

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 dolíčky 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-fölljetonist 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** Hìlandiagua kuñanguéra oho peteī sa'yju ypa'üme Gavõme ombo'e hāgua ingyleñe'ẽ mitānguérare ne'ēndy'ŷ.

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

**IPA** [gʷʃ?.nas.dɔ̃ŋ.kʰlja]  
[ɟan.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** Mulķa hipji 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	$\mu$	$10^{-6}$	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	$\lambda_c$	$2.426310215 \times 10^{-12}$
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	$1 \text{ eV}^{-1}$	$1.97 \times 10^{-7}$
reduced wavelength of hydrogen radiation	$1/R_\infty$	$9.112670505509 \times 10^{-8}$
Planck length	$\ell_p$	$1.616199 \times 10^{-35}$
Stoney unit of length	$l_s$	$1.381 \times 10^{-35}$
quantum chromodynamics (QCD) unit of length	$l_{\text{qcd}}$	$2.103 \times 10^{-16}$

	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

ⓘ

blackboard a  
u+1d552

ⓘ

blackboard S  
u+1d54a



heavy 12pt Pinwheel Star  
u+1f7d4



ace of spades  
u+1f0a1

ⓘ

zNot Rel Comp  
u+2a3e



smash product  
u+2a33

ⓘ

G circled  
u+24bc

ⓘ

cada una  
u+2106

ⓘ

segno  
u+1d10b



editorial coronis  
u+2e0e



cube root  
u+221b

ⓘ

Sharp S  
u+1e9e



speech bubble  
u+1f4ac



Roman 50K  
u+2187



white upperleft square  
u+25f0



eternity  
u+58e

fff

triple integral  
u+222d



DiffyQ  
u+e830

ⓘ

subset with +  
u+2abf

ⓘ

per  
u+214c



triple dagger  
u+2e4b



heavy right arrow  
u+1f882

ⓘ

hbar  
u+210f

ⓘ

alef  
u+2135

ⓘ

recycled paper  
u+267c

ⓘ

ⓘ

Mayan 14  
u+1d2ee

ⓘ

left arrow subset  
u+297a

ⓘ

swash ampersand  
u+1f675

ⓘ

earth ground  
u+23da

ⓘ

hryvnia  
u+20b4

ⓘ

git PR  
u+e726

ⓘ

coptic 800  
u+102fa

