

cat.face.laugh	cat.face.shock	cat.face.smile
cat.face.smirk	chain	chains
chair	champagne	chart.bar
chart.up	chart.down	chart.yen.up
checkmark.heavy	checkmark.box	cheese
cherries	chess	chestnut
chicken	chicken.baby	chicken.baby.egg
chicken.baby.head	chicken.leg	chicken.male
child	chipmunk	chocolate
chopsticks	church	church.love
cigarette	cigarette.not	circle.black
circle.blue	circle.brown	circle.green
circle.orange	circle.purple	circle.white
circle.red	circle.yellow	circle.stroked
circus	city	city.dusk
city.night	city.sunset	cl
clamp	clapperboard	climbing
clip	clipboard	clips
clock.one	clock.one.thirty	clock.two
clock.two.thirty	clock.three	clock.three.thirty
clock.four	clock.four.thirty	clock.five

⌚clock.five.thirty	⌚clock.six	⌚clock.six.thirty
⌚clock.seven	⌚clock.seven.thirty	⌚clock.eight
⌚clock.eight.thirty	⌚clock.nine	⌚clock.nine.thirty
⌚clock.ten	⌚clock.ten.thirty	⌚clock.eleven
⌚clock.eleven.thirty	⌚clock.twelve	⌚clock.twelve.thirty
⏰clock.alarm	⏰clock.old	⏰clock.timer
☁cloud	☁cloud.dust	☁cloud.rain
☁cloud.snow	☁cloud.storm	☁cloud.sun
☁cloud.sun.hidden	☁cloud.sun.rain	☁cloud.thunder
🧥coat	🧥coat.lab	roach
🍸cocktail.martini	🍸cocktail.tropical	🥥coconut
☕coffee	⚰coffin	coin
☄comet	🧭compass	💻computer
🖱computermouse	🎉confetti	🚧construction
🎮controller	🍪cookie	🥠cookie.fortune
饪cooking	cool	©copyright
珊瑚礁coral	🌽corn	🛋couch
👫couple	🐄cow	🐮cow.face
🦀crab	🏗crane	🖍crayon
🏏cricket	🏏cricketbat	🐊crocodile
🥐croissant	✖crossmark	✖crossmark.box

👑crown	▢crutch	⌚crystal
🥒cucumber	🥤cup.straw	🧁cupcake
🎳curling	🍛curry	🍮custard
⚠️customs	🍴cutlery	🌀cyclone
🕺dancing.man	💃dancing.woman	🐰dancing.women.bunny
🎯darts	~~dash.wave.double	🦌deer
🏜desert	🕵️detective	◆diamond.blue
◆diamond.blue.small	◆diamond.orange	◆diamond.orange.small
❖diamond.dot	🎲die	🦕dino.pod
🦕dino.rex	💿disc.cd	📀disc.dvd
💿disc.mini	▢discoball	🤿diving
▢dodo	🐕dog	🐶dog.face
🐩dog.guide	🐩dog.poodle	\$ dollar
🐬dolphin	🍩donut	🚪door
🕊dove.peace	🐉dragon	🐲dragon.face
👗dress	👘dress.kimono	👗dress.sari
💧drop	pletsdrops	🥁drum
▢drum.big	🦆duck	🥟dumpling
🦅eagle	👂ear	👂ear.aid
🥚egg	⓯eighteen.not	🐘elephant
▢elevator	🧙elf	✉️@email

!excl	!excl.white	!!excl.double
!?excl.quest	💥explosion	滅 extinguisher
👁eye	👀eyes	☺face.grin
😡face.angry	😠face.angry.red	😱face.anguish
😱face.astonish	🤕face.bandage	🤗face.beam
😐face.blank	🤡face.clown	🥶face.cold
😢face.concern	😎face.cool	幪 face.cover
🤠face.cowboy	😭face.cry	😈face.devil.smile
😈face.devil.frown	□face.diagonal	👤face.disguise
😵face.dizzy	□face.dotted	⬇face.down
😅face.down.sweat	🥳face.drool	🤯face.explode
😊face.eyeroll	😊face.friendly	😱face.fear
😓face.fear.sweat	🤒face.fever	😳face.flush
❗face.frown	❗face.frown.slight	❗face.frust
🤡face.goofy	光环 face.halo	😊face.happy
❤face.heart	💖face.hearts	🥵face.heat
🤗face.hug	⌚face.inv	😊face.joy
😘face.kiss	😘face.kiss.smile	😘face.kiss.heart
😘face.kiss.blush	👅face.lick	🤥face.lie
😷face.mask	😉face.meh	👤face.melt
💰face.money	👓face.monocle	🤮face.nausea

 face.nerd	 face.neutral	 face.open
 face.party	 face.peek	 face.plead
 face.relief	 face.rofl	 face.sad
 face.salute	 face.shock	 face.shush
 face.skeptic	 face.sleep	 face.sleepy
 face.smile	 face.smile.slight	 face.smile.sweat
 face.smile.tear	 face.smirk	 face.sneeze
 face.speak.not	 face.squint	 face.stars
 face.straight	 face.suffer	 face.surprise
 face.symbols	 face.tear	 face.tear.relief
 face.tear.withheld	 face.teeth	 face.think
 face.tired	 face.tongue	 face.tongue.squint
 face.tongue.wink	 face.triumph	 face.unhappy
 face.vomit	 face.weary	 face.wink
 face.woozy	 face.worry	 face.wow
 face.yawn	 face.zip	 factory
 fairy	 faith.christ	 faith.dharma
 faith.islam	 faith.judaism	 faith.menorah
 faith.om	 faith.orthodox	 faith.peace
 faith.star.dot	 faith.worship	 faith.yinyang
 falafel	 family	 fax

□ feather	🤱 feeding.breast	🤺 fencing
🎡 ferriswheel	🗃 filebox	📁 filedividers
🎞 film	⌚ finger.r	⌚ finger.l
👉 finger.t	☝ finger.t.alt	👈 finger.b
▢ finger.front	👆 finger.m	👆 fingers.cross
▢ fingers.pinch	▢ fingers.snap	🔥 fire
🧨 firecracker	🚒 fireengine	🎆 fireworks
🐟 fish	🐠 fish.tropical	🎣 fishing
✊ fist.front	👉 fist.r	👉 fist.l
✊ fist.raised	🚩 flag.black	🚩 flag.white
🏁 flag.goal	🚩 flag.golf	🚩 flag.red
🎌 flags.jp.crossed	🦢 flamingo	🔦 flashlight
▢ flatbread	🇫 fleur	💾 floppy
🌺 flower.hibiscus	▢ flower.lotus	🌺 flower.pink
🌹 flower.rose	☀️ flower.sun	🌷 flower.tulip
🌸 flower.white	🥀 flower.wilted	🌸 flower.yellow
▢ fly	🌫 fog	📁 folder
📁 folder.open	▢ fondue	🦶 foot
⚽ football	🏉 football.am	💵 forex
⛲ fountain	🦊 fox	🆓 free
🍟 fries	🥎 frisbee	🐸 frog.face

fuelpump	gachi	garlic
gear	gem	genie
ghost	giraffe	girl
glass.clink	glass.milk	glass.pour
glass.tumbler	glasses	glasses.sun
globe.am	globe.as.au	globe.eu.af
globe.meridian	gloves	go
goal	goat	goggles
golfing	gorilla	grapes
guard.man	guitar	gymnastics
haircut	hammer	hammer.pick
hammer.wrench	hamsa	hamster.face
hand.raised	hand.raised.alt	hand.r
hand.l	hand.t	hand.b
hand.ok	hand.call	hand.love
hand.part	hand.peace	hand.pinch
hand.rock	hand.splay	hand.wave
hand.write	handbag	handball
handholding.man.ma	handholding.woman.ma	handholding.woman.woma
n	n	n
hands.folded	hands.palms	hands.clap

□hands.heart	₩hands.open	₩hands.raised
握手 hands.shake	#hash	礼帽 hat.ribbon
🎩 hat.top	🎧 headphone	❤️ heart
心脏箭心 heart.arrow	❤️ heart.beat	🖤 heart.black
心脏蓝心 heart.blue	📦 heart.box	💔 heart.broken
心脏棕心 heart.brown	DOUBLE HEART heart.double	❗ heart.excl
心脏绿心 heart.green	❤️ heart.grow	🧡 heart.orange
心脏紫心 heart.purple	□ heart.real	💔 heart.revolve
丝带心 heart.ribbon	💖 heart.spark	🤍 heart.white
心脏黄心 heart.yellow	🦔 hedgehog	🚁 helicopter
螺旋 helix	⛑ helmet.cross	⛑ helmet.military
秘 hi	🦓 hippo	🏒 hockey
洞 hole	🍯 honey	🇨🇳 hongbao
钩子 hook	📇 horn.postal	🐎 horse
旋转木马 horse.carousel	🐴 horse.face	🏇 horse.race
医院 hospital	🌭 hotdog	🏨 hotel
酒店爱 hotel.love	♨hotspring	⏳ hourglass
沙漏流 hourglass.flow	🏡 house	🏚 house.derelict
花园房子 house.garden	🏡 house.multiple	💯 hundred
小屋 hut	🧊 ice	🍦 icecream
剃冰激凌 icecream.shaved	🍦 icecream.soft	🏒 icehockey

 id	 info	 izakaya
 jar	 jeans	 jigsaw
 joystick	 juggling	 juice
 ka	 kaaba	 kachi
 kadomatsu	 kangaroo	 kara
 kebab	 key	 key.old
 keyboard	 kiss	 kissmark
 kite	 kiwi	 knife
 knife.dagger	 knot	 koala
 koinobori	 koko	 kon
 label	 lacrosse	 ladder
 lamp.diya	 laptop	 leaf.clover.three
 leaf.clover.four	 leaf.fall	 leaf.herb
 leaf.maple	 leaf.wind	 leftluggage
 leg	 leg.mech	 lemon
 leopard	 letter.love	 liberty
 lightbulb	 lightning	 lion
 lipstick	 litter	 litter.not
 lizard	 llama	 lobster
 lock	 lock.key	 lock.open
 lock.pen	 lollipop	 lotion

 luggage	 lungs	 mage
 magnet	 magnify.r	 magnify.l
 mahjong.dragon.red	 mail	 mail.arrow
 mailbox.closed.empty	 mailbox.closed.full	 mailbox.open.empty
y		
 mailbox.open.full	 mammoth	 man
 man.box	 man.crown	 man.guapimao
 man.levitate	 man.old	 man.pregnant
 man.turban	 man.tuxedo	 mango
 map.world	 map.jp	 martialarts
 masks	 mate	 matryoshka
 meat	 meat.bone	 medal.first
 medal.second	 medal.third	 medal.sports
 medal.military	 megaphone	 megaphone.simple
 melon	 merperson	 metro
 microbe	 microphone	 microphone.studio
 microscope	 milkyway	 mirror
 mixer	 money.bag	 money.dollar
 money.euro	 money.pound	 money.yen
 money.wings	 monkey	 monkey.face
 monkey.hear.not	 monkey.see.not	 monkey.speak.not

🌙 moon.crescent	🌕 moon.full	🌖 moon.full.face
🌑 moon.new	🌒 moon.new.face	🌓 moon.wane.one
🌔 moon.wane.two	🌕 moon.wane.three.face	🌗 moon.wane.three
🌓 moon.wax.one	🌔 moon.wax.two	🌓 moon.wax.two.face
🌔 moon.wax.three	🎓 mortarboard	🕋 mosque
🐞 mosquito	🏍 motorcycle	🛣️ motorway
🏔️ mountain	🏔️ mountain.fuji	🏔️ mountain.snow
🏔️ mountain.sunrise	鼫 mouse	鼫 mouse.face
◻ mousetrap	👄 mouth	◻ mouth.bite
🔱 moyai	🈪 muryo	🏛️ museum
🍄 mushroom	🎼 musicalscore	💅 nails.polish
📛 namebadge	🧿 nazar	👔 necktie
◻ needle	◻ nest.empty	◻ nest.eggs
🆕 new	📰 newspaper	🗞️ newspaper.rolled
🆖 ng	🥋 ningyo	🥋 ninja
🚫 noentry	👃 nose	📓 notebook
📔 notebook.deco	📝 notepad	♪ notes
♫ notes.triple	🔢 numbers	ଓ o
🐙 octopus	🏢 office	שמן oil
🆗 ok	◻ olive	👹 oni
🥔 onion	🦍 orangutan	🦦 otter



owl



ox



oyster



package



paella



page



page.curl



page.pencil



pager



pages.tabs



painting



palette



pancakes



panda



parachute



spark



parking



parrot



partalteration



party



peach



peacock



peanuts



pear



pedestrian



pedestrian.not



pen.ball



pen.fountain



pencil



penguin



pepper



pepper.hot



person



person.angry



person.beard



person.blonde



person.bow



person.crown



person.deaf



person.facepalm



person.frown



person.hijab



person.kneel



person.lotus



person.massage



person.no



person.ok



person.old



person.pregnant



person.raise



person.sassy



person.shrug



person.stand



person.steam



petri



phone



phone.arrow



phone.classic



phone.not



phone.off



phone.receiver



phone.signal



phone.vibrate

🎹 piano	⛏ pick	🥧 pie
🐖 pig	🐗 pig.face	🐽 pig.nose
💊 pill	📌 pin	📍 pin.round
▫️ pinata	🍍 pineapple	🏓 pingpong
🔫 pistol	🍕 pizza	▢ placard
🛸 planet	▢ plant	🩹 plaster
🍽 plate.cutlery	⬇️ playback.down	⏏️ playback.eject
▶️ playback.forward	⏸️ playback.pause	⏺️ playback.record
🔁 playback.repeat	🔂 playback.repeat.once	🔁 playback.repeat.v
⏮️ playback.restart	⏪ playback.rewind	🔀 playback.shuffle
▶️▶️ playback.skip	⏹️ playback.stop	▶️▶️ playback.toggle
⬆️ playback.up	🎴 playingcard.flower	🃏 playingcard.joker
▢ plunger	👮 policeofficer	💩 poo
🍿 popcorn	🌐 post.eu	〒 post.jp
📮 postbox	🥔 potato	🥔 potato.sweet
👝 pouch	🔌 powerplug	🎁 present
🥨 pretzel	🖨 printer	👣 prints.foot
🐾 prints.paw	🚫 prohibited	📽 projector
🎃 pumpkin.lantern	👛 purse	❓ quest
❔ quest.white	🐰 rabbit	🐰 rabbit.face
🦭 raccoon	📻 radio	☢️ radioactive

 railway	 rainbow	 ram
 rat	 razor	 receipt
 recycling	 reg	 restroom
 rhino	 ribbon	 ribbon.remind
 rice	 rice.cracker	 rice.ear
 rice.onigiri	 ring	 ringbuoy
 robot	 rock	 rocket
 rollercoaster	 rosette	 rugby
 ruler	 ruler.triangle	 running
 sa	 safetypin	 safetyvest
 sake	 salad	 salt
 sandwich	 santa.man	 santa.woman
 satdish	 satellite	 saw
 saxophone	 scales	 scarf
 school	 scissors	 scooter
 scooter.motor	 scorpion	 screwdriver
 scroll	 seal	 seat
 seedling	 shark	 sheep
 shell.spiral	 shield	 shin
 ship	 ship.cruise	 ship.ferry
 shirt.sports	 shirt.t	 shoe

 shoe.ballet	 shoe.flat	 shoe.heel
 shoe.hike	 shoe.ice	 shoe.roller
 shoe.sandal.heel	 shoe.ski	 shoe.sneaker
 shoe.tall	 shoe.thong	 shopping
 shorts	 shoshinsha	 shower
 shrimp	 shrimp.fried	 shrine
祝 shuku	 sign.crossing	 sign.stop
 silhouette	 silhouette.double	 silhouette.hug
 silhouette.speak	 siren	 skateboard
 skewer.dango	 skewer.oden	 skiing
 skull	 skull.bones	 skunk
 sled	 slide	 slider
 sloth	 slots	 snail
 snake	 snowboarding	 snowflake
 snowman	 snowman.snow	 soap
 socks	 softball	 sos
 soup	 spaghetti	 sparkle.box
 sparkler	 sparkles	 speaker
 speaker.not	 speaker.wave	 speaker.waves
 spider	 spiderweb	 spinach
 sponge	 spoon	 square.black

▪ square.black.tiny	▪ square.black.small	▪ square.black.medium
□square.white	▪ square.white.tiny	▫ square.white.small
▫square.white.medium	▨square.blue	▨square.brown
▨square.green	▨square.orange	▨square.purple
▨square.red	▨square.yellow	🦑squid
🏟stadium	★star	⭐star.arc
✳star.box	✳star.glow	✳star.shoot
医用听诊器	🛍store.big	🛍store.small
苺strawberry	♣suit.club	♦suit.diamond
♥suit.heart	♠suit.spade	☀sun
☁sun.cloud	☀sun.face	🌅sunrise
🦸superhero	🦸supervillain	🏄surfing
🍣sushi	🦆swan	🏊swimming
👙swimsuit	⚔swords	㉔symbols
🕧synagogue	💉syringe	🌮taco
🍱takeout	▢tamale	🎋tanabata
귤tangerine	⠇tap	🚫tap.not
🚕taxi	🚕taxi.front	🍵teacup
▢teapot	🧸teddy	🔭telescope
🏯temple	🔟ten	👺tengu
🎾tennis	⛺tent	🧪testtube

🌡 thermometer	🧵 thread	👍 thumb.up
👎 thumb.down	🎫 ticket.event	🎫 ticket.travel
🐯 tiger	🐯 tiger.face	™ tm
🚻 toilet	🧻 toiletpaper	🉐 toku
🍅 tomato	⚰ tombstone	👅 tongue
.toolbox	🦷 tooth	🦷 toothbrush
🌪 tornado	🗼 tower.tokyo	🖱 trackball
🚜 tractor	🚦 trafficlight.v	🚦 trafficlight.h
🚂 train	🚃 train.car	🚡 train.light
🚇 train.metro	🚞 train.mono	🚞 train.mountain
🚄 train.speed	🚄 train.speed.bullet	🚂 train.steam
🚉 train.stop	🚞 train.suspend	🚊 train.tram
🚞 train.tram.car	⚧ transgender	📥 tray.inbox
✉️ tray.mail	📤 tray.outbox	🌳 tree.deciduous
🌲 tree.evergreen	🌴 tree.palm	🎄 tree.xmas
► triangle.r	◀ triangle.l	▲ triangle.t
▼ triangle.b	▲ triangle.t.red	▼ triangle.b.red
🔱 trident	Troll	🏆 trophy
🚚 truck	🚚 truck.trailer	🎺 trumpet
-tsukimi	🦃 turkey	🐢 turtle
📺 tv	🛸 ufo	☂️ umbrella.open

傘umbrella.closed	傘umbrella.rain	傘umbrella.sun
独角兽unicorn	未知unknown	↑up
urn	vampire	小提琴violin
火山volcano	排球volleyball	VSvs
华夫饼waffle	魔杖wand	⚠warning
手表watch	停止watch.stop	西瓜watermelon
水球waterpolo	海浪wave	WCwc
举重weightlifting	鲸鱼whale	鲸鱼喷泉whale.spout
轮子wheel	轮椅wheelchair	轮椅箱wheelchair.box
轮椅马达wheelchair.motor	风wind	风铃windchime
窗户window	酒wine	狼wolf
女人woman	女人盒子woman.box	女人皇冠woman.crown
老女人woman.old	孕妇woman.pregnant	木头wood
蠕虫worm	扳手wrench	摔跤wrestling
X光片xray	纱线yarn	yo-yo
悠悠球yoyo	指yubi	柔术yuryo
斑马zebra	星座水瓶座zodiac.aquarius	星座白羊座zodiac.aries
星座巨蟹座zodiac.cancer	星座海王星zodiac.capri	星座双子座zodiac.gemini
星座狮子座zodiac.leo	星座天秤座zodiac.libra	星座蛇夫座zodiac.ophi
星座双鱼座zodiac.pisces	星座射手座zodiac.sagit	星座天蝎座zodiac.scorpio
星座金牛座zodiac.taurus	星座处女座zodiac.virgo	僵尸zombie

`zzzzzz`

3.9.3 Symbol Type

A Unicode symbol.

Typst defines common symbols so that they can easily be written with standard keyboards. The symbols are defined in modules, from which they can be accessed using [field access notation](#):

- General symbols are defined in the [sym module](#)
- Emoji are defined in the [emoji module](#)

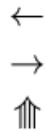
Moreover, you can define custom symbols with this type's constructor function.

```
#sym.arrow.r \
#sym.gt.eq.not \
$gt.eq.not$ \
#emoji.face.halo
```



Many symbols have different variants, which can be selected by appending the modifiers with dot notation. The order of the modifiers is not relevant. Visit the documentation pages of the symbol modules and click on a symbol to see its available variants.

```
$arrow.l$ \
$arrow.r$ \
$arrow.t.quad$
```



Constructor

Create a custom symbol with modifiers.

```
symbol(
  ...strarray
) -> symbol
  #let envelope = symbol(
    "?",
    ("stamped", "?"),
    ("stamped.pen", "?"),
    ("lightning", "?"),
    ("fly", "?"),
  )

#envelope
#envelope.stamped
#envelope.stamped.pen
#envelope.lightning
#envelope.fly
```



variants

str or array

RequiredPositional

Variadic

The variants of the symbol.

Can be a just a string consisting of a single character for the modifierless variant or an array with two strings specifying the modifiers and the symbol. Individual modifiers should be separated by dots. When displaying a symbol, Typst selects

the first from the variants that have all attached modifiers and the minimum number of other modifiers.

3.10 Layout

3.10.1 Align

Aligns content horizontally and vertically.

Example

Let's start with centering our content horizontally:

```
#set page(height: 120pt)
#set align(center)
```

```
Centered text, a sight to see \
In perfect balance, visually \
Not left nor right, it stands alone \
A work of art, a visual throne
```

```
Centered text, a sight to see
In perfect balance, visually
Not left nor right, it stands alone
A work of art, a visual throne
```

To center something vertically, use *horizon* alignment:

```
#set page(height: 120pt)
#set align(horizon)
```

```
Vertically centered, \
the stage had entered, \
a new paragraph.
```

Vertically centered,
the stage had entered,
a new paragraph.

Combining alignments

You can combine two alignments with the + operator. Let's also only apply this to one piece of content by using the function form instead of a set rule:

```
#set page(height: 120pt)
Though left in the beginning ...

#align(right + bottom) [
    ... they were right in the end, \
    and with addition had gotten, \
    the paragraph to the bottom!
]
```

Though left in the beginning ...

... they were right in the end,
and with addition had gotten,
the paragraph to the bottom!

Nested alignment

You can use varying alignments for layout containers and the elements within them. This way, you can create intricate layouts:

```
#align(center, block[
    #set align(left)
    Though centered together \
```

```
alone \
we \
are \
left.

])
```

Though centered together
 alone
 we
 are
 left.

Alignment within the same line

The `align` function performs block-level alignment and thus always interrupts the current paragraph. To have different alignment for parts of the same line, you should use [fractional spacing](#) instead:

```
Start #h(1fr) End
```

Start End

Parameters

```
align(
  alignment,
  content,
) -> content
alignment
  alignment
Positional
Settable
```

The [alignment](#) along both axes.

Default: `start + top`

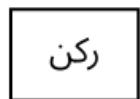
[View example](#)

```
#set page(height: 6cm)
#set text(lang: "ar")
```

مثال

```
#align(
    end + horizon,
    rect(inset: 12pt) [ركن]
)
```

مثال



body

content

RequiredPositional

The content to align.

3.10.2 Alignment

Where to [align](#) something along an axis.

Possible values are:

- **start**: Aligns at the [start](#) of the [text direction](#).
- **end**: Aligns at the [end](#) of the [text direction](#).
- **left**: Align at the left.
- **center**: Aligns in the middle, horizontally.
- **right**: Aligns at the right.

- `top`: Aligns at the top.
- `horizon`: Aligns in the middle, vertically.
- `bottom`: Align at the bottom.

These values are available globally and also in the alignment type's scope, so you can write either of the following two:

```
#align(center) [Hi]
#align(alignment.center) [Hi]
```

Hi

Hi

2D alignments

To align along both axes at the same time, add the two alignments using the `+` operator. For example, `top + right` aligns the content to the top right corner.

```
#set page(height: 3cm)
#align(center + bottom) [Hi]
```

Hi

Fields

The `x` and `y` fields hold the alignment's horizontal and vertical components, respectively (as yet another `alignment`). They may be `none`.

```
#{top + right}.x \
#left.x \
```

```
#left.y (none)
```

```
right
left
(none)
```

Definitions

`axis`

The axis this alignment belongs to.

- "horizontal" for start, left, center, right, and end
- "vertical" for top, horizon, and bottom
- none for 2-dimensional alignments

```
self.axis()
#left.axis() \
#bottom.axis()
```

```
horizontal
vertical
```

`inv`

The inverse alignment.

```
self.inv() -> alignment
#top.inv() \
#left.inv() \
#center.inv() \
#(left + bottom).inv()
```

```
bottom
right
center
right + top
```

3.10.3 Angle

An angle describing a rotation.

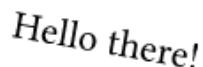
Typst supports the following angular units:

- Degrees: `180deg`

- Radians: `3.14rad`

Example

```
#rotate(10deg) [Hello there!]
```



Hello there!

Definitions

`rad`

Converts this angle to radians.

```
self.rad() -> float
```

`deg`

Converts this angle to degrees.

```
self.deg() -> float
```

3.10.4 Block

A block-level container.

Such a container can be used to separate content, size it, and give it a

background or border.

Examples

With a block, you can give a background to content while still allowing it to

break across multiple pages.

```
#set page(height: 100pt)
#block(
    fill: luma(230),
    inset: 8pt,
    radius: 4pt,
    lorem(30),
)
```

 Lorem ipsum dolor sit amet, consectetur
 adipiscing elit, sed do eiusmod tempor
 incididunt ut labore et dolore magna
 aliquam quaerat voluptatem. Ut enim

 aeque doleamus animo, cum corpore
 dolemus, fieri.

Blocks are also useful to force elements that would otherwise be inline to become block-level, especially when writing show rules.

```
#show heading: it => it.body
```

= Blockless

More text.

```
#show heading: it => block(it.body)
```

= Blocky

More text.

Blockless More text.

Blocky

More text.

Parameters

```
block(
    width: autorelative,
    height: autorelativefraction,
    breakable: bool,
    fill: nonecolorgradientpattern,
    stroke: nonelengthcolorgradientstrokepatterndictionary,
    radius: relativedictionary,
    inset: relativedictionary,
    outset: relativedictionary,
    spacing: relativefraction,
    above: autorelativefraction,
    below: autorelativefraction,
    clip: bool,
    sticky: bool,
    nonecontent,
) -> content
width
auto or relative
```

Settable

The block's width.

Default: `auto`

[View example](#)

```
#set align(center)
#block(
    width: 60%,
    inset: 8pt,
    fill: silver,
    lorem(10),
)
```

Lorem ipsum dolor sit
 amet, consectetur
 adipiscing elit, sed do.

height

auto or relative or fraction

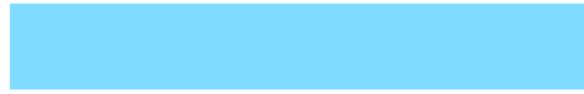
Settable

The block's height. When the height is larger than the remaining space on a page and breakable is true, the block will continue on the next page with the remaining height.

Default: auto

[View example](#)

```
#set page(height: 80pt)
#set align(center)
#block(
    width: 80%,
    height: 150%,
    fill: aqua,
)
```



breakable

bool

Settable

Whether the block can be broken and continue on the next page.

Default: `true`

[View example](#)

```
#set page(height: 80pt)
The following block will
jump to its own page.

#block(
    breakable: false,
    lorem(15),
)
```

The following block will jump to its own page.

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore.

`fill`

`none` or `color` or `gradient` or `pattern`

Settable

The block's background color. See the [rectangle's documentation](#) for more details.

Default: `none`

`stroke`

`none` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

Settable

The block's border color. See the [rectangle's documentation](#) for more details.

Default: (:

radius

[relative](#) or [dictionary](#)

Settable

How much to round the block's corners. See the [rectangle's documentation](#) for more details.

Default: (:

inset

[relative](#) or [dictionary](#)

Settable

How much to pad the block's content. See the [box's documentation](#) for more details.

Default: (:

outset

[relative](#) or [dictionary](#)

Settable

How much to expand the block's size without affecting the layout. See the [box's documentation](#) for more details.

Default: (:

spacing

[relative](#) or [fraction](#)

Settable

The spacing around the block. When `auto`, inherits the paragraph [spacing](#).

For two adjacent blocks, the larger of the first block's `above` and the second block's `below` spacing wins. Moreover, block spacing takes precedence over paragraph [spacing](#).

Note that this is only a shorthand to set `above` and `below` to the same value. Since the values for `above` and `below` might differ, a [context](#) block only provides access to `block.above` and `block.below`, not to `block.spacing` directly.

This property can be used in combination with a show rule to adjust the spacing around arbitrary block-level elements.

Default: `1.2em`

[View example](#)

```
#set align(center)
#show math.equation: set block(above: 8pt, below: 16pt)
```

This sum of `x` and `y`:

$$\$ x + y = z \$$$

A second paragraph.

This sum of x and y :

$$x + y = z$$

A second paragraph.

above

`auto` or `relative` or `fraction`

Settable

The spacing between this block and its predecessor.

Default: `auto`

below

`auto` or `relative` or `fraction`

Settable

The spacing between this block and its successor.

Default: `auto`

`clip`

`bool`

Settable

Whether to clip the content inside the block.

Clipping is useful when the block's content is larger than the block itself, as any

content that exceeds the block's bounds will be hidden.

Default: `false`

[View example](#)

```
#block(
    width: 50pt,
    height: 50pt,
    clip: true,
    image("tiger.jpg", width: 100pt, height: 100pt)
)
```



`sticky`

`bool`

Settable

Whether this block must stick to the following one, with no break in

between.

This is, by default, set on heading blocks to prevent orphaned headings at the bottom of the page.

Default: `false`

[View example](#)

```
// Disable stickiness of headings.  
#show heading: set block(sticky: false)  
#lorem(20)  
  
= Chapter  
#lorem(10)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. quaeat.

Chapter

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

body

`none` or `content`

Positional

Settable

The contents of the block.

Default: `none`

3.10.5 Box

An inline-level container that sizes content.

All elements except inline math, text, and boxes are block-level and cannot occur inside of a paragraph. The box function can be used to integrate such elements into a paragraph. Boxes take the size of their contents by default but can also be sized explicitly.

Example

```
Refer to the docs
#box(
    height: 9pt,
    image("docs.svg")
)
for more information.
```

Refer to the docs [🔗](#) for more information.

Parameters

```
box(
    width: autorelativefraction,
    height: autorelative,
    baseline: relative,
    fill: nonecolorgradientpattern,
    stroke: nonelengthcolorgradientstrokepatterndictionary,
    radius: relativedictionary,
    inset: relativedictionary,
    outset: relativedictionary,
    clip: bool,
    nonecontent,
) -> content
width
auto or relative or fraction
```

Settable

The width of the box.

Boxes can have [fractional](#) widths, as the example below demonstrates.

Note: Currently, only boxes and only their widths might be fractionally sized within paragraphs. Support for fractionally sized images, shapes, and more might be added in the future.

Default: `auto`

[View example](#)

```
Line in #box(width: 1fr, line(length: 100%)) between.
```

Line in _____ between.

height

`auto` or `relative`

Settable

The height of the box.

Default: `auto`

baseline

`relative`

Settable

An amount to shift the box's baseline by.

Default: `0% + 0pt`

[View example](#)

```
Image: #box(baseline: 40%, image("tiger.jpg", width: 2cm)).
```



fill

`none` or `color` or `gradient` or `pattern`

Settable

The box's background color. See the [rectangle's documentation](#) for more details.

Default: `none`

`stroke`

`none` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

Settable

The box's border color. See the [rectangle's documentation](#) for more details.

Default: `(::)`

`radius`

`relative` or `dictionary`

Settable

How much to round the box's corners. See the [rectangle's documentation](#) for more details.

Default: `(::)`

`inset`

`relative` or `dictionary`

Settable

How much to pad the box's content.

Note: When the box contains text, its exact size depends on the current [text edges](#).

Default: `(::)`

[View example](#)

`#rect (inset: 0pt) [Tight]`

ligh*t***outset**relative or dictionary***Settable***

How much to expand the box's size without affecting the layout.

This is useful to prevent padding from affecting line layout. For a generalized version of the example below, see the documentation for the [raw text's block parameter](#).

Default: `(::)`

[View example](#)

An inline

```
#box(
    fill: luma(235),
    inset: (x: 3pt, y: 0pt),
    outset: (y: 3pt),
    radius: 2pt,
) [rectangle].
```

An inline `rectangle`.

clipbool***Settable***

Whether to clip the content inside the box.

Clipping is useful when the box's content is larger than the box itself, as any content that exceeds the box's bounds will be hidden.

Default: `false`

[View example](#)

```
#box(
    width: 50pt,
    height: 50pt,
```

```
clip: true,
image("tiger.jpg", width: 100pt, height: 100pt)
)
```

**body**

none or content

Positional

Settable

The contents of the box.

Default: none

3.10.6 Column Break

Forces a column break.

The function will behave like a [page break](#) when used in a single column layout or the last column on a page. Otherwise, content after the column break will be placed in the next column.

Example

```
#set page(columns: 2)
Preliminary findings from our
ongoing research project have
revealed a hitherto unknown
phenomenon of extraordinary
significance.

#colbreak()
Through rigorous experimentation
and analysis, we have discovered
a hitherto uncharacterized process
that defies our current
```

understanding of the fundamental laws of nature.

Preliminary findings from our ongoing research project have revealed a hitherto unknown phenomenon of extraordinary significance.	Through rigorous experimentation and analysis, we have discovered a hitherto uncharacterized process that defies our current understanding of the fundamental laws of nature.
---	---

Parameters

```
colbreak(
weak: bool
) -> content
weak
bool
Settable
```

If `true`, the column break is skipped if the current column is already empty.

Default: `false`

3.10.7 Columns

Separates a region into multiple equally sized columns.

The `column` function lets you separate the interior of any container into

multiple columns. It will currently not balance the height of the columns.

Instead, the columns will take up the height of their container or the remaining

height on the page. Support for balanced columns is planned for the future.

Page-level columns

If you need to insert columns across your whole document, use the `page` function's [columns parameter](#) instead. This will create the columns directly at the page-level rather than wrapping all of your content in a layout container. As a result, things like [pagebreaks](#), [footnotes](#), and [line numbers](#) will continue to work as expected. For more information, also read the [relevant part of the page setup guide](#).

Breaking out of columns

To temporarily break out of columns (e.g. for a paper's title), use parent-scoped floating placement:

```
#set page(columns: 2, height: 150pt)

#place(
    top + center,
    scope: "parent",
    float: true,
    text(1.4em, weight: "bold") [
        My document
    ],
)
#lorem(40)
```

My document

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.	voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod
---	---

Parameters

```
columns (
    int,
    gutter: relative,
    content,
) -> content
count
int
Positional
Settable
```

The number of columns.

Default: 2

```
gutter
relative
Settable
```

The size of the gutter space between each column.

Default: 4% + 0pt

```
body
content
RequiredPositional
```

The content that should be laid out into the columns.

3.10.8 Direction

The four directions into which content can be laid out.

Possible values are:

- ltr: Left to right.
- rtl: Right to left.
- ttb: Top to bottom.
- btt: Bottom to top.

These values are available globally and also in the direction type's scope, so you can write either of the following two:

```
#stack(dir: rtl) [A] [B] [C]
#stack(dir: direction.rtl) [A] [B] [C]
```

CBA

CBA

Definitions

axis

The axis this direction belongs to, either "horizontal" or "vertical".

```
self.axis()
#ltr.axis() \
#ttb.axis()
```

horizontal
vertical

start

The start point of this direction, as an alignment.

```
self.start() -> alignment
#ltr.start() \
#rtl.start() \
#ttb.start() \
#btt.start()
```

left
right
top
bottom

end

The end point of this direction, as an alignment.

```
self.end() -> alignment
#ltr.end() \
#rtl.end() \
#ttb.end() \
#btt.end()
```

`right`
`left`
`bottom`
`top`

inv

The inverse direction.

```
self.inv() -> direction
#ltr.inv() \
#rtl.inv() \
#ttb.inv() \
#btt.inv()
```

`rtl`
`ltr`
`btt`
`ttb`

3.10.9 Fraction

Defines how the remaining space in a layout is distributed.

Each fractionally sized element gets space based on the ratio of its fraction to the sum of all fractions.

For more details, also see the [h](#) and [v](#) functions and the [grid function](#).

Example

```
Left #h(1fr) Left-ish #h(2fr) Right
```

Left

Left-ish

Right

3.10.10 Grid

Arranges content in a grid.

The grid element allows you to arrange content in a grid. You can define the number of rows and columns, as well as the size of the gutters between them. There are multiple sizing modes for columns and rows that can be used to create complex layouts.

While the grid and table elements work very similarly, they are intended for different use cases and carry different semantics. The grid element is intended for presentational and layout purposes, while the [table](#) element is intended for, in broad terms, presenting multiple related data points. In the future, Typst will annotate its output such that screenreaders will announce content in `table` as tabular while a grid's content will be announced no different than multiple content blocks in the document flow. Set and show rules on one of these elements do not affect the other.

A grid's sizing is determined by the track sizes specified in the arguments. Because each of the sizing parameters accepts the same values, we will explain them just once, here. Each sizing argument accepts an array of individual track sizes. A track size is either:

- `auto`: The track will be sized to fit its contents. It will be at most as large as the remaining space. If there is more than one `auto` track width, and together they

claim more than the available space, the `auto` tracks will fairly distribute the available space among themselves.

- A fixed or relative length (e.g. `10pt` or `20%` – `1cm`): The track will be exactly of this size.

- A fractional length (e.g. `1fr`): Once all other tracks have been sized, the remaining space will be divided among the fractional tracks according to their fractions. For example, if there are two fractional tracks, each with a fraction of `1fr`, they will each take up half of the remaining space.

To specify a single track, the array can be omitted in favor of a single value. To specify multiple `auto` tracks, enter the number of tracks instead of an array. For example, `columns: 3` is equivalent to `columns: (auto, auto, auto)`.

Examples

The example below demonstrates the different track sizing options. It also shows how you can use [grid.cell](#) to make an individual cell span two grid tracks.

```
// We use `rect` to emphasize the
// area of cells.
#set rect(
    inset: 8pt,
    fill: rgb("e4e5ea"),
    width: 100%,
)

#grid(
    columns: (60pt, 1fr, 2fr),
    rows: (auto, 60pt),
    gutter: 3pt,
    rect[Fixed width, auto height],
    rect[1/3 of the remains],
```

```
rect[2/3 of the remains],
rect(height: 100%) [Fixed height],
grid.cell(
    colspan: 2,
    image("tiger.jpg", width: 100%),
),
)
```



You can also [spread](#) an array of strings or content into a grid to populate its cells.

