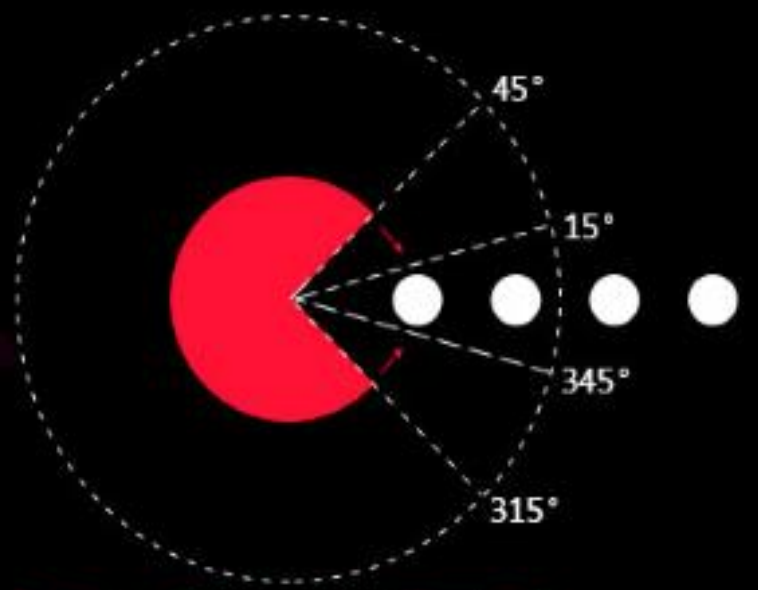
```
void setup() {
  size(500, 500);
  background(255);
}

void draw(){
  frameRate(10);
  background(0);
  fill(255, 18, 52 );
      switch(key){
        case 'd' :
          if(boolean(x%2)){
            arc(x,y,50,50,radians(45),radians(315),PIE);
          }
          else{
            arc(x,y,50,50,radians(15),radians(345),PIE);
          }
```



PROGRAMMING FOR ARTISTS 1

VARIABLES & IMAGES

VARIABLES
DECLARATION: SYNTAX

Before you use any variables, you must declare them!

int number = 3 ;

Data type (int, float, double, color, ...)
Name or identifier
Assignment sign
Value
Semicolon

The variable “number” is initialized with the value 3!

IDENTIFIER: NAMING CONVENTIONS

- Variable names always start with lowercase
- Variable name with multiple words: camelCasing
- ALLOWED: All letters or numbers, _ (underscore) and $ (dollar sign)
- NOT ALLOWED: start with number or a double declaration (same name and value)

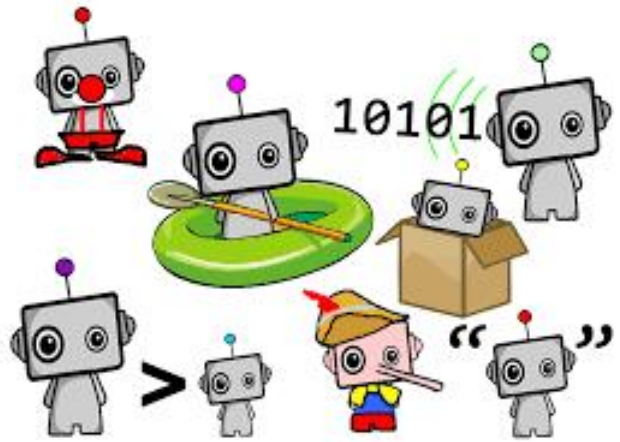
```
int number1 = 42;
int $myNumber = 15;
String myName = “Jack”;
```

```
int variable = 42;
int variable = 42;
int 4variable = 42;
```

DATA TYPES

Depends on which kind of information must be stored:

- int
- float
- double
- color
- boolean
- char
- String



DATA TYPES

- **int**

  used to store integers, without a decimal point. The value can be positive as negative. Default value is 0.

  int number1 = -256;

- **float**

  used to store floating-point numbers, e.g. numbers that have a decimal point (between 3.40282347E+38 and -3.40282347E+38). The value can be positive as negative.

  float number2 = 1.5387; → use a point, no comma!

