

JUlamono

a monospaced programming font
with reasonable Unicode support

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

JuliaMono Light

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

JuliaMono Regular

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

JuliaMono Medium

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

JuliaMono Bold

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

JuliaMono ExtraBold

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

JuliaMono Black

```
abcdefghijklmnoprstuvwxyz 12345
ABCDEFGHIJKLMNPQRSTUVWXYZ 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', ὡς ἔχει, λέξαι θέλω σοι πρὸν θανεῖν ἢ 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 lluït rëtol blavís d'onze kWh.

Czech Zvlášť zákeřný učeň s dholičky běží podél zóny úlů

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

English Sphinx of black quartz, judge my vow.

Estonian Põdur Zagrebi tšellomängija-följetonist Cigo 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.

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

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

Guarani H̄iñlandiagua kuñanguéra oho peteī sa'yuju ypa'üme Gavōme ombo'e hāhua ingyleñe'ẽ mitānguérare ne'ēndy'ŷ.

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

IPA [gʷʃ?.nas.do:j.kʰlja]
[jan.nɸiwo.çiuɛn.ɣwa]

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

Irish Čuaití bhé mórsáct le dlúthspád fíorfinn trí hata mo dea-porcáin big.

Latvian Muļķa hipiji mēģina brīvi nogaršot celofāna žņaudzējčūsku.

Lithuanian Įlinkdama fechtuotojo špaga sublykčiojusi pragręžė apvalų aarbūzą.

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 ośm 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 ж ти, прощайся без ғольфів!

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

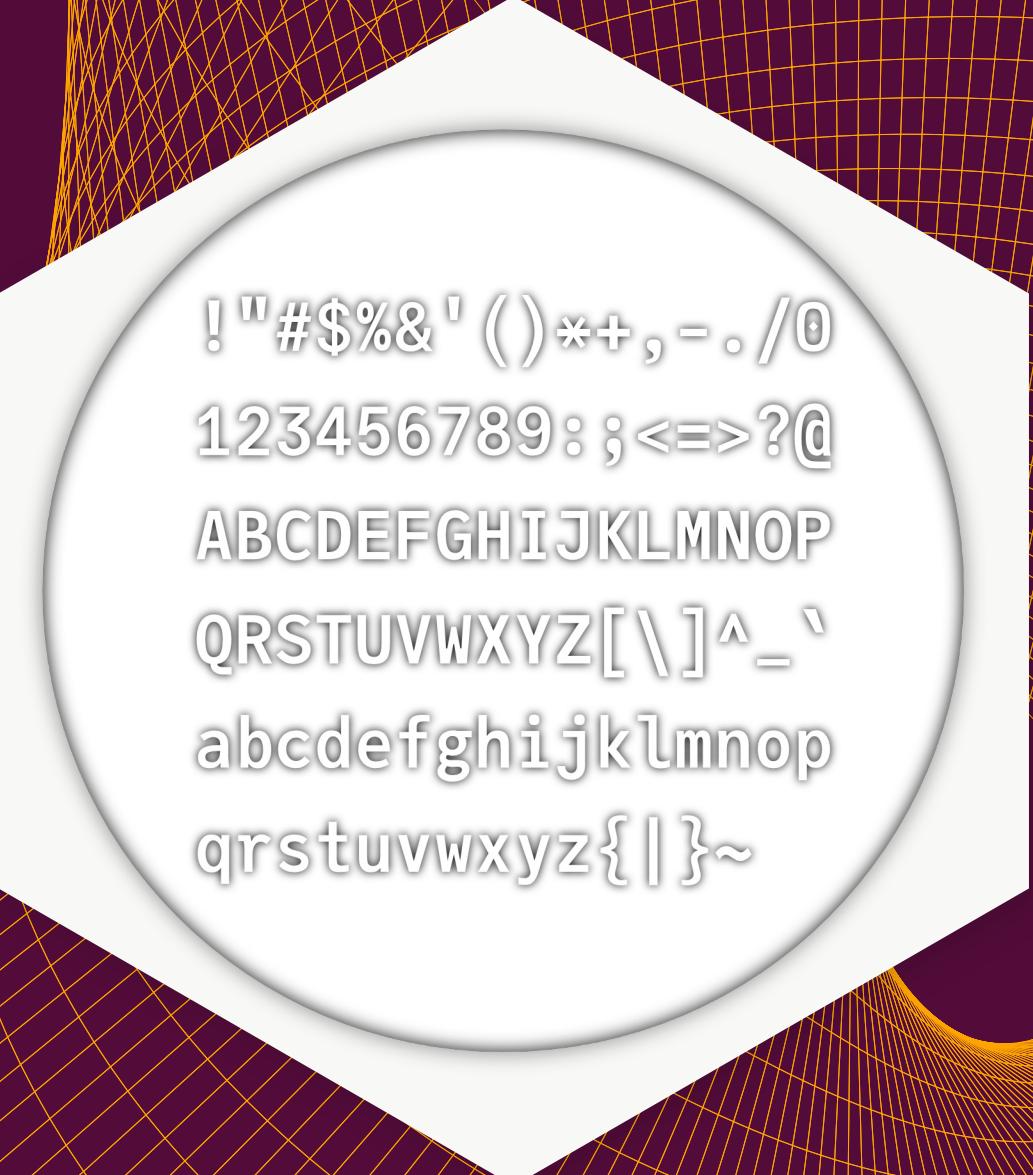
from 1513. According to Vasari, "after he had lingered over it four years, left it unfinished". In 1516, Leonardo was invited by King Francis I to work at the Clos Lucé near the Château d'Amboise; it is believed that he took the *Mona Lisa* with him and continued to work on it after he moved to France. 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

