Task1_AI_RafEngelen

October 6, 2023

1 Report task1: Cryptarithmetic puzzle

1.1 Introduction

In this file you will find some code written in comments. This is code that is only necessary to run on Streamlit. The version on Streamlit will have that code not in comments. I will explain the program step by step, I have created functions that will be called in the same order as you see in this file. These functions will be called in the main function.

Streamlit link: https://rafengelenaitask1.streamlit.app/

1.2 The Program

1.2.1 Imports and input

First we need to import al necessary modules and ask for a puzzle. It is possible to place spaces in the puzzle, for example in between the operator and the word. We remove all spaces.

[100]:
```
from simpleai.search import CspProblem, backtrack
import re
# import streamlit as st

puzzle = input("Give a cryptarithmetic puzzle. ").replace(" ","")
# st.header('Raf Engelen - r0901812 - 3APP01', divider='gray')
# st.title("Task 1 AI: Cryptarithmetic puzzle")
# puzzle = st.text_input(
#     label="Give a cryptarithmetic puzzle.",
#     placeholder="AI + IS = FUN",
#     label_visibility="visible")
```

1.2.2 Validating the input

We have gotten a string, but it might not be in a correct form (… + … = …). We will have check if the string is usable. The input needs has the following validation rules: 1. There is an operator (+, -, * or /) 2. There is an = sign 3. There are 3 words which are split up by the operator and the = sign

Problem: The text from the exception that is raised doesn't show on streamlit.

Solution: I add text to the GUI when an exeption is raised so that it is visible which mistake has been made. (picture af exceptions are found at the end of the report)

[101]:

```
# The puzzle has only one operator
def validate_operator(puzzle):
    pattern = r'^[^+\-*/]*[+\-*/][^+\-*/]*$'
    if re.match(pattern, puzzle):
        return True
    else:
        # st.write(":red[The cryptarithmetic puzzle is not in the correct form.␣
 ↪Use one operator (+, -, *, /) to define your calculation.]")
        raise Exception("The cryptarithmetic puzzle is not in the correct form.␣
 ↪Use one operator (+, -, *, /) to define your calculation.")

# the puzzle has an only one = sign
def validate_equal_sign(puzzle):
    if puzzle.count('=') == 1:
        return True
    else:
        # st.write(':red[The cryptarithmetic puzzle is not in the correct form.␣
 ↪Use one "=" sign.]')
        raise Exception('The cryptarithmetic puzzle is not in the correct form.␣
 ↪Use one "=" sign.')

# The puzzle has 3 words which are split up by the operator and the = sign
def validate_3words(puzzle):
    pattern = r'^[a-zA-Z]+\s*[+\-*/]\s*[a-zA-Z]+\s*=\s*[a-zA-Z]+$'

    modified_string = re.sub(r'\s', '', puzzle)
    if re.match(pattern, modified_string) :
        return True
    else:

        # st.write(':red[The cryptarithmetic puzzle is not in the correct form.␣
 ↪The correct form is: "ai + is = fun", you will need 3 words.]')
        raise Exception('The cryptarithmetic puzzle is not in the correct form.␣
 ↪The correct form is: "ai + is = fun", you will need 3 words.')

# Execute all validation functions
def validate_puzzle(puzzle):
    validate_operator(puzzle)
    validate_equal_sign(puzzle)
    validate_3words(puzzle)
```

1.2.3 Getting the operator and the words

The next step is to find a few things in the given puzzle: 1. The operator (+, -, *, /) 2. We need the 3 words, the first 2 words in the calculation and the result word.

We already know that there is an operator in the puzzle due to the validation, we merely have to check which one.

[102]:

```
def find_operator(puzzle):
     if "+" in puzzle:
          operator = "+"
     elif "-" in puzzle:
          operator = "-"
     elif "*" in puzzle:
          operator = "*"
     else:
           operator = "/"
     return operator
```

For the words we slice the given puzzle. Because we defined the correct form of the puzzle as 'word1 + word2 = result', between the 2 first there is an operator and in between the 2 last words there is an equal sign. Because we know this, we can slice the puzzle by finding the operator and the equal sign. After slicing the puzzle we capitalize the letters.

