Parte II - Aula 1
Laboratório de informática
Conceitos básicos do SageMath

Alex Abreu

Conteúdo

Nesta parte do curso vamos aprender a usar o Sage, e como ele pode nos ajudar a resolver problemas matemáticos.

Comece criando um *sageworksheet* nomeado *Aula1*.

1 Operações aritméticas

Experimente realizar as seguintes operações (você não precisa digitar o *sage:*). Use *shif+enter* para compilar.

```
sage: 3+5
8
sage: 3  +  5
8
sage: 4*7
28
sage: 23-40
-17
sage: 7^3
343
sage: 40/14
20/7
```

O acento circunflexo ^ é o símbolo para exponenciação, podendo também usarmos **. Note que no último comando o Sage simplesmente reproduz o 20/7. Se quisermos uma aproximação numérica podemos usar a função `N` (ou `n`).

```
sage: N(20/7)
2.85714285714286
sage: n(20/7)
```

```
2.85714285714286
```

Podemos também avaliar expressões do tipo

```
sage: 3+7^3*3/7+1
151
```

Você pode se perguntar em que ordem o Sage realiza as operações. Primeiro são calculados as exponenciações, depois multiplicações e divisões, por fim adições e subtrações. Quando em dúvida, use parentêses.

```
sage: 3+((7^3)*3)/7+1
151
```

Se quisermos somar antes de multiplicar, ou realizar operações que não estão na ordem usual, basta usar parentêses

```
sage: (3+7)^(3*3)/(7+1)
125000000
```

O Sage já vem equipado com outras operações aritméticas. Podemos por exemplo realizar divisão com resto. Usamos // para o quociente e % para o resto.

```
sage: 20//7
2
sage: 20%7
6
sage: 2*7+6
20
```

Podemos também calcular raízes quadradas usando o comando `sqrt`.

```
sage: sqrt(36)
6
sage: sqrt(3)
sqrt(3)
```

Note que, como antes, ele retorna $\sqrt{3}$. Se quisermos uma aproximação numérica usamos a função `N`, ou podemos escrever `sqrt(3.0)`.

```
sage: N(sqrt(3))
1.73205080756888
sage: sqrt(3.0)
1.73205080756888
```

O Sage responde $\sqrt{3}$ quando perguntado qual é a raiz quadrada de 3 porque é um sistema de computação simbólico em vez de numérico. Isto quer dizer que ele realiza cálculos exatamente e não aproximados. A principal vantagem é que evitamos erro de aproximações, como o que acontece abaixo.

```
sage: (1+10^50)-10^50
1
sage: n(1+10^50)-n(10^50)
0.000000000000000
```

Não existe uma função específica para raízes cúbicas, mas podemos usar a exponenciação.

```
sage: 36^(1/2)
6
sage: 125^(1/3)
```

```
5
sage: 5^(1/5)
5^(1/5)
sage: N((5)^(1/5), digits=30)
1.37972966146121483239006346422
```

Note que a exponenciação com expoentes racionais só funciona se o exponenciando for positivo, tente calcular a raiz cúbica de -8.

```
sage: (-8)^{1/3}
```

1.1 Atividades

1. Calcule $\frac{(2+3^4)}{5} + 45 \cdot 60 - 7^4$.
2. Calcule a raiz quadrada de 344569.
3. Calcule o resto da divisão de 504930234 por 46984

2 Constantes e funções predefinidas

No Sage, algumas constantes e funções já vem predefinidas. Por exemplo as constantes π e e já vem definidas, bem como a unidade imaginária i.

```
sage: pi
pi
sage: N(pi)
3.14159265358979
sage: e
e
sage: N(e)
2.71828182845905
sage: I
I
sage: I^2
-1
```

Uma outra constante que já vem definida é ∞.

```
sage: oo
+Infinity
```

Algumas funções já definidas incluem as funções trigonométricas e as funções logaritmo e exponencial. Lembramos que as funções trigonométricas são para ângulos em radianos.

