Computational Methods for Linguists

Ling 471

Olga Zamaraeva (Instructor)
Yuanhe Tian (TA)

04/13/21

Takeaways
From April 8

- Available tools:
  - GUIs: IDEs, text editors
    - Good for most users, especially to write code
      - And to **debug**
  - Command line
    - Most people don't like it because it is easy to write a wrong command and hard to understand what the problem is
    - Sometimes inevitable
    - More general and powerful than any GUI
  - Version control (via both GUI and command line)
    - Keep track of your development
    - Back up

Version control
What we need

- In this class:
  - Focus on **storing** code and keeping track of changes
  - Use GitHub or VS Code GitLens extension to look at **diffs**
  - If you actually need to go to a previous version:
    - Don't let it frustrate you and use office hours :)
- Retreiving **previous** versions is important
  - But not straightforward :(
  - Some hints at the end of slide deck
    - We will address this as needed

Reminders

- Assignment 1 is due tonight
- Make sure we can map **your full name** to your GitHub **username**!
    - e.g. put it in README
    - e.g. email the mapping to us
        - *"Full name: Mary Carrasco. GitHub: mcar22"*
- If no patas access:
    - Let Olga know asap, including date when you requested the access.

THANKS, EVERYONE,
for great Blog discussions!

Assignment 2
Published; due April 27

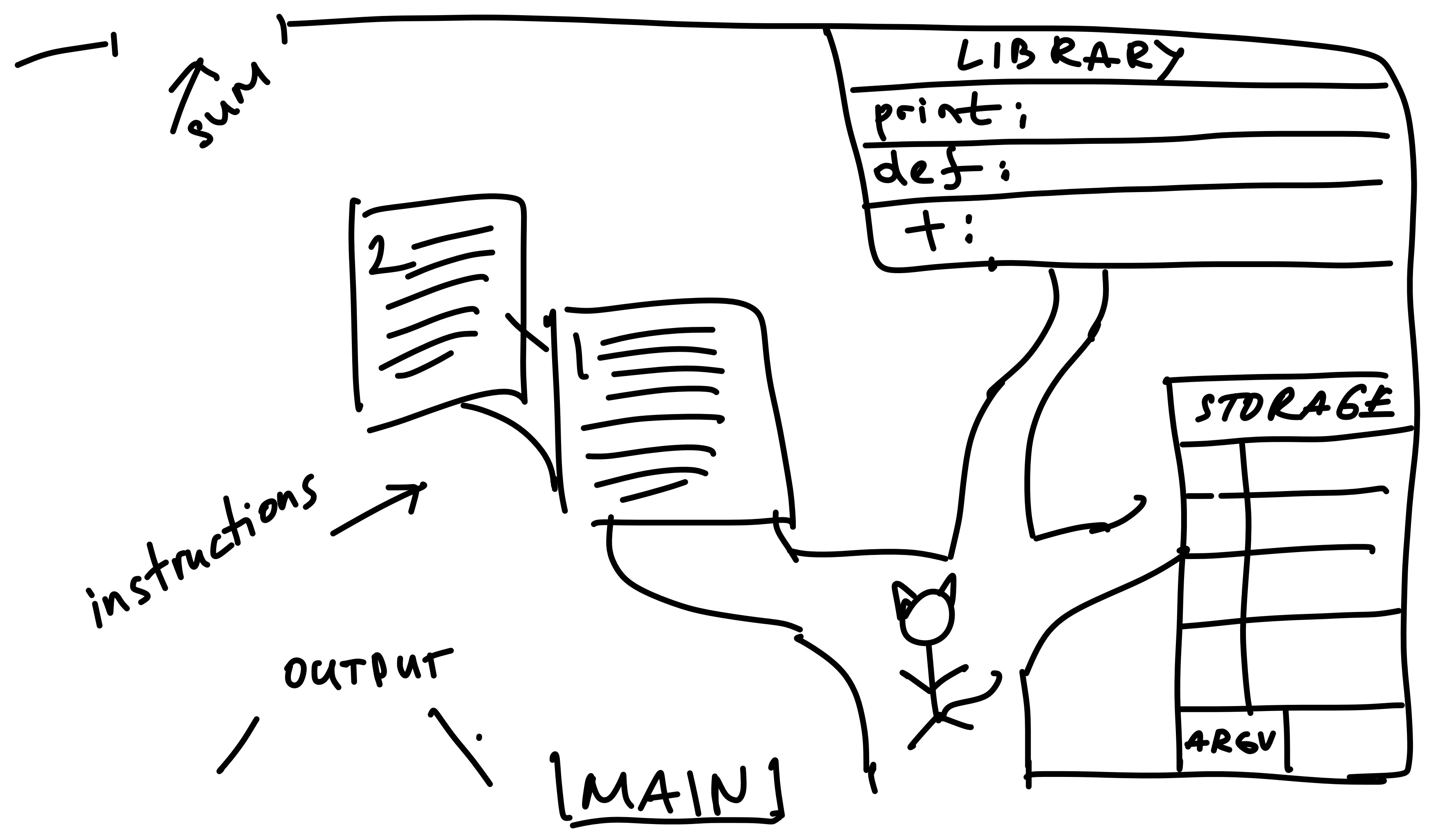
- Goals:
  - Continue practicing the tools
  - Write a small program
    - Open a file, read in text
    - Clean up the text and tokenize it
    - Count tokens
    - Based on simplistic logic, predict whether review is POS or NEG
- We will cover all of these topics by 04/20
  - Today: Conditionals (need to predict POS/NEG)

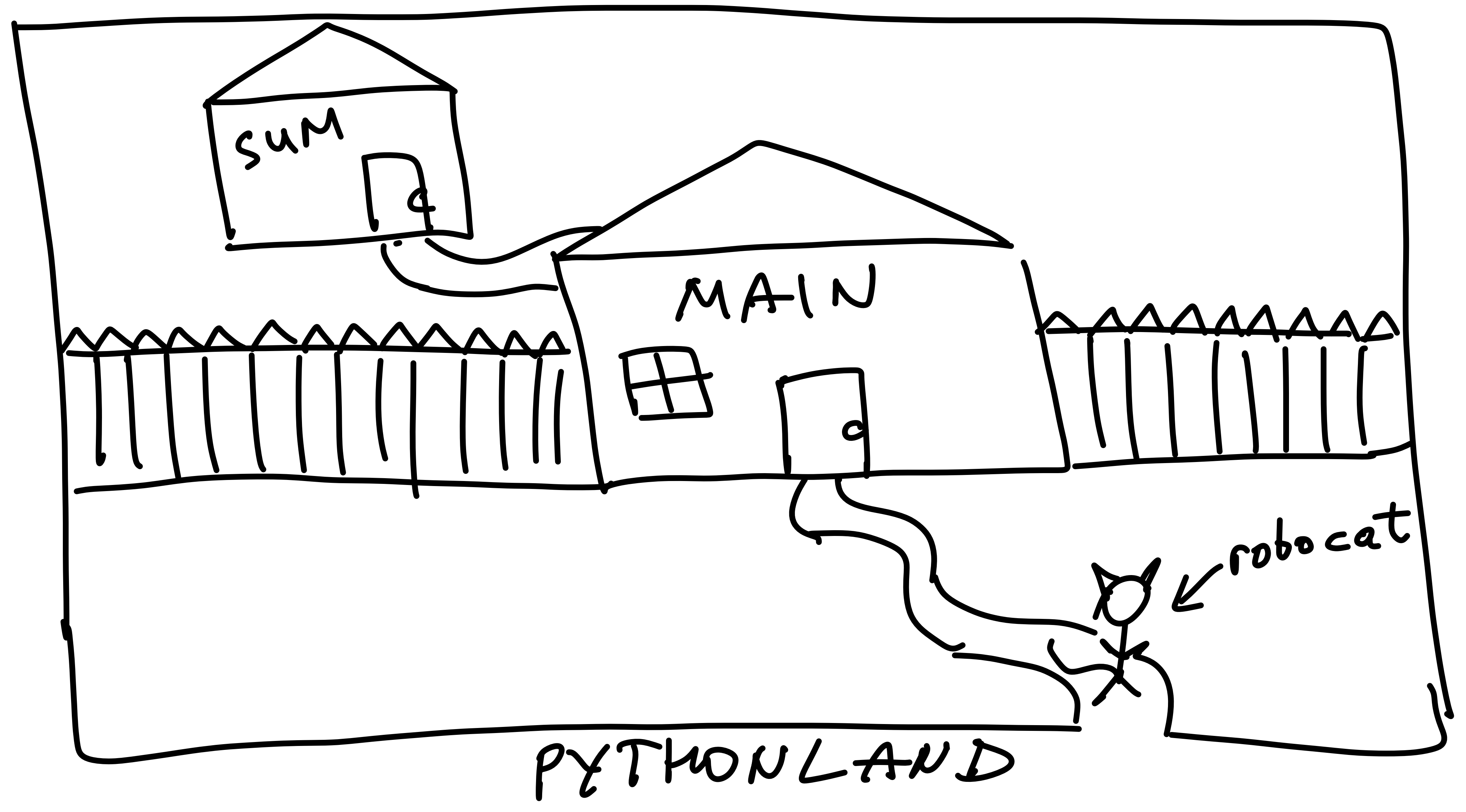
Plan for today

- Review:
  - Variables and assignment
- New concepts:
  - Scope
  - Functions
  - Control flow
- Methodology:
  - Look at **concepts** first
  - Then learn the specific **syntax** by looking at the code

Programming
Metaphors

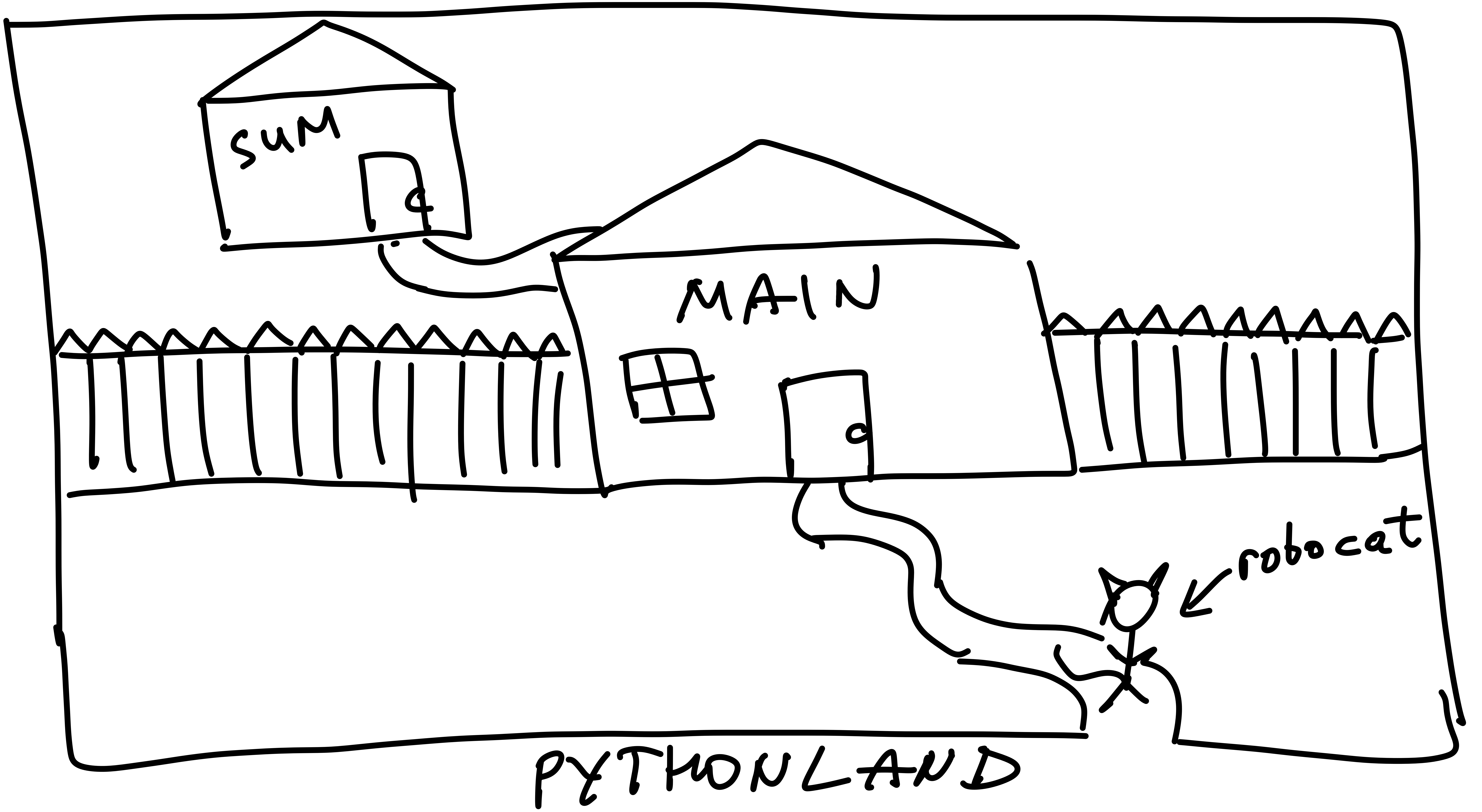
- Imagine a Robocat
    - The Robocat visits Pythonland where there is only one **etnrance** called **Main**
    - In the building, there are **instructions**, a library, **labeled boxes**, an "**output**" window, and **a door to another building** called "Sum".
    - Robocat can only follow instructions or go to library
    - Labeled boxes can **contain** things, but things go in and out only in some cases (**assignment**)
    - Otherwise, Robocat can only copy things labeled boxes contain

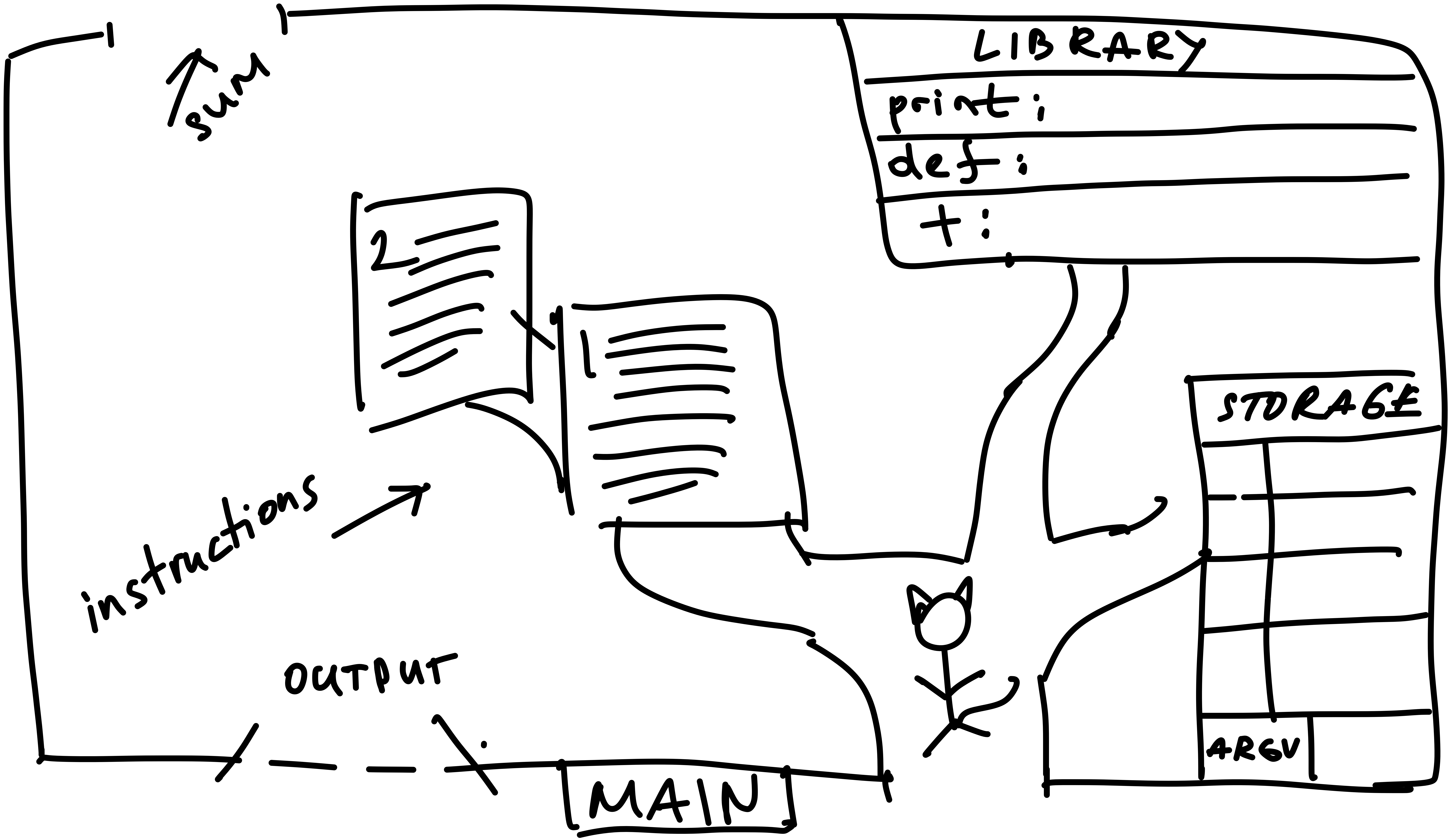


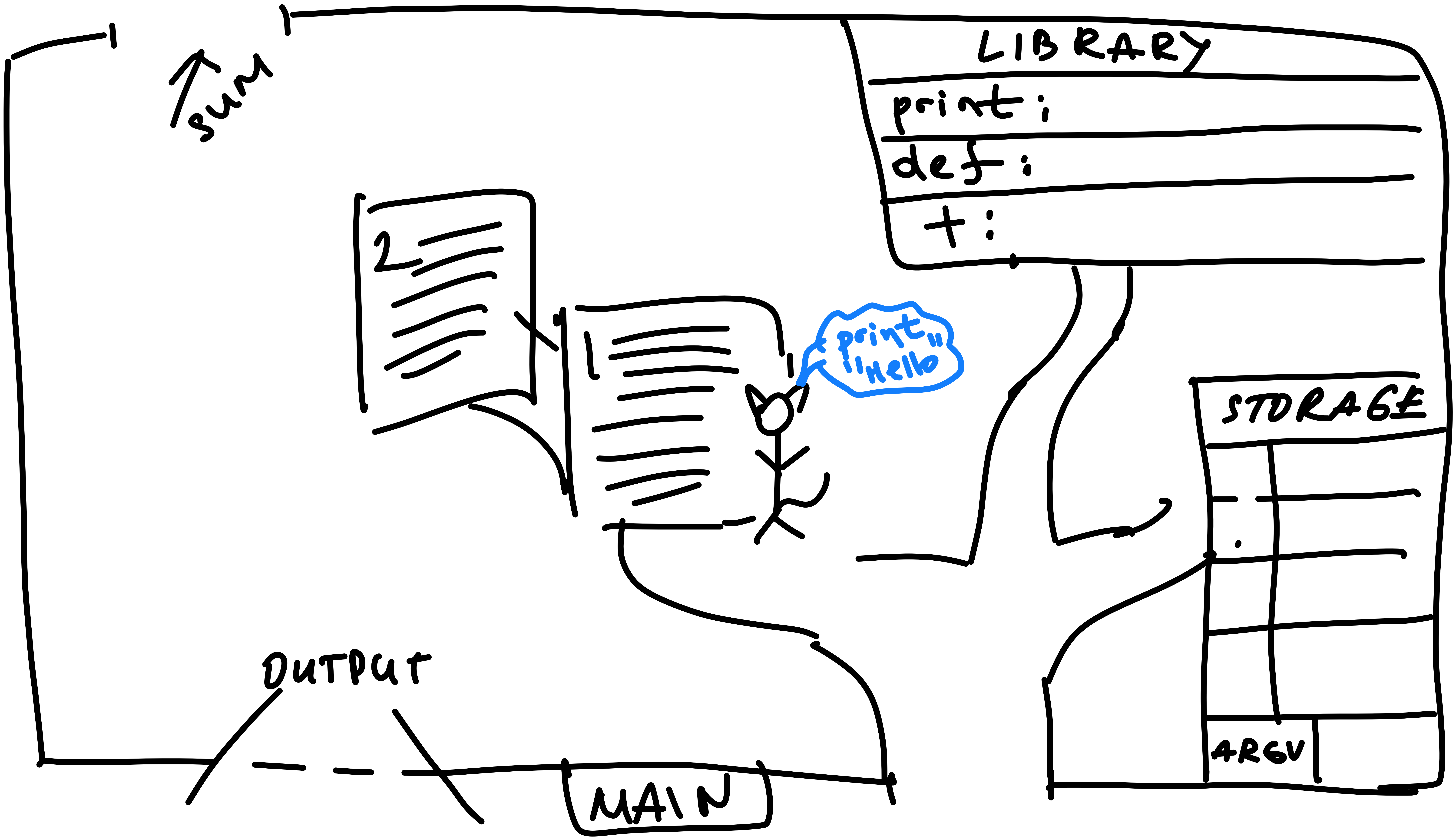


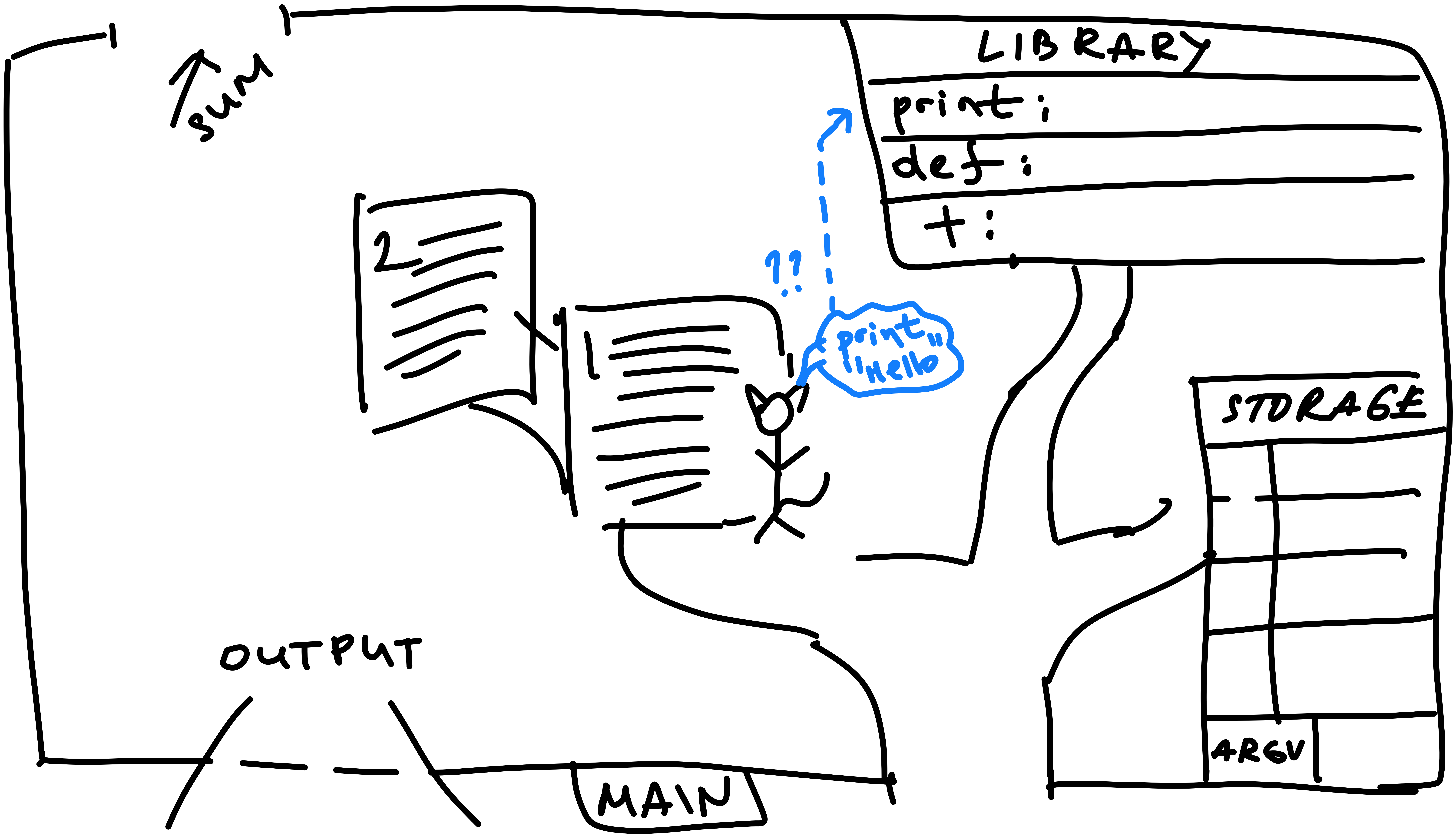
Warning/Disclaimer: Metaphors can help but they can also mislead!

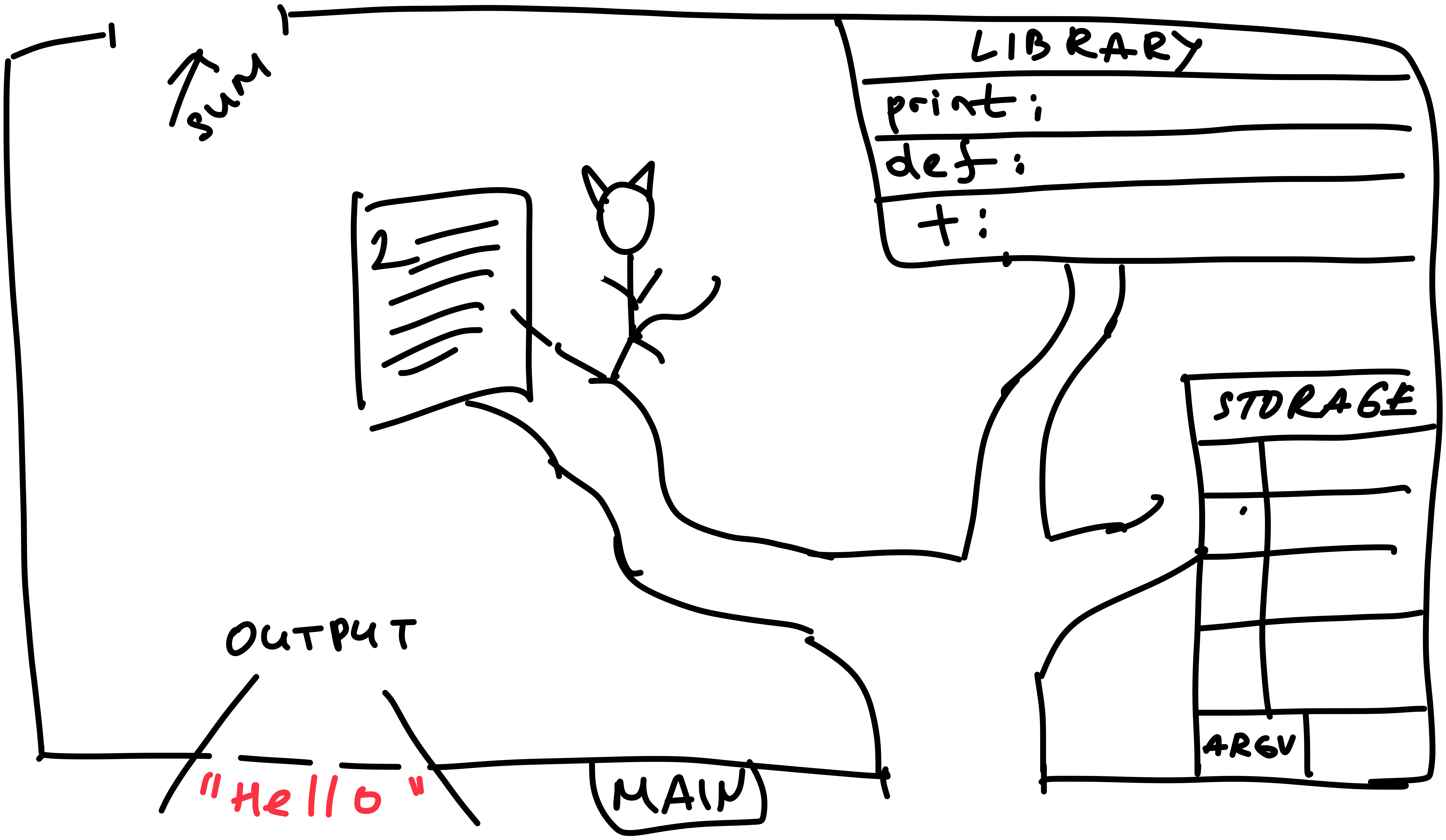
(There may be bugs in metaphors :))


