Functional Abstraction Reduces Complexity

Layering: Building Functions out of Functions

Lawrence Snyder
University of Washington, Seattle
Plan For Today

- Today – the two threads of class merge again as we introduce functions in Processing and use them in a layering technique to build a timer
  - Introduce Functions
  - Draw digital timer elements
  - Assemble elements into digits
  - Light digit segments to create numbers
  - Select number based on a digit
Functions, A Review

- Functions have been used in Lightbot 2.0: F1
- Functions were in Assignment 03: F.turn( ) ...
- We’ve used functions, also known as
  - procedures
  - methods
  - subroutines

in all of our Processing code: `size(200, 200)`

- Recall that functions have two parts:
  - function definition ... a statement of how it works
  - function call ... a request to have it performed
Functions In Processing

- Form of function definition in Processing
  
  \[<\text{return type}> \ <\text{name}> \ ( \ <\text{param list}> \ ) \ { \]
  
  \[<\text{body}> \]
  
  \}

as in

```java
void draw_a_box (int x_pos, int y_pos) {
  rect(x_pos, y_pos, 20, 20);
}
```

or

```java
color pink ( ) {
  return color(255, 200, 200);
}
```
Functions In Processing: Result

- Functions that do something, but do not return a value, have `void` as their `<return type>`
- Functions that return a value must say its type

```plaintext
void draw_a_box (int x_pos, int y_pos) {
  rect(x_pos, y_pos, 20, 20);
}

color pink () {
  return color(255, 200, 200);
}
```
Functions In Processing: Params

- Parameters are the values used as input to the function; parameters are not required, but the parentheses are.
- The type of each parameter must be given.

```cpp
void draw_a_box (int x_pos, int y_pos) {
    rect(x_pos, y_pos, 20, 20);
}

color pink () {
    return color(255, 200, 200);
}
```
A function returns its value with the `return` statement ... the stuff following return is the result

The function is done when it reaches return

```cpp
void draw_a_box (int x_pos, int y_pos) {
    rect(x_pos, y_pos, 20, 20);
}

color pink ( ) {
    return color(255, 200, 200);
}
```
Writing Functions

- Processing function definitions are typically listed after the standard blocks: setup(), draw(), mousePressed(), etc.

```java
void setup() {
  size(100, 100);
  background(0);
  noStroke();
}

void draw() {
  fill(255);
  hexa(20, 20);
}

void hexa(float xbase, float ybase) {
  rect(xbase, ybase+10, 20, 40);
  triangle(xbase, ybase+10, xbase+20, ybase+10, xbase+10, ybase);
  triangle(xbase, ybase+50, xbase+20, ybase+50, xbase+10, ybase+60);
}
```
Using Functions

- Once defined, functions can be called repeatedly … it’s the point of writing them!

```cpp
void setup() {
  size(110, 100);
  background(0);
  noStroke();
}

void draw() {
  fill(255);
  hexa(20, 20);
  hexa(50, 20);
  hexa(80, 20);
}

void hexa(float xbase, float ybase) {
  rect(xbase, ybase+10, 20, 40);
  triangle(xbase, ybase+10, xbase+20, ybase+10, xbase+10, ybase);
  triangle(xbase, ybase+50, xbase+20, ybase+50, xbase+10, ybase+60);
}
```
Arguments Become Parameters

- Notice that if the DEFINITION has \( n \) parameters, the CALL needs \( n \) arguments.
- The parameters and arguments correspond.

```c
void draw( ) {
    fill(255);
    hexa(20, 40);
    hexa(50, 40);
    hexa(80, 40);
}

void hexa(float xbase, float ybase) {
    rect(xbase, ybase+10, 20, 40);
    triangle(xbase, ybase+10, xbase+20, ybase+10, xbase+10, ybase);
    triangle(xbase, ybase+50, xbase+20, ybase+50, xbase+10, ybase+60);
}
```

Inside of the function, the parameter, e.g. `xbase`, is declared and initialized to the corresponding argument, e.g. 80. Then, the definition uses it, e.g. `rect(80, 40+10, 20, 40)`.
Parameters

- Parameters are automatically declared (and initialized) on a call, and remain in existence as long as the function remains unfinished.
- When the function ends, the parameters vanish, only to be recreated on the next call.
- It is wise to choose parameter names, e.g. x-b-a-s-e that are meaningful to you.
  - I chose xbase as the orientation point of the figure in the x direction.
  - Notice that I used that name a lot, and the meaning to me remained the same.
- Draw digital timer elements
- Assemble elements into digits
- Light digit segments to create numbers
- Select number based on a digit
Define hexa() and rexa()