```
+ 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 end
+ 33
+ 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 = fill!(t[i], val)
+ 65     end
+ 66     t
+ 67 end
+ 68
+ 69
+ 70 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+ 71     for i = 1:N
+ 72         t[i] = val
+ 73     end
+ 74     t
+ 75 end
+ 76
+ 77 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+ 78     for i = 1:N
+ 79         t[i] = val
+ 80     end
+ 81     t
+ 82 end
+ 83
+ 84 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+ 85     for i = 1:N
+ 86         t[i] = val
+ 87     end
+ 88     t
+ 89 end
+ 90
+ 91 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+ 92     for i = 1:N
+ 93         t[i] = val
+ 94     end
+ 95     t
+ 96 end
+ 97
+ 98 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+ 99     for i = 1:N
+100        t[i] = val
+101    end
+102    t
+103 end
+104
+105 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+106     for i = 1:N
+107         t[i] = val
+108     end
+109     t
+110 end
+111
+112 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+113     for i = 1:N
+114         t[i] = val
+115     end
+116     t
+117 end
+118
+119 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+120     for i = 1:N
+121         t[i] = val
+122     end
+123     t
+124 end
+125
+126 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+127     for i = 1:N
+128         t[i] = val
+129     end
+130     t
+131 end
+132
+133 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+134     for i = 1:N
+135         t[i] = val
+136     end
+137     t
+138 end
+139
+140 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+141     for i = 1:N
+142         t[i] = val
+143     end
+144     t
+145 end
+146
+147 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+148     for i = 1:N
+149         t[i] = val
+150     end
+151     t
+152 end
+153
+154 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+155     for i = 1:N
+156         t[i] = val
+157     end
+158     t
+159 end
+160
+161 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+162     for i = 1:N
+163         t[i] = val
+164     end
+165     t
+166 end
+167
+168 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+169     for i = 1:N
+170         t[i] = val
+171     end
+172     t
+173 end
+174
+175 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+176     for i = 1:N
+177         t[i] = val
+178     end
+179     t
+180 end
+181
+182 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+183     for i = 1:N
+184         t[i] = val
+185     end
+186     t
+187 end
+188
+189 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+190     for i = 1:N
+191         t[i] = val
+192     end
+193     t
+194 end
+195
+196 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+197     for i = 1:N
+198         t[i] = val
+199     end
+200     t
+201 end
+202
+203 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+204     for i = 1:N
+205         t[i] = val
+206     end
+207     t
+208 end
+209
+210 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+211     for i = 1:N
+212         t[i] = val
+213     end
+214     t
+215 end
+216
+217 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+218     for i = 1:N
+219         t[i] = val
+220     end
+221     t
+222 end
+223
+224 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+225     for i = 1:N
+226         t[i] = val
+227     end
+228     t
+229 end
+230
+231 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+232     for i = 1:N
+233         t[i] = val
+234     end
+235     t
+236 end
+237
+238 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+239     for i = 1:N
+240         t[i] = val
+241     end
+242     t
+243 end
+244
+245 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+246     for i = 1:N
+247         t[i] = val
+248     end
+249     t
+250 end
+251
+252 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+253     for i = 1:N
+254         t[i] = val
+255     end
+256     t
+257 end
+258
+259 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+260     for i = 1:N
+261         t[i] = val
+262     end
+263     t
+264 end
+265
+266 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+267     for i = 1:N
+268         t[i] = val
+269     end
+270     t
+271 end
+272
+273 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+274     for i = 1:N
+275         t[i] = val
+276     end
+277     t
+278 end
+279
+280 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+281     for i = 1:N
+282         t[i] = val
+283     end
