Computational Methods for Linguists

Ling 471

Olga Zamaraeva (Instructor)
Yuanhe Tian (TA)

04/15/21

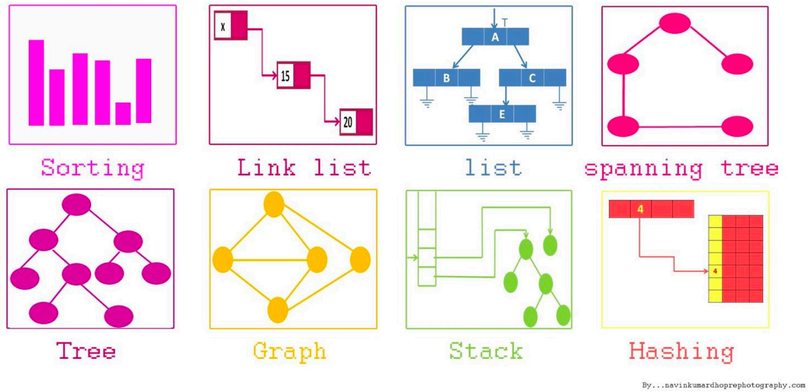
Reminders

- Start Assignment 2 (Part 1)
  - Make sure you figured out:
    - Running configurations
    - Debugging
    - Use **office hours** if there's issues with the above
      - Also monitor the discussion board
      - May use e.g. PyCharm if VS Code is not working
- Blog 2
  - Due April 20

Plan for today

- Further preparation for Assignment 2:
  - Loops
  - The *dict* data structure
    - Aka *dictionary* (C#), *map* (Java)…
  - Input/Output and string formatting

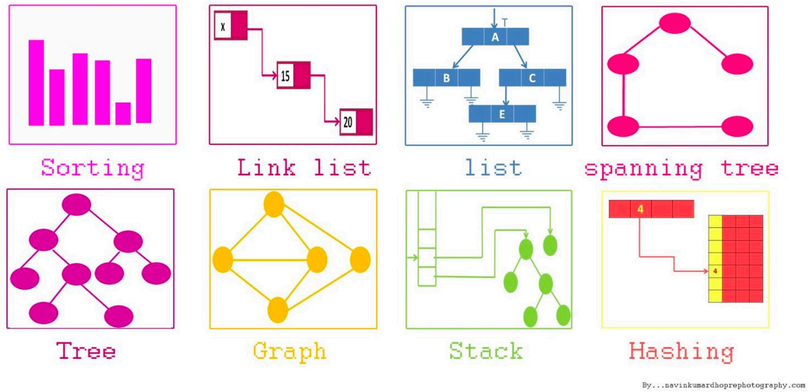
Data structures



https://medium.com/@fabianterh/how-to-improve-your-data-structures-algorithms-and-problem-solving-skills-af50971cba60

- **Proper** study can take one **or more** quarters
- We will be **opportunistic**
  - Only discuss things we **happen to need** (for HW)
  - This is not ideal but **OK**
  - Keep in mind that data structures are an important and well-studied topic:
    - how different languages implement them
    - what depends on which one you use
    - etc

Algorithms



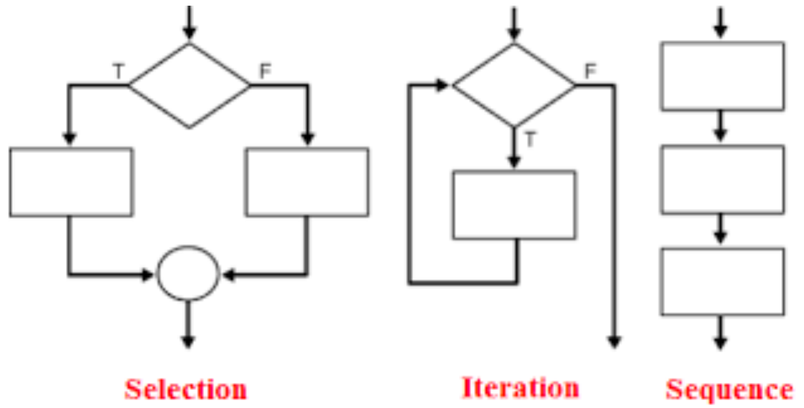
https://medium.com/@fabianterh/how-to-improve-your-data-structures-algorithms-and-problem-solving-skills-af50971cba60

- Proper study of programming includes **algorithms**
  - What **types** there are
  - **Why** the differences matter
    - Hint: some are more **efficient** than others
    - ...some are **simpler** than others!
      - means **fewer** bugs
- **No** proper study of algorithms in this class!
  - We will mostly use **in-built** algorithms
  - Which is **good** practice* ultimately
    - *understanding still important in the long run!

But first, more control flow!

Loops

Loops
and iteration

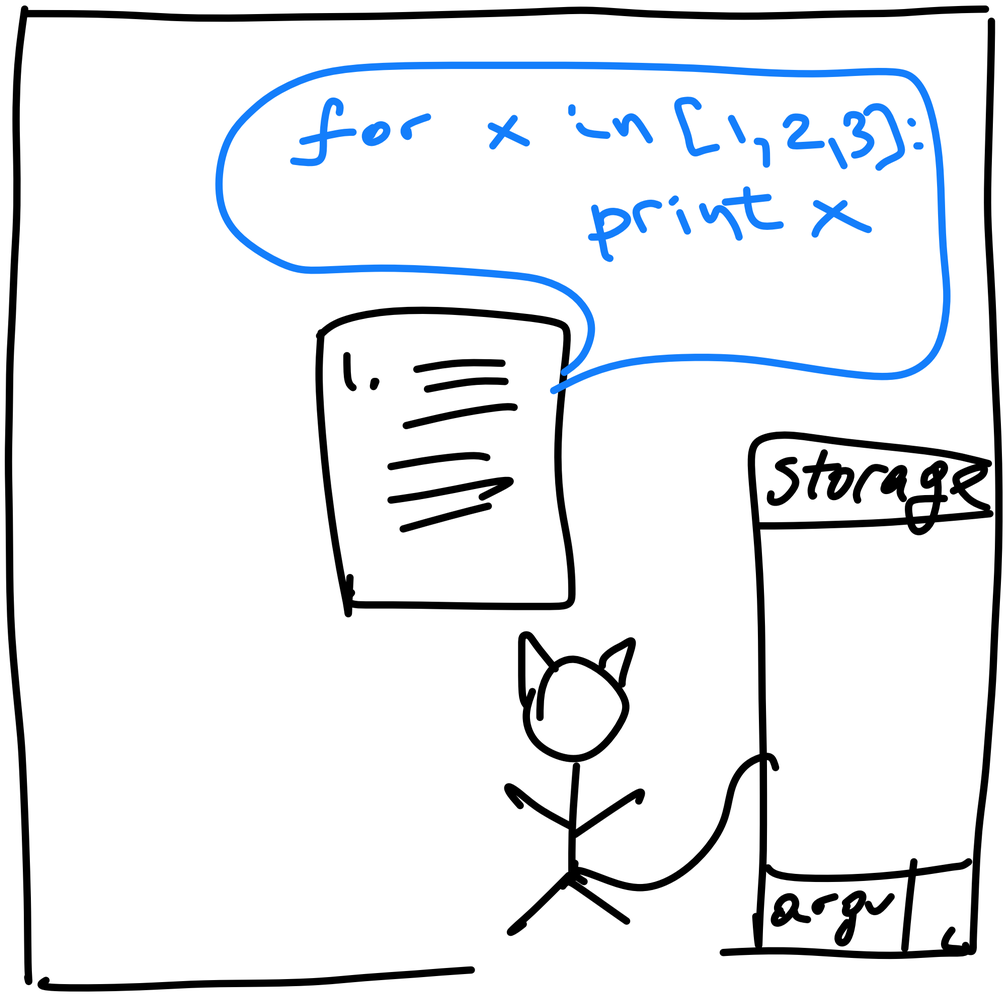


http://net-informations.com/python/flow/default.htm

- A way to control execution flow
  - (Which statement to execute?)
- The loop continues while the declared condition is True
  - e.g. “while x < 100”
  - Once it stops being true, program exits the loop
    - suppose we are incrementing x at each step
- (In)famous *infinite loop:*
  - *While True: {some code}*
  - Sometimes less explicit :)

Loops
while- and for-

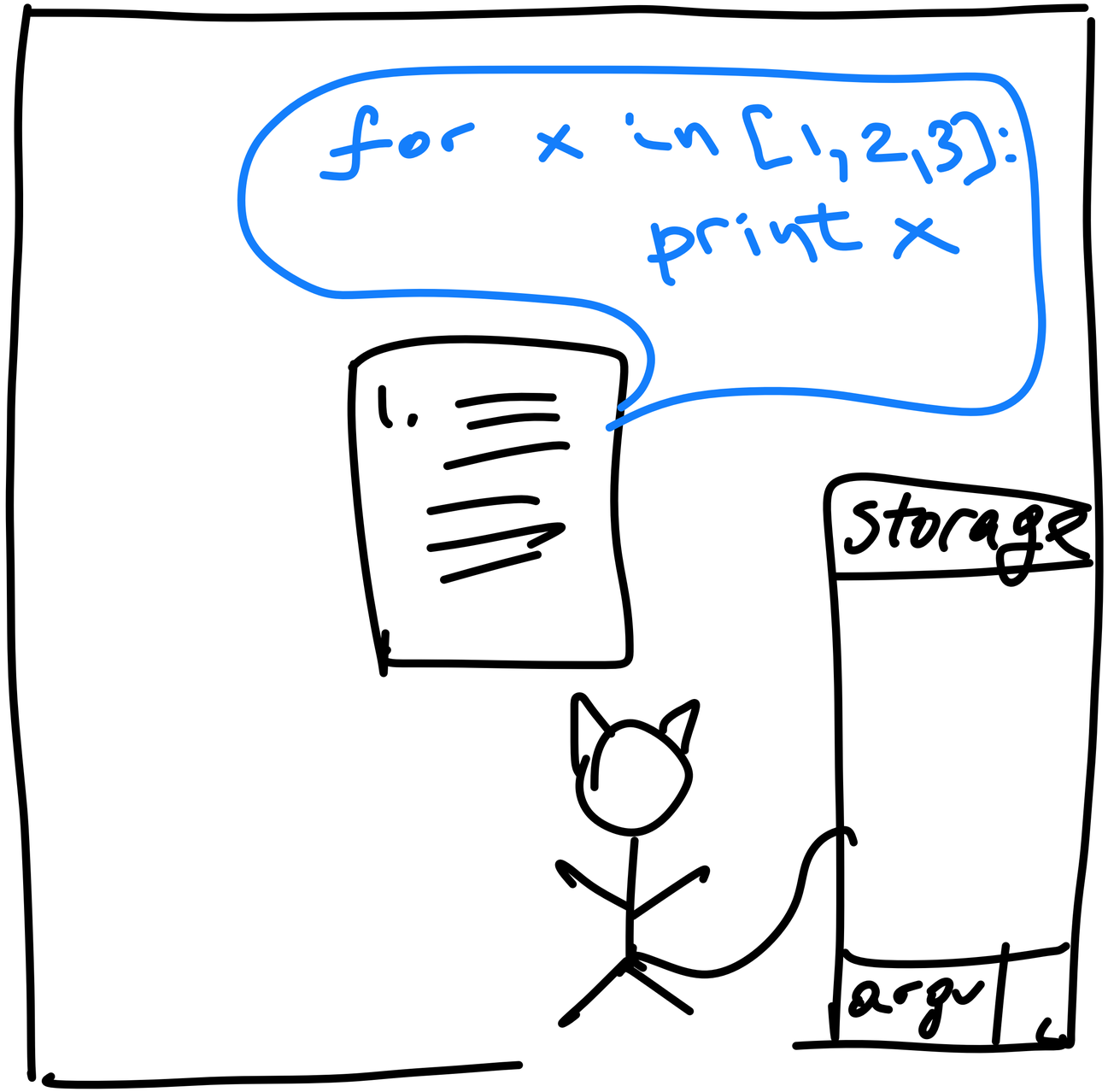
- *while* and *for* are both python keywords
  - *while* translates straightforwardly into:
    - “Do something, while this condition is true”
  - *for* is used for **iteration** over lists of values
- A **for-loop** still needs a condition to be true
  - But that condition is less explicit:
  - Do something, while the iteration variable is in the **range** of values
  - Can state what the range is explicitly or use range()



