

JUlamono

a monospaced programming font
with reasonable Unicode support

<https://github.com/cormullion/juliamono>

JuliaMono Light

abcdefghijklmnoprstuvwxyz 12345
ABCDEFGHIJKLMNOPQRSTUVWXYZ 67890
{ } [] () <> \$ * - + = / # _ % ^ @ \ & | ~ ? ' " ` ! , . ; :

JuliaMono Regular

abcdefghijklmnoprstuvwxyz 12345
ABCDEFGHIJKLMNOPQRSTUVWXYZ 67890
{ } [] () <> \$ * - + = / # _ % ^ @ \ & | ~ ? ' " ` ! , . ; :

JuliaMono Medium

abcdefghijklmnoprstuvwxyz 12345
ABCDEFGHIJKLMNOPQRSTUVWXYZ 67890
{ } [] () <> \$ * - + = / # _ % ^ @ \ & | ~ ? ' " ` ! , . ; :

JuliaMono Bold

abcdefghijklmnoprstuvwxyz 12345
ABCDEFGHIJKLMNOPQRSTUVWXYZ 67890
{ } [] () <> \$ * - + = / # _ % ^ @ \ & | ~ ? ' " ` ! , . ; :

JuliaMono ExtraBold

abcdefghijklmnoprstuvwxyz 12345
ABCDEFGHIJKLMNOPQRSTUVWXYZ 67890
{ } [] () <> \$ * - + = / # _ % ^ @ \ & | ~ ? ' " ` ! , . ; :

JuliaMono Black

abcdefghijklmnoprstuvwxyz 12345
ABCDEFGHIJKLMNOPQRSTUVWXYZ 67890
{ } [] () <> \$ * - + = / # _ % ^ @ \ & | ~ ? ' " ` ! , . ; :

```
function _nloops(N::Int, itersym::Symbol, rangeexpr::Expr, args::Expr...)
    if rangeexpr.head !== :→
        throw(ArgumentError("second argument must be an anonymous function expression to compute the range"))
    end
    if !(1 ≤ length(args) ≤ 3)
        throw(ArgumentError("number of arguments must be 1 ≤ length(args) ≤ 3, got $args"))
    end
    body = args[end]
    ex = Expr(:escape, body)
    for dim = 1:N
        itervar = inlineanonymous(itersym, dim)
        rng = inlineanonymous(rangeexpr, dim)
        preexpr = length(args) > 1 ? inlineanonymous(args[1], dim) : :(nothing)
        postexpr = length(args) > 2 ? inlineanonymous(args[2], dim) : :(nothing)
        ex = quote
            for $(esc(itervar)) = $(esc(rng))
                $(esc(preexpr))
                $ex
                $(esc(postexpr))
            end
        end
    end
    ex
end

function _nloops(N::Int, itersym::Symbol, rangeexpr::Expr, args::Expr...)
    if rangeexpr.head !== :→
        throw(ArgumentError("second argument must be an anonymous function expression to compute the range"))
    end
    if !(1 ≤ length(args) ≤ 3)
        throw(ArgumentError("number of arguments must be 1 ≤ length(args) ≤ 3, got $args"))
    end
    body = args[end]
    ex = Expr(:escape, body)
    for dim = 1:N
        itervar = inlineanonymous(itersym, dim)
        rng = inlineanonymous(rangeexpr, dim)
        preexpr = length(args) > 1 ? inlineanonymous(args[1], dim) : :(nothing)
        postexpr = length(args) > 2 ? inlineanonymous(args[2], dim) : :(nothing)
        ex = quote
            for $(esc(itervar)) = $(esc(rng))
                $(esc(preexpr))
                $ex
                $(esc(postexpr))
            end
        end
    end
    ex
end

function _nloops(N::Int, itersym::Symbol, rangeexpr::Expr, args::Expr...)
    if rangeexpr.head !== :→
        throw(ArgumentError("second argument must be an anonymous function expression to compute the range"))
    end
    if !(1 ≤ length(args) ≤ 3)
        throw(ArgumentError("number of arguments must be 1 ≤ length(args) ≤ 3, got $args"))
    end
    body = args[end]
    ex = Expr(:escape, body)
    for dim = 1:N
        itervar = inlineanonymous(itersym, dim)
        rng = inlineanonymous(rangeexpr, dim)
        preexpr = length(args) > 1 ? inlineanonymous(args[1], dim) : :(nothing)
        postexpr = length(args) > 2 ? inlineanonymous(args[2], dim) : :(nothing)
        ex = quote
            for $(esc(itervar)) = $(esc(rng))
                $(esc(preexpr))
                $ex
                $(esc(postexpr))
            end
        end
    end
    ex
end
```


Ancient Greek Ἀδμηθ', ὁρᾶς γὰρ τάμα πράγμαθ', ώς ἔχει, λέξαι θέλω σοι πρὶν θανεῖν ἀ βούλομαι.

Bulgarian Я, пазачът Валъо уж бди, а скришом хапва кюфтенца зад щайгите.

Catalan «Dóna amor que seràs feliç!». Això, il·lus company geniüt, ja és un lliüt rētol blavís d'onze kWh.

Czech Zvlášť zákeřný učeň s dholíčky běží podél zóny úlu

Danish Quizdeltagerne spiste jordbær med fløde, mens cirkusklovnen Walther spillede på xylofon.

English Sphinx of black quartz, judge my vow.

Estonian Põdur Zagrebi tšellomängija-fölljetonist Ciqo külmetas kehvas garaažis

Finnish Charles Darwin jammaili Åken hevixylofonilla Qatarin yöpub Zeligissä.

French Voix ambiguë d'un cœur qui au zéphyr préfère les jattes de kiwi.

Georgian სწრაფი ყავისფერი მელა ახტება ზარმაც ძაღლს.

German Victor jagt zwölf Boxkämpfer quer über den großen Sylter Deich.

Greek Ταχίστη αλώπηξ βαφής ψημένη γη, δρασκελίζει υπέρ νωθρού κυνός.

Guarani H̄ilandiaigua kuñanguéra oho peteī sa'uju ypa'ūme Gavōme ombo'e hāguia ingyleñe'ē mitānguérale ne'ēndy'ŷ.

Hungarian Jó foxim és don Quijote húszwattos lámpánál ülve egy pár bűvös cipőt készít.

IPA [gʷʃ?.nas.do:ŋ.kʰlja]
[ɛjan.ŋzliwo.çiuɛn.χwa]

Icelandic Kæmi ný öxi hér, ykist þjófum nú bæði víl og ádrepa.

Irish Čuaiġ bé mórsáċ le dlúitspád fiorfinn trí hata mo dea-þorcáin big.

Latvian Muļķa hipiji mēģina brīvi nogaršot celofāna žņaudzējčūsku.

Lithuanian Įlinkdama fechtuotojo špaga sublykčiojusi pragrėžę apvalų arbūzą.

Macedonian Сидарски пејзаж: шугав билмез со чудење џвака ќофте и кељ на туѓ цех.

Norwegian Jeg begynte å fortære en sandwich mens jeg kjørte taxi på vei til quiz

Polish Pchnąć w tę łódź jeża lub ósm skrzyń fig.

Portuguese Luís argüia à Júlia que «brações, fé, chá, óxido, pôr, zângão» eram palavras do português.

Romanian Înjurând pițigăiat, zoofobul comandă vexat whisky și tequila.

Russian Широкая электрификация южных губерний даст мощный толчок подъёму сельского хозяйства.

Scottish Mus d'fhàg Cèit-Ùna ròp ì le ob.

Serbian Ајшо, лепото и чежњо, за љубав срца мога дођи у Хаџиће на кафу.

Spanish Benjamín pidió una bebida de kiwi y fresa; Noé, sin vergüenza, la más champaña del menú.

