[301] JSON

Tyler Caraza-Harter

Learning Objectives Today

JSON

- differences with Python syntax
- creating JSON files
- reading JSON files

Read: Sweigart Ch 14
https://automatetheboringstuff.com/chapter14/

“JSON and APIs” to the end

Python Data Structures and File Formats

Python

```
[
 [“name”, “x”, “y”],
 [“alice”, 100, 150],
 [“bob”, -10, 80]
]
```

list of lists

File

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

We can use CSV files to store data we would want in lists of lists

Python Data Structures and File Formats

Python

```
[
 [“name”, “x”, “y”],
 [“alice”, 100, 150],
 [“bob”, -10, 80]
]
```

list of lists

File

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

?

Python Data Structures and File Formats

Python

```
[
 [“name”, “x”, “y”],
 [“alice”, 100, 150],
 [“bob”, -10, 80]
]
```

list of lists

File

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

JSON file

Python Data Structures and File Formats

Python

File

JSON files look almost identical to Python code for data structures!

```
[
  [“name”, “x”, “y”],
  [“alice”, 100, 150],
  [“bob”, -10, 80]
]
```

list of lists

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

JSON file

Python Data Structures and File Formats

Python

File

JSON files look almost identical to Python code for data structures!

```
[
  [“name”, “x”, “y”],
  [“alice”, 100, 150],
  [“bob”, -10, 80]
]
```

list of lists

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

dicts use curly braces

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

JSON file

Python Data Structures and File Formats

Python

File

JSON files look almost identical to Python code for data structures!

```
[
  [“name”, “x”, “y”],
  [“alice”, 100, 150],
  [“bob”, -10, 80]
]
```

list of lists

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

keys are separated from values with a colon

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

JSON file

Python Data Structures and File Formats

Python

File

JSON files look almost identical to Python code for data structures!

```
[
  [“name”, “x”, “y”],
  [“alice”, 100, 150],
  [“bob”, -10, 80]
]
```

list of lists

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

lists use square brackets

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

JSON file

Python Data Structures and File Formats

Python

File

JSON files look almost identical to Python code for data structures!

```
[
  [“name”, “x”, “y”],
  [“alice”, 100, 150],
  [“bob”, -10, 80]
]
```

list of lists

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

strings are in quotes

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

JSON file

Python Data Structures and File Formats

Python

File

JSON files look almost identical to Python code for data structures!

```
[
  [“name”, “x”, “y”],
  [“alice”, 100, 150],
  [“bob”, -10, 80]
]
```

list of lists

```
name,x,y
alice,100,150
bob,-10,80
```

CSV file

integers look like integers

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

dict of dicts

```
{
  “alice”: {
    “age”: 40,
    “scores”: [10,20,19]},
  “bob”: {
    “age”: 45,
    “scores”: [15,23,17,15]}
}
```

JSON file

JSON

Stands for **JavaScript Object Notation**

- JavaScript is a language for web development
- JSON was developed as a way for JavaScript programs to store/share data
- JavaScript is similar to Python, which is why JSON looks like Python code

JSON

Stands for **JavaScript Object Notation**

- JavaScript is a language for web development
- JSON was developed as a way for JavaScript programs to store/share data
- JavaScript is similar to Python, which is why JSON looks like Python code

Minor JavaScript vs. Python differences:

	Python	JSON
Booleans	True, False	true, false
No value	None	null
Quotes	Single (‘) or double (“)	Only double (“)
Commas	Extra allowed: [1,2,]	No extra: [1,2]
Keys	Any type: {1: “one”}	Str only: {“1”: “one”}