For loop

Explicit range

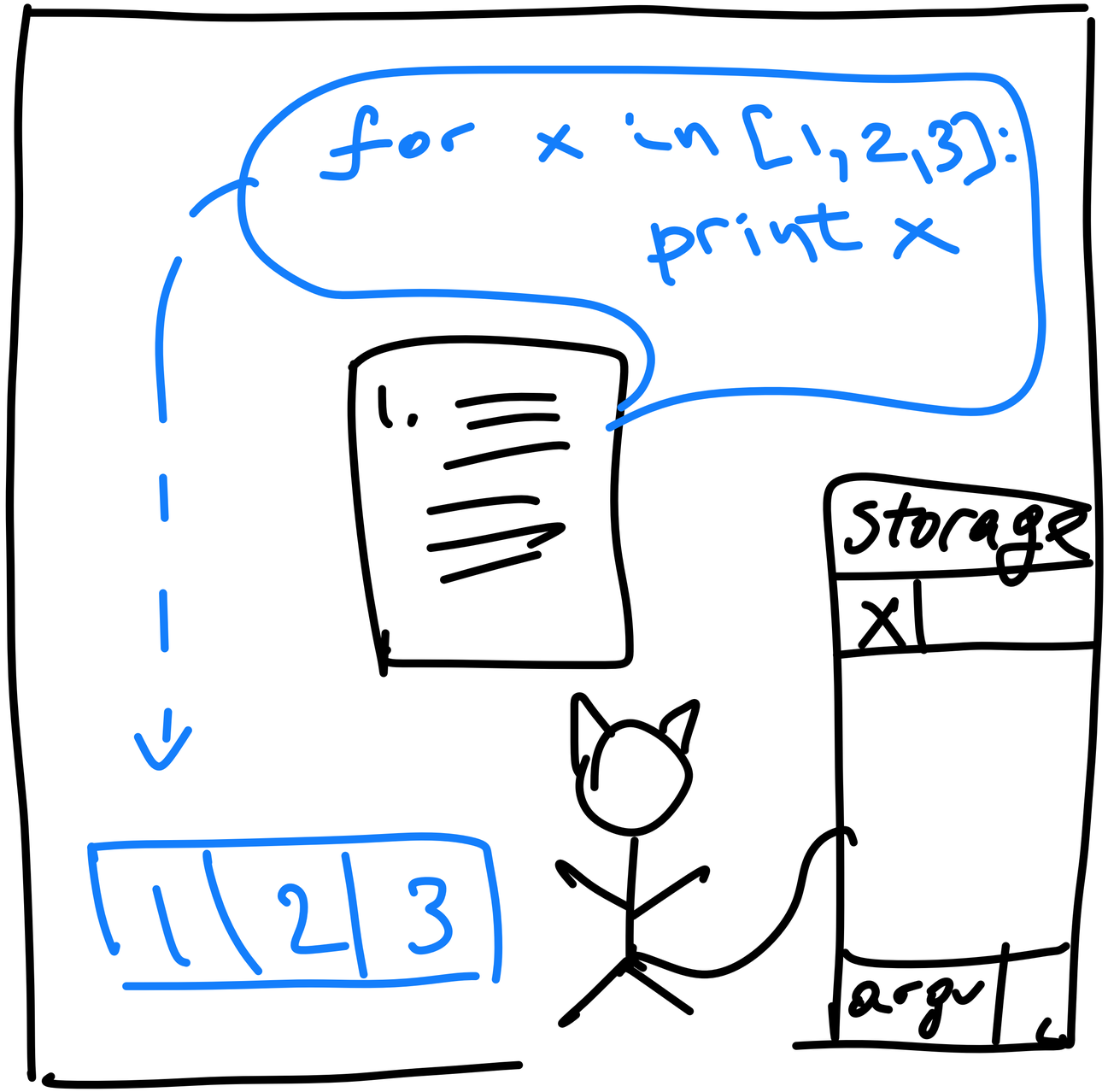
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

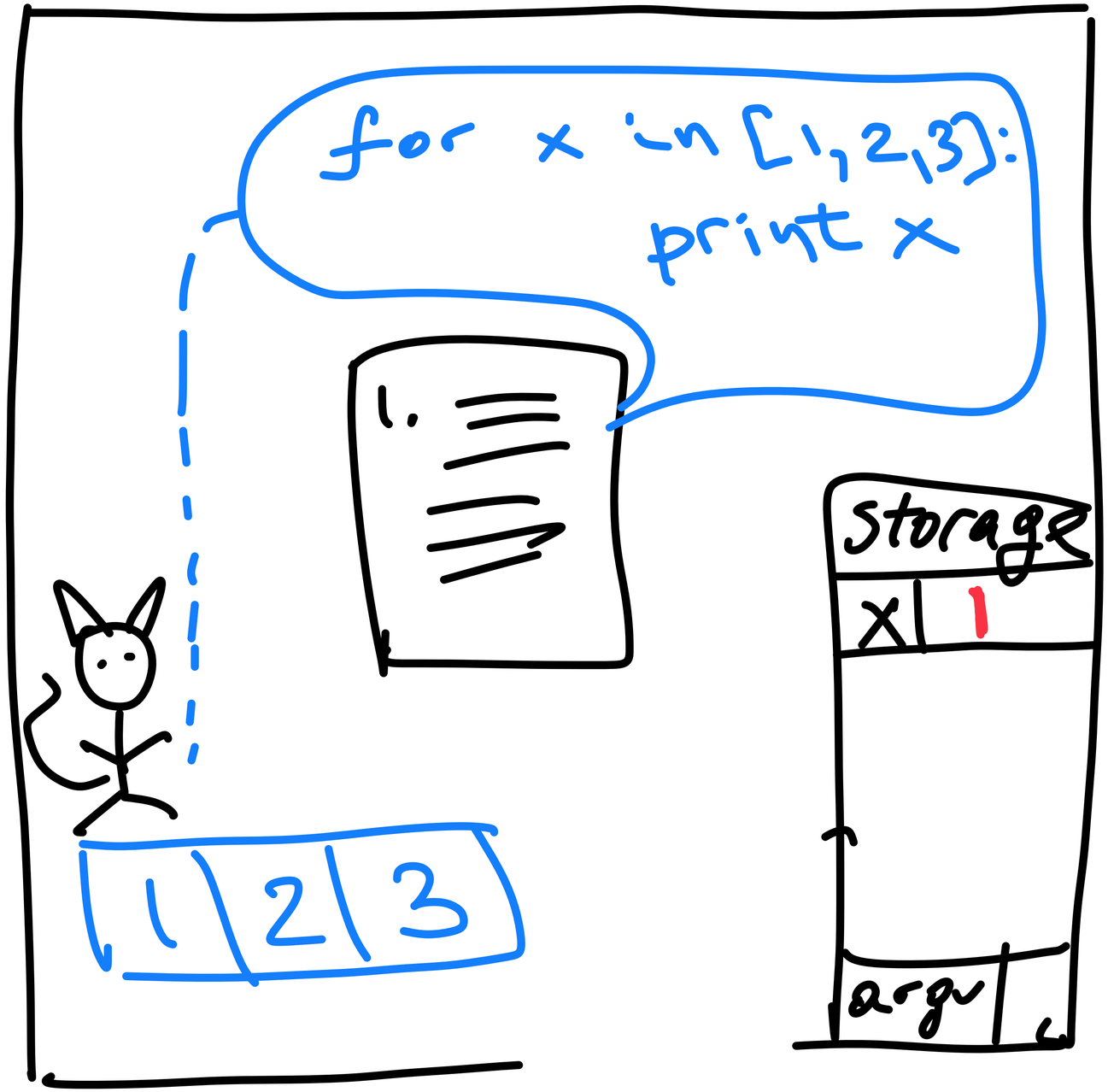
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

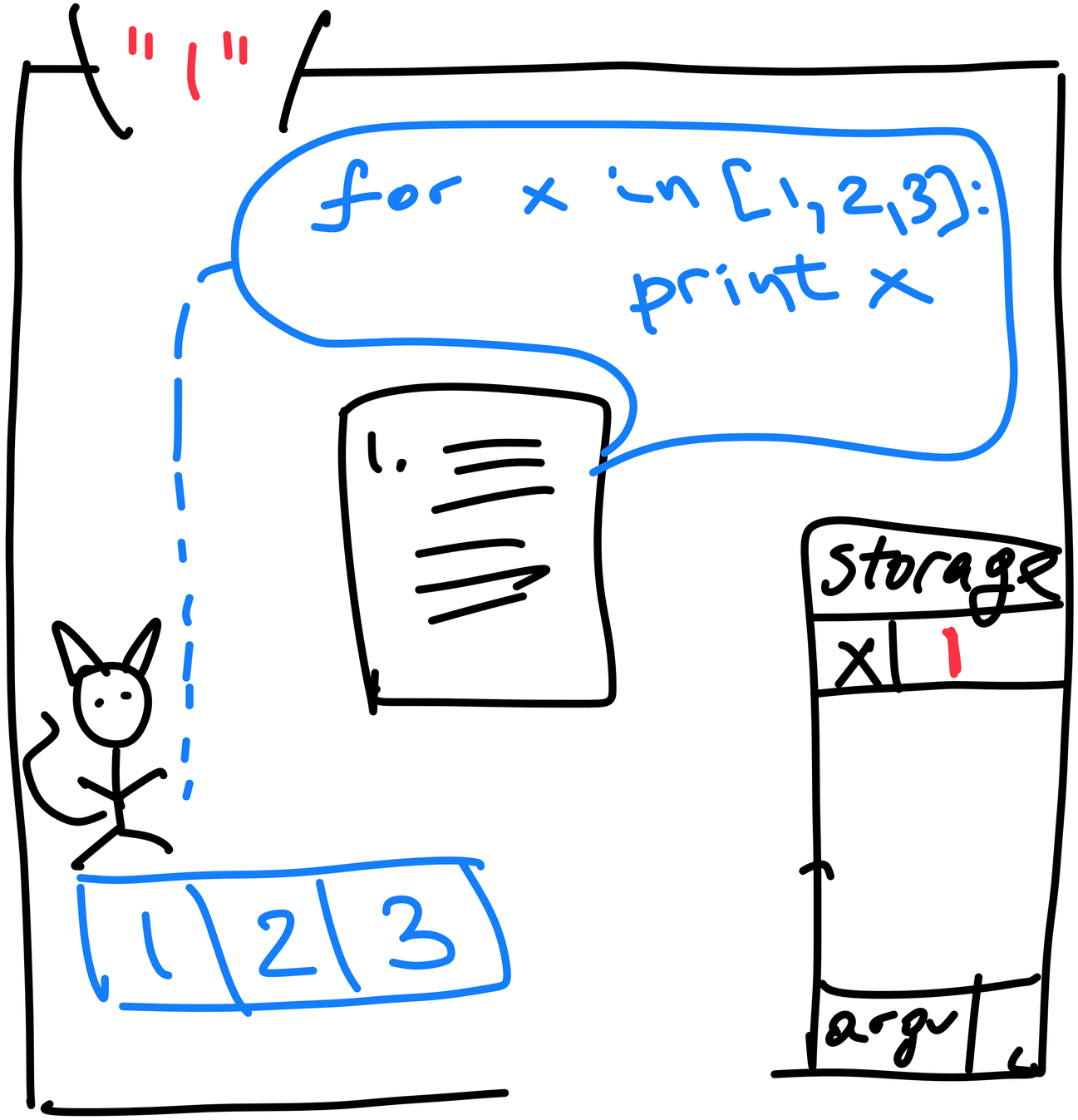
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
  - changing the value is **implicit** in the **for-**syntax
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