+284     t
+285 end
+286
+287 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+288     for i = 1:N
+289         t[i] = val
+290     end
+291     t
+292 end
+293
+294 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+295     for i = 1:N
+296         t[i] = val
+297     end
+298     t
+299 end
+300
+301 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+302     for i = 1:N
+303         t[i] = val
+304     end
+305     t
+306 end
+307
+308 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+309     for i = 1:N
+310         t[i] = val
+311     end
+312     t
+313 end
+314
+315 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+316     for i = 1:N
+317         t[i] = val
+318     end
+319     t
+320 end
+321
+322 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+323     for i = 1:N
+324         t[i] = val
+325     end
+326     t
+327 end
+328
+329 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+330     for i = 1:N
+331         t[i] = val
+332     end
+333     t
+334 end
+335
+336 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+337     for i = 1:N
+338         t[i] = val
+339     end
+340     t
+341 end
+342
+343 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+344     for i = 1:N
+345         t[i] = val
+346     end
+347     t
+348 end
+349
+350 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+351     for i = 1:N
+352         t[i] = val
+353     end
+354     t
+355 end
+356
+357 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+358     for i = 1:N
+359         t[i] = val
+360     end
+361     t
+362 end
+363
+364 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+365     for i = 1:N
+366         t[i] = val
+367     end
+368     t
+369 end
+370
+371 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+372     for i = 1:N
+373         t[i] = val
+374     end
+375     t
+376 end
+377
+378 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+379     for i = 1:N
+380         t[i] = val
+381     end
+382     t
+383 end
+384
+385 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+386     for i = 1:N
+387         t[i] = val
+388     end
+389     t
+390 end
+391
+392 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+393     for i = 1:N
+394         t[i] = val
+395     end
+396     t
+397 end
+398
+399 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+400     for i = 1:N
+401         t[i] = val
+402     end
+403     t
+404 end
+405
+406 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+407     for i = 1:N
+408         t[i] = val
+409     end
+410     t
+411 end
+412
+413 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+414     for i = 1:N
+415         t[i] = val
+416     end
+417     t
+418 end
+419
+420 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+421     for i = 1:N
+422         t[i] = val
+423     end
+424     t
+425 end
+426
+427 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+428     for i = 1:N
+429         t[i] = val
+430     end
+431     t
+432 end
+433
+434 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+435     for i = 1:N
+436         t[i] = val
+437     end
+438     t
+439 end
+440
+441 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+442     for i = 1:N
+443         t[i] = val
+444     end
+445     t
+446 end
+447
+448 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+449     for i = 1:N
+450         t[i] = val
+451     end
+452     t
+453 end
+454
+455 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+456     for i = 1:N
+457         t[i] = val
+458     end
+459     t
+460 end
+461
+462 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+463     for i = 1:N
+464         t[i] = val
+465     end
+466     t
+467 end
+468
+469 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+470     for i = 1:N
+471         t[i] = val
+472     end
+473     t
+474 end
+475
+476 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+477     for i = 1:N
+478         t[i] = val
+479     end
+480     t
+481 end
+482
+483 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+484     for i = 1:N
+485         t[i] = val
+486     end
+487     t
+488 end
+489
+490 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+491     for i = 1:N
+492         t[i] = val
+493     end
+494     t
+495 end
+496
+497 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+498     for i = 1:N
+499         t[i] = val
+500     end
+501     t
+502 end
+503
+504 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+505     for i = 1:N
+506         t[i] = val
+507     end
+508     t
