referência rápida Python + py5

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

estrutura do programa

Estruturando um sketch estático:

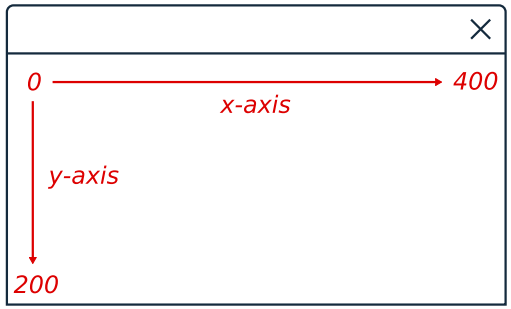
```
size(400, 200) # tamanho do desenho
código aqui é executado uma vez
```

Structuring an animated sketch:

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

Anatomia de uma chamada de função:

Sistema de coordenadas padrão:



comentários

```
# uma linha de comentário
'''
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)

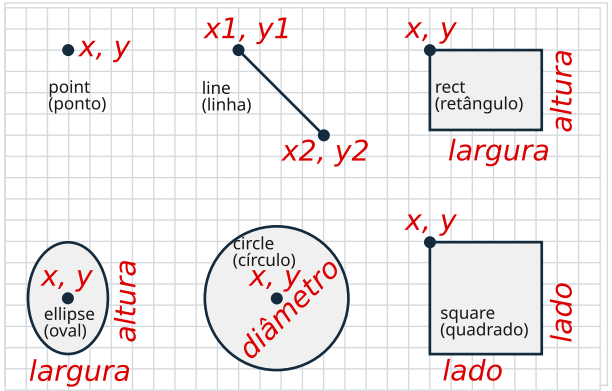
line(x1, y1, x2, y2)

rect(x, y, largura, altura)

ellipse(x, y, largura, altura)

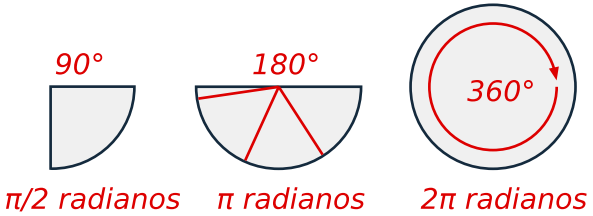
circle(x, y, diâmetro)

square(x, y, lado)



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

Para indicar ângulos usamos sempre radianos:



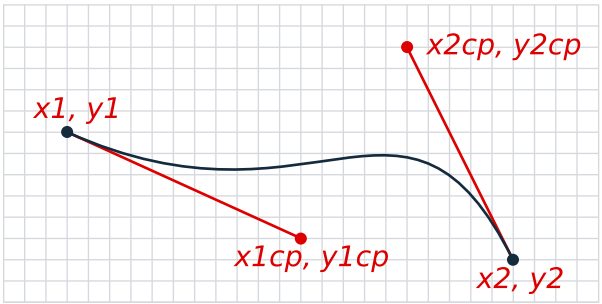
formas com vértices

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:

```
+  -  *  /  %
```

Use parenteses para sobrepujar a precedencia normal dos operadores:

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

pseudo-aleatoriedade (random)

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

```
random(10)  # 0.0 até 10, não inclui 10
random(5, 10)  # 5.0 até 10, não inclui 10
random_int(5, 10)  # valor inteiro, inclui 10
random_choice([5, 10]) # escolhe 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
```

Some useful Processing system variables:

```
width, height  # largura e altura do des.
mouse_x, mouse_y  # coordenadas
frame_count  # número do quadro atual
```

controle do fluxo de execução

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
```

Operadores relacionais (de comparação):

```
==   !=   >   <   >=   <=
```

Operadores lógicos:

```
and (e) or (ou) not (não)
```

Iteração com o laço for:

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

Usando um laço para percorrer uma lista:

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

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