Reading JSON Files

JSON file saved somewhere

```
{
  “alice”: 10,
  “bob”: 12,
  “cindy”: 15
}
```

Reading JSON Files

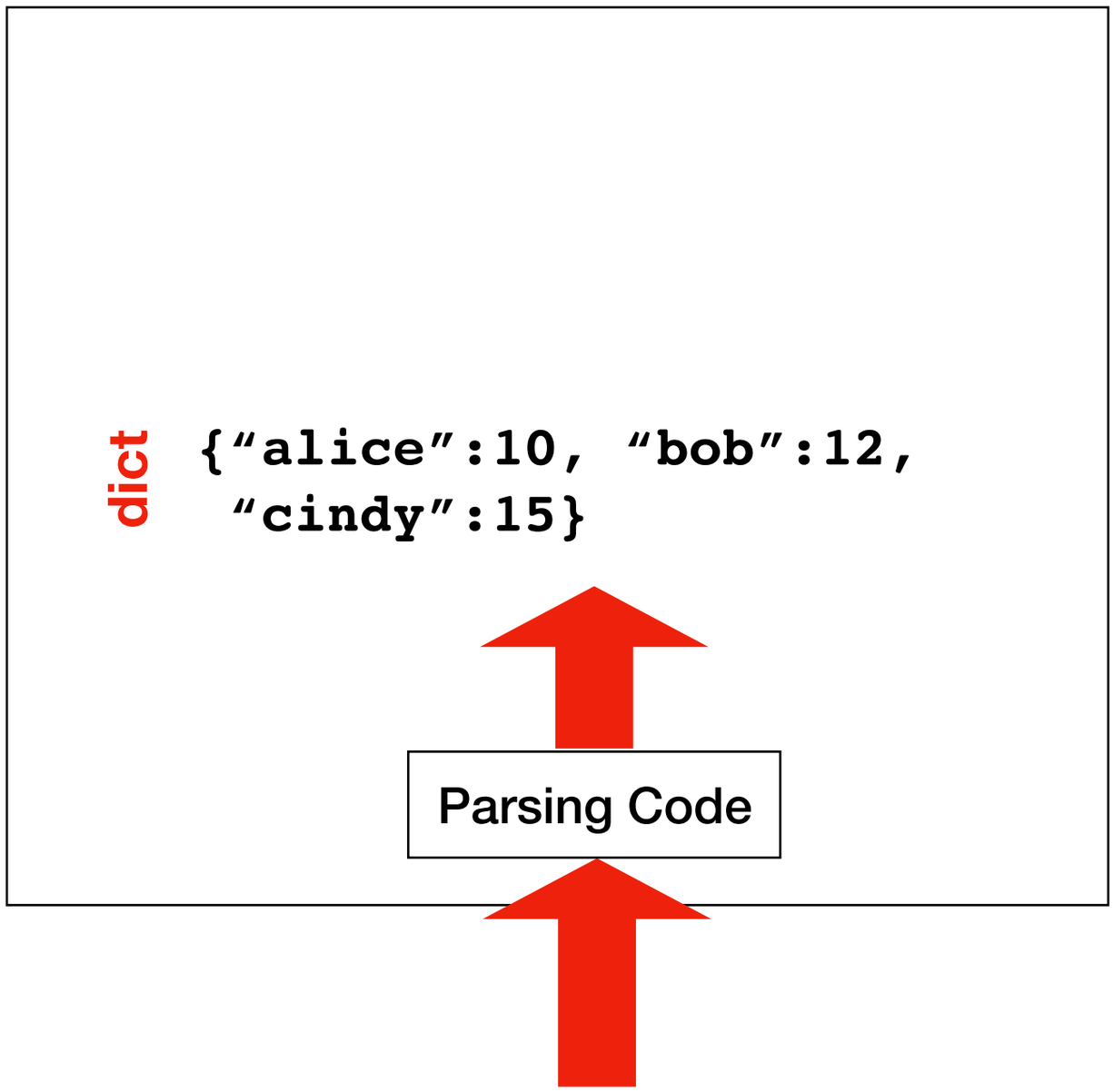
Python Program

Parsing Code

JSON file saved somewhere

```
{
  “alice”: 10,
  “bob”: 12,
  “cindy”: 15
}
```

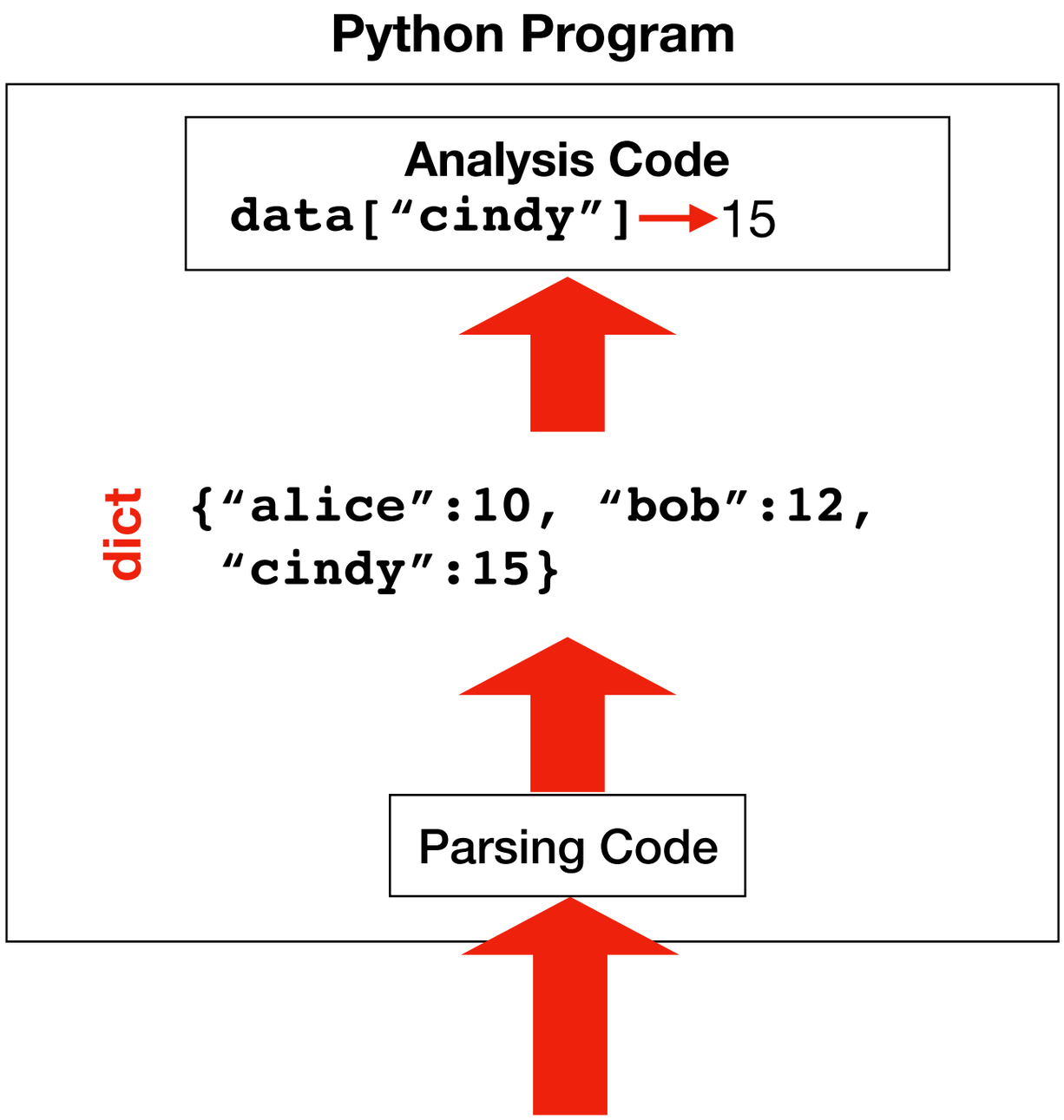
Reading JSON Files



JSON file saved somewhere

```
{
  “alice”: 10,
  “bob”: 12,
  “cindy”: 15
}
```

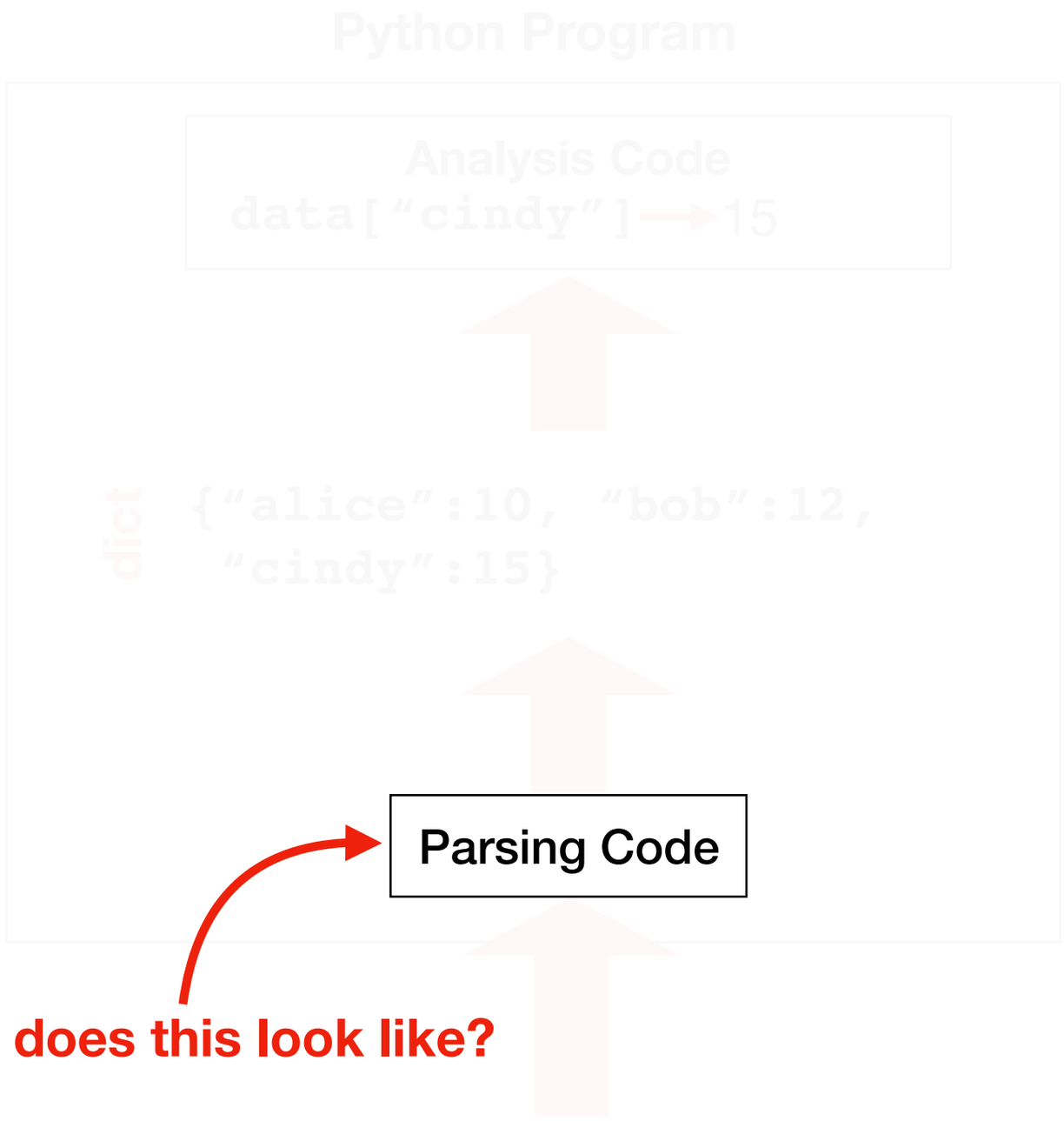
Reading JSON Files



JSON file saved somewhere

```
{
  “alice”: 10,
  “bob”: 12,