```
#grid(
    columns: 5,
    gutter: 5pt,
    ..range(25).map(str)
)
```

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Styling the grid

The grid's appearance can be customized through different parameters. These are the most important ones:

- [fill](#) to give all cells a background

- [align](#) to change how cells are aligned
- [inset](#) to optionally add internal padding to each cell
- [stroke](#) to optionally enable grid lines with a certain stroke

If you need to override one of the above options for a single cell, you can use the [grid.cell](#) element. Likewise, you can override individual grid lines with the [grid.hline](#) and [grid.vline](#) elements.

Alternatively, if you need the appearance options to depend on a cell's position (column and row), you may specify a function to `fill` or `align` of the form `(column, row) => value`. You may also use a show rule on [grid.cell](#) - see that element's examples or the examples below for more information.

Locating most of your styling in set and show rules is recommended, as it keeps the grid's or table's actual usages clean and easy to read. It also allows you to easily change the grid's appearance in one place.

Stroke styling precedence

There are three ways to set the stroke of a grid cell: through [grid.cell's stroke field](#), by using [grid.hline](#) and [grid.vline](#), or by setting the [grid's stroke field](#). When multiple of these settings are present and conflict, the `hline` and `vline` settings take the highest precedence, followed by the `cell` settings, and finally the `grid` settings.

Furthermore, strokes of a repeated grid header or footer will take precedence over regular cell strokes.

Parameters

```
grid(
    columns: autointrelativefractionarray,
    rows: autointrelativefractionarray,
    gutter: autointrelativefractionarray,
    column-gutter: autointrelativefractionarray,
    row-gutter: autointrelativefractionarray,
    fill: nonecolorgradientarraypatternfunction,
    align: autoarrayalignmentfunction,
    stroke: nonelengthcolorgradientarraystrokepatterndictionaryfunction,
    inset: relativearraydictionaryfunction,
    ...content,
) -> content
```

columns

auto or int or relative or fraction or array

Settable

The column sizes.

Either specify a track size array or provide an integer to create a grid with that many auto-sized columns. Note that opposed to rows and gutters, providing a single track size will only ever create a single column.

Default: ()

rows

auto or int or relative or fraction or array

Settable

The row sizes.

If there are more cells than fit the defined rows, the last row is repeated until there are no more cells.

Default: ()

gutter

auto or int or relative or fraction or array

Settable

The gaps between rows and columns.

If there are more gutters than defined sizes, the last gutter is repeated.

This is a shorthand to set `column-gutter` and `row-gutter` to the same value.

Default: `()`

column-gutter

`auto` or `int` or `relative` or `fraction` or `array`

Settable

The gaps between columns.

Default: `()`

row-gutter

`auto` or `int` or `relative` or `fraction` or `array`

Settable

The gaps between rows.

Default: `()`

fill

`none` or `color` or `gradient` or `array` or `pattern` or `function`

Settable

How to fill the cells.

This can be a color or a function that returns a color. The function receives the cells' column and row indices, starting from zero. This can be used to implement striped grids.

Default: `none`

[View example](#)

```
#grid(
    fill: (x, y) =>
        if calc.even(x + y) { luma(230) }
        else { white },
    align: center + horizon,
    columns: 4,
    inset: 2pt,
```

```
[X], [O], [X], [O],
[O], [X], [O], [X],
[X], [O], [X], [O],
[O], [X], [O], [X],
)
```

X	O	X	O
O	X	O	X
X	O	X	O
O	X	O	X

align

`auto` or `array` or `alignment` or `function`

Settable

How to align the cells' content.

This can either be a single alignment, an array of alignments (corresponding to each column) or a function that returns an alignment. The function receives the cells' column and row indices, starting from zero. If set to `auto`, the outer alignment is used.

You can find an example for this argument at the [table.align](#) parameter.

Default: `auto`

stroke

`none` or `length` or `color` or `gradient` or `array` or `stroke` or `pattern` or `dictionary` or `function`

Settable

How to [stroke](#) the cells.

Grids have no strokes by default, which can be changed by setting this option to the desired stroke.

If it is necessary to place lines which can cross spacing between cells produced by the `gutter` option, or to override the stroke between multiple specific cells, consider specifying one or more of [grid.hline](#) and [grid.vline](#) alongside your grid cells.

Default: `(::)`

[View example](#)

```
#set page(height: 13em, width: 26em)

#let cv(..jobs) = grid(
    columns: 2,
    inset: 5pt,
    stroke: (x, y) => if x == 0 and y > 0 {
        (right: (
            paint: luma(180),
            thickness: 1.5pt,
            dash: "dotted"
        ))
    },
    grid.header(grid.cell(colspan: 2) [
        *Professional Experience*
        #box(width: 1fr, line(length: 100%, stroke: luma(180)))
    ]),
    ..{
        let last = none
        for job in jobs.pos() {
            (
                if job.year != last [*#job.year*],
                [
                    *#job.company* - #job.role _ (#job.timeframe) _ \
                    #job.details
                ]
            )
            last = job.year
        }
    }
)

#cv(
    (

```

```
year: 2012,
company: [Pear Seed & Co.],
role: [Lead Engineer],
timeframe: [Jul - Dec],
details: [
    - Raised engineers from 3x to 10x
    - Did a great job
],
),
(
    year: 2012,
    company: [Mega Corp.],
    role: [VP of Sales],
    timeframe: [Mar - Jun],
    details: [- Closed tons of customers],
),
(
    year: 2013,
    company: [Tiny Co.],
    role: [CEO],
    timeframe: [Jan - Dec],
    details: [- Delivered 4x more shareholder value],
),
(
    year: 2014,
    company: [Glorbocorp Ltd],
    role: [CTO],
    timeframe: [Jan - Mar],
    details: [- Drove containerization forward],
),
)
```

Professional Experience

2012	Pear Seed & Co. - Lead Engineer (<i>Jul - Dec</i>) <ul style="list-style-type: none"> • Raised engineers from 3x to 10x • Did a great job Mega Corp. - VP of Sales (<i>Mar - Jun</i>) <ul style="list-style-type: none"> • Closed tons of customers
2013	Tiny Co. - CEO (<i>Jan - Dec</i>)

Professional Experience

2014	<ul style="list-style-type: none"> • Delivered 4x more shareholder value Glorbocorp Ltd - CTO (<i>Jan - Mar</i>) <ul style="list-style-type: none"> • Drove containerization forward
-------------	---

`inset`

relative or array or dictionary or function

Settable

How much to pad the cells' content.

You can find an example for this argument at the [table.inset](#) parameter.

Default: `(:)`

`children`

content

RequiredPositional

Variadic

The contents of the grid cells, plus any extra grid lines specified with the

[grid.hline](#) and [grid.vline](#) elements.

The cells are populated in row-major order.

Definitions

`cellElement`

A cell in the grid. You can use this function in the argument list of a grid to override grid style properties for an individual cell or manually positioning it within the grid. You can also use this function in show rules to apply certain styles to multiple cells at once.

For example, you can override the position and stroke for a single cell:

```
grid.cell(
    content,
    x: autoint,
    y: autoint,
    colspan: int,
    rowspan: int,
    fill: noneautoicolorgradientpattern,
    align: autoalignment,
    inset: autorelativedictionary,
    stroke: nonelengthcolorgradientstrokepatterndictionary,
    breakable: autobool,
) -> content
    #set text(15pt, font: "Noto Sans Symbols 2")
    #show regex("[皇-𠂇]"): set text(fill: rgb("21212A"))
    #show regex("[皇-𠂇]"): set text(fill: rgb("111015"))

#grid(
    fill: (x, y) => rgb(
        if calc.odd(x + y) { "7F8396" }
        else { "EFF0F3" }
    ),
    columns: (1em,) * 8,
    rows: 1em,
    align: center + horizon,

    [皇], [𠂇], [皇], [𠂇], [皇], [皇], [𠂇], [皇],
    [皇], [皇], [皇], [皇], [], [皇], [皇], [皇],
    grid.cell(
        x: 4, y: 3,
        stroke: blue.transparentize(60%)
    )[皇],
```

```
..(grid.cell(y: 6)[\u265f],) * 8,
..([\u265f], [\u265e], [\u265d], [\u265c], [\u265b], [\u265a], [\u265e], [\u265f])
    .map(grid.cell.with(y: 7)),
)
```



You may also apply a show rule on `grid.cell` to style all cells at once, which allows you, for example, to apply styles based on a cell's position. Refer to the examples of the [table.cell](#) element to learn more about this.

bodycontent**RequiredPositional**

The cell's body.

xauto or int**Settable**

The cell's column (zero-indexed). This field may be used in show rules to style a cell depending on its column.

You may override this field to pick in which column the cell must be placed. If no row (`y`) is chosen, the cell will be placed in the first row (starting at row 0) with that column available (or a new row if none). If both `x` and `y` are chosen, however, the cell will be placed in that exact position. An error is raised if that

position is not available (thus, it is usually wise to specify cells with a custom position before cells with automatic positions).

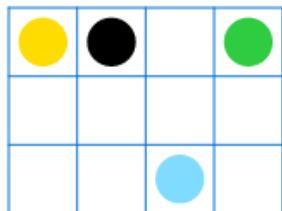
Default: `auto`

[View example](#)

```
#let circ(c) = circle(
    fill: c, width: 5mm
)

#grid(
    columns: 4,
    rows: 7mm,
    stroke: .5pt + blue,
    align: center + horizon,
    inset: 1mm,

    grid.cell(x: 2, y: 2, circ(aqua)),
    circ(yellow),
    grid.cell(x: 3, circ(green)),
    circ(black),
)
```



y

`auto` or `int`

Settable

The cell's row (zero-indexed). This field may be used in show rules to style a cell depending on its row.

You may override this field to pick in which row the cell must be placed. If no column (`x`) is chosen, the cell will be placed in the first column (starting at

column 0) available in the chosen row. If all columns in the chosen row are already occupied, an error is raised.

Default: `auto`

[View example](#)

```
#let tri(c) = polygon.regular(
    fill: c,
    size: 5mm,
    vertices: 3,
)

#grid(
    columns: 2,
    stroke: blue,
    inset: 1mm,

    tri(black),
    grid.cell(y: 1, tri(teal)),
    grid.cell(y: 1, tri(red)),
    grid.cell(y: 2, tri(orange))
)
```



`colspan`

`int`

Settable

The amount of columns spanned by this cell.

Default: `1`

`rowspan`

`int`

Settable

The amount of rows spanned by this cell.

Default: `1`

fill`none or auto or color or gradient or pattern`**Settable**

The cell's [fill](#) override.

Default: `auto`

align`auto or alignment`**Settable**

The cell's [alignment](#) override.

Default: `auto`

inset`auto or relative or dictionary`**Settable**

The cell's [inset](#) override.

Default: `auto`

stroke`none or length or color or gradient or stroke or pattern or dictionary`**Settable**

The cell's [stroke](#) override.

Default: `(::)`

breakable`auto or bool`**Settable**

Whether rows spanned by this cell can be placed in different pages. When equal to `auto`, a cell spanning only fixed-size rows is unbreakable, while a cell spanning at least one `auto`-sized row is breakable.

Default: `auto`

hline*Element*

A horizontal line in the grid.

Overrides any per-cell stroke, including stroke specified through the grid's stroke field. Can cross spacing between cells created through the grid's column-gutter option.

An example for this function can be found at the [table.hline](#) element.

```
grid.hline(
  y: auto|int,
  start: int,
  end: none|int,
  stroke: nonelengthcolorgradientstrokepatterndictionary,
  position: alignment,
) -> content
```

y

auto or int

Settable

The row above which the horizontal line is placed (zero-indexed). If the position field is set to bottom, the line is placed below the row with the given index instead (see that field's docs for details).

Specifying auto causes the line to be placed at the row below the last automatically positioned cell (that is, cell without coordinate overrides) before the line among the grid's children. If there is no such cell before the line, it is placed at the top of the grid (row 0). Note that specifying for this option exactly the total amount of rows in the grid causes this horizontal line to override the bottom border of the grid, while a value of 0 overrides the top border.

Default: auto

start
int

Settable

The column at which the horizontal line starts (zero-indexed, inclusive).

Default: `0`

end

`none` or `int`

Settable

The column before which the horizontal line ends (zero-indexed, exclusive).

Therefore, the horizontal line will be drawn up to and across column `end - 1`.

A value equal to `none` or to the amount of columns causes it to extend all the way towards the end of the grid.

Default: `none`

stroke

`none` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

Settable

The line's stroke.

Specifying `none` removes any lines previously placed across this line's range, including hlines or per-cell stroke below it.

Default: `1pt + black`

position

alignment

Settable

The position at which the line is placed, given its row (`y`) - either `top` to draw above it or `bottom` to draw below it.

This setting is only relevant when row gutter is enabled (and shouldn't be used otherwise - prefer just increasing the `y` field by one instead), since then the

position below a row becomes different from the position above the next row due to the spacing between both.

Default: `top`

vline*Element*

A vertical line in the grid.

Overrides any per-cell stroke, including stroke specified through the grid's `stroke` field. Can cross spacing between cells created through the grid's `row-gutter` option.

```
grid.vline(
  x: autoint,
  start: int,
  end: noneint,
  stroke: nonelengthcolorgradientstrokepatterndictionary,
  position: alignment,
) -> content
x
auto or int
```

Settable

The column before which the horizontal line is placed (zero-indexed). If the `position` field is set to `end`, the line is placed after the column with the given index instead (see that field's docs for details).

Specifying `auto` causes the line to be placed at the column after the last automatically positioned cell (that is, cell without coordinate overrides) before the line among the grid's children. If there is no such cell before the line, it is placed before the grid's first column (column 0). Note that specifying for this option exactly the total amount of columns in the grid causes this vertical line

to override the end border of the grid (right in LTR, left in RTL), while a value of 0 overrides the start border (left in LTR, right in RTL).

Default: `auto`

start

`int`

Settable

The row at which the vertical line starts (zero-indexed, inclusive).

Default: `0`

end

`none` or `int`

Settable

The row on top of which the vertical line ends (zero-indexed, exclusive).

Therefore, the vertical line will be drawn up to and across row `end - 1`.

A value equal to `none` or to the amount of rows causes it to extend all the way towards the bottom of the grid.

Default: `none`

stroke

`none` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

Settable

The line's stroke.

Specifying `none` removes any lines previously placed across this line's range, including vlines or per-cell stroke below it.

Default: `1pt + black`

position

`alignment`

Settable

The position at which the line is placed, given its column (`x`) - either

`start` to draw before it or `end` to draw after it.

The values `left` and `right` are also accepted, but discouraged as they cause your grid to be inconsistent between left-to-right and right-to-left documents.

This setting is only relevant when column gutter is enabled (and shouldn't be used otherwise - prefer just increasing the `x` field by one instead), since then the position after a column becomes different from the position before the next column due to the spacing between both.

Default: `start`

headerElement

A repeatable grid header.

If `repeat` is set to `true`, the header will be repeated across pages. For an example, refer to the [table.header](#) element and the [grid.stroke](#) parameter.

```
grid.header(
  repeat: bool,
  ..content,
) -> content
repeat
bool
```

Settable

Whether this header should be repeated across pages.

Default: `true`

children

`content`

RequiredPositional

Variadic

The cells and lines within the header.

footerElement

A repeatable grid footer.

Just like the [grid.header](#) element, the footer can repeat itself on every page of the table.

No other grid cells may be placed after the footer.

```
grid.footer(
  repeat: bool,
  ..content,
) -> content
repeat
bool
```

Settable

Whether this footer should be repeated across pages.

Default: **true**

children

[content](#)

RequiredPositional

Variadic

The cells and lines within the footer.

3.10.11 Hide

Hides content without affecting layout.

The `hide` function allows you to hide content while the layout still 'sees' it. This is useful to create whitespace that is exactly as large as some content. It may also be useful to redact content because its arguments are not included in the output.

Example

```
Hello Jane \
#hide [Hello] Joe
```

```
Hello Jane
Joe
```

Parameters

```
hide(
  content
) -> content
body
content
RequiredPositional
```

The content to hide.

3.10.12 Layout

Provides access to the current outer container's (or page's, if none) dimensions (width and height).

Accepts a function that receives a single parameter, which is a dictionary with keys `width` and `height`, both of type [length](#). The function is provided [context](#), meaning you don't need to use it in combination with the `context` keyword.

This is why [measure](#) can be called in the example below.

```
#let text = lorem(30)
#layout(size => [
  #let (height,) = measure(
    block(width: size.width, text),
  )
  This text is #height high with
  the current page width: \
  #text
])
```

This text is **64.79pt** high with the current page width:

 Lorem ipsum dolor sit amet, consectetur
 adipiscing elit, sed do eiusmod tempor
 incididunt ut labore et dolore magna
 aliquam quaerat voluptatem. Ut enim aequa
 doleamus animo, cum corpore dolemus, fieri.

Note that the `layout` function forces its contents into a [block](#)-level container, so placement relative to the page or pagebreaks are not possible within it.

If the `layout` call is placed inside a box with a width of **800pt** and a height of **400pt**, then the specified function will be given the argument `(width: 800pt, height: 400pt)`. If it is placed directly into the page, it receives the page's dimensions minus its margins. This is mostly useful in combination with [measurement](#).

You can also use this function to resolve [ratio](#) to fixed lengths. This might come in handy if you're building your own layout abstractions.

```
#layout(size => {
    let half = 50% * size.width
    [Half a page is #half wide.]
})
```

Half a page is **105pt** wide.

Note that the width or height provided by `layout` will be infinite if the corresponding page dimension is set to [auto](#).

Parameters

```
layout(
    function
) -> content
func
function
RequiredPositional
```

A function to call with the outer container's size. Its return value is displayed in the document.

The container's size is given as a [dictionary](#) with the keys `width` and `height`.

This function is called once for each time the content returned by `layout` appears in the document. This makes it possible to generate content that depends on the dimensions of its container.

3.10.13 Length

A size or distance, possibly expressed with contextual units.

Typst supports the following length units:

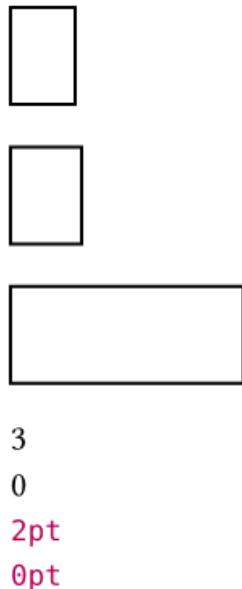
- Points: `72pt`
- Millimeters: `254mm`
- Centimeters: `2.54cm`
- Inches: `1in`
- Relative to font size: `2.5em`

You can multiply lengths with and divide them by integers and floats.

Example

```
#rect(width: 20pt)
#rect(width: 2em)
#rect(width: 1in)
```

```
#(3em + 5pt).em \
#(20pt).em \
#(40em + 2pt).abs \
#(5em).abs
```



Fields

- `abs`: A length with just the absolute component of the current length (that is, excluding the `em` component).
- `em`: The amount of `em` units in this length, as a [float](#).

Definitions

`pt`

Converts this length to points.

Fails with an error if this length has non-zero `em` units (such as `5em + 2pt` instead of just `2pt`). Use the `abs` field (such as in `(5em + 2pt).abs.pt()`) to ignore the `em` component of the length (thus converting only its absolute component).

`self.pt() -> float`

mm

Converts this length to millimeters.

Fails with an error if this length has non-zero `em` units. See the [pt](#) method for more details.

```
self.mm() -> float  
cm
```

Converts this length to centimeters.

Fails with an error if this length has non-zero `em` units. See the [pt](#) method for more details.

```
self.cm() -> float  
inches
```

Converts this length to inches.

Fails with an error if this length has non-zero `em` units. See the [pt](#) method for more details.

```
self.inches() -> float  
to-absolute
```

Resolve this length to an absolute length.

```
self.to-absolute() -> length  
  #set text(size: 12pt)  
  #context [  
    #(6pt).to-absolute() \  
    #(6pt + 10em).to-absolute() \  
    #(10em).to-absolute()  
  ]  
  
  #set text(size: 6pt)  
  #context [  
    #(6pt).to-absolute() \  
    #(6pt + 10em).to-absolute() \  
    #(10em).to-absolute()  
  ]
```

6pt
126pt
120pt

6pt
66pt
60pt

3.10.14 Measure

Measures the layouted size of content.

The `measure` function lets you determine the layouted size of content. By default an infinite space is assumed, so the measured dimensions may not necessarily match the final dimensions of the content. If you want to measure in the current layout dimensions, you can combine `measure` and [`layout`](#).

Example

The same content can have a different size depending on the [`context`](#) that it is placed into. In the example below, the `#content` is of course bigger when we increase the font size.

```
#let content = [Hello!]
#content
#set text(14pt)
#content
```

Hello! Hello!

For this reason, you can only measure when context is available.

```
#let thing(body) = context {
    let size = measure(body)
    [Width of "#body" is #size.width]
}
```

```
#thing[Hey] \
#thing[Welcome]
```

Width of “Hey” is **18.54pt**
 Width of “Welcome” is **41.26pt**

The measure function returns a dictionary with the entries `width` and `height`, both of type [length](#).

Parameters

```
measure(
  width: autolength,
  height: autolength,
  content,
  nonestyles,
) -> dictionary
width
auto or length
```

The width available to layout the content.

Setting this to `auto` indicates infinite available width.

Note that using the `width` and `height` parameters of this function is different from measuring a sized [block](#) containing the content. In the following example, the former will get the dimensions of the inner content instead of the dimensions of the block.

Default: `auto`

[View example](#)

```
#context measure(lorem(100), width: 400pt)

#context measure(block(lorem(100), width: 400pt))
```

```
(width: 395.89pt, height: 122.34pt)
(width: 400pt, height: 122.34pt)
```

height

`auto` or `length`

The height available to layout the content.

Setting this to `auto` indicates infinite available height.

Default: `auto`

content

`content`

RequiredPositional

The content whose size to measure.

styles

`none` or `styles`

Positional

Compatibility: This argument is deprecated. It only exists for compatibility with Typst 0.10 and lower and shouldn't be used anymore.

Default: `none`

3.10.15 Move

Moves content without affecting layout.

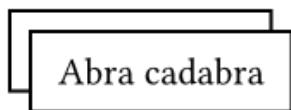
The `move` function allows you to move content while the layout still 'sees' it at the original positions. Containers will still be sized as if the content was not moved.

Example

```
#rect(inset: 0pt, move(
    dx: 6pt, dy: 6pt,
    rect(
```

```

    inset: 8pt,
    fill: white,
    stroke: black,
    [Abra cadabra]
)
)
)
```



Parameters

```

move (
  dx: relative,
  dy: relative,
  content,
) -> content
dx
relative
Settable
```

The horizontal displacement of the content.

Default: 0% + 0pt

```

dy
relative
Settable
```

The vertical displacement of the content.

Default: 0% + 0pt

```

body
content
RequiredPositional
```

The content to move.

3.10.16 Padding

Adds spacing around content.

The spacing can be specified for each side individually, or for all sides at once by specifying a positional argument.

Example

```
#set align(center)

#pad(x: 16pt, image("typing.jpg"))
_ Typing speeds can be
measured in words per minute._
```



Typing speeds can be measured in words per minute.

Parameters

```
pad(
  left: relative,
  top: relative,
  right: relative,
  bottom: relative,
  x: relative,
  y: relative,
  rest: relative,
  content,
) -> content
```

left

`relative`

Settable

The padding at the left side.

Default: `0% + 0pt`

`top`

`relative`

Settable

The padding at the top side.

Default: `0% + 0pt`

`right`

`relative`

Settable

The padding at the right side.

Default: `0% + 0pt`

`bottom`

`relative`

Settable

The padding at the bottom side.

Default: `0% + 0pt`

`x`

`relative`

Settable

A shorthand to set `left` and `right` to the same value.

Default: `0% + 0pt`

`y`

`relative`

Settable

A shorthand to set `top` and `bottom` to the same value.

Default: `0% + 0pt`

`rest`

`relative`

Settable

A shorthand to set all four sides to the same value.

Default: `0%` + `0pt`

`body`

`content`

RequiredPositional

The content to pad at the sides.

3.10.17 Page

Layouts its child onto one or multiple pages.

Although this function is primarily used in set rules to affect page properties, it can also be used to explicitly render its argument onto a set of pages of its own.

Pages can be set to use `auto` as their width or height. In this case, the pages will grow to fit their content on the respective axis.

The [Guide for Page Setup](#) explains how to use this and related functions to set up a document with many examples.

Example

```
#set page("us-letter")
```

There you go, US friends!

There you go, US friends!

Parameters

```
page(  
    paper: str,  
    width: autolength,  
    height: autolength,  
    flipped: bool,  
    margin: autorelativedictionary,
```

```

binding: autoalignment,
columns: int,
fill: noneautocolorgradientpattern,
numbering: nonestrfunction,
number-align: alignment,
header: noneautocontent,
header-ascent: relative,
footer: noneautocontent,
footer-descent: relative,
background: nonecontent,
foreground: nonecontent,
content,
) -> content
paper
str
Settable

```

A standard paper size to set width and height.

This is just a shorthand for setting `width` and `height` and, as such, cannot be retrieved in a context expression.

[View options](#)

Default: "a4"

[View paper sizes](#)

```

"a0", "a1", "a2", "a3", "a4", "a5", "a6", "a7", "a8", "a9", "a10", "a11", "iso-b1",
"iso-b2", "iso-b3", "iso-b4", "iso-b5", "iso-b6", "iso-b7", "iso-b8", "iso-
c3", "iso-c4", "iso-c5", "iso-c6", "iso-c7", "iso-c8", "din-d3", "din-d4",
"din-d5", "din-d6", "din-d7", "din-d8", "sis-g5", "sis-e5", "ansi-a", "ansi-
b", "ansi-c", "ansi-d", "ansi-e", "arch-a", "arch-b", "arch-c", "arch-d",
"arch-e1", "arch-e", "jis-b0", "jis-b1", "jis-b2", "jis-b3", "jis-b4", "jis-
b5", "jis-b6", "jis-b7", "jis-b8", "jis-b9", "jis-b10", "jis-b11", "sac-d0",
"sac-d1", "sac-d2", "sac-d3", "sac-d4", "sac-d5", "sac-d6", "iso-id-1", "iso-
id-2", "iso-id-3", "asia-f4", "jp-shiroku-ban-4", "jp-shiroku-ban-5", "jp-

```

```
shiroku-ban-6", "jp-kiku-4", "jp-kiku-5", "jp-business-card", "cn-
business-card", "eu-business-card", "fr-telli re", "fr-couronne- criture",
"fr-couronne- dition", "fr-raisin", "fr-carr ", "fr-j sus", "uk-brief",
"uk-draft", "uk-foolscap", "uk-quarto", "uk-crown", "uk-book-a", "uk-book-
b", "us-letter", "us-legal", "us-tabloid", "us-executive", "us-foolscap-
folio", "us-statement", "us-ledger", "us-oficio", "us-gov-letter", "us-gov-
legal", "us-business-card", "us-digest", "us-trade", "newspaper-compact",
"newspaper-berliner", "newspaper-broadsheet", "presentation-16-9",
"presentation-4-3"
```

width

auto or length

Settable

The width of the page.

Default: 595.28pt

[View example](#)

```
#set page(
    width: 3cm,
    margin: (x: 0cm),
)

#for i in range(3) {
    box(square(width: 1cm))
}
```

**height**

auto or length

Settable

The height of the page.

If this is set to `auto`, page breaks can only be triggered manually by inserting a [page break](#). Most examples throughout this documentation use `auto` for the height of the page to dynamically grow and shrink to fit their content.

Default: `841.89pt`

flipped

`bool`

Settable

Whether the page is flipped into landscape orientation.

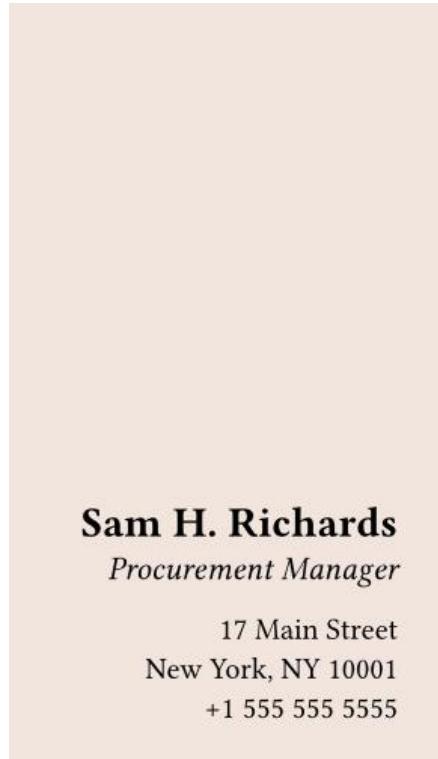
Default: `false`

[View example](#)

```
#set page(
    "us-business-card",
    flipped: true,
    fill: rgb("f2e5dd"),
)

#set align(bottom + end)
#text(14pt) [*Sam H. Richards*] \
_Procurement Manager_

#set text(10pt)
17 Main Street \
New York, NY 10001 \
+1 555 555 5555
```



margin

`auto` or `relative` or `dictionary`

Settable

The page's margins.

- `auto`: The margins are set automatically to 2.5/21 times the smaller dimension of the page. This results in 2.5cm margins for an A4 page.
- A single length: The same margin on all sides.

• A dictionary: With a dictionary, the margins can be set individually. The dictionary can contain the following keys in order of precedence:

`otop`: The top margin.

`oright`: The right margin.

`obottom`: The bottom margin.

`oleft`: The left margin.

`oinside`: The margin at the inner side of the page (where the [binding](#) is).

`ooutside`: The margin at the outer side of the page (opposite to the [binding](#)).

`ox`: The horizontal margins.

`oy`: The vertical margins.

`orest`: The margins on all sides except those for which the dictionary explicitly sets a size.

The values for `left` and `right` are mutually exclusive with the values for `inside` and `outside`.

Default: `auto`

[View example](#)

```
#set page(
    width: 3cm,
    height: 4cm,
    margin: (x: 8pt, y: 4pt),
)

#rect(
    width: 100%,
    height: 100%,
    fill: aqua,
)
```



binding

`auto` or [alignment](#)

Settable

On which side the pages will be bound.

- **auto**: Equivalent to `left` if the [text direction](#) is left-to-right and `right` if it is right-to-left.

- `left`: Bound on the left side.

- `right`: Bound on the right side.

This affects the meaning of the `inside` and `outside` options for margins.

Default: `auto`

columns

`int`

Settable

How many columns the page has.

If you need to insert columns into a page or other container, you can also use

the [columns function](#).

Default: `1`

[View example](#)

```
#set page(columns: 2, height: 4.8cm)
Climate change is one of the most
pressing issues of our time, with
the potential to devastate
communities, ecosystems, and
economies around the world. It's
clear that we need to take urgent
action to reduce our carbon
emissions and mitigate the impacts
of a rapidly changing climate.
```

Climate change is one of the most pressing issues of our time, with the potential to devastate communities, ecosystems, and economies around the world. It's clear that we need to take urgent action to reduce our carbon emissions and mitigate the impacts

fill

Settable

The page's background fill.

Setting this to something non-transparent instructs the printer to color the complete page. If you are considering larger production runs, it may be more environmentally friendly and cost-effective to source pre-dyed pages and not set this property.

When set to `none`, the background becomes transparent. Note that PDF pages will still appear with a (usually white) background in viewers, but they are actually transparent. (If you print them, no color is used for the background.)

The default of `auto` results in `none` for PDF output, and `white` for PNG and SVG.

Default: `auto`

[View example](#)

```
#set page(fill: rgb("444352"))
#set text(fill: rgb("fdfdfd"))

*Dark mode enabled.*
```

Dark mode enabled.

numbering

none or str or function

Settable

How to [number](#) the pages.

If an explicit `footer` (or `header` for top-aligned numbering) is given, the numbering is ignored.

Default: `none`

[View example](#)

```
#set page(
    height: 100pt,
    margin: (top: 16pt, bottom: 24pt),
    numbering: "1 / 1",
)
```

`#lorem(48)`

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna
aliquam quaerat voluptatem. Ut enim aequ

1 / 2

doleamus animo, cum corpore dolemus, fieri
tamen permagna accessio potest, si aliquod
aeternum et infinitum impendere malum
nobis opinemur. Quod idem licet transferre in.

2 / 2

number-align

alignment

Settable

The alignment of the page numbering.

If the vertical component is `top`, the numbering is placed into the header and if it is `bottom`, it is placed in the footer. Horizon alignment is forbidden. If an explicit matching `header` or `footer` is given, the numbering is ignored.

Default: `center + bottom`

[View example](#)

```
#set page(
    margin: (top: 16pt, bottom: 24pt),
    numbering: "1",
    number-align: right,
)

#lorem(30)
```

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna
aliquam quaerat voluptatem. Ut enim aequa
doleamus animo, cum corpore dolemus, fieri.

1

header

`none` or `auto` or `content`

Settable

The page's header. Fills the top margin of each page.

- **Content:** Shows the content as the header.
- **auto:** Shows the page number if a `numbering` is set and `number-align` is `top`.
- **none:** Suppresses the header.

Default: `auto`

[View example](#)

```
#set par(justify: true)
#set page(
    margin: (top: 32pt, bottom: 20pt),
    header: [
        #set text(8pt)
        #smallcaps[Typst Academcy]
        #h(1fr) _Exercise Sheet 3_
    ],
)
#lorem(19)
```

TYPST ACADEMY

Exercise Sheet 3

 Lorem ipsum dolor sit amet, consectetur adip-
 iscing elit, sed do eiusmod tempor incididunt ut
 labore et dolore magna aliquam.

header-ascent

relative

Settable

The amount the header is raised into the top margin.

Default: 30% + 0pt

footer

none or auto or content

Settable

The page's footer. Fills the bottom margin of each page.

- Content: Shows the content as the footer.

- auto: Shows the page number if a `numbering` is set and `number-align` is bottom.

- none: Suppresses the footer.

For just a page number, the `numbering` property typically suffices. If you want to create a custom footer but still display the page number, you can directly access the [page counter](#).

Default: `auto`

[View example](#)

```
#set par(justify: true)
#set page(
    height: 100pt,
    margin: 20pt,
    footer: context [
        #set align(right)
        #set text(8pt)
        #counter(page).display(
            "1 of I",
            both: true,
        )
    ]
)

#lorem(48)
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua
quaerat voluptatem. Ut enim aequo doleamus

1 of II

animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in.

2 of II

footer-descent

[relative](#)

Settable

The amount the footer is lowered into the bottom margin.

Default: `30%` + `0pt`

background

`none` or `content`

Settable

Content in the page's background.

This content will be placed behind the page's body. It can be used to place a background image or a watermark.

Default: `none`

[View example](#)

```
#set page (background: rotate(24deg,
    text(18pt, fill: rgb("FFCBC4")) [
        *CONFIDENTIAL*
    ]
) )
```

= Typst's secret plans

In the year 2023, we plan to take over the world (of typesetting).

Typst's secret plans

In the year 2023, we plan to take over the world (of typesetting).

foreground

`none` or `content`

Settable

Content in the page's foreground.

This content will overlay the page's body.

Default: `none`

[View example](#)

```
#set page(foreground: text(24pt) [?])
```

Reviewer 2 has marked our paper
 "Weak Reject" because they did
 not understand our approach...

Reviewer 2 has marked our paper "Weak Reject" because they ot understand our approach...

body
content
RequiredPositional

The contents of the page(s).

Multiple pages will be created if the content does not fit on a single page. A new page with the page properties prior to the function invocation will be created after the body has been typeset.

3.10.18 Page Break

A manual page break.

Must not be used inside any containers.

Example

The next page contains
 more details on compound theory.
`#pagebreak()`

== Compound Theory
 In 1984, the first ...

The next page contains more details on compound theory.

Compound Theory

In 1984, the first ...

Parameters

```
pagebreak(
    weak: bool,
    to: nonestr,
) -> content
weak
bool
Settable
```

If `true`, the page break is skipped if the current page is already empty.

Default: `false`

`to`

`none` or `str`

Settable

If given, ensures that the next page will be an even/odd page, with an empty page in between if necessary.

Variant	Details
<code>"even"</code>	Next page will be an even page.
<code>"odd"</code>	Next page will be an odd page.

Default: `none`

[View example](#)

```
#set page (height: 30pt)
```

First.

```
#pagebreak(to: "odd")
```

Third.

First.

Third.

3.10.19 Place

Places content relatively to its parent container.

Placed content can be either overlaid (the default) or floating. Overlaid content is aligned with the parent container according to the given [alignment](#), and shown over any other content added so far in the container. Floating content is placed at the top or bottom of the container, displacing other content down or up respectively. In both cases, the content position can be adjusted with [dx](#) and [dy](#) offsets without affecting the layout.

The parent can be any container such as a [block](#), [box](#), [rect](#), etc. A top level `place` call will place content directly in the text area of the current page. This can be used for absolute positioning on the page: with a `top + left` [alignment](#), the offsets `dx` and `dy` will set the position of the element's top left corner relatively to the top left corner of the text area. For absolute positioning on the full page including margins, you can use `place` in [page.foreground](#) or [page.background](#).

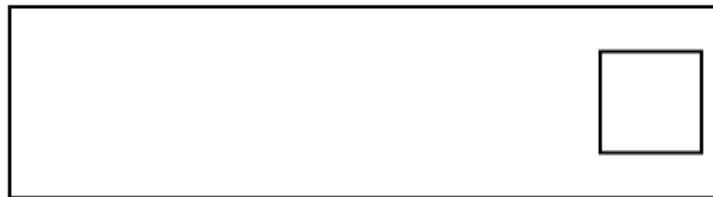
Examples

```
#set page(height: 120pt)
Hello, world!

#rect(
    width: 100%,
    height: 2cm,
    place(horizon + right, square()),
)

#place(
    top + left,
    dx: -5pt,
    square(size: 5pt, fill: red),
)
```

■Hello, world!



Effect on the position of other elements

Overlaid elements don't take space in the flow of content, but a `place` call inserts an invisible block-level element in the flow. This can affect the layout by breaking the current paragraph. To avoid this, you can wrap the `place` call in a [box](#) when the call is made in the middle of a paragraph. The alignment and offsets will then be relative to this zero-size box. To make sure it doesn't interfere with spacing, the box should be attached to a word using a word joiner.

For example, the following defines a function for attaching an annotation to the following word:

```
#let annotate(..args) = {
    box(place(..args))
    sym.wj
    h(0pt, weak: true)
}
```

A placed `#annotate(square(), dy: 2pt)`
square in my text.

A placed square in my text.



The zero-width weak spacing serves to discard spaces between the function call and the next word.

Parameters

```
place(
    autoalignment,
    scope: str,
    float: bool,
    clearance: length,
    dx: relative,
    dy: relative,
    content,
) -> content
alignment
```

`auto` or `alignment`

Positional

Settable

Relative to which position in the parent container to place the content.

- If `float` is `false`, then this can be any alignment other than `auto`.
- If `float` is `true`, then this must be `auto`, `top`, or `bottom`.

When `float` is `false` and no vertical alignment is specified, the content is placed at the current position on the vertical axis.

Default: start

scope

str

Settable

Relative to which containing scope something is placed.

The parent scope is primarily used with figures and, for this reason, the figure function has a mirrored [scope parameter](#). Nonetheless, it can also be more generally useful to break out of the columns. A typical example would be to [create a single-column title section](#) in a two-column document.

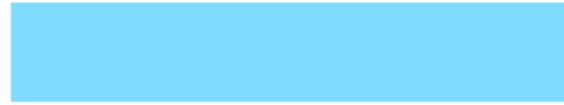
Note that parent-scoped placement is currently only supported if `float` is `true`. This may change in the future.

Variant	Details
<code>"column"</code>	Place into the current column.
<code>"parent"</code>	Place relative to the parent, letting the content span over all columns.

Default: `"column"`

[View example](#)

```
#set page(height: 150pt, columns: 2)
#place(
    top + center,
    scope: "parent",
    float: true,
    rect(width: 80%, fill: aqua),
)
#lorem(25)
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

float

`bool`

Settable

Whether the placed element has floating layout.

Floating elements are positioned at the top or bottom of the parent container, displacing in-flow content. They are always placed in the in-flow order relative to each other, as well as before any content following a later [place.flush](#) element.

Default: `false`

[View example](#)

```
#set page(height: 150pt)
#let note(where, body) = place(
  center + where,
  float: true,
  clearance: 6pt,
  rect(body),
)

#lorem(10)
#note(bottom) [Bottom 1]
#note(bottom) [Bottom 2]
#lorem(40)
#note(top) [Top]
#lorem(10)
```

Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna

Bottom 1

Bottom 2

Top

aliquam quaerat voluptatem. Ut enim aequa
dolet animo, cum corpore dolemus, fieri
tamen permagna accessio potest, si aliquod
aeternum et infinitum impendere.

Lore ipsum dolor sit amet, consectetur
adipiscing elit, sed do.

clearance

length

Settable

The spacing between the placed element and other elements in a floating layout.

Has no effect if `float` is `false`.

Default: `1.5em`

dx

relative

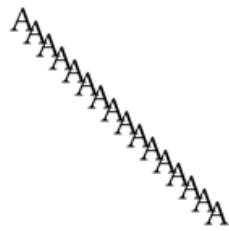
Settable

The horizontal displacement of the placed content.

Default: `0% + 0pt`

[View example](#)

```
#set page(height: 100pt)
#for i in range(16) {
    let amount = i * 4pt
    place(center, dx: amount - 32pt, dy: amount) [A]
}
```



This does not affect the layout of in-flow content. In other words, the placed content is treated as if it were wrapped in a [move](#) element.

dy

[relative](#)

Settable

The vertical displacement of the placed content.

This does not affect the layout of in-flow content. In other words, the placed content is treated as if it were wrapped in a [move](#) element.

Default: 0% + 0pt

body

[content](#)

RequiredPositional

The content to place.

Definitions

flushElement

Asks the layout algorithm to place pending floating elements before continuing with the content.

This is useful for preventing floating figures from spilling into the next section.

`place.flush() -> content`

```
#lorem(15)

#figure(
    rect(width: 100%, height: 50pt),
    placement: auto,
    caption: [A rectangle],
)

#place.flush()
```

This text appears after the figure.

 Lorem ipsum dolor sit
 amet, consectetur
 adipiscing elit, sed do
 eiusmod tempor
 incididunt ut labore.



Figure 1: A rectangle

This text appears after the figure.

3.10.20 Ratio

A ratio of a whole.

Written as a number, followed by a percent sign.

Example

```
#set align(center)
*scale(x: 150%) [
    Scaled apart.
]
```

Scaled apart.

3.10.21 Relative Length

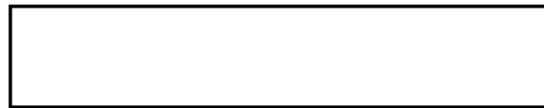
A length in relation to some known length.

This type is a combination of a [length](#) with a [ratio](#). It results from addition and subtraction of a length and a ratio. Wherever a relative length is expected, you can also use a bare length or ratio.

Example

```
#rect(width: 100% - 50pt)

#(100% - 50pt).length \
#(100% - 50pt).ratio
```



-50pt
100%

A relative length has the following fields:

- `length`: Its length component.
- `ratio`: Its ratio component.

3.10.22 Repeat

Repeats content to the available space.

This can be useful when implementing a custom index, reference, or outline.

Space may be inserted between the instances of the body parameter, so be sure to adjust the [justify](#) parameter accordingly.

Errors if there no bounds on the available space, as it would create infinite content.

Example

Sign on the dotted line:

```
#box(width: 1fr, repeat[.])

#set text(10pt)
#v(8pt, weak: true)
#align(right) [
    Berlin, the 22nd of December, 2022
]
```

Sign on the dotted line:

Berlin, the 22nd of December, 2022

Parameters

`repeat` (
`content`,
`gap`: `length`,
`justify`: `bool`,
`) -> content`
`body`
`content`

RequiredPositional

The content to repeat.

`gap`
`length`
Settable

The gap between each instance of the body.

Default: `0pt`

`justify`
`bool`
Settable

Whether to increase the gap between instances to completely fill the available space.

Default: `true`

3.10.23 Rotate

Rotates content without affecting layout.

Rotates an element by a given angle. The layout will act as if the element was not rotated unless you specify `reflow: true`.

Example

```
#stack(
    dir: ltr,
    spacing: 1fr,
    ..range(16)
    .map(i => rotate(24deg * i) [X]),
)
```

Parameters

```
rotate(
    angle,
    origin: alignment,
    reflow: bool,
    content,
) -> content
```

`angle`

`angle`

Positional

Settable

The amount of rotation.

Default: `0deg`

[View example](#)

```
#rotate(-1.571rad) [Space!]
```

Space!

origin

alignment

Settable

The origin of the rotation.

If, for instance, you wanted the bottom left corner of the rotated element to stay aligned with the baseline, you would set it to `bottom + left` instead.

Default: `center + horizon`

[View example](#)

```
#set text(spacing: 8pt)
#let square = square.with(width: 8pt)

#box(square())
#box(rotate(30deg, origin: center, square()))
#box(rotate(30deg, origin: top + left, square()))
#box(rotate(30deg, origin: bottom + right, square()))
```



reflow

bool

Settable

Whether the rotation impacts the layout.

If set to `false`, the rotated content will retain the bounding box of the original content. If set to `true`, the bounding box will take the rotation of the content into account and adjust the layout accordingly.

Default: `false`

[View example](#)

```
Hello #rotate(90deg, reflow: true) [World]!
```

Hello

World

!

body

content

RequiredPositional

The content to rotate.

3.10.24 Scale

Scales content without affecting layout.

Lets you mirror content by specifying a negative scale on a single axis.

Example

```
#set align(center)
#scale(x: -100%) [This is mirrored.]
#scale(x: -100%, reflow: true) [This is mirrored.]
```

This is mirrored.

This is mirrored.

Parameters

```
scale(
    autolengthratio,
    x: autolengthratio,
    y: autolengthratio,
    origin: alignment,
    reflow: bool,
    content,
) -> content
factor
```

`auto` or `length` or `ratio`

Positional

Settable

The scaling factor for both axes, as a positional argument. This is just an optional shorthand notation for setting `x` and `y` to the same value.

Default: `100%`

x

`auto` or `length` or `ratio`

Settable

The horizontal scaling factor.

The body will be mirrored horizontally if the parameter is negative.

Default: `100%`

y

`auto` or `length` or `ratio`

Settable

The vertical scaling factor.

The body will be mirrored vertically if the parameter is negative.

Default: `100%`

origin

alignment

Settable

The origin of the transformation.

Default: `center + horizon`

[View example](#)

```
A#box(scale(75%) [A])A \
B#box(scale(75%, origin: bottom + left) [B])B
```

```
AAA
BBB
```

reflowbool*Settable*

Whether the scaling impacts the layout.

If set to `false`, the scaled content will be allowed to overlap other content. If set to `true`, it will compute the new size of the scaled content and adjust the layout accordingly.

Default: `false`

[View example](#)

```
Hello #scale(x: 20%, y: 40%, reflow: true) [World]!
```

!cell

!

bodycontent*RequiredPositional*

The content to scale.

3.10.25 Skew

Skews content.

Skews an element in horizontal and/or vertical direction. The layout will act as if the element was not skewed unless you specify `reflow: true`.

Example

```
#skew(ax: -12deg) [
    This is some fake italic text.
```

]

This is some fake italic text.

Parameters

```
skew(
    ax: angle,
    ay: angle,
    origin: alignment,
    reflow: bool,
    content,
) -> content
```

ax
angle
Settable

The horizontal skewing angle.

Default: `0deg`

[View example](#)

```
#skew(ax: 30deg) [Skewed]
```



Skewed

ay

angle

Settable

The vertical skewing angle.

Default: `0deg`

[View example](#)

```
#skew/ay: 30deg) [Skewed]
```



Skewed

origin

alignment

Settable

The origin of the skew transformation.

The origin will stay fixed during the operation.

Default: center + horizon

[View example](#)

```
X #box(skew(ax: -30deg, origin: center + horizon) [X]) X \
X #box(skew(ax: -30deg, origin: bottom + left) [X]) X \
X #box(skew(ax: -30deg, origin: top + right) [X]) X
```

reflow

[bool](#)

Settable

Whether the skew transformation impacts the layout.

If set to `false`, the skewed content will retain the bounding box of the original content. If set to `true`, the bounding box will take the transformation of the content into account and adjust the layout accordingly.

Default: `false`

[View example](#)

```
Hello #skew(ax: 30deg, reflow: true, "World")!
```

body

[content](#)

RequiredPositional

The content to skew.

3.10.26 Spacing (H)

Inserts horizontal spacing into a paragraph.

The spacing can be absolute, relative, or fractional. In the last case, the remaining space on the line is distributed among all fractional spacings according to their relative fractions.

Example

```
First #h(1cm) Second \
First #h(30%) Second
```

First	Second
First	Second

Fractional spacing

With fractional spacing, you can align things within a line without forcing a paragraph break (like [align](#) would). Each fractionally sized element gets space based on the ratio of its fraction to the sum of all fractions.

