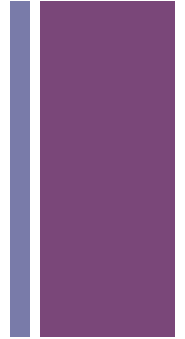


Word Clouds

+ Inheritance



- **Superclass (base class)** – higher in the hierarchy
- **Subclass (child class)** – lower in the hierarchy
- A subclass is **derived from** from a superclass
- Subclasses **inherit** the **fields** and **methods** of their superclass.
 - I.e. subclasses automatically "**get**" stuff in superclasses
- Subclasses can **override** a superclass method by redefining it.
 - They can replace anything by redefining locally

```

// Ellipse base class
class Ellipse {

    float X;
    float Y;
    float W;
    float H;

    // Ellipses are always red
    color fillColor =
        color(255,0,0); }

    Ellipse(float X, float Y,
            float W, float H)
    {
        this.X = X;
        this.Y = Y;
        this.W = W;
        this.H = H;
    }

    void draw() {
        ellipseMode(CENTER);
        fill(fillColor);
        ellipse(X, Y, W, H);
    }
}

```

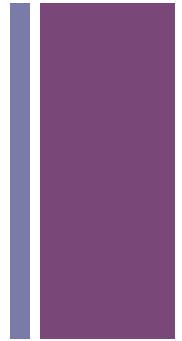
```

// Circle derived class
class Circle extends Ellipse {

    Circle(float X, float Y,
           float D) {
        super(X, Y, D, D);

        // Circles are always green
        fillColor = color(0,255,0);
    }
}

