COMP 1012 Winter 2015 Assignment 4

Due Date: Thursday, March 12, 2015, 11:59 PM

New Material Covered

- user input, including from a menu, and error checking
- `if … elif … else …`
- generating all permutations of a list of items
- substitution in strings
- evaluating string expressions

Notes:

- Name your script file as follows: `<LastName><FirstName>A4Q1.py`. For example, `LiJaneA4Q1.py` is a valid name for a student named Jane Li. If you wish to add a version number to your file, you may add it to the end of the file name. For example, `SmithRobA4Q1V2.py` is a valid name for Rob Smith's file.
- Follow the posted programming standards to avoid losing marks.
- You must submit a ***Blanket Honesty Declaration*** to have your assignment counted. This one honesty declaration applies to all assignments in COMP 1012.
- To submit the assignment follow the instructions on the course website carefully. You will upload both script file and output via a dropbox on the course website. There will be a period of several days before the due date when you can submit your assignment. ***Do not be late!*** If you try to submit your assignment within 48 hours ***after*** the deadline, it will be accepted, but you will receive a penalty (roughly 1% per hour).

Question 1—Verbal Arithmetic [14 marks]

Description

Verbal arithmetic (also known as word math, alphametics or cryptarithmetic) is a class of puzzle where you have to assign a digit to each letter in a word. For example, TWO + TWO == FOUR has the solution T = 7, W = 6, O = 5, F = 1, U = 3 and R = 0, so that 765 + 765 == 1530. This is the only solution using digits 0 to 7.

There are a few simple rules:

1. Each letter represents the same digit wherever it occurs. E.g., T is always 7 in the puzzle above.
2. No two letters represent the same digit, so T and W could not both be 7.
3. Letters that start a word (e.g., T and F in the example above) cannot represent 0.

Puzzlers solve these puzzles using logic and math arguments. Some puzzles are quite difficult, but you can always make extremely simple examples such as BANANA = 102020, where obviously B = 1, A = 0 and N = 2.

In this assignment you will write a computer program that solves any word math puzzle by brute force. That is, your program will try all possible substitutions and find ones that work.

A sample session is shown on the following pages. Note that the user is presented with a menu of choices, and that inappropriate inputs are captured and rejected. Some prompts are repeated; in some cases it is because the user's input is invalid, and in others because the user can solve multiple puzzles in the same session.

The session shown does not take a long time to run (check the date lines) because the puzzles chosen are simple. ***Solving the puzzles with all 10 digits can take minutes.*** It is best to debug your program using simple examples.