- **double**

  for floating-point numbers larger than those that can be stored in a float. The value can be positive as negative.
  Processing supports the double datatype from Java as well. However, none of the Processing functions use double values, which use more memory and are typically overkill for most work created in Processing.

  double number3 = -2.98456268;

DATA TYPES

- **color**

  used to store color values, using the RGB values or hexadecimal notation.

```
color clr1 = color(100,200,100);     // Use the method color() to define a color ' clr1'
fill(clr1);     // Use color variable 'clr1' as fill color

color clr2 = color(255);    // Uses 255 for red, green and blue
color chosenColor = #FF1234;

red(chosenColor);  // Extracts the red value from the given color
green(chosenColor); // Extracts the green value from the given color
blue(chosenColor); // Extracts the blue value from the given color
```

- **boolean**

  used to store the values true or false. It is common to use boolean values with control statements to determine the flow of a program. Default value of a boolean variable is false.

```
boolean clicked = false;
boolean canBeDividedBy2 = true;
boolean gameStarted = true;
```

DATA TYPES

- **char**

  used to store a single character. A char stores letters and symbols in the Unicode format, a coding system developed to support a variety of world languages (see ASCII table)!

```
char letter = 'A';
println(letter);        //prints "A" to the console
int n  = letter;
println(n);             //prints "65" to the console
```

Always between single quotes!

- **String**

  used to store a sequence of characters.
  The class String includes methods for examining individual characters, comparing strings, searching strings, extracting parts of strings, and for converting an entire string uppercase and lowercase (see later).

```
String greeting = "Hello World";
int number = 42;
String sentence = "The number is " + number;
```

Always between double quotes!

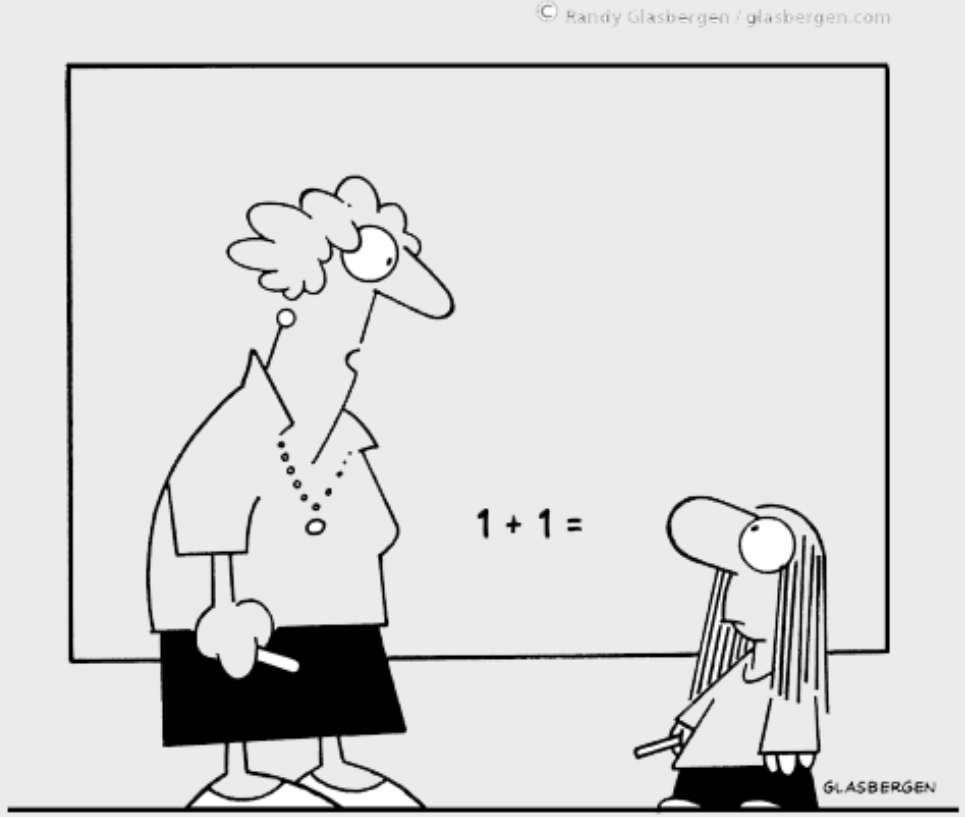
Use plus-sign to concatenate string variables with other variables

