La théorie des nombres

- Théorie des nombres: [HPS14; 1.2, 1.3, 1.4, 1.5, 2.2, 2.3]
- Discrete Logarithme Problem et Diffie-Hellman: [HPS14; 2.2, 2.3, 2.6, 2.7, 2.8]

[HPS14] J. Hoffstein, J. Pipher and J. H. Silverman. *An Introduction to Mathematical Cryptography*. *Undergraduate Texts in Mathematics* (Springer, New York, NY, 2014). Accessed on Nov 18, 2024.

Exemples

Si tous les mois avaient 30 jours, est-ce qu'il y a des jours de la semaine qui ne seront jamais le premier du mois ?

Reformulation: pour tout nombre $0 \leq j < 7$, existe-t-il x et y tels que $30x = j + 7y$. Notation modulo : $30x \equiv j \pmod 7$.

Si tous les ans avaient 365 jours, est-ce qu'il y a des jours de la semaine qui ne seront jamais le 25 Décembre ? Est si tous les ans avaient 366 jours ? Et s'ils avaient 364 jours ?

Reformulation: pour tout nombre $0 \leq j < 7$, existe-t-il x et y tels que $365x = j + 7y$. Notation modulo : $365x \equiv j \pmod 7$.

Théorème de Bézout

Définition Le *Greatest Common Divisor (GCD)* de deux nombres $a \in \mathbb{Z}$ et $b \in \mathbb{Z}$, noté $\gcd(a, b)$ est le plus grand nombre $g \in \mathbb{Z}$ qui divise a (noté $g \mid a$) et b (noté $g \mid b$). C'est à dire qu'il existe $x \in \mathbb{Z}$ tel que $a = gx$ et $y \in \mathbb{Z}$ tel que $b = gy$. En notation modulaire, $a \equiv 0 \pmod g$ et $b \equiv 0 \pmod g$.

Théorème de Bézout Il existe $x, y \in \mathbb{Z}$ tels que $ax + by = c$ si et seulement si $\gcd(a, b)$ divise c. En notation modulaire $ax \equiv c \pmod b$ et $by \equiv c \pmod a$.

▶ **Comment prouver que l'égalité $ax + by = c$ implique que $\gcd(a, b)$ divise c ?**

Algorithme d'Euclide : élaboration

> **Définition** Le résultat de la *division Euclidienne* de a par un diviseur d est un quotient q et un reste $0 \leq r < d$ tels que $a = qd + r$. En notation modulaire $a \equiv r \pmod{d}$.

▶ **Observation clé Que dit le théorème de Bézout par rapport à $\gcd(a, d)$ et r.**

▶ **Observation clé Que dit le théorème de Bézout par rapport à $\gcd(d, r)$ et a.**

Lemme: Si $a \equiv r \pmod{b}$ alors $\gcd(a, b) = \gcd(b, r)$.

▶ **Observation clé Si $a > b$, trouver un mono-variant.**

▶ **Observation finale Si a et b sont positifs et qu'on effectue la substitution $(a, b) \to (b, r)$ récursivement, le mono-variant impose qu'on ne puisse itérer qu'un nombre fini de fois, que va-t-il se passer ?**

Algorithme d'Euclide : implémentation

```
pgcd (generic function with 1 method)
```

```julia
function pgcd(a, b)
    println("gcd($a, $b) = ")
    if b == 0
        println(a)
        return a
    else
        return pgcd(b, mod(a, b))
    end
end
```

gcd_a = 90284599

gcd_b = 249357461

```
1
```

```julia
pgcd(gcd_a, gcd_b)
```

```
gcd(90284599, 249357461) =
gcd(249357461, 90284599) =
gcd(90284599, 68788263) =
gcd(68788263, 21496336) =
gcd(21496336, 4299255) =
gcd(4299255, 61) =
gcd(61, 36) =
gcd(36, 25) =
gcd(25, 11) =
gcd(11, 3) =
gcd(3, 2) =
gcd(2, 1) =
gcd(1, 0) =
1
```

The complexity is difficult to evaluate but can be shown to be $O(\log(\min(a, b)))$.

