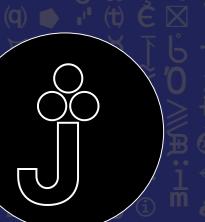


JuliaMono

<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 τὰμα πράγμαθ' ὡς ἔχει, λέξαι θέλω σοι πρὸν 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 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 spilledede 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 H̄īlandiagua kuñanguéra oho petēī sa'yuju ypa'üme Gavõme ombo'e hāgua ingyleñe'ẽ mitãnguérare ne'ëndy'ÿ.

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

IPA [gʷʃ?.nas.doŋ.kʰlja]
[jan.nɸiwo.çiuɛn.ɣwa]

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

Irish Čuaig bé mórsáć le dlúthspád fíorfinn trí hata mo ðea-þorcáin big.

Latvian Mulķa hipji mēģina brīvi nogaršot celofāna žņaudzējūsku.

Lithuanian Įlinkdama fechtuotojo špaga sublykčiojusi pragręžė apvalų arbūzą.

Macedonian Сидарски пејзаж: шугав билмез со чудење ұвака қофте и көл 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	sexillion
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	sexillionth
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

```
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
end
```

```
..._cav 101200  
typeintersect chown -  
redirect_stdin Iterators _  
opp! @timed walkdir IdDict ↳ pc  
ational unescape_string chop in  
g @threadcall replace! display gcd  
e \ CartesianIndex acot Cshort check.  
isassigned : @_MODULE__ frexp isabstv  
edException Cwchar_t LinearIndices Inde  
bty! eachcol kron mtime isinteractive si.  
hash unmark readlines stdout acos delete!  
ns isinf IOStream inpermute! falses count  
BigInt v hex2bytes empty IndexLinear cumsum  
tan startswith @ccall isequal rotr90 BitArra  
ngle issetequal lock isreadonly rotl90 >; ls  
ns exp promote_rule fill! findprev any Matrix ()  
cd intersect! seekstart adjoint ↳ titlecase c  
error seek @cmd propertynames splitdir ispat  
iterate flush acotd @label ENV nameof Pipe Cs  
te code_lowered symlink union atexit isupperc  
e Inf16 DimensionMismatch StridedVector Libc  
onString seed == @r_str BitSet CompositeExcept  
actDict fieldoffset ↳ reset run logip broadcast  
similar popat! tempdir ismissing UnitRange «  
safe_read foreach nextprod acosd RoundNeares  
typeintersect chown ischardev uppercasefirs  
redirect_stdin Iterators schedule Dict acc  
opp! @timed walkdir IdDict ↳ partialsort! s'  
ational unescape_string chop inperm get  
g @threadcall replace! display gcd @int1'  
e \ CartesianIndex acot Cshort checking  
isassigned : @_MODULE__ frexp isabstv  
edException Cwchar_t LinearIndices I  
bty! eachcol kron mtime isinteractive  
hash unmark readlines stdout acos  
ns isinf IOStream inpermute!  
BigInt v hex2bytes empty IndexLinear cumsum  
tan startswith @ccall isequal rotr90 BitArra  
ngle issetequal lock isreadonly rotl90 >; ls
```

```
    _ \exp isabstract
    \ LinearIndices IndexStyle
    \n mtime isinteractive sizehint! spc
    ,readlines stdout acos delete! isone NaN16
    JStream inpermutate! falses count! complex isar
    hex2bytes empty IndexLinear cumsum ComplexF64 *_
    \rtwith @ccall isEqual rotr90 BitArray asyncmap mapto
    issetequal lock isreadonly rotl90 >: lstrip @html _str pr
    \xp promote_rule fill! findprev any Matrix CartesianIndices =
    intersect! seekstart adjoint \c titlecase chop islowercase @_.
    error seek @cmd propertynames splitdir ispath denominator @gene
    iterate flush acotd @label ENV nameof Pipe Cstring tanh deg2rad c
    e code_lowered symlink union atexit isuppercase ExponentialBackoff
    Inf16 DimensionMismatch StridedVector Libc circshift asecd VecOr