[103]:

```
def find_words(puzzle, operator):
    word_1 = puzzle[:puzzle.index(operator)].upper()
    word_2 = puzzle[puzzle.index(operator)+1:puzzle.index("=")].upper()
    word_result = puzzle[puzzle.index("=")+1:].upper()
    return [word_1, word_2, word_result]
```

1.2.4 Unique letters and the possible numbers

Next we want to create a tuple of all the unique letters. The reason why we need the letters in a tuple is because simpleai needs it in a tuple. Yet because we want only the unique letters we first create a set.

[104]:

```
def find_letters(words):
    return tuple(set("".join(words)))
```

For simpleai we also need a dictionairy where each letter is the key for an array. In this array there are all possible numbers for the letter. Of course the first letters of the words cannot have 0 as a value.

[105]:

```
def possible_values(letters, words):
    domains = {}
    for letter in letters:
        if letter in [words[0][0], words[1][0], words[2][0]]:

            domains[letter] = list(range(1, 10))
        else:
            domains[letter] = list(range(0, 10))
    return domains
```

1.2.5 Constraints

We need 2 constraints, one constraint to make sure that there are no duplicates. The other calculates the result of the operation and checks this with the given result. Because the constraints will need to be in a certain form, the program will have to calculate somethings that we have already calculated.

In the constraint_unique we merely check if the length of the tuple values is the same as the length of the set of values. This is because a set only has the unique values.

In the constraint_calculation we first need the words. We use functions we have defines previously to find all the words. After that we need a way to create numbers from the words. I define another function for this which checkes which value each letter has and calculates the numerical values of the words. After that we check which operator is used so we can see if the calculation of the first 2 words (now as number) are equal to the third.

[106]:

```
# constraint 1
def constraint_unique(variables, values):
    return len(values) == len(set(values))

# helper function for constraint 2
def word_as_number(word, values, variables):
    number = ""
    for letter in word:
        number += str(values[variables.index(letter)])
    return int(number)

#constraint 2
def constraint_calculation(variables, values):
    operator = find_operator(puzzle)

    words = find_words(puzzle, operator)

    number_1 = word_as_number(words[0], values, variables)

    number_2 = word_as_number(words[1], values, variables)

    result = word_as_number(words[2], values, variables)

    if operator ==  "+":
        return (number_1 + number_2) == result
    elif operator == "-":
        return result == (number_1 - number_2) == result
    elif operator == "*":
        return (number_1 * number_2) == result
    else:
        return (number_1 / number_2) ==result
```

1.2.6 Output

I want the words and the numbers they represent in the output. So I create a function that gets me the numeric value of the words.

```
[107]: def number_result(word : str,solutions : dict):
           number = ""
           for letter in word:
               number += str(solutions[letter])
           return int(number)
```

1.2.7 Solution

All the previous functions will be used in the main fuction. This function will be used when puzzle has a value. This is important in streamlit to not get errors when loading the page.

the functions will be used in the same order as the order as I have defined them. First we validate the puzzle, because we have done this we can find the operator and the words. Next we can find all the unique letters and create the possible values they can take and we define the constraints. Further we can define the problem and let the program backtrack the possible values the letters can take to check if it finds a solution.

Finaly I want to print the results that we have gathered.

We only execute the main function if the puzzle variable is not null, which means that the function will not be executed when the input box is empty. For example when you first load the application on streamlit.

