thinking.

Some domestic teachers have adopted the method of "multiple solutions to one problem" in C language programming, and achieved good results. These teaching methods are mainly designed and practiced from the aspects of selecting structure, cyclic structure, function, application of recursion [1], finding the maximum number, printing Yang Hui triangle [2], finding factorial [3], designing piecewise function [4], finding fibnacci sequence [5], finding accumulation and [6]. In Python programming, there is no published literature on the use of "multiple solutions to one problem".

2. Advantages of Using Python for "Multiple Solutions to One Problem"

In Python programming, "multiple solutions to one problem" means that there are many different solutions or methods for the same problem to achieve the goal. This diversity not only shows the flexibility and powerful functions of Python language, but also provides different ideas and perspectives to solve problems. Python language has obvious advantages in this field.

2.1. Grammatical Flexibility

(1) Multi-paradigm support. The same problem can be solved by different programming paradigms. For example, the Fibonacci sequence problem can be solved in the following three ways: the first is the iterative method (process type), which uses yield generator to realize memory optimization. The second is recursion+cache (functional), which uses lru_cache decorator to avoid repeated calculations. The third is dynamic programming (algorithm optimization), which stores intermediate results through a list.

(2) Built-in Syntax sugar. Syntax sugar, a term invented by British computer scientist Peter J. landin, refers to a grammar added to computer language, which has no impact on the function of the language, but is more convenient for programmers. Generally speaking, the use of Syntax sugar can increase the readability of the program, thus reducing the chance of program code error. For example, string flipping can be realized by stack structure, recursive decomposition and double pointer exchange, and the amount of code is controlled within 10 lines.

2.2. Diversity of Tool Chains

(1) Combination of standard library and third-party library. For example, list de duplication can be quickly implemented with native set() or collections.ordereddict can be used to maintain the order. The prime number detection can be processed in batch by either trial division method or eratostheni sieve method.

(2) Cross domain solutions. For example, in the file processing scenario, the first method can use the basic scheme, using the OS module to traverse the directory. The second method can adopt an efficient scheme, using pathlib object-oriented operation. The third method can use a special scheme, using openpyxl to process Excel files, and so on.

2.3. Development Efficiency Advantage

(1) Quickly validate different algorithms. The dynamic type system allows developers to test different implementations without declaring variable types. For example, the same function can quickly switch the return list or generator.

(2) Community resource support. The active open source ecosystem provides a large number of ready-made cases. For example, the solution of leetcode often contains the comparison of 3+ solutions of Python.

2.4. Performance Optimization Space

(1) Mixed programming capability. Key code segments can be accelerated through Python compilation, which can improve performance while preserving Python multi solution.

(2) Gradual optimization path. From brute force solution (such as recursive Full Permutation) to pruning optimization, Python's concise syntax is easy to improve step by step.

Through the above features, Python can not only meet the needs of beginners' rapid implementation, but also support advanced developers' in-depth optimization, forming a unique problem-solving methodology.

3. Case 1: Calculate the Sum of All Integers From 1 to n

Problem: Writing a program to calculate the sum of all integers added from 1 to n.

(1) Solution 1: Loop structure

```
def sum_of_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
return total
```

Note: the program in this case is given in the form of a function.

(2) Solution 2: Mathematical formulas.This method is especially useful when dealing with large numbers.

```
def sum_of_n(n):
    return n * (n + 1) // 2
```

(3) Solution 3: Built-in functions(sum and range)

```
def sum_of_n(n):
    return sum(range(1, n + 1))
```

(4) Solution 4: Recursion

Although recursive method is not the optimal solution for this problem (because it is inefficient in theory), it can be used as an example of learning recursion.

```
def sum_of_n(n):
    if n == 1:
        return 1
    else:
        return n + sum_of_n(n - 1)
```

Each method has its applicable scenarios, advantages and disadvantages. It is important for beginners to understand loops and conditional statements. Although recursion is not the most efficient method here, it can help understand the complex algorithm structure. The use of built-in functions reflects the simplicity and efficiency of Python. Which method to choose depends on specific needs, performance considerations, and personal preferences. In practice, understanding and mastering a variety of methods can help students better solve problems and improve their programming skills.

4. Case 2: Narcissistic Number

Definition of Narcissus number: in three digits, the sum of each digit cube is equal to the number itself (e.g. $153=1^3+5^3+3^3$). In this example, nine solutions are given.

(1) Solution 1: Mathematical decomposition method

```
for num in range(100, 1000):
    a = num // 100
    b = num // 10 % 10
    c = num % 10
    if a**3 + b**3 + c**3 == num:
        print(num)
```

(2) Solution 2: Basic circulation method

```
for a in range(1, 10):
    for b in range(0,10):
        for c in range(0,10):
            num = a**3+b**3+c**3
            s=a*100+b*10+c
            if s == num:
                print(num)
```

(3) Solution 3: String conversion method 1. It adopts the method of converting integers into strings and forming a list of numbers.

```
for num in range(100, 1000):
    digits = [int(d) for d in str(num)]
    if sum(d**3 for d in digits) == num:
        print(num)
```

(4) Solution 4: String conversion method 2, which converts integers into strings.

```
for num in range(100,1000):
    strnum=str(num)
    s=0
    for d in strnum:
      s=s+int(d)**3
    if s==num:
        print(num)
```

(5) Solution 5: Function encapsulation method.

```
def is_narcissistic(n):
    return sum(int(d)**3 for d in str(n)) == n
for num in range(100, 1000):
    if is_narcissistic(num):
        print(num)
```

