WELCOME

CSCA20 REVIEW SEMINAR

What is the difference between

`return()` and `print()`

return()

- You can only return from inside a function.
- Values that are returned can be saved and re-used, but are not displayed.
- Return is used primarily to retrieve data that was modified/generated, at the end of a function call .

print()

- You can print anywhere in your program.
- Printing a value doesn't save it. It only displays the value in the shell.
- Print statements are used to display information to the shell. This can be useful in debugging and testing.

Type Conversions

Which of these will run?

```
int(“four”)
int(“4”)
int(15.39)
int(“15.39”)
int(“CSCA20”)
int(“1.1”)
float(int(“4”))
int(float(“5.34”))
```

Type Conversions

Which of these will run?

1	`int(“four”)`
🟩 2	`int(“4”)`
🟩 3	`int(15.39)`
4	`int(“15.39”)`
5	`int(“CSCA20”)`
6	`int(“1.1”)`
🟩 7	`float(int(“4”))`
🟩 8	`int(float(“5.34”))`

Review

What is wrong with this code?

```
def myFunction(num_1, num2):
    if num_1 > num2 and == 3:
        print(Tacos!)
    elif num_1 = num2:
        print(‘burgers!’)
    else
        print(“pizza”)
return(“Food!”)
```

Hint: There are 5 problems!

Review

How can we fix it?

```
def myFunction(num_1, num2):
    if num_1 > num2 and == 3:
        print(Tacos!)
    elif num_1 = num2:
        print(‘burgers!’)
    else
        print(“pizza”)
return(“Food!”)
```

Review

How can we fix it?

```
def myFunction(num_1, num2):
    if (num_1 > num2) and (num_1 == 3):
        print(“Tacos!”)
    elif num_1 == num2:
        print(‘burgers!’)
    else:
        print(“pizza”)
    return(“Food!”)
```

```python
def myFunction(num_1, num2):
    if (num_1 > num2) and (num_1 == 3):
        print(“Tacos!”)
    elif num_1 == num2:
        print(‘burgers!’)
    else:
        print(“pizza”)
    return(“Food!”)
```

What do we expect to see?

```python
result = myFunction(3, 1)
print(result)
```

```
Tacos!
Food!
```

```python
myFunction(3, 1)
```

```
Tacos!
```

Conditional Statements

```
if condition1:
    # Perform action 1

[if]/[elif] condition2:
    # Perform action 2

[if]/[elif] condition3:
    # Perform action 3

else:
    # Handle all other cases
```

Conditional Statements

What is the difference between `if` and `elif`?

A: Multiple if blocks can be executed,
but only one elif block can be executed.

AKA: elif blocks are mutually exclusive
with if blocks and other elif blocks.

How many `else` blocks am I allowed to have?

A: Only one at the end to catch every
other case that falls through your
ifs and elifs.

STRINGS

Strings

What is a string?

A String is a sequence of characters.

Think of it as a **word** or **phrase**

In Python, string literals are declared with **double or single quotes**

```
myString = ‘CSCA20’
myString2 = “CSCA20”
```

To the interpreter, **both** strings **are equal**

String Operations

Which of these will run?

```
print(“4” + “0”)
print(“4” + 0)
print(“hello” * 2)
print(“hello” * “2”)
print(“hello” + 2)
print(“CSCA” + “20”)
print(“CSCA” + 20)
print(“CSCA” + str(20))
```

String Operations

Which of these will run?

🟩 1	`print("4" + "0")`
2	`print("4" + 0)`
🟩 3	`print("hello" * 2)`
4	`print("hello" * "2")`
5	`print("hello" + 2)`
🟩 6	`print("CSCA" + "20")`
7	`print("CSCA" + 20)`
🟩 8	`print("CSCA" + str(20))`

String Operations

Concatenation

$$\text{“First” + “Second”} \longrightarrow \text{“FirstSecond”}$$

String Indexing

```
myString = “Hello World”
```

$$\texttt{myString[4]} \longrightarrow \text{“o”}$$

$$\texttt{myString[4:-1]} \longrightarrow \text{“orl”}$$

String Operations

String Slicing

```
myString = “CSCA20 is lots of fun”
```

$$\texttt{myString[:4]} \longrightarrow \texttt{“CSCA”}$$

$$\texttt{myString[4:]} \longrightarrow \texttt{“20 is lots of fun”}$$

$$\texttt{myString[4:6]} \longrightarrow \texttt{“20”}$$

String Slicing

String Slicing

```
mySTring[a : b]
```

Included

Not Included

Quiz Time

```
myString = "COMPUTERS"
```

What is `myString[:4]`?

A) "COMPU"

B) "COM"

✓ C) "COMP"

D) "UTERS"

Quiz Time

```
myString = “COMPUTERS”
```

What is myString[-1]?

[✓ A) “S”]

B) “COMPUTERS”

C) “C”

D) None of the above

Quiz Time

```
myString = “COMPUTERS”
```

What is `myString[:300]`?

A) “CO”

[✓ B) “COMPUTERS”]

C) “S”

D) None of the above

Quiz Time

```
myString = “COMPUTERS”
```

What is myString[12]?

A) “S”

B) “COMPUTERS”

C) “C”

[✓ D) None of the above]

Quiz Time

```
myString = “COMPUTERS”
```

What is `myString[5:]`?

A) “COMPU”

[✓ B) “TERS”]

C) “ERS”

D) “T”

Quiz Time

```
myString = “COMPUTERS”
```

What is myString[6:6]?

A) “TE”

B) “E”

[✓ C) “”]

D) None of the above

Quiz Time

```
myString = “COMPUTERS”
```

What is myString[6:7]?

A) “TE”

[✓ B) “E”]

C) “”

D) None of the above

Quiz Time

```
myString = “COMPUTERS”
```

What is `len(myString)`?

A) 8

[✓ B) 9]

C) “COMPUTERS”

D) None of the above

Quiz Time

```
myString = “COMPUTERS”
```

What is `len(myString[6:6])`?

A) 9

B) 1

[✓ C) 0]

D) 2

Quiz Time

```
myString = “COMPUTERS”
```

What is `len(myString[5:])`?

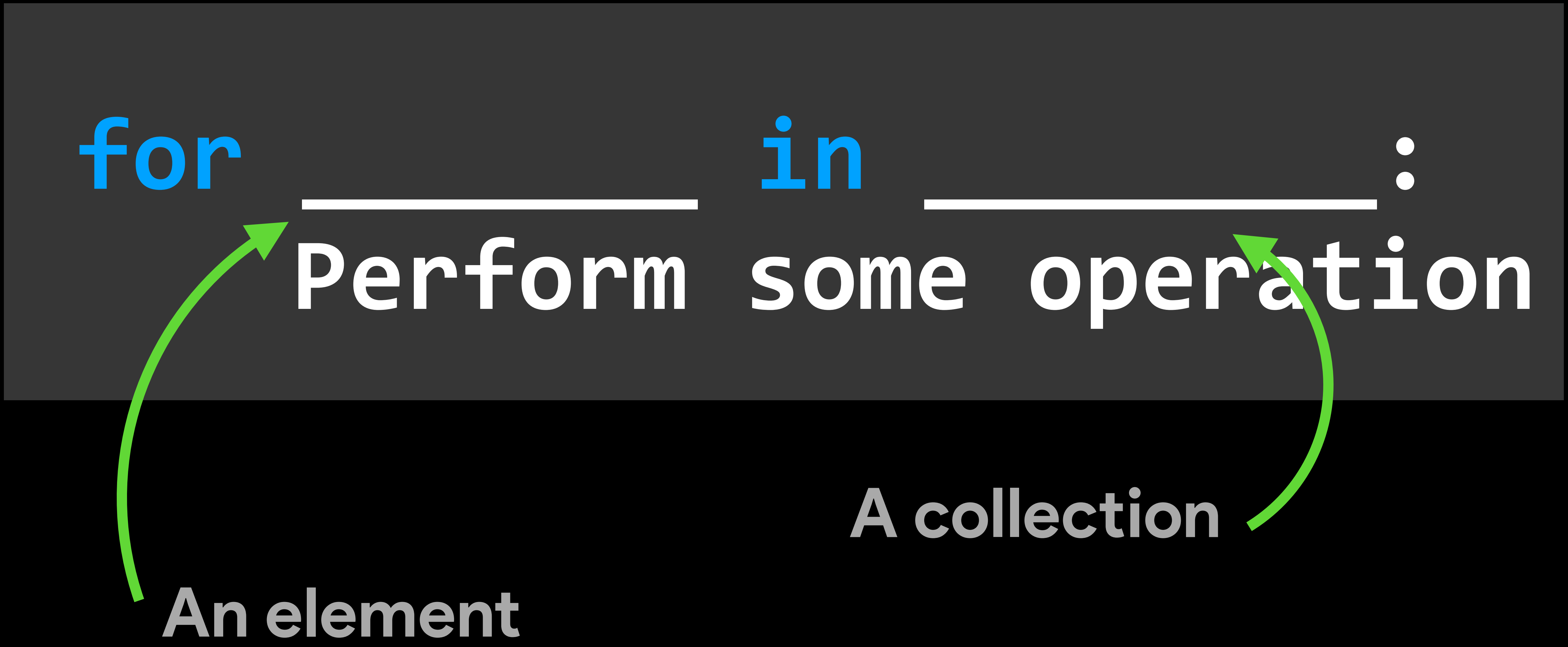
A) 5

[✓ B) 4]

C) 3

D) 0

For Loops



```
for ________ in __________:
      Perform some operation
```

A for loop iterates over the collection one element at a time, performing the operation you defined in the loop body.

For Loops

```python
for num in range(4):
   print(num)
```

```
0
1
2
3
```

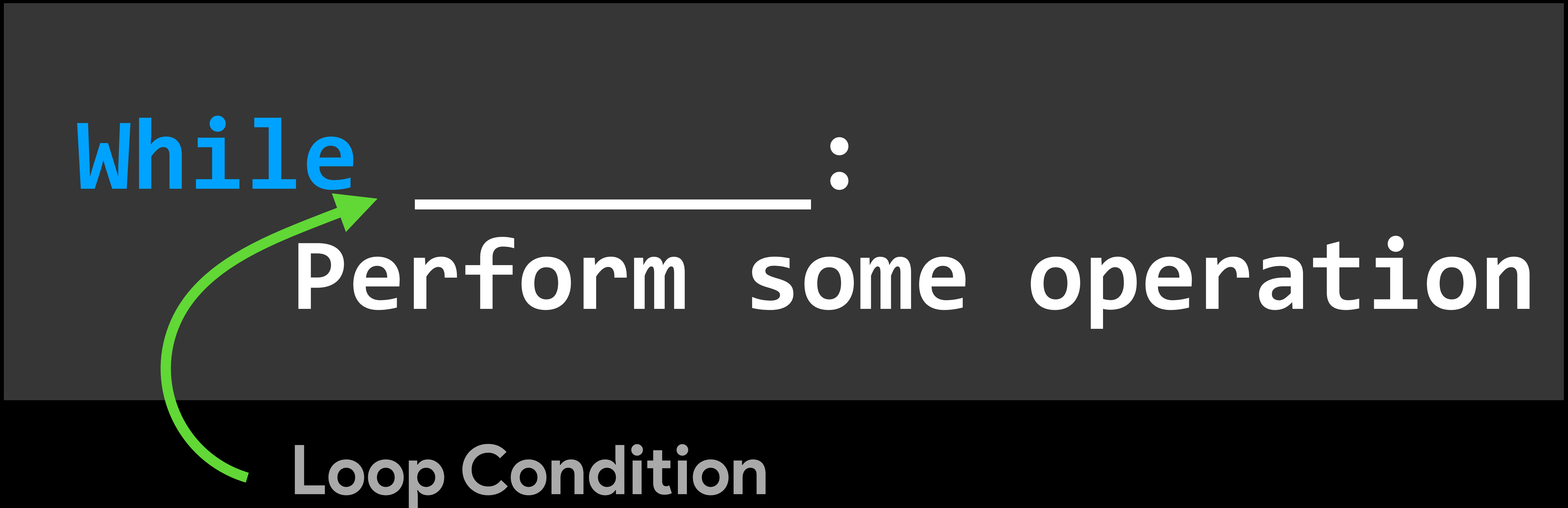
```python
for num in range(len("hi")):
   print(num)
```

```
0
1
```

```python
result = 1
for num in range(3):
   result +=(num)
   print("num: " + str(num))
   print("result: " + str(result))
```

```
num: 0
Result: 1
Num: 1
Result: 2
Num: 2
Result: 4
```

While Loops



```
While _________:
    Perform some operation
```

A while loop repeatedly performs the operation you defined in the loop body so long as the loop condition holds.