SUM
MAIN
robocat
PYTHONLAND


sum
LIBRARY
print;
def;
+;
2
1
instructions
STORAGE
ARGV
OUTPUT
MAIN


SUM
LIBRARY
print;
def;
+;
2
1
print "Hello"
STORAGE
ARGV
OUTPUT
MAIN

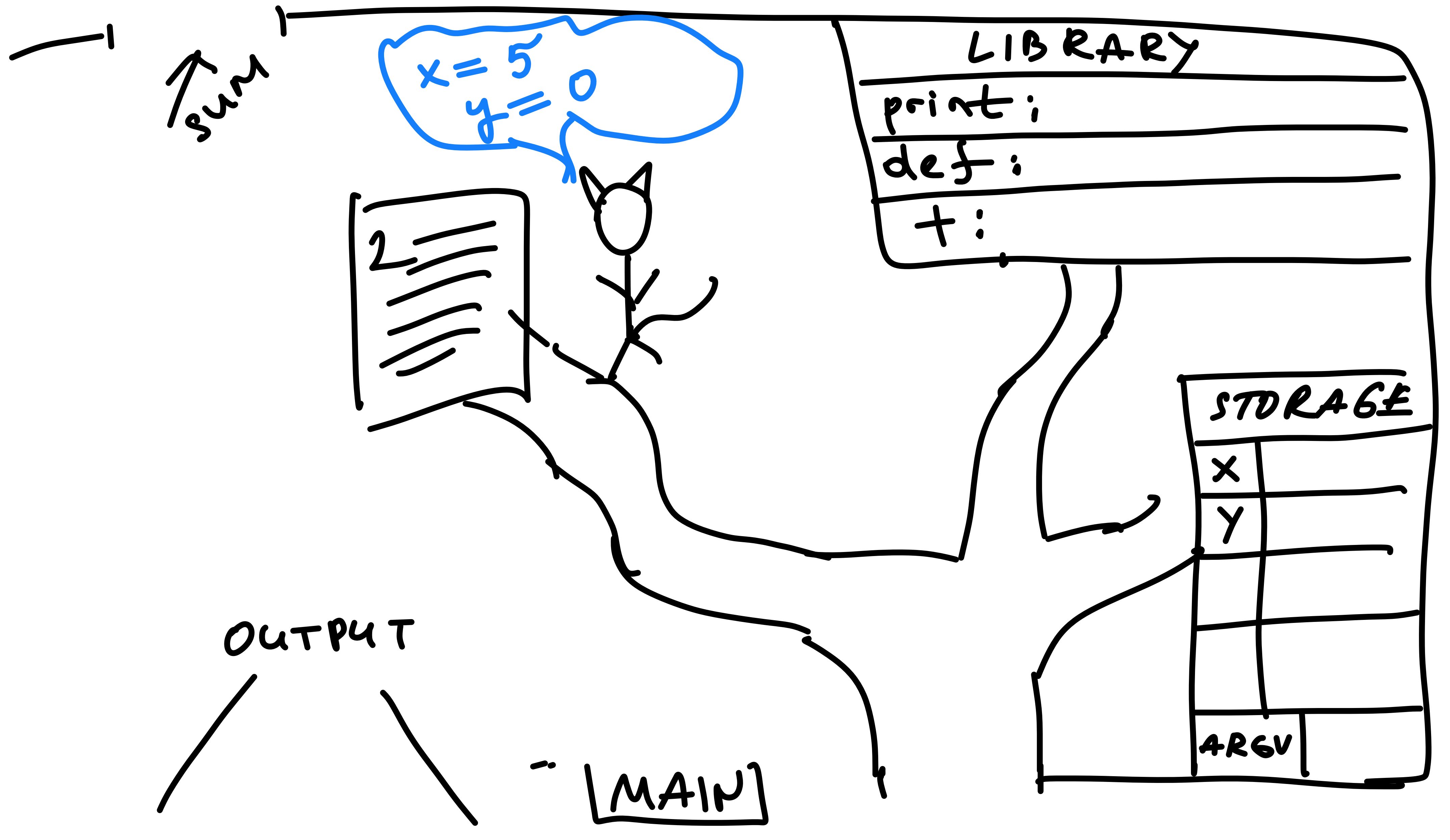

SUM
LIBRARY
print;
def;
+;
2
1
??
print "Hello"
STORAGE
ARGV
OUTPUT
MAIN

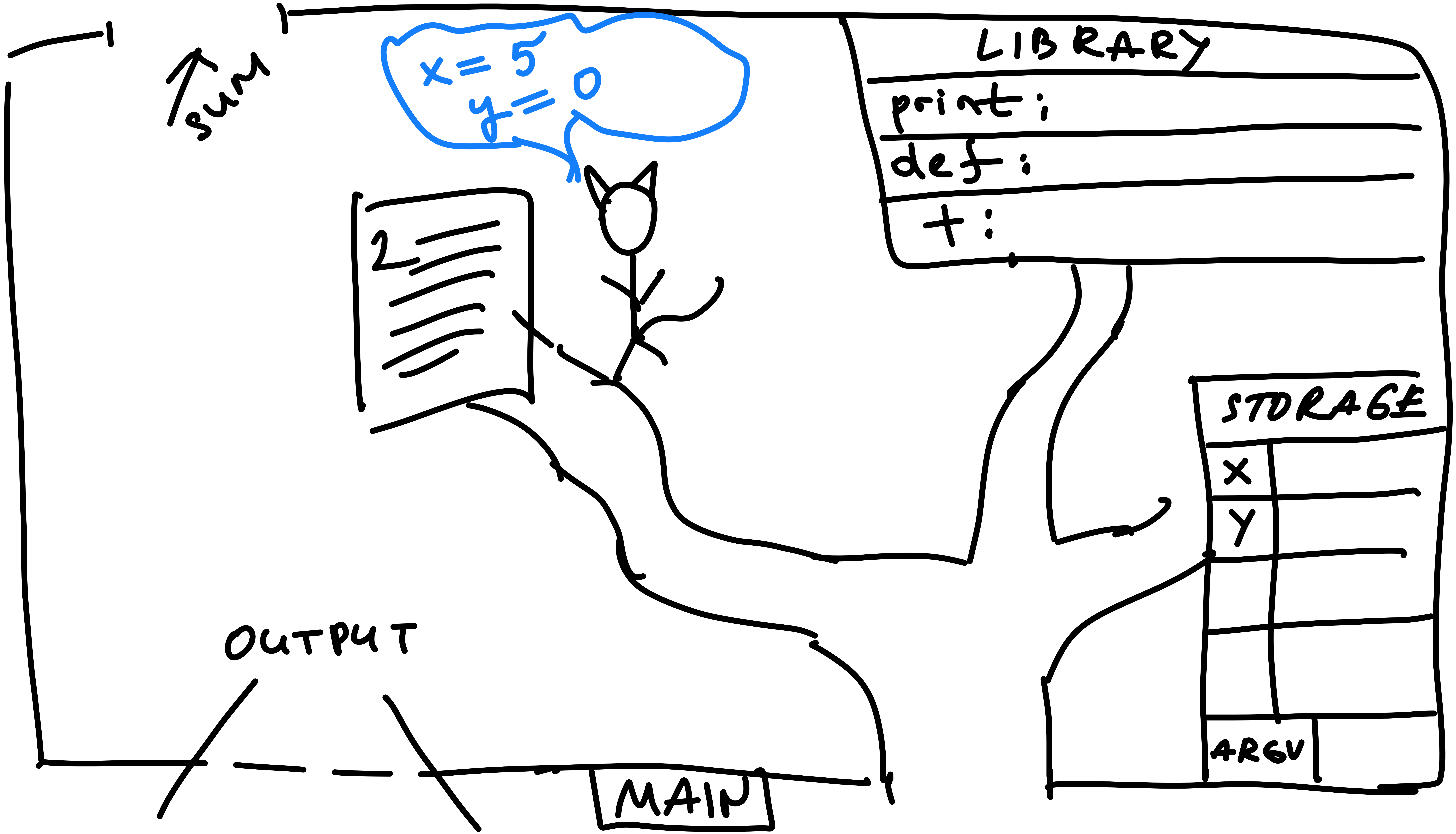

sum
LIBRARY
print;
def;
+;
2
STORAGE
ARGV
OUTPUT
"Hello"
MAIN

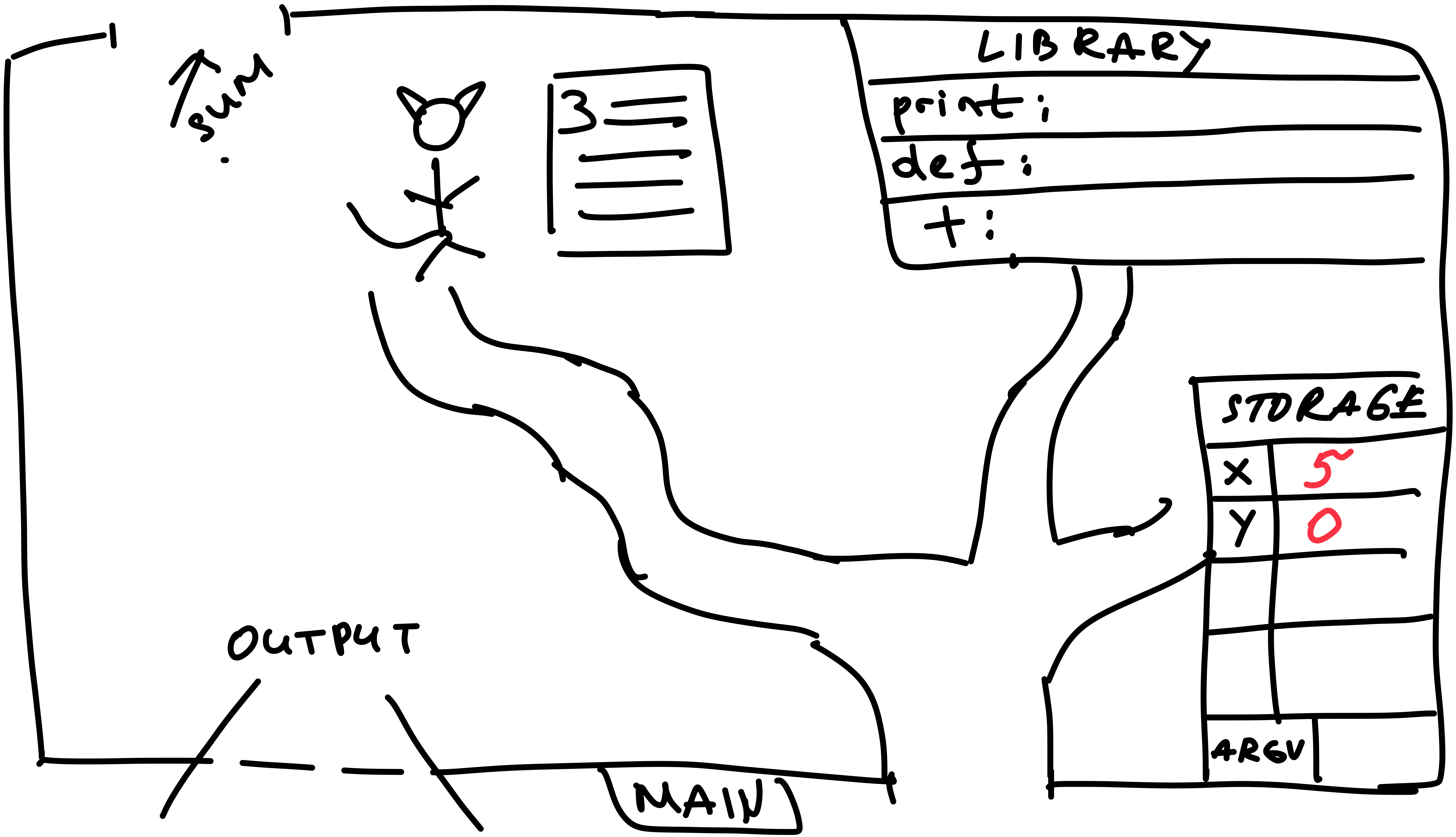
Variables

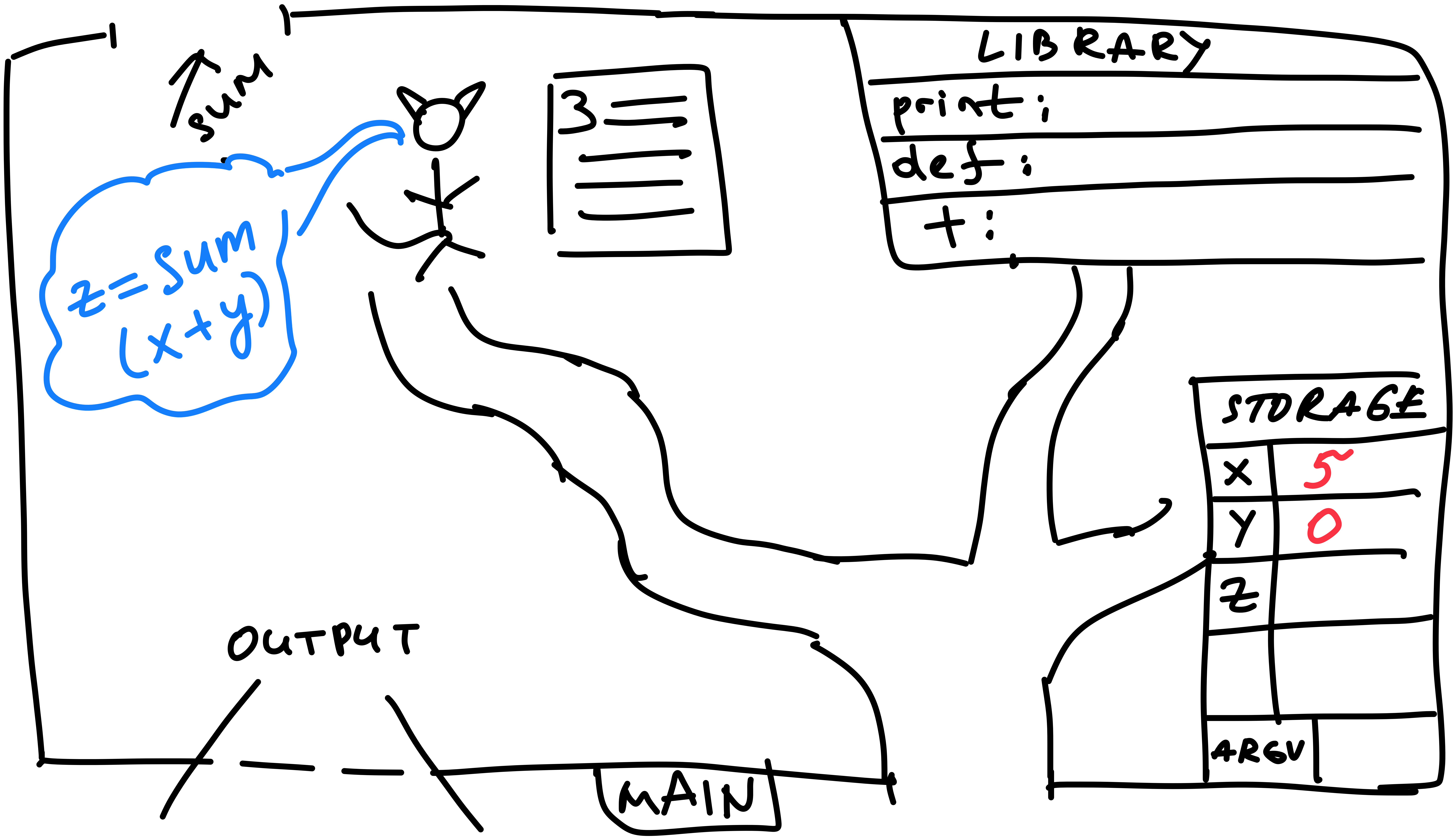
Scope

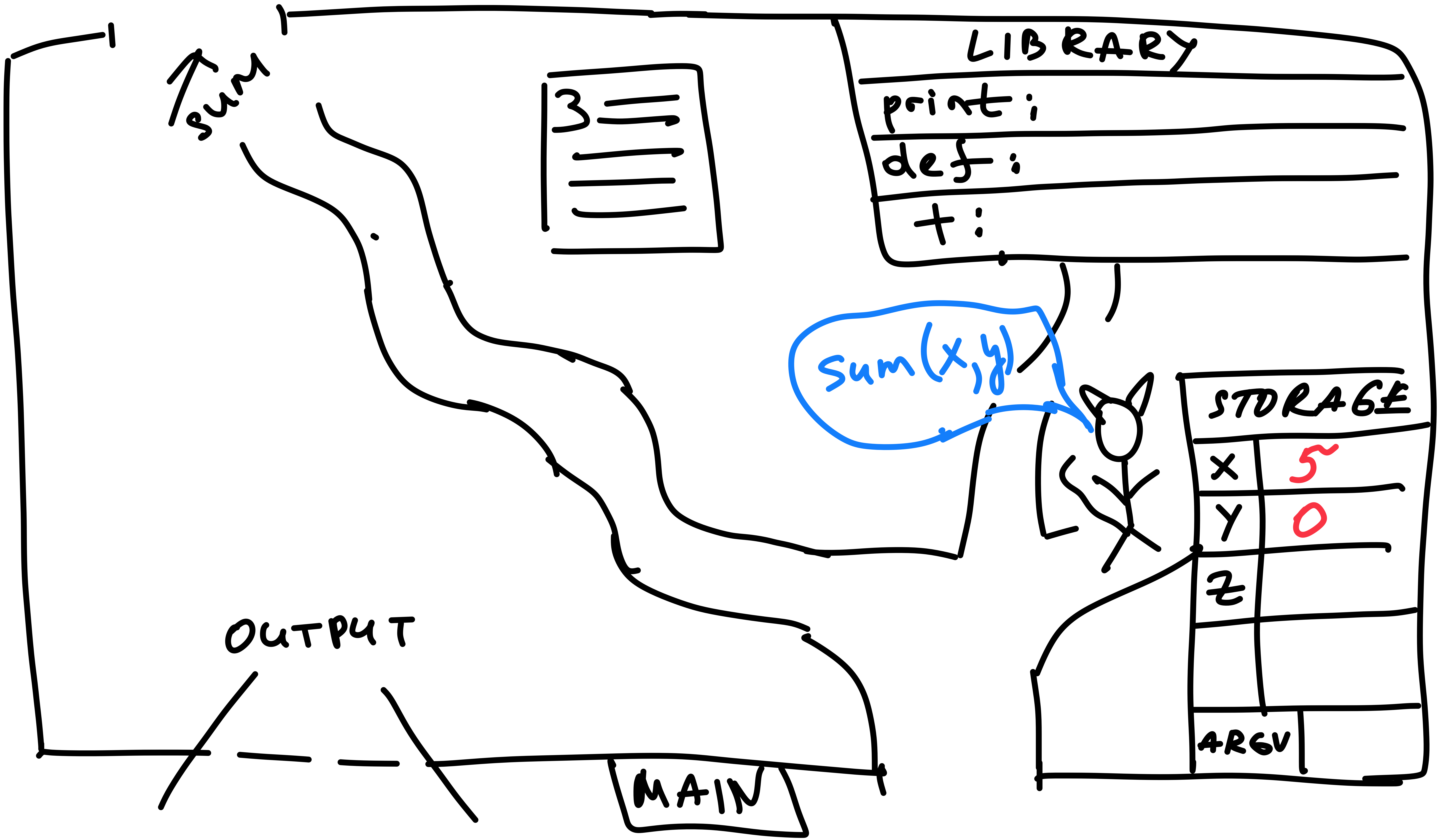
- Variables are locations in memory
    - Variables have **names**
- **Values** can be **stored** under those names
    - e.g. x = 5
        - **x** is the **name**, 5 is the **value**
        - **Each** storage allows only unique names! (e.g. one **x**)
- Variable names have **scope**
    - Can have more than one x in the same program
        - If separate scopes exist, like separate “buildings”/“storages”
        - e.g. there are different functions
        - Once in a function, the scope is specific to the function