Sample Session

```
            W O R D   M A T H   P U Z Z L E S

Enter the number of one of these puzzles (Q to quit):

0. BANANA == 102020 using digits from 0 to 2
1. HE + ME == WE using digits from 0 to 3
2. TWO + TWO == FOUR using digits from 0 to 7
3. FIVE + FIVE == SEVEN using digits from 0 to 7
4. SIX + SEVEN + SEVEN == TWENTY using digits from 0 to 8
5. SEND + MORE == MONEY using digits from 0 to 9
6. SPRING + RAINS + BRING + GREEN == PLAINS using digits from 0 to 9
7. CLOCK + TELLS + TIME == CHIMES using digits from 0 to 9
8. TERRIBLE + NUMBER == THIRTEEN using digits from 0 to 9
9. ENTER YOUR OWN MATHWORD PUZZLE

➢ BANANA
You entered BANANA; enter a valid digit, or Q

Enter the number of one of these puzzles (Q to quit):

0. BANANA == 102020 using digits from 0 to 2
1. HE + ME == WE using digits from 0 to 3
2. TWO + TWO == FOUR using digits from 0 to 7
3. FIVE + FIVE == SEVEN using digits from 0 to 7
4. SIX + SEVEN + SEVEN == TWENTY using digits from 0 to 8
5. SEND + MORE == MONEY using digits from 0 to 9
6. SPRING + RAINS + BRING + GREEN == PLAINS using digits from 0 to 9
7. CLOCK + TELLS + TIME == CHIMES using digits from 0 to 9
8. TERRIBLE + NUMBER == THIRTEEN using digits from 0 to 9
9. ENTER YOUR OWN MATHWORD PUZZLE

➢ 10
You entered 10; enter a valid digit, or Q

Enter the number of one of these puzzles (Q to quit):

0. BANANA == 102020 using digits from 0 to 2
1. HE + ME == WE using digits from 0 to 3
2. TWO + TWO == FOUR using digits from 0 to 7
3. FIVE + FIVE == SEVEN using digits from 0 to 7
4. SIX + SEVEN + SEVEN == TWENTY using digits from 0 to 8
5. SEND + MORE == MONEY using digits from 0 to 9
6. SPRING + RAINS + BRING + GREEN == PLAINS using digits from 0 to 9
7. CLOCK + TELLS + TIME == CHIMES using digits from 0 to 9
8. TERRIBLE + NUMBER == THIRTEEN using digits from 0 to 9
9. ENTER YOUR OWN MATHWORD PUZZLE

➢ 2
Date: ..............................Sat Feb 21 11:30:00 2015
```

```
Solving the puzzle: TWO + TWO == FOUR
Calculating permutations of 8 items; please wait ...
PUZZLE:    TWO + TWO == FOUR
SOLUTION:  765 + 765 == 1530

Date: .............................Sat Feb 21 11:30:01 2015

Enter the number of one of these puzzles (Q to quit):

0. BANANA == 102020 using digits from 0 to 2
1. HE + ME == WE using digits from 0 to 3
2. TWO + TWO == FOUR using digits from 0 to 7
3. FIVE + FIVE == SEVEN using digits from 0 to 7
4. SIX + SEVEN + SEVEN == TWENTY using digits from 0 to 8
5. SEND + MORE == MONEY using digits from 0 to 9
6. SPRING + RAINS + BRING + GREEN == PLAINS using digits from 0 to 9
7. CLOCK + TELLS + TIME == CHIMES using digits from 0 to 9
8. TERRIBLE + NUMBER == THIRTEEN using digits from 0 to 9
9. ENTER YOUR OWN MATHWORD PUZZLE

➢ 9

Enter a word math puzzle:
GRAB = 1234

Puzzle 'GRAB = 1234' does not contain '=='

Enter a word math puzzle:
GRAB == 1234

What is the largest digit allowed?
4

Date: .............................Sat Feb 21 11:30:46 2015

Solving the puzzle: GRAB == 1234
Calculating permutations of 5 items; please wait ...
PUZZLE:    GRAB == 1234
SOLUTION:  1234 == 1234

Date: .............................Sat Feb 21 11:30:46 2015

Enter the number of one of these puzzles (Q to quit):

0. BANANA == 102020 using digits from 0 to 2
1. HE + ME == WE using digits from 0 to 3
2. TWO + TWO == FOUR using digits from 0 to 7
3. FIVE + FIVE == SEVEN using digits from 0 to 7
4. SIX + SEVEN + SEVEN == TWENTY using digits from 0 to 8
5. SEND + MORE == MONEY using digits from 0 to 9
6. SPRING + RAINS + BRING + GREEN == PLAINS using digits from 0 to 9
7. CLOCK + TELLS + TIME == CHIMES using digits from 0 to 9
8. TERRIBLE + NUMBER == THIRTEEN using digits from 0 to 9
9. ENTER YOUR OWN MATHWORD PUZZLE

➢ q
```

