UNIT 8. TEXT IN PROCESSING: BASIC STRINGS

Summary

Up until now in the course, all of your computation and variables have been using numerical values. Here, we learn how to use text in processing.

In this section, you will…

- Learn how text is stored in Processing
- See ways to work with text, such as combining multiple segments into one, and converting back and forth to numbers
- Learn how to put text to the console, and on the canvas.
- Learn about the difference between a chunk of text and a single character.
- See how to get the length of text, and how to extract specific characters

Learning Objectives

After finishing this unit, you will be able to …

- Create variables to store text data, called Strings
- Combine multiple Strings together
- Convert between Strings and numerical data
- Get the length of a String, as an integer
- Create a character variable
- Get a specific character out of a string, from a specific index.

How to Proceed

- Read the unit content.
- Have a Processing window open while you read, to follow along with the examples.
- Do the sets of exercises in the **Check your Understanding** sections.
- Re-check the **Learning Objectives** once done.

© James Young, 2015

8.1 Introduction

Until now in the course we have only been dealing with numerical data: integer and floating point numbers. It is time to learn how we can work with text in Processing.

In computing, a piece of text is referred to as a ***string of characters***, or a string for short. To write a string (a piece of text) in processing, you encapsulate it in double quotes as follows:

```
println("hello world!");
```

Make sure to use double quotes `"` (the shift plus single quote) not the single quote `‘` or `'`, or two single quotes `''`. Processing takes everything from the first quote to the last quote as verbatim text. If you try to put text without the quotes, Processing tries to parse the text as programming commands and you'll get an error.

While strings give you a lot of power (text is very important), they also introduce a lot of problems. A lot of this stems from the fact that strings are a new kind of data. Until now, all our data types are *primitive types*, those that just store one simple piece of data. Strings are more complicated: they can have varying lengths, they can be short or long, etc. Because of this, strings fall into a different category of data called ***Objects. Objects are ways to encapsulate more complex kinds of data in easy to use ways.*** Unfortunately, you often use Objects differently than regular primitives, and even worse, different Objects are often used very differently. In this course, we won't learn Objects, but be aware of this as the reason why Strings can be so different from time to time, and be assured that it gets easier as you learn computer science (and Objects!)

The first difference is in the naming scheme. Unlike our primitive data types, Object types should start with a capital letter. In this case, to make a new string variable:

```
String variableName;
```

Other than that you can use this like your other types so far. For example, you can have combined declaration and instantiation:

```
String name = "Jim Young";
```

The variables themselves are also usable in many ways like other variables, for example, as above we can use it in our `println` statement:

```
String name = "Jim Young";
println(name);
```


© James Young, 2015

Many of the operations that we learned on numerical data (such as multiplication or modulo) do not work on String data. Luckily, this makes sense: what does multiplication mean with text? But, we have some new operators that we'll see shortly.

This is a good time to introduce the ***empty string. This is the simplest string that you can come up with, as it contains no information!*** (a little Zen?) Students sometimes struggle with this, but we'll get some practice with it. You specify an empty string by putting two double quotes together like this:

```
String empty = "";
```

If you print this out, as may be expected, you get nothing ☺. You can think of this as kind of a default, or starting value, for a string.

Advanced: how do you place a quote sign in a string? If you try to do it, it doesn't work, since Processing doesn't know the difference between a quote to end the string, and a quote as part of the string. The solution is to use what is called an escape sequence, which is a command inside a string. There are many such sequences, but the one we would use here is backslash quote. For example:

```
String message = "Hi ¥"friend¥"";
```

8.2 Concatenating Strings

The word *concatenate* may sound complex, but all that it means is to stick two things together. Concatenating strings just takes two strings and combines them to make one. For example, consider how silly the following code is:

```
String firstName = "Jim";
String lastName = "Young";
String fullName = "Jim Young";
```

This is odd because if we have the first and last name, we should be able to calculate the full name without having to type it in again. We can do this using the concatenate operator. All that you do, is put a plus sign (+) between two strings as follows:

```
String firstName = "Jim";
String lastName = "Young";
String fullName = firstName + lastName;
println(fullName);
```

I recommend that you try this to test it out. What is the output? Any problems? The

© James Young, 2015

concatenation worked, but we got `JimYoung` as our output with no space between them: Processing stuck them directly together. This I simple, because you can chain the concatenation operations together. Just use more concatenation to add a space:

```
String fullName = firstName + " " + lastName;
```

Concatenation is easy and scales up very intuitively. Let's try making a Madlibs game. Let's take a template string and plug in words. Here is our template:

"<exclamation>! He said <adverb> as he jumped into his convertible <noun> and drove off with his <adjective> partner."