  “cindy”: 15
}
```

Reading JSON Files



JSON file saved somewhere

```
{
  “alice”: 10,
  “bob”: 12,
  “cindy”: 15
}
```

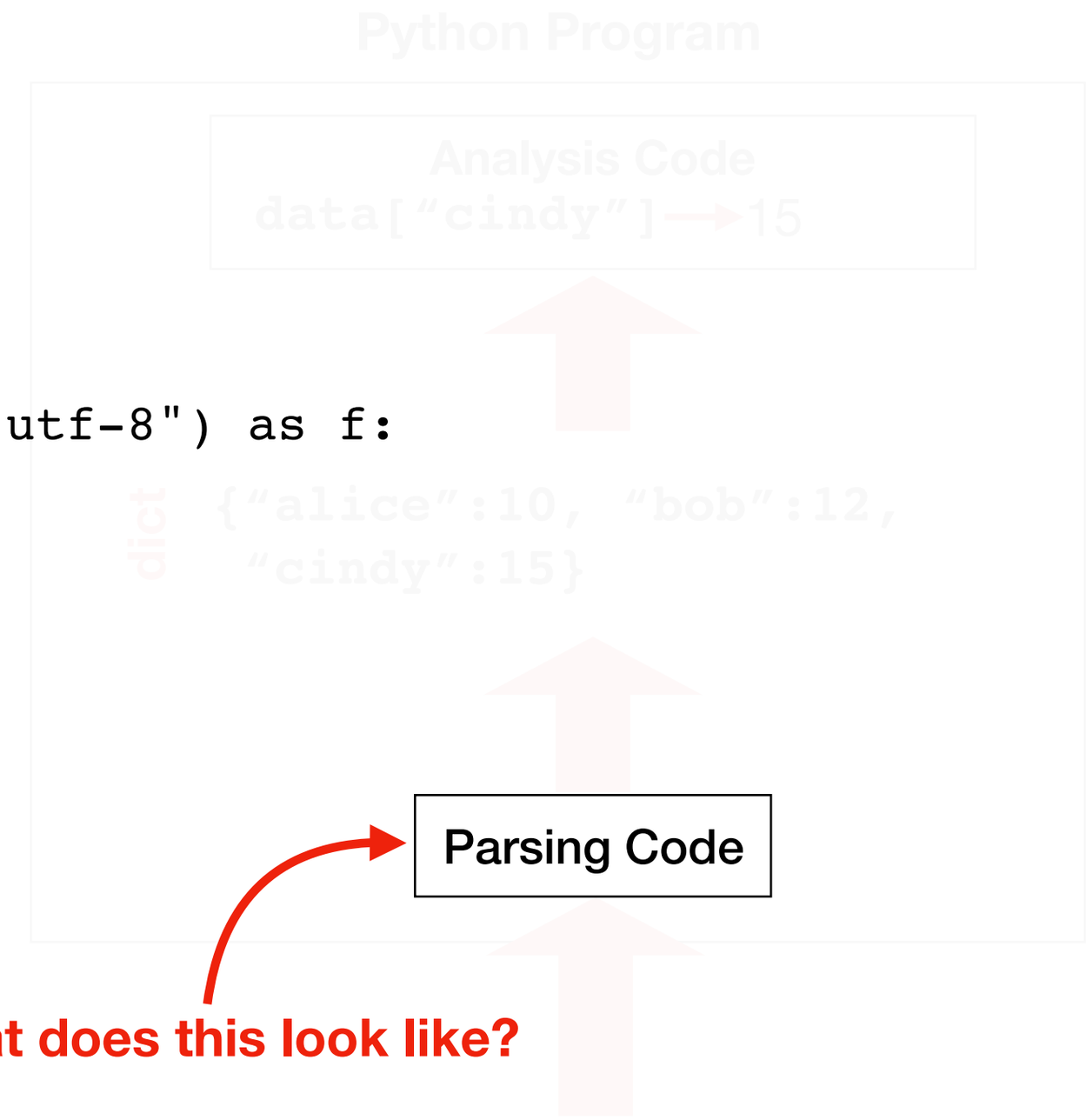
Reading JSON Files

```
import json

def read_json(path):
    with open(path, encoding="utf-8") as f:
        return json.load(f)
```

CTRL + C

don't need to understand this snippet yet



Reading JSON Files

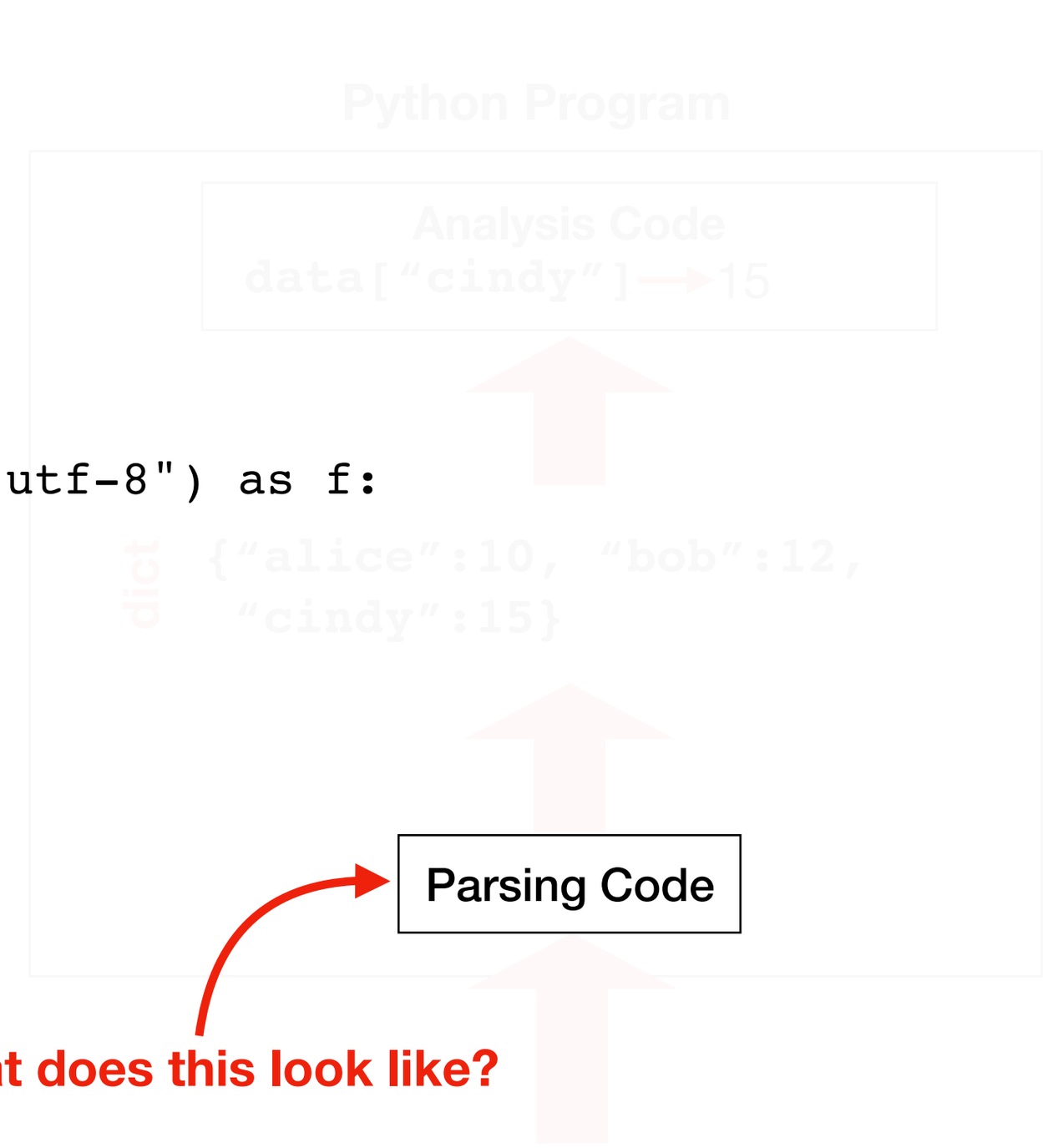
```
import json

def read_json(path):
    with open(path, encoding="utf-8") as f:
        return json.load(f)
```