ⓘ

ogham nion  
u+1685

ⓘ

numero  
u+2116

ⓘ

devangari 5  
u+96b

ⓘ

Julia dots  
u+e800

ⓘ

amalgamation Or Coproduct  
u+2a3f

ⓘ

subscript b  
u+e805

ⓘ

umbrella  
u+2602

ⓘ

Pluto  
u+2bd4

ⓘ

division times  
u+22c7

ⓘ

R fraktur  
u+211c

ⓘ

psi  
u+3c8

ⓘ

weierstrass  
u+2118

ⓘ

APL dot tack jot  
u+234e

ⓘ

tetragram completion  
u+1d34e

ⓘ

Braille 2345 (t)  
u+281e

ⓘ

circled asterisk  
u+229b

ⓘ

yod  
u+5d9

ⓘ

transversal intersect  
u+2adb

ⓘ



palm branch  
u+2e19

ⓘ



fist  
u+270a

ⓘ



alembic  
u+2697

ⓘ



hacker  
u+e826

ⓘ



v cat  
u+e838

ⓘ

trans Pluto  
u+2bd7

random Selection of Unicode characters

## contextual alternates

off	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

```
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, Isabela 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

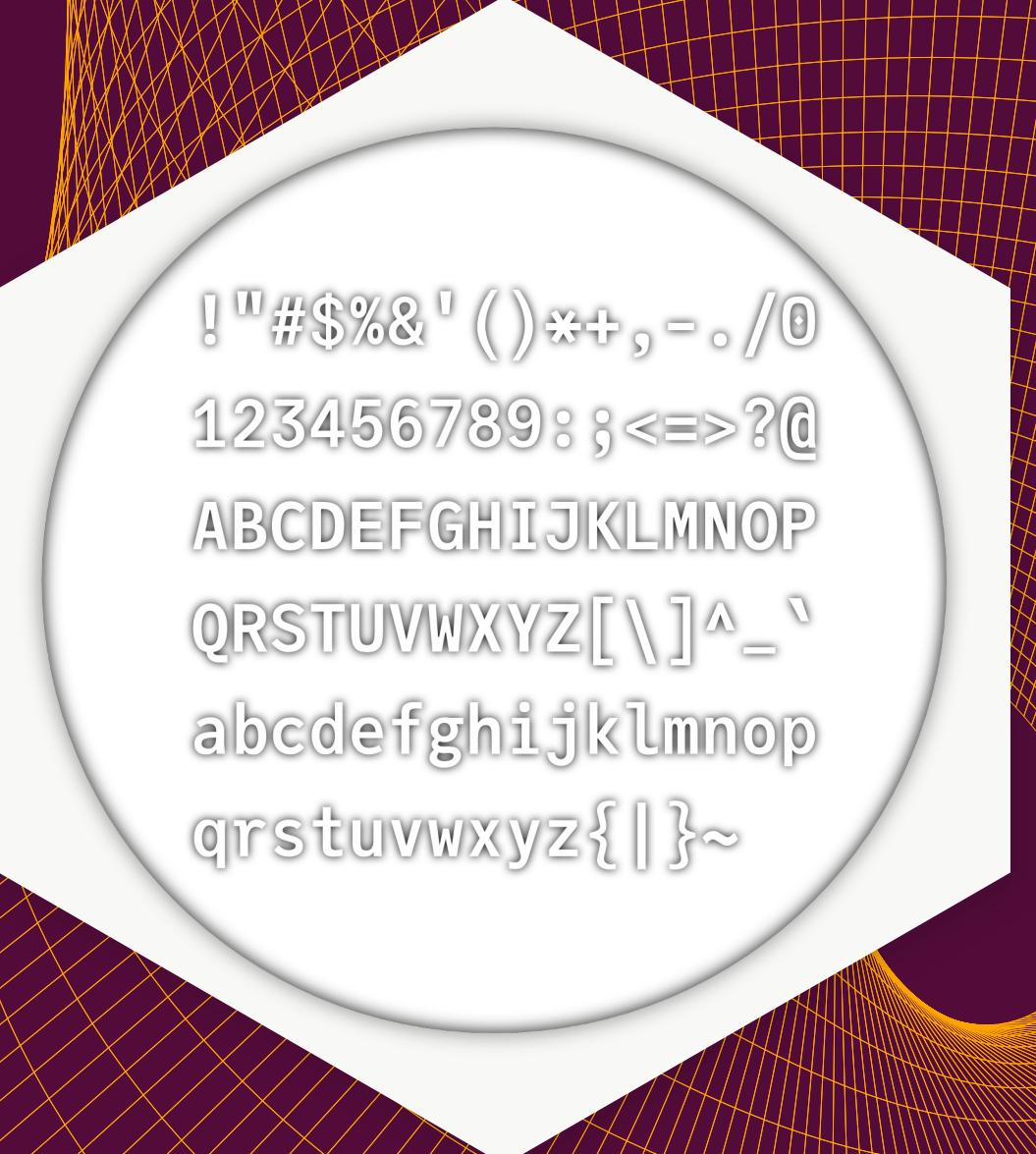
painted the work from 1513. According to Vasari, "after he had lingered over it four years, left it unfinished". In 1516, Leonardo was invited by King Francis I to work at the Clos Lucé near the Château d'Amboise; it is believed that he took the *Mona Lisa* with him and continued to work on it after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

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 to 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 1800s, 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 Valfierro, who claimed to have masterminded the theft. Forger Yves Chaudron was to have created ... [text by Wikipedia contributors. "Mona Lisa." Wikipedia, The Free Encyclopedia.]

may 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: ;<=>?@  
ABCDEFGHIJKLMNP  
QRSTUVWXYZ[\]^\_`  
abcdefghijklmnop  
qrstuvwxyz{|}~

JuliaMono Light

JuliaMono Regular

JuliaMono Medium

JuliaMono Bold

JuliaMono ExtraBold

JuliaMono Black

