How Efficient are Students at Solving Adaptive Parsons Problems vs. Equivalent Write Code Problems?

Conner Morgan and Pari Shah

Motivation

- Maximize students’ abilities to learn and understand programming concepts.

Introduction

- Novice programmers need lots of practice to learn how to program.
- They are often asked to write code from scratch which can be frustrating.
- Adaptive Parsons problems can be more efficient and require less cognitive load than writing the equivalent code.

Adaptive Parsons Problems

These problems require learners to place mixed-up code blocks in the correct order to solve a problem and offer students help after three failed attempts. Students were asked to complete one of two problem sets with five problems each.

Table 1: Type of Problem by Version

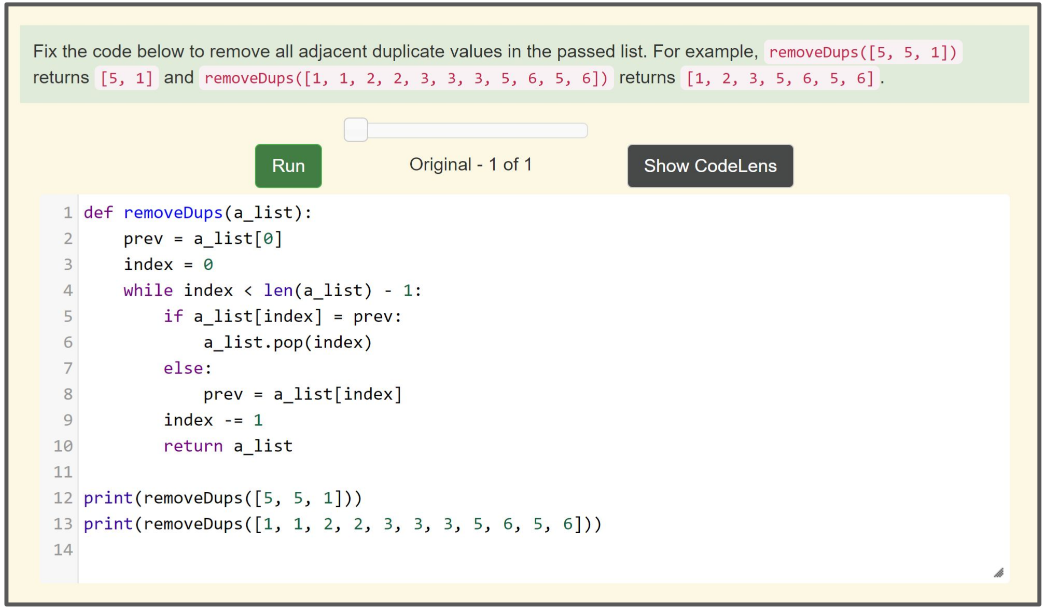
Version	Problem Type
A	Parsons, Write, Parsons, Write, Parsons
B	Write, Parsons, Write, Parsons, Write

Write-Code Parsons Problems

These problems require learners to write their solutions. The code then compiles and learners see what percentage of the unit tests have passed.

Table 1: Type of Problem by Version

Version	Problem Type
A	Parsons, Write, Parsons, Write, Parsons
B	Write, Parsons, Write, Parsons, Write



