Python para Matemáticos

Tertuliano Franco (UFBA)

IX Encontro da Pós em Matemática da UFBA
https://encontropgmat.ufba.br/

18/11/2024

Python

- **O que é Python?**

Python

- **O que é Python?** Python é uma linguagem de programação interpretada, de alto nível, multifuncional, de tipagem dinâmica e forte, e orientada ao objeto.

Python

- **O que é Python?** Python é uma linguagem de programação interpretada, de alto nível, multifuncional, de tipagem dinâmica e forte, e orientada ao objeto.
- **Por que Python?** Para imprimir *"Hello, world!"* em Python, basta fazer

```
print("Hello, world!")
```

Python

- **O que é Python?** Python é uma linguagem de programação interpretada, de alto nível, multifuncional, de tipagem dinâmica e forte, e orientada ao objeto.
- **Por que Python?** Para imprimir *"Hello, world!"* em Python, basta fazer

```
print("Hello, world!")
```

Python

- **O que é Python?** Python é uma linguagem de programação interpretada, de alto nível, multifuncional, de tipagem dinâmica e forte, e orientada ao objeto.
- **Por que Python?** Para imprimir *"Hello, world!"* em Python, basta fazer

```
print("Hello, world!")
```

☛ Crie um arquivo que imprima algo. Compile o arquivo no terminal ou acesse `https://www.programiz.com/python-programming/online-compiler/`

Python

- **Por que Python?** Em C++ seria necessário escrever algo como

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!";
    return 0;
}
```

Python

- **Por que Python?** Em C++ seria necessário escrever algo como

```
#include <iostream>

int main() {
    std::cout << "Hello, world!";
    return 0;
}
```

Em C seria necessário escrever algo como

```
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
```

Python

- **Por que Python?** Em Java seria necessário escrever algo como

```
public class HelloWorld{
    public static void main(String []args){
        System.out.println("Hello, world!");
    }
}
```

Estruturas Básicas

- Inteiro (integer)
- Ponto flutuante (float)
- Palavra (string)

👉 Abra uma sessão interativa de Python no terminal e digite `0.1+0.2`

Estruturas Básicas

- Inteiro (integer)
- Ponto flutuante (float)
- Palavra (string)

👉 Abra uma sessão interativa de Python no terminal e digite `0.1+0.2`

👉 Compile (em um arquivo ou sessão interativa)

```
x = 7
print(type(x))
y = '7'
print(type(y))
z = 14/2
print(type(z))
```

Métodos e Funções

- *Método*: função associada a uma classe específica

Métodos e Funções

- *Método*: função associada a uma classe específica
- *Função*: função, que a princípio não está associada a nenhuma classe específica

Métodos e Funções

- *Método*: função associada a uma classe específica
- *Função*: função, que a princípio não está associada a nenhuma classe específica

👉 Compile (em um arquivo ou sessão interativa)

```
x = 'FULANO DE TAL'
y = x.title()
print(y)
```

👉 Compile (em um arquivo ou sessão interativa)

```
from math import floor
x = floor(7.1)
print(x)
```

Métodos e Funções

- Compile

```
def quadrado(k):
    x = k**2
    return x

print(quadrado(7))
```

Métodos e Funções

- Compile

```
def quadrado(k):
    x = k**2
    return x

print(quadrado(7))
```

- Adivinhe o que aconteceria aqui:

```
a = 4
def quadrado(k):
    a = 5
    x = k**2
    return x

print(a)
```

Listas, Tuplas, Dicionários e Conjuntos

👉 Compile

```
x = [1, 2, 3]
x.append(4)
print(x)
```

Listas, Tuplas, Dicionários e Conjuntos

👉 Compile

```
x = [1, 2, 3]
x.append(4)
print(x)
```

👉 Compile

```
x = [6, 7, 2, 3, 1, 4, 0]
y = x[1:2]
print(x)
```

Listas, Tuplas, Dicionários e Conjuntos

👉 Compile