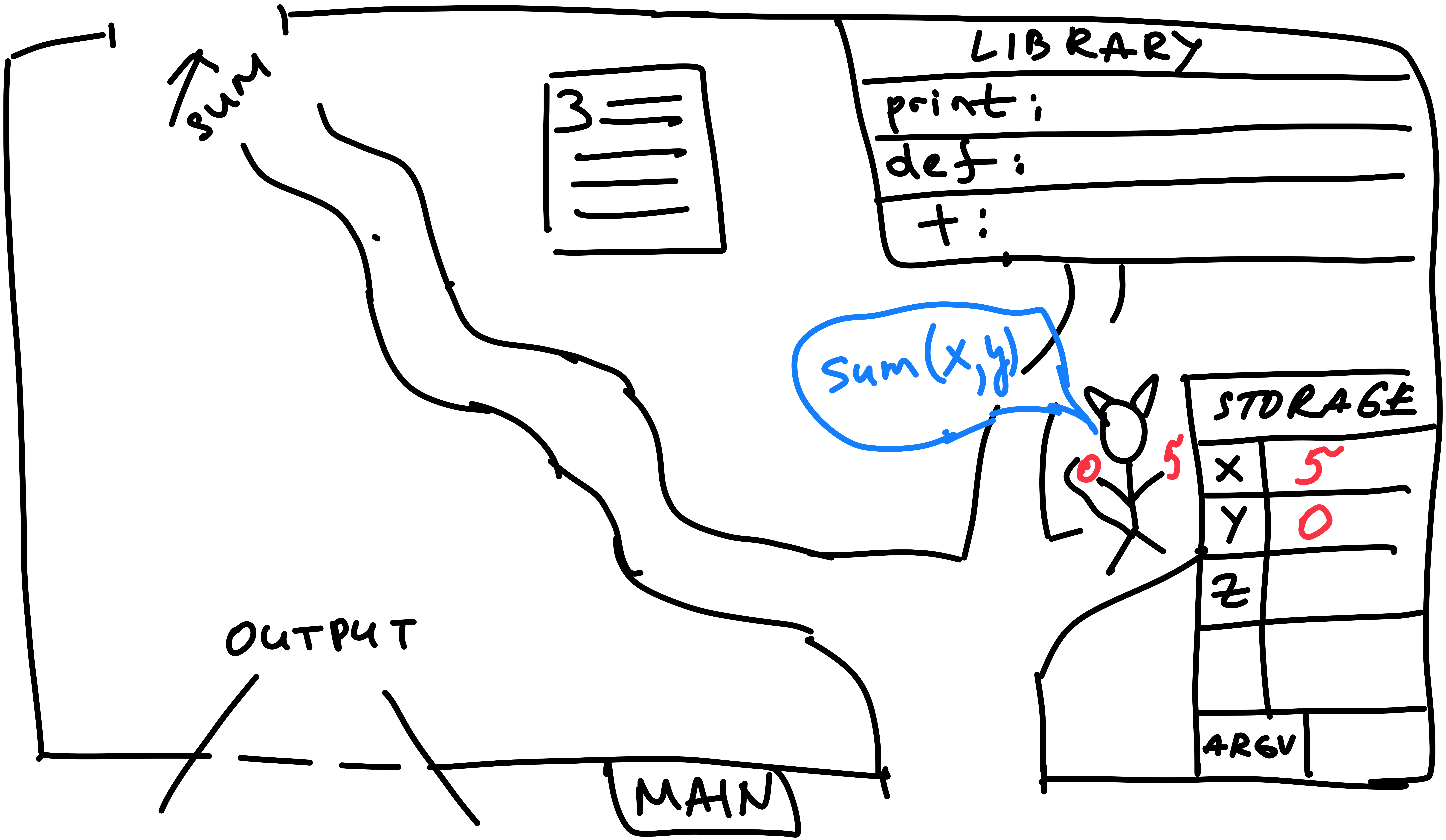


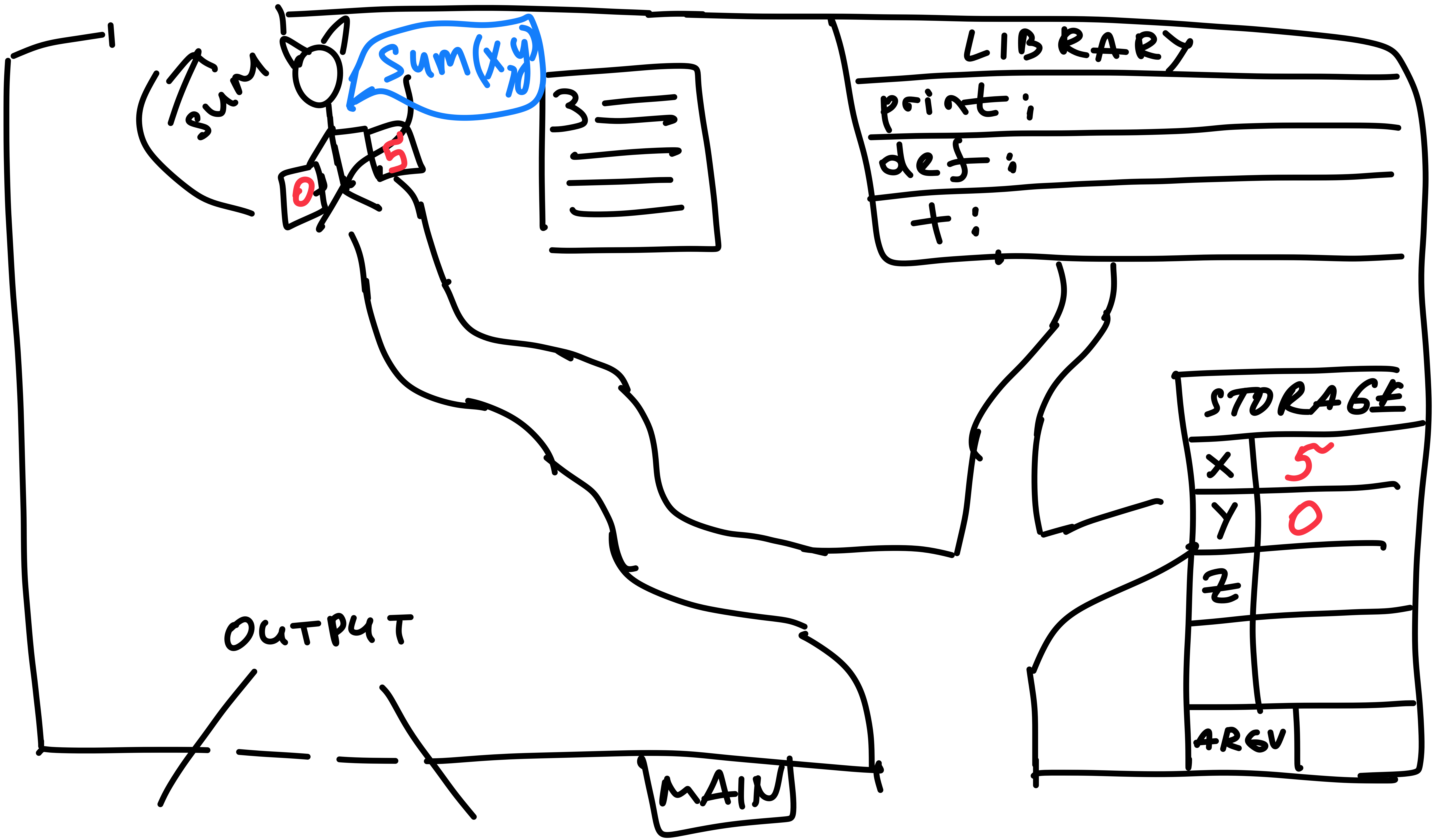

SUM
x = 5
y = 0
2
LIBRARY
print;
def;
+;
STORAGE
x
y
ARGV
OUTPUT
MAIN

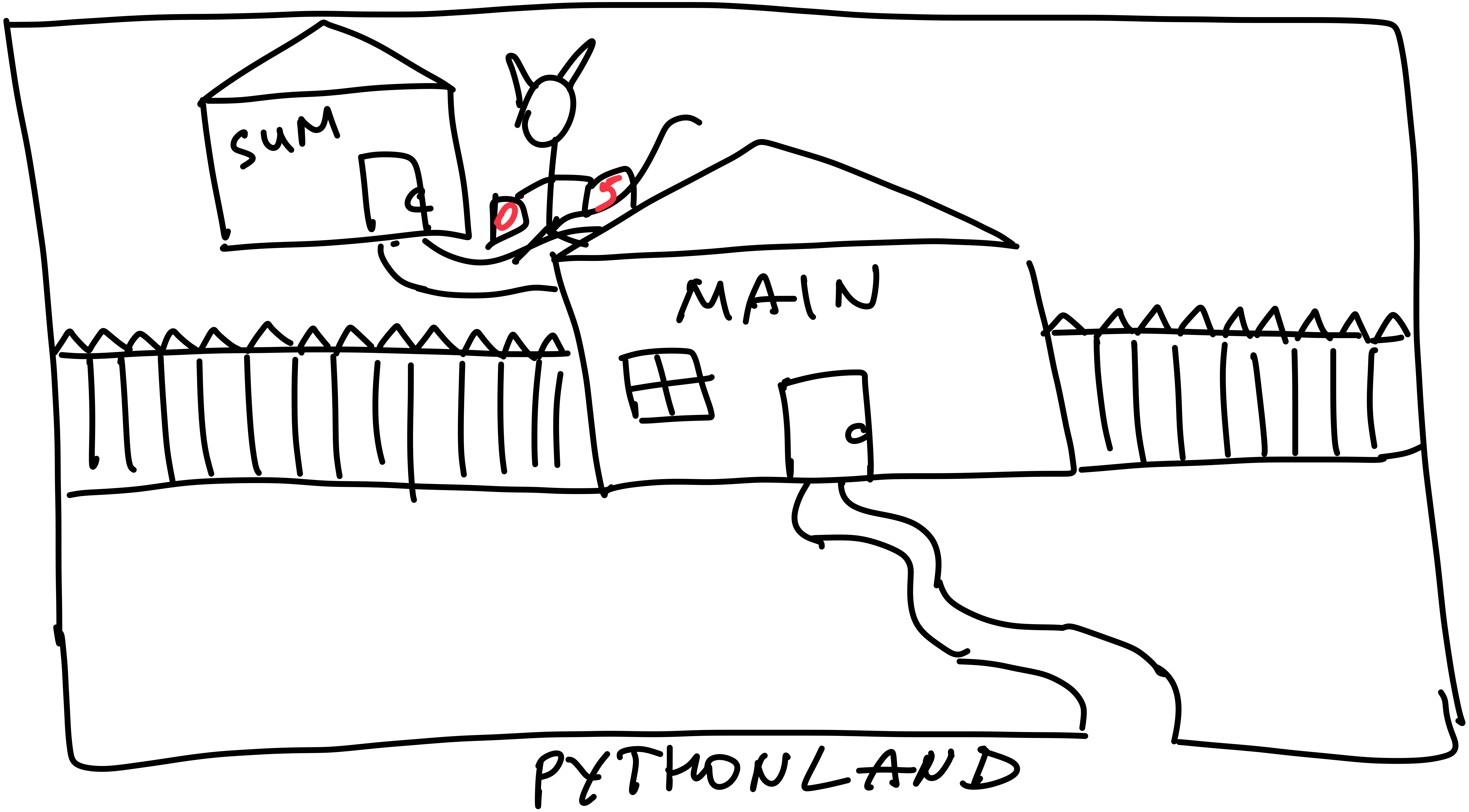

SUM
3
LIBRARY
print :
def :
+ :
STORAGE
X 5
Y 0
ARGV
OUTPUT
MAIN


SUM
z=sum(x+y)
3
LIBRARY
print:
def:
+:
STORAGE
X 5
Y 0
Z
ARGV
OUTPUT
MAIN


sum
3
LIBRARY
print;
def;
+:
sum(x,y)
STORAGE
x 5
y 0
z
ARGV
OUTPUT
MAIN

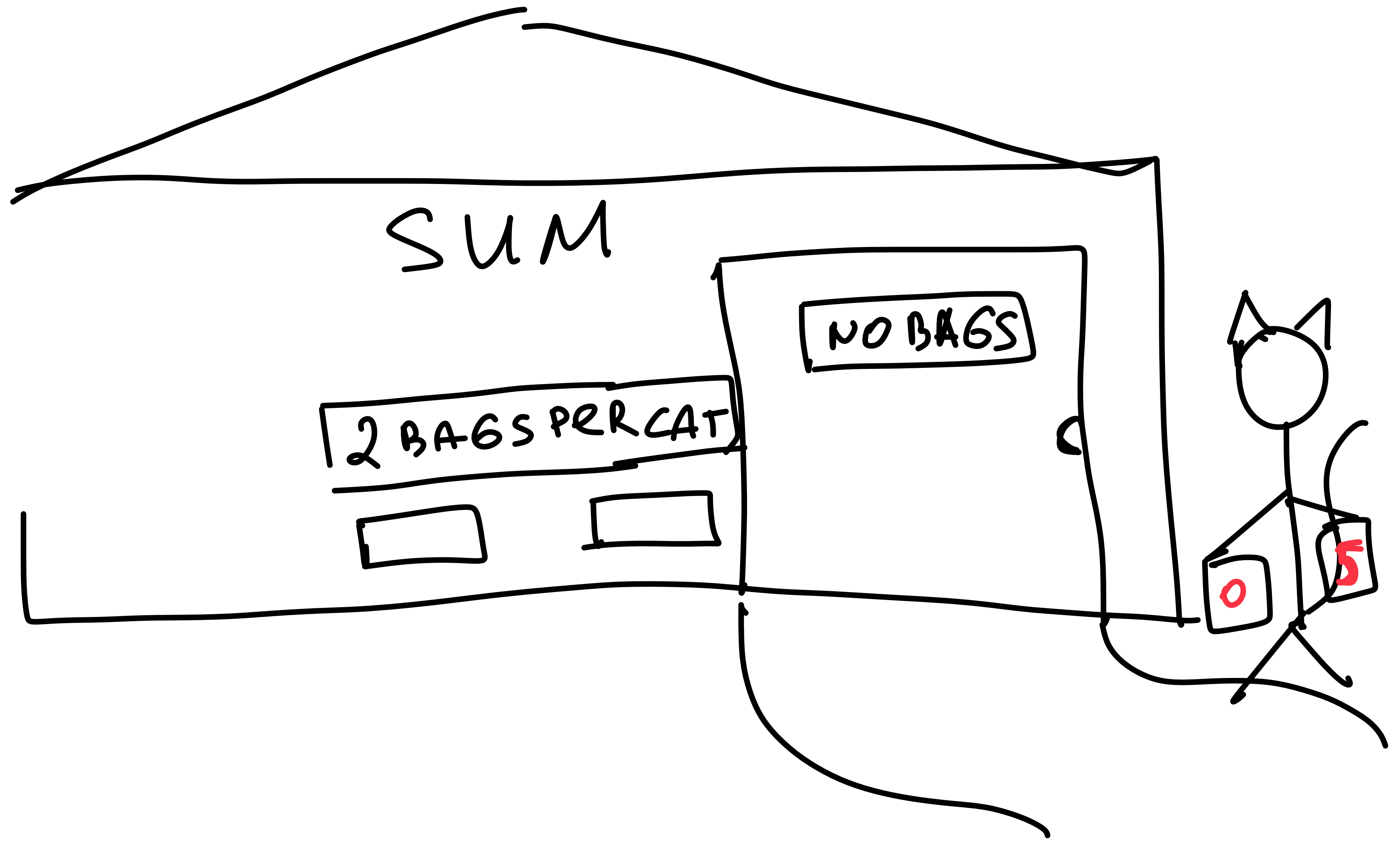

SUM
3
LIBRARY
print:
def:
+:
sum(x,y)
0
5
STORAGE
X
5
Y
0
Z
ARGV
OUTPUT
MAIN

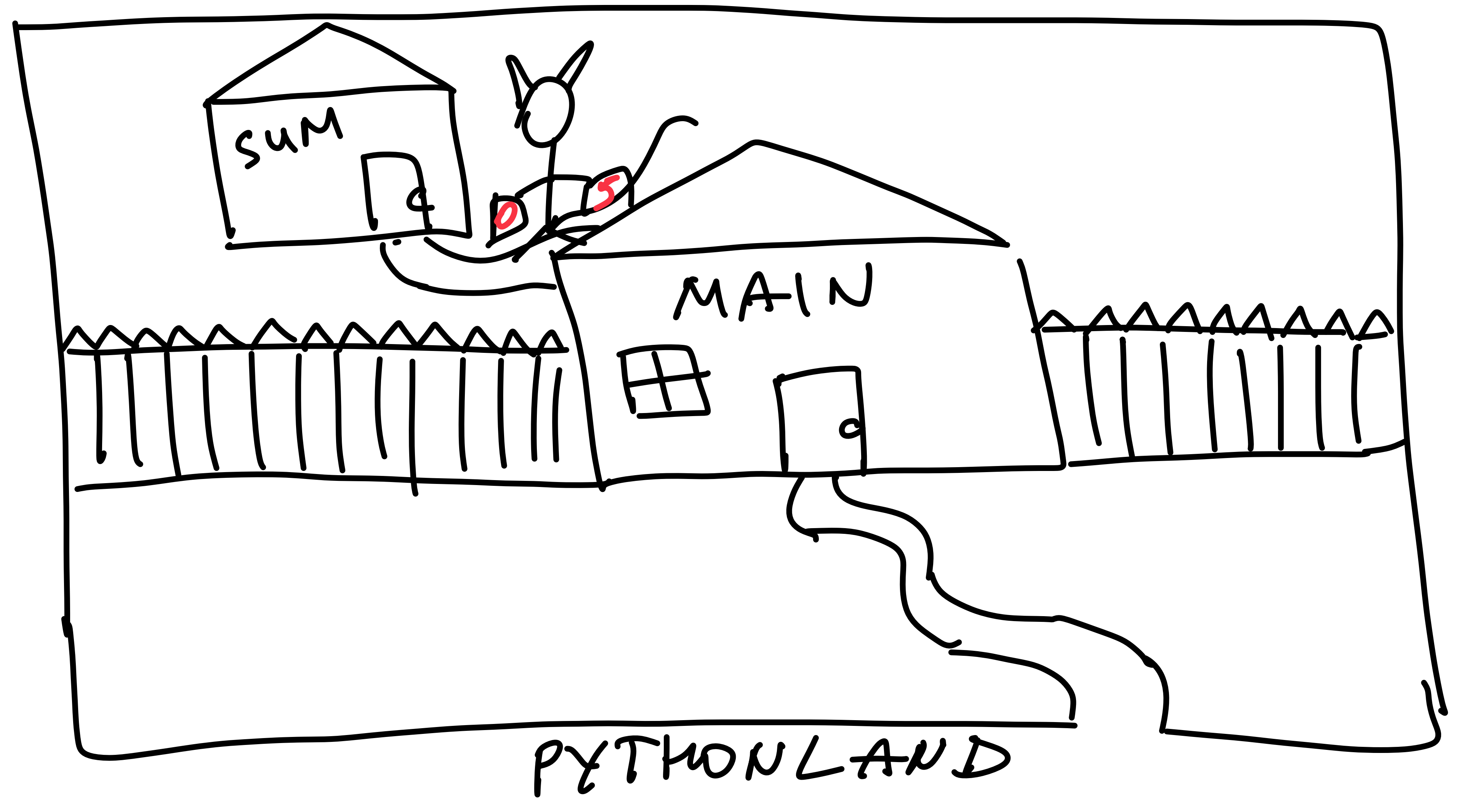

SUM
sum(x,y)
3
0
5
LIBRARY
print;
def;
+;
STORAGE
x 5
y 0
z
ARGV
OUTPUT
MAIN

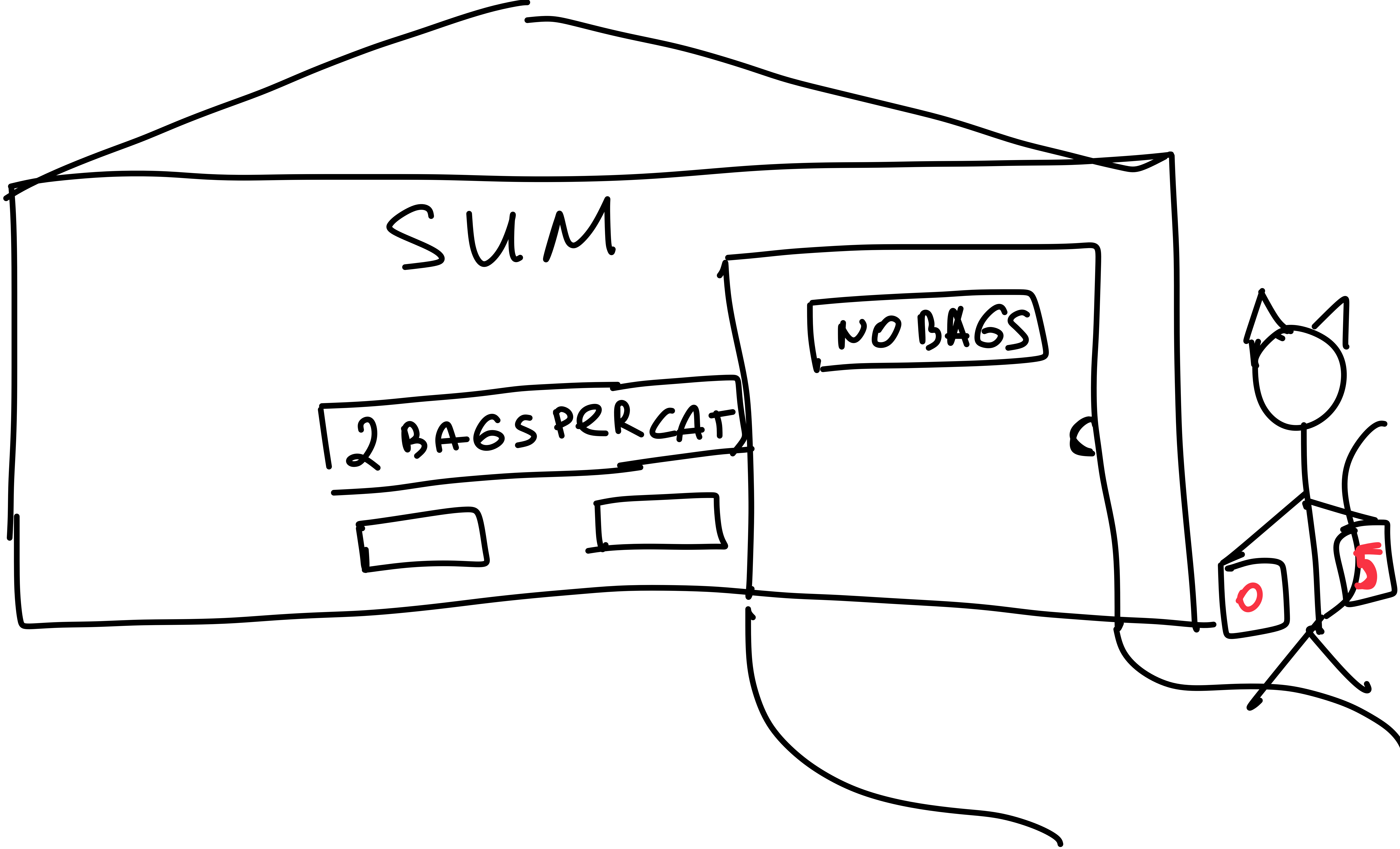

SUM
0
5
MAIN
PYTHONLAND

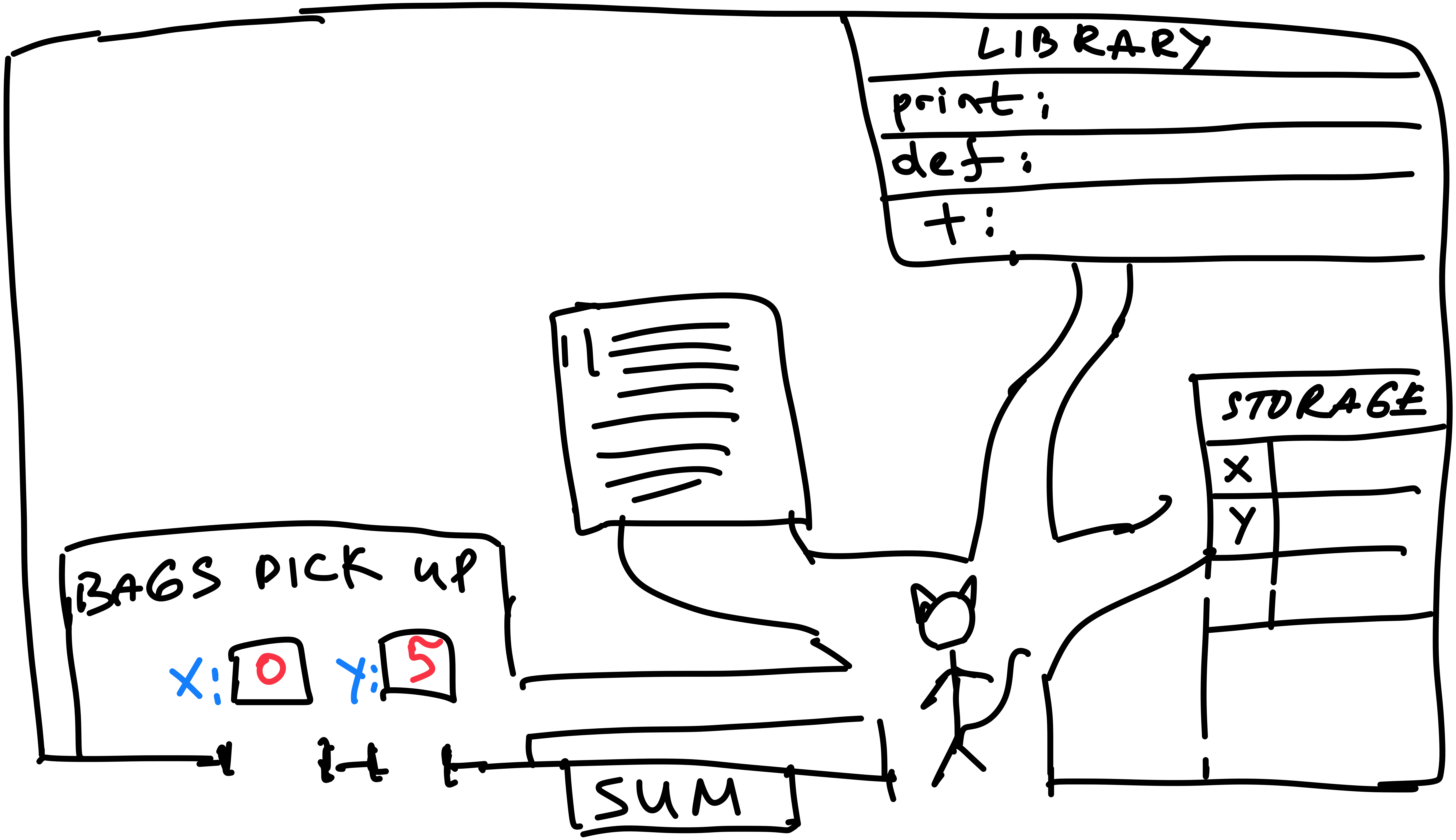
Arguments
of functions

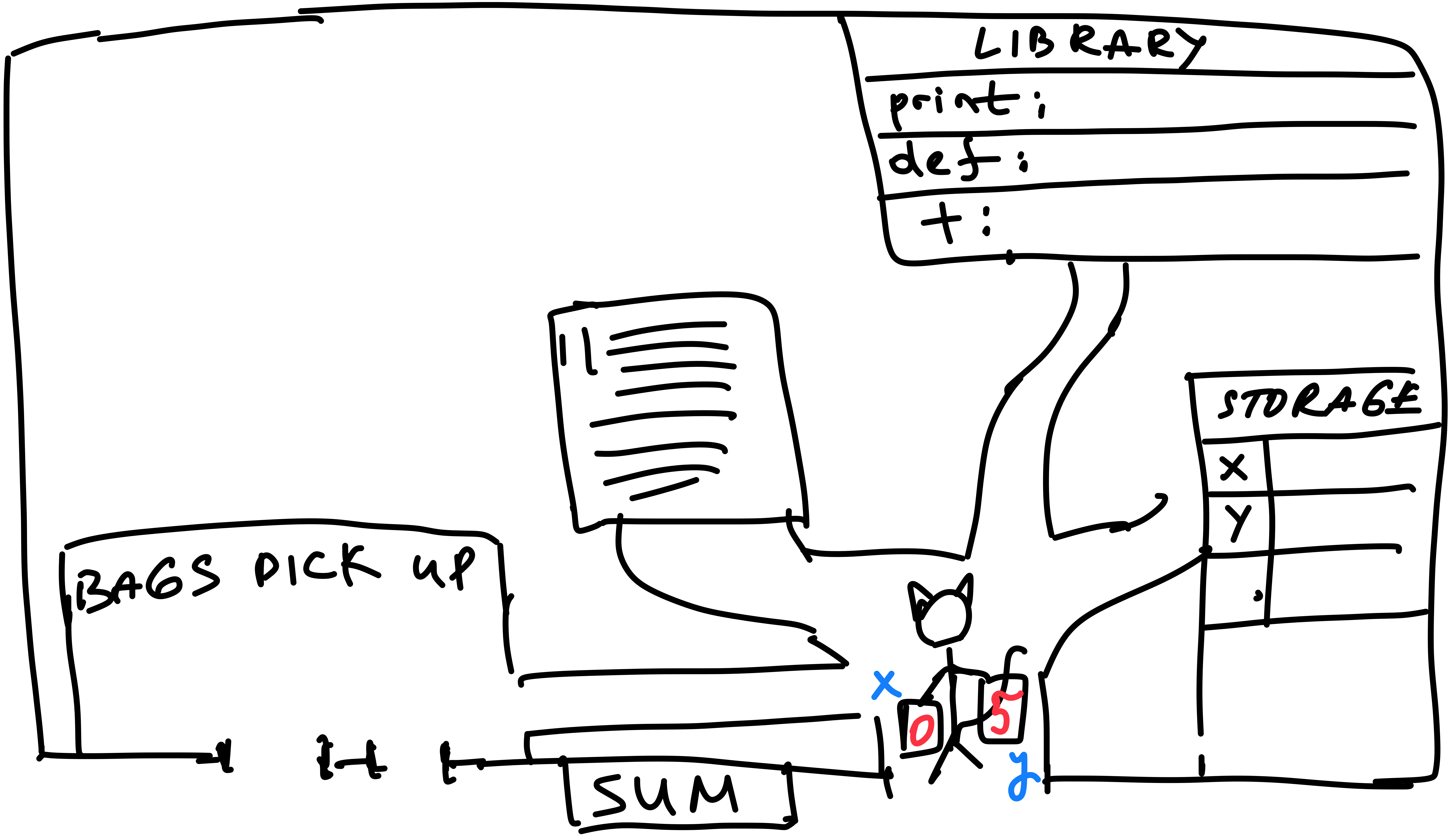
- Functions have **parameters** which are realized in the form of **arguments**
- All functions define **exactly how many** parameters they have (how many args can be passed) and in **which order** they come
  - Including the Main function!
  - main() has only one argument:
    - named **argv**
    - ...which happens to be a **list** (array) of things!