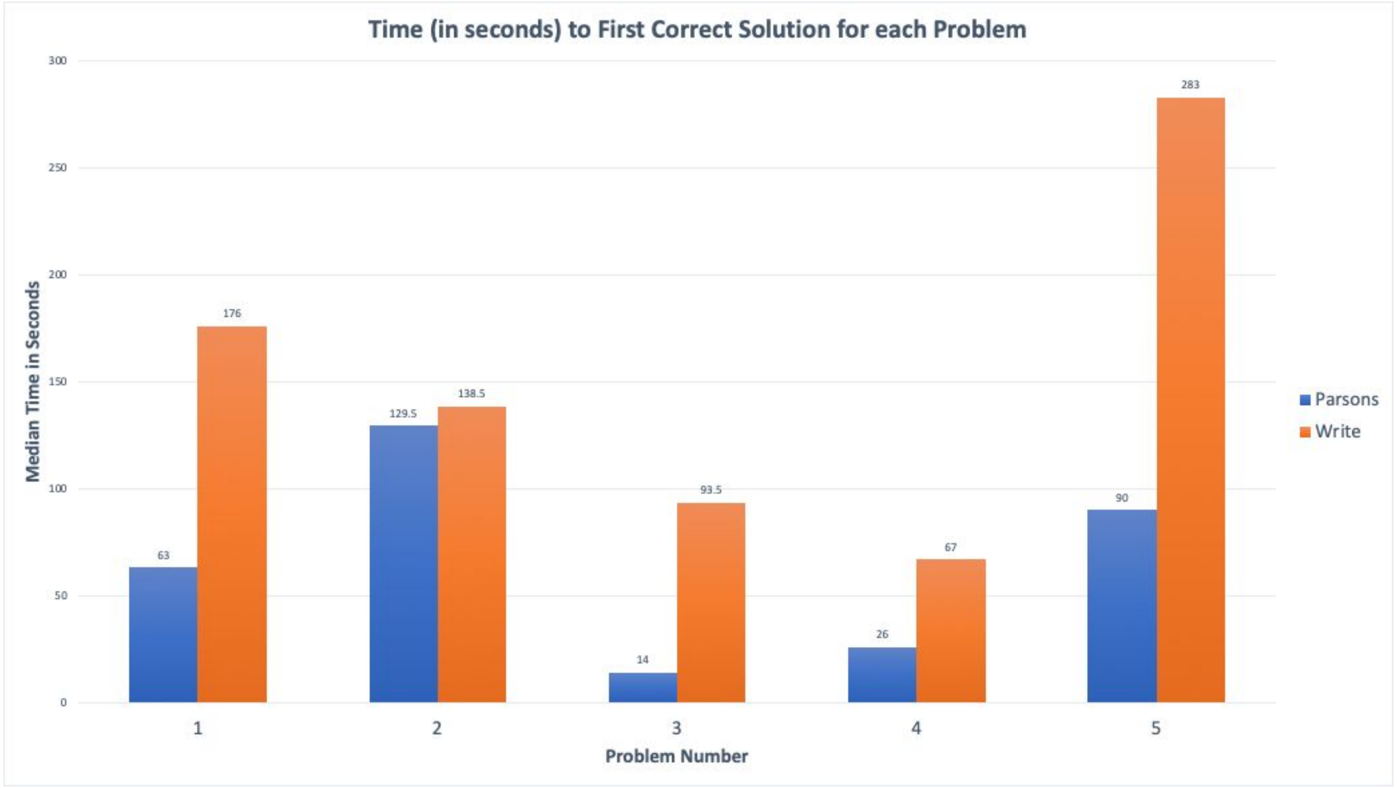
Fix the code below to remove all adjacent duplicate values in the passed list. For example, `removeDups([5, 5, 1])` returns `[5, 1]` and `removeDups([1, 1, 2, 2, 3, 3, 3, 5, 6, 5, 6])` returns `[1, 2, 3, 5, 6, 5, 6]`.

```python
def removeDups(a_list):
    prev = a_list[0]
    index = 0
    while index < len(a_list) - 1:
        if a_list[index] = prev:
            a_list.pop(index)
        else:
            prev = a_list[index]
        index -= 1
        return a_list

print(removeDups([5, 5, 1]))
print(removeDups([1, 1, 2, 2, 3, 3, 3, 5, 6, 5, 6]))

```

Think-Aloud Sessions

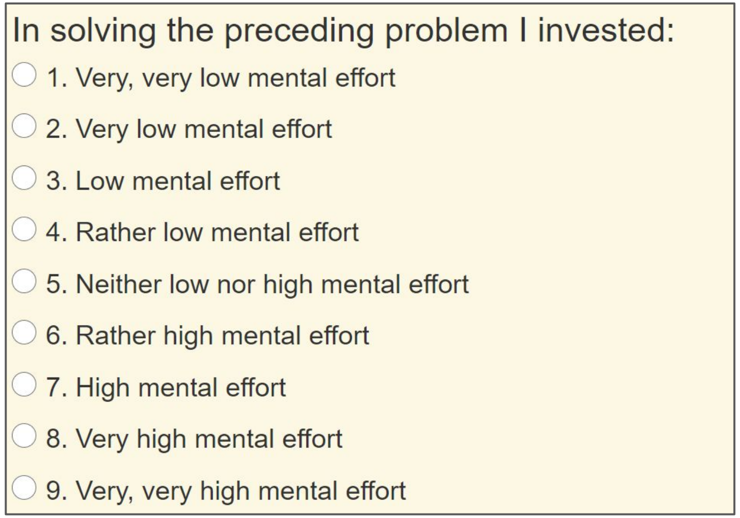
- Students were asked to verbalize their thoughts during the experiment.
- Students were asked several questions after completing the experiment such as:
  - Which do you prefer, solving code writing problems or solving mixed up code problems? Why?
  - Do you find the solution to mixed up code problems useful? Why or why not?