While Loops

```python
num = 0
while num < 4:
    print(num)
    num += 1
```

```
0
1
2
3
```

```python
result = 1;
num = 0
while num < 3:
    result +=(num)
    print("num: " + str(num))
    print("result: " + str(result))
    num += 1
```

```
num: 0
Result: 1
Num: 1
Result: 2
Num: 2
Result: 4
```

L STS

Lists

A list is an **ordered collection of objects.** ie:

```
[True,  3,  “bob”, 2.39]

[“Sara”, “Anna”, “Karen”]

[2, 4, 8, 16, 32, 64]
```

Lists

We can also slice and index lists the way we do with strings

```
myList = [True,  3,  “bob”, 2.39]

myList[:2]  →    [True, 3]
```

List Operations

We can concatenate two lists using the + operator.

```
Girls = [“Sara”,  “Anna”,  “Karen”]
Boys  = [“Billy”,  “John”]
```

```
Girls + Boys
```

```
[“Sara”, “Anna”, “Karen”, “Billy”, “John”]
```

List Operations

We can repeat elements within a list using the * operator.

```
[“Sara”] * 3
```

$$\begin{tikzcd} {} \arrow[r, bend right] & {} \end{tikzcd}$$

```
[“Sara”, “Sara”, “Sara”]
```

We can check for membership within a list using the in keyword.

Lists and Strings

Both Lists and Strings are Ordered Collections

A String is an array of characters

“H”	“E”	“L”	“L”	“O”	“ “	“W”	“O”	“R”	“L”	“D”

A list is an array of objects

“Hello”	3.14	True	42	False	“bloop”

List Functions

`cmp(list1, list2)`

Compares the two given lists

`min(list)`

Returns the minimum in the list

`max(list)`

Returns the maximum in the list

`len(list)`

Returns the length of the list

List Methods

- `List.append(obj)`

  Inserts obj to the end of the list

  `List.count(obj)`

  Returns the number of occurrences of obj

  `List.index(obj)`

  Returns the first index of obj in the list

- `List.insert(index,obj)`

  Inserts obj at the given index in the list

List Methods

`List.pop()`

Removes and returns the last object in list

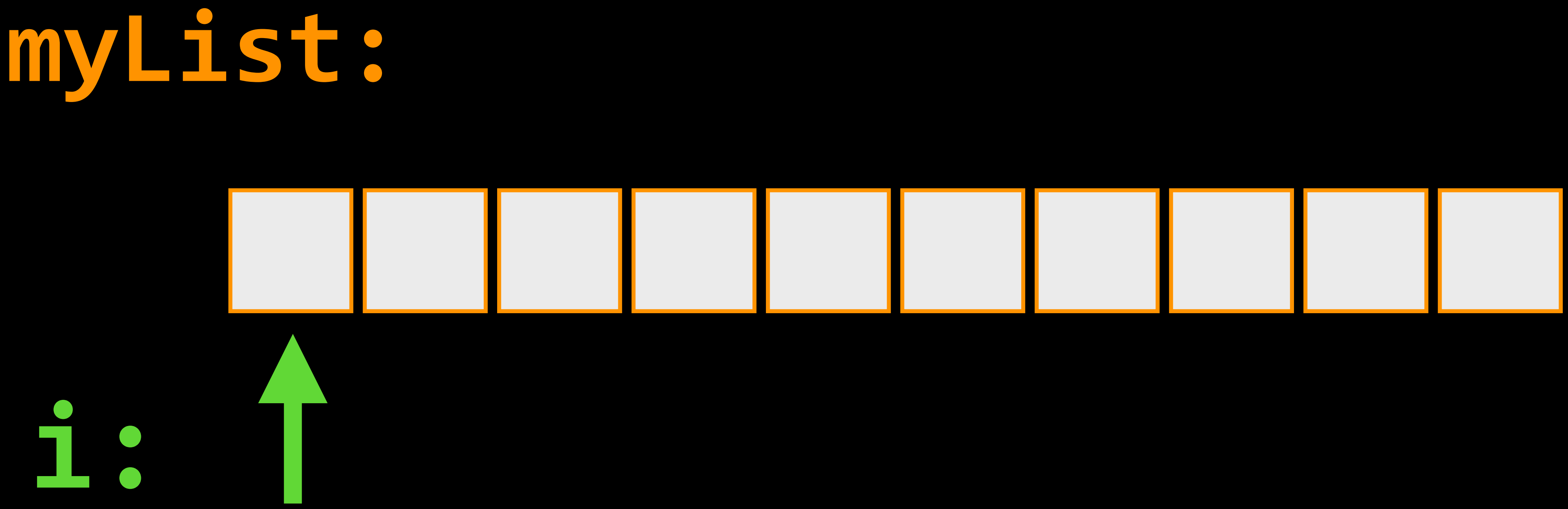
- `List.remove(obj)`

  Removes obj from the list

- `List.reverse(obj)`

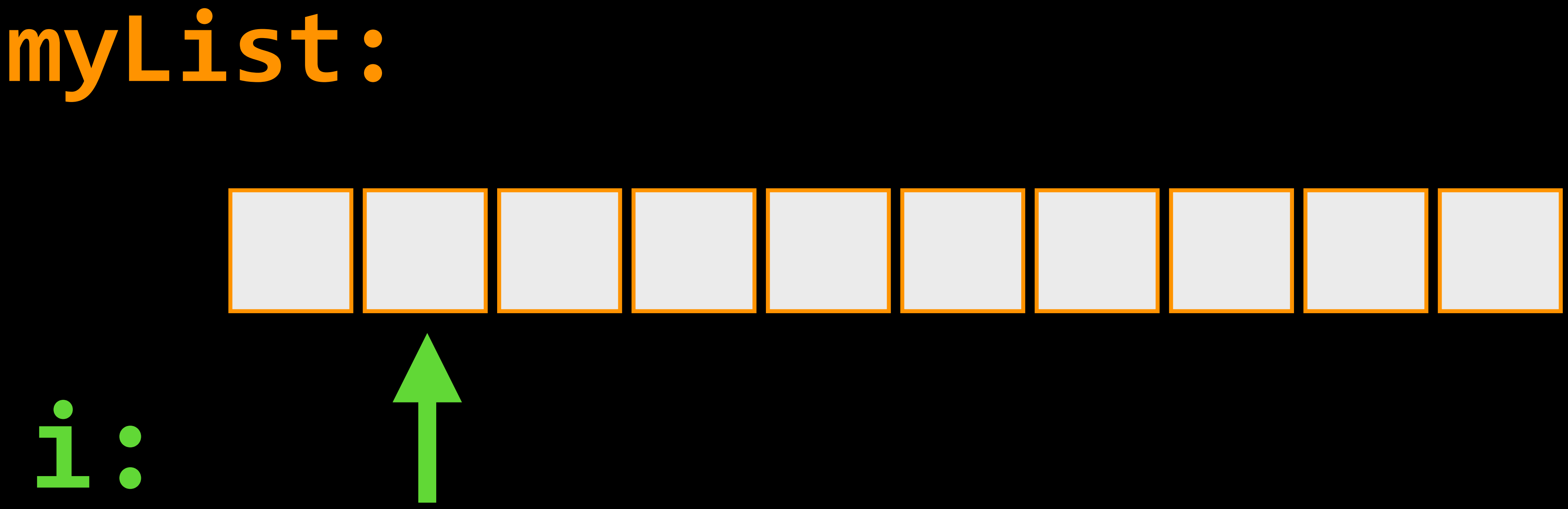
  Reverses the order of objects in the list

