EECS 1710
Programming for Digital Media

Week 2 :: Programming Basics

YORK U
UNIVERSITÉ
UNIVERSITY

This Week

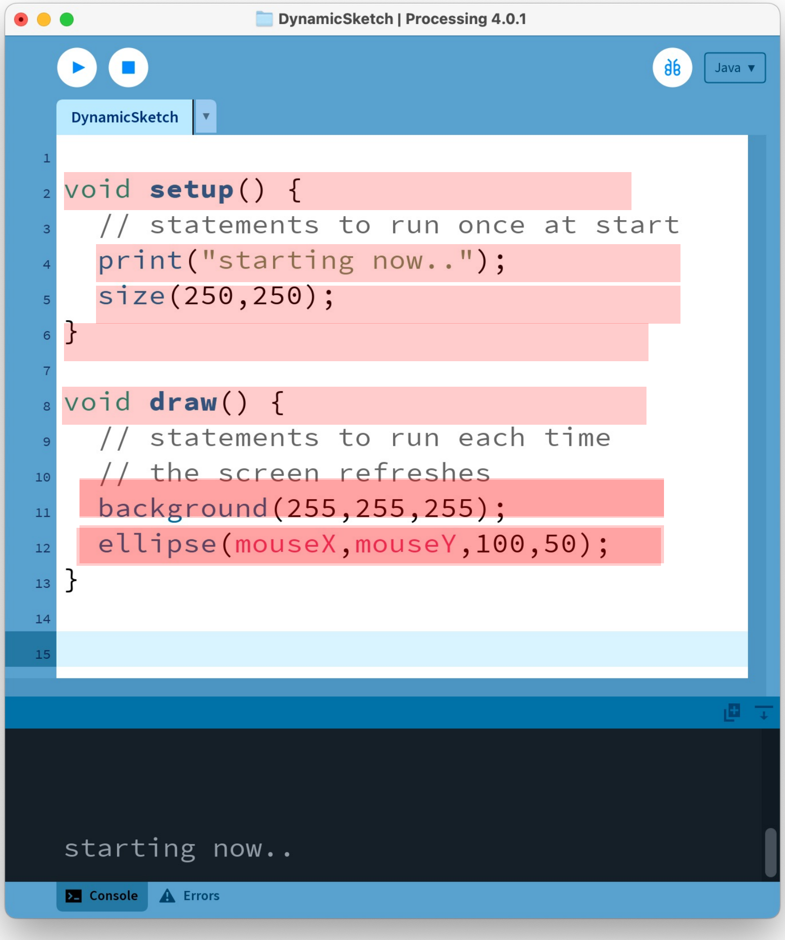
Lecture 2:

- Anatomies of a processing sketch
- Language elements & running a program
- Coordinate system in Processing
- Some drawing commands
- Tracing a program

Lecture 3:

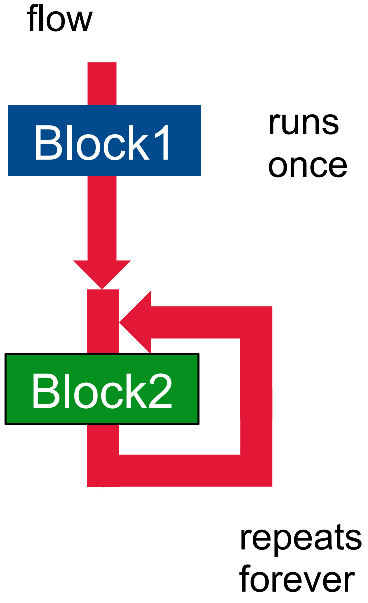
- Variables & Data Types
- Declaration and Assignment

Tracing DynamicSketch.pde?



```

void setup() {
  // statements to run once at start
  print("starting now..");
  size(250,250);
}

void draw() {
  // statements to run each time
  // the screen refreshes
  background(255,255,255);
  ellipse(mouseX,mouseY,100,50);
}


```

```
starting now..
```



Variables

- Variables are identifiers we create (names) for containers that will store certain types of data values

- We can create our own, or utilize some *pre-defined* variables that processing provides for us

- (mouseX, mouseY) are *pre-defined* variables that hold the current mouse position → i.e. the (x,y) position of the cursor on the application window (in image coordinates)

- This is useful as we can cause changes in our drawings by moving the mouse!

Simple program with our own variables

```
// Area.pde

/*
    a simple program to compute and store the area
    of a rectangle and displaying it to the console
 */

 int rectWidth;
 rectWidth = 8;

 int rectHeight;
 rectHeight = 3;

 int area = rectWidth * rectHeight;

 print("Area = ");
 println(area);
```