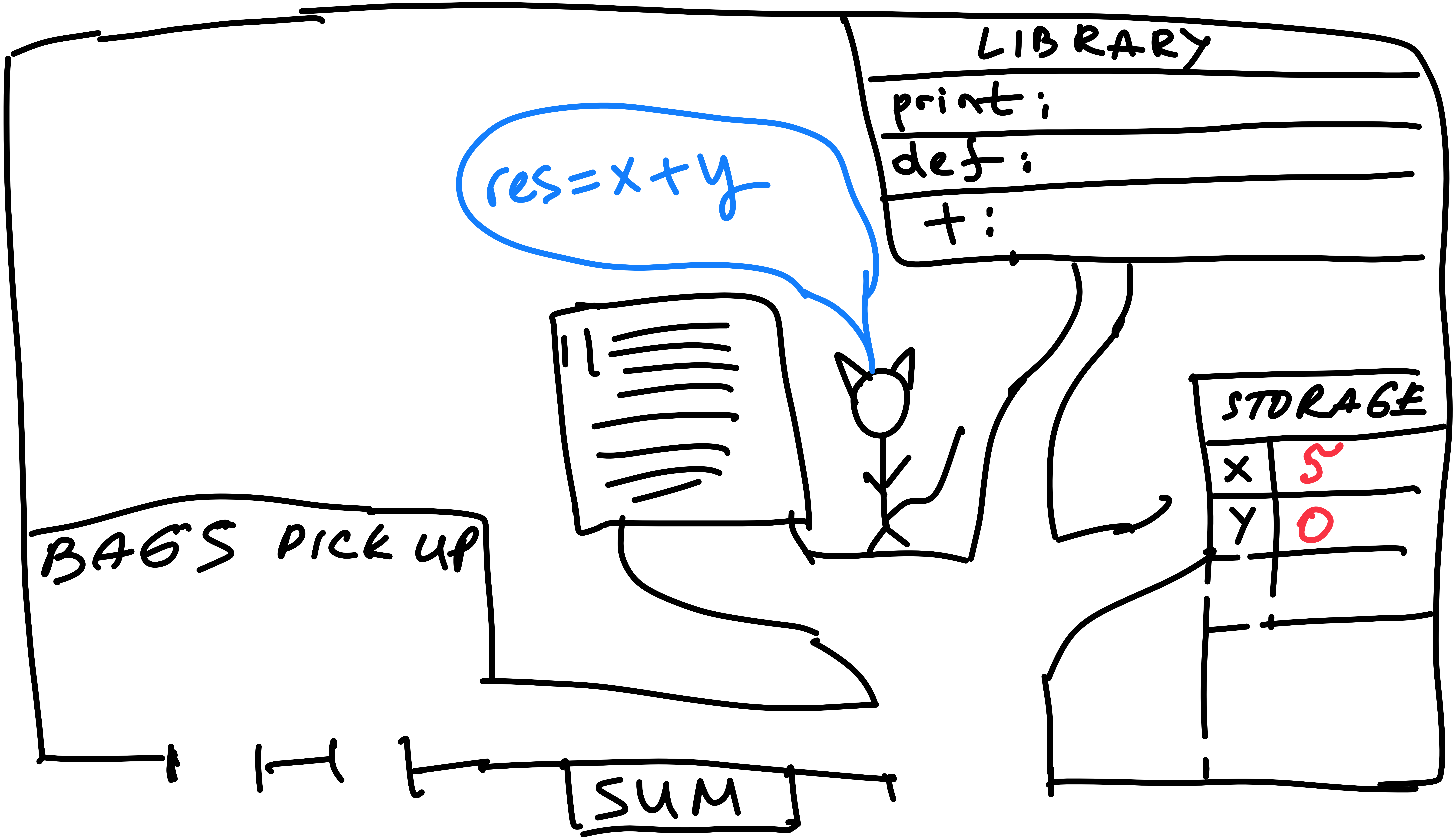


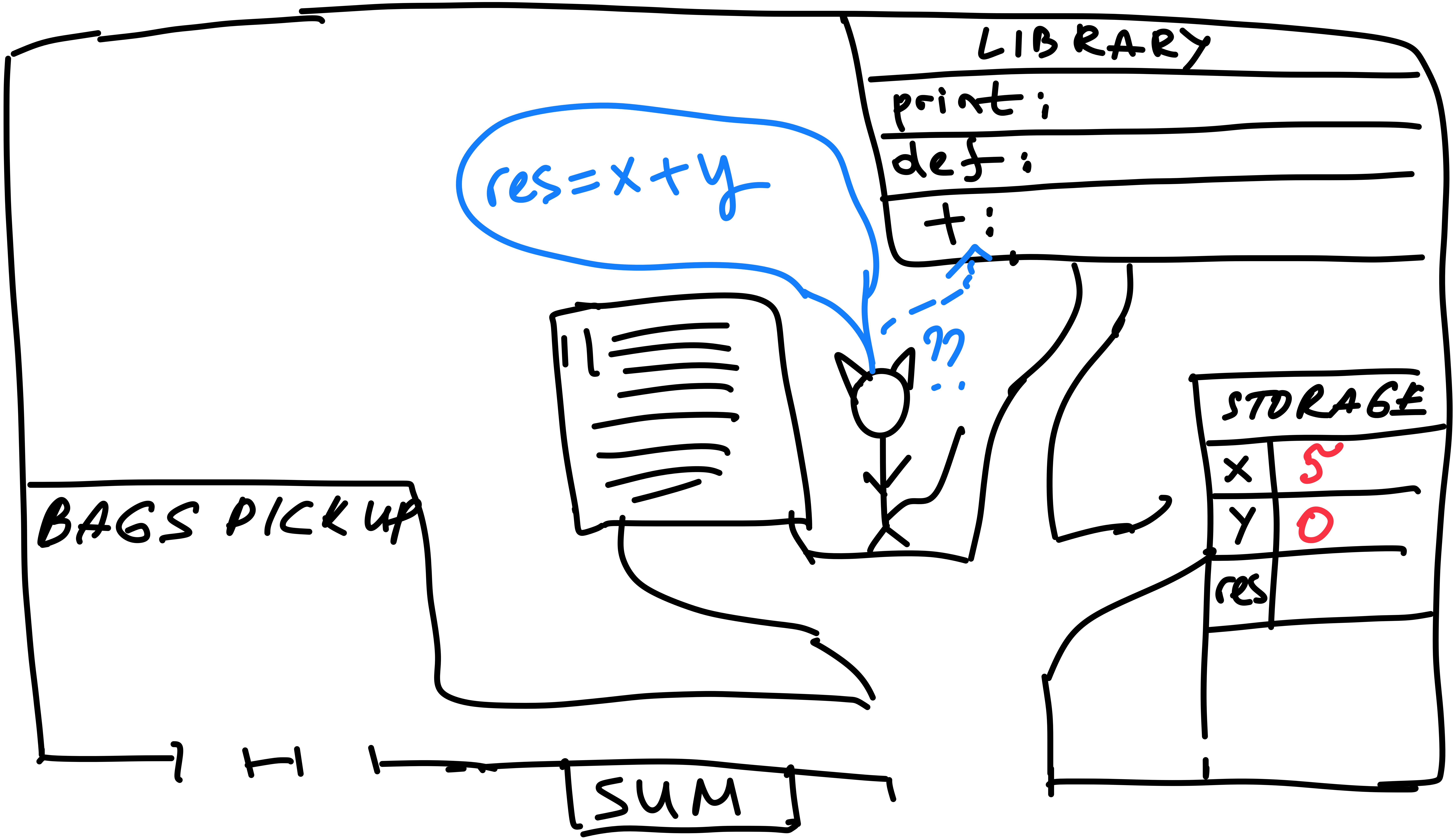


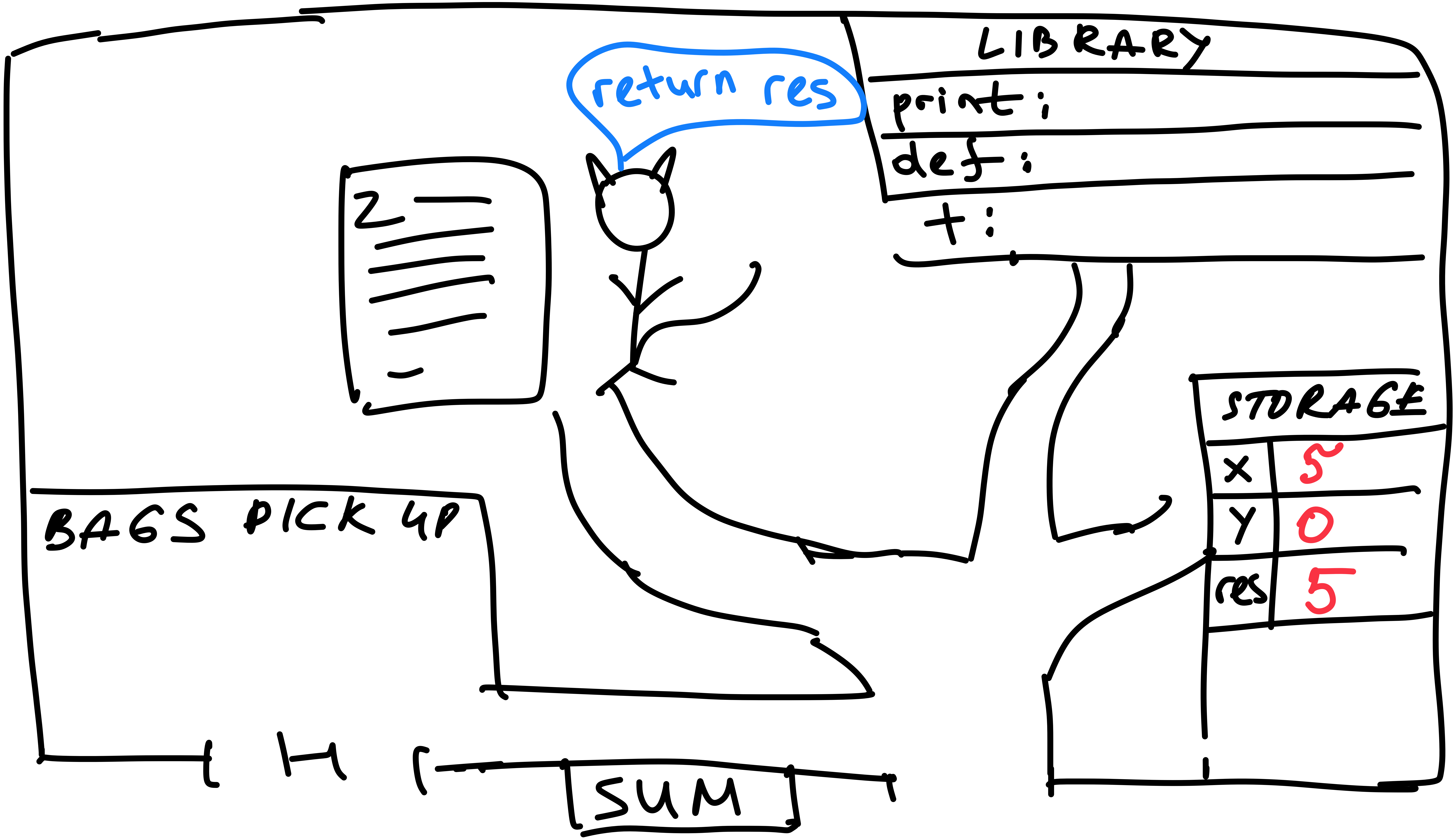

SUM
2 BAGS PER CAT
NO BAGS
0
5

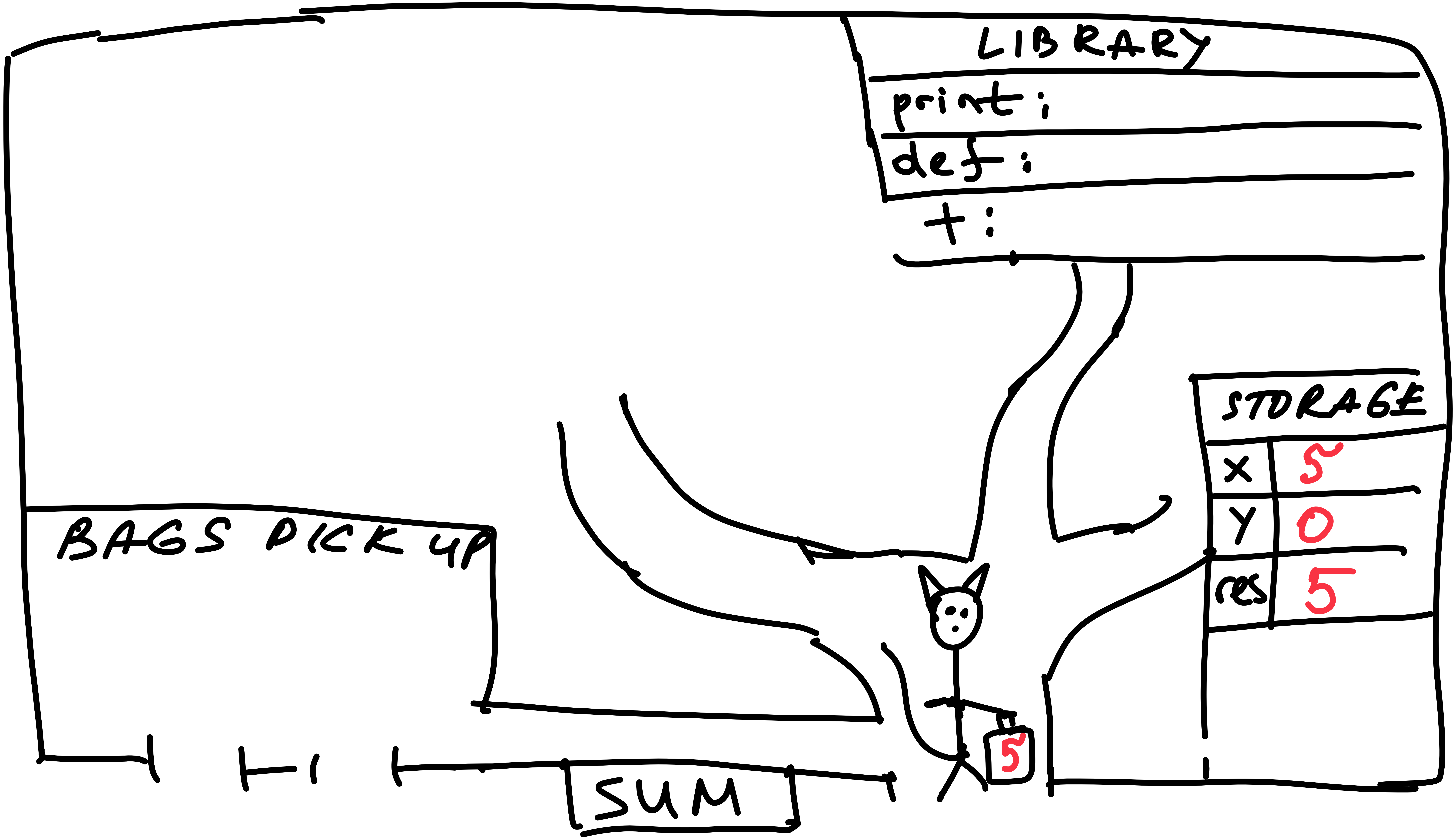

LIBRARY
print:
def:
+:
STORAGE
x
y
BAGS PICK UP
x: 0
y: 5
SUM

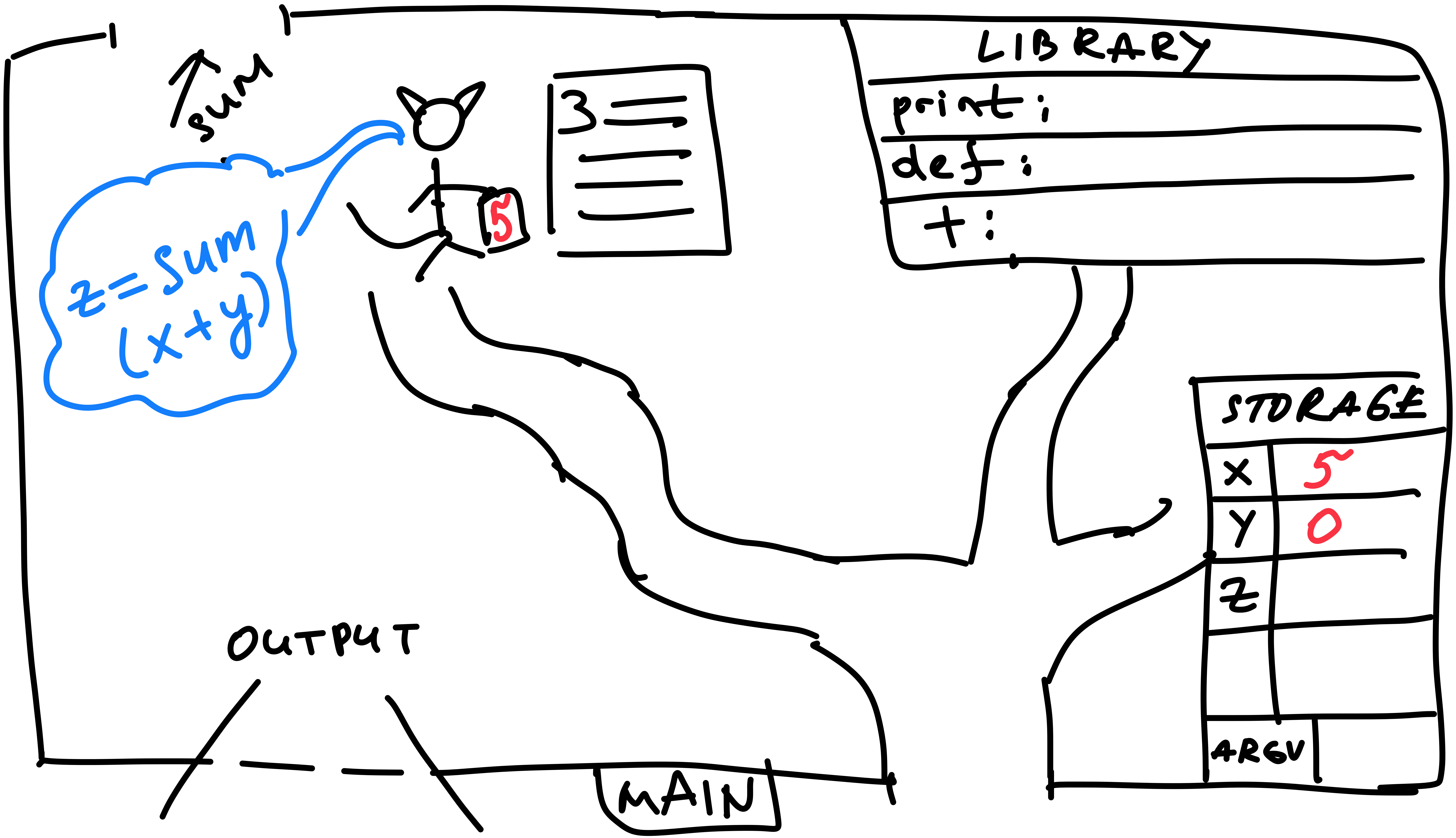

LIBRARY
print:
def:
+:
STORAGE
x
y
BAGS PICK UP
SUM
x
0
5
y

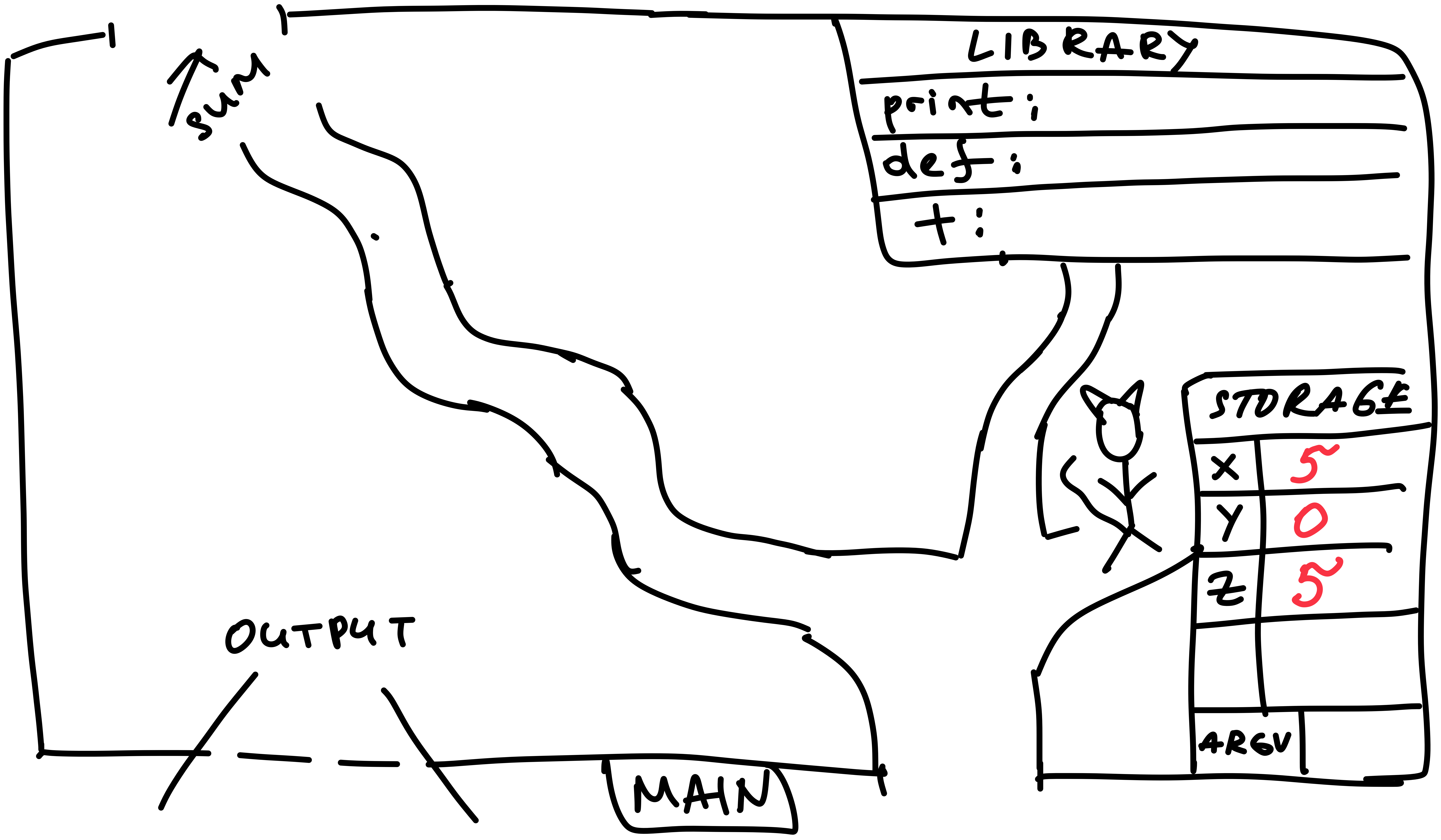

LIBRARY
print;
def;
+;
res=x+y
STORAGE
x 5
y 0
BAGS PICK UP
SUM


LIBRARY
print;
def;
+;
STORAGE
x 5
y 0
res
res=x+y
??
BAGS PICKUP
SUM

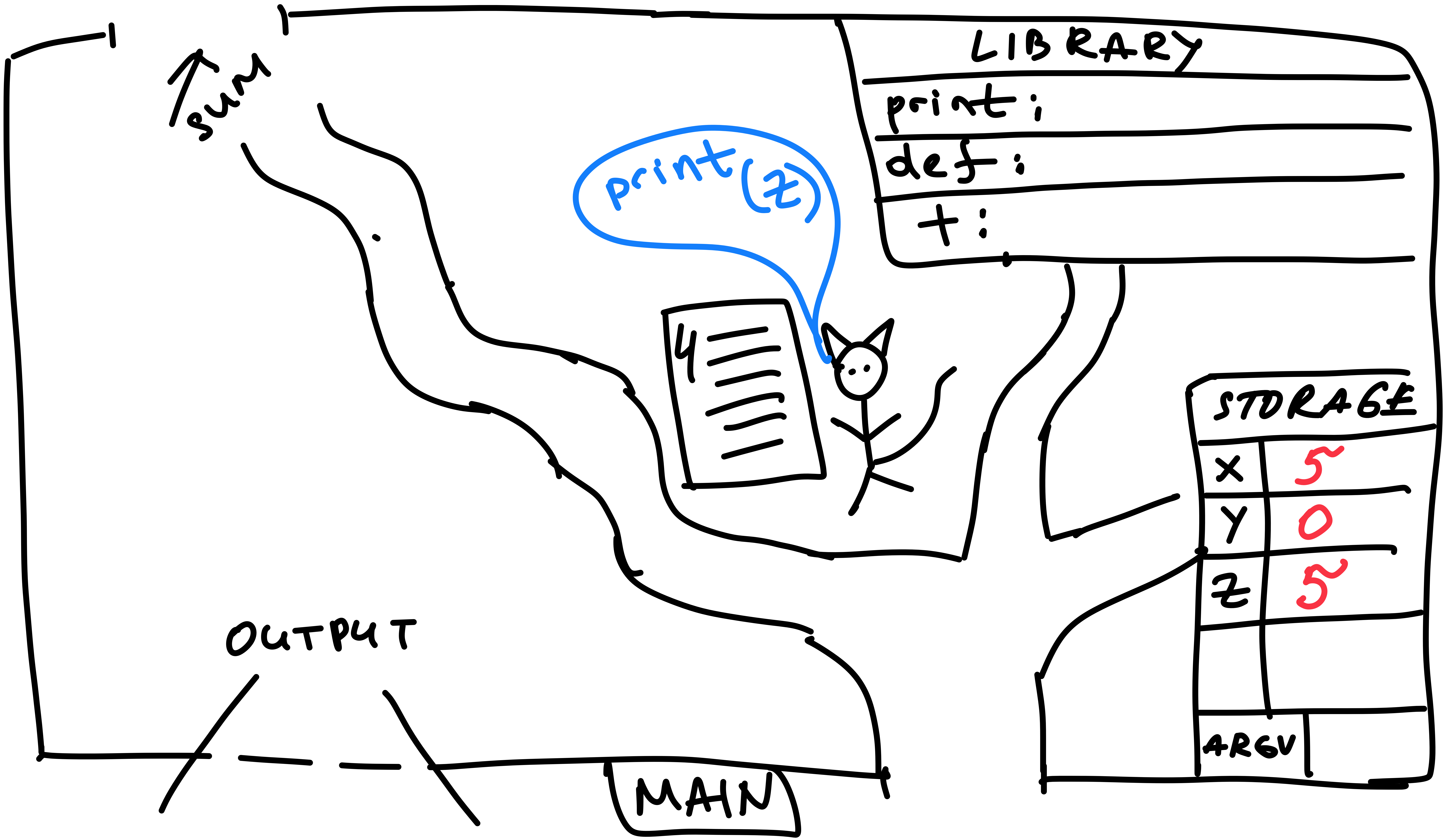

return res
LIBRARY
print;
def;
+;
STORAGE
x 5
y 0
res 5
BAGS PICK UP
2
SUM

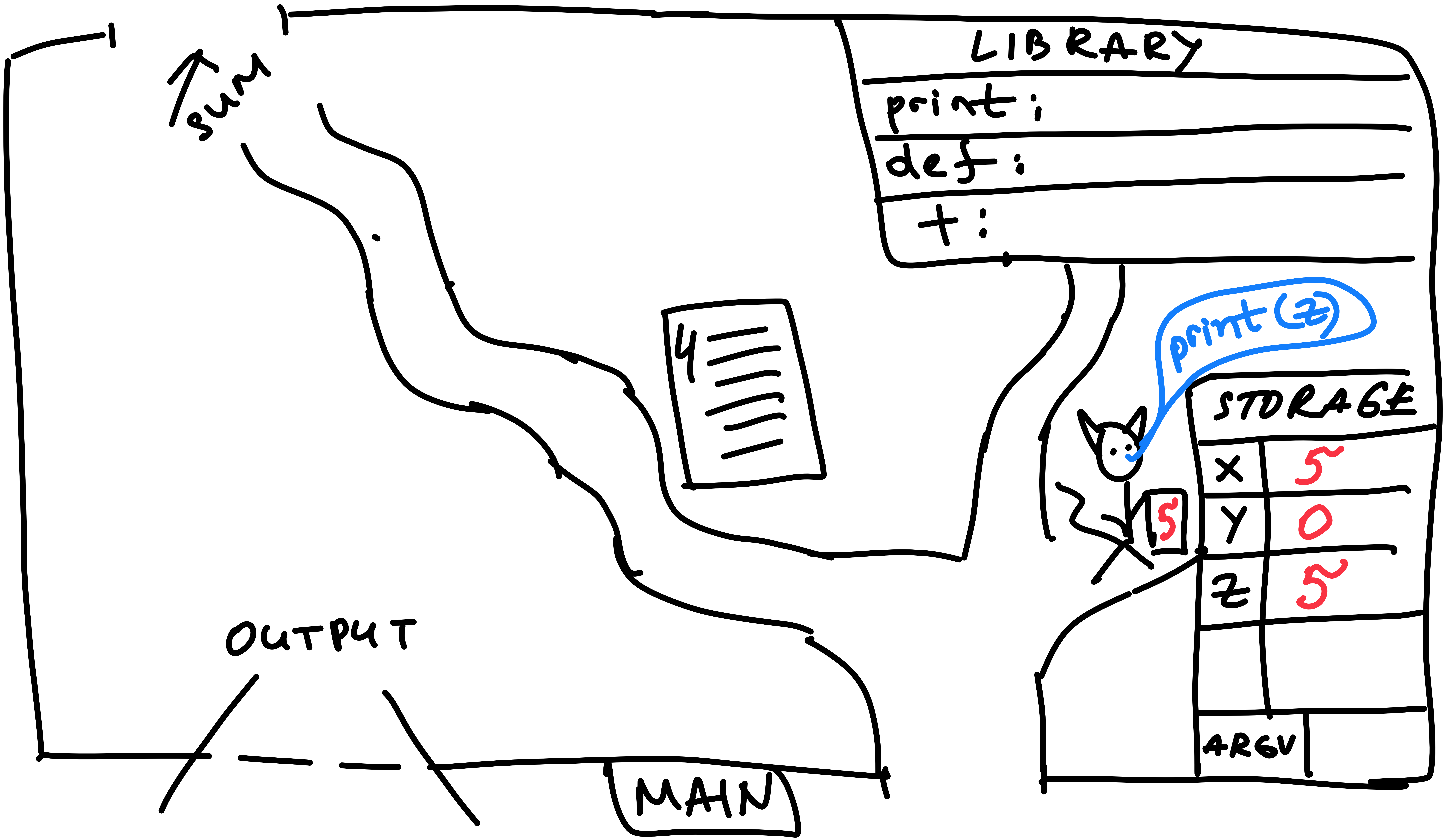

LIBRARY
print:
def:
+:
STORAGE
x 5
y 0
res 5
BAGS PICK UP
5
SUM

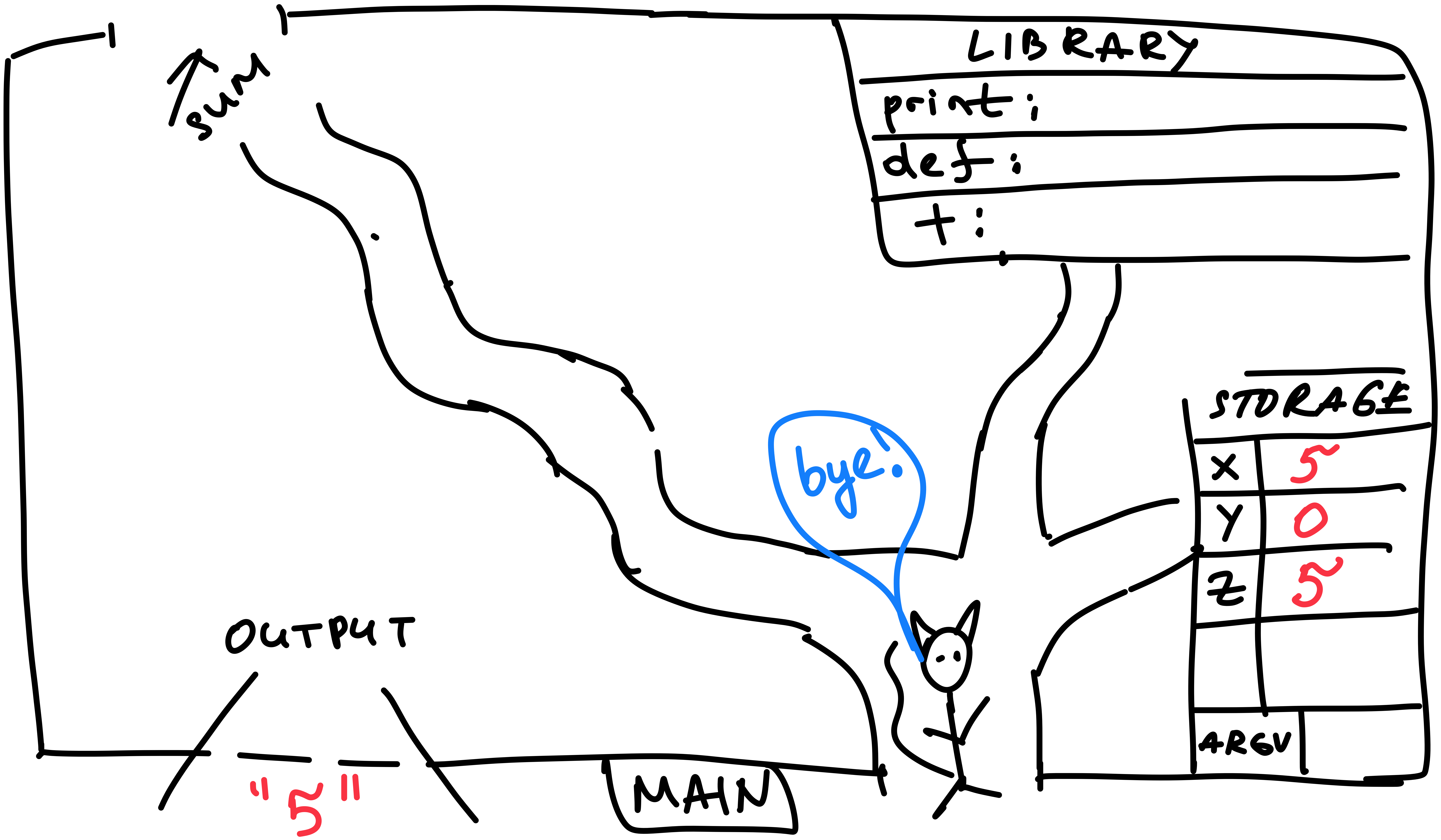

SUM
z=sum(x+y)
3
5
LIBRARY
print;
def;
+;
STORAGE
x
5
y
0
z
ARGV
OUTPUT
MAIN



SUM
LIBRARY
print:
def:
+:
STORAGE
x 5
y 0
z 5
ARGV
OUTPUT
MAIN


SUM
LIBRARY
print;
def;
+:
print(z)
4
STORAGE
x 5
y 0
z 5
ARGV
OUTPUT
MAIN


sum
LIBRARY
print ;
def ;
+ :
4
print(z)
STORAGE
x 5
y 0
z 5
ARGV
5
OUTPUT
MAIN



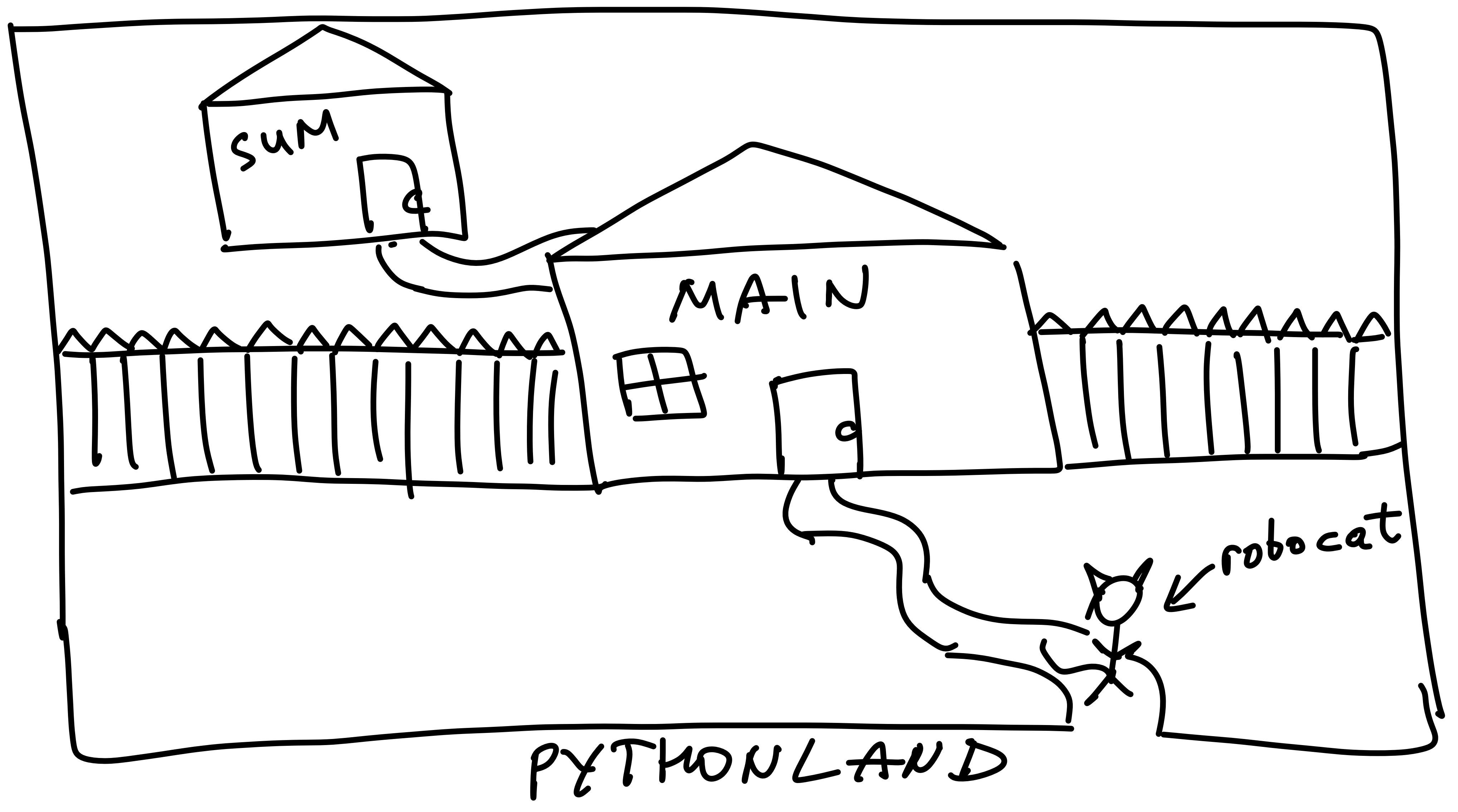
sum
LIBRARY
print :
def :
+ :
STORAGE
x 5
y 0
z 5
ARGV
bye!
OUTPUT
"5"
MAIN

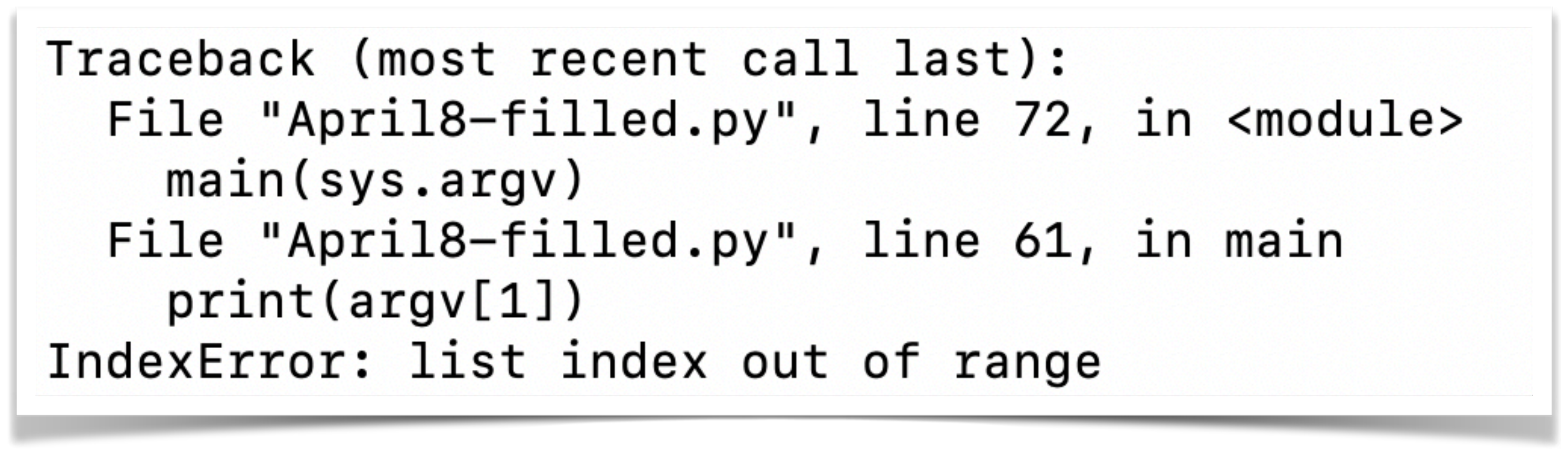
Let‘s look at this in VS Code again!
Questions?

