

Version 0.036

JULIAMONO

a monospaced programming font
with reasonable Unicode support

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

JuliaMono Light

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

JuliaMono Regular

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

JuliaMono Medium

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

JuliaMono Bold

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

JuliaMono ExtraBold

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

JuliaMono Black

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

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

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

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


Ancient Greek Ἀδμηθ', ὁρᾶς γὰρ τάμα πράγμαθ', ώς ἔχει, λέξαι θέλω σοι πρὶν θανεῖν 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 lliüt rētol blavís d'onze kWh.

Czech Zvlášť zákeřný učeň s doličky běží podél zóny úlu

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

English Sphinx of black quartz, judge my vow.

Estonian Põdur Zagrebi tšellomängija-fölljetonist Ciqo külmetas kehvas garaažis

Finnish Charles Darwin jammaili Åken hevixylofonilla Qatarin yöpub Zeligissä.

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

Georgian სწრაფი ყავისფერი მელა ახტება ზარმაც ძაღლს.

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

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

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

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

IPA [ɛwɔ?.nas.do:ŋ.kʰlja]
[ɛjan.ɛzliwo.çiuɛn.χwa]

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

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

Latvian Muļķa hipiji mēģina brīvi nogaršot celofāna žņaudzējčūsku.

Lithuanian Įlinkdama fechtuotojo špaga sublykčiojusi pragrėžę apvalų arbūzą.

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

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

Polish Pchnąć w tę łódź jeża lub ósm skrzyń fig.

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

Romanian Înjurând pițigăiat, zoofobul comandă vexat whisky și tequila.

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

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

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

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

Swedish Flygande bäckasiner söka hwila på mjuka tuvor.

Turkish Pijamalı hasta yağız şoföre çabucak güvendi.

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

yotta	Y	10^{24}	1 000 000 000 000 000 000 000 000	septillion
zetta	Z	10^{21}	1 000 000 000 000 000 000 000 000	sextrillion
exa	E	10^{18}	1 000 000 000 000 000 000 000 000	quintillion
peta	P	10^{15}	1 000 000 000 000 000 000 000 000	quadrillion
tera	T	10^{12}	1 000 000 000 000	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

ꝝ

weierstrass
u+2118

ꝑ

palm branch
u+2e19

Ꝉ

swash ampersand
u+1f675

♻

recycled paper
u+267c

Ꝋ

transversal intersect
u+2adb

Ꝋ

trans Pluto
u+2bd7

ꝉ

per
u+214c

Ꝏ

division times
u+22c7

ꝑ

cada una
u+2106

ꝑ

alef
u+2135

Ꝉ

G circled
u+24bc

ꝉ

Z Notation Schema Composition
u+2a1f

Ꝋ

git PR
u+e726

ꝉ

segno
u+1d10b

ꝉ

blackboard a
u+1d552

Ꝏ

triple dagger
u+2e4b

ꝑ

editorial coronis
u+2e0e

ꝑ

coptic 800
u+102fa

Ꝉ

Pluto
u+2bd4

ꝉ

psi
u+3c8

Ꝋ

circled asterisk
u+229b

Ꝋ

hbar
u+210f

Ꝏ

cube root
u+221b

ꝉ

blackboard S
u+1d54a

ꝉ

numero
u+2116

Ꝉ

random
Ꝉ right-pointing angle trigram divergence amalgamation Or Coproduct
u+2e16

ꝉ

u+1d310

ꝉ

u+2a3f

ꝉ

Sharp S
u+1e9e

ꝉ

speech bubble
u+1f4ac

ꝉ

triple integral
u+222d

ꝉ

devangari 5
u+96b

selection

random

selection

selection

selection

selection

ꝉ

sharp S
u+1e9e

ꝉ

hryvnia
u+20b4

ꝉ

APL dot tack jot
u+234e

ꝉ

hexagram completion
u+4df6

ꝉ

fist
u+270a

ꝉ

subset with +
u+2abf

ꝉ

ace of spades
u+1f0a1

ꝉ

heavy 12pt Pinwheel Star
u+1f7d4

ꝉ

earth ground
u+23da

ꝉ

smash product
u+2a33

Ꝏ

heavy right arrow
u+1f882

ꝉ

R fraktur
u+211c

ꝉ

white upperleft square
u+25f0

ꝉ

Tifinagh Yach
u+2d5e

ꝉ

Braille 2345 (t)
u+281e

ꝉ

subscript b
u+e805

ꝉ

earth ground
u+23da

ꝉ

zNot Rel Comp
u+2a3e

Ꝏ

Mayan 14
u+1d2ee

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
ss12	=====	=====	equal join (>= 4)

+ with calt off

git PR
u+e726

```
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 end
            acc_0 = acc
            Base.Cartesian.@nexprs $N i →
                acc_{i} = op(acc_{i-1},
                    return $(Symbol(:acc_, N)))
    quote end
    else
        for i in 1:N
            acc = op(acc, i)
        end
    end
end
```



