Conditionals and Loops

Review

- Primitive Data Types & Variables
  - int, long
  - float, double
  - boolean
  - char
- String
- Mathematical operators: `+ - * / %`
- Comparison: `< > <= >= ==`

A Foundation for Programming

any program you might want to write

objects

functions and modules

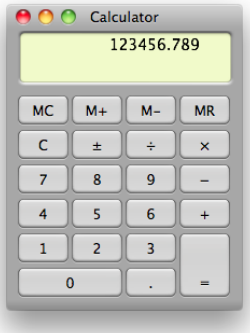
graphics, sound, and image I/O

arrays

conditionals and loops

Math | text I/O

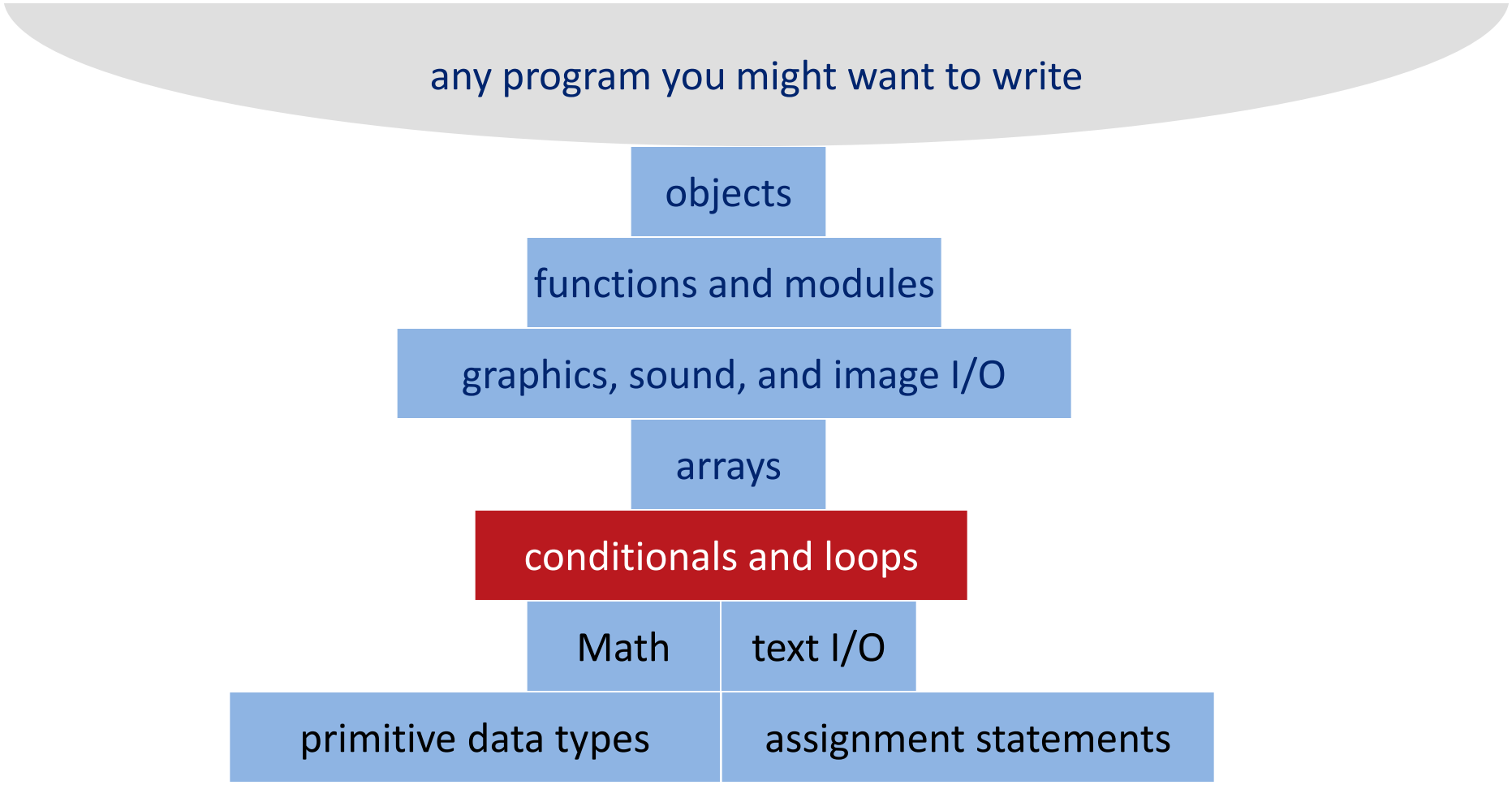
primitive data types | assignment statements



