Module 02

{ 'A', 'r', 'r', 'a', 'y', 's' } and "Strings"

Low-level array operations

```
int[] arr1;
int[] arr2 = new int[ 17 ];
int[] arr3 = { 1, 2, 3, 4 };
```

Declaration and initialization

```
arr2[15] = arr3[2];
```

Reading and writing elements

```
int els = arr2.length;
```

Array size

Array idioms

An idiom is not a single algorithm or line of code. It's a rough template that can be customized to a specific situation.

```
for( int idx = 0; idx < arr.length; ++idx ) {
                        arr[idx]
}
```

Arrays are just values...

```
int[] arr1 = { 1, 2, 3, 4 };
int[] arr2 = arr1;

int[] processArray( int[] arr, float val )
{
  ...
}

int[] arr3 = processArray( arr1, 3.14 );
```
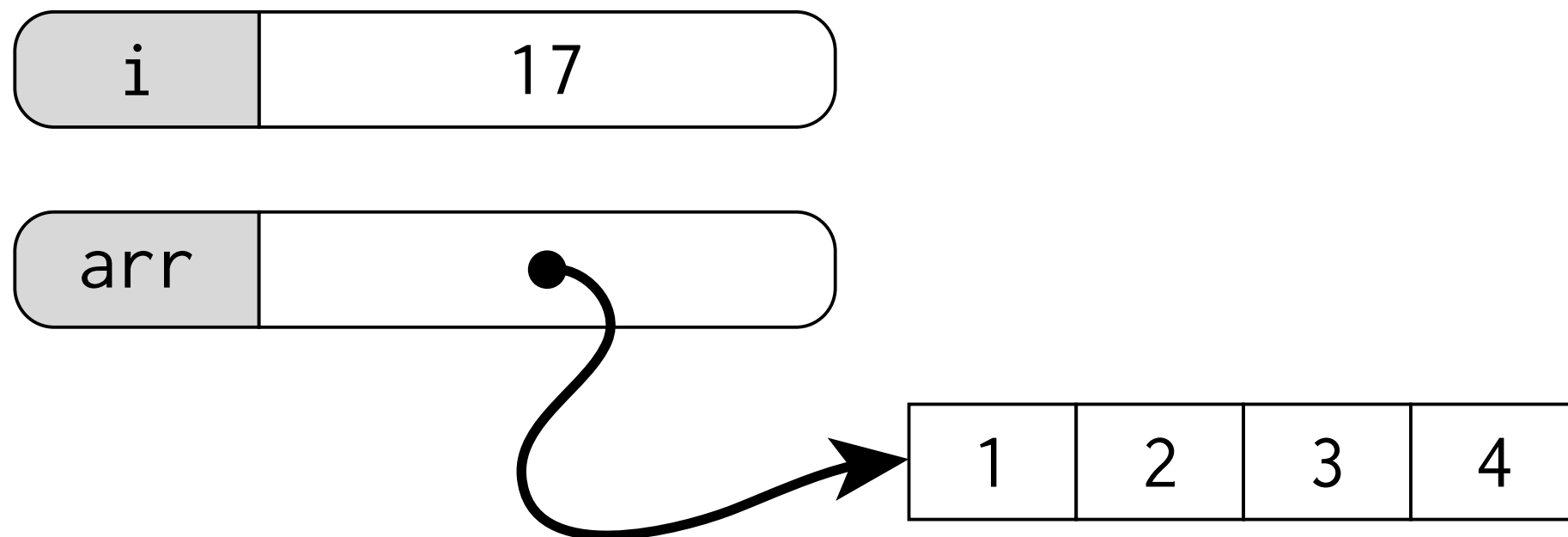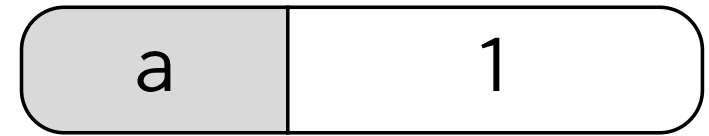
...aren't they?

An array value is really an arrow pointing to the place in memory where all the array elements are stored. We say that an array variable is a *reference*.

```
int i = 17;
int[] arr = { 1, 2, 3, 4 };
```

```
int a = 1;
```

