Any serious cryptography involves computations that need to be done by a machine. Let us see how to use the open-source computer algebra system **Sage** to do basic computations for us.

Sage is freely available at sagemath.org. Instead of installing it locally (it's huge!) we can conveniently use it in the cloud at cloud.sagemath.com from any browser.

Sage is built as a **Python** library, so any Python code is valid. For starters, we will use it as a fancy calculator.

**Example 50.** Let's start with some basics.

```
Sage] 17 % 12
```

$5$

```
Sage] (1 + 5) % 2  # don't forget the brackets
```

$0$

```
Sage] inverse_mod(17, 23)
```

$19$

```
Sage] xgcd(17, 23)
```

$(1, -4, 3)$

```
Sage] -4*17 + 3*23
```

$1$

```
Sage] euler_phi(84)
```

$24$

**Example 51.** Why is the following bad?

```
Sage] 3^1003 % 101
```

$27$

The reason is that this computes $3^{1003}$ first, and then reduces that huge number modulo $101$:

```
Sage] 3^1003
```

$$35695912125981779196042292013307897881066394884308000526952849942124372128361032287601\\
01447396641767302556399781555972361067577371671671062036425358196474919874574608035466\\
17047063989041820507144085408031748926871104815910218235498276622866724603402112436668\\
09387969298949770468720050187071564942882735677962417251222021721836167242754312973216\\
80102291029227131545307753863985171834477895265551139587894463150442112884933077598746\\
04125161734774642865878855686737747603770909400 27$$

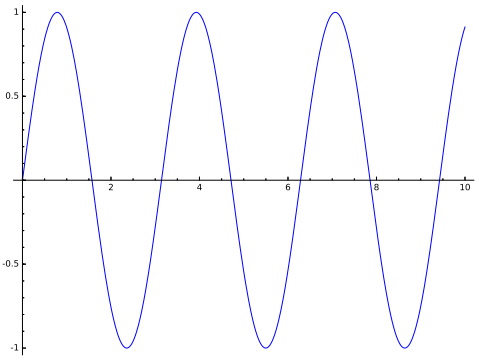
We know how to avoid computing huge intermediate numbers. Sage does the same if we instead use something like:

```
Sage] power_mod(3, 1003, 101)
```

$27$

**Example 52.** By the way, Sage is easy to use for plotting, too.

```
Sage] plot(sin(2*x), (0,10))
```




**Example 53.** We can solve equations modulo $n$. For instance:

```
Sage] solve_mod(x^2 == 1, 17)
```

$[(1), (16)]$

```
Sage]
```

Explain why the following has four solutions. [Hint: Chinese Remainder Theorem]

```
Sage] solve_mod(x^2 == 1, 21)
```

$[(1), (13), (8), (20)]$

```
Sage]
```

This even works with several variables (and is the reason why the solutions above are tuples):

```
Sage] y = var('y')

Sage] solve_mod(x^2 + y^2 == 1, 3)
```

$[(0, 1), (0, 2), (1, 0), (2, 0)]$

**Example 54. (bonus challenge!)** We can define our own functions. Here is a recursive implementation of Fibonacci numbers:

```
Sage] def fibo(n):
          if n==0: return 0
          if n==1: return 1
          return fibo(n-1)+fibo(n-2)

Sage] fibo(4)
```

$3$

```
Sage] [fibo(n) for n in [0..10]]
```

$[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]$

```
Sage] fibo(34)
```

$5702887$

The last comment takes quite a while (and `fibo(40)` will take ages). Can you explain why?

Even better, propose a solution to our issue.

**Example 55. (homework)** Check the solutions to today's quiz using Sage.