Iterating Through a List



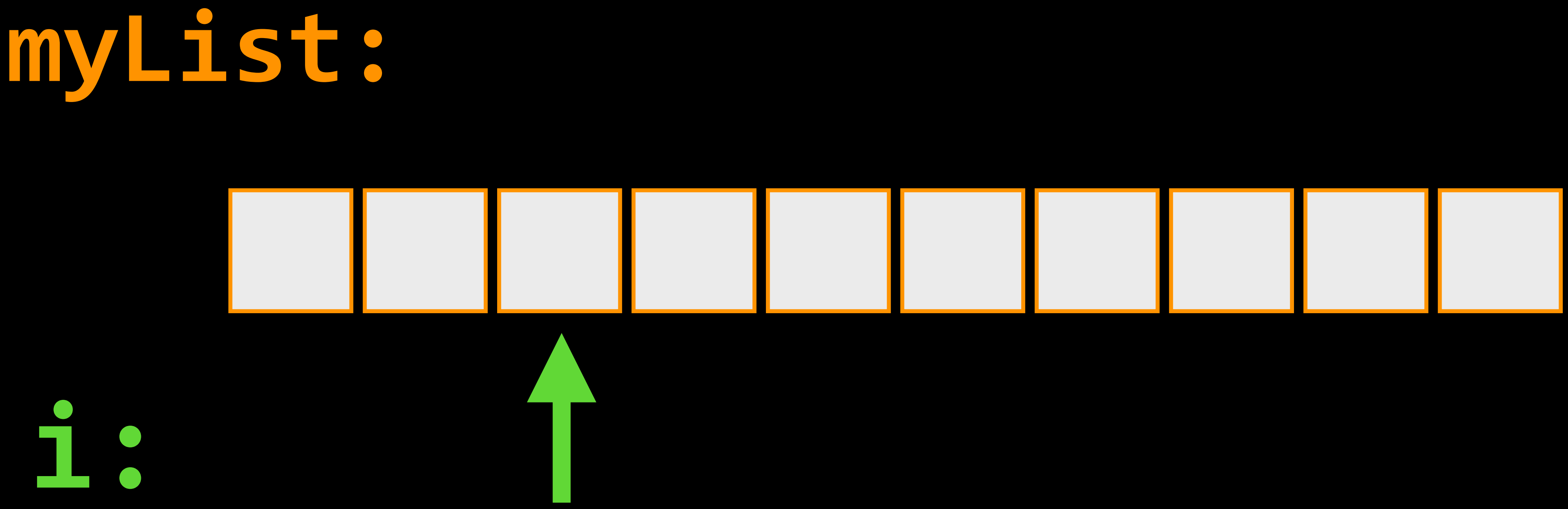
```
for i in myList:
```

Iterating Through a List



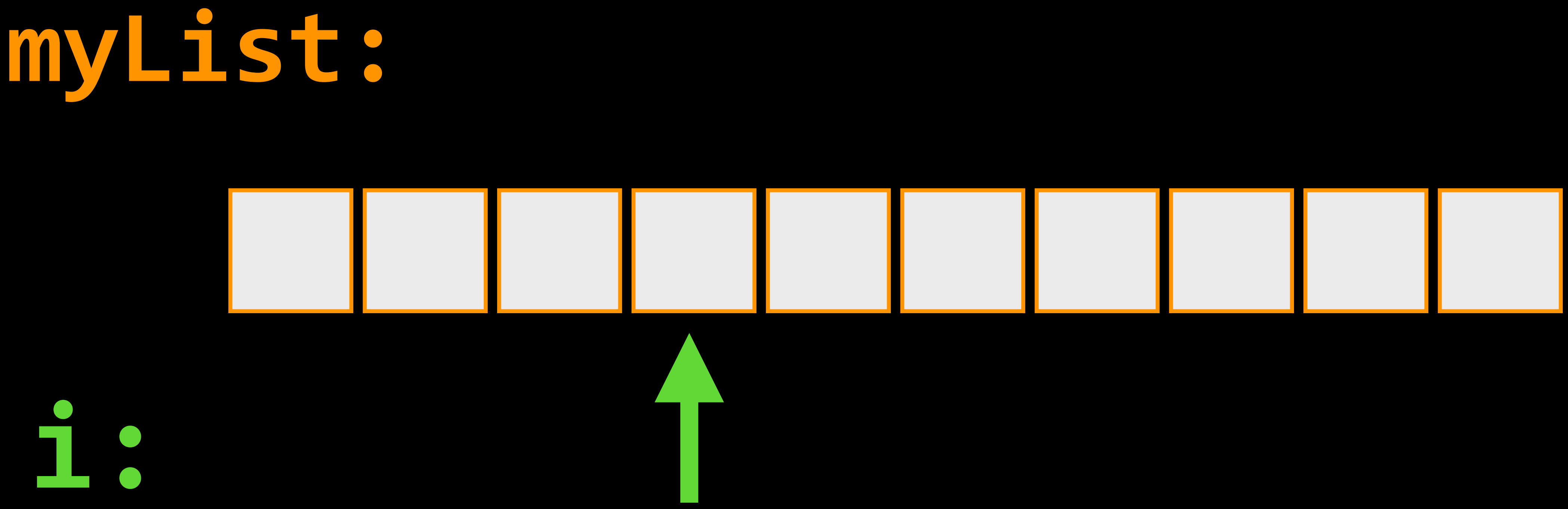
```
for i in myList:
```

Iterating Through a List



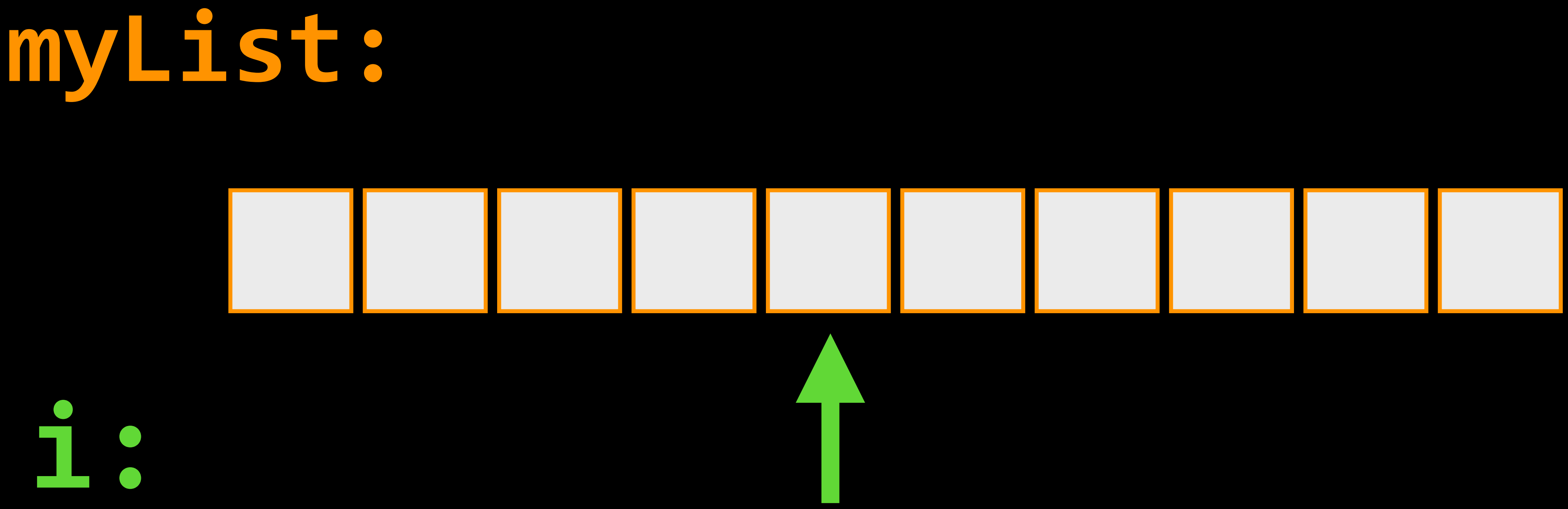
```python
for i in myList:
```

Iterating Through a List



```python
for i in myList:
```

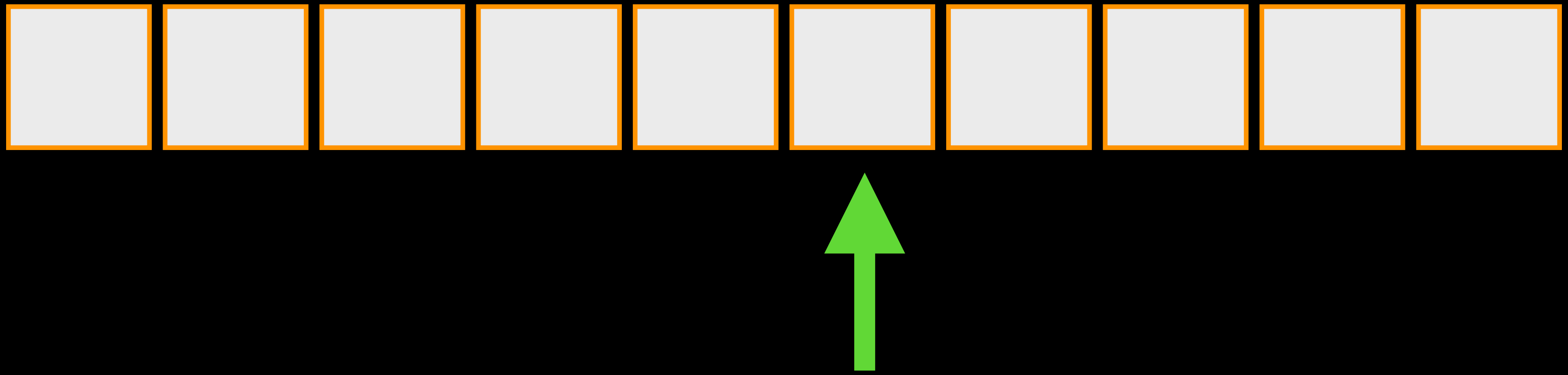
Iterating Through a List



```
for i in myList:
```

Iterating Through a List

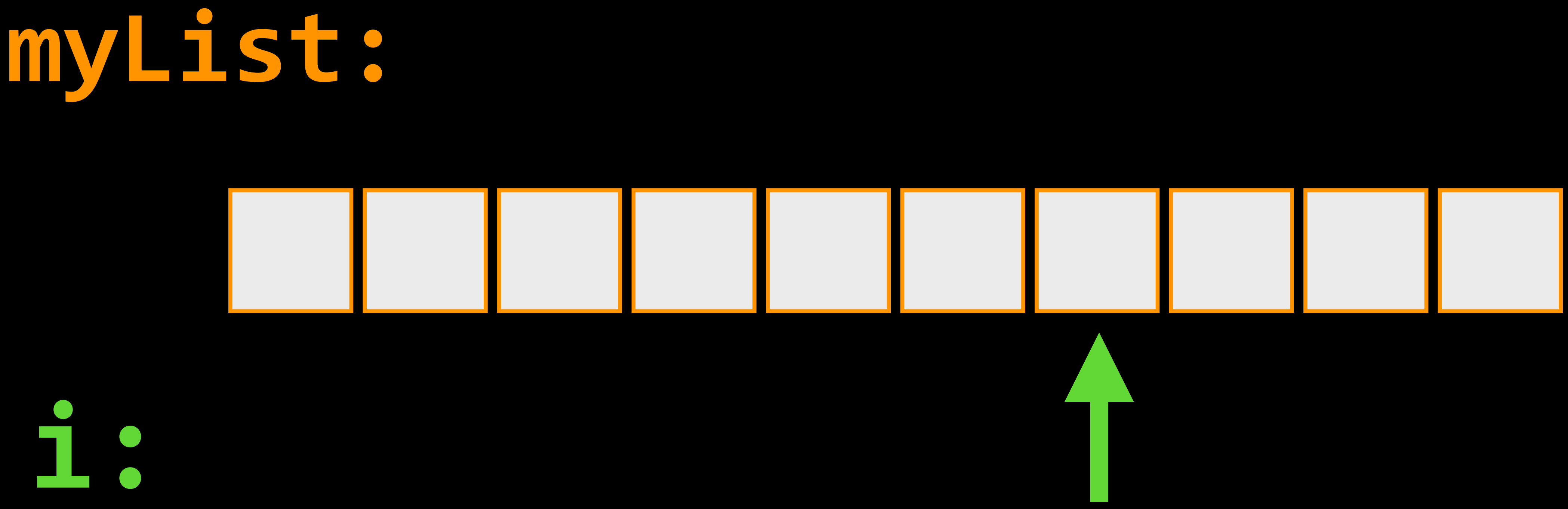
myList:



i:

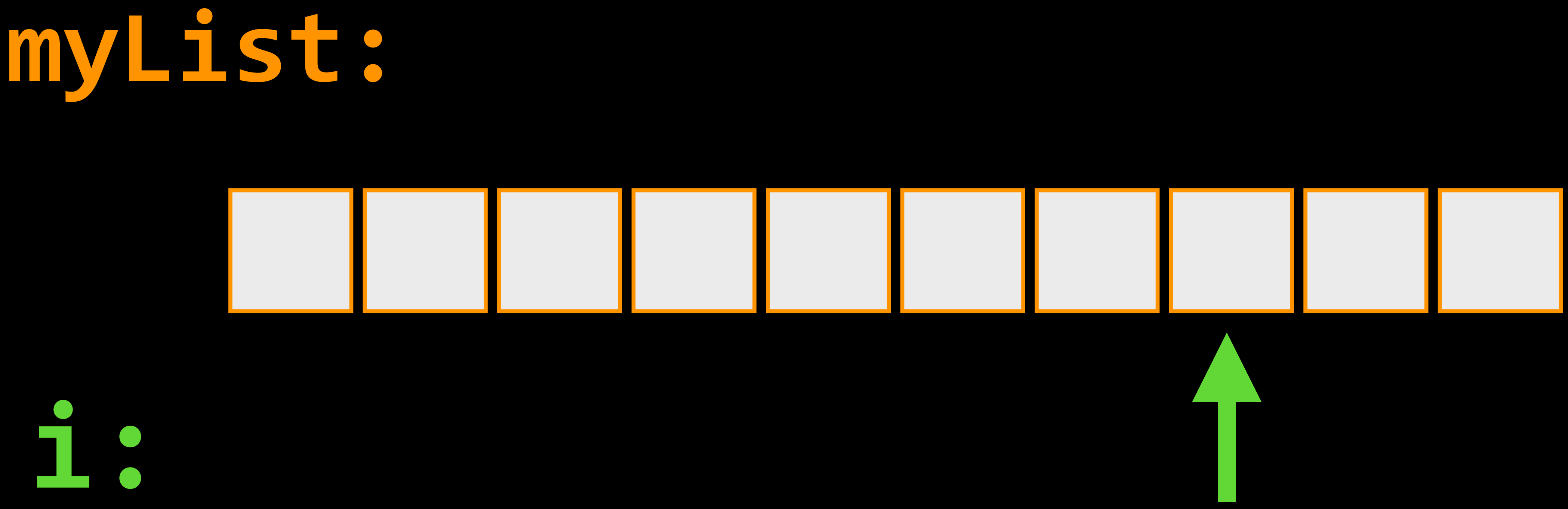
```
for i in myList:
```

Iterating Through a List



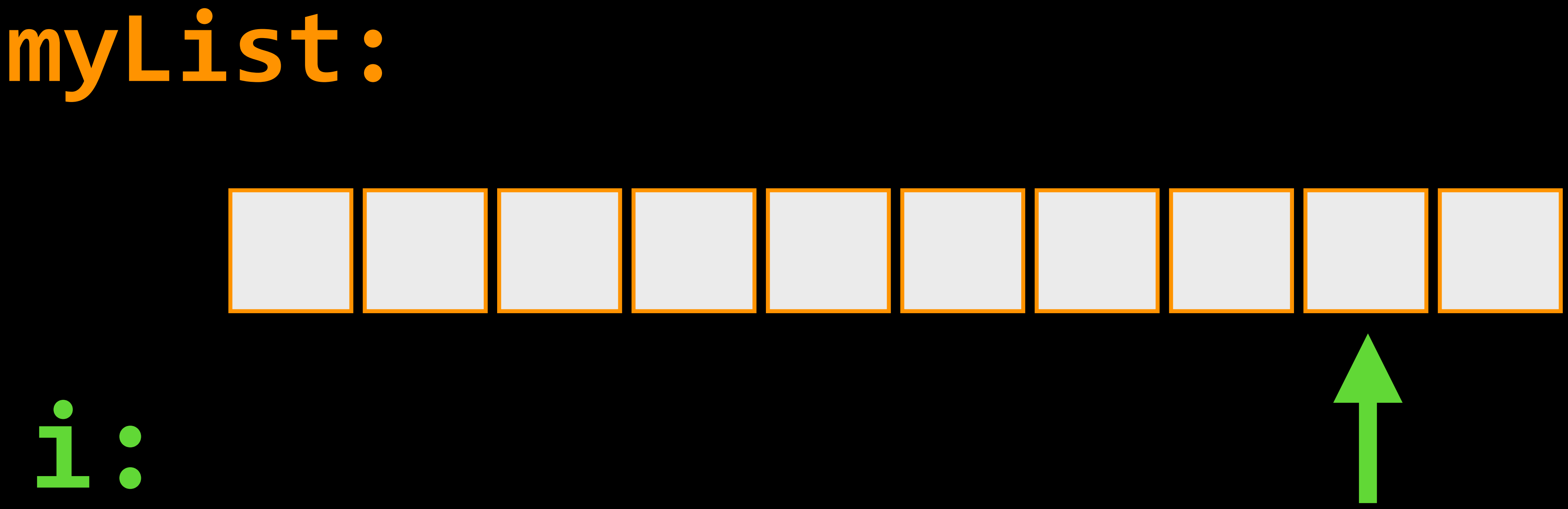
```python
for i in myList:
```

Iterating Through a List



```
for i in myList:
```

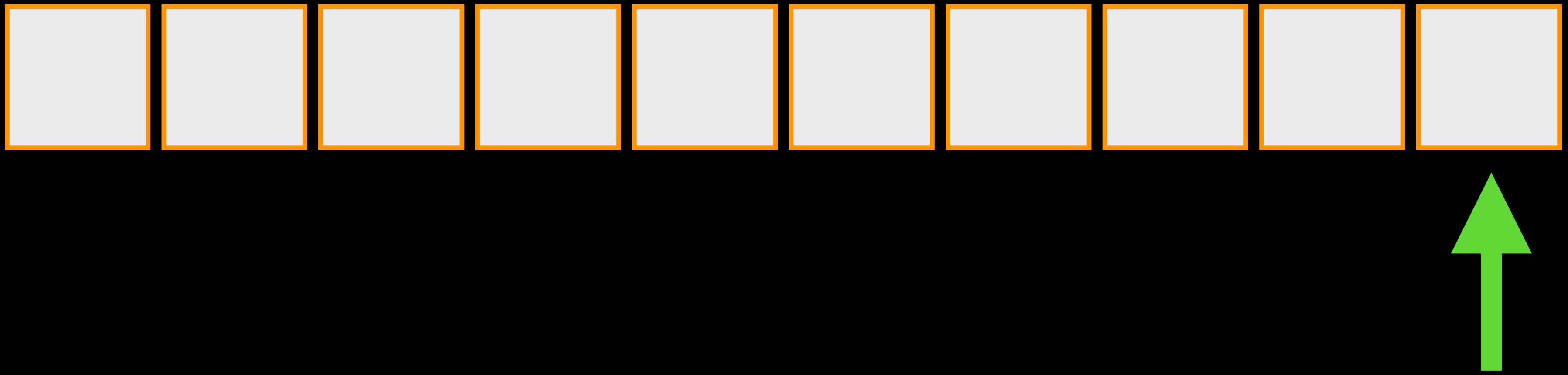
Iterating Through a List



```
for i in myList:
```

Iterating Through a List

myList:

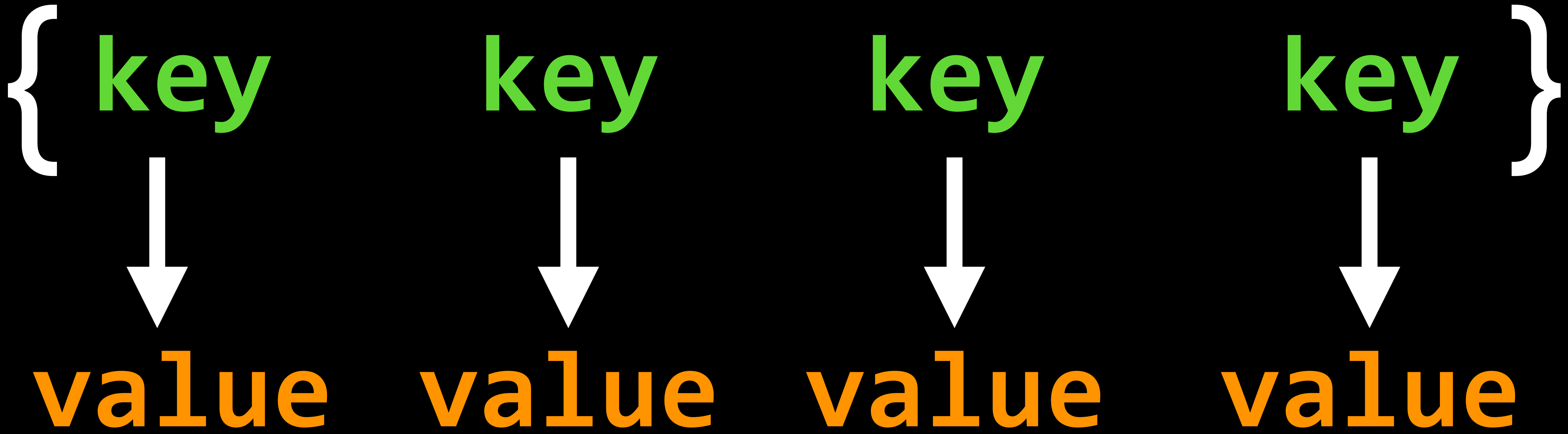


i:

```
for i in myList:
```

DICTIONARIES

Python Dictionaries



A dictionary is a collection which is unordered, mutable and indexed.
In Python dictionaries are written with curly brackets, and they have keys and values.

Python Dictionaries

```
        key     value
myDict = { "Name" : "Kara"
           "Age" : 19
           "Occupation" : "TA"
           "Program" : "CS" }
```

Setting Up a Dictionary

You know that to create:

- A new String: `myStr = “”`
- A new List: `myList = []`

We know that dictionaries are denoted with curly braces {} so, intuitively:

`myDict = {}` OR `myDict = dict()`

Adding Values to a Dictionary

To add a new value to a dictionary, we must add a key, and give it a value.

```
myDict[key] = value
```

For example:

```
kara = dict()
kara[“name”] = “Kara”
kara[“age”] = 19
kara[“job”] = “TA”
```

```
{“name”:“Kara”, “age”:19, “job”:“TA”}
```

Reading Values from a Dictionary

To read an existing value to a dictionary, we must reference a key.

```
value = myDict[key]
```

For example:

```
{“name”:“Kara”, “age”:19, “job”:“TA”}
```

```
kara[“name”]    ⟶  “Kara”
kara[“age”]     ⟶  19
kara[“job”]     ⟶  “TA”
kara[“address”] ⟶  ERROR
```

Removing Values from a Dictionary

To remove an existing value to a dictionary, we must pop the key value pair by referencing a key.

```
myDict.pop(key)
```

For example:

```
{“name”:“Kara”, “age”:19, “job”:“TA”}
```

```
kara.pop(“job”)
```

```
{“name”:“Kara”, “age”:19}
```

Merging Dictionaries

To merge two dictionaries, we can use the update method to join them into a single dict.

```
myDict.update(anotherDict)
```

For example:

```
{“name”:“Kara”, “age”:19}

kara.update({“job”:“TA”})

{“name”:“Kara”,“age”:19, “job”:“TA”}
```

Important Dictionary Methods

`Dictionary.copy()`

Returns a copy of the dictionary

`Dictionary.clear()`

Removes all elements from the dictionary

`Dictionary.keys()`

Returns a list of the dictionary’s keys

`Dictionary.values()`

Returns a list of the dictionary’s values

Old Friends We Can Rely On

`in` [KEYWORD]

Is a key in our dictionary?

`len()`

How many keys are in our dictionary?

`type()`

Is our variable a dictionary?

`del` [KEYWORD]

Clears the value of a variable

Combining Dictionary Methods

```
sorted(dict.keys())
```

Returns a list of sorted keys in the dictionary

```
sorted(dict.get(key))
```

Returns a sorted list of the values at the key

```
type(dict.get(key))
```

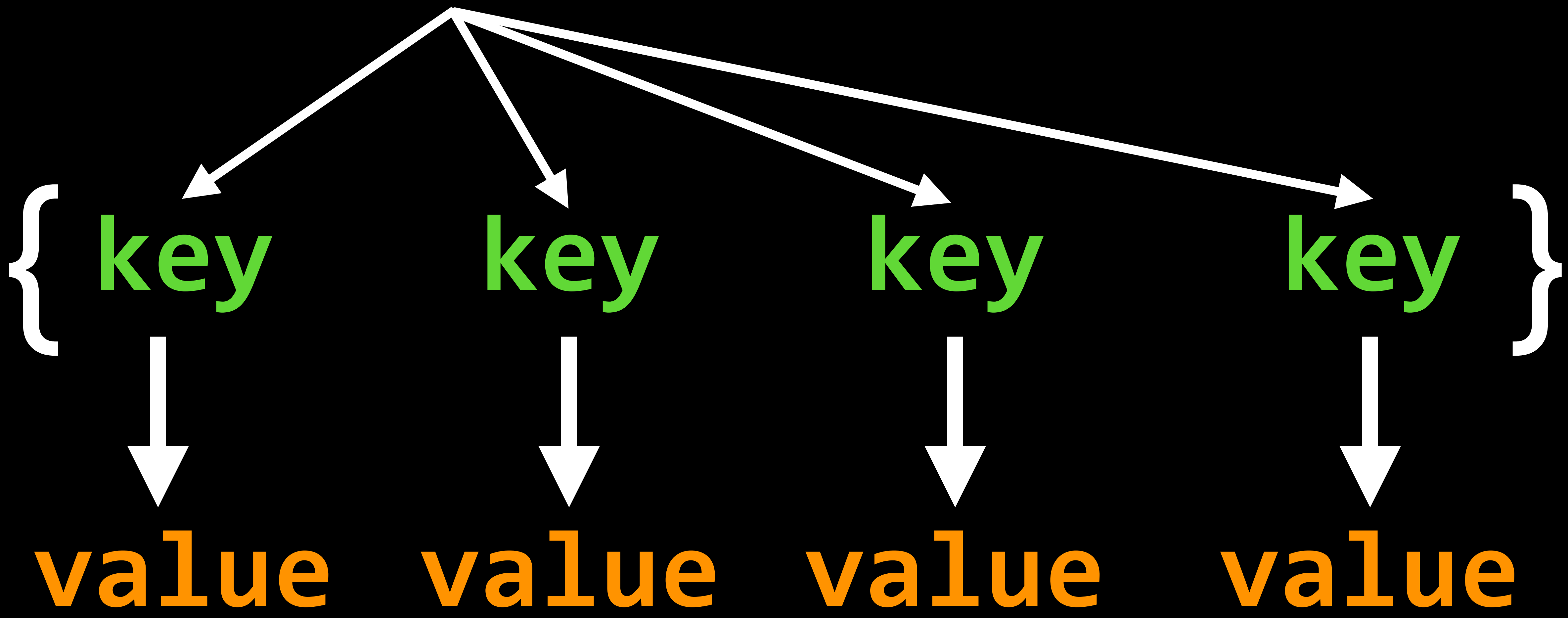
Tells us the type of value at the key

... And many more!

