Looking to explore formula basics? Check out our introduction article:

- Formulas

Notion Get Notion free

Properties

Formulas support many property types. For those that aren't supported directly, data is automatically converted into another data type (usually text).

Property Types	Examples	Formula Type
Title	`prop("Title")` `prop("Title").length()`	Text
Text	`prop("Text")` `prop("Text").length()`	Text
Select	`prop("Priority") == "High"`	Text
Multi-Select	`prop("Tags").length()` `prop("Tags").includes("Finance")`	Text
Checkbox	`prop("Checkbox")` `not prop("Checkbox")`	Boolean

Email, URL, Phone Number	`!empty(prop("Phone"))`
	`!empty(prop("Email"))`
	`link("Call", "tel:" +`
		Text

	`prop("Phone")`	
Unique ID	`prop("Task ID").split("-").first()` ← Prefix `prop("Task ID").split("-").last()` ← ID	Text
Created By, Edited By	`prop("Created By").name()` `prop("Created By").email()`	Person

Person	`prop("Person")`
	`prop("Person").at(0).name()`
	`prop("Person").map(current.email())`
		Person

Date, Created Time, Last Edited Time	`prop("Due Date") > now()` `dateBetween(prop("Birthday"), now(), "days")`	Date
Number	`prop("Number") / 2` `pi() * prop("Radius") ^ 2`	Number
Relation	`prop("Tasks").length()` `prop("Tasks").filter(current.prop("Status") !== "Done")`	Page
Rollup	`prop("Purchases").length()` `prop("Average cost") * 12`	Number, date, or list of any type. Depends on rollup configuration.

Built-ins

Built-ins are specific symbols and values that are built into the language to help designate a calculation.

Built-in	Example
Math operators: `+`, `-`, `*`, `%`	`2 * pi()` `"hello" + "world"`
Boolean values: `true`, `false`	`true` `false`
Comparison operators: `==`, `>`, `>=`, `<`, `<=`	`123 == 123` = `true` `"Notion" == "Motion"` = `false`

Logical operators:
`and`, `or`, `not`
	`and:`
	`true and false`
	`true && false`
	`and(true, false)`
	`or:`
	`true or false`
	`true || false`
	`or(true, false)`
	`not:`
	`not true`
	`!true`

Ternary operator: `? :`	`X ? Y : Z` is equivalent to `if(X, Y, Z)`

Functions

Notion formulas support the following functions.

Name	Description	Example

if	Returns the first value if the condition is true; otherwise, returns the second value.	`if(true, 1, 2)` = `1`
		`if(false, 1, 2)` = `2`
		`prop("Checked")`

		`== true ? "Complete" : "Incomplete"`
ifs	Returns the value that corresponds to the first true condition. This can be used as an alternative to multiple nested if() statements.	`ifs(true, 1, true, 2, 3)` = `1` `ifs(false, 1, false, 2, 3)` = `3`
empty	Returns true if the value is empty. 0, “”, and [] are considered empty.	`empty(0)` = `true` `empty([])` = `true`
length	Returns the length of the text or list value.	`length("hello")` = `5` `length([1, 2, 3])` = `3`
substring	Returns the substring of the text from the start index (inclusive) to the end index (optional and exclusive).	`substring("Notion", 0, 3)` = `"Not"` `substring("Notion", 3)` = `"ion"`
contains	Returns true if the search string is present in the value.	`contains("Notion", "ot")` = `true`
test	Returns true if the value matches the regular expression and false otherwise.	`test("Notion", "Not")` = `true` `test("Notion", "\\d")` = `false`
match	Returns all matches of the regular expression as a list.	`match("Notion Notion", "Not")` = `["Not", "Not"]` `match("Notion 123 Notion 456", "\\d+")` = `["123", "456"]`