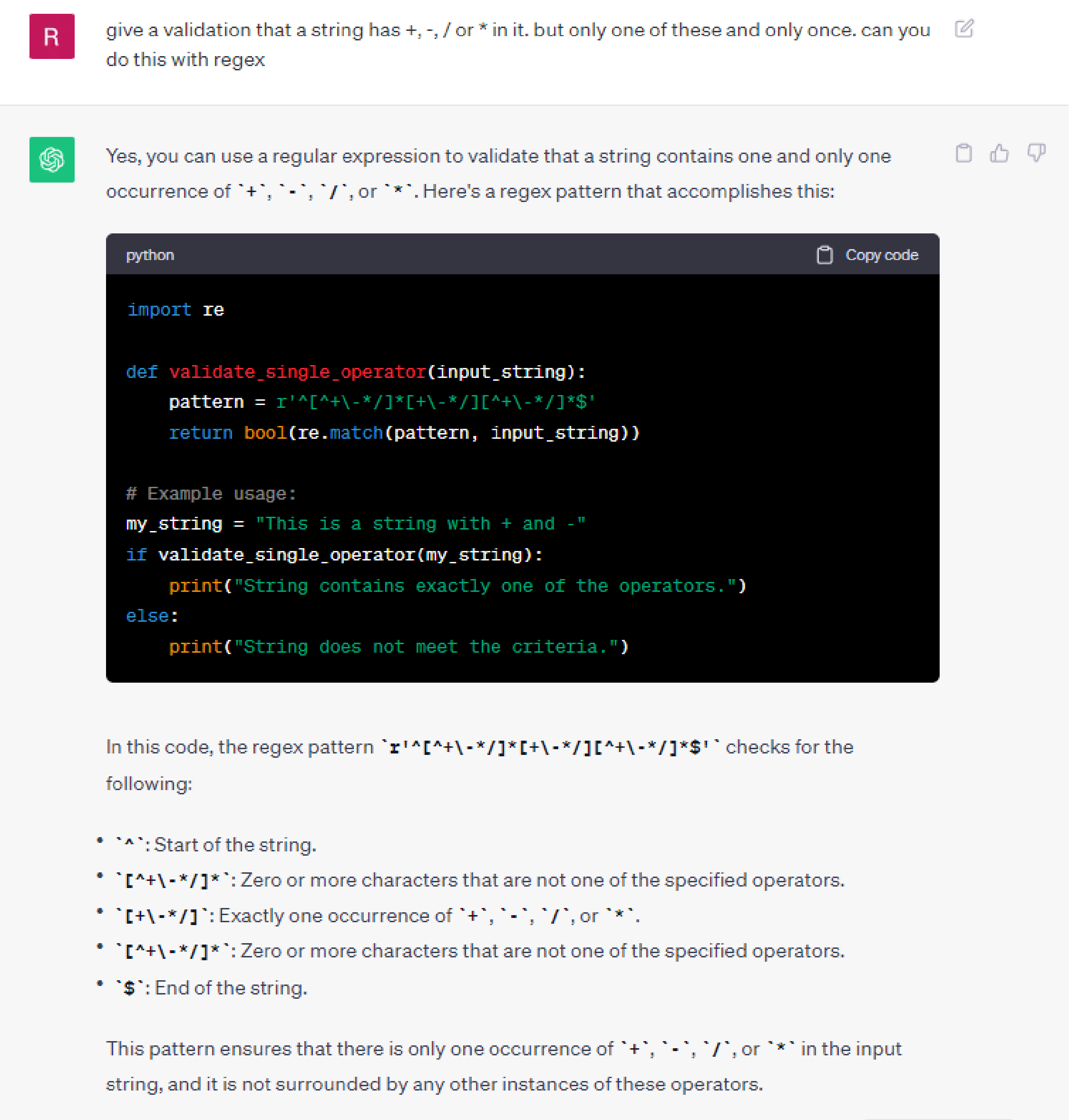
```
[108]: def main(puzzle):
           print(puzzle)
           validate_puzzle(puzzle)

           operator = find_operator(puzzle)
           print(f"Operator: {operator}")

           words = find_words(puzzle, operator)
           print(f"Words: {words}")

           letters = find_letters(words)
           print(f"Letters: {letters}")

           domains = possible_values(letters, words)
           print(f"Possible values for letters: {domains}")

           constraints = [
               (letters, constraint_unique),
               (letters, constraint_calculation),
           ]

           problem = CspProblem(letters, domains, constraints)

           output = backtrack(problem)
           try:
               print(f"\nSolution:",
                     f"{words[0]} {operator} {words[1]} = {words[2]}",
                     f"{number_result(words[0], output)} {operator}␣
       ↪{number_result(words[1], output)} = {number_result(words[2], output)}",
                    ␣
       ↪"___________________________________________________________________\n",␣
       ↪sep="\n")

               # st.write(f"{words[0]} {operator} {words[1]} = {words[2]}")
               # st.write(f"{number_result(words[0], output)} {operator}␣
       ↪{number_result(words[1], output)} = {number_result(words[2], output)}")
           except:
               print("No solutions found\n"
                    ␣
       ↪"___________________________________________________________________\n")
               # st.write("No solutions found")



       if puzzle:
           main(puzzle)
```

```
TO+GO=OUT
Operator: +
Words: ['TO', 'GO', 'OUT']
Letters: ('U', 'T', 'G', 'O')
Possible values for letters: {'U': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'T': [1, 2,
3, 4, 5, 6, 7, 8, 9], 'G': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'O': [1, 2, 3, 4, 5, 6,
7, 8, 9]}

Solution:
TO + GO = OUT
21 + 81 = 102
___________________________________________________________________
```

1.3 GenAI tools

In this task I have utilised chatGPT once. This was to generate regex for the “validate_operator” and the “validate_3words” functions.