a	1

```
int b = 2;
```

a	1

b	2

```
a = b;
```

a	2

b	2

```
b = 3;
```

a	2

b	3

```
int[] a = { 1 };
int[] b = { 2 };
a = b;
b[0] = 3;
println( a[0] + b[0] );
```
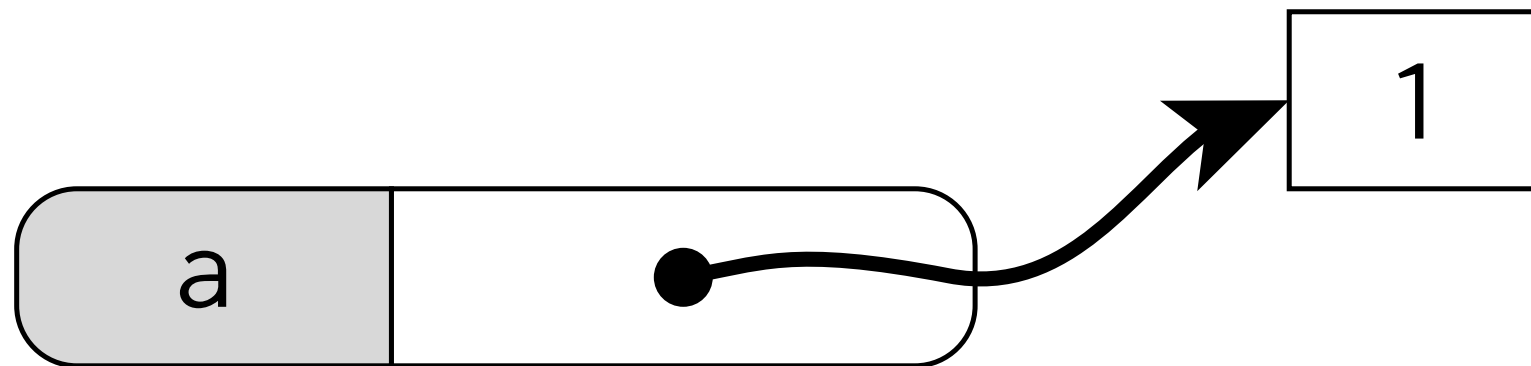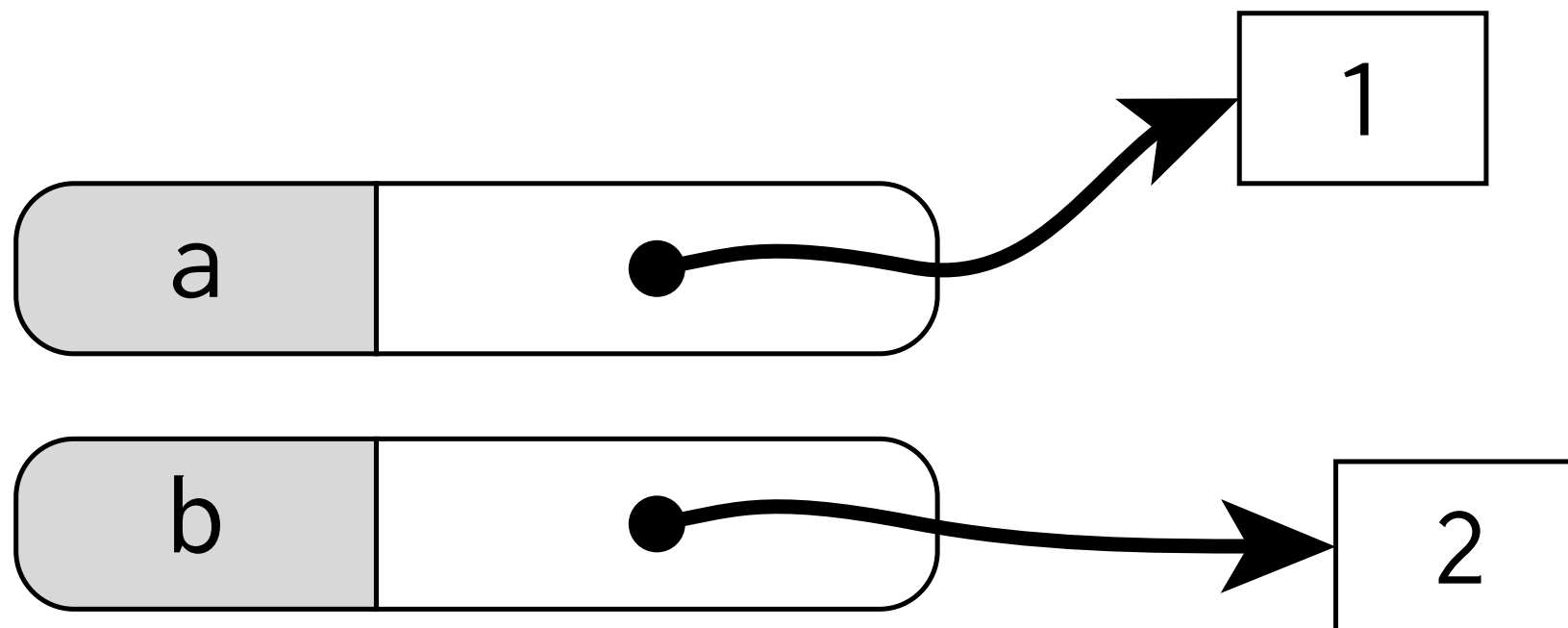
```
int[] a = { 1 };
int[] b = { 2 };
a = b;
b[0] = 3;
println( a[0] + b[0] );
```
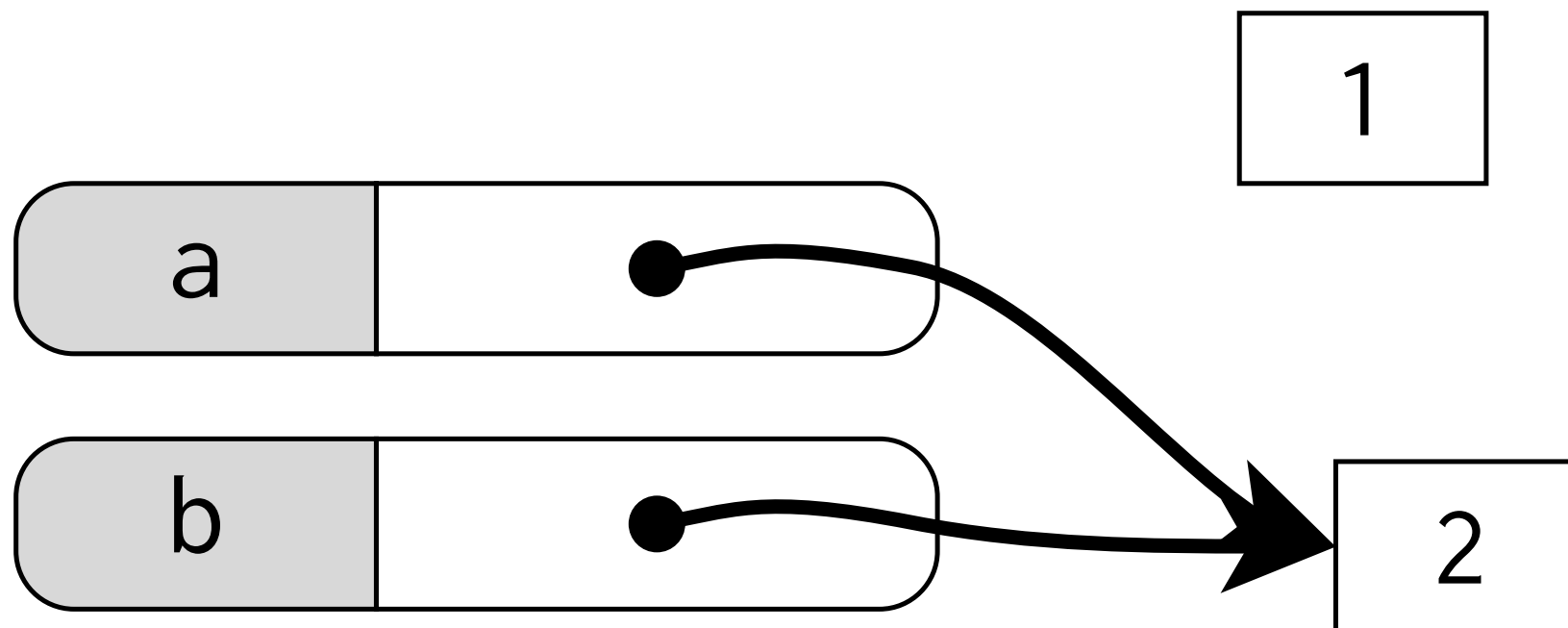
```
int[] a = { 1 };