Adaptive Parsons Problem Results



Think-Aloud Session Results

Cognitive Load Rating

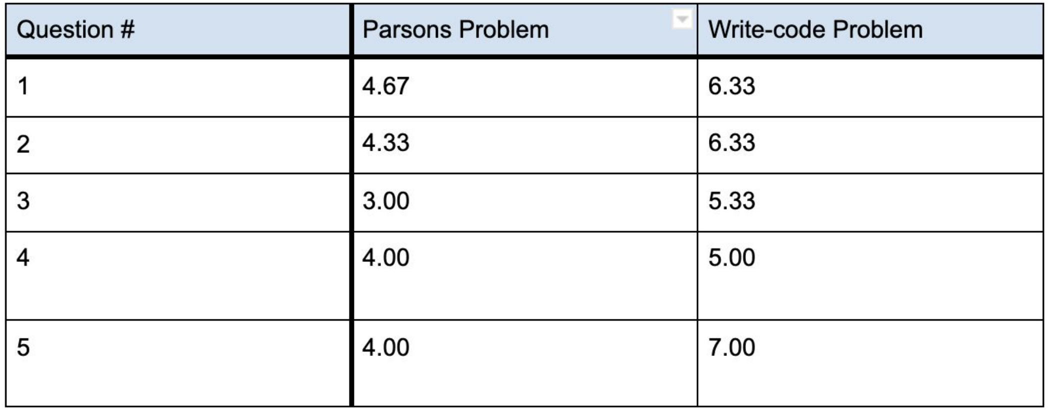


In solving the preceding problem I invested:

1. Very, very low mental effort
2. Very low mental effort
3. Low mental effort
4. Rather low mental effort
5. Neither low nor high mental effort
6. Rather high mental effort
7. High mental effort
8. Very high mental effort
9. Very, very high mental effort

Students were asked to complete this scale to measure their cognitive load.

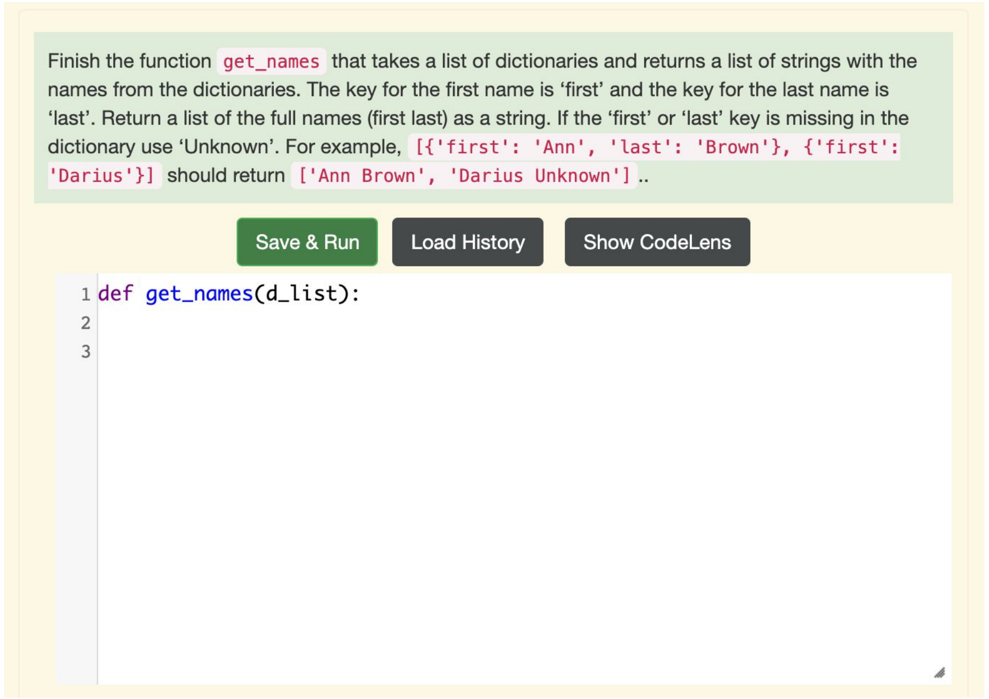
Average Cognitive Load Ratings



Question #	Parsons Problem	Write-code Problem
1	4.67	6.33
2	4.33	6.33
3	3.00	5.33
4	4.00	5.00
5	4.00	7.00

Why did Problem 5 take longer to solve?