Abaixo alguns exemplos para funções trigonométricas. Aqui também os cálculos são exatos.

```
sage: sin(pi/6)
1/2
sage: cos(pi/6)
1/2*sqrt(3)
sage: cos(2*pi/3)
-1/2
sage: tan(pi/4)
1
sage: tan(pi/2)
Infinity
sage: arctan(1)
1/4*pi
```

```
sage: arcsin(1/2)
1/6*pi
sage: arccos(0.7)
0.795398830184144
```

Lembre que um ângulo de medida π radianos mede 180°. Podemos então passar de radianos para graus dividindo por π e multiplicando por 180°.

```
sage: (pi/6)/pi*180
30
sage: 30*pi/180
1/6*pi
```

Abaixo alguns exemplos para funções logaritmo e exponencial. Lembramos que as funções logaritmo e exponencial geralmente são definidas na base e.

```
sage: log(3)
log(3)
sage: N(log(3))
1.09861228866811
sage: log(e)
1
sage: exp(2)
e^2
sage: e^2
e^2
sage: e^(log(3))
3
sage: 2^(log(7)/log(2))
7
```

Se quisermos calcular o logaritmo em outras bases, como $\log_3(9)$ ou $\log_2(64)$, usamos a seguinte função.

```
sage: log(9,3)
2
sage: log(10,3)
log(10)/log(3)
sage: log(64,2)
6
sage: 5^(log(20,5))
20
```

Também já está definida a função *módulo* $|x|$, ou valor absoluto. Relacionada a ela, temos a função *sinal*, que nos diz o sinal do número.

```
sage: abs(10)
10
sage: abs(-10)
10
sage: sign(-10)
-1
sage: sign(10)
1
```

2.1 Atividades

1. Calcule o seno, cosseno e tangente de $15°$.
2. Qual ângulo, em graus e aproximadamente, possui tangente 2.23.
3. Calcule com precisão de 20 dígitos o logaritmo na base e de $3\pi^3$.
4. Calcule uma aproximação numérica de $\sqrt[3]{-45}$.
5. Calcule $4^{\log_4(11)}$, $5^{\log_5(28)}$, $\pi^{\log_\pi(47)}$. Em geral, qual vai ser o resultado de $x^{\log_x(y)}$?
6. Calcule $\log_4(64)$ e $\log(64)/\log(4)$. Você sabe porque os resultados são iguais? Se não, pergunte ao seu professor(a) de Matemática básica.

3 Expressões Lógicas

Além de trabalhar com números inteiros, reais e complexos, o Sage também trabalha com valores Booleanos. São eles `Verdadeiro` e `Falso`, ou em inglês `True` e `False`. Por exemplo, podemos comparar números.

```
sage: 5>3
True
sage: 5>=3
True
sage: 5<3
False
sage: 2<4
True
sage: 2^2+3==7
True
sage: 59242==(59242//31)*31+59242%31
True
```

Para algumas igualdades, precisamos usar o comando `bool` para obter o valor Booleano como resposta.

```
sage: log(16,2)==log(16)/log(2)
4 == 4
sage: bool(log(16,2)==log(16)/log(2))
True
sage: 5^(log(23,5))==23
23 == 23
sage: bool(5^(log(23,5))==23)
True
```

Da mesma maneira que podemos realizar operações com números, podemos realizar operações com valores booleano. Estas operações são `e`, `ou` e `nao`, ou em inglês `and`, `or` e `not`

```
sage: True and False
False
sage: True or False
True
sage: not True
False
sage: 2<4 and 4>3
True
sage: not 5>3
```

```
False
sage: 3==4 or 4>3
True
```

3.1 Atividades

1. É verdade que 11^9 é maior que 9^{11}?
2. O resto da divisão de 349532 por 9 é igual a 8?
3. O número $\sqrt{2}-1$ é maior, igual ou menor que $\frac{1}{\sqrt{2}+1}$?
4. O número $\frac{46}{\sqrt{47}-1}$ é maior, igual ou menor que $\sqrt{47}+1$?

4 Atribuições

Você provavelmente notou que nos exemplos da seção anterior escrevemos `==` quando queremos saber se dois números são iguais. Isto se dá porque o sinal `=` é usado para atribuição de variáveis.

```
sage: a=3
sage: a
3
sage: 2*a^2
18
```

Podemos usar palavras e números para representar variáveis.

```
sage: resultado2=5
sage: resultado2
5
sage: resultado2-3
2
```

É recomendado nomear suas variáveis de maneira intuitiva. Por exemplo, se estivermos estudando um quadrado de lado 5, escreveríamos:

```
sage: lado=5
sage: lado
5
sage: area=lado^2
sage: area
25
```

Note que se tentarmos realizar operações com variáveis que não foram definidas, aparecerá uma mensagem de erro. Tente compilar o seguinte comando.

```
sage: b^2
```

Note que uma vez atribuído um valor a uma variável, ele continuará na memória.

```
sage: a
3
```