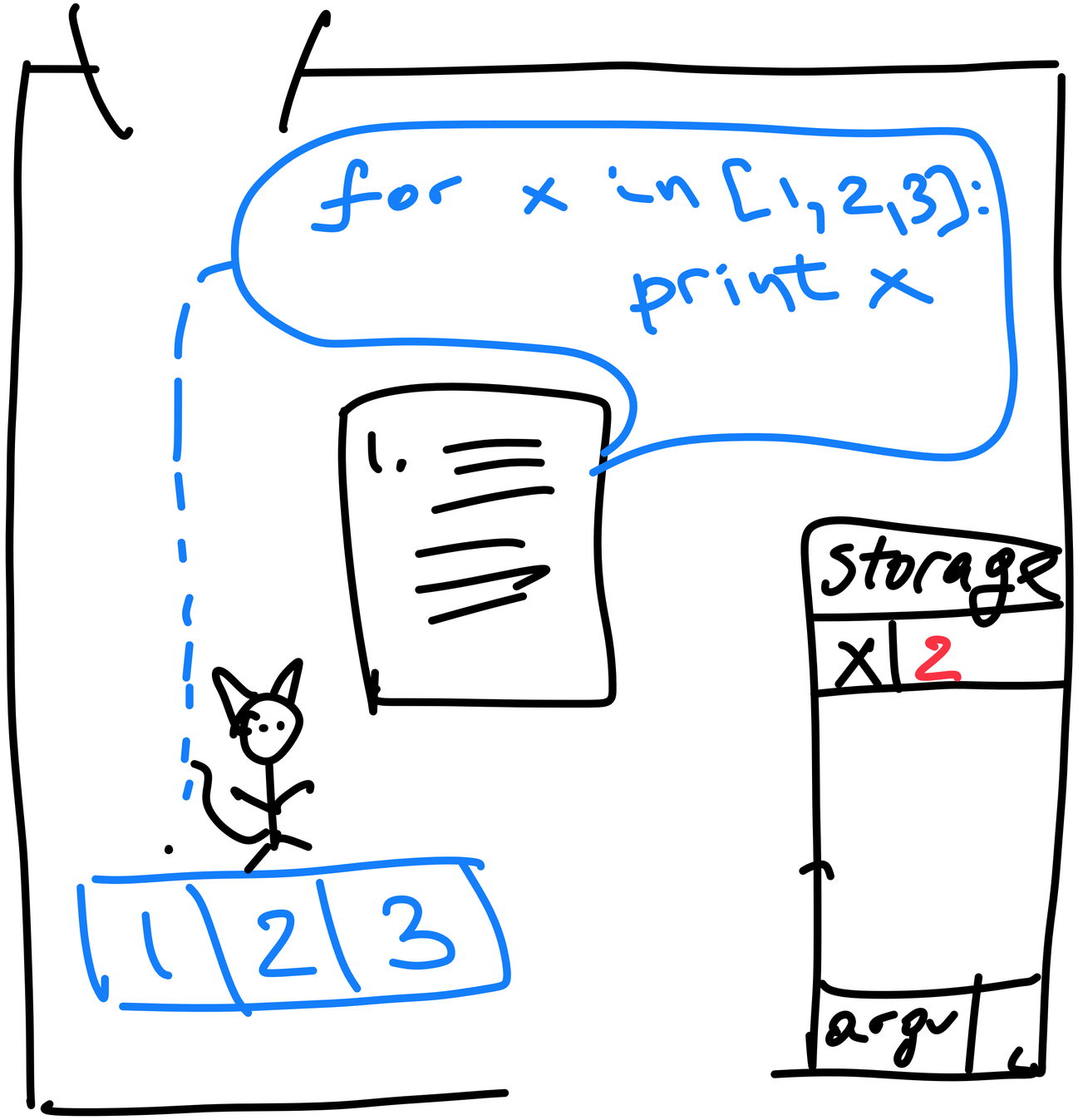
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
  - changing the value is **implicit** in the **for-**syntax
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

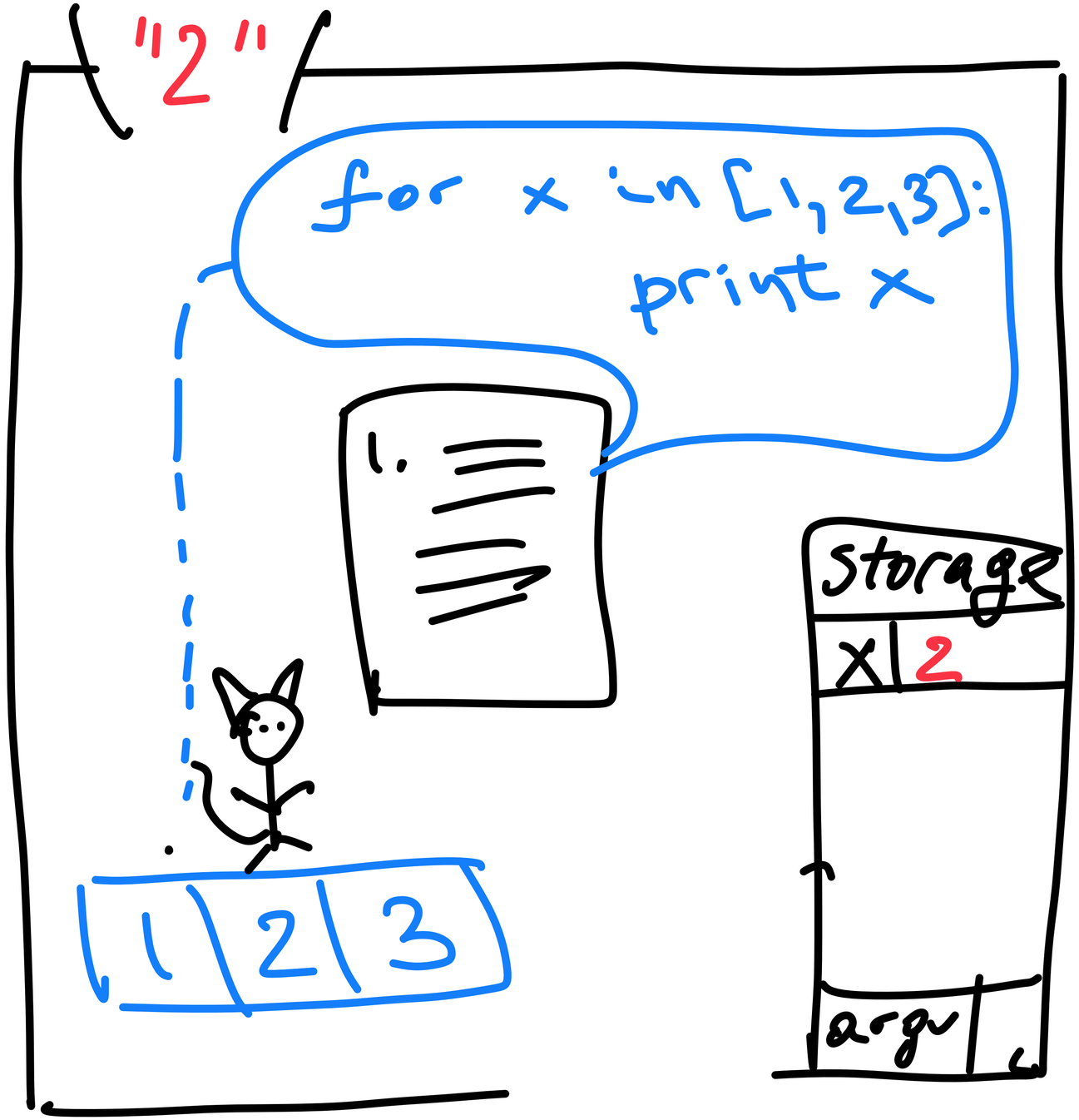
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
  - changing the value is **implicit** in the **for-**syntax
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

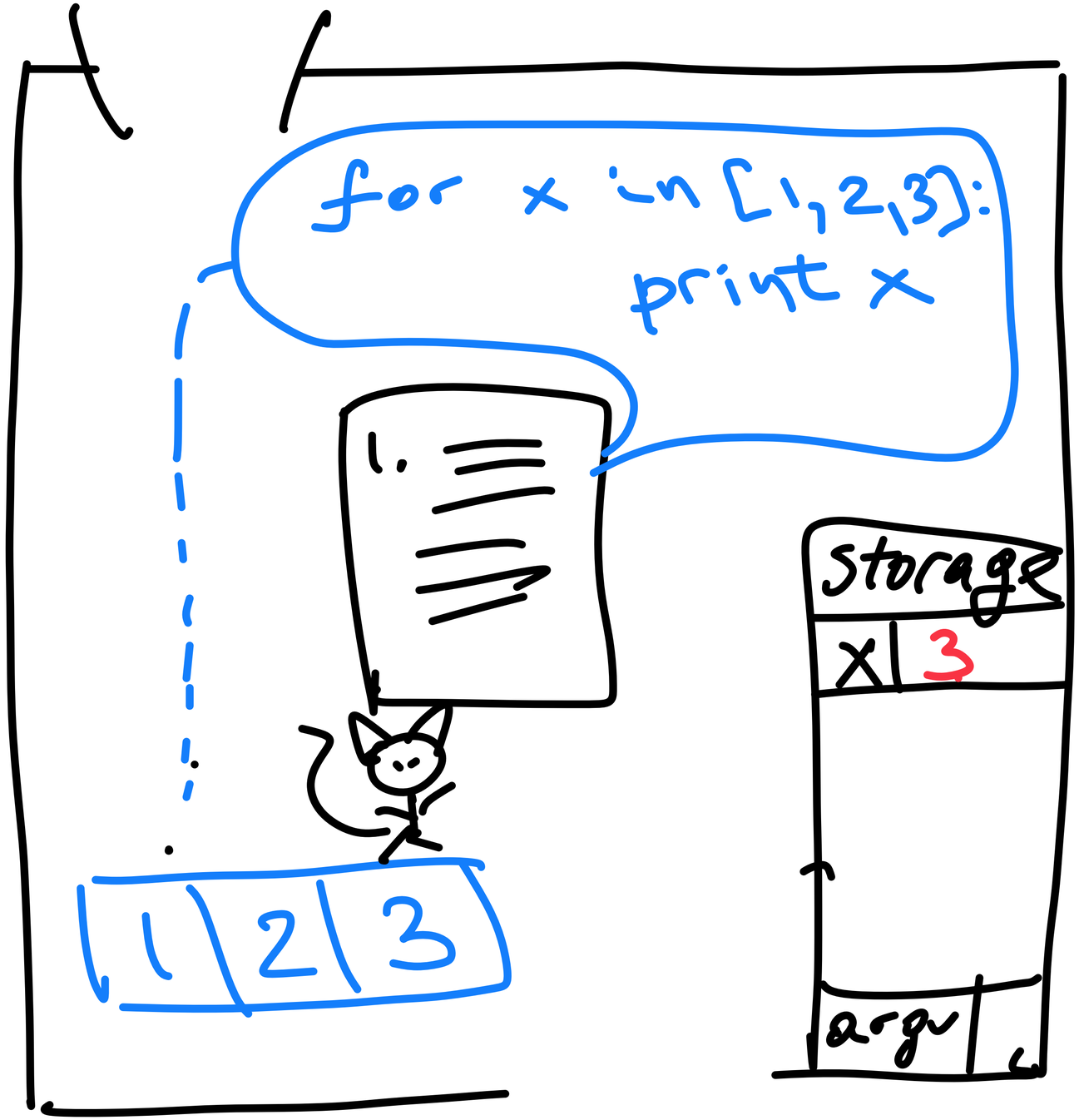
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
  - changing the value is **implicit** in the **for-**syntax
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

