Variables

- Processing, like many programming languages, uses variables to store information
- Variables are stored in computer memory with certain attributes
  - location — where in memory is the information stored
  - type — what sort of information is stored in that memory
    - sometimes the type is a primite value
    - sometimes the type is a reference to an aggregate called an object
  - name — how can we refer to that location of memory
    - can contain letters (a–z, A–Z), digits (0–9), and the characters $ and _
    - first character cannot be a digit

Primitive types

- An `int` type stores a 32-bit integer value: -1, 50000, 2012, etc (-2^{31} to $2^{31} - 1$)
- A `float` type stores real values using 32 bits: 1.0, 3.141592, -459.67, etc
- A `char` type stores single character values: 'A', 'l', 'b', 'i', 'o', 'n', etc
- A `boolean` type stores one of two values: true or false
- A `byte` type stores an 8-bit integer: -128 to 127
- A `short` type stores a 16-bit integer: -32768 to 32767
- A `long` type stores a 64-bit integer: -2^{63} to $2^{63} - 1$
- A `double` type stores real values using 64 bits: 1.0, 3.141592, -459.67, etc

Literals

- A value that is explicitly given is called a literal
- The literal 10 is literally the `int` value of 10
  - `int` is the default type of an integer
  - A `long` type can be created by appending an ‘L’ to the integer. For example 100L.
- The literal 3.14 is literally the `double` value of 3.14
  - `double` is the default type of a floating point literal
  - A `double` type can be created by appending an ‘D’ or ’d’ to the literal value. For example 100d.
  - A `float` type can be created by appending an ‘F’ or ’f’ to the literal value. For example 3.14f.
- `true` and `false` are boolean literals
- `"Computer Science"` is a `String` literal

Using variables

- To use a variable it must first be declared
- Here we declare the identifier `amount` to be a variable that will store integer values

```
int amount;
```

- Here we assign a value 10 to the variable `amount`

```
amount = 10;
```

- note we are not using the = to mean equivalence, rather it is a verb that means and should be read ’is assigned’
- We can combine declaration and assignment, which is good practice

```
int amount = 10;
```

Good use variables

- Give variables names that are meaningful and related to the data they store
- Avoid names already used, such as `mouseX`
- Use a comment in the declaration to help clarify what the variable stores

```
int amount; // The amount of items
```

- Variables store things, so they should be given a noun-phrase
- Start variables with a lower case letter and capitalize intermediate words, such as `numberOfStudents`
- Unless you are using simple constants such as 0 or 1, you should use a variable, especially if that value is used more than once!

Assignments

- An assignment statement has a left-hand side (LHS) and a right-hand side (RHS)
- The assignment operator = assigns the RHS to the LHS
- The assignment operator = should be read 'is assigned'

```
amount = 10;  // The location referred to
    by amount is assigned the value 10
```

- The left-hand side is ALWAYS an identifier associated with a memory location where a result can be stored
- The right-hand side is ALWAYS an expression that can be evaluated to a value

```
10 = value;  // WRONG WRONG WRONG
```

- The type of the RHS value in an assignment must match, or be compatible with, the type of the LHS
- What about the following?

```
value = value + 1;
```

Scope and lifetime of variables

- Where a variable can be used is called the scope of the variable
- The time period (statement t_1 to t_2) when a variable can be used is called its lifetime.
- A variable's scope is limited to point in the block in which it is created until the end of that block.
  - a local variable is one declared inside a particular block
  - a global variable is one declared outside any block
- A variable's lifetime ends after the final statement in the block in which it was declared has been executed.
  - A variable maintains its value when other functions are called, such as `line()`.

Processing Variables

- Several variables are global variables declared by the Processing language
- Several varaibles are always a part of every program, including
  - width — width of the window
  - height — height of the window
  - screen.width — width of the entire screen
  - screen.height — height of the entire screen
  - frameCount — the number of frames that have been displayed in the current run
  - frameRate — the number of frames to display every second
  - mouseX and mouseY — position of the mouse
  - key — A `char` that contains the value of the last key pressed on the keyboard
  - keyCode — A vaiable that contains the code value of the last key pressed on the keyboard
  - mousePressed — A boolean that is true if a mouse button is pressed
  - mouseButton — A varaible representing a code (LEFT, RIGHT, or CENTER) for the most recent ouse button pressed

Numerical system functions

The Math class defines many functions:

- arithmetic: `abs()`, `ceil()`, `floor()`, `round()`, `log()`, `pow()`, `sq()`, `sqrt()`, `max()`, `min()`
- conversion: `lerp()`, `map()`, `norm()`
- geometric: `dist()`, `mag()`
- trigonometric: `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`, `atan2()`, `degrees()`, `radians()`

Arithmetic Operators

- The operator + add two numbers:

```
int sum = 1 + 2 + 3 + 4 + 5; // 15
float x = 2.8 + 1.5; // 4.3
```

- The operator - subtracts two numbers:

```
int difference = 21 + 13; // 8
```

- These operators work from left to right
- parentheses can be used to group operations to change normal order of operations

Arithmetic Operators

- The operator * multiplies two numbers:

```
int factorial = 1*2*3*4*5; // 120
float x = 0.25*80.8;  // 20.2
```

- The operator / divides two numbers:

```
int factorial = 1/2; // 0
float x = 1.0/2.0;  // 0.5
```

- The operator % divides two numbers and returns the remainder:

```
int rem1 = 1%2; // 1
int rem2 = 73%7; // 3
int rem3 = 23.9%4; // 3.9
```