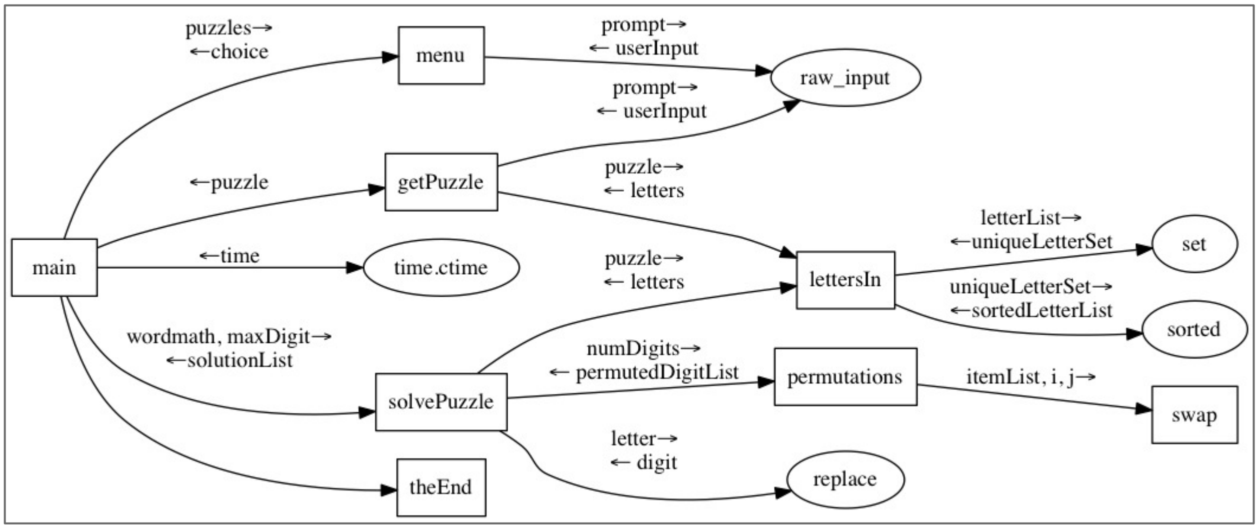
```
Programmed by The Instructors
Date: Sat Feb 21 11:31:03 2015
** End of processing **
```

Program Details

The following call diagram shows the functions you should have in your solution, and which other functions they call, what gets passed down as an argument, and what is returned. At the left is the function `main`. Your main code should be in a function called `main` in this program. In your script file, apart from constant and function definitions, you should have only one line at the left margin, namely a call to `main` at the end of the file. It should look like this:

```
main()   # required call to main to start program
```

To the right of `main` in the diagram are the other functions in the program. Built-in functions of note are shown in ovals (not every built-in function you will use is shown, just the interesting ones). You have to program the function shown as boxes.



The following list explains what each of these functions is supposed to do. When developing your solution to the assignment, treat each function definition as a little project. First develop `main`. It calls `menu`, `getPuzzle`, `solvePuzzle` and `theEnd`. Write very simple stubs for each of these called functions, stubs that are only one or two lines long and that always return the same value. Get `main` working correctly. Then one by one get the called functions working. When necessary, introduce stubs for functions like `lettersIn` and `permutations`, and later fill in their details.

The list below is in alphabetical order (except that main comes first) for convenience in finding them. It is rarely appropriate to develop the functions in this order. However, it is a good idea to list your function definitions in your script file in alphabetical order as well, to make them easier to find.

- `main()`: This function is where your main program resides.
  - Variables defined here will be local variables, so you will ***not*** be able to access them in the shell window after running your code, unlike in past assignments.
  - This function implements a loop to ask the user over and over which puzzle he/she wants solved. It should use the design from the notes for repetitive input (not the loop for error checking), and it should call `menu` for each input to determine the user's choice.

- When calling menu, `main` should pass down a tuple of possible puzzles. This tuple should be defined as a global constant, something like this:

```
PUZZLES = ( "2BANANA == 102020",
            "3HE + ME == WE",
            "7TWO + TWO == FOUR",
            ...
            "9TERRIBLE + NUMBER == THIRTEEN",
            "0ENTER YOUR OWN MATHWORD PUZZLE")
```

  Note that `each` line looks similar to the sample output prompts shown previously. (You will fill in the missing puzzles from the sample output.) It is the job of menu to take this input and format it to present to the user.

- The digit at the start of each puzzle specifies the largest digit in the desired solution. It is a good idea to keep this as small as possible, for two reasons. If the largest digit is *d*, then *d*! permutations will need to be considered, and *d*! grows rapidly with *d* so puzzles with *d* = 9 take a long time to solve. Also, if we allow larger digits, some of these puzzles have many solutions.
- In the case that the user chooses to enter their own word math puzzle, `main` calls `getPuzzle` to read in the puzzle to be solved.
- `main` calls `solvePuzzle` to solve each puzzle selected. Before and after calling `solvePuzzle` it prints out a date line so that the user can see how long the solution took.
- When the user enters 'Q' or 'q' as their choice of puzzle, a value of 'Q' should be returned to `main` from menu, and that should be the condition to quit the input loop.
- At the end of `main` it should call `theEnd` to print a termination message, as in the last assignment. You may use the same function you defined there.