```

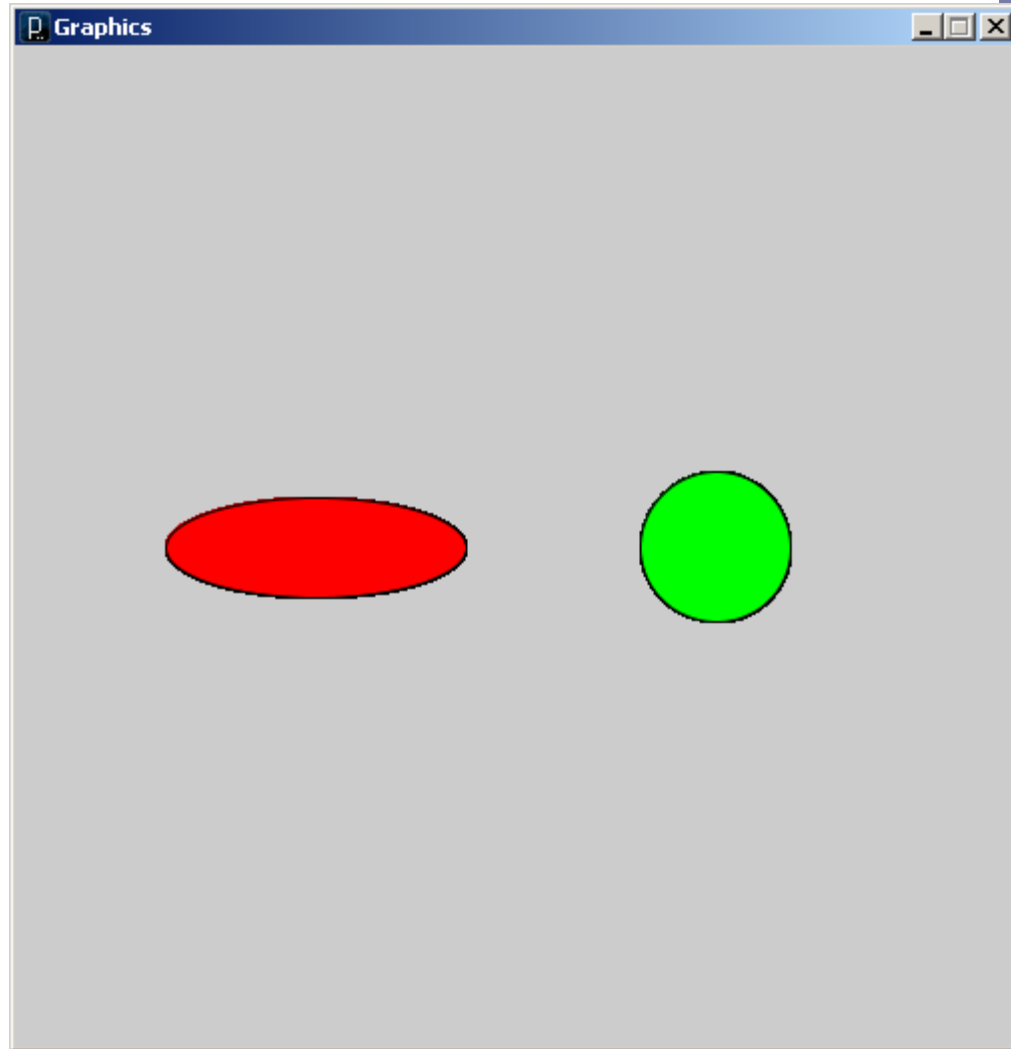


- The **extends** keyword creates hierarchical relationship between classes.
- The Circle class gets all fields and methods of the Ellipse class, automatically.
- The **super** keyword refers to the base class in the relationship.
- The **this** keyword refers to the object itself.

```
+ // Graphics
  Ellipse e = new Ellipse(150, 250, 150, 50);
  Circle c = new Circle(350, 250, 75);

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

void draw() {
  e.draw();
  c.draw();
}
```



Graphics.pde

```

// Graphics2
Ellipse[] e = new Ellipse[20];

void setup() {
  size(500, 500);
  smooth();

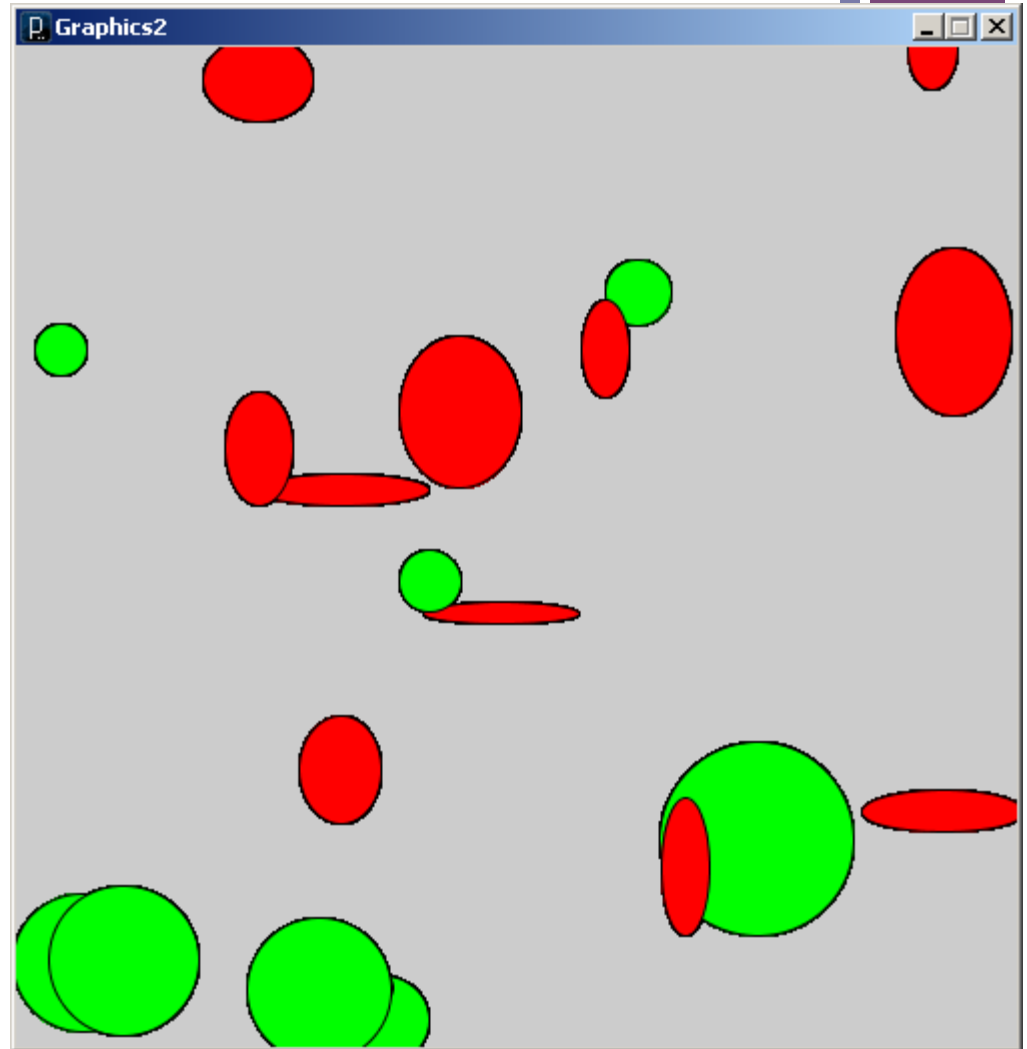
  for (int i=0; i<e.length; i++) {

    float X = random(0, width);
    float Y = random(0, height);
    float W = random(10, 100);
    float H = random(10, 100);

    // Ellipses are Circles are
    // stored in the same array
    if (random(1.0) < 0.5)
      e[i] = new Ellipse(X,Y,W,H);
    else
      e[i] = new Circle(X,Y,W);
  }
}

void draw() {
  for (int i=0; i<e.length; i++)
    e[i].draw();
}