Arithmétique modulaire : somme

$$a \equiv \alpha \pmod{n} \quad \text{et} \quad b \equiv \beta \pmod{n} \quad \Rightarrow \quad a + b \equiv \alpha + \beta \pmod{n}$$

a	α	b	β	n
11	1	13	3	5

```
abn_picker
```

4

```
mod(a + b, n)
```

4

```
mod(mod(a, n) + mod(b, n), n)
```

Arithmétique modulaire : produit

$$a \equiv \alpha \pmod{n} \quad \text{et} \quad b \equiv \beta \pmod{n} \quad \Rightarrow \quad ab \equiv \alpha\beta \pmod{n}$$

a	α	b	β	n
11	1	13	3	5

3

```
mod(a * b, n)
```

3

```
mod(mod(a, n) * mod(b, n), n)
```

Corollaire

$$n \mid a \quad \text{et} \quad n \mid b \quad \Rightarrow \quad n \mid (ab)$$

À ne pas confondre avec

$$a \mid n \quad \text{et} \quad b \mid n \quad \Rightarrow \quad (ab/\gcd(a,b)) \mid n$$

Division par 3 et 9

Est-ce que 2345 est divisible par 3 ou 9?

$$\begin{aligned} 2\cdot 10^3+3\cdot 10^2+4\cdot 10+5 &\equiv \ ? \pmod 9 \\ 2\cdot 1^3+3\cdot 1^2+4\cdot 1+5 &\equiv \ ? \pmod 9 \\ 2+3+4+5 &\equiv 14 \pmod 9 \end{aligned}$$

Est-ce que 2345 est divisible par 11?

$$\begin{aligned} 2\cdot (10)^3+3\cdot 10^2+4\cdot 10+5 &\equiv \ ? \pmod{11} \\ 2\cdot (-1)^3+3\cdot (-1)^2+4\cdot (-1)+5 &\equiv \ ? \pmod{11} \\ -2+3-4+5 &\equiv 2 \pmod{11} \end{aligned}$$

Inverse et division modulaire

- **Inverse modulaire** : étant donné a, n, trouver x (noté a^{-1}) tel que $xa \equiv 1 \pmod{n}$
- **Division modulaire** : étant donné a, b, n, trouver x tel que $xa \equiv b \pmod{n}$ → $x \equiv a^{-1}b \pmod{n}$.

▶ **Est-ce que l'inverse modulaire existe toujours ?**

▶ **Comment trouver l'inverse modulaire ?**

Algorithme d'Euclide étendu

$$xb + yr = g \quad \text{et} \quad r = a - qb \quad \Rightarrow \quad (x - yq)b + ya = g$$

Solution *homogène* $x = b, y = -a$ → $ba - ab = 0$. Donc si (x, y) est solution, $(x + b, y - a)$ aussi.

```
pgcdx (generic function with 1 method)
```

```julia
function pgcdx(a, b)
    if b == 0
        return a, one(a), zero(a)
    else
        q, r = divrem(a, b)
        g, x, y = pgcdx(b, r)
        return g, y, x - y * q
    end
end
```

```
(1, 89932200, -32561659)
```

```julia
gcd_g, gcd_x, gcd_y = pgcdx(gcd_a, gcd_b)
```

```
1
```

```julia
gcd_x * gcd_a + gcd_y * gcd_b
```

Pas une solution unique:

```
(1, 89932200, -32561659)
```

```julia
gcdx(gcd_a, gcd_b)
```

Inversion modulaire par Euclide étendu

Ensemble de solutions: $(x+kb, y-ka)$ pour un $k \in \mathbb{Z}$ arbitraire. Prenons k tel que $0 \leq x+kb < b$ avec `mod`.

```
modinv (generic function with 1 method)
```

```julia
function modinv(a, n)
    g, x, y = gcdx(a, n)
    return mod(x, n)
end
```