• `getPuzzle()`: This function has no parameters. It interacts with the user to acquire a puzzle to be solved.
  - The puzzle is returned to `main` in the same format as the puzzles in PUZZLES shown above, specifically a string containing the largest digit in the 0 position, followed by the puzzle itself, all in capital letters.
  - The two parts of the puzzle (the text and the largest digit) are input from the user in separate steps. Each one has an input loop to deal with invalid inputs.
  - The text of the puzzle is invalid if it does not contain a test for equality, '==', or if it contains more than 10 letters, since there are only 10 digits. (It could also be invalid for other reasons, but these are the one you need to check.) This function calls `lettersIn` to determine how many letters are in the puzzle to test that number.
  - The largest digit is invalid if it has more than one character, or if the one character it does have is not a digit. If it passes that test, you can convert it to an `int` and check its value. If the largest digit is 5, say, then the digits to use in the puzzle solution are 0, 1, 2, 3, 4, 5. There are six of them. In general, the largest digit + 1 must be at least as big as the number of distinct letters in the puzzle.

• `lettersIn(text)`: This function returns a sorted list of the distinct letters in the string `text`. There are many ways to do this, but we want you to use a very simple way, explained here.
  - In addition to lists and tuples and strings, Python has a set data type. In the Canopy shell, type '`set("BANANA")`' and see what you get. Sets, unlike lists, can contain only one copy

of each entry. A very simple way of determining the unique entries in a collection is to convert it to a set, as in '`mySet = set(collection)`'. If the collection is a string, then you get a set of the distinct characters in the string.

  - Before converting the text to a set of characters, it is important to convert everything to upper case; otherwise `'E'` and `'e'` will be two separate characters; `text.upper()` returns a string that is the same as `text` except that every letter is in uppercase. Non-letters are unchanged.
  - After converting the text to a set of letters, you can extract just the letters (no blanks, no mathematical operators) by taking a *set intersection*[1] with the set of capital letters in the alphabet, "ABCDEFGHIJKLMNOPQRSTUVWXYZ" (use the constant string `ascii_uppercase` from the library `string` to avoid making a mistake). To find the intersection of two sets, `set1` and `set2`, use the operation `set1.intersection(set2)`.
  - Finally, we want to convert the set of letters back to a list in sorted order. The function `sorted(aSet)` turns it into a list sorted in alphabetical order.

- `menu(listPuzzles)`: This function finds out from the user which one of the puzzles in `listPuzzles` the user wants.
  - `menu` should do all its work with `listPuzzles`, not with the global variable `PUZZLES` mentioned above. It is `main` that is in charge, and it might send down only a subset of the puzzle list for some reason.
  - The prompt provided by `menu` consists of a formatted version of the puzzles in `listPuzzles`. This prompt does not all have to be output by `raw_input`, although it can be. It is also acceptable to print the list of puzzles before calling `raw_input`.
  - The formatting of each puzzle includes a) putting a count at the beginning of each line so that the user will have something to choose, b) removing the leading puzzle character, which is the largest digit allowed in the solution, and printing it at the end of the line as "`using digits from 0 to` *d*" where *d* is the largest digit, as shown in the sample output., c) not displaying the largest digit if that largest digit is 0, as it would not be appropriate for the last entry.
  - The big character pointing out where the user should enter his input is the Unicode character `U"\N{BLACK RIGHTWARDS ARROWHEAD}"`.
  - If the user's entry is `'q'` or `'Q'` it should be returned to the calling code as `'Q'`. This indicates the user wants to quit.
  - Otherwise, user entries should be checked for errors, and the prompting repeated as long as the entry isn't valid. We are looking for a single digit, so an entry that is more than one character long is invalid, as is an entry not one of the line numbers displayed.
  - Return either 'Q' or a single digit choice as an `int` (e.g., 2 or 7), and nothing else.
- `permutations(NN)`: The parameter is a positive integer. Return a list of all the permutations of the digits from 0 to NN-1.
  - `permutations(3)` should return: ['012', '102', '201', '021', '120', '210'] of length 3! = 6.