```



Ellipses and Circles in the same array! Graphics2.pde

```

// Ellipse base class
class Ellipse {

  float X;
  float Y;
  float W;
  float H;

  // Ellipses are always red
  color fillColor =
    color(255,0,0);

  Ellipse(float X, float Y,
    float W, float H)
  {
    this.X = X;
    this.Y = Y;
    this.W = W;
    this.H = H;
  }

  void draw() {
    ellipseMode(CENTER);
    fill(fillColor);
    ellipse(X, Y, W, H);
  }

  // Do nothing
  void mousePressed() {}
}

```

```

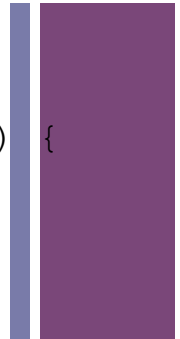
// Circle derived class
class Circle extends Ellipse {

  Circle(float X, float Y, float D) {
    super(X, Y, D, D);

    // Circles are always green
    fillColor = color(0,255,0);
  }

  // Change color of circle when clicked
  void mousePressed() {
    if (dist(mouseX, mouseY, X, Y) < 0.5*W)
      fillColor = color(0,0,255);
  }
}

```



- The mousePressed behavior of the Circle class **overrides** the default behavior of the Ellipse class.



