Datatypes and Variables

Lawrence Snyder
University of Washington, Seattle
Today’s Goals

- We have three basic ideas to cover –
  - Datatypes
  - Declarations
  - Variables

- They all interact ... we’ll just start on these ideas today
void setup() {
    size(500,500);
    noStroke();
}

void draw() {
    background(255, 245, 220);
    raff();
}

void raff() {
    fill(0,100,0);
    rect(240, 260, 40, 45);
    fill(219,136,0);
    rect(240, 210, 40, 50);
    fill(0,100.0);
    rect(240, 190, 40, 20);
    fill(255,0,0);
    rect(240, 184, 40, 6);
    fill(0,100,0);
    rect(240, 169, 40, 15);
}
Variables

- **variables** are names used in a program for quantities that vary ... get it? Variables vary!
- So, one thing we can do is give them values:
  - \( x = 12; \)
    - \( x \) is the variable, and it’s being given the value 12
- Now, whenever I use the variable \( x \), as in
  - \( y = x + 1; \)
    - it is as if I had used its value (12) directly: \( y=12+1 \)
- It’s pretty obvious ... but there’s more to it

Caution: variables are NOT unknowns
The data that variables name 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.
Give Datatypes in Declarations

- Processing has a largish set of **datatypes**
- The most important datatypes for us are **int, float, boolean** and **color**

... we add more later

- Find details in the references
Tell Processing About Your Values

- Processing (and all languages) need to know the types of data you are working with
- We tell them the type by **declaring** a variable’s datatype
- 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);`
Variables are case sensitive

```c
int leftSide, left_side, leftside; // declare 3 vars
```

Variables can be initialized

```c
float temperature = 98.6; // declare & initialize
```

Variables names are meaningless to computers, but meaningful to people ... don't lie

```c
color myWhite = color(0,0,0); // White ... ha, ha!
```

Variables are best declared at top of a program
Add A Variable

- Raphael gets a var
- Adding the variable value (0) to each horizontal position results in no change

```java
int ra = 0;
void setup() {
  size(500, 500);
  noStroke();
}
void draw() {
  background(255, 245, 220);
  raff();
}
void raff() {
  fill(0, 100, 0);
  rect(240 + ra, 260, 40, 45);
  fill(219, 136, 0);
  rect(240 + ra, 210, 40, 50);
  fill(0, 100, 0);
  rect(240 + ra, 190, 40, 20);
  fill(255, 0, 0);
  rect(240 + ra, 184, 40, 6);
  fill(0, 100, 0);
  rect(240 + ra, 169, 40, 15);
}
When ra has the value of 200, Raff’s position is changed

```cpp
int ra = 200;
void setup() {
  size(500, 500);
  noStroke();
}
void draw() {
  background(255, 245, 220);
  raff();
}
void raff() {
  fill(0, 100, 0);
  rect(240 + ra, 260, 40, 45);
  fill(219, 136, 0);
  rect(240 + ra, 210, 40, 50);
  fill(0, 100, 0);
  rect(240 + ra, 190, 40, 20);
  fill(255, 0, 0);
  rect(240 + ra, 184, 40, 6);
  fill(0, 100, 0);
  rect(240 + ra, 169, 40, 15);
}```
Recall setup() and draw()

- The functions setup() and draw() allow the Processing computations to be dynamic
- Recall that they work as follows:

  ```
  setup()
  draw()
  ```

- Make Raphael run!
`int ra = -200;
void setup() {
    size(500, 500);
    noStroke();
}
void draw() {
    background(255, 245, 220);
    raff();
    ra = ra + 1;
}
void raff() {
    fill(0, 100, 0);
    rect(240 + ra, 260, 40, 45);
    fill(219, 136, 0);
    rect(240 + ra, 210, 40, 50);
    fill(0, 100, 0);
    rect(240 + ra, 190, 40, 20);
    fill(255, 0, 0);
    rect(240 + ra, 184, 40, 6);
    fill(0, 100, 0);
    rect(240 + ra, 169, 40, 15);`
Make Him Appear

- Start Raff off-screen to right, by initializing him to ... ?
- Then make him move left by ... ?
- And speed his movement up by ... ?
Note 400 is enough to hide him off screen
Subtracting moves him left
Changing ra by 2 speeds him up

```cpp
int ra = 400;

void setup() {
  size(500, 500);
  noStroke();
}

void draw() {
  background(255, 245, 220);
  raff();
  ra = ra - 2; // Add 1 to ra
}

void raff() {
  fill(0, 100, 0);
  rect(240 + ra, 260, 40, 45);
  fill(219, 136, 0);
  rect(240 + ra, 210, 40, 50);
  fill(0, 100, 0);
  rect(240 + ra, 190, 40, 20);
  fill(255, 0, 0);
  rect(240 + ra, 184, 40, 6);
  fill(0, 100, 0);
  rect(240 + ra, 169, 40, 15);
}```
Return to basic Raff, and declare five new variables of type float ... and add to vertical dimension