acoth cumprod keys ismount getproperty retry U @view include_dependency AbstractVector showerror seek @cmd propertynames splitdir ispath denominator @generated include_string strides AbstractIrrational i process_running Cushort & :: Inf maplices cld rounding InsertionSort indexin NaN showable iterate flush acotd @label ENV nameof Pipe Cstring tanh deg2rad cosd sortperm! eps hasmethod maximum! mark foldr RoundNearestTiesAway KeyError BitVector partialsortperm! trues @inbounds rpad cosc enumerate code_lowered symlink union atexit isuppercase ExponentialBackOff NaN64 ascii findmin! fill issubset log10 oper Complex ispunct coth fldmod1 zip put! < ~ findmax isdigit Threads numerator oneunit valtype Inf16 DimensionMismatch StridedVector Libc circshift asecd VecOrMat 2 factorial leading_zeros isetuid >> se split @s_str NaN32 IOContext asinh @macroexpand sech partialsortperm truncate v SubstitutionString secd = @r_str BitSet CompositeException RoundNearestTiesUp Docs cospi task_local_storage isfifo OrdinalR widemul uppercase cscd redisplay istaskdone isopen prevflat time TextDisplay findlast AbstractDict fieldoffset & reset run logip broadcast! filter foldl ifelse isprimitivetype SubArray lstat eachindex > last stat atanh broadcast skipmissing AbstractDisplay abs eltype istaskstarted simpi Cchar similar popat! tempdir ismissing UnitRange min rem2pi partialsort eachline Dims @async StridedArray @inline setd AbstractVecOrMat sortslices diff ! filesize length argmin hcat setrounding ndims syndiff unsafe_read foreach nextprod acosd RoundNearest sinc Cintmax_t significand | hypot sleep all! countlines yieldto is homedir pointer_from_objref relpath parent map! bytesavailable typejoin isreal unsafe_write typeintersect chown ischardev uppercasefirst |> sortperm SubString firstindex RoundingMode findall modf Csize_t isspace tryparse @doc searchsorted @views RoundDown kill splitpath Pair readybytes! AbstractChannel StepRange prevpow muladd digits pop! @timed walkdir IdDict 3 partialsort! skip C_NULL @raw_str cumsum! Rational digits! bitstring exponent fullname isprint all signbit hasfield get_zero_subnormals DenseVecOrMat BitMatrix abs2 Clonglong Irrational unescape_string chomp invperm getindex < VersionNumber isdispatchtuple / IOBuffer isblockdev @specialize circcopy! isready sqrt minimum mod1 ReentrantLock BigFloat view % gensym eof count_ones big @threadcall replace! display gcd @int128_str bytes2hex withenv append! mv vec argmax tempname ComplexF16 xor e isbits dirname π rand yield copy! iswritable reverse! println @goto supertype @nospecialize \ CartesianIndex acot Cshort checkindex issubnormal @_DIR_ joinpath merge @deprecated Array reduce reim floor cmp RawFD rem AbstractSet pairs endswith stderr pathof floatmin timewait sign DenseVector isassigned : @_MODULE_ frexp isabstracttype powermod fld normpath Cmd redirect_stdout signed Cptrdiff_t tand c range ones fd csch invmod isdirpath log lpad open htn RoundToZero GC mapfoldr ProcessFailedException Cwchart LinearIndices IndexStyle pipeline transpose parentmodule selectdim @time ^ div rstrip sort L repr count isascii pi isempty string Cfloat extrema QuickSort clamp maxintfloat im isinteger redirect_stdin Iterators schedule Dict accumulate ComplexF32 insert! isspace tryparse @doc searchsorted @views RoundDown kill splitpath Pair readybytes! AbstractChannel StepRange prevpow muladd digits pop! @timed walkdir IdDict 3 partialsort! skip C_NULL @raw_str cumsum! Rational digits! bitstring exponent fullname isprint all signbit hasfield get_zero_subnormals DenseVecOrMat BitMatrix abs2 Clonglong Irrational unescape_string chomp invperm getindex < VersionNumber isdispatchtuple / IOBuffer isblockdev @specialize circcopy! isready sqrt minimum mod1 ReentrantLock BigFloat view % gensym eof count_ones big @threadcall replace! display gcd @int128_str bytes2hex withenv append! mv vec argmax tempname ComplexF16 xor e isbits dirname π rand yield copy! iswritable reverse! println @goto supertype @nospecialize \ CartesianIndex acot Cshort checkindex issubnormal @_DIR_ joinpath merge @deprecated Array reduce reim floor cmp RawFD rem AbstractSet pairs endswith stderr pathof floatmin timewait sign DenseVector isassigned : @_MODULE_ frexp isabstracttype powermod fld normpath Cmd redirect_stdout signed Cptrdiff_t tand c range ones fd csch invmod isdirpath log lpad open htn RoundToZero GC mapfoldr ProcessFailedException Cwchart LinearIndices IndexStyle pipeline transpose parentmodule selectdim @time ^ div rstrip sort L read ismutable ignorestatus findmax! redirect_stderr ARGS notify readline step fieldcount empty! eachcol kron mtime isinteractive sizehint! splitdrive TaskFailedException download @eval CapturedException permutedims! inv values Missing size isnan StackTraces show sizeof cosh bitrotate findfirst hash unmark readlines stdout aco delete! isone NaN16 Cint StridedMatrix nextpow isfile acsch fma cp @text_str & sincos prod hvcat VERSION StringIndexError allunique lastindex ctime haskey Inf64 contains isinf IOStream invpermute! falses count! complex isabspath occurin one pointer PartialQuickSort Set merge! fl IndexCartesian ispow2 isbitstype convert @isdefined sec escape_string filter! RoundFromZero BigInt v hex2bytes empty IndexLinear cumsum ComplexF64 * filemode get! fetch popfirst! @MIME_str @_dot_ isdir @big_str float basename searchsortedfirst circshift! unsafe_wrap pkgdir issorted asyncmap! atan startswith @ccall isequal rotr90 BitArray asyncmap mapfoldl process_exited getkey unlock issocket Regex isyndiff! cumprod! exit imag cos displayable evalfile isstructtype invalid replace Cdouble angle issetequal lock isreadonly rotl90 >: lstrip @html_str promote lowercasestart floatmax asech nextind round c splitext precision // mergewith! e unsafe_trunc codeunits reinterpret wait sincosd fieldtypes exp promote_rule fill! findprev any Matrix CartesianIndices => mergewith cbrt gethostname stdin ntuple Csize_t prod! Cuint @simd isapprox @warn ception Cwchart LinearIndices IndexStyle pipeline transpose parentmodule selectdim @time ^ div rstrip sort L read ismutable ignorestatus findmax! redirect_stdin chardev uppercase step fieldcount empty! eachcol kron mtime isinteractive sizehint! splitdrive TaskFailedException download @eval CapturedException permutedims! inv values Missing size isnan & Iterators schedule Dict accumulate bitrotate findfirst hash unmark readlines stdout aco truncate v SubstitutionString siedMatrix nextpow isfile acsch fma cp @text_str & sincos prod hvcat VERSION StringIndexE<dir IdDict 3 partialsort! skip C_Nu_Inf64 contains isinf IOStream invpermute! TextDisplay findlast AbstractDict field in one pointer PartialQuickSort Set merge! fl IndexCartesian ispow2 isbitstype convert escape_string chomp invperm getindex < Ve_InfFromZero BigInt v hex2bytes empty je istaskstarted simpi Cchar similar popat! ! fetch popfirst! @MIME_str @_dot_ isdir @big_str float basename searchsortedfirst circshift! unsafe_wrap pkgdir issorted asyncmap! atan startswith @ccall isat setrounding ndims syndiff unsafe_read foreach exited getkey unlock issocket Regex isyndiff! cumprod! exit imag cos d; tesianIndex acot Cshort checkindex issubnormal @_double angle issetequal lock able typejoin isreal unsafe_write typeintersect chown isfirst floatmax asech nextind round c splitext precision // mergewith! signed : @_MODULE_ frexp isabstracttype powermod fld fieldtypes exp promote_rule fill! findprev any Matrix CartesianIndices => mergewith cbrt gethostname stdin ntuple Csize_t prod! Cuint @simd isapprox @warn ception Cwchart LinearIndices IndexStyle pipeline transpose parentmodule selectdim @time ^ div rstrip sort L read ismutable ignorestatus findmax! redirect_stdin step fieldcount empty! eachcol kron mtime istype SubArray lstat eachindex > last stat atanh broadcast skipmissing AbstractDisplay abs eltype istaskstarted simpi Cchar similar popat! tempdir ismissing UnitRange min rem2pi partialsort eachline Dims @async StridedArray @inline setd AbstractVecOrMat sortslices diff ! filesize length argmin hcat setrounding ndims syndiff unsafe read foreach nextprod acosd RoundNearest sinc Cintmax_t significand | hypot sleep all! countlines yieldto is homedir pointer_from_objref relpath parent map! bytesavailable typejoin isreal unsafe_write typeintersect chown ischardev uppercasefirst |> sortperm SubString firstindex RoundingMode findall modf Csize_t isspace tryparse @doc searchsorted @views RoundDown kill splitpath Pair readybytes! AbstractChannel StepRange prevpow muladd digits pop! @timed walkdir IdDict 3 partialsort! skip C_NULL @raw_str cumsum! Rational digits! bitstring exponent fullname isprint all signbit hasfield get_zero_subnormals DenseVecOrMat BitMatrix abs2 Clonglong Irrational unescape_string chomp invperm getindex < VersionNumber isdispatchtuple / IOBuffer isblockdev @specialize circcopy! isready sqrt minimum mod1 ReentrantLock BigFloat view % gensym eof count_ones big @threadcall replace! display gcd @int128_str bytes2hex withenv append! mv vec argmax tempname ComplexF16 xor e isbits dirname π rand yield copy! iswritable reverse! println @goto supertype @nospecialize \ CartesianIndex acot Cshort checkindex issubnormal @_DIR_ joinpath merge @deprecated Array reduce reim floor cmp RawFD rem AbstractSet pairs endswith stderr pathof floatmin timewait sign DenseVector isassigned : @_MODULE_ frexp isabstracttype powermod fld normpath Cmd field Dict field ARGS notify readline step fieldcount empty! eachcol kron mtime istype SubArray lstat eachindex > last stat atanh broadcast skipmissing AbstractDisplay abs eltype istaskstarted simpi Cchar similar popat! tempdir ismissing UnitRange min rem2pi partialsort eachline Dims @async StridedArray @inline setd AbstractVecOrMat sortslices diff ! filesize length argmin hcat setrounding ndims syndiff unsafe read foreach nextprod acosd RoundNearest sinc Cintmax_t significand | hypot sleep all! countlines yieldto is names in Julia Base lay istaskdone isopen preclnt TextDisplay findlast AbstractDict fieldoffset & reset run logip broadcast! filter foldl ifelse isprimitivetype SubArray lstat eachindex > last stat atanh broadcast skipmissing AbstractDisplay abs eltype istaskstarted simpi Cchar similar popat! tempdir ismissing UnitRange min rem2pi partialsort eachline Dims @async StridedArray @inline setd AbstractVecOrMat sortslices diff ! filesize length argmin hcat setrounding ndims syndiff unsafe read foreach nextprod acosd RoundNearest sinc Cintmax_t significand | hypot sleep all! countlines yieldto is

