1 Top down design

Top Down Design is a problem solving technique where you:

1. Start with general description of problem
2. Break it into several high-level steps
3. Iteratively break the steps into smaller steps until you have steps that are easy to solve.

It is similar to writing a paper where you start with an outline, then fill in the second-level details, and so on until you can start writing each section/function.

Whether or not you practice top down design, your code should look like it was written using top down design. In other words, the code should have structure. Almost always, that structure is hierarchical: some functions carry out “high level” tasks and other functions are helper functions that take care of “low level” details.

The same is true for writing. You may or may not write your essay by starting with an outline. However, given the final essay, it should be easy to extract an outline from it.

Programs that are written using top-down design tend to have functions that are SOFA and DRY (which stands for Don’t repeat yourself). If you don’t use top-down design, you still must meet these criteria!

2 SOFA

In well-designed programs, each function meets the SOFA criteria:

- Short: more than 10-15 lines is probably too much
- does One thing: if the function does more than one task, break it down into smaller functions.
- takes Few parameters: Alan Perlis, a famous computer scientist said, “If you have a [function] with 10 parameters, you probably missed some.”
- maintains a single level of Abstraction: a function should focus on either high-level or low-level, and not try to do both.

The first three criteria are straightforward. The A needs some additional explanation. Here are some examples from the last homework (Higher or Lower Game).

- The main and play_game functions are high level. They focus on game logic. It’s quite easy even for a non-programmer to read it and understand exactly what is happening.
- The shuffle function is low level. The function’s purpose is narrow: to return a lists of the cards in random order. It would be confusing to someone who hasn’t taken 101. However, the docstring should be clear enough that someone can understand what your program does without having to read the body of shuffle.
3 One more design tip

Test your code often. There are two ways to test it.

- Run the entire main program and check that the latest feature you added works correctly. For example, with higher or lower, to test shuffle I could play the entire game many times, making sure the cards on the board are different each time.

- Comment out the call to the main program and “Run Module” in IDLE. This will effectively import your code. Try out individual functions in the interpreter. For example, I can focus exclusively on shuffle by trying it out in the interpreter.

```python
>>> shuffle()
```