

INSTITUTO FEDERAL
Santa Catarina

Introdução à Programação com Python

Bibliotecas Math, Fraction e Turtle

Jackson Meires

jackson.meires@ifsc.edu.br

O que é uma Biblioteca em Python?

É uma coleção de módulos de script acessíveis a um programa Python para simplificar o processo de programação e remover a necessidade de reescrever os comandos mais usados

Eles podem ser usados chamando-os / importando-os no início de um script

O que é a biblioteca Math?

- Contém várias funções para efetuar cálculos matemáticos
- Dentre esses cálculos matemáticos temos: raiz quadrada, potência, arredondamento, entre outros
- O módulo matemático possui um conjunto de métodos e constantes
- Todos os valores retornam do tipo flutuante

Diferença entre a Biblioteca Math e cMath

Math – Funções Matemáticas

O módulo `math` define funções logarítmicas, de exponenciação, trigonométricas, hiperbólicas e conversões angulares, entre outras.

cMath – Funções Matemáticas com Números Complexos

Já o módulo `cMath`, implementa funções similares, porém feitas para processar números complexos

Principais funções matemáticas Math

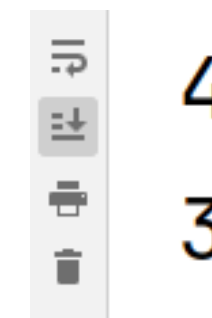
- “floor(x)”: efetua arredondamento para baixo;
- “ceil(x)”: efetua arredondamento para cima;
- “sqrt(x)”: calcula a raiz quadrada;
- “pow(base, pot)”: eleva base ao pot (potência);
- “sin(x)”: calcula o valor de seno;
- “cos(x)”: calcula o valor de cosseno;
- “log(x, base)”: calcula o log da base;
- “pi”: constante valor de PI 3,1415...;

Funções floor(x) e ceil(x)

- “floor(x)”: efetua arredondamento para baixo;
- “ceil(x)”: efetua arredondamento para cima;

```
1 import math
2
3 a = 3.8
4
5 print(math.ceil(a))
6 print(math.floor(a))
```

Resultado




```
4
3
```

Funções sqrt(x) e pow(x)

- “sqrt(x)”: calcula a raiz quadrada;
- “pow(base, pot)”: eleva base ao pot (potência);

```
1 import math
2
3 raiz = math.sqrt(36)
4 potencia = math.pow(2,3)
5
6 print(raiz)
7 print(potencia)
```

Resultado



6.0
8.0

Funções $\sin(x)$ e $\cos(x)$

- “ $\sin(x)$ ”: calcula o valor de seno;
- “ $\cos(x)$ ”: calcula o valor de cosseno;

```
1 import math
2
3 seno = math.sin(3.14/6) # 30 graus
4 cos seno = math.cos(3.14/4) # 45 graus
5
6 print("0 seno é: ", seno)
7 print("0 cos seno é: ", cos seno)
```

Resultado

```
0 seno é: 0.4997701026431024
0 cos seno é: 0.7073882691671998
```


Funções tan(x) e log(x, base)

- “tan()”: calcula o valor da tangente;
- “log(x, base)”: calcula o valor do logaritmo;

```
1 import math
2
3 # log e tan
4 tan = math.tan(3.14/6)
5 log = math.log(1000, 10)
6
7 print("0 tangente é: ", tan)
8 print("0 log é: ", log)
```

Resultado

```
0 tangente é: 0.576996400392873
0 log é: 2.9999999999999996
```

Constantes

Ex: $\pi = 3.14$ com a biblioteca Math

OBS: No Python não tem como declarar uma variável do tipo constante. Portanto, basta não fazer uma atribuição a variável

```
1 import math
2
3 pi = math.pi
4
5 print("O valor de PI é: ", pi)
```

Resultado

```
O valor de PI é: 3.141592653589793
```

Biblioteca Fractions

- Torna disponível um novo tipo de número: o tipo Fraction. O tipo Fraction é usado para representar frações e realizar aritmética racional, como:

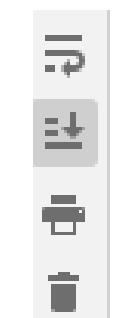
$$\frac{1}{2} + \frac{3}{4} = \frac{5}{4}$$

Biblioteca Fractions

- Para usar o módulo fractions, primeiro precisamos importá-lo “*import fractions*”
- Para criar um objeto Fraction, usamos o construtor Fraction() com dois argumentos: um numerador e um denominador. Veja aqui como definir 1/2 e 3/4:

```
1 import fractions
2
3 a = fractions.Fraction(1, 2)
4 b = fractions.Fraction(3, 4)
5
6 print(a)
7 print(b)
```

Resultado



1/2
3/4

Biblioteca Fractions

- Observe que a variável “a” não é avaliado como 0,75.
- Assim como outros números, objetos Fraction podem ser somados, e o resultado é um objeto Fraction:

```
1 import fractions
2
3 a = fractions.Fraction(1, 2)
4 b = fractions.Fraction(3, 4)
5
6 c = a + b
7
8 print(c)
```

Resultado

 5/4

Por quer não usar o fractions ao invés do tipo float?