    nString seed = @r_str BitSet CompositeException RoundNearestTiesUp
    ctDict fieldoffset ! reset run log1p broadcast! filter foldl ifelse
    similar popat! tempdir ismissing UnitRange min rem2pi partialsort
    afe_read foreach nextprod acosd RoundNearest sinc Cintmax_t signific
    typeintercept chown ischardev uppercasefirst > sortperm SubString
    redirect_stdin Iterators schedule Dict accumulate ComplexF32 insert
    p! @timed walkdir IdDict \c partialsort! skip C_NULL @raw_str cumsum
    tional unescape_string chomp inperm getindex < VersionNumber isdi:
    @threadcall replace! display gcd @int128._str bytes2hex withenv api
    \ CartesianIndex acot Cshort checkindex issubnormal @_DIR_ join
    sassigned : @_MODULE_ frexp isabstract type powermod fd normpath
    dException Cwchart LinearIndices IndexStyle pipeline transpose p
    tity! eachcol kron mtime isinteractive sizehint! splitdrive TaskFa
    sh unmark readlines stdout acos delete! isone NaN16 Cint Strid
    isint IOStream inpermutate! falses count! complex isabspath o
    \nt v hex2bytes empty IndexLinear cumsum ComplexF64 * filemo
    \rtwith @ccall isEqual rotr90 BitArray asyncmap mapto
    issetequal lock isreadonly rotl90 >: lstrip @html _str pr
    promote_rule fill! findprev any Matrix CartesianIndices =
    intersect! seekstart adjoint \c titlecase chop islower
    @cmd propertynames splitdir ispath denominator
    acotd @label ENV nameof Pipe Cstring
    \link union atexit isuppercase
    \h StridedVector 1 ;
```

```
    .isinteger redirect
    evpow muladd digits pop! @timed .
    tMatrix abs2 Clonglong Irrational unescape_
    % gensym eof count_ones big @threadcall rep
    oto supertype @nospecialize \ CartesianIndex .
    imedwait sign DenseVector isassigned : @_MODULE_
    ro GC mapfldr ProcessFailedException Cwchar_t L
    readline step fieldcount empty! eachcol kron mtme
    of cosh bitrotate findfirst hash umark readlines sta
    ctime haskey Inf64 contains isinf IOStream invperm
    ring filter! RoundFromZero BigInt y hex2bytes empty In
    kgdir issorted asynemap! atan startswith @ccall isequa
    l invalid replace Cdoubule angle isseteqequal lock isreadad
    rypet wait sincos fieldtypes exp promote_rule fill! findp
    m push! @b_lstr isreadable cd intersect! seekstart adjoi
    dependency AbstractVector showerror seek @cmd propertynames
    nSort indexin NaN showable iterate flush acotd @label ENV
    inbounds rpad cosc enumerate code_lowered symlink union
    s numerator oneunit valtype Inf16 DimensionMismatch Stri
    perm truncate / SubstitutionString seqd = @r_lstr Bitset run
    TextDisplay findlast AbstractDict fieldoffset ≠ reset run
    istaskstarted sinpi Cchar similar popat! tempdir ismiss
    trounding ndims symdiff unsafe_read foreach nextprod acc
    ypejoin isreal unsafe_write typeintersect chown ischard
    mp maxintfloat im isinteger redirect_stdin Iterators sch
    ge prevpow muladd digits pop! @timed walkdir IdDict e
    tMatrix abs2 Clonglong Irrational unescape_string cho
    % gensym eof count_ones big @threadcall replace! dis
    oto supertype @nospecialize \ CartesianIndex acot C
    imedwait sign DenseVector isassigned : @_MODULE_
    ro GC mapfldr ProcessFailedException Cwchar_t L
    readline step fieldcount empty! eachcol kron mt;
    of cosh bitrotate findfirst hash umark readlin
    ctime haskey Inf64 contains isinf IOStream
    ring filter! RoundFromZero BigInt y hex2byt
    issorted asynemap! atan startswi
    replace Cdoubule angle i
    ad fieldtypes
```