int[] b = { 2 };
a = b;
b[0] = 3;
println( a[0] + b[0] );
```

```
int[] a = { 1 };
int[] b = { 2 };
a = b;
b[0] = 3;
println( a[0] + b[0] );
```
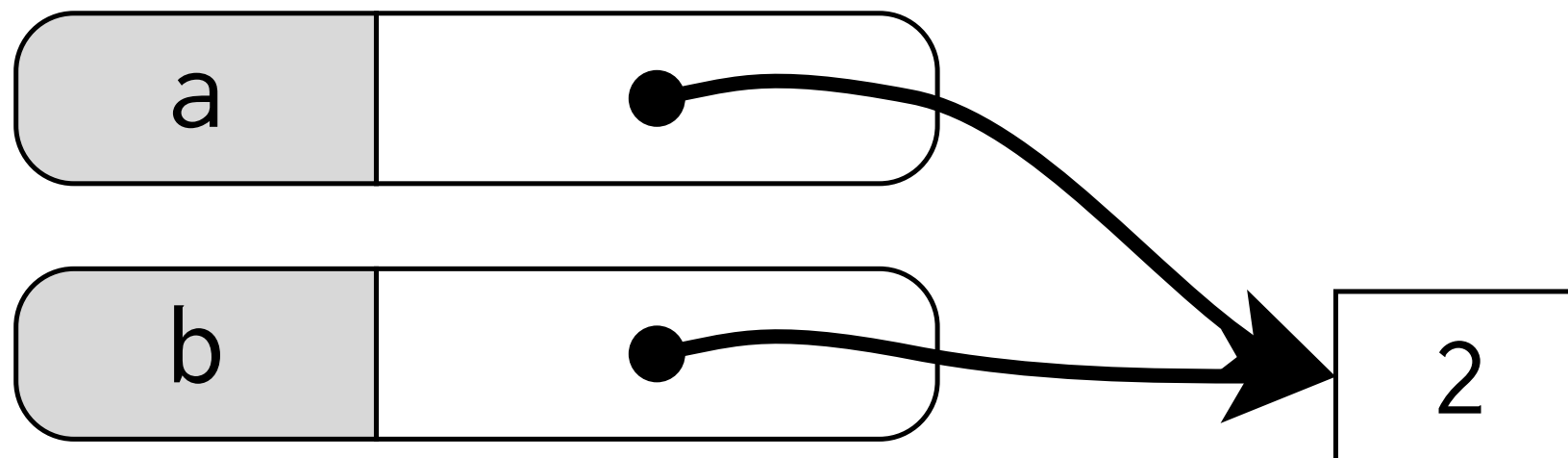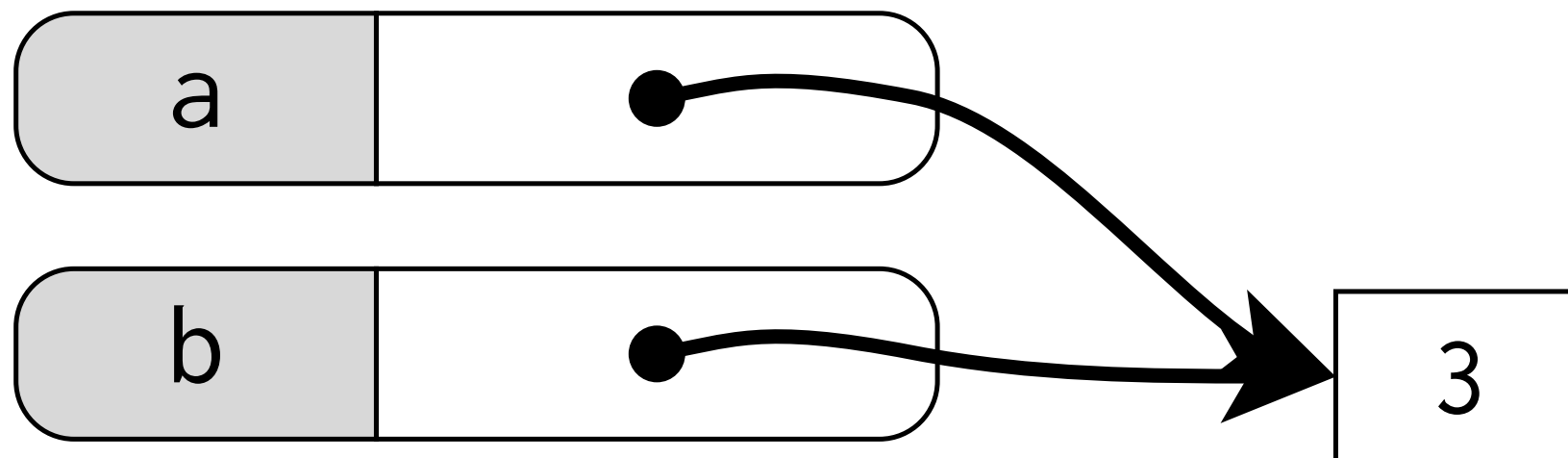
```
int[] a = { 1 };
int[] b = { 2 };
a = b;
b[0] = 3;
println( a[0] + b[0] );
```

```
int[] a = { 1 };
int[] b = { 2 };
a = b;
b[0] = 3;
println( a[0] + b[0] );
```

