referência rápida Python + py5

referência completa do py5 em:
https://py5.ixora.io/reference

estrutura do programa

Estruturando um sketch estático:

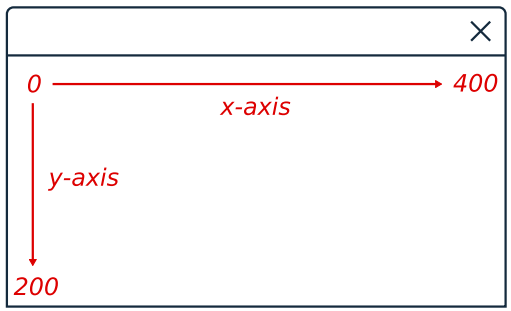
```
size(400, 200) # tamanho do desenho
código aqui é executado uma vez
```

A estrutura de um sketch com movimento:

```
def setup():
    size(400, 200)
    código excutado uma vez no início
def draw():
    código repetindo a cada quadro
```

Anatomia de uma chamada de função:

Sistema de coordenadas padrão:



comentários

```
# para anotações no meio do código
# serve pra desligar código também!
# no_fill()  <- não vai executar

"""
para comentários de várias linhas
usamos aspas triplas (doctstrings)
"""
```

preenchimento & traço

```
background(color) # cor de fundo
fill(color) # cor de preenchimento
no_fill() # sem preenchimento
stroke(color) # cor de traço
stroke_weight() # espessura do traço
no_stroke() # sem traço de contorno
```

Exemplo de preenchimento vermelho:

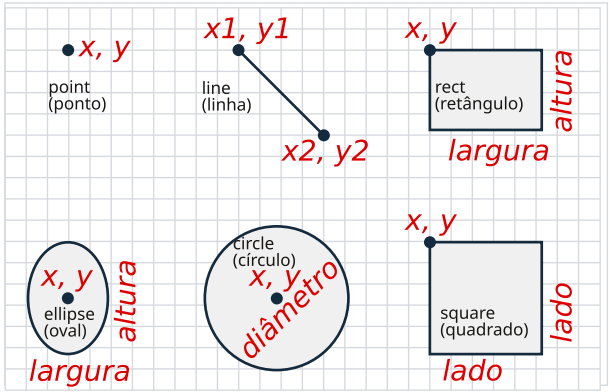
```
fill('#FF0000')  # hexadecimal
fill(255, 0, 0)  # R, G, B
# HSB (Matiz, Saturação, Brilho)
color_mode(HSB, 360, 100, 100)
fill(0, 100, 100)
```

exibir um valor

```
print(value)  # exibe to console
```

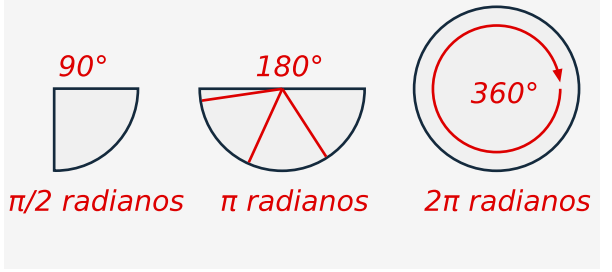
elementos 2D

```
point(x, y)
line(x1, y1, x2, y2)
rect(x, y, largura, altura)
ellipse(x, y, largura, altura)
circle(x, y, diâmetro)
square(x, y, lado)
```




```
arc(x, y, width, height, start, end)
```

Para indicar ângulos usamos sempre radianos:



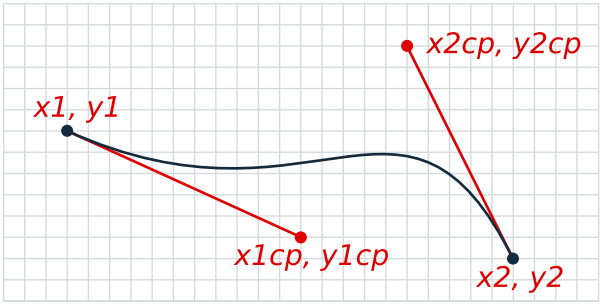
formas com vértices

Desenhe formas usando a função vertex() entre begin_shape() e end_shape() :

```
begin_shape()
vertex(x1, y1)
vertex(x2, y2)
# mais vertices aqui
end_shape(CLOSE)
```

Para formas curvas:

```
begin_shape()
vertex(x1, y1)  # vértice 1
bezier_vertex(
  x1cp, y1cp,   # ponto de controle 1
  x2cp, y2cp,   # ponto de controle 2
  x2, y2)       # vértice 2
# mais vértices bezier aqui
end_shape()
```



tipografia

```
text_font(create_font(fonte, corpo_base))
text_size(corpo_aplicado)
text('texto', x, y)
```

matemática

Operadores aritiméticos, soma, subtração, multiplicação, divisão e resto da divisão:

```
+  -  *  /  %
```

```
resto = 7 % 2   # resulta 1
```

Use parenteses para sobrepujar a precedencia normal dos operadores:

```
conta = (1 + 2) * 3 # resulta 9, não 7
```

pseudo-aleatoriedade (random)

Para obter "valores inesperados" podemos usar a função random:

```
f = random(10)  # 0.0 até 10, mas não inclui 10
r = random(5, 10)# 5.0 até 10, mas não inclui 10
i = random_int(5, 10) # inteiro, e inclui 10!
c = random_choice(['a', 'b']) # sorteia um!
```

É possível fixar o início do gerador de números pseudo-aleatórios usando:

```
random_seed(inteiro)
```

constantes / variáveis do sistema

O número pi, e algumas frações e múltiplos:

```
PI,  HALF_PI, QUARTER_PI, TAU
```

Algumas variáveis que o Processing provê:

```
width, height # largura e altura do desenho
mouse_x, mouse_y # posição do mouse
frame_count # número do quadro atual
```

controle do fluxo de execução

Operadores relacionais (de comparação):

```
a == b   # se a é igual a b
a != b   # se a é diferente de b
a > b    # se a é maior que b
a < b    # se a é menor que b
a >= b   # se a é maior ou igual a b
a <= b   # se a é menor ou igual a b
```

Operadores lógicos:

and (e) **or** (ou) **not** (não)

Execução condicional (se então):

```
if  cond_a:
    se condição a for verdadeira/True, faça isto
elif cond_b:
    se condição b for verdadeira/True, faça isto
else:
    se ambas forem falsas/False, faça isto outro
```

Iteração com o laço for:

```
for i in range(3):
    print(i) # exibe 0, 1, 2 no console
```

Usando um laço para percorrer uma lista:

```
numeros = [3, 4, 5]
for i in numeros:
    print(i) # exibe 3, 4, 5 no console
```

adaptado de:
github.com/tabreturn/processing.py-cheat-sheet