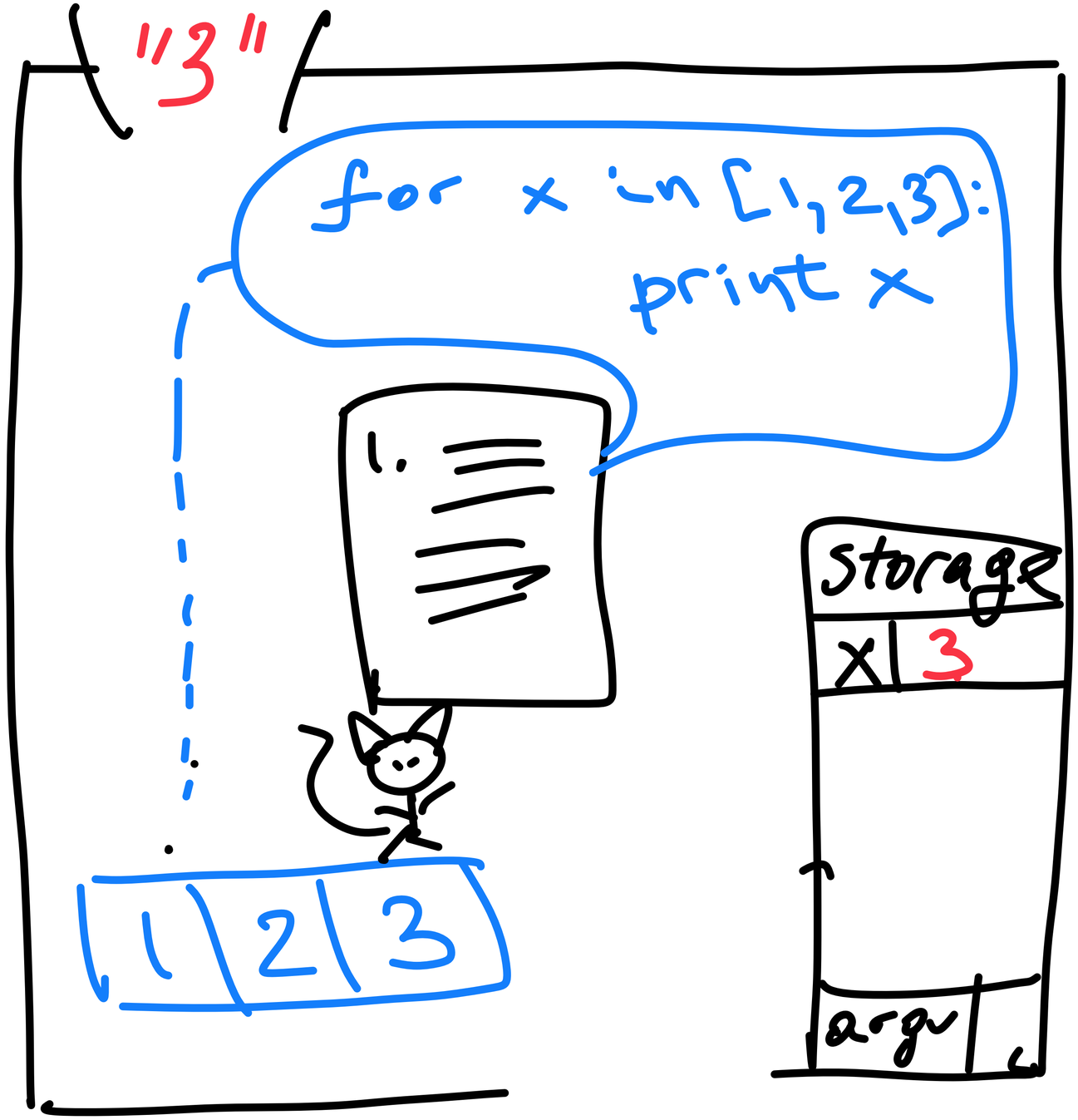
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
  - changing the value is **implicit** in the **for-**syntax
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

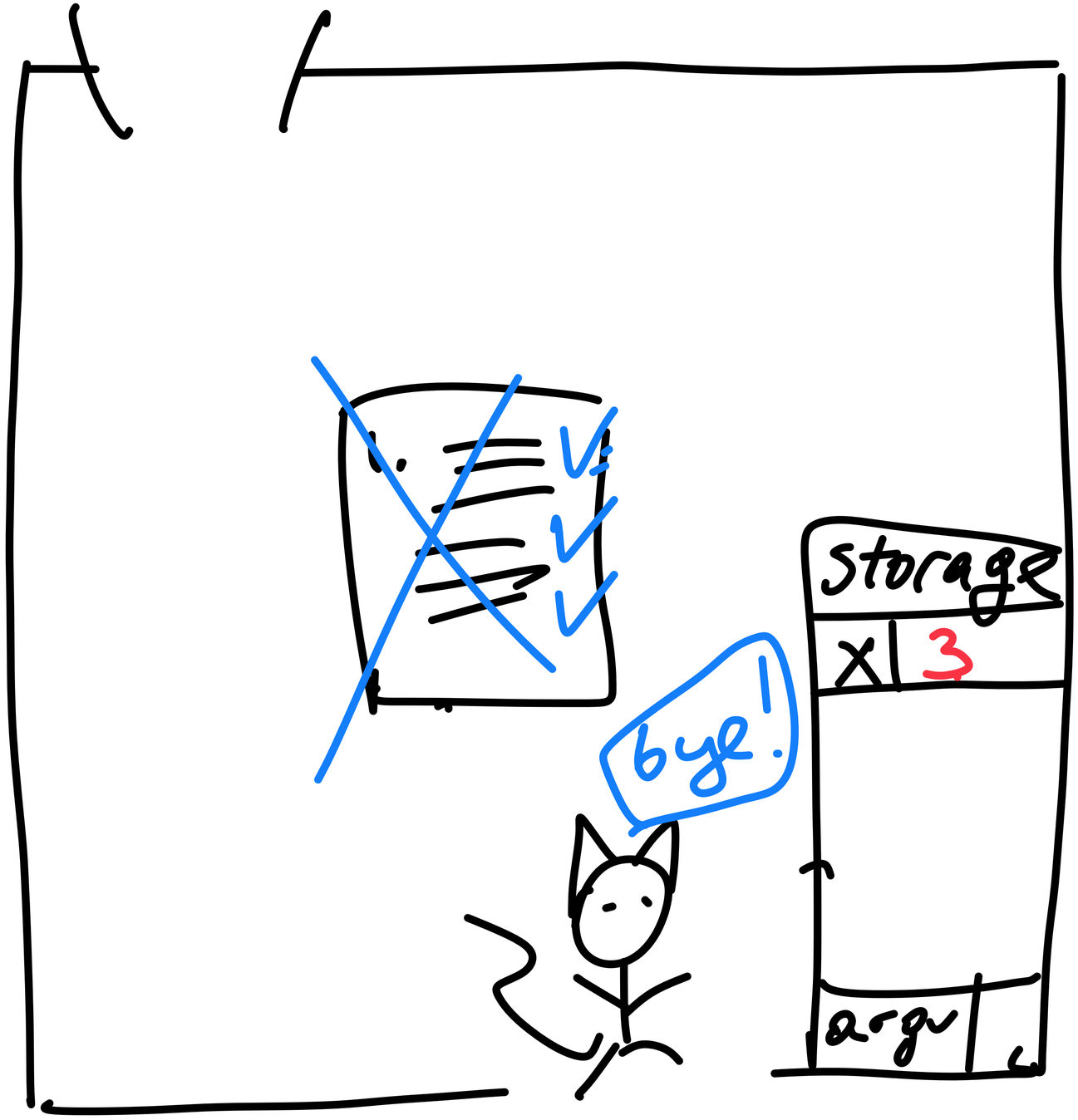
- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
  - changing the value is **implicit** in the **for-**syntax
- [1,2,3] is equivalent to:
  - range(1,4)



For loop

Explicit range

- Variable x is initialized
- Its value will be changing as we iterate through the list [1,2,3]
  - changing the value is **implicit** in the **for-**syntax
- [1,2,3] is equivalent to:
  - range(1,4)
- Once **out** of range, the loop **condition** is **no longer** true and we **exit**



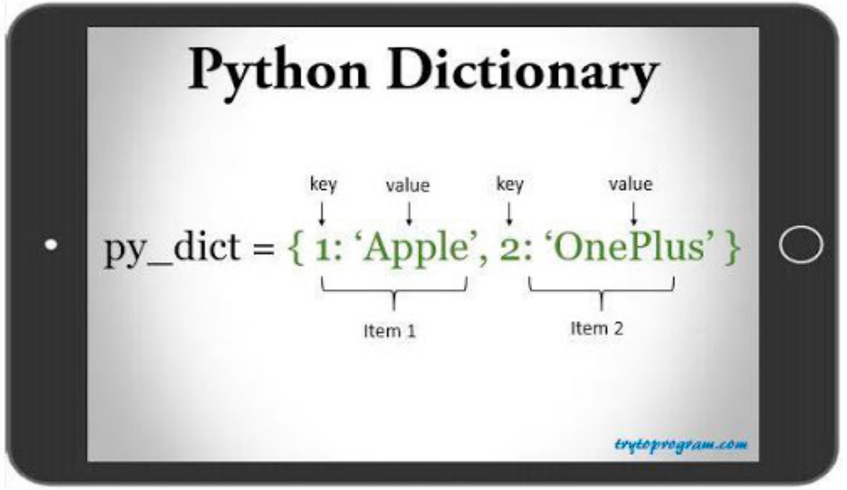
For-loop demo

Dict data structure

Dict

Dictionary, map...

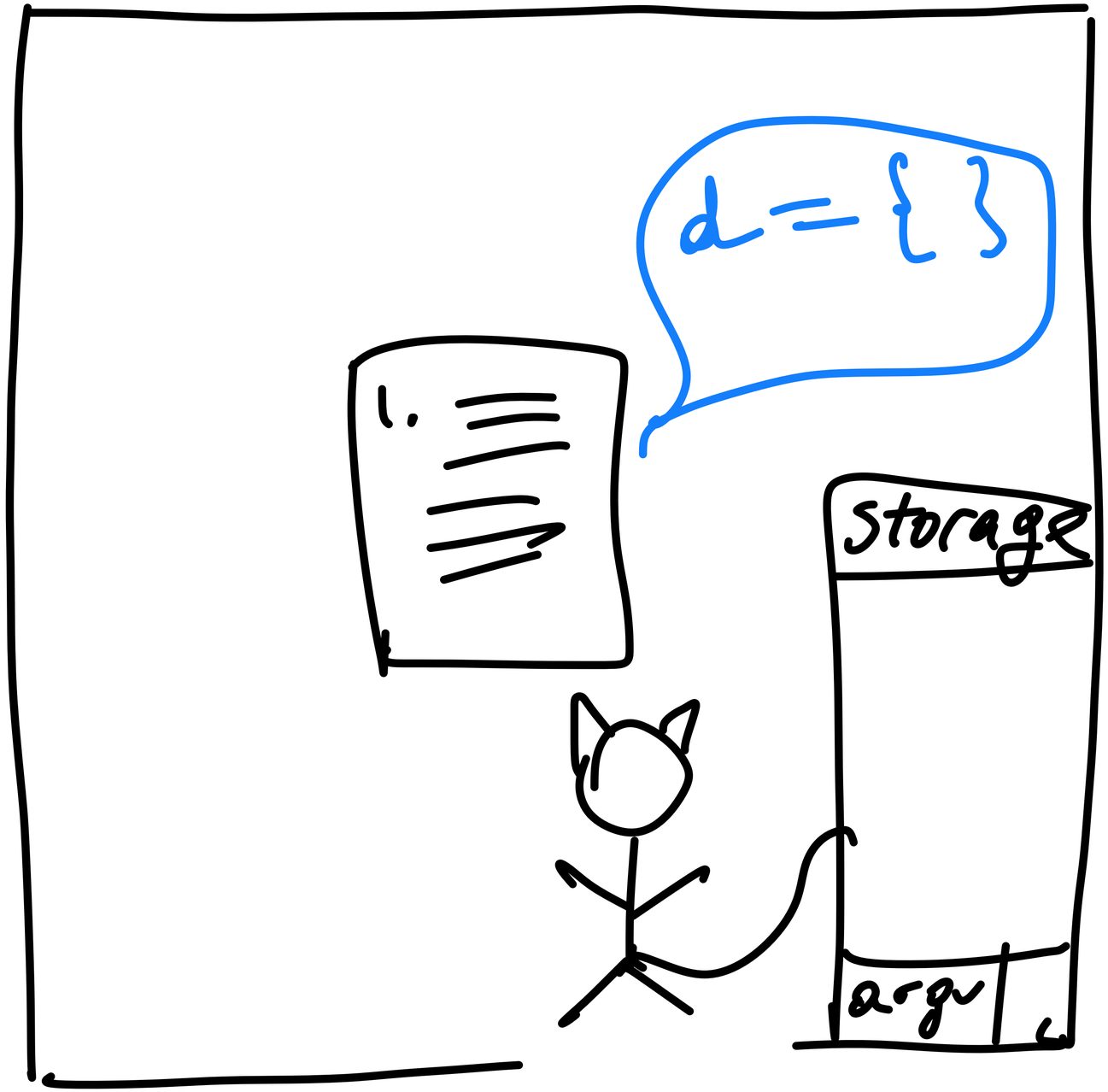
- A data structure for fast look up
- A *dict* is a **table** which maps **keys** to **values**
  - (It is **fast** because the interpreter can **immediately** access a key **without** iterating over **all** of them)
  - But **sometimes**, you need to **iterate** over all keys or all items in a dict
    - e.g. when you want to **update each item**
- **Tables** are very important in data science
  - because data is usually **vectorized**
  - vectors become **rows** in tables
  - more on this **later** in the course!