Swedish Flygande bäckasiner söka hwila på mjuka tuvor.

Turkish Pijamalı hasta yağız şoföre çabucak güvendi.

Ukrainian Чуєш іх, доцю, га? Кумедна жти, прощайся без ґольфів!

yotta	Y	10^{24}	1 000 000 000 000 000 000 000 000	septillion
zetta	Z	10^{21}	1 000 000 000 000 000 000 000 000	sextrillion
exa	E	10^{18}	1 000 000 000 000 000 000 000 000	quintillion
peta	P	10^{15}	1 000 000 000 000 000 000 000 000	quadrillion
tera	T	10^{12}	1 000 000 000 000 000	trillion
giga	G	10^9	1 000 000 000	billion
mega	M	10^6	1 000 000	million
kilo	k	10^3	1 000	thousand
hecto	h	10^2	100	hundred
deca	da	10^1	10	ten
deci	d	10^{-1}	0.1	tenth
centi	c	10^{-2}	0.01	hundredth
milli	m	10^{-3}	0.001	thousandth
micro	μ	10^{-6}	0.000 001	millionth
nano	n	10^{-9}	0.000 000 001	billionth
pico	p	10^{-12}	0.000 000 000 001	trillionth
femto	f	10^{-15}	0.000 000 000 000 001	quadrillionth
atto	a	10^{-18}	0.000 000 000 000 000 001	quintillionth
zepto	z	10^{-21}	0.000 000 000 000 000 000 001	sextillionth
yocto	y	10^{-24}	0.000 000 000 000 000 000 000 001	septillionth

classical electron radius	r_e	$2.817940285 \times 10^{-15}$
Compton wavelength of the electron	λ_c	$2.426310215 \times 10^{-12}$
reduced Compton wavelength of the electron	$\tilde{\lambda}_c$	$3.8615926764 \times 10^{-13}$
Bohr radius of the hydrogen atom	a_0	$5.291772083 \times 10^{-11}$
natural units based on the electronvolt	1 eV^{-1}	1.97×10^{-7}
reduced wavelength of hydrogen radiation	$1/R_\infty$	$9.112670505509 \times 10^{-8}$
Planck length	ℓ_p	1.616199×10^{-35}
Stoney unit of length	l_s	1.381×10^{-35}
quantum chromodynamics (QCD) unit of length	l_{qcd}	2.103×10^{-16}

	mm		inches		points	
	w	h	w	h	w	h
A0	841	1189	33.11	46.81	2384	3370
A1	594	841	23.39	33.11	1684	2384
A2	420	594	16.54	23.39	1190	1684
A3	297	420	11.69	16.54	842	1190
A4	210	297	8.27	11.69	595	842
A5	148	210	5.83	8.27	420	595
A6	105	148	4.13	5.83	298	420
A7	74	105	2.91	4.13	210	298
A8	52	74	2.05	2.91	148	210
Letter (ANSI A)	215.9	279.4	8.5	11	612	792
Legal	215.9	355.6	8.5	14	612	1008
Ledger (ANSI B)	279.4	431.8	11	17	792	1224
Tabloid (ANSI B)	431.8	279.4	17	11	1224	792
Executive	184.1	266.7	7.25	10.55	522	756
ANSI C	559	432	22	17	1584	1224
ANSI D	864	559	34	22	2448	1584
ANSI E	1118	864	44	34	3168	2448
Foolscap	336	419	13.25	16.5	954	1188
Small Post	368	469	14.5	18.5	1044	1332
Sheet and 1/3 cap	336	588	13.25	22	954	1584
Sheet and 1/2 cap	336	628	13.25	24.75	954	1782
Demy	394	507	15.5	20	1116	1440
Large Post	419	533	16.5	21	1188	1512
Small medium	444	558	17.5	22	1260	1584
Medium	457	584	18	23	1296	1656
Small Royal	482	609	19	24	1368	1728
Royal	507	634	20	25	1440	1800
Imperial	559	761	22	30	1584	2160



v cat

u+e838



Braille 2345 (t)

u+281e



blackboard S

u+1d54a



Julia dots

u+e800



editorial coronis

u+2e0e



triple integral

u+222d



alembic

u+2697



APL dot tack jot

u+234e



left arrow subset

u+297a



transversal intersect

u+2adb



weierstrass

u+2118



Pluto

u+2bd4



circled asterisk

u+229b



heavy right arrow

u+1f882



G circled

u+24bc



Mayan 14

u+1d2ee



yod

u+5d9



numero

u+2116



heavy 12pt Pinwheel Star

u+1f7d4



subscript b

u+e805



fist

u+270a



division times

u+22c7



amalgamation Or Coproduct

u+2a3f



ace of spades

u+1f0a1



Sharp S

u+1e9e



cada una

u+2106



subset with +

u+2abf



ogham nion

u+1685



segno

u+1d10b



tetragram divergence

u+1d310



white upperleft square

u+25f0



recycled paper

u+267c



Roman 50K

u+2187



eternity

u+58e



psi

u+3c8



umbrella

u+2602



triple dagger

u+2e4b



blackboard a

u+1d552



alef

u+2135



devangari 5

u+96b



swash ampersand

u+1f675



coptic 800

u+102fa



cube root

u+221b



git PR

u+e726



hacker

u+e826



zNot Rel Comp

u+2a3e



R fraktur

u+211c



speech bubble

u+1f4ac



hbar

u+210f



earth ground

u+23da



trans Pluto

u+2bd7



palm branch

u+e819



per

u+214c



hexagram completion

u+4df6



DifyQ

u+e830



smash product

u+2a33

random Selection of Unicode characters

contextual alternates

calt off	calt on
->	→
=>	⇒
>	▷
<	△
::	::

stylistic sets

zero	0	0	slashed zero
ss01	g	g	alternate g
ss02	@	@	alternate @
ss03	j	j	alternate j
ss04	ø	ø	alternate ø
ss05	*	*	lighter asterisk
ss06	a	a	simple a
ss07	`	`	smaller grave
ss08	->	→	distinct ligatures†
ss09	f	f	alternate f
ss10	r	r	alternate r
ss11	'	'	thinner grave

+ with calt off

```
45 @inline function ntuple(f::F, ::Val{N}) where {F,N}
46     N::Int
47     (N >= 0) || throw(ArgumentError(string("tuple length should be ≥ 0, got ", N)))
48     if @generated
49         quote
50             @nexprs $N i → t_i = f(i)
51             @ncall $N tuple t
52         end
53     else
54         Tuple(f(i) for i = 1:N)
55     end
56 end
57
58 @inline function fill_to_length(t::Tuple, val, ::Val{_N}) where {_N}
59     M = length(t)
60     N = _N::Int
61     M > N && throw(ArgumentError("input tuple of length $M, requested $N"))
62     if @generated
63         quote
64             (t..., $(fill(:val, (_N::Int) - length(t.parameters))...))
65         end
66     else
67         (t..., fill(val, N-M)...)
```

```
@inline function _foldoneto(op, acc, ::Val{N}) where N
    @assert N::Integer > 0
    if @generated
        quote
            acc_0 = acc
            Base.Cartesian.@nexprs $N i ->
                acc_{i} = op(acc_{i-1},
            return $(Symbol(:acc_, N)))
        end
    else
        quote
            for i in 1:N
                acc = op(acc, i)
            end
        end
    end
end
```