EXAMPLES

```
int numberOne = 1;
int numberTwo = -99;

float numberThree = 1.5;
float numberFour = -99.5;

char letter = 'k';
char myInitial = 'S';

String answer = "No";
String meeting = "Hi, we can meet at 5pm.";

boolean isChecked = true;
boolean isEmpty = false;
```

VARIABLES
ARITHMETICS

```
int number1 = 2;
int number2 = 4;

int addition = number1 + number2;
int subtraction  = number1 - number2;
int multiplication = number1 * number2;
int division = number1 / number2;
int remainder = number1 % number2;
```



"Yes, this will be useful to you later in life."

ORDER OF OPERATIONS:

Multiplicative	* / %
Additive	+ -
Assignment	=

Use round brackets to change to order!

INCREMENT/DECREMENT OPERATOR

Shortcut to add/subtract the value of a variable by 1

```
int x = 1;
x++;
println(x);          → x = 2
```

```
int y = 1;
y--;
println(y);          → y = 0
```

COMPOUND ASSIGNMENT OPERATOR

Provides a shorter syntax for assigning the result of an arithmetic or bitwise operator.

→ combining addition and assignment in 1 statement:

```
int x = 1;
x+=5;
println(x);          → x = 6
```

→ can also be used for subtraction, multiplication, modulo and division:

```
-=      x-=5;
*=      x*=5;
/=      x/=5;
%=      x%=5;
```

Always put the sign before the equal-sign!

Always put the sign before the equal-sign!

Always put the sign before the equal-sign!

Always put the sign before the equal-sign!

Always put the sign before the equal-sign!

Always put the sign before the equal-sign!

BUILT-IN METHODS SETUP AND DRAW

setup() and draw() are special functions, in the sense you will never call them. You create them and Processing will call them automatically. When you run a sketch, a number of steps are processed:

- the variables are initialized with the given values, or default values (0, null, false, etc.) if none are given.
- setup() is called, only once per sketch run. It can be used to define the base sketch setup, like its size, its mode (default, OpenGL, etc.), for loading resources, etc.
- draw() is called on each frame: a sketch is like a movie, it is made of successive frames, images, and their quick succession (by default 60 per second) makes the eye believe there is a continuous movement.

```
int xPos, yPos = 50, radius = 15;

void setup(){
  size(500,100);
  frameRate(10);
  background(0);
}

void draw(){
  fill(255);
  ellipse(xPos, yPos, radius*2,radius*2);
  xPos +=radius*2;
}
```

BUILT-IN METHODS SETUP AND DRAW

If you run the code, you will see that ellipses fill up the window, because the method draw() repeats itself each frame.
So every frame, the x-position is raised by 30 pixels. When it reaches the right border, the x-position goes on beyond the screen.
The background is not reset, therefor you can still see the ellipses!

If you want to animate the ellipse and show 1 ellipse that goes from left to right:
- reset the background in the draw
- reset the x-position, using the modulo

Check the difference!

```
int xPos, yPos = 50, radius = 15;

void setup(){
  size(500,100);
  frameRate(10);
  background(0);
}

void draw(){
  fill(255);
  ellipse(xPos, yPos, radius*2,radius*2);
  xPos +=radius*2;
}
```

```
int xPos, yPos = 50, radius = 15;

void setup(){
  size(500,100);
  frameRate(10);
  background(0);
}

void draw(){
  background(0);
  fill(255);
  ellipse(xPos, yPos, radius*2,radius*2);
  xPos +=radius*2;
  xPos%=width;
}
```

xPos =

0/500 = 0,.. →rest 0
1/500 = 0,.. →rest 1
2/500 = 0,.. →rest 2
250/500 =0,5 →rest 250