Main function

argv

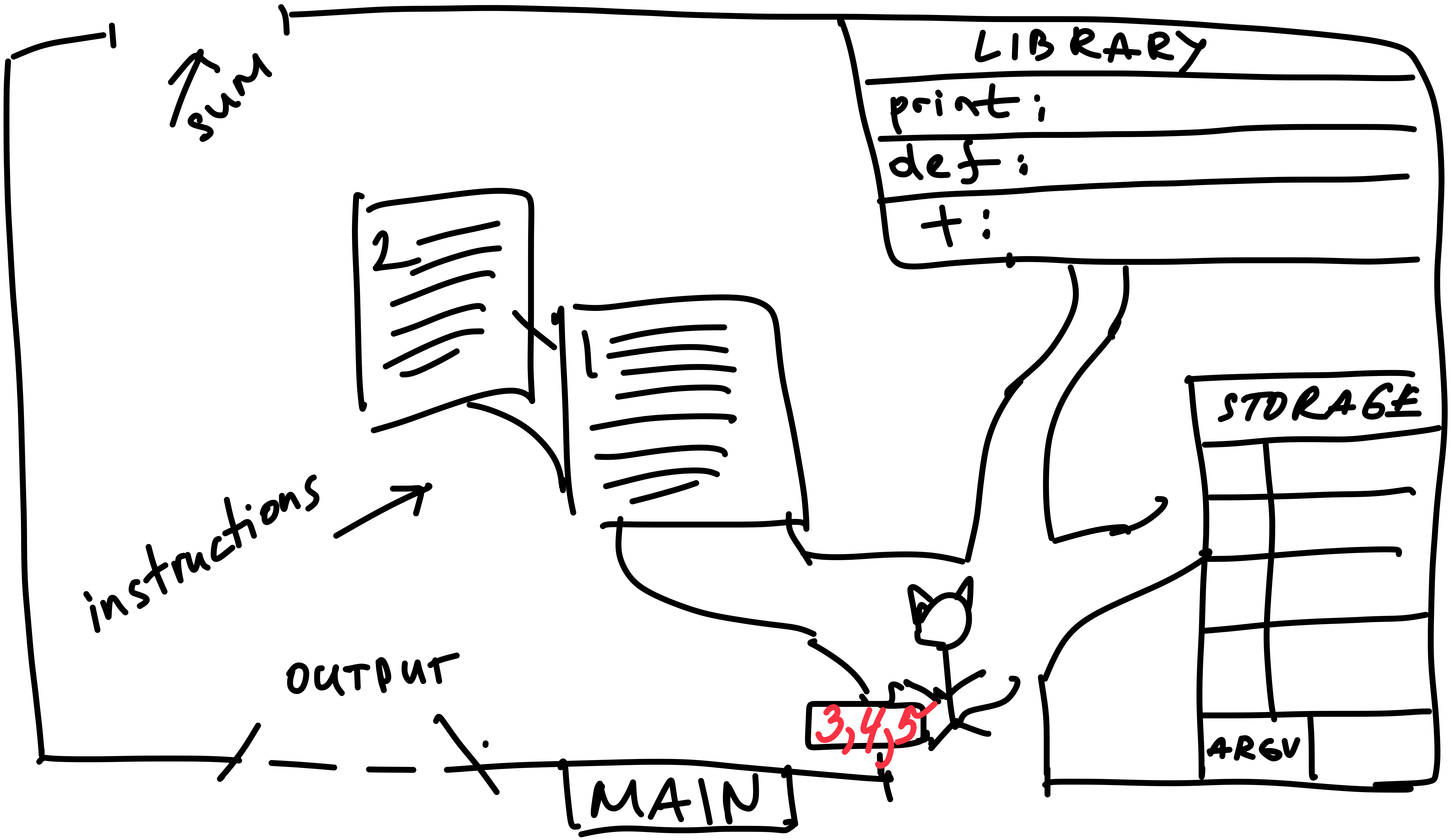
- **argv** is the sole argument of main
  - You can't call it anything else; it's defined already
  - It is a list
  - It can be "empty"
    - from the programmer's perspective
    - argv[0] is always the program name
      - (by convention)
      - argv[1], argv[2], etc., won't exist unless you **pass** them as the programmer
      - This is what **running config** is for, in VS Code
      - In command line, you pass argumens simply by typing them after the program name

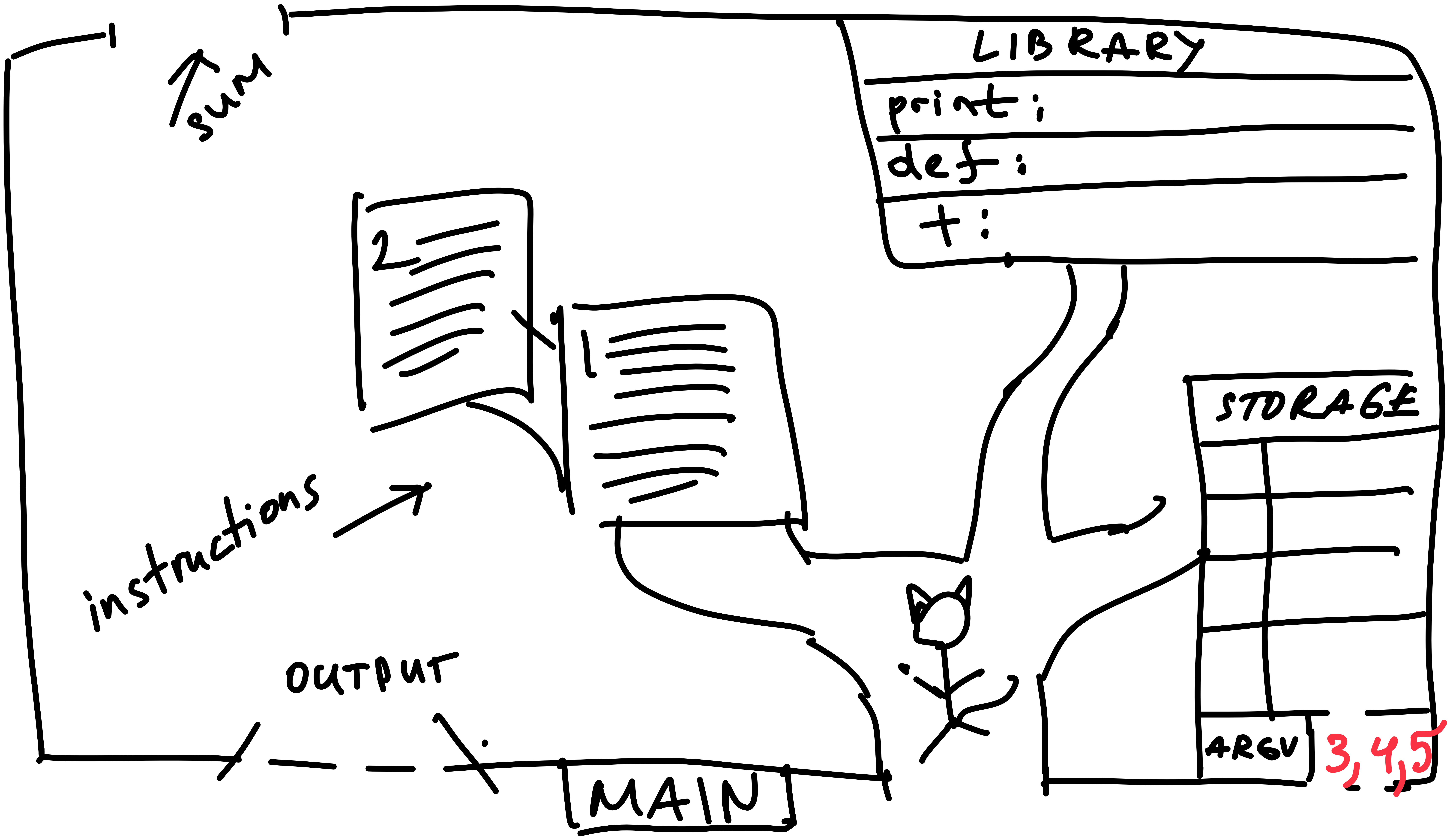




```
Traceback (most recent call last):
  File "April8-filled.py", line 72, in <module>
    main(sys.argv)
  File "April8-filled.py", line 61, in main
    print(argv[1])
IndexError: list index out of range
```

Python interpreter is complaining about not being provided an argument for main()


sum
LIBRARY
print:
def:
+:
2
1
instructions
STORAGE
ARGV
OUTPUT
3,4,5
MAIN

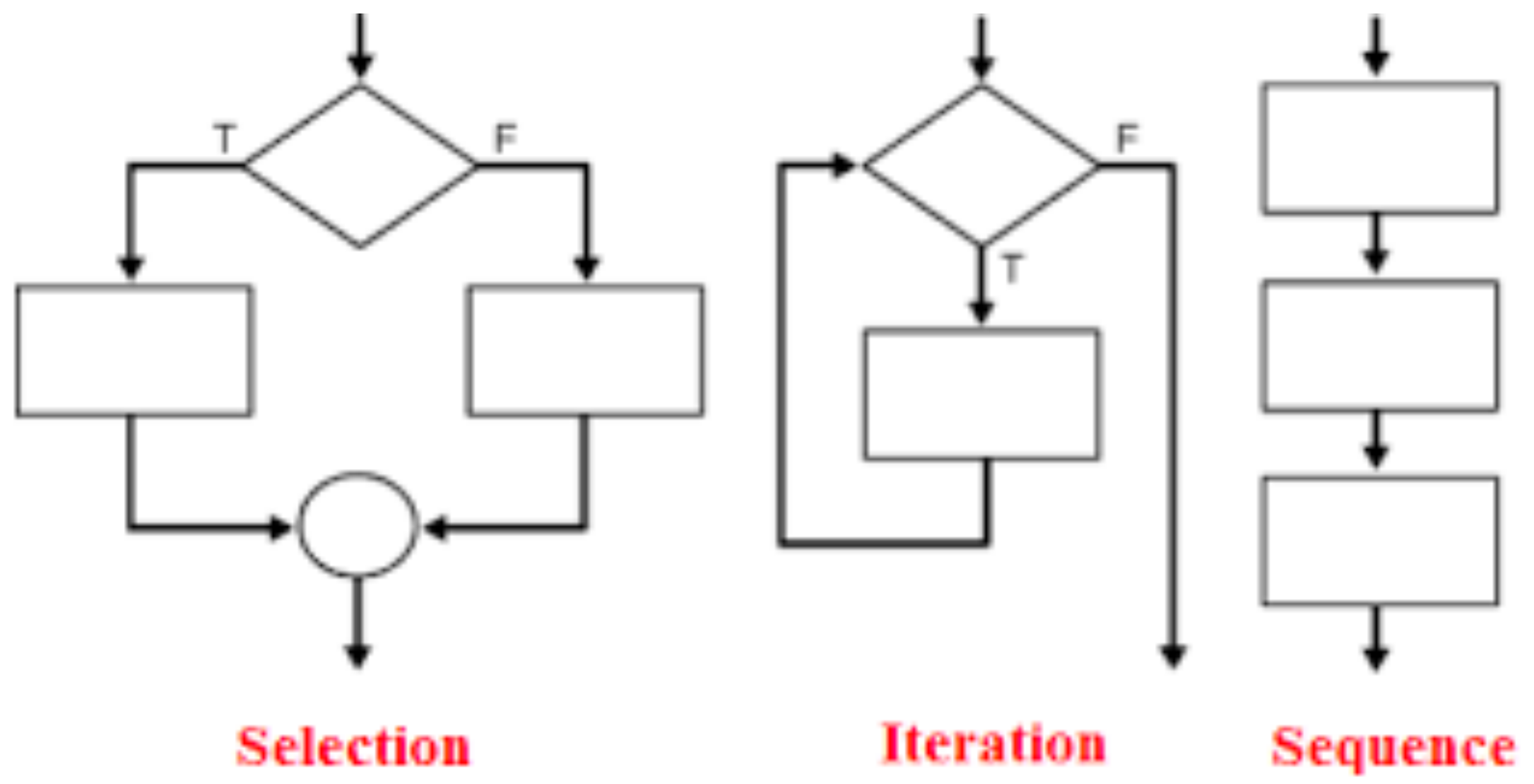

sum
LIBRARY
print;
def;
+;
2
1
instructions
STORAGE
ARGV
3,4,5
OUTPUT
MAIN

main() demo

Control flow

Which statement to execute?

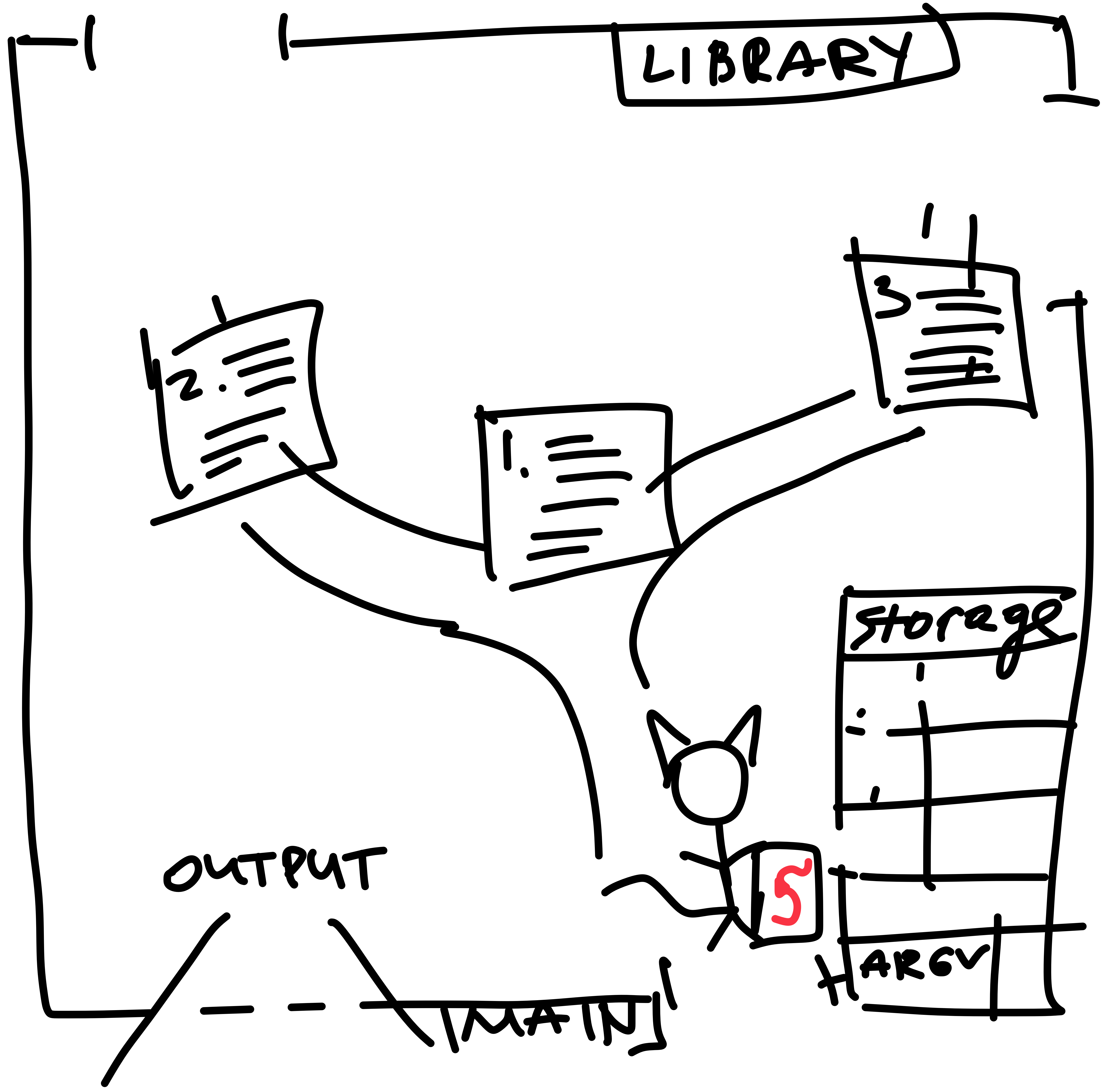
- By default, the one on the next line
- But this can change:
  - Maybe we call a function
  - Maybe we are in a loop
  - Maybe we have a **conditional** statement
    - It will only execute **if the condition is true**
    - **Relevant example:**
      - **IF** condition A is true: Predict POSITIVE review
      - **ELSE:** Predict NEGATIVE



http://net-informations.com/python/flow/default.htm

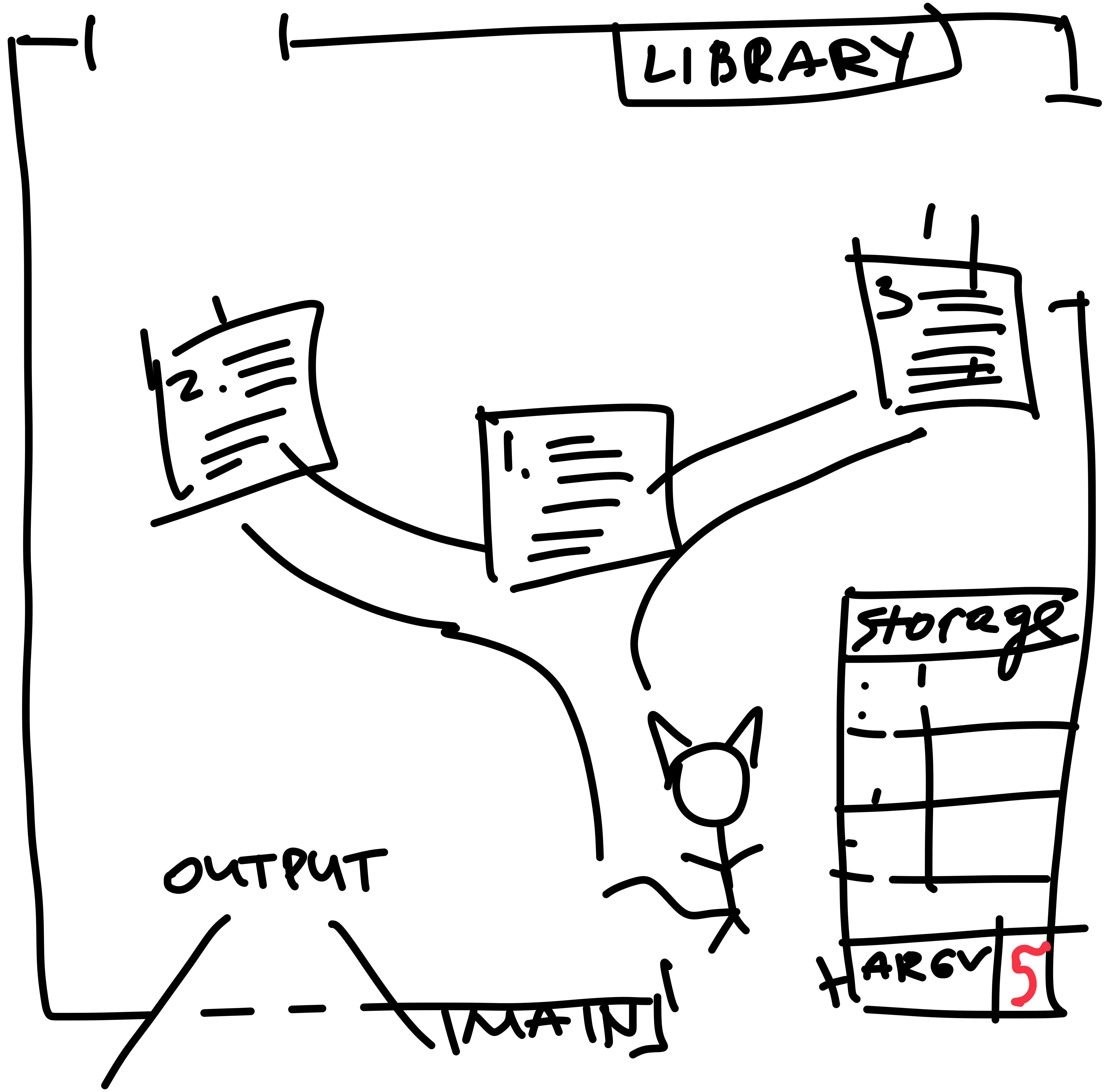
Starting a new program!

- First, we put whatever we brought in the **argument** bag, into the box labeled **argv**



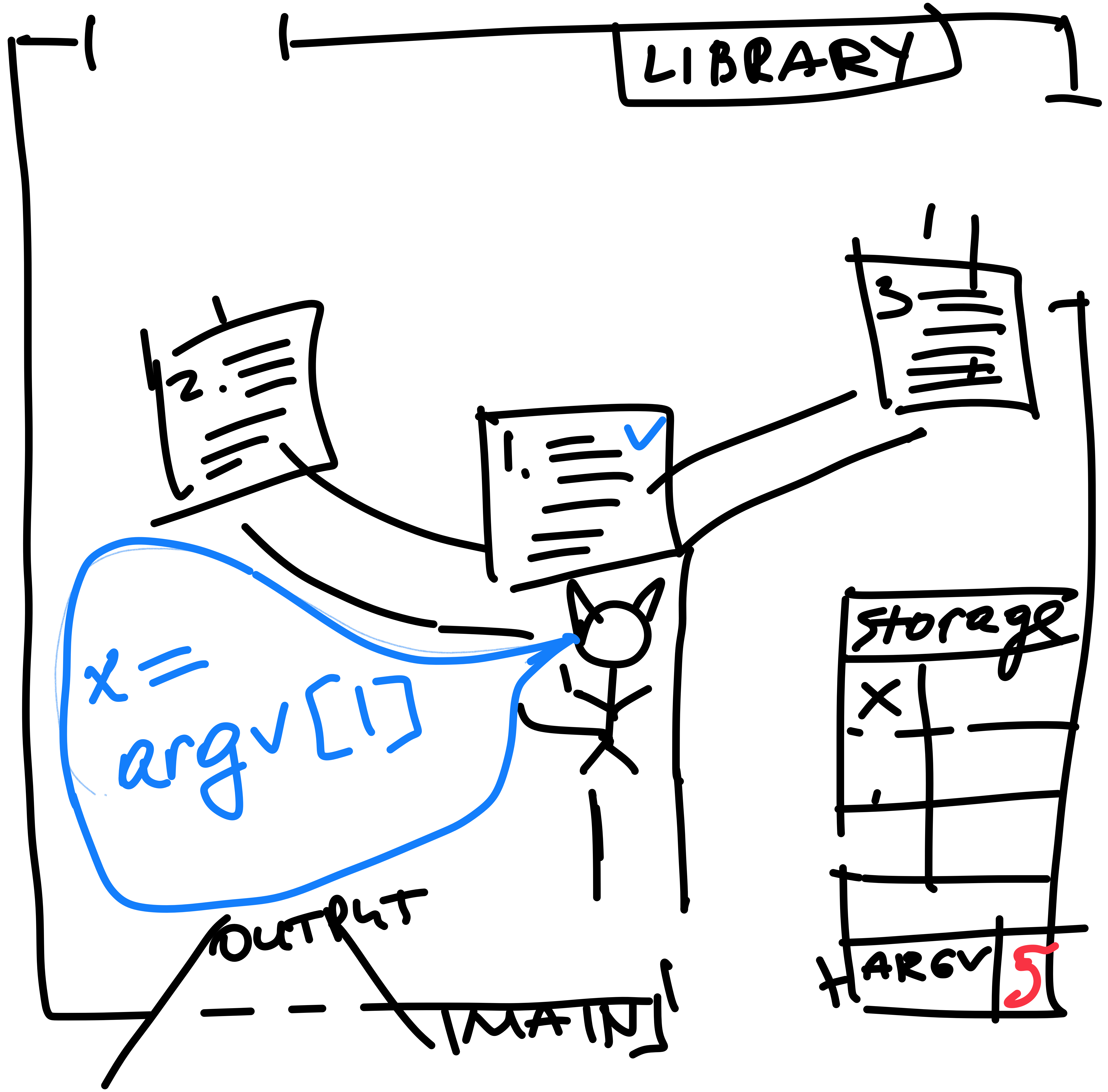
Starting a new program!

- First, we put whatever we brought in the **argument** bag, into the box labeled **argv**



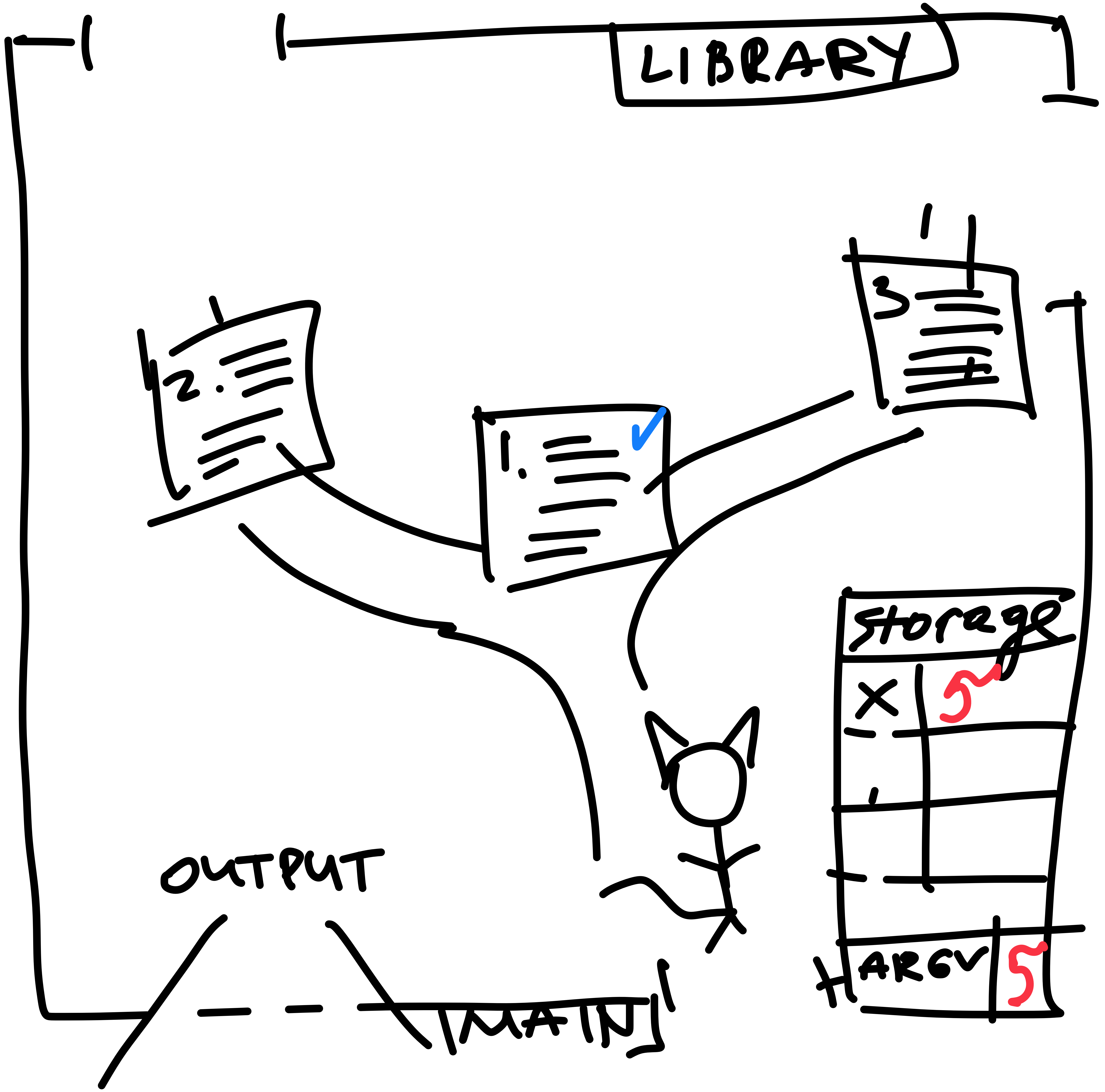
Starting a new program!

- Then, we **assign** whatever is we put in argv to the variable x



Starting a new program!