- Patter: Parameterize the functions by a consistent position – upper left corner is good

```c
void draw( ) {
    fill(255);
    hexa(20, 40);
    rexa(30, 20);
}

void hexa(float xbase, float ybase) {
    rect(xbase, ybase+10, 20, 40);
    triangle(xbase, ybase+10, xbase+20, ybase+10, xbase+10, ybase);
    triangle(xbase, ybase+50, xbase+20, ybase+50, xbase+10, ybase+60);
}

void rexa(float xbase, float ybase) {
    triangle(xbase, ybase+10, xbase+10, ybase, xbase+10, ybase+20);
    rect(xbase+10, ybase, 40, 20);
    triangle(xbase+50, ybase, xbase+50, ybase+20, xbase+60, ybase+10);
}
```
void draw() {
    fill(255);
    digit(50, 20);
    digit(140, 20);
}

void hexa(float xbase, float ybase) {
    rect(xbase, ybase+10, 20, 40);
    triangle(xbase, ybase+10, xbase+20, ybase+10, xbase+10, ybase);
    triangle(xbase, ybase+50, xbase+20, ybase+50, xbase+10, ybase+60);
}

void rexa(float xbase, float ybase) {
    triangle(xbase, ybase+10, xbase+10, ybase, xbase+10, ybase+20);
    rect(xbase+10, ybase, 40, 20);
    triangle(xbase+50, ybase, xbase+50, ybase+20, xbase+60, ybase+10);
}

void digit(float xbase, float ybase) {
    hexa(xbase, ybase+10); //left upper
    hexa(xbase, ybase+70); //left lower
    rexa(xbase+10, ybase); //mid horizontal
    rexa(xbase+10, ybase+60); //top horizontal
    rexa(xbase+10, ybase+120); //bot horizontal
    hexa(xbase+60, ybase+10); //right upper
    hexa(xbase+60, ybase+70); //right lower
}
Let There Be Light (and Dark)

- Define the illumination of the digit
  - Must declare two color variables, initialize to proper colors, use them in fill, and check ‘em

```cpp
color dark, lite;

void setup( ) {
  size(250, 180);
  background(0);
  stroke(0);
}

void draw( ) {
  lite = color(255, 185, 0);
  dark = color(64, 48, 0);

  fill(dark);
  digit(50, 20);
  fill(lite);
  digit(140, 20);
}
```
Count In Lights

- Light up the digit for each number: ^C ^P

```c
void digit(float xbase, float ybase) {
    hexa(xbase, ybase+10);  // left upper
    hexa(xbase, ybase+70);  // left lower
    rexa(xbase+10, ybase);  // top horizontal
    rexa(xbase+10, ybase+60);  // mid horizontal
    rexa(xbase+10, ybase+120);  // bot horizontal
    hexa(xbase+60, ybase+10);  // right upper
    hexa(xbase+60, ybase+70);  // right lower
}

void one (float xbase, float ybase) {
    hexa(xbase+60, ybase+10);  // right upper
    hexa(xbase+60, ybase+70);  // right lower
}

void two (float xbase, float ybase) {
    rexa(xbase+10, ybase);    // top horizontal
    rexa(xbase+10, ybase+60);  // mid horizontal
    rexa(xbase+10, ybase+120);  // bot horizontal
    hexa(xbase+60, ybase+10);  // right upper
    hexa(xbase, ybase+70);     // left lower
}
```
Select A Number To Display

- Given an integer, display it in lights

```c
void sel(int n, float xbase, float ybase) {
    fill(lite);
    if (n == 0) {
        zero(xbase, ybase);
    }
    if (n==1) {
        one(xbase, ybase);
    }
    if (n==2) {
        two(xbase, ybase);
    }
    if (n==3) {
        three(xbase, ybase);
    }
    if (n==4) {
        four(xbase, ybase);
    }
    if (n==5) {
        five(xbase, ybase);
    }
    if (n==6) {
        six(xbase, ybase);
    }
    ...
```
Create a 3 Digit Display

```c
void three_digit(int n, float xbase, float ybase) {
    fill(dark);
    digit(50, 90);
    digit(140, 90);
    digit(260, 90);
    fill(lite);
    rect(xbase+185, ybase+125, 15, 15);
    sel(((n/100)%10, xbase, ybase);
    sel(((n/10)%10, xbase+90, ybase);
    sel(n%10, xbase+210, ybase);
}
```

Here’s The Action
Count up At The Frame Rate

color dark, lite;
int i;

void setup() {
    size(400, 300);
    background(0);
    noStroke();
    frameRate(10);
}

void draw() {
    lite = color(255, 185, 0);
    dark = color(64, 48, 0);
    i = i + 1;
    three_digit(i, 50, 90);
}
Review What We Did

The computation is ONLY drawing triangles and rectangles, but we don’t think of it that way ... to us, it’s a timer