Topics

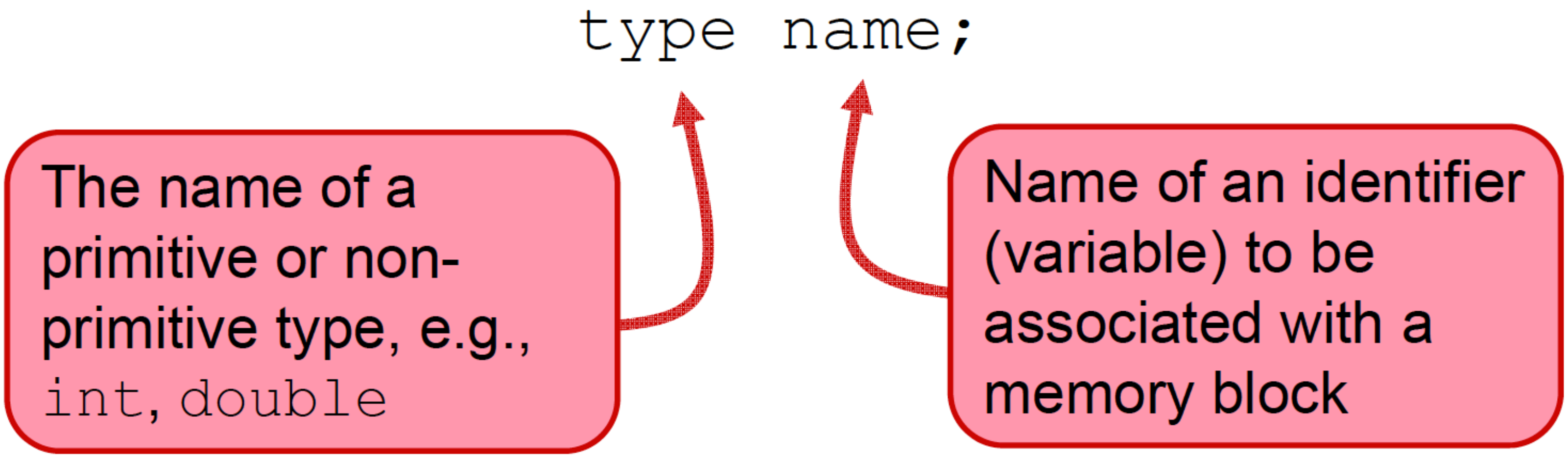
- Anatomy of a program
- The declaration statement
- The assignment statement

Declaration Statement

- The statement (from `Area.pde`)

```
int rectWidth;
```

is of the general form



```
type name;
```

The name of a primitive or non-primitive type, e.g., `int`, `double`

Name of an identifier (variable) to be associated with a memory block

Variable Scope

- Variables have *scope*
- A variable’s scope is the variable’s *enclosing block*
- The variable is not known outside of its scope

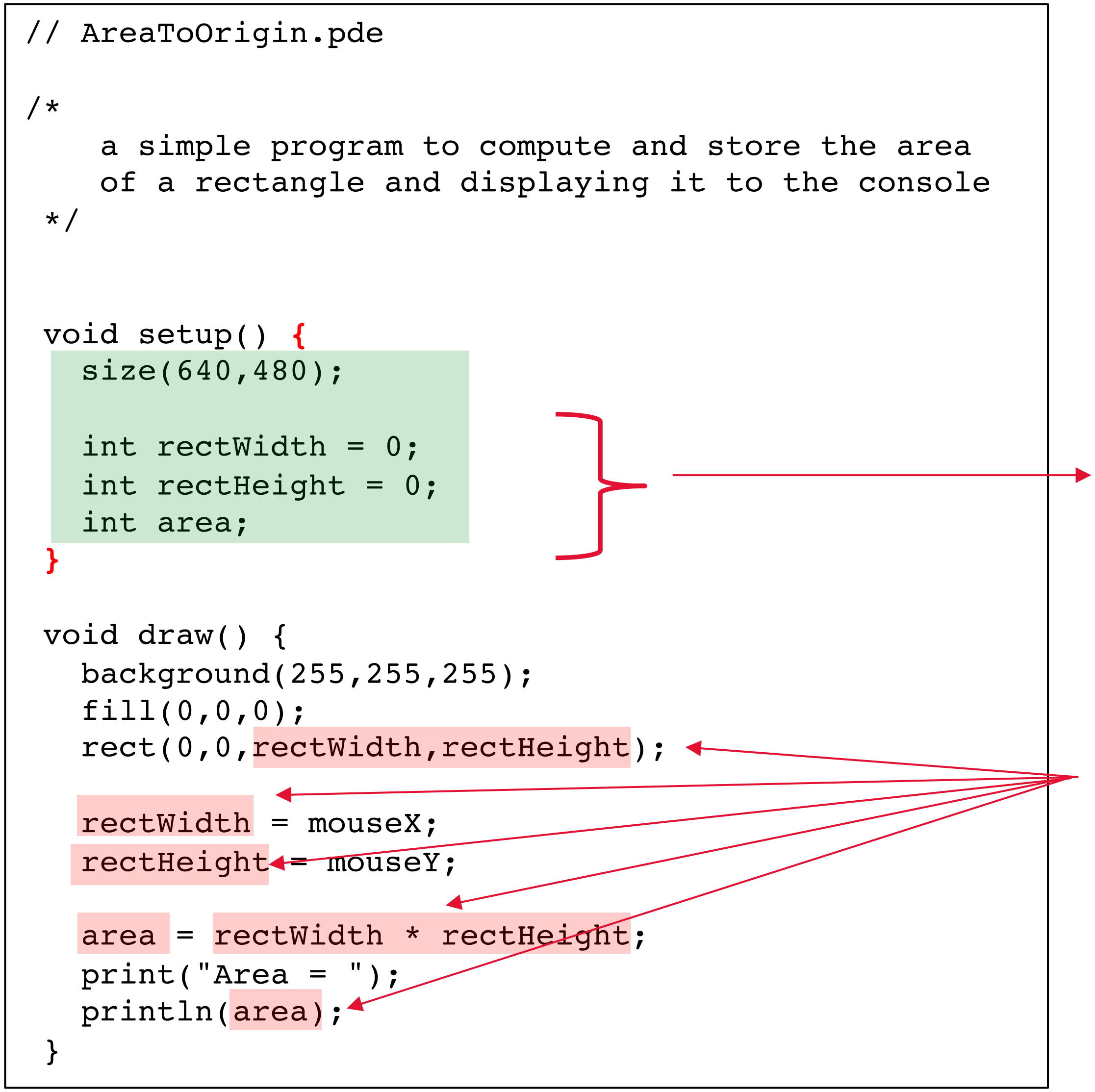
Another version of Area

```
// AreaToOrigin.pde

/*
    a simple program to compute and store the area
    of a rectangle and displaying it to the console
 */

 int rectWidth = 0;
 int rectHeight = 0;
 int area;

 void setup() {
   size(640,480);
 }

 void draw() {
   background(255,255,255);
   fill(0,0,0);
   rect(0,0,rectWidth,rectHeight);

   rectWidth = mouseX;
   rectHeight = mouseY;

   area = rectWidth * rectHeight;
   print("Area = ");
   println(area);
 }
```