```

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)
2       return t

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

```

```
11 ...
12
13 function f(n, m) {
14     if (n < 0) {
15         return 0;
16     }
17     if (n == 0) {
18         return 1;
19     }
20     if (n == 1) {
21         return 2;
22     }
23     if (n == 2) {
24         return 3;
25     }
26     if (n == 3) {
27         return 4;
28     }
29     if (n == 4) {
30         return 5;
31     }
32     if (n == 5) {
33         return 6;
34     }
35     if (n == 6) {
36         return 7;
37     }
38     if (n == 7) {
39         return 8;
40     }
41     if (n == 8) {
42         return 9;
43     }
44     if (n == 9) {
45         return 10;
46     }
47     if (n == 10) {
48         return 11;
49     }
50     if (n == 11) {
51         return 12;
52     }
53     if (n == 12) {
54         return 13;
55     }
56     if (n == 13) {
57         return 14;
58     }
59     if (n == 14) {
60         return 15;
61     }
62     if (n == 15) {
63         return 16;
64     }
65     if (n == 16) {
66         return 17;
67     }
68     if (n == 17) {
69         return 18;
70     }
71     if (n == 18) {
72         return 19;
73     }
74     if (n == 19) {
75         return 20;
76     }
77     if (n == 20) {
78         return 21;
79     }
80     if (n == 21) {
81         return 22;
82     }
83     if (n == 22) {
84         return 23;
85     }
86     if (n == 23) {
87         return 24;
88     }
89     if (n == 24) {
90         return 25;
91     }
92     if (n == 25) {
93         return 26;
94     }
95     if (n == 26) {
96         return 27;
97     }
98     if (n == 27) {
99         return 28;
100    }
101 }
102
103 function g(n, m) {
104     if (n < 0) {
105         return 0;
106     }
107     if (n == 0) {
108         return 1;
109     }
110     if (n == 1) {
111         return 2;
112     }
113     if (n == 2) {
114         return 3;
115     }
116     if (n == 3) {
117         return 4;
118     }
119     if (n == 4) {
120         return 5;
121     }
122     if (n == 5) {
123         return 6;
124     }
125     if (n == 6) {
126         return 7;
127     }
128     if (n == 7) {
129         return 8;
130     }
131     if (n == 8) {
132         return 9;
133     }
134     if (n == 9) {
135         return 10;
136     }
137     if (n == 10) {
138         return 11;
139     }
140     if (n == 11) {
141         return 12;
142     }
143     if (n == 12) {
144         return 13;
145     }
146     if (n == 13) {
147         return 14;
148     }
149     if (n == 14) {
150         return 15;
151     }
152     if (n == 15) {
153         return 16;
154     }
155     if (n == 16) {
156         return 17;
157     }
158     if (n == 17) {
159         return 18;
160     }
161     if (n == 18) {
162         return 19;
163     }
164     if (n == 19) {
165         return 20;
166     }
167     if (n == 20) {
168         return 21;
169     }
170     if (n == 21) {
171         return 22;
172     }
173     if (n == 22) {
174         return 23;
175     }
176     if (n == 23) {
177         return 24;
178     }
179     if (n == 24) {
180         return 25;
181     }
182     if (n == 25) {
183         return 26;
184     }
185     if (n == 26) {
186         return 27;
187     }
188     if (n == 27) {
189         return 28;
190     }
191     if (n == 28) {
192         return 29;
193     }
194     if (n == 29) {
195         return 30;
196     }
197     if (n == 30) {
198         return 31;
199     }
200     if (n == 31) {
201         return 32;
202     }
203     if (n == 32) {
204         return 33;
205     }
206     if (n == 33) {
207         return 34;
208     }
209     if (n == 34) {
210         return 35;
211     }
212     if (n == 35) {
213         return 36;
214     }
215     if (n == 36) {
216         return 37;
217     }
218     if (n == 37) {
219         return 38;
220     }
221     if (n == 38) {
222         return 39;
223     }
224     if (n == 39) {
225         return 40;
226     }
227     if (n == 40) {
228         return 41;
229     }
230     if (n == 41) {
231         return 42;
232     }
233     if (n == 42) {
234         return 43;
235     }
236     if (n == 43) {
237         return 44;
238     }
239     if (n == 44) {
240         return 45;
241     }
242     if (n == 45) {
243         return 46;
244     }
245     if (n == 46) {
246         return 47;
247     }
248     if (n == 47) {
249         return 48;
250     }
251     if (n == 48) {
252         return 49;
253     }
254     if (n == 49) {
255         return 50;
256     }
257     if (n == 50) {
258         return 51;
259     }