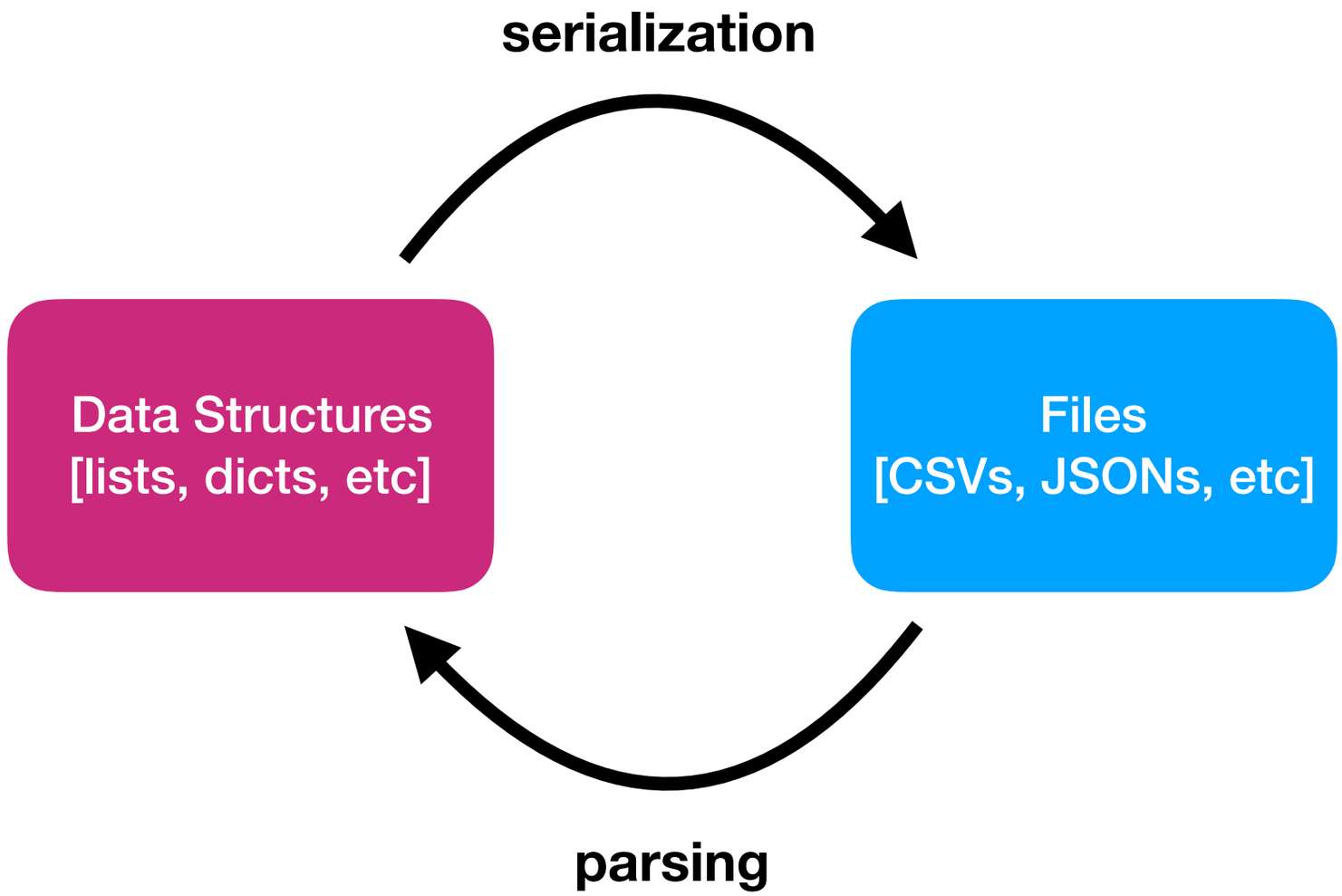
CTRL + C

don't need to understand this snippet yet

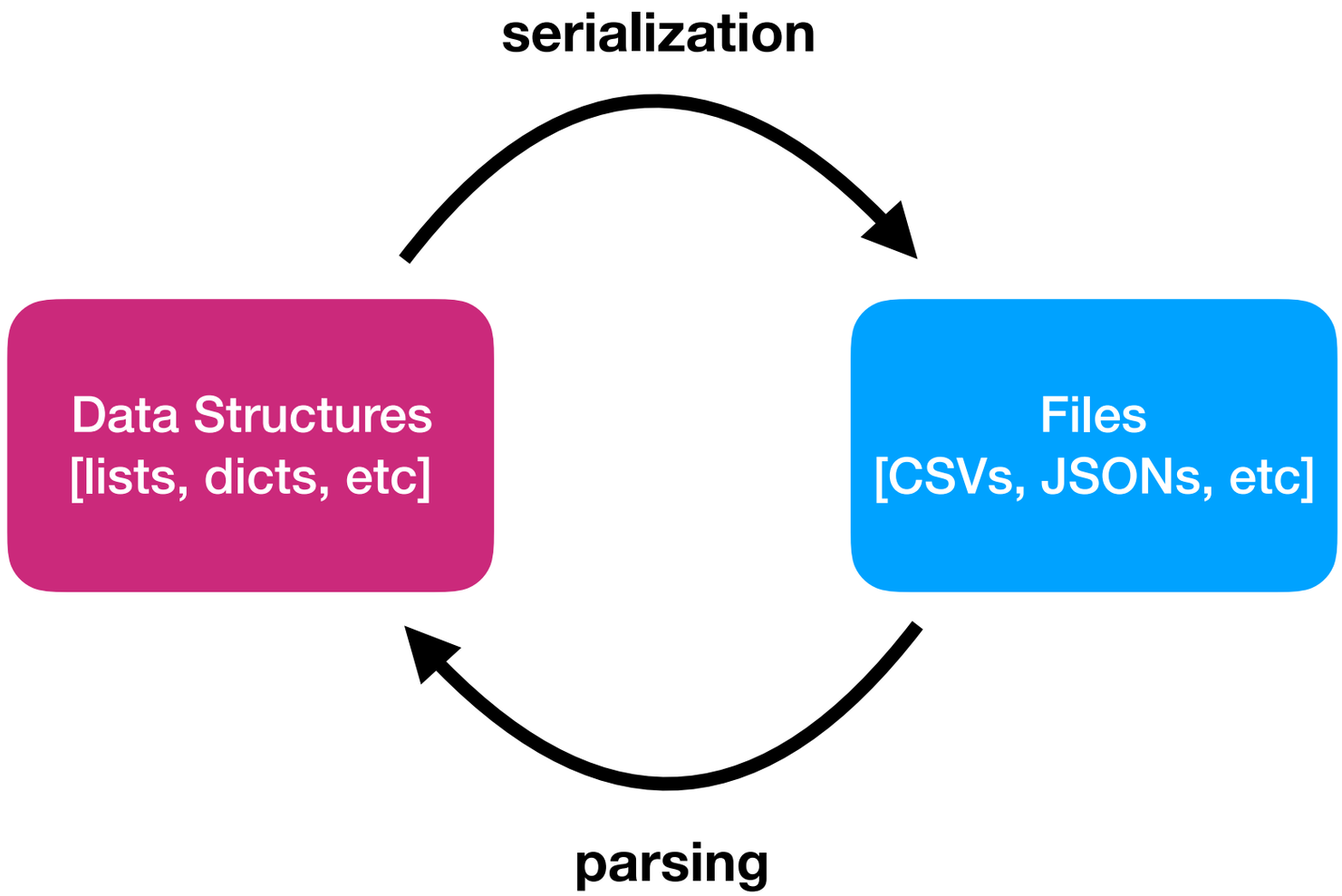
what about writing?