Scope?

= all blocks (defined outside)

Another version of Area



```
// AreaToOrigin.pde

/*
    a simple program to compute and store the area
    of a rectangle and displaying it to the console
 */


 void setup() {
   size(640,480);

   int rectWidth = 0;
   int rectHeight = 0;
   int area;
 }

 void draw() {
   background(255,255,255);
   fill(0,0,0);
   rect(0,0,rectWidth,rectHeight);

   rectWidth  = mouseX;
   rectHeight = mouseY;

   area = rectWidth * rectHeight;
   print("Area = ");
   println(area);
 }
```

What if defined here?

Scope =
`void setup() { // here }`

all of these don’t exist inside void draw() { .. }. !!

Another version of Area

```
// AreaToOrigin.pde

/*
    a simple program to compute and store the area
    of a rectangle and displaying it to the console
 */


 void setup() {
   size(640,480);

}

 void draw() {
   int rectWidth = 0;
   int rectHeight = 0;
   int area;

   background(255,255,255);
   fill(0,0,0);
   rect(0,0,rectWidth,rectHeight);

   rectWidth = mouseX;
   rectHeight = mouseY;

   area = rectWidth * rectHeight;
   print("Area = ");
   println(area);
 }
```

How about here?

Variable Names

- Rules and guidelines for names of variables
  - Must be an identifier
  - Must not be in the scope of another variable with the same name
  - A good name reflects the content stored in the variable
  - Style
    - Use lowercase letters, but for multi-word names, capitalize the first letter of each subsequent word

Integer Types

- A type is a range of values and a set of operations on these values
- Operators: `+` (add), `–` (subtract), `*` (multiply), `/` (divide), `%` (remainder)
- Variations

Type	Range	Memory size
`byte`	≈ ±100	1 byte (= 8 bits)
`short`	≈ ±30,000	2 bytes (= 16 bits)
`int`	≈ $\pm 2x10^9$	4 bytes (= 32 bits)
`long`	≈ $\pm 9x10^{18}$	8 bytes (= 64 bits)

Default literal

As a literal, L or l suffix (e.g., `long x = 5L;`)

Exact Range

Type	Bits	Low	High
`byte`	8	-2^{7}	$2^{7} - 1$
`short`	16	-2^{15}	$2^{15} - 1$
`int`	32	-2^{31}	$2^{31} - 1$
`long`	64	-2^{63}	$2^{63} - 1$

Quick primer on number systems!

  - What is a bit?
  - What is a byte??

- Basic Number Systems:
  - Decimal vs. Binary?

Basics of Data Representation

- What do computers understand?
  - Numbers – in fact, even less.. just high/low (on/off) voltages

- What is the concept of an “encoding”?
  - Uses high/low to “encode” things
    - how many things can be encoded with a single “wire”?
    - Multiple “wires”, encode more things (numbers, symbols, etc)

Imagine a “wire”
in 1 of 2 states:

● has a voltage (ON)

○ no voltage (OFF)

Each state can represent a (symbolize) a different thing:
Therefore 2 things can be represented (e.g. 2 digits)?

How many wires needed to represent 10 digits (0,1,2,…,9)?

An encoding is a way of storing information

- We can store information in such encodings!
  - 10 digits requires 10 combinations of on/off
    - 1 wire = 2 combinations ● ○
    - 2 wires = 4 combinations ○○ ●○ ○● ●●
    - 3 wires = 8 combinations ○○○ ●○○ ○●○ … ●●○ ●●●
    - 4 wires = 16 combinations

need at least 4 "wires" to represent 10 digits

  - on/off voltages (***bits***) are the most basic unit of information understood by a computer
  - A (***byte***) is a set of 8 bits!
  - numbers can be used to compute & store new numbers:
    - 2 + 4
    - 13 * 5 + (8 – 2)/3

Decimal vs Binary Encoding:

Decimal: (10 digit system/ base 10)

1 0 3 6

$$= 1 * 1000 \quad + 0 * 100 \quad + 3 * 10 \quad + 6 * 1$$

$$= 1 * 10^3 \quad + 0 * 10^2 \quad + 3 * 10^1 \quad + 6 * 10^0$$

$$= 1036$$

Binary: (2 digit system/ base 2)