+509 end
+510
+511 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+512     for i = 1:N
+513         t[i] = val
+514     end
+515     t
+516 end
+517
+518 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+519     for i = 1:N
+520         t[i] = val
+521     end
+522     t
+523 end
+524
+525 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+526     for i = 1:N
+527         t[i] = val
+528     end
+529     t
+530 end
+531
+532 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+533     for i = 1:N
+534         t[i] = val
+535     end
+536     t
+537 end
+538
+539 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+540     for i = 1:N
+541         t[i] = val
+542     end
+543     t
+544 end
+545
+546 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+547     for i = 1:N
+548         t[i] = val
+549     end
+550     t
+551 end
+552
+553 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+554     for i = 1:N
+555         t[i] = val
+556     end
+557     t
+558 end
+559
+560 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+561     for i = 1:N
+562         t[i] = val
+563     end
+564     t
+565 end
+566
+567 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+568     for i = 1:N
+569         t[i] = val
+570     end
+571     t
+572 end
+573
+574 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+575     for i = 1:N
+576         t[i] = val
+577     end
+578     t
+579 end
+580
+581 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+582     for i = 1:N
+583         t[i] = val
+584     end
+585     t
+586 end
+587
+588 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+589     for i = 1:N
+590         t[i] = val
+591     end
+592     t
+593 end
+594
+595 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+596     for i = 1:N
+597         t[i] = val
+598     end
+599     t
+600 end
+601
+602 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+603     for i = 1:N
+604         t[i] = val
+605     end
+606     t
+607 end
+608
+609 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+610     for i = 1:N
+611         t[i] = val
+612     end
+613     t
+614 end
+615
+616 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+617     for i = 1:N
+618         t[i] = val
+619     end
+620     t
+621 end
+622
+623 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+624     for i = 1:N
+625         t[i] = val
+626     end
+627     t
+628 end
+629
+630 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+631     for i = 1:N
+632         t[i] = val
+633     end
+634     t
+635 end
+636
+637 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+638     for i = 1:N
+639         t[i] = val
+640     end
+641     t
+642 end
+643
+644 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+645     for i = 1:N
+646         t[i] = val
+647     end
+648     t
+649 end
+650
+651 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+652     for i = 1:N
+653         t[i] = val
+654     end
+655     t
+656 end
+657
+658 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+659     for i = 1:N
+660         t[i] = val
+661     end
+662     t
+663 end
+664
+665 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+666     for i = 1:N
+667         t[i] = val
+668     end
+669     t
+670 end
+671
+672 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+673     for i = 1:N
+674         t[i] = val
+675     end
+676     t
+677 end
+678
+679 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+680     for i = 1:N
+681         t[i] = val
+682     end
+683     t
+684 end
+685
+686 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+687     for i = 1:N
+688         t[i] = val
+689     end
+690     t
+691 end
+692
+693 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+694     for i = 1:N
+695         t[i] = val
+696     end
+697     t
+698 end
+699
+700 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+701     for i = 1:N
+702         t[i] = val
+703     end
+704     t
+705 end
+706
+707 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+708     for i = 1:N
+709         t[i] = val
+710     end
+711     t
+712 end
+713
+714 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+715     for i = 1:N
+716         t[i] = val
+717     end
+718     t
+719 end
+720
+721 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+722     for i = 1:N
+723         t[i] = val
+724     end
+725     t
+726 end
+727
+728 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+729     for i = 1:N
+730         t[i] = val
+731     end
+732     t
+733 end
+734