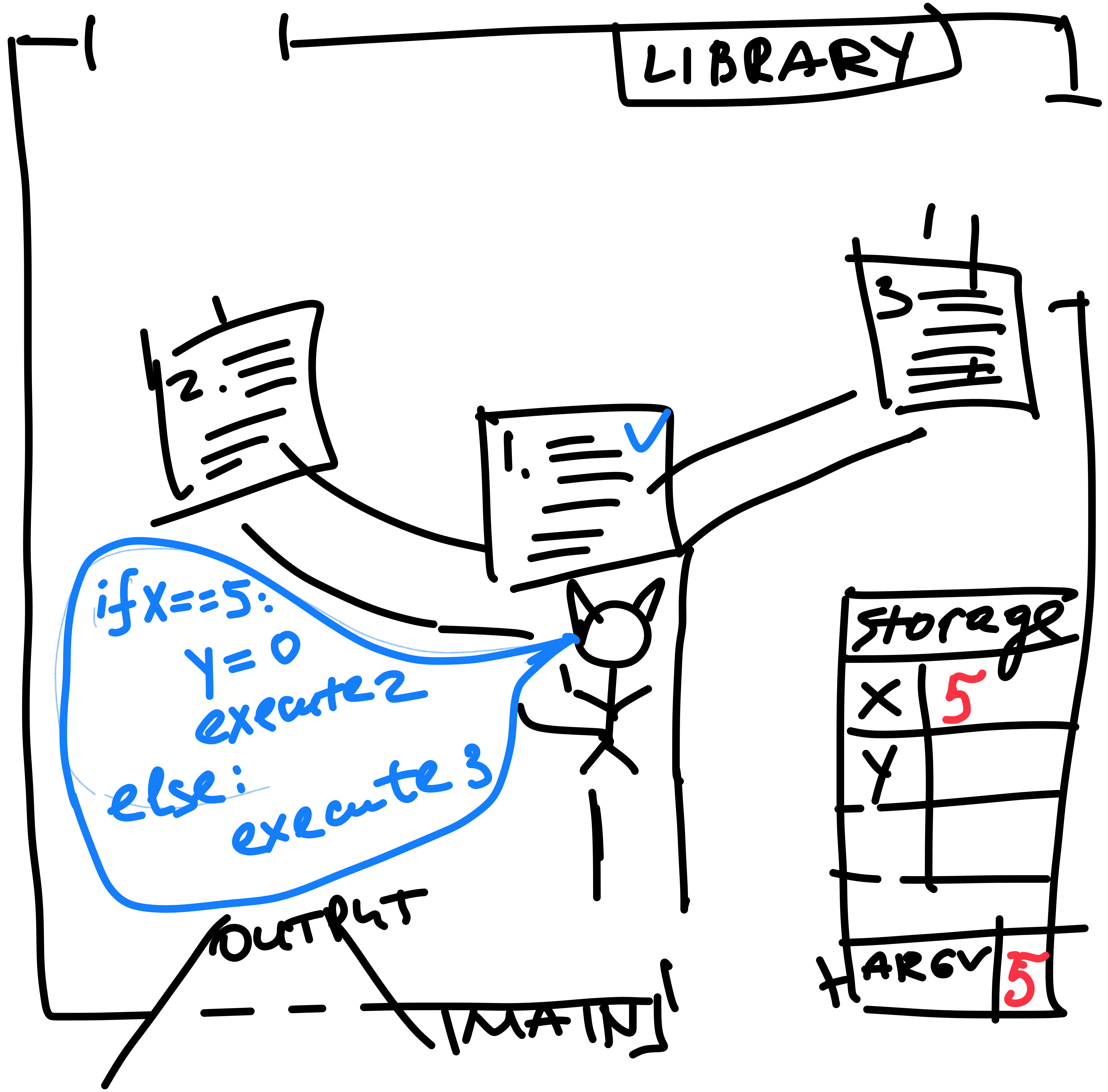
- Then, we **assign** whatever is we put in argv to the variable x



Control flow

The If-Else block

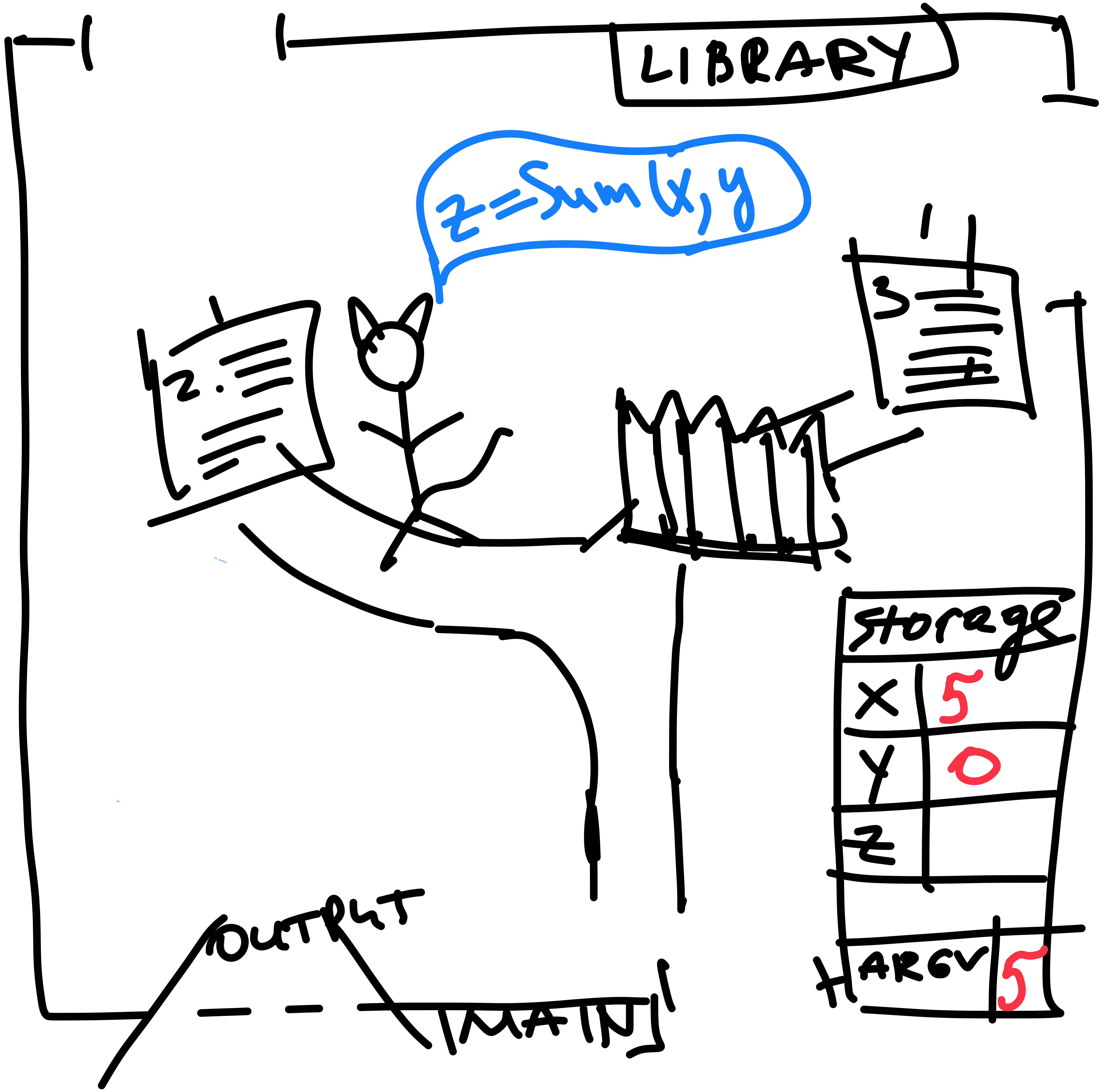
- Then, suppose instruction 1 has a conditional
  - A condition can be either **true** or **false**
    - **5 == 5** is true
    - 5 == 3 is false
    - 5 != 3 is true
    - 5 < 3 is false
    - 3 in [1,2,3] is true
    - 'a' in "apple" is true
  - Syntax:
    - **==** means "is (already/currently) equal to
    - Note the difference with the **assignment** operator **=**
    - **!=** means "is not equal to"
    - **>/<** "greater than"/"less than"
    - **>=** "greater or equal to"
    - **in** is a keyword for list membership (strings are lists of characters!)



Control flow

The If-Else block

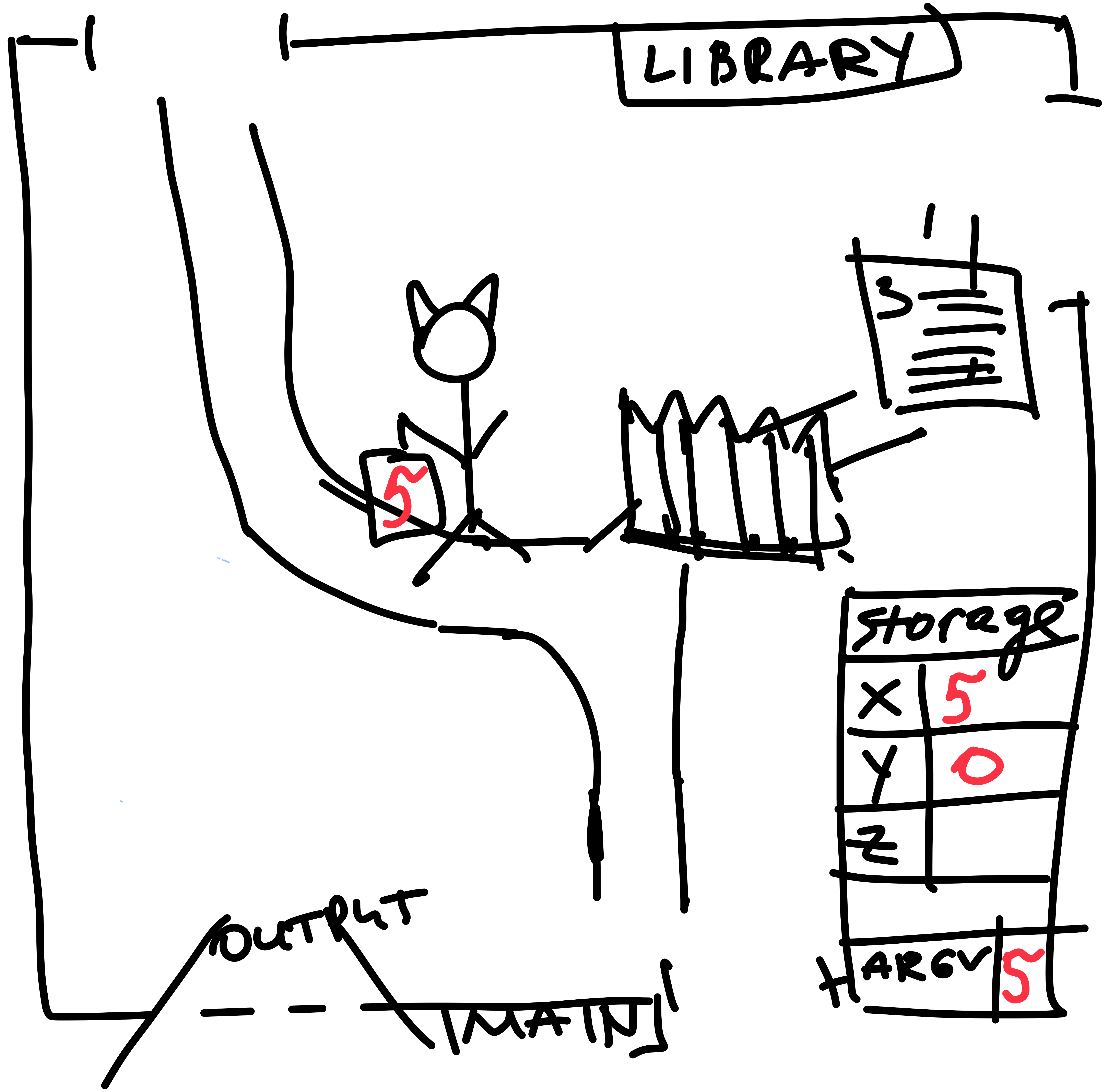
- Because x was indeed equal to 5:
  - We put **0** into the **y**-box
  - We noted that we will also need a new variable, **z**
  - And we went on to execute the next instruction on the execution path
  - We will now **never** be able to exectute instruction 3!



Control Flow

The If-Else block

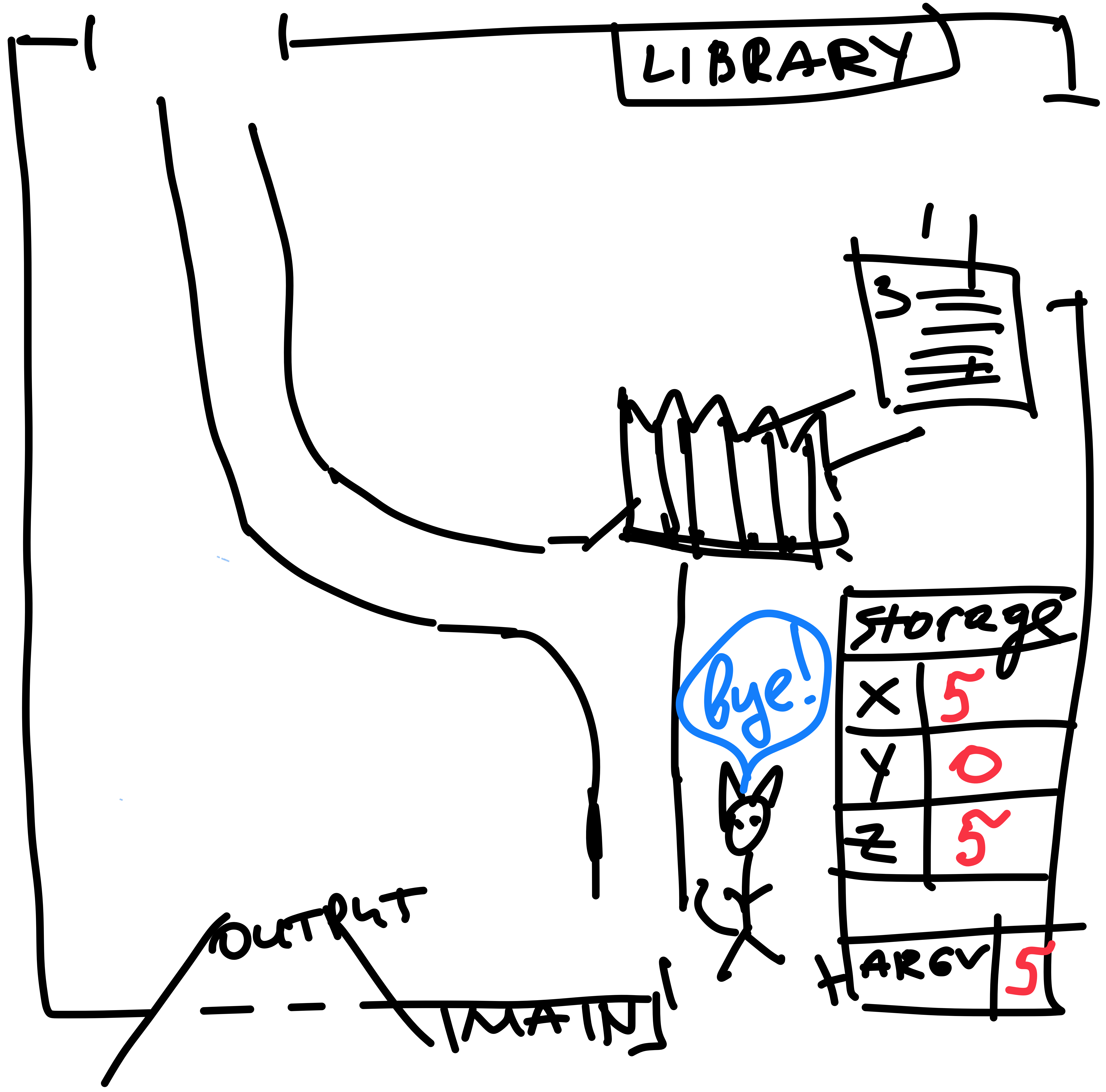
- We are done with instructions 2
- We **cannot** get to instructions 3!



Control Flow

The If-Else block

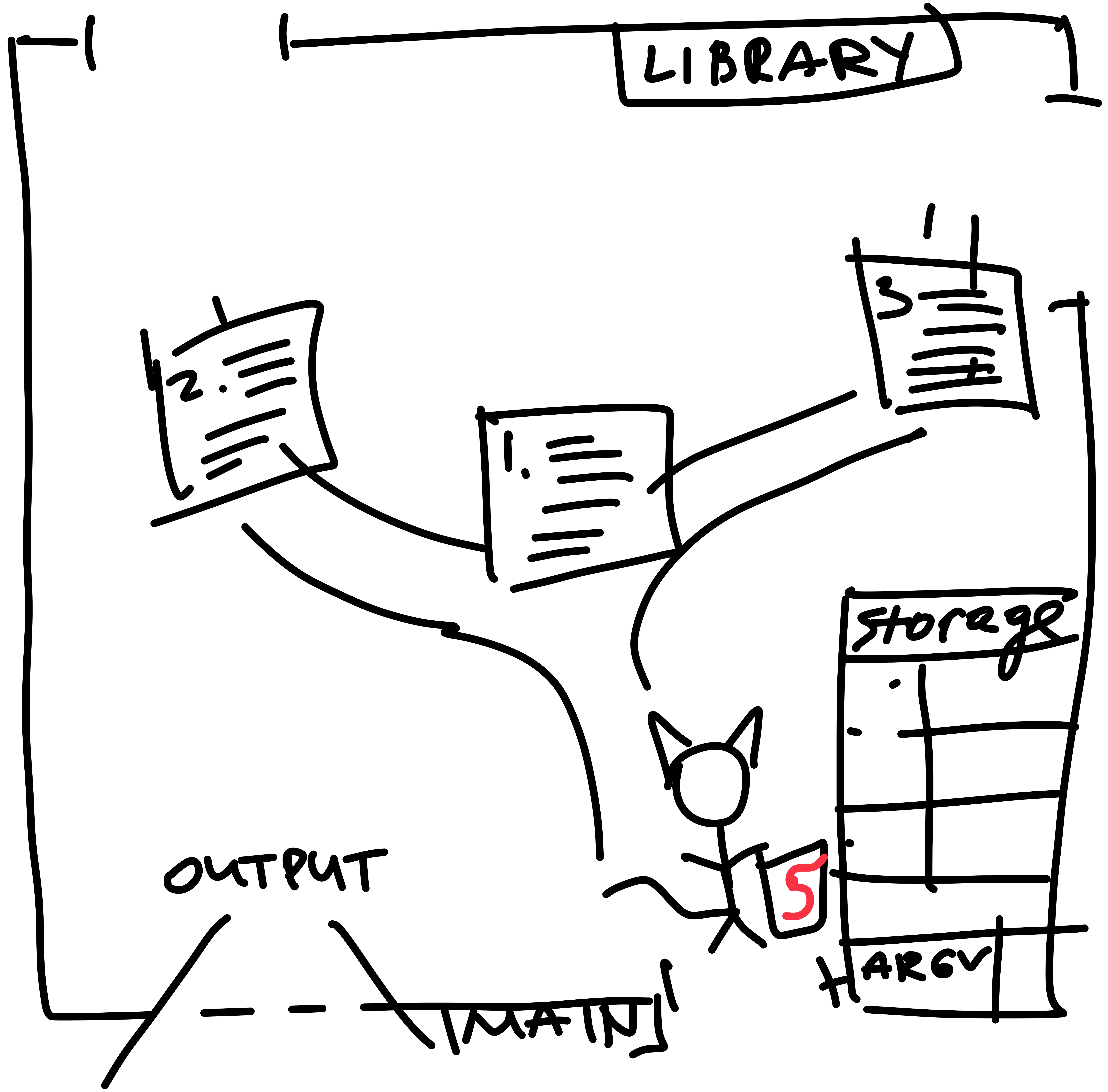
- Instructions 3 never got exectuted!



Control Flow

if—elif—elif—elif—else

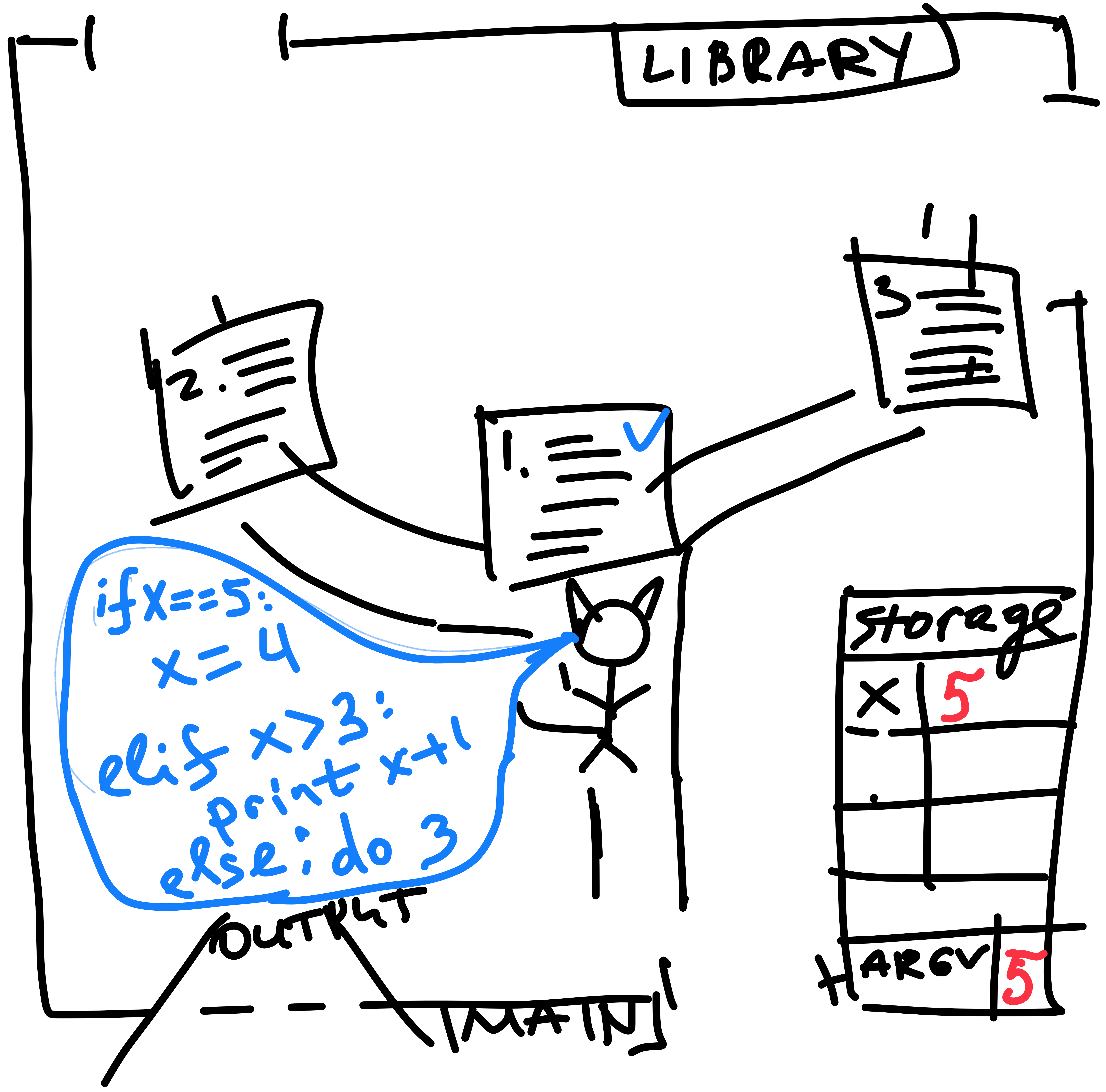
- Check for a series of conditions, one by one
  - Only **ONE** of the blocks will be executed
    - ("**else** if" = "elif")
  - The code in the first block for which condition is true
- Or, if none of the conditions is true:
  - Execute the code in the Else-block



Control Flow

if—elif—elif—elif—else

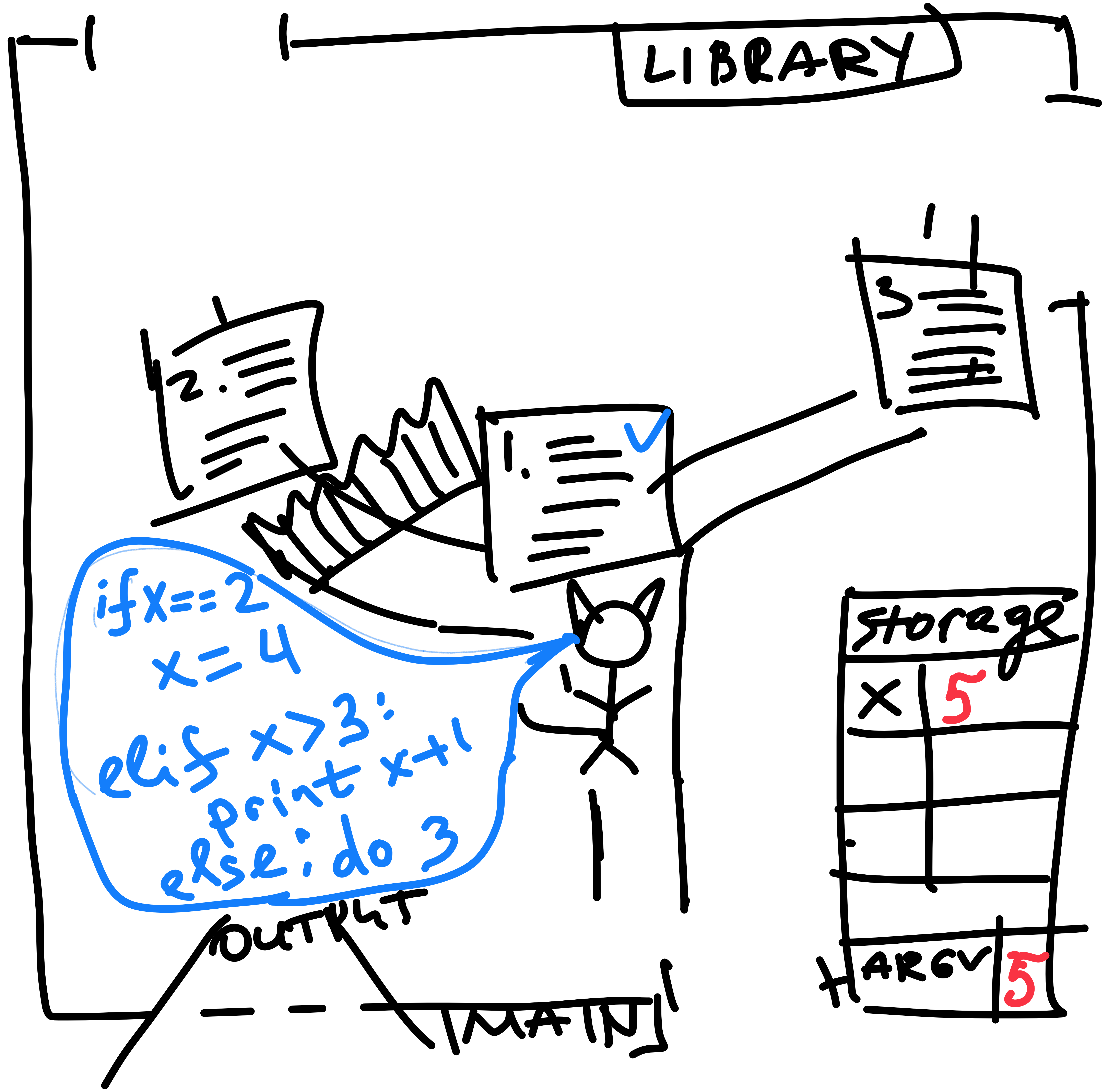
- Check for a series of conditions, one by one
  - Only **ONE** of the blocks will be executed
    - (“**else** if” = “elif”)
  - The code in the first block for which condition is true
- Or, if none of the conditions is true:
  - Execute the code in the Else-block



Control Flow

if—elif—elif—elif—else

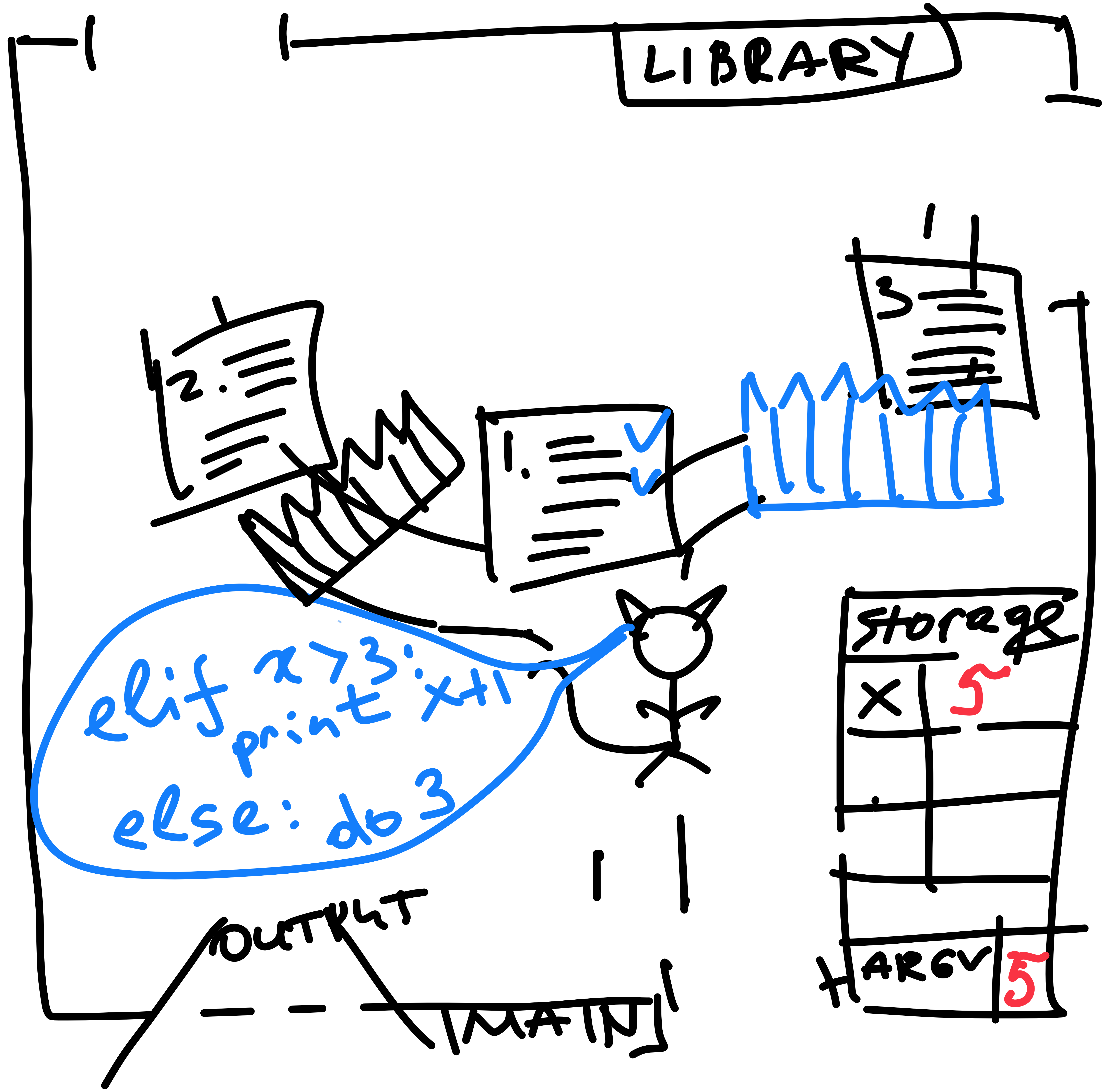
- Check for a series of conditions, one by one
  - Only **ONE** of the blocks will be executed
    - (“**else** if” = “elif”)
  - The code in the first block for which condition is true
- Or, if none of the conditions is true:
  - Execute the code in the Else-block
- NB: instructions 2 is “dead code”!