The title of the painting, which is known in English as *Mona Lisa*, comes from a description by Renaissance art historian Giorgio Vasari, who wrote "Leonardo undertook to paint, for Francesco del Giocondo, the portrait of *Mona Lisa*, his wife." *Mona* in Italian is a polite form of address originating as *ma donna* – similar to *Ma'am*, *Madam*, or *my lady* in English. This became *madonna*, and its contraction *mona*. The title of the painting, though traditionally spelled *Mona* (as used by Vasari), is also commonly spelled in modern Italian as *Monna Lisa* (*mona* being a vulgarity in some Italian dialects), but this is rare in English. Vasari's account of the *Mona Lisa* comes from his biography of Leonardo published in 1550, 31 years after the artist's death. It has long been the best-known source of information on the provenance of the work and identity of the sitter. Leonardo's assistant Salai, at his death in 1524, owned a portrait which in his personal papers was named *la Gioconda*, a painting bequeathed to him by Leonardo. That Leonardo painted such a work, and its date, were confirmed in 2005 when a scholar at Heidelberg University discovered a marginal note in a 1477 printing of a volume by ancient Roman philosopher Cicero. Dated October 1503, the note was written by Leonardo's contemporary Agostino Vespucci. This note likens Leonardo to renowned Greek painter Apelles, who is mentioned in the text, and states that Leonardo was at that time working on a painting of *Lisa del Giocondo*. In response to the announcement of the discovery of this document, Vincent Delieuvin, the Louvre representative, stated "Leonardo da Vinci was painting, in 1503, the portrait of a Florentine lady by the name of *Lisa del Giocondo*. About this we are now certain. Unfortunately, we cannot be absolutely certain that this portrait of *Lisa del Giocondo* is the painting of the Louvre." The model, *Lisa del Giocondo*, was a member of the Gherardini family of Florence and Tuscany, and the wife of wealthy Florentine silk merchant Francesco del Giocondo. The painting is thought to have been commissioned for their new home, and to celebrate the birth of their second son, Andrea. The Italian name for the painting, *La Gioconda*, means 'jocund' ('happy' or 'joyful') or, literally, 'the jocund one', a pun on the feminine form of *Lisa*'s married name, *Giocondo*. In French, the title *La Joconde* has the same meaning. Before that discovery, scholars had developed several alternative views as to the subject of the painting. Some argued that *Lisa del Giocondo* was the subject of a different portrait, identifying at least four other paintings as the *Mona Lisa* referred to by Vasari. Several other women have been proposed as the subject of the painting. Isabella of Aragon, Cecilia Gallerani, Costanza d'Avalos, Duchess of Francavilla, Isabella d'Este, Pacifica Brandano or Brandino, Isabella Gualanda, Caterina Sforza, Bianca Giovanna Sforza—even Salai and Leonardo himself—are all among the list of posited models portrayed in the painting. The consensus of art historians in the 21st century maintains the long-held traditional opinion that the painting depicts *Lisa del Giocondo*. History: Leonardo da Vinci had begun working on a portrait of *Lisa del Giocondo*, the model of the *Mona Lisa*, by October 1503. It is believed by some that the *Mona Lisa* was begun in 1503 or 1504 in Florence. Although the Louvre states that it was "doubtless painted between 1503 and 1506", art historian Martin Kemp says that there are some difficulties in confirming the dates with certainty. In addition, many Leonardo experts, such as Carlo Pedretti and Alessandro Vezzosi, are of the opinion that the painting is characteristic of Leonardo's style in the final years of his life, post-1513. Other academics argue that, given the historical documentation, Leonardo would have painted the work

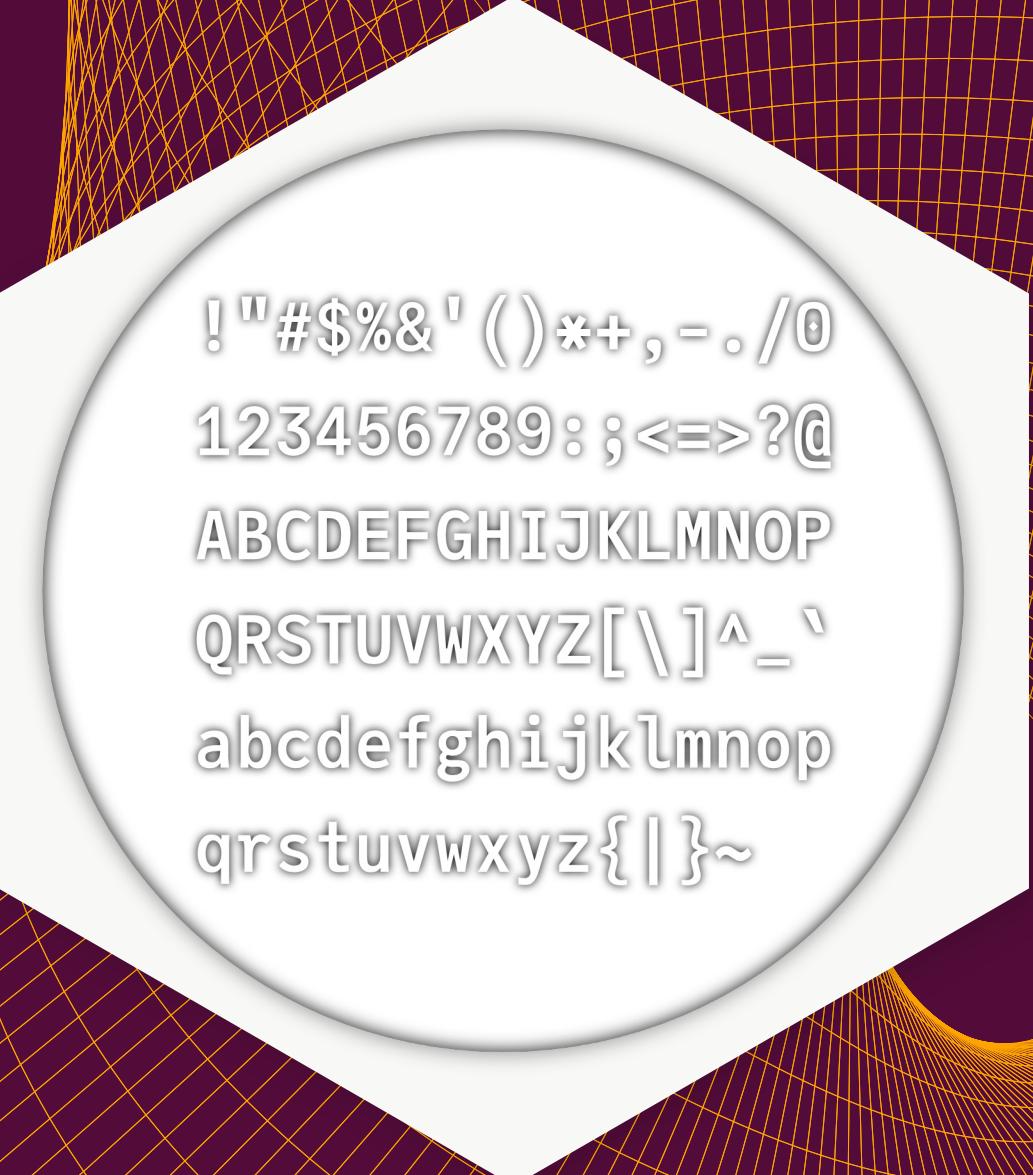
from 1513. According to Vasari, "after he had lingered over it four years, left it unfinished". In 1516, Leonardo was invited by King Francis I to work at the Clos Lucé near the Château d'Amboise; it is believed that he took the *Mona Lisa* with him and continued to work on it after he moved to France. Art historian Carmen C. Bambach has concluded that Leonardo probably continued refining the work until 1516 or 1517. Leonardo's right hand was paralytic circa 1517, which may