...
500/500= 1 →rest 0

IMPLICIT DATA CONVERSION

Data conversion is the conversion of computer data from one format to another. Implicit means that the computer assigns the value according to the specific datatype:

int a = 4 / 3;	OK : assigns 1 to a
int b = 3 / 4;	OK : assigns 0 to b
float c = 4.0 / 3.0;	OK : assigns 1.33334 to c
float d = 4.0 / 3;	OK : assigns 1.33334 to d (at least one number is float → returns a float)
float e = 4 / 3;	OK : assigns 1.0 to e (division of integers is integer, but displayed as float)
int f = 4.0 / 3;	ERROR: float can not be assigned to int without explicit data conversion!

EXPLICIT DATA CONVERSION (= TYPE CASTING)

Assigning a value of one type to a variable of another type is known as type casting.
It can only be used to convert numeric and char types to integer or float.
So it is fine if the variable is one of the following: char, int, float, double. It will not work with strings!

(float)int_*variable* → type cast int to float
(int)char_variable → type cast char to int
(int)float_*variable* → type cast float to int

You can also call a function which takes a value of a certain data type and attempts to convert it to another datatype:

int(variable) → converts any value of a primitive data type (boolean, char, String, color, float, int, double) to its integer representation

float(variable) → converts an integer or String to float

str(*variable*) → converts any value of a primitive data type (boolean, char, String, color, float, int, double) to its String representation

boolean(*variable*) → convert int or String to boolean *(0 or "false"→ false / 1 or "true"→ true)*

EXPLICIT DATA CONVERSION: EXAMPLE

```
float number1=65.9, floatVar1, floatVar2;
int number2=44;
int intVar1, intVar2, intVar3=12, intVar4;
int strVar1, strVar2, charVar;
String myStr="12", myText="text", strVar3;
char c='E';
boolean boolVar1, boolVar2, boolVar3;

void setup(){
  size(500,500);
  background(255);
}
```

```
void draw() {

  //TYPECASTING number1 from float to int
  intVar1 = (int)number1 / 3;
  println("intVar1=", intVar1);

  //CONVERTING DIVISION RESULT
  intVar2 = int(number1/3);
  println("intVar2=", intVar2);

  //TYPECASTING number2 from int to float
  floatVar1 = (float)number2 / 3;  //result 14.666667
  floatVar2 = number2 / 3;         //result 14.0
  println("floatVar1=", floatVar1);
  println("floatVar2=", floatVar2);

  //CONVERTING STRING TO INT
  strVar1 = int(myStr);
  println("strVar=", strVar1); //return 12, because string is a number!

  strVar2 = int(myText);
  println("strVar2=", strVar2); //returns 0!

  //CONVERTING CHAR TO INT
  charVar = int(c); //  TYPECASTING ALSO POSSIBLE HERE (int)c;
  println(c + " : " + charVar);  // Prints "E : 69"

  boolVar1 = boolean(intVar3); //returns true
  println("boolVar1=", boolVar1);

  /*
  intVar4 = 0.1;
   boolVar2 = boolean(intVar4); //ERROR: expects int
   println("boolVar2=", boolVar2);
   */

  strVar3 = "true";
  boolVar3 = boolean(strVar3); //returns true
  println("boolVar3=", boolVar3);
}
```

LOADING IMAGES

- Datatype: PImage
- Declaration: PImage myImage;
- Add image file using menu '**Sketch – Add File**'
  - **Folder** "***data***" is automatically created in the folder of the sketch
  - Processing currently can load GIF, JPEG, TGA and PNG images
- Load the image to the sketch: to load correctly, images must be located in the data directory of the current sketch!

myImage = loadImage("myExample.png", *extension*);
Loads an image into a variable of type Pimage. The extension parameter (optional) is used to determine the image type in cases where the image filename does not end with a proper extension.