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

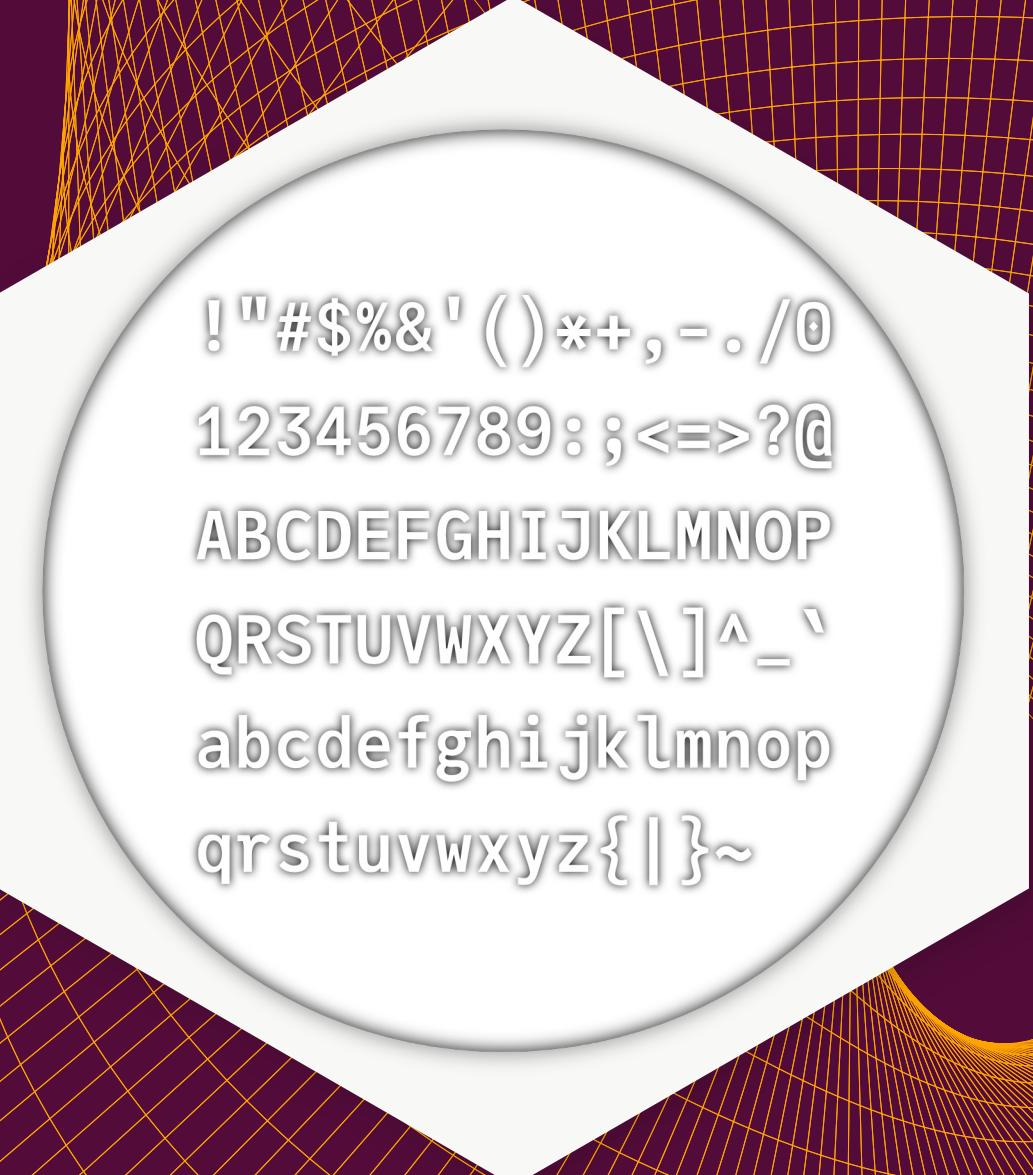
painted the work from 1513. According to Vasari, "after he had lingered over it four years, left it unfinished". In 1516, Leonardo was invited by King Francis I to work at the Clos Lucé near the Château d'Amboise; it is believed that he took the *Mona Lisa* with him and continued to work on it after 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 == 10 ? (†(1), †(2), †(3), †(4), †(5), †(6), †(7), †(8), †(9), †(10)) :
```

42 ntuple

2
3
4
5
6