2004. In view of this, Vincent Delieuvin, curator of 16th-century Italian painting at the Louvre, states that the sketch and these other copies must have been inspired by another version, while Zöllner states that the sketch may be after another Leonardo portrait of the same subject. The record of an October 1517 visit by Louis d'Aragon states that the *Mona Lisa* was executed for the deceased Giuliano de' Medici, Leonardo's steward at the Belvedere Palace between 1513 and

and assistant Salai's possession until his death in 1524. The second, commissioned by Giuliano de' Medici circa 1513, would have been sold by Salai to Francis I in 1518 and is the one in the Louvre today. Others believe that there was only one true *Mona Lisa*, but are divided as to the two aforementioned fates. At some point in the 16th century, a varnish was applied to the painting. It was kept at the Palace of Fontainebleau until Louis XIV moved it to the Palace of Versailles, where it remained until the French Revolution. In 1797, it was put on permanent display at the Louvre. In the early 21st century, French scientist Pascal Cotte hypothesized a hidden portrait underneath the surface of the painting, circumstantial evidence for which was produced using reflective light technology. The underlying portrait appears to be of a model looking to the side, but lacks the flanking columns drawn by Raphael. Having been given access to the painting by the Louvre in 2004, Cotte spent ten years studying the painting with layer-amplification methods. However, the alleged portrait does not fit with historical descriptions of the painting: both Vasari and Gian Paolo Lomazzo describe the subject as smiling, unlike the subject in Cotte's portrait. Cotte admits that his reconstitution had been carried out only in support of his hypotheses and should not be considered as objective proof of an underlying portrait. Refuge, theft and vandalism After the French Revolution, the painting was moved to the Louvre, but spent a brief period in the bedroom of Napoleon (d. 1821) in the Tuileries Palace. The *Mona Lisa* was not widely known outside the art world, but in the 1860s, a portion of the French intelligentsia began to hail it as a masterpiece of Renaissance painting. During the Franco-Prussian War (1870–1871), the painting was moved from the Louvre to the Brest Arsenal. Vacant wall in the Louvre's Salon Carré after the painting was stolen in 1911 "La Joconde est Retrouvée" ("Mona Lisa is Found"), Le Petit Parisien, 13 December 1913. In 1911, the painting was still not popular among the lay-public. On 21 August 1911, the painting was stolen from the Louvre. The missing painting was first noticed the next day by painter Louis Béroud. After some confusion as to whether the painting was being photographed somewhere, the Louvre was closed for a week for investigation. French poet Guillaume Apollinaire came under suspicion and was arrested and imprisoned. Apollinaire implicated his friend Pablo Picasso, who was brought in for questioning. Both were later exonerated. The real culprit was Louvre employee Vincenzo Peruggia, who had helped construct the painting's glass case. He carried out the theft by entering the building during regular hours, hiding in a broom closet, and walking out with the painting hidden under his coat after the museum had closed. Peruggia was an Italian patriot who believed that Leonardo's painting should have been returned to an Italian museum. Peruggia may have been motivated by an associate whose copies of the original would significantly rise in value after the painting's theft. After having kept the *Mona Lisa* in his apartment for two years, Peruggia grew impatient and was caught when he attempted to sell it to Giovanni Poggi, director of the Uffizi Gallery in Florence. It was exhibited in the Uffizi Gallery for over two weeks and returned to the Louvre on 4 January 1914. Peruggia served six months in prison for the crime and was hailed for his patriotism in Italy. A year after the theft, Saturday Evening Post journalist Karl Decker met an alleged accomplice named Eduardo de Valforno, who claimed to have masterminded the theft. Forger Yves Chaudron was to have created ... [text by Wikipedia contributors. "Mona Lisa." Wikipedia, The Free Encyclopedia.]

indicate why he left the *Mona Lisa* unfinished. Circa 1505, Raphael executed a pen-and-ink sketch, in which the columns flanking the subject are more apparent. Experts universally agree that it is based on Leonardo's portrait. Other later copies of the *Mona Lisa*, such as those in the National Museum of Art, Architecture and Design and The Walters Art Museum, also display large flanking columns. As a result, it was thought that the *Mona Lisa* had been trimmed. However, by 1993, Frank Zöllner observed that the painting surface had never been trimmed; this was confirmed through a series of tests in

1516—but this was likely an error. According to Vasari, the painting was created for the model's husband, Francesco del Giocondo. A number of experts have argued that Leonardo made two versions (because of the uncertainty concerning its dating and commissioner, as well as its fate following Leonardo's death in 1519, and the difference of details in Raphael's sketch—which may be explained by the possibility that he made the sketch from memory). The hypothetical first portrait, displaying prominent columns, would have been commissioned by Giocondo circa 1503, and left unfinished in Leonardo's pupil



!"#\$%&' ()*+, -./0
123456789: ;<=>?@
ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`
abcdefghijklmnp
qrstuvwxyz{|}~