Podemos trocar o valor de uma variável, simplesmente atribuindo outro valor.

```
sage: a=5
sage: a
5
```

Note que podemos usar a própria variável para atribuir novos valores.

```
sage: a=a+1
sage: a
6
sage: a=a^2
sage: a
36
sage: a=a^2
sage: a
1296
```

Também podemos definir várias variáveis em uma mesma linha.

```
sage: a, b, c = 3, 7, 11
sage: a
3
sage: b
7
sage: c
11
sage: a+b+c
21
```

À variáveis também podem estar atribuídos valores Booleanos.

```
sage: V= True
sage: V
True
sage: F=False
sage: F
False
sage: V and F
False
```

Ou até mesmo texto (em linguagem de programação se diz cadeia de caracteres ou *string*).

```
sage: u='Ola '
sage: v='mundo'
sage: u+v
Ola mundo
```

4.1 Atividades

1. Defina uma variável `b` com valor 9 e depois atribua a `b` o valor `2*b` três vezes. Qual o valor final da variável `b`.

2. Considere o código

   ```
   sage: a=5
   sage: a=3*a
   sage: b=3
   sage: a=2
   sage: a=3*b+a
   ```

   Sem compilar, qual o valor final da variável a.

5 Variáveis simbólicas e expressões

Nas últimas seções aprendemos a atribuir valores numéricos a variáveis, ou até mesmo valores booleanos e strings. Mas, em matemática, muitas vezes precisamos trabalhar com variáveis que, a princípio, não sabemos que número representam (e muitas vezes as variáveis são apenas objetos simbólicos).

Por exemplo, se quisermos resolver a equação $a^2 + 3a - 4 = 0$, podemos checar usando o Sage, que $a = 1$ e $a = -4$ são soluções.

```
sage: a=1
sage: a^2+3*a-4==0
True
sage: a=-4
sage: a^2+3*a-4==0
True
```

Obviamente, seria muito mais útil se pudéssemos usar o Sage para obter as soluções para a equação. O comando `solve` faz exatamente isso (vamos aprender mais sobre este comando na próxima aula). Mas observe que se escrevermos

```
sage: solve(a^2+3*a-4==0,a)
```

o Sage reclama que a variável `a` não foi definida. Dessa forma, precisamos definir uma variável simbólica utilizando o comando `var`, da seguinte maneira:

```
sage: a=var('a')
sage: solve(a^2+3*a-4==0,a)
[
a == 1,
a == -4
]
```

Recapitulando, existem dois tipos de variáveis. As variáveis de programação, que chamaremos apenas de variáveis. Estas variáveis servem para guardar na memória do computador determinados objetos matemáticos (ou texto), como vimos na seção anterior. As variáveis matemáticas, que chamaremos de variáveis simbólicas, são alguns desses objetos matemáticos.

O Sage quando é inicializado, não possui variáveis simbólicas definidas (com excessão da variável simbólica x), portanto, os únicos objetos matemáticos que podemos usar são números e expressões simbólicas envolvendo o x. Se quisermos usar outras variáveis simbólicas precisamos defini-las como acima.

Uma vez definida uma variável simbólica, podemos atribui-la à variáveis de programação. (Note que a variável simbólica a já foi definida anteriormente.)

```
sage: equacao= a^2 +3*a-4==0
sage: equacao
a^2 + 3*a - 4 == 0
sage: solve(equacao,a)
[
a == 1,
a == -4
]
```

Note que a variável de progamação `equacao` está atribuída, na memória do computador, à equação $a^2 + 3a - 4 = 0$ que é um objeto matemático que usa a variável simbólica a. Note que poderíamos ter escolhido outro nome para a variável `equacao`, como, por exemplo

```
sage: b= a^2+3*a-4==0
sage: b
```

```
a^2 + 3*a - 4 == 0
sage: solve(b,a)
[
a == 1,
a == -4
]
```

mas, é usual nomear suas variáveis (de programação) de forma sugestiva. Como a variável representa uma equação, a nomeamos de `equacao`.