1 1 1 0

$$= 1 * 2^3 \quad + 1 * 2^2 \quad + 1 * 2^1 \quad + 0 * 2^0$$

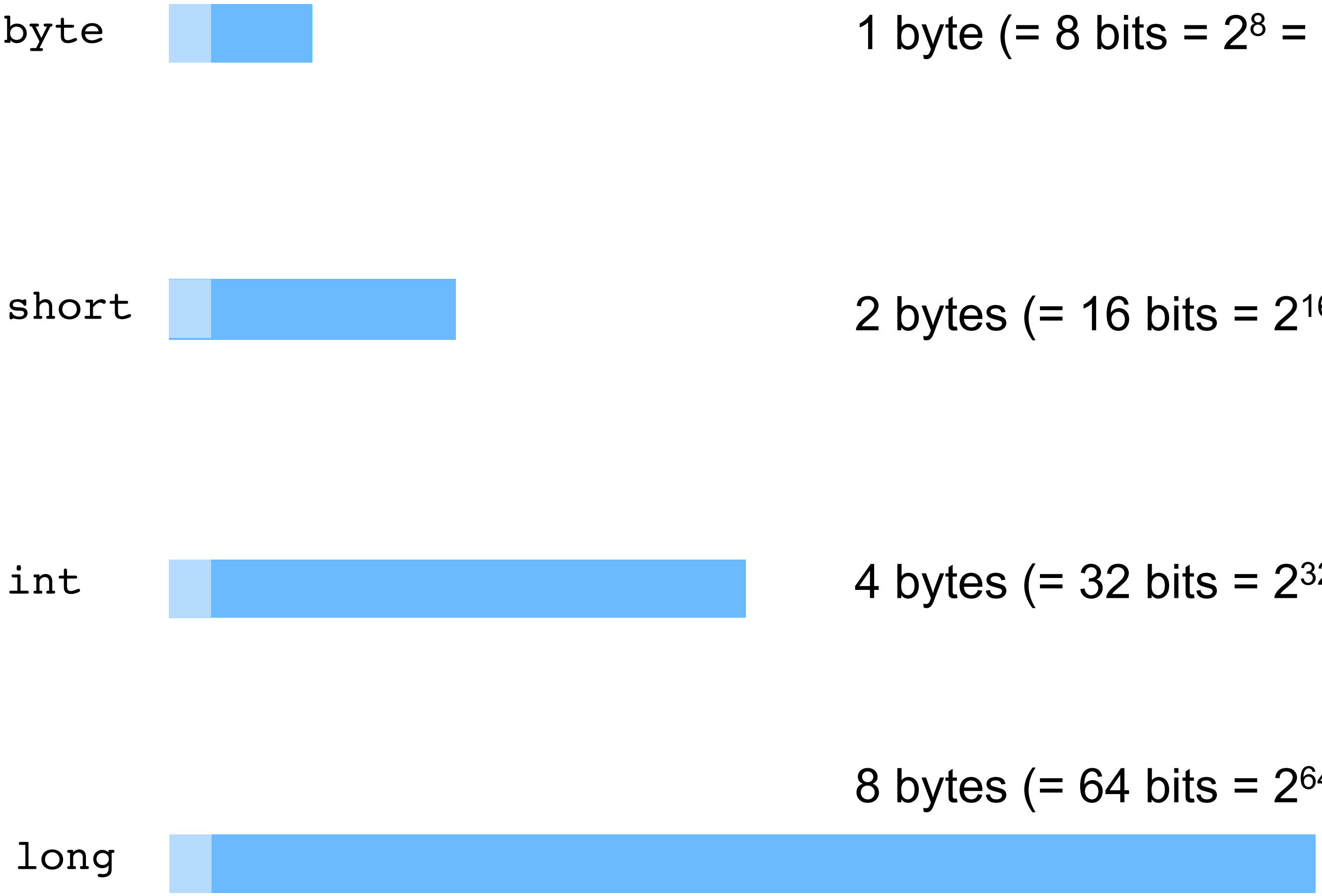
$$= 8 \quad + 4 \quad + 2 \quad + 0$$

= 14 (decimal equivalent)

Exact Range

Type	Bits	Low	High
`byte`	8	-2^{7}	$2^{7} - 1$
`short`	16	-2^{15}	$2^{15} - 1$
`int`	32	-2^{31}	$2^{31} - 1$
`long`	64	-2^{63}	$2^{63} - 1$

Comparison of Integer types (*bits* used)



1 byte (= 8 bits = 2^8 = 256 values)

2 bytes (= 16 bits = 2^{16} = 65,536 values)

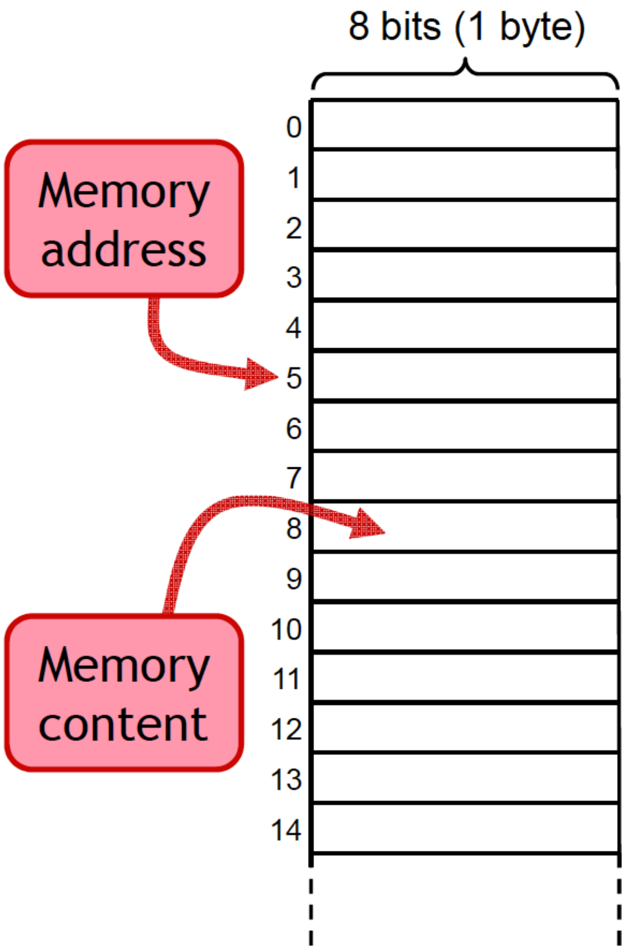
4 bytes (= 32 bits = 2^{32} = a lot!)

