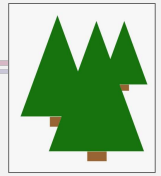
Abstraction and Modularity – Functions

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* or *methods*) are a way of creating modules that do tasks.

- a list of instructions given a name

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?”
  - *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

Functions

We have been using system-defined functions already –

```
rectMode(CENTER);
fill(255,0,0);
stroke(0);
rect(100,200,50,100);
x = x+random(-5,5);
y = map(noise(t),0,1,100,200);
```

function calls

We have also written function definitions for functions with a special role –

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

function definitions

Programmer-Defined Functions

Functions generally have one of two jobs –

- do stuff
  - e.g. `size`, `background`, `fill`, `stroke`, `rectMode`, `ellipseMode`, `rect`, `ellipse`, …
    - written as a statement by themselves
- compute a value for use elsewhere in the program
  - e.g. `random`, `noise`, `sin`, `cos`, `max`, `min`, …
    - written as an expression (or part of an expression) in a place where a value is needed

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

Programmer-Defined Functions

Many functions are defined by the system or provided by libraries.

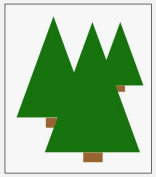
You can also define your own.

Drawing Function Questions, Part 1

Do we need a drawing function?

Yes, if –

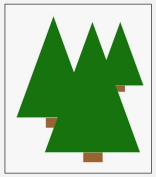
- a thing consists of more than a few shapes
- there is more than one copy of a thing (including variations) in a single frame or over a series of frames (animation or interaction)



For each drawing function identified, we need to consider both the function definition and the function call(s).

Drawing Function Questions, Part 2

For the drawing function definition –



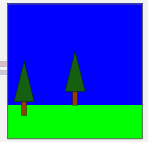
- What is being drawn?
  - e.g. tree, car, …
  - just one purpose!
- What differs from one copy to the next?
  - position, size, color, …
  - consider both multiple copies within one frame and "copies" over multiple frames due to animation or interaction
- How is it drawn?
  - include all necessary state (`rectMode`/`ellipseMode`, `stroke`, `fill`, etc) as well as the shapes to be drawn
    - draw a picture and label the necessary elements
  - can use system variables and "what differs"
    - it is legal to use animation variables but better to view that as "what differs" instead

Drawing Function Questions, Part 3

For the function call(s) –

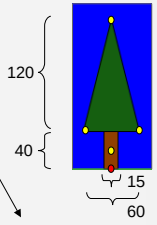
- What are the specific values for the "what differs" things?

Example



Consider the two trees in the scene.

- Do we need a drawing function?
  - → yes, each tree has two shapes and there's more than one tree
- What is being drawn?
  - → a tree
- What differs from one copy to the next?
  - position (both x and y)
- How is it drawn?
  - dark green triangle, brown rectangle
  - let "position" be the center of the bottom of the trunk (red dot)
- What are the specific values for the "what differs" things?
  - tree on the left: x: 50, y: `height`-70
  - tree in the middle: x: `width`/2, y: `height`-100



position (red dot) – (x,y)
rect center – (x,y-20)
triangle – (x-30,y-40), (x,y-160), (x+30,y-40)