Lab 3

COMP9021, Session 1, 2018

1 ☞ Finding particular sequences of prime numbers

Write a program consecutive_primes.py that finds all sequences of 6 consecutive prime 5-digit numbers, say (a, b, c, d, e, f), with $b = a + 2$, $c = b + 4$, $d = c + 6$, $e = d + 8$, and $f = e + 10$. So a, b, c, d, e and f are all 5-digit prime numbers and no number between a and b, between b and c, between c and d, between d and e, and between e and f is prime.

The expected output is:

```
The solutions are:

    13901   13903   13907   13913   13921   13931
    21557   21559   21563   21569   21577   21587
    28277   28279   28283   28289   28297   28307
    55661   55663   55667   55673   55681   55691
    68897   68899   68903   68909   68917   68927
```

2 ☞ Finding particular sequences of triples

Write a program triples_1.py that finds all triples of positive integers (i, j, k) such that i, j and k are two digit numbers, no digit occurs more than once in i, j and k, and the set of digits that occur in i, j or k is equal to the set of digits that occur in the product of i, j and k.

The expected output is:

```
20 x 79 x 81 = 127980
21 x 76 x 80 = 127680
28 x 71 x 90 = 178920
31 x 60 x 74 = 137640
40 x 72 x 86 = 247680
46 x 72 x 89 = 294768
49 x 50 x 81 = 198450
56 x 87 x 94 = 457968
```

3 ☞ Finding special triples of the form $(n, n+1, n+2)$

Write a program triples_2.py that finds all triples of consecutive positive three-digit integers each of which is the sum of two squares, that is, all triples of the form $(n, n+1, n+2)$ such that:

- n, $n+1$ and $n+2$ are integers at least equal to 100 and at most equal to 999;
- each of n, $n+1$ and $n+2$ is of the form $a^2 + b^2$.

Hint: As we are not constrained by memory space for this problem, we might use a list that stores an integer for all indexes n in $[100, 999]$, equal to 1 in case n is the sum of two squares, and to 0 otherwise. Then it is just a matter of finding three consecutive 1's in the list. This idea can be refined (by not storing 1s, but suitable nonzero values) to not only know that some number is of the form $a^2 + b^2$, but also know such a pair (a, b)...

The output of the program could be (the decompositions into sums of squares could differ):

```
(144, 145, 146) (equal to (0^2+12^2, 8^2+9^2, 5^2+11^2)) is a solution.
(232, 233, 234) (equal to (6^2+14^2, 8^2+13^2, 3^2+15^2)) is a solution.
(288, 289, 290) (equal to (12^2+12^2, 8^2+15^2, 11^2+13^2)) is a solution.
(360, 361, 362) (equal to (6^2+18^2, 0^2+19^2, 1^2+19^2)) is a solution.
(520, 521, 522) (equal to (14^2+18^2, 11^2+20^2, 9^2+21^2)) is a solution.
(576, 577, 578) (equal to (0^2+24^2, 1^2+24^2, 17^2+17^2)) is a solution.
(584, 585, 586) (equal to (10^2+22^2, 12^2+21^2, 15^2+19^2)) is a solution.
(800, 801, 802) (equal to (20^2+20^2, 15^2+24^2, 19^2+21^2)) is a solution.
(808, 809, 810) (equal to (18^2+22^2, 5^2+28^2, 9^2+27^2)) is a solution.
```

4 Decoding a sequence of operations (Moderately advanced, optional)

Write a program `evaluates_to_100.py` that finds all possible ways of inserting `+` and `-` signs in the sequence `123456789` (at most one sign before any digit) such that the resulting arithmetic expression evaluates to `100`.

Here are a few hints.

- `1` can either be preceded by `-`, or optionally be preceded by `+`; so `1` starts a negative or a positive number.
- All other digits can be preceded by `-` and start a new number to be subtracted to the running sum, or be preceded by `+` and start a new number to be added to the running sum, or not be preceded by any sign and be part of a number which it is not the leftmost digit of. That gives 3^8 possibilities for all digits from `2` to `9`. We can generate a number N in $[0, 3^8 - 1]$. Then we can:
  - consider the remainder division of N by 3 to decide which of the three possibilities applies to `2`;
  - consider the remainder division of $\frac{N}{3}$ by 3 to decide which of the three possibilities applies to `3`;
  - consider the remainder division of $\frac{N}{3^2}$ by 3 to decide which of the three possibilities applies to `4`;
  - ...