```
First #h(1fr) Second \
First #h(1fr) Second #h(1fr) Third \
First #h(2fr) Second #h(1fr) Third
```

First	Second	Third
First	Second	Third
First	Second	Third

Mathematical Spacing

In [mathematical formulas](#), you can additionally use these constants to add spacing between elements: `thin` (1/6 em), `med` (2/9 em), `thick` (5/18 em), `quad` (1 em), `wide` (2 em).

Parameters

```
h (
    relativefraction,
    weak: bool,
) -> content
amount
```

`relative` or `fraction`

RequiredPositional

How much spacing to insert.

`weak`

`bool`

Settable

If `true`, the spacing collapses at the start or end of a paragraph. Moreover, from multiple adjacent weak spacings all but the largest one collapse.

Weak spacing in markup also causes all adjacent markup spaces to be removed, regardless of the amount of spacing inserted. To force a space next to weak spacing, you can explicitly write `#"` (for a normal space) or `~` (for a non-breaking space). The latter can be useful to create a construct that always attaches to the preceding word with one non-breaking space, independently of whether a markup space existed in front or not.

Default: `false`

[View example](#)

```
#h(1cm, weak: true)
```

We identified a group of `_weak_` specimens that fail to manifest in most cases. However, when

```
#h(8pt, weak: true) supported
#h(8pt, weak: true) on both sides,
they do show up.
```

Further `#h(0pt, weak: true)` more,
even the smallest of them swallow
adjacent markup spaces.

We identified a group of *weak* specimens that fail to manifest in most cases. However, when `supported` on both sides, they do show up.

Furthermore, even the smallest of them swallow adjacent markup spaces.

3.10.27 Spacing (V)

Inserts vertical spacing into a flow of blocks.

The spacing can be absolute, relative, or fractional. In the last case, the remaining space on the page is distributed among all fractional spacings according to their relative fractions.

Example

```
#grid(
    rows: 3cm,
    columns: 6,
    gutter: 1fr,
    [A #parbreak() B],
    [A #v(0pt) B],
    [A #v(10pt) B],
    [A #v(0pt, weak: true) B],
    [A #v(40%, weak: true) B],
    [A #v(1fr) B],
)
```

```

A      A      A      A      A      A
B      B      B      B      B      B

```

Parameters

```

v(
relativefraction,
weak: bool,
) -> content
amount
relative or fraction

```

RequiredPositional

How much spacing to insert.

weak

bool

Settable

If `true`, the spacing collapses at the start or end of a flow. Moreover, from multiple adjacent weak spacings all but the largest one collapse. Weak spacings will always collapse adjacent paragraph spacing, even if the paragraph spacing is larger.

Default: `false`

[View example](#)

The following theorem is foundational to the field:

```
#v(4pt, weak: true)
$ x^2 + y^2 = r^2 $
#v(4pt, weak: true)
```

The proof is simple:

The following theorem is foundational to the field:

$$x^2 + y^2 = r^2$$

The proof is simple:

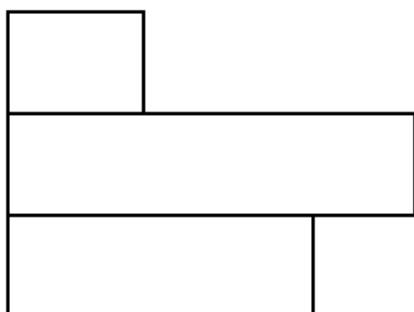
3.10.28 Stack

Arranges content and spacing horizontally or vertically.

The stack places a list of items along an axis, with optional spacing between each item.

Example

```
#stack(
    dir: ttb,
    rect(width: 40pt),
    rect(width: 120pt),
    rect(width: 90pt),
)
```



Parameters

```
stack(
    dir: direction,
    spacing: nonrelativefraction,
    ..relativefractioncontent,
) -> content
dir
direction
Settable
```

The direction along which the items are stacked. Possible values are:

- `ltr`: Left to right.
- `rtl`: Right to left.
- `ttb`: Top to bottom.
- `btt`: Bottom to top.

You can use the `start` and `end` methods to obtain the initial and final points (respectively) of a direction, as `alignment`. You can also use the `axis` method to determine whether a direction is "`horizontal`" or "`vertical`". The `inv` method returns a direction's inverse direction.

For example, `ttb.start()` is `top`, `ttb.end()` is `bottom`, `ttb.axis()` is "`vertical`" and `ttb.inv()` is equal to `btt`.

Default: `ttb`

spacing

`none` or `relative` or `fraction`

Settable

Spacing to insert between items where no explicit spacing was provided.

Default: `none`

children

`relative` or `fraction` or `content`

RequiredPositional

Variadic

The children to stack along the axis.

3.11 Visualize

Drawing and data visualization.

If you want to create more advanced drawings or plots, also have a look at the [CetZ](#) package as well as more specialized [packages](#) for your use case.

Definitions

- [circle](#)A circle with optional content.
- [color](#)A color in a specific color space.
- [ellipse](#)An ellipse with optional content.
- [gradient](#)A color gradient.
- [image](#)A raster or vector graphic.
- [line](#)A line from one point to another.
- [path](#)A path through a list of points, connected by Bezier curves.
- [pattern](#)A repeating pattern fill.
- [polygon](#)A closed polygon.
- [rect](#)A rectangle with optional content.
- [square](#)A square with optional content.
- [stroke](#)Defines how to draw a line.

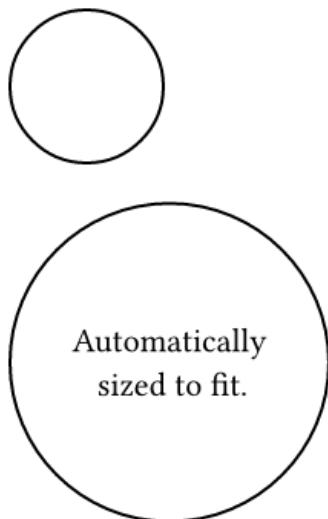
3.11.1 Circle

A circle with optional content.

Example

```
// Without content.
#circle(radius: 25pt)
```

```
// With content.
#circle[
    #set align(center + horizon)
    Automatically \
    sized to fit.
]
```



Parameters

```
circle(
    radius: length,
    width: autorelative,
    height: autorelativefraction,
    fill: nonecolorgradientpattern,
    stroke: noneautolengthcolorgradientstrokepatterndictionary,
    inset: relativedictionary,
    outset: relativedictionary,
    nonecontent,
) -> content
```

radius

length

Settable

The circle's radius. This is mutually exclusive with `width` and `height`.

Default: `0pt`

width

`auto` or `relative`

Settable

The circle's width. This is mutually exclusive with `radius` and `height`.

In contrast to `radius`, this can be relative to the parent container's width.

Default: `auto`

height

`auto` or `relative` or `fraction`

Settable

The circle's height. This is mutually exclusive with `radius` and `width`.

In contrast to `radius`, this can be relative to the parent container's height.

Default: `auto`

fill

`none` or `color` or `gradient` or `pattern`

Settable

How to fill the circle. See the [rectangle's documentation](#) for more

details.

Default: `none`

stroke

`none` or `auto` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

Settable

How to stroke the circle. See the [rectangle's documentation](#) for more

details.

Default: `auto`

inset

`relative` or `dictionary`

Settable

How much to pad the circle's content. See the [box's documentation](#) for

more details.

Default: `0% + 5pt`

outset

`relative` or `dictionary`

Settable

How much to expand the circle's size without affecting the layout. See the [box's documentation](#) for more details.

Default: `(::)`

body

`none` or `content`

Positional

Settable

The content to place into the circle. The circle expands to fit this content, keeping the 1-1 aspect ratio.

Default: `none`

3.11.2 Color

A color in a specific color space.

Typst supports:

- sRGB through the [rgb function](#)
- Device CMYK through [cmyk function](#)
- D65 Gray through the [luma function](#)
- Oklab through the [oklab function](#)
- Oklch through the [oklch function](#)
- Linear RGB through the [color.linear-rgb function](#)
- HSL through the [color.hsl function](#)

- HSV through the [color.hsv function](#)

Example

```
#rect(fill: aqua)
```

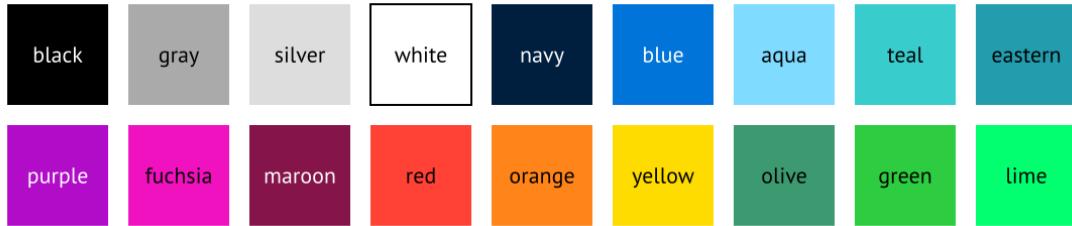


Predefined colors

Typst defines the following built-in colors:

Color	Definition
black	<code>luma(0)</code>
gray	<code>luma(170)</code>
silver	<code>luma(221)</code>
white	<code>luma(255)</code>
navy	<code>rgb("#001f3f")</code>
blue	<code>rgb("#0074d9")</code>
aqua	<code>rgb("#7fdbff")</code>
teal	<code>rgb("#39cccc")</code>
eastern	<code>rgb("#239dad")</code>
purple	<code>rgb("#b10dc9")</code>
fuchsia	<code>rgb("#f012be")</code>
maroon	<code>rgb("#85144b")</code>
red	<code>rgb("#ff4136")</code>
orange	<code>rgb("#ff851b")</code>
yellow	<code>rgb("#ffdc00")</code>
olive	<code>rgb("#3d9970")</code>
green	<code>rgb("#2ecc40")</code>
lime	<code>rgb("#01ff70")</code>

The predefined colors and the most important color constructors are available globally and also in the color type's scope, so you can write either `color.red` or just `red`.



Predefined color maps

Typst also includes a number of preset color maps that can be used for [gradients](#). These are simply arrays of colors defined in the module `color.map`.

```
#circle(fill: gradient.linear(..color.map.crest))
```



Map

turbo

Details

A perceptually uniform rainbow-like color map. Read [this blog post](#) for more details.

cividis

A blue to gray to yellow color map. See [this blog post](#) for more details.

rainbow

Cycles through the full color spectrum. This color map is best used by setting the interpolation color space to [HSL](#). The rainbow gradient is **not suitable** for data visualization because it is not perceptually uniform, so the differences between values become unclear to your readers. It should only be used for decorative purposes.

spectral Red to yellow to blue color map.

viridis A purple to teal to yellow color map.

`inferno` A black to red to yellow color map.

`magma` A black to purple to yellow color map.

`plasma` A purple to pink to yellow color map.

`rocket` A black to red to white color map.

`mako` A black to teal to yellow color map.

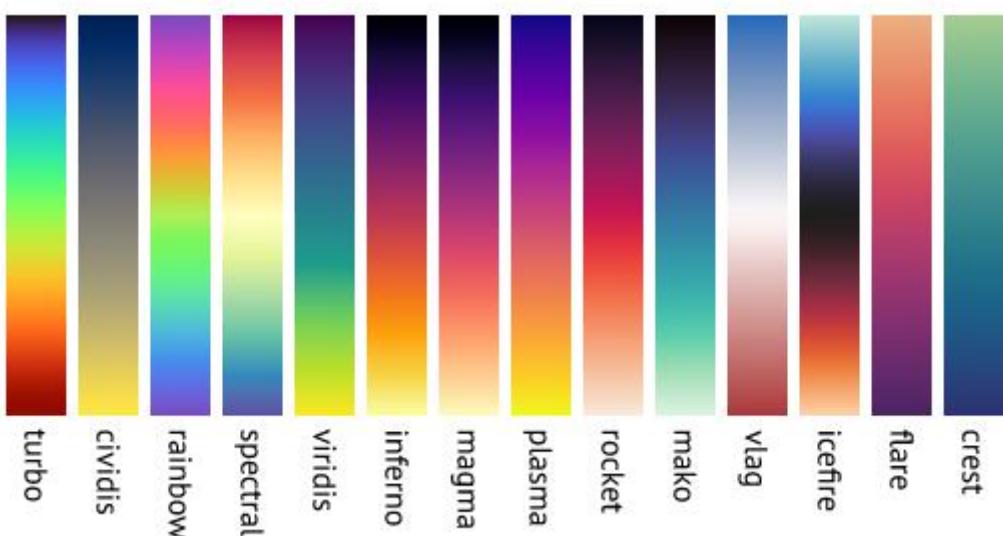
`vlag` A light blue to white to red color map.

`icefire` A light teal to black to yellow color map.

`flare` A orange to purple color map that is perceptually uniform.

`crest` A blue to white to red color map.

Some popular presets are not included because they are not available under a free licence. Others, like [Jet](#), are not included because they are not color blind friendly. Feel free to use or create a package with other presets that are useful to you!



Definitions

luma

Create a grayscale color.

A grayscale color is represented internally by a single `lightness` component.

These components are also available using the [components](#) method.

```
color.luma(  
    intratio,  
    ratio,  
    color,  
    ) -> color  
    #for x in range(250, step: 50) {  
        box(square(fill: luma(x)))  
    }
```

**lightness**

int or ratio

RequiredPositional

The lightness component.

alpha

ratio

RequiredPositional

The alpha component.

color

color

RequiredPositional

Alternatively: The color to convert to grayscale.

If this is given, the `lightness` should not be given.

oklab

Create an [Oklab](#) color.

This color space is well suited for the following use cases:

- Color manipulation such as saturating while keeping perceived hue
- Creating grayscale images with uniform perceived lightness
 - Creating smooth and uniform color transition and gradients

A linear Oklab color is represented internally by an array of four components:

- lightness ([ratio](#))
- a ([float](#) or [ratio](#)). Ratios are relative to `0.4`; meaning `50%` is equal to `0.2`)
- b ([float](#) or [ratio](#)). Ratios are relative to `0.4`; meaning `50%` is equal to `0.2`)
- alpha ([ratio](#))

These components are also available using the [components](#) method.

```
color.oklab(
    ratio,
    floatratio,
    floatratio,
    ratio,
    color,
) -> color
    #square(
        fill: oklab(27%, 20%, -3%, 50%)
    )
```



lightness
ratio
RequiredPositional

The lightness component.

a
float or ratio
RequiredPositional

The a ("green/red") component.

b`float or ratio`***RequiredPositional***

The b ("blue/yellow") component.

alpha`ratio`***RequiredPositional***

The alpha component.

color`color`***RequiredPositional***

Alternatively: The color to convert to Oklab.

If this is given, the individual components should not be given.

oklch

Create an [Oklch](#) color.

This color space is well suited for the following use cases:

- Color manipulation involving lightness, chroma, and hue
- Creating grayscale images with uniform perceived lightness
 - Creating smooth and uniform color transition and gradients

A linear Oklch color is represented internally by an array of four components:

- lightness ([ratio](#))
- chroma ([float](#) or [ratio](#)). Ratios are relative to `0.4`; meaning `50%` is equal to `0.2`)
- hue ([angle](#))
- alpha ([ratio](#))

These components are also available using the [components](#) method.

```
color.oklch()
```

```

ratio,
floatratio,
angle,
ratio,
color,
) -> color
#square(
  fill: oklch(40%, 0.2, 160deg, 50%)
)

```

**lightness**ratio*RequiredPositional*

The lightness component.

chromafloat or ratio*RequiredPositional*

The chroma component.

hueangle*RequiredPositional*

The hue component.

alpharatio*RequiredPositional*

The alpha component.

colorcolor*RequiredPositional*

Alternatively: The color to convert to Oklch.

If this is given, the individual components should not be given.

linear-rgb

Create an RGB(A) color with linear luma.

This color space is similar to sRGB, but with the distinction that the color component are not gamma corrected. This makes it easier to perform color operations such as blending and interpolation. Although, you should prefer to use the [oklab function](#) for these.

A linear RGB(A) color is represented internally by an array of four components:

- red ([ratio](#))
- green ([ratio](#))
- blue ([ratio](#))
- alpha ([ratio](#))

These components are also available using the [components](#) method.

```
color.linear-rgb(
    intratio,
    intratio,
    intratio,
    intratio,
    color,
) -> color
    #square(fill: color.linear-rgb(
        30%, 50%, 10%,
    ))
```



red

[int or ratio](#)

RequiredPositional

The red component.

greenint or ratio*RequiredPositional*

The green component.

blueint or ratio*RequiredPositional*

The blue component.

alphaint or ratio*RequiredPositional*

The alpha component.

colorcolor*RequiredPositional*

Alternatively: The color to convert to linear RGB(A).

If this is given, the individual components should not be given.

rgb

Create an RGB(A) color.

The color is specified in the sRGB color space.

An RGB(A) color is represented internally by an array of four components:

•red ([ratio](#))

•green ([ratio](#))

•blue ([ratio](#))

•alpha ([ratio](#))

These components are also available using the [components](#) method.

```
color.rgb(  
    intratio,
```

```

intratio,
intratio,
intratio,
str,
color,
) -> color
#square(fill: rgb("#b1f2eb"))
#square(fill: rgb(87, 127, 230))
#square(fill: rgb(25%, 13%, 65%))

```

**red**`int or ratio`*RequiredPositional*

The red component.

green`int or ratio`*RequiredPositional*

The green component.

blue`int or ratio`*RequiredPositional*

The blue component.

alpha`int or ratio`*RequiredPositional*

The alpha component.

hex

str

RequiredPositional

Alternatively: The color in hexadecimal notation.

Accepts three, four, six or eight hexadecimal digits and optionally a leading hash.

If this is given, the individual components should not be given.

[View example](#)

```
#text(16pt, rgb("#239dad")) [  
    *Typst*  
]
```

Typst

color

color

RequiredPositional

Alternatively: The color to convert to RGB(a).

If this is given, the individual components should not be given.

cmyk

Create a CMYK color.

This is useful if you want to target a specific printer. The conversion to RGB for display preview might differ from how your printer reproduces the color.

A CMYK color is represented internally by an array of four components:

- cyan ([ratio](#))

- magenta ([ratio](#))

- yellow ([ratio](#))

- **key** ([ratio](#))

These components are also available using the [components](#) method.

Note that CMYK colors are not currently supported when PDF/A output is enabled.

```
color.cmyk(
    ratio,
    ratio,
    ratio,
    ratio,
    color,
) -> color
    #square(
        fill: cmyk(27%, 0%, 3%, 5%)
    )
```



cyan
`ratio`
RequiredPositional

The cyan component.

magenta
`ratio`
RequiredPositional

The magenta component.

yellow
`ratio`
RequiredPositional

The yellow component.

key
`ratio`
RequiredPositional

The key component.

`color``color`*RequiredPositional*

Alternatively: The color to convert to CMYK.

If this is given, the individual components should not be given.

`hsl`

Create an HSL color.

This color space is useful for specifying colors by hue, saturation and lightness.

It is also useful for color manipulation, such as saturating while keeping perceived hue.

An HSL color is represented internally by an array of four components:

- hue ([angle](#))
- saturation ([ratio](#))
- lightness ([ratio](#))
- alpha ([ratio](#))

These components are also available using the [components](#) method.

```
color.hsl(
    angle,
    intratio,
    intratio,
    intratio,
    color,
    ) -> color
        #square(
            fill: color.hsl(30deg, 50%, 60%)
        )
```



hueangle*RequiredPositional*

The hue angle.

saturationint or ratio*RequiredPositional*

The saturation component.

lightnessint or ratio*RequiredPositional*

The lightness component.

alphaint or ratio*RequiredPositional*

The alpha component.

colorcolor*RequiredPositional*

Alternatively: The color to convert to HSL.

If this is given, the individual components should not be given.

hsv

Create an HSV color.

This color space is useful for specifying colors by hue, saturation and value. It is also useful for color manipulation, such as saturating while keeping perceived hue.

An HSV color is represented internally by an array of four components:

- hue (angle)

- saturation (ratio)

- **value** ([ratio](#))

- **alpha** ([ratio](#))

These components are also available using the [components](#) method.

```
color.hsv(
    angle,
    intratio,
    intratio,
    intratio,
    color,
) -> color
    #square(
        fill: color.hsv(30deg, 50%, 60%)
    )
```



hue

angle

RequiredPositional

The hue angle.

saturation

int or ratio

RequiredPositional

The saturation component.

value

int or ratio

RequiredPositional

The value component.

alpha

int or ratio

RequiredPositional

The alpha component.

color**color****RequiredPositional**

Alternatively: The color to convert to HSL.

If this is given, the individual components should not be given.

components

Extracts the components of this color.

The size and values of this array depends on the color space. You can obtain the color space using [space](#). Below is a table of the color spaces and their components:

Color space	C1	C2	C3	C4
luma	Lightness			
oklab	Lightness ^a	b	Alpha	
oklch	Lightness Chroma	Hue	Alpha	
linear-rgb	Red	Green	Blue	Alpha
rgb	Red	Green	Blue	Alpha
cmyk	Cyan	Magenta	Yellow	Key
hsl	Hue	Saturation	Lightness	Alpha
hsv	Hue	Saturation	Value	Alpha

For the meaning and type of each individual value, see the documentation of the corresponding color space. The alpha component is optional and only included if the `alpha` argument is `true`. The length of the returned array depends on the number of components and whether the alpha component is included.

```
self.components(
alpha: bool
) -> array
// note that the alpha component is included by default
#rgb(40%, 60%, 80%).components()
```

(40%, 60%, 80%, 100%)

alpha
bool

Whether to include the alpha component.

Default: `true`

space

Returns the constructor function for this color's space:

- [luma](#)
 - [oklab](#)
 - [oklch](#)
 - [linear-rgb](#)
 - [rgb](#)
 - [cmyk](#)
 - [hsl](#)
 - [hsv](#)
- ```
self.space() -> any
let color = cmyk(1%, 2%, 3%, 4%)
#(color.space() == cmyk)
```

`true`

**to-hex**

Returns the color's RGB(A) hex representation (such as `#ffaa32` or `#020304fe`).

The alpha component (last two digits in `#020304fe`) is omitted if it is equal to `ff` (255 / 100%).

```
self.to-hex() -> str
lighten
```

Lightens a color by a given factor.

```
self.lighten(
 ratio
) -> color
factor
 ratio
RequiredPositional
```

The factor to lighten the color by.

**darken**

Darkens a color by a given factor.

```
self.darken(
 ratio
) -> color
factor
 ratio
RequiredPositional
```

The factor to darken the color by.

**saturate**

Increases the saturation of a color by a given factor.

```
self.saturate(
 ratio
) -> color
factor
 ratio
RequiredPositional
```

The factor to saturate the color by.

**desaturate**

Decreases the saturation of a color by a given factor.

```
self.desaturate(
 ratio
```

```
) -> color
factor
ratio
RequiredPositional
```

The factor to desaturate the color by.

**negate**

Produces the complementary color using a provided color space. You can think of it as the opposite side on a color wheel.

```
self.negate(
space: any
) -> color
#square(fill: yellow)
#square(fill: yellow.negate())
#square(fill: yellow.negate(space: rgb))
```



**space**

any

The color space used for the transformation. By default, a perceptual color space is used.

Default: oklab

**rotate**

Rotates the hue of the color by a given angle.

```
self.rotate(
angle,
space: any,
) -> color
```

**angle**angle**RequiredPositional**

The angle to rotate the hue by.

**space**any

The color space used to rotate. By default, this happens in a perceptual color space ([oklch](#)).

Default: `oklch`

**mix**

Create a color by mixing two or more colors.

In color spaces with a hue component (hsl, hsv, oklch), only two colors can be mixed at once. Mixing more than two colors in such a space will result in an error!

```
color.mix(
..colorarray,
space: any,
) -> color
#set block(height: 20pt, width: 100%)
#block(fill: red.mix(blue))
#block(fill: red.mix(blue, space: rgb))
#block(fill: color.mix(red, blue, white))
#block(fill: color.mix((red, 70%), (blue, 30%)))
```



**colors**`color` or `array`**RequiredPositional****Variadic**

The colors, optionally with weights, specified as a pair (array of length two) of color and weight (float or ratio).

The weights do not need to add to `100%`, they are relative to the sum of all weights.

**space**`any`

The color space to mix in. By default, this happens in a perceptual color space ([oklab](#)).

Default: `oklab`

**transparentize**

Makes a color more transparent by a given factor.

This method is relative to the existing alpha value. If the scale is positive,

calculates `alpha - alpha * scale`. Negative scales behave like

```
color.opacify(-scale).
self.transparentize(
ratio
) -> color
#block(fill: red) [opaque]
#block(fill: red.transparentize(50%)) [half red]
#block(fill: red.transparentize(75%)) [quarter red]
```

**opaque****half red****quarter red**

**scale**ratio*RequiredPositional*

The factor to change the alpha value by.

**opacify**

Makes a color more opaque by a given scale.

This method is relative to the existing alpha value. If the scale is positive,

**calculates** `alpha + scale - alpha * scale`. Negative scales behave like

`color.transparentize(-scale)`.

```
self.opacify(
```

ratio

```
) -> color
```

```
#let half-red = red.transparentize(50%)
```

```
#block(fill: half-red.opacify(100%)) [opaque]
```

```
#block(fill: half-red.opacify(50%)) [three quarters red]
```

```
#block(fill: half-red.opacify(-50%)) [one quarter red]
```

**opaque****three quarters red****one quarter red****scale**ratio*RequiredPositional*

The scale to change the alpha value by.

### 3.11.3 Ellipse

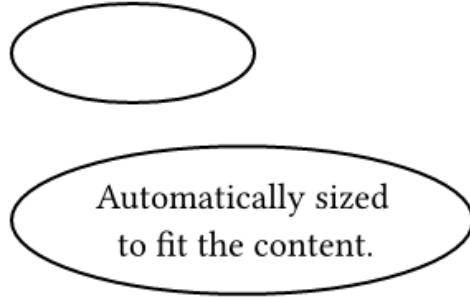
An ellipse with optional content.

#### Example

```
// Without content.
#ellipse(width: 35%, height: 30pt)

// With content.
```

```
#ellipse[
 #set align(center)
 Automatically sized \
 to fit the content.
]
```



## Parameters

```
ellipse(
 width: autorelative,
 height: autorelativefraction,
 fill: nonecolorgradientpattern,
 stroke: noneautolengthcolorgradientstrokepatterndictionary,
 inset: relativedictionary,
 outset: relativedictionary,
 nonecontent,
) -> content
width
auto or relative
```

### *Settable*

The ellipse's width, relative to its parent container.

Default: `auto`

### `height`

`auto` or `relative` or `fraction`

### *Settable*

The ellipse's height, relative to its parent container.

Default: `auto`

### `fill`

`none` or `color` or `gradient` or `pattern`

**Settable**

How to fill the ellipse. See the [rectangle's documentation](#) for more details.

Default: `none`

**stroke**

`none` or `auto` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

**Settable**

How to stroke the ellipse. See the [rectangle's documentation](#) for more details.

Default: `auto`

**inset**

`relative` or `dictionary`

**Settable**

How much to pad the ellipse's content. See the [box's documentation](#) for more details.

Default: `0% + 5pt`

**outset**

`relative` or `dictionary`

**Settable**

How much to expand the ellipse's size without affecting the layout. See the [box's documentation](#) for more details.

Default: `(::)`

**body**

`none` or `content`

**Positional**

**Settable**

The content to place into the ellipse.

When this is omitted, the ellipse takes on a default size of at most `45pt` by `30pt`.

Default: `none`

## 3.11.4 Gradient

## 3.11.5 Image

A raster or vector graphic.

You can wrap the image in a [figure](#) to give it a number and caption.

Like most elements, images are *block-level* by default and thus do not integrate themselves into adjacent paragraphs. To force an image to become inline, put it into a [box](#).

## Example

```
#figure(
 image("molecular.jpg", width: 80%),
 caption: [
 A step in the molecular testing
 pipeline of our lab.
],
)
```



Figure 1: A step in the molecular testing pipeline of our lab.

## Parameters

```
image (
 str,
 format: autostr,
 width: autorelative,
 height: autorelativefraction,
 alt: nonestr,
 fit: str,
) -> content
```

**path**  
**str**  
*RequiredPositional*

Path to an image file

For more details, see the [Paths section](#).

**format**

auto or str

*Settable*

The image's format. Detected automatically by default.

Supported formats are PNG, JPEG, GIF, and SVG. Using a PDF as an image is [not currently supported](#).

| Variant | Details |
|---------|---------|
|---------|---------|

`"png"` Raster format for illustrations and transparent graphics.

`"jpg"` Lossy raster format suitable for photos.

`"gif"` Raster format that is typically used for short animated clips.

`"svg"` The vector graphics format of the web.

Default: `auto`

**width**

`auto` or `relative`

*Settable*

The width of the image.

Default: `auto`

**height**

`auto` or `relative` or `fraction`

*Settable*

The height of the image.

Default: `auto`

**alt**

`none` or `str`

*Settable*

A text describing the image.

Default: `none`

**fit**

`str`

*Settable*

How the image should adjust itself to a given area (the area is defined by

the `width` and `height` fields). Note that `fit` doesn't visually change

anything if the area's aspect ratio is the same as the image's one.

| Variant   | Details                                                                                                                                                                          |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "cover"   | The image should completely cover the area (preserves aspect ratio by cropping the image only horizontally or vertically). This is the default.                                  |
| "contain" | The image should be fully contained in the area (preserves aspect ratio; doesn't crop the image; one dimension can be narrower than specified).                                  |
| "stretch" | The image should be stretched so that it exactly fills the area, even if this means that the image will be distorted (doesn't preserve aspect ratio and doesn't crop the image). |

Default: "cover"

[View example](#)

```
#set page(width: 300pt, height: 50pt, margin: 10pt)
#image("tiger.jpg", width: 100%, fit: "cover")
#image("tiger.jpg", width: 100%, fit: "contain")
#image("tiger.jpg", width: 100%, fit: "stretch")
```



# Definitions

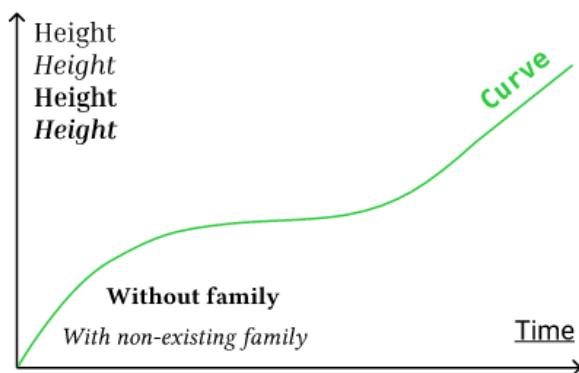
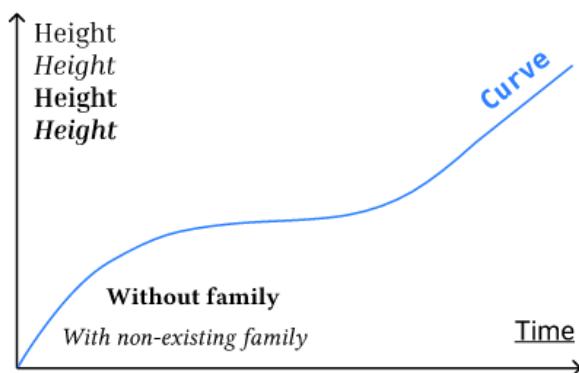
## decode

Decode a raster or vector graphic from bytes or a string.

```
image.decode(
 strbytes,
 format: autostr,
 width: autorelative,
 height: autorelativefraction,
 alt: nonestr,
 fit: str,
) -> content

let original = read("diagram.svg")
let changed = original.replace(
 "#2B80FF", // blue
 green.to-hex(),
)

#image.decode(original)
#image.decode(changed)
```



**data**`str or bytes`**RequiredPositional**

The data to decode as an image. Can be a string for SVGs.

**format**`auto or str`

The image's format. Detected automatically by default.

**Variant****Details**

`"png"` Raster format for illustrations and transparent graphics.

`"jpg"` Lossy raster format suitable for photos.

`"gif"` Raster format that is typically used for short animated clips.

`"svg"` The vector graphics format of the web.

**width**`auto or relative`

The width of the image.

**height**`auto or relative or fraction`

The height of the image.

**alt**`none or str`

A text describing the image.

**fit**`str`

How the image should adjust itself to a given area.

**Variant****Details**

The image should completely cover the area (preserves aspect ratio "cover" by cropping the image only horizontally or vertically). This is the default.

The image should be fully contained in the area (preserves aspect "contain" ratio; doesn't crop the image; one dimension can be narrower than specified).

The image should be stretched so that it exactly fills the area, even "stretch" if this means that the image will be distorted (doesn't preserve aspect ratio and doesn't crop the image).

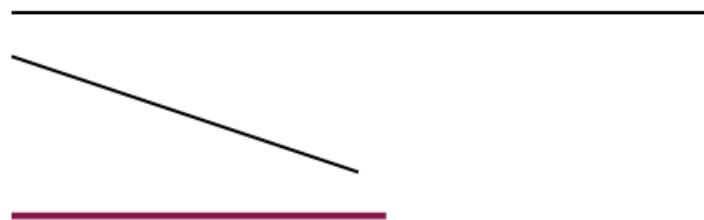
## 3.11.6 Line

A line from one point to another.

### Example

```
#set page(height: 100pt)

#line(length: 100%)
#line(end: (50%, 50%))
#line(
 length: 4cm,
 stroke: 2pt + maroon,
)
```



## Parameters

```
line(
 start: array,
 end: nonearray,
 length: relative,
 angle: angle,
 stroke: lengthcolorgradientstrokepatterndictionary,
) -> content
```

**start**  
**array**  
**Settable**

The start point of the line.

Must be an array of exactly two relative lengths.

Default: (0% + 0pt, 0% + 0pt)

**end**

none or array

**Settable**

The offset from `start` where the line ends.

Default: none

**length**

relative

**Settable**

The line's length. This is only respected if `end` is none.

Default: 0% + 30pt

**angle**

angle

**Settable**

The angle at which the line points away from the origin. This is only respected if `end` is none.

Default: 0deg

**stroke**

length or color or gradient or stroke or pattern or dictionary

**Settable**

How to [stroke](#) the line.

Default: `1pt + black`

[View example](#)

```
#set line(length: 100%)
#stack(
 spacing: 1em,
 line(stroke: 2pt + red),
 line(stroke: (paint: blue, thickness: 4pt, cap: "round")),
 line(stroke: (paint: blue, thickness: 1pt, dash: "dashed")),
 line(stroke: (paint: blue, thickness: 1pt, dash: ("dot", 2pt,
4pt, 2pt))),
)
```



## 3.11.7 Path

A path through a list of points, connected by Bezier curves.

### Example

```
#path(
 fill: blue.lighten(80%),
 stroke: blue,
 closed: true,
 (0pt, 50pt),
 (100%, 50pt),
 ((50%, 0pt), (40pt, 0pt)),
)
```



## Parameters

```
path(
 fill: none|color|gradient|pattern,
 fill-rule: str,
 stroke: none|auto|length|color|gradient|stroke|pattern|dictionary,
 closed: bool,
 ...array,
) -> content
```

**fill**

`none` or `color` or `gradient` or `pattern`

**Settable**

How to fill the path.

When setting a fill, the default stroke disappears. To create a rectangle with both fill and stroke, you have to configure both.

Default: `none`

**fill-rule**

`str`

**Settable**

The drawing rule used to fill the path.

**Variant**

**Details**

`"non-zero"` Specifies that "inside" is computed by a non-zero sum of signed edge crossings.

`"even-odd"` Specifies that "inside" is computed by an odd number of edge crossings.

Default: `"non-zero"`

**View example**

```
// We use `with` to get a new
// function that has the common
// arguments pre-applied.
#let star = path.with(
 fill: red,
```

```

closed: true,
(25pt, 0pt),
(10pt, 50pt),
(50pt, 20pt),
(0pt, 20pt),
(40pt, 50pt),
)

#star(fill-rule: "non-zero")
#star(fill-rule: "even-odd")

```

**stroke**

[none](#) or [auto](#) or [length](#) or [color](#) or [gradient](#) or [stroke](#) or [pattern](#) or [dictionary](#)

*Settable*

How to [stroke](#) the path. This can be:

Can be set to [none](#) to disable the stroke or to [auto](#) for a stroke of [1pt](#) black if and if only if no fill is given.

Default: [auto](#)

**closed**

[bool](#)

*Settable*

Whether to close this path with one last bezier curve. This curve will takes into account the adjacent control points. If you want to close with a straight line, simply add one last point that's the same as the start point.

Default: `false`

**vertices**

array

**RequiredPositional**

**Variadic**

The vertices of the path.

Each vertex can be defined in 3 ways:

- A regular point, as given to the [line](#) or [polygon](#) function.
- An array of two points, the first being the vertex and the second being the control point. The control point is expressed relative to the vertex and is mirrored to get the second control point. The given control point is the one that affects the curve coming *into* this vertex (even for the first point). The mirrored control point affects the curve going out of this vertex.
- An array of three points, the first being the vertex and the next being the control points (control point for curves coming in and out, respectively).

## 3.11.8 Pattern

A repeating pattern fill.

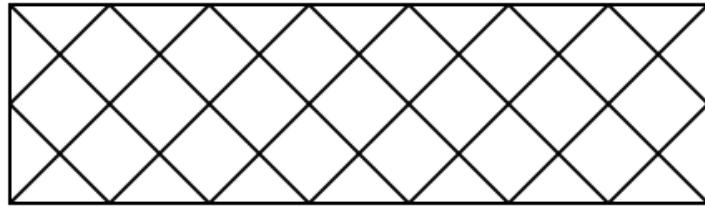
Typst supports the most common pattern type of tiled patterns, where a pattern is repeated in a grid-like fashion, covering the entire area of an element that is filled or stroked. The pattern is defined by a tile size and a body defining the content of each cell. You can also add horizontal or vertical spacing between the cells of the pattern.

## Examples

```
#let pat = pattern(size: (30pt, 30pt)) [
 #place(line(start: (0%, 0%), end: (100%, 100%)))
```

```
#place(line(start: (0%, 100%), end: (100%, 0%)))
]

#rect(fill: pat, width: 100%, height: 60pt, stroke: 1pt)
```



Patterns are also supported on text, but only when setting the [relativeness](#) to either `auto` (the default value) or `"parent"`. To create word-by-word or glyph-by-glyph patterns, you can wrap the words or characters of your text in [boxes](#) manually or through a [show rule](#).

```
#let pat = pattern(
 size: (30pt, 30pt),
 relative: "parent",
 square(
 size: 30pt,
 fill: gradient
 .conic(..color.map.rainbow),
)
)

#set text(fill: pat)
#lorem(10)
```

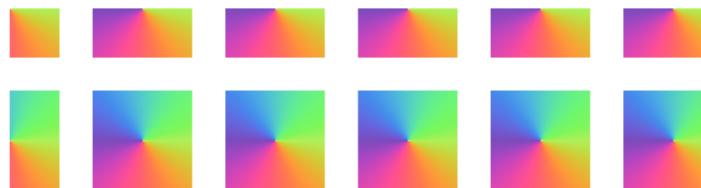
**Lorem ipsum dolor sit amet, consectetur  
  adipiscing elit, sed do.**

You can also space the elements further or closer apart using the [spacing](#) feature of the pattern. If the spacing is lower than the size of the

pattern, the pattern will overlap. If it is higher, the pattern will have gaps of the same color as the background of the pattern.

```
#let pat = pattern(
 size: (30pt, 30pt),
 spacing: (10pt, 10pt),
 relative: "parent",
 square(
 size: 30pt,
 fill: gradient
 .conic(..color.map.rainbow),
),
)

#rect(
 width: 100%,
 height: 60pt,
 fill: pat,
)
```



## Relativeness

The location of the starting point of the pattern is dependent on the dimensions of a container. This container can either be the shape that it is being painted on, or the closest surrounding container. This is controlled by the `relative` argument of a pattern constructor. By default, patterns are relative to the shape they are being painted on, unless the pattern is applied on text, in which case they are relative to the closest ancestor container.

Typst determines the ancestor container as follows:

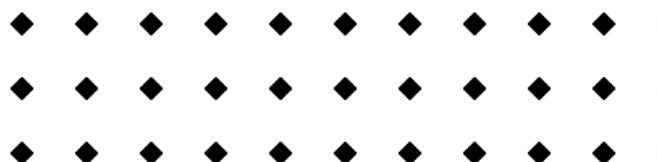
- For shapes that are placed at the root/top level of the document, the closest ancestor is the page itself.
- For other shapes, the ancestor is the innermost `block` or `box` that contains the shape. This includes the boxes and blocks that are implicitly created by show rules and elements. For example, a `rotate` will not affect the parent of a gradient, but a `grid` will.

## Constructor

Construct a new pattern.

```
pattern(
 size: autoarray,
 spacing: array,
 relative: autostr,
 content,
) -> pattern
 #let pat = pattern(
 size: (20pt, 20pt),
 relative: "parent",
 place(
 dx: 5pt,
 dy: 5pt,
 rotate(45deg, square(
 size: 5pt,
 fill: black,
)),
),
)
)

#rect(width: 100%, height: 60pt, fill: pat)
```



**size**`auto` or `array`

The bounding box of each cell of the pattern.

Default: `auto`

**spacing**`array`

The spacing between cells of the pattern.

Default: `(0pt, 0pt)`

**relative**`auto` or `str`

The [relative placement](#) of the pattern.

For an element placed at the root/top level of the document, the parent is the page itself. For other elements, the parent is the innermost block, box, column, grid, or stack that contains the element.

**Variant****Details**

`"self"` The gradient is relative to itself (its own bounding box).

`"parent"` The gradient is relative to its parent (the parent's bounding box).

Default: `auto`

**body**`content`

*RequiredPositional*

The content of each cell of the pattern.

## 3.11.9 Polygon

## 3.11.10 Rectangle

## Example

```
// Without content.
#rect(width: 35%, height: 30pt)

// With content.
#rect[
 Automatically sized \
 to fit the content.
]
```



Automatically sized  
to fit the content.

## Parameters

```
rect(
 width: autorelative,
 height: autorelativefraction,
 fill: nonecolorgradientpattern,
 stroke: noneautolengthcolorgradientstrokepatterndictionary,
 radius: relativedictionary,
 inset: relativedictionary,
 outset: relativedictionary,
 nonecontent,
) -> content
```

**width**

auto or relative

*Settable*

The rectangle's width, relative to its parent container.

Default: auto

**height**

auto or relative or fraction

*Settable*

The rectangle's height, relative to its parent container.

Default: `auto`

### **fill**

`none` or `color` or `gradient` or `pattern`

#### **Settable**

How to fill the rectangle.

When setting a fill, the default stroke disappears. To create a rectangle with

both fill and stroke, you have to configure both.

Default: `none`

View example

`#rect (fill: blue)`



### **stroke**

`none` or `auto` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

#### **Settable**

How to stroke the rectangle. This can be:

- `none` to disable stroking
- `auto` for a stroke of `1pt + black` if and if only if no fill is given.
- Any kind of `stroke`
- A dictionary describing the stroke for each side individually. The dictionary can contain the following keys in order of precedence:

`otop`: The top stroke.

`oright`: The right stroke.

`obottom`: The bottom stroke.

`oleft`: The left stroke.

`ox`: The horizontal stroke.

`oy`: The vertical stroke.

`orest`: The stroke on all sides except those for which the dictionary explicitly sets a size.

Default: `auto`

[View example](#)

```
#stack(
 dir: ltr,
 spacing: 1fr,
 rect(stroke: red),
 rect(stroke: 2pt),
 rect(stroke: 2pt + red),
)
```



### `radius`

[relative](#) or [dictionary](#)

*Settable*

How much to round the rectangle's corners, relative to the minimum of

the width and height divided by two. This can be:

- A relative length for a uniform corner radius.
- A dictionary: With a dictionary, the stroke for each side can be set individually.

The dictionary can contain the following keys in order of precedence:

`otop-left`: The top-left corner radius.

`otop-right`: The top-right corner radius.

`obottom-right`: The bottom-right corner radius.

`obottom-left`: The bottom-left corner radius.

`oleft`: The top-left and bottom-left corner radii.

`otop`: The top-left and top-right corner radii.

`oright`: The top-right and bottom-right corner radii.

`obottom`: The bottom-left and bottom-right corner radii.

`orest`: The radii for all corners except those for which the dictionary explicitly

sets a size.

Default: `(:)`

[View example](#)

```
#set rect(stroke: 4pt)
#rect(
 radius: (
 left: 5pt,
 top-right: 20pt,
 bottom-right: 10pt,
),
 stroke: (
 left: red,
 top: yellow,
 right: green,
 bottom: blue,
),
)
```



**inset**

[relative](#) or [dictionary](#)

**Settable**

How much to pad the rectangle's content. See the [box's documentation](#) for more details.

Default: `0% + 5pt`

**outset**

[relative](#) or [dictionary](#)

*Settable*

How much to expand the rectangle's size without affecting the layout.

See the [box's documentation](#) for more details.

Default: `(::)`

**body**

[none](#) or [content](#)

*Positional*

*Settable*

The content to place into the rectangle.

When this is omitted, the rectangle takes on a default size of at most `45pt` by `30pt`.

Default: `none`

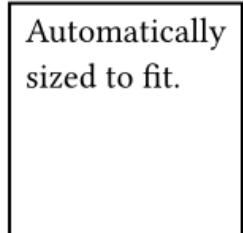
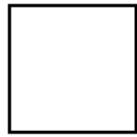
### 3.11.11 Square

A square with optional content.

#### Example

```
// Without content.
#square(size: 40pt)

// With content.
#square[
 Automatically \
 sized to fit.
]
```



## Parameters

```
square (
 size: autolength,
 width: autorelative,
 height: autorelativedefraction,
 fill: nonecolorgradientpattern,
 stroke: noneautolengthcolorgradientstrokepatterndictionary,
 radius: relativedictionary,
 inset: relativedictionary,
 outset: relativedictionary,
 nonecontent,
) -> content
```

**size**

auto or length

*Settable*

The square's side length. This is mutually exclusive with `width` and `height`.

Default: `auto`

**width**

auto or relative

*Settable*

The square's width. This is mutually exclusive with `size` and `height`.

In contrast to `size`, this can be relative to the parent container's width.

Default: `auto`

**height**

`auto` or `relative` or `fraction`

**Settable**

The square's height. This is mutually exclusive with `size` and `width`.

In contrast to `size`, this can be relative to the parent container's height.

Default: `auto`

**fill**

`none` or `color` or `gradient` or `pattern`

**Settable**

How to fill the square. See the [rectangle's documentation](#) for more details.

Default: `none`

**stroke**

`none` or `auto` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

**Settable**

How to stroke the square. See the [rectangle's documentation](#) for more details.

Default: `auto`

**radius**

`relative` or `dictionary`

**Settable**

How much to round the square's corners. See the [rectangle's documentation](#) for more details.

Default: `(:)`

**inset**

`relative` or `dictionary`

**Settable**

How much to pad the square's content. See the [box's documentation](#) for more details.

Default: `0% + 5pt`

#### **outset**

[relative](#) or [dictionary](#)

#### *Settable*

How much to expand the square's size without affecting the layout. See the [box's documentation](#) for more details.

Default: `(::)`

#### **body**

[none](#) or [content](#)

#### *Positional*

#### *Settable*

The content to place into the square. The square expands to fit this content, keeping the 1-1 aspect ratio.

When this is omitted, the square takes on a default size of at most `30pt`.

Default: `none`

## 3.11.12 Stroke

Defines how to draw a line.

A stroke has a *paint* (a solid color or gradient), a *thickness*, a line *cap*, a line *join*, a *miter limit*, and a *dash* pattern. All of these values are optional and have sensible defaults.

## Example

```
#set line(length: 100%)
#stack(
 spacing: 1em,
```

```

line(stroke: 2pt + red),
line(stroke: (paint: blue, thickness: 4pt, cap: "round")),
line(stroke: (paint: blue, thickness: 1pt, dash:
"dashed")),
line(stroke: 2pt + gradient.linear(..color.map.rainbow)),
)

```



## Simple strokes

You can create a simple solid stroke from a color, a thickness, or a combination of the two. Specifically, wherever a stroke is expected you can pass any of the following values:

- A length specifying the stroke's thickness. The color is inherited, defaulting to black.
- A color to use for the stroke. The thickness is inherited, defaulting to 1pt.
- A stroke combined from color and thickness using the + operator as in 2pt + red.

For full control, you can also provide a [dictionary](#) or a `stroke` object to any function that expects a stroke. The dictionary's keys may include any of the parameters for the constructor function, shown below.

## Fields

On a stroke object, you can access any of the fields listed in the constructor function. For example, `(2pt + blue).thickness` is 2pt. Meanwhile,

`stroke(red)`.`cap` is `auto` because it's unspecified. Fields set to `auto` are inherited.

## Constructor

Converts a value to a stroke or constructs a stroke with the given parameters.

Note that in most cases you do not need to convert values to strokes in order to use them, as they will be converted automatically. However, this constructor can be useful to ensure a value has all the fields of a stroke.