Control Flow

if—elif—elif—elif—else

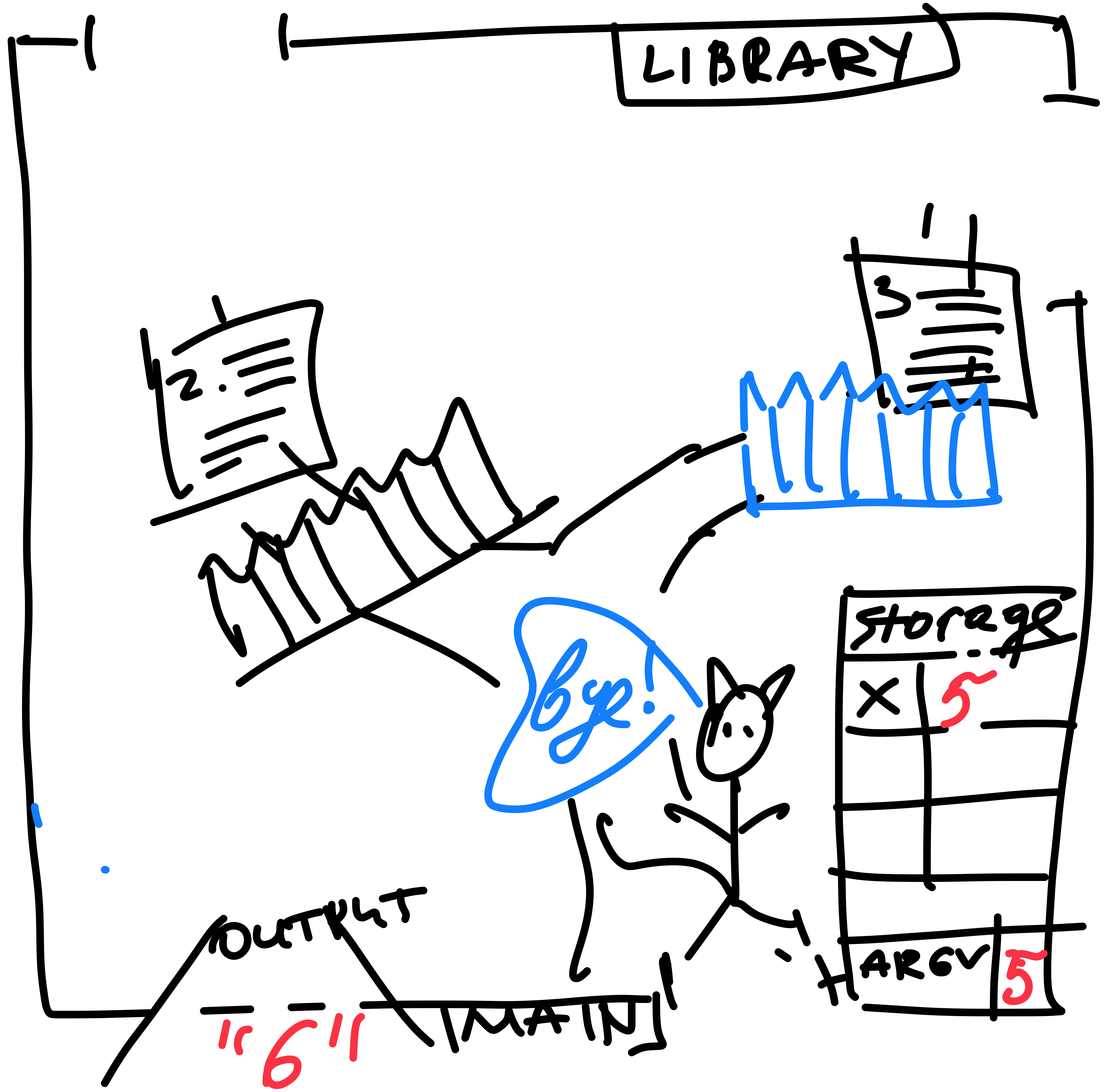
- Check for a series of conditions, one by one
  - Only **ONE** of the blocks will be executed
    - (“**else** if” = “elif”)
  - The code in the first block for which condition is true
- Or, if none of the conditions is true:
  - Execute the code in the Else-block
- NB: instructions 2 is “dead code”!



Control Flow

if—elif—elif—elif—else

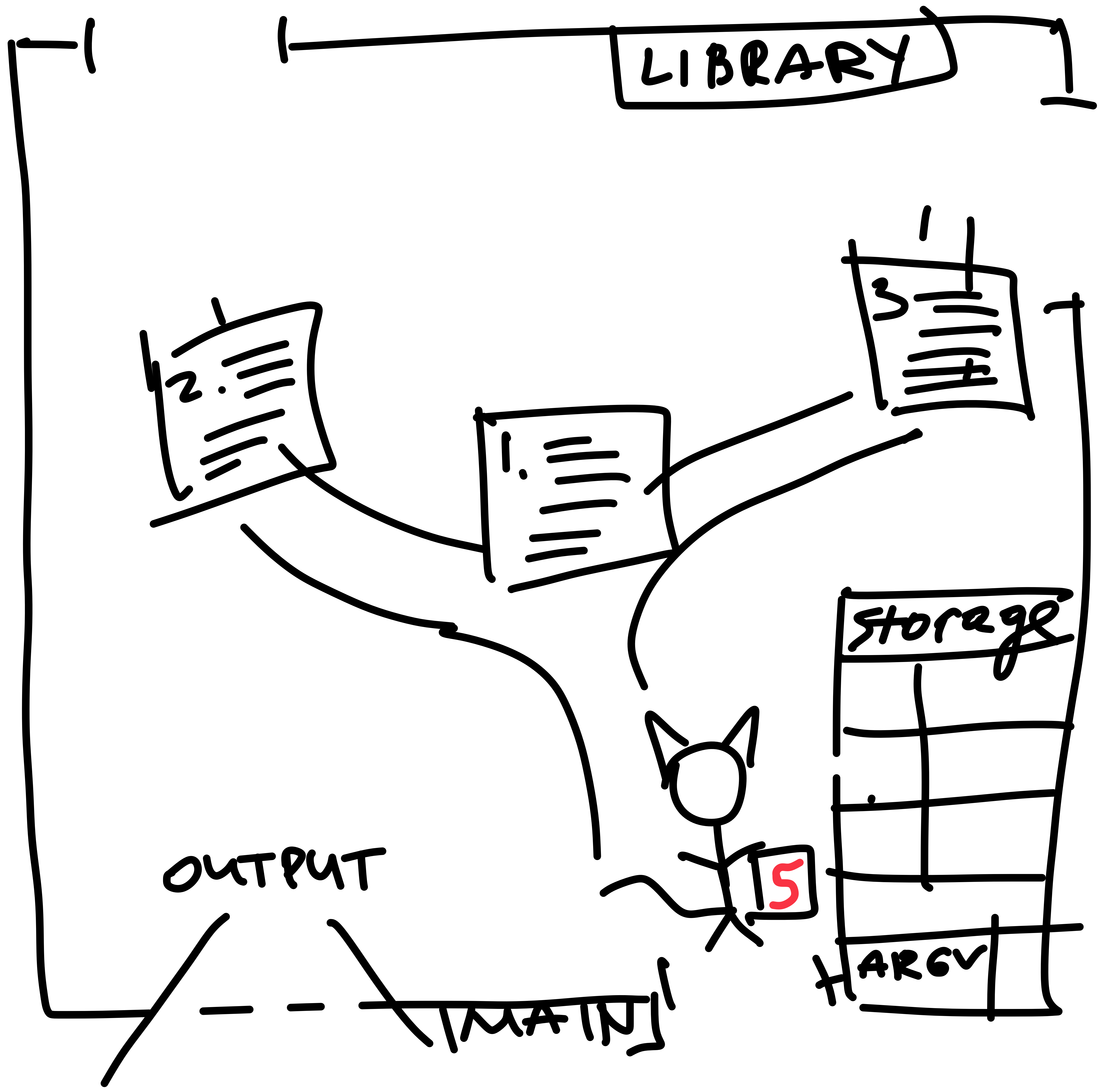
- Check for a series of conditions, one by one
  - Only **ONE** of the blocks will be executed
    - (“**else** if” = “elif”)
  - The code in the first block for which condition is true
- Or, if none of the conditions is true:
  - Execute the code in the Else-block
- NB: instructions 2 is “dead code”!



Control Flow

If—if—if—if—(else?)

- In a series of If-blocks:
  - Each **if** is independent, and **ALL** blocks for which condition is true will be executed.
- Only the **LAST if—else** is a **true if—else** block
  - The code in the **else** block will be executed only if the condition for the **LAST if** is false
  - All the other **ifs** don't matter for this last **else**!



Control Flow

If—if—if—if—(else?)

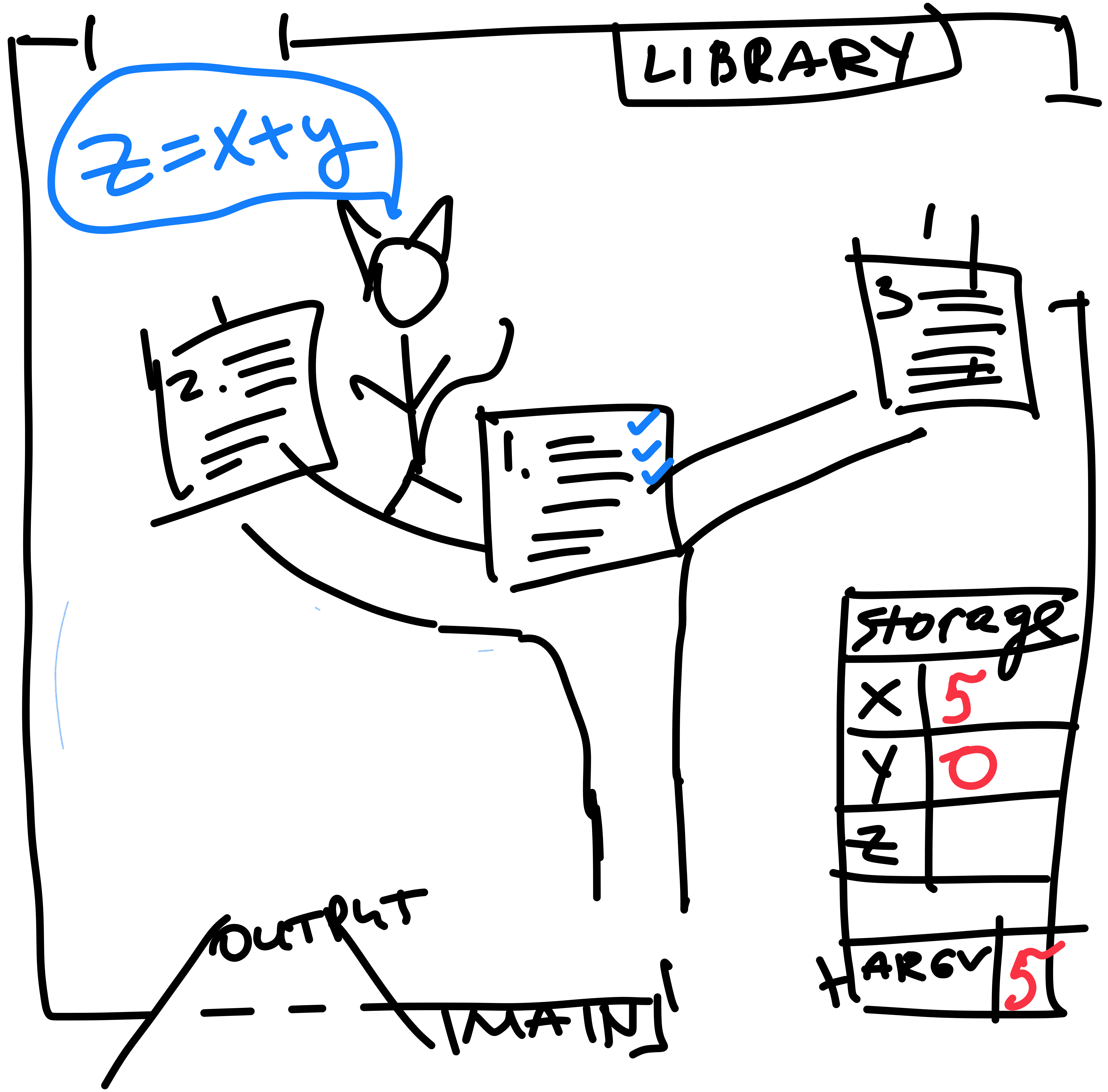
- In a series of If-blocks:
  - Each **if** is independent, and **ALL** blocks for which condition is true will be executed.
- Only the **LAST if—else** is a **true if—else** block
  - The code in the **else** block will be executed only if the condition for the **LAST if** is false
  - All the other **ifs** don't matter for this last **else**!

LIBRARY
1.
2.
3
if x==5:
y=0
execute 2
if x !=0:
execute 3
Storage
X
5
Y
ARGV
5
OUTPUT
MAIN

Control Flow

If—if—if—if—(else?)

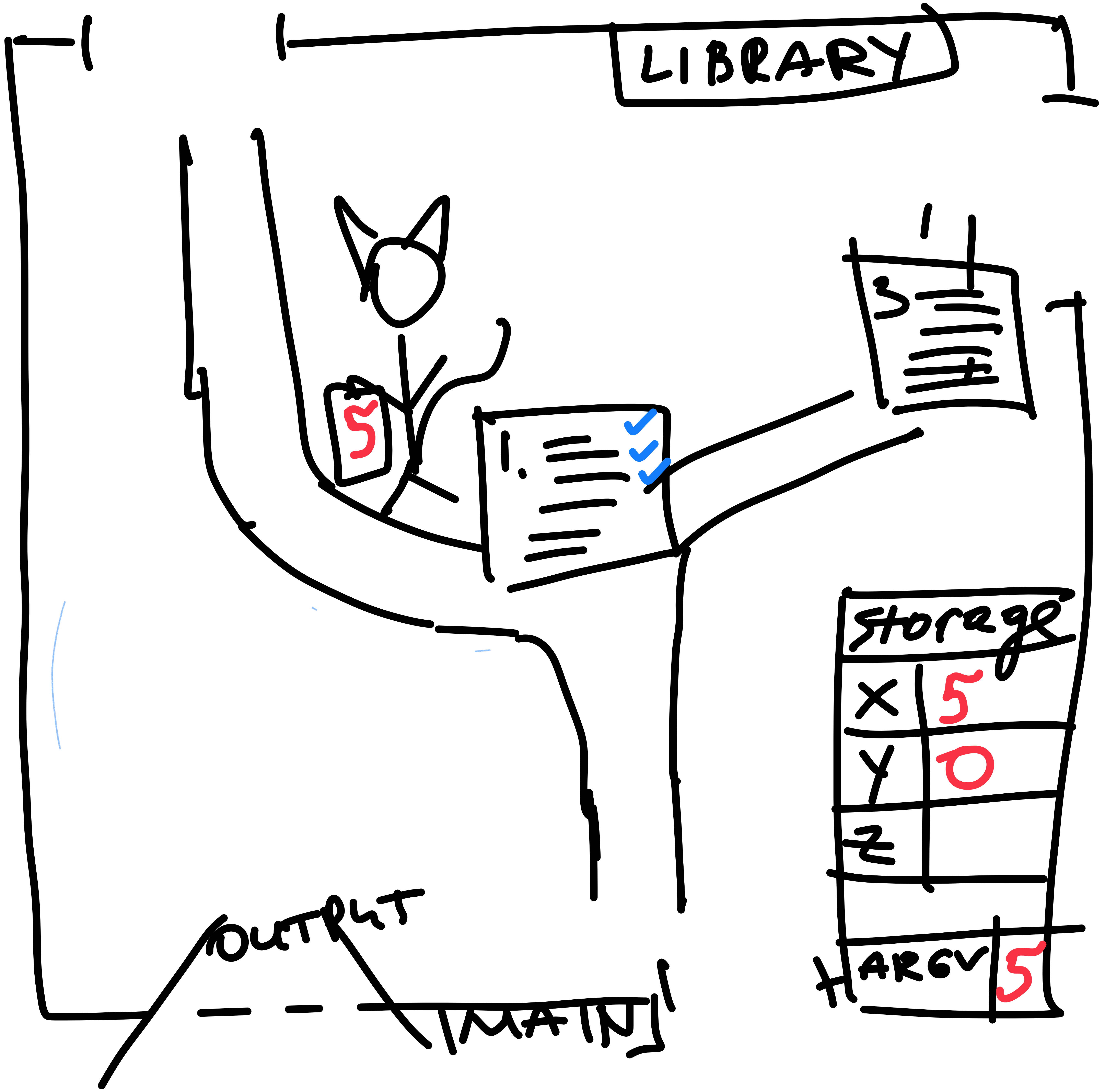
- In a series of If-blocks:
  - Each **if** is independent, and **ALL** blocks for which condition is true will be executed.
- Only the **LAST if—else** is a **true if—else** block
  - The code in the **else** block will be executed only if the condition for the **LAST if** is false
  - All the other **ifs** don't matter for this last **else**!



Control Flow

If—if—if—if—(else?)

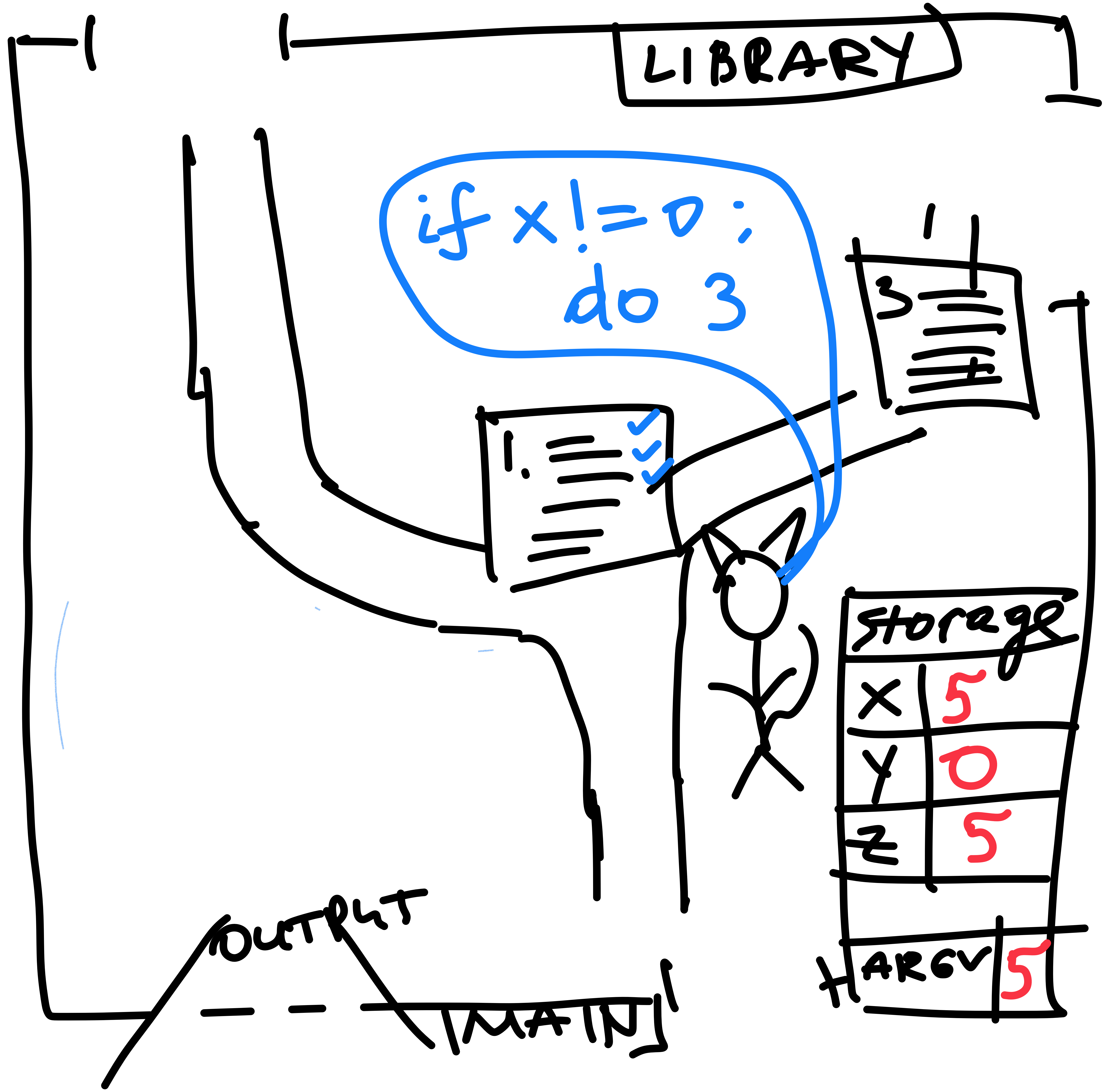
- In a series of If-blocks:
  - Each **if** is independent, and **ALL** blocks for which condition is true will be executed.
- Only the **LAST if—else** is a **true if—else** block
  - The code in the **else** block will be executed only if the condition for the **LAST if** is false
  - All the other **ifs** don't matter for this last **else**!



Control Flow

If—if—if—if—(else?)

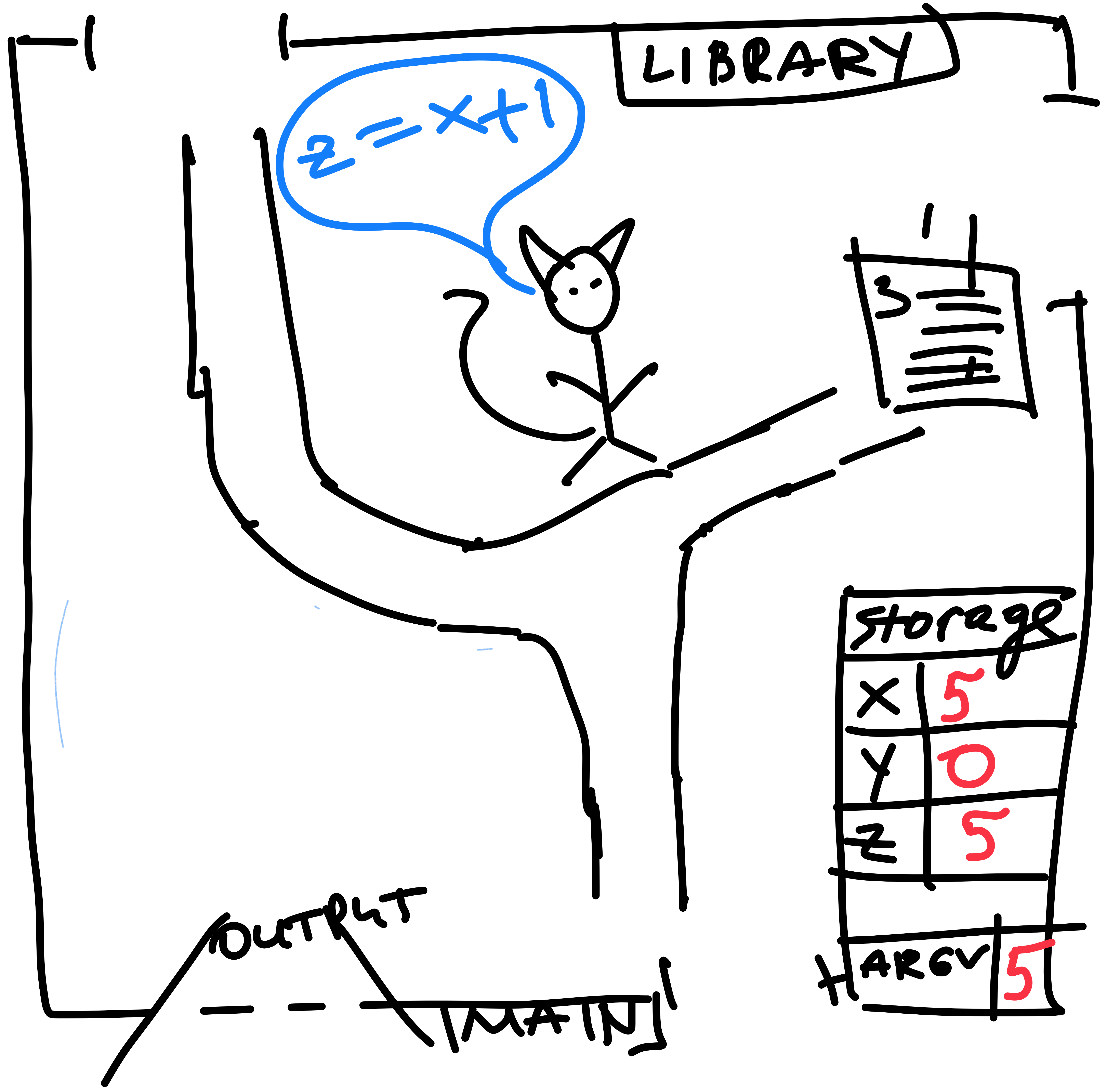
- In a series of If-blocks:
  - Each **if** is independent, and **ALL** blocks for which condition is true will be executed.
- Only the **LAST if—else** is a **true if—else** block
  - The code in the **else** block will be executed only if the condition for the **LAST if** is false
  - All the other **ifs** don't matter for this last **else**!



Control Flow

If—if—if—if—(else?)

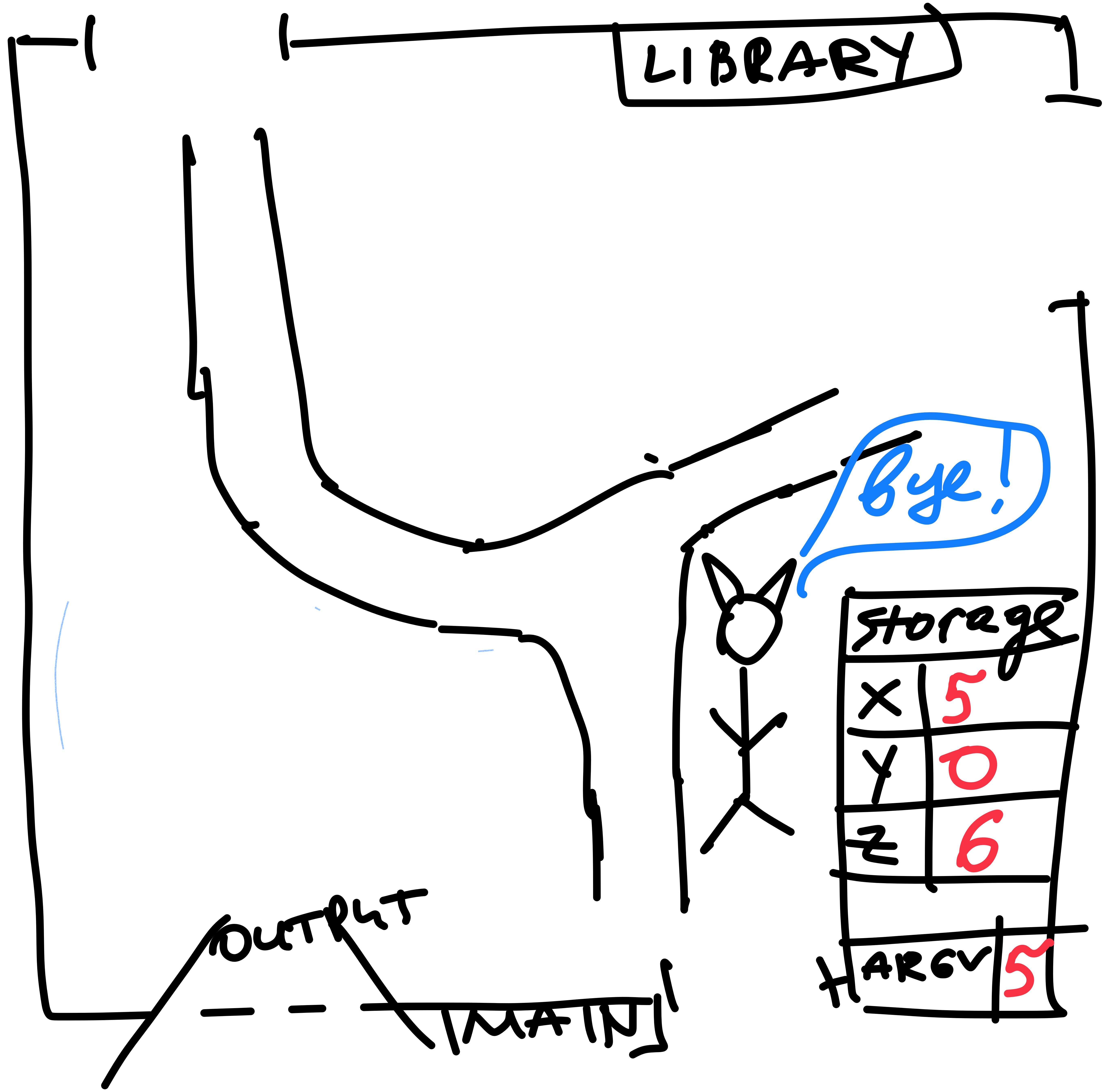
- In a series of If-blocks:
  - Each **if** is independent, and **ALL** blocks for which condition is true will be executed.
- Only the **LAST if—else** is a **true if—else** block
  - The code in the **else** block will be executed only if the condition for the **LAST if** is false
  - All the other **ifs** don't matter for this last **else**!