http://www.trytoprogram.com/python-programming/python-dictionary/

Dict

- Initializing empty dict
- d = {}

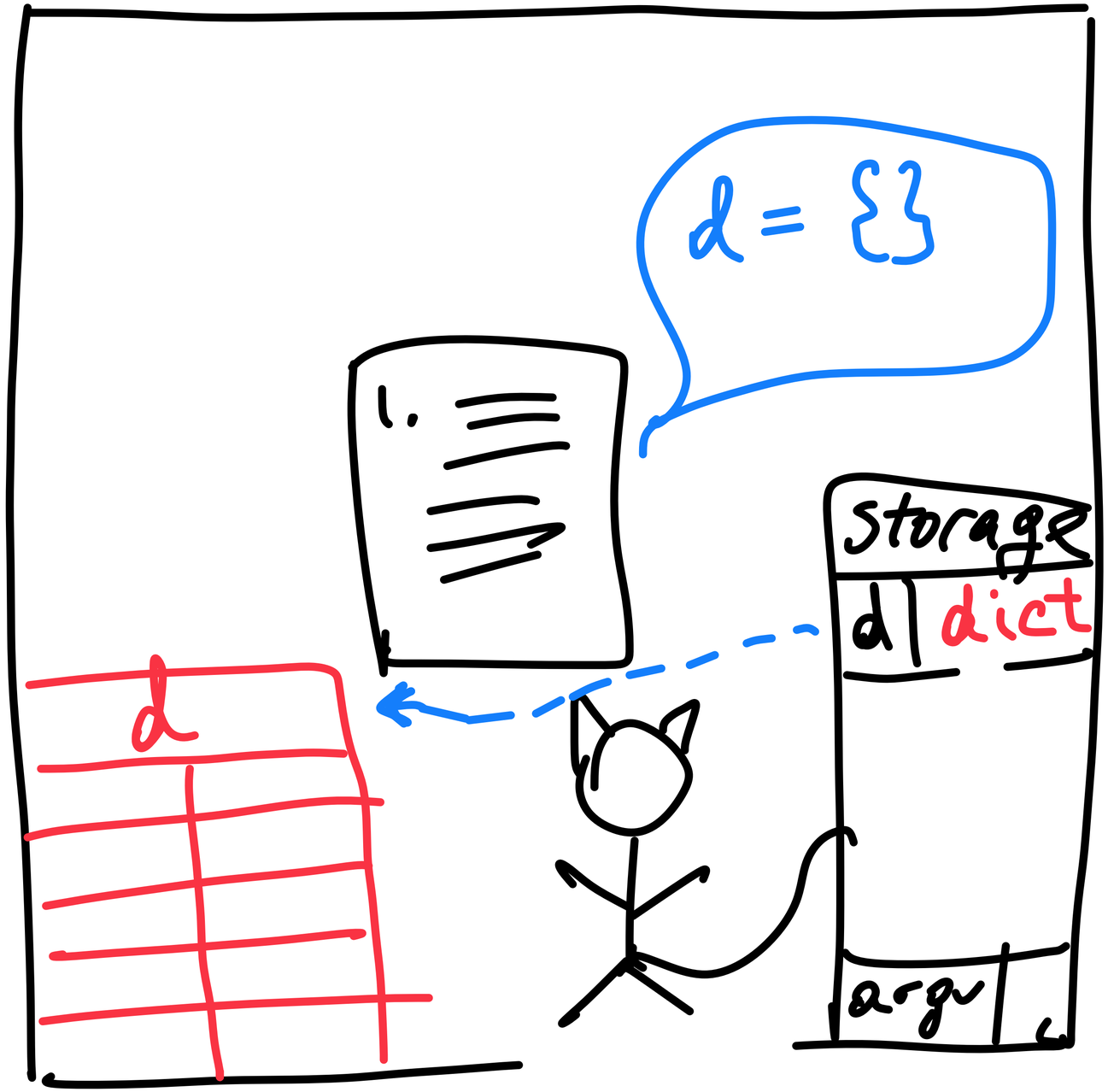


Dict

- Initializing empty dict
  - Python knows {} is an empty *dict*
    - It knows d can contain keys and values, mapped to each other!

Dict

- For now, it is an empty table

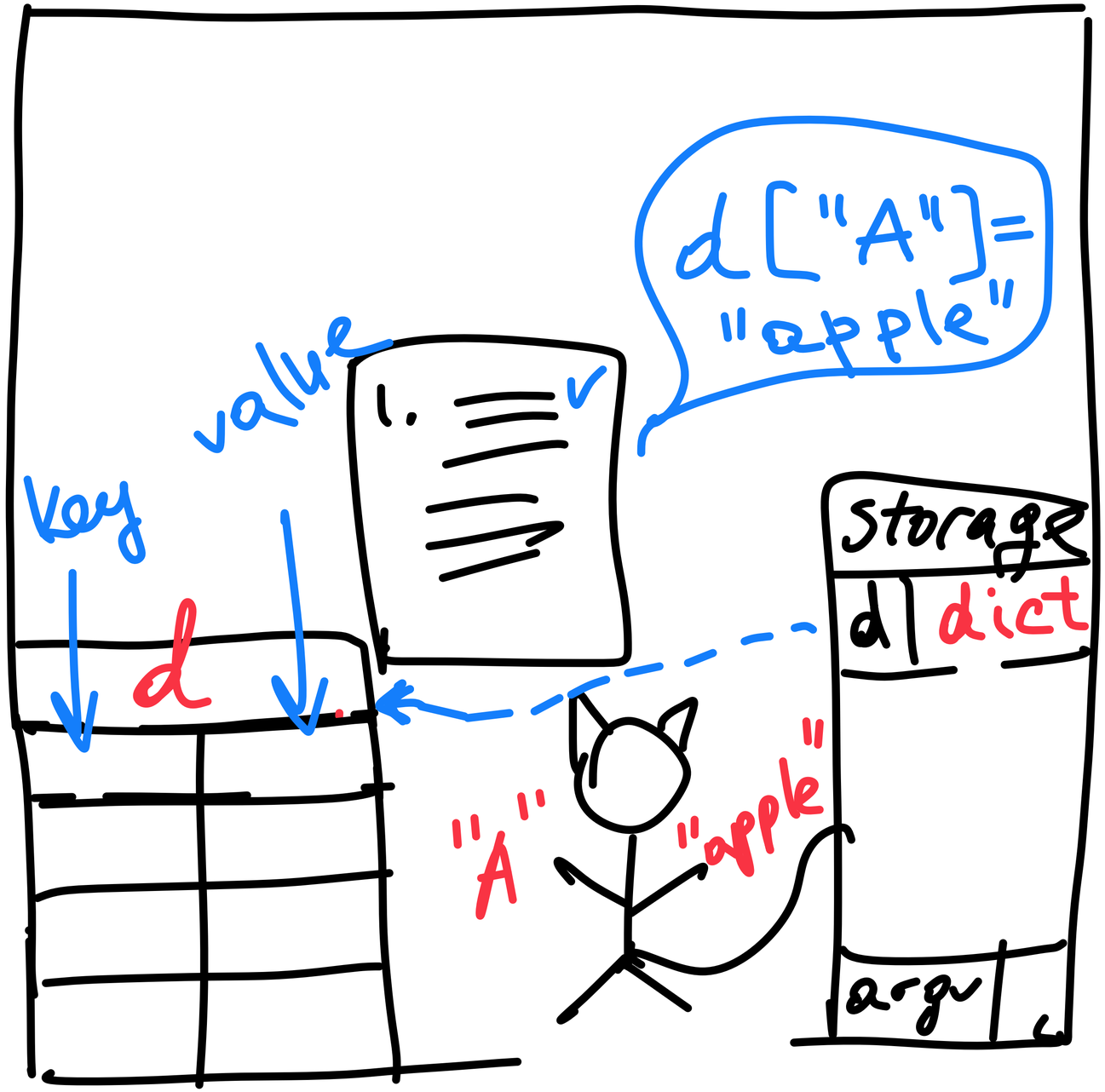


Dict

- Suppose instructions say:
  - Assign value “apple” to the key “A”
    - Keys can be anything!
      - Numbers, characters, words…

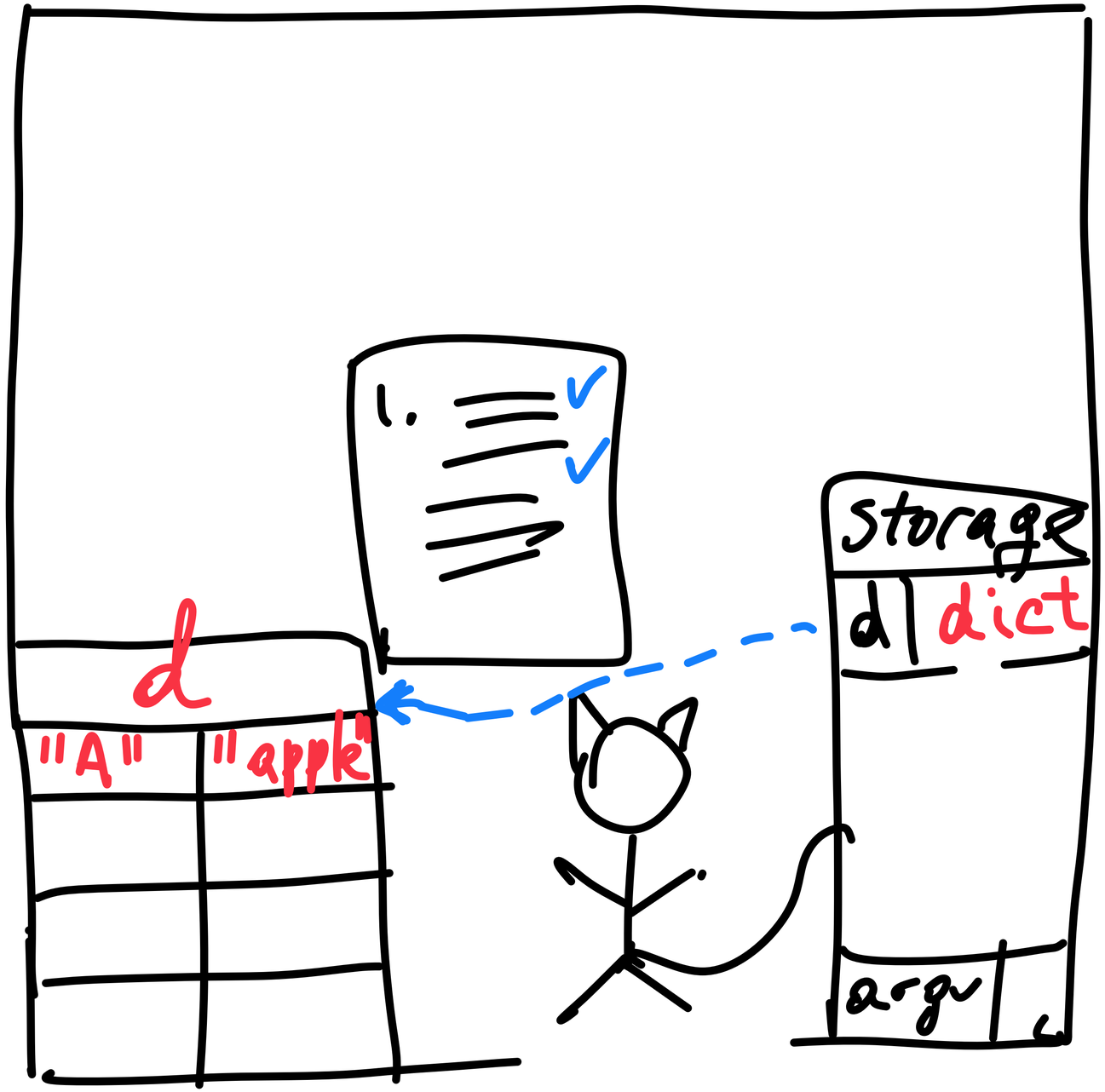
Dict

- Suppose instructions say:
  - Assign value “apple” to the key “A”
    - Keys can be anything!
      - Numbers, characters, words…



Dict

- Suppose instructions say:
  - Assign value “apple” to the key “A”
    - Keys can be anything!
      - Numbers, characters, words…