```
stroke(
 autocolorgradientpattern,
 autolength,
 autostr,
 autostr,
 noneautostrarraydictionary,
 autofloat,
) -> stroke
 #let my-func(x) = {
 x = stroke(x) // Convert to a stroke
 [Stroke has thickness #x.thickness.]
 }
 #my-func(3pt) \
 #my-func(red) \
 #my-func(stroke(cap: "round", thickness: 1pt))
```

Stroke has thickness **3pt**.  
 Stroke has thickness **auto**.  
 Stroke has thickness **1pt**.

**paint**  
auto or color or gradient or pattern

*RequiredPositional*

The color or gradient to use for the stroke.

If set to `auto`, the value is inherited, defaulting to `black`.

### **thickness**

`auto` or `length`

#### *RequiredPositional*

The stroke's thickness.

If set to `auto`, the value is inherited, defaulting to `1pt`.

### **cap**

`auto` or `str`

#### *RequiredPositional*

How the ends of the stroke are rendered.

If set to `auto`, the value is inherited, defaulting to `"butt"`.

| <b>Variant</b>        | <b>Details</b>                                             |
|-----------------------|------------------------------------------------------------|
| <code>"butt"</code>   | Square stroke cap with the edge at the stroke's end point. |
| <code>"round"</code>  | Circular stroke cap centered at the stroke's end point.    |
| <code>"square"</code> | Square stroke cap centered at the stroke's end point.      |

### **join**

`auto` or `str`

#### *RequiredPositional*

How sharp turns are rendered.

If set to `auto`, the value is inherited, defaulting to `"miter"`.

| <b>Variant</b>       | <b>Details</b>                                                                                    |
|----------------------|---------------------------------------------------------------------------------------------------|
| <code>"miter"</code> | Segments are joined with sharp edges. Sharp bends exceeding the miter limit are bevelled instead. |
| <code>"round"</code> | Segments are joined with circular corners.                                                        |

Segments are joined with a bevel (a straight edge connecting the `"bevel"` butts of the joined segments).

**dash**

`none` or `auto` or `str` or `array` or `dictionary`

*RequiredPositional*

The dash pattern to use. This can be:

- One of the predefined patterns:

- `"solid"` or `none`
- `"dotted"`
- `"densely-dotted"`
- `"loosely-dotted"`
- `"dashed"`
- `"densely-dashed"`
- `"loosely-dashed"`
- `"dash-dotted"`
- `"densely-dash-dotted"`
- `"loosely-dash-dotted"`

- An `array` with alternating lengths for dashes and gaps. You can also use the string `"dot"` for a length equal to the line thickness.

- A `dictionary` with the keys `array` (same as the array above), and `phase` (of type `length`), which defines where in the pattern to start drawing.

If set to `auto`, the value is inherited, defaulting to `none`.

[View options](#)

| <b>Variant</b> | <b>Details</b> |
|----------------|----------------|
|----------------|----------------|

|                               |  |
|-------------------------------|--|
| <code>"solid"</code>          |  |
| <code>"dotted"</code>         |  |
| <code>"densely-dotted"</code> |  |
| <code>"loosely-dotted"</code> |  |

```
"dashed"
"densely-dashed"
"loosely-dashed"
"dash-dotted"
"densely-dash-dotted"
"loosely-dash-dotted"
```

[View example](#)

```
#set line(length: 100%, stroke: 2pt)
#stack(
 spacing: 1em,
 line(stroke: (dash: "dashed")),
 line(stroke: (dash: (10pt, 5pt, "dot", 5pt))),
 line(stroke: (dash: (array: (10pt, 5pt, "dot", 5pt), phase:
10pt))),
)
```



### **miter-limit**

auto or float

*RequiredPositional*

Number at which protruding sharp bends are rendered with a bevel instead or a miter join. The higher the number, the sharper an angle can be before it is bevelled. Only applicable if `join` is "miter".

Specifically, the miter limit is the maximum ratio between the corner's protrusion length and the stroke's thickness.

If set to `auto`, the value is inherited, defaulting to `4.0`.

[View example](#)

```
#let points = ((15pt, 0pt), (0pt, 30pt), (30pt, 30pt), (10pt,
20pt))
```

```
#set path(stroke: 6pt + blue)
#stack(
 dir: ltr,
 spacing: 1cm,
 path(stroke: (miter-limit: 1), ..points),
 path(stroke: (miter-limit: 4), ..points),
 path(stroke: (miter-limit: 5), ..points),
)
)
```



## 3.12 Introspection

Interactions between document parts.

This category is home to Typst's introspection capabilities: With the `counter` function, you can access and manipulate page, section, figure, and equation counters or create custom ones. Meanwhile, the `query` function lets you search for elements in the document to construct things like a list of figures or headers which show the current chapter title.

Most of the functions are *contextual*. It is recommended to read the chapter on [context](#) before continuing here.

## Definitions

- [counter](#) Counts through pages, elements, and more.
- [here](#) Provides the current location in the document.
- [locate](#) Determines the location of an element in the document.
- [location](#) Identifies an element in the document.

- [metadata](#) Exposes a value to the query system without producing visible content.

- [query](#) Finds elements in the document.
- [state](#) Manages stateful parts of your document.

## 3.12.1 Counter

Counts through pages, elements, and more.

With the counter function, you can access and modify counters for pages, headings, figures, and more. Moreover, you can define custom counters for other things you want to count.

Since counters change throughout the course of the document, their current value is *contextual*. It is recommended to read the chapter on [context](#) before continuing here.

## Accessing a counter

To access the raw value of a counter, we can use the [get](#) function. This function returns an [array](#): Counters can have multiple levels (in the case of headings for sections, subsections, and so on), and each item in the array corresponds to one level.

```
#set heading(numbering: "1.")

= Introduction
Raw value of heading counter is
#context counter(heading).get()
```

# 1. Introduction

Raw value of heading counter is (1,)

## Displaying a counter

Often, we want to display the value of a counter in a more human-readable way. To do that, we can call the [display](#) function on the counter. This function retrieves the current counter value and formats it either with a provided or with an automatically inferred [numbering](#).

```
#set heading(numbering: "1.")

= Introduction
Some text here.

= Background
The current value is: #context {
 counter(heading).display()
}

Or in roman numerals: #context {
 counter(heading).display("I")
}
```

# 1. Introduction

Some text here.

## 2. Background

The current value is: 2.

Or in roman numerals: II

## Modifying a counter

To modify a counter, you can use the `step` and `update` methods:

- The `step` method increases the value of the counter by one. Because counters can have multiple levels , it optionally takes a `level` argument. If given, the counter steps at the given depth.
- The `update` method allows you to arbitrarily modify the counter. In its basic form, you give it an integer (or an array for multiple levels). For more flexibility, you can instead also give it a function that receives the current value and returns a new value.

The heading counter is stepped before the heading is displayed, so `Analysis` gets the number seven even though the counter is at six after the second update.

```
#set heading(numbering: "1.")

= Introduction
#counter(heading).step()

= Background
#counter(heading).update(3)
#counter(heading).update(n => n * 2)

= Analysis
Let's skip 7.1.
#counter(heading).step(level: 2)

== Analysis
Still at #context {
 counter(heading).display()
}
```

## 1. Introduction

## 3. Background

## 7. Analysis

Let's skip 7.1.

### 7.2. Analysis

Still at 7.2.

## Page counter

The page counter is special. It is automatically stepped at each pagebreak. But like other counters, you can also step it manually. For example, you could have Roman page numbers for your preface, then switch to Arabic page numbers for your main content and reset the page counter to one.

```
#set page(numbering: "(i)")
```

#### = Preface

The preface is numbered with roman numerals.

```
#set page(numbering: "1 / 1")
#counter(page).update(1)
```

#### = Main text

Here, the counter is reset to one. We also display both the current page and total number of pages in Arabic numbers.

## Preface

The preface is numbered with roman numerals.

(i)

## Main text

Here, the counter is reset to one. We also display both the current page and total number of pages in Arabic numbers.

1 / 1

## Custom counters

To define your own counter, call the `counter` function with a string as a key.

This key identifies the counter globally.

```
#let mine = counter("mycounter")
#context mine.display() \
#mine.step()
#context mine.display() \
#mine.update(c => c * 3)
#context mine.display()
```

0  
1  
3

## How to step

When you define and use a custom counter, in general, you should first step the counter and then display it. This way, the stepping behaviour of a counter can depend on the element it is stepped for. If you were writing a counter for, let's

say, theorems, your theorem's definition would thus first include the counter step and only then display the counter and the theorem's contents.

```
#let c = counter("theorem")
#let theorem(it) = block[
 #c.step()
 Theorem #context c.display() :
 #it
]

#theorem[$1 = 1$]
#theorem[$2 < 3$]
```

**Theorem 1:**  $1 = 1$

**Theorem 2:**  $2 < 3$

The rationale behind this is best explained on the example of the heading counter: An update to the heading counter depends on the heading's level. By stepping directly before the heading, we can correctly step from 1 to 1.1 when encountering a level 2 heading. If we were to step after the heading, we wouldn't know what to step to.

Because counters should always be stepped before the elements they count, they always start at zero. This way, they are at one for the first display (which happens after the first step).

## Time travel

Counters can travel through time! You can find out the final value of the counter before it is reached and even determine what the value was at any particular location in the document.

```
#let mine = counter("mycounter")

= Values
#context [
 Value here: #mine.get() \
 At intro: #mine.at(<intro>) \
 Final value: #mine.final()
]

#mine.update(n => n + 3)

= Introduction <intro>
#lorem(10)

#mine.step()
#mine.step()
```

## Values

Value here: (0,)  
At intro: (3,)  
Final value: (5,)

## Introduction

Lorem ipsum dolor sit amet, consectetur  
adipiscing elit, sed do.

## Other kinds of state

The `counter` type is closely related to [state](#) type. Read its documentation for more details on state management in Typst and why it doesn't just use normal variables for counters.

## Constructor

Create a new counter identified by a key.

```
counter(
 str label selector location function
) -> counter
```

**key**

`str or label or selector or location or function`

**RequiredPositional**

The key that identifies this counter.

- If it is a string, creates a custom counter that is only affected by manual updates,
- If it is the [page](#) function, counts through pages,
- If it is a [selector](#), counts through elements that matches with the selector. For example,
  - provide an element function: counts elements of that type,
  - provide a [`<label>`](#): counts elements with that label.

## Definitions

**getContextual**

Retrieves the value of the counter at the current location. Always returns an array of integers, even if the counter has just one number.  
This is equivalent to `counter.at(here())`.

```
self.get() -> intarray
displayContextual
```

Displays the current value of the counter with a numbering and returns the formatted output.

*Compatibility:* For compatibility with Typst 0.10 and lower, this function also works without an established context. Then, it will create opaque contextual content rather than directly returning the output of the numbering. This behaviour will be removed in a future release.

```
self.display()
```

```
autostrfunction,
both: bool,
) -> any
numbering
auto or str or function
```

**Positional**

A [numbering pattern or a function](#), which specifies how to display the counter. If given a function, that function receives each number of the counter as a separate argument. If the amount of numbers varies, e.g. for the heading argument, you can use an [argument sink](#).

If this is omitted or set to `auto`, displays the counter with the numbering style for the counted element or with the pattern `"1.1"` if no such style exists.

Default: `auto`

`both`  
`bool`

If enabled, displays the current and final top-level count together. Both can be styled through a single numbering pattern. This is used by the page numbering property to display the current and total number of pages when a pattern like `"1 / 1"` is given.

Default: `false`

**atContextual**

Retrieves the value of the counter at the given location. Always returns an array of integers, even if the counter has just one number.

The `selector` must match exactly one element in the document. The most useful kinds of selectors for this are [labels](#) and [locations](#).

*Compatibility:* For compatibility with Typst 0.10 and lower, this function also works without a known context if the `selector` is a location. This behaviour will be removed in a future release.

```
self.at(
 labelselectorlocationfunction
) -> intarray
selector

label or selector or location or function
```

#### **RequiredPositional**

The place at which the counter's value should be retrieved.

#### **finalContextual**

Retrieves the value of the counter at the end of the document. Always returns an array of integers, even if the counter has just one number.

```
self.final(
 nonelocation
) -> intarray
location

none or location
```

#### **Positional**

*Compatibility:* This argument is deprecated. It only exists for compatibility with Typst 0.10 and lower and shouldn't be used anymore.

Default: `none`

#### **step**

Increases the value of the counter by one.

The update will be in effect at the position where the returned content is inserted into the document. If you don't put the output into the document, nothing happens! This would be the case, for example, if you write `let _ =`

`counter(page).step()`. Counter updates are always applied in layout order

and in that case, Typst wouldn't know when to step the counter.

```
self.step(
 level: int
) -> content
level
int
```

The depth at which to step the counter. Defaults to `1`.

Default: `1`

**update**

Updates the value of the counter.

Just like with `step`, the update only occurs if you put the resulting content into the document.

```
self.update(
 intarrayfunction
) -> content
update
```

`int or array or function`

*RequiredPositional*

If given an integer or array of integers, sets the counter to that value. If given a function, that function receives the previous counter value (with each number as a separate argument) and has to return the new value (integer or array).

## 3.12.2 Here

Provides the current location in the document.

You can think of `here` as a low-level building block that directly extracts the current location from the active [context](#). Some other functions use it internally:

For instance, `counter.get()` is equivalent to `counter.at(here())`.

Within show rules on [locatable](#) elements, `here()` will match the location of the shown element.

If you want to display the current page number, refer to the documentation of the [counter](#) type. While `here` can be used to determine the physical page number, typically you want the logical page number that may, for instance, have been reset after a preface.

## Examples

Determining the current position in the document in combination with the [position](#) method:

```
#context [
 I am located at
 #here().position()
]
```

I am located at (page: 1, x: 15pt, y: 15pt)

Running a [query](#) for elements before the current position:

```
= Introduction
= Background
```

There are

```
#context query(
 selector(heading).before(here())
).len()
headings before me.
```

= Conclusion

## Introduction

### Background

There are 2 headings before me.

### Conclusion

Refer to the [selector](#) type for more details on before/after selectors.

## Parameters

`here()` -> [location](#)

### 3.12.3 Locate

Determines the location of an element in the document.

Takes a selector that must match exactly one element and returns that

element's [location](#). This location can, in particular, be used to retrieve the physical [page](#) number and [position](#) (page, x, y) for that element.

## Examples

Locating a specific element:

```
#context [
 Introduction is at: \
 #locate(<intro>).position()
]
```

= Introduction <intro>

Introduction is at:  
 (page: 1, x: 15pt, y: 55.87pt)

## Introduction

### Compatibility

In Typst 0.10 and lower, the `locate` function took a closure that made the current location in the document available (like [here](#) does now). This usage pattern is deprecated. Compatibility with the old way will remain for a while to give package authors time to upgrade. To that effect, `locate` detects whether it received a selector or a user-defined function and adjusts its semantics accordingly. This behaviour will be removed in the future.

### Parameters

```
locate(
 labelselectorlocationfunction
) -> contentlocation

selector
```

`label` or `selector` or `location` or `function`

*RequiredPositional*

A selector that should match exactly one element. This element will be located.

Especially useful in combination with

- [here](#) to locate the current context,
- a [location](#) retrieved from some queried element via the [location\(\)](#) method on content.

## 3.12.4 Location

Identifies an element in the document.

A location uniquely identifies an element in the document and lets you access its absolute position on the pages. You can retrieve the current location with the [here](#) function and the location of a queried or shown element with the [location\(\)](#) method on content.

## Locatable elements

Currently, only a subset of element functions is locatable. Aside from headings and figures, this includes equations, references, quotes and all elements with an explicit label. As a result, you *can* query for e.g. [strong](#) elements, but you will find only those that have an explicit label attached to them. This limitation will be resolved in the future.

## Definitions

### `page`

Returns the page number for this location.

Note that this does not return the value of the [page counter](#) at this location, but the true page number (starting from one).

If you want to know the value of the page counter, use

`counter(page).at(loc)` instead.

Can be used with [here](#) to retrieve the physical page position of the current context:

```
self.page() -> int
#context [
 I am located on
```

```
page #here().page()
]
```

I am located on page 1

#### **position**

Returns a dictionary with the page number and the x, y position for this location. The page number starts at one and the coordinates are measured from the top-left of the page.

If you only need the page number, use `page()` instead as it allows Typst to skip unnecessary work.

```
self.position() -> dictionary
page-numbering
```

Returns the page numbering pattern of the page at this location. This can be used when displaying the page counter in order to obtain the local numbering. This is useful if you are building custom indices or outlines.

If the page numbering is set to `none` at that location, this function returns `none`.

```
self.page-numbering() -> nonestrfunction
```

## 3.12.5 Metadata

Exposes a value to the query system without producing visible content.

This element can be retrieved with the [query](#) function and from the command line with [typst query](#). Its purpose is to expose an arbitrary value to the introspection system. To identify a metadata value among others, you can attach a [label](#) to it and query for that label.

The `metadata` element is especially useful for command line queries because it allows you to expose arbitrary values to the outside world.

```
// Put metadata somewhere.
#metadata("This is a note") <note>

// And find it from anywhere else.
#context {
 query(<note>).first().value
}
```

This is a note

## Parameters

```
metadata(
any
) -> content
value
any
RequiredPositional
```

The value to embed into the document.

## 3.12.6 Query

Finds elements in the document.

The `query` functions lets you search your document for elements of a particular type or with a particular label. To use it, you first need to ensure that [context](#) is available.

## Finding elements

In the example below, we manually create a table of contents instead of using the [outline](#) function.

To do this, we first query for all headings in the document at level 1 and where `outlined` is true. Querying only for headings at level 1 ensures that, for the purpose of this example, sub-headings are not included in the table of contents. The `outlined` field is used to exclude the "Table of Contents" heading itself.

Note that we open a `context` to be able to use the `query` function.

```
#set page(numbering: "1")

#heading(outlined: false) [
 Table of Contents
]
#context {
 let chapters = query(
 heading.where(
 level: 1,
 outlined: true,
)
)
 for chapter in chapters {
 let loc = chapter.location()
 let nr = numbering(
 loc.page-numbering(),
 ..counter(page).at(loc),
)
 [#chapter.body #h(1fr) #nr \]
 }
}

= Introduction
#lorem(10)
#pagebreak()

== Sub-Heading
#lorem(8)

= Discussion
#lorem(18)
```

## Table of Contents

|              |   |
|--------------|---|
| Introduction | 1 |
| Discussion   | 2 |

## Introduction

Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do.

1

## Sub-Heading

Lore ipsum dolor sit amet, consectetur adipiscing elit.

## Discussion

Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna.

2

To get the page numbers, we first get the location of the elements returned by query with [location](#). We then also retrieve the [page numbering](#) and [page counter](#) at that location and apply the numbering to the counter.

## A word of caution

To resolve all your queries, Typst evaluates and layouts parts of the document multiple times. However, there is no guarantee that your queries can actually be

completely resolved. If you aren't careful a query can affect itself—leading to a result that never stabilizes.

In the example below, we query for all headings in the document. We then generate as many headings. In the beginning, there's just one heading, titled `Real`. Thus, `count` is 1 and one `Fake` heading is generated. Typst sees that the query's result has changed and processes it again. This time, `count` is 2 and two `Fake` headings are generated. This goes on and on. As we can see, the output has a finite amount of headings. This is because Typst simply gives up after a few attempts.

In general, you should try not to write queries that affect themselves. The same words of caution also apply to other introspection features like [counters](#) and [state](#).

```
= Real
#context {
 let elems = query(heading)
 let count = elems.len()
 count * [= Fake]
}
```

**Real**

**Fake**

**Fake**

**Fake**

**Fake**

## Command line queries

You can also perform queries from the command line with the `typst query` command. This command executes an arbitrary query on the document and returns the resulting elements in serialized form. Consider the following example.typ file which contains some invisible [metadata](#):

```
#metadata("This is a note") <note>
```

You can execute a query on it as follows using Typst's CLI:

```
$ typst query example.typ "<note>"
[
 {
 "func": "metadata",
 "value": "This is a note",
 "label": "<note>"
 }
]
```

Frequently, you're interested in only one specific field of the resulting elements. In the case of the `metadata` element, the `value` field is the interesting one. You can extract just this field with the `--field` argument.

```
$ typst query example.typ "<note>" --field value
["This is a note"]
```

If you are interested in just a single element, you can use the `--one` flag to extract just it.

```
$ typst query example.typ "<note>" --field value --one
"This is a note"
```

## Parameters

```
query(
 labelselectorlocationfunction,
 nonelocation,
```

```
) -> array
target
label or selector or location or function
```

**RequiredPositional**

Can be

- an element function like a `heading` or `figure`,
- a `<label>`,
- a more complex selector like `heading.where(level: 1)`,
- or `selector(heading).before(here())`.

Only [locatable](#) element functions are supported.

**location**

```
none or location
```

**Positional**

*Compatibility:* This argument is deprecated. It only exists for compatibility with Typst 0.10 and lower and shouldn't be used anymore.

Default: `none`

## 3.12.7 State

Manages stateful parts of your document.

Let's say you have some computations in your document and want to remember the result of your last computation to use it in the next one. You might try something similar to the code below and expect it to output 10, 13, 26, and 21. However this **does not work** in Typst. If you test this code, you will see that Typst complains with the following error message: *Variables from outside the function are read-only and cannot be modified.*

```
// This doesn't work!
#let x = 0
#let compute(expr) = {
 x = eval(
 expr.replace("x", str(x))
)
 [New value is #x.]
}

#compute("10") \
#compute("x + 3") \
#compute("x * 2") \
#compute("x - 5")
```

## State and document markup

Why does it do that? Because, in general, this kind of computation with side effects is problematic in document markup and Typst is upfront about that. For the results to make sense, the computation must proceed in the same order in which the results will be laid out in the document. In our simple example, that's the case, but in general it might not be.

Let's look at a slightly different, but similar kind of state: The heading numbering. We want to increase the heading counter at each heading. Easy enough, right? Just add one. Well, it's not that simple. Consider the following example:

```
#set heading(numbering: "1.")
#let template(body) = [
 = Outline
 ...
 #body
]

#show: template
```

```
= Introduction
```

...

## 1. Outline

...

## 2. Introduction

...

Here, Typst first processes the body of the document after the `show` rule, sees the `Introduction` heading, then passes the resulting content to the `template` function and only then sees the `Outline`. Just counting up would number the `Introduction` with 1 and the `Outline` with 2.

## Managing state in Typst

So what do we do instead? We use Typst's state management system. Calling the `state` function with an identifying string key and an optional initial value gives you a state value which exposes a few functions. The two most important ones are `get` and `update`:

- The `get` function retrieves the current value of the state. Because the value can vary over the course of the document, it is a *contextual* function that can only be used when `context` is available.
- The `update` function modifies the state. You can give it any value. If given a non-function value, it sets the state to that value. If given a function, that function receives the previous state and has to return the new state.

Our initial example would now look like this:

```
#let s = state("x", 0)
#let compute(expr) = [
 #s.update(x =>
 eval(expr.replace("x", str(x)))
)
 New value is #context s.get().
]
```

```
#compute("10") \
#compute("x + 3") \
#compute("x * 2") \
#compute("x - 5")
```

New value is 10.  
 New value is 13.  
 New value is 26.  
 New value is 21.

State managed by Typst is always updated in layout order, not in evaluation order. The `update` method returns content and its effect occurs at the position where the returned content is inserted into the document.

As a result, we can now also store some of the computations in variables, but they still show the correct results:

```
...
#let more = [
 #compute("x * 2") \
 #compute("x - 5")
]

#compute("10") \
#compute("x + 3") \
#more
```

```
New value is 10.
New value is 13.
New value is 26.
New value is 21.
```

This example is of course a bit silly, but in practice this is often exactly what you want! A good example are heading counters, which is why Typst's [counting system](#) is very similar to its state system.

## Time Travel

By using Typst's state management system you also get time travel capabilities! We can find out what the value of the state will be at any position in the document from anywhere else. In particular, the `at` method gives us the value of the state at any particular location and the `final` methods gives us the value of the state at the end of the document.

...

```
Value at `<here>` is
#context s.at(<here>)

#compute("10") \
#compute("x + 3") \
Here. <here> \
#compute("x * 2") \
#compute("x - 5")
```

```
Value at <here> is 13
```

```
New value is 10.
```

```
New value is 13.
```

```
Here.
```

```
New value is 26.
```

```
New value is 21.
```

## A word of caution

To resolve the values of all states, Typst evaluates parts of your code multiple times. However, there is no guarantee that your state manipulation can actually be completely resolved.

For instance, if you generate state updates depending on the final value of a state, the results might never converge. The example below illustrates this. We initialize our state with `1` and then update it to its own final value plus 1. So it should be `2`, but then its final value is `2`, so it should be `3`, and so on. This example displays a finite value because Typst simply gives up after a few attempts.

```
// This is bad!
#let s = state("x", 1)
#context s.update(s.final() + 1)
#context s.get()
```

5

In general, you should try not to generate state updates from within context expressions. If possible, try to express your updates as non-contextual values or functions that compute the new value from the previous value. Sometimes, it

cannot be helped, but in those cases it is up to you to ensure that the result converges.

## Constructor

Create a new state identified by a key.

```
state (
 str,
 any,
) -> state
key
str
RequiredPositional
```

The key that identifies this state.

```
init
any
Positional
```

The initial value of the state.

Default: `none`

## Definitions

`getContextual`

Retrieves the value of the state at the current location.

This is equivalent to `state.at(here())`.

```
self.get() -> any
atContextual
```

Retrieves the value of the state at the given selector's unique match.

The `selector` must match exactly one element in the document. The most

useful kinds of selectors for this are [labels](#) and [locations](#).

*Compatibility:* For compatibility with Typst 0.10 and lower, this function also works without a known context if the `selector` is a location. This behaviour will be removed in a future release.

```
self.at(
 labelselectorlocationfunction
) -> any
selector

label or selector or location or function
```

**RequiredPositional**

The place at which the state's value should be retrieved.

**finalContextual**

Retrieves the value of the state at the end of the document.

```
self.final(
 nonelocation
) -> any
location

none or location
```

**Positional**

*Compatibility:* This argument is deprecated. It only exists for compatibility with Typst 0.10 and lower and shouldn't be used anymore.

Default: none

**update**

Update the value of the state.

The update will be in effect at the position where the returned content is inserted into the document. If you don't put the output into the document, nothing happens! This would be the case, for example, if you write `let _ = state("key").update(7)`. State updates are always applied in layout order and in that case, Typst wouldn't know when to update the state.

```
self.update(
 anyfunction
) -> content
update

any or function
```

***RequiredPositional***

If given a non function-value, sets the state to that value. If given a function, that function receives the previous state and has to return the new state.

**display**

Displays the current value of the state.

**Deprecation planned:** Use [get](#) instead.

```
self.display(
 nonefunction
) -> content
func
none or function
```

***Positional***

A function which receives the value of the state and can return arbitrary content which is then displayed. If this is omitted, the value is directly displayed.

Default: `none`

## 3.13 Data Loading

Data loading from external files.

These functions help you with loading and embedding data, for example from the results of an experiment.

### Definitions

- [cbor](#) Reads structured data from a CBOR file.
- [csv](#) Reads structured data from a CSV file.
- [json](#) Reads structured data from a JSON file.
- [read](#) Reads plain text or data from a file.

- [toml](#) Reads structured data from a TOML file.

- [xml](#) Reads structured data from an XML file.

- [yaml](#) Reads structured data from a YAML file.

### 3.13.1 CBOR

Reads structured data from a CBOR file.

The file must contain a valid CBOR serialization. Mappings will be converted

into Typst dictionaries, and sequences will be converted into Typst arrays.

Strings and booleans will be converted into the Typst equivalents, null-values

(`null`, `~` or empty ``) will be converted into `none`, and numbers will be

converted to floats or integers depending on whether they are whole numbers.

Be aware that integers larger than  $2^{63}-1$  will be converted to floating point

numbers, which may result in an approximative value.

### Parameters

```
cbor (
 str
) -> any
path
str
RequiredPositional
```

Path to a CBOR file.

For more details, see the [Paths section](#).

### Definitions

**decode**

Reads structured data from CBOR bytes.

```
cbor.decode (
 bytes
) -> any
```

**data**  
bytes  
*RequiredPositional*

cbor data.

**encode**

Encode structured data into CBOR bytes.

`cbor.encode (`

`any`

`) -> bytes`

**value**

`any`

*RequiredPositional*

Value to be encoded.

## 3.13.2 CSV

Reads structured data from a CSV file.

The CSV file will be read and parsed into a 2-dimensional array of strings: Each row in the CSV file will be represented as an array of strings, and all rows will be collected into a single array. Header rows will not be stripped.

### Example

```
#let results = csv("example.csv")

#table(
 columns: 2,
 [*Condition*], [*Result*],
 ..results.flatten(),
)
```

| Condition | Result |
|-----------|--------|
| 0..2      | small  |
| 3..5      | medium |
| 6..       | big    |

## Parameters

```
csv (
 str,
 delimiter: str,
 row-type: type,
) -> array
path
str
RequiredPositional
```

Path to a CSV file.

For more details, see the [Paths section](#).

**delimiter**  
str

The delimiter that separates columns in the CSV file. Must be a single ASCII character.

Default: `", "`

**row-type**  
type

How to represent the file's rows.

- If set to `array`, each row is represented as a plain array of strings.
- If set to `dictionary`, each row is represented as a dictionary mapping from header keys to strings. This option only makes sense when a header row is present in the CSV file.

Default: `array`

## Definitions

**decode**

Reads structured data from a CSV string/bytes.

```
csv.decode (
 strbytes,
 delimiter: str,
```

`row-type: type,`

`) -> array`

**data**

`str or bytes`

**RequiredPositional**

CSV data.

**delimiter**

`str`

The delimiter that separates columns in the CSV file. Must be a single ASCII character.

Default: `" , "`

**row-type**

`type`

How to represent the file's rows.

- If set to `array`, each row is represented as a plain array of strings.
- If set to `dictionary`, each row is represented as a dictionary mapping from header keys to strings. This option only makes sense when a header row is present in the CSV file.

Default: `array`

### 3.13.3 JSON

Reads structured data from a JSON file.

The file must contain a valid JSON value, such as object or array. JSON objects will be converted into Typst dictionaries, and JSON arrays will be converted into Typst arrays. Strings and booleans will be converted into the Typst equivalents, `null` will be converted into `none`, and numbers will be converted to floats or integers depending on whether they are whole numbers.

Be aware that integers larger than  $2^{63}-1$  will be converted to floating point numbers, which may result in an approximative value.

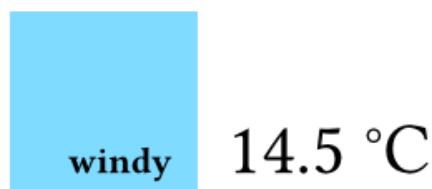
The function returns a dictionary, an array or, depending on the JSON file, another JSON data type.

The JSON files in the example contain objects with the keys `temperature`, `unit`, and `weather`.

## Example

```
#let forecast(day) = block[
 #box(square(
 width: 2cm,
 inset: 8pt,
 fill: if day.weather == "sunny" {
 yellow
 } else {
 aqua
 },
 align(
 bottom + right,
 strong(day.weather),
),
))
 #h(6pt)
 #set text(22pt, baseline: -8pt)
 #day.temperature °#day.unit
]

#forecast(json("monday.json"))
#forecast(json("tuesday.json"))
```



## Parameters

```
json(
 str
) -> any
path
 str
RequiredPositional
```

Path to a JSON file.

For more details, see the [Paths section](#).

## Definitions

**decode**

Reads structured data from a JSON string/bytes.

```
json.decode(
 strbytes
) -> any
data
 str or bytes
RequiredPositional
```

JSON data.

**encode**

Encodes structured data into a JSON string.

```
json.encode(
 any,
)
```

```
pretty: bool,
) -> str
value
any
RequiredPositional
```

Value to be encoded.

**pretty**  
bool

Whether to pretty print the JSON with newlines and indentation.

Default: `true`

## 3.13.4 Read

Reads plain text or data from a file.

By default, the file will be read as UTF-8 and returned as a [string](#).

If you specify `encoding: none`, this returns raw [bytes](#) instead.

## Example

```
An example for a HTML file: \
#let text = read("example.html")
#raw(text, lang: "html")
```

```
Raw bytes:
#read("tiger.jpg", encoding: none)
```

An example for a HTML file:

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8" />
 <title>Example document</title>
 </head>
 <body>
 <h1>Hello, world!</h1>
 </body>
</html>
```

Raw bytes: [bytes\(116679\)](#)

## Parameters

```
read(
 str,
 encoding: nonestr,
) -> strbytes
path
str
RequiredPositional
```

Path to a file.

For more details, see the [Paths section](#).

**encoding**

[none](#) or [str](#)

The encoding to read the file with.

If set to [none](#), this function returns raw bytes.

Variant	Details
---------	---------

["utf8"](#) The Unicode UTF-8 encoding.

Default: ["utf8"](#)

## 3.13.5 TOML

Reads structured data from a TOML file.

The file must contain a valid TOML table. TOML tables will be converted into Typst dictionaries, and TOML arrays will be converted into Typst arrays. Strings, booleans and datetimes will be converted into the Typst equivalents and numbers will be converted to floats or integers depending on whether they are whole numbers.

The TOML file in the example consists of a table with the keys `title`, `version`, and `authors`.

### Example

```
#let details = toml("details.toml")

Title: #details.title \
Version: #details.version \
Authors: #(details.authors
 .join(", ", last: " and "))
```

```
Title: Secret project
Version: 2
Authors: Mr Robert and Miss Enola
```

### Parameters

```
toml(
 str
) -> any
path
str
RequiredPositional
```

Path to a TOML file.

For more details, see the [Paths section](#).

## Definitions

### **decode**

Reads structured data from a TOML string/bytes.

```
toml.decode (
 strbytes
) -> any
```

**data**

```
str or bytes
```

*RequiredPositional*

TOML data.

### **encode**

Encodes structured data into a TOML string.

```
toml.encode (
 any,
 pretty: bool,
) -> str
```

**value**

```
any
```

*RequiredPositional*

Value to be encoded.

### **pretty**

```
bool
```

Whether to pretty-print the resulting TOML.

Default: `true`

## 3.13.6 XML

Reads structured data from an XML file.

The XML file is parsed into an array of dictionaries and strings. XML nodes can be elements or strings. Elements are represented as dictionaries with the following keys:

- `tag`: The name of the element as a string.

- `attrs`: A dictionary of the element's attributes as strings.

- `children`: An array of the element's child nodes.

The XML file in the example contains a root `news` tag with multiple `article` tags. Each article has a `title`, `author`, and `content` tag. The `content` tag contains one or more paragraphs, which are represented as `p` tags.

## Example

```
#let find-child(elem, tag) = {
 elem.children
 .find(e => "tag" in e and e.tag == tag)
}

#let article(elem) = {
 let title = find-child(elem, "title")
 let author = find-child(elem, "author")
 let pars = find-child(elem, "content")

 heading(title.children.first())
 text(10pt, weight: "medium") [
 Published by
 #author.children.first()
]

 for p in pars.children {
 if (type(p) == "dictionary") {
 parbreak()
 p.children.first()
 }
 }
}

#let data = xml("example.xml")
#for elem in data.first().children {
 if (type(elem) == "dictionary") {
 article(elem)
 }
}
```

## 2022 Budget approved

Published by John Doe

The 2022 budget has been approved by the Senate.

The budget is \$1.2 trillion.

It is expected to be signed by the President next week.

## Tigers win the World Series

Published by Jane Doe

The Tigers have won the World Series.

They beat the Giants 4 to 3.

## Parameters

```
xml (
 str
) -> any
path
str
RequiredPositional
```

Path to an XML file.

For more details, see the [Paths section](#).

## Definitions

### decode

Reads structured data from an XML string/bytes.

```
xml.decode (
 strbytes
) -> any
data
str or bytes
RequiredPositional
```

XML data.

## 3.13.7 YAML

Reads structured data from a YAML file.

The file must contain a valid YAML object or array. YAML mappings will be converted into Typst dictionaries, and YAML sequences will be converted into Typst arrays. Strings and booleans will be converted into the Typst equivalents, null-values (`null`, `~` or empty ``) will be converted into `none`, and numbers will be converted to floats or integers depending on whether they are whole numbers. Custom YAML tags are ignored, though the loaded value will still be present.

Be aware that integers larger than  $2^{63}-1$  will be converted to floating point numbers, which may give an approximative value.

The YAML files in the example contain objects with authors as keys, each with a sequence of their own submapping with the keys "title" and "published"

### Example

```
#let bookshelf(contents) = {
 for (author, works) in contents {
 author
 for work in works [
 - #work.title (#work.published)
]
 }
}

#bookshelf(
 yaml("scifi-authors.yaml")
)
```

Arthur C. Clarke

- Against the Fall of Night (1978)
- The songs of distant earth (1986)

Isaac Asimov

- Quasar, Quasar, Burning Bright (1977)
- Far as Human Eye Could See (1987)

## Parameters

```
yaml (
 str
) -> any
path
str
RequiredPositional
```

Path to a YAML file.

For more details, see the [Paths section](#).

## Definitions

**decode**

Reads structured data from a YAML string/bytes.

```
yaml.decode (
 strbytes
) -> any
data
str or bytes
RequiredPositional
```

YAML data.

**encode**

Encode structured data into a YAML string.

```
yaml.encode (
 any
) -> str
value
any
```

***RequiredPositional***

Value to be encoded.

## 4. Guides

Welcome to the Guides section! Here, you'll find helpful material for specific user groups or use cases. Currently, two guides are available: An introduction to Typst for LaTeX users, and a detailed look at page setup. Feel free to propose other topics for guides!

### List of Guides

- [Guide for LaTeX users](#)
- [Page setup guide](#)
- [Table guide](#)

## 4.1 Guide for LaTeX users

This page is a good starting point if you have used LaTeX before and want to try out Typst. We will explore the main differences between these two systems from a user perspective. Although Typst is not built upon LaTeX and has a different syntax, you will learn how to use your LaTeX skills to get a head start. Just like LaTeX, Typst is a markup-based typesetting system: You compose your document in a text file and mark it up with commands and other syntax. Then, you use a compiler to typeset the source file into a PDF. However, Typst also differs from LaTeX in several aspects: For one, Typst uses more dedicated syntax (like you may know from Markdown) for common tasks. Typst's commands are

also more principled: They all work the same, so unlike in LaTeX, you just need to understand a few general concepts instead of learning different conventions for each package. Moreover Typst compiles faster than LaTeX: Compilation usually takes milliseconds, not seconds, so the web app and the compiler can both provide instant previews.

In the following, we will cover some of the most common questions a user switching from LaTeX will have when composing a document in Typst. If you prefer a step-by-step introduction to Typst, check out our tutorial.

## Installation

You have two ways to use Typst: In [our web app](#) or by [installing the compiler](#) on your computer. When you use the web app, we provide a batteries-included collaborative editor and run Typst in your browser, no installation required. If you choose to use Typst on your computer instead, you can download the compiler as a single, small binary which any user can run, no root privileges required. Unlike LaTeX, packages are downloaded when you first use them and then cached locally, keeping your Typst installation lean. You can use your own editor and decide where to store your files with the local compiler.

## How do I create a new, empty document?

That's easy. You just create a new, empty text file (the file extension is `.typ`). No boilerplate is needed to get started. Simply start by writing your text. It will be set on an empty A4-sized page. If you are using the web app, click "+ Empty

document" to create a new project with a file and enter the editor. [Paragraph](#)

[breaks](#) work just as they do in LaTeX, just use a blank line.

```
Hey there!
```

Here are two paragraphs. The output is shown to the right.

Hey there!

Here are two paragraphs. The output is shown to the right.

If you want to start from an preexisting LaTeX document instead, you can use

[Pandoc](#) to convert your source code to Typst markup. This conversion is also

built into our web app, so you can upload your `.tex` file to start your project in

Typst.

## How do I create section headings, emphasis, ...?

LaTeX uses the command `\section` to create a section heading. Nested

headings are indicated with `\subsection`, `\subsubsection`, etc. Depending on

your document class, there is also `\part` or `\chapter`.

In Typst, [headings](#) are less verbose: You prefix the line with the heading on it

with an equals sign and a space to get a first-order heading: `= Introduction`.

If you need a second-order heading, you use two equals signs: `== In this`

[paper](#). You can nest headings as deeply as you'd like by adding more equals signs.