To validate that there is an operator and only one operator in the puzzle, I have given ChatGPT the following prompt:



R

give a validation that a string has +, -, / or * in it. but only one of these and only once. can you do this with regex

Yes, you can use a regular expression to validate that a string contains one and only one occurrence of `+`, `-`, `/`, or `*`. Here's a regex pattern that accomplishes this:

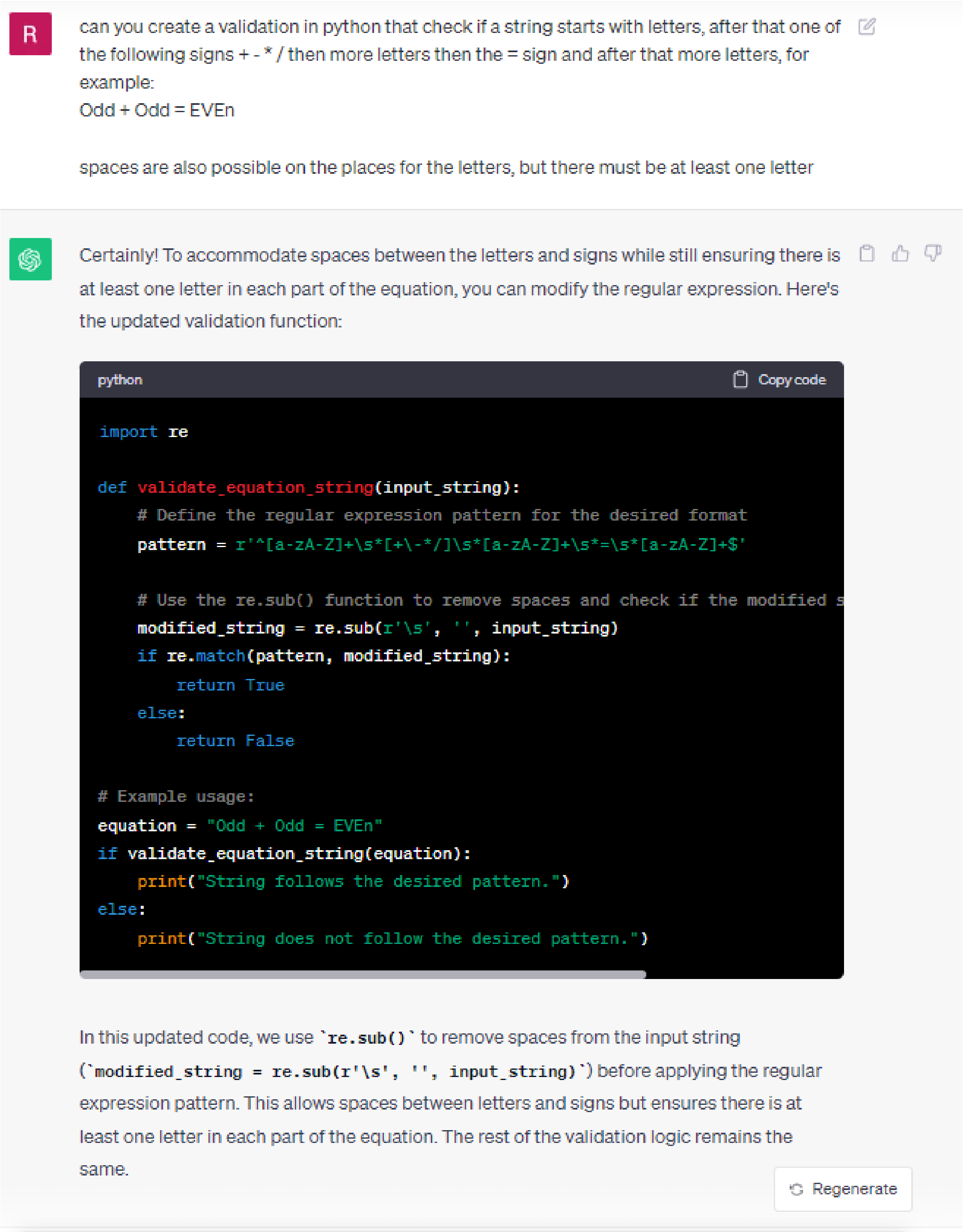
```python
import re

def validate_single_operator(input_string):
    pattern = r'^[^+\-*/]*[+\-*/][^+\-*/]*$'
    return bool(re.match(pattern, input_string))

# Example usage:
my_string = "This is a string with + and -"
if validate_single_operator(my_string):
    print("String contains exactly one of the operators.")
else:
    print("String does not meet the criteria.")
```