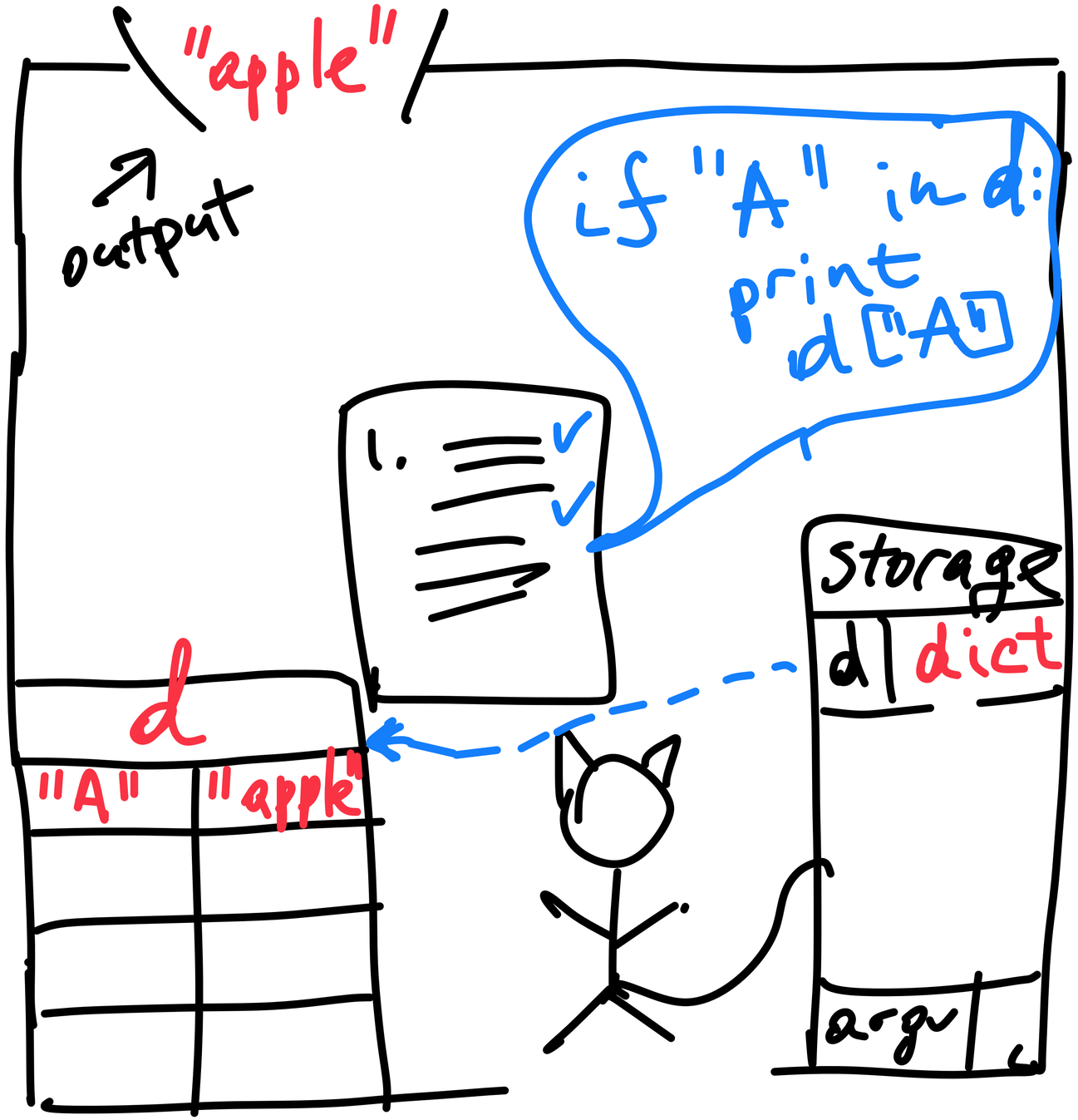


Dict

- Next instruction:
  - If there is a **key** “A” in the dict:
    - Print the **value** corresponding to that key

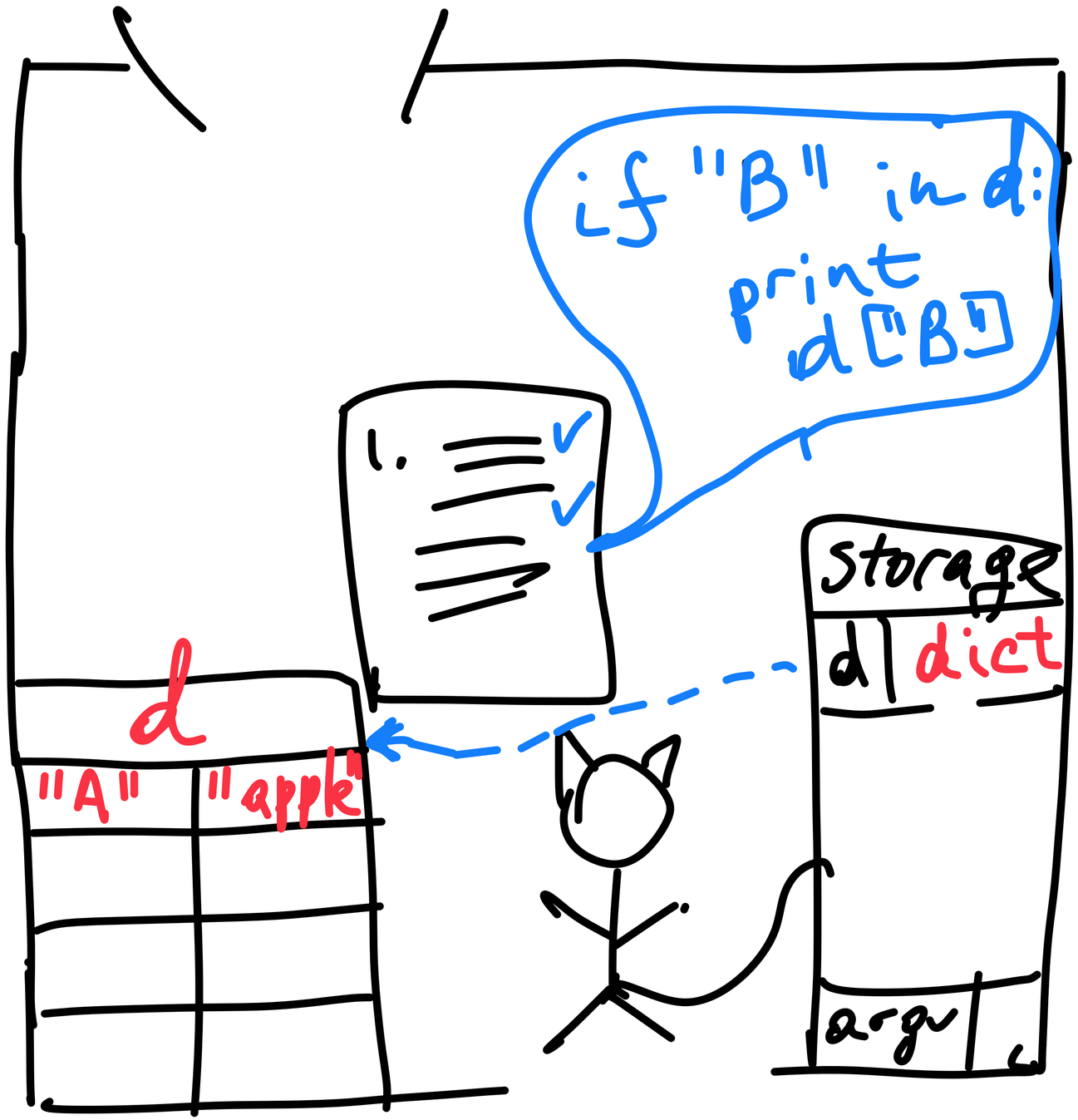
Dict

- Next instruction:
  - If there is a **key** “A” in the dict:
    - Print the **value** corresponding to that key



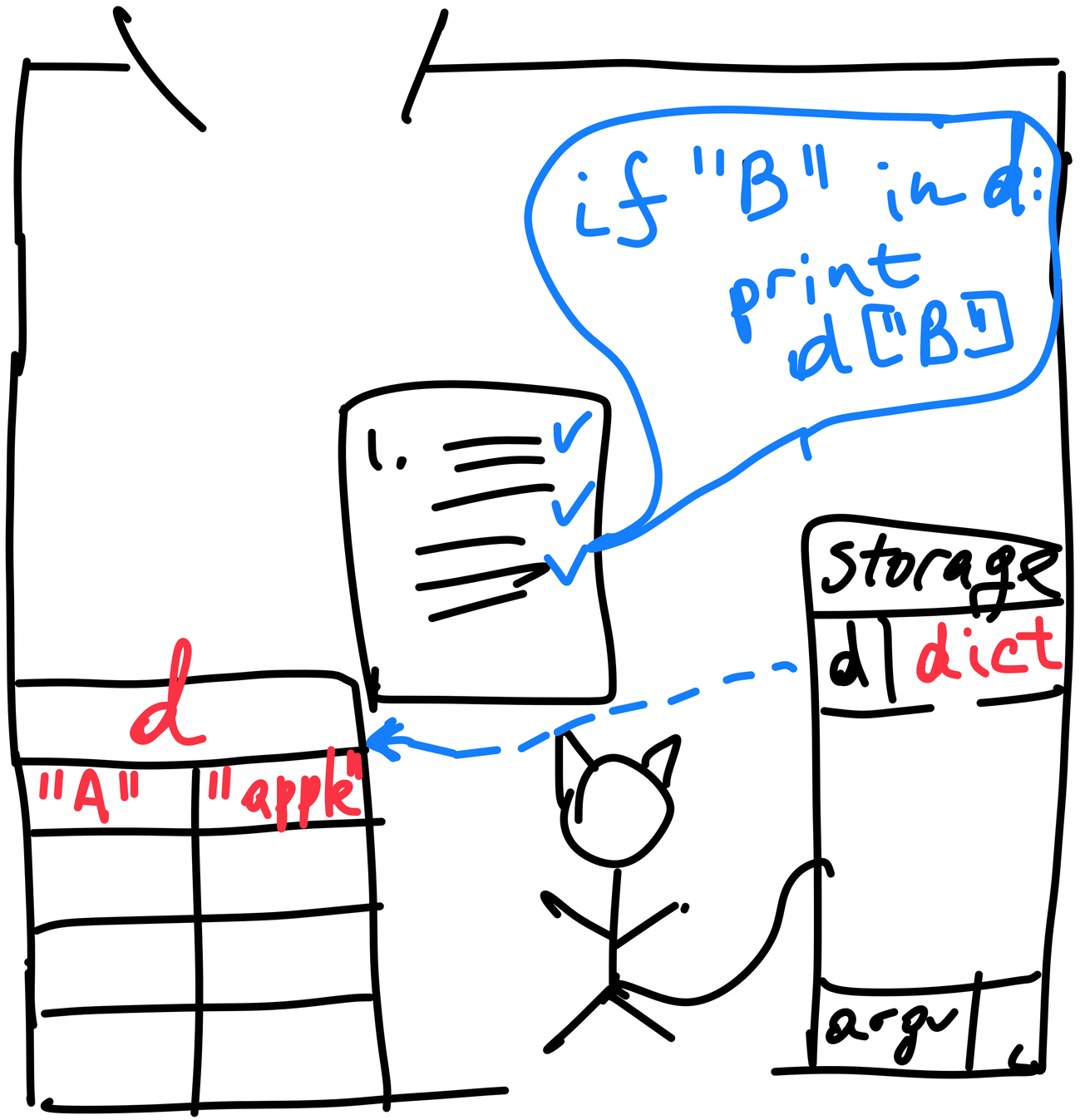
Dict

- Next instruction:
  - If there is a **key** “B” in the dict:
    - Print the **value** corresponding to that key
  - What is going to happen now?