Emphasis (usually rendered as italic text) is expressed by enclosing text in \_underscores\_ and strong emphasis (usually rendered in boldface) by using **\*stars\*** instead.

Here is a list of common markup commands used in LaTeX and their Typst equivalents. You can also check out the [full syntax cheat sheet](#).

Element	LaTeX	Typst	See
Strong emphasis	\textbf{strong}	*strong*	<a href="#">strong</a>
Emphasis	\emph{emphasis}	_emphasis_	<a href="#">emph</a>
Monospace / code	\texttt{print(1)}	`print(1)`	<a href="#">raw</a>
Link	\url{https://typst.app}	<a href="https://typst.app/">https://typst.app/</a> <a href="#">link</a>	
Label	\label{intro}	<intro>	<a href="#">label</a>
Reference	\ref{intro}	@intro	<a href="#">ref</a>
Citation	\cite{humphrey97}	@humphrey97	<a href="#">cite</a>
Bullet list	itemize environment	- List	<a href="#">list</a>
Numbered list	enumerate environment	+ List	<a href="#">enum</a>
Term list	description environment	/ <b>Term:</b> List	<a href="#">terms</a>
Figure	figure environment	figure function	<a href="#">figure</a>
Table	table environment	table function	<a href="#">table</a>

`$x$, align /`  
**Equation** `$x$, $ x = y $` [equation](#)  
 equation environments

[Lists](#) do not rely on environments in Typst. Instead, they have lightweight syntax like headings. To create an unordered list (`itemize`), prefix each line of an item with a hyphen:

To write this list in Typst...

```
```latex
\begin{itemize}
\item Fast
\item Flexible
\item Intuitive
\end{itemize}
```

```

...just type this:

- Fast
- Flexible
- Intuitive

To write this list in Typst...

```
\begin{itemize}
\item Fast
\item Flexible
\item Intuitive
\end{itemize}
```

...just type this:

- Fast
- Flexible
- Intuitive

Nesting lists works just by using proper indentation. Adding a blank line in between items results in a more [widely](#) spaced list.

To get a [numbered list](#) (`enumerate`) instead, use a `+` instead of the hyphen. For a [term list](#) (`description`), write `/ Term: Description` instead.

## How do I use a command?

LaTeX heavily relies on commands (prefixed by backslashes). It uses these *macros* to affect the typesetting process and to insert and manipulate content.

Some commands accept arguments, which are most frequently enclosed in curly braces: `\cite{rasmus}`.

Typst differentiates between [markup mode and code mode](#). The default is markup mode, where you compose text and apply syntactic constructs such as **\*stars for bold text\***. Code mode, on the other hand, parallels programming languages like Python, providing the option to input and execute segments of code.

Within Typst's markup, you can switch to code mode for a single command (or rather, *expression*) using a hash (#). This is how you call functions to, for example, split your project into different [files](#) or render text based on some [condition](#). Within code mode, it is possible to include normal markup [content](#) by using square brackets. Within code mode, this content is treated just as any other normal value for a variable.

```
First, a rectangle:
#rect()
```

```
Let me show how to do
#underline([_underlined_ text])
```

```
We can also do some maths:
#calc.max(3, 2 * 4)
```

And finally a little loop:

```
#for x in range(3) [
 Hi #x.
]
```

First, a rectangle:



Let me show how to do *underlined* text

We can also do some maths: 8

And finally a little loop: Hi 0. Hi 1. Hi 2.

A function call always involves the name of the function ([rect](#), [underline](#), [calc.max](#), [range](#)) followed by parentheses (as opposed to LaTeX where the square brackets and curly braces are optional if the macro requires no arguments). The expected list of arguments passed within those parentheses depends on the concrete function and is specified in the reference.

## Arguments

A function can have multiple arguments. Some arguments are positional, i.e., you just provide the value: The function `#lower ("SCREAM")` returns its argument in all-lowercase. Many functions use named arguments instead of positional arguments to increase legibility. For example, the dimensions and stroke of a rectangle are defined with named arguments:

```
#rect(
 width: 2cm,
 height: 1cm,
 stroke: red,
```

)



You specify a named argument by first entering its name (above, it's `width`, `height`, and `stroke`), then a colon, followed by the value (`2cm`, `1cm`, `red`).

You can find the available named arguments in the reference page for each function or in the autocomplete panel when typing. Named arguments are similar to how some LaTeX environments are configured, for example, you would type `\begin{enumerate} [label={\alph*}) ]` to start a list with the labels `a)`, `b)`, and so on.

Often, you want to provide some [content](#) to a function. For example, the LaTeX command `\underline{Alternative A}` would translate to

`#underline([Alternative A])` in Typst. The square brackets indicate that a value is [content](#). Within these brackets, you can use normal markup. However, that's a lot of parentheses for a pretty simple construct. This is why you can also move trailing content arguments after the parentheses (and omit the parentheses if they would end up empty).

Typst is an `#underline[alternative]` to LaTeX.

`#rect(fill: aqua) [Get started here!]`

Typst is an [alternative](#) to LaTeX.

[Get started here!](#)

## Data types

You likely already noticed that the arguments have distinctive data types. Typst supports many [data types](#). Below, there is a table with some of the most important ones and how to write them. In order to specify values of any of these types, you have to be in code mode!

| Data type                             | Example               |
|---------------------------------------|-----------------------|
| <a href="#">Content</a>               | [*fast* typesetting]  |
| <a href="#">String</a>                | "Pietro S. Author"    |
| <a href="#">Integer</a>               | 23                    |
| <a href="#">Floating point number</a> | 1.459                 |
| <a href="#">Absolute length</a>       | 12pt, 5in, 0.3cm, ... |
| <a href="#">Relative length</a>       | 65%                   |

The difference between content and string is that content can contain markup, including function calls, while a string really is just a plain sequence of characters.

Typst provides [control flow constructs](#) and [operators](#) such as + for adding things or == for checking equality between two variables.

You can also store values, including functions, in your own [variables](#). This can be useful to perform computations on them, create reusable automations, or

reference a value multiple times. The variable binding is accomplished with the `let` keyword, which works similar to `\newcommand`:

```
// Store the integer `5`.
#let five = 5

// Define a function that
// increments a value.
#let inc(i) = i + 1

// Reference the variables.
I have #five fingers.

If I had one more, I'd have
#inc(five) fingers. Whoa!
```

I have 5 fingers.

If I had one more, I'd have 6 fingers. Whoa!

## Commands to affect the remaining document

In LaTeX, some commands like `\textbf{bold text}` receive an argument in curly braces and only affect that argument. Other commands such as `\bfseries bold text` act as switches (LaTeX calls this a declaration), altering the appearance of all subsequent content within the document or current scope.

In Typst, the same function can be used both to affect the appearance for the remainder of the document, a block (or scope), or just its arguments. For example, `#text(weight: "bold") [bold text]` will only embolden its argument, while `#set text(weight: "bold")` will embolden any text until the end of the current block, or, if there is none, document. The effects of a

function are immediately obvious based on whether it is used in a call or a [set rule](#).

I am starting out with small text.

```
#set text(14pt)
```

This is a bit `#text(18pt)` [larger,  
don't you think?

I am starting out with small text.

**This is a bit larger, don't you  
think?**

Set rules may appear anywhere in the document. They can be thought of as default argument values of their respective function:

```
#set enum(numbering: "I.")
```

Good results can only be obtained by  
 + following best practices  
 + being aware of current results  
   of other researchers  
 + checking the data for biases

Good results can only be obtained by  
 I. following best practices  
 II. being aware of current results of other  
   researchers  
 III. checking the data for biases

The + is syntactic sugar (think of it as an abbreviation) for a call to the [enum](#) function, to which we apply a set rule above. [Most syntax is linked to a function in this way](#). If you need to style an element beyond what its

arguments enable, you can completely redefine its appearance with a [show rule](#) (somewhat comparable to `\renewcommand`).

You can achieve the effects of LaTeX commands like `\textbf`, `\textsf`, `\rmfamily`, `\mdseries`, and `\itshape` with the [font](#), [style](#), and [weight](#) arguments of the `text` function. The `text` function can be used in a set rule (declaration style) or with a content argument. To replace `\textsc`, you can use the [smallcaps](#) function, which renders its content argument as `smallcaps`. Should you want to use it declaration style (like `\scshape`), you can use an [everything show rule](#) that applies the function to the rest of the scope:

```
#show: smallcaps
```

Boisterous Accusations

## BOISTEROUS ACCUSATIONS

# How do I load a document class?

In LaTeX, you start your main `.tex` file with the `\documentclass{article}` command to define how your document is supposed to look. In that command, you may have replaced `article` with another value such as `report` and `amsart` to select a different look.

When using Typst, you style your documents with [functions](#). Typically, you use a template that provides a function that styles your whole document. First, you import the function from a template file. Then, you apply it to your whole

document. This is accomplished with a [show rule](#) that wraps the following document in a given function. The following example illustrates how it works:

```
#import "conf.typ": conf
#show: conf.with(
 title: [
 Towards Improved Modelling
],
 authors: (
 (
 name: "Theresa Tungsten",
 affiliation: "Artos Institute",
 email: "tung@artos.edu",
),
 (
 name: "Eugene Deklan",
 affiliation: "Honduras State",
 email: "e.dekhan@hstate.hn",
),
),
 abstract: lorem(80),
)
```

Let's get started writing this article by putting insightful paragraphs right here!

## Towards Improved Modelling

Theresa Tungsten  
 Artos Institute  
 tung@artos.edu

Eugene Deklan  
 Honduras State  
 e.dekhan@hstate.hn

### Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et.

Let's get started writing this article by putting insightful paragraphs right here! Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedit, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nolle me ab eo delec-

tari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeo, familiari suo, geometrica discere maluisset quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini eruditio in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi quidem videtur, inermis ac nudus est. Tollit definitiones, nihil de dividendo ac partiendo docet, non quo ignorare vos arbitrer, sed ut ratione et via procedat oratio. Quaerimus igitur, quid sit extrellum et ultimum bonorum, quod omnium philosophorum sententia tale debet esse, ut eius magnitudinem celeritas, diuturnitatem allevatio consoletur. Ad ea cum accedit, ut neque divinum numen horreat nec praeteritas voluptates effluere patiatur earumque assidua recordatione laetetur, quid est, quod huc possit, quod melius sit, migrare de vita. His rebus instructus semper est in voluptate esse aut in armatum hostem impetum fecisse aut in poetis evolvendis, ut ego et Triarius te hortatore facimus, consumeret, in quibus hoc primum est in quo admirer, cur in gravissimis rebus non delectet eos sermo patrius, cum idem fabellas Latinas ad verbum e Graecis expressas non invitit legant. Quis enim tam inimicus paene nomini Romano est, qui Ennii Medeam aut Antiopam Pacuvii spernat aut reiciat, quod se isdem Euripidis fabulis delectari dicat, Latinas litteras oderit? Synephebos ego,

The `import` statement makes `functions` (and other definitions) from another file available. In this example, it imports the `conf` function from the `conf.typ` file. This function formats a document as a conference article. We use a `show` rule to apply it to the document and also configure some

metadata of the article. After applying the show rule, we can start writing our article right away!

You can also use templates from Typst Universe (which is Typst's equivalent of CTAN) using an import statement like this:

```
#import "@preview/elsearticle:0.2.1": elsearticle. Check the documentation of an individual template to learn the name of its template function. Templates and packages from Typst Universe are automatically downloaded when you first use them.
```

In the web app, you can choose to create a project from a template on Typst Universe or even create your own using the template wizard. Locally, you can use the `typst init` CLI to create a new project from a template. Check out [the list of templates](#) published on Typst Universe. You can also take a look at the [awesome-typst repository](#) to find community templates that aren't available through Universe.

You can also [create your own, custom templates](#). They are shorter and more readable than the corresponding LaTeX `.sty` files by orders of magnitude, so give it a try!

Functions are Typst's "commands" and can transform their arguments to an output value, including document *content*. Functions are "pure", which means that they cannot have any effects beyond creating an output value / output content. This is in stark contrast to LaTeX macros that can have arbitrary effects on your document.

To let a function style your whole document, the show rule processes everything that comes after it and calls the function specified after the colon with the result as an argument. The `.with` part is a *method* that takes the `conf` function and pre-configures some of its arguments before passing it on to the show rule.

## How do I load packages?

Typst is "batteries included," so the equivalent of many popular LaTeX packages is built right-in. Below, we compiled a table with frequently loaded packages and their corresponding Typst functions.

| LaTeX Package               | Typst Alternative                                                           |
|-----------------------------|-----------------------------------------------------------------------------|
| graphicx, svg               | <a href="#">image</a> function                                              |
| tabularx                    | <a href="#">table</a> , <a href="#">grid</a> functions                      |
| fontenc, inputenc, unicode- | Just start writing!                                                         |
| math                        |                                                                             |
| babel, polyglossia          | <a href="#">text</a> function: <code>#set text(lang: "zh")</code>           |
| amsmath                     | <a href="#">Math mode</a>                                                   |
| amsfonts, amssymb           | <a href="#">sym</a> module and <a href="#">syntax</a>                       |
| geometry, fancyhdr          | <a href="#">page</a> function                                               |
| xcolor                      | <a href="#">text</a> function: <code>#set text(fill: rgb("#0178A4"))</code> |
| hyperref                    | <a href="#">link</a> function                                               |
| bibtex, biblatex, natbib    | <a href="#">cite</a> , <a href="#">bibliography</a> functions               |

|                    |                                                                               |
|--------------------|-------------------------------------------------------------------------------|
| lstlisting, minted | <a href="#">raw</a> function and syntax                                       |
| parskip            | <a href="#">block</a> and <a href="#">par</a> functions                       |
| csquotes           | Set the <a href="#">text</a> language and type " or '                         |
| caption            | <a href="#">figure</a> function                                               |
| enumitem           | <a href="#">list</a> , <a href="#">enum</a> , <a href="#">terms</a> functions |

Although *many* things are built-in, not everything can be. That's why Typst has its own [package ecosystem](#) where the community share its creations and automations. Let's take, for instance, the *cetz* package: This package allows you to create complex drawings and plots. To use *cetz* in your document, you can just write:

```
#import "@preview/cetz:0.2.1"
```

(The `@preview` is a *namespace* that is used while the package manager is still in its early and experimental state. It will be replaced in the future.)

Aside from the official package hub, you might also want to check out the [awesome-typst repository](#), which compiles a curated list of resources created for Typst.

If you need to load functions and variables from another file within your project, for example to use a template, you can use the same [import](#) statement with a file name rather than a package specification. To instead include the textual content of another file, you can use an [include](#) statement. It will retrieve the content of the specified file and put it in your document.

## How do I input maths?

To enter math mode in Typst, just enclose your equation in dollar signs. You can enter display mode by adding spaces or newlines between the equation's contents and its enclosing dollar signs.

The sum of the numbers from \$1\$ to \$n\$ is:

```
$ sum_(k=1)^n k = (n(n+1))/2 $
```

The sum of the numbers from 1 to  $n$  is:

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

[Math mode](#) works differently than regular markup or code mode.

Numbers and single characters are displayed verbatim, while multiple consecutive (non-number) characters will be interpreted as Typst variables.

Typst pre-defines a lot of useful variables in math mode. All Greek (`alpha`, `beta`, ...) and some Hebrew letters (`alef`, `bet`, ...) are available through their name. Some symbols are additionally available through shorthands, such as `<=`, `>=`, and `->`.

Refer to the [symbol pages](#) for a full list of the symbols. If a symbol is missing, you can also access it through a [Unicode escape sequence](#).

Alternate and related forms of symbols can often be selected by [appending a modifier](#) after a period. For example, `arrow.l.squiggly` inserts a squiggly left-pointing arrow. If you want to insert multiletter text in your expression instead, enclose it in double quotes:

```
$ delta "if" x <= 5 $
```

$$\delta \text{ if } x \leq 5$$

In Typst, delimiters will scale automatically for their expressions, just as if

`\left` and `\right` commands were implicitly inserted in LaTeX. You can customize delimiter behaviour using the [lx function](#). To prevent a pair of delimiters from scaling, you can escape them with backslashes.

Typst will automatically set terms around a slash / as a fraction while honoring operator precedence. All round parentheses not made redundant by the fraction will appear in the output.

```
$ f(x) = (x + 1) / x $
```

$$f(x) = \frac{x+1}{x}$$

[Sub- and superscripts](#) work similarly in Typst and LaTeX. `$x^2$` will produce a superscript, `$x_2$` yields a subscript. If you want to include more than one value in a sub- or superscript, enclose their contents in parentheses:

```
$x_(a -> epsilon)$.
```

Since variables in math mode do not need to be prepended with a # or a /, you can also call functions without these special characters:

```
$ f(x, y) := cases(
 1 "if" (x dot y)/2 <= 0,
 2 "if" x "is even",
 3 "if" x in NN,
 4 "else",
```

```
) $
```

$$f(x, y) := \begin{cases} 1 & \text{if } \frac{x \cdot y}{2} \leq 0 \\ 2 & \text{if } x \text{ is even} \\ 3 & \text{if } x \in \mathbb{N} \\ 4 & \text{else} \end{cases}$$

The above example uses the [cases function](#) to describe f. Within the cases function, arguments are delimited using commas and the arguments are also interpreted as math. If you need to interpret arguments as Typst values instead, prefix them with a #:

```
$ (a + b)^2
= a^2
+ text(fill: #maroon, 2 a b)
+ b^2 $
```

$$(a + b)^2 = a^2 + 2ab + b^2$$

You can use all Typst functions within math mode and insert any content. If you want them to work normally, with code mode in the argument list, you can prefix their call with a #. Nobody can stop you from using rectangles or emoji as your variables anymore:

```
$ sum^10_(?=1)
#rect(width: 4mm, height: 2mm) /?
= ? maltese $
```

$$\sum_{\mathfrak{z}=1}^{10} \boxed{\square} = \text{🧠} \times$$

If you'd like to enter your mathematical symbols directly as Unicode,  
that is possible, too!

Math calls can have two-dimensional argument lists using ; as a delimiter. The most common use for this is the [mat function](#) that creates matrices:

```
$ mat (
 1, 2, ..., 10;
 2, 2, ..., 10;
 dots.v, dots.v, dots.down, dots.v;
 10, 10, ..., 10;
) $
```

$$\begin{pmatrix} 1 & 2 & \dots & 10 \\ 2 & 2 & \dots & 10 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}$$

## How do I get the "LaTeX look?"

Papers set in LaTeX have an unmistakeable look. This is mostly due to their font, Computer Modern, justification, narrow line spacing, and wide margins.

The example below

- sets wide [margins](#)
- enables [justification](#), [tighter lines](#) and [first-line-indent](#)
- [sets the font](#) to "New Computer Modern", an OpenType derivative of

Computer Modern for both text and [code blocks](#)

- disables paragraph [spacing](#)

- increases [spacing](#) around [headings](#)

```
#set page(margin: 1.75in)
#set par(leading: 0.55em, spacing: 0.55em, first-line-
```

```
indent: 1.8em, justify: true)
#set text(font: "New Computer Modern")
#show raw: set text(font: "New Computer Modern Mono")
#show heading: set block(above: 1.4em, below: 1em)
```

This should be a good starting point! If you want to go further, why not create a reusable template?

## Bibliographies

Typst includes a fully-featured bibliography system that is compatible with BibTeX files. You can continue to use your `.bib` literature libraries by loading them with the [bibliography](#) function. Another possibility is to use [Typst's YAML-based native format](#).

Typst uses the Citation Style Language to define and process citation and bibliography styles. You can compare CSL files to BibLaTeX's `.bbx` files. The compiler already includes [over 80 citation styles](#), but you can use any CSL-compliant style from the [CSL repository](#) or write your own.

You can cite an entry in your bibliography or reference a label in your document with the same syntax: `@key` (this would reference an entry called `key`). Alternatively, you can use the [cite](#) function.

Alternative forms for your citation, such as year only and citations for natural use in prose (cf. `\citet` and `\textcite`) are available with [`#cite\(<key>, form: "prose"\)`](#).

You can find more information on the documentation page of the [bibliography](#) function.

# What limitations does Typst currently have compared to LaTeX?

Although Typst can be a LaTeX replacement for many today, there are still features that Typst does not (yet) support. Here is a list of them which, where applicable, contains possible workarounds.

- **Well-established plotting ecosystem.** LaTeX users often create elaborate charts along with their documents in PGF/TikZ. The Typst ecosystem does not yet offer the same breadth of available options, but the ecosystem around the [cetz](#) package is catching up quickly.

- **Change page margins without a pagebreak.** In LaTeX, margins can always be adjusted, even without a pagebreak. To change margins in Typst, you use the [page function](#) which will force a page break. If you just want a few paragraphs to stretch into the margins, then reverting to the old margins, you can use the [pad function](#) with negative padding.

- **Include PDFs as images.** In LaTeX, it has become customary to insert vector graphics as PDF or EPS files. Typst supports neither format as an image format, but you can easily convert both into SVG files with [online tools](#) or [Inkscape](#). The web app will automatically convert PDF files to SVG files upon uploading them.

## 4.2 Page setup guide

Your page setup is a big part of the first impression your document gives. Line lengths, margins, and columns influence [appearance](#) and [legibility](#) while the right headers and footers will help your reader easily navigate your document.

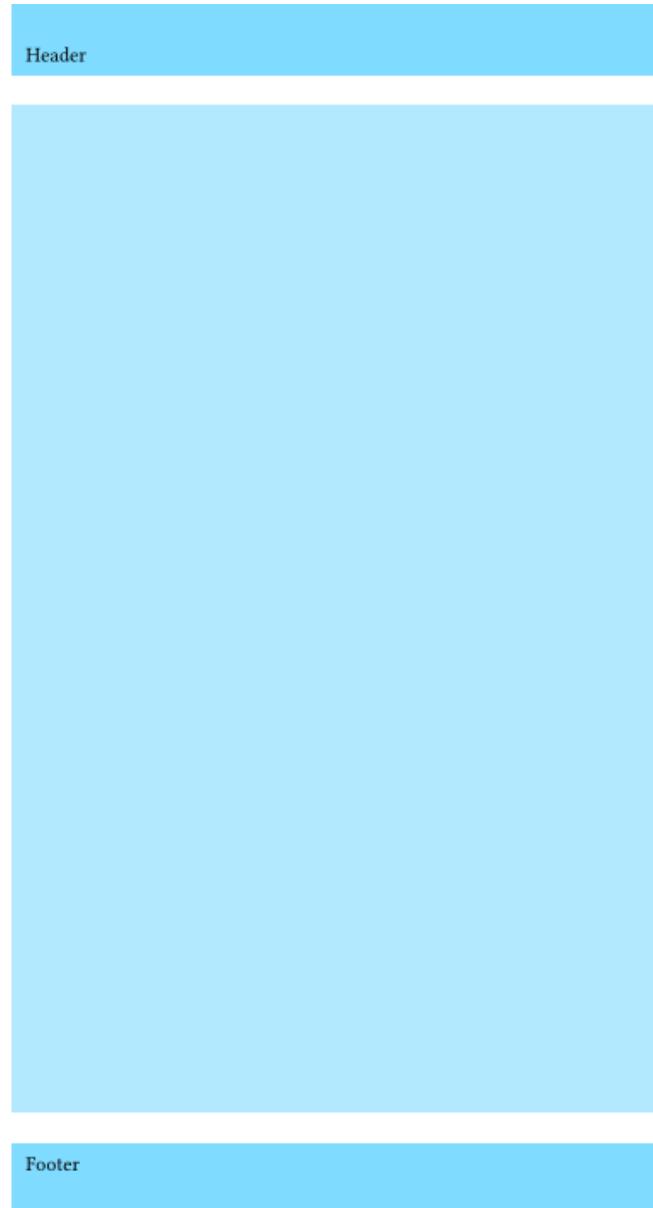
This guide will help you to customize pages, margins, headers, footers, and page numbers so that they are the right fit for your content and you can get started with writing.

In Typst, each page has a width, a height, and margins on all four sides. The top and bottom margins may contain a header and footer. The set rule of the [page](#) element is where you control all of the page setup. If you make changes with this set rule, Typst will ensure that there is a new and conforming empty page afterward, so it may insert a page break. Therefore, it is best to specify your [page](#) set rule at the start of your document or in your template.

```
#set rect(
 width: 100%,
 height: 100%,
 inset: 4pt,
)

#set page(
 paper: "iso-b7",
 header: rect(fill: aqua) [Header],
 footer: rect(fill: aqua) [Footer],
 number-align: center,
)

#rect(fill: aqua.lighten(40%))
```



This example visualizes the dimensions for page content, headers, and footers. The page content is the page size (ISO B7) minus each side's default margin. In the top and the bottom margin, there are stroked rectangles visualizing the header and footer. They do not touch the main content, instead, they are offset by 30% of the respective margin. You can control this offset by specifying the [header-ascent](#) and [footer-descent](#) arguments.

Below, the guide will go more into detail on how to accomplish common page setup requirements with examples.

## Customize page size and margins

Typst's default page size is A4 paper. Depending on your region and your use case, you will want to change this. You can do this by using the [page](#) set rule and passing it a string argument to use a common page size. Options include the complete ISO 216 series (e.g. "iso-a4", "iso-c2"), customary US formats like "us-legal" or "us-letter", and more. Check out the reference for the [page's paper argument](#) to learn about all available options.

```
#set page("us-letter")
```

This page likes freedom.

This page likes freedom.

If you need to customize your page size to some dimensions, you can specify the named arguments `width` and `height` instead.

```
#set page(width: 12cm, height: 12cm)
```

This page is a square.

This page is a square.

## Change the page's margins

Margins are a vital ingredient for good typography: [Typographers consider lines that fit between 45 and 75 characters best length for legibility](#) and your margins and [columns](#) help define line widths. By default, Typst will create margins proportional to the page size of your document. To set custom margins, you will use the [margin](#) argument in the [page](#) set rule.

The `margin` argument will accept a length if you want to set all margins to the same width. However, you often want to set different margins on each side. To do this, you can pass a dictionary:

```
#set page(margin: (
 top: 3cm,
 bottom: 2cm,
 x: 1.5cm,
))
```

```
#lorem(100)
```

Lorem ipsum dolor sit amet,  
  consectetur adipiscing elit, sed do  
  eiusmod tempor incididunt ut  
  labore et dolore magna aliquam  
  quaerat voluptatem. Ut enim  
  aeque doleamus animo, cum  
  corpore dolemus, fieri tamen  
  permagna accessio potest, si  
  aliquod aeternum et infinitum  
  impendere malum nobis  
  opinemur. Quod idem licet  
  transferre in voluptatem, ut  
  postea variari voluptas  
  distinguique possit, augeri  
  amplificarique non possit. At  
  etiam Athenis, ut e patre  
  audiebam facete et urbane Stoicos  
  irridente, statua est in quo a nobis  
  philosophia defensa et collaudata  
  est, cum id, quod maxime placeat,  
  facere possimus, omnis voluptas  
  assumenda est, omnis dolor  
  repellendus. Temporibus autem  
  quibusdam et.

The page margin dictionary can have keys for each side (`top`, `bottom`, `left`, `right`), but you can also control left and right together by setting the `x` key

of the margin dictionary, like in the example. Likewise, the top and bottom margins can be adjusted together by setting the `y` key.

If you do not specify margins for all sides in the margin dictionary, the old margins will remain in effect for the unset sides. To prevent this and set all remaining margins to a common size, you can use the `rest` key. For example,

```
#set page(margin: (left: 1.5in, rest: 1in))
```

will set the left margin to 1.5 inches and the remaining margins to one inch.

## Different margins on alternating pages

Sometimes, you'll need to alternate horizontal margins for even and odd pages, for example, to have more room towards the spine of a book than on the outsides of its pages. Typst keeps track of whether a page is to the left or right of the binding. You can use this information and set the `inside` or `outside` keys of the margin dictionary. The `inside` margin points towards the spine, and the `outside` margin points towards the edge of the bound book.

```
#set page(margin: (inside: 2.5cm, outside: 2cm, y: 1.75cm))
```

Typst will assume that documents written in Left-to-Right scripts are bound on the left while books written in Right-to-Left scripts are bound on the right. However, you will need to change this in some cases: If your first page is output by a different app, the binding is reversed from Typst's perspective. Also, some books, like English-language Mangas are customarily bound on the right, despite English using Left-to-Right script.

To change the binding side and explicitly set where the `inside` and `outside` are, set the [binding](#) argument in the [page](#) set rule.

```
// Produce a book bound on the right,
// even though it is set in Spanish.
#set text(lang: "es")
#set page(binding: right)
```

If `binding` is `left`, `inside` margins will be on the left on odd pages, and vice versa.

## Add headers and footers

Headers and footers are inserted in the top and bottom margins of every page.

You can add custom headers and footers or just insert a page number.

In case you need more than just a page number, the best way to insert a header and a footer are the [header](#) and [footer](#) arguments of the [page](#) set rule. You can pass any content as their values:

```
#set page(header: [
 Lisa Strassner's Thesis
 #h(1fr)
 National Academy of Sciences
])
```

```
#lorem(150)
```

*Lisa Strassner's Thesis*

National Academy of Sciences

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed  
  do eiusmod tempor incididunt ut labore et dolore magna  
  aliquam quaerat voluptatem. Ut enim aequo doleamus animo,  
  cum corpore dolemus, fieri tamen permagna accessio potest,  
  si aliquod aeternum et infinitum impendere malum nobis  
  opinemur. Quod idem licet transferre in voluptatem, ut  
  postea variari voluptas distingue possit, augeri  
  amplificarique non possit. At etiam Athenis, ut e patre  
  audiebam facete et urbane Stoicos irridente, statua est in quo  
  a nobis philosophia defensa et collaudata est, cum id, quod  
  maxime placeat, facere possimus, omnis voluptas assumenda  
  est, omnis dolor repellendus. Temporibus autem quibusdam  
  et aut officiis debitibus aut rerum necessitatibus saepe eveniet,  
  ut et voluptates repudiandae sint et molestiae non  
  recusandae. Itaque earum rerum defuturum, quas natura non  
  depravata desiderat. Et quem ad me accedit, saluto: 'chaere,'  
  inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!'  
  hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius.

Headers are bottom-aligned by default so that they do not collide with the top edge of the page. You can change this by wrapping your header in the [align](#) function.

## Different header and footer on specific pages

You'll need different headers and footers on some pages. For example, you may not want a header and footer on the title page. The example below shows how to conditionally remove the header on the first page:

```
#set page(header: context {
 if counter(page).get().first() > 1 [
 Lisa Strassner's Thesis
 #h(1fr)
 National Academy of Sciences
]
})

#lorem(150)
```

This example may look intimidating, but let's break it down: By using the `context` keyword, we are telling Typst that the header depends on where we are in the document. We then ask Typst if the page `counter` is larger than one at our (context-dependent) current position. The page counter starts at one, so we are skipping the header on a single page. Counters may have multiple levels. This feature is used for items like headings, but the page counter will always have a single level, so we can just look at the first one.

You can, of course, add an `else` to this example to add a different header to the first page instead.

## Adapt headers and footers on pages with specific elements

The technique described in the previous section can be adapted to perform more advanced tasks using Typst's labels. For example, pages with big tables

could omit their headers to help keep clutter down. We will mark our tables with a `<big-table>` [label](#) and use the [query system](#) to find out if such a label exists on the current page:

```
#set page(header: context {
 let page-counter =
 let matches = query(<big-table>)
 let current = counter(page).get()
 let has-table = matches.any(m =>
 counter(page).at(m.location()) == current
)

 if not has-table [
 Lisa Strassner's Thesis
 #h(1fr)
 National Academy of Sciences
]
 }))

#lorem(100)
#pagebreak()


```

Here, we query for all instances of the `<big-table>` label. We then check that none of the tables are on the page at our current position. If so, we print the header. This example also uses variables to be more concise. Just as above, you could add an `else` to add another header instead of deleting it.

## Add and customize page numbers

Page numbers help readers keep track of and reference your document more easily. The simplest way to insert page numbers is the [numbering](#) argument of

the [page](#) set rule. You can pass a [\*numbering pattern\*](#) string that shows how you want your pages to be numbered.

```
#set page(numbering: "1")
```

This is a numbered page.

This is a numbered page.

Above, you can check out the simplest conceivable example. It adds a single Arabic page number at the center of the footer. You can specify other characters than "1" to get other numerals. For example, "i" will yield lowercase Roman numerals. Any character that is not interpreted as a number will be output as-is. For example, put dashes around your page number by typing this:

```
#set page(numbering: "- 1 -")
```

This is a — numbered — page.

This is a — numbered — page.

— 1 —

You can add the total number of pages by entering a second number character in the string.

```
#set page (numbering: "1 of 1")
```

This is one of many numbered pages.

This is one of many numbered pages.

1 of 1

Go to the [numbering function reference](#) to learn more about the arguments you can pass here.

In case you need to right- or left-align the page number, use the [number-align](#) argument of the [page](#) set rule. Alternating alignment between even and

odd pages is not currently supported using this property. To do this, you'll need to specify a custom footer with your footnote and query the page counter as described in the section on conditionally omitting headers and footers.

## Custom footer with page numbers

Sometimes, you need to add other content than a page number to your footer. However, once a footer is specified, the [numbering](#) argument of the [page](#) set rule is ignored. This section shows you how to add a custom footer with page numbers and more.

```
#set page(footer: context [
 American Society of Proceedings
 #h(1fr)
 #counter(page).display(
 "1/1",
 both: true,
)
])
```

This page has a custom footer.

This page has a custom footer.

**American Society of Proceedings**

1/1

First, we add some strongly emphasized text on the left and add free space to fill the line. Then, we call `counter(page)` to retrieve the page counter and use its `display` function to show its current value. We also set `both` to

`true` so that our numbering pattern applies to the current *and* final page number.

We can also get more creative with the page number. For example, let's insert a circle for each page.

```
#set page(footer: context [
 Fun Typography Club
 #h(1fr)
 #let (num,) = counter(page).get()
 #let circles = num * (
 box(circle(
 radius: 2pt,
 fill: navy,
)),
)
 #box(
 inset: (bottom: 1pt),
 circles.join(h(1pt))
)
])
```

This page has a custom footer.

This page has a custom footer.

## Fun Typography Club



In this example, we use the number of pages to create an array of [circles](#). The circles are wrapped in a [box](#) so they can all appear on the same

line because they are blocks and would otherwise create paragraph breaks.

The length of this [array](#) depends on the current page number.

We then insert the circles at the right side of the footer, with 1pt of space between them. The `join` method of an array will attempt to [join](#) the different values of an array into a single value, interspersed with its argument. In our case, we get a single content value with circles and spaces between them that we can use with the `align` function. Finally, we use another box to ensure that the text and the circles can share a line and use the [inset argument](#) to raise the circles a bit so they line up nicely with the text.

## Reset the page number and skip pages

Do you, at some point in your document, need to reset the page number?

Maybe you want to start with the first page only after the title page. Or maybe you need to skip a few page numbers because you will insert pages into the final printed product.

The right way to modify the page number is to manipulate the page [counter](#).

The simplest manipulation is to set the counter back to 1.

```
#counter(page).update(1)
```

This line will reset the page counter back to one. It should be placed at the start of a page because it will otherwise create a page break. You can also update the counter given its previous value by passing a function:

```
#counter(page).update(n => n + 5)
```

In this example, we skip five pages. `n` is the current value of the page counter and `n + 5` is the return value of our function.

In case you need to retrieve the actual page number instead of the value of the page counter, you can use the [page](#) method on the return value of the [here](#) function:

```
#counter(page).update(n => n + 5)

// This returns one even though the
// page counter was incremented by 5.
#context here().page()
```

1

You can also obtain the page numbering pattern from the location returned by `here` with the [page-numbering](#) method.

## Add columns

Add columns to your document to fit more on a page while maintaining legible line lengths. Columns are vertical blocks of text which are separated by some whitespace. This space is called the gutter.

To lay out your content in columns, just specify the desired number of columns in a [page](#) set rule. To adjust the amount of space between the columns, add a set rule on the [columns function](#), specifying the `gutter` parameter.

```
#set page(columns: 2)
#set columns(gutter: 12pt)

#lorem(30)
```

|                                                                                                                                    |                                                                                                   |
|------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| Lorem ipsum dolor<br>sit amet, consectetur<br>adipiscing elit, sed do<br>eiusmod tempor<br>incididunt ut labore<br>et dolore magna | aliquam quaerat<br>voluptatem. Ut enim<br>aequo doleamus<br>animo, cum corpore<br>dolemus, fieri. |
|------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|

Very commonly, scientific papers have a single-column title and abstract, while the main body is set in two-columns. To achieve this effect, Typst's [place function](#) can temporarily escape the two-column layout by

specifying `float: true` and `scope: "parent"`:

```
#set page(columns: 2)
#set par(justify: true)

#place(
 top + center,
 float: true,
 scope: "parent",
 text(1.4em, weight: "bold") [
 Impacts of Odobenidae
],
)
== About seals in the wild
#lorem(80)
```

## Impacts of Odobenidae

### About seals in the wild

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna ali-

  quam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opine-

*Floating placement* refers to elements being pushed to the top or bottom of the column or page, with the remaining content flowing in between. It is also frequently used for [figures](#).

### Use columns anywhere in your document

To create columns within a nested layout, e.g. within a rectangle, you can use the [columns function](#) directly. However, it really should only be used within nested layouts. At the page-level, the page set rule is preferable because it has better interactions with things like page-level floats, footnotes, and line numbers.

```
#rect(
 width: 6cm,
 height: 3.5cm,
 columns(2, gutter: 12pt) [
 In the dimly lit gas station,
 a solitary taxi stood silently,
 its yellow paint fading with
 time. Its windows were dark,
 its engine idle, and its tires
 rested on the cold concrete.
]
)
```

|                                                                                                           |                                                                                                         |
|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| In the dimly lit<br>gas station, a<br>solitary taxi<br>stood silently,<br>its yellow paint<br>fading with | time. Its<br>windows were<br>dark, its engine<br>idle, and its tires<br>rested on the<br>cold concrete. |
|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|

## Balanced columns

If the columns on the last page of a document differ greatly in length, they may create a lopsided and unappealing layout. That's why typographers will often equalize the length of columns on the last page. This effect is called balancing columns. Typst cannot yet balance columns automatically. However, you can balance columns manually by placing [#colbreak\(\)](#) at an appropriate spot in your markup, creating the desired column break manually.

## One-off modifications

You do not need to override your page settings if you need to insert a single page with a different setup. For example, you may want to insert a page that's flipped to landscape to insert a big table or change the margin and columns for your title page. In this case, you can call [page](#) as a function with your content as an argument and the overrides as the other arguments. This will insert enough new pages with your overridden settings to place your content on them. Typst will revert to the page settings from the set rule after the call.

```
#page (flipped: true) [
 = Multiplication table

 #table (
```

```

columns: 5 * (1fr,) ,
..for x in range(1, 10) {
 for y in range(1, 6) {
 (str(x*y),)
 }
}
]

```

## Multiplication table

|   |    |    |    |    |
|---|----|----|----|----|
| 1 | 2  | 3  | 4  | 5  |
| 2 | 4  | 6  | 8  | 10 |
| 3 | 6  | 9  | 12 | 15 |
| 4 | 8  | 12 | 16 | 20 |
| 5 | 10 | 15 | 20 | 25 |
| 6 | 12 | 18 | 24 | 30 |
| 7 | 14 | 21 | 28 | 35 |
| 8 | 16 | 24 | 32 | 40 |
| 9 | 18 | 27 | 36 | 45 |

## 4.3 Table guide

Tables are a great way to present data to your readers in an easily readable, compact, and organized manner. They are not only used for numerical values, but also survey responses, task planning, schedules, and more. Because of this wide set of possible applications, there is no single best way to lay out a table. Instead, think about the data you want to highlight, your document's overarching design, and ultimately how your table can best serve your readers.

Typst can help you with your tables by automating styling, importing data from other applications, and more! This guide takes you through a few of the most common questions you may have when adding a table to your document with Typst. Feel free to skip to the section most relevant to you – we designed this guide to be read out of order.

If you want to look up a detail of how tables work, you should also [check out their reference page](#). And if you are looking for a table of contents rather than a normal table, the reference page of the [outline function](#) is the right place to learn more.

## How to create a basic table?

In order to create a table in Typst, use the [table function](#). For a basic table, you need to tell the table function two things:

- The number of columns
  - The content for each of the table cells

So, let's say you want to create a table with two columns describing the ingredients for a cookie recipe:

```
#table(
 columns: 2,
 [*Amount*], [*Ingredient*],
 [360g], [Baking flour],
 [250g], [Butter (room temp.)],
 [150g], [Brown sugar],
 [100g], [Cane sugar],
 [100g], [70% cocoa chocolate],
 [100g], [35–40% cocoa chocolate],
 [2], [Eggs],
 [Pinch], [Salt],
 [Drizzle], [Vanilla extract],
```

)

| Amount  | Ingredient             |
|---------|------------------------|
| 360g    | Baking flour           |
| 250g    | Butter (room temp.)    |
| 150g    | Brown sugar            |
| 100g    | Cane sugar             |
| 100g    | 70% cocoa chocolate    |
| 100g    | 35-40% cocoa chocolate |
| 2       | Eggs                   |
| Pinch   | Salt                   |
| Drizzle | Vanilla extract        |

This example shows how to call, configure, and populate a table. Both the column count and cell contents are passed to the table as arguments.

The [argument list](#) is surrounded by round parentheses. In it, we first pass the column count as a named argument. Then, we pass multiple [content blocks](#) as positional arguments. Each content block contains the contents for a single cell.

To make the example more legible, we have placed two content block arguments on each line, mimicking how they would appear in the table. You could also write each cell on its own line. Typst does not care on which line you place the arguments. Instead, Typst will place the content cells from left to right (or right to left, if that is the writing direction of your language) and then from top to bottom. It will automatically add enough rows to your table so that it fits all of your content.

It is best to wrap the header row of your table in the [table.header function](#).

This clarifies your intent and will also allow future versions of Typst to make the output more accessible to users with a screen reader:

```
#table(
 columns: 2,
 table.header[*Amount*] [*Ingredient*],
 [360g], [Baking flour],
 // ... the remaining cells
)
```

| Amount  | Ingredient             |
|---------|------------------------|
| 360g    | Baking flour           |
| 250g    | Butter (room temp.)    |
| 150g    | Brown sugar            |
| 100g    | Cane sugar             |
| 100g    | 70% cocoa chocolate    |
| 100g    | 35-40% cocoa chocolate |
| 2       | Eggs                   |
| Pinch   | Salt                   |
| Drizzle | Vanilla extract        |

You could also write a show rule that automatically [strongly emphasizes](#) the contents of the first cells for all tables. This quickly becomes useful if your document contains multiple tables!

```
#show table.cell.where(y: 0): strong

#table(
 columns: 2,
 table.header[Amount][Ingredient],
 [360g], [Baking flour],
 // ... the remaining cells
)
```

| Amount  | Ingredient             |
|---------|------------------------|
| 360g    | Baking flour           |
| 250g    | Butter (room temp.)    |
| 150g    | Brown sugar            |
| 100g    | Cane sugar             |
| 100g    | 70% cocoa chocolate    |
| 100g    | 35-40% cocoa chocolate |
| 2       | Eggs                   |
| Pinch   | Salt                   |
| Drizzle | Vanilla extract        |

We are using a show rule with a selector for cell coordinates here instead of applying our styles directly to `table.header`. This is due to a current limitation of Typst that will be fixed in a future release.

Congratulations, you have created your first table! Now you can proceed to [change column sizes](#), [adjust the strokes](#), [add striped rows](#), and more!

## How to change the column sizes?

If you create a table and specify the number of columns, Typst will make each column large enough to fit its largest cell. Often, you want something different, for example, to make a table span the whole width of the page. You can provide a list, specifying how wide you want each column to be, through the `columns` argument. There are a few different ways to specify column widths:

- First, there is `auto`. This is the default behavior and tells Typst to grow the column to fit its contents. If there is not enough space, Typst will try its best to distribute the space among the `auto`-sized columns.

•[Lengths](#) like `6cm`, `0.7in`, or `120pt`. As usual, you can also use the font-dependent `em` unit. This is a multiple of your current font size. It's useful if you want to size your table so that it always fits about the same amount of text, independent of font size.