In this code, the regex pattern `r'^[^+\-*/]*[+\-*/][^+\-*/]*$'` checks for the following:

- `^`: Start of the string.
- `[^+\-*/]*`: Zero or more characters that are not one of the specified operators.
- `[+\-*/]`: Exactly one occurrence of `+`, `-`, `/`, or `*`.
- `[^+\-*/]*`: Zero or more characters that are not one of the specified operators.
- `$`: End of the string.

This pattern ensures that there is only one occurrence of `+`, `-`, `/`, or `*` in the input string, and it is not surrounded by any other instances of these operators.

To validate that the puzzle contains 3 'words' which are split up by an operator and a '=' sign. Note that the validation ignores spaces, this means that it is allowed that there are spaces between a word and the operator or '=' sign, but the spaces are not necessary. I have used chatGPT to generate the following response:



can you create a validation in python that check if a string starts with letters, after that one of the following signs + - * / then more letters then the = sign and after that more letters, for example:
Odd + Odd = EVEn

spaces are also possible on the places for the letters, but there must be at least one letter

Certainly! To accommodate spaces between the letters and signs while still ensuring there is at least one letter in each part of the equation, you can modify the regular expression. Here's the updated validation function:

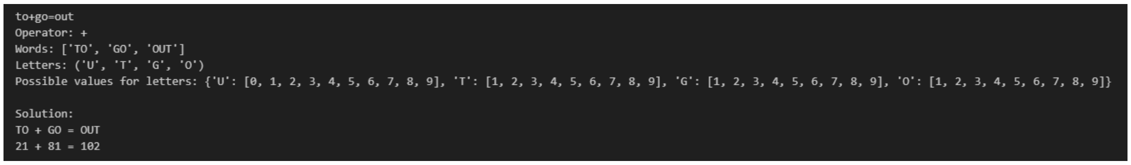
```python
import re

def validate_equation_string(input_string):
    # Define the regular expression pattern for the desired format
    pattern = r'^[a-zA-Z]+\s*[+\-*/]\s*[a-zA-Z]+\s*=\s*[a-zA-Z]+$'

    # Use the re.sub() function to remove spaces and check if the modified s
    modified_string = re.sub(r'\s', '', input_string)
    if re.match(pattern, modified_string):
        return True
    else:
        return False

# Example usage:
equation = "Odd + Odd = EVEn"
if validate_equation_string(equation):
    print("String follows the desired pattern.")
else:
    print("String does not follow the desired pattern.")
```

In this updated code, we use `re.sub()` to remove spaces from the input string (`modified_string = re.sub(r'\s', '', input_string)`) before applying the regular expression pattern. This allows spaces between letters and signs but ensures there is at least one letter in each part of the equation. The rest of the validation logic remains the same.

1.4 Examples of outputs

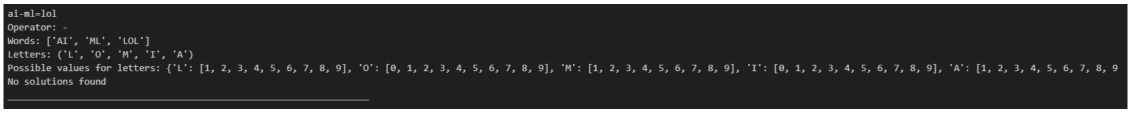
I will show some examples of cryptarithmetic puzzles with the result the program gives.

