Aula 15 – O módulo turtle – exemplo de módulo gráfico

O módulo turtle (tartaruga) é um módulo de demonstração de processamento gráfico muito interessante. E na verdade um módulo educacional construído originalmente para mostrar possibilidades gráficas da computação. Apenas como motivação, designamos o objeto que traça o desenho como uma tartaruga, mas poderia ser simplesmente uma caneta.

O exemplo abaixo constrói um retângulo 162x100. A unidade são pixels da tela

```
import turtle             # vamos usar as funções/objetos do módulo turtle

alex = turtle.Turtle()    # cria um turtle chamado alex
alex.forward(162)         # manda o alex se mover 162 unidades para frente
alex.left(90)             # roda de 90 graus para a esquerda
alex.forward(100)         # desenha o segundo lado do retângulo
alex.left(90)             # roda de 90 graus para a esquerda
alex.forward(162)         # desenha o terceiro lado do retângulo
alex.left(90)             # roda de 90 graus para a esquerda
alex.forward(100)         # desenha o quarto lado do retângulo.
```

Teste esse programa no seu computador e veja o resultado.

Funções e objetos utilizados:

turtle.Turtle():
Cria um objeto turtle

forward(d) ou fd(d):
Movimenta o objeto turtle na direção apontada por uma distância d.

left(g) ou lt(g):
Gira à esquerda o objeto turtle de g graus.

Exercícios:

P15.1) Dados a e b inteiros menores ou iguais a 100, desenhar um retângulo de lados a e b. Estamos limitando o tamanho apenas para o objeto caber na tela padrão.

P15.2) Dado a inteiro menor ou igual a 100, desenhar um triângulo equilátero (dica - o ângulo interno é 60. – qual o ângulo de desvio?)

P15.3) Idem para um pentágono.

P15.4) Idem para um polígono regular qualquer, ou seja, dado um comprimento de lado L, um número de lados N, construir um polígono regular com N lados. Lembre-se que o ângulo interno de um polígono regular é dado por 180*(N-1)/N.

Uma corrida de tartarugas

Dadas três tartarugas, vamos agora promover uma disputa entre elas. Uma corrida linear de uma dada distância. Para diferenciar a velocidade das tartarugas vamos a cada ciclo fazer com que deem 0, 1 ou 2 passos aleatoriamente. Há também um juiz que traça a linha de chegada e se posiciona para verificar quem chegou primeiro.

Teste o programa abaixo e verifique o que ocorre:

```
import turtle  # permite usar as funções e objetos do módulo turtle
import random  # para determinar os passos em cada ciclo

# criar os competidores
alex = turtle.Turtle()
blex = turtle.Turtle()
clex = turtle.Turtle()

# define a quantidade de passos da corrida
passos = 200    # corrida de 200 passos

# criar o juiz, traçar a linha de chegada e se posicionar
juiz = turtle.Turtle()
juiz.penup()
juiz.setpos(200, 100)
juiz.right(90)
juiz.pendown()
juiz.forward(200)
juiz.right(180)

# contadores
n_venc = 0          # número de vencedores da corrida
alex_passos = 0     # número de passos de alex
blex_passos = 0     # número de passos de blex
clex_passos = 0     # número de passos de clex

# posiciona alex e clex. blex já está posicionado

alex.penup()
alex.setpos(0, 50)
alex.pendown()

clex.penup()
clex.setpos(0, -50)
clex.pendown()

# cores
alex.color("blue")
blex.color("green")
clex.color("red")

# aparência
alex.shape("turtle")
blex.shape("turtle")
clex.shape("turtle")

# largura da linha
alex.pensize(5)
blex.pensize(5)
clex.pensize(5)

# inicia a corrida
# vamos usar 1000 ciclos - suficiente para terminar uma corrida
# random.seed()

for i in range(1000):
```

```
    # movimenta alex
    k = random.randrange(5)
    alex.forward(k)
    alex_passos += k
    if alex_passos >= passos:
        print("alex venceu a corrida")
        n_venc += 1

    # movimenta blex
    k = random.randrange(5)
    blex.forward(k)
    blex_passos += k
    if blex_passos >= passos:
        print("blex venceu a corrida")
        n_venc += 1

    # movimenta clex
    k = random.randrange(5)
    clex.forward(k)
    clex_passos += k
    if clex_passos >= passos:
        print("clex venceu a corrida")
        n_venc += 1

    if (n_venc > 0): break

#temos um ou mais vencedores
if (n_venc == 1): print("temos um vencedor")
else: print("houve empate")
```