8 bytes (= 64 bits = 2^{64} = even more!)

A simple model of computer memory: (analogy of a theatre)

- Theatre: memory block (storage – X number of seats)
- Seats: memory element (individual location in theatre)
- People: values (temporarily resides in a seat)
- Tickets: variables (an identifier connecting name to seat)

Computer Memory



- Memory is viewed as a one-dimensional arrangement of cells
- Each cell is 8 bits (*Note*: 1 byte = 8 bits)
- The total number of cells is the size of the memory
- Size is articulated in multiples of…
  - Kilobyte (1 KB = 1024 bytes)
  - Megabyte (1 MB = 1024 KB)
  - Gigabyte (1 GB = 1024 MB)
  - *Note*: 2^{10} = 1024
- Memory addresses start at 0 and extend upward (see figure at left)

Declaration and Memory

- With the declaration

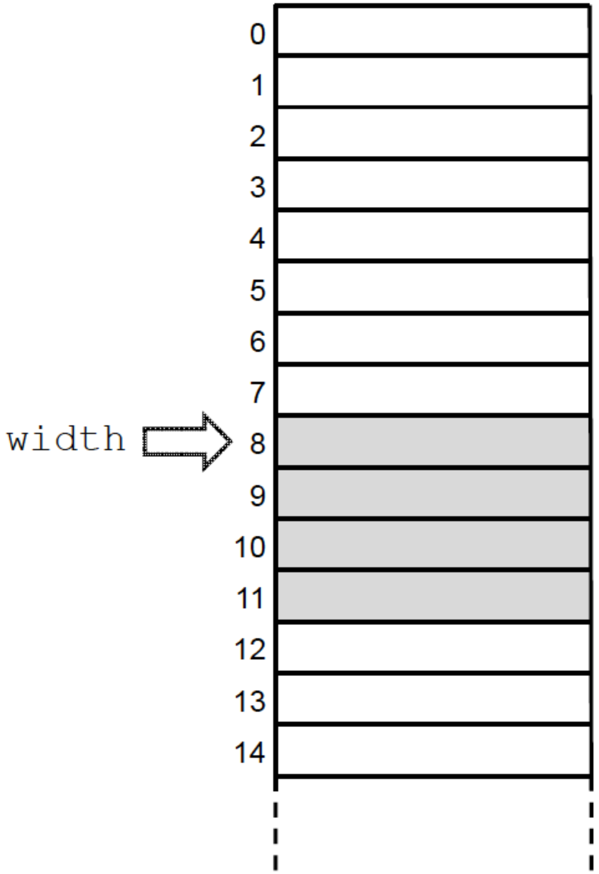
```
int rectWidth;
```

the compiler will set aside a 4-byte (32-bit) block of memory (see right)

- The compiler has a symbol table, which will have an entry such as

Identifier	Type	Block Address
rectWidth	int	8

- *Note*: No initialization is involved; there is only an association of a name with an address.



Variables & Type

- Two categories of variable (sometimes called types):

1. PRIMITIVE TYPES (built in)

   e.g.

numeric:	`int`, `long`, `float`, `double`, etc.
other:	`boolean`, `char`

2. NON-PRIMITIVE TYPES (user defined/composite)

   e.g.

   `String`

Java Keywords

Reserved words:

abstract	assert				
boolean	break	byte			
case	catch	char	class	const	continue
default	do	double			
else	enum	extends			
final	finally	float	for		
goto					
if	implements	import	instanceof	int	interface
long					
native	new				
package	private	protected	public		
return					
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	
void	volatile				
while					

Literals: `true, false, null`

Numeric types

`int`, `long`, **`float`**, **`double`**, etc.