```
retret  
nd  
unction tunction  
@_newlineut  
(n >= 0) || throw(  
([f(i) for i =where { f
```

```
43 tuple(f,  
44   e-mem); f(1);  
45 tuple(f,  
46   e-mem); f(1);  
47 # @inline f  
48 N::In  
49   (N >=  
50     if @g_<  
51       q<  
52     else  
53       T<  
54     end  
55   end  
56   ) @inline f  
57   M = L<  
58   N = -N<  
59   +2 ~-1 -0 ntuple(j
```

```
fun1 N::Int  
  (N  
    if , ::Val  
    , ::Val.  
    function  
      quote int  
        := 0) ||  
        generate quote  
        @ne,  
        @nc;  
    end  
  else;  
end  
@nexprs
```

```
+ 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
32         _ntuple(f, n)
33     return t
34
35 function _ntuple(f, n)
36     @_noinline_meta
37     (n >= 0) || throw(ArgumentError(string("tuple length sho
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 sho
50     if @_generated
51         quote
52             @_nexprs $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
```

```
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 = 1:N
64         t[i] = val
65     end
66     t
67 end
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
389
390
391
392
393
394
395
396
397
398
399
400
```

```
le t
= 1:N
inline TuncT
length{N} = 1:N
    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),
    n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7),
    _ntuple(f, n)
  return t
end

function _ntuple(f, n)
  @_noinline_meta
  (n >= 0) || throw(ArgumentError(string("tu
  ([f(i) for i = 1:n]...),
end

# inferable ntuple (enough for bootstrap)
ntuple(f, ::Val{}) = ()
_ntuple(f, ::Val{1}) = (@_inline_m
_ntuple(f, ::Val{2}) = (@_inline_m
_ntuple(f, ::Val{3}) = (@_inline_m
```

