SOFTWARE 1 PRACTICAL

Debugging & Recursion

Week 8 – Practical 12

Debugging a program with a DeBugger

A computer program will always do what you tell it to, but what you tell the program to do might not be the same as what you wanted the program to do. These errors are bugs in a computer program. Bugs happen when the programmer has not carefully thought about what exactly the program is doing. There are three types of bugs that can happen with your program:

1. **Syntax Errors** are a type of bug that comes from typos. When the Python interpreter sees a syntax error, it is because your code isn't written in proper Python language. A Python program with even a single syntax error won't run.
2. **Runtime Errors** are bugs that happen while the program is running. The program will work up until it reaches the line of code with the error, and then the program terminates with an error message (this is called crashing). The Python interpreter will display a "traceback" and show the line where the problem happens.
3. **Semantic Errors** are the trickiest to fix. These bugs don't crash the program, but it isn't doing what the programmer intended for the program to do. For example, if the programmer wants the variable total to be the sum of the values in variables a, b, and c but writes total = a * b * c, then the value in total will be wrong. This could crash the program later on, but it is not immediately obvious where the semantic bug happened.

Finding bugs in a program can be hard, if you even notice them at all! When running your program, you may discover that sometimes functions are not called when they are supposed to be, or you may code the condition for a while loop wrong, so that it loops the wrong number of times.

It can be hard to figure out how your code could be causing a bug. The lines of code get executed quickly and the values in variables change so often. We can use several `print` statements to display the state of some variables to guide us toward the error. A better alternative is to use a debugger.

A debugger is a program that lets you step through your code one line at a time in the same order that Python executes them. The debugger also shows you what values are stored in variables at each step.

TO DO at Home: *debugger tutoriasl*

1. If you are not familiar with debugging, follow the tutorial at: https://www.cs.uky.edu/~keen/help/debug-tutorial/debug.html

2. Once you are more familiar with debugging, watch the YouTube video on how to debug using VS Code. You could also practice using the debugger by using the example given in the Idle tutorial.

Exercise 1: *wildcard pattern*

Given a binary pattern that contains `'?'` wildcard character at few positions, find all possible combinations of binary strings that can be formed by replacing the wildcard character by either 0 or 1. For example, for wildcard pattern `1?11?00?1?`, the possible combinations are

```
1011000010    1011000011    1011001100    1011000111
1011100010    1011100011    1011101100    1011100111
1111000010    1111000011    1111001100    1111000111
1111100010    1111100011    1111101100    1111100111
```

We can easily solve this problem using recursion. The idea is to process each character of the pattern one by one and recur for the remaining pattern. If the current digit is `0` or `1`, we ignore it and if the current character is a wildcard character `'?'`, we replace it with 0 & 1 and recur for the remaining pattern.

Exercise 2: *Text on 9 keys – T9 (from practical 08 – Exercise 2)*

T9, which stands for **Text on 9 keys**, is a predictive text technology for mobile phones, specifically those that contain a 3x4 numeric keypad as shown in Figure 1. T9's objective is to make it easier to type text messages. It allows words to be entered by a single keypress for each letter, as opposed to the multi-tap approach used in conventional mobile phone text entry, in which several letters are associated with each key, and selecting one letter often requires multiple keypresses.

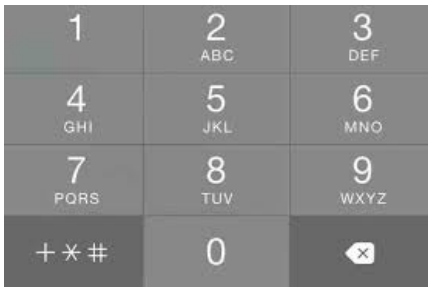

Figure 1: Example of a **3 × 4** numeric keypad used for T9

Given a sequence of numbers between [2-9], return a **set** of all possible combinations of words formed from the mobile keypad shown in Figure 1. The mobile keypad is store in a dictionary as shown below:

```
t9_keypad = {'2':['a','b','c'], '3':['d','e','f'],
             '4':['g','h','i'], '5':['j','k','l'],
             '6':['m','n','o'], '7':['p','q','r','s'],
             '8':['t','u','v'], '9':['w','x','y','z']}
```

Given the digits `'24'` the function `allwords(digits, keypad)` should return the set of 9 elements `{'AG', 'AH', 'AI', 'BG', 'BH', 'BI', 'CG', 'CH', 'CI'}`.