To convert this into processing, let's first create the words as variables. In your own case, select your own instances of words:

```
String exclamation = "smeg!";
String adverb = "happily";
String noun = "dog";
String adjective = "red";

// calculate our output text
String output = exclamation +"! He said "+adverb+
       " as he jumped into his convertible " + noun +
       " and drove off with his " + adjective + " partner.";
println(output);
```

There is also shorthand concatenation. Commonly, just like with numbers, we do the following kind of scenario to keep adding to a string:

```
String output = "";
output = output + exclamation;
output = output + "! He said";
```

and so on. In this case, we can use the += shorthand just like with numerical data:

```
output += exclamation;
output += "! He said";
```

It just saves a few keystrokes, but you'll appreciate it in practice ☺

8.3 **Graphical Text in Processing**

So we can toss text to the console for testing, but how do we include it in our

© James Young, 2015

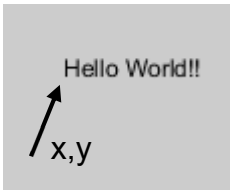
graphical programs? We need some new commands.

```
text(string data, x, y); // draw string at x,y
```

For example,

```
text("Hello World!!", 50, 50);
```

Be careful, the x and y are the bottom left corner of the base line of the graphical text, not the top left like with a rectangle. Some text may go below this, e.g., a lowercase "g".



You can set the color of the text using the same `fill` command that you already know.

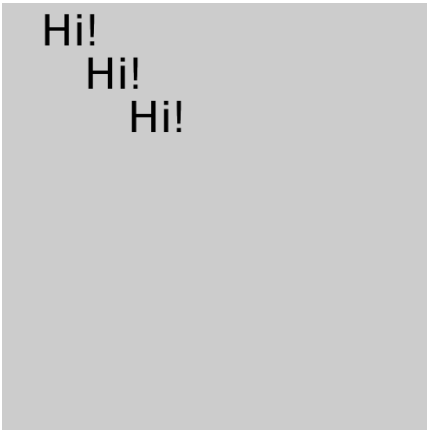
You can also set the size of the text in pixels:

```
textSize(size in pixels);
```

Make sure to set this size *before* you draw the text.

For example:

```
int size = 50;
textSize(size);
text("Hi!", size, size);
size = 100;
text("Hi!", size, size);
size = 150;
text("Hi!", size, size);
```



Processing also has other great functions, like determining how many pixels long a string will be when displayed on the canvas, or how tall it will be – these measurements can be useful for doing layouts of your interface. However, they are beyond the scope of this class. Feel free to look them up online!

8.4 Converting between text and numbers

Fundamentally, the *idea* of a number and its *textual representation* are quite different. For example, the number 5 is an abstract concept that you can store in an integer, and do mathematical operations on it. However, the textual representation of "5" is arbitrary and depends on the language. For example, these are all representation of the same number: 5 V ||||| いつつ 五 오. As another example, the number 1234.56 can be textually written as 1,234.56 or 12,34.56 and still have the same intrinsic

© James Young, 2015

numerical meaning and value.

As such, the string “5” is different in a computer than the integer 5. Try the following:

```
String s = "5";
int i = s;
```

Processing complains that you cannot convert from a `String` to an `int`, and it doesn’t work. Likewise:

```
int i = 5;
String s = i;
```

Doesn’t work. T***ext and numbers are fundamentally different concepts, and computers see them as fundamentally different data types***. The computer does not see them the same as you or I would.

The workaround is that ***we need specific commands to convert back and forth between numerical and string data.*** We have two commands for this:

```
int int(StringData); // converts string to an int
float float(Stringdata); // converts string to a float
String str(numericalData); // converts a number to a String
```

We can use these to fix our above examples:

```
String s = "5";
int i = int(s); // convert the String to an integer
```

Likewise:

```
int i = 5;
String s = str(i); // convert an integer to a String
```

This fixes our problem, and these tools will be an important part of your toolkit.