last lecture: equivalent to a calculator

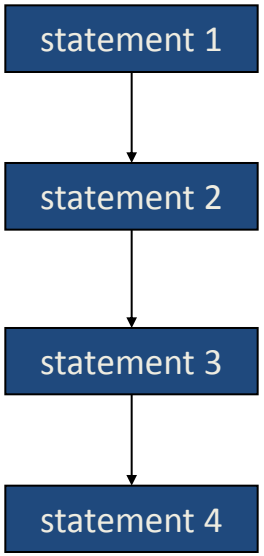
Penn Engineering

A Foundation for Programming

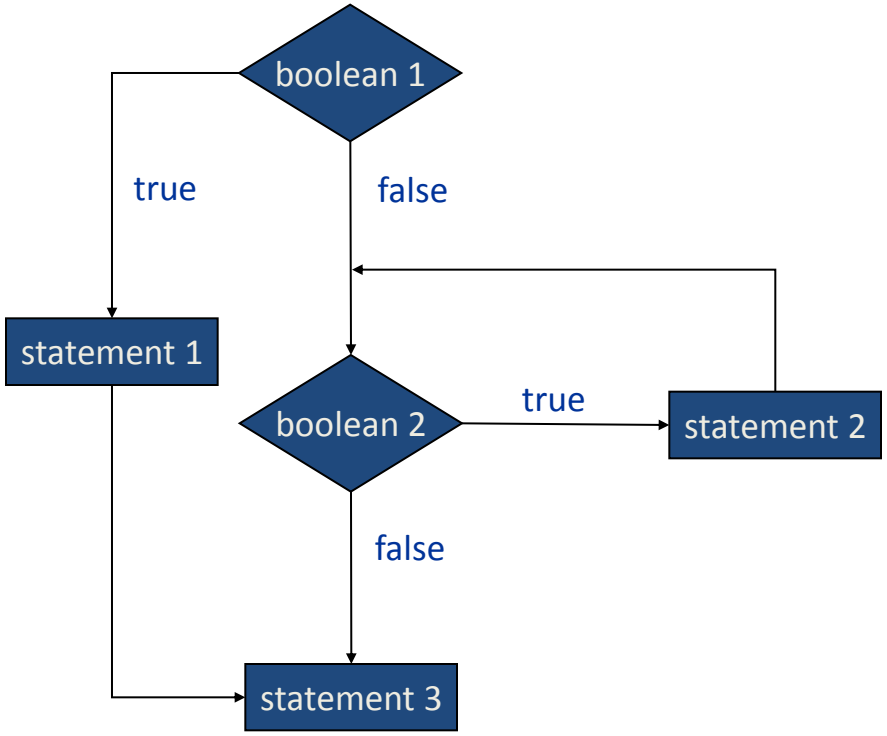


Control Flow

- Programs execute one statement after another
- Conditionals and loops allow us to control the flow