References

The base types `int`, `float`, `boolean`, and `char` are "primitive": their values are "naked" and copied around directly.

All other types (arrays and objects, including `String`) are passed around by reference (arrows).

Functions on arrays

1. Distillation

There are many natural operations on arrays that involve "reducing" the array down to a single value:

- Largest element
- Smallest element
- Is X in the array?
- Find the index of X
- Sum of elements
- Average of elements
- Number of positive elements

```
float distill( float[] arr )
{
  float result = ░░░░░░
  for( int idx = 0; idx < arr.length; ++idx ) {
    result = ░ ░░░░░░░░░ ░ arr[idx] ░░░ result ░░░░░░░░
  }

  return result;
}
```

```
float largestElement( float[] arr )
{
  float result = arr[0];
  for( int idx = 0; idx < arr.length; ++idx ) {
    if( arr[idx] > result ) {
      result = arr[idx];
    }
  }

  return result;
}
```

```java
float largestElement( float[] arr )
{
  float result = arr[0];
  for( int idx = 1; idx < arr.length; ++idx ) {
    if( arr[idx] > result ) {
      result = arr[idx];
    }
  }

  return result;
}
```

This is a common enough operation that `max()` and `min()` already work on arrays of numbers.

```
float[] arr = { 1.0, 4.2, -129832, PI, 2.718 };
println( max( arr ) );
println( min( arr ) );
```

Functions on arrays

2. Generation

Sometimes we want to conjure an array from nothing. We can do that in a function that takes values as input an returns an array.

Example: write a function that takes an integer `n` as input and produces the integer array `{0, 1, 2, …, n-1}`.

```
int[] upto( int n )
{
  int[] ret = new int[ n ];
  for ( int idx = 0; idx < n; ++idx ) {
    ret[idx] = idx;
  }
  return ret;
}
```

Functions on arrays

3. Transformation

Often we want to transform an array element-by-element into a new array. Sort of a combination of distillation and generation.

```
Type2[] transform( Type1[] arr )
{
  Type2[] ret = new Type2[ arr.length ];

  for( int idx = 0; idx < arr.length; ++idx ) {
    ret[idx] =            arr[idx]
  }

  return ret;
}
```

```
int[] badArrayClone( int[] arr )
{
  return arr;
}
```

```java
int[] badArrayClone( int[] arr )
{
  return arr;
}

int[] goodArrayClone( int[] arr )
{
  int[] ret = new int[ arr.length ];

  for( int idx = 0; idx < arr.length; ++idx ) {
    ret[idx] = arr[idx];
  }

  return ret;
}
```

Growing an array

Exercise: add one new element to the end of an array.

There's no way to grow an array "in place". Instead, we have to produce a new array that has all the original elements together with the new one.