- These operators work from left to right
- The operators *, /, and % are evaluated before + and -
- parentheses can be used to group operations to change normal order of operations

Relational Operators

- Relational operators return a boolean result (`true` or `false`) based on the result of the comparison

```
int a = 10; int b = 11;
int c = 12; int d = 11;
boolean result1 = a < b; // true
boolean result1 = a <= b; // true
boolean result1 = a > b; // false
boolean result1 = a >= b; // false
boolean result1 = a == b; // false
boolean result1 = a != b; // true
boolean result1 = d <= b; // true
```

- Note when compare two primitive values, == tests for equivalence and is not assignment

Boolean Operators

- && — AND — true if both operands are true
- || — OR — false if both operands are false
- ! — NOT — inverts: true if operand is false, false if operand is true

X	Y	X && Y	X || Y	!X	!Y
false	false	false	false	true	true
false	true	false	true	true	false
true	false	false	true	false	true
true	true	true	true	false	false

- MEMORIZE the above table!!!!
- Be careful so that you do not use the operators | and &, which perform bitwise operations on integer operands

Compound assignment operators

- Processing has several operators that provide a combination of operator and assignment

operator	example	equivalent
+=	`x += 1`	`x = x + 1`
-=	`x -= 1`	`x = x - 1`
*=	`x *= 2`	`x = x * 2`
/=	`x /= 2`	`x = x / 2`
%=	`x %= 2`	`x = x % 2`
++	`x++`	`x = x + 1`
--	`x--`	`x = x - 1`

- Main advantages: less typing, enforces variable consistency, easier to change variable names

Functions

- Functions allow a form of structural abstraction in programming
- The goal is to improve our reading, writing, reuse, robustness, and revision of code
- We have used many predefined Processing functions: line, rect, size, sin, cos, atan2, . . .
- We have also written several functions: setup, draw, mousePressed, mouseReleased, keyPressed, . . .
- Functions are an important component in modular design
- Functions perform an action, and thus are usually named using a verb-phrase
- In Java, functions are called methods

Example: Drawing a face

- Code to draw a face

```
size(200,200);
background(255);
int x = 100;
int y = 100;
ellipseMode(CENTER);
stroke(0);
ellipse(x,y,40,40); // head
ellipse(x-7,y-7,5,5); // left eye
ellipse(x+7,y-7,5,5);  // right eye
arc(x,y,20,20,PI/6,5*PI/6);  // mouth
```

- What if we wanted to draw 50 faces at random locations?

Drawing faces using a function

```
void setup() {
  size(200, 200);
}
void draw() {
  background(128);
  face(random(width), random(height));
}
void face(float x, float y) {
  ellipseMode(CENTER);
  ellipse(x,y,40,40); // head
  ellipse(x-7,y-7,5,5); // left eye
  ellipse(x+7,y-7,5,5);  // right eye
  arc(x,y,20,20,PI/6,5*PI/6);  // mouth
}
```

Anatomy of a function

- The definition of a function needs the following:
  - return type
  - name
  - parenthesized parameter list
  - body in curly braces
  - body needs a return statement if return type is not `void`
  - Example

```
void face(float x, float y) {
  ellipseMode(CENTER);
  ellipse(x,y,40,40); // head
  ellipse(x-7,y-7,5,5); // left eye
  ellipse(x+7,y-7,5,5);  // right eye
  arc(x,y,20,20,PI/6,5*PI/6);  // mouth
}
```

- We call or invoke a function by using its name and a compatible list of parameters: `face(100, 100);`

Using functions

- Functions can call other functions
- Example

```
void face(float x, float y) {
  ellipseMode(CENTER);
  ellipse(x, y, 40, 40);  // head
  eye(x-7, y-7); // left eye
  eye(x+7, y-7);  // right eye
  arc(x,y,20,20,PI/6,5*PI/6);  // mouth
}

void eye(float x, float y) {
  ellipseMode(CENTER);
  ellipse(x, y, 8, 8);  // eye
  ellipse(x, y, 2, 2);  // iris/pupil
}
```

Function Parameters

- Copies of parameters are passed by value into a function based on the position of the parameter in the list
- Example

```
void draw() { face(10,20); }
void face(float x, float y) {
// x <- 10, y <- 20
. . .
  eye(x-7, y-7); // left eye (1st call)
  eye(x+7, y-7);  // right eye (2nd call)
. . .
}
void eye(float x, float y) {
// 1st call: x <- 3, y <- 13
// 2nd call: x <- 17, y <- 13
. . .
}
```

Function variables

- variables and parameters have the scope and lifetime of the function where they are defined; they are local variables of the function
- What is printed when `testFunction1` is invoked?

```
void testFunction1() {
  int x = 5;
  System.out.println(x);
  function1(x);
  System.out.println(x);
}
void function1(int x) {
  System.out.println(x);
  x = 10;
  System.out.println(x);
}
```

Returning a value from a function

- variables and parameters have the scope and lifetime of the function where they are defined; they are local variables of the function
- `void` indicated a function has no return value
- `return value;` sends `value` back to the calling function
- What is printed when `testFunction2` is invoked?

```
void testFunction2() {
  int x = 5;
  System.out.println(function2(x));
  System.out.println(function2(x+1));
}
int function2(int x) {
  return 2*x;
}
```

Test Driven Development

- Writing good code requires testing
- Process
  - Write a test
  - run test
  - write code
  - run all tests, fixing code if tests fail
  - iterate as needed
- Can be hard with graphical applications
- Code quality depends on quality of tests