Unfortunately, these commands aren’t that powerful. If the command gets confused, it just gives you a bad result. You need to watch for that. For example:

```
String s = "1,234";
int i = int(s);
float f = float(s);
println(i);
```


© James Young, 2015

```
println(f);
```

In this case, the `i` gives you a 0, and the `f` gives you `NaN` – This is short for "not a number". Yeah, THAT makes sense. Basically, if you get a `NaN`, it just means it's broken.

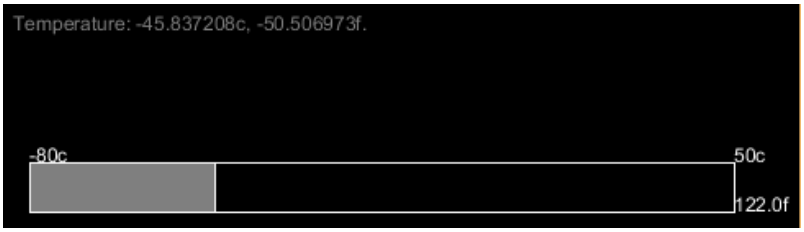
You will find that, in particular, converting from a number to a string for output is very common. This is so common, in fact, that Processing has a shortcut for it. If at any point, you combine a string and a number with a plus sign – that is, you try to concatenate a string with a number – Processing does the number-to-string conversion for you. For example:

```
String s = "My age: ";
s = s + 99; // actually s = s + str(99)
```

This is another nice thing that will save you a bunch of time.

8.5 **Example: Celsius and Fahrenheit Scale**

Let's make an interactive Celsius and Fahrenheit scale, where you can move the mouse to select a temperature and see both the c and the f reading.



Let's work through our globals to start setting up this project. Let's start with the basic dimensions:

```
final int S_TOP = 100; // scale top
final int S_LEFT = 30;
final int S_WIDTH = 400;
final int S_HEIGHT = 30;
```

We also need to setup the range of the scale (hot to cold):

```
final int HOT = 50; // celcius
final int COLD = -80;
final int TEMP_RANGE = HOT-COLD;
```

Now that we have our basic setup thought out, we can start drawing our scale. Let's use top-down programming to setup the draw block, defining the jobs that need to be done.

© James Young, 2015

```
void draw()
{
  background(0);
  drawScaleBackground();
  drawScaleLabels();
  fillScaleUsingMouse();
  drawScaleReading();
}
```

Drawing the scale background just requires a rectangle with the correct colors, using the globals that we already setup. I use a white outline and black fill. This is simple, so I don't put the code here.

Next, we need the labels at the left and the right of the scales. This is slightly trickier, as there are three problems to solve. First, where on the screen are they drawn? Second, we need the Celsius and Fahrenheit for both, and this should be calculated and not hard-coded, as the `HOT` and `COLD` variables may change. Third, we need to construct the required string output.

The locations are not that hard. The leftmost top label actually goes at the scale left and top. Remember that text's y coordinates, when drawn, refer to the bottom of the string. So, if we want a string on the top of our scale rectangle, we use the same coordinates.

```
int x = S_LEFT;
int y = S_TOP;
```

Then, to calculate the actual Fahrenheit we use the standard formula. This was an exercise in the previous chapter. Make sure to use floating points, and be careful of integer division!

```
float f = 9.0/5.0*COLD+32;
```

To construct the labels, we use our conversion functions to convert the numbers to text, and then, use concatenation to attach the appropriate "c" or "f" suffix.

```
String celsius = str(COLD)+"c";
String fahrenheit = str(f)+"f";
```

Note: do you see a shortcut here? Concatenating a number with a string automatically does the conversion for you, so the `str` function calls are actually not

 © James Young, 2015

necessary in this case.

Finally, we need to draw these. We have the x and y for the top label, and for the bottom, we just add the height of the scale. We also already have the strings:

```
text(celsius, x, y);
text(fahrenheit, x, y+S_HEIGHT);
```

Now that you have left labels, calculating the right ones is trivial. Try it on your own – you need new numbers, new strings, and a new location. Notice that you can reuse your existing variables for this.