260     if (n == 51) {
261         return 52;
262     }
263     if (n == 52) {
264         return 53;
265     }
266     if (n == 53) {
267         return 54;
268     }
269     if (n == 54) {
270         return 55;
271     }
272     if (n == 55) {
273         return 56;
274     }
275     if (n == 56) {
276         return 57;
277     }
278     if (n == 57) {
279         return 58;
280     }
281     if (n == 58) {
282         return 59;
283     }
284     if (n == 59) {
285         return 60;
286     }
287     if (n == 60) {
288         return 61;
289     }
290     if (n == 61) {
291         return 62;
292     }
293     if (n == 62) {
294         return 63;
295     }
296     if (n == 63) {
297         return 64;
298     }
299     if (n == 64) {
300         return 65;
301     }
302     if (n == 65) {
303         return 66;
304     }
305     if (n == 66) {
306         return 67;
307     }
308     if (n == 67) {
309         return 68;
310     }
311     if (n == 68) {
312         return 69;
313     }
314     if (n == 69) {
315         return 70;
316     }
317     if (n == 70) {
318         return 71;
319     }
320     if (n == 71) {
321         return 72;
322     }
323     if (n == 72) {
324         return 73;
325     }
326     if (n == 73) {
327         return 74;
328     }
329     if (n == 74) {
330         return 75;
331     }
332     if (n == 75) {
333         return 76;
334     }
335     if (n == 76) {
336         return 77;
337     }
338     if (n == 77) {
339         return 78;
340     }
341     if (n == 78) {
342         return 79;
343     }
344     if (n == 79) {
345         return 80;
346     }
347     if (n == 80) {
348         return 81;
349     }
350     if (n == 81) {
351         return 82;
352     }
353     if (n == 82) {
354         return 83;
355     }
356     if (n == 83) {
357         return 84;
358     }
359     if (n == 84) {
360         return 85;
361     }
362     if (n == 85) {
363         return 86;
364     }
365     if (n == 86) {
366         return 87;
367     }
368     if (n == 87) {
369         return 88;
370     }
371     if (n == 88) {
372         return 89;
373     }
374     if (n == 89) {
375         return 90;
376     }
377     if (n == 90) {
378         return 91;
379     }
380     if (n == 91) {
381         return 92;
382     }
383     if (n == 92) {
384         return 93;
385     }
386     if (n == 93) {
387         return 94;
388     }
389     if (n == 94) {
390         return 95;
391     }
392     if (n == 95) {
393         return 96;
394     }
395     if (n == 96) {
396         return 97;
397     }
398     if (n == 97) {
399         return 98;
400     }
401     if (n == 98) {
402         return 99;
403     }
404     if (n == 99) {
405         return 100;
406     }
407 }
408
409 function h(n, m) {
410     if (n < 0) {
411         return 0;
412     }
413     if (n == 0) {
414         return 1;
415     }
416     if (n == 1) {
417         return 2;
418     }
419     if (n == 2) {
420         return 3;
421     }
422     if (n == 3) {
423         return 4;
424     }
425     if (n == 4) {
426         return 5;
427     }
428     if (n == 5) {
429         return 6;
430     }
431     if (n == 6) {
432         return 7;
433     }
434     if (n == 7) {
435         return 8;
436     }
437     if (n == 8) {
438         return 9;
439     }
440     if (n == 9) {
441         return 10;
442     }
443     if (n == 10) {
444         return 11;
445     }
446     if (n == 11) {
447         return 12;
448     }
449     if (n == 12) {
450         return 13;
451     }
452     if (n == 13) {
453         return 14;
454     }
455     if (n == 14) {
456         return 15;
457     }
458     if (n == 15) {
459         return 16;
460     }
461     if (n == 16) {
462         return 17;
463     }
464     if (n == 17) {
465         return 18;
466     }
467     if (n == 18) {
468         return 19;
469     }
470     if (n == 19) {
471         return 20;
472     }
473     if (n == 20) {
474         return 21;
475     }
476     if (n == 21) {
477         return 22;
478     }
479     if (n == 22) {
480         return 23;
481     }
482     if (n == 23) {
483         return 24;
484     }
485     if (n == 24) {
486         return 25;
487     }
488     if (n == 25) {
489         return 26;
490     }
491     if (n == 26) {
492         return 27;
493     }
494     if (n == 27) {
495         return 28;
496     }
497     if (n == 28) {
498         return 29;
499     }
500     if (n == 29) {
501         return 30;
502     }
503     if (n == 30) {
504         return 31;
505     }
506     if (n == 31) {
507         return 32;
508     }