•A [ratio in percent](#) such as `40%`. This will make the column take up 40% of the total horizontal space available to the table, so either the inner width of the page or the table's container. You can also mix ratios and lengths into [relative lengths](#). Be mindful that even if you specify a list of column widths that sum up to 100%, your table could still become larger than its container. This is because there can be [gutter](#) between columns that is not included in the column widths.

If you want to make a table fill the page, the next option is often very useful.

•A [fractional part of the free space](#) using the `fr` unit, such as `1fr`. This unit allows you to distribute the available space to columns. It works as follows: First, Typst sums up the lengths of all columns that do not use `frs`. Then, it determines how much horizontal space is left. This horizontal space then gets distributed to all columns denominated in `frs`. During this process, a `2fr` column will become twice as wide as a `1fr` column. This is where the name comes from: The width of the column is its fraction of the total fractionally sized columns.

Let's put this to use with a table that contains the dates, numbers, and descriptions of some routine checks. The first two columns are `auto`-sized and the last column is `1fr` wide as to fill the whole page.

```
#table(
 columns: (auto, auto, lfr),
 table.header[Date] [°No] [Description],
 [24/01/03], [813], [Filtered participant pool],
 [24/01/03], [477], [Transitioned to sec. regimen],
 [24/01/11], [051], [Cycled treatment substrate],
)
```

| Date     | °No | Description                     |
|----------|-----|---------------------------------|
| 24/01/03 | 813 | Filtered participant pool       |
| 24/01/03 | 477 | Transitioned to sec.<br>regimen |
| 24/01/11 | 051 | Cycled treatment substrate      |

Here, we have passed our list of column lengths as an [array](#), enclosed in round parentheses, with its elements separated by commas. The first two columns are automatically sized, so that they take on the size of their content and the third column is sized as `lfr` so that it fills up the remainder of the space on the page.

If you wanted to instead change the second column to be a bit more spacious, you could replace its entry in the `columns` array with a value like `6em`.

## How to caption and reference my table?

A table is just as valuable as the information your readers draw from it. You can enhance the effectiveness of both your prose and your table by making a clear connection between the two with a cross-reference. Typst can help you with automatic [references](#) and the [figure function](#).

Just like with images, wrapping a table in the `figure` function allows you to add a caption and a label, so you can reference the figure elsewhere. Wrapping your

table in a figure also lets you use the figure's `placement` parameter to float it to the top or bottom of a page.

Let's take a look at a captioned table and how to reference it in prose:

```
#show table.cell.where(y: 0): set text(weight: "bold")

#figure(
 table(
 columns: 4,
 stroke: none,

 table.header[Test Item] [Specification] [Test
Result] [Compliance],
 [Voltage], [220V ± 5%], [218V], [Pass],
 [Current], [5A ± 0.5A], [4.2A], [Fail],
),
 caption: [Probe results for design A],
) <probe-a>
```

The results from `@probe-a` show that the design is not yet optimal.

We will show how its performance can be improved in this section.

| Test Item | Specification | Test Result | Compliance |
|-----------|---------------|-------------|------------|
| Voltage   | 220V ± 5%     | 218V        | Pass       |
| Current   | 5A ± 0.5A     | 4.2A        | Fail       |

Table 1: Probe results for design A

The results from Table 1 show that the design is not yet optimal. We will show how its performance can be improved in this section.

The example shows how to wrap a table in a figure, set a caption and a label, and how to reference that label. We start by using the `figure` function. It expects the contents of the figure as a positional argument. We just put the `table` function call in its argument list, omitting the `#` character because it is only needed when calling a function in markup

mode. We also add the caption as a named argument (above or below) the table.

After the figure call, we put a label in angle brackets (`<probe-a>`). This tells Typst to remember this element and make it referenceable under this name throughout your document. We can then reference it in prose by using the at sign and the label name `@probe-a`. Typst will print a nicely formatted reference and automatically update the label if the table's number changes.

## How to get a striped table?

Many tables use striped rows or columns instead of strokes to differentiate between rows and columns. This effect is often called *zebra stripes*. Tables with zebra stripes are popular in Business and commercial Data Analytics applications, while academic applications tend to use strokes instead.

To add zebra stripes to a table, we use the `table` function's `fill` argument. It can take three kinds of arguments:

- A single color (this can also be a gradient or a pattern) to fill all cells with. Because we want some cells to have another color, this is not useful if we want to build zebra tables.
- An array with colors which Typst cycles through for each column. We can use an array with two elements to get striped columns.
- A function that takes the horizontal coordinate `x` and the vertical coordinate `y` of a cell and returns its fill. We can use this to create horizontal stripes or [checkerboard patterns](#).

Let's start with an example of a horizontally striped table:

```
#set text(font: "IBM Plex Sans")

// Medium bold table header.
#show table.cell.where(y: 0): set text(weight: "medium")

// Bold titles.
#show table.cell.where(x: 1): set text(weight: "bold")

// See the strokes section for details on this!
#let frame(stroke) = (x, y) => (
 left: if x > 0 { 0pt } else { stroke },
 right: stroke,
 top: if y < 2 { stroke } else { 0pt },
 bottom: stroke,
)

#set table(
 fill: (rgb("EAF2F5"), none),
 stroke: frame(rgb("21222C")),
)


```

| Month    | Title                         | Author              | Genre         |
|----------|-------------------------------|---------------------|---------------|
| January  | <b>The Great Gatsby</b>       | F. Scott Fitzgerald | Classic       |
| February | <b>To Kill a Mockingbird</b>  | Harper Lee          | Drama         |
| March    | <b>1984</b>                   | George Orwell       | Dystopian     |
| April    | <b>The Catcher in the Rye</b> | J.D. Salinger       | Coming-of-Age |

This example shows a book club reading list. The line `fill`:

`(rgb("EAF2F5"), none)` in `table`'s `set` rule is all that is needed to add striped columns. It tells Typst to alternate between coloring columns with a light blue (in the `rgb` function call) and nothing (`none`). Note that we extracted all of our styling from the `table` function call itself into `set` and `show` rules, so that we can automatically reuse it for multiple tables.

Because setting the stripes itself is easy we also added some other styles to make it look nice. The other code in the example provides a dark blue `stroke` around the table and below the first line and emboldens the first row and the column with the book title. See the [strokes](#) section for details on how we achieved this stroke configuration.

Let's next take a look at how we can change only the `set` rule to achieve horizontal stripes instead:

```
#set table(
 fill: (_, y) => if calc.odd(y) { rgb("EAF2F5") },
 stroke: frame(rgb("21222C")),
)
```

| Month    | Title                  | Author              | Genre         |
|----------|------------------------|---------------------|---------------|
| January  | The Great Gatsby       | F. Scott Fitzgerald | Classic       |
| February | To Kill a Mockingbird  | Harper Lee          | Drama         |
| March    | 1984                   | George Orwell       | Dystopian     |
| April    | The Catcher in the Rye | J.D. Salinger       | Coming-of-Age |

We just need to replace the `set` rule from the previous example with this one and get horizontal stripes instead. Here, we are passing a function to `fill`. It discards the horizontal coordinate with an underscore and then

checks if the vertical coordinate `y` of the cell is odd. If so, the cell gets a light blue fill, otherwise, no fill is returned.

Of course, you can make this function arbitrarily complex. For example, if you want to stripe the rows with a light and darker shade of blue, you could do something like this:

```
#set table(
 fill: (_, y) => (none, rgb("EAF2F5"),
 rgb("DDEAEF")).at(calc.rem(y, 3)),
 stroke: frame(rgb("21222C")),
)
```

| Month    | Title                  | Author              | Genre         |
|----------|------------------------|---------------------|---------------|
| January  | The Great Gatsby       | F. Scott Fitzgerald | Classic       |
| February | To Kill a Mockingbird  | Harper Lee          | Drama         |
| March    | 1984                   | George Orwell       | Dystopian     |
| April    | The Catcher in the Rye | J.D. Salinger       | Coming-of-Age |

This example shows an alternative approach to write our fill function. The function uses an array with three colors and then cycles between its values for each row by indexing the array with the remainder of `y` divided by 3.

Finally, here is a bonus example that uses the `stroke` to achieve striped rows:

```
#set table(
 stroke: (x, y) => (
 y: 1pt,
 left: if x > 0 { 0pt } else if calc.even(y) { 1pt },
 right: if calc.even(y) { 1pt },
),
)
```

| Month    | Title                  | Author              | Genre         |
|----------|------------------------|---------------------|---------------|
| January  | The Great Gatsby       | F. Scott Fitzgerald | Classic       |
| February | To Kill a Mockingbird  | Harper Lee          | Drama         |
| March    | 1984                   | George Orwell       | Dystopian     |
| April    | The Catcher in the Rye | J.D. Salinger       | Coming-of-Age |

## Manually overriding a cell's fill color

Sometimes, the fill of a cell needs not to vary based on its position in the table, but rather based on its contents. We can use the [table.cell element](#) in the table's parameter list to wrap a cell's content and override its fill.

For example, here is a list of all German presidents, with the cell borders colored in the color of their party.

```
#set text(font: "Roboto")

#let cdu(name) = ([CDU], table.cell(fill: black, text(fill: white, name)))
#let spd(name) = ([SPD], table.cell(fill: red, text(fill: white, name)))
#let fdp(name) = ([FDP], table.cell(fill: yellow, name))


```

```
[2017-], .. spd[Frank-Walter-Steinmeier],
)
```

| Tenure    | Party | President               |
|-----------|-------|-------------------------|
| 1949-1959 | FDP   | Theodor Heuss           |
| 1959-1969 | CDU   | Heinrich Lübke          |
| 1969-1974 | SPD   | Gustav Heinemann        |
| 1974-1979 | FDP   | Walter Scheel           |
| 1979-1984 | CDU   | Karl Carstens           |
| 1984-1994 | CDU   | Richard von Weizsäcker  |
| 1994-1999 | CDU   | Roman Herzog            |
| 1999-2004 | SPD   | Johannes Rau            |
| 2004-2010 | CDU   | Horst Köhler            |
| 2010-2012 | CDU   | Christian Wulff         |
| 2012-2017 | n/a   | Joachim Gauck           |
| 2017-     | SPD   | Frank-Walter-Steinmeier |

In this example, we make use of variables because there only have been a total of three parties whose members have become president (and one unaffiliated president). Their colors will repeat multiple times, so we store a function that produces an array with their party's name and a table cell with that party's color and the president's name (`cdu`, `spd`, and `fdp`). We then use these functions in the `table` argument list instead of directly adding the name. We use the [spread operator](#) `...` to turn the items of the arrays into single cells. We could also write something like `[FDP]`, `table.cell(fill: yellow) [Theodor Heuss]` for each cell directly in the `table`'s argument list, but that becomes unreadable, especially for the

parties whose colors are dark so that they require white text. We also delete vertical strokes and set the font to Roboto.

The party column and the cell color in this example communicate redundant information on purpose: Communicating important data using color only is a bad accessibility practice. It disadvantages users with vision impairment and is in violation of universal access standards, such as the [WCAG 2.1 Success Criterion 1.4.1](#). To improve this table, we added a column printing the party name. Alternatively, you could have made sure to choose a color-blindness friendly palette and mark up your cells with an additional label that screen readers can read out loud. The latter feature is not currently supported by Typst, but will be added in a future release. You can check how colors look for color-blind readers with [this Chrome extension](#), [Photoshop](#), or [GIMP](#).

## How to adjust the lines in a table?

By default, Typst adds strokes between each row and column of a table. You can adjust these strokes in a variety of ways. Which one is the most practical, depends on the modification you want to make and your intent:

- Do you want to style all tables in your document, irrespective of their size and content? Use the `table` function's [`stroke`](#) argument in a set rule.
- Do you want to customize all lines in a single table? Use the `table` function's [`stroke`](#) argument when calling the `table` function.
- Do you want to change, add, or remove the stroke around a single cell? Use the `table.cell` element in the argument list of your `table` call.

- Do you want to change, add, or remove a single horizontal or vertical stroke in a single table? Use the [table.hline](#) and [table.vline](#) elements in the argument list of your table call.

We will go over all of these options with examples next! First, we will tackle the `table` function's [stroke](#) argument. Here, you can adjust both how the table's lines get drawn and configure which lines are drawn at all.

Let's start by modifying the color and thickness of the stroke:

```
#table(
 columns: 4,
 stroke: 0.5pt + rgb("666675"),
 [*Monday*], [11.5], [13.0], [4.0],
 [*Tuesday*], [8.0], [14.5], [5.0],
 [*Wednesday*], [9.0], [18.5], [13.0],
)
```

|                  |      |      |      |
|------------------|------|------|------|
| <b>Monday</b>    | 11.5 | 13.0 | 4.0  |
| <b>Tuesday</b>   | 8.0  | 14.5 | 5.0  |
| <b>Wednesday</b> | 9.0  | 18.5 | 13.0 |

This makes the table lines a bit less wide and uses a bluish gray. You can see that we added a width in point to a color to achieve our customized stroke. This addition yields a value of the [stroke type](#). Alternatively, you can use the dictionary representation for strokes which allows you to access advanced features such as dashed lines.

The previous example showed how to use the `stroke` argument in the `table` function's invocation. Alternatively, you can specify the `stroke` argument in the `table`'s `set` rule. This will have exactly the same effect on all subsequent

`table` calls as if the `stroke` argument was specified in the argument list. This is useful if you are writing a template or want to style your whole document.

```
// Renders the exact same as the last example
#set table(stroke: 0.5pt + rgb("666675"))


```

For small tables, you sometimes want to suppress all strokes because they add too much visual noise. To do this, just set the `stroke` argument to `none`:

```
#table(
 columns: 4,
 stroke: none,
 [*Monday*], [11.5], [13.0], [4.0],
 [*Tuesday*], [8.0], [14.5], [5.0],
 [*Wednesday*], [9.0], [18.5], [13.0],
)
```

|                  |      |      |      |
|------------------|------|------|------|
| <b>Monday</b>    | 11.5 | 13.0 | 4.0  |
| <b>Tuesday</b>   | 8.0  | 14.5 | 5.0  |
| <b>Wednesday</b> | 9.0  | 18.5 | 13.0 |

If you want more fine-grained control of where lines get placed in your table, you can also pass a dictionary with the keys `top`, `left`, `right`, `bottom` (controlling the respective cell sides), `x`, `y` (controlling vertical and horizontal strokes), and `rest` (covers all strokes not styled by other dictionary entries). All keys are optional; omitted keys will be treated as if

their value was the default value. For example, to get a table with only horizontal lines, you can do this:

```
#table(
 columns: 2,
 stroke: (x: none),
 align: horizon,
 [☒], [Close cabin door],
 [☐], [Start engines],
 [☐], [Radio tower],
 [☐], [Push back],
)
```

|                                     |                  |
|-------------------------------------|------------------|
| <input checked="" type="checkbox"/> | Close cabin door |
| <input type="checkbox"/>            | Start engines    |
| <input type="checkbox"/>            | Radio tower      |
| <input type="checkbox"/>            | Push back        |

This turns off all vertical strokes and leaves the horizontal strokes in place.

To achieve the reverse effect (only horizontal strokes), set the stroke argument to `(y: none)` instead.

[Further down in the guide](#), we cover how to use a function in the stroke argument to customize all strokes individually. This is how you achieve more complex stroking patterns.

## Adding individual lines in the table

If you want to add a single horizontal or vertical line in your table, for example to separate a group of rows, you can use the [`table.hline`](#) and [`table.vline`](#) elements for horizontal and vertical lines, respectively. Add them

to the argument list of the `table` function just like you would add individual cells and a header.

Let's take a look at the following example from the reference:

```
#set table.hline(stroke: 0.6pt)


```

|       |                      |
|-------|----------------------|
| 14:00 | Talk: Tracked Layout |
| 15:00 | Talk: Automations    |
| 16:00 | Workshop: Tables     |
| <hr/> |                      |
| 19:00 | Day 1 Attendee Mixer |

In this example, you can see that we have placed a call to `table.hline` between the cells, producing a horizontal line at that spot.

We also used a set rule on the element to reduce its stroke width to make it fit better with the weight of the font.

By default, Typst places horizontal and vertical lines after the current row or column, depending on their position in the argument list. You can also manually move them to a different position by adding the `y` (for `hline`) or `x` (for `vline`) argument. For example, the code below would produce the same result:

```
#set table.hline(stroke: 0.6pt)


```

Let's imagine you are working with a template that shows none of the table strokes except for one between the first and second row. Now, since you have one table that also has labels in the first column, you want to add an extra vertical line to it. However, you do not want this vertical line to cross into the top row. You can achieve this with the `start` argument:

```
// Base template already configured tables, but we need some
// extra configuration for this table.

#{

 set table(align: (x, _) => if x == 0 { left }
 else { right })
 show table.cell.where(x: 0): smallcaps
 table(
 columns: (auto, 1fr, 1fr, 1fr),
 table.vline(x: 1, start: 1),
 table.header[Trainset][Top Speed][Length][Weight],
 [TGV Réseau], [320 km/h], [200m], [383t],
 [ICE 403], [330 km/h], [201m], [409t],
 [Shinkansen N700], [300 km/h], [405m], [700t],
)
}
```

| TRAINSET        | Top Speed | Length | Weight |
|-----------------|-----------|--------|--------|
| TGV RÉSEAU      | 320 km/h  | 200m   | 383t   |
| ICE 403         | 330 km/h  | 201m   | 409t   |
| SHINKANSEN N700 | 300 km/h  | 405m   | 700t   |

In this example, we have added `table.vline` at the start of our positional argument list. But because the line is not supposed to go to the left of the first column, we specified the `x` argument as `1`. We also set the `start` argument to `1` so that the line does only start after the first row.

The example also contains two more things: We use the `align` argument with a function to right-align the data in all but the first column and use a `show` rule to make the first column of table cells appear in small capitals. Because these styles are specific to this one table, we put everything into a [code block](#), so that the styling does not affect any further tables.

## Overriding the strokes of a single cell

Imagine you want to change the stroke around a single cell. Maybe your cell is very important and needs highlighting! For this scenario, there is the [table.cell function](#). Instead of adding your content directly in the argument list of the table, you wrap it in a `table.cell` call. Now, you can use `table.cell`'s argument list to override the table properties, such as the stroke, for this cell only.

Here's an example with a matrix of two of the Big Five personality factors, with one intersection highlighted.

```
#table(
 columns: 3,
 stroke: (x: none),
 [],
 [*High Neuroticism*], [*Low Neuroticism*],
 [*High Agreeableness*],
 table.cell(stroke: orange + 2pt) [
 Sensitive \ Prone to emotional distress but very empathetic.
],
 [_Compassionate_ \ Caring and stable, often seen as a supportive figure.],
 [*Low Agreeableness*],
 [_Contentious_ \ Competitive and easily agitated.],
 [_Detached_ \ Independent and calm, may appear aloof.],
)
```

|                    | High Neuroticism                                                     | Low Neuroticism                                                               |
|--------------------|----------------------------------------------------------------------|-------------------------------------------------------------------------------|
| High Agreeableness | <i>Sensitive</i><br>Prone to emotional distress but very empathetic. | <i>Compassionate</i><br>Caring and stable, often seen as a supportive figure. |
| Low Agreeableness  | <i>Contentious</i><br>Competitive and easily agitated.               | <i>Detached</i><br>Independent and calm, may appear aloof.                    |

Above, you can see that we used the `table.cell` element in the table's argument list and passed the cell content to it. We have used its `stroke` argument to set a wider orange stroke. Despite the fact that we disabled vertical strokes on the table, the orange stroke appeared on all sides of the modified cell, showing that the table's stroke configuration is overwritten.

## Complex document-wide stroke customization

This section explains how to customize all lines at once in one or multiple tables. This allows you to draw only the first horizontal line or omit the outer

lines, without knowing how many cells the table has. This is achieved by providing a function to the table's `stroke` parameter. The function should return a stroke given the zero-indexed x and y position of the current cell. You should only need these functions if you are a template author, do not use a template, or need to heavily customize your tables. Otherwise, your template should set appropriate default table strokes.

For example, this is a set rule that draws all horizontal lines except for the very first and last line.

```
#show table.cell.where(x: 0): set text(style: "italic")
#show table.cell.where(y: 0): set text(style: "normal",
weight: "bold")
#set table(stroke: (_, y) => if y > 0 { (top: 0.8pt) })


```

| <b>Technique</b>     | <b>Advantage</b>   | <b>Drawback</b>       |
|----------------------|--------------------|-----------------------|
| <i>Diegetic</i>      | Immersive          | May be contrived      |
| <i>Extradiegetic</i> | Breaks immersion   | Obtrusive             |
| <i>Omitted</i>       | Fosters engagement | May fracture audience |

In the set rule, we pass a function that receives two arguments, assigning the vertical coordinate to `y` and discarding the horizontal coordinate. It

then returns a stroke dictionary with a `0.8pt` top stroke for all but the first line. The cells in the first line instead implicitly receive `none` as the return value. You can easily modify this function to just draw the inner vertical lines instead as `(x, _) => if x > 0 { (left: 0.8pt) }`.

Let's try a few more stroking functions. The next function will only draw a line below the first row:

```
#set table(stroke: (_, y) => if y == 0 { (bottom: 1pt) })

// Table as seen above
```

| Technique            | Advantage          | Drawback              |
|----------------------|--------------------|-----------------------|
| <i>Diegetic</i>      | Immersive          | May be contrived      |
| <i>Extradiegetic</i> | Breaks immersion   | Obtrusive             |
| <i>Omitted</i>       | Fosters engagement | May fracture audience |

If you understood the first example, it becomes obvious what happens here. We check if we are in the first row. If so, we return a bottom stroke. Otherwise, we'll return `none` implicitly.

The next example shows how to draw all but the outer lines:

```
#set table(stroke: (x, y) => (
 left: if x > 0 { 0.8pt },
 top: if y > 0 { 0.8pt },
)

// Table as seen above
```

| Technique            | Advantage          | Drawback              |
|----------------------|--------------------|-----------------------|
| <i>Diegetic</i>      | Immersive          | May be contrived      |
| <i>Extradiegetic</i> | Breaks immersion   | Obtrusive             |
| <i>Omitted</i>       | Fosters engagement | May fracture audience |

This example uses both the `x` and `y` coordinates. It omits the left stroke in the first column and the top stroke in the first row. The right and bottom lines are not drawn.

Finally, here is a table that draws all lines except for the vertical lines in the first row and horizontal lines in the table body. It looks a bit like a calendar.

```
#set table(stroke: (x, y) => (
 left: if x == 0 or y > 0 { 1pt } else { 0pt },
 right: 1pt,
 top: if y <= 1 { 1pt } else { 0pt },
 bottom: 1pt,
))

// Table as seen above
```

| Technique            | Advantage          | Drawback              |
|----------------------|--------------------|-----------------------|
| <i>Diegetic</i>      | Immersive          | May be contrived      |
| <i>Extradiegetic</i> | Breaks immersion   | Obtrusive             |
| <i>Omitted</i>       | Fosters engagement | May fracture audience |

This example is a bit more complex. We start by drawing all the strokes on the right of the cells. But this means that we have drawn strokes in the top row, too,

and we don't need those! We use the fact that `left` will override `right` and only draw the left line if we are not in the first row or if we are in the first column. In all other cases, we explicitly remove the left line. Finally, we draw the horizontal lines by first setting the bottom line and then for the first two rows with the `top` key, suppressing all other top lines. The last line appears because there is no `top` line that could suppress it.

## How to achieve a double line?

Typst does not yet have a native way to draw double strokes, but there are multiple ways to emulate them, for example with [patterns](#). We will show a different workaround in this section: Table gutters.

Tables can space their cells apart using the `gutter` argument. When a gutter is applied, a stroke is drawn on each of the now separated cells. We can selectively add gutter between the rows or columns for which we want to draw a double line. The `row-gutter` and `column-gutter` arguments allow us to do this. They accept arrays of gutter values. Let's take a look at an example:

```
#table(
 columns: 3,
 stroke: (x: none),
 row-gutter: (2.2pt, auto),
 table.header[Date] [Exercise Type] [Calories Burned],
 [2023-03-15], [Swimming], [400],
 [2023-03-17], [Weightlifting], [250],
 [2023-03-18], [Yoga], [200],
)
```

| Date       | Exercise Type | Calories Burned |
|------------|---------------|-----------------|
| 2023-03-15 | Swimming      | 400             |
| 2023-03-17 | Weightlifting | 250             |
| 2023-03-18 | Yoga          | 200             |

We can see that we used an array for `row-gutter` that specifies a `2.2pt` gap between the first and second row. It then continues with `auto` (which is the default, in this case `0pt` gutter) which will be the gutter between all other rows, since it is the last entry in the array.

## How to align the contents of the cells in my table?

You can use multiple mechanisms to align the content in your table. You can either use the `table` function's `align` argument to set the alignment for your whole table (or use it in a set rule to set the alignment for tables throughout your document) or the `align` function (or `table.cell`'s `align` argument) to override the alignment of a single cell.

When using the `table` function's `align` argument, you can choose between three methods to specify an [alignment](#):

- Just specify a single alignment like `right` (aligns in the top-right corner) or `center + horizon` (centers all cell content). This changes the alignment of all cells.
- Provide an array. Typst will cycle through this array for each column.

- Provide a function that is passed the horizontal `x` and vertical `y` coordinate of a cell and returns an alignment.

For example, this travel itinerary right-aligns the day column and left-aligns everything else by providing an array in the `align` argument:

```
#set text(font: "IBM Plex Sans")
#show table.cell.where(y: 0): set text(weight: "bold")


```

| <b>Day</b> | <b>Location</b>      | <b>Hotel or Apartment</b> | <b>Activities</b>                |
|------------|----------------------|---------------------------|----------------------------------|
| 1          | Paris, France        | Hotel de L'Europe         | Arrival, Evening River Cruise    |
| 2          | Paris, France        | Hotel de L'Europe         | Louvre Museum, Eiffel Tower      |
| 3          | Lyon, France         | Lyon City Hotel           | City Tour, Local Cuisine Tasting |
| 4          | Geneva, Switzerland  | Lakeview Inn              | Lake Geneva, Red Cross Museum    |
| 5          | Zermatt, Switzerland | Alpine Lodge              | Visit Matterhorn, Skiing         |

However, this example does not yet look perfect — the header cells should be bottom-aligned. Let's use a function instead to do so:

```
#set text(font: "IBM Plex Sans")
#show table.cell.where(y: 0): set text(weight: "bold")


```

| <b>Day</b> | <b>Location</b>      | <b>Hotel or Apartment</b> | <b>Activities</b>                |
|------------|----------------------|---------------------------|----------------------------------|
| 1          | Paris, France        | Hotel de L'Europe         | Arrival, Evening River Cruise    |
| 2          | Paris, France        | Hotel de L'Europe         | Louvre Museum, Eiffel Tower      |
| 3          | Lyon, France         | Lyon City Hotel           | City Tour, Local Cuisine Tasting |
| 4          | Geneva, Switzerland  | Lakeview Inn              | Lake Geneva, Red Cross Museum    |
| 5          | Zermatt, Switzerland | Alpine Lodge              | Visit Matterhorn, Skiing         |

In the function, we calculate a horizontal and vertical alignment based on whether we are in the first column (`x == 0`) or the first row (`y == 0`). We then make use of the fact that we can add horizontal and vertical alignments with `+` to receive a single, two-dimensional alignment.

You can find an example of using `table.cell` to change a single cell's alignment on [its reference page](#).

## How to merge cells?

When a table contains logical groupings or the same data in multiple adjacent cells, merging multiple cells into a single, larger cell can be advantageous.

Another use case for cell groups are table headers with multiple rows: That way, you can group for example a sales data table by quarter in the first row and by months in the second row.

A merged cell spans multiple rows and/or columns. You can achieve it with the [table.cell](#) function's `rowspan` and `colspan` arguments: Just specify how many rows or columns you want your cell to span.

The example below contains an attendance calendar for an office with in-person and remote days for each team member. To make the table more glanceable, we merge adjacent cells with the same value:

```
#let ofi = [Office]
#let rem = [_Remote_]
#let lea = [*On leave*]

#show table.cell.where(y: 0): set text(
 fill: white,
 weight: "bold",
)


```

```
[Lila Montgomery],
 table.cell(colspan: 5, lea),
[Nolan Pearce],
 rem,
 table.cell(colspan: 2, ofi),
 rem,
 ofi,
)
```

| Team member     | Monday | Tuesday | Wednesday | Thursday | Friday |
|-----------------|--------|---------|-----------|----------|--------|
| Evelyn Archer   |        | Office  |           | Remote   | Office |
| Lila Montgomery |        |         | On leave  |          |        |
| Nolan Pearce    | Remote |         | Office    | Remote   | Office |

In the example, we first define variables with "Office", "Remote", and "On leave" so we don't have to write these labels out every time. We can then use these variables in the table body either directly or in a `table.cell` call if the team member spends multiple consecutive days in office, remote, or on leave.

The example also contains a black header (created with `table`'s `fill` argument) with white strokes (`table`'s `stroke` argument) and white text (set by the `table.cell` set rule). Finally, we align all the content of all table cells in the body in the center. If you want to know more about the functions passed to `align`, `stroke`, and `fill`, you can check out the sections on [alignment](#), [strokes](#), and [striped tables](#).

This table would be a great candidate for fully automated generation from an external data source! Check out the [section about importing data](#) to learn more about that.

## How to rotate a table?

When tables have many columns, a portrait paper orientation can quickly get cramped. Hence, you'll sometimes want to switch your tables to landscape orientation. There are two ways to accomplish this in Typst:

- If you want to rotate only the table but not the other content of the page and the page itself, use the [rotate function](#) with the `reflow` argument set to `true`.
- If you want to rotate the whole page the table is on, you can use the [page function](#) with its `flipped` argument set to `true`. The header, footer, and page number will now also appear on the long edge of the page. This has the advantage that the table will appear right side up when read on a computer, but it also means that a page in your document has different dimensions than all the others, which can be jarring to your readers.

Below, we will demonstrate both techniques with a student grade book table.

First, we will rotate the table on the page. The example also places some text on the right of the table.

```
#set page("a5", columns: 2, numbering: "- 1 -")
#show table.cell.where(y: 0): set text(weight: "bold")

#rotate(
 -90deg,
 reflow: true,

 table(
 columns: (1fr,) + 5 * (auto,),
 inset: (x: 0.6em,),
 stroke: (_, y) => (
 x: 1pt,
 top: if y <= 1 { 1pt } else { 0pt },
 bottom: 1pt,
),
 align: (left, right, right, right, right, left),
)
)
```

```
table.header (
 [Student Name],
 [Assignment 1], [Assignment 2],
 [Mid-term], [Final Exam],
 [Total Grade],
),
[Jane Smith], [78%], [82%], [75%], [80%], [B],
[Alex Johnson], [90%], [95%], [94%], [96%], [A+],
[John Doe], [85%], [90%], [88%], [92%], [A],
[Maria Garcia], [88%], [84%], [89%], [85%], [B+],
[Zhang Wei], [93%], [89%], [90%], [91%], [A-],
[Marina Musterfrau], [96%], [91%], [74%], [69%], [B-],
),
)

#lorem(80)
```

| Student Name      | Assignment 1 | Assignment 2 | Mid-term | Final Exam | Total Grade |
|-------------------|--------------|--------------|----------|------------|-------------|
| Jane Smith        | 78%          | 82%          | 75%      | 80%        | B           |
| Alex Johnson      | 90%          | 95%          | 94%      | 96%        | A+          |
| John Doe          | 85%          | 90%          | 88%      | 92%        | A           |
| Maria Garcia      | 88%          | 84%          | 89%      | 85%        | B+          |
| Zhang Wei         | 93%          | 89%          | 90%      | 91%        | A-          |
| Marina Musterfrau | 96%          | 91%          | 74%      | 69%        | B-          |

Lorem ipsum dolor sit amet,  
 consectetur adipiscing elit, sed do  
 eiusmod tempor incididunt ut  
 labore et dolore magna aliquam  
 quaerat voluptatem. Ut enim  
 aeque doleamus animo, cum  
 corpore dolemus, fieri tamen  
 permagna accessio potest, si  
 aliquod aeternum et infinitum  
 impendere malum nobis  
 opinemur. Quod idem licet  
 transferre in voluptatem, ut  
 postea variari voluptas  
 distingue possit, augeri  
 amplificarique non possit. At  
 etiam Athenis, ut e patre  
 audiebam facete et urbane Stoicos  
 irridente, statua est in quo a nobis  
 philosophia defensa et.

— 1 —

What we have here is a two-column document on ISO A5 paper with page  
 numbers on the bottom. The table has six columns and contains a few  
 customizations to [stroke](#), alignment and spacing. But the most important

part is that the table is wrapped in a call to the `rotate` function with the `reflow` argument being `true`. This will make the table rotate 90 degrees counterclockwise. The reflow argument is needed so that the table's rotation affects the layout. If it was omitted, Typst would lay out the page as if the table was not rotated (`true` might become the default in the future).

The example also shows how to produce many columns of the same size: To the initial `1fr` column, we add an array with five `auto` items that we create by multiplying an array with one `auto` item by five. Note that arrays with just one item need a trailing comma to distinguish them from merely parenthesized expressions.

The second example shows how to rotate the whole page, so that the table stays upright:

```
#set page("a5", numbering: "- 1 -")
#show table.cell.where(y: 0): set text(weight: "bold")

#page(flipped: true) [
 #table(
 columns: (1fr,) + 5 * (auto,),
 inset: (x: 0.6em,),
 stroke: (_, y) => (
 x: 1pt,
 top: if y <= 1 { 1pt } else { 0pt },
 bottom: 1pt,
),
 align: (left, right, right, right, left),
)
 table.header(
 [Student Name],
 [Assignment 1], [Assignment 2],
 [Mid-term], [Final Exam],
)
]
```

```
[Total Grade],
) ,
[Jane Smith], [78%], [82%], [75%], [80%], [B],
[Alex Johnson], [90%], [95%], [94%], [96%], [A+],
[John Doe], [85%], [90%], [88%], [92%], [A],
[Maria Garcia], [88%], [84%], [89%], [85%], [B+],
[Zhang Wei], [93%], [89%], [90%], [91%], [A-],
[Marina Musterfrau], [96%], [91%], [74%], [69%], [B-],
)

#pad(x: 15%, top: 1.5em) [
= Winter 2023/24 results
#lorem(80)
]
]
```

| Student Name      | Assignment 1 | Assignment 2 | Mid-term | Final Exam | Total Grade |
|-------------------|--------------|--------------|----------|------------|-------------|
| Jane Smith        | 78%          | 82%          | 75%      | 80%        | B           |
| Alex Johnson      | 90%          | 95%          | 94%      | 96%        | A+          |
| John Doe          | 85%          | 90%          | 88%      | 92%        | A           |
| Maria Garcia      | 88%          | 84%          | 89%      | 85%        | B+          |
| Zhang Wei         | 93%          | 89%          | 90%      | 91%        | A-          |
| Marina Musterfrau | 96%          | 91%          | 74%      | 69%        | B-          |

### Winter 2023/24 results

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam quaerat voluptatem. Ut enim aequo doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distingue possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et.

– 1 –

Here, we take the same table and the other content we want to set with it and put it into a call to the [page](#) function while supplying `true` to the `flipped` argument. This will instruct Typst to create new pages with width and height swapped and place the contents of the function call onto a new page.

Notice how the page number is also on the long edge of the paper now. At the bottom of the page, we use the [pad](#) function to constrain the width of the paragraph to achieve a nice and legible line length.

## How to break a table across pages?

It is best to contain a table on a single page. However, some tables just have many rows, so breaking them across pages becomes unavoidable. Fortunately, Typst supports breaking tables across pages out of the box. If you are using the [table.header](#) and [table.footer](#) functions, their contents will be repeated on each page as the first and last rows, respectively. If you want to disable this behavior, you can set `repeat` to `false` on either of them.

If you have placed your table inside of a [figure](#), it becomes unable to break across pages by default. However, you can change this behavior. Let's take a look:

```
#set page(width: 9cm, height: 6cm)
#show table.cell.where(y: 0): set text(weight: "bold")
#show figure: set block(breakable: true)

#figure(
 caption: [Training regimen for Marathon],
 table(
 columns: 3,
 fill: (_, y) => if y == 0 { gray.lighten(75%) },
 table.header[Week][Distance (km)][Time (hh:mm:ss)],
 [1], [5], [00:30:00],
 [2], [7], [00:45:00],
 [3], [10], [01:00:00],
 [4], [12], [01:10:00],
 [5], [15], [01:25:00],
 [6], [18], [01:40:00],
 [7], [20], [01:50:00],
)
)
```

```
[8], [22], [02:00:00],
[...], [...], [...],
table.footer[_Goal_] [_42.195_] [_02:45:00_],
)
)
```

| Week        | Distance (km) | Time (hh:mm:ss) |
|-------------|---------------|-----------------|
| 1           | 5             | 00:30:00        |
| 2           | 7             | 00:45:00        |
| 3           | 10            | 01:00:00        |
| 4           | 12            | 01:10:00        |
| 5           | 15            | 01:25:00        |
| 6           | 18            | 01:40:00        |
| <i>Goal</i> | 42.195        | 02:45:00        |

| Week        | Distance (km) | Time (hh:mm:ss) |
|-------------|---------------|-----------------|
| 7           | 20            | 01:50:00        |
| 8           | 22            | 02:00:00        |
| ...         | ...           | ...             |
| <i>Goal</i> | 42.195        | 02:45:00        |

Table 1: Training regimen for Marathon

A figure automatically produces a [block](#) which cannot break by default.

However, we can reconfigure the block of the figure using a show rule to make it `breakable`. Now, the figure spans multiple pages with the headers and footers repeating.

## How to import data into a table?

Often, you need to put data that you obtained elsewhere into a table. Sometimes, this is from Microsoft Excel or Google Sheets, sometimes it is from a dataset on the web or from your experiment. Fortunately, Typst can load many [common file formats](#), so you can use scripting to include their data in a table.

The most common file format for tabular data is CSV. You can obtain a CSV file from Excel by choosing "Save as" in the *File* menu and choosing the file format "CSV UTF-8 (Comma-delimited) (.csv)". Save the file and, if you are using the web app, upload it to your project.

In our case, we will be building a table about Moore's Law. For this purpose, we are using a statistic with [how many transistors the average microprocessor consists of per year from Our World in Data](#). Let's start by pressing the "Download" button to get a CSV file with the raw data.

Be sure to move the file to your project or somewhere Typst can see it, if you are using the CLI. Once you did that, we can open the file to see how it is structured:

```
Entity,Code,Year,Transistors per microprocessor
World,OWID_WRL,1971,2308.2417
World,OWID_WRL,1972,3554.5222
World,OWID_WRL,1974,6097.5625
```

The file starts with a header and contains four columns: Entity (which is to whom the metric applies), Code, the year, and the number of transistors per microprocessor. Only the last two columns change between each row, so we can disregard "Entity" and "Code".

First, let's start by loading this file with the [csv](#) function. It accepts the file name of the file we want to load as a string argument:

```
#let moore = csv("moore.csv")
```

We have loaded our file (assuming we named it `moore.csv`) and [bound](#) [it](#) to the new variable `moore`. This will not produce any output, so there's nothing to see yet. If we want to examine what Typst loaded, we can either hover the name of the variable in the web app or print some items from the array:

```
#let moore = csv("moore.csv")

#moore.slice(0, 3)

(
(
 "Entity",
 "Code",
 "Year",
 "Transistors per microprocessor",
),
 ("World", "OWID_WRL", "1995",
"9646616"),
 ("World", "OWID_WRL", "1996",
"9646616"),
)
```

With the arguments `(0, 3)`, the [slice](#) method returns the first three items in the array (with the indices 0, 1, and 2). We can see that each row is its own array with one item per cell.

Now, let's write a loop that will transform this data into an array of cells that we can use with the `table` function.

```
#let moore = csv("moore.csv")


```

| Year | Transistors per microprocessor |
|------|--------------------------------|
| 1995 | 9646616                        |
| 1996 | 9646616                        |
| 1997 | 9646616                        |
| 1998 | 15261378                       |
| 1999 | 21673922                       |
| 2000 | 37180264                       |
| 2001 | 42550656                       |
| 2002 | 220673400                      |
| 2003 | 220673400                      |
| 2004 | 273842000                      |
| 2005 | 305052770                      |
| 2006 | 582941600                      |
| 2007 | 805842200                      |
| 2008 | 805842200                      |
| 2009 | 2308241400                     |
| 2010 | 2308241400                     |
| 2011 | 2600000000                     |
| 2012 | 2600000000                     |
| 2013 | 5000000000                     |
| 2014 | 5700000000                     |
| 2016 | 8000000000                     |
| 2017 | 19200000000                    |
| 2018 | 21100000000                    |
| 2019 | 39500000000                    |
| 2020 | 39500000000                    |
| 2021 | 58200000000                    |

The example above uses a for loop that iterates over the rows in our CSV file and returns an array for each iteration. We use the for loop's

[destructuring](#) capability to discard all but the last two items of each row.

We then create a new array with just these two. Because Typst will concatenate the array results of all the loop iterations, we get a one-dimensional array in which the year column and the number of transistors alternate. We can then insert the array as cells. For this we use the [spread operator](#) (...). By prefixing an array, or, in our case an expression that yields an array, with two dots, we tell Typst that the array's items should be used as positional arguments.

Alternatively, we can also use the [map](#), [slice](#), and [flatten](#) array methods to write this in a more functional style:

```
#let moore = csv("moore.csv")

#table(
 columns: moore.first().len(),
 ..moore.map(m => m.slice(2)).flatten(),
)
```

This example renders the same as the previous one, but first uses the `map` function to change each row of the data. We pass a function to `map` that gets run on each row of the CSV and returns a new value to replace that row with. We use it to discard the first two columns with `slice`. Then, we spread the data into the `table` function. However, we need to pass a one-dimensional array and `moore`'s value is two-dimensional (that means that each of its row values contains an array with the cell data). That's why we call `flatten` which converts it to a one-dimensional array. We also extract the number of columns from the data itself.

Now that we have nice code for our table, we should try to also make the table itself nice! The transistor counts go from millions in 1995 to trillions in 2021 and changes are difficult to see with so many digits. We could try to present our data logarithmically to make it more digestible:

```
#let moore = csv("moore.csv")
#let moore-log = moore.slice(1).map(m => {
 let (... , year, count) = m
 let log = calc.log(float(count))
 let rounded = str(calc.round(log, digits: 2))
 (year, rounded)
})

#show table.cell.where(x: 0): strong


```

| Year | Transistor count ( $\log_{10}$ ) |
|------|----------------------------------|
| 1995 | 6.98                             |
| 1996 | 6.98                             |
| 1997 | 6.98                             |
| 1998 | 7.18                             |
| 1999 | 7.34                             |
| 2000 | 7.57                             |
| 2001 | 7.63                             |
| 2002 | 8.34                             |
| 2003 | 8.34                             |
| 2004 | 8.44                             |
| 2005 | 8.48                             |
| 2006 | 8.77                             |
| 2007 | 8.91                             |
| 2008 | 8.91                             |
| 2009 | 9.36                             |
| 2010 | 9.36                             |
| 2011 | 9.41                             |
| 2012 | 9.41                             |
| 2013 | 9.7                              |
| 2014 | 9.76                             |
| 2016 | 9.9                              |
| 2017 | 10.28                            |
| 2018 | 10.32                            |
| 2019 | 10.6                             |
| 2020 | 10.6                             |
| 2021 | 10.76                            |

In this example, we first drop the header row from the data since we are adding our own. Then, we discard all but the last two columns as above. We do this by [destructuring](#) the array `m`, discarding all but the two last items. We then convert the string in `count` to a floating point number, calculate

its logarithm and store it in the variable `log`. Finally, we round it to two digits, convert it to a string, and store it in the variable `rounded`. Then, we return an array with `year` and `rounded` that replaces the original row. In our table, we have added our custom header that tells the reader that we've applied a logarithm to the values. Then, we spread the flattened data as above.

We also styled the table with [stripes](#), a [horizontal line](#) below the first row, [aligned](#) everything to the right, and emboldened the first column. Click on the links to go to the relevant guide sections and see how it's done!

## What if I need the `table` function for something that isn't a table?

Tabular layouts of content can be useful not only for matrices of closely related data, like shown in the examples throughout this guide, but also for presentational purposes. Typst differentiates between grids that are for layout and presentational purposes only and tables, in which the arrangement of the cells itself conveys information.