```

1 n == 1 ? (f(1)) :
2 n == 2 ? (f(1), f(2)) :
3 n == 3 ? (f(1), f(2), f(3)) :
4 n == 4 ? (f(1), f(2), f(3), f(4)) :
5 n == 5 ? (f(1), f(2), f(3), f(4), f(5)) :
6 n == 6 ? (f(1), f(2), f(3), f(4), f(5), f(6)) :
7 n == 7 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7)) :
8 n == 8 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8)) :
9 n == 9 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9)) :
10 n == 10 ? (f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)) :
11 _ntuple(f, n)
12
13 return t
14
15
16 function _ntuple(f, n)
17     @_noinline_meta
18     (n >= 0) || throw(ArgumentError(string("tuple length should be ≥ "
19     ([f(i) for i = 1:n]...),
20
21 end
22
23
24 # inferable ntuple (enough for bootstrapping)
25 ntuple(f, ::Val{}) = ()
26 ntuple(f, ::Val{1}) = (@_inline_meta; (f(1,)))
27 ntuple(f, ::Val{2}) = (@_inline_meta; (f(1), f(2)))
28 ntuple(f, ::Val{3}) = (@_inline_meta; (f(1), f(2), f(3)))
29
30 #
31
32 # @inline function ntuple(f::F, ::Val{N}) where {
33 #     N::Int
34 #     (N >= 0) || throw(ArgumentError(string('
35 #         if @generated
36 #             quote
37 #                 @exprs $N i → t_i = f(
38 #                     @call $N tuple t
39 #                 end
40 #             else
41 #                 Tuple(f(i) for i = 1
42 #             end
43 #         end
44 #     end
45 #     @inline function f!
46 #         M = length(t)
47 #         N = _N::Int
48 #         if N < M
49 #             error("tuple length must be ≤ "
50 #                   string(N, " but got ", M))
51 #         end
52 #         if N > M
53 #             error("tuple length must be ≥ "
54 #                   string(M, " but got ", N))
55 #         end
56 #         if N == M
57 #             return t
58 #         end
59 #         if N == 1
60 #             return f()
61 #         end
62 #         if N == 2
63 #             return (f(), f())
64 #         end
65 #         if N == 3
66 #             return (f(), f(), f())
67 #         end
68 #         if N == 4
69 #             return (f(), f(), f(), f())
70 #         end
71 #         if N == 5
72 #             return (f(), f(), f(), f(), f())
73 #         end
74 #         if N == 6
75 #             return (f(), f(), f(), f(), f(), f())
76 #         end
77 #         if N == 7
78 #             return (f(), f(), f(), f(), f(), f(), f())
79 #         end
80 #         if N == 8
81 #             return (f(), f(), f(), f(), f(), f(), f(), f())
82 #         end
83 #         if N == 9
84 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
85 #         end
86 #         if N == 10
87 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
88 #         end
89 #         if N > 10
90 #             error("tuple length must be ≤ 10")
91 #         end
92 #         if N < 1
93 #             error("tuple length must be ≥ 1")
94 #         end
95 #         if N > M
96 #             error("tuple length must be ≥ "
97 #                   string(M, " but got ", N))
98 #         end
99 #         if N < M
100 #             error("tuple length must be ≤ "
101 #                   string(N, " but got ", M))
102 #         end
103 #         if N == M
104 #             return t
105 #         end
106 #         if N == 1
107 #             return f()
108 #         end
109 #         if N == 2
110 #             return (f(), f())
111 #         end
112 #         if N == 3
113 #             return (f(), f(), f())
114 #         end
115 #         if N == 4
116 #             return (f(), f(), f(), f())
117 #         end
118 #         if N == 5
119 #             return (f(), f(), f(), f(), f())
120 #         end
121 #         if N == 6
122 #             return (f(), f(), f(), f(), f(), f())
123 #         end
124 #         if N == 7
125 #             return (f(), f(), f(), f(), f(), f(), f())
126 #         end
127 #         if N == 8
128 #             return (f(), f(), f(), f(), f(), f(), f(), f())
129 #         end
130 #         if N == 9
131 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
132 #         end
133 #         if N == 10
134 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
135 #         end
136 #         if N > 10
137 #             error("tuple length must be ≤ 10")
138 #         end
139 #         if N < 1
140 #             error("tuple length must be ≥ 1")
141 #         end
142 #         if N > M
143 #             error("tuple length must be ≥ "
144 #                   string(M, " but got ", N))
145 #         end
146 #         if N < M
147 #             error("tuple length must be ≤ "
148 #                   string(N, " but got ", M))
149 #         end
150 #         if N == M
151 #             return t
152 #         end
153 #         if N == 1
154 #             return f()
155 #         end
156 #         if N == 2
157 #             return (f(), f())
158 #         end
159 #         if N == 3
160 #             return (f(), f(), f())
161 #         end
162 #         if N == 4
163 #             return (f(), f(), f(), f())
164 #         end
165 #         if N == 5
166 #             return (f(), f(), f(), f(), f())
167 #         end
168 #         if N == 6
169 #             return (f(), f(), f(), f(), f(), f())
170 #         end
171 #         if N == 7
172 #             return (f(), f(), f(), f(), f(), f(), f())
173 #         end
174 #         if N == 8
175 #             return (f(), f(), f(), f(), f(), f(), f(), f())
176 #         end
177 #         if N == 9
178 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
179 #         end
180 #         if N == 10
181 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
182 #         end
183 #         if N > 10
184 #             error("tuple length must be ≤ 10")
185 #         end
186 #         if N < 1
187 #             error("tuple length must be ≥ 1")
188 #         end
189 #         if N > M
190 #             error("tuple length must be ≥ "
191 #                   string(M, " but got ", N))
192 #         end
193 #         if N < M
194 #             error("tuple length must be ≤ "
195 #                   string(N, " but got ", M))
196 #         end
197 #         if N == M
198 #             return t
199 #         end
200 #         if N == 1
201 #             return f()
202 #         end
203 #         if N == 2
204 #             return (f(), f())
205 #         end
206 #         if N == 3
207 #             return (f(), f(), f())
208 #         end
209 #         if N == 4
210 #             return (f(), f(), f(), f())
211 #         end
212 #         if N == 5
213 #             return (f(), f(), f(), f(), f())
214 #         end
215 #         if N == 6
216 #             return (f(), f(), f(), f(), f(), f())
217 #         end
218 #         if N == 7
219 #             return (f(), f(), f(), f(), f(), f(), f())
220 #         end
221 #         if N == 8
222 #             return (f(), f(), f(), f(), f(), f(), f(), f())
223 #         end
224 #         if N == 9
225 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
226 #         end
227 #         if N == 10
228 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
229 #         end
230 #         if N > 10
231 #             error("tuple length must be ≤ 10")
232 #         end
233 #         if N < 1
234 #             error("tuple length must be ≥ 1")
235 #         end
236 #         if N > M
237 #             error("tuple length must be ≥ "
238 #                   string(M, " but got ", N))
239 #         end
240 #         if N < M
241 #             error("tuple length must be ≤ "
242 #                   string(N, " but got ", M))
243 #         end
244 #         if N == M
245 #             return t
246 #         end
247 #         if N == 1
248 #             return f()
249 #         end
250 #         if N == 2
251 #             return (f(), f())
252 #         end
253 #         if N == 3
254 #             return (f(), f(), f())
255 #         end
256 #         if N == 4
257 #             return (f(), f(), f(), f())
258 #         end
259 #         if N == 5
260 #             return (f(), f(), f(), f(), f())
261 #         end
262 #         if N == 6
263 #             return (f(), f(), f(), f(), f(), f())
264 #         end
265 #         if N == 7
266 #             return (f(), f(), f(), f(), f(), f(), f())
267 #         end