Looping Through a Dictionary

When we use a for loop with a dictionary the way we're used to doing it, we're iterating through the keys.

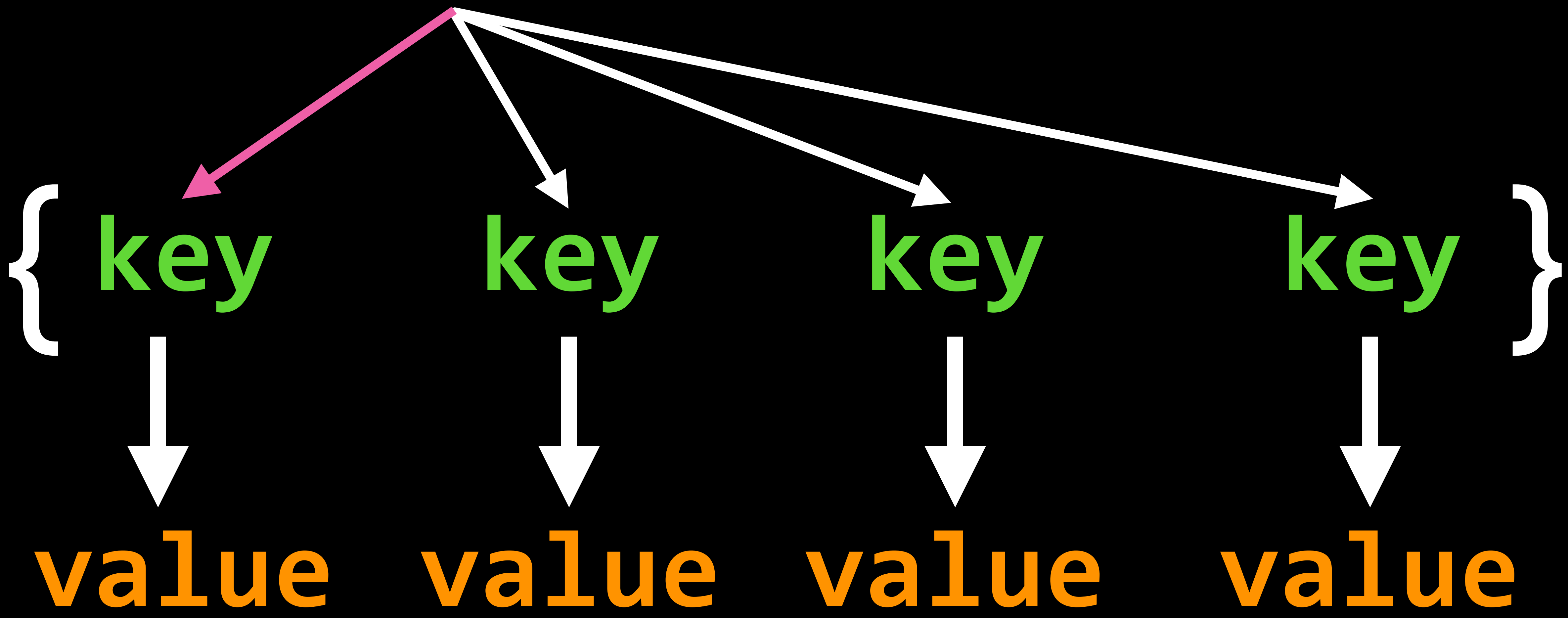
```
for i in dictionary:
```



Looping Through a Dictionary

When we use a for loop with a dictionary the way we're used to doing it, we're iterating through the keys.

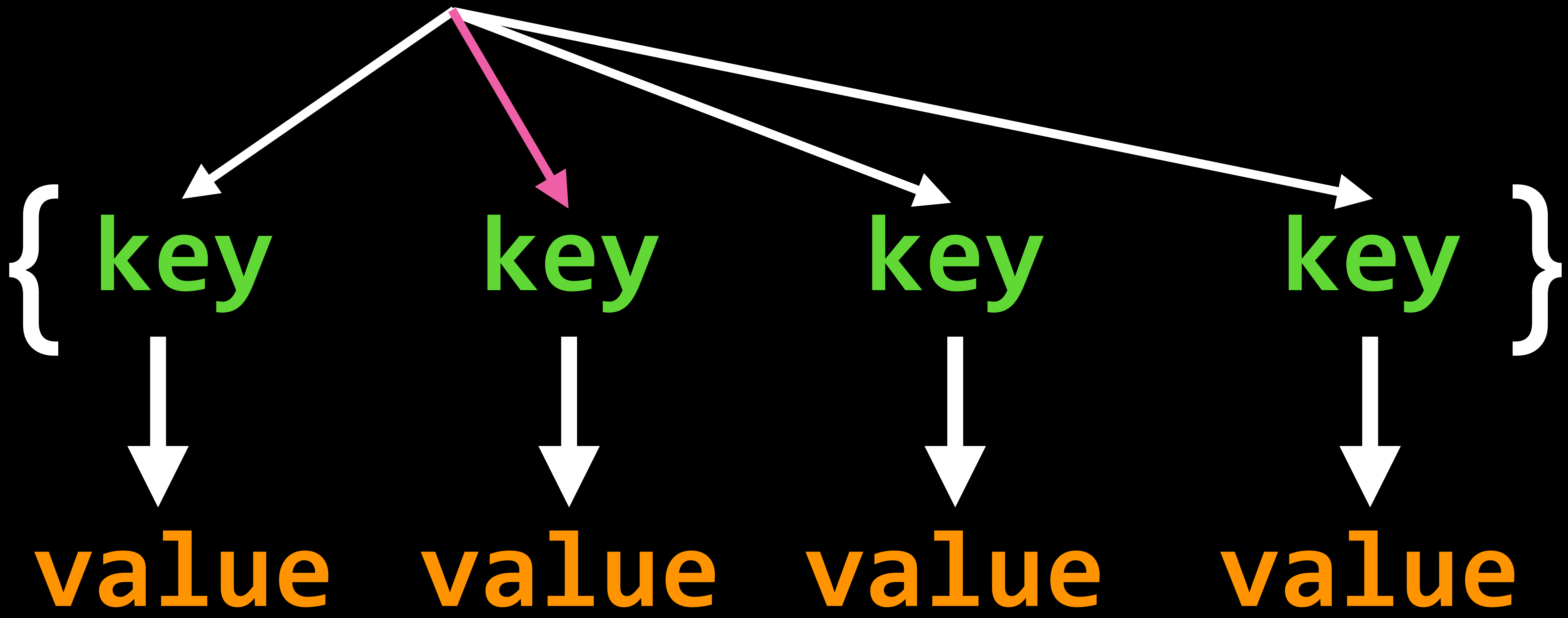
```
for i in dictionary:
```



Looping Through a Dictionary

When we use a for loop with a dictionary the way we're used to doing it, we're iterating through the keys.

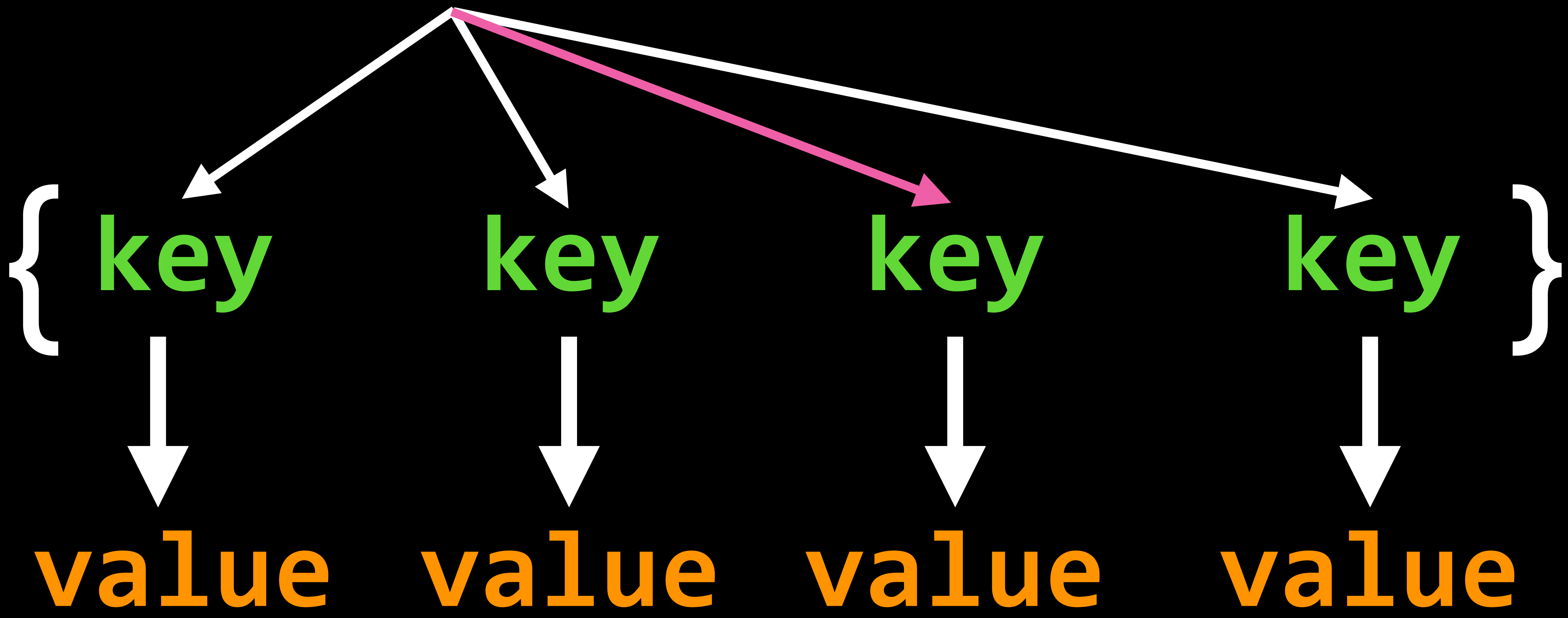
```
for i in dictionary:
```



Looping Through a Dictionary

When we use a for loop with a dictionary the way we're used to doing it, we're iterating through the keys.