myImage = requestImage("myExample.png");
This function loads images on a separate thread so that your sketch doesn't freeze while images load during setup().

```
void setup() {
  String url = "https://processing.org/img/processing-web.png";
  // Load image from a web server
  webImg = loadImage(url, "png");
}
```

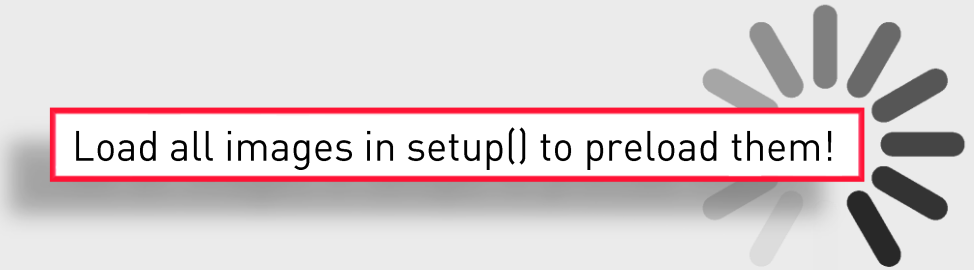


IMAGE METHODS

- The method image() draws an image to the display window:
  - image(myImage,x,y); → 'x' and 'y' are the x and y coordinates
  - image(myImage,x,y,w,h); → 'w' and 'h' are the width and height of the image

  The image is displayed at its original size unless the w and h parameters specify a different size.
  You can use '.width' and '.height' to get the size of the image:
  - int imgWidth = imgExample.width;
  - int imgHeight = imgExample.height;

- The imageMode() function can be used to change the way these parameters draw the image
  - CORNER (default): upper-left corner of the image
  - CORNERS: 2nd and 3rd parameter are one corner, the 4th and 5th the parameters of the opposite corner
  - CENTER: takes the center point of the image

- Examples:

```
image(myImage, 0, 0);
imageMode(CENTER);
image(myImage, 20, 50, imgWidth/2, imgHeight/2);
```

IMAGE METHODS

resize()

Resize the image to a new width and height. To make the image scale proportionally, **use 0** as the value for the **width or height parameter**. For instance, to make the width of an image 150 pixels, and change the height using the same proportion, use resize(150, 0).

```
PImage myImage;
myImage = loadImage("myExample.png");
image(myImage, 0, 0);
myImage.resize(70, 50);
image(myImage, 0, 0);
```

copy()

Copies a region of pixels from the display window to another area of the display window.

```
copy(sourceX, sourceY, sourceWidth, sourceHeight, destinationX, destinationY, destinationWidth, destinationHeight);
```

```
PImage img = loadImage("eames.jpg");
image(img, 0, 0, width, height);
img.copy(7, 22, 10, 10, 35, 25, 50, 50);
```

IMAGE METHODS

get()

Reads the color of any pixel or grabs a section of an image. If no parameters are specified, the entire image is returned

imgExample.get();
imgExample.get(x,y); → *value of 1 pixel at position x and y*
imgExample.get(x,y,w,h); → *specify the width and height to get a section. X and y are defined as the upper left corner*

This function can be used to animate the frames of an image:

```
PImage imgTheBoy, imgFrame;
int frameCounter = 0, numFrames = 5;
int frameWidth, frameHeight;

void setup(){
  size(200,541);
  frameRate(10);

  imgTheBoy = loadImage("theBoy.jpg");
  frameWidth = imgTheBoy.width / numFrames;
  frameHeight = imgTheBoy.height;
}

void draw(){
  background(255);

  //get(x,y,width,height)
  imgFrame = imgTheBoy.get(frameCounter * frameWidth, 0, frameWidth, frameHeight);
  image(imgFrame,0,0);

  frameCounter++;
  frameCounter%=numFrames;
}
```