Growing an array

The built-in function `append()` **adds a single new element to an array, returning the enlarged array.**

```
int[] arr1 = { 1, 2, 3, 4 };
float[] arr2 = { 1.2, 3.4, 5.6, 7.8 };

arr1 = append( arr1, 5 );
arr2 = append( arr2, cos( 2.0 * PI / 5.0 ) );
```

Casting

The `append()` **function and other array functions tend to work fine with built-in types, but "need help" with other types.**

```
class Circle
{ … }

Circle[] circs = …
circs = append( circs, new Circle( 10, 20, 30 ) );
```

Type mismatch, "java.lang.Object" does not match with "sketch_170108c.Circle[]"

Casting

A "casting operator" is a way to remind Processing of what type you're working with in cases where it forgets.

```
Circle[] circs = …
circs =
  (Circle[])append( circs, new Circle( 10, 20, 30 ) );
```

Force the expression that follows to be treated as an array of Circle**s.**

Other occasionally useful array operations:

```
int[] a = { 6, 3, 4, 1, 2, 5 };
int[] b = { 5, 6, 7 };
```

concat(a, b) ⇒

6	3	4	1	2	5	5	6	7

reverse(a) ⇒

5	2	1	4	3	6

shorten(b) ⇒

5	6

sort(a) ⇒

1	2	3	4	5	6

subset(a, 2, 3) ⇒

4	1	2

Strings

In many programming situations, we want to deal with blocks of text.

- Text boxes in a web form
- Text drawn to the screen
- Analyzing text documents for patterns

We need a type to hold blocks of text. Processing includes the type `String`, which is inherits from Java.

Strings and characters

A *character* is one symbol or letter in a string, including whitespace and other control characters. Characters are represented using the built-in type `char`.

Literals

To give an explicit character (a *literal*), put it in single quotes.

```
char a = 'a';
char b = 'd';
char c = ' ';
char d = '*';
```

To give an explicit string, put it in double quotes.

```
String name = "Jyn Erso";
String title = "Rogue One: A Star Wars Story";
String line = "It was the best of times, it was the worst of times.";
```

```
println( "mouse is pressed" );

img = loadImage( "bird.png" );
```

String literals

Special characters

And now the leather-covered sphere came hurtling through the
    air,
And Casey stood a-watching it in haughty grandeur there.
Close by the sturdy batsman the ball unheeded sped—
"That ain't my style," said Casey. "Strike one!" the umpire said.

```
String lastline = ""That ain't my style," sai
```

Ernest Lawrence Thayer, *Casey at the Bat* **(1888)**

Special characters

Use the backslash \ to tell Processing about upcoming special characters.

```
char single_quote = '\'';    // Only in chars
String double_quote = "\"";  // Only in strings
char newline = '\n';          // Like pressing return
char uni = '\u2603';          // 16-bit Unicode
```

Special characters

Use the backslash \ to tell Processing about upcoming special characters.

```
char single_quote = '\'';      // Only in chars
String double_quote = "\"";    // Only in strings
char newline = '\n';           // Like pressing return
char uni = '\u2603';           // 16-bit Unicode
char backslash = '\\';
```

\ ———————— BACKSLASH
\\ ———————— REAL BACKSLASH
\\\ ———————— *REAL REAL BACKSLASH*
\\\\ ———————— ACTUAL BACKSLASH, FOR REAL THIS TIME
\\\\\ ———————— ELDER BACKSLASH
\\\\\\\ ———————— BACKSLASH WHICH ESCAPES THE SCREEN AND ENTERS YOUR BRAIN
\\\\\\\\\ ———————— BACKSLASH SO REAL IT TRANSCENDS TIME AND SPACE
\\\\\\\\\\ ———————— BACKSLASH TO END ALL OTHER TEXT
\\\\\\\\\\\\\\\... — THE TRUE NAME OF BA'AL, THE SOUL-EATER

```
String lines = "Close by the
    sturdy batsman the ball
    unheeded sped—\n\"That ain't my
    style,\" said Casey. \"Strike
    one!\" the umpire said.";
```

This would be one long line in your program!

Strings are just values

```
String str1 = "Hello";
String str2 = str1;

String processString( String str, float val )
{
  …
}

String str3 = processString( str1, 3.14 );

String[] columns = { "Doric", "Ionic", "Corinthian" };
```

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};              String wd = "…";
```

```
char[] wd = {'h','e','l','l','o'};
```

```
String wd = "hello";
```

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};
int len = wd.length;
char c = wd[2];
wd[4] = '!';
```

```
String wd = "…";
```

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};
int len = wd.length;
char c = wd[2];
wd[4] = '!';
```

```
String wd = "…";
int len = wd.length();
```

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};
int len = wd.length;
char c = wd[2];
wd[4] = '!';
```

```
String wd = "…";
int len = wd.length();
char c = wd.charAt(2);
```

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};
int len = wd.length;
char c = wd[2];
wd[4] = '!';
```

```
String wd = "…";
int len = wd.length();
char c = wd.charAt(2);
```
NOTHING!

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};
int len = wd.length;
char c = wd[2];
wd[4] = '!';
```

```
String wd = "…";
int len = wd.length();
char c = wd.charAt(2);
```
NOTHING!

Strings are *immutable*: once you create one, you can't change its contents. Instead, assign a new string to the same variable.