Note que os nomes entre as variáveis e as variáveis simbólicas não precisam ser relacionados. Mais ainda, a variável `y` e a variável simbólica y não interagem. Podemos escrever.

```
sage: z=var('y')
sage: y=4
sage: y^2*z
16*y
```

Acima, à variável `y` está atribuído o número 4 e à variável `z` está atribuída a variável simbólica y. Apesar das variáveis e variáveis simbólicas não interagirem, é convenção que a variável simbólica y seja atribuída à variável `y`.

```
sage: y=var('y')
sage: y
y
```

Agora que aprendemos a diferença entre variáveis e variáveis simbólicas, vamos aprender a manipular expressões simbólicas. Começando com substituição. (Lembre que as variáveis simbólicas a, y e x já estão definidas.)

```
sage: expressao = a*x^2+2*a*x-3*y^2
sage: expressao(a=2)
2*x^2 - 3*y^2 + 4*x
sage: expressao(x=y)
a*y^2 + 2*a*y - 3*y^2
```

Acima, definimos a variável `expressao` como a expressão $ax^2 + 2ax - 3y^2$, na segunda linha, substituímos $a = 2$ na expressão, enquanto que na terceira linha, substituímos $x = y$ na expressão. Podemos fazer os dois ao mesmo tempo.

```
sage: expressao(a=2,x=y)
-y^2 + 4*y
```

Podemos então reescrever nosso teste do início da seção do seguinte modo.

```
sage: expressao2= a^2+3*a-4
sage: expressao2(a=1)
0
sage: expressao2(a=-4)
0
```

Nossas expressões não são limitadas a polinômios, podemos ter expressões envolvendo funções logarítmicas, exponenciais e trigonométricas.

```
sage: expressao3= x^2-sin(x)*log(x)
sage: expressao3(x=2)
-log(2)*sin(2) + 4
```

Note que como antes, o Sage não substitui valores como log(2) e sen(2) por suas aproximações.

Podemos realizar algumas operações com expressões. A primeira será a de expandir uma ex-

pressão.

```
sage: expr=(x+sqrt(3))^4
sage: expr
(x + sqrt(3))^4
sage: expr.expand()
x^4 + 4*sqrt(3)*x^3 + 18*x^2 + 12*sqrt(3)*x + 9
sage: expr1= (x+y)^2*(x+2)
sage: expr1.expand()
x^3 + 2*x^2*y + x*y^2 + 2*x^2 + 4*x*y + 2*y^2
```

Note que aqui a sintaxe é um pouco diferente. Se quisermos expandir a expressão `expr` escrevemos `expr.expand()`. Esta é a sintaxe da linguagem Python, que é a linguagem de programação usada pelo Sage.

Podemos também fatorar expressões, usando o comando `factor`.

```
sage: expr2=x^4-y^4
sage: expr2.factor()
(x^2 + y^2)*(x + y)*(x - y)
sage: expr3=(x+1)^3-x^3-1
sage: expr3.factor()
3*(x + 1)*x
```

Podemos também coletar variáveis com o comando `collect`.

```
sage: expr4=expr1.expand()
sage: expr4
x^3 + 2*x^2*y + x*y^2 + 2*x^2 + 4*x*y + 2*y^2
sage: expr4.collect(x)
x^3 + 2*x^2*(y + 1) + (y^2 + 4*y)*x + 2*y^2
sage: expr4.collect(y)
x^3 + (x + 2)*y^2 + 2*x^2 + 2*(x^2 + 2*x)*y
```

Temos comandos análogos para funções envolvendo logaritmos, são eles `log_expand` e `log_simplify`.

```
sage: expr5= log(x*y^2)
sage: expr5.log_expand()
log(x) + 2*log(y)
sage: expr6= log(x)+2*log(y)
sage: expr6.log_simplify()
log(x*y^2)
```

E também para funções trigonométricas, são eles `trig_expand`, `trig_simplify` e `trig_reduce`

```
sage: expr7=cos(x)^2+sin(x)^2
sage: expr7.trig_simplify()
1
sage: expr8=sin(x+y)
sage: expr8.trig_expand()
cos(y)*sin(x) + cos(x)*sin(y)
sage: expr9=sin(x)^2
sage: expr9.trig_reduce()
-1/2*cos(2*x) + 1/2
sage: expr10=sin(x)*sin(y)+cos(x)*cos(y)
sage: expr10.trig_reduce()
cos(-x + y)
```

Mas o comando mais eficiente em simplificar expressões é o `canonicalize_radical`. Mas temos que ter cuidado com este comando, uma vez que suas simplificações podem funcionar apenas parcialmente.

```
sage: expr11= sqrt(x^2)
sage: expr11.canonicalize_radical()
x
```