JuliaMono Light

JuliaMono Regular

JuliaMono Medium

JuliaMono Bold

JuliaMono ExtraBold

JuliaMono Black

```
n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)) :  
n == 1 ? (f(1),) :  
n == 2 ? (f(1), f(2)) :  
n == 3 ? (f(1), f(2), f(3)) :  
n == 4 ? (f(1), f(2), f(3), f(4)) :  
n == 5 ? (f(1), f(2), f(3), f(4), f(5)) :  
n == 6 ? (f(1), f(2), f(3), f(4), f(5), f(6)) :  
n == 7 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7)) :  
n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)) :  
n == 9 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9)) :  
n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)) :  
return t  
end  
function ntuple(f::F, n::Integer) where F  
    t = n == 0 ? () :  
    n == 1 ? (f(1),) :  
    n == 2 ? (f(1), f(2)) :  
    n == 3 ? (f(1), f(2), f(3)) :  
    n == 4 ? (f(1), f(2), f(3), f(4)) :  
    n == 5 ? (f(1), f(2), f(3), f(4), f(5)) :  
    n == 6 ? (f(1), f(2), f(3), f(4), f(5), f(6)) :  
    n == 7 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7)) :  
    n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)) :  
    n == 9 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9)) :  
    n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)) :  
    _ntuple(f, n)  
    return t  
end  
function _ntuple(f, n)  
    @_noinline_meta  
    (n >= 0) || throw(ArgumentError(string("tuple length should be non-negative, got ", n)))  
    if @generated  
        quote  
            @nexprs $N i → t_i = f(i)  
            @ncall $N tuple t  
        end  
    else  
        Tuple(f(i) for i = 1:$N)  
    end  
end  
# inferable ntuple (enough for bootstrapping)  
ntuple(f, ::Val{0}) = ()  
ntuple(f, ::Val{1}) = (@_inline_meta (f(1),))  
ntuple(f, ::Val{2}) = (@_inline_meta (f(1), f(2)))  
ntuple(f, ::Val{3}) = (@_inline_meta (f(1), f(2), f(3)))  
#  
@inline function ntuple(f::F, ::Val{N}) where {F,N}  
    N::Int  
    (N >= 0) || throw(ArgumentError(string("tuple length should be non-negative, got ", N)))  
    if @generated  
        quote  
            @nexprs $N i → t_i = f(i)  
            @ncall $N tuple t  
        end  
    else  
        Tuple(f(i) for i = 1:N)  
    end  
end  
@inline function fill_to_length(t::Tuple, val, ::Val{_N}) where {_N}  
    M = length(t)  
    N = _N::Int  
    if M < N  
        ntuple(f::F, ::Val{N})  
    else  
        quote  
            @nexprs $N i → t_i = val  
            @ncall $N tuple t  
        end  
    end  
end  
# inferable ntuple (enough for bootstrapping)  
ntuple(f, ::Val{0}) = ()  
ntuple(f, ::Val{1}) = (@_inline_meta (f(1),))  
ntuple(f, ::Val{2}) = (@_inline_meta (f(1), f(2)))  
#  
@inline function fill_to_length(t::Tuple, val, ::Val{_N}) where {_N}  
    M = length(t)  
    N = _N::Int  
    if M < N  
        ntuple(f::F, ::Val{N})  
    else  
        quote  
            @nexprs $N i → t_i = val  
            @ncall $N tuple t  
        end  
    end  
end  
# inferable ntuple (enough for bootstrapping)  
ntuple(f, ::Val{0}) = ()  
ntuple(f, ::Val{1}) = (@_inline_meta (f(1),))  
ntuple(f, ::Val{2}) = (@_inline_meta (f(1), f(2)))  
#  
@inline function fill_to_length(t::Tuple, val, ::Val{_N}) where {_N}  
    M = length(t)  
    N = _N::Int  
    if M < N  
        ntuple(f::F, ::Val{N})  
    else  
        quote  
            @nexprs $N i → t_i = val  
            @ncall $N tuple t  
        end  
    end  
end
```

```
+ 15
+ 16 `````
+ 17 """
+ 18 function ntuple(f::F, n::Integer) where F
+ 19     t = n == 0 ? () :
+ 20     n == 1 ? (f(1),) :
+ 21     n == 2 ? (f(1), f(2)) :
+ 22     n == 3 ? (f(1), f(2), f(3)) :
+ 23     n == 4 ? (f(1), f(2), f(3), f(4)) :
+ 24     n == 5 ? (f(1), f(2), f(3), f(4), f(5)) :
+ 25     n == 6 ? (f(1), f(2), f(3), f(4), f(5), f(6)) :
+ 26     n == 7 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7)) :
+ 27
+ 28     n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7),
+ 29     n == 9 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7),
+ 30     n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7),
+ 31     _ntuple(f, n)
+ 32     return t
+ 33 end
+ 34
+ 35 function _ntuple(f, n)
+ 36     @_noinline_meta
+ 37     (n >= 0) || throw(ArgumentError(string("tuple length sh
+ 38 ([f(i) for i = 1:n]...),
+ 39 end
+ 40
+ 41 # inferrable ntuple (enough for bootstrapping)
+ 42 ntuple(f, ::Val{0}) = ()
+ 43 ntuple(f, ::Val{1}) = (@_inline_meta; (f(1),))
+ 44 ntuple(f, ::Val{2}) = (@_inline_meta; (f(1), f(2)))
+ 45 ntuple(f, ::Val{3}) = (@_inline_meta; (f(1), f(2), f(3)))
+ 46 #
+ 47 @inline function ntuple(f::F, ::Val{N}) where {F,N}
+ 48     N::Int
+ 49     (N >= 0) || throw(ArgumentError(string("tuple length sh
+ 50     if @_generated
+ 51         quote
+ 52             @exprs $N i → t_i = f(i)
+ 53             @call $N tuple t
+ 54         end
+ 55     else
+ 56         Tuple(f(i) for i = 1:N)
+ 57     end
+ 58 end
+ 59
+ 60 @inline function fill_to_length(t::Tuple, val, ::Val{-N}) where {-N}
+ 61     M = length(t)
+ 62     N = -N::Int
+ 63     for i = M+1:N
+ 64         t = tuple(t..., val)
+ 65     end
+ 66     t
+ 67 end
+ 68
+ 69 @inline function fill!(t::AbstractVector, val, ::Val{-N}) where {-N}
+ 70     M = length(t)
+ 71     N = -N::Int
+ 72     for i = M+1:N
+ 73         t[i] = val
+ 74     end
+ 75     t
+ 76 end
+ 77
+ 78 @inline function fill!(t::AbstractMatrix, val, ::Val{-N,M}) where {-N,M}
+ 79     M = size(t, 1)
+ 80     N = -N::Int
+ 81     for i = 1:M
+ 82         fill!(t[i], val)
+ 83     end
+ 84     t
+ 85 end
+ 86
+ 87 @inline function fill!(t::AbstractVector{,}, val, ::Val{-N}) where {T,N}
+ 88     M = length(t)
+ 89     N = -N::Int
+ 90     for i = 1:M
+ 91         t[i] = val
+ 92     end
+ 93     t
+ 94 end
+ 95
+ 96 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M}) where {T,N,M}
+ 97     M = size(t, 1)
+ 98     N = -N::Int
+ 99     for i = 1:M
+ 100        fill!(t[i], val)
+ 101    end
+ 102    t
+ 103 end
+ 104
+ 105 @inline function fill!(t::AbstractVector{,}, val, ::Val{-N,M}) where {T,N,M}
+ 106     M = size(t, 2)
+ 107     N = -N::Int
+ 108     for j = 1:M
+ 109         fill!(t[:,j], val)
+ 110     end
+ 111     t
+ 112 end
+ 113
+ 114 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L}) where {T,N,M,L}
+ 115     M = size(t, 1)
+ 116     N = -N::Int
+ 117     L = -L::Int
+ 118     for i = 1:M
+ 119         fill!(t[i], val)
+ 120     end
+ 121     t
+ 122 end
+ 123
+ 124 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L}) where {T,N,M,L}
+ 125     M = size(t, 2)
+ 126     N = -N::Int
+ 127     L = -L::Int
+ 128     for j = 1:L
+ 129         fill!(t[:,j], val)
+ 130     end
+ 131     t
+ 132 end
+ 133
+ 134 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K}) where {T,N,M,L,K}
+ 135     M = size(t, 1)
+ 136     N = -N::Int
+ 137     L = -L::Int
+ 138     K = -K::Int
+ 139     for i = 1:M
+ 140         fill!(t[i], val)
+ 141     end
+ 142     t
+ 143 end
+ 144
+ 145 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K}) where {T,N,M,L,K}
+ 146     M = size(t, 2)
+ 147     N = -N::Int
+ 148     L = -L::Int
+ 149     K = -K::Int
+ 150     for j = 1:L
+ 151         fill!(t[:,j], val)
+ 152     end
+ 153     t
+ 154 end
+ 155
+ 156 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O}) where {T,N,M,L,K,O}
+ 157     M = size(t, 1)
+ 158     N = -N::Int
+ 159     L = -L::Int
+ 160     K = -K::Int
+ 161     O = -O::Int
+ 162     for i = 1:M
+ 163         fill!(t[i], val)
+ 164     end
+ 165     t
+ 166 end
+ 167
+ 168 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O}) where {T,N,M,L,K,O}
+ 169     M = size(t, 2)
+ 170     N = -N::Int
+ 171     L = -L::Int
+ 172     K = -K::Int
+ 173     O = -O::Int
+ 174     for j = 1:L
+ 175         fill!(t[:,j], val)
+ 176     end
+ 177     t
+ 178 end
+ 179
+ 180 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P}) where {T,N,M,L,K,O,P}
+ 181     M = size(t, 1)
+ 182     N = -N::Int
+ 183     L = -L::Int
+ 184     K = -K::Int
+ 185     O = -O::Int
+ 186     P = -P::Int
+ 187     for i = 1:M
+ 188         fill!(t[i], val)
+ 189     end
+ 190     t
+ 191 end
+ 192
+ 193 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P}) where {T,N,M,L,K,O,P}
+ 194     M = size(t, 2)
+ 195     N = -N::Int
+ 196     L = -L::Int
+ 197     K = -K::Int
+ 198     O = -O::Int
+ 199     P = -P::Int
+ 200     for j = 1:L
+ 201         fill!(t[:,j], val)
+ 202     end
+ 203     t
+ 204 end
+ 205
+ 206 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q}) where {T,N,M,L,K,O,P,Q}
+ 207     M = size(t, 1)
+ 208     N = -N::Int
+ 209     L = -L::Int
+ 210     K = -K::Int
+ 211     O = -O::Int
+ 212     P = -P::Int
+ 213     Q = -Q::Int
+ 214     for i = 1:M
+ 215         fill!(t[i], val)
+ 216     end
+ 217     t
+ 218 end
+ 219
+ 220 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q}) where {T,N,M,L,K,O,P,Q}
+ 221     M = size(t, 2)
+ 222     N = -N::Int
+ 223     L = -L::Int
+ 224     K = -K::Int
+ 225     O = -O::Int
+ 226     P = -P::Int
+ 227     Q = -Q::Int
+ 228     for j = 1:L
+ 229         fill!(t[:,j], val)
+ 230     end
+ 231     t
+ 232 end
+ 233
+ 234 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R}) where {T,N,M,L,K,O,P,Q,R}
+ 235     M = size(t, 1)
+ 236     N = -N::Int
+ 237     L = -L::Int
+ 238     K = -K::Int
+ 239     O = -O::Int
+ 240     P = -P::Int
+ 241     Q = -Q::Int
+ 242     R = -R::Int
+ 243     for i = 1:M
+ 244         fill!(t[i], val)
+ 245     end
+ 246     t
+ 247 end
+ 248
+ 249 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R}) where {T,N,M,L,K,O,P,Q,R}
+ 250     M = size(t, 2)
+ 251     N = -N::Int
+ 252     L = -L::Int
+ 253     K = -K::Int
+ 254     O = -O::Int
+ 255     P = -P::Int
+ 256     Q = -Q::Int
+ 257     R = -R::Int
+ 258     for j = 1:L
+ 259         fill!(t[:,j], val)
+ 260     end
+ 261     t
+ 262 end
+ 263
+ 264 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S}) where {T,N,M,L,K,O,P,Q,R,S}
+ 265     M = size(t, 1)
+ 266     N = -N::Int
+ 267     L = -L::Int
+ 268     K = -K::Int
+ 269     O = -O::Int
+ 270     P = -P::Int
+ 271     Q = -Q::Int
+ 272     R = -R::Int
+ 273     S = -S::Int
+ 274     for i = 1:M
+ 275         fill!(t[i], val)
+ 276     end
+ 277     t
+ 278 end
+ 279
+ 280 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S}) where {T,N,M,L,K,O,P,Q,R,S}
+ 281     M = size(t, 2)
+ 282     N = -N::Int
+ 283     L = -L::Int
+ 284     K = -K::Int
+ 285     O = -O::Int
+ 286     P = -P::Int
+ 287     Q = -Q::Int
+ 288     R = -R::Int
+ 289     S = -S::Int
+ 290     for j = 1:L
+ 291         fill!(t[:,j], val)
+ 292     end
+ 293     t
+ 294 end
+ 295
+ 296 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T}) where {T,N,M,L,K,O,P,Q,R,S,T}
+ 297     M = size(t, 1)
+ 298     N = -N::Int
+ 299     L = -L::Int
+ 300     K = -K::Int
+ 301     O = -O::Int
+ 302     P = -P::Int
+ 303     Q = -Q::Int
+ 304     R = -R::Int
+ 305     S = -S::Int
+ 306     T = -T::Int
+ 307     for i = 1:M
+ 308         fill!(t[i], val)
+ 309     end
+ 310     t
+ 311 end
+ 312
+ 313 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T}) where {T,N,M,L,K,O,P,Q,R,S,T}
+ 314     M = size(t, 2)
+ 315     N = -N::Int
+ 316     L = -L::Int
+ 317     K = -K::Int
+ 318     O = -O::Int
+ 319     P = -P::Int
+ 320     Q = -Q::Int
+ 321     R = -R::Int
+ 322     S = -S::Int
+ 323     T = -T::Int
+ 324     for j = 1:L
+ 325         fill!(t[:,j], val)
+ 326     end
+ 327     t
+ 328 end
+ 329
+ 330 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U}) where {T,N,M,L,K,O,P,Q,R,S,T,U}
+ 331     M = size(t, 1)
+ 332     N = -N::Int
+ 333     L = -L::Int
+ 334     K = -K::Int
+ 335     O = -O::Int
+ 336     P = -P::Int
+ 337     Q = -Q::Int
+ 338     R = -R::Int
+ 339     S = -S::Int
+ 340     T = -T::Int
+ 341     U = -U::Int
