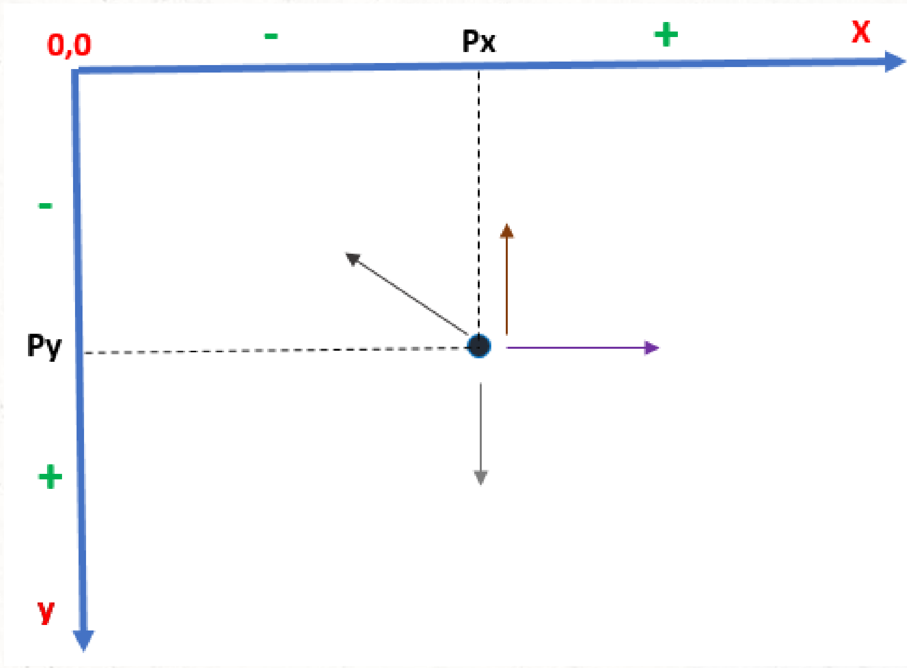
How to Move 2D primitive

Grosu Olga

The coordinate system in a computer window

- There are four possible steps to move.
- a step to the right can be simulated by incrementing x (x++);
- to the left by decrementing x (x--);
- forward by going down a pixel (y++);
- and backward by going up a pixel (y--)



random()

- Generates random numbers.
- Each time the random() function is called, it returns an unexpected value within the specified range.
- If only one parameter is passed to the function, it will return a float between zero and the value of the high parameter.
- ex1) starting at zero, and up to, but not including 8

Syntax:

```
random(high)
ex1) Random(8);
random(low, high)
ex2) random(-3, 7.6);
```

random()

- If two parameters are specified, the function will return a float with a value between the two values.
- ex1) returns values starting at -3 and up to (but not including) 7.6.
- To convert a floating-point random number to an integer, use the int() function.

Syntax:
random(high)
ex1) Random(8);
random(low, high)
ex2) random(-3, 7.6);

frameRate()

- Specifies the number of frames to be displayed every second.
- Ex.) frameRate(30)
- will attempt to refresh 30 times a second.
- If the processor is not fast enough to maintain the specified rate, the frame rate will not be achieved.
- Setting the frame rate within setup() is recommended. The default rate is 60 frames per second.

constrain()

- Constrains a value to not exceed a maximum and minimum value.
- amt - (float, int) - the value to constrain
- low - (float, int) -minimum limit
- high - (float, int) - maximum limit

Syntax:
constrain(amt, low, high)

ex:
```
void draw()
{
  background(204);
  float mx = constrain(mouseX, 30, 70);
  rect(mx-10, 40, 20, 20);
}
```

mouseX, mouseY

- The system variable **mouseX** always contains the current horizontal coordinate of the mouse.
- The system variable **mouseY** always contains the current vertical coordinate of the mouse.