To make this difference clear to other software and allow templates to heavily style tables, Typst has two functions for grid and table layout:

- The [table](#) function explained throughout this guide which is intended for tabular data.
- The [grid](#) function which is intended for presentational purposes and page layout.

Both elements work the same way and have the same arguments. You can apply everything you have learned about tables in this guide to grids. There are only three differences:

- You'll need to use the `grid.cell`, `grid.vline`, and `grid.hline` elements instead of `table.cell`, `table.vline`, and `table.hline`.
- The grid has different defaults: It draws no strokes by default and has no spacing (inset) inside of its cells.
- Elements like `figure` do not react to grids since they are supposed to have no semantical bearing on the document structure.

## **5. Changelog**

Learn what has changed in the latest Typst releases and move your documents forward. This section documents all changes to Typst since its initial public release.

### **Versions**

- [Typst 0.12.0](#)
- [Typst 0.11.1](#)
- [Typst 0.11.0](#)
- [Typst 0.10.0](#)
- [Typst 0.9.0](#)
- [Typst 0.8.0](#)
- [Typst 0.7.0](#)

- [Typst 0.6.0](#)

- [Typst 0.5.0](#)

- [Typst 0.4.0](#)

- [Typst 0.3.0](#)

- [Typst 0.2.0](#)

- [Typst 0.1.0](#)

- [Earlier](#)

## 5.1 0.12.0

### Highlights

- Added support for multi-column floating [placement](#) and [figures](#)
- Added support for automatic [line numbering](#) (often used in academic papers)
- Typst's layout engine is now multithreaded. Typical speedups are 2-3x for larger documents. The multithreading operates on page break boundaries, so explicit page breaks are necessary for it to kick in.
- Paragraph justification was optimized with a new two-pass algorithm. Speedups are larger for shorter paragraphs and go up to 6x.
- Highly reduced PDF file sizes due to better font subsetting (thanks to [@LaurenzV](#))
- Emoji are now exported properly in PDF
- Added initial support for PDF/A. For now, only the PDF/A-2b profile is supported, but more is planned for the future.

- Added various options for configuring the CLI's environment (fonts, package paths, etc.)
- Text show rules now match across multiple text elements
- Block-level equations can now optionally break over multiple pages
- Fixed a bug where some fonts would not print correctly on professional printers
- Fixed a long-standing bug which could cause headings to be orphaned at the bottom of the page

## Layout

- Added support for multi-column floating placement and figures via `place.scope` and `figure.scope`. Two-column documents should now prefer `set page(columns: 2) over show: column.with(2)` (see the [page setup guide](#)).
- Added support for automatic [line numbering](#) (often used in academic papers)
- Added [par.spacing](#) property for configuring paragraph spacing. This should now be used instead of `show par: set block(spacing: ...)` (**Breaking change**)
- Block-level elements like lists, grids, and stacks now show themselves as blocks and are thus affected by all block properties (e.g. `stroke`) rather than just spacing (**Breaking change**)
- Added [block.sticky](#) property which prevents a page break after a block

- Added [`place.flush`](#) function which forces all floating figures to be placed before any further content
- Added [`skew`](#) function
- Added `auto` option for [`page.header`](#) and [`page.footer`](#) which results in an automatic header/footer based on the numbering (which was previously inaccessible after a change)
- Added `gap` and `justify` parameters to [`repeat`](#) function
- Added `width` and `height` parameters to the [`measure`](#) function to define the space in which the content should be measured. Especially useful in combination with [`layout`](#).
- The height of a `block`, `image`, `rect`, `square`, `ellipse`, or `circle` can now be specified in [`fractional units`](#)
- The [`scale`](#) function now supports absolute lengths for `x`, `y`, `factor`. This way an element of unknown size can be scaled to a fixed size.
- The values of `block.above` and `block.below` can now be retrieved in context expressions.
- Increased accuracy of conversions between absolute units (pt, mm, cm, in)
- Fixed a bug which could cause headings to be orphaned at the bottom of the page
- Fixed footnotes within breakable blocks appearing on the page where the breakable block ends instead of at the page where the footnote marker is
- Fixed numbering of nested footnotes and footnotes in floats

- Fixed empty pages appearing when a [context](#) expression wraps whole pages
- Fixed `set block` (`spacing: x`) behaving differently from `set block` (`above: x, below: x`)
- Fixed behavior of [rotate](#) and [scale](#) with `reflow: true`
- Fixed interaction of [align](#) (`horizon`) and [v\(1fr\)](#)
- Fixed various bugs where floating placement would yield overlapping results
- Fixed a bug where widow/orphan prevention would unnecessarily move text into the next column
- Fixed [weak spacing](#) not being trimmed at the start and end of lines in a paragraph (only at the start and end of paragraphs)
- Fixed interaction of weak page break and [pagebreak.to](#)
- Fixed compilation output of a single weak page break
- Fixed crash when [padding](#) by `100%`

## Text

- Tuned hyphenation: It is less eager by default and hyphenations close to the edges of words are now discouraged more strongly (**May lead to larger layout reflows**)
- New default font: Libertinus Serif. This is the maintained successor to the old default font Linux Libertine. (**May lead to smaller reflows**)
- Setting the font to an unavailable family will now result in a warning
- Implemented a new smart quote algorithm, fixing various bugs where smart quotes weren't all that smart

- Added `text.costs` parameter for tweaking various parameters that affect the choices of the layout engine during text layout
- Added `typm` highlighting mode for math in `raw blocks`
- Added basic i18n for Galician, Catalan, Latin, Icelandic, Hebrew
- Implemented hyphenation duplication for Czech, Croatian, Lower Sorbian, Polish, Portuguese, Slovak, and Spanish.
- The `smallcaps` function is now an element function and can thereby be used in show(-set) rules.
- The `raw.theme` parameter can now be set to `none` to disable highlighting even in the presence of a language tag, and to `auto` to reset to the default
- Multiple `stylistic sets` can now be enabled at once
- Fixed the Chinese translation for "Equation"
- Fixed that hyphenation could occur outside of words
- Fixed incorrect layout of bidirectional text in edge cases
- Fixed layout of paragraphs with explicit trailing whitespace
- Fixed bugs related to empty paragraphs created via `#""`
- Fixed accidental trailing spaces for line breaks immediately preceding an inline equation
- Fixed `text.historical-ligatures` not working correctly
- Fixed accidental repetition of Thai characters around line breaks in some circumstances
- Fixed `smart quotes` for Swiss French

- New font metadata exceptions for Archivo, Kaiti SC, and Kaiti TC

- Updated bundled New Computer Modern fonts to version 6.0

## Math

- Block-level equations can now break over multiple pages if enabled via

```
show math.equation: set block(breakable: true).
```

- Matrix and vector sizing is now more consistent across different cell contents

- Added [stretch](#) function for manually or automatically stretching characters

like arrows or parentheses horizontally or vertically

- Improved layout of attachments on parenthesized as well as under- or overlined expressions

- Improved layout of nested attachments resulting from code like `#let a0 = $a_0$; $a0^1$`

- Improved layout of primes close to superscripts

- Improved layout of fractions

- Typst now makes use of math-specific height-dependent kerning information in some fonts for better attachment layout

- The `floor` and `ceil` functions in math are now callable symbols, such that

```
$ floor(x) = lr(floor.l x floor.r) $
```

- The [mat.delim](#), [vec.delim](#), and [cases.delim](#) parameters now allow any

character that is considered a delimiter or "fence" (e.g. `|`) by Unicode. The

`delim: "||"` notation is *not* supported anymore and should be replaced by

`delim: bar.double` **(Minor breaking change)**

- Added [vec.align](#) and [mat.align](#) parameters

- Added [underparen](#), [overparen](#), [undershell](#), and [overshell](#)
- Added ~ shorthand for `tilde.op` in math mode (**Minor breaking change**)
- Fixed baseline alignment of equation numbers
- Fixed positioning of corner brackets (⟨, ⟩, ⟲, ⟳)
- Fixed baseline of large roots
- Fixed multiple minor layout bugs with attachments
- Fixed that alignment points could affect line height in math
- Fixed that spaces could show up between text and invisible elements like [metadata](#) in math
- Fixed a crash with recursive show rules in math
- Fixed [lr.size](#) not affecting characters enclosed in [mid](#) in some cases
- Fixed resolving of em units in sub- and superscripts
- Fixed bounding box of inline equations when a [text edge](#) is set to "bounds"

## Introspection

- Implemented a new system by which Typst tracks where elements end up on the pages. This may lead to subtly different behavior in introspections.

### (Breaking change)

- Fixed various bugs with wrong counter behavior in complex layout situations, through a new, more principled implementation
- Counter updates can now be before the first, in between, and after the last page when isolated by weak page breaks. This allows, for instance, updating a counter before the first page header and background.

- Fixed logical ordering of introspections within footnotes and figures
- Fixed incorrect `here().position()` when `place` was used in a context expression
- Fixed resolved positions of elements (in particular, headings) whose show rule emits an invisible element (like a state update) before a page break
- Fixed behavior of stepping a counter at a deeper level than its current state has
- Fixed citation formatting not working in table headers and a few other places
- Displaying the footnote counter will now respect the footnote numbering style

## Model

- Document set rules do not need to be at the very start of the document anymore. The only restriction is that they must not occur inside of layout containers.
- The `spacing` property of [lists](#), [enumerations](#), and [term lists](#) is now also respected for tight lists
- Tight lists now only attach (with tighter spacing) to preceding paragraphs, not arbitrary blocks
- The [quote](#) element is now locatable (can be used in queries)
- The bibliography heading now uses `depth` instead of `level` so that its level can still be configured via a show-set rule
- Added support for more [numbering](#) formats: Devanagari, Eastern Arabic, Bengali, and circled numbers

- Added `hanging-indent` parameter to heading function to tweak the appearance of multi-line headings and improved default appearance of multi-line headings
- Improved handling of bidirectional text in outline entry
- Fixed document set rules being ignored in an otherwise empty document
- Fixed document set rules not being usable in context expressions
- Fixed bad interaction between `set` document and `set` page
- Fixed `show figure: set align(..)`. Since the default figure alignment is now a show-set rule, it is not revoked by `show figure: it => it.body` anymore. (**Minor breaking change**)

- Fixed numbering of footnote references
- Fixed spacing after bibliography heading

## Bibliography

- The Hayagriva YAML `publisher` field can now accept a dictionary with a `location` key. The top-level `location` key is now primarily intended for event and item locations.
- Multiple page ranges with prefixes and suffixes are now allowed
- Added `director` and catch-all editor types to BibLaTeX parsing
- Added support for disambiguation to alphanumeric citation style
- The year 0 will now render as 1BC
- Fixes for sorting of bibliography entries
- Fixed pluralization of page range labels

- Fixed sorting of citations by their number
- Fixed how citation number ranges collapse
- Fixed when the short form of a title is used
- Fixed parsing of unbalanced dollars in BibLaTeX `url` field
- Updated built-in citation styles

## Visualization

- Added `fill-rule` parameter to [path](#) and [polygon](#) functions
- Fixed color mixing and gradients for [Luma colors](#)
- Fixed conversion from Luma to CMYK colors
- Fixed offset gradient strokes in PNG export
- Fixed unintended cropping of some SVGs
- SVGs with foreign objects now produce a warning as they will likely not render correctly in Typst

## Syntax

- Added support for nested imports like `import "file.typ": module.item`
- Added support for parenthesized imports like `import "file.typ": (a, b, c)`. With those, the import list can break over multiple lines.
- Fixed edge case in parsing of reference syntax
- Fixed edge case in parsing of heading, list, enum, and term markers immediately followed by comments
- Fixed rare crash in parsing of parenthesized expressions

## Scripting

- Added new fixed-point [decimal](#) number type for highly precise arithmetic on numbers in base 10, as needed for finance
- Added `std` module for accessing standard library definitions even when a variable with the same name shadows/overwrites it
- Added [array.to-dict](#), [array.reduce](#), [array.windows](#) methods
- Added `exact` argument to [array.zip](#)
- Added [arguments.at](#) method
- Added [int.from-bytes](#), [int.to-bytes](#), [float.from-bytes](#), and [float.to-bytes](#)
- Added proper support for negative values of the `digits` parameter of [calc.round](#) (the behaviour existed before but was subtly broken)
- Conversions from [int](#) to [float](#) will now error instead of saturating if the float is too large (**Minor breaking change**)
- Added `float.nan` and `float.inf`, removed `calc.nan` (**Minor breaking change**)
- Certain symbols are now generally callable like functions and not only specifically in math. Examples are accents or [floor](#) and [ceil](#).
- Improved [repr](#) of relative values, sequences, infinities, NaN, `type(None)` and `type(auto)`
- Fixed crash on whole packages (rather than just files) cyclically importing each other
- Fixed return type of [calc.round](#) on integers when a non-zero value is provided for `digits`

## Styling

- Text show rules now match across multiple text elements
- The string " in a text show rule now matches smart quotes
- Fixed a long-standing styling bug where the header and footer would incorrectly inherit styles from a lone element on the page (e.g. a heading)
- Fixed `set page` not working directly after a counter/state update
- Page fields configured via an explicit `page(...)[...]` call can now be properly retrieved in context expressions

## Export

- Highly reduced PDF file sizes due to better font subsetting
- Emoji are now exported properly in PDF
- Added initial support for PDF/A. For now, only the standard PDF/A-2b is supported, but more is planned for the future. Enabled via `--pdf-standard a-2b` in the CLI and via the UI in File > Export as > PDF in the web app.
- Setting `page.fill` to `none` will now lead to transparent pages instead of white ones in PNG and SVG. The new default of `auto` means transparent for PDF and white for PNG and SVG.
- Improved text copy-paste from PDF in complex scenarios
- Exported SVGs now contain the `data-typst-label` attribute on groups resulting from labelled [boxes](#) and [blocks](#)
- Fixed a bug where some fonts would not print correctly on professional printers

- Fixed a bug where transparency could leak from one PDF object to another
- Fixed a bug with CMYK gradients in PDF
- Fixed various bugs with export of Oklab gradients in PDF
- Fixed crashes related to rendering of non-outline glyphs
- Two small fixes for PDF standard conformance

## Performance

- Typst's layout engine is now multithreaded. Typical speedups are 2-3x for larger documents. The multithreading operates on page break boundaries, so explicit page breaks are necessary for it to kick in.
- Paragraph justification was optimized with a new two-pass algorithm. Speedups are larger for shorter paragraphs and range from 1-6x.

## Command Line Interface

- Added `--pages` option to select specific page ranges to export
- Added `--package-path` and `--package-cache-path` as well as `TYPST_PACKAGE_PATH` and `TYPST_PACKAGE_CACHE_PATH` environment variables for configuring where packages are loaded from and cached in, respectively
- Added `--ignore-system-fonts` flag to disable system fonts fully for better reproducibility
- Added `--make-deps` argument for outputting the dependencies of the current compilation as a Makefile

- Added `--pretty` option to `typst query`, with the default now being to minify (only applies to JSON format)
- Added `--backup-path` to `typst update` to configure where the previous version is backed up
- Added useful links to help output
- The CLI will now greet users who invoke just `typst` for the first time
- The document can now be written to stdout by passing `-` as the output filename (for PDF or single-page image export)
- Typst will now emit a proper error message instead of failing silently when the certificate specified by `--cert` or `TYPST_CERT` could not be loaded
- The CLI now respects the `SOURCE_DATE_EPOCH` environment variable for better reproducibility
- When exporting multiple images, you can now use `t` (total pages), `p` (current page), and `0p` (zero-padded current page, same as current `n`) in the output path
- The input and output paths now allow non-UTF-8 values
- Times are now formatted more consistently across the CLI
- Fixed a bug related to the `--open` flag
- Fixed path completions for `typst` not working in zsh

## Tooling and Diagnostics

- The "compiler" field for specifying the minimum Typst version required by a package now supports imprecise bounds like `0.11` instead of `0.11.0`

- Added warning when a label is ignored by Typst because no preceding labellable element exists
- Added hint when trying to apply labels in code mode
- Added hint when trying to call a standard library function that has been shadowed/overwritten by a local definition
- Added hint when trying to set both the language and the region in the `lang` parameter
- Added hints when trying to compile non-Typst files (e.g. after having typed `typst c file.pdf` by accident)
- Added hint when a string is used where a label is expected
- Added hint when a stray end of a block comment (`*/`) is encountered
- Added hints when destructuring arrays with the wrong number of elements
- Improved error message when trying to use a keyword as an identifier in a `let` binding
- Improved error messages when accessing nonexistent fields
- Improved error message when a package exists, but not the specified version
- Improved hints for unknown variables
- Improved hint when trying to convert a length with non-zero em component to an absolute unit
- Fixed a crash that could be triggered by certain hover tooltips
- Fixed an off-by-one error in to-source jumps when first-line-indent is enabled
- Fixed suggestions for `.` after the end of an inline code expressions

- Fixed autocompletions being duplicated in a specific case

## Symbols

- **New:** parallelogram, original, image, crossmark, rest, natural, flat, sharp, tiny, miny, copyleft, trademark, emoji.beet, emoji.fingerprint, emoji.harp, emoji.shovel, emoji.splatter, emoji.tree.leafless,

- **New variants:** club.stroked, diamond.stroked, heart.stroked, spade.stroked, gt.neq, lt.neq, checkmark.heavy, paren.double, brace.double, shell.double, arrow.turn, plus.double, plus.triple, infinity.bar, infinity.incomplete, infinity.tie, multimap.double, ballot.check, ballot.check.heavy, emptyset.bar, emptyset.circle, emptyset.arrow.l, emptyset.arrow.r, parallel.struck, parallel.eq, parallel.equiv, parallel.slanted, parallel.tilde, angle.l.curly, angle.l.dot, angle.r.curly, angle.r.dot, angle.oblique, angle.s, em.two, em.three

- **Renamed:** turtle to shell, notes to note, ballot.x to ballot.cross, succ.eq to succ.curly.eq, prec.eq to prec.curly.eq, servicemark to trademark.service, emoji.face.tired to emoji.face.distress (**Breaking change**)

- **Changed codepoint:** prec.eq, prec.neq, succ.eq, succ.neq, triangle from ▷ to Δ, emoji.face.tired (**Breaking change**)
- **Removed:** lt.curly in favor of prec, gt.curly in favor of succ (**Breaking change**)

## Deprecations

- `counter.display` without an established context
- `counter.final` with a location
- `state.final` with a location
- `state.display`
- `query` with a location as the second argument
- `locate` with a callback function
- `measure` with styles
- `style`

## Development

- Added `typst-kit` crate which provides useful APIs for `World` implementors
- Added go-to-definition API in `typst-ide`
- Added package manifest parsing APIs to `typst-syntax`
- As the compiler is now capable of multithreading, `World` implementations must satisfy `Send` and `Sync`
- Changed signature of `World::main` to allow for the scenario where the main file could not be loaded
- Removed `Tracer` in favor of `Warned<T>` and `typst::trace` function
- The `xz2` dependency used by the self-updater is now statically linked
- The Dockerfile now has an `ENTRYPOINT` directive

## Contributors

Thanks to everyone who contributed to this release!

## 5.2 0.11.1

### Security

- Fixed a vulnerability where image files at known paths could be embedded into the PDF even if they were outside of the project directory

### Bibliography

- Fixed et-al handling in subsequent citations
- Fixed suppression of title for citations and bibliography references with no author
- Fixed handling of initials in citation styles without a delimiter
- Fixed bug with citations in footnotes

### Text and Layout

- Fixed interaction of `first-line-indent` and `outline`
- Fixed compression of CJK punctuation marks at line start and end
- Fixed handling of `rectangles` with negative dimensions
- Fixed layout of `path` in explicitly sized container
- Fixed broken `raw` text in right-to-left paragraphs
- Fixed tab rendering in `raw` text with language `typ` or `typc`
- Fixed highlighting of multi-line `raw` text enclosed by single backticks
- Fixed indentation of overflowing lines in `raw` blocks
- Fixed extra space when `raw` text ends with a backtick

### Math

- Fixed broken `equations` in right-to-left paragraphs

- Fixed missing [blackboard bold](#) letters
- Fixed error on empty arguments in 2D math argument list
- Fixed stretching via [mid](#) for various characters
- Fixed that alignment points in equations were affected by `set align(...)`

## Export

- Fixed [smart quotes](#) in PDF outline
- Fixed [patterns](#) with spacing in PDF
- Fixed wrong PDF page labels when [page numbering](#) was disabled after being previously enabled

## Scripting

- Fixed overflow for large numbers in external data files (by converting to floats instead)
- Fixed [str.trim\(regex, at: end\)](#) when the whole string is matched

## Miscellaneous

- Fixed deformed strokes for specific shapes and thicknesses
- Fixed newline handling in code mode: There can now be comments within chained method calls and between an `if` branch and the `else` keyword
- Fixed inefficiency with incremental reparsing
- Fixed autocompletions for relative file imports
- Fixed crash in autocompletion handler
- Fixed a bug where the path and entrypoint printed by `typst init` were not properly escaped

- Fixed various documentation errors

## Contributors

Thanks to everyone who contributed to this release!

# 5.3 0.11.0

## Tables

- Tables are now *much* more flexible, read the new [table guide](#) to get started
- Added `table.cell` element for per-cell configuration
- Cells can now span multiple [columns](#) or [rows](#)
- The [stroke](#) of individual cells can now be customized
- The [align](#) and [inset](#) arguments of the table function now also take `(x, y)`  
=> .. functions
- Added `table.hline` and `table.vline` for convenient line customization
- Added `table.header` element for table headers that repeat on every page
- Added `table.footer` element for table footers that repeat on every page
- All the new table functionality is also available for [grids](#)
  - Fixed gutter-related bugs

Thanks to [@PgBiel](#) for his work on tables!

## Templates

- You can now use template packages to get started with new projects. Click [Start from template](#) on the web app's dashboard and choose your preferred template or run the `typst init <template>` command in the CLI. You can [browse the available templates here](#).

- Switching templates after the fact has become easier. You can just import a styling function from a different template package.
- Package authors can now submit their own templates to the [package repository](#). Share a template for a paper, your institution, or an original work to help the community get a head start on their projects.
- Templates and packages are now organized by category and discipline. Filter packages by either taxonomy in the *Start from template* wizard. If you are a package author, take a look at the new documentation for [categories](#) and [disciplines](#).

## Context

- Added *context expressions*: Read the chapter on [context](#) to get started
- With context, you can access settable properties, e.g. `context text.lang` to access the language set via `set text(lang: "...")`
- The following existing functions have been made contextual: [query](#), [locate](#), [measure](#), [counter.display](#), [counter.at](#), [counter.final](#), [state.at](#), and [state.final](#)
- Added contextual methods [counter.get](#) and [state.get](#) to retrieve the value of a counter or state in the current context
- Added contextual function [here](#) to retrieve the [location](#) of the current context
- The [locate](#) function now returns the location of a selector's unique match. Its old behavior has been replaced by context expressions and only remains temporarily available for compatibility.

- The `counter.at` and `state.at` methods are now more flexible: They directly accept any kind of `locatable` selector with a unique match (e.g. a label) instead of just locations
- When context is available, `counter.display` now directly returns the result of applying the numbering instead of yielding opaque content. It should not be used anymore without context. (Deprecation planned)
- The `state.display` function should not be used anymore, use `state.get` instead (Deprecation planned)
- The `location` argument of `query`, `counter.final`, and `state.final` should not be used anymore (Deprecation planned)
- The `styles` argument of the `measure` function should not be used anymore (Deprecation planned)
- The `style` function should not be used anymore, use context instead (Deprecation planned)
- The correct context is now also provided in various other places where it is available, e.g. in show rules, layout callbacks, and numbering functions in the outline

## Styling

- Fixed priority of multiple `show-set rules`: They now apply in the same order as normal set rules would
- Show-set rules on the same element (e.g. `show heading.where(level: 1): set heading(numbering: "1.")`) now work properly

- Setting properties on an element within a transformational show rule (e.g.

`show heading: it => { set heading(..); it })` is **not** supported anymore (previously it also only worked sometimes); use show-set rules instead

### (Breaking change)

- Text show rules that match their own output now work properly (e.g.

`show "cmd": `cmd``

- The elements passed to show rules and returned by queries now contain all fields of their respective element functions rather than just specific ones
- All settable properties can now be used in [where](#) selectors
- [And](#) and [or](#) selectors can now be used with show rules
- Errors within show rules and context expressions are now ignored in all but the last introspection iteration, in line with the behavior of the old [locate](#)
- Fixed a bug where document set rules were allowed after content

## Layout

- Added `reflow` argument to [rotate](#) and [scale](#) which lets them affect the layout
- Fixed a bug where [floating placement](#) or [floating figures](#) could end up out of order
- Fixed overlap of text and figure for full-page floating figures
- Fixed various cases where the [hide](#) function didn't hide its contents properly
- Fixed usage of [h](#) and [v](#) in [stacks](#)

- Invisible content like a counter update will no longer force a visible block for just itself
- Fixed a bug with horizontal spacing followed by invisible content (like a counter update) directly at the start of a paragraph

## Text

- Added [stroke](#) property for text
- Added basic i18n for Serbian and Catalan
- Added support for contemporary Japanese [numbering](#) method
- Added patches for various wrong metadata in specific fonts
- The [text direction](#) can now be overridden within a paragraph
- Fixed Danish [smart quotes](#)
- Fixed font fallback next to a line break
- Fixed width adjustment of JIS-style Japanese punctuation
- Fixed Finnish translation of "Listing"
- Fixed Z-ordering of multiple text decorations (underlines, etc.)
- Fixed a bug due to which text [features](#) could not be overridden in consecutive set rules

## Model

- Added [depth](#) and [offset](#) arguments to heading to increase or decrease the heading level for a bunch of content; the heading syntax now sets `depth` rather than `level` (**Breaking change**)
- List [markers](#) now cycle by default

- The [quote](#) function now more robustly selects the correct quotes based on language and nesting
- Fixed indent bugs related to the default show rule of [terms](#)

## Math

- Inline equations now automatically linebreak at appropriate places
- Added [number-align](#) argument to equations
- Added support for adjusting the [size](#) of accents relative to their base
- Improved positioning of accents
- [Primes](#) are now always attached as [scripts](#) by default
- Exposed [math.primes](#) element which backs the `$f'` syntax in math
- Math mode is not affected by [strong](#) and [emph](#) anymore
- Fixed [attach](#) under [fractions](#)
- Fixed that [math.class](#) did not affect smart limit placement
- Fixed weak spacing in [lr](#) groups
- Fixed layout of large operators for Cambria Math font
- Fixed math styling of Hebrew symbol codepoints

## Symbols

- Added [gradient](#) as an alias for nabla
- Added [partial](#) as an alias for diff, diff will be deprecated in the future
- Added colon.double, gt.approx, gt.napprox, lt.approx, and lt.napprox
- Added arrow.r.tilde and arrow.l.tilde
- Added tilde.dot

- Added `forces` and `forces.not`
- Added `space.nobreak.narrow`
- Added `lrm` (Left-to-Right Mark) and `rlm` (Right-to-Left Mark)
- Fixed `star.stroked` symbol (which previously had the wrong codepoint)

## Scripting

- Arrays can now be compared lexicographically
- Added contextual method [`to-absolute`](#) to lengths
- Added [`calc.root`](#)
- Added [`int.signum`](#) and [`float.signum`](#) methods
- Added [`float.is-nan`](#) and [`float.is-infinite`](#) methods
- Added [`int.bit-not`](#), [`int.bit-and`](#), [`int.bit-or`](#), [`int.bit-xor`](#), [`int.bit-lshift`](#), and [`int.bit-rshift`](#) methods
- Added [`array.chunks`](#) method
- A module can now be converted to a dictionary with the [`dictionary`](#) [`constructor`](#) to access its contents dynamically
- Added [`row-type`](#) argument to `csv` function to configure how rows will be represented
- [`XML parsing`](#) now allows DTDs (document type definitions)
- Improved formatting of negative numbers with [`str`](#) and [`repr`](#)
- For loops can now iterate over [`bytes`](#)
- Fixed a bug with pattern matching in for loops
- Fixed a bug with labels not being part of [`.fields\(\)`](#) dictionaries

- Fixed a bug where unnamed argument sinks wouldn't capture excess arguments
- Fixed typo in `repr` output of strokes

## Syntax

- Added support for nested [destructuring patterns](#)
- Special spaces (like thin or non-breaking spaces) are now parsed literally instead of being collapsed into normal spaces (**Breaking change**)
- Korean text can now use emphasis syntax without adding spaces (**Breaking change**)
- The token `context` is now a keyword and cannot be used as an identifier anymore (**Breaking change**)
- Nested line comments aren't allowed anymore in block comments (**Breaking change**)
- Fixed a bug where `x..` would be treated as a field access
- Text elements can now span across curly braces in markup
- Fixed silently wrong parsing when function name is parenthesized
- Fixed various bugs with parsing of destructuring patterns, arrays, and dictionaries

## Tooling & Diagnostics

- Click-to-jump now works properly within [raw](#) text
- Added suggestion for accessing a field if a method doesn't exist
- Improved hint for calling a function stored in a dictionary

- Improved errors for mutable accessor functions on arrays and dictionaries
- Fixed error message when calling constructor of type that doesn't have one
- Fixed confusing error message with nested dictionaries for strokes on different sides
- Fixed autocompletion for multiple packages with the same name from different namespaces

## Visualization

- The `image` function doesn't upscale images beyond their natural size anymore
- The `image` function now respects rotation stored in EXIF metadata
- Added support for SVG filters
- Added alpha component to `luma` colors
- Added `color.transparentize` and `color.opacify` methods
- Improved `color.negate` function
- Added `stroke` and `radius` arguments to `highlight` function
- Changed default `highlight` color to be transparent
- CMYK to RGB conversion is now color-managed
- Fixed crash with gradients in Oklch color space
- Fixed color-mixing for hue-based spaces
- Fixed bugs with color conversion
- SVG sizes are not rounded anymore, preventing slightly wrong aspect ratios
- Fixed a few other SVG-related bugs
- `color.components` doesn't round anything anymore

## Export

- PDFs now contain named destinations for headings derived from their labels
- The internal PDF structure was changed to make it easier for external tools to extract or modify individual pages, avoiding a bug with Typst PDFs in Apple Preview
- PDFs produced by Typst should now be byte-by-byte reproducible when `set document(date: none)` is set
- Added missing flag to PDF annotation
- Fixed multiple bugs with gradients in PDF export
- Fixed a bug with patterns in PDF export
- Fixed a bug with embedding of grayscale images in PDF export
- Fixed a bug with To-Unicode mapping of CFF fonts in PDF export
- Fixed a bug with the generation of the PDF outline
- Fixed a sorting bug in PDF export leading to non-reproducible output
- Fixed a bug with transparent text in PNG export
- Exported SVG files now include units in their top-level `width` and `height`

## Command line interface

- Added support for passing [inputs](#) via a CLI flag
- When passing the filename `-`, Typst will now read input from `stdin`
- Now uses the system-native TLS implementation for network fetching which should be generally more robust
- Watch mode will now properly detect when a previously missing file is created

- Added `--color` flag to configure whether to print colored output
- Fixed user agent with which packages are downloaded
- Updated bundled fonts to the newest versions

## Development

- Added `--vendor-openssl` to CLI to configure whether to link OpenSSL statically instead of dynamically (not applicable to Windows and Apple platforms)
- Removed old tracing (and its verbosity) flag from the CLI
- Added new `--timings` flag which supersedes the old flamegraph profiling in the CLI
- Added minimal CLI to `typst-docs` crate for extracting the language and standard library documentation as JSON
- The `typst_pdf::export` function's `ident` argument switched from `Option` to `Smart`. It should only be set to `Smart::Custom` if you can provide a stable identifier (like the web app can). The CLI sets `Smart::Auto`.

## Contributors

Thanks to everyone who contributed to this release!

## 5.4.10.0

### Bibliography management

- Added support for citation collapsing (e.g. [1]–[3] instead of [1], [2], [3]) if requested by a CSL style
- Fixed bug where an additional space would appear after a group of citations

- Fixed link show rules for links in the bibliography
- Fixed show-set rules on citations
- Fixed bibliography-related crashes that happened on some systems
- Corrected name of the GB/T 7714 family of styles from 7114 to 7714
  - Fixed missing title in some bibliography styles
  - Fixed printing of volumes in some styles
  - Fixed delimiter order for contributors in some styles (e.g. APA)
  - Fixed behavior of alphanumeric style
  - Fixed multiple bugs with GB/T 7714 style
  - Fixed escaping in Hayagriva values
  - Fixed crashes with empty dates in Hayagriva files
  - Fixed bug with spacing around math blocks
  - Fixed title case formatting after verbatim text and apostrophes
  - Page ranges in `.bib` files can now be arbitrary strings
  - Multi-line values in `.bib` files are now parsed correctly
  - Entry keys in `.bib` files now allow more characters
  - Fixed error message for empty dates in `.bib` files
  - Added support for years of lengths other than 4 without leading zeros in `.bib` files
  - More LaTeX commands (e.g. for quotes) are now respected in `.bib` files

## Visualization

- Added support for [patterns](#) as fills and strokes

- The `alpha` parameter of the [components](#) function on colors is now a named parameter (**Breaking change**)

- Added support for the [Oklch](#) color space
- Improved conversions between colors in different color spaces
- Removed restrictions on [Oklab](#) chroma component
- Fixed [clipping](#) on blocks and boxes without a stroke
- Fixed bug with [gradients](#) on math
- Fixed bug with gradient rotation on text
- Fixed bug with gradient colors in PDF
- Fixed relative base of Oklab chroma ratios
- Fixed Oklab color negation

## Text and Layout

- CJK text can now be emphasized with the `*` and `_` syntax even when there are no spaces
- Added basic i18n for Greek and Estonian
- Improved default [figure caption separator](#) for Chinese, French, and Russian
- Changed default [figure supplement](#) for Russian to short form
- Fixed [CJK-Latin-spacing](#) before line breaks and in [locate](#) calls
- Fixed line breaking at the end of links

## Math

- Added [mid](#) function for scaling a delimiter up to the height of the surrounding [lr](#) group

- The [op](#) function can now take any content, not just strings
- Improved documentation for [math alignment](#)
- Fixed swallowing of trailing comma when a symbol is used in a function-like way (e.g. `pi(a,b,)`)

## Scripting

- Any non-identifier dictionary key is now interpreted as an expression: For instance, `((key) : value)` will create a dictionary with a dynamic key
- The [stroke](#) type now has a constructor that converts a value to a stroke or creates one from its parts
- Added constructor for [arguments](#) type
- Added [calc.div-euclid](#) and [calc.rem-euclid](#) functions
- Fixed equality of [arguments](#)
- Fixed [repr](#) of [cmyk](#) colors
- Fixed crashes with provided elements like figure captions, outline entries, and footnote entries

## Tooling and Diagnostics

- Show rules that match on their own output now produce an appropriate error message instead of a crash (this is a first step, in the future they will just work)
- Too highly or infinitely nested layouts now produce error messages instead of crashes
- Added hints for invalid identifiers

- Added hint when trying to use a manually constructed footnote or outline entry
- Added missing details to autocompletions for types
- Improved error message when passing a named argument where a positional one is expected
- Jump from click now works on raw blocks

## Export

- PDF compilation output is now again fully byte-by-byte reproducible if the document's [date](#) is set manually
- Fixed color export in SVG
- Fixed PDF metadata encoding of multiple [authors](#)

## Command line interface

- Fixed a major bug where `typst watch` would confuse files and fail to pick up updates
- Fetching of the release metadata in `typst update` now respects proxies
- Fixed bug with `--open` flag on Windows when the path contains a space
- The `TYPST_FONT_PATHS` environment variable can now contain multiple paths (separated by ; on Windows and : elsewhere)
- Updated embedded New Computer Modern fonts to version 4.7
- The watching process doesn't stop anymore when the main file contains invalid UTF-8

## Miscellaneous Improvements

- Parallelized image encoding in PDF export
- Improved the internal representation of content for improved performance
- Optimized introspection (query, counter, etc.) performance
- The [document title](#) can now be arbitrary content instead of just a string
- The [number-align](#) parameter on numbered lists now also accepts vertical alignments
- Fixed selectors on [quote](#) elements
- Fixed parsing of `#return` expression in markup
- Fixed bug where inline equations were displayed in equation outlines
- Fixed potential CRLF issue in [raw](#) blocks
- Fixed a bug where Chinese numbering couldn't exceed the number 255

## Development

- Merged `typst` and `typst-library` and extracted `typst-pdf`, `typst-svg`, and `typst-render` into separate crates
- The Nix flake now includes the git revision when running `typst --version`

## Contributors

Thanks to everyone who contributed to this release!

## 5.5.0.9.0

## Bibliography management

- New bibliography engine based on [CSL](#) (Citation Style Language). Ships with about 100 commonly used citation styles and can load custom `.csl` files.

- Added new `form` argument to the `cite` function to produce different forms of citations (e.g. for producing a citation suitable for inclusion in prose)
- The `cite` function now takes only a single label/key instead of allowing multiple. Adjacent citations are merged and formatted according to the citation style's rules automatically. This works both with the reference syntax and explicit calls to the `cite` function. (**Breaking change**)
- The `cite` function now takes a `label` instead of a string (**Breaking change**)
- Added `full` argument to bibliography function to print the full bibliography even if not all works were cited
- Bibliography entries can now contain Typst equations (wrapped in `$..$` just like in markup), this works both for `.yml` and `.bib` bibliographies
- The hayagriva YAML format was improved. See its [changelog](#) for more details.

### (**Breaking change**)

- A few bugs with `.bib` file parsing were fixed
- Removed `brackets` argument of `cite` function in favor of `form`

## Visualization

- Gradients and colors (thanks to [@Dherse](#))
  - Added support for [gradients](#) on shapes and text
  - Supports linear, radial, and conic gradients
  - Added support for defining colors in more color spaces, including [Oklab](#), [Linear RGB\(A\)](#), [HSL](#), and [HSV](#)
  - Added [saturate](#), [desaturate](#), and [rotate](#) functions on colors

- Added [color.map](#) module with predefined color maps that can be used with gradients
- Rename `kind` function on colors to [space](#)
- Removed `to-rgba`, `to-cmyk`, and `to-luma` functions in favor of a new [components](#) function
  - Improved rendering of [rectangles](#) with corner radius and varying stroke widths
  - Added support for properly clipping [boxes](#) and [blocks](#) with a border radius
  - Added background parameter to [overline](#), [underline](#), and [strike](#) functions
  - Fixed inaccurate color embedding in PDFs
  - Fixed ICC profile handling for images embedded in PDFs

## Text and Layout

- Added support for automatically adding proper [spacing](#) between CJK and Latin text (enabled by default)
- Added support for automatic adjustment of more CJK punctuation
- Added [quote](#) element for inserting inline and block quotes with optional attributions
- Added [raw.line](#) element for customizing the display of individual lines of raw text, e.g. to add line numbers while keeping proper syntax highlighting
- Added support for per-side [inset](#) customization to `table` function
- Added Hungarian and Romanian translations
- Added support for Czech hyphenation

- Added support for setting custom [smart quotes](#)
- The default [figure separator](#) now reacts to the currently set language and region
- Improved line breaking of links / URLs (especially helpful for bibliographies with many URLs)
- Improved handling of consecutive hyphens in justification algorithm
- Fixed interaction of justification and hanging indent
- Fixed a bug with line breaking of short lines without spaces when justification is enabled
- Fixed font fallback for hyphen generated by hyphenation
- Fixed handling of word joiner and other no-break characters during hyphenation
- Fixed crash when hyphenating after an empty line
- Fixed line breaking of composite emoji like ??
- Fixed missing text in some SVGs
- Fixed font fallback in SVGs
- Fixed behavior of [to](#) argument on [pagebreak](#) function
- Fixed `set align(..)` for equations
- Fixed spacing around [placed](#) elements
- Fixed coalescing of [above](#) and [below](#) spacing if given in em units and the font sizes differ

- Fixed handling of `extent` parameter of [underline](#), [overline](#), and [strike](#) functions
- Fixed crash for [floating placed elements](#) with no specified vertical alignment
- Partially fixed a bug with citations in footnotes

## Math

- Added `gap` argument for [vec](#), [mat](#), and [cases](#) function
- Added `size` argument for [abs](#), [norm](#), [floor](#), [ceil](#), and [round](#) functions
- Added [reverse](#) parameter to [cases](#) function
- Added support for multinomial coefficients to [binom](#) function
- Removed `rotation` argument on [cancel](#) function in favor of a new and more flexible `angle` argument (**Breaking change**)
- Added `wide` constant, which inserts twice the spacing of `quad`
- Added `csch` and `sech` [operators](#)
- $\leftarrow$ ,  $\rightarrow$ ,  $\leftrightarrow$ , and  $\leftrightarrow$  can now be used as [accents](#)
- Added `integral.dash`, `integral.dash.double`, and `integral.slash` [symbols](#)
- Added support for specifying negative indices for [augmentation](#) lines to position the line from the back
- Fixed default color of matrix [augmentation](#) lines
- Fixed attachment of primes to inline expressions
- Math content now respects the text [baseline](#) setting

## Performance

- Fixed a bug related to show rules in templates which would effectively disable incremental compilation in affected documents
- Micro-optimized code in several hot paths, which brings substantial performance gains, in particular in incremental compilations
- Improved incremental parsing, which affects the whole incremental compilation pipeline
- Added support for incremental parsing in the CLI
- Added support for incremental SVG encoding during PDF export, which greatly improves export performance for documents with many SVG

## Tooling and Diagnostics

- Improved autocompletion for variables that are in-scope
- Added autocompletion for package imports
- Added autocompletion for [labels](#)
- Added tooltip that shows which variables a function captures (when hovering over the equals sign or arrow of the function)
- Diagnostics are now deduplicated
- Improved diagnostics when trying to apply unary + or - to types that only support binary + and -
- Error messages now state which label or citation key isn't present in the document or its bibliography

- Fixed a bug where function argument parsing errors were shadowed by function execution errors (e.g. when trying to call `array.sorted` and passing the key function as a positional argument instead of a named one).

## Export

- Added support for configuring the document's creation `date`. If the `date` is set to `auto` (the default), the PDF's creation date will be set to the current date and time.
- Added support for configuring document `keywords`
- Generated PDFs now contain PDF document IDs
- The PDF creator tool metadata now includes the Typst version

## Web app

- Added version picker to pin a project to an older compiler version (with support for Typst 0.6.0+)
- Fixed desyncs between editor and compiler and improved overall stability
- The app now continues to highlight the document when typing while the document is being compiled

## Command line interface

- Added support for discovering fonts through fontconfig
- Now clears the screen instead of resetting the terminal
- Now automatically picks correct file extension for selected output format
- Now only regenerates images for changed pages when using `typst` watch with PNG or SVG export

## Miscellaneous Improvements

- Added [version](#) type and `sys.version` constant specifying the current compiler version. Can be used to gracefully support multiple versions.
- The U+2212 MINUS SIGN is now used when displaying a numeric value, in the [repr](#) of any numeric value and to replace a normal hyphen in text mode when before a digit. This improves, in particular, how negative integer values are displayed in math mode.
- Added support for specifying a default value instead of failing for `remove` function in [array](#) and [dictionary](#)
- Simplified page setup guide examples
- Switched the documentation from using the word "hashtag" to the word "hash" where appropriate
- Added support for [array.zip](#) without any further arguments
- Fixed crash when a plugin tried to read out of bounds memory
- Fixed crashes when handling infinite [lengths](#)
- Fixed introspection (mostly bibliography) bugs due to weak page break close to the end of the document

## Development

- Extracted `typst::ide` into separate `typst_ide` crate
- Removed a few remaining '`static`' bounds on `&dyn World`
- Removed unnecessary dependency, which reduces the binary size
- Fixed compilation of `typst` by itself (without `typst-library`)

- Fixed warnings with Nix flake when using `lib.getExe`

## Contributors

Thanks to everyone who contributed to this release!