The expected output is (the ordering could be different):

```
 1 + 23 - 4 + 5 + 6 + 78 - 9 = 100
 123 - 4 - 5 - 6 - 7 + 8 - 9 = 100
 123 + 45 - 67 + 8 - 9 = 100
 123 + 4 - 5 + 67 - 89 = 100
 12 + 3 + 4 + 5 - 6 - 7 + 89 = 100
 123 - 45 - 67 + 89 = 100
 12 - 3 - 4 + 5 - 6 + 7 + 89 = 100
 1 + 2 + 34 - 5 + 67 - 8 + 9 = 100
 1 + 2 + 3 - 4 + 5 + 6 + 78 + 9 = 100
-1 + 2 - 3 + 4 + 5 + 6 + 78 + 9 = 100
 12 + 3 - 4 + 5 + 67 + 8 + 9 = 100
 1 + 23 - 4 + 56 + 7 + 8 + 9 = 100
```

Write two versions, one using the `eval()` function, and one not using it (the `eval()` function is very powerful as it can evaluate any string, but unsafe as a consequence of that power).

5 Dice rolls (optional, needs a module not installed on CSE computers)

Write a program `dice_rolls.py` that prompts the user twice, for strictly positive integers $s_1, \ldots, s_k$ intended to represent the number of sides of some dice, and for an integer N meant to represent the number of times these dice should be cast. If the first input is empty, then a single six-sided die will be used. If the first input is not empty, then any part of it which is not a strictly positive integer will be replaced by 6 (so for instance, inputting `12 0 3 -1 python 4 5A` is equivalent to inputting `12 6 3 6 6 4 6`). If the second input is empty or is not a strictly positive integer, then the number of rolls will be set to 1,000.

Here are possible interactions:

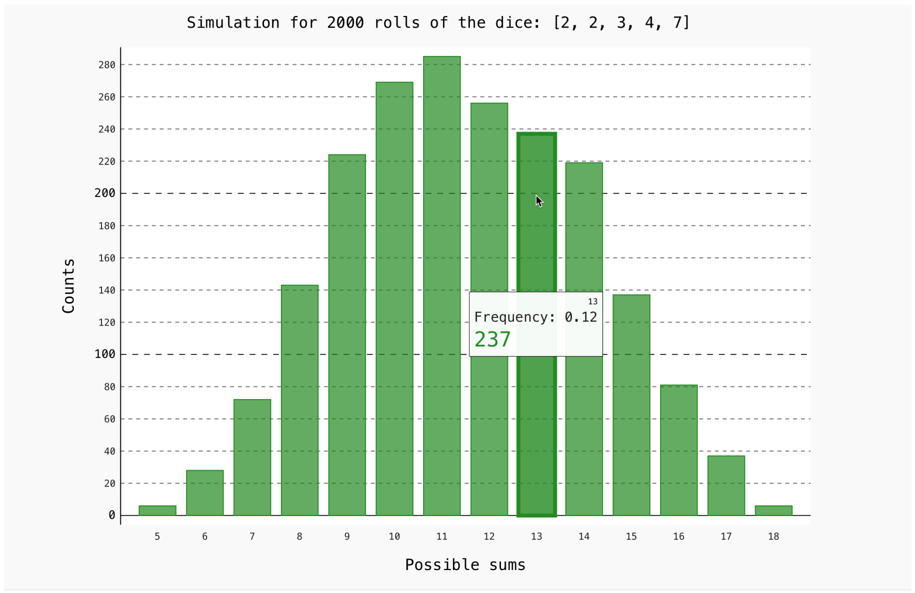
```
$ python3 dice_rolls.py
Enter N strictly positive integers (number of sides of N dice):
You did not enter any value, a single standard six-sided die will be rolled.

Enter the desired number of rolls:
Input was not provided or invalid, so the default value of 1,000 will be used.
$ python3 dice_rolls.py
Enter N strictly positive integers (number of sides of N dice): 2 0 3 python
Some of the values, incorrect, have been replaced with the default value of 6.

Enter the desired number of rolls: 0
Input was not provided or invalid, so the default value of 1,000 will be used.
$ python3 dice_rolls.py
Enter N strictly positive integers (number of sides of N dice): 2 4 2 7 3

Enter the desired number of rolls: 2000
```

The program should generate N times k random numbers between 1 and $s_1, \ldots, s_k$, respectively, sum them up, and display the N sums in the form of a histogram, created as an object of class `Bar` of the `pygal` module, that can be displayed in a browser by opening a file named `dice_rolls.svg`—check out `render_to_file()`. To create the histogram from the N sums, check out `add()`. The histogram should have—check out the `Style` class from the `pygal.style` module:

- as title for the histogram, `Simulation for` *N* `rolls of the dice:` *L* where *L* is the ordered list of the number of sides of the dice;
- as labels on the x-axis, all possible sums;
- as title for the x-axis, `Possible sums`;
- the major labels of the y-axis having a font size of 12 pt;
- as title for the y-axis, `Counts`;

- tooltips displaying, besides the count, `Frequency:` *f* where *f* is the count divided by *N*, displayed with 2 digits after the decimal point;
- bars having the colour whose rgb code is `#228B22`;
- no legend.

Here is one possible such histogram obtained with `2 4 2 7 3` as first input and `2000` as second input, with the cursor hovering over the bar for the sum of 13.


Simulation for 2000 rolls of the dice: [2, 2, 3, 4, 7]
Counts
Possible sums
Frequency: 0.12