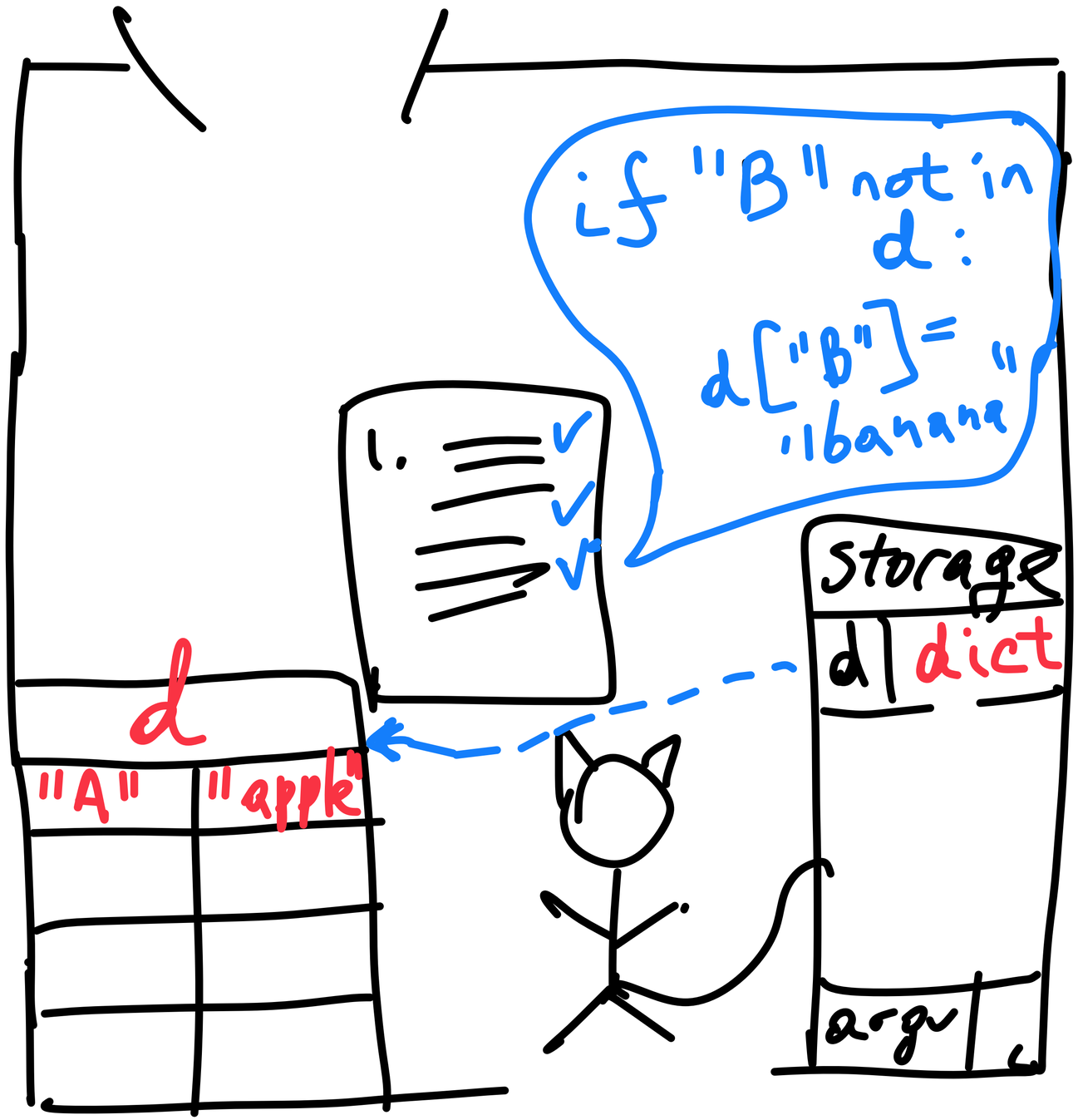
Dict

- Next instruction:
  - If there is a **key** “B” in the dict:
    - Print the **value** corresponding to that key
  - What is going to happen now?
    - **Nothing!**
    - The code is skipped.



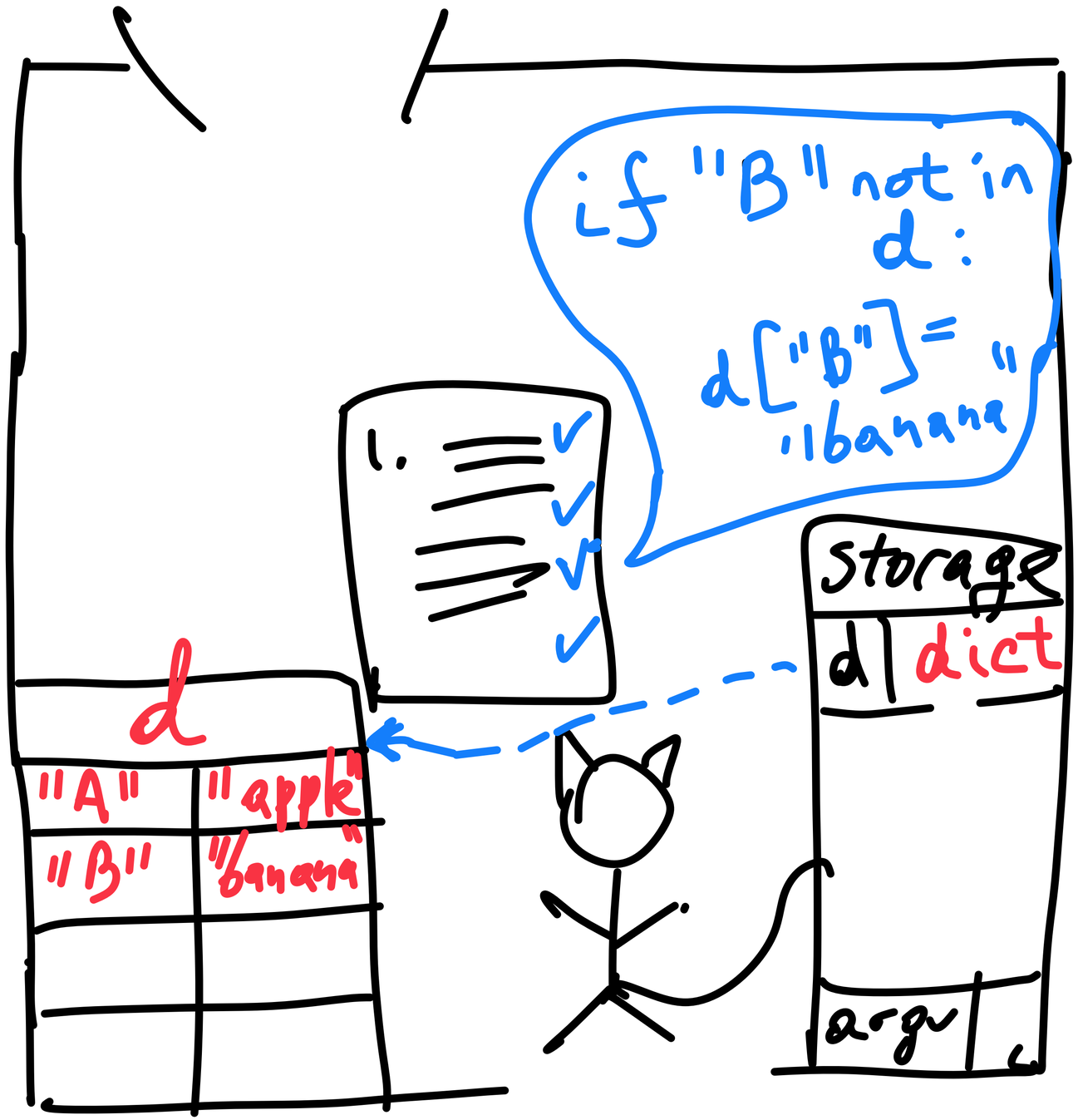
Dict

- Next instruction:
  - If there is no such key yet:
    - Put “banana” under “B”



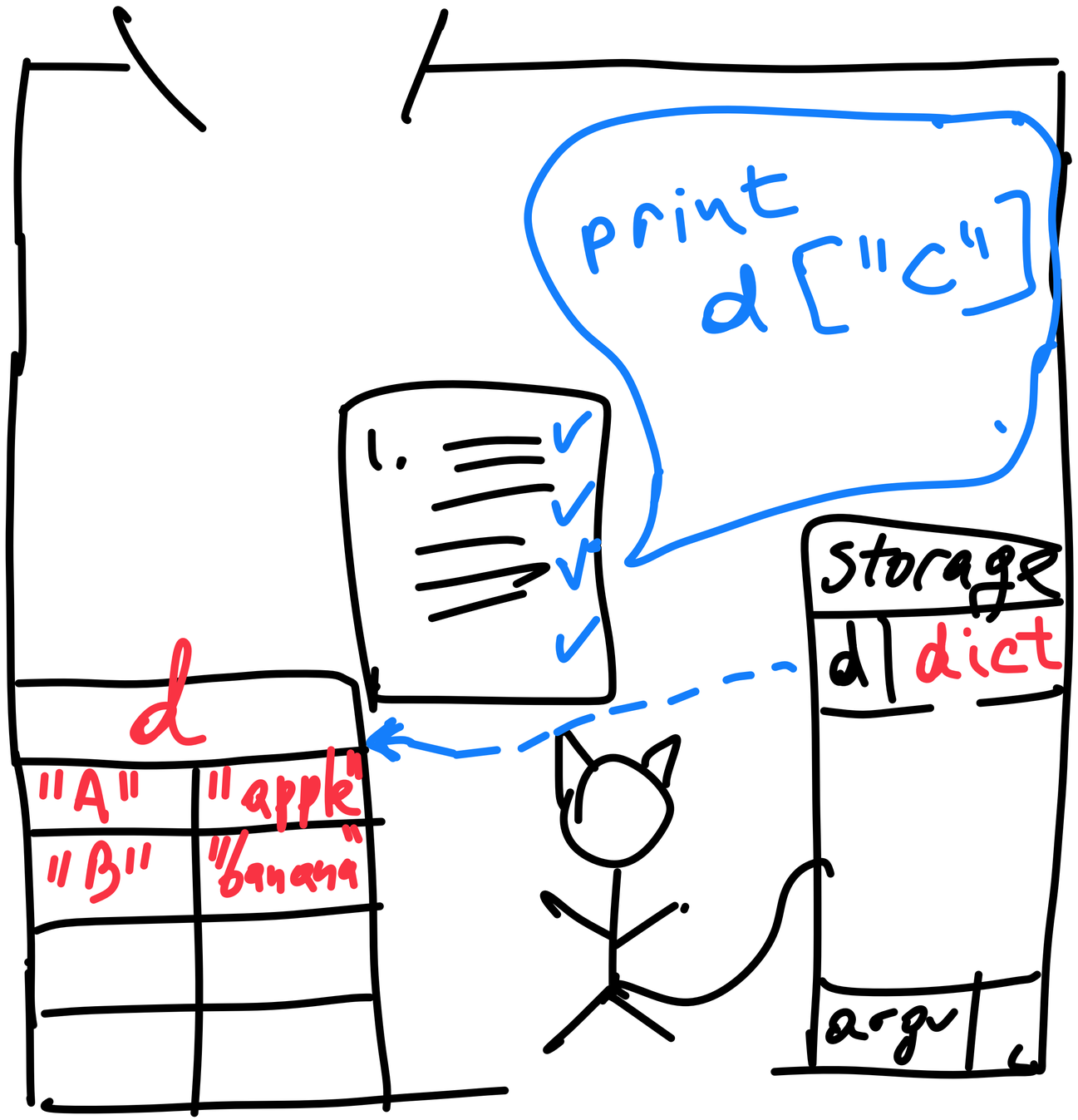
Dict

- Next instruction:
  - If there is no such key yet:
    - Put “banana” under “B”



Dict

- Trying to access a **nonexistent** key
  - Without **checking** that it exists, first
  - What is going to happen now?



Dict

- Trying to access a **nonexistent** key
  - Without **checking** that it exists, first
  - What is going to happen now?
    - The execution exploded
    - The cat is kicked out

Demo

Counting letters in words

- Write a function which takes one word as an argument
  - And returns how many of each letter of the alphabet the word contains

Formatting output

Formatting output

- Typically, have some numbers/results to report
  - Often, along with some text explaining the numbers
  - (Though only if results are for human reading!)
- Enter string format() function!
  - Accepts arguments of all types
  - Inserts their string representations in indicated spots!

