Sage as a Calculator

By Samaneh shafi naderi

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Assignment

Sage uses $=$ for assignment. It uses $==, \leq, \geq$, $<$ and $>$ for comparison:

Example

```
sage: a = 5
sage: a
5
sage: 2 == 2
True
sage: 2 == 3
False
sage: 2 < 3
True
sage: a == 5
True
```

Basic Arithmetic

The basic arithmetic operators are $+$, -, *, and $/$ for addition,subtraction,multiplication and division, while ^ is used for exponents.

Example

```
sage: 1+1
2
sage: 103-101
2
sage: 7*9
63
sage: 7337/11
667
sage: 11/4
11/4
sage: 2^5
32
```

The following table lists the operators that are available in Sage:

Example

Operator	Function	Operator	Function
=	Assignment	==	Equality
+	Addition	>	Greater than
-	Subtraction	>=	Greater than or equal to
*	Multiplication	<	Less than
/	Division	<=	Less than or equal to
** or ^	Power	!=	Not equal to
%	Modulo (remainder)		
//	Integer quotient		

As we would expect, Sage adheres to the standard order of operations, PEMDAS (parenthesis, exponents, multiplication, division, addition, subtraction).

Example

```
sage: 2*4^2+1
33
sage: (2*4)^2+1
65
sage: 2*4^(2+1)
128
sage: -3^2
-9
sage: (-3)^2
9
```

When dividing two integers, there is a subtlety; whether Sage will return a fraction or its decimal approximation. Unlike most graphing calculators, Sage will attempt to be as precise as possible and will return the fraction unless told otherwise.

Example

```
sage: 11/4.0
2.75000000000000
sage: 11/4.
2.75000000000000
sage: 11.0/4
2.75000000000000
sage: 11/4*1.
2.75000000000000
```

Integer Division and Factoring

To calculate the quotient we use the // operator and the % operator is used for the remainder.

Example

```
sage: 2**3     #  ** means exponent
8
sage: 2^3      #  ^ is a synonym for ** (unlike in Python)
8
sage: 10 % 3   #  for integer arguments, % means mod, i.e., remainder
1
sage: 10/4
5/2
sage: 10//4    #  for integer arguments, // returns the integer quotient
2
sage: 4 * (10 // 4) + 10 % 4 == 10
True
sage: 3^2*4 + 2%5
38
```

Integer Division and Factoring

If we want both the quotient and the remainder all at once, we use the `divmod()` command

Example

```
sage: divmod(14,4)
(3, 2)
```

The integers in Sage have a built-in command (or method) which allows us to check whether one integer divides another.

Example

```
sage: 5.divides(17)
False
```

Integer Division and Factoring

A related command is the `divisors()` method. This method returns a list of all positive divisors of the integer specified.

Example

```
sage: 12.divisors()
[1, 2, 3, 4, 6, 12]
sage: 101.divisors()
[1,101]
```

When the divisors of an integer are only and itself then we say that the number is prime. To check if a number is prime in sage, we use its `is_prime()` method.

Example

```
sage: (2^19-1).is_prime()
True
sage: 153.is_prime()
False
```

Integer Division and Factoring

We use the `factor()` method to compute the *prime factorization* of an integer

Example

```
sage: 62.factor()
2 * 31
sage: 63.factor()
3^2 * 7
```

If we are interested in simply knowing which prime numbers divide an integer, we may use its `prime_divisors()` (or `prime_factors()`) method.

Example

```
sage: 24.prime_divisors()
[2, 3]
sage: 63.prime_factors()
[3, 7]
```

The greatest common divisor (gcd), not too surprisingly, is the largest of all of these common divisors. The `gcd()` command is used to calculate this divisor.

Example

```
sage: gcd(14,63)
7
sage: gcd(15,19)
1
```

Notice that if two integers share no common divisors,then their gcd will be 1.

The least common multiple is the smallest integer which both integers divide. The `lcm()` command is used to calculate the least common multiple.

Example

```
sage: lcm(4,5)
20
sage: lcm(14,21)
42
```

Standard Functions and Constants

Sage includes nearly all of the standard functions that one encounters when studying mathematics. In this section, we shall cover some of the most commonly used functions: the *maximum*, *minimum*, *floor*, *ceiling*, *trigonometric*, *exponential*, and *logarithm* functions. We will also see many of the standard mathematical constants; such as *Eulers constant(e)*, π , and the *golden ratio*(ϕ).
The *max()* and *min()* commands return the largest and smallest of a set of numbers.

Example

```
sage: max(1,5,8)
8
sage: min(1/2,1/3)
1/3
```

We may input any number of arguments into the max and min functions.

for example : `max(sqrt(v),sqrt(t))`

Standard Functions and Constants

In Sage we use the `abs()` command to compute the *absolute value* of a real number.

Example

```
sage: abs(-10)
10
sage: abs(4)
4
```

The `floor()` command rounds a number down to the nearest integer, while `ceil()` rounds up.

Example

```
sage: floor(2.1)
2
sage: ceil(2.1)
3
```

Example

```
sage: floor(1/(2.1-2))
9
```

This is clearly not correct: $\lfloor 1/(2.1-2) \rfloor = \lfloor 1/.1 \rfloor = \lfloor 10 \rfloor = 10$ .So what happend?

Example

```
sage: 1/(2.1-2)
9.99999999999999
```

Due to this, it is often a good idea to use rational numbers whenever possible instead of decimals, particularly if a high level of precision is required.

Example

```
sage: floor(1/(21/10-2))
10
```

The `sqrt()` command calculates the square root of a real number.