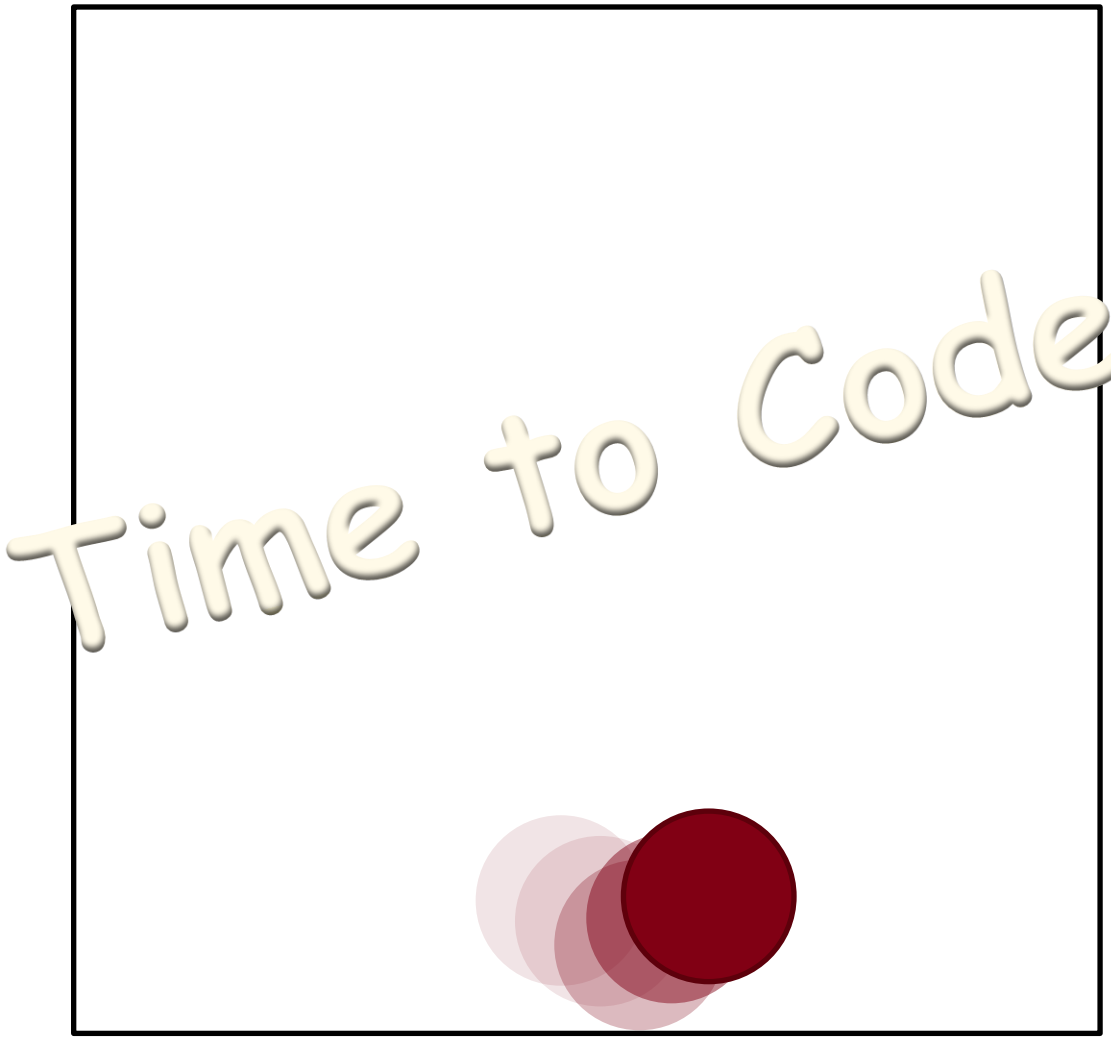


straight-line control flow



control flow with conditionals and loops

In-Class Demo: Bouncing Ball



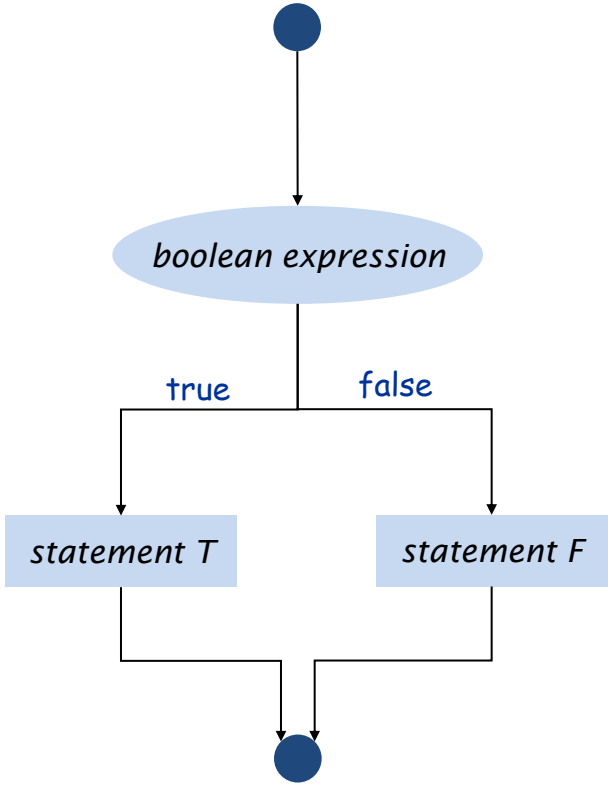
Conditionals

If Statement

- The **`if`** statement: A common branching structure
  - Evaluate a **`boolean`** expression
  - If **`true`**, execute some statements
  - If **`false`**, execute other statements

```
if (boolean expression) {
    //statement T;
}
else {
    //statement F;
}
```

can be any sequence of statements



Penn Engineering

Relational Expressions

<	less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equivalent
!=	is not equivalent

Relational Expressions: Examples

```
1. if ( true ) { … }
2. if ( 10 > 10 ) { … }
3. if ( 10 >= 10 ) { … }
4. if ( 'a' == 'a' ) { … }
5. if ( 'a' != 'a' ) { … }
```

Logical Expressions

&& logical conjunction (and)

- both expressions must be true for conjunction to be true

|| logical disjunction (or)

- either expression must be true for disjunction to be true

! logical negation (not)

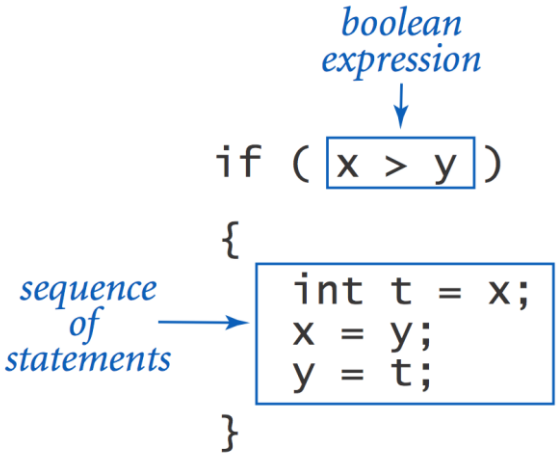
- true $\rightarrow$ false, false $\rightarrow$ true

