Types review

• We’ve covered the following types:
  ○ int
  ○ float
  ○ char
  ○ boolean

• Today we'll cover the type **String**
The String type

- A **String** is an ordered sequence of characters (the `char` type), one-after-the-other
- Strings can be used to store text-information within a program (names, emails, messages, and other text)
- Examples:
  - "You win!"
  - "I wandered lonely as a cloud."
  - "CSc 101"
Strings and text

- You’ve already written code like the following for sections and assignments:

```javascript
text("You Win!", 50, 100);
```

- The `text()` function takes three arguments
  - The first one is an argument of type String, the other two are numbers
- But, we can do more with Strings than just this
Declaring String variables

- As with other types, we can declare String variables

```java
String myName = "Jordan Smith";
String myEmail = "jsmith@email.arizona.edu";
```

- As with all variables, these can be global or local
- Strings can be passed as arguments to functions that take String types (either custom or built-in, like `text()`)  
- Notice: Strings are surrounded by double-quotes, chars surrounded by single-quotes
Declaring String variables

- Strings can be **concatenated** ("combined") using the plus-sign

```java
String myName = "Jordan Smith";
String myEmail = "jsmith@email.arizona.edu"
String nameAndEmail = myName + myEmail;
```

does this

- As with all variables, these can be global or local
- They can also be passed as parameters to functions that take String types (either custom or built-in, like `text()`)
Declaring String variables

String myName = "Jordan Smith";
String myEmail = "jsmith@email.arizona.edu"
String nameAndEmail = myName + myEmail;
text(nameAndEmail, 50, 50);

// what will this display?
Declaring String variables

String myName = "Jordan Smith";
String myEmail = "jsmith@email.arizona.edu"
String nameAndEmail = myName + myEmail;
text(nameAndEmail, 50, 50);

// how to get a space?
Declaring String variables

String end = "XYZ";
String begin = "ABC"
String letters = end + " " + begin + " " + end;
text(letters, 50, 50);

// what will this display?
Strings vs String literals

- A **String literal** is when we “literally” write out a string, surrounded by double-quotes
- A **String variable** is when we declare or use a string stored in a variable
Strings vs String literals

What are the variables? What are the literals?

```java
String end = "finish";
String begin = "start";
String letters = end + " -- " + begin + " // " + end;
text(letters, 50, 50);
```
Multiple lines

- You’ve used `text()` several times
- What if you want multiple lines of text? ... how is this done?
- We would need to use several calls to `text`
  ```java
  String quote1 = "If I fall...
  String quote2 = "don't bring me back.";
  text(quote1, 50, 50);
  text(quote2, 50, 50);
  ```
- But we can also use a special character, the newline character
Multiple lines

```java
void setup() {
  size(700, 200);
  textSize(30);
}
void draw() {
  String quote1 = "If I fall..."
  String quote2 = "don't bring me back.";
  text(quote1, 50, 50);
  text(quote2, 50, 90);
}
```

If I fall...
don't bring me back.
Multiple lines

- The newline character is a special character-combo that can be used to put a line-break into a string
- Newline character: "\n"
- Example:
  ```java
  String quote1 = "If I fall...\ndon't bring me back.";
  text(quote1, 50, 50);
  ```
void setup() {
  size(700,200);
  textSize(30);
}

void draw() {
  String quote1 = "If I fall...\ndon't bring me back.";
  text(quote1, 50, 50);
}
What will this show?

```java
String sparta = "THIS\nIS\nSPARTA!";
text(sparta, 50, 50); // what will this display?
```

- Use "\n" to automatically break to the next line
- "\n" counts as 1 character, not two!
String with newlines

- Using **only one string variable** and **one use of the text()** function, write a program that produces the following canvas:
Practice problem

String drSuess = "one fish\ntwo fish\nred fish\nblue fish";

void setup() {
  size(300, 300);
  textSize(50);
}

void draw() {
  background(100);
  fill(100, 255, 255);
  text(drSuess, 20, 50);
}
Concatenating strings with other types

- We saw strings concatenated with "+"
- You can also use the plus-sign to concatenate strings and other types together
- It is often useful to combine a String(s) with integers or floats when you want to display information
Concatenating strings with other types

- You can also use the plus-sign to concatenate strings and other types together.
- It is often useful to combine a String(s) with integers or floats when you want to display information.
- For example...

```java
void draw() {
    int slideNum = 2;
    String message = "you are on slide ";
    String full = message + slideNum;
    text(full, 50, 50);
}
```