- Os valores float são armazenados no computador usando um número limitado de bits, normalmente 64 deles.
- Isso significa que o intervalo de valores que os objetos float podem armazenar é limitado. Por exemplo, $0,5^{1075}$ não pode ser representado como um valor float e, portanto, é avaliado como 0:

```
print(0.5**1075)
```

Resultado

5e-324

Por que não usar o fractions ao invés do tipo float?

O intervalo de valores representáveis com objetos fractions. Fraction é muito, muito maior e limitado apenas pela memória disponível, assim como para o tipo int. Assim, podemos facilmente calcular $1/2^{1075}$:

```
1 import fractions
2
3 a = fractions.Fraction(1, 2)
4
5 print(a**1075)
```

Resultado

```
1/4048045066146212367049906934378346140991132
99528284236713802716054860679135990693783920
76740287424899037415572863362382277961747477
15869537340267998814770198430348485531327227
28933815484186432682479535356945490137124014
96684938539723620671129831911268162011302471
75391046668292304610050643726550172920125266
15415482186989568
```

Por que não usar sempre o tipo fractions.Fraction?

Porque as expressões envolvendo valores float são avaliadas muito mais rapidamente do que as expressões envolvendo valores fractions.Fraction.

Biblioteca Turtle graphics

- Permite que um usuário desenhe linhas e formas de um modo semelhante ao uso de uma caneta sobre o papel. Ela foi desenvolvida para fins de ensino de programação
- Ela possui mais de 80 métodos de classe e funções.



IDE Online Python com o Turtle: https://replit.com/languages/python_turtle

Documentação: <https://docs.python.org/pt-br/3.9/library/turtle.html?highlight=turtle>

Biblioteca Turtle graphics – Comandos Básicos

```
1 import turtle
2
3 t = turtle.Turtle() # cria a tartaruga
4 t.shape("turtle") # define o tipo do objeto
5 t.color("orange") # define a cor do objeto
6 t.pensize(5) # largura da seta
7 t.forward(100) # quantos pixels vai se mover
8
9 turtle.done() #mantem janela aberta
```

Resultado



Biblioteca Turtle graphics – Comandos Básicos

Movimentando a Tartaruga

- Para frente
 - **turtle.forward(valor)**
 - **turtle.fd(valor)**
 - Um número real
- Para trás
 - **turtle.backward(valor)**
 - **turtle.bk(valor)**
 - **turtle.back(valor)**
 - Um número real

Biblioteca Turtle graphics

Girando a tartaruga

- Girando para a direita
 - **turtle.right(valor)**
 - **turtle.rt(valor)**
- Girando para a esquerda
 - **turtle.left(valor)**
 - **turtle.lt(valor)**
- Trabalhando com ângulos
 - Graus (padrão)
 - **turtle.degrees()**
 - Radianos
 - **turtle.radians()**
- Valor do ângulo para virar a tartaruga

Biblioteca Turtle graphics

Largura do Traço

- **turtle.pensize(valor)**
- **turtle.width(valor)**
 - Valor deve ser um número positivo

Biblioteca Turtle graphics

Mudando o formato da tartaruga

–**turtle.shape([valor])**

- “arrow”
- “turtle”
- “circle”
- “square”
- “triangle”
- “classic”

Biblioteca Turtle graphics

Indo de um ponto a ponto

- `turtle.goto(valor)`
- `turtle.setpos(valor)`
- `turtle.setposition(valor)`
 - Valor é um par de coordenadas cartesianas tal qual 0,0

Biblioteca Turtle graphics

Risca ou não Risca

- Para riscar, abaixe a caneta
 - `turtle.pendown()`
 - `turtle.pd()`
 - `turtle.down()`
- Para não riscar, levante a caneta
 - `turtle.penup()`
 - `turtle.pu()`
 - `turtle.up()`

Biblioteca Turtle graphics

Cores

- Cor da Caneta
 - **turtle.pencolor**([valor])
- Cor do preenchimento
 - **turtle.fillcolor**([valor])
- Valor pode ser
 - Uma string como "red", "green" ou "#336699"
 - Uma tupla rgb (100, 200, 150)
 - Três números inteiros representando rgb
 - fillcolor(100, 200, 150)

Biblioteca Turtle graphics

Mais comandos úteis

- Limpando a tela
 - `turtle.clear()`
- Começando tudo de novo
 - `turtle.reset()`
- Preenchendo um desenho
 - Execute `turtle.fill(True)` antes de começar o desenho e `turtle.fill(False)` após terminá-lo
- Levando a tartaruga para (0, 0)
 - `turtle.home()`

Biblioteca Turtle graphics

Mais comandos úteis

- Cade a tartaruga?