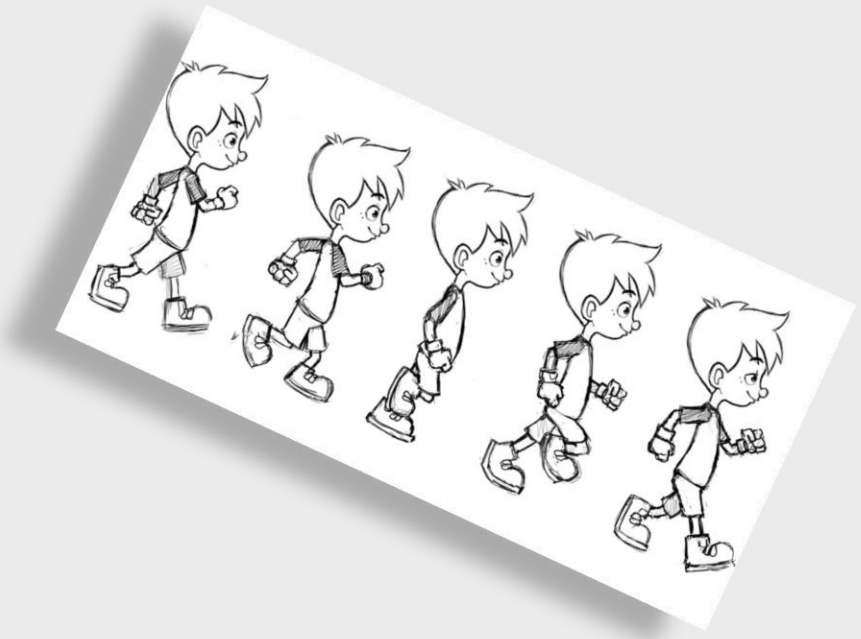


IMAGE METHODS

set()

Changes the color of any pixel, or writes an image directly to the display window.
The x and y parameters specify the pixel to change and the c parameter specifies the color value

```
set( 30, 85, color(0) );

PImage myImage = loadImage("apples.jpg");
set(0, 0, myImage);              // when setting an image, the x and y coordinates are defined as the upper left corner
```

mask()

Masks part of an image from displaying by loading another image and using it as an alpha channel.
This method is useful for creating dynamically generated alpha masks. Mask() can only be used with an image of the same size!

```
photo = loadImage("test.jpg");
maskImage = loadImage("mask.jpg");
photo.mask(maskImage);
```

filter()

Filters the display window using a preset filter or with a custom shader.
There are several preset options (see reference)

IMAGE METHODS

tint()

Sets the fill value for displaying images. Images can be tinted to specified colors or made transparent by including an alpha value.

```
PImage myImage;
myImage = loadImage("myExample.png");
image(myImage, 0, 0);
tint(0, 153, 204, 126);  // Tint blue and set transparency
image(myImage, 50, 0);
```

To apply transparency to an image without affecting its color, use white as the tint color and specify an alpha value.
For instance, tint(255, 128) will make an image 50% transparent.

```
tint(255, 126);  // Apply transparency without changing color
```

noTint()

Removes the current fill value for displaying images and reverts to displaying images with their original hues.

```
...
tint(0, 153, 204);  // Tint blue
image(img, 0, 0);
noTint();  // Disable tint
image(img, 50, 0);
```

IMAGE METHODS

blend()

Blends a region of pixels from one image into another (or in itself again) with full alpha channel support.
Different modes to blend (see reference).

```
imgAirport = loadImage("airport.jpg");
myImage.blend(imgAirport, 12, 12, 76, 76, 12, 12, 76, 76, ADD);
```

save()

Saves an image from the display window. Append a file extension to the name of the file, to indicate the file format to be used: either TIFF (.tif), TARGA (.tga), JPEG (.jpg), or PNG (.png). If no extension is included in the filename, the image will save in TIFF format and .tif will be added to the name.

```
PImage img;
img = loadImage(" myExample.png");
tint(0, 153, 204);  // Tint blue
image(img, 0, 0);
save("myImage.tif"); // Saves a TIFF file named "myImage.tif"
```

saveFrame()

Saves a numbered sequence of images, one image each time the function is run.