# 5.6 0.8.0

## Scripting

- Plugins (thanks to [@astrale-sharp](#) and [@arnaudgolfouse](#))
  - Typst can now load [plugins](#) that are compiled to WebAssembly
  - Anything that can be compiled to WebAssembly can thus be loaded as a plugin
  - These plugins are fully encapsulated (no access to file system or network)
  - Plugins can be shipped as part of [packages](#)
  - Plugins work just the same in the web app
- Types are now first-class values (**Breaking change**)
  - A [type](#) is now itself a value
  - Some types can be called like functions (those that have a constructor), e.g. [int](#) and [str](#)
    - Type checks are now of the form `type(10) == int` instead of the old `type(10) == "integer"`. [Compatibility](#) with the old way will remain for a while to give package authors time to upgrade, but it will be removed at some point.
    - Methods are now syntax sugar for calling a function scoped to a type, meaning that `"hello".len()` is equivalent to `str.len("hello")`

- Added support for [import](#) renaming with `as`
- Added a [duration](#) type
- Added support for [CBOR](#) encoding and decoding
- Added encoding and decoding functions from and to bytes for data formats:

[json.decode](#), [json.encode](#), and similar functions for other formats

- Added [array.intersperse](#) function
- Added [str.rev](#) function
- Added `calc.tau` constant
- Made [bytes](#) joinable and addable
- Made [array.zip](#) function variadic
- Fixed bug with [eval](#) when the mode was set to "math"
- Fixed bug with [ends-with](#) function on strings
- Fixed bug with destructuring in combination with break, continue, and return
- Fixed argument types of [hyperbolic functions](#), they don't allow angles anymore

### (Breaking change)

- Renamed some color methods: `rgba` becomes `to-rgba`, `cmyk` becomes `to-cmyk`, and `luma` becomes `to-luma` (**Breaking change**)

## Export

- Added SVG export (thanks to [@Enter-tainer](#))
- Fixed bugs with PDF font embedding
- Added support for page labels that reflect the [page numbering](#) style in the PDF

## Text and Layout

- Added [highlight](#) function for highlighting text with a background color
- Added [polygon.regular](#) function for drawing a regular polygon
- Added support for tabs in [raw](#) elements alongside [tab-width](#) parameter
- The layout engine now tries to prevent "runt" (final lines consisting of just a single word)
- Added Finnish translations
- Added hyphenation support for Polish
- Improved handling of consecutive smart quotes of different kinds
- Fixed vertical alignments for [number-align](#) argument on page function

### (Breaking change)

- Fixed weak pagebreaks after counter updates
- Fixed missing text in SVG when the text font is set to "New Computer Modern"
- Fixed translations for Chinese
- Fixed crash for empty text in show rule
- Fixed leading spaces when there's a linebreak after a number and a comma
- Fixed placement of floating elements in columns and other containers
- Fixed sizing of block containing just a single box

## Math

- Added support for [augmented matrices](#)

- Removed support for automatic matching of fences like `|` and `||` as there were too many false positives. You can use functions like [abs](#) or [norm](#) or an explicit [lr](#) call instead. (**Breaking change**)

- Fixed spacing after number with decimal point in math
- Fixed bug with primes in subscript
- Fixed weak spacing
- Fixed crash when text within math contains a newline

## Tooling and Diagnostics

- Added hints when trying to call a function stored in a dictionary without extra parentheses
- Fixed hint when referencing an equation without numbering
- Added more details to some diagnostics (e.g. when SVG decoding fails)

## Command line interface

- Added `typst update` command for self-updating the CLI (thanks to [@jimvdl](#))
- Added download progress indicator for packages and updates
- Added `--format` argument to explicitly specify the output format
- The CLI now respects proxy configuration through environment variables and has a new `--cert` option for setting a custom CA certificate
- Fixed crash when field wasn't present and `--one` is passed to `typst query`

## Miscellaneous Improvements

- Added [page setup guide](#)

- Added [`figure.caption`](#) function that can be used for simpler figure customization (**Breaking change** because `it.caption` now renders the full caption with supplement in figure show rules and manual outlines)
- Moved `caption-pos` argument to `figure.caption` function and renamed it to `position` (**Breaking change**)
- Added [`separator`](#) argument to `figure.caption` function
- Added support for combination of and/or and before/after [`selectors`](#)
- Packages can now specify a [`minimum compiler version`](#) they require to work
- Fixed parser bug where method calls could be moved onto their own line for `#let` expressions in markup (**Breaking change**)
- Fixed bugs in sentence and title case conversion for bibliographies
- Fixed supplements for alphanumeric and author-title bibliography styles
- Fixed off-by-one error in APA bibliography style

## Development

- Made `Span` and `FileId` more type-safe so that all error conditions must be handled by `World` implementors

## Contributors

Thanks to everyone who contributed to this release!

## 5.7 0.7.0

### Text and Layout

- Added support for floating figures through the [`placement`](#) argument on the `figure` function

- Added support for arbitrary floating content through the [float](#) argument on the place function
- Added support for loading `.sublime-syntax` files as highlighting [syntaxes](#) for raw blocks
- Added support for loading `.tmTheme` files as highlighting [themes](#) for raw blocks
- Added *bounds* option to [top-edge](#) and [bottom-edge](#) arguments of text function for tight bounding boxes
- Removed nonsensical top- and bottom-edge options, e.g. *ascender* for the bottom edge (**Breaking change**)
- Added [script](#) argument to text function
- Added [alternative](#) argument to smart quote function
- Added basic i18n for Japanese
- Added hyphenation support for `nb` and `nn` language codes in addition to `no`
- Fixed positioning of [placed elements](#) in containers
- Fixed overflowing containers due to optimized line breaks

## Export

- Greatly improved export of SVG images to PDF. Many thanks to [@LaurenzV](#) for their work on this.
- Added support for the alpha channel of RGBA colors in PDF export
- Fixed a bug with PPI (pixels per inch) for PNG export

## Math

- Improved layout of primes (e.g. in `$a'₁$`)
- Improved display of multi-primes (e.g. in `$a''$`)
- Improved layout of [roots](#)
- Changed relations to show attachments as [limits](#) by default (e.g. in `$a ->^x b$`)
- Large operators and delimiters are now always vertically centered
- [Boxes](#) in equations now sit on the baseline instead of being vertically centered by default. Notably, this does not affect [blocks](#) because they are not inline elements.
- Added support for [weak spacing](#)
- Added support for OpenType character variants
- Added support for customizing the [math class](#) of content
- Fixed spacing around `.`, `\!/`, and `\dots`
- Fixed spacing between closing delimiters and large operators
- Fixed a bug with math font weight selection
- Symbols and Operators (**Breaking changes**)
  - Added `id`, `im`, and `tr` text [operators](#)
  - Renamed `ident` to `equiv` with alias `eq.triple` and removed `ident.strict` in favor of `eq.quad`
  - Renamed `ast.sq` to `ast.square` and `integral.sq` to `integral.square`
  - Renamed `.eqq` modifier to `.equiv` (and `.neqq` to `.nequiv`) for `tilde`, `gt`, `lt`, `prec`, and `succ`

- Added `emptyset` as alias for `nothing`
- Added `lt.curly` and `gt.curly` as aliases for `prec` and `succ`
- Added `aleph`, `beth`, and `gimmel` as alias for `alef`, `bet`, and `gimel`

## Scripting

- Fields
  - Added `abs` and `em` field to [lengths](#)
  - Added `ratio` and `length` field to [relative lengths](#)
  - Added `x` and `y` field to [2d alignments](#)
  - Added `paint`, `thickness`, `cap`, `join`, `dash`, and `miter-limit` field to [strokes](#)
- Accessor and utility methods
  - Added [dedup](#) method to arrays
  - Added `pt`, `mm`, `cm`, and `inches` method to [lengths](#)
  - Added `deg` and `rad` method to [angles](#)
  - Added `kind`, `hex`, `rgba`, `cmyk`, and `luma` method to [colors](#)
  - Added `axis`, `start`, `end`, and `inv` method to [directions](#)
  - Added `axis` and `inv` method to [alignments](#)
  - Added `inv` method to [2d alignments](#)
  - Added `start` argument to [enumerate](#) method on arrays
- Added [color.mix](#) function
- Added `mode` and `scope` arguments to [eval](#) function
- Added [bytes](#) type for holding large byte buffers

- Added [encoding](#) argument to read function to read a file as bytes instead of a string
- Added [image.decode](#) function for decoding an image directly from a string or bytes
- Added [bytes](#) function for converting a string or an array of integers to bytes
- Added [array](#) function for converting bytes to an array of integers
- Added support for converting bytes to a string with the [str](#) function

## Tooling and Diagnostics

- Added support for compiler warnings
- Added warning when compilation does not converge within five attempts due to intense use of introspection features
- Added warnings for empty emphasis (`_` and `**`)
- Improved error message for invalid field assignments
- Improved error message after single `#`
- Improved error message when a keyword is used where an identifier is expected
- Fixed parameter autocompletion for functions that are in modules
- Import autocompletion now only shows the latest package version until a colon is typed
- Fixed autocompletion for dictionary key containing a space
- Fixed autocompletion for `for` loops

## Command line interface

- Added `typst query` subcommand to execute a [query](#) on the command line
- The `--root` and `--font-paths` arguments cannot appear in front of the command anymore (**Breaking change**)
- Local and cached packages are now stored in directories of the form `{namespace}/{name}/{version}` instead of `{namespace}/{name}-{version}` (**Breaking change**)
- Now prioritizes explicitly given fonts (via `--font-paths`) over system and embedded fonts when both exist
- Fixed `typst watch` not working with some text editors
- Fixed displayed compilation time (now includes export)

## Miscellaneous Improvements

- Added [bookmarked](#) argument to heading to control whether a heading becomes part of the PDF outline
- Added [caption-pos](#) argument to control the position of a figure's caption
- Added [metadata](#) function for exposing an arbitrary value to the introspection system
- Fixed that a [state](#) was identified by the pair `(key, init)` instead of just its key
- Improved indent logic of [enumerations](#). Instead of requiring at least as much indent as the end of the marker, they now require only one more space indent than the start of the marker. As a result, even long markers like `12.` work with just 2 spaces of indent.
- Fixed bug with indent logic of [raw](#) blocks

- Fixed a parsing bug with dictionaries

## Development

- Extracted parser and syntax tree into `typst-syntax` crate
- The `World::today` implementation of Typst dependents may need fixing if they have the same [bug](#) that the CLI world had

## Contributors

Thanks to everyone who contributed to this release!

# 5.8 0.6.0

## Package Management

- Typst now has built-in [package management](#)
- You can import [published](#) community packages or create and use [system-local](#) ones
- Published packages are also supported in the web app

## Math

- Added support for optical size variants of glyphs in math mode
- Added argument to enable [limits](#) conditionally depending on whether the equation is set in [display](#) or [inline](#) style
- Added `gt.eq.slant` and `lt.eq.slant` symbols
- Increased precedence of factorials in math mode (`$1/n!$` works correctly now)
- Improved [underlines](#) and [overlines](#) in math mode
- Fixed usage of [limits](#) function in show rules
- Fixed bugs with line breaks in equations

## Text and Layout

- Added support for alternating page [margins](#) with the `inside` and `outside` keys
- Added support for specifying the page [binding](#)
- Added [to](#) argument to `pagebreak` function to skip to the next even or odd page
- Added basic i18n for a few more languages (TR, SQ, TL)
- Fixed bug with missing table row at page break
- Fixed bug with [underlines](#)
- Fixed bug superfluous table lines
- Fixed smart quotes after line breaks
- Fixed a crash related to text layout

## Command line interface

- **Breaking change:** Added requirement for `--root/TYPST_ROOT` directory to contain the input file because it designates the *project* root. Existing setups that use `TYPST_ROOT` to emulate package management should switch to [local packages](#)
- **Breaking change:** Now denies file access outside of the project root
- Added support for local packages and on-demand package download
- Now watches all relevant files, within the root and all packages
- Now displays compilation time

## Miscellaneous Improvements

- Added [outline.entry](#) to customize outline entries with show rules
- Added some hints for error messages
- Added some missing syntaxes for [raw](#) highlighting
- Improved rendering of rotated images in PNG export and web app
- Made [footnotes](#) reusable and referenceable
- Fixed bug with citations and bibliographies in [locate](#)
- Fixed inconsistent tense in documentation

## Development

- Added [contribution guide](#)
- Reworked `worlD` interface to accommodate for package management and make it a bit simpler to implement (*Breaking change for implementors*)

## Contributors

Thanks to everyone who contributed to this release!

## 5.9.0.5.0

### Text and Layout

- Added [raw](#) syntax highlighting for many more languages
- Added support for Korean [numbering](#)
- Added basic i18n for a few more languages (NL, SV, DA)
- Improved line breaking for East Asian languages
- Expanded functionality of outline [indent](#) property
- Fixed footnotes in columns
- Fixed page breaking bugs with [footnotes](#)

- Fixed bug with handling of footnotes in lists, tables, and figures
- Fixed a bug with CJK punctuation adjustment
- Fixed a crash with rounded rectangles
- Fixed alignment of [line](#) elements

## Math

• **Breaking change:** The syntax rules for mathematical [attachments](#) were improved: `$f^abs (3)$` now parses as `$f^(abs (3))$` instead of `$ (f^abs) (3)$`.

To disambiguate, add a space: `$f^zeta (3)$`.

- Added [forced size](#) commands for math (e.g., [display](#))
- Added [supplement](#) parameter to [equation](#), used by [references](#)
- New [symbols](#): bullet, xor, slash.big, sigma.alt, tack.r.not, tack.r.short, tack.r.double.not
- Fixed a bug with symbols in matrices
- Fixed a crash in the [attach](#) function

## Scripting

- Added new [datetime](#) type and [datetime.today](#) to retrieve the current date
- Added [str.from-unicode](#) and [str.to-unicode](#) functions
- Added [fields](#) method on content
- Added `base` parameter to [str](#) function
- Added [calc.exp](#) and [calc.ln](#)
- Improved accuracy of [calc.pow](#) and [calc.log](#) for specific bases
- Fixed [removal](#) order for dictionary

- Fixed .at(default: ...) for [strings](#) and [content](#)
- Fixed field access on styled elements
- Removed deprecated `calc.mod` function

## Command line interface

- Added PNG export via `typst compile source.typ output-{n}.png`. The output path must contain `{n}` if the document has multiple pages.
- Added `--diagnostic-format=short` for Unix-style short diagnostics
- Doesn't emit color codes anymore if stderr isn't a TTY
- Now sets the correct exit when invoked with a nonexistent file
- Now ignores UTF-8 BOM in Typst files

## Miscellaneous Improvements

- Improved errors for mismatched delimiters
- Improved error message for failed length comparisons
- Fixed a bug with images not showing up in Apple Preview
- Fixed multiple bugs with the PDF outline
- Fixed citations and other searchable elements in [hide](#)
- Fixed bugs with [reference supplements](#)
- Fixed Nix flake

## Contributors

Thanks to everyone who contributed to this release!

## 5.10 0.4.0

## Footnotes

- Implemented support for footnotes
  - The [footnote](#) function inserts a footnote
  - The [footnote.entry](#) function can be used to customize the footnote listing
  - The "chicago-notes" [citation style](#) is now available

## Documentation

- Added a [Guide for LaTeX users](#)
- Now shows default values for optional arguments
- Added richer outlines in "On this Page"
- Added initial support for search keywords: "Table of Contents" will now find the [outline](#) function. Suggestions for more keywords are welcome!
- Fixed issue with search result ranking
- Fixed many more small issues

## Math

- **Breaking change:** Alignment points (&) in equations now alternate between left and right alignment
- Added support for writing roots with Unicode: For example, `$root (x+y) $` can now also be written as `$\sqrt (x+y) $`
- Fixed uneven vertical [attachment](#) alignment
- Fixed spacing on decorated elements (e.g., spacing around a [canceled](#) operator)
- Fixed styling for stretchable symbols
- Added `tack.r.double`, `tack.l.double`, `dotless.i` and `dotless.j` [symbols](#)

- Fixed show rules on symbols (e.g. `show sym.tack: set text(blue)`)

- Fixed missing rename from `ast.op` to `ast` that should have been in the previous release

## Scripting

- Added function scopes: A function can now hold related definitions in its own scope, similar to a module. The new `assert.eq` function, for instance, is part of the `assert` function's scope. Note that function scopes are currently only available for built-in functions.

- Added `assert.eq` and `assert.ne` functions for simpler equality and inequality assertions with more helpful error messages

- Exposed `list`, `enum`, and `term list` items in their respective functions' scope
- The `at` methods on `strings`, `arrays`, `dictionaries`, and `content` now support specifying a default value

- Added support for passing a function to `replace` that is called with each match.

- Fixed `replacement` strings: They are now inserted completely verbatim instead of supporting the previous (unintended) magic dollar syntax for capture groups

- Fixed bug with trailing placeholders in destructuring patterns
- Fixed bug with underscore in parameter destructuring
- Fixed crash with nested patterns and when hovering over an invalid pattern
- Better error messages when casting to an `integer` or `float` fails

## Text and Layout

- Implemented sophisticated CJK punctuation adjustment
- Disabled [overhang](#) for CJK punctuation
- Added basic translations for Traditional Chinese
- Fixed [alignment](#) of text inside raw blocks (centering a raw block, e.g. through a figure, will now keep the text itself left-aligned)
- Added support for passing an array instead of a function to configure table cell [alignment](#) and [fill](#) per column
- Fixed automatic figure [kind](#) detection
- Made alignment of [enum numbers](#) configurable, defaulting to `end`
- Figures can now be made breakable with a show-set rule for blocks in figure
- Initial fix for smart quotes in RTL languages

## Export

- Fixed ligatures in PDF export: They are now copyable and searchable
- Exported PDFs now embed ICC profiles for images that have them
- Fixed export of strokes with zero thickness

## Web app

- Projects can now contain folders
- Added upload by drag-and-drop into the file panel
- Files from the file panel can now be dragged into the editor to insert them into a Typst file
- You can now copy-paste images and other files from your computer directly into the editor

- Added a button to resend confirmation email
- Added an option to invert preview colors in dark mode
- Added tips to the loading screen and the Help menu. Feel free to propose more!
- Added syntax highlighting for YAML files
- Allowed middle mouse button click on many buttons to navigate into a new tab
- Allowed more project names
- Fixed overridden Vim mode keybindings
- Fixed many bugs regarding file upload and more

## Miscellaneous Improvements

- Improved performance of counters, state, and queries
- Improved incremental parsing for more efficient recompilations
- Added support for `.yaml` extension in addition to `.yml` for bibliographies
- The CLI now emits escape codes only if the output is a TTY
- For users of the `typst` crate: The `Document` is now `Sync` again and the `World` doesn't have to be `'static` anymore

## Contributors

Thanks to everyone who contributed to this release!

## 5.11.0.3.0

## Breaking changes

- Renamed a few symbols: What was previous `dot.op` is now just `dot` and the basic dot is `dot.basic`. The same applies to `ast` and `tilde`.
- Renamed `mod` to [`rem`](#) to more accurately reflect the behavior. It will remain available as `mod` until the next update as a grace period.
- A lone underscore is not a valid identifier anymore, it can now only be used in patterns
- Removed `before` and `after` arguments from [`query`](#). This is now handled through flexible [`selectors`](#) combinator methods
- Added support for [`attachments`](#) (sub-, superscripts) that precede the base symbol. The `top` and `bottom` arguments have been renamed to `t` and `b`.

## New features

- Added support for more complex [`strokes`](#) (configurable caps, joins, and dash patterns)
- Added [`cancel`](#) function for equations
- Added support for [`destructuring`](#) in argument lists and assignments
- Added [`alt`](#) text argument to `image` function
- Added [`toml`](#) function for loading data from a TOML file
- Added [`zip`](#), [`sum`](#), and [`product`](#) methods for arrays
- Added `fact`, `perm`, `binom`, `gcd`, `lcm`, `atan2`, `quo`, `trunc`, and `fract` [`calculation`](#) functions

## Improvements

- Text in SVGs now displays properly

- Typst now generates a PDF heading outline
- [References](#) now provides the referenced element as a field in show rules
- Refined linebreak algorithm for better Chinese justification
- Locations are now a valid kind of selector
- Added a few symbols for algebra
- Added Spanish smart quote support
- Added [selector](#) function to turn a selector-like value into a selector on which combinator methods can be called
- Improved some error messages
- The outline and bibliography headings can now be styled with show-set rules
- Operations on numbers now produce an error instead of overflowing

## Bug fixes

- Fixed wrong linebreak before punctuation that follows inline equations, citations, and other elements
- Fixed a bug with [argument sinks](#)
- Fixed strokes with thickness zero
- Fixed hiding and show rules in math
- Fixed alignment in matrices
- Fixed some alignment bugs in equations
- Fixed grid cell alignment
- Fixed alignment of list marker and enum markers in presence of global alignment settings

- Fixed [path](#) closing
- Fixed compiler crash with figure references
- A single trailing line break is now ignored in math, just like in text

## Command line interface

- Font path and compilation root can now be set with the environment variables

`TYPST_FONT_PATHS` and `TYPST_ROOT`

- The output of `typst fonts` now includes the embedded fonts

## Development

- Added instrumentation for debugging and optimization
- Added `--update` flag and `UPDATE_EXPECT` environment variable to update reference images for tests
- You can now run a specific subtest with `--subtest`
- Tests now run on multiple threads

## Contributors

Thanks to everyone who contributed to this release!

## 5.12.0.2.0

### Breaking changes

- Removed support for iterating over index and value in [for loops](#). This is now handled via unpacking and enumerating. Same goes for the [map](#) method.
- [Dictionaries](#) now iterate in insertion order instead of alphabetical order.

### New features

- Added [unpacking syntax](#) for let bindings, which allows things like `let (1, 2) = array`
- Added [enumerate](#) method
- Added [path](#) function for drawing Bézier paths
- Added [layout](#) function to access the size of the surrounding page or container
- Added `key` parameter to [sorted](#) method

## Command line interface

- Fixed --open flag blocking the program
- New Computer Modern font is now embedded into the binary
- Shell completions and man pages can now be generated by setting the `GEN_ARTIFACTS` environment variable to a target directory and then building Typst

## Miscellaneous improvements

- Fixed page numbering in outline
- Added basic i18n for a few more languages (AR, NB, CS, NN, PL, SL, ES, UA, VI)
- Added a few numbering patterns (Ihora, Chinese)
- Added `sinc` [operator](#)
- Fixed bug where math could not be hidden with [hide](#)
- Fixed sizing issues with box, block, and shapes
- Fixed some translations
- Fixed inversion of "R" in [cal](#) and [frak](#) styles
- Fixed some styling issues in math

- Fixed supplements of references to headings
- Fixed syntax highlighting of identifiers in certain scenarios
- [Ratios](#) can now be multiplied with more types and be converted to [floats](#) with the [float](#) function

## Contributors

Thanks to everyone who contributed to this release!

## 5.13 0.1.0

### Breaking changes

- When using the CLI, you now have to use subcommands:

`otypst compile file.typ` or `typst c file.typ` to create a PDF

`otypst watch file.typ` or `typst w file.typ` to compile and watch

`otypst fonts` to list all fonts

- Manual counters now start at zero. Read the "How to step" section [here](#) for more details

- The [bibliography styles](#) "author-date" and "author-title" were renamed to "chicago-author-date" and "chicago-author-title"

### Figure improvements

- Figures now automatically detect their content and adapt their behavior. Figures containing tables, for instance, are automatically prefixed with "Table X" and have a separate counter
- The figure's supplement (e.g. "Figure" or "Table") can now be customized

- In addition, figures can now be completely customized because the show rule gives access to the automatically resolved kind, supplement, and counter

## Bibliography improvements

- The [bibliography](#) now also accepts multiple bibliography paths (as an array)
- Parsing of BibLaTeX files is now more permissive (accepts non-numeric edition, pages, volumes, dates, and Jabref-style comments; fixed abbreviation parsing)
- Labels and references can now include : and . except at the end
- Fixed APA bibliography ordering

## Drawing additions

- Added [polygon](#) function for drawing polygons
- Added support for clipping in [boxes](#) and [blocks](#)

## Command line interface

- Now returns with non-zero status code if there is an error
- Now watches the root directory instead of the current one
- Now puts the PDF file next to input file by default
- Now accepts more kinds of input files (e.g. /dev/stdin)
- Added --open flag to directly open the PDF

## Miscellaneous improvements

- Added [yaml](#) function to load data from YAML files
- Added basic i18n for a few more languages (IT, RU, ZH, FR, PT)
- Added numbering support for Hebrew
- Added support for [integers](#) with base 2, 8, and 16

- Added symbols for double bracket and laplace operator
- The [link](#) function now accepts [labels](#)
- The link syntax now allows more characters
- Improved justification of Japanese and Chinese text
- Calculation functions behave more consistently w.r.t to non-real results
- Replaced deprecated angle brackets
- Reduced maximum function call depth from 256 to 64
- Fixed [first-line-indent](#) being not applied when a paragraph starts with styled text
- Fixed extraneous spacing in unary operators in equations
- Fixed block spacing, e.g. in [block](#) (above: `1cm`, below: `1cm`, ...)
- Fixed styling of text operators in math
- Fixed invalid parsing of language tag in raw block with a single backtick
- Fixed bugs with displaying counters and state
- Fixed crash related to page counter
- Fixed crash when [symbol](#) function was called without arguments
- Fixed crash in bibliography generation
- Fixed access to label of certain content elements
- Fixed line number in error message for CSV parsing
- Fixed invalid autocompletion after certain markup elements

## Contributors

Thanks to everyone who contributed to this release!

# 5.14 Earlier

## March 28, 2023

- **Breaking changes:**

- Enumerations now require a space after their marker, that is, `1. ok` must now be written as `1. ok`
- Changed default style for [term lists](#): Does not include a colon anymore and has a bit more indent

- Command line interface

- Added `--font-path` argument for CLI
- Embedded default fonts in CLI binary
- Fixed build of CLI if `git` is not installed

- Miscellaneous improvements

- Added support for disabling [matrix](#) and [vector](#) delimiters. Generally with
 

```
#set math.mat(delim: none) or one-off with $mat(delim: #none, 1, 2;
3, 4) $.
```

- Added [separator](#) argument to term lists

- Added [round](#) function for equations

- Numberings now allow zeros. To reset a counter, you can write

```
#counter(..).update(0)
```

- Added documentation for [page\(\)](#) and [position\(\)](#) methods on [location](#) type

- Added symbols for double, triple, and quadruple dot accent

- Added smart quotes for Norwegian Bokmål

- Added Nix flake

- Fixed bibliography ordering in IEEE style
- Fixed parsing of decimals in math:  $\$1.2/3.4\$$
- Fixed parsing of unbalanced delimiters in fractions:  $\$1/(2 \times \$$
- Fixed unexpected parsing of numbers as enumerations, e.g. in 1.2
- Fixed combination of page fill and header
- Fixed compiler crash if `repeat` is used in page with automatic width
- Fixed `matrices` with explicit delimiter
- Fixed `indent` property of term lists
- Numerous documentation fixes
- Links in bibliographies are now affected by link styling
- Fixed hovering over comments in web app

Thanks to everyone who contributed to this release!

## March 21, 2023

- Reference and bibliography management
- [Bibliographies](#) and [citations](#) (currently supported styles are APA, Chicago Author Date, IEEE, and MLA)
  - You can now [reference](#) sections, figures, formulas, and works from the bibliography with `@label`
  - You can make an element referenceable with a label:
- `= Introduction <intro>`
- `$ A = pi r^2 $ <area>`
- Introspection system for interactions between different parts of the document

- [counter](#) function

- Access and modify counters for pages, headings, figures, and equations
- Define and use your own custom counters
- Time travel: Find out what the counter value was or will be at some other point in the document (e.g. when you're building a list of figures, you can determine the value of the figure counter at any given figure).

- Counters count in layout order and not in code order

- [state](#) function

- Manage arbitrary state across your document
- Time travel: Find out the value of your state at any position in the document
- State is modified in layout order and not in code order
- [query](#) function
- Find all occurrences of an element or a label, either in the whole document or before/after some location
- Link to elements, find out their position on the pages and access their fields
- Example use cases: Custom list of figures or page header with current chapter title

- [locate](#) function

- Determines the location of itself in the final layout
- Can be accessed to get the page and x, y coordinates
- Can be used with counters and state to find out their values at that location
- Can be used with queries to find elements before or after its location

- New [measure](#) function
  - Measure the layouted size of elements
  - To be used in combination with the new [style](#) function that lets you generate different content based on the style context something is inserted into (because that affects the measured size of content)
- Exposed content representation
  - Content is not opaque anymore
  - Content can be compared for equality
    - The tree of content elements can be traversed with code
  - Can be observed in hover tooltips or with [repr](#)
  - New [methods](#) on content: `func`, `has`, `at`, and `location`
  - All optional fields on elements are now settable
  - More uniform field names (`heading.title` becomes `heading.body`, `list.items` becomes `list.children`, and a few more changes)
- Further improvements
  - Added [figure](#) function
  - Added [numbering](#) parameter on equation function
  - Added [numbering](#) and [number-align](#) parameters on page function
    - The page function's [header](#) and [footer](#) parameters do not take functions anymore. If you want to customize them based on the page number, use the new [numbering](#) parameter or [counter](#) function instead.
  - Added [footer-descent](#) and [header-ascent](#) parameters

- Better default alignment in header and footer
- Fixed Arabic vowel placement
- Fixed PDF font embedding issues
- Renamed `math.formula` to [math.equation](#)
- Font family must be a named argument now: `#set text(font: "...")`
- Added support for [hanging indent](#)
- Renamed paragraph `indent` to [first-line-indent](#)
- More accurate [logarithm](#) when base is 2 or 10
- Improved some error messages
- Fixed layout of [terms](#) list
- Web app improvements
  - Added template gallery
  - Added buttons to insert headings, equations, raw blocks, and references
  - Jump to the source of something by clicking on it in the preview panel (works for text, equations, images, and more)
  - You can now upload your own fonts and use them in your project
  - Hover debugging and autocompletion now takes multiple files into account and works in show rules
  - Hover tooltips now automatically collapse multiple consecutive equal values
  - The preview now automatically scrolls to the right place when you type
  - Links are now clickable in the preview area
  - Toolbar, preview, and editor can now all be hidden

- Added autocompletion for raw block language tags
- Added autocompletion in SVG files
- New back button instead of four-dots button
- Lots of bug fixes

## February 25, 2023

- Font changes
  - New default font: Linux Libertine
  - New default font for raw blocks: DejaVu Sans Mono
  - New default font for math: Book weight of New Computer Modern Math
  - Lots of new math fonts available
  - Removed Latin Modern fonts in favor of New Computer Modern family
  - Removed unnecessary smallcaps fonts which are already accessible through the corresponding main font and the [smallcaps](#) function
- Improved default spacing for headings
- Added [panic](#) function
- Added [clusters](#) and [codepoints](#) methods for strings
- Support for multiple authors in [set document](#)
- Fixed crash when string is accessed at a position that is not a char boundary
- Fixed semicolon parsing in `#var ;`
- Fixed incremental parsing when inserting backslash at end of `#"abc"`
- Fixed names of a few font families (including Noto Sans Symbols and New Computer Modern families)

- Fixed autocompletion for font families
- Improved incremental compilation for user-defined functions

## February 15, 2023

- [Box](#) and [block](#) have gained `fill`, `stroke`, `radius`, and `inset` properties
- Blocks may now be explicitly sized, fixed-height blocks can still break across pages
- Blocks can now be configured to be [breakable](#) or not
- [Numbering style](#) can now be configured for nested enums
- [Markers](#) can now be configured for nested lists
- The [eval](#) function now expects code instead of markup and returns an arbitrary value. Markup can still be evaluated by surrounding the string with brackets.
- PDFs generated by Typst now contain XMP metadata
- Link boxes are now disabled in PDF output
- Tables don't produce small empty cells before a pagebreak anymore
- Fixed raw block highlighting bug

## February 12, 2023

- Shapes, images, and transformations (move/rotate/scale/repeat) are now block-level. To integrate them into a paragraph, use a [box](#) as with other elements.
- A colon is now required in an "everything" show rule: Write `show:` `it => ..` instead of `show it => ..`. This prevents intermediate states that ruin your whole document.

- Non-math content like a shape or table in a math formula is now centered vertically
- Support for widow and orphan prevention within containers
- Support for [RTL](#) in lists, grids, and tables
- Support for explicit `auto` sizing for boxes and shapes
- Support for fractional (i.e. `1fr`) widths for boxes
- Fixed bug where columns jump to next page
- Fixed bug where list items have no leading
- Fixed relative sizing in lists, squares and grid auto columns
- Fixed relative displacement in [place](#) function
- Fixed that lines don't have a size
- Fixed bug where `set document(...)` complains about being after content
- Fixed parsing of `not in` operation
- Fixed hover tooltips in math
- Fixed bug where a heading show rule may not contain a pagebreak when an outline is present
- Added [baseline](#) property on [box](#)
- Added [tg](#) and [ctg](#) operators in math
- Added delimiter setting for [cases](#) function
- Parentheses are now included when accepting a function autocompletion

**February 2, 2023**

- Merged text and math symbols, renamed a few symbols (including `infty` to `infinity` with the alias `oo`)
- Fixed missing italic mappings
- Math italics correction is now applied properly
- Parentheses now scale in `$zeta(x/2)$`
- Fixed placement of large root index
- Fixed spacing in `$abs(-x)$`
- Fixed inconsistency between text and identifiers in math
- Accents are now ignored when positioning superscripts
- Fixed vertical alignment in matrices
- Fixed `text set` rule in `raw show` rule
- Heading and list markers now parse consistently
- Allow arbitrary math directly in content

## January 30, 2023

[Go to the announcement blog post.](#)

- New expression syntax in markup/math
  - Blocks cannot be directly embedded in markup anymore
  - Like other expressions, they now require a leading hash
  - More expressions available with hash, including literals (`#"string"`) as well as field access and method call without space: `#emoji.face`
- New import syntax
  - `#import "module.typ"` creates binding named `module`

- `#import "module.typ": a, b` or `#import "module.typ": * to import items`

- `#import emoji: face, turtle` to import from already bound module

- New symbol handling

- Removed symbol notation

- Symbols are now in modules: `sym`, `emoji`, and `math`

- Math module also reexports all of `sym`

- Modified through field access, still order-independent

- Unknown modifiers are not allowed anymore

- Support for custom symbol definitions with `symbol` function

- Symbols now listed in documentation

- New `math` module

- Contains all math-related functions

- Variables and function calls directly in math (without hash) access this module instead of the global scope, but can also access local variables

- Can be explicitly used in code, e.g. `#set math.vec (delim: "[")`

- Delimiter matching in math

- Any opening delimiters matches any closing one

- When matched, they automatically scale

- To prevent scaling, escape them

- To forcibly match two delimiters, use `lr` function

- Line breaks may occur between matched delimiters

- Delimiters may also be unbalanced
- You can also use the `lr` function to scale the brackets (or just one bracket) to a specific size manually
- Multi-line math with alignment
  - The `\` character inserts a line break
  - The `&` character defines an alignment point
  - Alignment points also work for underbraces, vectors, cases, and matrices
  - Multiple alignment points are supported
- More capable math function calls
  - Function calls directly in math can now take code expressions with hash
  - They can now also take named arguments
  - Within math function calls, semicolons turn preceding arguments to arrays to support matrices: `$mat(1, 2; 3, 4)$`
- Arbitrary content in math
  - Text, images, and other arbitrary content can now be embedded in math
  - Math now also supports font fallback to support e.g. CJK and emoji
- More math features
  - New text operators: `op` function, `lim`, `max`, etc.
  - New matrix function: `mat`
  - New n-ary roots with `root` function: `$root(3, x)$`
  - New under- and overbraces, -brackets, and -lines
  - New `abs` and `norm` functions

- New shorthands: `[|, |]`, and `||`
- New `attach` function, overridable attachments with `script` and `limit`
- Manual spacing in math, with `h`, `thin`, `med`, `thick` and `quad`
- Symbols and other content may now be used like a function, e.g. `$zeta(x)$`
- Added Fira Math font, removed Noto Sans Math font
- Support for alternative math fonts through `#show math.formula`:

```
set text("Fira Math")
```

- More library improvements
  - New `calc module`, `abs`, `min`, `max`, `even`, `odd` and `mod` moved there
  - New `message` argument on `assert` function
  - The `pairs` method on dictionaries now returns an array of length-2 arrays instead of taking a closure
    - The method call `dict.at("key")` now always fails if "key" doesn't exist Previously, it was allowed in assignments. Alternatives are `dict.key = x` and `dict.insert("key", x)`.
- Smarter editor functionality
  - Autocompletion for local variables
  - Autocompletion for methods available on a value
  - Autocompletion for symbols and modules
  - Autocompletion for imports
  - Hover over an identifier to see its value(s)
- Further editor improvements
  - New Font menu with previews

- Single projects may now be shared with share links
- New dashboard experience if projects are shared with you
- Keyboard Shortcuts are now listed in the menus and there are more of them
- New Offline indicator
- Tooltips for all buttons
- Improved account protection
- Moved Status indicator into the error list button
- Further fixes
  - Multiple bug fixes for incremental parser
  - Fixed closure parameter capturing
  - Fixed tons of math bugs
  - Bugfixes for performance, file management, editing reliability
  - Added redirection to the page originally navigated to after signin

## **6. Roadmap**

This page lists planned features for the Typst language, compiler, library and web app. Since priorities and development realities change, this roadmap is not set in stone. Features that are listed here will not necessarily be implemented and features that will be implemented might be missing here. Moreover, this roadmap only lists larger, more fundamental features and bugs.

Are you missing something on the roadmap? Typst relies on your feedback as a user to plan for and prioritize new features. Get started by filing a new issue on [GitHub](#) or discuss your feature request with the community.

## Language and Compiler

### • Styling

- Support for revoking style rules
- Ancestry selectors (e.g., within)
- ~~Fix show rule recursion crashes~~
- ~~Fix show-set issues~~

### • Scripting

- Function for debug logging
- Fix issues with paths being strings
- Custom types (that work with set and show rules)
- Type hints
- Function hoisting (if feasible)
- ~~Data loading functions~~
- ~~Support for compiler warnings~~
- ~~Types as first-class values~~
- ~~More fields and methods on primitives~~
- ~~WebAssembly plugins~~
- ~~Get values of set rules~~
- ~~Replace locate, etc. with unified context system~~

◦ ~~Allow expressions as dictionary keys~~

◦ ~~Package management~~

• **Model**

◦ Fix issues with numbering patterns

◦ Better support for custom referenceable things

◦ Richer built-in outline customization

◦ Enum continuation

◦ ~~Support a path or bytes in places that currently only support paths,~~

◦ ~~superseding .decode-scoped functions (not yet released)~~

◦ ~~Bibliography and citation customization via CSL (Citation Style Language)~~

◦ ~~Relative counters, e.g. for figure numbering per section~~

• **Text**

◦ Font fallback warnings

◦ Bold, italic, and smallcaps synthesis

◦ Variable fonts support

◦ Ruby and Warichu

◦ Kashida justification

◦ ~~Support for basic CJK text layout rules~~

◦ ~~Fix SVG font fallback~~

◦ ~~Themes for raw text and more/custom syntaxes~~

• **Math**

◦ Fix syntactic quirks

- Fix single letter strings
- Fix font handling
- Fix attachment parsing priorities
- Provide more primitives
- Improve equation numbering
- Big fractions

- **Layout**

- Fix issues with list (in particular baselines & alignment)
- Improve widow & orphan prevention
- Support for side-floats and other "collision" layouts
- Better support for more canvas-like layouts
- Unified layout primitives across normal content and math
- Page adjustment from within flow
- Chained layout regions
- Grid-based typesetting
- Balanced columns
- Drop caps
- End notes, maybe margin notes
- ~~Expand floating layout (e.g. over two columns)~~
- ~~Support for "sticky" blocks that stay with the next one~~
- ~~Fix footnote issues~~ (not all released yet)
- ~~Footnotes~~

- ~~Basic floating layout~~
- ~~Row span and column span in table~~
- ~~Per-cell table stroke customization~~

- **Visualize**

- Arrows
- Better path drawing, possibly path operations
- Color management
- ~~More configurable strokes~~

- ~~Gradients~~

- ~~Patterns~~

- **Introspection**

- Support for freezing content, so that e.g. numbers in it remain the same if it appears multiple times

- **Export**

- HTML export
- Tagged PDF for Accessibility
- PDF/X support
- EPUB export
- ~~PDF/A support~~
- ~~PNG export~~
- ~~SVG export~~
- ~~Support for transparency in PDF~~

◦ Fix issues with SVGs in PDF

◦ Fix emoji export in PDF

◦ Selectable text in SVGs in PDF

◦ Better font subsetting for smaller PDFs

### • **CLI**

◦ Support for downloading fonts on-demand automatically

◦ ~~typst query for querying document elements~~

◦ ~~typst init for creating a project from a template~~

◦ ~~typst update for self updating the CLI~~

### • **Tooling**

◦ Documentation generator and doc comments

◦ Autoformatter

◦ Linter

### • **Performance**

◦ Reduce memory usage

◦ Optimize runtime of optimal paragraph layout

◦ Parallelize layout engine

### • **Development**

◦ Benchmarking

◦ Better contributor documentation

## **Web App**

### • **Editing**

- Smarter & more action buttons
  - Inline documentation
  - Preview autocomplete entry
  - Color Picker
  - Symbol picker
  - Basic, built-in image editor (cropping, etc.)
  - GUI inspector for editing function calls
  - Cursor in preview
  - Hover tooltips for debugging
  - Scroll to cursor position in preview
  - Folders in projects
  - Outline panel
  - More export options
  - Preview in a separate window
  - Sync literature with Zotero and Mendeley
  - Paste modal
  - Improve panel
- **Writing**
- Word count
  - Structure view
  - Text completion by LLM
  - Spell check

•**Collaboration**

- Change tracking
- Version history
- ~~Chat-like comments~~
- ~~Git integration~~

•**Project management**

- Drag-and-drop for projects
- Template generation by LLM
- ~~LaTeX, Word, Markdown import~~
- ~~Thumbnails for projects~~

•**Settings**

- Keyboard shortcuts configuration
- Better project settings
- Avatar Cropping
- ~~System Theme setting~~

•**Other**

- Offline PWA
- Mobile improvements
- Two-Factor Authentication
- Advanced search in projects
- Private packages in teams
- ~~LDAP Single sign-on~~

- ~~Compiler version picker~~
- ~~Presentation mode~~
- ~~Support for On-Premises deployment~~
- ~~Typst Universe~~

## 7. Community

Hey and welcome to the Community page! We're so glad you're here. Typst is developed by an early-stage startup and it is still early days, but it would be pointless without people like you who are interested in it.

We would love to not only hear from you but to also provide spaces where you can discuss any topic around Typst, typesetting, writing, the sciences, and typography with other likeminded people.

**Our [Forum](#) is the best place to get answers for questions on Typst and to show off your creations.** If you would like to chat with the community and shape the future development of Typst, we would like to also invite you to our [Discord server](#). We coordinate our Open-Source work there, but you can also iterate on Typst projects and discuss off-topic things with the community members. Both the Forum and the Discord server are open for everyone. Of course, you are also very welcome to connect with us on social media ([Mastodon](#), [Bluesky](#), [Instagram](#), [LinkedIn](#), and [GitHub](#)).

### **What to share?**

For our community, we want to foster versatility and inclusivity. You are welcome to post about any topic that you think would interest other community members, but if you need a little inspiration, here are a few ideas:

- Share and discuss your thoughts and ideas for new features or improvements you'd like to see in Typst
- Showcase documents you've created with Typst, or share any unique or creative ways you've used the platform
- Share importable files or templates that you use to style your documents
- Alert us of bugs you encounter while using Typst

## Following the development

Typst is still under very active development and breaking changes can occur at any point. The compiler is developed in the open on [GitHub](#).

We will update the members of our Discord server and our social media followers when new features become available. We'll also update you on the development progress of large features.

## How to support Typst

If you want to support Typst, there are multiple ways to do that! You can [contribute to the code](#) or [translate the strings in Typst](#) to your native language if it's not supported yet. You can also help us by [subscribing to the paid tier of our web app](#) or [sponsoring our Open Source efforts!](#) Multiple recurring sponsorship tiers are available and all of them come with a set of goodies. No matter how you contribute, thank you for your support!

## Community Rules

We want to make our community a safe and inclusive space for everyone. Therefore, we will not tolerate any sexual harassment, sexism, political attacks, derogatory language or personal insults, racism, doxing, and other inappropriate behaviour. We pledge to remove members that are in violation of these rules. [Contact us](#) if you think another community member acted inappropriately towards you. All complaints will be reviewed and investigated promptly and fairly.

In addition, our [privacy policy](#) applies on all community spaces operated by us, such as the Discord server. Please also note that the terms of service and privacy policies of the respective services apply.

## See you soon!

Thanks again for learning more about Typst. We would be delighted to meet you on our [Discord server](#)!