Funções e objetos utilizados:

penup() ou pu() ou up():
Levanta a pena de desenho. Movimenta o objeto sem desenhar.

pendown() ou pd() ou down()
Abaixa a pena de desenho. O movimento do objeto vai provocar o rastro (desenho).

setpos(x, y = None) ou goto(x, y = None) ou setposition(x, y = None)
Movimenta o objeto para a posição x, y. Se não tem o parâmetro y, x deve ser um par de coordenadas na forma pos(a, b).

color(“nome da cor”)
Cor da linha a ser desenhada. Existem centenas de cores. Veja mais exemplos abaixo.

shape(“forma do objeto”)
Forma do objeto que simula a caneta. Pode ser: “arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”. Mais exemplos abaixo.

pensize(“largura da linha”)
Largura da linha traçada. Mais exemplos abaixo.

O posicionamento padrão e dimensões da tela

A dimensão padrão da tela é de 400 x 300 pixels. As coordenadas padrões são -200 a +200 no eixo horizontal e -150 a +150 no eixo vertical. Pode ser alterada pela função screensize:

turtle.screensize(largura, altura, cor-de-fundo)

Exemplos:

turtle.screensize(1000, 800, “orange”) – redefine a tela com largura 1000, altura 800 e cor de fundo laranja.

turtle.screensize() – devolve a largura e altura da tela atual

Todo novo objeto turtle quando criado (turtle.Turtle()) estará na posição de coordenadas (0, 0).

Assim, podemos usar toda a tela para posicionar nossa caneta ou tartaruga. Exemplos:

t1 = turtle.Turtle
t1.goto(-100, 200)
t1.goto(50, -100)
t1.goto(0, -50)

Atributos visuais do objeto turtle

Execute o exemplo a seguir. Os comentários são autoexplicativos. Os comandos alteram as características visuais do objeto turtle e da tela.

```
import turtle
from math import sqrt

wn = turtle.Screen()          # dá um nome ao objeto tela
wn.bgcolor("lightgreen")      # define a cor de fundo da janela
                              # pode usar as cores tradicionais:
                              # red, yellow, black, blue, etc.

tess = turtle.Turtle()
tess.color("blue")               # define a cor da caneta
tess.pensize(5)                  # define a espessura da caneta

tess.shape("turtle")    # define a forma do objeto turtle
                        # pode ser também arrow, blank, circle, classic,
                        # square, triangle

tess.speed(4)     # controla a velocidade da tartaruga.
                  # de 1 (mais lenta) a 10 (mais rápida)

tess.forward(200)
tess.stamp()        # deixa um carimbo por onde passa
tess.left(90)

tess.forward(200)
tess.stamp()
tess.left(135)

tess.forward(sqrt(200**2 + 200**2))
tess.stamp()

wn.exitonclick()    # apaga a janela após um click
```

O exemplo abaixo posiciona o objeto, sem traçar, nas extremidades dos ângulos de 0, 45°, 90°, ..., 315°. Execute e veja o resultado. Veja também os comentários autoexplicativos.

```
import turtle
from math import sqrt, sin, cos, pi

wn = turtle.Screen()          # dá um nome ao objeto tela
wn.bgcolor("lightsalmon")     # define a cor de fundo da tela

wn.screensize(600, 600)       # define o tamanho da tela
                              #(-300, 300) nos eixos x e y

ttt = turtle.Turtle()         # cria o objeto turtle ttt
ttt.shape("turtle")           # forma do objeto
ttt.color("blue")             # cor do objeto
ttt.penup()                   # levanta a pena da caneta

# posiciona e marca o objeto nos ângulos 0, 45, 90, 135, ..., 315
for i in range(8):
    alfa = pi / 4 * i
    x = 300 * cos(alfa)
    y = 300 * sin(alfa)
    ttt.goto(x, y)
    ttt.stamp()

wn.exitonclick()     # apaga a janela após um click
```

Exercícios

P15.5) Executar o programa acima alterando os vários parâmetros (cores, formas, velocidades, etc.) para se familiarizar com o resultado das várias funções.

P15.6) Refaça o P15.2 desenhando cada lado do triângulo de cor diferente – sugestão: use um comando for da seguinte forma:

```
for cores in [“blue”, “black”, “green”]:
     ...
```

P15.7) Refaça o P15.3 desenhando cada lado do pentágono de cor diferente

P15.8) Refaça o P15.4 alternado a cor dos lados entre azul e vermelho.