509     if (n == 32) {
510         return 33;
511     }
512     if (n == 33) {
513         return 34;
514     }
515     if (n == 34) {
516         return 35;
517     }
518     if (n == 35) {
519         return 36;
520     }
521     if (n == 36) {
522         return 37;
523     }
524     if (n == 37) {
525         return 38;
526     }
527     if (n == 38) {
528         return 39;
529     }
530     if (n == 39) {
531         return 40;
532     }
533     if (n == 40) {
534         return 41;
535     }
536     if (n == 41) {
537         return 42;
538     }
539     if (n == 42) {
540         return 43;
541     }
542     if (n == 43) {
543         return 44;
544     }
545     if (n == 44) {
546         return 45;
547     }
548     if (n == 45) {
549         return 46;
550     }
551     if (n == 46) {
552         return 47;
553     }
554     if (n == 47) {
555         return 48;
556     }
557     if (n == 48) {
558         return 49;
559     }
560     if (n == 49) {
561         return 50;
562     }
563     if (n == 50) {
564         return 51;
565     }
566     if (n == 51) {
567         return 52;
568     }
569     if (n == 52) {
570         return 53;
571     }
572     if (n == 53) {
573         return 54;
574     }
575     if (n == 54) {
576         return 55;
577     }
578     if (n == 55) {
579         return 56;
580     }
581     if (n == 56) {
582         return 57;
583     }
584     if (n == 57) {
585         return 58;
586     }
587     if (n == 58) {
588         return 59;
589     }
590     if (n == 59) {
591         return 60;
592     }
593     if (n == 60) {
594         return 61;
595     }
596     if (n == 61) {
597         return 62;
598     }
599     if (n == 62) {
600         return 63;
601     }
602     if (n == 63) {
603         return 64;
604     }
605     if (n == 64) {
606         return 65;
607     }
608     if (n == 65) {
609         return 66;
610     }
611     if (n == 66) {
612         return 67;
613     }
614     if (n == 67) {
615         return 68;
616     }
617     if (n == 68) {
618         return 69;
619     }
620     if (n == 69) {
621         return 70;
622     }
623     if (n == 70) {
624         return 71;
625     }
626     if (n == 71) {
627         return 72;
628     }
629     if (n == 72) {
630         return 73;
631     }
632     if (n == 73) {
633         return 74;
634     }
635     if (n == 74) {
636         return 75;
637     }
638     if (n == 75) {
639         return 76;
640     }
641     if (n == 76) {
642         return 77;
643     }
644     if (n == 77) {
645         return 78;
646     }
647     if (n == 78) {
648         return 79;
649     }
650     if (n == 79) {
651         return 80;
652     }
653     if (n == 80) {
654         return 81;
655     }
656     if (n == 81) {
657         return 82;
658     }
659     if (n == 82) {
660         return 83;
661     }
662     if (n == 83) {
663         return 84;
664     }
665     if (n == 84) {
666         return 85;
667     }
668     if (n == 85) {
669         return 86;
670     }
671     if (n == 86) {
672         return 87;
673     }
674     if (n == 87) {
675         return 88;
676     }
677     if (n == 88) {
678         return 89;
679     }
680     if (n == 89) {
681         return 90;
682     }
683     if (n == 90) {
684         return 91;
685     }
686     if (n == 91) {
687         return 92;
688     }
689     if (n == 92) {
690         return 93;
691     }
692     if (n == 93) {
693         return 94;
694     }
695     if (n == 94) {
696         return 95;
697     }
698     if (n == 95) {
699         return 96;
700     }
701     if (n == 96) {
702         return 97;
703     }
704     if (n == 97) {
705         return 98;
706     }
707     if (n == 98) {
708         return 99;
709     }
710     if (n == 99) {
711         return 100;
712     }
713 }
714
715 function i(n, m) {
716     if (n < 0) {
717         return 0;
718     }
719     if (n == 0) {
720         return 1;
721     }
722     if (n == 1) {
723         return 2;
724     }
725     if (n == 2) {
726         return 3;
727     }
728     if (n == 3) {
729         return 4;
730     }
731     if (n == 4) {
732         return 5;
733     }
734     if (n == 5) {
735         return 6;
736     }
737     if (n == 6) {
738         return 7;
739     }
740     if (n == 7) {
741         return 8;
742     }
743     if (n == 8) {
744         return 9;
745     }
746     if (n == 9) {
747         return 10;
748     }
749     if (n == 10) {
750         return 11;
751     }
752     if (n == 11) {
753         return 12;
754     }
755     if (n == 12) {
756         return 13;
757     }
