PCS 2190 - TEXNUM: Texto e Números em Ponto Flutuante

Ricardo Nakamura e Romero Tori

2015

1 Introdução

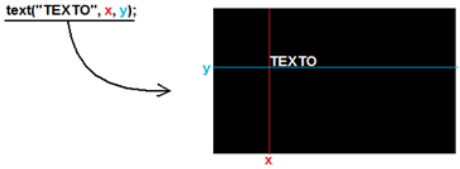
Nesta aula vamos ver como utilizar textos e números reais em um programa e como desenhar textos no Processing.

2 Escrevendo textos no Processing

Para escrever um texto na janela gráfica, podemos utilizar o comando **text**. O texto deve ser colocado entre aspas, como mostrado no exemplo a seguir:

```
size(400, 200);
background(0);
text("The quick brown fox jumped over...", 20, 20);
```

Os dois números escritos depois do texto são a posição (horizontal, vertical) em que o texto deve ser escrito na janela gráfica. Uma observação importante: o padrão do Processing é utilizar a posição indicada como linha de base, como ilustrado a seguir.

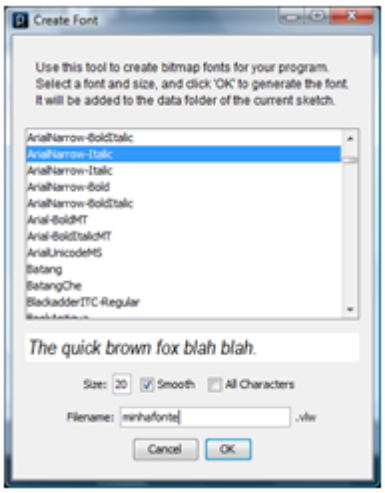


Atividade 1

Você pode alterar o tamanho da fonte usando o comando **textSize**, a cor através do comando **fill** e o alinhamento do texto através do comando **textAlign**. Modifique o programa anterior para escrever três linhas de texto, usando os comandos citados.

3 Configurando uma fonte específica

Normalmente, o Processing escolhe uma fonte padrão disponível no computador para escrever os textos. No entanto, você também pode selecionar uma fonte específica. Para isso, o primeiro passo consiste em criar um **arquivo de fonte**, o que é feito através da opção de menu Tools → Create Font. Será aberta uma janela semelhante à da figura a seguir.



Nesta janela, podemos escolher a fonte a ser utilizada, bem como o seu tamanho – em pixels e não em pontos (de tipografia). Na caixa de texto na parte de baixo da janela, devemos digitar o nome do arquivo de fonte a ser criado. Na figura anterior, o nome do arquivo é "minha-fonte.vlw".

O Processing utiliza fontes criadas através desta ferramenta para permitir que os programas sejam distribuídos de forma independente. Se ele carregasse diretamente uma fonte do sistema, seria preciso distribuir esta fonte juntamente com o programa e pedir para que cada usuário a instalasse.

Por outro lado, este sistema também tem suas limitações: como as fontes são exportadas em formato matricial, elas só têm uma aparência boa na tela se forem usadas do tamanho que foram exportadas. Tamanhos maiores ou menores farão com que elas tenham distorções. A listagem a seguir mostra como usar um arquivo de fonte no Processing para escrever texto.

```
size(500, 200);
background(0);
PFont f = loadFont("minhafonte.vlw");
textFont(f);
text("Teste␣1␣2␣3", 0, 20);
textSize(40);
text("Teste␣1␣2␣3", 0, 60);
textSize(80);
text("Teste␣1␣2␣3", 0, 140);
```

Para carregar o arquivo de fonte criado anteriormente, usamos o comando **loadFont**. Ao testar o programa, lembre-se de mudar o nome para o arquivo que você criou. Da mesma forma que imagens são armazenadas em variáveis do tipo **PImage**, fontes são armazenadas em variáveis do tipo **PFont**. Assim, três coisas ocorrem na terceira linha do programa: definimos uma variável chamada “f” do tipo **PFont**, carregamos nossa fonte do arquivo “minhafonte.vlw” e armazenamos a fonte carregada na variável “f”.

O comando **textFont**, na quarta linha do programa é utilizado para definir qual a fonte a ser utilizada para desenhar texto na janela. O comando text, na quinta linha do programa escreve o texto “teste 1 2 3” na janela. A sexta linha do programa utiliza o comando **textSize** para indicar a altura (em pixels) do texto a ser escrito. Note que, neste exemplo, a fonte “minhafonte.vlw” foi exportada com 20 pixels de altura, então nesta linha fazemos com que ela seja desenhada com o dobro do tamanho.