Problem 5 asked students to complete the function get_names to return a list of strings from a dictionary and return “Unknown’ if a string was missing.

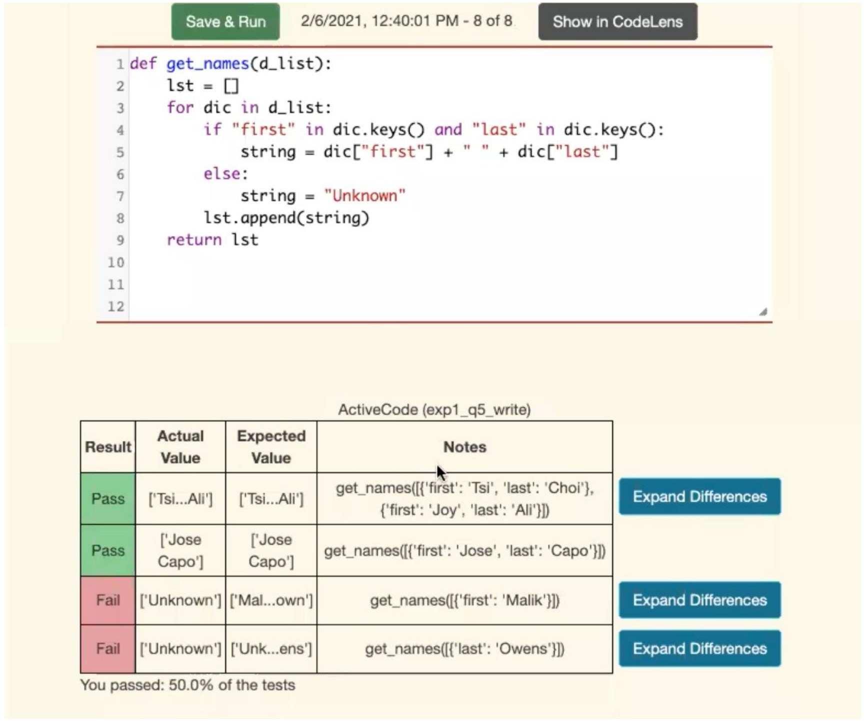


Finish the function `get_names` that takes a list of dictionaries and returns a list of strings with the names from the dictionaries. The key for the first name is ‘first’ and the key for the last name is ‘last’. Return a list of the full names (first last) as a string. If the ‘first’ or ‘last’ key is missing in the dictionary use ‘Unknown’. For example, `[{'first': 'Ann', 'last': 'Brown'}, {'first': 'Darius'}]` should return `['Ann Brown', 'Darius Unknown']`..

```python
def get_names(d_list):


```

Why did Problem 5 take longer to solve?

Subjects 3, 4, and 6 all struggled with problem 5 as a write-code problem. They all reported investing relatively high cognitive load for this task, yielding an average cognitive load rating of 7.0. Students did not understand the concept of Python dictionaries and for loops, which made it difficult to solve the problem.



```python
def get_names(d_list):
    lst = []
    for dic in d_list:
        if "first" in dic.keys() and "last" in dic.keys():
            string = dic["first"] + " " + dic["last"]
        else:
            string = "Unknown"
        lst.append(string)
    return lst
```