Syntax:

```
constrain(amt, low, high)
```

ex:

```
void draw()
{
  background(204);
  float mx = constrain(mouseX, 30, 70);
  rect(mx-10, 40, 20, 20);
}
```

PmouseX

- The system variable pmouseX always contains the horizontal position of the mouse in the frame previous to the current frame.

ex:

```
void draw() {
  background(204);
  line(mouseX, 20, pmouseX, 80);
  println(mouseX + " : " + pmouseX);
}
```

PmouseY

- The system variable pmouseY always contains the vertical position of the mouse in the frame previous to the current frame.
- Inside draw(), pmouseX and pmouseY update only once per frame.

ex:
```
void draw() {
  background(204);
  line(20, mouseY, 80, pmouseY);
  println(mouseY + " : " + pmouseY);
}
```

Translate()

translate(x, y)

- Specifies an amount to displace objects within the display window.
- The x parameter specifies left/right translation,
- the y parameter specifies up/down translation

ex:
```
size(400, 400);
rect(0, 0, 220, 220);  // Draw rect at original 0,0
translate(120, 80);
rect(0, 0, 220, 220);  // Draw rect at new 0,0
translate(56, 56);
rect(0, 0, 220, 220);  // Draw rect at new 0,0
```

•statements
•}

Conditonals -IF

- Allows the program to make a decision about which code to execute.
- If the test evaluates to true, the statements enclosed within the block are executed
- and if the test evaluates to false the statements are not executed.

```
if (test) {
statements
}
```

ex:
```
if(mouseX > width/2)
{
    r = r + 1;
  }
```

IF

```
if (expression) {
statements
} else {statements}
if (expression) {statements}
else if (expression)
{statements}
 else { statements}
```

ex:

```
if(mouseX > width/2)
{if (mouseX < 240 && mouseY < 135) {
    rect(0, 0, 240, 135);
  } else if (mouseX > 240 && mouseY <
135) {
    rect(240, 0, 240, 135);
  } else if (mouseX < 240 && mouseY >
135) {
    rect(0, 135, 240, 135);
  } else if
(mouseX > 240 && mouseY > 135) {
    rect(240, 135, 240, 135);} }
```

?: (conditional)

A shortcut for writing an if and else structure.

```
result = test ? expression1 : expression2
```

is equivalent to this structure:

```
if (test) {
result = expression1
} else {
  result = expression2
 }
```

ex:

```
s = (i < 50) ? 0 : 255;
```

Loop function -For

- Controls a sequence of repetitions.

for (init; test; update) {statements}

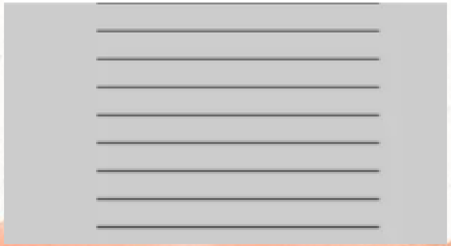
init - statement executed once when beginning loop

test - if the test evaluates to true, the statements execute

update - executes at the end of each iteration

ex:

```
size(400, 400);
for (int i = 0; i < 320; i = i+20)
{
  line(120, i, 320, i);
}
```

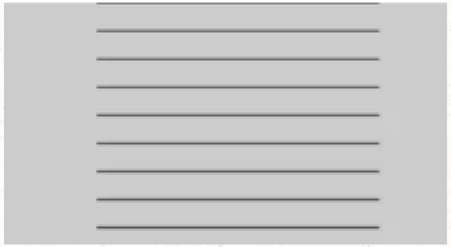


While

- Controls a sequence of repetitions.
- The while structure executes a series of statements continuously while the expression is true.
- The expression must be updated during the repetitions or the program will never "break out" of while.

```
while (expression) {
statements
}
```

ex:
```
size(400, 400);
int i = 0;
while (i < 320) {
  line(120, i, 320, i);
  i = i + 20;
}
```



DOCUMENTATION

- https://processing.org/reference
- DANIEL SHIFFMAN “Learning Processing” http://learningprocessing.com/
- DANIEL SHIFFMAN “The Nature of Code”

https://natureofcode.com/book/introduction/

Also:

https://processing.org/books/