Logical Expression Examples

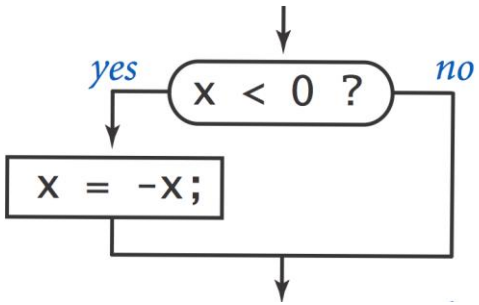
```
1.  if ( (2 > 1) && (3 > 4) ) { … }
2.  if ( ('b' == 'b') && (1 + 2 == 3) ) { … }
3.  if ( !false ) { … }
4.  if ( !(1 < -1) ) { … }
5.  if ( !(10 < 20) || false  ) { … }
6.  if ( !(10 > 20) && (10 < 20) ) { … }
7.  if ( (true || false) && true ) { … }
8.  if ( (true && false) || true ) ) { … }
9.  …
```

If Statement

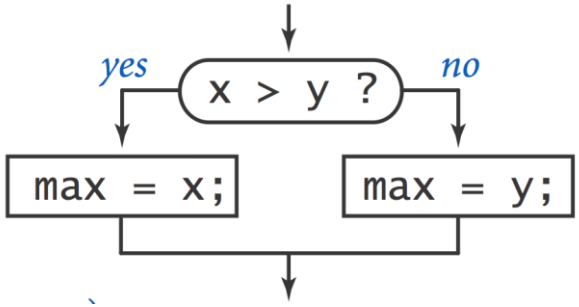
- The **`if`** statement: A common branching structure
  - Evaluate a **`boolean`** expression
  - If **`true`**, execute some statements
  - If **`false`**, execute other statements




```
if (x < 0) x = -x;
```




```
if (x > y) max = x;
else       max = y;
```



If Statement

- Ex. Take different actions depending on the value of a variable

```
void setup() {
}

void draw() {
    if (Math.random() < 0.5) {
        println("Heads");
    } else {
        println("Tails");
    }
}
```

OUTPUT

```
Heads
Heads
Tails
Heads
...
```

If Statement Examples

absolute value	`if (x < 0) x = -x;`

put x and y into sorted order	`if (x > y)`
	`{`
	`  int t = x;`
	`  x = y;`
	`  y = t;`
	`}`

maximum of x and y	`if (x > y) max = x;` `else  max = y;`
error check for division operation	`if (den == 0) System.out.println("Division by zero");` `else  System.out.println("Quotient = " + num/den);`

error check for quadratic formula	`double discriminant = b*b - 4.0*c;`
	`if (discriminant < 0.0)`
	`{`
	`  System.out.println("No real roots");`
	`}`
	`else`
	`{`
	`  System.out.println((-b + Math.sqrt(discriminant))/2.0);`
	`  System.out.println((-b - Math.sqrt(discriminant))/2.0);`
	`}`

Equations of Motion (Simplified)

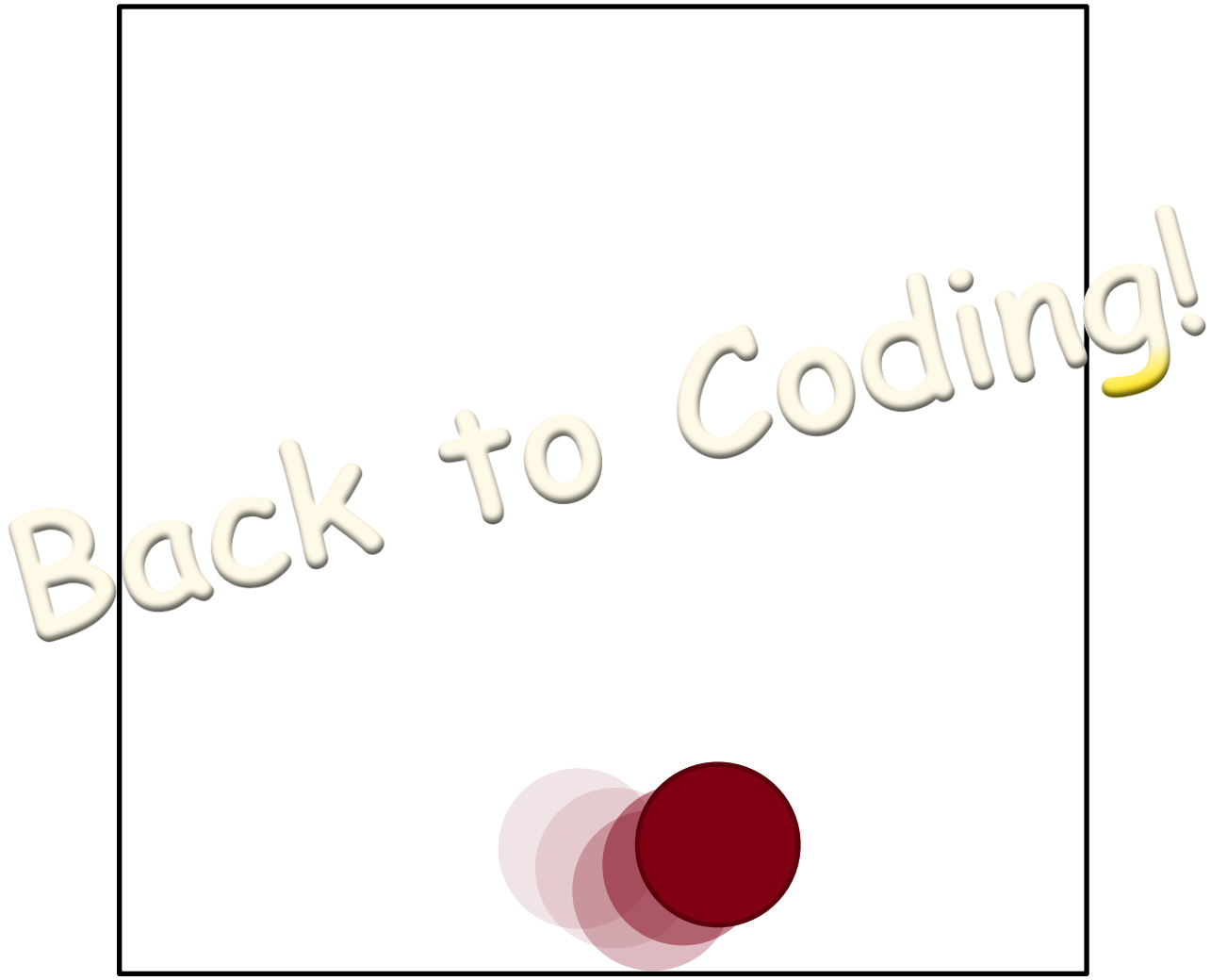
s = displacement
t = time
v = velocity
a = acceleration