replace	Replaces the first match of the regular expression with the replacement value.	`replace("Notion Notion", "N", "M")` = `"Motion Notion"`
replaceAll	Replaces all matches of the regular expression with the replacement value.	`replaceAll("Notio n Notion", "N", "M")` = `"Motion Motion"` `replaceAll("Notio n 123", "\\d", "")` = `"Notion"`
lower	Converts the text to lowercase.	`lower("NOTION")` = `"notion"`
upper	Converts the text to uppercase.	`upper("notion")` = `"NOTION"`
repeat	Repeats the text a given number of times.	`repeat("0", 4)` = `"0000"` `repeat("~=", 10)` = `"~=~=~=~=~=~=~= ~=~=~=~=~="`
link	Creates a hyperlink from the label text and the URL.	`link("Notion", "https://notion.so")` = "Notion"
style	Adds styles and colors to the text. Valid formatting styles: `"b"` (bold), `"u"` (underline), `"i"` (italics), `"c"` (code), or `"s"` (strikethrough). Valid colors: `"gray"`, `"brown"`, `"orange"`, `"yellow"`, `"green"`, `"blue"`, `"purple"`, `"pink"`, and `"red"`. Add `"_background"` to colors to set background colors.	`style("Notion", "b", "u")` = " Notion " `style("Notion", "blue", "gray_background")`

unstyle	Removes formatting styles from the text. If no styles are specified, all styles are removed.	`unstyle("Text")` `unstyle("Text", "b")`
format	Returns the value formatted as text.	`format(1234)` = `"1234"` `format(now())` = `"August 30, 2023 17:55"`
add	Returns the sum of two numbers.	`add(5, 10)` = `15` `5 + 10` = `15`
subtract	Returns the difference of two numbers.	`subtract(5, 10)` = `-5` `5 - 10` = `-5`
multiply	Returns the product of two numbers.	`multiply(5, 10)` = `50` `5 * 10` = `50`
mod	Returns the first number modulo the second number.	`mod(5, 10)` = `5` `5 % 10` = `5`
pow	Returns the result of a base number raised to an exponent power.	`pow(5, 10)` = `9765625` `5 ^ 10` = `9765625`
divide	Returns the quotient of two numbers.	`divide(5, 10)` = `0.5` `5 / 10` = `0.5`
min	Returns the smallest number of the arguments.	`min(1, 2, 3)` = `1` `min([1, 2, 3])` = `1`
		`max(1, 2, 3)` =

max	Returns the largest number of the arguments.	`3` `max([1, 2, 3])` = `3`
sum	Returns the sum of its arguments.	`sum(1, 2, 3)` = `6` `sum([1, 2, 3], 4, 5)` = `15`
abs	Returns the absolute value of the number.	`abs(10)` = `10` `abs(-10)` = `10`
round	Returns the value of a number rounded to the nearest integer.	`round(0.4)` = `0` `round(-0.6)` = `-1`
ceil	Returns the smallest integer greater than or equal to the number.	`ceil(0.4)` = `1` `ceil(-0.6)` = `0`
floor	Returns the largest integer less than or equal to the number.	`floor(0.4)` = `0` `floor(-0.6)` = `-1`
sqrt	Returns the positive square root of the number.	`sqrt(4)` = `2` `sqrt(7)` = `2.6457513110645907`
cbrt	Returns the cube root of the number.	`cbrt(9)` = `2.080083823051904` `cbrt(64)` = `4`
exp	Returns e^x, where x is the argument, and e is Euler's number (2.718...), the base of the natural logarithm.	`exp(1)` = `2.718281828459045` `exp(-1)` = `0.36787944117144233`

ln	Returns the natural logarithm of the number.	`ln(2.718281828459045)` = `1` `ln(10)` = `2.302585092994046`
log10	Returns the base 10 logarithm of the number.	`log10(10)` = `1` `log10(100000)` = `5`
log2	Returns the base 2 logarithm of the number.	`log2(4)` = `2` `log2(1024)` = `10`
sign	Returns 1 if the number is positive, -1 if it is negative, and 0 if it is zero.	`sign(-10)` = `-1` `sign(10)` = `1`
pi	Returns the ratio of a circle's circumference to its diameter.	`pi()` = `3.14159265358979393`
e	Returns the base of the natural logarithm.	`e()` = `2.718281828459045`