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};
int len = wd.length;
char c = wd[2];
wd[4] = '!';
```

```
String wd = "…";
int len = wd.length();
char c = wd.charAt(2);
```
NOTHING!

```
char[] wd3 = concat( wd1, wd2 );
```

String **vs.** char[]

Strings wish they were arrays of characters, but they aren't quite. Still, your knowledge of arrays will help you.

```
char[] wd = {…};
int len = wd.length;
char c = wd[2];
wd[4] = '!';
```

```
String wd = "…";
int len = wd.length();
char c = wd.charAt(2);
```
NOTHING!

```
char[] wd3 = concat( wd1, wd2 );
```

```
String str3 = str1 + str2;
```

Concatenating strings

The + **operator on strings is very flexible.**

```
"Call me" + " " + "Ishmael."

"Ours go to " + 11

"The value of PI is " + PI

"A " + true + " or " + false + " question"

float x, y;
"The point is at (" + x + ", " + y + ")"
```

Parsing strings

We often obtain "raw text" from external sources, and need to *parse* it into meaningful data.

The built-in functions `int()` **and** `float()` **work on strings and arrays of strings.**

```
int a = int( "1234" );
float b = float( "567.89" );

String[] strs = { "-81", "0", "36" };
int[] arr = int( strs );
```

String equality

We often want to compare two strings to see whether they have the same text. The `String` class has an `equals()` method for that purpose.

```java
if( str1.equals( str2 ) ) {
  // The strings are equal.
}
```

String equality

We often want to compare two strings to see whether they have the same text. **The** `String` **class has an** `equals()` **method for that purpose.**

```
if( str1.equals( str2 ) ) {
    // The strings are equal.
}
```

WARNING! The following is legal code, but probably not what you want.

```
if( str1 == str2 ) ) {
    // What can go wrong?
}
```
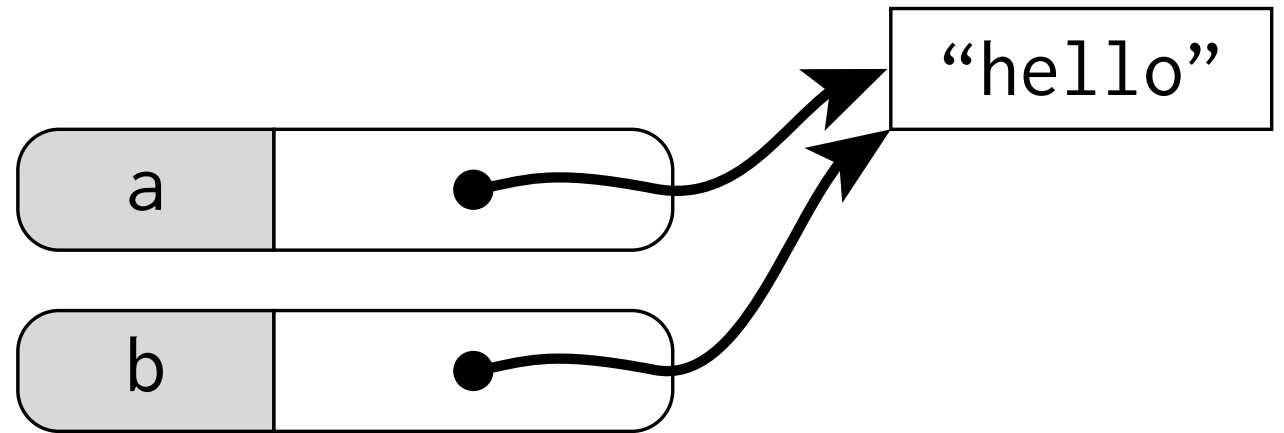
```
String s = ”He”;
println( ”Hello” );
println( s + ”llo” );
println( ”Hello” == (s+”llo”) );
```
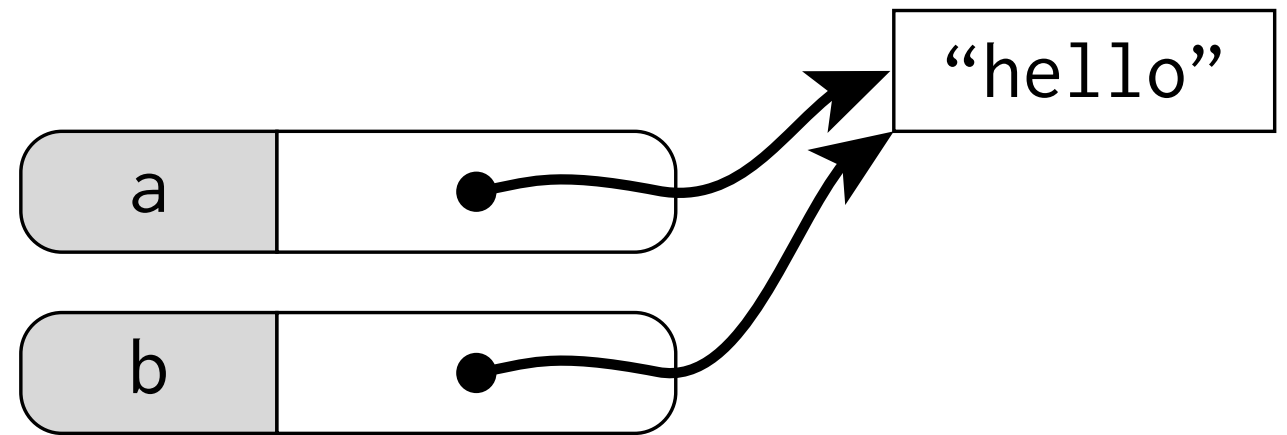
```
String a = "Hello";
String b = a;
```
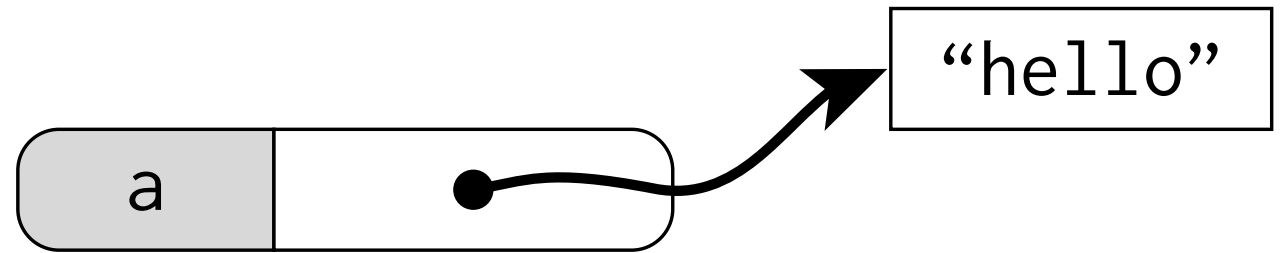


