

Creative Programming

Arrays and Functions

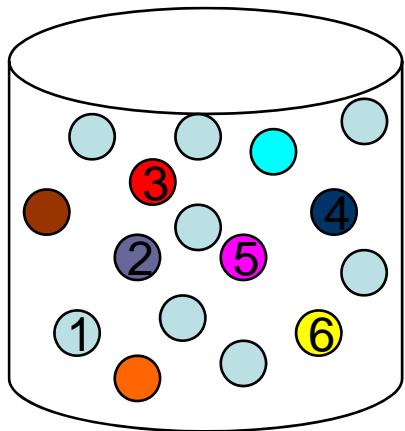
Creative Programming

Overview of this lecture

- Arrays
- Functions

Arrays Intro

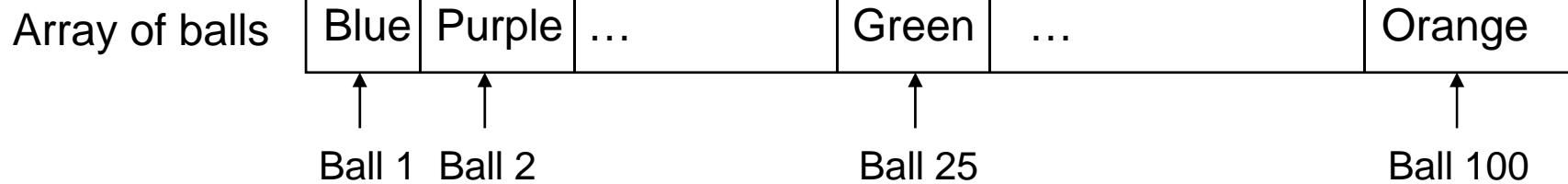
- Suppose you have a bag of balls
- For each of them you want to save their color value



```
String ball1 = "blue";  
String ball2 = "purple";  
...  
String ball100 = "orange";
```

Arrays: the Idea

- One container, datastructure to store of all them



- Compare to a desk with multiple drawers

Arrays: the Idea

- Single entity with one name
- Can hold multiple values
- Accessible through indexing
- Important datastructure in coding

Arrays: Syntax

- Declare with a datatype
String[] listofballs;
- Same declaration as a variable that holds a single string, with [] added
- More examples

int[] xpos;

float[] xspeed;

Arrays : types and size

- float, int are lowercase: primitive datatype
- String, initial cap: name of predefined class
- Can be of any size (limited to size of int datatype)
- Can hold any legal type
- Once declared, type cannot be changed

Arrays : types and size ctd.

- Example:

float[] xspeed;

- Now it can only hold values of type float
- Initialization:

xspeed = new float[100];

- “new” reserves memory
- *xspeed* now has 100 places in memory

Arrays : types and size ctd.

- Declaration and initialization can be merged into one statement:

float[] xspeed = new float[100];

- Another example:

Object[] obs = new Object[0];

Arrays: alternative initialization

int[] speeds = {2, 4, 445, -120, 3, 54};

String[] names = {"Jun", "Loe", "Peter"}

Arrays: indexing

- Start with “0”
- *int firstelement = speeds[0];*
- Length is the actual number of items in the array
- *int lenght = speeds.length;*
- Compile error when you go out of range

Looping over an array

```
String firstNames={"Rene","Karl","Shriek","Peter"};
```

```
for(int i=0;i<firstNames.length; i++){  
    println(firstNames[i]);  
    println();  
}
```

Results in?

Functions

Creative Programming

Functions: what?

- Add structure and flexibility to your programs
- Reusable blocks of code
- Has parameters and can return a value

```
void setup(){  
    size(400,400);  
    background(255)  
    drawRectangle(150, 150, 100, 100);  
}  
  
void drawRectangle(float x, float y, float w, float h){  
    rect(x, y, w, h);  
}
```

Functions: general structure

```
return_type function_name(optional parameters){  
    code to execute when function is called  
}
```

- A function starts with a return type
- An identifier (the name of the function)
- Open and closed parentheses: optional parameters
- Open and closed curly braces

Functions: Another example

```
void setup(){  
    size(400, 400);  
    background(255);  
    for(int i=0; i<100; i++)  
        drawRectangle(random(width), random  
height, random(200), random(200));  
}  
}
```

- What will happen?

Functions: strengths

- Use of different parameters
- Efficient by avoiding redundant code
- Freed from writing linearly code
- Once a function is defined, you can call it whenever you need it

Functions: parameters & arguments

- Parameters should match arguments passed to function

```
void myFunction(int x, int y){}
void myFunction(int x, int y, int z){}
void myFunction(float x, float y, float z){}
```

```
myFunction(2, 5);
myFunction(2, 5, 7);
myFunction(2.0, 3.4, 2.33);
```

Functions: return

- Each function that has a return type other than void should have a return statement as the last line

Functions: do not use hard coded numbers

- Does not facilitate evolution and flexibility of code

Bad:

```
size(200,200);  
background(255);  
strokeWeight(5);  
stroke(20);  
fill(100);  
rect(50, 50, 100, 100);
```

Functions: do not use hard coded numbers, ctd.

- Use parameterized function

```
void setup(){  
    size(200,200);  
    background(255);  
    createRect(50, 50, 100, 100, 20, 5, 100);  
}  
// parameterized function  
void createRect(int xpos, int ypos, int wdth, int ht, int strokeCol, int strokeWt,  
int fillCol) {  
    stroke(strokeCol);  
    ...  
    rect(xpos, ypos, wdth, ht);  
}
```