Now, we have a nice scale with Celsius and Fahrenheit on it!!! Yay!

Filling the scale based on the mouse position is a little bit tricky, and requires the following steps to be done in our `fillScaleUsingMouse` function:

- Calculate how far along the scale the mouse is. Take the mouse position and subtract the left end of the scale
- Make sure we're not off either end of the scale!!
- Draw the filling using that width

```
int xOffset = mouseX - S_LEFT; // how far along scale
xOffset = max(0, xOffset); // if < 0, make 0
xOffset = min(S_WIDTH, xOffset); // if > width, make width
rect(S_LEFT, S_TOP, xOffset, S_HEIGHT);
```

Choose a color for the filling, too, before drawing that fill. I used grey. At this point, your program should draw a filling in the scale that animates as you move the mouse. It should be aligned with the mouse cursor's X position.

Finally, the last part is to draw the label at the top of the screen, which gives the current reading at the mouse position in both Celsius and Fahrenheit. This has several steps – we need to get how far the mouse is along the scale in pixels, then convert that to how many degrees that represents. Then we need to convert that to Fahrenheit, and construct our message.

We already calculated our mouse position along the scale in the `xOffset` variable in a different function. You can either re-calculate that here, or, make the previous variable global so that you can re-use the value. In that case, pay attention to the order that the functions are called in, to ensure that the variable is properly calculated first.

We convert the mouse offset along the scale (how far it is from the scale left) to a percentage of the scale first, and then mapping that to the temperature range.

 © James Young, 2015

First, we are hit with a problem. If we try to generate our percentage:

```
float tempPerc = xOffset/S_WIDTH;
```

Can you see the problem? Both `xOffset` and `S_WIDTH` are integers, giving integer division. We can fix this by making one of the variables, the `xOffset`, a `float`.

Now that we have the temperature as a percentage, we map it to our temperature range. First we find out where in the range we are, then we add in the minimum temperature.

```
float tempC = tempPerc*TEMP_RANGE+COLD;
```

Now we can also calculate the Fahrenheit:

```
float tempF = 9.0/5.0*tempC+32;
```

And construct our output message:

```
String message = "Temperature: "+tempC+"c, "+tempF+"f.";
text(message,20,20);
```

You should make sure to set the color, too.

All done! Here is my final code:

```
final int S_TOP = 100;
final int S_LEFT = 30;
final int S_WIDTH = 430;
final int S_HEIGHT = 30;
final int HOT = 50; // celcius
final int COLD = -80;
final int TEMP_RANGE = HOT-COLD;
final int xOffset = 0;

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

void drawScaleBackground()
{
```


© James Young, 2015

```
  stroke(255);
  fill(0);
  rect(S_LEFT, S_TOP, S_WIDTH, S_HEIGHT);
}

void drawScaleLabels()
{
  fill(255);

  int x = S_LEFT;
  int y = S_TOP;

  float f = 9.0/5.0*COLD+32;

  String celsius = str(COLD)+"c";
  String fahrenheit = str(f)+"f";
  text(celsius, x, y);
  text(fahrenheit, x, y+S_HEIGHT);

  // draw right marks
  x = S_LEFT + S_WIDTH;
  f = 9.0/5.0*HOT+32;
  celsius = HOT+"c";
  fahrenheit = f+"f";
  text(celsius, x, y);
  text(fahrenheit, x, y+S_HEIGHT);
}

void fillScaleUsingMouse()
{
  // fill in thermometer
  int xOffset = mouseX - S_LEFT; // how far along scale
  xOffset = max(0, xOffset); // if <0, make 0
  xOffset = min(S_WIDTH, xOffset); // if >width, make width
  fill(127);
  rect(S_LEFT, S_TOP, xOffset, S_HEIGHT);
}

void drawScaleReading()
{
```


© James Young, 2015

```
  // output the reading from the mouse
  float xOffset = mouseX - S_LEFT; // how far along scale
  float tempPerc = xOffset/(float)(S_WIDTH);
  float tempC = tempPerc*TEMP_RANGE+COLD;
  float tempF = 9.0/5.0*tempC+32;
  String message = "Temperature: "+tempC+"c, "+tempF+"f.";
  stroke(255);
  text(message, 20, 20);
}

void draw()
{
  background(0);
  drawScaleBackground();
  drawScaleLabels();
  fillScaleUsingMouse();
  drawScaleReading();
}
```

8.6 Characters

This may seem strange at first, but in Processing, in addition to the String type, we have a type for single characters. The reason for this is that while String is an object (complex!), the character is a primitive type. They are otherwise very like the other primitive types. In fact, Strings are actually internally made up of a collection of this character primitive type.