Revenons aux exemples:

$$30x \equiv j \pmod{7} \quad \Rightarrow \quad x \equiv (30)^{-1} j \pmod{7}$$

```
[0, 4, 1, 5, 2, 6, 3]
```

```julia
collect(mod.(modinv(30, 7) .* (0:6), 7))
```

$$365x \equiv j \pmod{7} \quad \Rightarrow \quad x \equiv (365)^{-1} j \pmod{7}$$

```
[0, 1, 2, 3, 4, 5, 6]
```

```julia
collect(mod.(modinv(365, 7) .* (0:6), 7))
```

$$366x \equiv j \pmod{7} \quad \Rightarrow \quad x \equiv (366)^{-1} j \pmod{7}$$

```
[0, 4, 1, 5, 2, 6, 3]
```

```julia
collect(mod.(modinv(366, 7) .* (0:6), 7))
```

S'il y avait 364 jours par ans, les fêtes seraient toujours le même jour de la semaine!

```
7
```

```julia
gcd(364, 7)
```

Fast powering

Comment calculer a^m pour un large m ?

power = 251

3618502788666131106986593281521497120414687020801267626233049500247285301248

```
@time big(2)^power
```

```
0.000012 seconds (5 allocations: 160 bytes)
```

▶ **Supposons que m est pair, c'est à dire $m = 2k$...**

▶ **Que faire si que m est impair, c'est à dire $m = 2k + 1$...**

Recursive implementation

```
fast_power (generic function with 1 method)
```

```julia
function fast_power(prod_func::Function, a, power)
    if power == 0
        return one(a)
    elseif mod(power, 2) == 1
        return prod_func(fast_power(prod_func, a, power - 1), a)
    else
        b = fast_power(prod_func, a, div(power, 2))
        return prod_func(b, b)
    end
end
```

power = 251

```
3618502788666131106986593281521497120414687020801267626233049500247285301248
```

```julia
@time big(2)^power
```

```
0.000005 seconds (5 allocations: 160 bytes)
```

```
3618502788666131106986593281521497120414687020801267626233049500247285301248
```

```julia
@time fast_power(*, big(2), power)
```

```
0.008110 seconds (3.40 k allocations: 177.469 KiB, 99.74% compilation time)
```

▶ **Quelle est la complexité temporelle ?**

Fast modular powering

```
fast_mod_power (generic function with 1 method)
```

Last 3 digit:

```
@time pow_1000 = 248
```

```
@time pow_1000 = fast_mod_power(2, power, 1000)
```

```
0.000001 seconds
```

Et modulo 999 ?

```
@time pow_999 = 500
```

```
@time pow_999 = fast_mod_power(2, power, 999)
```

```
0.000001 seconds
```

Par l'algo d'Euclide, $\gcd(n, n-1) = 1$ donc $\gcd(1000, 999) = 1$.

▶ **Comment trouver `mod(2^power, 999000)` en utilisant `pow_1000` et `pow_999` ?**

```
252248
```

```
fast_mod_power(2, power, 999000)
```

```
252248
```

```
mod(pow_1000 * 999 * modinv(999, 1000) + pow_999 * 1000 * modinv(1000, 999), 999000)
```

Chinese remainder theorem

Voir [HPS14; Section 2.8].

```
chinese_remainder_theorem (generic function with 1 method)
```

```
252248
```

```
chinese_remainder_theorem([pow_1000, pow_999], [1000, 999])
```

```
primes_upper = 100
```

```
prime_list =
[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

```
prime_list = primes(3, primes_upper)
```

```
[2, 3, 4, 2, 7, 8, 10, 6, 15, 2, 19, 39, 22, 12, 50, 14, 35, 41, 64, 37, 11, 32, 67, 11]
```

```
fast_mod_power.(2, power, prime_list)
```

```
986325341584402112586461440731025613
```

```
chinese_remainder_theorem(big.(fast_mod_power.(2, power, prime_list)), big.
(prime_list))
```

```
3618502788666131106986593281521497120414687020801267626233049500247285301248
```