- Constant acceleration (a)

  $s_{i+1} = s_i + v_i \Delta t$

  $v_{i+1} = v_i + a \Delta t$

In-Class Demo: Bouncing Ball



Conditionals: if-else-if-statement

```
if ( boolean_expression_1 ) {
        statements;
} else if ( boolean_expression_2 ) {
        statements;
} else if ( boolean_expression_3 ) {
        statements;
} else {
        statements;
}
```

Example: Graduated Income Tax

Pay a certain income tax rate depending on income:

Income	Rate
0 - 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

5 mutually exclusive alternatives

Nested If Statements

Use nested **`if`** statements to handle multiple alternatives

```
if (income <  47450) rate = 0.22;
else {
   if (income < 114650) rate = 0.25;
   else {
      if (income < 174700) rate = 0.28;
      else {
         if (income < 311950) rate = 0.33;
         else rate = 0.35;
      }
   }
}
```

Income	Rate
0 - 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

Nested If Statements

Income	Rate
0 - 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

5 mutually exclusive alternatives

Alternative shortened version:

```
double rate;
if      (income <  47450) rate = 0.22;
else if (income < 114650) rate = 0.25;
else if (income < 174700) rate = 0.28;
else if (income < 311950) rate = 0.33;
else                      rate = 0.35;
```

Nested If Statements

What is wrong with the following implementation?

Income	Rate
0 - 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

5 mutually exclusive alternatives

```
double rate = 0.35;
if (income <  47450) rate = 0.22;
if (income < 114650) rate = 0.25;
if (income < 174700) rate = 0.28;
if (income < 311950) rate = 0.33;
```

Conditionals: switch-statement

- Works like a if-else statement.
- Convenient for large numbers of value tests

```
switch( expression ) {
  case label1:          // label1 equals expression
     statements;
     break;
  case label2:          // label2 equals expression
     statements;
     break;
  default:              // Nothing matches
     statements;
}
```

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

void draw() {}

void keyPressed() {
  switch(key) {
    case 'l':
    case 'L':
      println("Turning left");
      break;
    case 'r':
    case 'R':
      println("Turning right");
      break;
  }
}
```

What does this do?

```
int positionX = 250;
int positionY = 250;
int deltaX = 0;
int deltaY = 0;

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

void draw() {
  background(255);

  positionX = positionX + deltaX;
  positionY = positionY + deltaY;

  if (positionX < 0)
    positionX = 0;
  if (positionX > width)
    positionX = width;
  if (positionY < 0)
    positionY = 0;
  if (positionY > height)
    positionY = height;