In the short term, we won't use characters much. Later, when we learn more advanced programming techniques, we will do more work with taking a string apart into individual characters.

To create a character variable, you use the keyword `char`, which can be pronounced like "car" (short for character), or char (like charbroiled).

```
char c;
```

You create a character literal by using the single quotes. Be careful, using double quotes gives you a string, which is different.

```
char c = 'j';
```

Characters can be numbers, letters, upper case, lower case, symbols, etc. Basically,

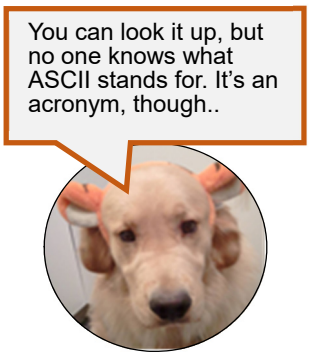
© James Young, 2015

anything that can go in a string, is a character. But be careful! You cannot place two characters in one variable:

```
char c = 'ja'; // doesn't work
```

8.7 What is a Character anyway?

Everything in a computer is stored as a number. Characters are no exception. When early computer people started to store characters, they had to think up a way to convert a random character like the letter 'q' into a number. The solution was to develop a standardized lookup table, where each character would be assigned a number. For example, let's say that 'q' is 113. Every time the computer encounters the character 113, it draws a q from its font. This kind of system requires standardization – all computers that talk to each other need to agree on this.



Decimal	Char	Decimal	Char	Decimal	Char
0	[NULL]	48	0	96	`
1	[START OF HEADING]	49	1	97	a
2	[START OF TEXT]	50	2	98	b
3	[END OF TEXT]	51	3	99	c
4	[END OF TRANSMISSION]	52	4	100	d
5	[ENQUIRY]	53	5	101	e
6	[ACKNOWLEDGE]	54	6	102	f
7	[BELL]	55	7	103	g
8	[BACKSPACE]	56	8	104	h
9	[HORIZONTAL TAB]	57	9	105	i
10	[LINE FEED]	58	:	106	j
11	[VERTICAL TAB]	59	;	107	k
12	[FORM FEED]	60	<	108	l
13	[CARRIAGE RETURN]	61	=	109	m
14	[SHIFT OUT]	62	>	110	n
15	[SHIFT IN]	63	?	111	o
16	[DATA LINK ESCAPE]	64	@	112	p
17	[DEVICE CONTROL 1]	65	A	113	q
18	[DEVICE CONTROL 2]	66	B	114	r
19	[DEVICE CONTROL 3]	67	C	115	s
20	[DEVICE CONTROL 4]	68	D	116	t
21	[NEGATIVE ACKNOWLEDGE]	69	E	117	u
22	[SYNCHRONOUS IDLE]	70	F	118	v
23	[ENG OF TRANS. BLOCK]	71	G	119	w
24	[CANCEL]	72	H	120	x
25	[END OF MEDIUM]	73	I	121	y
26	[SUBSTITUTE]	74	J	122	z
27	[ESCAPE]	75	K	123	{
28	[FILE SEPARATOR]	76	L	124	|
29	[GROUP SEPARATOR]	77	M	125	}
30	[RECORD SEPARATOR]	78	N	126	~
31	[UNIT SEPARATOR]	79	O	127	[DEL]
32	[SPACE]	80	P		
33	!	81	Q		
34	"	82	R		
35	#	83	S		
36	$	84	T		
37	%	85	U		
38	&	86	V		
39	'	87	W		
40	(	88	X		
41	)	89	Y		
42	*	90	Z		
43	+	91	[		
44	,	92	\		
45	-	93	]		
46	.	94	^		
47	/	95	_		

Image cc, derived from commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

One early standard for this was called ASCII (pronounced ass-key). In ASCII, all the basic characters and some "control" characters (to send commands to, e.g., old printers), were put onto this table. Everyone agreed, so now that every time the computer encounters the number 84 as a character, it knows that it's a capital T.