Reading data from files

Opening files

- Special syntax
  - Will be included in HW2 skeleton
    - *with open(filename, ‘r’, encoding=‘utf8’) as f:*
      - Typycally, call the read() or readlines()
  - *with* is a **keyword** which creates special scope
    - and makes sure the file closes **automatically**
    - **Leaving** open files is **bad** practice
      - Must either explicitly close them or use *with*

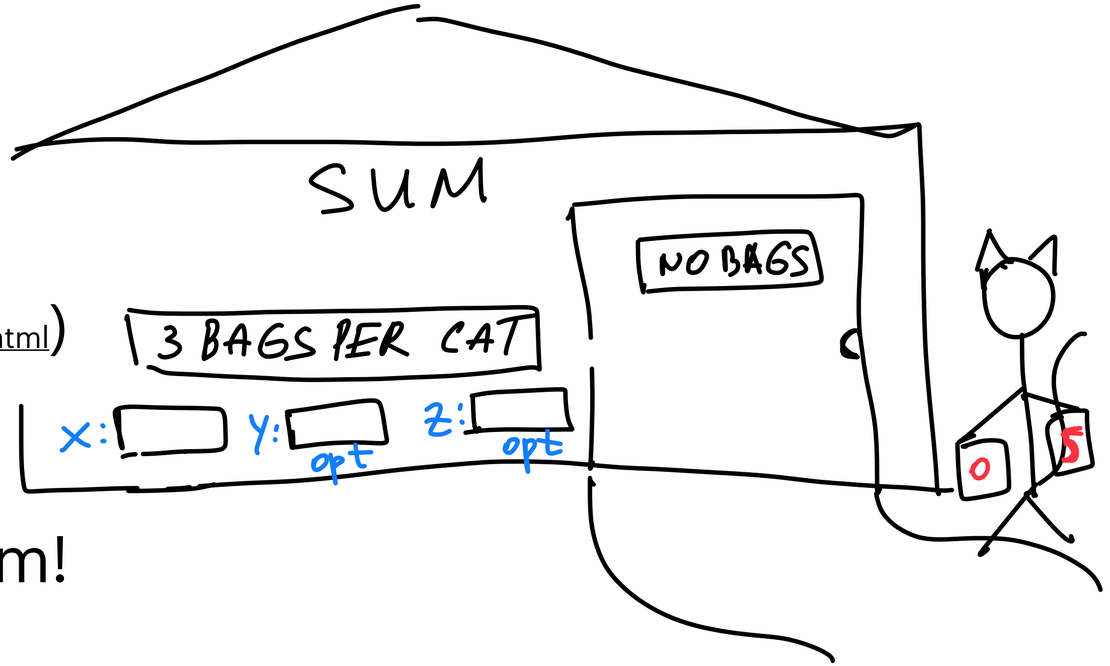
Opening files
For reading

- Special syntax
  - Will be **included** in HW2 skeleton
    - *with open(filename, ‘r’, encoding=‘utf8’) as f:*
      - Typycally, call the read() or readlines()
  - You must indicate as **arguments**:
    - Which file to open (**exact path**!)
    - Whether to open it for **reading** or for **writing**
    - Which **encoding**

Optional arguments

labeling arguments while calling a function

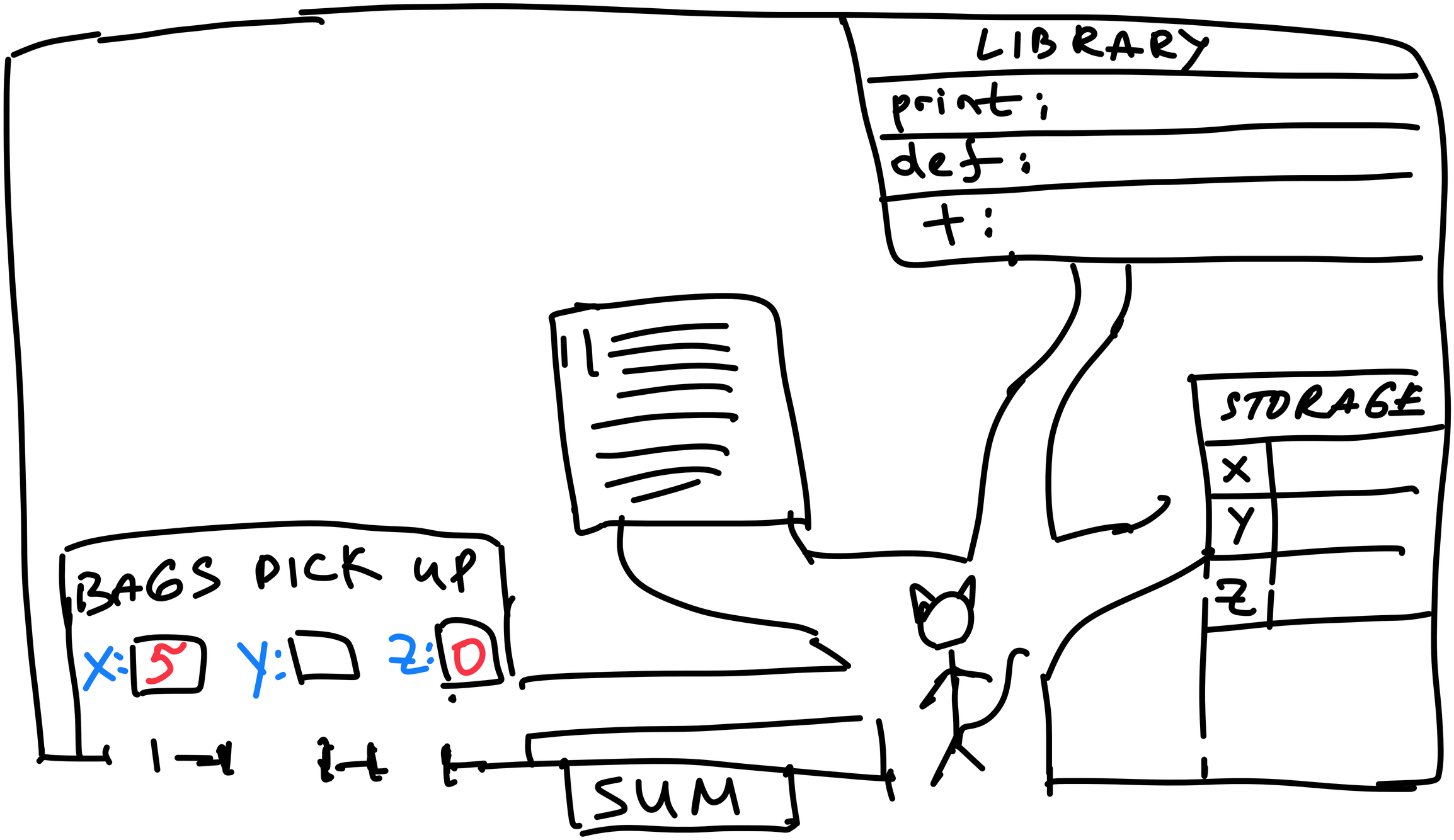
- file open() function has many possible parameters/arguments
  - see the docs! (7.2 in https://docs.python.org/3/tutorial/inputoutput.html)
- You don’t need to pass all of them
- But if you one pass some, better label them!
  - unless you are passing the first ones, in strict order

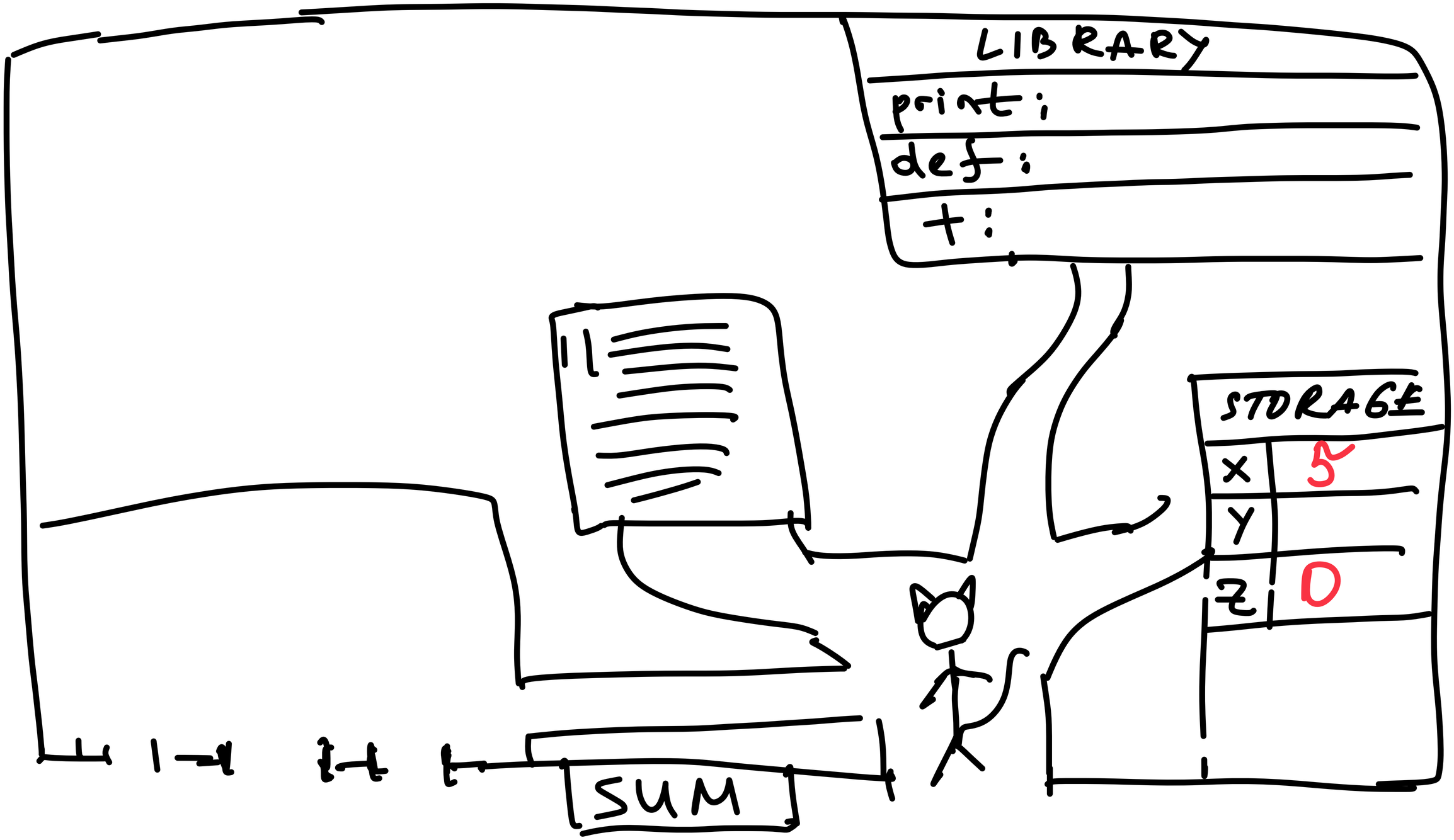



SUM
NO BAGS
3 BAGS PER CAT
x:
y:
opt
z:
opt
0
5


SUM
NO BAGS
3 BAGS PER CAT
x:
y:
opt
z:
opt
5
0
"z"


SUM
NO BAGS
3 BAGS PER CAT
x:
y:
opt
z:
opt
5
z = 0


LIBRARY
print;
def:
t:
STORAGE
x
y
z
BAGS PICK UP
X: 5
Y:
Z: 0
SUM

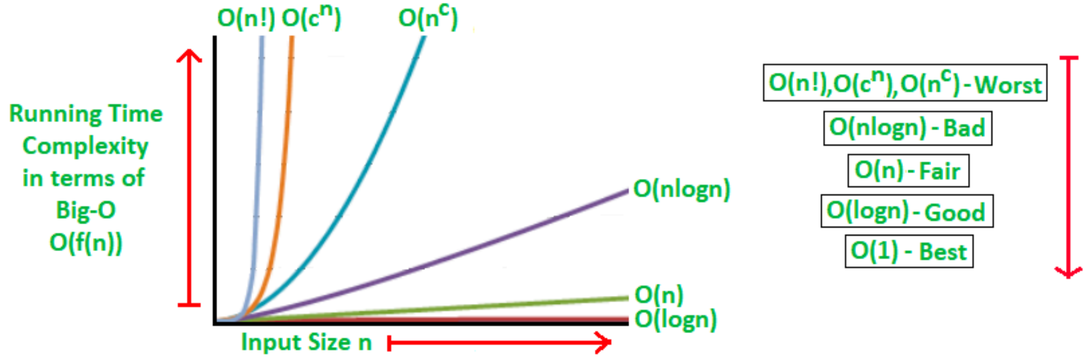

LIBRARY
print:
def:
t:
STORAGE
x
5
y
z
0
SUM

Demo opening files for reading

Bonus material

Program complexity

The “Big-O” analysis



https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

- Some programs are **faster** than others!
  - This is often talked about in terms of “the Big O”
  - “O” refers to “order” (of the function)
    - ...the function of how program **running time** grows with the increase of **input size**
  - E.g., the longer the list, the longer it takes to find an element in that list
    - e.g., iterating over the full list **once** is an example of **linear** time complexity
  - The dict structure allows for O(1) look-up speed!
    - Because of how the entire structure is organized and stored in memory
    - “Hashing”: a **fast** function sending your right to the **key** location
  - But may take more **space** than other data structures
- **Time** and **Space** complexity engage in **trade-offs**

```
>>> hash('brown')
-8795079360369488223
>>> hash(2.018)
41505174165846018
>>> hash(1)
1
```

http://www.jessicayung.com/how-python-implements-dictionaries/