EXAMPLE 1:



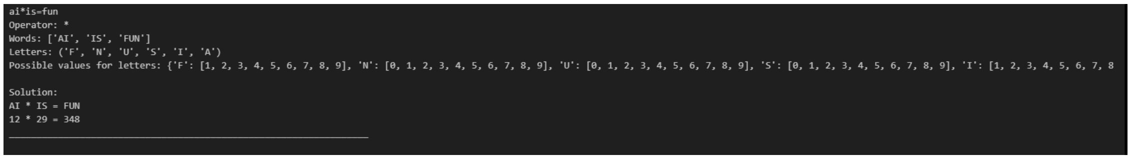
```
to+go=out
Operator: +
Words: ['TO', 'GO', 'OUT']
Letters: ('U', 'T', 'G', 'O')
Possible values for letters: {'U': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'T': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'G': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'O': [1, 2, 3, 4, 5, 6, 7, 8, 9]}

Solution:
TO + GO = OUT
21 + 81 = 102
```

EXAMPLE 2:



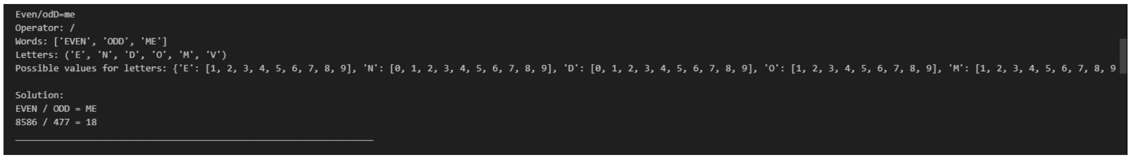
```
ai-ml=lol
Operator: -
Words: ['AI', 'ML', 'LOL']
Letters: ('L', 'O', 'M', 'I', 'A')
Possible values for letters: {'L': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'O': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'M': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'I': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'A': [1, 2, 3, 4, 5, 6, 7, 8, 9
No solutions found
__________________________________________________________________
```

EXAMPLE 3:



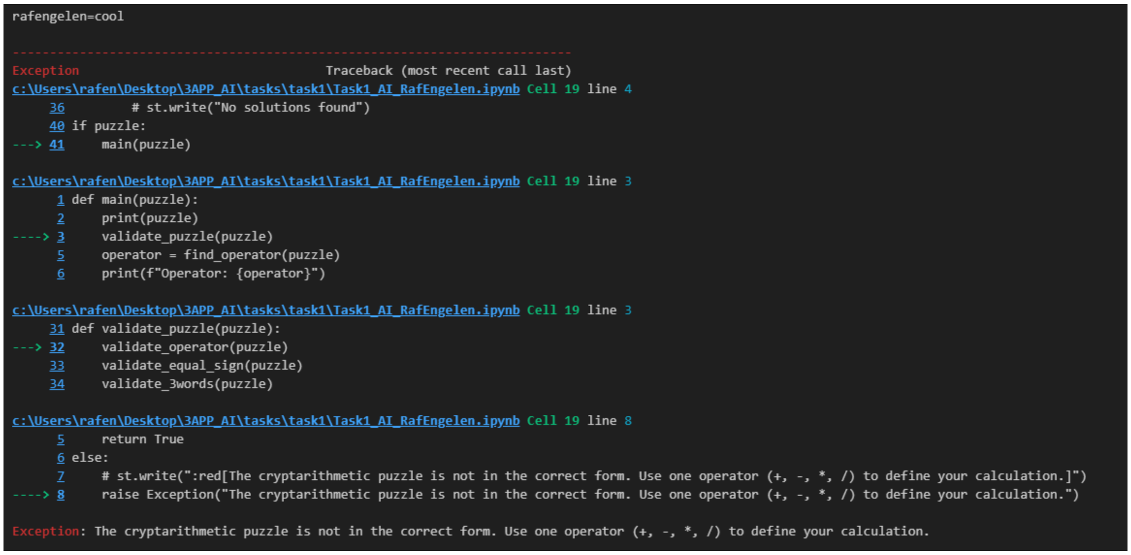
```
ai*is=fun
Operator: *
Words: ['AI', 'IS', 'FUN']
Letters: ('F', 'N', 'U', 'S', 'I', 'A')
Possible values for letters: {'F': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'N': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'U': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'S': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'I': [1, 2, 3, 4, 5, 6, 7, 8

Solution:
AI * IS = FUN
12 * 29 = 348
__________________________________________________________________
```

