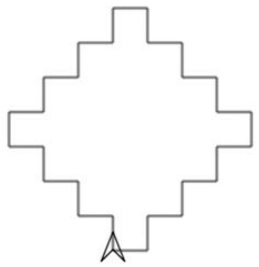
Name(s)_______________________________________ Period ______ Date __________________

Worksheet - Top-Down Design

Functions and Managing Complexity



In the previous lesson, we saw how we could **use functions to manage the complexity of a programming task**. We identified repeated patterns in the diamond that could be programmed as individual functions and then recombined these building blocks to make the full figure. In that example, the problem was broken down for you into layers of functions (and abstraction), but going forward you'll be designing these layers of functions yourself. The question is, "How?"

Top-Down Problem Solving

One method for approaching programming problems is called "Top-Down Design" or "Stepwise Refinement". This strategy is an informal way of repeatedly dividing a system into simpler subsystems. As these pieces of the original problem get smaller and smaller, eventually you will arrive at pieces that you can program solutions to in a straightforward manner. These pieces of code can then be recombined to solve your original, complex problem. As a result you can eventually solve a large problem, but along the way you only ever had to address smaller ones. Another benefit of this strategy is that it can help you choose names for the different functions you'll design. Here's an example of how we used this strategy to develop our solution to the diamond problem.

Top-Down Design Strategy	**Talking through the problem...**	**Function Name**
Look at the big picture...	"Well, I need to draw a diamond…"	drawDiamond()
Identify a sub-task...	"...the diamond has 4 sides I need to draw…"	drawSide()
Break down that sub-task into smaller sub-task(s)...	"...and each side is really 3 zig-zag steps…"	drawStep()
Keep going until you're down to the commands you already have access to.	"...each step is just forward-left-forward-right…"	Stop; this is simple enough to begin programming.

Writing the Code

Once you've identified the layers of functions you want to design, you begin by writing and testing the bottom layer first and work your way up. Higher layers of functions depend on lower layers working perfectly, since they use them without needing to know the exact details of how they are programmed. The benefit of having these **layers of abstraction** is that it lets you approach increasingly complex problems in an organized way. This typically **makes reasoning about your code much easier**, and your code will read almost like *you are describing your solution to the problem in plain English.*

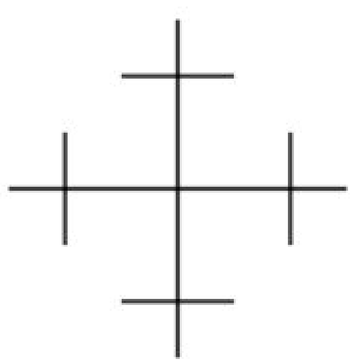
Personal Expression in Programming

The solution you were presented to the diamond problem is only one of many possible solutions. In fact, when you originally saw that figure, you may have seen a postage stamp or two pyramids stacked on one another. There are always innumerable ways to break the problem into subproblems, name the associated functions, and program their behavior. **How *you* decide to write a program is a reflection of the way *you* think and approach problems** and no two people will always do it exactly identically. Programming is a form of personal expression as unique to you as any other way you might express yourself.

Practice with Top-Down Design

Let's practice using Top-Down Design by breaking down a simple problem.

- With a partner look at the figure that you need to draw.
- Talk through the problem starting from the big picture and identifying sub-tasks.
- Decide the names of the functions you would write to solve this problem.
- Use the space provided to do scratch work.
- Iterate on your ideas; as you discuss the problem you might change your mind about the approach, or come up with better, more descriptive, function names.

The problem: Write a program for a turtle to draw this figure	**Functions you would write**
	

Compare with Another Group

Trade your solution with another group. Did you break down the problem in the same way? Do your functions have the same names?