268 #         if N == 8
269 #             return (f(), f(), f(), f(), f(), f(), f(), f())
270 #         end
271 #         if N == 9
272 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
273 #         end
274 #         if N == 10
275 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
276 #         end
277 #         if N > 10
278 #             error("tuple length must be ≤ 10")
279 #         end
280 #         if N < 1
281 #             error("tuple length must be ≥ 1")
282 #         end
283 #         if N > M
284 #             error("tuple length must be ≥ "
285 #                   string(M, " but got ", N))
286 #         end
287 #         if N < M
288 #             error("tuple length must be ≤ "
289 #                   string(N, " but got ", M))
290 #         end
291 #         if N == M
292 #             return t
293 #         end
294 #         if N == 1
295 #             return f()
296 #         end
297 #         if N == 2
298 #             return (f(), f())
299 #         end
300 #         if N == 3
301 #             return (f(), f(), f())
302 #         end
303 #         if N == 4
304 #             return (f(), f(), f(), f())
305 #         end
306 #         if N == 5
307 #             return (f(), f(), f(), f(), f())
308 #         end
309 #         if N == 6
310 #             return (f(), f(), f(), f(), f(), f())
311 #         end
312 #         if N == 7
313 #             return (f(), f(), f(), f(), f(), f(), f())
314 #         end
315 #         if N == 8
316 #             return (f(), f(), f(), f(), f(), f(), f(), f())
317 #         end
318 #         if N == 9
319 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
320 #         end
321 #         if N == 10
322 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
323 #         end
324 #         if N > 10
325 #             error("tuple length must be ≤ 10")
326 #         end
327 #         if N < 1
328 #             error("tuple length must be ≥ 1")
329 #         end
330 #         if N > M
331 #             error("tuple length must be ≥ "
332 #                   string(M, " but got ", N))
333 #         end
334 #         if N < M
335 #             error("tuple length must be ≤ "
336 #                   string(N, " but got ", M))
337 #         end
338 #         if N == M
339 #             return t
340 #         end
341 #         if N == 1
342 #             return f()
343 #         end
344 #         if N == 2
345 #             return (f(), f())
346 #         end
347 #         if N == 3
348 #             return (f(), f(), f())
349 #         end
350 #         if N == 4
351 #             return (f(), f(), f(), f())
352 #         end
353 #         if N == 5
354 #             return (f(), f(), f(), f(), f())
355 #         end
356 #         if N == 6
357 #             return (f(), f(), f(), f(), f(), f())
358 #         end
359 #         if N == 7
360 #             return (f(), f(), f(), f(), f(), f(), f())
361 #         end
362 #         if N == 8
363 #             return (f(), f(), f(), f(), f(), f(), f(), f())
364 #         end
365 #         if N == 9
366 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
367 #         end
368 #         if N == 10
369 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
370 #         end
371 #         if N > 10
372 #             error("tuple length must be ≤ 10")
373 #         end
374 #         if N < 1
375 #             error("tuple length must be ≥ 1")
376 #         end
377 #         if N > M
378 #             error("tuple length must be ≥ "
379 #                   string(M, " but got ", N))
380 #         end
381 #         if N < M
382 #             error("tuple length must be ≤ "
383 #                   string(N, " but got ", M))
384 #         end
385 #         if N == M
386 #             return t
387 #         end
388 #         if N == 1
389 #             return f()
390 #         end
391 #         if N == 2
392 #             return (f(), f())
393 #         end
394 #         if N == 3
395 #             return (f(), f(), f())
396 #         end
397 #         if N == 4
398 #             return (f(), f(), f(), f())
399 #         end
400 #         if N == 5
401 #             return (f(), f(), f(), f(), f())
402 #         end
403 #         if N == 6
404 #             return (f(), f(), f(), f(), f(), f())
405 #         end
406 #         if N == 7
407 #             return (f(), f(), f(), f(), f(), f(), f())
408 #         end
409 #         if N == 8
410 #             return (f(), f(), f(), f(), f(), f(), f(), f())
411 #         end
412 #         if N == 9
413 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
414 #         end
415 #         if N == 10
416 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
417 #         end
418 #         if N > 10
419 #             error("tuple length must be ≤ 10")
420 #         end
421 #         if N < 1
422 #             error("tuple length must be ≥ 1")
423 #         end
424 #         if N > M
425 #             error("tuple length must be ≥ "
426 #                   string(M, " but got ", N))
427 #         end
428 #         if N < M
429 #             error("tuple length must be ≤ "
430 #                   string(N, " but got ", M))
431 #         end
432 #         if N == M
433 #             return t
434 #         end
435 #         if N == 1
436 #             return f()
437 #         end
438 #         if N == 2
439 #             return (f(), f())
440 #         end
441 #         if N == 3
442 #             return (f(), f(), f())
443 #         end
444 #         if N == 4
445 #             return (f(), f(), f(), f())
446 #         end
447 #         if N == 5
448 #             return (f(), f(), f(), f(), f())
449 #         end
450 #         if N == 6
451 #             return (f(), f(), f(), f(), f(), f())
452 #         end
453 #         if N == 7
454 #             return (f(), f(), f(), f(), f(), f(), f())
455 #         end
456 #         if N == 8
457 #             return (f(), f(), f(), f(), f(), f(), f(), f())
458 #         end
459 #         if N == 9
460 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
461 #         end
462 #         if N == 10
463 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
464 #         end
465 #         if N > 10
466 #             error("tuple length must be ≤ 10")
467 #         end
468 #         if N < 1
469 #             error("tuple length must be ≥ 1")
470 #         end
471 #         if N > M
472 #             error("tuple length must be ≥ "
473 #                   string(M, " but got ", N))
474 #         end
475 #         if N < M
476 #             error("tuple length must be ≤ "
477 #                   string(N, " but got ", M))
478 #         end
479 #         if N == M
480 #             return t
481 #         end
482 #         if N == 1
483 #             return f()
484 #         end
485 #         if N == 2
486 #             return (f(), f())
487 #         end
488 #         if N == 3
489 #             return (f(), f(), f())
490 #         end
491 #         if N == 4
492 #             return (f(), f(), f(), f())
493 #         end
494 #         if N == 5
495 #             return (f(), f(), f(), f(), f())
496 #         end
497 #         if N == 6
498 #             return (f(), f(), f(), f(), f(), f())
499 #         end
500 #         if N == 7
501 #             return (f(), f(), f(), f(), f(), f(), f())
502 #         end
503 #         if N == 8
504 #             return (f(), f(), f(), f(), f(), f(), f(), f())
505 #         end
506 #         if N == 9
507 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
508 #         end
509 #         if N == 10
510 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
511 #         end
512 #         if N > 10
513 #             error("tuple length must be ≤ 10")
514 #         end
515 #         if N < 1
516 #             error("tuple length must be ≥ 1")
517 #         end
518 #         if N > M
519 #             error("tuple length must be ≥ "
520 #                   string(M, " but got ", N))
521 #         end
522 #         if N < M
523 #             error("tuple length must be ≤ "
524 #                   string(N, " but got ", M))
525 #         end
526 #         if N == M
527 #             return t
528 #         end
529 #         if N == 1