```
big(2)^power
```

▶ **Comment savoir si `prime_list` contient assez de nombres pour avoir la bonne réponse ?**

Fibonacci sequence

Équation de récurrence:

$$x_{k+1} = x_k + x_{k-1}$$

Reformulation sans $(k-1)$

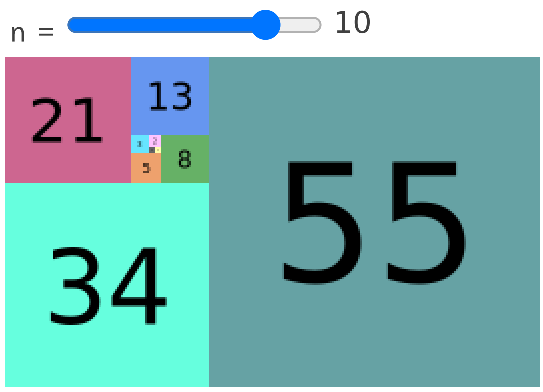
$$\begin{aligned} x_{k+1} &= x_k + y_k \\ y_{k+1} &= x_k \end{aligned}$$

Forme matricielle:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

Matrix power:

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$



Fast powering for matrices

```
fib_rec (generic function with 1 method)
```

```julia
fib_rec(n) = (n == 0 ? 0 : (n == 1 ? 1 : fib_rec(n - 1) + fib_rec(n - 2)))
```

```
267914296
```

```julia
@time fib_rec(42)
```

```
1.643827 seconds
```

```
fib_seq (generic function with 1 method)
```

```julia
function fib_seq(n)
    f = zeros(BigInt, n + 1)
    f[2] = 1
    for k in 2:n
        f[k + 1] = f[k] + f[k - 1]
    end
    return f[end]
end
```

```
25311623237323612422401550035206072917663564858024852789519298419913127817605413152301534
```

```julia
@time fib_seq(20000)
```

```
0.014594 seconds (40.01 k allocations: 17.620 MiB, 71.38% gc time)
```

```
fib_pow (generic function with 1 method)
```

```julia
function fib_pow(n)
    A = BigInt[1 1
               1 0]
    x = A^(n-1) * [1, 0]
    return x[1]
end
```

```
25311623237323612422401550035206072917663564858024852789519298419913127817605413152301534
```

```julia
@time fib_pow(20000)
```

```
0.000386 seconds (1.85 k allocations: 201.844 KiB)
```

Diagonalization to speed up powering

```
E = Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
    values:
    2-element Vector{Float64}:
     -0.6180339887498948
      1.618033988749895
    vectors:
    2×2 Matrix{Float64}:
      0.525731  -0.850651
     -0.850651  -0.525731
```

```
E = eigen([1 1; 1 0])
```

```
D = 2×2 Diagonal{BigFloat, Vector{BigFloat}}:
     -0.618034   ⋅
       ⋅        1.61803
```

```
D = Diagonal([(1 - √big(5)) / 2, (1 + √big(5)) / 2])
```

```
2×2 Matrix{Float64}:
  0.525731  -0.850651
 -0.850651  -0.525731
```

```
E.vectors
```

```
2×2 Matrix{Float64}:
 1.0   1.0
 1.0  -1.11022e-16
```

```
E.vectors * Diagonal(E.values) * inv(E.vectors)
```

```
fib_diag (generic function with 1 method)
```

```
function fib_diag(n)
    x = E.vectors * D^(n - 1) * (E.vectors \ [1, 0])
    return x[1]
end
```

```
2.531162323732361578998428490601662084769270923897910687031954402219719762339543e+4179
```

```
@time fib_diag(20000)
```

```
  1.271829 seconds (2.96 M allocations: 148.988 MiB, 2.58% gc time, 99.99% compilation time)
```

```
25311623237323612422401550035206072917663564858024852789519298419913127817605413152301534
```

```julia
@time fib_pow(20000)
```

```
0.000300 seconds (1.85 k allocations: 201.922 KiB)
```

Closed form solution

Trouver b tel que x_k est solution:

$$x_k = b^k \quad \to \quad b^{k+1} = b^k + b^{k-1} \quad \to \quad b^2 - b - 1 = 0 \quad \to \quad b = \frac{1 \pm \sqrt{5}}{2}$$

On a donc une famille de solutions:

$$x_k = a_1 \left(\frac{1-\sqrt{5}}{2} \right)^k + a_2 \left(\frac{1+\sqrt{5}}{2} \right)^k$$

Il reste à trouver a_1 et a_2 tels que $x_0 = 0$ et $x_1 = 1$. Ça correspond à calculer `E.vectors \ [1, 0]`, etc...

$$\begin{aligned} x_0 &= 0 & a_1 + a_2 &= 0 \\ x_1 &= 1 & a_1 \frac{1-\sqrt{5}}{2} + a_2 \frac{1+\sqrt{5}}{2} &= 1 \end{aligned}$$

Donc $a_1 = -1/\sqrt{5}$ et $a_2 = 1/\sqrt{5}$.