```
x = ['a', 'b', 'c']
x[0]= 'd'
print(x)
```

Listas, Tuplas, Dicionários e Conjuntos

👉 Compile

```
x = ['a', 'b', 'c']
x[0]= 'd'
print(x)
```

👉 Compile

```
x = ('a', 'b', 'c')
x[0]= 'd'
print(x)
```

Listas, Tuplas, Dicionários e Conjuntos

👉 Compile

```
Dicionario = {'Tertu': 5,
   'Dirk': 4,
   'Eduardo': 7 }
x = Dicionario['Tertu']
print(x)
```

Listas, Tuplas, Dicionários e Conjuntos

👉 Compile

```
Dicionario = {'Tertu': 5,
   'Dirk': 4,
   'Eduardo': 7 }
x = Dicionario['Tertu']
print(x)
```

👉 Compile

```
Conjunto = {'a', 'b', 'c'}
x = len(Conjunto)
print(x)
y = 'a' in Conjunto
print(y)
```

`if`, `elif`, `else`

- 👉 Entenda o que acontecerá aqui antes de compilar.

if, elif, else

👉 Entenda o que acontecerá aqui antes de compilar.

```
nome = input('Digite a sua idade')

if nome >= 18:
    print('Você émaior de idade!')
elif nome >= 15:
    print('Você éadolescente')
else:
    print('Você écriança!')
```

if, elif, else

👉 Entenda o que acontecerá aqui antes de compilar.

```
nome = input('Digite a sua idade')

if nome >= 18:
    print('Você émaior de idade!')
elif nome >= 15:
    print('Você éadolescente')
else:
    print('Você écriança!')
```

For, While, Break, Continue

- 👉 Some os números de 1 a 1000.

For, While, Break, Continue

- Some os números de 1 a 1000.

```
S = 0
for k in range(1000):
    S = S + (k+1)
print(S)
```

For, While, Break, Continue

- Some os números de 1 a 1000.

```
S = 0
for k in range(1000):
    S = S + (k+1)
print(S)
```

- Note que `range()` é um iterador, não uma lista! Some novamente os números de 1 a 1000.

For, While, Break, Continue

- Some os números de 1 a 1000.

```
S = 0
for k in range(1000):
    S = S + (k+1)
print(S)
```

- Note que `range()` é um iterador, não uma lista! Some novamente os números de 1 a 1000.

```
import time
S = 0
lista = list(range(1000))
t_0 = time.perf_counter()
for k in lista:
S = S + (k+1)
print(S)
t_1 = time.perf_counter()
print("tempo gasto em segundos:", t1_stop-t1_start)
```

For, While, Break, Continue

- Defina a função fatorial usando `while`

```
def fatorial(n):
    S = 1
    while n > 1:
        S *= n
        n -= 1
    return S
x = fatorial(4)
print(x)
```

For, While, Break, Continue

- Defina a função fatorial usando `while`

```
def fatorial(n):
    S = 1
    while n > 1:
        S *= n
        n -= 1
    return S
x = fatorial(4)
print(x)
```

- `break` and `continue`

class

👉 Classe é uma estrutura de dados personalizada, a qual pode ter métodos e propriedades, e pode criar instâncias.

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
    def info(self):
        print(f'{self.nome} tem {self.idade} anos')

pessoa_1 = Pessoa('Tertu', 42)

print(pessoa_1.nome)
print(pessoa_1.idade)
pessoa_1.info()
```

class

👉 Classe é uma estrutura de dados personalizada, a qual pode ter métodos e propriedades, e pode criar instâncias.