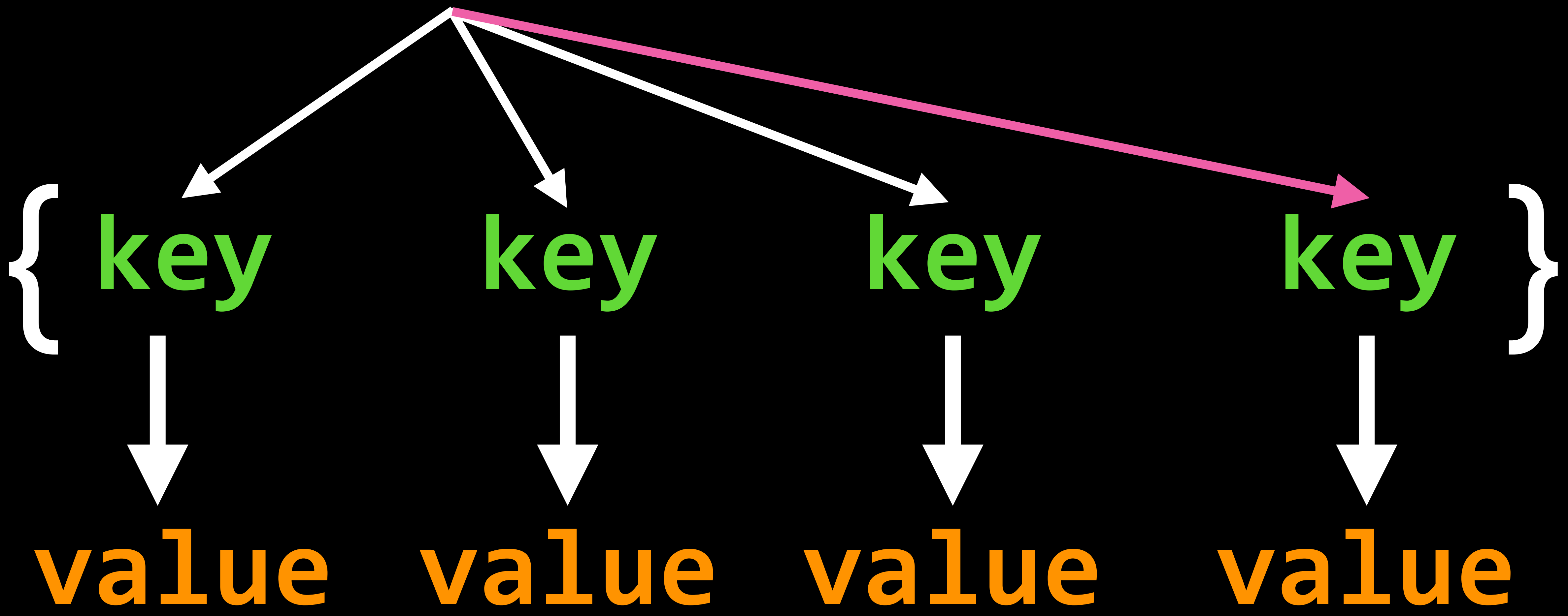
```
for i in dictionary:
```



Looping Through a Dictionary

When we use a for loop with a dictionary the way we're used to doing it, we're iterating through the keys.

```
for i in dictionary:
```



F LES

Opening files

```
with open(file) as myFile:
```

Open a file by name
(In the same directory)

Store the file
into a variable

Now we can do something with our file inside the `with` block.

When the block finishes executing, the file will be closed automatically.

Opening files

```
myFile = open(file)
```

Store the file into a variable

Open a file by name
(In the same directory)

myFile will be the variable that holds the open file. We can work with it the same way we would in a with block, except we must remember to close the file when we're done.

Opening files

```
myFile = open(file, mode)
```

Open a file by name
(In the same directory)

Indicate what we want to do with the file

We can (and should) indicate what we intend to do with our open file:

“r”: read.	Read the contents of the file only
“w”: write	Clears the file for writing into
“a”: append	Write into the file after its content

Closing files

```
myFile.close()
```

We need to make sure that
the file we indicate is currently open

If we opened a file manually, as was shown on the previous slide, we must ensure that we close it before the program exits.
This is very important!

Why Should I Close A File?

Why do you shut your computer down instead of pulling the power cord out?

We don’t want to cause conflicts with other applications that might use the file later.

We don’t want to hog more memory (RAM) than we need.

It’s like clicking “eject” before pulling out a flash drive.

Which way is better?

`with` Block	Manually
You don't need to worry about closing files	It's easy to forget to close open files
If your code causes an error, the file will close automatically	If your code causes an error, your program will crash
You have to remember to indent the block	No indenting is needed. (Yay?)

Looping Through a File

```
for ______ in myFile:
```

Line in you open file
(In the same directory)

myFile is an
open file

When we loop through a file by element, we read one line at a time; up to each newline ("\n") character — what you get when you hit the enter key on your keyboard

Important File Methods:

`File.readline()`

Reads the next unread line in the file.

(This tracks your place in the file)

`File.readlines()`

Returns a list containing all the lines in the file.

`File.write(text)`

Writes the given text to the open file.

Like `print`, except the output goes into the file.

CSV Files

CSV stands for Comma Separated Values.

A CSV file is a translation of a table into text. Programs like MS Excel, and Numbers read and generate CSVs out of spreadsheets.

Values in the table are separated with commas, without spaces. Think of these commas as dividers in a table.

CSV Files

Name	Age	Gender
Linda	34	F
Joseph	8	M

```
Name,Age,Gender
Linda,34,F
Joseph,8,M
```

Introduction to Databases

Why talk about databases?

Databases are one of the most important topics in computer sciences!

Almost all organizations, whether private or public, use databases in one way or another

You use databases every single day without even realizing it!

What is a Database?

A database is just a well-structured collection of data.

Data should be easily stored and retrieved

Often data is stored in the form of tables where the headers are properties, and each row represents an entry

What is a Database?

The format of which data is stored in a database is called its schema.

Uploads:

← Attributes →

↑ Entries ↓

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

What is a Database?

Does this sound familiar?
It should! This is how CSV files are formatted!

`Uploads:`

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Database Example

Suppose that this is the schema that an image search platform uses to store data.

`Uploads:`

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Database Example

How do we search for names of images that contain the tag “weekend”?

Uploads:

Attributes

Entries

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Our query will be something along the lines of:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Our query will be something along the lines of:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Our query will be something along the lines of:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Our query will be something along the lines of:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Alternatively, you can think of it this way:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Alternatively, you can think of it this way:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name	uploader	image_size	search_tags
“img_1134.png”	“mrBubbles123”	30	[“cats”, “weekend”, “12”]
“img_6126.jpg”	“hanna_mclean”	13	[“beach”, “sun”, “trip”]
“dsc_2342.tiff”	“ms_skittles”	45	[“baby”, “weekend”, “cute”]
“img_4911.jpeg”	“space_invader”	35	[“game”, “fortnite”, “boy”]
…	…	…	…

Alternatively, you can think of it this way:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS "weekend"
```

Uploads:

image_name	uploader	image_size	search_tags
"img_1134.png"	"mrBubbles123"	30	["cats", "weekend", "12"]
"dsc_2342.tiff"	"ms_skittles"	45	["baby", "weekend", "cute"]

Alternatively, you can think of it this way:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name
“img_1134.png”
“dsc_2342.tiff”

Alternatively, you can think of it this way:

```
SELECT image_name
FROM uploads
WHERE search_tags CONTAINS “weekend”
```

Uploads:

image_name
“img_1134.png”
“dsc_2342.tiff”

→

```
(“img_1134.png”, “dsc_2342.tiff”)
```

Writing A Query

A query defines the parameters for the search that we want to perform on a database.

```
SELECT [some attribute or column]
FROM   [some table]
WHERE  [some condition is true]
```

Depending on the version of SQL that you use, the exact syntax will vary, but the idea is always the same.

Writing A Query

```
FROM [The name of the table we examine]
```

Before we can do a SELECT operation, we must first indicate which table we want SELECT from.

The FROM block will always be run first so that the query has a starting point.

Writing A Query

```
SELECT [some attribute or column]
```

When we select from a database, we want to make sure that the argument is a column or set of columns in our table.

We can also use SELECT * to denote that we want to select ALL the columns.

Writing A Query

```
WHERE [some condition holds true]
```

When we select from a database, we want can include a WHERE block to narrow down our search results to just a certain entries.

The WHERE block is technically optional, but it's what gives you the actual search functionality.

SQL

SQL is a database management system that can be integrated into various programs and have numerous implementations that work with many programming languages.

In this course, we'll be using pySQLite using the `sqlite3` API (This is the module you have to import)

SQL is not quite like Python:
Python is used to do general computations,
SQL is used manipulate tables in a database.

SQL

SQL is a database management system that can be integrated into various programs and have numerous implementations that work with many programming languages.

In this course, we'll be using pySQLite using the `sqlite3` API (This is the module you have to import)

SQL is not quite like Python:
Python is used to do general computations,
SQL is used manipulate tables in a database.

Working With A Database

The first thing we need to do is **import** the sqlite3 module.

```
import sqlite3
```

Next, we need to **connect** to our database
and **link** to it using a cursor. Now we can do some **work**.

```
connection = sqlite3.connect(name of database)
cursor = connection.cursor()
```

Once we are done making changes, we need to **save**.

```
connection.commit()
```

After all changes have been saved, **close** all connections.

```
cursor.close()
connection.close()
```

Manipulating the Database

The cursor is a link to your database.
in other words, if you want to do something to your database, you must reference it using the cursor.

If you want to think of the database as a Object like a String, List, Dictionary etc, then the cursor is the database object that contains a set of database tools.

.execute()

The database cursor's execute method isn't a conventional method like those that you're used to seeing. It doesn't do any one thing...

.execute() does to the database whatever you tell it to do in SQL!

In other words, it's the bridge between your Python code and the SQL that modifies the database.

`.execute()`

SQL queries are always written and passed to `.execute()` as a string.

The SQL itself specifies the operation that `.execute()` performs on your database!

Common Table Tasks

Here are some common tasks that can be done using SQL and the cursor's `.execute()` method:

```
DROP TABLE IF EXISTS table_name
```

If the table already exists, erase it and set it up all over again. This should be used inside your functions before you create any new table.

```
CREATE TABLE table_name(columnName TYPE …)
```

Creates a new table with the given name and columns.

Columns must indicate the names of each column and the type of data that should go into that column. These types are not the same across Python and SQL!

Python Type	SQL Type
`Str`	`TEXT`
`Float`	`REAL`
`Int`	`INTEGER`

Common Table Tasks

```
SELECT columns FROM table WHERE condition
```

SQL queries (Of the format we discussed earlier) Can also be passed into .execute(), indicating that we want to search the database.

```
(INSERT INTO table VALUES (?, ?, …), data)
```

Add an entry (also called a VALUE or row) into the table. This is the query that must be paired with an actual dataset. Each “?” Is a placeholder for an attribute of the actual dataset.