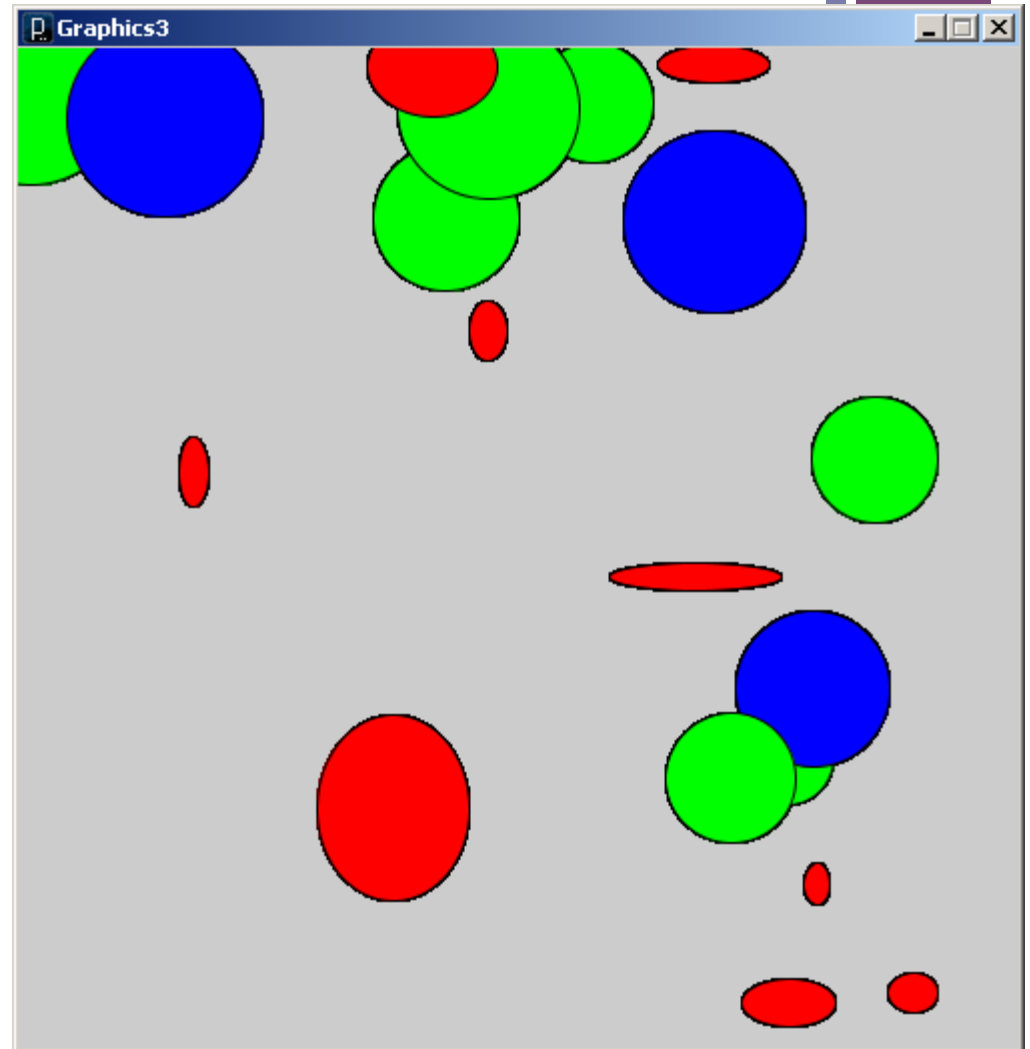
```
// Graphics3
Ellipse[] e = new Ellipse[20];

void setup() {
  size(500, 500);
  smooth();

  // Stuff removed ...
}

void draw() {
  for (int i=0; i<e.length; i++)
    e[i].draw();
}

void mousePressed() {
  for (int i=0; i<e.length; i++)
    e[i].mousePressed();
}
```



+

What is a word cloud?



Source:

http://www.huffingtonpost.com/2013/09/01/1100-words-to-describe-your-summer00-words-to-describe-you_n_3853071.html

+ Text Processing

Data Visualization Process

- Acquire - Obtain the data from some source
- Parse - Give the data some structure, clean up
- Filter - Remove all but the data of interest
- Mine - Use the data to derive interesting properties
- Represent - Chose a visual representation
- Refine – Improve to make it more visually engaging
- Interact - Make it interactive

Text Visualization

- Source = Document
- Parse = Words
- Filter = Word Set with counts
- Mine = Get relevant words
- Represent = Fonts/Placement
- Refine/Interact



What's a string?

Characters enclosed by double quotes

```
"this is a String"
```

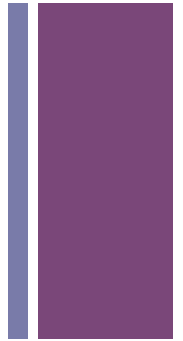
```
"  this String starts with spaces"
```

```
"12345"
```

```
"the above String is made up of digit characters"
```

Print Strings to the Console using println()

```
println( "The mouse was pressed" );
```



+ Strings are Objects

Defined using a class

Have fields, methods, one or more constructors

String objects hold an array of 'chars'

What's a char?

- A character enclosed by single quotes ('A')

```
String msg = "I Love CS 110!";
```

msg

0	1	2	3	4	5	6	7	8	9	10	11	12	13
'I'	' '	'L'	'o'	'v'	'e'	' '	'C'	'S'	' '	'1'	'1'	'0'	'!'