758     if (n == 13) {
759         return 14;
760     }
761     if (n == 14) {
762         return 15;
763     }
764     if (n == 15) {
765         return 16;
766     }
767     if (n == 16) {
768         return 17;
769     }
770     if (n == 17) {
771         return 18;
772     }
773     if (n == 18) {
774         return 19;
775     }
776     if (n == 19) {
777         return 20;
778     }
779     if (n == 20) {
780         return 21;
781     }
782     if (n == 21) {
783         return 22;
784     }
785     if (n == 22) {
786         return 23;
787     }
788     if (n == 23) {
789         return 24;
790     }
791     if (n == 24) {
792         return 25;
793     }
794     if (n == 25) {
795         return 26;
796     }
797     if (n == 26) {
798         return 27;
799     }
800     if (n == 27) {
801         return 28;
802     }
803     if (n == 28) {
804         return 29;
805     }
806     if (n == 29) {
807         return 30;
808     }
809     if (n == 30) {
810         return 31;
811     }
812     if (n == 31) {
813         return 32;
814     }
815     if (n == 32) {
816         return 33;
817     }
818     if (n == 33) {
819         return 34;
820     }
821     if (n == 34) {
822         return 35;
823     }
824     if (n == 35) {
825         return 36;
826     }
827     if (n == 36) {
828         return 37;
829     }
830     if (n == 37) {
831         return 38;
832     }
833     if (n == 38) {
834         return 39;
835     }
836     if (n == 39) {
837         return 40;
838     }
839     if (n == 40) {
840         return 41;
841     }
842     if (n == 41) {
843         return 42;
844     }
845     if (n == 42) {
846         return 43;
847     }
848     if (n == 43) {
849         return 44;
850     }
851     if (n == 44) {
852         return 45;
853     }
854     if (n == 45) {
855         return 46;
856     }
857     if (n == 46) {
858         return 47;
859     }
860     if (n == 47) {
861         return 48;
862     }
863     if (n == 48) {
864         return 49;
865     }
866     if (n == 49) {
867         return 50;
868     }
869     if (n == 50) {
870         return 51;
871     }
872     if (n == 51) {
873         return 52;
874     }
875     if (n == 52) {
876         return 53;
877     }
878     if (n == 53) {
879         return 54;
880     }
881     if (n == 54) {
882         return 55;
883     }
884     if (n == 55) {
885         return 56;
886     }
887     if (n == 56) {
888         return 57;
889     }
890     if (n == 57) {
891         return 58;
892     }
893     if (n == 58) {
894         return 59;
895     }
896     if (n == 59) {
897         return 60;
898     }
899     if (n == 60) {
900         return 61;
901     }
902     if (n == 61) {
903         return 62;
904     }
905     if (n == 62) {
906         return 63;
907     }
908     if (n == 63) {
909         return 64;
910     }
911     if (n == 64) {
912         return 65;
913     }
914     if (n == 65) {
915         return 66;
916     }
917     if (n == 66) {
918         return 67;
919     }
920     if (n == 67) {
921         return 68;
922     }
923     if (n == 68) {
924         return 69;
925     }
926     if (n == 69) {
927         return 70;
928     }
929     if (n == 70) {
930         return 71;
931     }
932     if (n == 71) {
933         return 72;
934     }
935     if (n == 72) {
936         return 73;
937     }
938     if (n == 73) {
939         return 74;
940     }
941     if (n == 74) {
942         return 75;
943     }
944     if (n == 75) {
945         return 76;
946     }
947     if (n == 76) {
948         return 77;
949     }
950     if (n == 77) {
951         return 78;
952     }
953     if (n == 78) {
954         return 79;
955     }
956     if (n == 79) {
957         return 80;
958     }
959     if (n == 80) {
960         return 81;
961     }
962     if (n == 81) {
963         return 82;
964     }
965     if (n == 82) {
966         return 83;
967     }
968     if (n == 83) {
969         return 84;
970     }
971     if (n == 84) {
972         return 85;
973     }
974     if (n == 85) {
975         return 86;
976     }
977     if (n == 86) {
978         return 87;
979     }
980     if (n == 87) {
981         return 88;
982     }
983     if (n == 88) {
984         return 89;
985     }
986     if (n == 89) {
987         return 90;
988     }
989     if (n == 90) {
990         return 91;
991     }
992     if (n == 91) {
993         return 92;
994     }
995     if (n == 92) {
996         return 93;
997     }
998     if (n == 93) {
999         return 94;
999 }
1000 }
```

```

# 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

```