+735 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+736     for i = 1:N
+737         t[i] = val
+738     end
+739     t
+740 end
+741
+742 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+743     for i = 1:N
+744         t[i] = val
+745     end
+746     t
+747 end
+748
+749 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+750     for i = 1:N
+751         t[i] = val
+752     end
+753     t
+754 end
+755
+756 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+757     for i = 1:N
+758         t[i] = val
+759     end
+760     t
+761 end
+762
+763 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+764     for i = 1:N
+765         t[i] = val
+766     end
+767     t
+768 end
+769
+770 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+771     for i = 1:N
+772         t[i] = val
+773     end
+774     t
+775 end
+776
+777 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+778     for i = 1:N
+779         t[i] = val
+780     end
+781     t
+782 end
+783
+784 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+785     for i = 1:N
+786         t[i] = val
+787     end
+788     t
+789 end
+790
+791 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+792     for i = 1:N
+793         t[i] = val
+794     end
+795     t
+796 end
+797
+798 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+799     for i = 1:N
+800         t[i] = val
+801     end
+802     t
+803 end
+804
+805 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+806     for i = 1:N
+807         t[i] = val
+808     end
+809     t
+810 end
+811
+812 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+813     for i = 1:N
+814         t[i] = val
+815     end
+816     t
+817 end
+818
+819 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+820     for i = 1:N
+821         t[i] = val
+822     end
+823     t
+824 end
+825
+826 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+827     for i = 1:N
+828         t[i] = val
+829     end
+830     t
+831 end
+832
+833 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+834     for i = 1:N
+835         t[i] = val
+836     end
+837     t
+838 end
+839
+840 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+841     for i = 1:N
+842         t[i] = val
+843     end
+844     t
+845 end
+846
+847 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+848     for i = 1:N
+849         t[i] = val
+850     end
+851     t
+852 end
+853
+854 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+855     for i = 1:N
+856         t[i] = val
+857     end
+858     t
+859 end
+860
+861 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+862     for i = 1:N
+863         t[i] = val
+864     end
+865     t
+866 end
+867
+868 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+869     for i = 1:N
+870         t[i] = val
+871     end
+872     t
+873 end
+874
+875 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+876     for i = 1:N
+877         t[i] = val
+878     end
+879     t
+880 end
+881
+882 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+883     for i = 1:N
+884         t[i] = val
+885     end
+886     t
+887 end
+888
+889 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+890     for i = 1:N
+891         t[i] = val
+892     end
+893     t
+894 end
+895
+896 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+897     for i = 1:N
+898         t[i] = val
+899     end
+900     t
+901 end
+902
+903 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+904     for i = 1:N
+905         t[i] = val
+906     end
+907     t
+908 end
+909
+910 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+911     for i = 1:N
+912         t[i] = val
+913     end
+914     t
+915 end
+916
+917 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+918     for i = 1:N
+919         t[i] = val
+920     end
+921     t
+922 end
+923
+924 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+925     for i = 1:N
+926         t[i] = val
+927     end
+928     t
+929 end
+930
+931 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+932     for i = 1:N
+933         t[i] = val
+934     end
+935     t
+936 end
+937
+938 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+939     for i = 1:N
+940         t[i] = val
+941     end
+942     t
+943 end
+944
+945 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+946     for i = 1:N
+947         t[i] = val
+948     end
+949     t
+950 end
+951
+952 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+953     for i = 1:N
+954         t[i] = val
+955     end
+956     t
+957 end
+958