+ 342     for i = 1:M
+ 343         fill!(t[i], val)
+ 344     end
+ 345     t
+ 346 end
+ 347
+ 348 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U}) where {T,N,M,L,K,O,P,Q,R,S,T,U}
+ 349     M = size(t, 2)
+ 350     N = -N::Int
+ 351     L = -L::Int
+ 352     K = -K::Int
+ 353     O = -O::Int
+ 354     P = -P::Int
+ 355     Q = -Q::Int
+ 356     R = -R::Int
+ 357     S = -S::Int
+ 358     T = -T::Int
+ 359     U = -U::Int
+ 360     for j = 1:L
+ 361         fill!(t[:,j], val)
+ 362     end
+ 363     t
+ 364 end
+ 365
+ 366 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V}
+ 367     M = size(t, 1)
+ 368     N = -N::Int
+ 369     L = -L::Int
+ 370     K = -K::Int
+ 371     O = -O::Int
+ 372     P = -P::Int
+ 373     Q = -Q::Int
+ 374     R = -R::Int
+ 375     S = -S::Int
+ 376     T = -T::Int
+ 377     U = -U::Int
+ 378     V = -V::Int
+ 379     for i = 1:M
+ 380         fill!(t[i], val)
+ 381     end
+ 382     t
+ 383 end
+ 384
+ 385 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V}
+ 386     M = size(t, 2)
+ 387     N = -N::Int
+ 388     L = -L::Int
+ 389     K = -K::Int
+ 390     O = -O::Int
+ 391     P = -P::Int
+ 392     Q = -Q::Int
+ 393     R = -R::Int
+ 394     S = -S::Int
+ 395     T = -T::Int
+ 396     U = -U::Int
+ 397     V = -V::Int
+ 398     for j = 1:L
+ 399         fill!(t[:,j], val)
+ 400     end
+ 401     t
+ 402 end
+ 403
+ 404 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W}
+ 405     M = size(t, 1)
+ 406     N = -N::Int
+ 407     L = -L::Int
+ 408     K = -K::Int
+ 409     O = -O::Int
+ 410     P = -P::Int
+ 411     Q = -Q::Int
+ 412     R = -R::Int
+ 413     S = -S::Int
+ 414     T = -T::Int
+ 415     U = -U::Int
+ 416     V = -V::Int
+ 417     W = -W::Int
+ 418     for i = 1:M
+ 419         fill!(t[i], val)
+ 420     end
+ 421     t
+ 422 end
+ 423
+ 424 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W}
+ 425     M = size(t, 2)
+ 426     N = -N::Int
+ 427     L = -L::Int
+ 428     K = -K::Int
+ 429     O = -O::Int
+ 430     P = -P::Int
+ 431     Q = -Q::Int
+ 432     R = -R::Int
+ 433     S = -S::Int
+ 434     T = -T::Int
+ 435     U = -U::Int
+ 436     V = -V::Int
+ 437     W = -W::Int
+ 438     for j = 1:L
+ 439         fill!(t[:,j], val)
+ 440     end
+ 441     t
+ 442 end
+ 443
+ 444 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X}
+ 445     M = size(t, 1)
+ 446     N = -N::Int
+ 447     L = -L::Int
+ 448     K = -K::Int
+ 449     O = -O::Int
+ 450     P = -P::Int
+ 451     Q = -Q::Int
+ 452     R = -R::Int
+ 453     S = -S::Int
+ 454     T = -T::Int
+ 455     U = -U::Int
+ 456     V = -V::Int
+ 457     W = -W::Int
+ 458     X = -X::Int
+ 459     for i = 1:M
+ 460         fill!(t[i], val)
+ 461     end
+ 462     t
+ 463 end
+ 464
+ 465 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X}
+ 466     M = size(t, 2)
+ 467     N = -N::Int
+ 468     L = -L::Int
+ 469     K = -K::Int
+ 470     O = -O::Int
+ 471     P = -P::Int
+ 472     Q = -Q::Int
+ 473     R = -R::Int
+ 474     S = -S::Int
+ 475     T = -T::Int
+ 476     U = -U::Int
+ 477     V = -V::Int
+ 478     W = -W::Int
+ 479     X = -X::Int
+ 480     for j = 1:L
+ 481         fill!(t[:,j], val)
+ 482     end
+ 483     t
+ 484 end
+ 485
+ 486 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y}
+ 487     M = size(t, 1)
+ 488     N = -N::Int
+ 489     L = -L::Int
+ 490     K = -K::Int
+ 491     O = -O::Int
+ 492     P = -P::Int
+ 493     Q = -Q::Int
+ 494     R = -R::Int
+ 495     S = -S::Int
+ 496     T = -T::Int
+ 497     U = -U::Int
+ 498     V = -V::Int
+ 499     W = -W::Int
+ 500     X = -X::Int
+ 501     Y = -Y::Int
+ 502     for i = 1:M
+ 503         fill!(t[i], val)
+ 504     end
+ 505     t
+ 506 end
+ 507
+ 508 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y}
+ 509     M = size(t, 2)
+ 510     N = -N::Int
+ 511     L = -L::Int
+ 512     K = -K::Int
+ 513     O = -O::Int
+ 514     P = -P::Int
+ 515     Q = -Q::Int
+ 516     R = -R::Int
+ 517     S = -S::Int
+ 518     T = -T::Int
+ 519     U = -U::Int
+ 520     V = -V::Int
+ 521     W = -W::Int
+ 522     X = -X::Int
+ 523     Y = -Y::Int
+ 524     for j = 1:L
+ 525         fill!(t[:,j], val)
+ 526     end
+ 527     t
+ 528 end
+ 529
+ 530 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z}
+ 531     M = size(t, 1)
+ 532     N = -N::Int
+ 533     L = -L::Int
+ 534     K = -K::Int
+ 535     O = -O::Int
+ 536     P = -P::Int
+ 537     Q = -Q::Int
+ 538     R = -R::Int
+ 539     S = -S::Int
+ 540     T = -T::Int
+ 541     U = -U::Int
+ 542     V = -V::Int
+ 543     W = -W::Int
+ 544     X = -X::Int
+ 545     Y = -Y::Int
+ 546     Z = -Z::Int
+ 547     for i = 1:M
+ 548         fill!(t[i], val)
+ 549     end
+ 550     t
+ 551 end
+ 552
+ 553 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z}
+ 554     M = size(t, 2)
+ 555     N = -N::Int
+ 556     L = -L::Int
+ 557     K = -K::Int
+ 558     O = -O::Int
+ 559     P = -P::Int
+ 560     Q = -Q::Int
+ 561     R = -R::Int
+ 562     S = -S::Int
+ 563     T = -T::Int
+ 564     U = -U::Int
+ 565     V = -V::Int
+ 566     W = -W::Int
+ 567     X = -X::Int
+ 568     Y = -Y::Int
+ 569     Z = -Z::Int
+ 570     for j = 1:L
+ 571         fill!(t[:,j], val)
+ 572     end
+ 573     t
+ 574 end
+ 575
+ 576 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A}
+ 577     M = size(t, 1)
+ 578     N = -N::Int
+ 579     L = -L::Int
+ 580     K = -K::Int
+ 581     O = -O::Int
+ 582     P = -P::Int
+ 583     Q = -Q::Int
+ 584     R = -R::Int
+ 585     S = -S::Int
+ 586     T = -T::Int
+ 587     U = -U::Int
+ 588     V = -V::Int
+ 589     W = -W::Int
+ 590     X = -X::Int
+ 591     Y = -Y::Int
+ 592     Z = -Z::Int
+ 593     A = -A::Int
+ 594     for i = 1:M
+ 595         fill!(t[i], val)
+ 596     end
+ 597     t
+ 598 end
+ 599
+ 600 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A}
+ 601     M = size(t, 2)
+ 602     N = -N::Int
+ 603     L = -L::Int
+ 604     K = -K::Int
+ 605     O = -O::Int
+ 606     P = -P::Int
+ 607     Q = -Q::Int
+ 608     R = -R::Int
+ 609     S = -S::Int
+ 610     T = -T::Int
+ 611     U = -U::Int
+ 612     V = -V::Int
+ 613     W = -W::Int
+ 614     X = -X::Int
+ 615     Y = -Y::Int
+ 616     Z = -Z::Int
+ 617     A = -A::Int
+ 618     for j = 1:L
+ 619         fill!(t[:,j], val)
+ 620     end
+ 621     t
+ 622 end
+ 623
+ 624 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B}
+ 625     M = size(t, 1)
+ 626     N = -N::Int
+ 627     L = -L::Int
+ 628     K = -K::Int
+ 629     O = -O::Int
+ 630     P = -P::Int
+ 631     Q = -Q::Int
+ 632     R = -R::Int
+ 633     S = -S::Int
+ 634     T = -T::Int
+ 635     U = -U::Int
+ 636     V = -V::Int
+ 637     W = -W::Int
+ 638     X = -X::Int
+ 639     Y = -Y::Int
+ 640     Z = -Z::Int
+ 641     A = -A::Int
+ 642     B = -B::Int
+ 643     for i = 1:M
+ 644         fill!(t[i], val)
+ 645     end
+ 646     t
+ 647 end
+ 648
+ 649 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B}
+ 650     M = size(t, 2)
+ 651     N = -N::Int
+ 652     L = -L::Int
+ 653     K = -K::Int
+ 654     O = -O::Int
+ 655     P = -P::Int
+ 656     Q = -Q::Int
+ 657     R = -R::Int
+ 658     S = -S::Int
+ 659     T = -T::Int
+ 660     U = -U::Int
+ 661     V = -V::Int
+ 662     W = -W::Int
+ 663     X = -X::Int
+ 664     Y = -Y::Int
+ 665     Z = -Z::Int
+ 666     A = -A::Int
+ 667     B = -B::Int
+ 668     for j = 1:L
+ 669         fill!(t[:,j], val)
+ 670     end
+ 671     t
+ 672 end
+ 673
+ 674 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C}
+ 675     M = size(t, 1)
+ 676     N = -N::Int
+ 677     L = -L::Int
+ 678     K = -K::Int
+ 679     O = -O::Int
+ 680     P = -P::Int
+ 681     Q = -Q::Int
+ 682     R = -R::Int
+ 683     S = -S::Int
+ 684     T = -T::Int
+ 685     U = -U::Int
+ 686     V = -V::Int
+ 687     W = -W::Int
+ 688     X = -X::Int
+ 689     Y = -Y::Int
+ 690     Z = -Z::Int
+ 691     A = -A::Int
+ 692     B = -B::Int
+ 693     C = -C::Int
+ 694     for i = 1:M
+ 695         fill!(t[i], val)
+ 696     end
+ 697     t
+ 698 end
+ 699
+ 700 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C}
+ 701     M = size(t, 2)
+ 702     N = -N::Int
+ 703     L = -L::Int
+ 704     K = -K::Int
+ 705     O = -O::Int
+ 706     P = -P::Int
+ 707     Q = -Q::Int
+ 708     R = -R::Int
+ 709     S = -S::Int
+ 710     T = -T::Int
+ 711     U = -U::Int
+ 712     V = -V::Int
+ 713     W = -W::Int
+ 714     X = -X::Int
+ 715     Y = -Y::Int
+ 716     Z = -Z::Int
+ 717     A = -A::Int
+ 718     B = -B::Int
+ 719     C = -C::Int
+ 720     for j = 1:L
+ 721         fill!(t[:,j], val)
+ 722     end
+ 723     t
+ 724 end
+ 725
+ 726 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D}
+ 727     M = size(t, 1)
+ 728     N = -N::Int
+ 729     L = -L::Int
+ 730     K = -K::Int
+ 731     O = -O::Int
+ 732     P = -P::Int
+ 733     Q = -Q::Int
+ 734     R = -R::Int
+ 735     S = -S::Int
+ 736     T = -T::Int
+ 737     U = -U::Int
+ 738     V = -V::Int
+ 739     W = -W::Int
+ 740     X = -X::Int
+ 741     Y = -Y::Int
+ 742     Z = -Z::Int
+ 743     A = -A::Int
+ 744     B = -B::Int
+ 745     C = -C::Int
+ 746     D = -D::Int
+ 747     for i = 1:M
+ 748         fill!(t[i], val)
+ 749     end
+ 750     t
+ 751 end
+ 752
+ 753 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D}
+ 754     M = size(t, 2)
+ 755     N = -N::Int
+ 756     L = -L::Int
+ 757     K = -K::Int
+ 758     O = -O::Int
+ 759     P = -P::Int
+ 760     Q = -Q::Int
+ 761     R = -R::Int
+ 762     S = -S::Int
+ 763     T = -T::Int
+ 764     U = -U::Int
+ 765     V = -V::Int
+ 766     W = -W::Int
+ 767     X = -X::Int
+ 768     Y = -Y::Int
+ 769     Z = -Z::Int
+ 770     A = -A::Int
+ 771     B = -B::Int
+ 772     C = -C::Int
+ 773     D = -D::Int
+ 774     for j = 1:L
+ 775         fill!(t[:,j], val)
+ 776     end
+ 777     t
+ 778 end
+ 779
+ 780 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D,E}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D,E}
+ 781     M = size(t, 1)
+ 782     N = -N::Int
+ 783     L = -L::Int
+ 784     K = -K::Int
+ 785     O = -O::Int
+ 786     P = -P::Int
+ 787     Q = -Q::Int
+ 788     R = -R::Int
+ 789     S = -S::Int
+ 790     T = -T::Int
+ 791     U = -U::Int
+ 792     V = -V::Int
+ 793     W = -W::Int
+ 794     X = -X::Int
+ 795     Y = -Y::Int
+ 796     Z = -Z::Int
+ 797     A = -A::Int
+ 798     B = -B::Int
+ 799     C = -C::Int
+ 800     D = -D::Int
+ 801     E = -E::Int
+ 802     for i = 1:M
+ 803         fill!(t[i], val)
+ 804     end
+ 805     t
+ 806 end
+ 807
+ 808 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D,E}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D,E}
+ 809     M = size(t, 2)
+ 810     N = -N::Int
+ 811     L = -L::Int
+ 812     K = -K::Int
+ 813     O = -O::Int
+ 814     P = -P::Int
+ 815     Q = -Q::Int
+ 816     R = -R::Int
+ 817     S = -S::Int
+ 818     T = -T::Int
+ 819     U = -U::Int
+ 820     V = -V::Int
+ 821     W = -W::Int
+ 822     X = -X::Int
+ 823     Y = -Y::Int
+ 824     Z = -Z::Int
+ 825     A = -A::Int
+ 826     B = -B::Int
+ 827     C = -C::Int
+ 828     D = -D::Int
+ 829     E = -E::Int
+ 830     for j = 1:L
+ 831         fill!(t[:,j], val)
+ 832     end
+ 833     t
+ 834 end
+ 835
+ 836 @inline function fill!(t::AbstractMatrix{,}, val, ::Val{-N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D,E,F}) where {T,N,M,L,K,O,P,Q,R,S,T,U,V,W,X,Y,Z,A,B,C,D,E,F}
+ 837     M = size(t, 1)
+ 838     N = -N::Int
+ 839     L = -L::Int
+ 840     K = -K::Int
+ 841     O = -O::Int
+ 842     P = -P::Int
+ 843     Q = -Q::Int
+ 844     R = -R::Int
+ 845     S = -S::Int
+ 846     T = -T::Int
+ 847     U = -U::Int
+ 848     V = -V::Int
+ 849     W = -W::Int
+ 850     X = -X::Int
+ 851     Y = -Y::Int
+ 852     Z = -Z::Int
+ 853     A = -A::Int
+ 854     B = -B::Int
+ 855     C = -C::Int
+ 856     D = -
```