Example

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

To compute other roots, we use a rational exponent. Sage can compute any rational power. If either the exponent or the base is a decimal then the output will be a decimal.

Example

```
sage: 3^(1/2)
sqrt(3)
sage: (3.0)^(1/2)
1.73205080756888
sage: 8^(1/2)
2*sqrt(2)
sage: 8^(1/3)
2
```

Sage also has available all of the standard trigonometric functions: for sine and cosine we use `sin()` and `cos()`.

Example

```
sage: sin(1)
sin(1)
sage: sin(1.0)
0.841470984807897
sage: cos(3/2)
cos(3/2)
sage: cos(3/2.0)
0.0707372016677029
```

Sage has a built-in symbolic π, and understands this identity:

Example

```
sage: pi
pi
sage: sin(pi/3)
1/2*sqrt(3)
```

When we type `pi` in Sage we are dealing exactly with π , not some numerical approximation. However, we can call for a numerical approximation using the `n()` method:

Example

```
sage: pi.n()
3.14159265358979
sage: sin(pi)
0
sage: sin(pi.n())
1.22464679914735e-16
```

To get a numerical approximation, use either the function `n` or the method `n` (and both of these have a longer name, `numerical_approx`, and the function `N` is the same as `n`). These take optional arguments `prec`, which is the requested number of bits of precision, and `digits`, which is the requested number of decimal digits of precision the default is 53 bits of precision.

Example

```
sage: exp(2)
e^2
sage: n(exp(2))
7.38905609893065
sage: sqrt(pi).numerical_approx()
1.77245385090552
sage: sin(10).n(digits=5)
-0.54402
sage: N(sin(10),digits=10)
-0.5440211109
sage: numerical_approx(pi, prec=200)
3.1415926535897932384626433832795028841971693993751058209749
```

Standard Functions and Constants

Here are a few examples of using the symbolic, precise π as the numerical approximation:

Example

```
sage: sin(pi/6)
1/2
sage: sin(pi.n()/6)
0.500000000000000
sage: sin(pi/4)
1/2*sqrt(2)
sage: sin(pi.n()/4)
0.707106781186547
```

Other trigonometric functions, the inverse trigonometric functions and hyperbolic functions are also available.

Example

```
sage: arctan(1.0)
0.785398163397448
sage: sinh(9.0)
4051.54190208279
```

Similar to `pi` Sage has a built-in symbolic constant for the number e, the base of the natural logarithm.

Example

```
sage: e
e
sage: e.n()
2.71828182845905
```

While some might be familiar with using `ln(x)` for natural log and `log(x)` to represent logarithm base 10, in Sage both represent logarithm base e. We may specify a different base as a second argument to the command: to compute $log_b(x)$ in Sage we use the command `log(x,b)`.

Example

```
sage: ln(e)
1
sage: log(e)
1
sage: log(e^2)
2
sage: log(10)
log(10)
sage: log(10.0)
2.30258509299405
sage: log(100,10)
2
```

Exponentiation base e can done using both the `exp()` function and by raising the symbolic constant `e` to a specified power.

Example

```
sage: exp(2)
e^2
sage: exp(2.0)
7.38905609893065
sage: exp(log(pi))
pi
sage: e^(log(2))
2
```

Getting Help and Search

There are three ways to get help from the Sage command line. To see the documentation for a command or function, type a ? on the command line after the command. For example, to learn about the `exp` function type following:

Example

```
sage: tan?
Type:         <class 'sage.calculus.calculus.Function_tan'>
Definition:   tan( [noargspec] )
Docstring:

    The tangent function

    EXAMPLES:
        sage: tan(pi)
        0
```

Use the arrows to scroll up and down. Use the Spacebar to page down, and the b key to page up. Press q to leave the help screen and return to the command line. If you want to see the documentation and the source code for the function (if the code is available), type two question marks after the function name:
sage: `exp??`
Finally, to see the complete class documentation, use the help function:

sage: `help(exp)`

Tab completion can also make your life easier. Type the first letter (or first few letters) of a command at the prompt, and press *Tab* to see a list of possible completions. For example, type `pl` and press *Tab*:

Example

```
sage: pl
plot                    plot_step_function
plot3d                  plot_vector_field
plot_slope_field        plot_vector_field3d
```

Search

To search the documentation, type `search_doc("my query")` in an empty input cell and evaluate the cell. You can also search the source code by using `search_src ("my query")`.

The following shortcuts are useful for working with cells:

Example

Evaluate cell	With cursor in cell, hold *Shift* and press *Enter*
Insert new input cell	Move cursor between cells and click when solid bar appears
Insert new text cell	Move cursor between cells and *Shift*-click when solid bar appears
Delete cell	Delete cell contents, and then press *Backspace*
Split cell at cursor	Press *Ctrl-;*
Join two cells	Click in the lower cell and press *Ctrl-backspace*

Getting Help and Search
working with codes

The notebook interface also provides some shortcuts to make it easier to edit code in input cells:

Example

Tab completion	Start typing the name of a command, function, or object and press *Tab* to see possible completions.
Indent block of text	Highlight block and press > to indent or < to unindent. In Firefox, highlight block and press *Tab* to indent or *Shift-Tab* to unindent.
Comment a block of code	Highlight code and press *Ctrl-.*
Uncomment a block of code	Highlight code and press *Ctrl-,*
Close parenthesis	Press *Ctrl-0* to automatically insert a closing parenthesis (if needed). Press *Ctrl-0* multiple times to close multiple parentheses.