toNumber	Parses a number from text.	`toNumber("2")` = `2`
		`toNumber(now())` = `1693443300000`
		`toNumber(true)` = `1`

now	Returns the current date and time.	`now()` = `@August 30, 2023 5:55 PM`
		`minute(parseDate("2023-07-`

minute	Returns the minute of the date (0-59).	`10T17:35Z"))` = `35`
hour	Returns the hour of the date (0-23).	`hour(parseDate("2023-07-10T17:35Z"))` = `17`
day	Returns the day of the week of the date, between 1 (Monday) and 7 (Sunday).	`day(parseDate("2023-07-10T17:35Z"))` = `1`
date	Returns the day of the month from the date (1-31).	`date(parseDate("2023-07-10T17:35Z"))` = `10`
week	Returns the ISO week of the year of the date (1-53).	`week(parseDate("2023-01-02"))` = `1`
month	Returns the month of the date (1-12).	`month(parseDate("2023-07-10T17:35Z"))` = `7`
year	Returns the year of the date.	`year(now())` = `2023`
dateAdd	Adds time to the date. The unit argument can be one of: `"years"`, `"quarters"`, `"months"`, `"weeks"`, `"days"`, `"hours"`, or `"minutes"`.	`dateAdd(now(), 1, "days")` = `@August 31, 2023 5:55 PM` `dateAdd(now(), 2, "months")` = `@October 30, 2023 5:55`

		`PM` `dateAdd(now(), 3, "years")` = `@August 30, 2026 5:55 PM`
dateSubtract	Subtracts time from the date. The unit argument can be one of: `"years"`, `"quarters"`, `"months"`, `"weeks"`, `"days"`, `"hours"`, or `"minutes"`.	`dateSubtract(now(), 1, "days")` = `@August 29, 2023 5:55 PM` `dateSubtract(now(), 2, "months")` = `@June 30, 2023 5:55 PM` `dateSubtract(now(), 3, "years")` = `@August 30, 2020 5:55 PM`
dateBetween	Returns the difference between two dates. The unit argument can be one of: `"years"`, `"quarters"`, `"months"`, `"weeks"`, `"days"`, `"hours"`, or `"minutes"`.	`dateBetween(now(), parseDate("2022-09-07"), "days")` = `357` `dateBetween(parseDate("2030-01-01"), now(), "years")` = `6`
dateRange	Returns a date range constructed from the start and end dates.	`dateRange(prop("Start Date"), prop("End Date"))` = `@September 7, 2022 → September 7, 2023`
		`dateStart(prop("Date Range"))` =

dateStart	Returns the start of the date range.	`@September 7, 2022` `dateBetween(dateStart(prop("Date Range")), dateEnd(prop("Date Range")), "days")` = `-365`
dateEnd	Returns the end of the date range.	`dateEnd(prop("Date range"))` = `@September 7, 2023` `dateBetween(dateEnd(prop("Date Range")), dateStart(prop("Date Range")), "days")` = `365`
timestamp	Returns the current Unix timestamp, representing the number of milliseconds that have elapsed since January 1, 1970.	`timestamp(now())` = `1693443300000`
fromTimestamp	Returns the date from the given Unix timestamp. The timestamp represents the number of milliseconds that have elapsed since January 1, 1970. Note: the returned date will not retain the seconds & milliseconds.	`fromTimestamp(1689024900000)` = `@July 10, 2023 2:35 PM`
formatDate	Formats the date using a custom format string. The format string can contain the following text to represent parts of the date: `"YYYY"` for year, `"MM"` for month, `"DD"` for day, `"HH"` for hour, `"mm"` for	`formatDate(now(), "MMMM D, Y")` = `"August 30, 2023"` `formatDate(now(), "MM/DD/YYYY")` = `"08/30/2023"`