Exercise 3:

We are given a set of items, each with a weight and a value and we need to determine the number of each item to include in a collection (a bag for example) so that the total weight is less than or equal to a given limit and the total value is as large as possible. In addition, the items are indivisible; we can either take an item or not. Note that there might be more than one item with same value and same weight. For example,

```
Input:
value = [20, 5, 5, 10, 40, 15, 25]
weight = [1, 2, 2,  3,  8,  7,  4]
max_weight = 10

Output: 60
value = 20 + 40 = 60
weight = 1 + 8 = 9
```

The idea is to use **recursion** to solve this problem. For each item, there are two possibilities

1. We include current item in the bag and recur for remaining items with decreased capacity of the bag. If the capacity becomes negative, do not recur or return -INFINITY.

2. We exclude current item from the bag and recur for remaining items

Finally, we return maximum value we get by including or excluding current item. The base case of the recursion would be when no items are left, or capacity becomes 0.

Problem: *Introduction to the Predator Prey Problem*

The predator problem is a simulation that attempts to predict the relationship in populations between a population of foxes and rabbits isolated on an island.

Objectives:

To investigate how populations are affected by predator-prey relationships over several generations.

Assumptions:

1. Rabbits only die by being eaten by foxes

2. Foxes only die from natural causes.

3. The interaction between Foxes and Rabbits can be described by a function.

Populations

For Rabbit Populations if no foxes were present the new rabbit population is calculated from the previous population by the following equation.

$$P_{R_n} = P_{R_{(n-1)}} + B_R * P_{R_{(n-1)}}$$

where :

P_{R_n} is the new rabbit population after n generations

$P_{R_{(n-1)}}$ is the beginning rabbit population at generation n-1

B_R is the Birth Rate for Rabbits

(Under these conditions the rabbit population would continually increase.)

For populations if no hunters are present the new fox population is calculated from the previous population by the following equation:

$$P_{F_n} = P_{F_{n-1}} - D_F * P_{F_{n-1}}$$

where:

P_{F_n} is the new fox population after n generations

$P_{F_{n-1}}$ is the beginning fox population at generation n-1

D_F is the Death Rate for Foxes

(Under these conditions the fox population only will decrease.)

When foxes can eat rabbits the population equations have another term to describe the interaction. The population of Rabbits is the same as above with a loss of population due to foxes:

$$P_{R_n} = P_{R_{n-1}} + B_R * P_{R_{n-1}} - A * P_{R_{n-1}} * P_{F_{n-1}}$$

where:

A is the interaction constant between the foxes and the rabbits

The population of foxes is increased with the addition of the interaction term:

$$P_{F_n} = P_{F_{n-1}} - D_F * P_{F_{n-1}} + A * P_{F_{n-1}} * P_{R_{n-1}}$$

The Program

1. Write a function returning the population of rabbit at generation N given an initial population of rabbits, a rabbit birth rate and no fox interaction.

2. Write a function returning the population of foxes at generation N given an initial population of foxes, a fox death rate and no rabbit interaction.

3. Same as 1) and 2) but with a given interaction ***A*** between foxes and rabbits.

Advanced *(outside the scope of the module)*
Draw a population graph using the functions developed in 3) and the module matplotlib. Try several values for each parameter to model population interaction over time.

MatplotLib resources:

- matplotlib module webpage
- Link to external resources from MatplotLib webpage
- matplotlib tutorial