+ Making Strings

- Declaring String objects with no chars

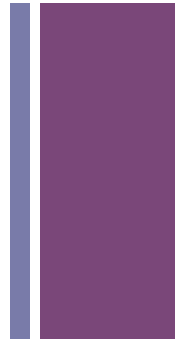
```
String myName;
```

```
String myName = new String();
```

- Declaring String objects init'd w/ char array

```
String myName = "Dianna";
```

```
String myName = new String("Dianna");
```





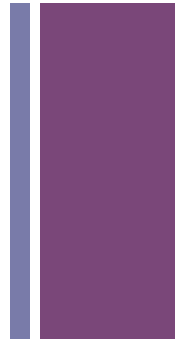
Chars are encoded by bytes

ASCII

- *American Standard Code for Information Interchange*
- An early character encoding standard
- glyph \leftrightarrow byte mapping
- 127 characters
- Forms the basis of new encoding standards
- Unicode: more than 109,000 characters covering 93 scripts

Note:

- Numbers are different than the digit characters
- Includes special characters and punctuation

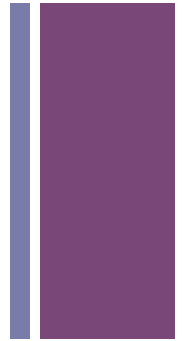




Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec
(nul)	0	(dc4)	20	(40	<	60	P	80	d	100	x	120
(soh)	1	(nak)	21)	41	=	61	Q	81	e	101	y	121
(stx)	2	(syn)	22	*	42	>	62	R	82	f	102	z	122
(etx)	3	(etb)	23	+	43	?	63	S	83	g	103	{	123
(eot)	4	(can)	24	,	44	@	64	T	84	h	104		124
(enq)	5	(em)	25	-	45	A	65	U	85	i	105	}	125
(ack)	6	(sub)	26	.	46	B	66	V	86	j	106	~	126
(bel)	7	(esc)	27	/	47	C	67	W	87	k	107	(del)	127
(bs)	8	(fs)	28	0	48	D	68	X	88	l	108		
(ht)	9	(gs)	29	1	49	E	69	Y	89	m	109		
(nl)	10	(rs)	30	2	50	F	70	Z	90	n	110		
(vt)	11	(us)	31	3	51	G	71	[91	o	111		
(np)	12	(sp)	32	4	52	H	72	\	92	p	112		
(cr)	13	!	33	5	53	I	73]	93	q	113		
(so)	14	"	34	6	54	J	74	^	94	r	114		
(si)	15	#	35	7	55	K	75	_	95	s	115		
(dle)	16	\$	36	8	56	L	76	`	96	t	116		
(dc1)	17	%	37	9	57	M	77	a	97	u	117		
(dc2)	18	&	38	:	58	N	78	b	98	v	118		
(dc3)	19	'	39	;	59	O	79	c	99	w	119		

+ String class methods

- `charAt(index)`
 - Returns the character at the specified index
- `equals(anotherString)`
 - Compares a string to a specified object
- `equalsIgnoreCase(anotherString)`
 - S/A ignoring case (i.e. 'A' == 'a')
- `indexOf(char)`
 - Returns the index value of the first occurrence of a character within the input string
- `length()`
 - Returns the number of characters in the input string
- `substring(startIndex, endIndex)`
 - Returns a new string that is part of the input string
- `toLowerCase()`
 - Converts all the characters to lower case
- `toUpperCase()`
 - Converts all the characters to upper case
- `concat(anotherString)`
 - Concatenates String with anotherString



+ Try it!

```
String s1 = "abcdefg";  
println( s1.charAt(0) );
```

```
String s1 = "abcdefg";  
String s2 = "abcdefg";  
if (s1.equals(s2)) println("They are equal");
```

```
String s1 = "abcdefg";  
println( s1.indexOf('c') );
```

```
String s1 = "abcdefg";  
println( s1.substring(2, 5) );
```

