Python projects

I wrote some for the edX course, others out of interest. A place to collect all my insights from each project. For version control I’ve just been saving subsequent versions of the files. I wonder if other IDEs will save it like Github.

Course:

* Alphabetiser  
  Learning how to avoid IndexError  
  Char can be directly compared in an alphabetical format
* Bob counter  
  Similarly, I wrote my own version of str.count(‘sub’). The idea is being aware of how to search for a description of desired function to avoid doing double work.
* Secret number guesser  
  Remembered about docstrings here  
  Use of bisection search, which I’ve encountered before in Fast Fourier Transform.  
  This was much more challenging than I expected. I suspect it’s because I didn’t properly clarify what I wanted out of this, which was to define boundaries and then slowly shift the appropriate one.
* Vowel counter  
  I realised that you can place any variable name as a parameter here, because I used a type.
* Midterm
* List flattener: Uses recursion and I’m unable to figure out how to write it.

Self:

* Calculator using iterative addition  
  Got very laggy with all the additions, especially with pow(). Also only works with int.
* Babbage()  
  I prepared this in Excel beforehand, and learned about how coding was more compact compared to Excel’s need for defined domains of cells.  
  Also the first time I began calling functions and returning data from within their scopes.
  + Random polynomial generator  
    Surprisingly simple when all’s said and done. Also imports **random** module within the code. Also begun using list comprehension here to instantiate the various arrays for power, coefficients and input value.
  + Difference Engine  
    I began to see the convoluted nature of nested for loops. But in the end it’s still fairly straightforward to read through.
  + Polynomial root reader  
    Serves as a translator between human language and computer language: Converting algebraic factors into a pair of arrays through string slicing.
  + Polynomial generator using roots (WIP)  
    Involves combinations and exclusive multiplication. Very convoluted. **itertools** seems promising though and I’ve been reading through the code for itertools.combinations().
  + Polynomial reconstructor  
    Depends on the above to generate reference polynomials to then reconstruct the random polynomial from babbage()
* Python web scraper
* Data diode

Polyn gen with roots:

Maths theory – Permutations and combinations  
Max number for x terms with n choices = n^x  
🡪 For a sequence of binary terms, max number = 2^x of combinations  
Permutation/List – Order matters, P(n,r) = n!/(n-r)!  
Combination/Group – Presence only so remove redundancies, C(n,r) = n!/(n-r)!/r!  
[Combinations sound simple so less, permutations sound complicated so more]

Examples: For 5 terms, max = 2^5 = 32  
0 ON: 1  
1 ON: 5!/(5-1)!/1! = 5  
2 ON: 5!/(5-2)!/2! = 5\*4/2 = 10  
3 ON = 2 OFF  
4 ON = 1 OFF  
5 ON = 0 OFF  
🡪 Total = (1 + 5 + 10) \* 2 = 32

Examples: For 6 terms, max = 2^6 = 64  
0 ON: 1  
1 ON: 6!/(6-1)!/1! = 6  
2 ON: 6!/(6-2)!/2! = 6\*5/2 = 15  
3 ON = 3 OFF = 6!/(6-3)!/3! = 6\*5\*4/6 = 20  
4 ON = 2 OFF  
5 ON = 1 OFF  
6 ON = 0 OFF  
🡪 Total = (1 + 6 + 15) \* 2 + 20 = 44 + 20 = 64

This serves as a check for the number of combinations generated

COEF and CONST arrays are reflections i.e. (0,1,0) <-> (1,0,1)

Might have to use loops as well

Recursion is possible, but I haven’t been able to wrap my head around it. But what if we slice the original list into smaller lists?  
The outer list proceeds along one loop, and the inner list along another. The outer list moves one step every time the inner one completes.  
Recursion in this sense might involve first generating the initial list with r choices, and then slicing it into smaller lists. I’ve done it with r = 2, and if I get r =3 I think I will be that much closer to making it happen.  
[1,1,1,0,0,0,0] 🡪 [1,0,0,0,0,0,0]; [1,0,0,0,0,0]; [1,0,0,0,0]  
But every time the outer list shifts right, the whole set must be generated again  
[0,1,1,1,0,0,0,0] 🡪 [0,1,0,0,0,0,0]; [1,0,0,0,0]; [1,0,0,0]  
What matters is that the inner loop completes before the outer loop shifts once, and this will apply to any such case of inner/outer loops. I might be able to write it recursively, or at least create a function that works like that.  
The advantage is that it can run for any value of r, if I get it to work automatically.  
However, it might be computationally easier to convert r so that we find the number of 0s instead. This is done by taking (n-r), and applying it to the reflected list.

[1,0,0,0,0] 🡪 [0,0,0,0], then run the formula on this?  
My base case will be a list comprising of zeroes, in which the function will run  
We can do this by checking for the presence of 1, which will produce a ValueError in the base case. That can be handled with try-except.

For r = 2, # of combo = n(n-1)/2

For sliceClocker(n,r), there will be a few loops going on. Hopefully they can capture all the possibilities.  
Loop for number of choices, up till r.  
Inner/outer loop clocking. I ran into the problem while coding the Flash game, which seems to involve generating a certain number of loops and then running them. But that’s simply not possible. Instead, I might need another kind of loop.  
Place shifter. Inner/outer loop is the base case, and this shifts that base case outwards. It will have to work a lot though with each case. (To avoid excessive computations, I could reflect existing combos.)

Finally found a function that can generate the combinations I need: itertools.product(). The actual expression is a bit more complicated than that though. I think my next step will be to sort them according to the value of r.  
The print() keeps returning the generator object address instead of its contents.

lst = list(map(list, itertools.product([0, 1], repeat=n)))  
Appears clunkier because of double use of list()  
OR  
lst = [list(i) for i in itertools.product([0, 1], repeat=n)]  
List comprehension

list() – Converts an iterable into a list  
map(function, iterable) – Applies function across the iterable

At any rate, the simplest case to try this out on would be for r = 3.  
My current issue is with indexes, which falls under the tasks of the placeShifter().

X Process the coefficients and constants from the roots into lists

Generate coef for each n^x term for x in range(k)

Sort coefs by value of x

Generate combinations based on number of terms from COEF (and thus from CONST)  
 Starting with k # of terms:

X = 0 🡪 1

X = 1 🡪 k

X = 2 🡪 (k-1)!

X Select exclusive entries from COEF and CONST lists using itertools.compress()

Sum sub-coefs together for n^x term

Return final expression in linear form

'' in table form

I did it. I completed the function. It was all thanks to the itertools library, notably product() and compress(). Only took two months, and about one week’s worth of coding.

Minecraft

* Recipe exporter
* Slimycraft