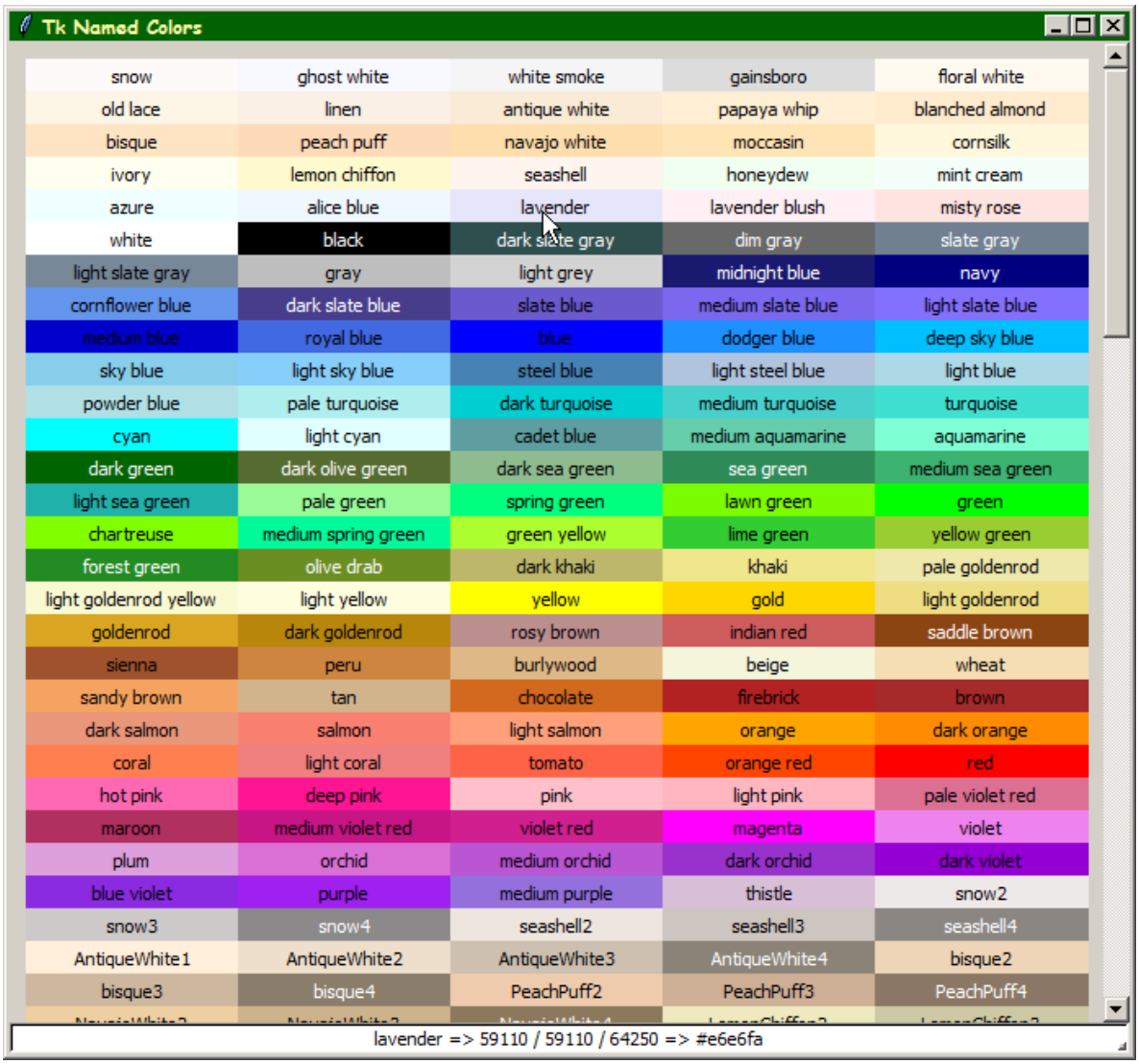
P15.9) Refaça agora a corrida das tartarugas mudando: a cor de fundo da tela, a cor do juiz e da reta de chegada, usando cores diferentes para cada tartaruga, usando a posição inicial na extrema esquerda da tela, aumentando o tamanho da tela para uma corrida mais disputada, alterando a velocidade das tartarugas.