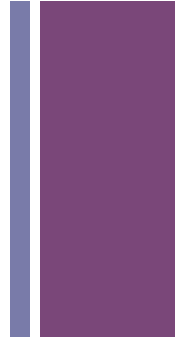
```
println( "abcdefg".length() );
```

```
println( "abcdefg".toUpperCase() );
```



Comparing Strings : Always use equals()

- Never use '==' ... Why?
 - String are objects
 - The '==' operator checks that two items are identical
 - Two objects can contain the same data, but be different object instances
 - The '==' operator tests that the two objects are the same object ... generally, that's not what we want
 - The equals() method tests the data of the two String objects for equality



+ Other forms of indexOf()

Return s	Description
int	<u>indexOf</u> (int ch) Returns the index within this string of the first occurrence of the specified character.
int	<u>indexOf</u> (int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	<u>indexOf</u> (String str) Returns the index within this string of the first occurrence of the specified substring.
int	<u>indexOf</u> (String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

+ Other forms of substring()

Returns	Description
String	<u>substring</u> (int beginIndex) Returns a new string that is a substring of this string.
String	<u>substring</u> (int beginIndex, int endIndex) Returns a new string that is a substring of this string

+ Digit chars in a String are not integers

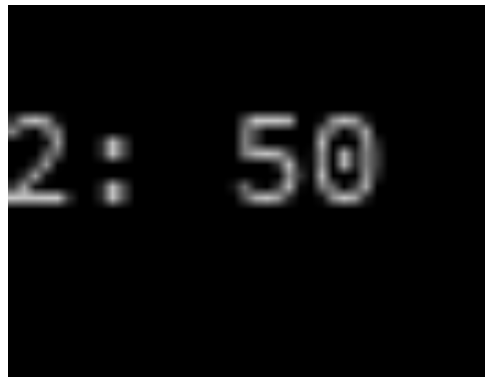
```
String s = "12345";

void setup() {

    char myChar = s.charAt(1);
    byte myByte = byte(myChar);
    print(myChar);
    print(": ");
    println(myByte);

}
```

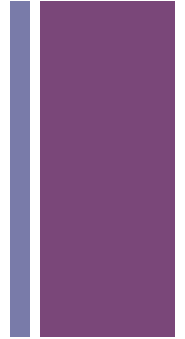
Result:



2: 50



Building Strings – Use '+'



```
void setup() {  
  String s1 = "Hello";  
  String s2 = "World";  
  String s3 = one + " " + two;  
  println( s3 );  
}
```

```
void setup() {  
  String s1 = "She is number ";  
  String s2 = " in computer science.";  
  String s3 = s1 + 1 + s2;  
  println( s3 );  
}
```

Numbers are converted to Strings prior to concatenation

+ Special chars in a String using escape char(\)

Use the escape character to embed special characters in a String

```
'\n'  new line
```

```
'\t'  tab
```

```
void setup() {  
    println("This is line 1\nThis is line 2");  
}
```



Strings can be held by Arrays

- (Just like any other object or primitive type)

```
String[] tokens = new String[5];
```

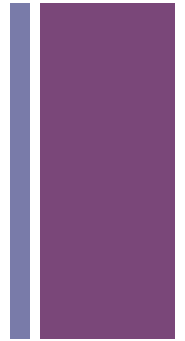
```
void setup() {
```

```
    tokens[0] = "one";  
    tokens[1] = "two";  
    tokens[2] = "three";  
    tokens[3] = "four";  
    tokens[4] = "five";
```

```
    println(tokens);
```

```
}
```

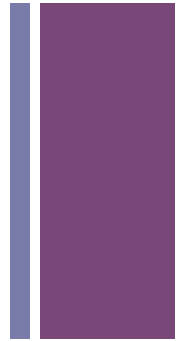
```
[0] "one"  
[1] "two"  
[2] "three"  
[3] "four"  
[4] "five"
```





Strings can be held by Arrays

- Initialized when declared



```
String[] tokens = new String[] {"one", "two", "three", "four", "five"};
```

```
void setup() {  
    println(tokens);  
}
```

```
[0] "one"  
[1] "two"  
[2] "three"  
[3] "four"  
[4] "five"
```