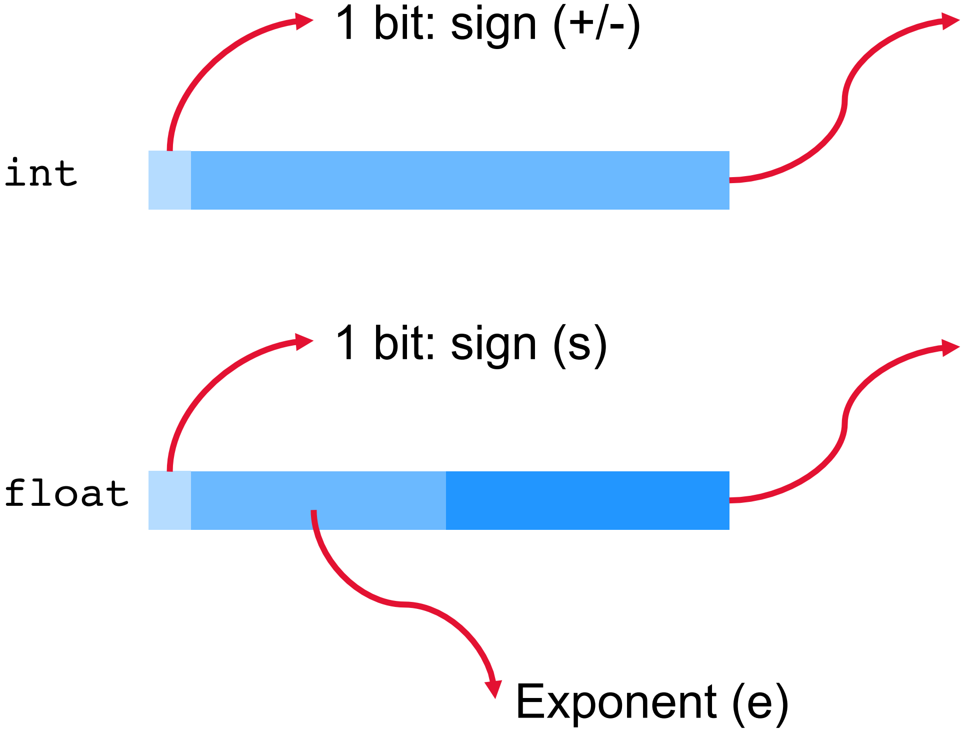
Real (decimal) types

Reals (format, storage, range)

- Format
  - Formatted according to the IEEE-754 standard for floating point arithmetic
  - Includes a fractional part and a power
- Storage
  - `float` → 4 bytes
  - `double` → 8 bytes
- Range
  - `float` → $\pm 10^{38}$ with 7 significant digits
  - `double` → $\pm 10^{308}$ with 15 significant digits

How can `float` and `int` encode different ranges using same number of bits??

- Answer:
  - Different representations! (i.e. bits configured differently)



$$\boxed{(-1)^{s} * (m) * B^{e}}$$

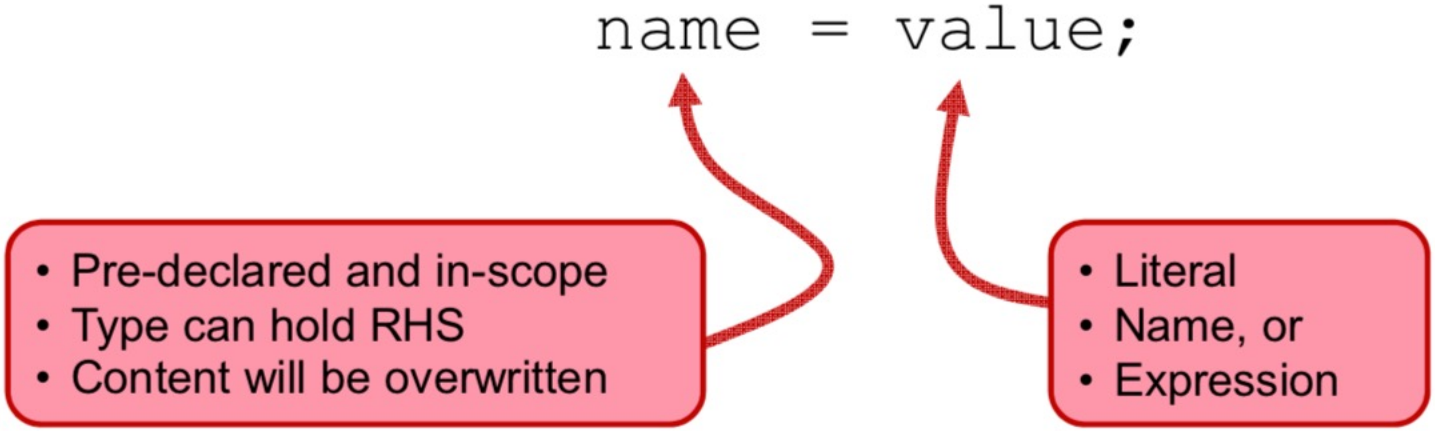
where B = base (e.g. 2 or 10)

Assignment

- The statement (from `Area.pde`)

```
rectWidth = 8;
```

is of the general form



```
name = value;
```

- Pre-declared and in-scope
- Type can hold RHS
- Content will be overwritten

- Literal
- Name, or
- Expression

Note: RHS = right-hand side, LHS = left-hand side

Assigning Literals to Real Types

```
double x;

double interestRate = 1.5;

float z = -1.1f;

double abc = 3.4E-5;
```

Float literal (default is double)