Atividade 2

Crie um programa para desenhar uma mesma linha de texto cinco vezes, com tamanhos cada vez maiores. Utilize **while** ou **for** no seu programa.

Atividade 3

Para mudar a cor com que o texto será escrito, usamos o comando **fill** (o mesmo comando usado para escolher a cor de preenchimento de formas geométricas). Faça um programa que desenha um texto na tela cinco vezes, com cores diferentes.

4 Manipulando Texto

Na seção anterior, escrevemos o texto a ser desenhado na tela diretamente no comando **text**. No entanto, também é possível armazenar texto em variáveis do tipo **String**. O programa a seguir mostra o uso de uma variável deste tipo. Na quinta linha do programa, definimos a variável t, do tipo **String**. Na sexta linha, atribuímos o texto "teste 1 2 3" a esta variável e, na última linha, usamos a variável para desenhar o texto na tela.

```
size(500, 200);
background(0);
PFont f = loadFont("minhafonte.vlw");
textFont(f);
String t;
t = "teste␣1␣2␣3";
text(t, 50, 50);
```

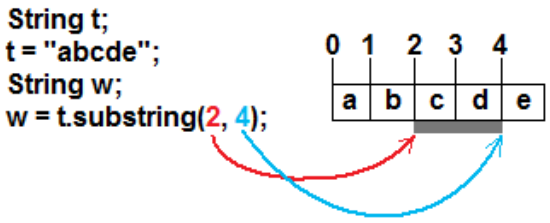
Observe que o conteúdo a ser atribuído para a variável t é indicado entre aspas. Isso é necessário para identificar os limites do texto, caso contrário não haveria como determinar o que é texto e o que é programa. Importante: ao copiar texto da apostila para o Processing, as

aspas podem ser convertidas para caracteres especiais “”. Neste caso, será preciso digitar novamente as aspas no editor do Processing para que elas sejam reconhecidas corretamente!!

O tipo **String** é um tipo composto e, assim, possui alguns recursos adicionais em relação a outros tipos de variáveis. Por exemplo, se a variável t é do tipo String, podemos escrever t.length() para obter o comprimento do texto (em número de caracteres). Também podemos utilizar o comando t.substring para obter uma parte do texto original, como ilustrado nos exemplos abaixo:

```
String t;
size(300, 200);
background(0);
t = "teste 1 2 3";
int c;
c = t.length(); // c irá receber o valor 11
String w;
w = t.substring(0, 5); // w irá receber o valor "teste"
String k;
k = t.substring(6, 11); // k irá receber o valor "1 2 3"
text(t, 20, 20);
text(w, 20, 40);
text(k, 20, 60);
```

A figura abaixo ajuda a entender os parâmetros do comando substring: “cortamos uma fatia” do texto original entre as duas posições passadas para o comando. É importante observar que a contagem começa do zero, que corresponde à posição do primeiro caractere do texto. Da mesma forma, a “fatia” não inclui o caractere indicado na posição final.



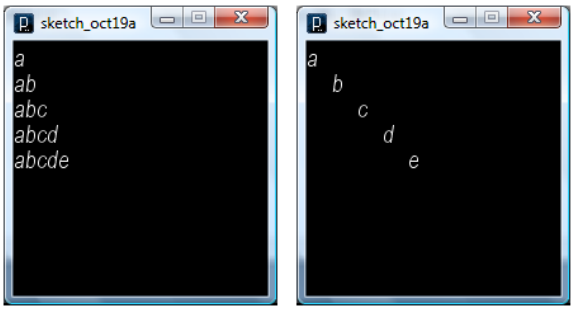
Existem ainda outros comandos que podem ser utilizados com variáveis do tipo String. Consulte a referência do Processing para conhecê-los. Curiosamente, é possível utilizar a operação de adição para agrupar duas ou mais variáveis do tipo String. Veja o exemplo abaixo:

```
size(200, 200);
background(0);
String a;
a = "teste";
String b;
b = "␣1␣2␣3";
String c;
c = a + b;
text(c, 30, 30);
```

Atividade 4

O programa abaixo desenha uma sequência de textos (mostrada na figura da esquerda), usando os comandos length e substring. Modifique o programa para desenhar somente uma linha diagonal de letras, como na figura da direita.

```
size(200, 200);
background(0);
String t;
t = "abcde";
int i;
for (i = 1; i <= t.length(); i = i + 1) {
  String k = t.substring(0, i);
  text(k, 0, 20*i);
}
```



5 Utilizando números reais

Até o momento, todos os números que manipulamos foram números inteiros. Um outro tipo de variável, chamado **float**, é utilizado para representar números reais. O nome deste tipo de variável se refere à forma de representação utilizada internamente pelo computador, denominada "ponto flutuante".