  ellipse(positionX, positionY, 50, 50);
}
```

```
void keyPressed() {
    switch (keyCode) {
    case 37:
      deltaX = -2;
      deltaY = 0;
      break;
    case 39:
      deltaX = 2;
      deltaY = 0;
      break;
    case 38:
      deltaX = 0;
      deltaY = -2;
      break;
    case 40:
      deltaX = 0;
      deltaY = 2;
      break;
    case 32:
      deltaX = 0;
      deltaY = 0;
      break;
  }
}
```

An aside … Operators

`+`, `-`, `*`, `/` and …

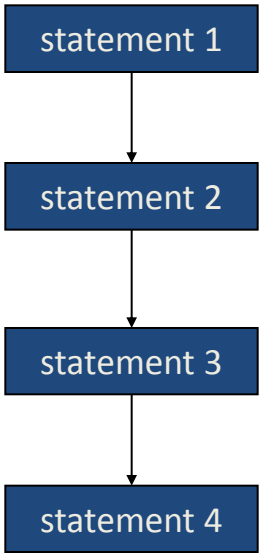
```
i++;      equivalent to  i = i + 1;
i += 2;   equivalent to  i = i + 2;
i--;      equivalent to  i = i - 1;
i -= 3;   equivalent to  i = i - 3;
i *= 2;   equivalent to  i = i * 2;
i /= 4;   equivalent to  i = i / 4;
```

`i % 3;` remainder after i is divided by 3 (modulo)

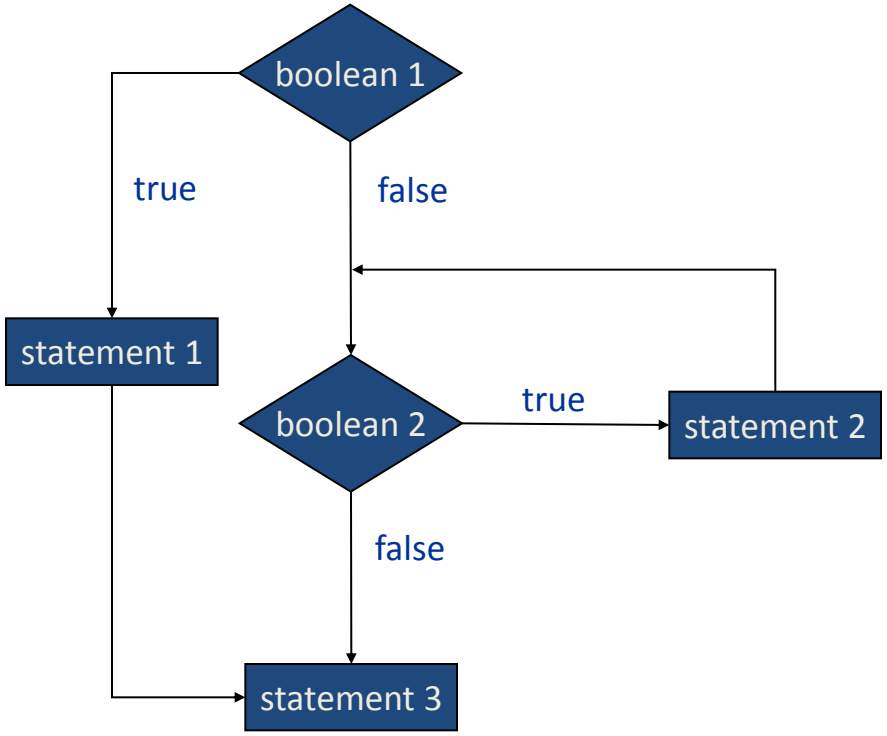
Iteration

Control Flow

- Programs execute one statement after another
- Conditionals and loops allow us to control the flow



straight-line control flow



control flow with conditionals and loops

Iteration

Repetition of a program block

- Iterate when a block of code is to repeated multiple times.

Options

- The while-loop
- The for-loop

The While Loop

While Loop

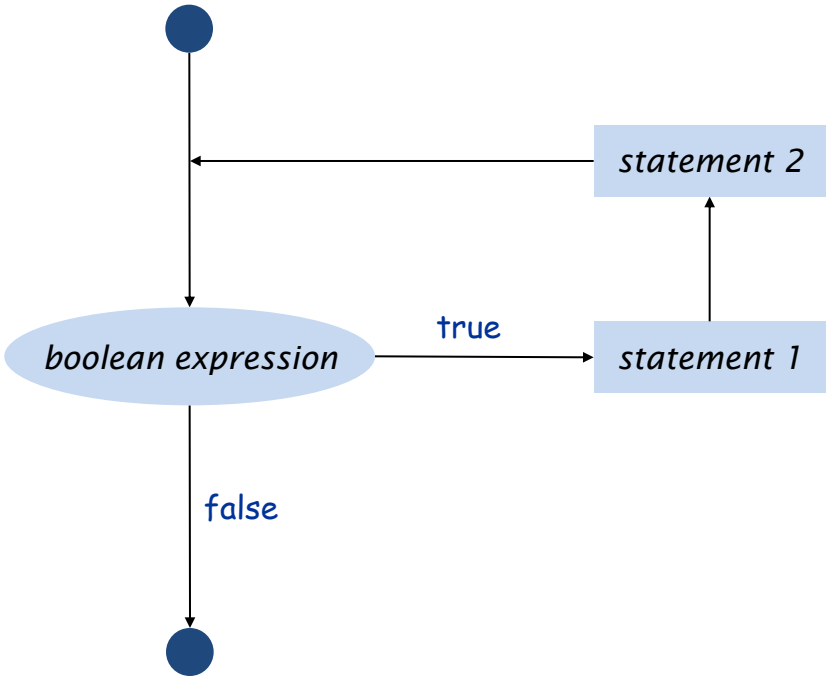
The **while** loop: A common repetition structure

- Evaluate a **boolean** expression
- If **true**, execute some statements
- Repeat

loop continuation condition

```
while (boolean expression) {
    statement 1;
    statement 2;
}
```

loop body



What will this do?

