[count)[register] operator
[register] [count) operator

"h yy
2"hp → execute the cmd 2 times
"hp

++ Register types ++
- Unnamed - default = ""
- Numbered = "0 "1 ... "9
- Named
- The black hole = "_

registers preceded with "" the unnamed register is actually double quote the double quote

→ the unnamed register contains the last bit of text from an operation like delete or yank as you've already seen

Registers
"" holds text from d, c, s, x and y operations
"0 holds last text yanked (y)
"1 holds last text deleted (d) or changed (c).

with each successive deletion or change vim shifts the previous contents of register one into register two and register 2 to 3 and so forth losing the previous contents of register 9 if it falls off

to look at the registers we can type
:reg ↵

"_ : black hole
"_dd // text in "" still the same
"_p // that line gets past

3yy + :reg
↳ in the register there ...^J
↓
stop represent line character
it will replace with ↵ enter

the 26 named registers from "a to "z | "ayy → yank holding in a register

if you want to append more text to the register use the capital letter of register name
"Ayy or "shift+a yy → in register "a — "A
Vim registers, the basics and beyond

You can think of registers as a bunch of space in memory that vim uses to store some text. Each of these spaces have a identifier, so it can be accessed later, it's no different than when you copy some text to your clipboard, except you usually have just one clipboard to copy to, while vim allows you to have multiple places to store different texts.

the basic usage:
every register is accessed using a double quote before its name.
For example: we can access the content that is in the register r with "r
You could add selected text to the register r by doing "ry. You are copying (yanking) the selected text, and then adding it to the register "r. to past the content of this register, the logic is the same: "rp. You are pasting the data that is in this register.
You can also access the registers in insert/command mode with ~~ctrl-r+~~ Ctrl-r + register name, like in Ctrl-r r. It will just paste the text in your current buffer. You can use :reg command to see all the registers and their content, or filter just the one that you are interested in :reg a bc

:reg abc

The unnamed register
vim has unnamed (or default) register that can be accessed with "". Any text you delete (with d, c, s, or x) or yank (with y) will be placed there, and that's what vim uses to past when no explicit register is given, A simple p is the same thing as doing ""p
Never lose a yanked text again: it has happend to all of us. we yank some text then delete other, and when we try the yanked text. it's not there anymore, vim replaced it with the text that you deleted, then you need to go there and yanked that text again.

well, as I said vim will always replace the unnamed register, but of course we didn't lost the yanked text, vim would not have survived that long if it was that dumb, right?

vim automatically populates what is called the numbered registers for us expected. there are registers from "0 to "9.

"0 will always have the content of latest yank, and the others will have last 9 deleted text; being "1 the newest, and "9 the oldest. So if you yanked some text, you can always refer to it using "0p.

The read only registers

there are 4 read only registers: ". | "% | ": | "#

The last inserted text is stored on ". , and it's quite handy if you need to write the same text twice, in different places, not needing to yank and past

"% has the current file path, starting from the directory where vim was first opened. what I usually use it for is to copy the current file to clipboard, so I can use it externally (running a script in another terminal, for instance). You could execute :let @+=@% to do that. let is used to write to a register, and "+ is the clipboard register, so we are copying the current file path to the clipboard.

": is the most recently executed command. If ~~you save the current buffer with :w~~

++ ~~test~~ inserting changing replacing and joining ++

shift+i or I // -> the cursor jumps to the first non_blank
capital i // character in the line and you're placed into
// insert mode
a CMD // a command appends text after the current cursor position
A cmd // appends at the end of line
o CMD // it begins a new line below the cursor and places
// you in insert mode
O or shift+o // ~~sst~~ start new line above the cursor position

---

You already know how you can make if CMD repeat as many time as you want by proceeding that with a number [which is called a count]
↳ it also work with insert CMD

30i*
↳ will insert * 80 times

=> Creat 5 line that begin with "#".

>> 5o esc
lower case O

=> Create 4 lines that begin "10.11.12."
4o 10.11.12. esc

There is a very similar mode to insert mode in ~~Vim~~ Vim and it's called
++ replace mode ++
↳ when entre replace mode each character you type replaces an existing character.

R or shift+r

-- REPLACE -- -> so in replace mode one character in the line is deleted for every character you type, if there is no character to the right at the end of the line then that type character is appended to the ~~end of the~~ line

r => replace one character
cw => change word ~~dw~~ => delete word and enter insert mode
"a cw => will store the replaced text to "a register

c$ // replace from your current cursor position all the way through the end of line

shift case =

C or shift + c

cc // change entire line

↳ you can also use register to sort the replaced text + reset it