Same as

`0.000034`

= 3.4 x 10^{-5}

Expressions & Operators

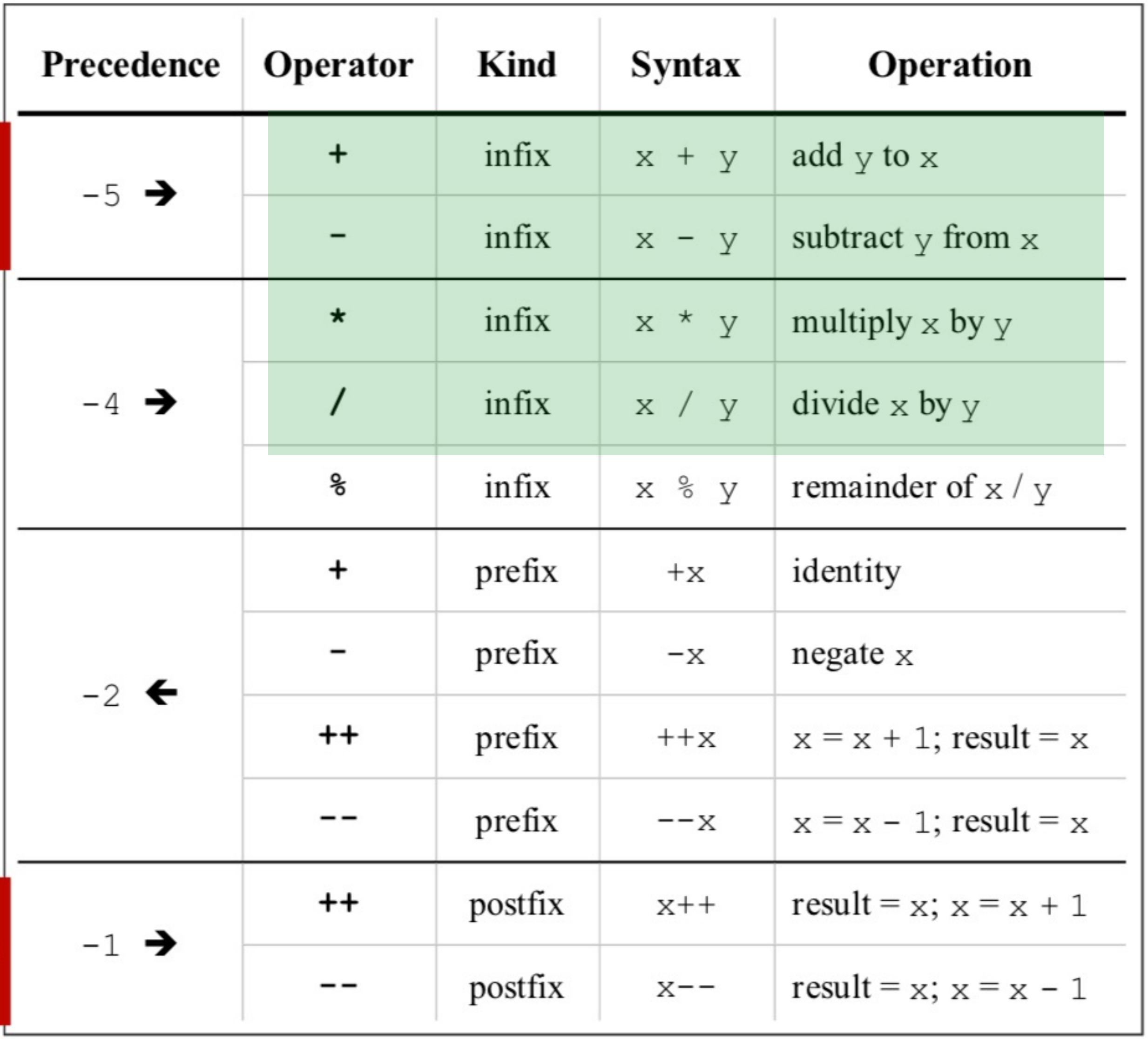
- *Expressions* involve one or more data values that appear together with *operators*
- *Operators* define specific actions on data
- *Operators* are usually specific to a given type
  - E.g. standard operators + - * / in general, work on integer and real types
  - Their function may differ slightly depending on the type they are operating on

- Expressions are typically processed from left to right (though there are exceptions that give some operators precedence over others)

int arithmetic operators (summary)

Lowest priority

Highest priority



Precedence	Operator	Kind	Syntax	Operation
-5 →	+	infix	x + y	add y to x
	-	infix	x - y	subtract y from x
-4 →	*	infix	x * y	multiply x by y
	/	infix	x / y	divide x by y
	%	infix	x % y	remainder of x / y
-2 ←	+	prefix	+x	identity
	-	prefix	-x	negate x
	++	prefix	++x	x = x + 1; result = x
	--	prefix	--x	x = x - 1; result = x
-1 →	++	postfix	x++	result = x; x = x + 1
	--	postfix	x--	result = x; x = x - 1

Special Cases

- What happens if…
  - Division by zero
    - Integers: throws an arithmetic exception
    - Reals: assigns a fictitious value, `NaN` (“not a number”)
  - Out of range result
    - Integers: range is treated as circular
    - Reals: assigns a fictitious value, `Infinity`

Strong/Weak Types

- Java is considered a “strongly typed” language
  - When you create a variable, its type MUST be specified
  - Only values (data) of the same type may be assigned to that variable
  - Less ambiguous

- Some languages (e.g. python) are “weakly typed”
  - Type does not need to be specified
  - Can assign any values (data types) to the variable
  - More ambiguous