```
print("Program running");
while (true) {
    print(".");
}
println();
println("Program Exiting");
```

While Loop: Powers of Two

Example: Print powers of 2 that are $\leq 2^N$

- Increment **i** from **0** to **N**
- Double **v** each time

```
int i = 0;
int v = 1;
while (i <= N) {
  System.out.println(i + " " + v);
  i++;
  v = 2 * v;
}
```

Output:

```
0 1
1 2
2 4
3 8
4 16
```

N = 4

i	v	i <= N
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	false

While Loop Challenge

Q: Is there anything wrong with the following code for printing powers of 2?

```
int i = 0;
int v = 1;
int N = 4;
while (i <= N)
  println(i + " " + v);
  i = i + 1;
  v = 2 * v;
```

While Loop Challenge

Q: Is there anything wrong with the following code for printing powers of 2?

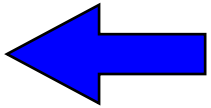
```
int i = 0;
int v = 1;
int N = 4;
while (i <= N)
  println(i + " " + v);
  i = i + 1;
  v = 2 * v;
```

A: Need curly braces around statements in while loop

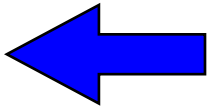
- otherwise it enters an infinite loop, printing "0 1"

The 3 Parts of a Loop

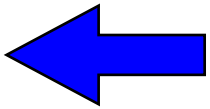
```
...
int i = 1 ;

// count from 1 to 100
while ( i < 101 ) {
   println("i") ;
   i = i + 1 ;
}
```



initialization of **loop control variable**



test of **loop termination condition**



modification of loop control variable

Example: Factorial

```
...
int factorial = 1 ;
while (myNumber > 0) {
      factorial *= myNumber ;
      myNumber-- ;
}
println(factorial) ;
```

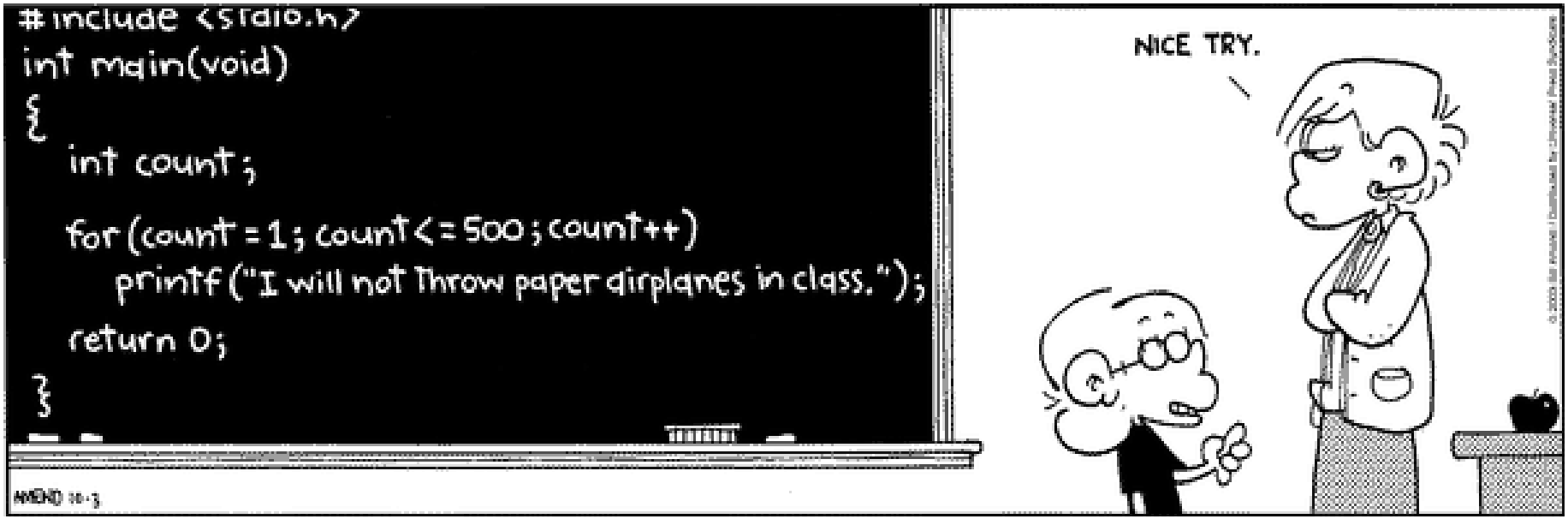
```
void setup() {
  size(500, 500);
  float diameter = 500.0f;
  while ( diameter > 1.0 ) {
      ellipse(width/2, height/2, diameter, diameter);
      diameter = diameter * 0.9;
  }
}
```

What does this do?