	minute.	`formatDate(now(), "HH:mm A")` = `"17:55 PM"`
parseDate	Returns the date parsed according to the ISO 8601 standard.	`parseDate("2022-01-01")` = `@January 1, 2022` `parseDate("2022-01-01T00:00Z")` = `@December 31, 2021 4:00 PM`
name	Returns the name of a person.	`name(prop("Created By"))` `prop("Pioneers").map(name(current)).join(", ")` = `"Grace Hopper, Ada Lovelace"`
email	Returns the email address of a person.	`email(prop("Created By"))` `prop("People").map(email(current)).join(", ")`
at	Returns the value at the specified index in a list.	`at([1, 2, 3], 1)` = `2`
first	Returns the first item in the list.	`first([1, 2, 3])` = `1`
last	Returns the last item in the list.	`last([1, 2, 3])` = `3`
	Returns the items of the list from the	`slice([1, 2, 3], 1, 2)` = `[2]`

slice	provided start index (inclusive) to the end index (optional and exclusive).	`slice(["a", "b", "c"], 1)` = `["b", "c"]`
concat	Returns the concatenation of multiple lists.	`concat([1, 2], [3, 4])` = `[1, 2, 3, 4]` `concat(["a", "b"], ["c", "d"])` = `["a", "b", "c", "d"]`
sort	Returns the list in sorted order.	`sort([3, 1, 2])` = `[1, 2, 3]`
reverse	Returns the reversed list.	`reverse(["green", "eggs", "ham"])` = `["ham", "eggs", "green"]`
join	Returns the values of the list with the joiner placed between each of the values.	`join(["a", "b", "c"], ", ")` = `"a, b, c"` `join(["dog", "go"], "")` = `"doggo"`
split	Returns the list of values created by splitting a text input by a separator.	`split("apple,pear, orange", ",")` = `["apple", "pear", "orange"]`
unique	Returns the list of unique values in the input list.	`unique([1, 1, 2])` = `[1, 2]`
includes	Returns true if the list contains the specified value, and false otherwise.	`includes(["a", "b", "c"], "b")` = `true` `includes([1, 2, 3], 4)` = `false`
		`find(["a", "b", "c"], current == "b")` =

find	Returns the first item in the list for which the condition evaluates to true.	`"b"` `find([1, 2, 3], current > 100)` = `Empty`
findIndex	Returns the index of the first item in the list for which the condition is true.	`findIndex(["a", "b", "c"], current == "b")` = `1` `findIndex([1, 2, 3], current > 100)` = `-1`
filter	Returns the values in the list for which the condition is true.	`filter([1, 2, 3], current > 1)` = `[2, 3]` `filter(["a", "b", "c"], current == "a")` = `["a"]`
some	Returns true if any item in the list satisfies the given condition, and false otherwise.	`some([1, 2, 3], current == 2)` = `true` `some(["a", "b", "c"], current.length > 2)` = `false`
every	Returns true if every item in the list satisfies the given condition, and false otherwise.	`every([1, 2, 3], current > 0)` = `true` `every(["a", "b", "c"], current == "b")` = `false`
map	Returns the list populated with the results of calling the expression on every item in the input list.	`map([1, 2, 3], current + 1)` = `[2, 3, 4]` `map([1, 2, 3], current + index)`

		= `[1, 3, 5]`
flat	Flattens a list of lists into a single list.	`flat([1, 2, 3])` = `[1, 2, 3]` `flat([[1, 2], [3, 4]])` = `[1, 2, 3, 4]`
id	Returns the id of the page. If no page is provided, returns the id of the page the formula is on.	`id()` `id(prop("Relation").first())`
equal	Returns true if both values are equal and false otherwise.	`equal(1, 1)` = `true` `"a" == "b"` = `false`
unequal	Returns false if both values are equal and true otherwise.	`unequal(1, 2)` = `true` `"a" != "a"` = `false`
let	Assigns a value to a variable and evaluates the expression using that variable.	`let(person, "Alan", "Hello, " + person + "!")` = `"Hello, Alan!"` `let(radius, 4, round(pi() * radius ^ 2))` = `50`
lets	Assigns values to multiple variables and evaluates the expression using those variables.	`lets(a, "Hello", b, "world", a + " " + b)` = `"Hello world"` `lets(base, 3, height, 8, base * height / 2)` = `12`