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

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 1516-

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 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 1911 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. Wikipedia, The Free Encyclopedia, 4 Jul. 2020. Web. 22 Jul. 2020.]

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 2004.

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 and





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

JuliaMono Light

JuliaMono Regular

JuliaMono Medium

JuliaMono Bold

JuliaMono ExtraBold

JuliaMono Black

```
base — vim ntuple.jl - 92x43
1 # This file is a part of Julia. License is MIT: https://julia...
2
3 # `ntuple`, for constructing tuples of a given length
4
5 """
6     ntuple(f::Function, n::Integer)
7
8 Create a tuple of length `n`, computing each element as `f(i)`
9 where `i` is the index of the element.
10
11 # Examples
12 ````jl doctest
13 julia> ntuple(i → 2*i, 4)
14 (2, 4, 6, 8)
15 `````
16 """
17 @inline function ntuple(f::F, n::Integer) where F
18     # marked inline since this benefits from const
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         n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)) :
28         n == 9 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9)) :
29         n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)) :
30         _ntuple(f, n)
31     return t
32 end
33
34 function _ntuple(f, n)
35     @_noinline_meta
36     (n >= 0) || throw(ArgumentError(string("tuple length should be ≥ 0")))
37     ([f(i) for i = 1:n]...)
38 end
39
40 # inferrable ntuple (enough for bootstrapping)
NORMAL ntuple.jl julia ✘ utf-8 ✘ 11
```

```
    /  
8 Create a tuple of length `n`, computing each element as `f(i)`,  
9 where `i` is the index of the element.  
10  
11 # Examples  
12 ````jldoctest  
13 julia> ntuple(i → 2*i, 4)  
14 (2, 4, 6, 8)  
15 ````  
16 """"  
17 @inline function ntuple(f::F, n::Integer) where F  
    # marked inline since this benefits from constant propagation  
    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      n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)) :  
    28      n == 9 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9)) :  
    29      n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)) :  
    30      _ntuple(f, n)  
    31      return t  
    32 end  
    33  
34 function _ntuple(f, n)  
    35      @_noinline_meta  
    36      (n >= 0) || throw()  
    37      ([f(i) for i = 1  
    38 end  
    39  
40 # inferrable  
NORMAL ➤ ntuple
```

```
base -- vim ntuple.jl -- 92:43
1 # This file is a part of Julia. License is MIT: https://julialang.org/license
2
3 # `ntuple`, for constructing tuples of a given length
4
5 """
6     ntuple(f::Function, n::Integer)
7
8 Create a tuple of length `n`, computing each element as `f(i)`,  

9 where `i` is the index of the element.
0
1 # Examples
2 ````jldoctest
3 julia> ntuple(i → 2*i, 4)
4 (2, 4, 6, 8)
5 `````
6 """
7 @inline function ntuple(f::F, n::Integer) where F
8     # marked inline since this benefits from tail-call optimization
9     t = n == 0 ? () :
10        n == 1 ? (f(1),) :
11        n == 2 ? (f(1), f(2)) :
12        n == 3 ? (f(1), f(2), f(3)) :
13        n == 4 ? (f(1), f(2), f(3), f(4)) :
14        n == 5 ? (f(1), f(2), f(3), f(4), f(5)) :
15        n == 6 ? (f(1), f(2), f(3), f(4), f(5), f(6)) :
16        n == 7 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7)) :
17        n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)) :
18        error("ntuple: n must be less than or equal to 8")
19
20 end
```

```
1 // This file is a part of Julia. License is MIT: http://julia.readthedocs.org/licenses
2
3 # "tuple", for constructing tuples of a given length
4 #
5 # Examples
6 # julia> tuple(f1Function, m1Integer)
7 # Create a tuple of length "n", computing each element as "f(i)" where "i" is the index of the element.
8
9 n = 5
10
11 # Examples
12 # julia> tuple(f1Function, m1Integer) where n=5
13 #   marked inline since this benefits from constant propagation
14 #   f1Function(1) + f1Function(2) + f1Function(3) + f1Function(4) + f1Function(5)
15 #   m1Integer + 2*m1Integer + 3*m1Integer + 4*m1Integer + 5*m1Integer
16 # (f1, f2, f3, f4, f5)
17 # (f1, f2, f3, f4, f5)
18 # (f1, f2, f3, f4, f5)
19 # (f1, f2, f3, f4, f5)
20 # (f1, f2, f3, f4, f5)
21 # (f1, f2, f3, f4, f5)
22 # (f1, f2, f3, f4, f5)
23 # (f1, f2, f3, f4, f5)
24 # (f1, f2, f3, f4, f5)
25 # (f1, f2, f3, f4, f5)
26 # (f1, f2, f3, f4, f5)
27 # (f1, f2, f3, f4, f5)
28 # (f1, f2, f3, f4, f5)
29 # (f1, f2, f3, f4, f5)
30 # (f1, f2, f3, f4, f5)
31 # (f1, f2, f3, f4, f5)
32 end
33 return t
34 end
35
36 function _tupleof(t, n)
37     @nospecialize(n)
38     if n < 0
39         ArgumentError(string("tuple length should be ≥ 0, got ", n))
40     end
41     n == 0 ? () : _tupleof(t, n - 1) * (t())
42 end
43
44 #internal _tupleof (enough for bootstrapping)
45 @nospecialize(_tupleof)
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
686
687
688
689
689
690
691
692
693
694
695
696
696
697
698
699
699
700
701
702
703
704
705
706
706
707
708
709
709
710
711
712
713
713
714
715
715
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
```



```

# 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

```