EXAMPLE 4:



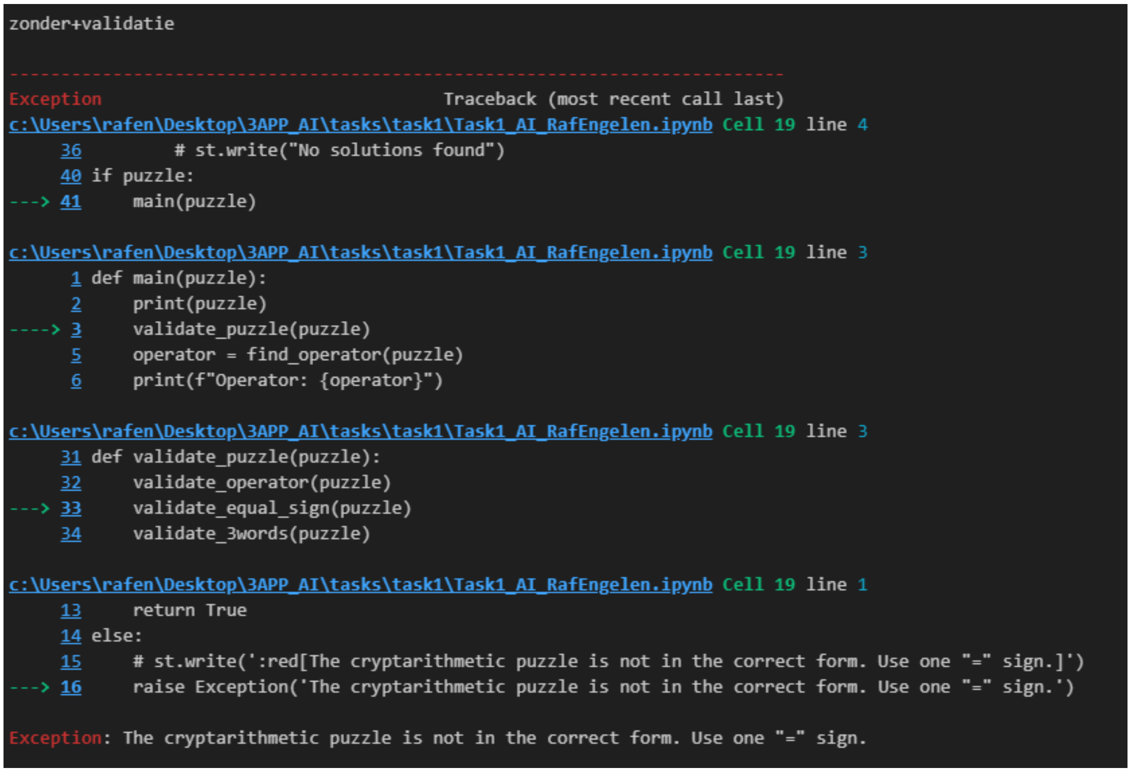
```
Even/odD=me
Operator: /
Words: ['EVEN', 'ODD', 'ME']
Letters: ('E', 'N', 'D', 'O', 'M', 'V')
Possible values for letters: {'E': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'N': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'D': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 'O': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'M': [1, 2, 3, 4, 5, 6, 7, 8, 9

Solution:
EVEN / ODD = ME
8586 / 477 = 18
__________________________________________________________________
```