```c
float ua = 0;
float ub = 0;
float uc = 0;
float ud = 0;
float ue = 0;

... 

void raff() {
    fill(0,100,0);
    rect(240,260+ua, 40, 45);
    fill(219,136,0);
    rect(240,210+ub, 40, 50);
    fill(0,100,0);
    rect(240,190+uc, 40, 20);
    fill(255,0,0);
    rect(240, 184+ud, 40, 6);
    fill(0,100,0);
    rect(240, 169+ue, 40, 15);
}
```
Add Some Action!

- We want Raff to drop down
  - Translate his position by -150
  - Add 1 to each new variable
- ... but, he doesn’t stop

```cpp
float ua = -150;
float ub = -150;
float uc = -150;
float ud = -150;
float ue = -150;

void setup() {
  size(500, 500);
  noStroke();
}

void draw() {
  background(255, 245, 220);
  raff();
  ua = ua + 1;
  ub = ub + 1;
  uc = uc + 1;
  ud = ud + 1;
  ue = ue + 1;
}

...
As the value of ua, say, changes, Raff’s position changes ... fill(0,100,0);
rect(240,260+ua, 40, 40);
... ua = ua + 1;

Consider changes [position blue; ra red]

- 110 = 260+(-150)  // first time
  - -149 = -150 + 1

- 111 = 260+(-149)  // second time
  - -148 = -149 + 1

- 112 = 260+(-148)  // third time
  - -147 = -148 + 1
  ...
Continuing The Analysis

- The offset $u_a$ gets less and less negative, eventually getting to zero
  - $259 = 260 + (-1)$
    - $0 = -1 + 1$
  - $260 = 260 + 0$
    - $1 = 0 + 1$
    - ...

- We want to stop when $u_a$ gets to 0
- So, don’t do $u_a = u_a + 1$, write $u_a = \min(0, u_a + 1)$
- What happens???
  - $\min(a, b)$ gives the smaller of $a, b$
Check Out The min( ) Function

- $110 = 260 + (-150)$
  - $-149 = \min(0, -150 + 1)$ \hspace{1cm} As before!

- $111 = 260 + (-149)$
  - $-148 = \min(0, -149 + 1)$ \hspace{1cm} As before!

- $112 = 260 + (-148)$
  - $-147 = \min(0, -148 + 1)$ \hspace{1cm} As before!

  ...

- $259 = 260 + -1$
  - $0 = \min(0, -1 + 1)$ \hspace{1cm} No difference, as before!

- $260 = 260 + 0$
  - $0 = \min(0, 0 + 1)$ \hspace{1cm} Stays at $0 \ldots$ forever!
The code simply applies the `min()` function

```cpp
void draw() {
  background(255, 245, 220);
  raff();
  ua = min(ua + 1, 0);
  ub = min(ub + 1, 0);
  uc = min(uc + 1, 0);
  ud = min(ud + 1, 0);
  ue = min(ue + 1, 0);
}

void raff() {
  fill(0, 100, 0);
  rect(240, 260 + ua, 40, 45);
  fill(219, 136, 0);
  rect(240, 210 + ub, 40, 50);
  fill(0, 100, 0);
  rect(240, 190 + uc, 40, 20);
  fill(255, 0, 0);
  rect(240, 184 + ud, 40, 6);
  fill(0, 100, 0);
  rect(240, 169 + ue, 40, 15);
}
```
Best Stunt Of All: Reform

- Change the amount Raff’s parts fall so he appears to reassemble!

```java
void draw() {
    background(255, 245, 220);
    raff();
    ua = min(ua + 5, 0);
    ub = min(ub + 4, 0);
    uc = min(uc + 3, 0);
    ud = min(ud + 0.75, 0);
    ue = min(ue + 1, 0);
}
```

Requires float ud

Just Do It!
Today, we learned about

- variables ... names for quantities that vary in the program
- datatypes ... forms of data like integers, floating point numbers (decimal numbers), colors, booleans, etc.
- declarations ... statements that define what datatype variables are, as in \texttt{int ra = 0};
- And we learned the \texttt{min( )} function