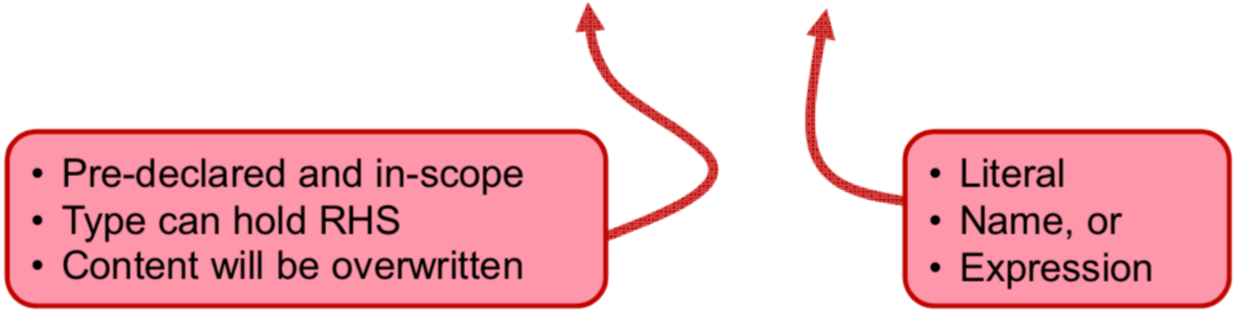
Assignment

- The statement (from Area.java)

```
width = 8;
```

is of the general form

```
name = value;
```

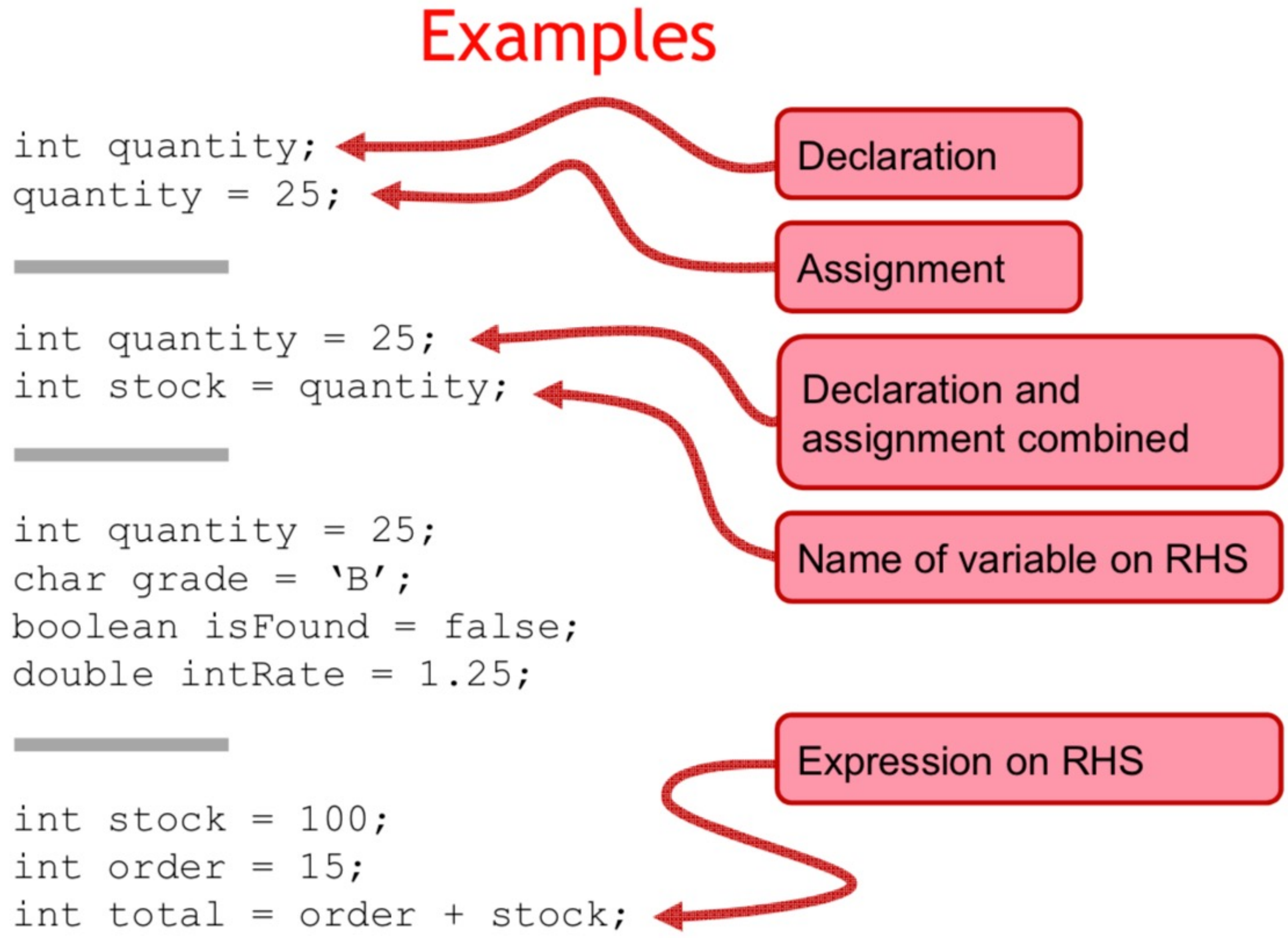


- Pre-declared and in-scope
- Type can hold RHS
- Content will be overwritten

- Literal
- Name, or
- Expression

Note: RHS = right-hand side, LHS = left-hand side

Assignment



Examples

```
int quantity;
quantity = 25;
```

```
int quantity = 25;
int stock = quantity;
```

```
int quantity = 25;
char grade = ‘B’;
boolean isFound = false;
double intRate = 1.25;
```

```
int stock = 100;
int order = 15;
int total = order + stock;
```

Declaration

Assignment

Declaration and assignment combined

Name of variable on RHS

Expression on RHS

Coming up…

- Other primitive types
- More operators
- Operator precedence
- More Expressions
- The String type
- Heterogeneous Expressions