```
fib_closed (generic function with 1 method)
```

```julia
fib_closed(n) = (((1 + √big(5)) / 2)^n - ((1 - √big(5)) / 2)^n) / √big(5)
```

```
2.531162323732361242240155003520607291766356485802485278951929841991312781989138e+4179
```

```julia
@time fib_closed(20000)
```

```
0.000028 seconds (36 allocations: 1.695 KiB)
```

```
2.531162323732361578998428490601662084769270923897910687031954402219719762339543e+4179
```

```julia
@time fib_diag(20000)
```

```
0.000031 seconds (58 allocations: 2.750 KiB)
```

```
25311623237323612422401550035206072917663564858024852789519298419913127817605413152301534
```

```julia
@time fib_pow(20000)
```

```
0.000274 seconds (1.85 k allocations: 201.781 KiB)
```

Fermat’s Little Theorem

Fermat's little theorem [HPS14; Theorem 1.24]

$$\text{Si}\quad p \text{ est premier}\quad \text{et}\quad p \nmid g,\quad \text{alors}\quad g^{p-1} \equiv 1 \pmod{p}.$$

Définition g est une *racine primitive* modulo p si g^k prend toutes les valeurs $1, 2, \ldots, p-1$.

$$\text{Si}\quad p \nmid b,\quad \text{alors}\quad b^{p-1} \equiv 1 \pmod{p}$$

g = 2

p = 11

```
all_powers = [2, 4, 8, 5, 10, 9, 7, 3, 6, 1]
all_powers = fast_mod_power.(g, 1:(p-1), p)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sort(all_powers)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
unique(sort(all_powers))
```

Le nombre 2 **est** une racine primitive modulo 11

Dicrete logarithm

Étant donné un nombre premier p et une racine primitive g modulo p et un entier a tel que $p \nmid a$, le *Discrete logarithme problem* consiste à retrouver x tel que $g^x \equiv a \pmod{p}$.

```
discrete_log (generic function with 1 method)
```

```
function discrete_log(a, g, p)
    gx = one(a)
    for x = 0:(p-2)
        if a == gx
            return x
        end
        gx = mod(gx * g, p)
    end
end
```

g = 2   p = 11

Le nombre 2 **est** une racine primitive modulo 11

```
x = 8
```

```
x = discrete_log(3, g, p)
```

```
3
```

```
fast_mod_power(g, x, p)
```

▶ **Quelle est la complexité spatiale et temporelle de `discrete_log` ?**

▶ **Est-ce une complexité linéaire ou exponentielle en fonction de la taille de l'input**

Meet in the middle approach

La méthode *meet in the middle* est une méthode générique permettant de passer d'une complexité de $\mathcal{O}(N)$ à $\mathcal{O}(\sqrt{N})$.

```
[1, 2, 4, 8, 16, 15, 13, 9, 1, 2, 4, 8, 16, 15, 13, 9]
```

```
fast_mod_power.(g, 0:15, 17)
```

On peut mettre le vecteur de taille n^2 sous forme de matrice de taille $n \times n$

```
4×4 Matrix{Int64}:
 1  16  1  16
 2  15  2  15
 4  13  4  13
 8   9  8   9
```

```
reshape(fast_mod_power.(g, 0:15, 17), 4, 4)
```

On remarque que la matrice est de rang 1. Elle vaut

$$\begin{bmatrix} 1 & g^n & \cdots & g^{n^2-n} \\ g & g^{n+1} & \ddots & g^{n^2-n+1} \\ \vdots & \ddots & \ddots & \vdots \\ g^{n-1} & g^{2n-1} & \cdots & g^{n^2-1} \end{bmatrix} \equiv \begin{bmatrix} 1 \\ g \\ g^2 \\ \vdots \\ g^{n-1} \end{bmatrix} \begin{bmatrix} 1 & g^n & g^{2n} & \cdots & g^{n^2-n} \end{bmatrix} \pmod{p}$$