Data Structures and Files



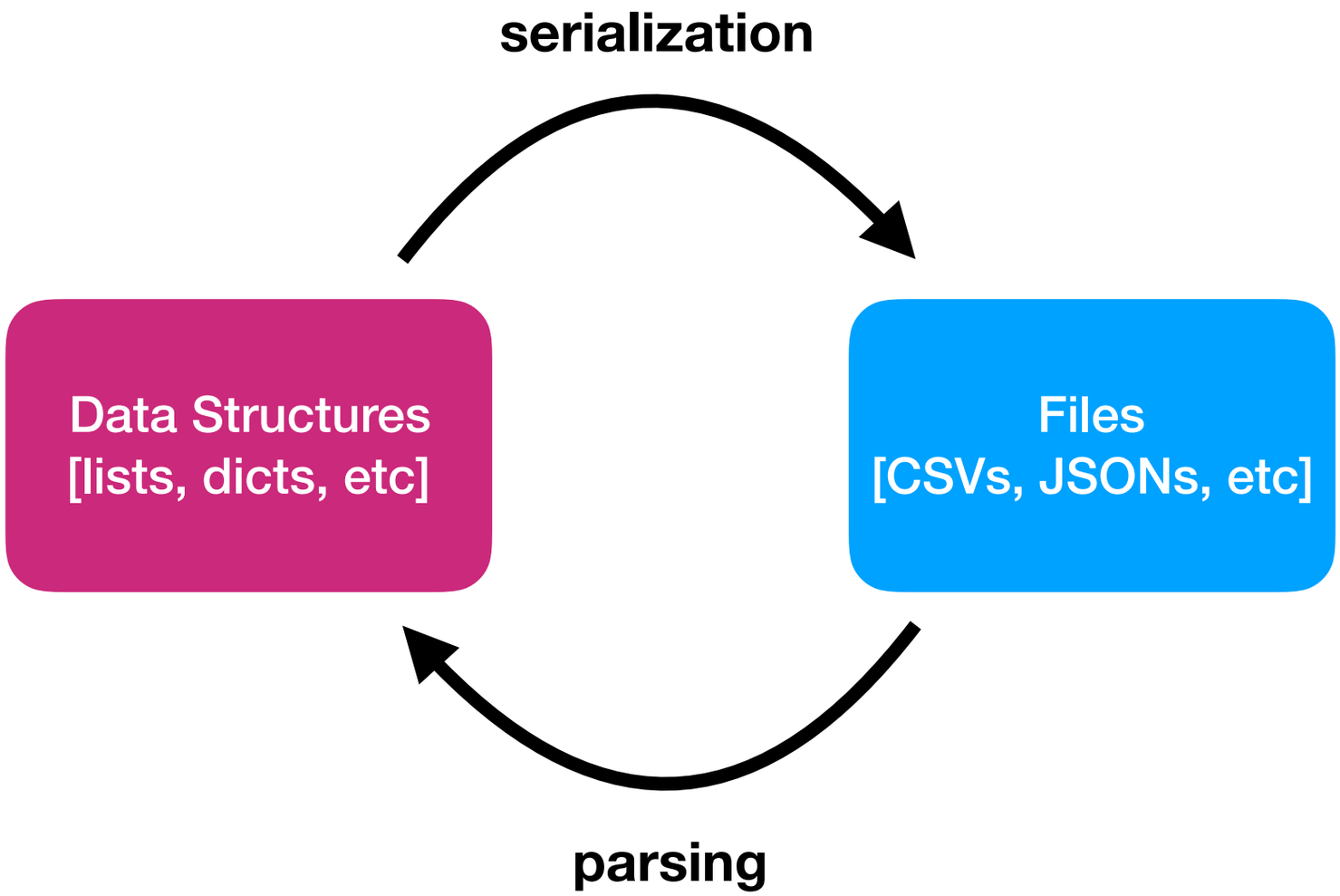
Data Structures and Files



why not just have data structures?

because our data needs to live somewhere when our programs aren't running

Data Structures and Files



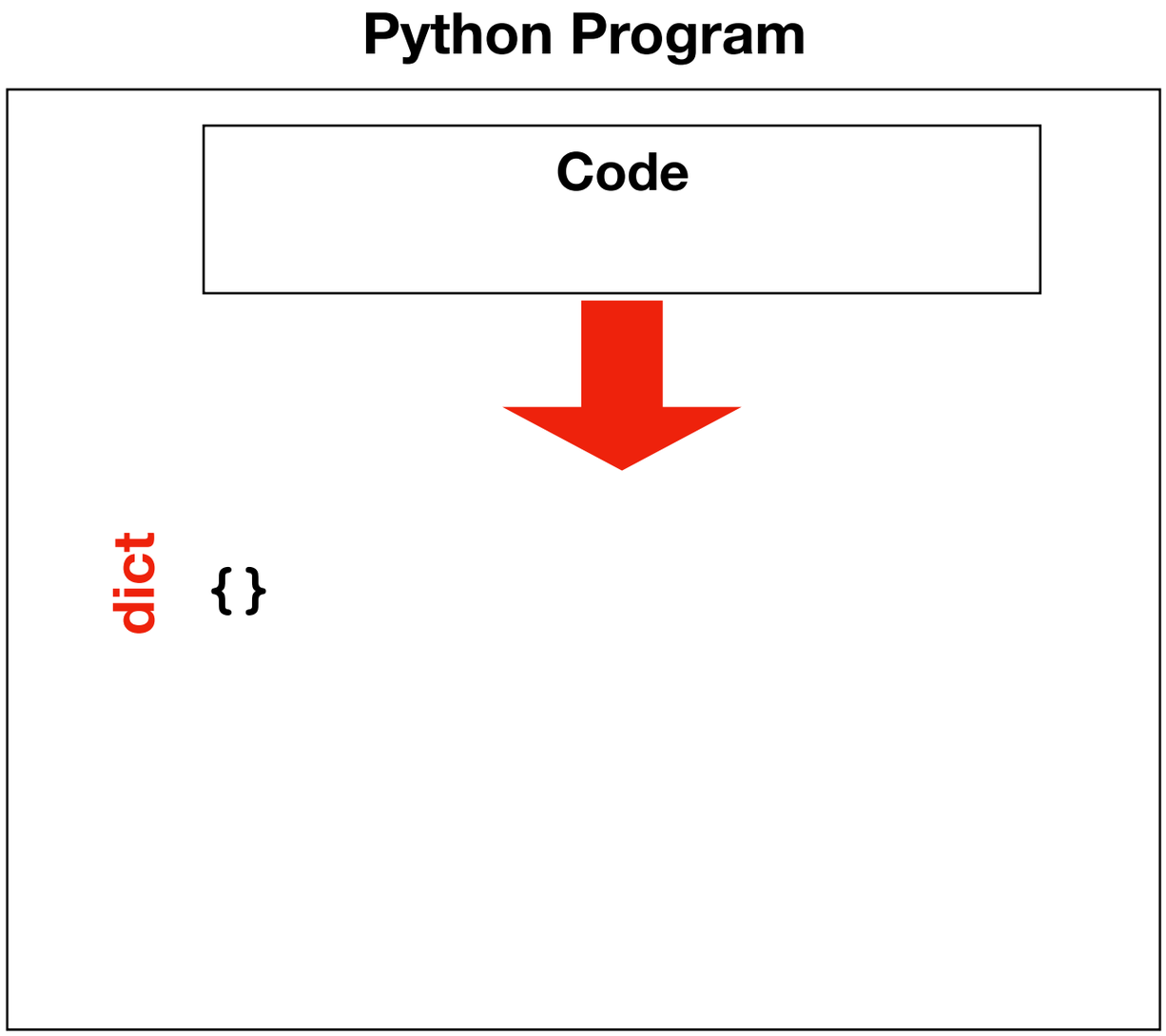
why not just have data structures?