You are on slide 2
String indices

- Each character in a String has an associated index number
- The numbers start at zero and increase
- Using indices, we can refer to parts of a string
String indices

- Each character in a String has an associated index number
- The numbers start at zero and increase
- Using an index, we can refer to a specific character of a string
- Ex: `String word` at index 2 is 'p'

```java
String word = "alphabet";
```

<table>
<thead>
<tr>
<th>letter</th>
<th>a</th>
<th>l</th>
<th>p</th>
<th>h</th>
<th>a</th>
<th>b</th>
<th>e</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Extracting parts of a String

- We can also extract parts or “chunks” of a string
- These are called **substrings**
- `substring()` creates substrings
  - requires these arguments
    - the index of the beginning character
    - the index of the ending character (plus one)
Extracting parts of a String

• Given the definition of `word` below, would expect to write
  ```java
  substring(word,0,5);  //can't do this!!!
  ```
• Cannot do this because `substring()` is a method, not a function

```java
String word = "alphabet";
```

<table>
<thead>
<tr>
<th>letter</th>
<th>a</th>
<th>l</th>
<th>p</th>
<th>h</th>
<th>a</th>
<th>b</th>
<th>e</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Side note: methods vs functions

- Many types in Processing have both built-in functions and methods
- A method is called with different syntax
Side note: methods vs functions

- Given the definition of `word` below, would expect to write
  ```java
  substring(word,0,5);  //Can't do this!!!
  ```
- Methods are called differently
  ```java
  word.substring(0,5);  //Yes, do this!
  ```

```java
String word = "alphabet";
```

<table>
<thead>
<tr>
<th>letter</th>
<th>a</th>
<th>l</th>
<th>p</th>
<th>h</th>
<th>a</th>
<th>b</th>
<th>e</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Side note: methods vs functions

- Methods are called differently
  ```
  word.substring(0,5);  //Yes, do this!
  ```
- Use `substring()` to extract these strings:
  "ha"
  "abe"
  "bet"

```
String word = "alphabet";
```

<table>
<thead>
<tr>
<th>letter</th>
<th>a</th>
<th>l</th>
<th>p</th>
<th>h</th>
<th>a</th>
<th>b</th>
<th>e</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Side note: methods vs functions

- Let's look closely at how the method is called
- To call the `substring()` method, you type:
  - name of the String, a dot, the method name, then the arguments
- Here is another example of calling the `substring()` method:

```java
String sentence = "cool story bro tell it again";
String w1 = sentence.substring(0, 4);
String w2 = sentence.substring(11, 14);
String message = w1 + w2;
text(message, 50, 50);
```
What does this program display?

String sentence = "cool story bro tell it again";
String w1 = sentence.substring(0, 4);
String w2 = sentence.substring(11, 14);
String message = w1 + w2;
text(message, 50, 50);
Extracting a word

- Write the code to extract the word "bookkeepers" using only the below string, and the substring method

```java
String sentence = "book is keeping clovers";
```
Extracting a word

- Write the code to extract the word "bookkeepers" using only the below string, and the substring method

```java
String sentence = "book is keeping clovers";
```
void setup () { 
  size(500, 200); 
  fill(0); 
  textSize(40); 
}

void draw() { 
  String sentence = "book is keeping clovers"; 
  String a = sentence.substring(0, 4); 
  String b = sentence.substring(8, 12); 
  String c = sentence.substring(20, 23); 
  String bk = a + b + c; 
  text(bk, 30, 100); 
}
Review: substrings

- Examples of using `substring()` to extract strings:

```java
String word = "alphabet";
word.substring(0, 5); // "alpha"
word.substring(4, 7); // "abe"
```

<table>
<thead>
<tr>
<th>letter</th>
<th>a</th>
<th>l</th>
<th>p</th>
<th>h</th>
<th>a</th>
<th>b</th>
<th>e</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Extracting another word

- Extract the word "flowing" using only the below string, and the substring method

String f1 = "flowers are wilting downward"
Extracting another word

- Extract the word "flowing" using only the below string, and the substring method

String f1 = "flowers are wilting downward"
Comparing Strings

- What do you think will happen when this code is run?

```java
void setup() {
  size(500, 500);
}
void draw() {
  background(100);
  String strA = "cs101";
  String strB = "cs";
  strB = strB + "101";
  if (strA == strB) {
    fill(100, 255, 100);
    rect(100, 100, 300, 300);
  } else {
    fill(0, 150, 255);
    ellipse(250, 250, 300, 300);
  }
}
```
Comparing Strings

- What do you think will happen when this code is run?
- Why did it do what it did?

