Datatypes and Functions

Lawrence Snyder
University of Washington, Seattle
Two ideas – data types and functions – are already familiar to you, because you’ve been using them.

Today, we teach their details:
- Data types
- Functions

Also, today, we’ll cover some handy “tricks” using those ideas.
Information has certain properties ... we group information with similar properties into “types” --
- integers, or whole numbers
- floating point, usually called decimal numbers
- colors, a triple of numbers for R, G and B
- Etc.

In order for computers to process data, they need to know its type
- So, we always specify the data’s type
Give Datatypes in Declarations

- Processing has a series of datatypes
- The most important datatypes for us are int, float, boolean and color
  - Find details in the references
- When declaring variables we list them after the type, as in
  - int x, y, z;
  - float half_step = 0.5, whole = 1.0;
  - color yellow = color(200,200,0);
Examples: At Top of a Program

```java
int i = 0;
int L, m, eN, Oh, pea;

float myTemp = 98.6;
float waterFreeze = 32;

color red = color(0,255,0), turquoise = color(100, 231, 192);

void setup() {
    size(100,100);
    background(red);
}
void draw() {
    fill(turquoise);
    rect(20, 20, 20, 20);
}
```
At The Top of Functions

```
int i = 30;

color turquoise = color(100, 231, 192);

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

void draw() {
    int inside = 30;
    fill(turquoise);
    rect(20, 20, 20+inside, 20+i);
}
```
Global Variable Preserve Info

```java
int i = 30;

color turquoise = color(100, 231, 192);

void setup() {
    size(100, 100);
    background(0);
    i = i / 3;
}

void draw() {
    int inside = 10;
    fill(turquoise);
    rect(20, 20, 20 + i, 20 + inside);
}
```
Hiding In Another Function ...

```cpp
int i = 30;

color turquoise = color(100, 231, 192);

void setup() {
    int inside = 30;
    size(100,100);
    background(0);
}
void draw() {
    fill(turquoise);
    rect(20, 20, 20+inside, 20+i);
}
```

---

Cannot find anything named “inside”
Functions, A Review

- Functions have been used in Lightbot 2.0: F1
- Functions in HW 03: F.turn( ), HW 05: Cols ...
- 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 ... declaration of its instructions
  - function call ... a request to run the function
- Let’s get the details down now ...
Functions In Processing

- The form of function definition in Processing
  
  `<return type> <name> ( <param list> ) { 
  
  <body>
  
  }

  }

  as in

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

  or

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

- The form of function definition in Processing

  \[
  \langle \text{return type} \rangle \ \langle \text{name} \rangle \ ( \ \langle \text{param list} \rangle \ ) \ \{ \\
  \langle \text{body} \rangle \\
  \}
  \]

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 that do something, but do not return a value, have **void** as their *return type*.
- Functions that return a value must say the datatype of the value returned.

```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);
}
```
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);
}
```
Functions In Processing: Args

- An argument is the input value given to a parameter when a function is called, as in `draw_a_box(50, 200);`
- The value of the argument becomes the value of the corresponding parameter:
  - `draw_a_box(50, 200);`
  - `void draw_a_box (int x_pos, int y_pos) {
    rect(x_pos, y_pos, 20, 20);
  }`
Functions In Processing: Return

- 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);
}
```
Plan The Function in Declaration

- The function has a name, parameters, def
  - Name: `drawColumn`
  - Parameters: `int offset`;
  - Definition:
    ```
    rect(20+offset, 250, 60, 20);
    rect(30+offset, 270, 40, 10);
    ellipse(30+offset, 275, 10, 10);
    ellipse(70+offset, 275, 10, 10);
    rect(35+offset, 280, 30, 60);
    ```
  - Nothing has to be returned
The Function Declaration & Calls

- The result:

```c
void drawCol( int offset) {
    rect(20+offset, 250, 60, 20); // Top stone
    rect(30+offset, 270, 40, 10);  // Stone below it
    ellipse(30+offset, 275, 10, 10); // Left curl
    ellipse(70+offset, 275, 10, 10); // Right curl
    rect(35+offset, 280, 30, 60);  // Actual column
}
```

- The calls

```c
fill(255);
drawCol(0);
drawCol(100);
drawCol(200);
drawCol(300);
}```
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. o-f-f-s-e-t that are meaningful to you.
  - I chose offset 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.
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. \( x_{\text{base}} \), is declared and initialized to the corresponding argument, e.g. 80. Then, the definition uses it, e.g. \( \text{rect}(80, 40+10, 20, 40) \)
Using Functions + Global Variables

```cpp
int x = 0;
int dir = 1;

void setup () {
    size(400,100);
    noStroke();
}

void draw() {
    background(200);
    fill(100, 255, 100);
    rect(20+x, 20, 50, 50);
    x = x+dir;
}

void mousePressed() {
    dir = 0 - dir;
}
```

Mouse clicks control the direction ... dir flips from 1 to -1 on each click
Summary on Functions

- When we have something we have to produce and perhaps use repeatedly, we need a function
- We need two things: a declaration, and calls
  - The declaration is the function “package”
  - The call is the use, where we ask to “run” it
- So we declared the function, and called it
  - Need to follow the rules on form: “void”, parens, give type of parameters, curly braces, indent, comment, ...
  - Use parameter values where changes are needed