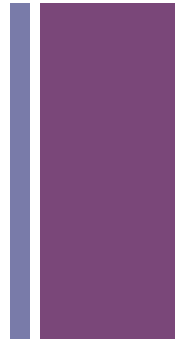
Strings can be held by Arrays

- Not initialized

```
String[] tokens = new String[5];
```

```
void setup() {  
    println(tokens);  
}
```

```
[0] null  
[1] null  
[2] null  
[3] null  
[4] null
```



+ Built-in String functions (not methods)

`split(bigString, splitChar)`

- Breaks a String into a String Array, splitting on `splitChar`
- Returns new String Array

`splitTokens(bigString, splitCharString)`

- Breaks a String into a String Array, splitting on any char in `splitCharString`

`join(stringArray, joinChar)`

- Builds a new String by concatenating all Strings in `stringArray`, placing `joinChar` between each
- Inverse of `split()` function

`nf(intValue, digits)`

`nf(floatValue, left, right)`

- Formats a number as a String

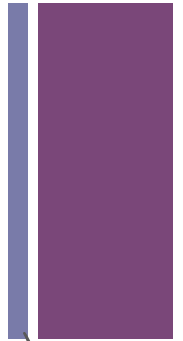
`trim(theString)`

- Removes whitespace from the beginning and end of *theString*

`text(theString, x, y)`

`text(theString, x, y, width, height)`

- Draws *theString* on the sketch at (x, y)



+ Split a String based on a single or multiple separator chars

```
String s1 = "12, 34, 56";  
String[] as;
```

```
void setup() {  
    as = split(s1, ",");  
    //as = trim(as);  
    println( as );  
}
```

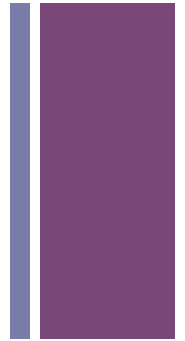
```
[0] "12"  
[1] " 34"  
[2] " 56"
```

```
String s1 = "Data: 12, 34, 56";  
String[] as;
```

```
void setup() {  
    as = splitTokens(s1, ":",");  
    //as = trim(as);  
    println( as );  
}
```

```
[0] "Data"  
[1] " 12"  
[2] " 34"  
[3] " 56"
```

+ Join a String Array with a join char



```
String[] as = new String[] {"one", "two", "buckle my shoe"};

void setup() {
  String s1 = join( as, " | " );
  println( s1 );
}
```

```
one | two | buckle my shoe
```



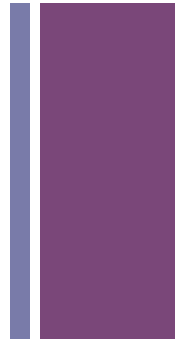
Numbers can be formatted as Strings

```
String s1 = "She is the";
```

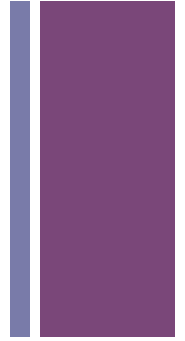
```
String s2 = "programmer.";
```

```
phrase = s1 + nf(7, 3) + " " + s2;  
// nf( integer, number of digits )  
// "She is the 007 programmer."
```

```
phrase = s1 + nf(3.14159, 3, 2) + " " + s2;  
// nf( float, digits before decimal, digits after decimal )  
// "She is the 003.14 programmer."
```

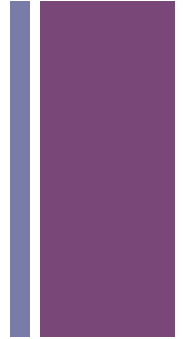


+ Acquire data: Source = Document



- `// Sketch 7-1: Parsing an input text file`
`String inputFile = "Obama.txt";`
`String [] fileContents;`
`fileContents = loadStrings(inputFile);`
- `fileContents` has the source!
- What next?

+ Parse



- How do we turn fileContents into words?