Na maior parte do tempo, podemos usar variáveis float de forma muito parecida com variáveis int, com a diferença que podemos atribuir números que não são inteiros. Para isso, utilizamos o ponto (e não vírgula) para separar a porção decimal:

```
float x, y;
x = 4.5;
y = 3.0/5.0;
```

Quando escrevemos um cálculo diretamente com números, o Processing sempre assume que os números são inteiros, a menos que um ponto seja utilizado. O efeito disso é visível no caso da divisão:

```
float x, y;
x = 4 / 7; // divisão inteira: 4/7 = 0
y = 4.0 / 7.0; // divisão entre números reais: 4/7 = 0.5714...
// vamos conferir:
println(x);
println(y);
```

Note que, devido à diferença de precisão, não é possível atribuir diretamente o valor de uma variável float para uma variável int. Para isso, precisamos fazer a conversão usando o comando int:

```
float a;
int b;
a = 5.7;
// se eu escrever b = a, causarei um erro do Processing
b = int(a);
println(a);
println(b);
```

Quando a conversão de **float** para **int** é feita, toda a parte decimal do número é eliminada. Note que isso é diferente de "arredondar" o número. Por exemplo, int(3.02) = 3 e int(3.9999) = 3 também.

Um ponto importante a se lembrar quando se utiliza variáveis do tipo **float** é que sua precisão é limitada. Por exemplo, matematica-

mente a fração 1/3 corresponde ao número 0,3333333333333... No entanto, como a memória do computador não é infinita, não há como representar os infinitos "3" depois da vírgula. Portanto, o número será arredondado para algo como 0,33333333333. Os efeitos desta precisão limitada são especialmente importantes em programas que fazem simulações de física e outros cálculos matemáticos complexos. O programa a seguir mostra um exemplo dessa falta de precisão.

```
float p = 1/99.0;
float q = 0;
// soma 99 vezes p. Deveria ser o
// mesmo que 99*p
for (int i = 0; i < 99; i = i + 1) {
  q = q + p;
}
// vamos conferir...
println(99*p);
println(q);
```

Normalmente, somar um número 99 vezes ou multiplicar esse número por 99 deveria ser exatamente a mesma coisa, mas o programa anterior mostra que os resultados das duas operações são diferentes, porque o valor da variável **p** não consegue representar exatamente a fração 1/99.

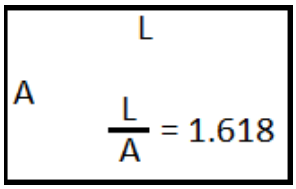
Também podemos escrever números na janela usando o comando **text**. Se usarmos um valor float, o número será escrito com casas decimais. O programa abaixo mostra a diferença.

```
size(200, 200);
background(0);
PFont f = loadFont("minhafonte.vlw");
textFont(f);
text(12, 0, 20);
text(12.0, 0, 40);
```

Atividade 5

A razão áurea (ou proporção áurea) representa uma proporção encontrada frequentemente na natureza e corresponde, aproximadamente, ao número 1.61803. Um retângulo apresenta a proporção

áurea quando a divisão entre seu comprimento e altura correspondem a esse número. Faça um programa que desenha vários retângulos de tamanhos diferentes mas todos com a proporção áurea. Utilize variáveis do tipo **float** em seu programa.



Note que, utilizando números reais, podemos fazer animações mais precisas, com movimentos de frações de pixels, como ilustrado no próximo exemplo, em que um dos retângulos se move a uma velocidade de 1/2 pixel por repetição:

```
float x1, x2;

void setup() {
  size(400, 300);
  x1 = 0;
  x2 = 0;
  background(0);
}
void draw() {
  rect(x1, 0, 20, 150);
  rect(x2, 150, 20, 150);
  x1 = x1 + 1;
  // fazemos o segundo retângulo se mover
  // a apenas 1/2 pixel por repetição!
  x2 = x2 + 0.5;
}
```

Atividade 6

Modifique o programa acima, acrescentando um terceiro retângulo que se move à velocidade de 1/3 de pixel por repetição.