because our data needs to live somewhere when our programs aren't running

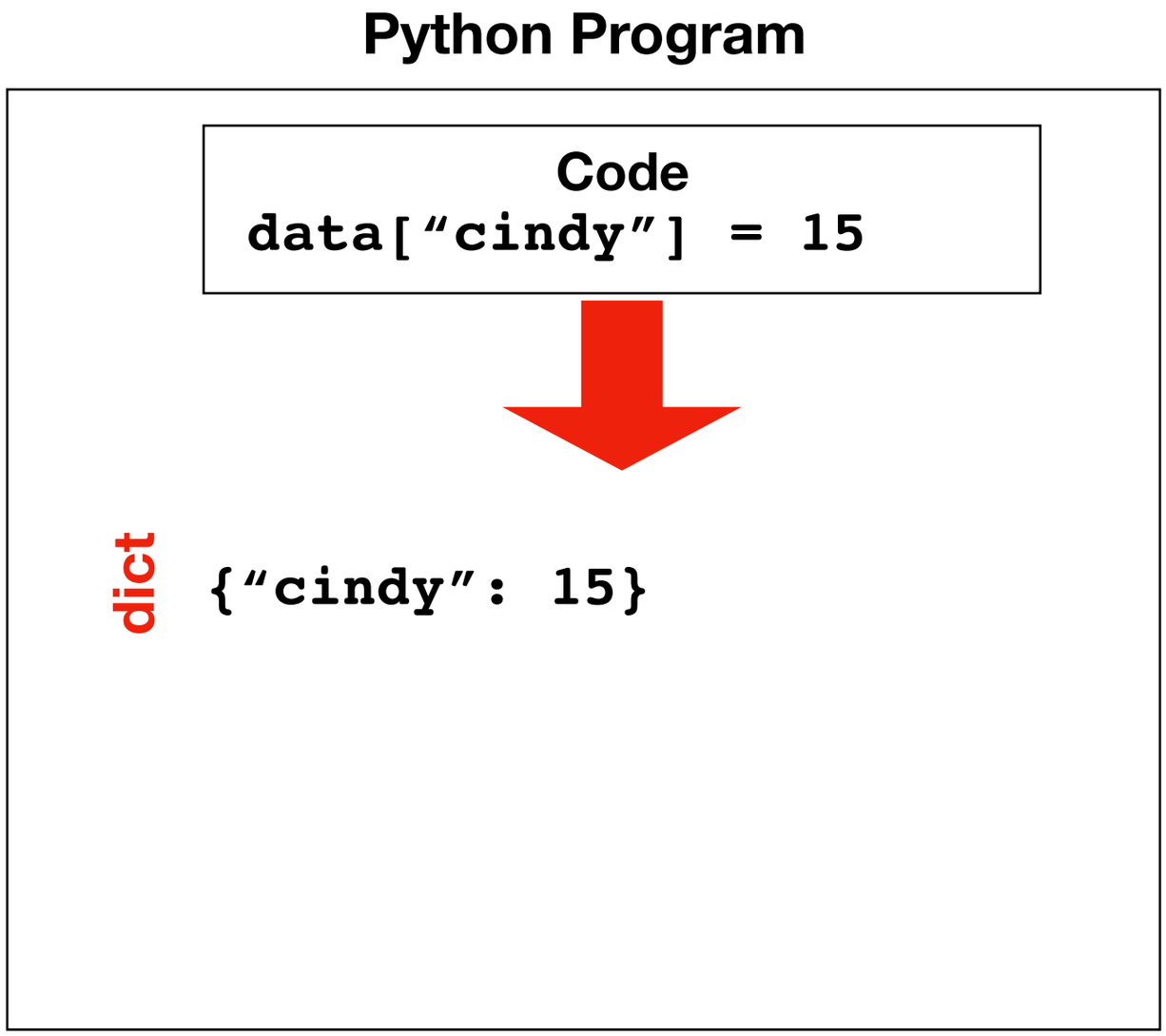
why not just have files?