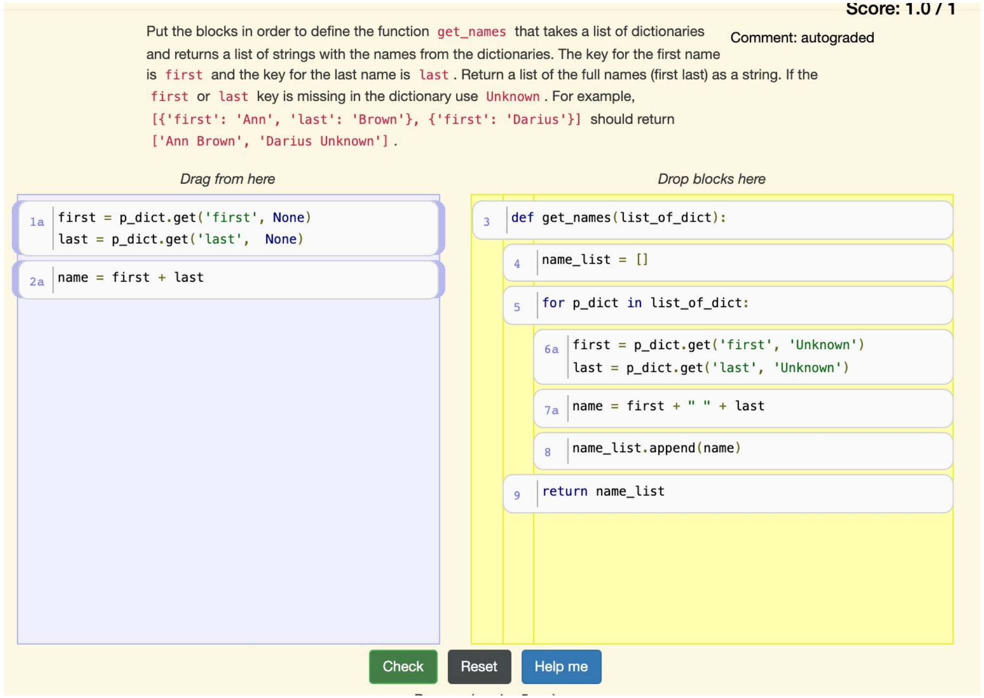
ActiveCode (exp1_q5_write)

Result	Actual Value	Expected Value	Notes
Pass	['Tsi...Ali']	['Tsi...Ali']	get_names([{'first': 'Tsi', 'last': 'Choi'}, {'first': 'Joy', 'last': 'Ali'}])
Pass	['Jose Capo']	['Jose Capo']	get_names([{'first': 'Jose', 'last': 'Capo'}])
Fail	['Unknown']	['Mal...own']	get_names([{'first': 'Malik'}])
Fail	['Unknown']	['Unk...ens']	get_names([{'last': 'Owens'}])

You passed: 50.0% of the tests

Why did Problem 5 take longer to solve?

If students could have switched to the Adaptive Parsons problem for problem 5 they may have been more efficient and invested less cognitive load.



Score: 1.0 / 1

Comment: autograded

Put the blocks in order to define the function `get_names` that takes a list of dictionaries and returns a list of strings with the names from the dictionaries. The key for the first name is `first` and the key for the last name is `last` . Return a list of the full names (first last) as a string. If the `first` or `last` key is missing in the dictionary use `Unknown` . For example, `[{'first': 'Ann', 'last': 'Brown'}, {'first': 'Darius'}]` should return `['Ann Brown', 'Darius Unknown']` .

Drag from here

```
first = p_dict.get('first', None)
last = p_dict.get('last',  None)
```

```
name = first + last
```

Drop blocks here

```
def get_names(list_of_dict):
    name_list = []
    for p_dict in list_of_dict:
        first = p_dict.get('first', 'Unknown')
        last = p_dict.get('last', 'Unknown')
        name = first + " " + last
        name_list.append(name)
    return name_list
```

Conclusion

- Adaptive **Parsons** problems are **usually faster to solve** than writing the equivalent code.
- Adaptive **Parsons** problems **usually require less cognitive load** than writing the equivalent code.
- **Write-code** problems can take **longer to solve** when students are unfamiliar with the concepts being used.

Future Work

- Provide students with the **option to choose** an adaptive parsons problems or equivalent write-code problem.
- Use an **objective measure**, such as eye-tracking, to compliment the self-report cognitive load questionnaires.
- Collect self-efficacy data to explore the effect of solving adaptive Parsons problems on **affective learning gains**.

Reference

Haynes, C. C., & Ericson, B. (n.d.). Problem-Solving Efficiency and Cognitive Load for Adaptive Parsons Problems vs. Writing the Equivalent Code (Rep.).