+959 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+960     for i = 1:N
+961         t[i] = val
+962     end
+963     t
+964 end
+965
+966 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+967     for i = 1:N
+968         t[i] = val
+969     end
+970     t
+971 end
+972
+973 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+974     for i = 1:N
+975         t[i] = val
+976     end
+977     t
+978 end
+979
+980 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+981     for i = 1:N
+982         t[i] = val
+983     end
+984     t
+985 end
+986
+987 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+988     for i = 1:N
+989         t[i] = val
+990     end
+991     t
+992 end
+993
+994 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+995     for i = 1:N
+996         t[i] = val
+997     end
+998     t
+999 end
+1000
+1001 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1002     for i = 1:N
+1003         t[i] = val
+1004     end
+1005     t
+1006 end
+1007
+1008 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1009     for i = 1:N
+1010         t[i] = val
+1011     end
+1012     t
+1013 end
+1014
+1015 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1016     for i = 1:N
+1017         t[i] = val
+1018     end
+1019     t
+1020 end
+1021
+1022 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1023     for i = 1:N
+1024         t[i] = val
+1025     end
+1026     t
+1027 end
+1028
+1029 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1030     for i = 1:N
+1031         t[i] = val
+1032     end
+1033     t
+1034 end
+1035
+1036 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1037     for i = 1:N
+1038         t[i] = val
+1039     end
+1040     t
+1041 end
+1042
+1043 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1044     for i = 1:N
+1045         t[i] = val
+1046     end
+1047     t
+1048 end
+1049
+1050 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1051     for i = 1:N
+1052         t[i] = val
+1053     end
+1054     t
+1055 end
+1056
+1057 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1058     for i = 1:N
+1059         t[i] = val
+1060     end
+1061     t
+1062 end
+1063
+1064 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1065     for i = 1:N
+1066         t[i] = val
+1067     end
+1068     t
+1069 end
+1070
+1071 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1072     for i = 1:N
+1073         t[i] = val
+1074     end
+1075     t
+1076 end
+1077
+1078 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1079     for i = 1:N
+1080         t[i] = val
+1081     end
+1082     t
+1083 end
+1084
+1085 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1086     for i = 1:N
+1087         t[i] = val
+1088     end
+1089     t
+1090 end
+1091
+1092 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1093     for i = 1:N
+1094         t[i] = val
+1095     end
+1096     t
+1097 end
+1098
+1099 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1100     for i = 1:N
+1101         t[i] = val
+1102     end
+1103     t
+1104 end
+1105
+1106 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1107     for i = 1:N
+1108         t[i] = val
+1109     end
+1110     t
+1111 end
+1112
+1113 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1114     for i = 1:N
+1115         t[i] = val
+1116     end
+1117     t
+1118 end
+1119
+1120 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1121     for i = 1:N
+1122         t[i] = val
+1123     end
+1124     t
+1125 end
+1126
+1127 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1128     for i = 1:N
+1129         t[i] = val
+1130     end
+1131     t
+1132 end
+1133
+1134 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1135     for i = 1:N
+1136         t[i] = val
+1137     end
+1138     t
+1139 end
+1140
+1141 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1142     for i = 1:N
+1143         t[i] = val
+1144     end
+1145     t
+1146 end
+1147
+1148 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1149     for i = 1:N
+1150         t[i] = val
+1151     end
+1152     t
+1153 end
+1154
+1155 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1156     for i = 1:N
+1157         t[i] = val
+1158     end
+1159     t
+1160 end
+1161
+1162 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1163     for i = 1:N
+1164         t[i] = val
+1165     end
+1166     t
+1167 end
+1168
+1169 @_inline function fill!(t::NTuple{N,T}, val) where {N,T}
+1170     for i =
```

length should be

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

```