```
(INSERT INTO table VALUES (?, ?, …), data)
```

Suppose we have a table called Uploads:

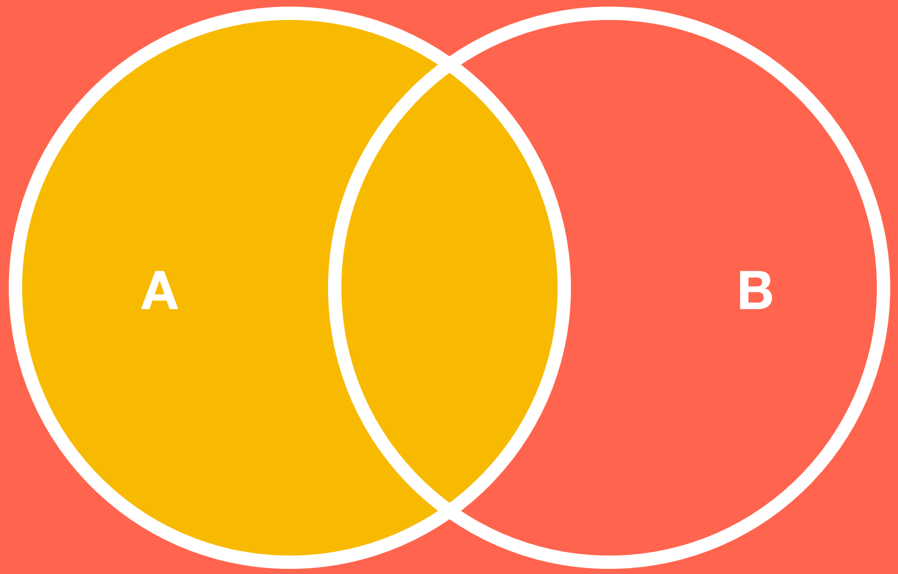
image_name	uploader	image_size
“img_1134.png”	“mrBubbles123”	30
“img_6126.jpg”	“hanna_mclean”	13

We want to add this row:

“dsc_2342.tiff”	“ms_skittles”	45

```
query = “INSERT INTO uploads VALUES (?, ?, ?)”
data = (“dsc_2342.tiff”, “ms_skittles”, 45)
cursor.execute(query, data)
```

table joins



When Are Joins Useful?

Sometimes the information we need is Spread across more than one table

We need some way to relate this data in a way that makes sense, and is still easy to access

Solution:

We can combine multiple smaller tables into a single larger table that contains all the information we want!

Types of Joins

There are many types of joins that we can use depending on the data we're working with:

In lecture, David talked about:

- Left Joins
- Right Joins (Not In SQLite)
- Inner Joins
- Full Outer Joins (Not In SQLite)
- Cross Joins (This one is very different!)

Types of Joins

There are many types of joins that we can use depending on the data we're working with:

In lecture, David talked about:

Left Joins

Right Joins (Not In SQLite)

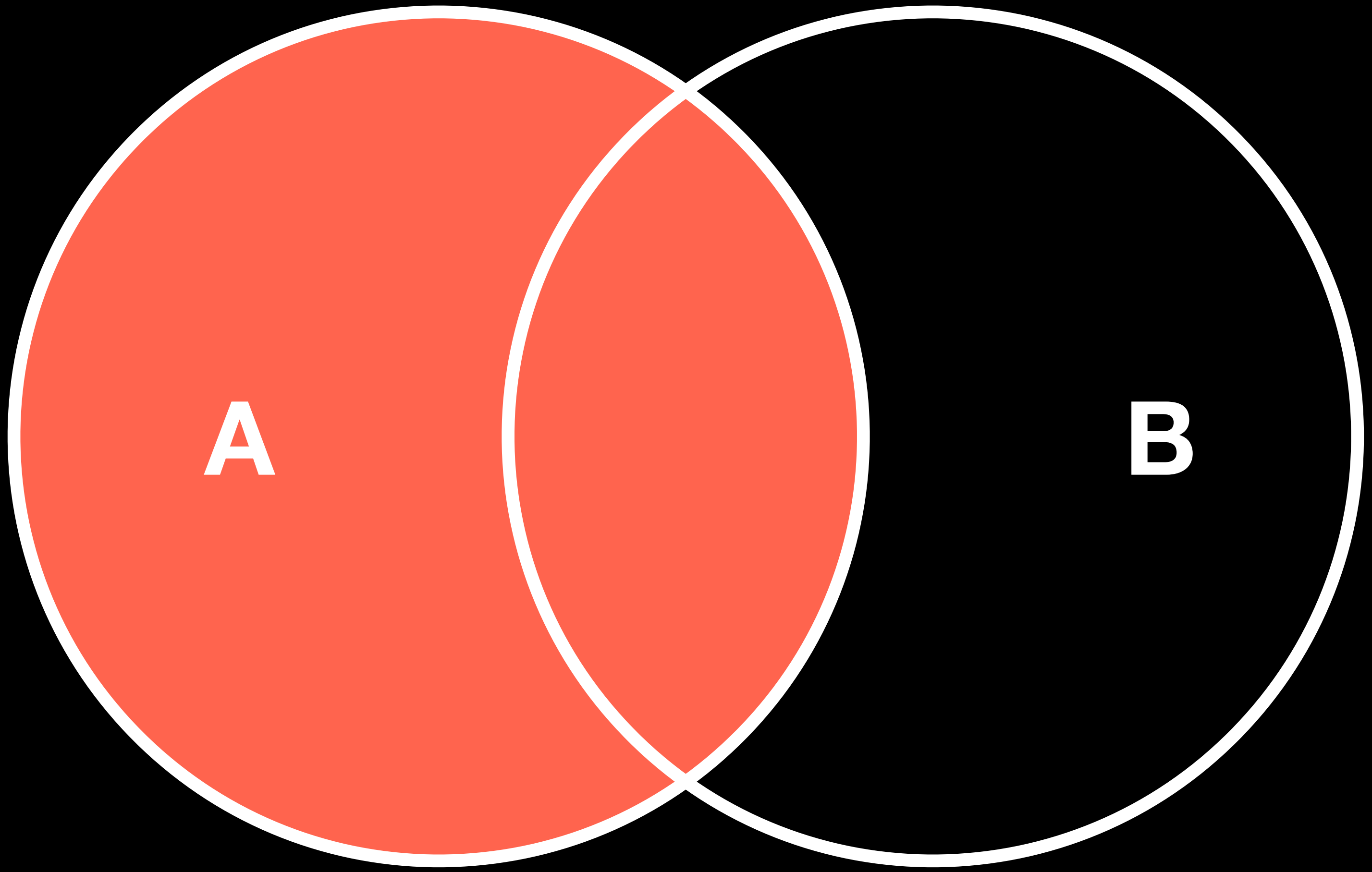
Inner Joins

Full Outer Joins (Not In SQLite)

Cross Joins

We'll talk about these today.

Left Join



```
SELECT [some attribute or column]
FROM   A LEFT JOIN B
ON     A.key = B.key
WHERE  [some condition is true]
```

Left Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM   A LEFT JOIN B
ON     A.movie = B.movie
WHERE  …
```

Left Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM    A LEFT JOIN B
ON      A.movie = B.movie
WHERE   …
```

Left Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM   A LEFT JOIN B
ON     A.movie = B.movie
WHERE  …
```

Left Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

Movie	Year	Genre
Titanic	1998	**NULL**
Avatar	2009	Action

```
SELECT …
FROM    A LEFT JOIN B
ON      A.movie = B.movie
WHERE   …
```

Left Join

A =

Movie	Year
Titanic	1997
Avatar	2009

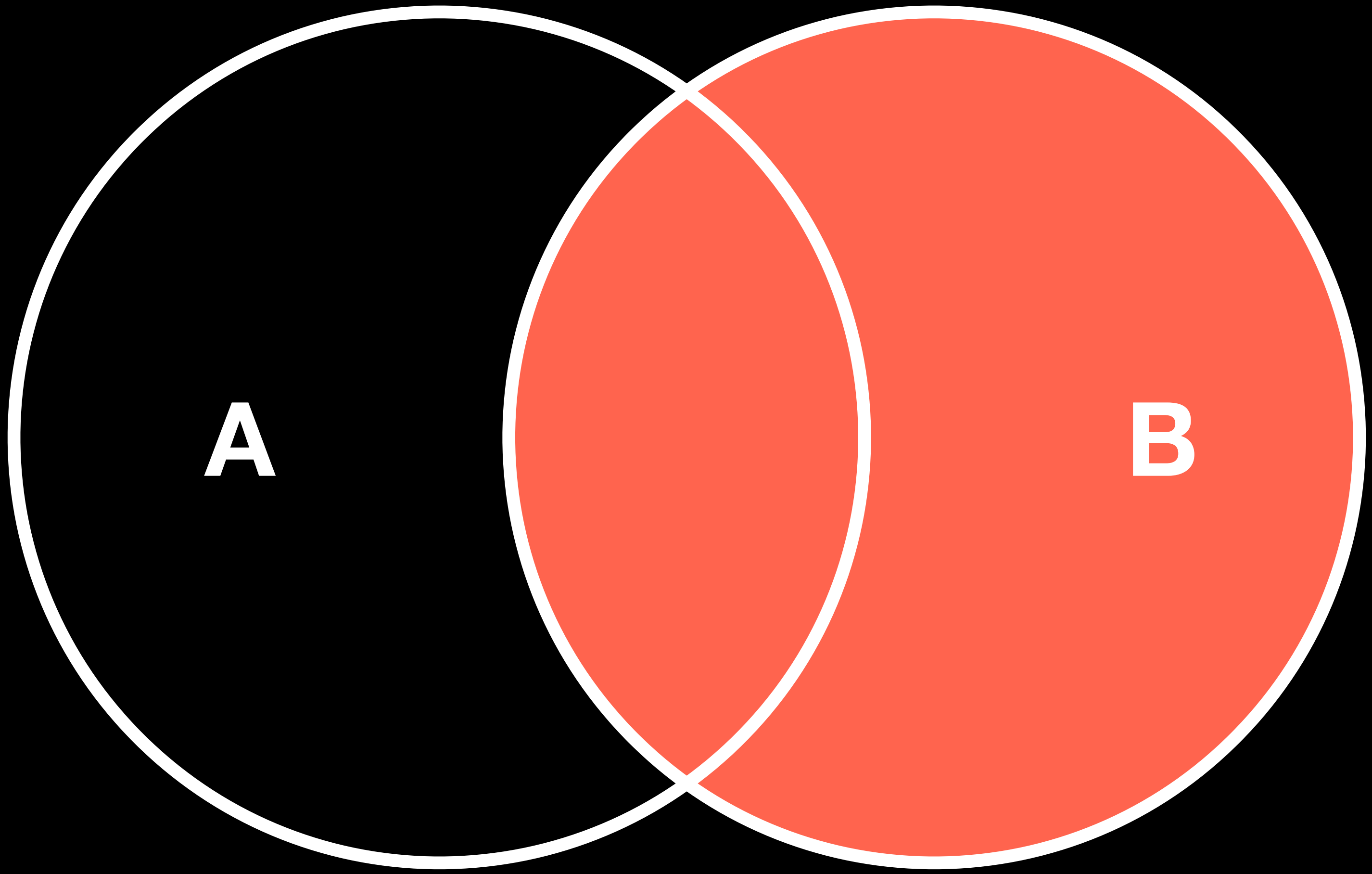
B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

Movie	Year	Genre
Titanic	1998	**NULL**
Avatar	2009	Action

```
SELECT …
FROM   A LEFT JOIN B
ON     A.movie = B.movie
WHERE  …
```

Right Join



```
SELECT [some attribute or column]
FROM   A LEFT JOIN B
ON     A.key = B.key
WHERE  [some condition is true]
```

Right Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM   A RIGHT JOIN B
ON     A.movie = B.movie
WHERE  …
```

Right Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM    A RIGHT JOIN B
ON      A.movie = B.movie
WHERE   …
```

Right Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT ...
FROM    A RIGHT JOIN B
ON      A.movie = B.movie
WHERE   ...
```

Right Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

Movie	Genre	Year
Avatar	Action	2009
Grown Ups	Comedy	**NULL**