6 Números Aleatórios

Agora que as variáveis de ponto flutuante foram apresentadas, podemos utilizar um recurso muito interessante do Processing: o gerador de números aleatórios. Na verdade, um computador não é capaz de produzir números verdadeiramente aleatórios, mas existem formas de se gerar sequências "imprevisíveis" o bastante para que não seja possível notar este fato.

Para sortear um número aleatório, utilizamos o comando **random**. Este comando pode receber um ou dois valores. No caso de um valor, será sorteado um número real maior ou igual a zero e menor do que o valor recebido. No caso de dois valores, será sorteado um número real maior ou igual ao primeiro valor e menor do que o segundo. Por exemplo, random(3) sorteia um número entre 0 e 2.999999999999999... e random(12, 17) sorteia um número entre 12 e 16.9999999999999...

Note que o valor sorteado pelo comando **random** é um **float**. Assim, se quisermos somente valores inteiros, temos que usar o comando **int**. Por exemplo, o seguinte programa simula o lançamento de um dado de seis faces:

```
float r;
int dado;
// sorteia um numero entre 1 e 6.9999999999...
r = random(1, 7);
// despreza a parte decimal, resultando em um número entre 1 e 6
dado = int(r);
```

Combinando-se o comando **random** com outros comandos do Processing, podemos obter desenhos e animações "imprevisíveis". O programa abaixo desenha dez pequenos círculos na tela em posições aleatórias.

```
void setup() {
  size(400, 400);
  // atualiza a tela 1 vez por segundo
  // para podermos ver os círculos mudando
  frameRate(1);
}
void draw() {
  background(0);
  float x
  float y;
  for (int i = 0; i < 10; i = i + 1) {
    // sorteia posições dentro da janela
```

```
    x = random(width);
    y = random(height);
    ellipse(x, y, 10, 10);
  }
}
```

O comando **frameRate(1)** utilizado no bloco **setup** indica que o bloco **draw** deve ser executado somente uma vez por segundo. O padrão do Processing é executar a 60 quadros por segundo (correspondente a frameRate(60)). Desta forma, é possível ver cada conjunto de círculos antes dele ser substituído.

Atividade 7

Modifique o programa anterior para que cada círculo seja preenchido com uma cor aleatória. Dica: utilize o comando **color** (veja a referência do Processing).

Atividade 8

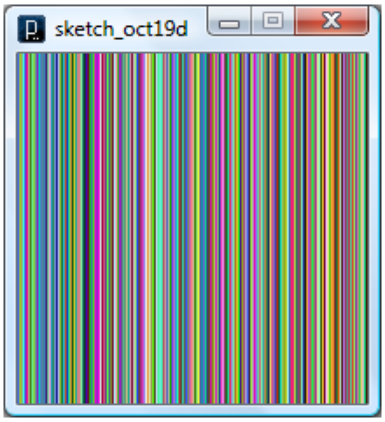
Modifique o programa anterior para desenhar dez segmentos de reta entre pontos aleatórios. Para um desafio maior, tente fazer com que as linhas sejam desenhadas de forma conectada (isto é, o ponto final de uma reta é o começo da próxima).

7 Resumo

Nesta aula tivemos contato com variáveis para representar texto e números reais e vimos algumas formas de utilizar estas variáveis no Processing.

Atividade 9

Crie um programa que preenche uma janela com uma sequência de linhas verticais com cores aleatórias, como mostrado na figura abaixo:



Atividade 10

Crie um programa que mostra uma contagem regressiva que começa em 10 e vai diminuindo em 1 a cada segundo. O contador deve ser desenhado na janela gráfica, com uma fonte de sua preferência. Quando a contagem atinge zero, a janela toda é pintada de vermelho.

Atividade 11

Desafio: Crie um programa que desenha polígonos regulares com um número aleatório de lados entre 3 e 20. A cada segundo, um

novo polígono é desenhado na janela, substituindo o anterior. A cor de desenho do polígono também deve ser aleatória. Opcionalmente, além do contorno do polígono, desenhe também linhas que conectam os vértices até o centro. As figuras abaixo mostram alguns exemplos.