- `turtle.showturtle()`

- `turtle.st()`

- E se eu quiser escondê-la?

- `turtle.hideturtle()`

- `turtle.ht()`

Biblioteca Turtle graphics

Exemplo Prático

```
1  import turtle
2
3  u = turtle.Turtle()
4  u.right(90)
5  u.forward(100)
6  u.left(90)
7  u.forward(100)
8  u.left(90)
9  u.forward(100)
10 turtle.done() #mantem janela aberta
```

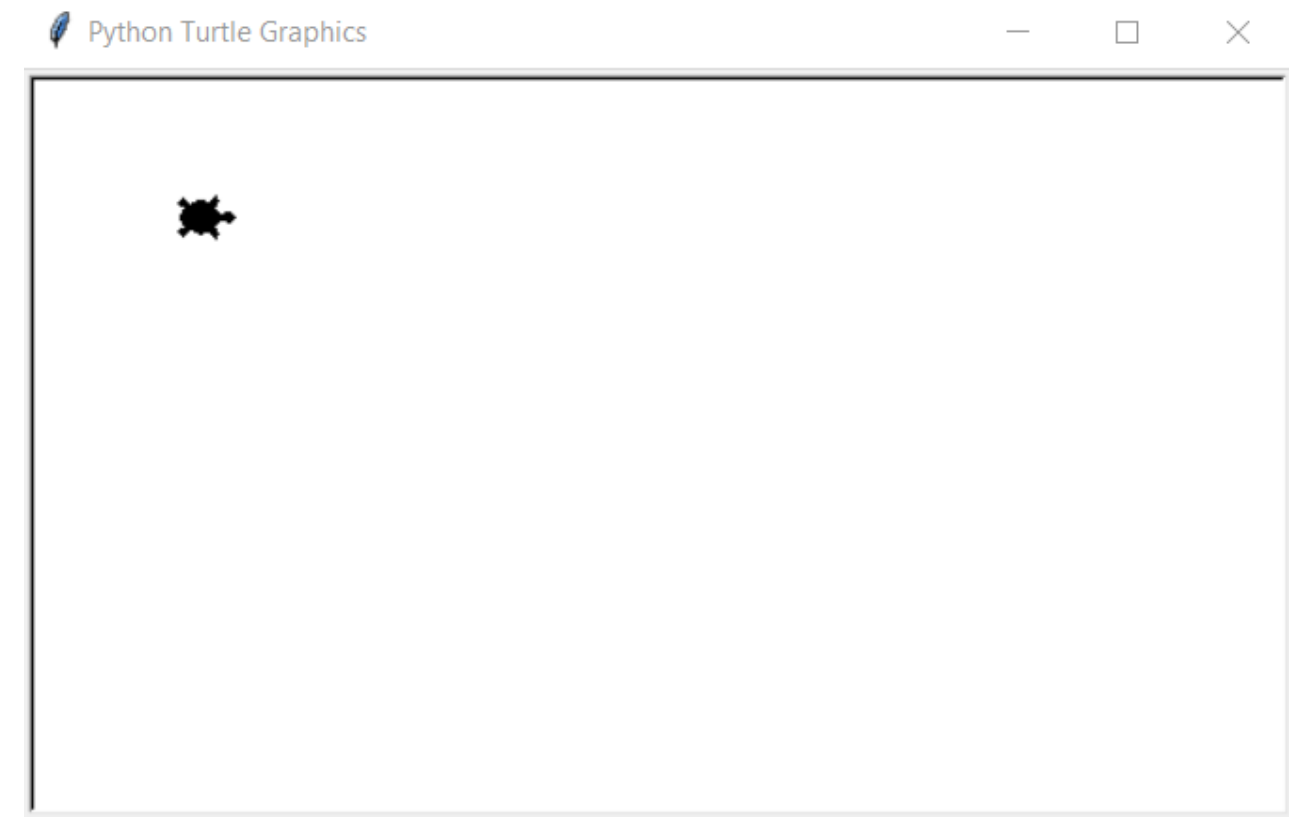
Resultado



Biblioteca Turtle graphics

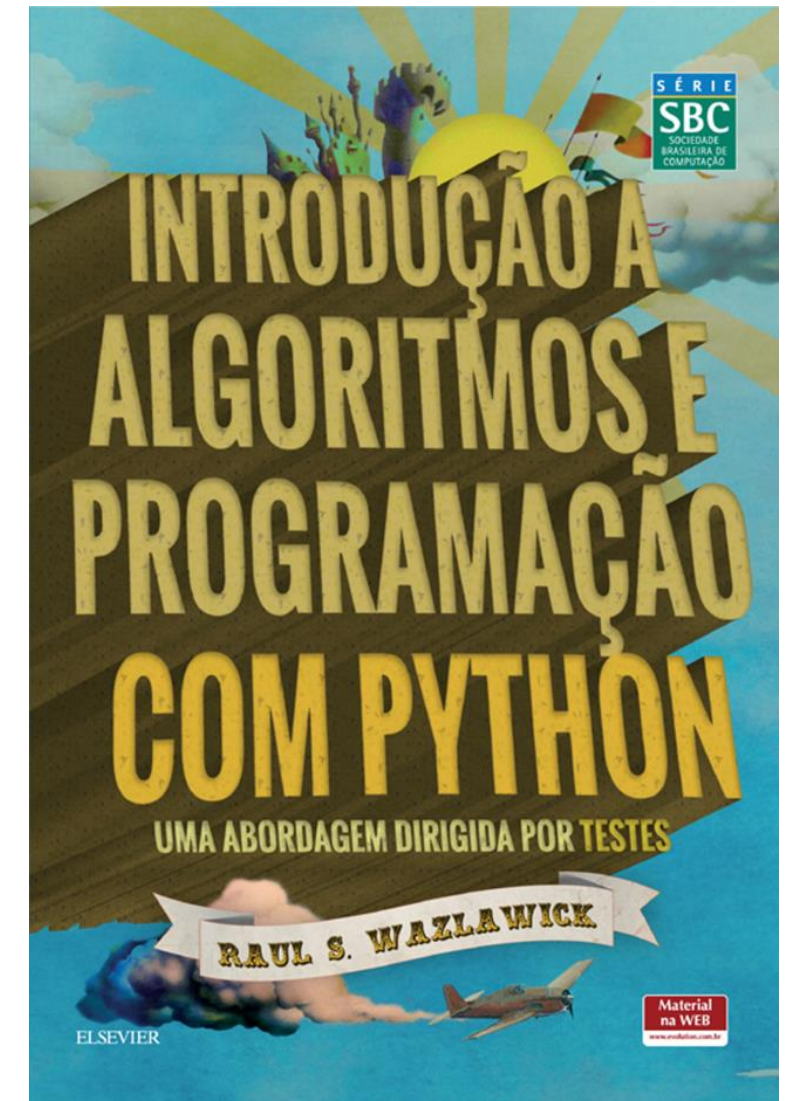
Mudando a posição de Início da tartaruga na tela

```
1 import turtle
2
3 t = turtle.Turtle() #inicia o objeto tartaruga
4 t.penup() # oculta a caneta da tartaruga
5 t.setx(-200) # define posição eixo X