```
SELECT …
FROM   A RIGHT JOIN B
ON     A.movie = B.movie
WHERE  …
```

Right Join

A =

Movie	Year
Titanic	1997
Avatar	2009

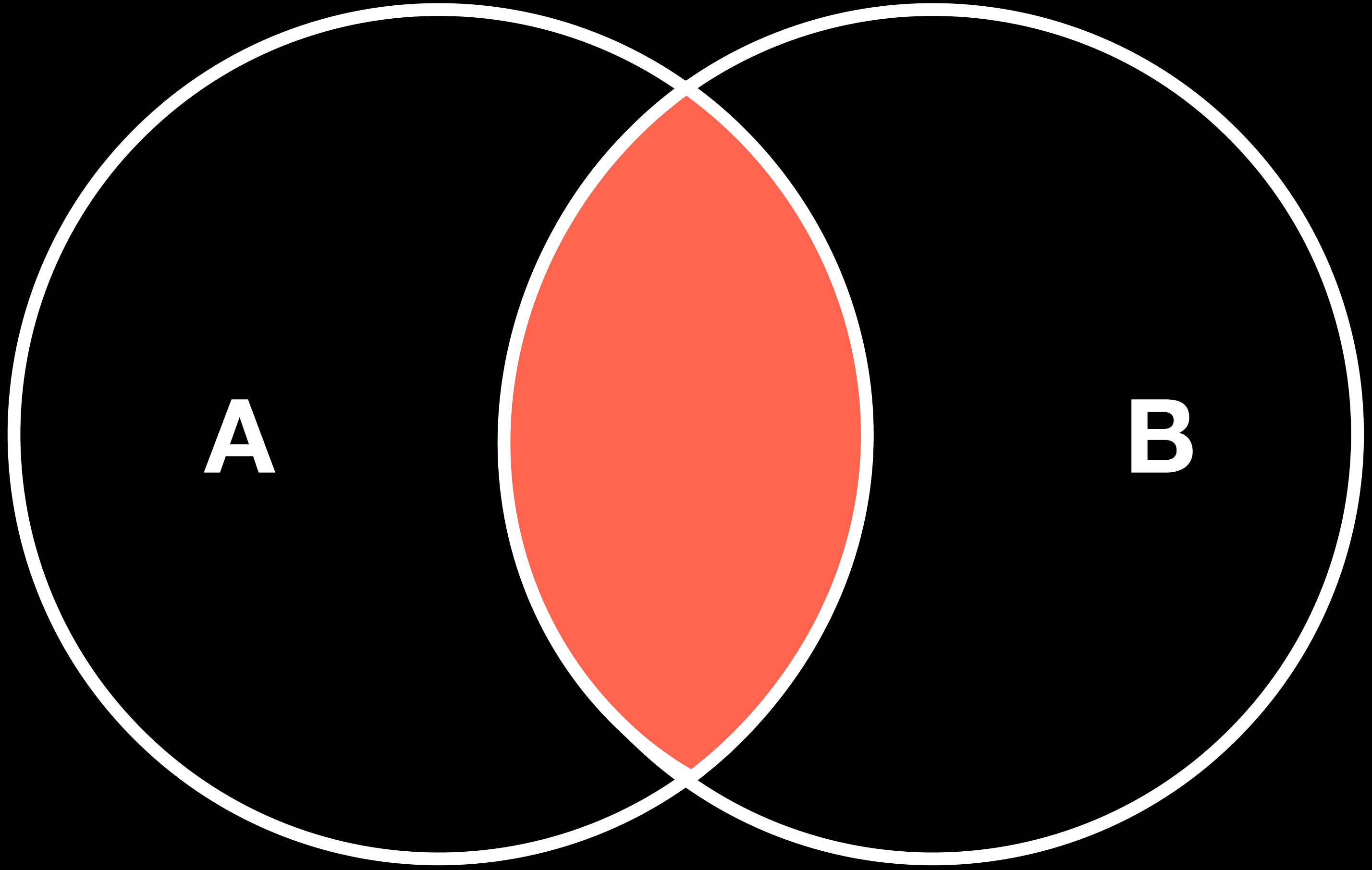
B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

Movie	Genre	Year
Avatar	Action	2009
Grown Ups	Comedy	**NULL**

```
SELECT …
FROM    A RIGHT JOIN B
ON      A.movie = B.movie
WHERE   …
```

Inner Join



```
SELECT [some attribute or column]
FROM   A INNER JOIN B
ON     A.key = B.key
WHERE  [some condition is true]
```

Inner Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM   A INNER JOIN B
ON     A.movie = B.movie
WHERE  …
```

Inner Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM    A INNER JOIN B
ON      A.movie = B.movie
WHERE   …
```

Inner Join

A =

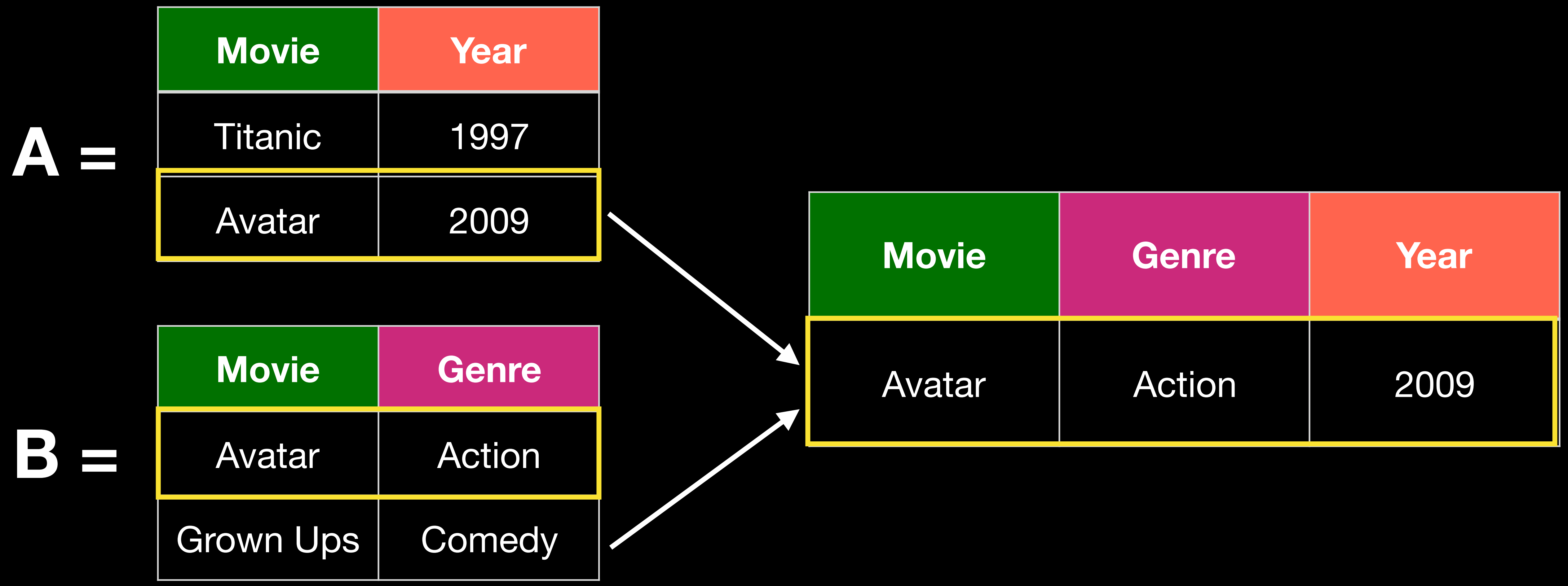
Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT ...
FROM    A INNER JOIN B
ON      A.movie = B.movie
WHERE   ...
```

Inner Join



A =

Movie	Year
Titanic	1997
Avatar	2009

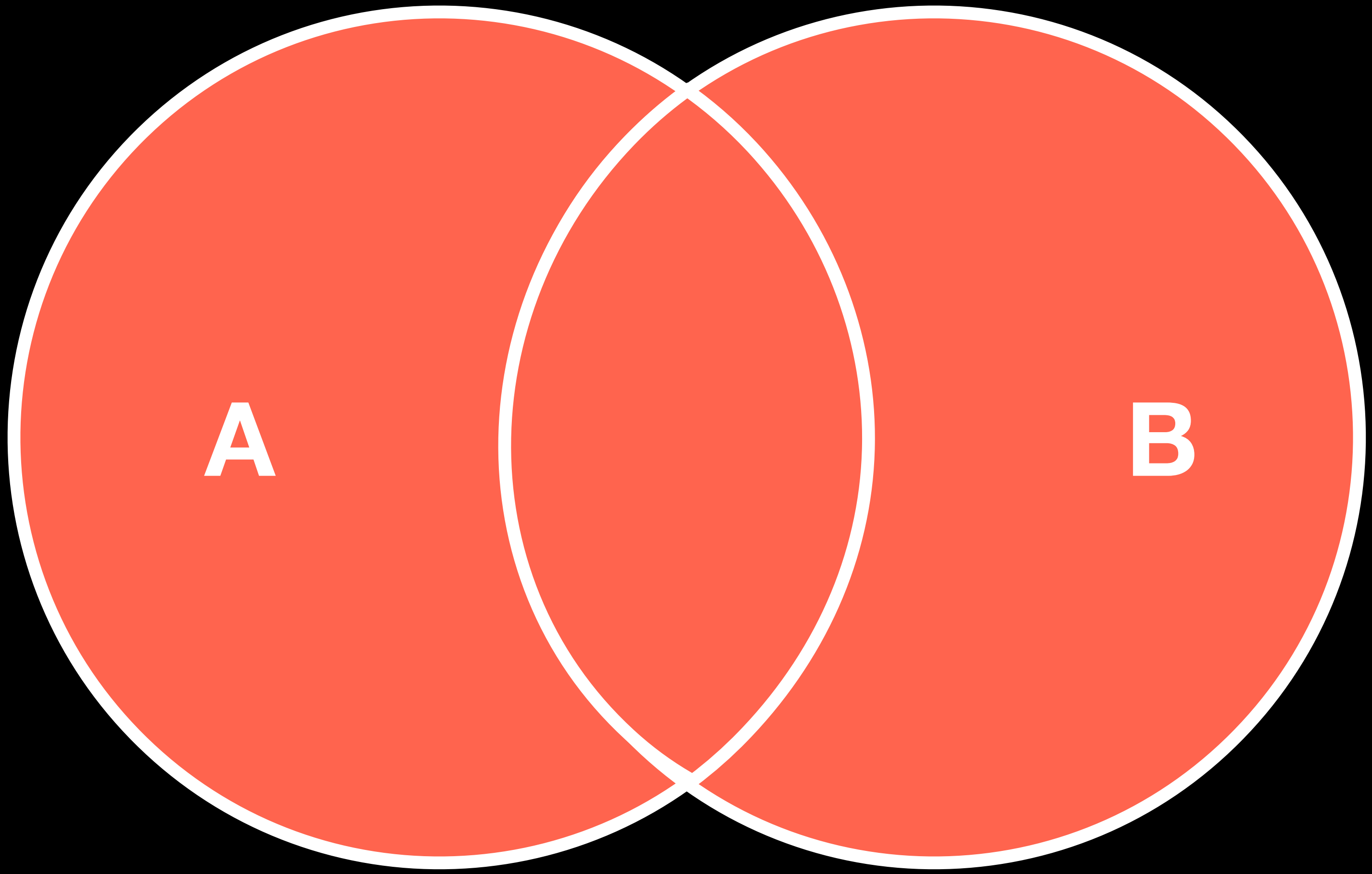
B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

Movie	Genre	Year
Avatar	Action	2009

```
SELECT …
FROM   A INNER JOIN B
ON     A.movie = B.movie
WHERE  …
```

Full Outer Join



```
SELECT [some attribute or column]
FROM   A FULL OUTER JOIN B
ON     A.key = B.key
WHERE  [some condition is true]
```

Full Outer Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM   A FULL OUTER JOIN B
ON     A.movie = B.movie
WHERE  …
```

Full Outer Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM    A FULL OUTER JOIN B
ON      A.movie = B.movie
WHERE   …
```

Full Outer Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

```
SELECT …
FROM    A FULL OUTER JOIN B
ON      A.movie = B.movie
WHERE   …
```

Full Outer Join

A =

Movie	Year
Titanic	1997
Avatar	2009

B =

Movie	Genre
Avatar	Action
Grown Ups	Comedy

Movie	Genre	Year
Titanic	1997	**NULL**
Avatar	2009	Action
Grown Ups	**NULL**	Comedy

```
SELECT ...
FROM   A FULL OUTER JOIN B
ON     A.movie = B.movie
WHERE  ...
```