No exemplo acima vimos que a expressão $\sqrt{x^2}$ é simplificada para x. O que não é verdade para todo x, apenas para x real maior igual a 0.

```
sage: expr11(x=-3)
3
```

Abaixo temos mais exemplos de como simplificar expressões.

```
sage: expr12=log(x^2+2*x*y+y^2)
sage: expr12.canonicalize_radical()
2*log(x + y)
sage: expr13=(x-1)/(x^(1/2)-1)
sage: expr13.canonicalize_radical()
sqrt(x) + 1
sage: expr15=x^(log(y)/log(x))
sage: expr15.canonicalize_radical()
y
```

5.1 Atividades

1. Verifique a identidade de Brahmagupta-Fibonacci

$$(a^2+b^2)(x^2+y^2) = (ax+by)^2 + (ay-bx)^2.$$

   Dica: Expanda a diferença das duas expressões ou calcule o valor Booleano da igualdade.

2. Verifique a identidade dos quatro quadrados de Euler.

$$\begin{aligned}(a^2+b^2+c^2+d^2)(x^2+y^2+z^2+w^2) =& (ax-by-cz-dw)^2 + (ay+bx+cw-dz)^2 + \\ &+ (az-bw+cx+dy)^2 + (aw+bz-cy+dx)^2\end{aligned}$$

   Dica: Não esqueça de definir todas as variáveis.

3. Expanda os polinômios.
   (a) $(x+y)^2$.
   (b) $(x-y)^2$.
   (c) $(x+y)^3$.
   (d) $(x+y)(x-y)$.

4. Fatore os polinômios
   (a) x^2-y^2.
   (b) x^4-14x^2+1
   (c) x^3-1
   (d) x^5-1
   (e) x^7-1
   (f) Você consegue generalizar as fatorações acima para x^n-1 para todo n.

5. Use o comando `trig_expand` para aprender as fórmulas para

   (a) $\text{sen}(x+y)$, $\text{sen}(x-y)$, $\text{sen}(2x)$ e $\text{sen}(5x)$.

   (b) $\cos(x+y)$, $\cos(x-y)$ e $\cos(2x)$.

   (c) $\tan(x+y)$.

6. Use o comando `trig_simplify` e calcule $\text{sen}(x)^6+\cos(x)^6+3*\text{sen}(x)^2*\cos(x)^2$. Você consegue provar esta igualdade?

7. Use os comandos `factor` e `log_expand` para expandir os logaritmos abaixo. (Fatore os polinômios e depois expanda o logaritmo)

   (a) $\log(x^2-y^2)$.

   (b) $\log(x^3+3x^2+3x+1)$.

8. Use o comando `canonicalize_radical` para simplificar as expressões abaixo. Em cada item diga em que casos vale a simplificação.

   (a) $\sqrt{x^2+2xy+y^2}$.

   (b) $e^{\log(x)}$.

   (c) $\frac{x^{\frac{1}{3}}-2}{x-8}$.

Constantes predefinidas	
Número π	`pi`
Número e	`e`
Unidade imaginária i	`I`
Inifnito	`Infinity` ou `oo`
Comandos para o número `x`	
Raiz quadrada	`sqrt(x)`
Logaritmo na base e	`log(x)`
Logaritmo na base a	`log(x,a)`
Exponencial na base e	`exp(x)` ou `e^x`
Seno	`sin(x)`
Cosseno	`cos(x)`
Tangente	`tan(x)`
Secante	`sec(x)`
Cossecante	`csc(x)`
Cotangente	`cot(x)`
Arco-seno	`arcsen(x)`
Arco-cosseno	`arccos(x)`
Arco-tangente	`arctan(x)`
Valor absoluto	`abs(x)`
Sinal	`sign(x)`
Aproximação numérica	`N(x)` ou `n(x)` ou `x.n()`
Comandos para a expressão simbólica `expr`	
Expandir expressão polinomial	`expr.expand()`
Fatorar expressão polinomial ou racional	`expr.factor()`
Coletar variável x em uma expressão polinomial	`expr.collect(x)`
Expandir expressão logarítmica	`expr.log_expand()`
Simplificar expressão logarítmica	`expr.log_simplify()`
Expandir expressão trigonométrica	`expr.trig_expand()`
Simplificar expressão trigonométrica	`expr.trig_simplify()`
Reduzir expressão trigonométrica	`expr.trig_reduce()`
Simplificar qualquer expressão	`expr.canonicalize_radical()`

Tabela 1: Comandos aprendidos na aula de hoje