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

void draw() {
  background(100);
  String strA = "cs101";
  String strB = "cs";
  strB = strB + "101";
  if (strA == strB) {
    fill(100, 255, 100);
    rect(100, 100, 300, 300);
  } else {
    fill(0, 150, 255);
    ellipse(250, 250, 300, 300);
  }
}
```
Comparing Strings

- What do you think will happen when this code is run?
- Why did it do what it did?
- When comparing strings, don’t use ==

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

void draw() {
  background(100);
  String strA = "cs101";
  String strB = "cs";
  strB = strB + "101";
  if (strA == strB) {
    fill(100, 255, 100);
    rect(100, 100, 300, 300);
  } else {
    fill(0, 150, 255);
    ellipse(250, 250, 300, 300);
  }
}
```
Comparing Strings

- Instead, use the `equals()` built-in method to compare the contents of strings.
- This makes sure to compare the entire strings with each other, character-by-character.
Searching in a String

- Often we want to find a string within a string. For example:

  ```java
  String poem = "Two roads diverged in a yellow wood";
  ```

- Does "yellow" exist in poem? If so, where does it start?
- We use the method `indexOf()` to do this
  - This method takes one argument - the string you want to search for (find the index of)
  - It will `return` the index
    - Or -1, if it cannot be found
Searching in a String

String intro = "one fish two fish, red fish blue fish";

• To find where “two” is I can do ...
Searching in a String

String intro = "one fish two fish, red fish blue fish";
int twoPos = intro.indexOf("two"); // what is twoPos?

• To find where “two” is I can do ...
Searching in a String

String intro = "one fish two fish, red fish blue fish";
int twoPos = intro.indexOf("two"); // what is twoPos?

- To find where “red” is I can do . . .
Searching in a String

String intro = "one fish two fish, red fish blue fish";
int twoPos = intro.indexOf("two");  // what is twoPos ?
int redPos = intro.indexOf("red");  // what is redPos ?

• To find where “red” is I can do ...
Searching in a String

String intro = "one fish two fish, red fish blue fish";
int twoPos = intro.indexOf("two"); // what is twoPos ?
int redPos = intro.indexOf("red"); // what is redPos ?

• To search for "brick" ...
Searching in a String

String intro = "one fish two fish, red fish blue fish";
int twoPos = intro.indexOf("two"); // what is twoPos ?
int redPos = intro.indexOf("red"); // what is redPos ?
int brickPos = intro.indexOf("brick"); // what is brickPos ?

- To search for "brick" ...
Using `indexOf()`

- What will happen when this code is run?

```java
void setup() {
  size(500, 200);
  textSize(50);
}

void draw() {
  background(100);
  String sentence = "This class is cs101";
  int a = sentence.indexOf("class");
  int b = sentence.indexOf(" is");
  String sub = sentence.substring(a, b);
  text(sub, 50, 100);
}
```
Checking and Searching

- Write a program that . . .
  - Has a global string named unknown.
    - pretend you don’t know its contents
  - The program should get the substring between "#" and "+
    in the unknown and display it to the canvas
  - See starter code on website (string_starter_b.pde)
String unknown = "These # are some characters + in a string";

void setup() {
    size(700, 200);
    textSize(50);
}

void draw() {
    background(100);
    int start = unknown.indexOf("#");
    int end = unknown.indexOf("+");
    String sub = unknown.substring(start+1, end);
    text(sub, 50, 100);
}
More String methods

• We also have methods to get the length of a string and the character at a certain index

```java
String poem = "Two roads diverged in a yellow wood";
```

• What is the length of `poem`?
• What is the character at index 4?
• Processing has these two methods:
  o `String.length()` - tells you the number of characters in the string
  o `String.charAt()` - tells you the character at an index
More String methods

- We also have methods to get the length of a string and the character at a certain index

```java
String poem = "Two roads diverged in a yellow wood";
int poemLen = poem.length(); // poemLen is 35
char c = poem.charAt(4); // c is 'r'
```
Looping through a String