```
void setup() {
  size(500, 500);
  float diameter = 500.0f;
  while (true) {
      ellipse(width/2, height/2, diameter, diameter);
      diameter = diameter * 0.9;
      if (diameter <= 1.0) break;
  }
}
```

What about this?

The For Loop



For Loops

- Handles details of the counter-controlled loop "automatically"
- The for loop structure includes:
  - the initialization of the the loop control variable,
  - the termination condition test, and
  - control variable modification

```
for ( int i = 1 ; i < 101 ; i = i + 1 ) {

}
```

For Loop: Powers of Two

Example: Print powers of 2 that are $\leq 2^N$

- Increment **i** from **0** to **N**
- Double **v** each time

```
int v = 1;
for (int i = 0; i <= N; i++) {
  System.out.println(i + " " + v);
  v = 2 * v;
}
```

Output:

```
0 1
1 2
2 4
3 8
4 16
```

N = 4

v	i	i <= N
1	0	true
2	1	true
4	2	true
8	3	true
16	4	true
32	5	false

For Loop Examples

- A *for* loop that counts from 0 to 9:

```
// modify part can be simply “i++”
for ( i = 0;  i < 10;  i = i + 1 ) {
  System.out.println( i ) ;
}
```

- …or we can count backwards by 2’s :

```
// modify part can be “i -= 2”
for ( i = 10;  i > 0;  i = i - 2 ) {
  System.out.println( i ) ;
}
```

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

  float diameter = 500.0f;
  while ( diameter > 1.0 ) {
        ellipse(250, 250, diameter, diameter);
        diameter = diameter - 10.0;
  }
}
```

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

  for ( float diameter = 500.0f; diameter > 1.0; diameter -= 10.0 ) {
        ellipse(250, 250, diameter, diameter);
  }
}
```

When Does a *for* Loop Initialize, Test and Modify?

- Just as with a while loop, a for loop
  - initializes the loop control variable before beginning the first loop iteration
  - performs the loop termination test before each iteration of the loop
  - modifies the loop control variable at the **very end** of each iteration of the loop
- The for loop is easier to write and read for counter-controlled loops.

Loop Examples

print largest power of two less than or equal to N	`int v = 1;`
	`while (v <= N/2)`
	`  v = 2*v;`
	`System.out.println(v);`

compute a finite sum $(1 + 2 + \ldots + N)$	`int sum = 0;`
	`for (int i = 1; i <= N; i++)`
	`  sum += i;`
	`System.out.println(sum);`

compute a finite product $(N! = 1 \times 2 \times \ldots \times N)$	`int product = 1;`
	`for (int i = 1; i <= N; i++)`
	`  product *= i;`
	`System.out.println(product);`

print a table of function values	`for (int i = 0; i <= N; i++)` `  System.out.println(i + " " + 2*Math.PI*i/N);`

The *break & continue* Statements

- The `break` & `continue` statements can be used in **while** and **for** loops to skip the remaining statements in the loop body:
  - `break` causes the looping itself to abort
  - `continue` causes the next turn of the loop to start
    - In a **for** loop, the modification step will still be executed

Example: Break in a For-Loop

```
...
int i;
for (i = 1; i < 10; i = i + 1) {
    if (i == 5) {
        break;
    }
    System.out.print(i);
}
System.out.println("Broke out of loop at i = " + i);
```

OUTPUT:

1 2 3 4

Broke out of loop at i = 5

Example: Continue in a For-Loop

```
...
int i;
for (i = 1; i < 10; i = i + 1) {
    if (i == 5) {
        continue;
    }
    System.out.print(i);
}
System.out.println("Done");
```

OUTPUT:

1 2 3 4 6 7 8 9

Done

Problem: Continue in While-Loop

```
// This seems equivalent to for loop
// in previous slide—but is it??
...
int i = 1;
while (i < 10) {
    if (i == 5) {
        continue;
    }
    System.out.print (i);
    i = i + 1;
}
System.out.println("Done");
```

OUTPUT:

???

Variable Scope

***Variable scope*:**

- That set of code statements in which the variable is known to the compiler
- Where it can be referenced in your program
- Limited to the ***code block*** in which it is defined
  - A ***code block*** is a set of code enclosed in braces (***{ }***)

One interesting application of this principle allowed in Java involves the for loop construct

Scoping and the For-Loop Index

- Can declare and initialize variables in the heading of a for loop
- These variables are local to the for-loop
- They may be reused in other loops

```
int count = 1;
for (int i = 0; i < 10; i++){
   count *= 2;
}
 //using 'i' here generates a compiler error
```