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
    def info(self):
        print(f'{self.nome} tem {self.idade} anos')

pessoa_1 = Pessoa('Tertu', 42)

print(pessoa_1.nome)
print(pessoa_1.idade)
pessoa_1.info()
```

Bibliotecas (dentre mais de 60.000)

- `math`
- `random`
- `matplotlib`
- `numpy`
- `pygame` and `arcade`
- `networkx`
- `itertools`
- `sympy`

ChatGPT

- 👉 ChatGPT: Pouco inteligente, mas muito bem preparado!

Sympy

Sympy

- `sympy`: computação simbólica em Python

Sympy

- `sympy`: computação simbólica em Python

- se não tiver notebook com python e sympy, navegue para `https://www.sympy.org/en/shell.html`

Sympy

- `sympy`: computação simbólica em Python
- se não tiver notebook com python e sympy, navegue para `https://www.sympy.org/en/shell.html`
- O acontecerá aqui?

Sympy

- `sympy`: computação simbólica em Python

- se não tiver notebook com python e sympy, navegue para `https://www.sympy.org/en/shell.html`

- O acontecerá aqui?

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x, y, z = symbols('x y z')
pprint(x+y+z+2*x)
```

Sympy

Sympy

👉 Você pode usar qualquer nome que quiser para um símbolo em `sympy`, mas é melhor evitar as letras `Q`, `O`, `S`, `I`, `N` e `E`, porque elas possuem significados especiais em `sympy`.

Sympy

👉 Você pode usar qualquer nome que quiser para um símbolo em `sympy`, mas é melhor evitar as letras `Q`, `O`, `S`, `I`, `N` e `E`, porque elas possuem significados especiais em `sympy`.

```
from sympy import *
print(ask(Q.prime(7)))
```

Sympy

☛ Você pode usar qualquer nome que quiser para um símbolo em `sympy`, mas é melhor evitar as letras `Q`, `O`, `S`, `I`, `N` e `E`, porque elas possuem significados especiais em `sympy`.

```
from sympy import *
print(ask(Q.prime(7)))
```

☛ A função `S()` é a função `sympify` que não deve ser confundida com a função `simplify`, que simplifica expressões.

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

L = sympify('2**2/3 + 5 + 34/8')
pprint(L)
```

Sympy

👉 A letra `O` corresponde a “o grande”, a notação matemática para dizer que uma função é menor ou igual a uma constante vezes a outra.

Sympy

- 👉 A letra `O` corresponde a “o grande”, a notação matemática para dizer que uma função é menor ou igual a uma constante vezes a outra.
- 👉 A letra `I` representa o número complexo imaginário $i = \sqrt{-1}$.

Sympy

- A letra `O` corresponde a "o grande", a notação matemática para dizer que uma função é menor ou igual a uma constante vezes a outra.
- A letra `I` representa o número complexo imaginário $i = \sqrt{-1}$.
- A função `N()` serve para obter valores numéricos de uma expressão.
- A letra E representa o número de Euler e.

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)
pprint(sqrt(2)*pi)
L = N(sqrt(2)*pi)
print(L)
```

Sympy

☛ É possível especificar hipóteses sobre o símbolo em Python, se é real, se é inteiro, se é função, entre muitas outras. Além disso, há métodos disponíveis para checar as hipóteses sobre o símbolo.

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x = Symbol('x', positive=True, integer=False)
pprint(simplify(sqrt(x**2)))

print(x.is_positive)
print(x.assumptions0)
```

Sympy

👉 A biblioteca `sympy` possui uma sub-biblioteca com todas as letras gregas e latinas já tranformadas em símbolos, chamada abc.

Sympy

👉 A biblioteca `sympy` possui uma sub-biblioteca com todas as letras gregas e latinas já tranformadas em símbolos, chamada abc.

```
from sympy import *
from sympy.abc import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

pprint(alpha+beta+lamda+alpha)
```

Sympy

👉 Caso você precise utilizar uma quantidade grande de símbolos, é possível defini-los todos de uma vez.

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x, y, z = symbols('x y z')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
```

Sympy

☛ Caso você precise utilizar uma quantidade grande de símbolos, é possível defini-los todos de uma vez.