Advanced you can test this out. If you force a character to be read as a number, then Processing will tell you the ASCII number. How can we do this? We can just store the character into an integer. There is some things happening under the hood here that we'll learn soon. People don't really do this, though, so it's just a bit of a toy example.

```
char c = 'X';
int number = c;
println("The ASCII number for "+c+" is: "+number);
```


© James Young, 2015

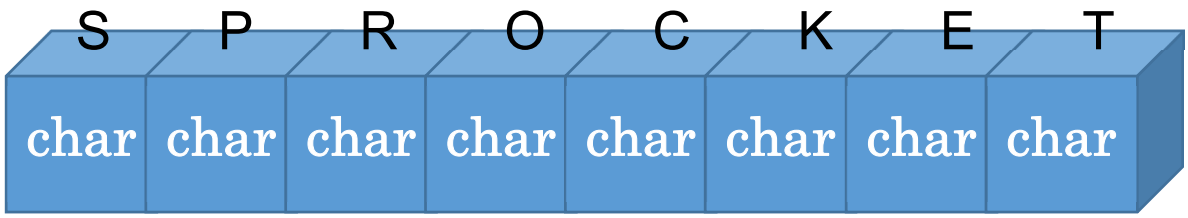
Looking at the above ASCII table, can you see any limitations? This is extremely limited! No accented letters! Does not handle complex writing systems! We need a new standard that can handle Chinese, Arabic, Hebrew, and Korean!

こんにちは！　中国語　안 녕　하 세 요　שָׁלוֹם

A new standard was developed for all languages – it's called Unicode. One code to rule them all, one code to bind them. We don't cover Unicode in this course, but it's good to know a rough idea of what it is.

8.8 **Characters and Strings**

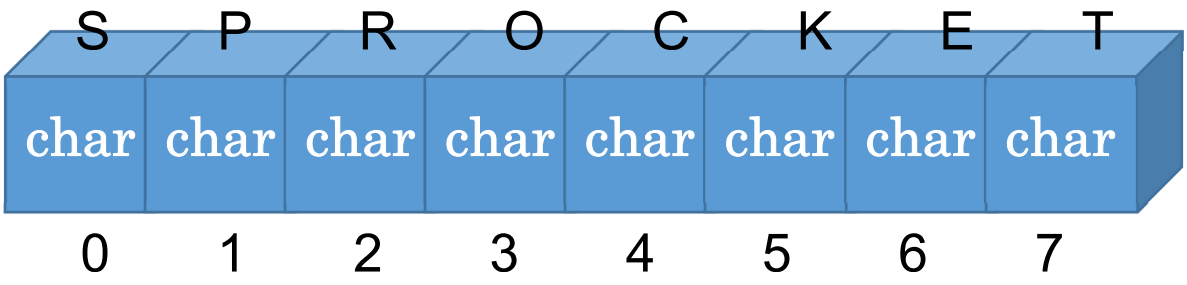
So I told you that Strings are made of characters, so now let's learn a little more about that. You should think of strings as a series of boxes, where each box is a character. For example, the string "SPROCKET" is stored internally as follows:



You can see that each character in the string gets its own box. Even spaces, symbols, etc. Now, internally Processing numbers them in a very specific way that is simple at first but will end up causing you all kinds of grief:

String Character numbering: the characters are numbered in order, starting at 0.

As follows:



How many characters are in the string "SPROCKET"? 8. What is the index of the last character? 7!!!!!! This is called the dreaded off-by-one error which will plague you for the rest of your computer science career.

© James Young, 2015

Off by one error: Since computers start counting at 0, and we generally start counting at 1, your intuition is often off by 1.

For example, with strings, the index of the last bin is always the length of the string -1, as above.