- A for-loop can be used to loop through all of the characters of a string, one-at-a-time.
- We can use the for loop just like we usually do.
- Need to know the length of the string to control the for loop.
- Need to get the character at each index.
  - `String.length()` - tells you the number of characters in the string!
  - `String.charAt()` - tells you the character at an index.
Looping

- How would you go about creating a canvas that appears like so?
Looping

- Create a program that generates this canvas
Loopying

- Use this code as a baseline starting point

```java
void setup() {
  size(530, 530);
  // What text size?
}

void draw() {
  background(100);
  String alph = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  // You must loop through the characters
  // of the string alph
  // Start at index 0 and stop at the length
  // of alph!
}
```
Looping

- A solution:

```java
void setup() {
  size(530, 530);
  textSize(24);
}

void draw() {
  background(100);
  String alph = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  for(int i = 0; i < alph.length(); i=i+1) {
    char current = alph.charAt(i);
    text(current, i*20+3, i*20+20);
  }
}
```
Looping

- A solution:
- What function could we use instead of `charAt()`?

```java
void setup()
{
  size(530, 530);
  textSize(24);
}

void draw()
{
  background(100);
  String alph = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  for(int i = 0; i < alph.length(); i=i+1) {
    text( alph.charAt(i), i*20+3, i*20+20);
  }
}
```
The empty String

- To "build" a string from nothing, we start with the empty string, which is ""

```java
String poem = "Two roads diverged in a yellow wood";
String word = "";

word = word + poem.charAt(0);
word = word + poem.charAt(1);
word = word + poem.charAt(2);

// word is now "Two"
```
Building a string

- What is `word` at the end of the loop? What is shown on the canvas?

```java
void setup() {
  size(500, 200);
  textSize(50);
}
void draw() {
  background(100);
  String poem = "Two roads diverged in a yellow wood";
  String word = "";
  for (int i = 4; i < 8; i = i + 1) {
    word = word + poem.charAt(i);
  }
  text(word, 50, 150);
}
```
Building a String

- To "build" a string from nothing, we start with the empty string, which is ""
- We can build a string from the keys that the user types!
- We need to use keyPressed to capture the current key that is pressed

```java
String input = "";

void keyPressed() {
    input += key;
}
```
Building a string from input characters

Write a program that behaves like the one below. The user can type input on the keyboard, and it shows up on the canvas!
void setup() {
    size(800, 175);
    textSize(50);
}

String input = "";

void keyPressed() {
    input += key;
}

void draw() {
    background(100);
    fill(100, 255, 100);
    text("input: " + input, 30, 100);
}
Input

Modify the program so that the text can be reset when the enter key is pressed.
void setup() {
  size(800, 175);
  textSize(50);
}

String input = "";
void keyPressed() {
  if (key == '\n') {
    input = "";
  }
  else {
    input += key;
  }
}

void draw() {
  background(100);
  fill(100, 255, 100);
  text("input: " + input, 30, 100);
}
String Search

- Let’s build a program that
  - Can determine if a search string exists within some text
  - Behavior shown to the right
  - Search for "fox" (that will be the input)
  - If the searched string exists, say “yes” otherwise say “no”
  - Hit ENTER to reset the search

Search:         Found: yes
The quick brown fox jumped over the lazy dog on a cloudy day
String Search

- First get the text input working
String Search

- Next, add in the text and yes/no label
- Don’t worry about the search logic yet

Search: Found: ?
The quick brown fox jumped over the lazy dog on a cloudy day
String Search

- Now, get search working!

Search: The quick brown fox jumped over the lazy dog on a cloudy day

Found: yes
String Search

- Now, get search working!

Search:       Found: yes
The quick brown fox jumped over the lazy dog on a cloudy day
//Set found to no at the beginning
//If the input string is found in content,
//change it to yes

found = "no";
int index = content.indexOf(input);
if (index != -1) {
    found = "yes";
}
String Search

Or . . . .

// Do the search.
// Reset the found variable to "no" at beginning.
// Change to "yes" if the string is found while looping.

found = "no";
for(int i = 0; i <= content.length() - input.length(); i=i+1) {
  String compare = content.substring(i, i+input.length());
  if (compare.equals(input)) {
    found = "yes";
  }
}
Materials

- Required Materials
  - https://processing.org/reference/String.html
  - https://processing.org/reference/String_substring_.html