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x, y, z = symbols('x y z')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
```

ou para muitos mesmo:

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

tupla = symbols('x:100')
pprint(tupla[0]+tupla[0]+tupla[99])
```

Sympy

- Simplificações (há muitos tipos!)

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)
x, y = symbols('x y')

# simplifica
L = simplify(sin(x)**2 + cos(x)**2-1)
pprint(L)
# expande
L = expand((x+y)**18)
pprint(L)
# fatora
H = factor(L)
pprint(H)
# lista os fatores
H = factor_list(L)
#pprint(H)
```

Sympy

```
# expande em frações parciais
expr = (4*x**3 + 21*x**2 + 10*x + 12)/(x**4 + 5*x**3 +
    5*x**2 + 4*x)
L = apart(expr)
print_latex(L)
pprint(L)
# cancela funções racionais
H = cancel((x**2 + 2*x + 1)/(x**2 + x))
pprint(H)
# simplifica fatoriais
H = combsimp(factorial(n)/factorial(n - 3))
pprint(H)
L = combsimp(binomial(n+1, k+1)/binomial(n, k))
pprint(L)
# simplifica funções trigonométricas
H = trigsimp(sin(x)**4 - 2*cos(x)**2*sin(x)**2 +
    cos(x)**4)
pprint(H)
```

Sympy

```
# expande funções trigonometricas
H = expand_trig(sin(x + y))
pprint(H)
# coloca em evidência
expressao = x*y + x - 3 + 2*x**2 - z*x**2 + x**3
H = collect(expressao, x)
pprint(H)
# faz o mmc de frações
expressao = (x**2+y)/(x*y+4) +(x**3*y+y**2)/(x+y+x*y)
L = together(expressao)
pprint(L)
# para reescrever em termos de uma dada função
L = Eq(tan(x), tan(x).rewrite(sin))
pprint(L)
H = Eq(factorial(x), factorial(x).rewrite(gamma))
pprint(H)
```

Sympy

👉 Para compor funções, usamos o método subs:

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x, y, f = symbols('x y f')
f = 1 + 1/x
f2 = f.subs(x,f)
pprint(f2)
```

Sympy

- Para calcular limites, usamos a função `limit()`

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x = Symbol('x')

# limite em zero
L = limit(sin(x)/x, x, 0)
pprint(L)

# limite no infinito
M = limit(exp(x), x, oo)
pprint(M)

# limite lateral
N = limit(1/x, x, 0, '-')
pprint(N)
```

Sympy

- Para derivar uma função, usamos a função `diff()`

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x = Symbol('x')
f = Function('f')
L = diff(f(x)*cos(x), x, 2)
pprint(L)
```

Sympy

- Para integrar, usamos `integrate()` ou `Integral`

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x = Symbol('x')

#integral indefinida
L = integrate(exp(x) * sin(x), x)
pprint(L)
#integral definida
M = integrate(sin(x), (x, 0, pi))
pprint(M)
#integral nao aplicada
N = Integral(acos(x),x)
H = Eq(N, N.doit())
pprint(H)
```

Sympy

👉 Para obter expansão em séries de potências, usamos a função `series()`

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x = Symbol('x')

#expansao em serie
L = series(cos(x)**2, x, n=10)
pprint(L)
```

Sympy

- Para resolver equações, podemos usar `solveset()`

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x = Symbol('x')

# para resolver equações
L = solveset(x**2 - 1, x)
pprint(L)
```

Sympy

☛ Podemos resolver equações diferenciais através da função `dsolve()`.

```
from sympy import *
init_printing(use_unicode=True, wrap_line=False,
    no_global=True)

x = Symbol('x')
f = Function('f')

# equação diferencial
eq_dif = Eq(f(x) - 2*f(x).diff(x) + f(x), sin(x))
pprint(eq_dif)

# solução da equação diferencial
L = dsolve(eq_dif, f(x))
pprint(L)
```

👉 Compile: `import this`

👉 Compile: `import this`

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
*Although never is often better than *right* now.*
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!