```
4×4 Matrix{Int64}:
 1  16  1  16
 2  15  2  15
 4  13  4  13
 8   9  8   9
```

```
mod.(fast_mod_power.(g, 0:3, 17) * fast_mod_power.(g^4, 0:3, 17)', 17)
```

On doit donc trouver la ligne i et la ligne et j tels que

$$\begin{aligned} g^{i-1}g^{(j-1)n} &\equiv a & \pmod{p} \\ g^{i-1} &\equiv a(g^{-n})^{j-1} & \pmod{p} \end{aligned}$$

Ils ne reste plus qu'à chercher une collision entre les listes de restes modulo p pour g^{i-1} et $a(g^{-n})^{j-1}$. L'identification des collision peut se faire en $\mathcal{O}(\sqrt{n}\log(n))$ avec une recherche dichotomique our en $\mathcal{O}(\sqrt{n})$ amorti avec un dictionaire.

Shanks's Babystep–Giantstep Algorithm

Voir [HPS14; Section 2.7]

giant_steps (generic function with 1 method)

```
function giant_steps(g, n, p)
    gn = fast_mod_power(g, n, p)
    return baby_steps.(modinv(gn, p), n, p)
end
```

baby_steps (generic function with 1 method)

```
function baby_steps(g, n, p)
    steps = [one(g)]
    for i in 1:n
        push!(steps, mod(steps[end] * g, p))
    end
    return steps
end
```

collision (generic function with 1 method)

```
function collision(a, b)
    d = Dict(a[i] => i for i in eachindex(a))
    for j in eachindex(b)
        if haskey(d, b[j])
            return d[b[j]], j
        end
    end
end
```

shanks_discrete_log (generic function with 1 method)

```
function shanks_discrete_log(a, g, p)
    n = isqrt(p) + 1
    i, j = collision(baby_steps(g, n, p), mod.(a .* giant_steps(g, n, p), p))
    return i - 1 + (j - 1) * n
end
```

shanks_x = 8

```
shanks_x = shanks_discrete_log(3, g, p)
```

3

```
fast_mod_power(g, shanks_x, p)
```

▶ **Quelle est la complexité?**

Diffie-Hellman

Étant donné un nombre premier p et une racine primitive g modulo p, Alice (resp. Bob) génère un nombre secret a (resp. b). Ils communique ensuite publiquement A et B.

$$A \equiv g^a \pmod{p} \qquad B \equiv g^b \pmod{p}$$

$$A' \equiv B^a \pmod{p} \qquad B' \equiv A^b \pmod{p}$$

▶ **What is the relation between A' and B' ?**

Voir [HPS14; Section 2.3]

Utils

▸ [2, 3, 5, 7]

```julia
Primes.primes(10)
```

```julia
using PlutoUI, Primes, DataFrames, Luxor, Colors, LinearAlgebra
```

```
Precompiling DataFrames...
   1769.3 ms  ✓ StringManipulation
  18430.0 ms  ✓ PrettyTables
  42220.8 ms  ✓ DataFrames
  3 dependencies successfully precompiled in 63 seconds. 30 already precompile
d.
Precompiling Luxor...
    653.3 ms  ✓ Bzip2_jll
    661.4 ms  ✓ Libgpg_error_jll
    697.6 ms  ✓ Libiconv_jll
    760.4 ms  ✓ XZ_jll
    701.8 ms  ✓ FreeType2_jll
    697.2 ms  ✓ Libgcrypt_jll
    682.0 ms  ✓ Libtiff_jll
    759.8 ms  ✓ XML2_jll
    523.5 ms  ✓ XSLT_jll
    735.0 ms  ✓ Gettext_jll
    932.1 ms  ✓ Fontconfig_jll
    664.6 ms  ✓ Glib_jll
   1241.6 ms  ✓ Xorg_libxcb_jll
    533.8 ms  ✓ Xorg_libX11_jll
    498.9 ms  ✓ Xorg_libXext_jll
    674.1 ms  ✓ Xorg_libXrender_jll
    745.6 ms  ✓ gdk_pixbuf_jll
    593.8 ms  ✓ Cairo_jll
    599.4 ms  ✓ HarfBuzz_jll
    562.3 ms  ✓ libass_jll
    608.6 ms  ✓ Pango_jll
```