530 #             return f()
531 #         end
532 #         if N == 2
533 #             return (f(), f())
534 #         end
535 #         if N == 3
536 #             return (f(), f(), f())
537 #         end
538 #         if N == 4
539 #             return (f(), f(), f(), f())
540 #         end
541 #         if N == 5
542 #             return (f(), f(), f(), f(), f())
543 #         end
544 #         if N == 6
545 #             return (f(), f(), f(), f(), f(), f())
546 #         end
547 #         if N == 7
548 #             return (f(), f(), f(), f(), f(), f(), f())
549 #         end
550 #         if N == 8
551 #             return (f(), f(), f(), f(), f(), f(), f(), f())
552 #         end
553 #         if N == 9
554 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
555 #         end
556 #         if N == 10
557 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
558 #         end
559 #         if N > 10
560 #             error("tuple length must be ≤ 10")
561 #         end
562 #         if N < 1
563 #             error("tuple length must be ≥ 1")
564 #         end
565 #         if N > M
566 #             error("tuple length must be ≥ "
567 #                   string(M, " but got ", N))
568 #         end
569 #         if N < M
570 #             error("tuple length must be ≤ "
571 #                   string(N, " but got ", M))
572 #         end
573 #         if N == M
574 #             return t
575 #         end
576 #         if N == 1
577 #             return f()
578 #         end
579 #         if N == 2
580 #             return (f(), f())
581 #         end
582 #         if N == 3
583 #             return (f(), f(), f())
584 #         end
585 #         if N == 4
586 #             return (f(), f(), f(), f())
587 #         end
588 #         if N == 5
589 #             return (f(), f(), f(), f(), f())
590 #         end
591 #         if N == 6
592 #             return (f(), f(), f(), f(), f(), f())
593 #         end
594 #         if N == 7
595 #             return (f(), f(), f(), f(), f(), f(), f())
596 #         end
597 #         if N == 8
598 #             return (f(), f(), f(), f(), f(), f(), f(), f())
599 #         end
600 #         if N == 9
601 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
602 #         end
603 #         if N == 10
604 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
605 #         end
606 #         if N > 10
607 #             error("tuple length must be ≤ 10")
608 #         end
609 #         if N < 1
610 #             error("tuple length must be ≥ 1")
611 #         end
612 #         if N > M
613 #             error("tuple length must be ≥ "
614 #                   string(M, " but got ", N))
615 #         end
616 #         if N < M
617 #             error("tuple length must be ≤ "
618 #                   string(N, " but got ", M))
619 #         end
620 #         if N == M
621 #             return t
622 #         end
623 #         if N == 1
624 #             return f()
625 #         end
626 #         if N == 2
627 #             return (f(), f())
628 #         end
629 #         if N == 3
630 #             return (f(), f(), f())
631 #         end
632 #         if N == 4
633 #             return (f(), f(), f(), f())
634 #         end
635 #         if N == 5
636 #             return (f(), f(), f(), f(), f())
637 #         end
638 #         if N == 6
639 #             return (f(), f(), f(), f(), f(), f())
640 #         end
641 #         if N == 7
642 #             return (f(), f(), f(), f(), f(), f(), f())
643 #         end
644 #         if N == 8
645 #             return (f(), f(), f(), f(), f(), f(), f(), f())
646 #         end
647 #         if N == 9
648 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
649 #         end
650 #         if N == 10
651 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
652 #         end
653 #         if N > 10
654 #             error("tuple length must be ≤ 10")
655 #         end
656 #         if N < 1
657 #             error("tuple length must be ≥ 1")
658 #         end
659 #         if N > M
660 #             error("tuple length must be ≥ "
661 #                   string(M, " but got ", N))
662 #         end
663 #         if N < M
664 #             error("tuple length must be ≤ "
665 #                   string(N, " but got ", M))
666 #         end
667 #         if N == M
668 #             return t
669 #         end
670 #         if N == 1
671 #             return f()
672 #         end
673 #         if N == 2
674 #             return (f(), f())
675 #         end
676 #         if N == 3
677 #             return (f(), f(), f())
678 #         end
679 #         if N == 4
680 #             return (f(), f(), f(), f())
681 #         end
682 #         if N == 5
683 #             return (f(), f(), f(), f(), f())
684 #         end
685 #         if N == 6
686 #             return (f(), f(), f(), f(), f(), f())
687 #         end
688 #         if N == 7
689 #             return (f(), f(), f(), f(), f(), f(), f())
690 #         end
691 #         if N == 8
692 #             return (f(), f(), f(), f(), f(), f(), f(), f())
693 #         end
694 #         if N == 9
695 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
696 #         end
697 #         if N == 10
698 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
699 #         end
700 #         if N > 10
701 #             error("tuple length must be ≤ 10")
702 #         end
703 #         if N < 1
704 #             error("tuple length must be ≥ 1")
705 #         end
706 #         if N > M
707 #             error("tuple length must be ≥ "
708 #                   string(M, " but got ", N))
709 #         end
710 #         if N < M
711 #             error("tuple length must be ≤ "
712 #                   string(N, " but got ", M))
713 #         end
714 #         if N == M
715 #             return t
716 #         end
717 #         if N == 1
718 #             return f()
719 #         end
720 #         if N == 2
721 #             return (f(), f())
722 #         end
723 #         if N == 3
724 #             return (f(), f(), f())
725 #         end
726 #         if N == 4
727 #             return (f(), f(), f(), f())
728 #         end
729 #         if N == 5
730 #             return (f(), f(), f(), f(), f())
731 #         end
732 #         if N == 6
733 #             return (f(), f(), f(), f(), f(), f())
734 #         end
735 #         if N == 7
736 #             return (f(), f(), f(), f(), f(), f(), f())
737 #         end
738 #         if N == 8
739 #             return (f(), f(), f(), f(), f(), f(), f(), f())
740 #         end
741 #         if N == 9
742 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
743 #         end
744 #         if N == 10
745 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
746 #         end
747 #         if N > 10
748 #             error("tuple length must be ≤ 10")
749 #         end
750 #         if N < 1
751 #             error("tuple length must be ≥ 1")
752 #         end
753 #         if N > M
754 #             error("tuple length must be ≥ "
755 #                   string(M, " but got ", N))
756 #         end
757 #         if N < M
758 #             error("tuple length must be ≤ "
759 #                   string(N, " but got ", M))
760 #         end
761 #         if N == M
762 #             return t
763 #         end
764 #         if N == 1
765 #             return f()
766 #         end
767 #         if N == 2
768 #             return (f(), f())
769 #         end
770 #         if N == 3
771 #             return (f(), f(), f())
772 #         end
773 #         if N == 4
774 #             return (f(), f(), f(), f())
775 #         end
776 #         if N == 5
777 #             return (f(), f(), f(), f(), f())
778 #         end
779 #         if N == 6
780 #             return (f(), f(), f(), f(), f(), f())
781 #         end
782 #         if N == 7
783 #             return (f(), f(), f(), f(), f(), f(), f())
784 #         end
785 #         if N == 8
786 #             return (f(), f(), f(), f(), f(), f(), f(), f())
787 #         end
788 #         if N == 9
789 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
790 #         end
791 #         if N == 10
792 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
793 #         end