6 t.sety(100) # define posição eixo Y
7 t.pendown() # exibe a caneta da tartaruga
8 t.shape("turtle")
```



Material

- **Livro**
 - WAZLAWICK , Raul. Introdução a Algoritmos e Programação com Python. LTC, 2017.
- **Documentação Oficial – Python**
 - <https://docs.python.org/pt-br/3/>
- **W3schools**
 - <https://www.w3schools.com/python/>
- **IDE Online - Gratuita**
 - <https://replit.com/languages/python3>



Mais bibliotecas Python

- **Postar no Instagram**

- <https://instagrabot.github.io/docs/en/>

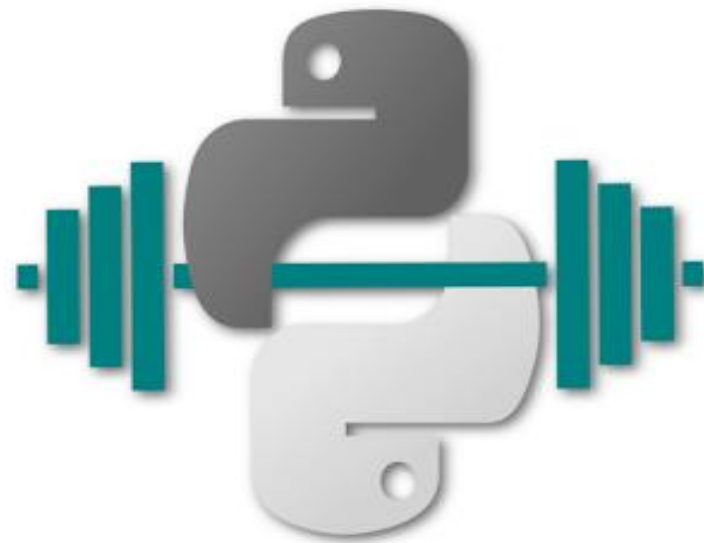
- **Diversas por Finalidades**

- <https://www.hashtagtreinamentos.com/bibliotecas-mais-importantes-do-python>



Exercícios

Vamos lá!



Dúvidas?