```julia
import DocumenterCitations
```

```
Precompiling DocumenterCitations...
    477.2 ms  ✓ StringEncodings
    516.1 ms  ✓ Git_jll
    401.2 ms  ✓ Git
   1107.9 ms  ✓ YAML
    765.0 ms  ✓ BibParser
   1271.7 ms  ✓ Bibliography
  25804.5 ms  ✓ Documenter
   3217.5 ms  ✓ DocumenterCitations
  8 dependencies successfully precompiled in 30 seconds. 60 already precompile
d.
Precompiling ParsersExt...
    529.8 ms  ✓ InlineStrings → ParsersExt
  1 dependency successfully precompiled in 1 seconds. 9 already precompiled.
```

slider_a = 11

```julia
slider_a = @bind a Slider(1:100, default=11, show_value = true)
```

slider_b = 13

```julia
slider_b = @bind b Slider(1:100, default=13, show_value = true)
```

slider_n = 5

```julia
slider_n = @bind n Slider(1:100, default=5, show_value = true)
```

abn_picker =

a	α	b	β	n
11	1	13	3	5

gp_picker =
g = 2   p = 11

```julia
gp_picker = HAlign(
    md"`g` = $(@bind g Slider(2:(p-1), default = 2, show_value = true))",
    md"`p` = $p_picker",
)
```

power_slider = 251

```julia
power_slider = @bind power Slider(1:10000, default = 256, show_value = true)
```

draw_fib (generic function with 2 methods)

```
function draw_fib(n, size = 400)
    f = [0, 1, 1]
    for i in 3:(n+1)
        push!(f, f[end] + f[end - 1])
    end
    scale = div(size, 2maximum(f[end-1]))
    #Luxor.scale(scale)
    colors = distinguishable_colors(n)
    if iseven(n)
        Δx = f[end]
        Δy = f[end - 1]
    else
        Δx = f[end - 1]
        Δy = f[end]
    end
    left_most = sum(f[i] for i in 1:(n+1) if mod(i, 4) == 1; init = 0)
    up_most = sum(f[i] for i in 1:(n+1) if mod(i, 4) == 0; init = 0)
    shift = Point(left_most - Δx / 2, up_most - Δy / 2)
    pos(x, y) = scale * (Point(x, y) + shift)
    @draw begin
        x = 0
        y = 0
        j = 1
        for i in 2:(n+1)
            left = x
            if isodd(i)
                if iseven(div(i - 1, 2))
                    left -= f[i]
                else
                    left += f[i - 1]
                end
            end
            up = y
            if iseven(i)
                if iseven(div(i, 2))
                    up -= f[i]
                else
                    up += f[i-1]
                end
            end
            sethue(colors[i - 1])
            setopacity(0.6)
            rect(pos(left, up), scale * f[i], scale * f[i], action=:fill)
            setopacity(1)
            sethue("black")
            fontsize(div(scale * f[i], 2))
            text(string(f[i]), pos(left + f[i] / 2, up + f[i] / 2), halign = :center, valign = :middle)
```

```
            x = min(x, left)
            y = min(y, up)
        end
    end Δx * scale Δy * scale
```

fib_picker = 10

```
fib_picker = @bind fib_n Slider(1:12, default = 10, show_value = true)
```

p_picker = 11

```
p_picker = @bind p Slider(primes(20), default = 11, show_value = true)
```

qa (generic function with 2 methods)

```
include("utils.jl")
```

biblio =
▸CitationBibliography("/home/runner/work/LSINC1113/LSINC1113/Lectures/biblio.bib", AlphaSt

```
biblio = load_biblio!()
```

ⓘ Loading bibliography from `/home/runner/work/LSINC1113/LSINC1113/Lectures/biblio.bib`...

ⓧ Entry west2022Introduction is missing the publisher field(s).

ⓘ Loading completed.

cite (generic function with 1 method)

```
cite(args...) = bibcite(biblio, args...)
```

refs (generic function with 1 method)

```
refs(keys) = bibrefs(biblio, keys)
```