length should be

```

1      n == 1 ? (f(1),) :
2      n == 2 ? (f(1), f(2)) :
3      n == 3 ? (f(1), f(2), f(3)) :
4      n == 4 ? (f(1), f(2), f(3), f(4)) :
5      n == 5 ? (f(1), f(2), f(3), f(4), f(5)) :
6      n == 6 ? (f(1), f(2), f(3), f(4), f(5), f(6)) :
7      n == 7 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7)) :
8      n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)) :
9      n == 9 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9)) :
10     n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)) :
11     _ntuple(f, n)
12
13     return t
14
15
16     function _ntuple(f, n)
17       @_noinline_meta
18       (n >= 0) || throw(ArgumentError(string("tuple length should be ≥
19           ${f(i)} for i = 1:n")))
20   end
21
22   41 # inferable ntuple (enough for bootstrapping)
23   42 ntuple(f, ::Val{0}) = {}
24   43 ntuple(f, ::Val{1}) = (@_inline_meta; (f(1),))
25   44 ntuple(f, ::Val{2}) = (@_inline_meta; (f(1), f(2)))
26   45 ntuple(f, ::Val{3}) = (@_inline_meta; (f(1), f(2), f(3)))
27   46 #
28   47 @inline function ntuple(f::F, ::Val{N}) where {
29     N::Int
30     (N >= 0) || throw(ArgumentError(string(
31       "N must be ≥ 0")))
32     if @_generated
33       quote
34         @nexprs $N i → t_i = f(i)
35         @call $N tuple t
36       end
37     else
38       Tuple(f(i) for i = 1:N)
39     end
40   end
41
42   60 @inline function fil
43     M = length(t)
44     N = @_n:Int
45
46     NORMAL +2 -1 -0