(6) Solution 6: Generator expression.

```
def find_narcissistic_numbers():
    for num in range(100, 1000):
        digit_sum = sum(int(d) ** 3 for d in str(num))
        if num == digit_sum:
            yield num
for num in find_narcissistic_numbers():
print(num)
```

(7) Solution 7: Recursive implementation (applicable to learning recursive ideas).

```
def digit_cube_sum(n):
    if n == 0: return 0
    return (n % 10)**3 + digit_cube_sum(n // 10)
for num in range(100, 1000):
    if digit_cube_sum(num) == num:
        print(num)
```

(8) Solution 8: Map function implementation

```
for num in range(100, 1000):
    if sum(map(lambda x: int(x)**3, str(num))) == num:
        print(num)
```

(9) Solution 9: Bit operation implementation (less used)

```
for num in range(100, 1000):
    a = num >> 6 & 0x7
    b = num >> 3 & 0x7
    c = num & 0x7
    if (a**3 + b**3 + c**3) == num:
        print(num)
```

The above nine methods can correctly output all three Narcissus numbers (153, 370, 371, 407), showing different programming skills and python features.

5. Case 3: Sorting

In this case, a total of five solutions are given, including: built-in sorted function, list with sort method, bubble sort, simple selection sort, and quick sort. Each method tests the time-consuming of sorting 100 integers. In addition, merge sort and heap sort are complex and are not shown in this case.

```
import random
import time
#Generate 100 random integers
nums = [random.randint(1, 1000) for _ in range(100)]

#Solution 1: built-in sorted function
start = time.time()
sorted_nums = sorted(nums)
print(f"Built-in sorted function time consuming: {time.time()-start:.6f}seconds")

#Solution 2: Sort method of list
start = time.time()
nums.sort()
print(f" List sort method time consuming: {time.time()-start:.6f} seconds ")
```

```
#Solution 3: Bubble sort implementation
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
start = time.time()
sorted_nums = bubble_sort(nums)
print(f" Bubble sort time consuming: {time.time()-start:.6f} seconds ")

#Solution 4: Simple select sort implementation
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
start = time.time()
sorted_nums = bubble_sort(nums)
print(f" Simple select sort time consuming: {time.time()-start:.6f} seconds ")

#Solution 5: Quick sort implementation
def quick_sort(arr):
    if len(arr) <= 1: return arr
    pivot = arr[len(arr)//2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)
start = time.time()
sorted_nums = quick_sort(nums.copy())
print(f" Quick sort time consuming: {time.time()-start:.6f} seconds ")
```

6. Conclusions

In addition to the above cases, the author has also designed and constructed a large number of cases with multiple solutions to one problem in the chapters of branch structure, circular structure, function, sequence, file, set, dictionary, etc., which have been well applied in teaching. After three years of practice, it can be found that the teaching method of "multiple solutions to one problem " in Python programming teaching has the following effects.

(1) Cultivate divergent thinking

Providing multiple solutions to the same problem can effectively train students' multidimensional thinking ability and avoid thinking stereotypes. This teaching method is especially in line with Piaget's view of "accumulating experience through diversified activities" in his cognitive development theory.

(2) Improve problem solving ability

By comparing different solutions (such as algorithm efficiency, code brevity, etc.), students can learn to choose the optimal solution according to the actual problem. For example, in the teaching of sorting algorithm, the implementation of bubble sort and quick sort can be displayed at the same time.

(3) Stimulate innovation potential

Diversified problem-solving paths encourage students to break through the conventional thinking mode. For example, in the problem of narcissus number, it can be calculated by mathematical methods or realized by string operation.

(4) Enhance the ability of knowledge transfer

By analyzing the correlation between different solutions (such as recursive and non recursive implementation), students can better understand the essence of programming concepts.

(5) Meet the requirements of engineering thinking training

When solving practical engineering problems, we often need to think from multiple perspectives. This teaching method can help students establish a complete ability chain of "from situation to solution".

(6) Enhance interest in learning

Compared with a single solution, diversified implementation methods (such as using built-in functions vs manual implementation) can keep learning fresh and avoid homogeneous learning.

(7) Promote cooperation and exchange

When discussing different solutions in groups, students need to express their ideas and understand the views of others. This interaction can significantly improve their communication skills.

This teaching method is highly consistent with the characteristics of "solving the same problem in multiple ways" of Python language, which can help students establish a more flexible programming thinking paradigm and strengthen the cultivation of computational thinking.

References

[1] Yu Yan. (2010) Applying Multiple Solutions to One Problem to Cultivate Students' Innovative Ability, Computer Education, 13, 17-19.

[2] Hu Ming. (2013) In C Language Teaching, One Problem With Multiple Solutions Is Used to Expand Students' Thinking of Problem Solving, Modern Computer (Professional Edition), 1, 27-29.

[3] Chen Cong. (2020) Case Teaching Method of C Language Programming Course, Fujian Computer, 3, 84-86.

[4] Chen Gang, Wang RiFeng, Li Hui, et al(2017) Teaching Reform and Practice of Computational Thinking Oriented Advanced Language Programming, Journal of Guangxi Normal University of Science and Technology, 4, 22-25.

[5] Zhang Jun. (2013) Multiple Solutions to One Problem in C Language, Computer Programming Skills and Maintenance, 18,135-137.

[6] Yin Xiaoling, Xia Qishou. (2008) Using "Multiple Solutions to One Problem" to Develop Students' Thinking in C Language Teaching, Computer Education, 18,95+92.