- join array into one long string

```
String rawText;  
rawText = join(fileContents, " ");
```

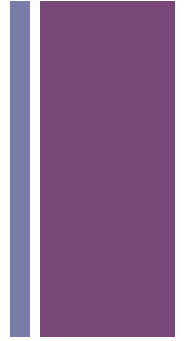
- make all same case

```
rawText = rawText.toLowerCase();
```

- remove symbols and split string into words

```
String delimiters = " ,./?<>;:'\"[{}]\\"|=+~_()*&^%$#@!~";  
tokens = splitTokens(rawText, delimiters);
```


+ Filtering: Word Frequency List

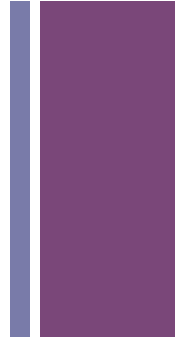


- Create a set of word frequency pairs.
- Algorithm:
 - create empty set pairs
 - for each token
 - if pairs has (token,count)
 - increment count
 - otherwise
 - add (token, 1)

+ The word class

```
class Word {
    // Each Word is a pair: the word, and its frequency
    String word;
    int freq;
    Word(String newWord) { // Constructor
        word = newWord;
        freq = 1;
    } // Word()
    String getWord() {
        return word;
    } // getWord()
    int getFreq() {
        return freq;
    } // getFreq()
    void incr() { // increments the word count
        freq++;
    } // incr()
    String toString() { // print representation of Word objects
        return "<" + word + ", " + freq + ">";
    }
} // class Word
```

+ Data Structures



- Ways of storing and organizing data
- Arrays
 - Must know the size ahead of time
 - Can not grow and shrink at will

+ Built-in Collection Classes



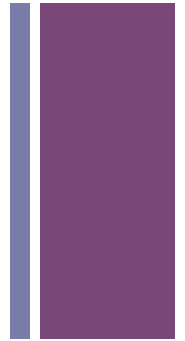
■ ArrayList

- A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
- Objects in an ArrayList are accessed by **index** [0..size-1]

■ HashMap

- A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
- Objects in a HashMap are accessed by a **key**, which can be another Object, frequently a String.

+ ArrayList



■ Constructors

```
ArrayList lst1 = new ArrayList();  
ArrayList lst2 = new ArrayList(int initialSize);
```

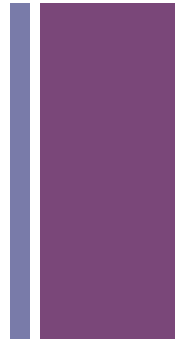
■ Fields

■ Methods

```
size() // Returns the num of items held.  
add(Object o) // Appends o to end.  
add(int idx, Object o) // Inserts o at pos idx.  
remove(int idx) // Removes item at pos idx.  
get(int idx) // Gets items at idx. No removal.  
set(int idx, Object o) // Replaces item at idx with o.  
clear() // Removes all items.  
isEmpty() // true if empty.  
toArray() // returns an array that contains  
// the contents of the list
```

+ Make the set using an ArrayList

```
ArrayList<Word> wordFrequency = new ArrayList();  
  
// Compute the wordFrequency table using tokens  
for (String t : tokens) {  
    // See if token t is already a known word  
    int index = search(t, wordFrequency);  
    if (index >= 0) {  
        wordFrequency.get(index).incr();  
    }  
    else {  
        wordFrequency.add(new Word(t));  
    } // if  
} // for
```



+ HashMap

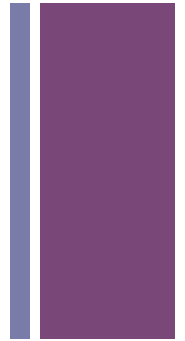
■ Constructors

```
HashMap map1 = new HashMap();  
HashMap map2 = new HashMap(int initialCapacity);
```

■ Fields

■ Methods

```
size() // Returns num of items held.  
  
put(Object key, Object o) // Puts o in map at key  
  
remove(Object key) // Remove Object at key  
  
get(Object key) // Get Object at key  
  
containsKey(Object key) // True if map contains key  
  
containsValue(Object val) // True if map contains val  
  
clear() // Removes all items.  
  
isEmpty() // true if empty.
```



+ Make the set using a HashMap?