```

```

# inferable ntuple (enough for bootstrapping)
ntuple(f, ::Val{0}) = ()
ntuple(f, ::Val{1}) = (@_inline_meta; (f(1),))
ntuple(f, ::Val{2}) = (@_inline_meta; (f(1), f(2)))
ntuple(f, ::Val{3}) = (@_inline_meta; (f(1), f(2), f(3)))

@inline function ntuple(f)::F, ::Val{N}) where {F,N}
    N::Int
    (N >= 0) || throw(ArgumentError(string("tuple length should be ≥ 0, got ", N)))
    if @generated
        quote
            @nexprs $N i → t_i = f(i)
            @ncall $N tuple t
        end
    else
        Tuple(f(i) for i = 1:N)
    end
end

```

All text set in JuliaMono.

JuliaMono is licensed with the
SIL OpenFont licence.



Copyright Creative Commons
CC BY-SA 3.0
(Attribution-ShareAlike 3.0)

```

@inline function fill_to_length(t::Tuple, val, ::Val{_N}) where {_N}
    M = length(t)
    N = _N::Int
    M > N && throw(ArgumentError("input tuple of length $M, requested $N"))
    if @generated
        quote
            (t..., $(fill(:val, (_N::Int) - length(t.parameters))...))
        end
    else
        Tuple(val for i = 1:N)
    end
end

```