slow, and Python doesn't understand structure until it is parsed

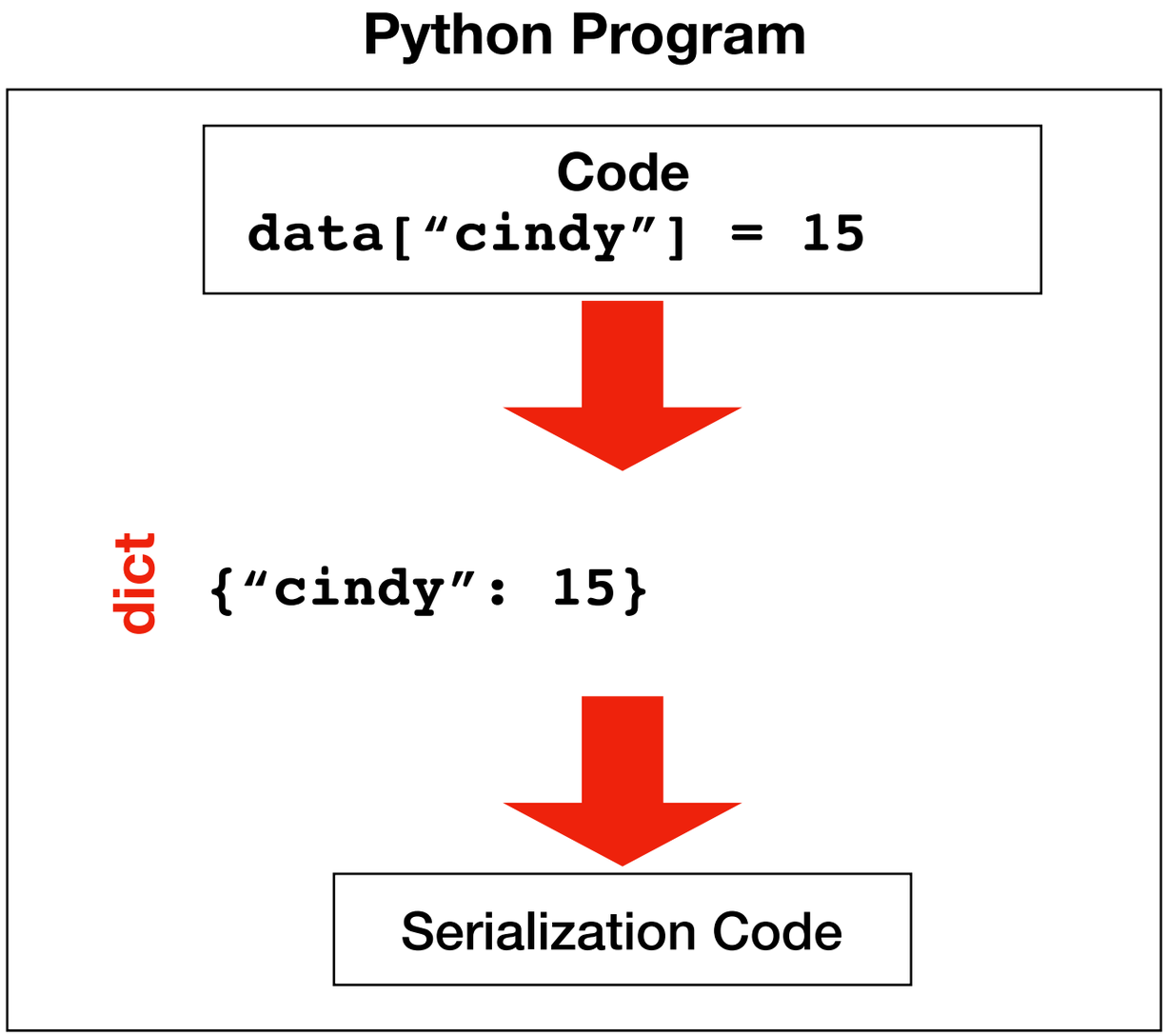
Writing JSON Files



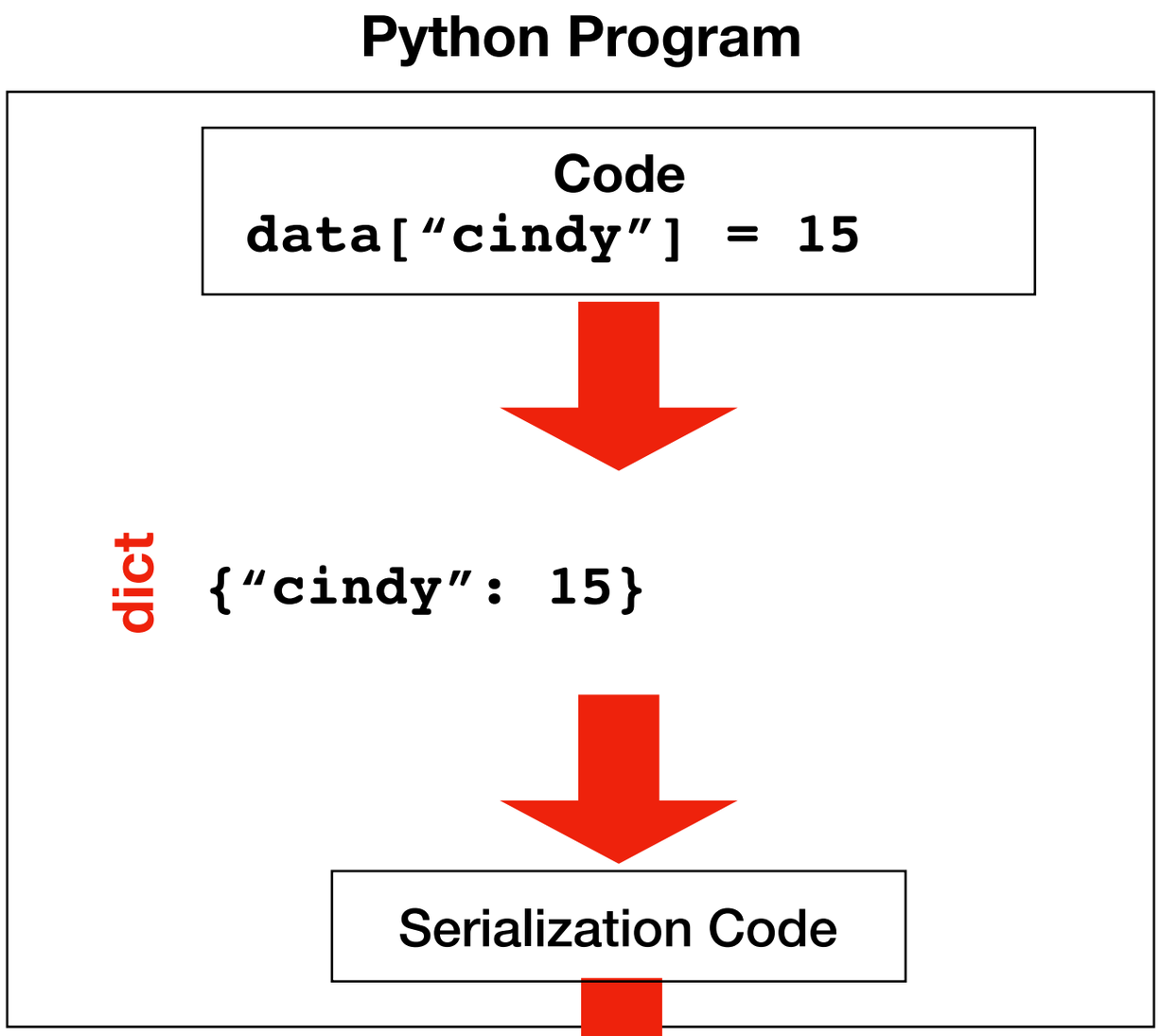
Writing JSON Files



Writing JSON Files



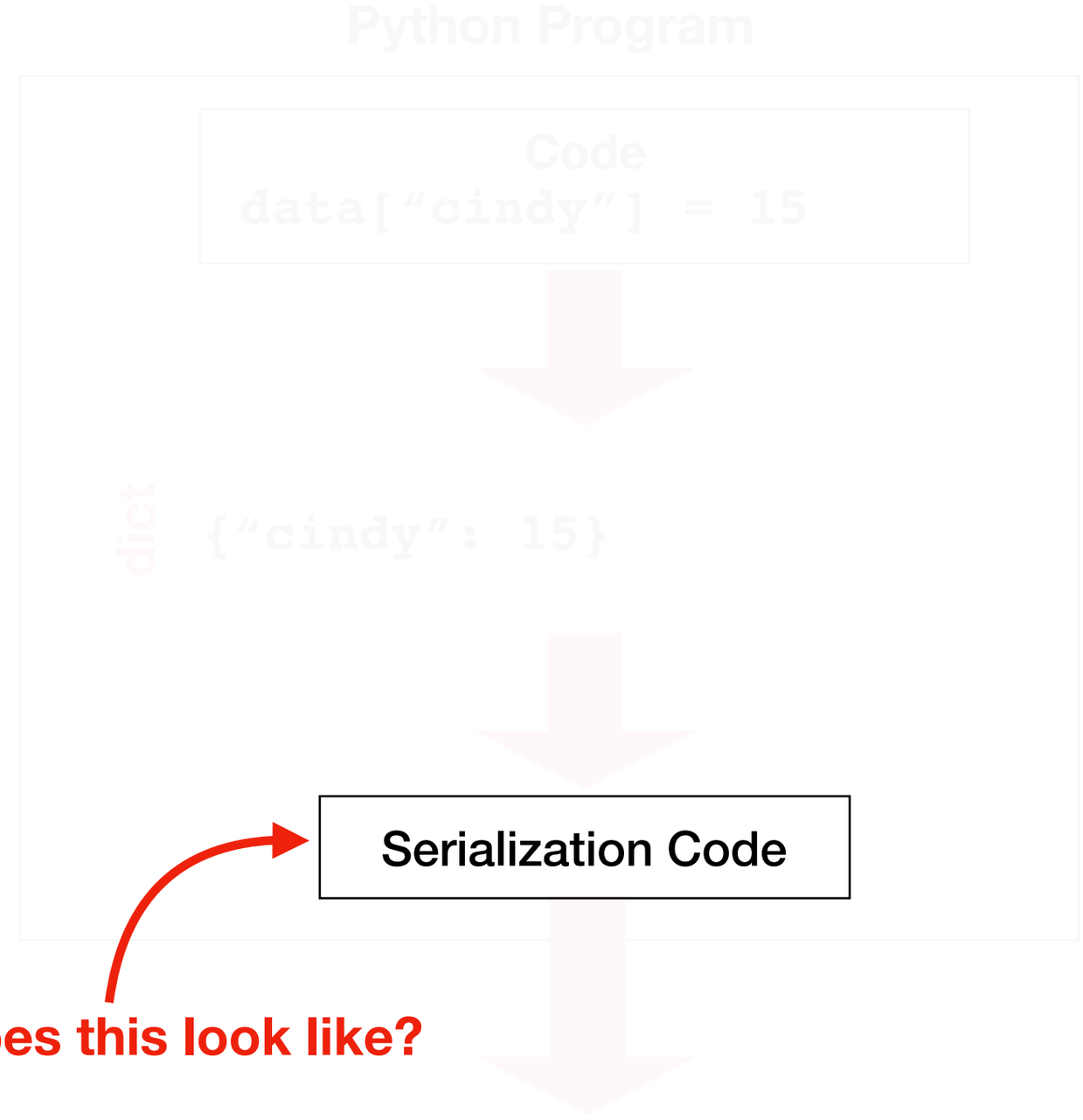
Writing JSON Files



JSON file saved somewhere

```
{
  “cindy”: 15
}
```

Writing JSON Files



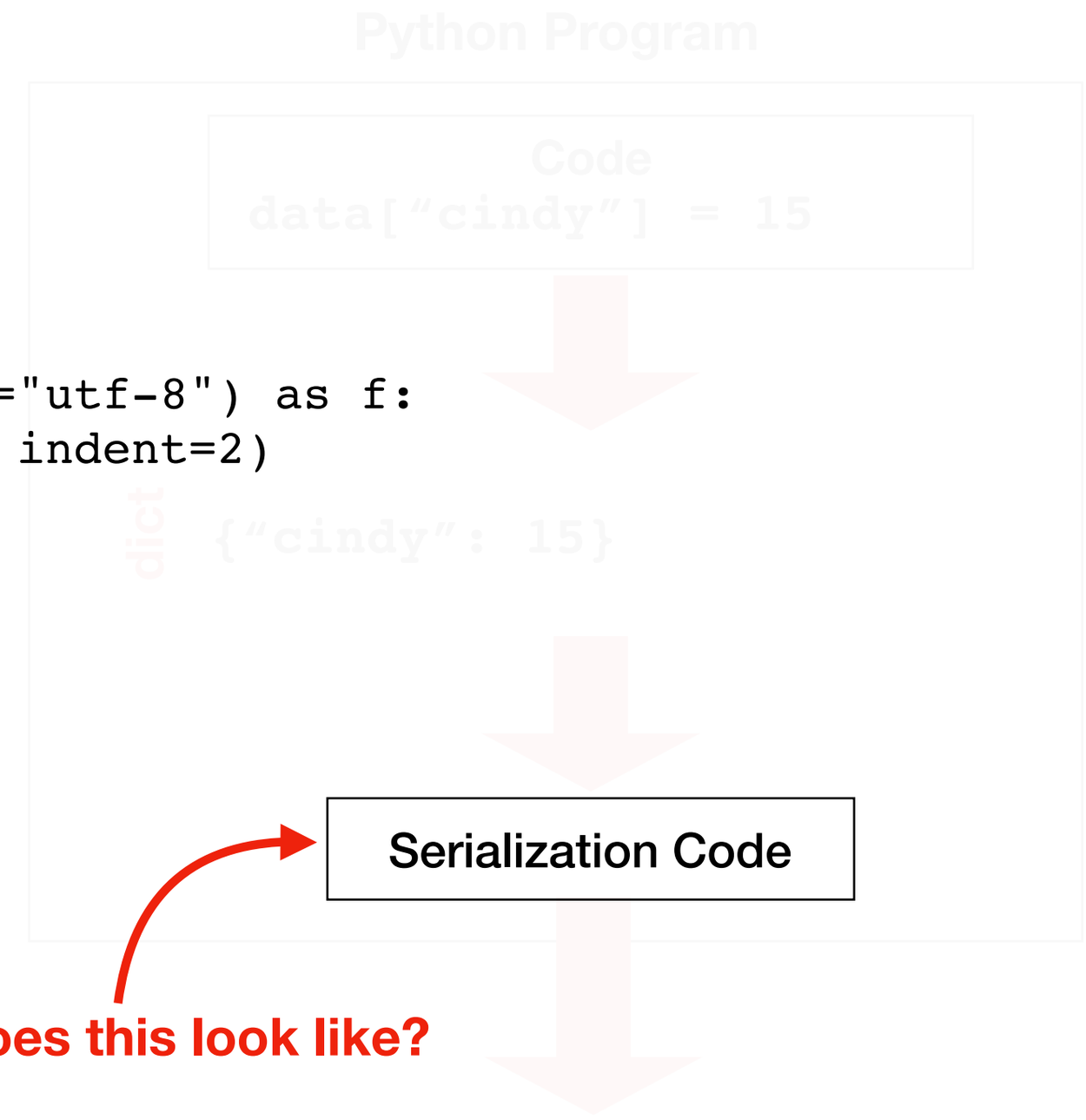
Writing JSON Files

```
import json

def write_json(path, data):
    with open(path, 'w', encoding="utf-8") as f:
        return json.dump(data, f, indent=2)
```

CTRL + C

don't need to understand this snippet yet



Demo 1: Number Count

Goal: count the numbers in a list saved as a JSON file

Input:

- Location of the file

Output:

- The sum

Example:

prompt> **python sum.py fileA.json**
6

fileA.json

```
[1,2,3]
```

Demo 2: Fifa JSON

Goal: lookup stats about players

Input:

- Player ID and column

Output:

- The value

Example:

prompt> **python lookup.py 20801 name**
Cristiano Ronaldo

fifa.json

```
{
  "20801": {
    "name": "Cristiano Ronaldo",
    "Age": 32,
    "nationality": "Portugal",
    "club": "Real Madrid CF",
    "league": "Spanish Primera Divisi\u00f3n",
    "euro_wage": 565000,
    "networth": 95500000,
    "score_of_100": 94
...
```

Demo 3: Score Tracker

Goal: record scores (save across runs) and print average

Input:

- A **name** and a **score** to record

Output:

- Running average for that person

Example:

prompt> **python record.py alice 10**
Alice Avg: 10
prompt> **python record.py alice 20**
Alice Avg: 15
prompt> **python record.py bob 13**
Bob Avg: 13

Demo 4: Prime Cache

Goal: find number of primes less than N,
remembering previous answers

Input:

- An integer N

Output:

- How many primes are less than that number

Demo 5: Upper Autocomplete

Goal: record scores (save across runs) and print average

Input:

- A complete phrase
- A partial phrase ending with a *

Output:

- The upper case version of it
- Options to autocomplete

autocomplete must work
across multiple runs

Example:

prompt> **python shout.py**
msg: hi
HI
msg: hello
HELLO
msg: h*
1: hi
2: hello
select: 1
HI