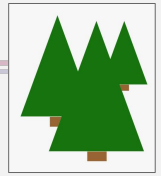
Functions

Abstraction and Modularity



Two motivating factors –

- laziness
  - it would be nice to draw a forest without having to write `rect(…)` and `triangle(…)` for every tree

    (perhaps we could define how to draw one tree, and then place a bunch of trees)
- feasibility
  - making a complex scene gets very difficult if you have to think about the whole thing at the levels of rects and triangles and ellipses

    (perhaps we could define how to draw a tree and a car and a house, and then position the tree and car and house to make the scene)

Abstraction and Modularity

- abstraction
  - be able to think of complex things in terms of higher level concepts, instead of only as their component parts
  - example
    - separate the arrangement of trees and cars in the sketch from the details of how to draw a tree or a car – think of the scene as an arrangement of trees and cars, and a tree or a car as rects, ellipses, triangles
  - goal is simplifying complexity
- modularity
  - create distinct components that can be used in a variety of situations
  - example
    - create a “tree” module so you can have a scene with lots of trees instead of repeating the individual drawing commands over and over
  - goals are to help with abstraction and to facilitate reuse

Functions

Functions (also known as *procedures* or *subroutines*) are a way of creating modules that do tasks.

- a programmer-defined list of instructions given a name

Functions generally have one of two roles –

- do stuff
- compute a value for use elsewhere in the program

We will consider specifically “do stuff” functions to draw things.

Create a drawing function if –

- a thing consists of more than a few shapes
- there is more than one copy of a thing (variations OK)

Functions

There are two parts to working with functions –

- a *function definition* associates a name with a list of instructions
  - "hey computer, here's what this name means, OK?"

```
void setup () { … }
void draw () { … }
void mouseClicked () { … }
```

  - *parameters* allow the function definition to be a template into which different values can be plugged (similar to variables)
- a *function call* tells the system to actually execute those statements
  - "hey computer, do that stuff now!"
  - the call provides values for the parameters

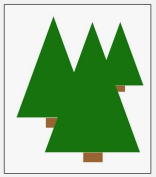
```
rectMode(CENTER);
fill(255,0,0);
stroke(0);
rect(100,200,50,100);
```


Drawing Function Questions, Part 1

Do we need a drawing function?

Yes, if

- a thing consists of more than a few shapes
- there is more than one copy of a thing (including variations)



Drawing Function Questions, Part 2

For the drawing function definition –

- put the definition outside other function definitions

- What is being drawn? → function *name* (and comment)
  - e.g. tree, car, …
  - just one purpose!
- What differs from one copy to the next? → *parameters*
  - also consider future flexibility (within reason) – there might be only one copy now, but maybe you are thinking about some extra credit enhancements…
- How is it drawn? → function *body*
  - the drawing commands – include all necessary state (`rectMode`/`ellipseMode`, `stroke`, `fill`, etc) as well as the shapes
  - scope – can use parameters, system variables
    - it is legal to use animation variables but better to use parameters instead

Defining a Drawing Function

What is being drawn? - a tree

```
// draw a tree, centered, with its base at
// the bottom edge of the window
void drawTree () {
  ...
}
```

name of the function
- must be different from other names being used
- should be descriptive of function's purpose

comments describe what the function does (draw a tree in a certain place)

Comments are essential for abstraction – they must describe everything someone needs to know to use the function without seeing the body.

Defining a Drawing Function

```
// draw a tree, centered, with its base at
// the bottom edge of the window
void drawTree () {
  rectMode(CENTER);
  stroke(0);

  // trunk
  fill(144,63,0);
  rect(width/2,height-20,15,40);

  // tree top
  fill(21,95,16);
  triangle(width/2-30,height-40,
           width/2,height-160,
           width/2+30,height-40);
}
```

How is it drawn?

function body – what the function does

Drawing Function Questions, Part 3

For the function call(s) –

- When should the task be performed? → function *call*
  - put the call where you want it to happen
- What are the specific values for the things that can vary? → *arguments*

Calling a Drawing Function

```
void setup () {
  size(400,400);
}
void draw () {
  background(0,0,255);

  // grass
  rectMode(CORNER);
  fill(0,255,0);
  stroke(0,255,0);
  rect(0,height-100,width,100);

  // tree
  drawTree();
}
```