```
String a = "Hello";
String b = "Hello";
```

```
String a = "Hello";
String b = a;
```



```
String a = "Hello";
String b = "Hello";
```
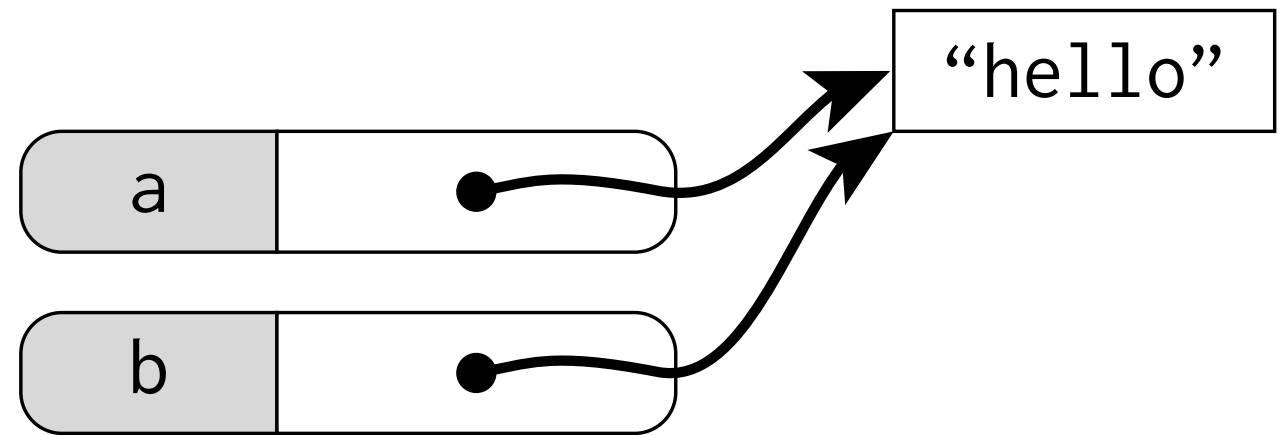
```java
String a = "Hello";
String b = a;
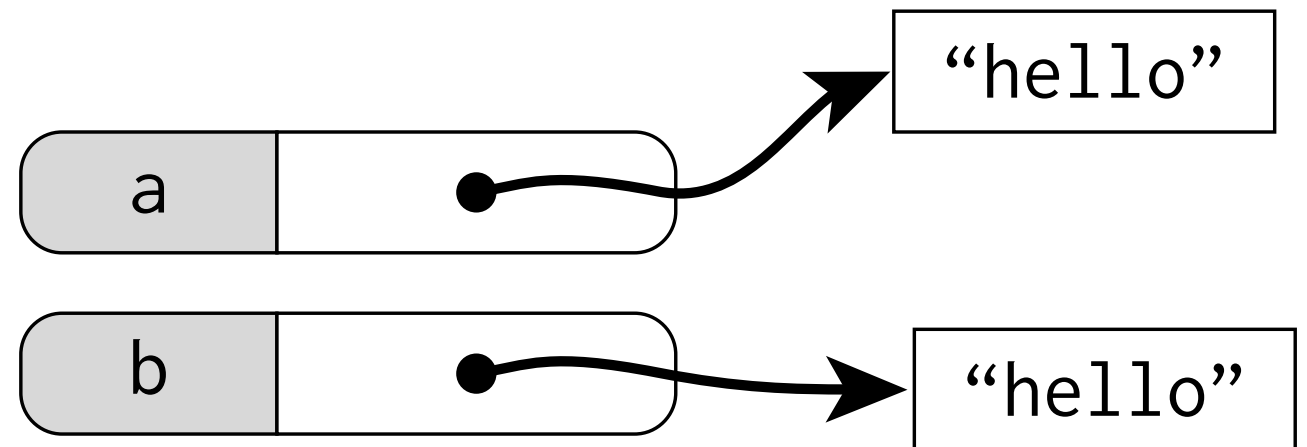```



```java
a.equals( b ) ✔
a == b ✔
```

```java
String a = "Hello";
String b = "Hello";
```

```java
String a = "Hello";
String b = a;
```