As cores

Existem centenas de cores que podem ser utilizadas. Uma relação completa com os nomes respectivos pode ser encontrada em:

http://wiki.tcl.tk/37701

Uma pequena amostra:



As funções ou métodos do módulo turtle

Existem várias outras funções dentro do módulo além das usadas aqui. Uma relação completa pode ser encontrada em

https://docs.python.org/3.0/library/turtle.html

Objetos, atributos e métodos

Uma tartaruga (turtle) é um **objeto** do módulo turtle. A cada objeto criado dizemos que estamos criando uma instância do mesmo. O objeto contém **atributos**. No caso, a cor, a forma, etc. Esses atributos são adquiridos pelo objeto através de funções ou **métodos** deste módulo. Usamos indistintamente as palavras função e método para designar as operações que podem ser feitas sobre os objetos ou não deste módulo.

Outra aplicação – gráfico de funções

Podemos usar o módulo turtle para traçar o gráfico de funções. Para tanto basta estabelecermos o intervalo e verificar se o gráfico cabe no espaço fornecido. Senão é necessário alterar a escala no eixo y para mais ou menos.

Para exemplificar vamos determinar o tamanho da tela de 600 x 600 e uma função que no intervalo [-300, 300] tenha valores também nesse intervalo. Por exemplo f(x) = 300 * sin (x / 25). Veja o programa abaixo:

```
import turtle
from math import sin

def f(a):
    return 300 * sin(a / 25)

wn = turtle.Screen()         # dá um nome ao objeto tela
wn.bgcolor("lightsalmon")    # define a cor de fundo da tela

wn.screensize(600, 600)      # define o tamanho da tela
                             #(-300, 300) nos eixos x e y

# traçar os eixos x e y
tx = turtle.Turtle()         # cria o objeto turtle ttt
ty = turtle.Turtle()

tx.penup()
tx.goto(-300, 0)
tx.pendown()
tx.goto(300, 0)

ty.penup()
ty.goto(0, -300)
ty.pendown()
ty.goto(0, 300)
ty.left(90)

# traçar o gráfico
tg = turtle.Turtle()
tg.penup()
tg.goto(-300, f(-300))       # posiciona a caneta no inicio do grafico
tg.pendown()
tg.color("red")

for i in range(-300, 300):
    tg.goto(i, f(i))

wn.exitonclick()    # apaga a janela após um click
```

P15.10) Modifique o programa acima para traçar o gráfico da função f(x) = | x |

P15.11) Escolha outras funções que cabem neste intervalo e modifique o programa.

Modificando a escala

Para traçar o gráfico de funções genéricas, temos que ter mais informações sobre as mesmas para que caibam no gráfico. Por exemplo: funções crescentes no intervalo [0, 10]. Podemos dedicar toda a tela a esse gráfico. Vamos traçar o gráfico da função $x^3 + x^2 + x + 1$ no intervalo [0, 10].

```
import turtle
from math import sin

def f(x):
    return x * x * x + x * x + x + 1

wn = turtle.Screen()         # dá um nome ao objeto tela
wn.bgcolor("palegreen")      # define a cor de fundo da tela

wn.screensize(600, 600)      # define o tamanho da tela
                             #(-300, 300) nos eixos x e y

# traçar os eixos x e y
tx = turtle.Turtle()         # cria o objeto turtle ttt
ty = turtle.Turtle()

fmin = f(0)
fmax = f(10)

# adequa a escala do gráfico ao tamanho da tela
wn.setworldcoordinates(0, fmin, 10, fmax)

# traçar os eixos x e y
tx.penup()
tx.goto(0, 0)
tx.pendown()
tx.goto(10, 0)

ty.penup()
ty.goto(0, fmin)
ty.pendown()
ty.goto(0, fmax)
ty.left(90)

# traçar o gráfico
tg = turtle.Turtle()
tg.penup()
tg.goto(0, f(0))       # posiciona a caneta no inicio do grafico
tg.pendown()
tg.color("blueviolet")
tg.pensize(4)

i = 0
while i <= 10:
    tg.goto(i, f(i))
    i = i + 0.1

wn.exitonclick()    # apaga a janela após um click
```

A função setworldcoordinates(xmin, ymin, xmax, ymax) normaliza o tamanho do gráfico para o tamanho da tela. xmin e ymin são as coordenadas do ponto mais a esquerda e mais abaixo. xmax e ymax os da direita.

P15.12) Adapte o programa acima para traçar o gráfico no intervalo [-10, 10].

P15.13) Procure funções que você possa determinar o máximo e o mínimo num determinado intervalo e adapte o programa acima para traçar o gráfico da mesma.

P15.13) Generalize o programa acima para qualquer função no intervalo [a, b] da qual se conhece o máximo e mínimo neste intervalo. Ou seja, dados a, b, fmax, fmin e um determinado passo (exemplo 0.01), traçar o gráfico desta função.