EXAMPLE 5: No operator in puzzle



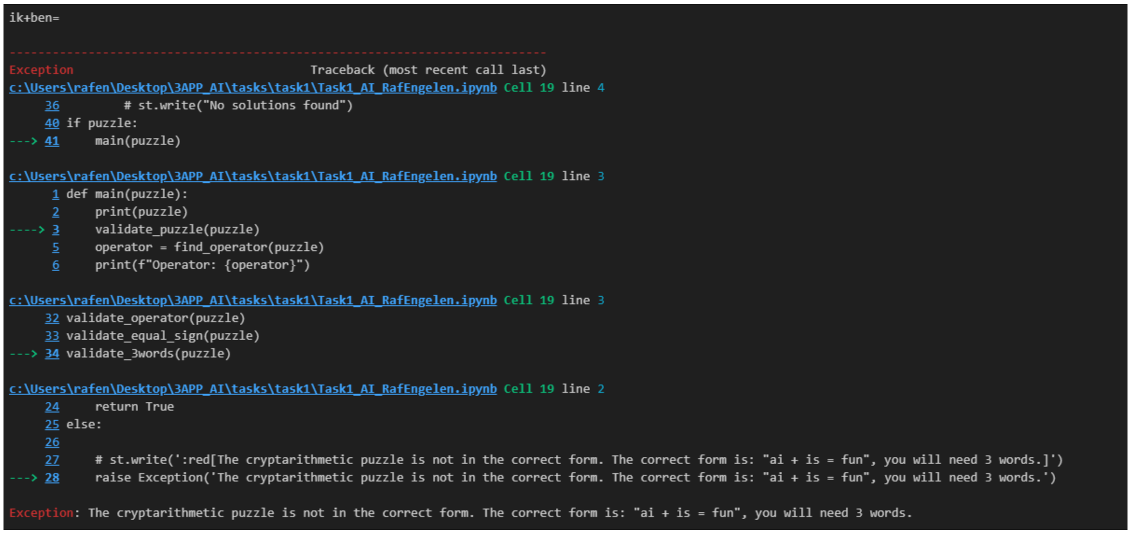
```
rafengelen=cool

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 4
     36         # st.write("No solutions found")
     40 if puzzle:
---> 41     main(puzzle)

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 3
      1 def main(puzzle):
      2     print(puzzle)
----> 3     validate_puzzle(puzzle)
      5     operator = find_operator(puzzle)
      6     print(f"Operator: {operator}")

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 3
     31 def validate_puzzle(puzzle):
---> 32     validate_operator(puzzle)
     33     validate_equal_sign(puzzle)
     34     validate_3words(puzzle)

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 8
      5     return True
      6 else:
      7     # st.write(":red[The cryptarithmetic puzzle is not in the correct form. Use one operator (+, -, *, /) to define your calculation.]")
----> 8     raise Exception("The cryptarithmetic puzzle is not in the correct form. Use one operator (+, -, *, /) to define your calculation.")

Exception: The cryptarithmetic puzzle is not in the correct form. Use one operator (+, -, *, /) to define your calculation.
```

EXAMPLE 6: No = sign



```
zonder+validatie

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 4
     36         # st.write("No solutions found")
     40 if puzzle:
---> 41     main(puzzle)

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 3
      1 def main(puzzle):
      2     print(puzzle)
----> 3     validate_puzzle(puzzle)
      5     operator = find_operator(puzzle)
      6     print(f"Operator: {operator}")

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 3
     31 def validate_puzzle(puzzle):
     32     validate_operator(puzzle)
---> 33     validate_equal_sign(puzzle)
     34     validate_3words(puzzle)

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 1
     13     return True
     14 else:
     15     # st.write(':red[The cryptarithmetic puzzle is not in the correct form. Use one "=" sign.]')
---> 16     raise Exception('The cryptarithmetic puzzle is not in the correct form. Use one "=" sign.')

Exception: The cryptarithmetic puzzle is not in the correct form. Use one "=" sign.
```

EXAMPLE 7: No 3 words



```
ik+ben=

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 4
     36         # st.write("No solutions found")
     40 if puzzle:
---> 41     main(puzzle)

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 3
      1 def main(puzzle):
      2     print(puzzle)
----> 3     validate_puzzle(puzzle)
      5     operator = find_operator(puzzle)
      6     print(f"Operator: {operator}")

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 3
     32 validate_operator(puzzle)
     33 validate_equal_sign(puzzle)
---> 34 validate_3words(puzzle)

c:\Users\rafen\Desktop\3APP_AI\tasks\task1\Task1_AI_RafEngelen.ipynb Cell 19 line 2
     24     return True
     25 else:
     26 
     27     # st.write(':red[The cryptarithmetic puzzle is not in the correct form. The correct form is: "ai + is = fun", you will need 3 words.]')
---> 28     raise Exception('The cryptarithmetic puzzle is not in the correct form. The correct form is: "ai + is = fun", you will need 3 words.')

Exception: The cryptarithmetic puzzle is not in the correct form. The correct form is: "ai + is = fun", you will need 3 words.
```