794 #         if N > 10
795 #             error("tuple length must be ≤ 10")
796 #         end
797 #         if N < 1
798 #             error("tuple length must be ≥ 1")
799 #         end
800 #         if N > M
801 #             error("tuple length must be ≥ "
802 #                   string(M, " but got ", N))
803 #         end
804 #         if N < M
805 #             error("tuple length must be ≤ "
806 #                   string(N, " but got ", M))
807 #         end
808 #         if N == M
809 #             return t
810 #         end
811 #         if N == 1
812 #             return f()
813 #         end
814 #         if N == 2
815 #             return (f(), f())
816 #         end
817 #         if N == 3
818 #             return (f(), f(), f())
819 #         end
820 #         if N == 4
821 #             return (f(), f(), f(), f())
822 #         end
823 #         if N == 5
824 #             return (f(), f(), f(), f(), f())
825 #         end
826 #         if N == 6
827 #             return (f(), f(), f(), f(), f(), f())
828 #         end
829 #         if N == 7
830 #             return (f(), f(), f(), f(), f(), f(), f())
831 #         end
832 #         if N == 8
833 #             return (f(), f(), f(), f(), f(), f(), f(), f())
834 #         end
835 #         if N == 9
836 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
837 #         end
838 #         if N == 10
839 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
840 #         end
841 #         if N > 10
842 #             error("tuple length must be ≤ 10")
843 #         end
844 #         if N < 1
845 #             error("tuple length must be ≥ 1")
846 #         end
847 #         if N > M
848 #             error("tuple length must be ≥ "
849 #                   string(M, " but got ", N))
850 #         end
851 #         if N < M
852 #             error("tuple length must be ≤ "
853 #                   string(N, " but got ", M))
854 #         end
855 #         if N == M
856 #             return t
857 #         end
858 #         if N == 1
859 #             return f()
860 #         end
861 #         if N == 2
862 #             return (f(), f())
863 #         end
864 #         if N == 3
865 #             return (f(), f(), f())
866 #         end
867 #         if N == 4
868 #             return (f(), f(), f(), f())
869 #         end
870 #         if N == 5
871 #             return (f(), f(), f(), f(), f())
872 #         end
873 #         if N == 6
874 #             return (f(), f(), f(), f(), f(), f())
875 #         end
876 #         if N == 7
877 #             return (f(), f(), f(), f(), f(), f(), f())
878 #         end
879 #         if N == 8
880 #             return (f(), f(), f(), f(), f(), f(), f(), f())
881 #         end
882 #         if N == 9
883 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
884 #         end
885 #         if N == 10
886 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
887 #         end
888 #         if N > 10
889 #             error("tuple length must be ≤ 10")
890 #         end
891 #         if N < 1
892 #             error("tuple length must be ≥ 1")
893 #         end
894 #         if N > M
895 #             error("tuple length must be ≥ "
896 #                   string(M, " but got ", N))
897 #         end
898 #         if N < M
899 #             error("tuple length must be ≤ "
900 #                   string(N, " but got ", M))
901 #         end
902 #         if N == M
903 #             return t
904 #         end
905 #         if N == 1
906 #             return f()
907 #         end
908 #         if N == 2
909 #             return (f(), f())
910 #         end
911 #         if N == 3
912 #             return (f(), f(), f())
913 #         end
914 #         if N == 4
915 #             return (f(), f(), f(), f())
916 #         end
917 #         if N == 5
918 #             return (f(), f(), f(), f(), f())
919 #         end
920 #         if N == 6
921 #             return (f(), f(), f(), f(), f(), f())
922 #         end
923 #         if N == 7
924 #             return (f(), f(), f(), f(), f(), f(), f())
925 #         end
926 #         if N == 8
927 #             return (f(), f(), f(), f(), f(), f(), f(), f())
928 #         end
929 #         if N == 9
930 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
931 #         end
932 #         if N == 10
933 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
934 #         end
935 #         if N > 10
936 #             error("tuple length must be ≤ 10")
937 #         end
938 #         if N < 1
939 #             error("tuple length must be ≥ 1")
940 #         end
941 #         if N > M
942 #             error("tuple length must be ≥ "
943 #                   string(M, " but got ", N))
944 #         end
945 #         if N < M
946 #             error("tuple length must be ≤ "
947 #                   string(N, " but got ", M))
948 #         end
949 #         if N == M
950 #             return t
951 #         end
952 #         if N == 1
953 #             return f()
954 #         end
955 #         if N == 2
956 #             return (f(), f())
957 #         end
958 #         if N == 3
959 #             return (f(), f(), f())
960 #         end
961 #         if N == 4
962 #             return (f(), f(), f(), f())
963 #         end
964 #         if N == 5
965 #             return (f(), f(), f(), f(), f())
966 #         end
967 #         if N == 6
968 #             return (f(), f(), f(), f(), f(), f())
969 #         end
970 #         if N == 7
971 #             return (f(), f(), f(), f(), f(), f(), f())
972 #         end
973 #         if N == 8
974 #             return (f(), f(), f(), f(), f(), f(), f(), f())
975 #         end
976 #         if N == 9
977 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
978 #         end
979 #         if N == 10
980 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
981 #         end
982 #         if N > 10
983 #             error("tuple length must be ≤ 10")
984 #         end
985 #         if N < 1
986 #             error("tuple length must be ≥ 1")
987 #         end
988 #         if N > M
989 #             error("tuple length must be ≥ "
990 #                   string(M, " but got ", N))
991 #         end
992 #         if N < M
993 #             error("tuple length must be ≤ "
994 #                   string(N, " but got ", M))
995 #         end
996 #         if N == M
997 #             return t
998 #         end
999 #         if N == 1
1000 #             return f()
1001 #         end
1002 #         if N == 2
1003 #             return (f(), f())
1004 #         end
1005 #         if N == 3
1006 #             return (f(), f(), f())
1007 #         end
1008 #         if N == 4
1009 #             return (f(), f(), f(), f())
1010 #         end
1011 #         if N == 5
1012 #             return (f(), f(), f(), f(), f())
1013 #         end
1014 #         if N == 6
1015 #             return (f(), f(), f(), f(), f(), f())
1016 #         end
1017 #         if N == 7
1018 #             return (f(), f(), f(), f(), f(), f(), f())
1019 #         end
1020 #         if N == 8
1021 #             return (f(), f(), f(), f(), f(), f(), f(), f())
1022 #         end
1023 #         if N == 9
1024 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f())
1025 #         end
1026 #         if N == 10
1027 #             return (f(), f(), f(), f(), f(), f(), f(), f(), f(), f())
1028 #         end
1029 #         if N > 10
1030 #             error("tuple length must be ≤ 10")
1031 #         end
1032 #         if N < 1
1033 #             error("tuple length must be ≥ 1")
1034 #         end
1035 #         if N > M
1036 #             error("tuple length must be ≥ "
1037 #                   string(M, " but got ", N))
1038 #         end
1039 #         if N < M
1040 #             error("tuple length must be ≤ "
1041 #                   string(N, " but got ", M))
1042 #         end
1043 #         if N == M
1044 #             return t
1045 #         end
1046 #         if N == 1
1047 #             return f()
1048 #         end
1049 #         if N == 2

```

```

# 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.

Graphics made with Julia and Luxor.jl

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

```