Control Flow

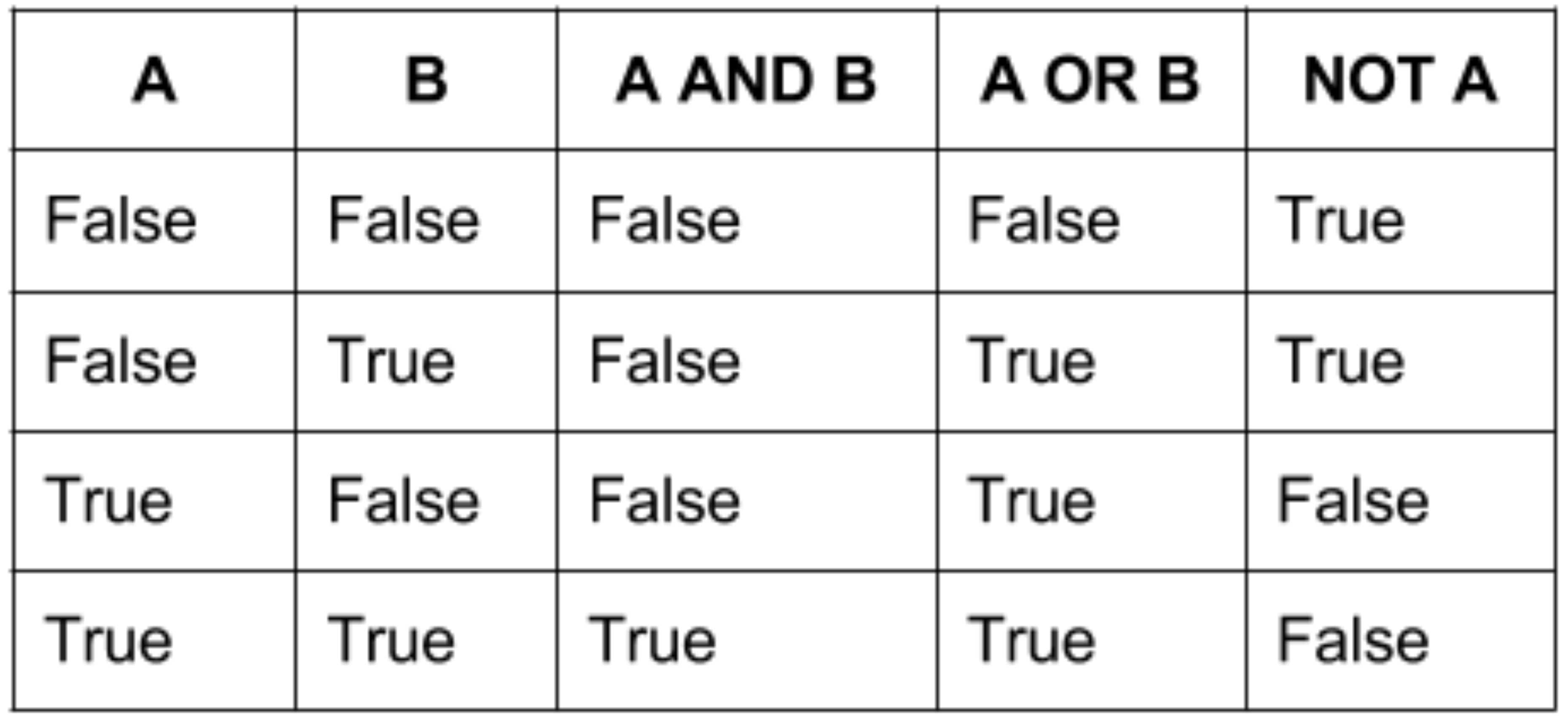
If—if—if—if—(else?)

- In a series of If-blocks:
  - Each **if** is independent, and **ALL** blocks for which condition is true will be executed.
- Only the **LAST if—else** is a **true if—else** block
  - The code in the **else** block will be executed only if the condition for the **LAST if** is false
  - All the other **ifs** don't matter for this last **else**!



Boolean logic

- Boolean logic:
  - Every statement is either True or False
  - Logical operators: AND, OR, NOT
    - There is also XOR, not shown in table
  - e.g.:
    - (5 > 3) AND (5 > 10) is FALSE
    - (5 > 3) OR (5 > 10) is TRUE
    - (5 > 3) AND (NOT (5 > 10)) is TRUE
      - NOT (5 > 10) is TRUE



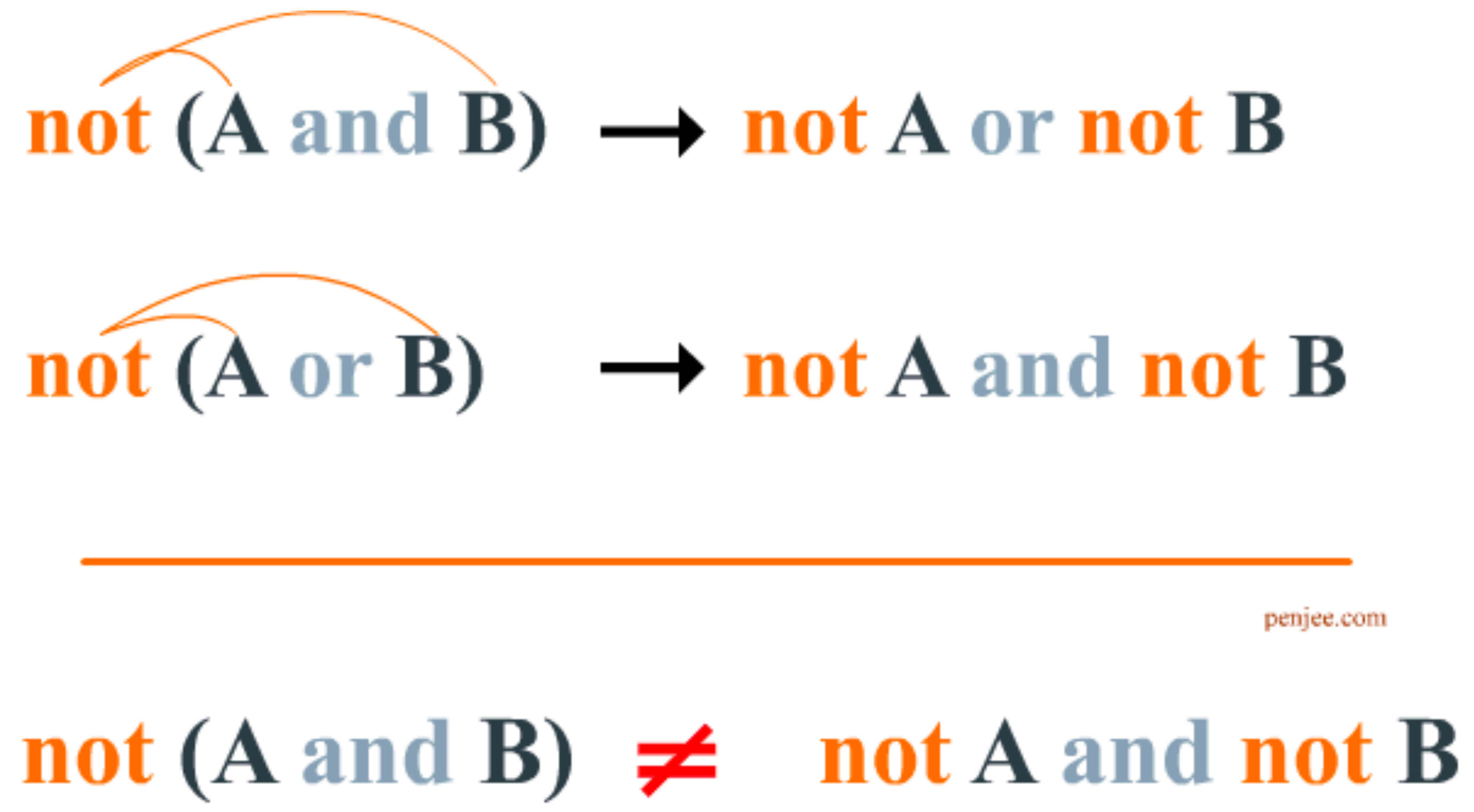
A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

https://en.wikiversity.org/wiki/File:Truth_table_for_AND,_OR,_and_NOT.png

Boolean logic

De Morgan's law

- Boolean logic:
  - Every statement is either True or False
  - Logical operators: AND, OR, NOT
    - e.g.:
      - "It canNOT be [both winter AND summer] (at the same time)
      - translates into:
      - At any point of time, [it is NOT winter] OR [it is NOT summer]



not (A and B) $\rightarrow$ not A or not B

not (A or B) $\rightarrow$ not A and not B

penjee.com

not (A and B) $\neq$ not A and not B

https://blog.penjee.com/wp-content/uploads/2016/12/demorgans-law-formula_all.png

The FizzBuzz problem

Conditionals example

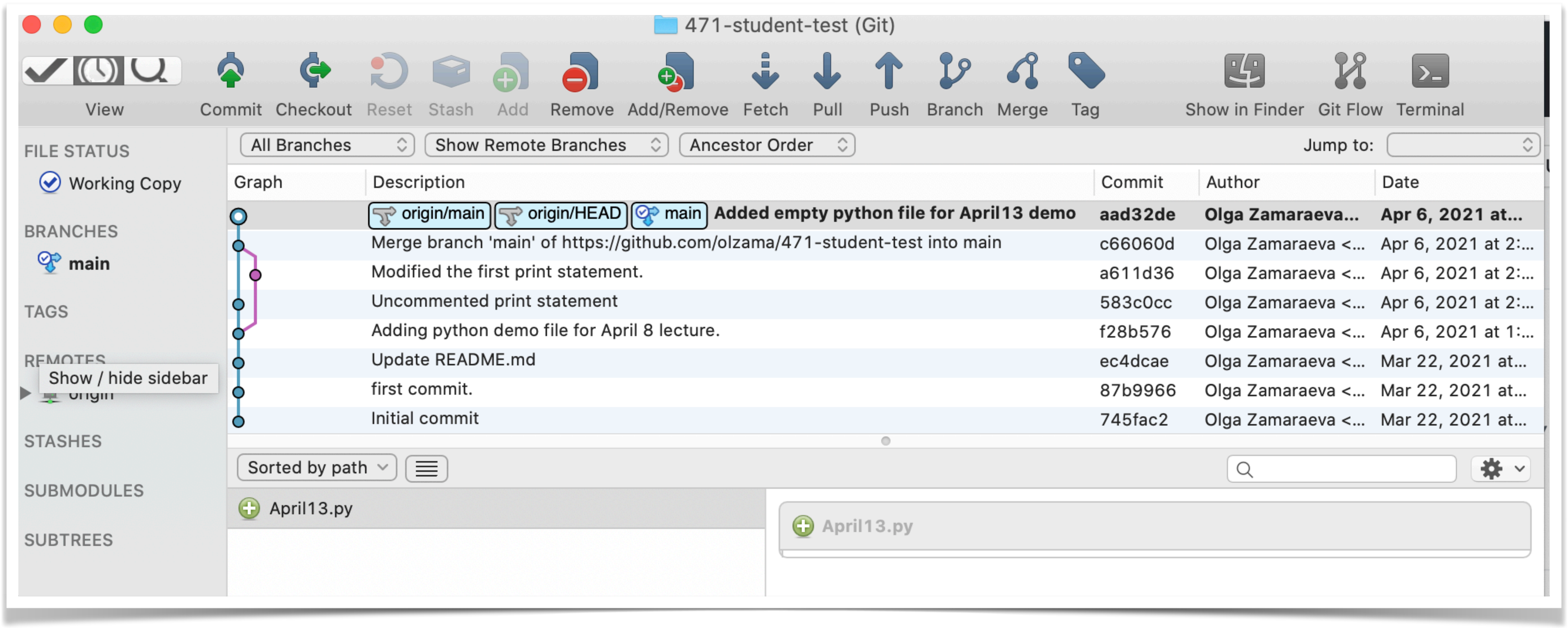
- This **classic** problem **still** is sometimes assigned on **real** interviews
  - And **many** programmers still get it wrong!
- Spec:
  - Iterate over numbers from 1 to 100.
    - If the number is divisible by 3:
      - Print “Fizz”
    - If the number is divisible by 5:
      - Print “Buzz”
    - If the number is divisible by both 3 and 5:
      - Print “FizzBuzz”
    - Otherwise, print the number itself!
- Let’s GO!

https://code.kx.com/q/learn/reading/fizzbuzz/

Addenda

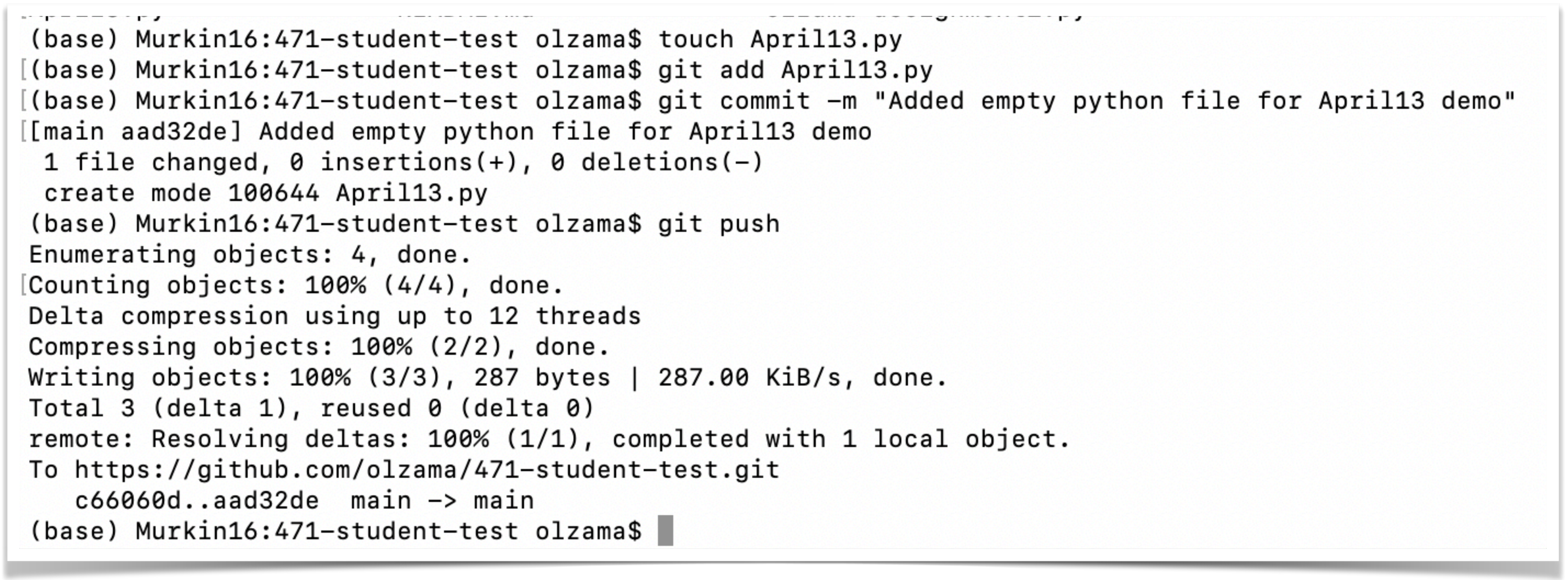
Using git
with a GUI

- GitHub is a GUI!
- VS Code also has a GUI for git!
    - I also use **SourceTree**
        - Not required for this class
            - But it's pretty good
            - Nice visualization
        - Try it if you like!
            - (It's additional setup though)



Graph	Description	Commit	Author	Date
	origin/main origin/HEAD main **Added empty python file for April13 demo**	**aad32de**	**Olga Zamaraeva...**	**Apr 6, 2021 at...**
	Merge branch 'main' of https://github.com/olzama/471-student-test into main	c66060d	Olga Zamaraeva <...	Apr 6, 2021 at 2:...
	Modified the first print statement.	a611d36	Olga Zamaraeva <...	Apr 6, 2021 at 2:...
	Uncommented print statement	583c0cc	Olga Zamaraeva <...	Apr 6, 2021 at 2:...
	Adding python demo file for April 8 lecture.	f28b576	Olga Zamaraeva <...	Apr 6, 2021 at 1:...
	Update README.md	ec4dcae	Olga Zamaraeva <...	Mar 22, 2021 at...
	first commit.	87b9966	Olga Zamaraeva <...	Mar 22, 2021 at...
	Initial commit	745fac2	Olga Zamaraeva <...	Mar 22, 2021 at...

Using git
with command line



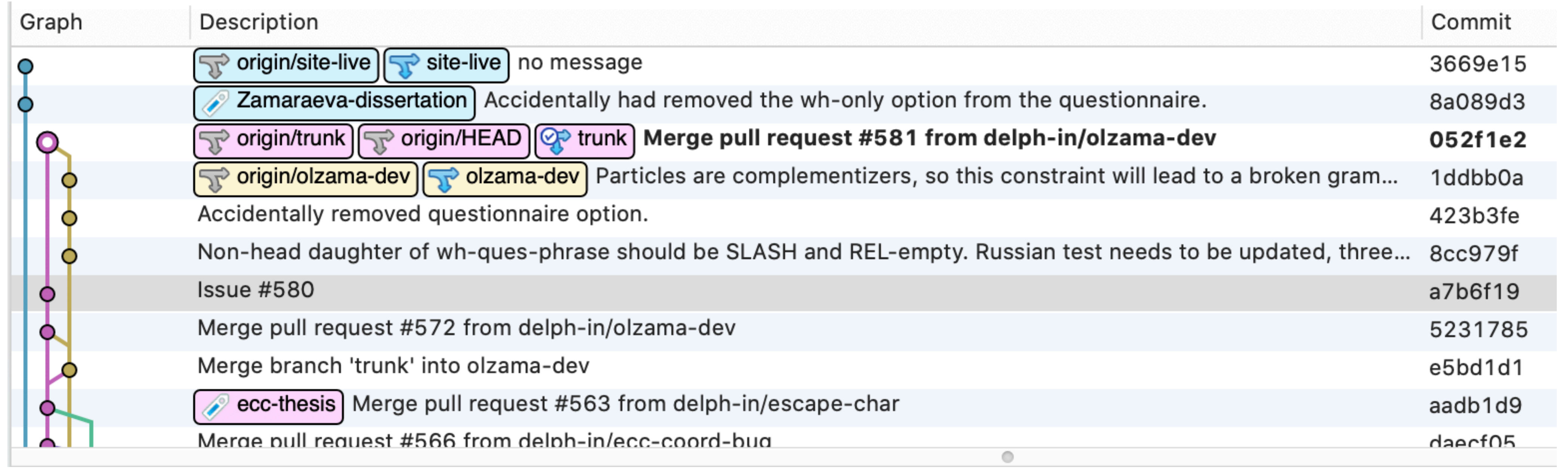
```
(base) Murkin16:471-student-test olzama$ touch April13.py
[(base) Murkin16:471-student-test olzama$ git add April13.py
[(base) Murkin16:471-student-test olzama$ git commit -m "Added empty python file for April13 demo"
[[main aad32de] Added empty python file for April13 demo
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 April13.py
(base) Murkin16:471-student-test olzama$ git push
Enumerating objects: 4, done.
[Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 287 bytes | 287.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/olzama/471-student-test.git
   c66060d..aad32de  main -> main
(base) Murkin16:471-student-test olzama$ 
```

- add, commit, push, pull
  - And merge, if required
  - Sometimes you can “force push” using command line when all else fails
  - Hopefully no need for that in this class
    - But good to know
    - Idea: Command line is more powerful when it comes to git than GUIs

Version control

Going back in time

- If you want to **go back** to a previous version:
    - Recommended for now, in VS Code:
        - Install **GitLens extension**
            - Will need to **log in** GitHub with it
        - Use File History to **restore** the version you want
        - Can also do **commit** history -> **reset**, for all files
    - Also ways to do that in command line
        - command line will **always** work
        - But can be more confusing, which command:
            - git revert / git reset
            - “Revert” **destroys** the “reverted” commit
            - “Reset” resets your working copy to **that** commit



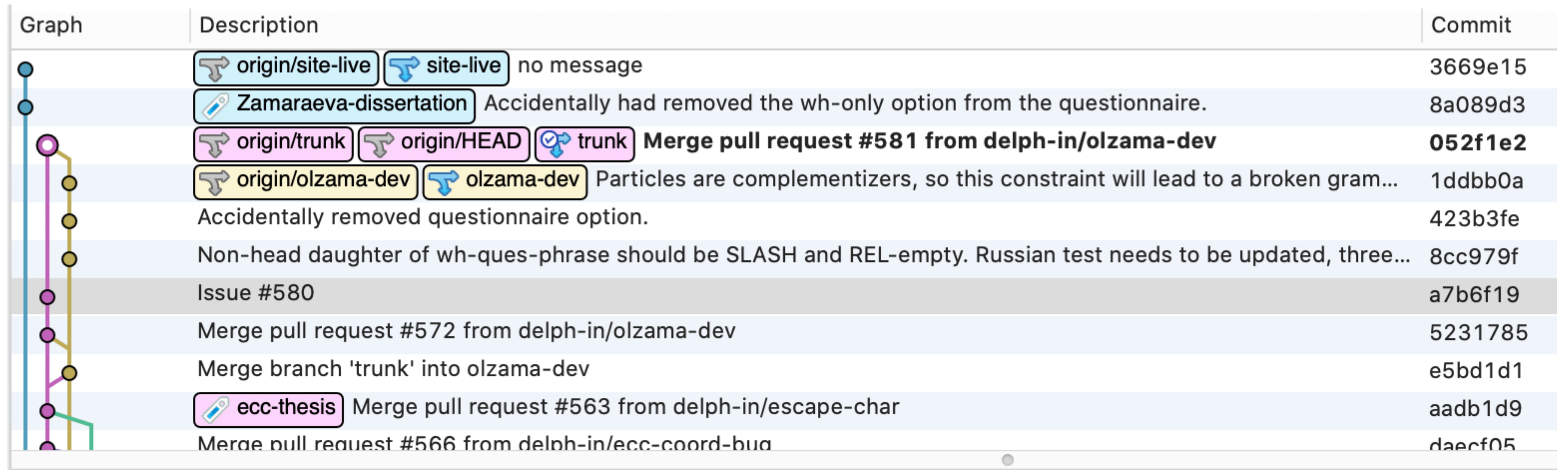
Graph	Description	Commit
	origin/site-live site-live no message	3669e15
	Zamaraeva-dissertation Accidentally had removed the wh-only option from the questionnaire.	8a089d3
	origin/trunk origin/HEAD trunk **Merge pull request #581 from delph-in/olzama-dev**	**052f1e2**
	origin/olzama-dev olzama-dev Particles are complementizers, so this constraint will lead to a broken gram...	1ddbb0a
	Accidentally removed questionnaire option.	423b3fe
	Non-head daughter of wh-ques-phrase should be SLASH and REL-empty. Russian test needs to be updated, three...	8cc979f
	Issue #580	a7b6f19
	Merge pull request #572 from delph-in/olzama-dev	5231785
	Merge branch 'trunk' into olzama-dev	e5bd1d1
	ecc-thesis Merge pull request #563 from delph-in/escape-char	aadb1d9
	Merge pull request #566 from delph-in/ecc-coord-bug	daecf05

A repository shown in SourceTree software (compatible with git)

Version control

Branches

- Keep different development tracks
  - With different commits etc.
- A branch can be either:
  - abandoned, if the track didn’t work out
  - Or merged into main
- Consider:
  - Having a branch for each major step of HW
  - Merging it into main once satisfied

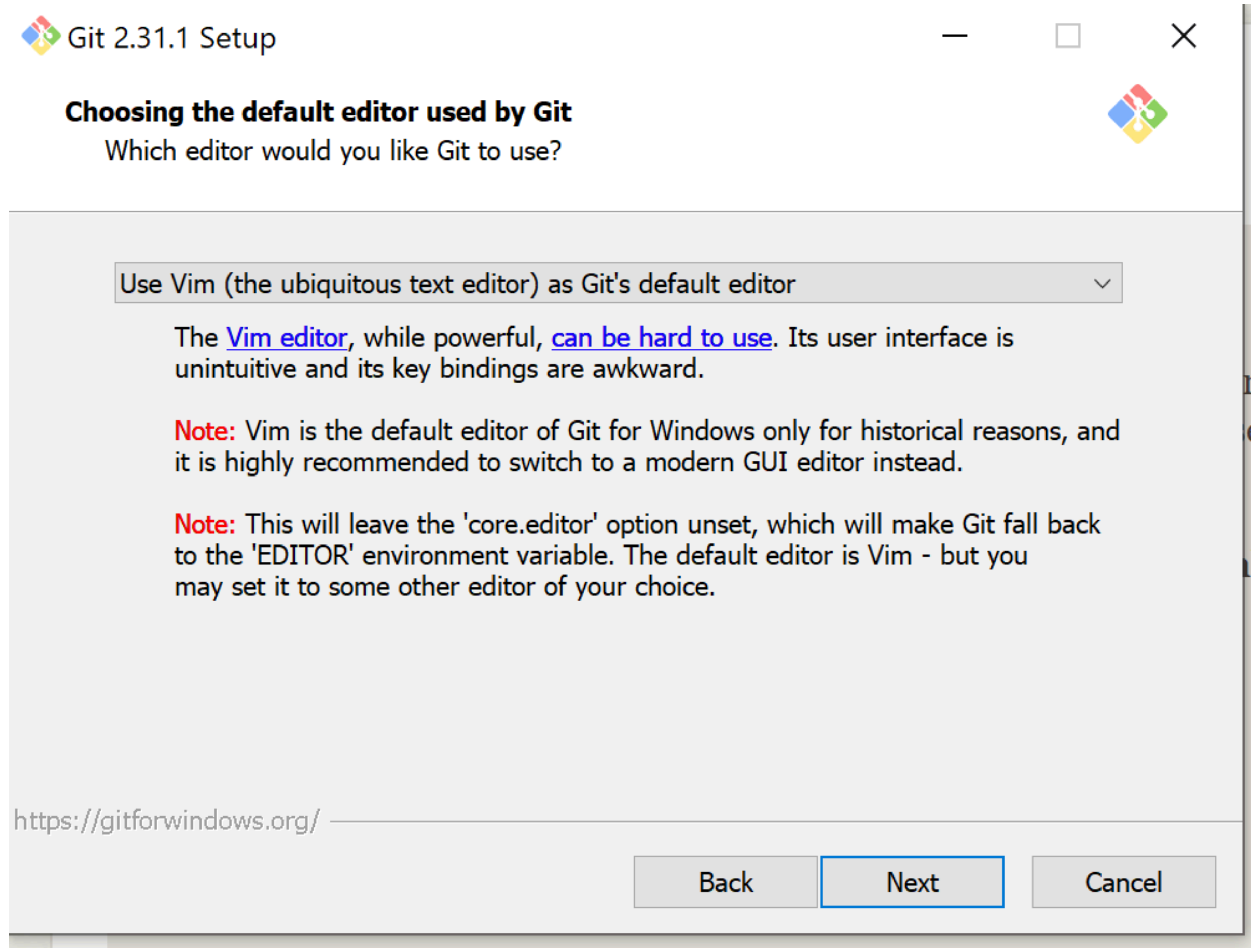


Graph	Description	Commit
	origin/site-live site-live no message	3669e15
	Zamaraeva-dissertation Accidentally had removed the wh-only option from the questionnaire.	8a089d3
	origin/trunk origin/HEAD trunk **Merge pull request #581 from delph-in/olzama-dev**	**052f1e2**
	origin/olzama-dev olzama-dev Particles are complementizers, so this constraint will lead to a broken gram...	1ddbb0a
	Accidentally removed questionnaire option.	423b3fe
	Non-head daughter of wh-ques-phrase should be SLASH and REL-empty. Russian test needs to be updated, three...	8cc979f
	Issue #580	a7b6f19
	Merge pull request #572 from delph-in/olzama-dev	5231785
	Merge branch 'trunk' into olzama-dev	e5bd1d1
	ecc-thesis Merge pull request #563 from delph-in/escape-char	aadb1d9
	Merge pull request #566 from delph-in/ecc-coord-bug	daecf05

Branches in SourceTree software (compatible with git)

Using git
with command line

- Forgetting to write a commit message in command line mode will open a command-line text editor
  - These aren't trivial to exit :)
  - By default, git opens the VIM editor
  - It can be exited by hitting ":wq"
  - You can also merge/resolve conflicts there
    - (I'd never do that unless I have to, but some people prefer them.)
    - (Edit files using GUI editors, use command line to commit and push if necessary or if you find that easier)



Choosing the default editor used by Git
Which editor would you like Git to use?

Use Vim (the ubiquitous text editor) as Git's default editor

The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.

Note: Vim is the default editor of Git for Windows only for historical reasons, and it is highly recommended to switch to a modern GUI editor instead.

Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.