[1] It is expected you know some set theory, but if not, the intersection of two sets is the set of entries in both of them.

- `permutations(4)` should return: ['0123', '1023', '2013', '0213', '1203', '2103', '2130', '1230', '3210', '2310', '1320', '3120', '3021', '0321', '2301', '3201', '0231', '2031', '1032', '0132', '3102', '1302', '0312', '3012'] of length 4! = 24.
- Either of these results, `permutations(3)` or `permutations(4)`, could be used as the value of a call to `permutations` when using it as a stub.
- In general, the return value should contain NN! entries. Each one should be a string containing the digits from 0 to NN-1 in some order. All the entries in the list should be different. The first entry in the returning list should be '012…d' where d = NN – 1. Every string in the list should differ from the immediately previous entry by swapping two digits.
- Now the good news: you don't have to write `permutations` from scratch. Instead, go to http://www.quickperm.org and find on that page the algorithm "A Pseudo-Code Representation (Countdown)". Adapt that algorithm so that it is Python code. ***Make sure to put a citation to this webpage in the comments for your code.***
- In addition to converting the syntax (that is, how the instructions are written), you need to add three things to the algorithm provided to complete your function `permutations`: 1) for the arbitrary list a, use `['0', '1', '2', … , 'd']` where d = NN -1 (note that this is a list of one-character strings); 2) before the outer loop begins, initialize a list of permutations and put the initial permutation string in it; 3) after each time you perform a swap (there is only one call to swap in the loop), add the new permutation to the list.
- Whenever you add a permutation to the growing list of permutations, first convert it to a string. So for example if NN = 5, the first permutation would be `'01234'`. This can be made from the list containing ['0', '1', '2', '3', '4'] by using join.
- After the loops are complete, return the list of permutations.

- `solvePuzzle(wordMath, maxDigit)`: This function finds and prints all solutions to the given word math puzzle using digits from 0 to `maxDigit`. In addition to printing them, it also returns a list of the permutations that provided solutions.
  - If a word math puzzle like "`BANANA == 102020`" has three distinct letters, A, B and N, we want to try all possible ways of assigning 0, 1 and 2 to these letters. The code can find the list of distinct letters by calling `lettersIn`. It can find all the possible permutations of the digits from 0 to `maxDigit` by calling `permutations`. The only step left is to figure out how to apply a given permutation to a given puzzle.
  - Given a puzzle like "`BANANA == 102020`", with distinct letters `['A', 'B', 'N']` from `lettersIn`, apply a permutation like "`012`" by replacing `'A'` with `'0'`, `'B'` with `'1'` and `'N'` with `'2'`. If `ss` is a string, the built-in function call `ss.replace(str1,str2)` produces a new string that is just like `ss` except that everywhere there was a copy of `str1` in `ss`, there is now a copy of `str2`. For example, "`BANANA == 102020".replace('A','1')` returns "`B1N1N1 == 102020`".[2]
  - Once all the letters have been replaced by digits, two tests for validity can be applied. First, check that there is no occurrence of `' 0'` (a blank and a zero) in the resulting string. That would be invalid. Second, try `eval` on the string. Since each puzzle has an '==' operator in it, the ideal result would be that the string evaluate to `True`. If it does, that permutation is a

[2] If you are eager to make your code run as fast as possible, look up `string.maketrans` and `string.translate` in the string library.

solution. If not, this permutation is not a solution. Whenever a solution is found, print it out and add it to the list of solution permutations to be returned.

  - Note that the test for a leading zero is rather crude. It will fail if the first letter is at the start of the string, or if there are not blanks around every operator. Consider making it better.

- `swap(aList, pos0, pos1)`: This function swaps two entries in `aList`, the ones in positions `pos0` and `pos1`. Use tuple assignments to do it simply. Remember that if you change the entries of `aList` in the called code, they will also be changed in the calling code. There is no need to return the list.
- `theEnd()`: This function has no parameters. It prints three lines of output to show the program is terminating, as shown in the sample output.

Hand-in

Hand in the script file. Hand in sample output showing a session where you try three puzzles, of which at least one uses 10 digits, and at least one is entered at the prompt. Also make intentional errors so that you can demonstrate correct error handling of at least three different error conditions.