a.equals(b) ✔
a == b ✔

```java
String a = "Hello";
String b = "Hello";
```
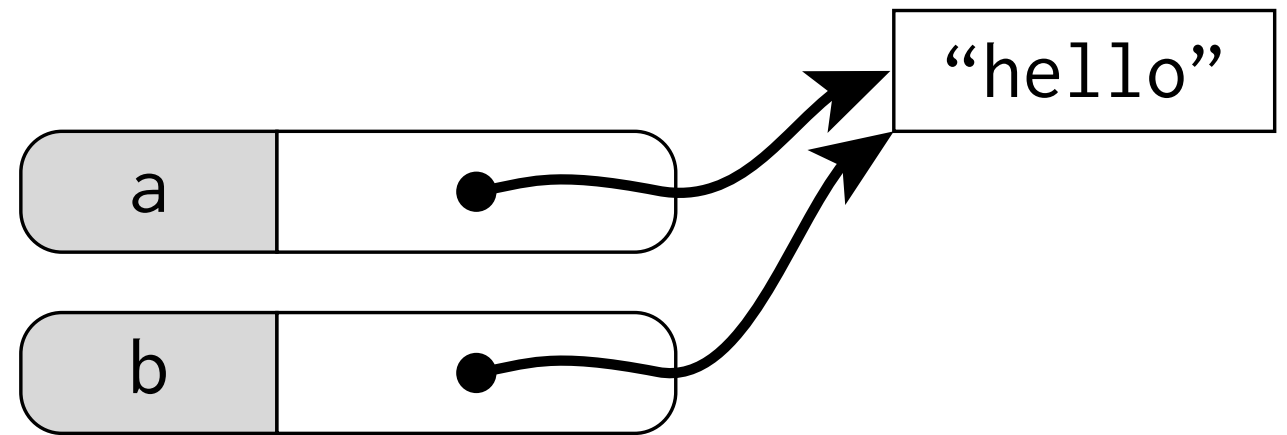
```
String a = "Hello";
String b = a;
```



```
a.equals( b ) ✔
a == b ✔
```

```
String a = "Hello";
String b = "Hello";
```
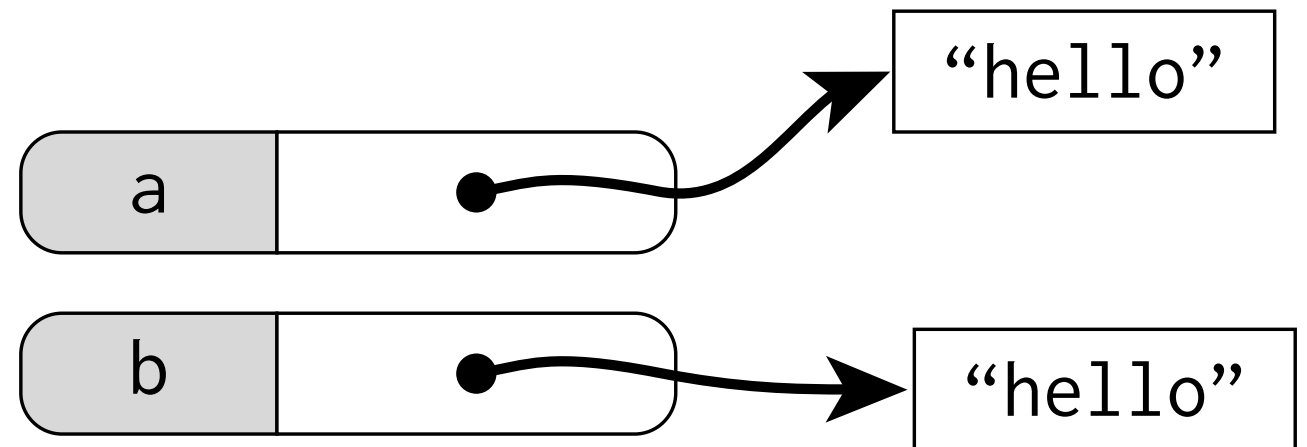


```
a.equals( b ) ✔
a == b ✗
```

The `.equals()` method checks if two strings have the same characters.

The `==` operator checks if they're the same string in the computer's memory.

(A bit like == vs. === in Javascript?)

Outputting text

The built-in `println()` function will write any text (or really, any value at all) to the console. Handy for debugging!

The built-in `text()` function will draw text at a given position in the sketch window, using the current fill colour.

See also `textSize()`, `textFont()`, `createFont()`, `textAlign()`.

```
void setup()
{
  size( 275, 400 );

  textSize( 72 );
  colorMode( HSB, 255 );
  background( 0, 0, 255 );
  for ( int y = 80; y < 380; y += 15 ) {
    fill( map( y, 80, 380, 0, 255 ),
       255, 255 );
    text( "CS 106", 10, y );
  }
}
```