8.9 String methods

Strings, since they are objects, can be used differently than other variables. There are a new kind of command that we can do on Strings. Up until now, we call commands functions. When a command is attached to an object, we call them methods. Here is some syntax (not exactly how you use it, that's next):

```
int stringVariable.length(); // number of characters
char stringVariable.charAt(int index);
```

These are commands that can tell you how long a string is in number of characters (including spaces, punctuation, symbols, etc.), and, that can get you a character in a specific bin.

These commands can be used as follows:

```
String s = "Hello world!";
println(s.length());
println(s.charAt(0)); // first character
```

The output here is 12, and H. 12 is the number of characters in the string, and H is the character at bin 0, which is the first bin. How would you get the last character using these commands?

```
s.charAt(s.length()); // ERROR! Off by one
s.charAt(s.length()-1);
```

For now, you will mostly use strings to put text on your screen relating to your program. Soon, however, we will learn more advanced techniques that will let us do things more with the character and length methods, for example, calculating someone's initials.

© James Young, 2015

8.10 Check Your Understanding: Exercises

Exercise 1. Computers often use template text in a situation, like a website, and then populate it depending on the user. Write a program to use the following template text:
"Hello <name>, from <city>, <country>. Thank you for shopping at <store>."

a. Create variables for each unknown, and assign them reasonable values.
b. Using string concatenation, construct a single new string with the entire message in it
c. Put the string to console
d. Put the string onto a canvas graphically

Exercise 2. Make a program that takes two digits, as Strings, such as "1" and "5" (call them number 1 and number 2), and create two numbers by concatenating them in both orders. In this case, you get "15" and "51". Convert them to numbers, and divide the first (number 1 then 2) by the second (2 then 1), in this case, 15/51. In what instance do you get 1 as a result? In what instance may this crash?

Exercise 3. Your friend is having trouble understanding the coordinate space in Processing, so you came up with a plan to help them. Make a program that displays the mouse's current position on the screen, so that as the mouse moves around they can see how the position changes.

a. Construct a string that stores "<mouseX>, <mouseY>"
b. Set the text size to 20
c. Display the text at x=0, and half way down the screen

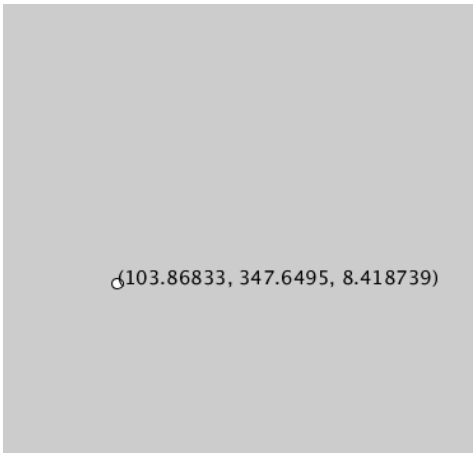
Exercise 4. The same friend has trouble understanding how the font sizes change. Update Exercise 3, and make the font size equal to the mouse's y coordinate, divided by 5.

Exercise 5. Make a program that prints out a string to the console in a special formatting. Given a string variable called data, construct the following: "-<firstCharater>-<string>-<lastCharacter>-(<length>)" and output it to console. For example, if data was set to "Hello!", then the output would be "-H-Hello!-!-(6)".

Exercise 6. Update Example 8.5, the temperature slider, to remember the

© James Young, 2015

maximum temperature that the mouse has ever reached. At that spot, draw a line through the scale, and put the text “Max: <maximum value>”.



Exercise 7. Make a program that plots a circle at a random spot and size on the screen, and displays the coordinates and size of the circle. How would you get whole numbers only, and not real numbers? We haven’t learned how yet ☺.

a. Move the text slightly away from the circle, up and to the right.

Exercise 8. Make a program that turns a number into a special circle on the screen. Given a number, first get how many digits that number has (hint: convert to a string). Then, the x coordinate of the circle is the length to the power of 4, modulo the screen width. The y coordinate is the length times the number itself, times the last digit in the number, modulo the height. Make the ellipse 50x50. (The output here is pretty boring – but as long as it works reasonably well, you are good to go. Just for practicing all the new stuff you learned).

How did you do?

Learning Objectives

How did you do? Go back to the beginning of the unit and check how you measure up to the learning objectives.

© James Young, 2015

(page intentionally left blank)

© James Young, 2015