INTRODUCTION TO

PYTHON

COOL THINGS ABOUT PYTHON

- In 2016, one of the top ten requested programming skills
- Intuitive, with a flatter learning curve than most programming languages
- Open Source

THE ZEN OF PYTHON

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

...

There should be one – and preferably only one – obvious way to do it.

INTEGERS

- Integer: a whole negative or positive number that can be represented without a fraction
- You can apply mathematical operations to integers
  - 1+2 = 3
  - 1-2 = -1
- Use them for:
  - Rounded mathematical operations
  - Indexing into lists
- ***Floats*:** like integers, but with fractional components

STRINGS

- String: A sequence of characters, often enclosed in single or double quotes
  - ‘Hello’ | “Hello”
- Adding strings means concatenation:
  - “Hello” + “World” = “HelloWorld”
  - “Hello” + “ “ + “World” + “!” = “Hello World!”

BOOLEANS

- Boolean Variable: Either True (1) or False (0)
- Recall the logical operations:
  - True AND True -> Returns True
  - True AND False -> Returns False
  - True OR False -> Returns True
  - False OR False -> Returns False
- Adding two boolean variables acts like OR

LISTS

- A list is a collection of objects of any data type
- List of strings: [“Hello”, “ “, “World”, “!”]
- List of integers: [1, 2, 3, -1]
- List of lists: [[“Hello”, “ “, “World”, “!”], [1, 2, 3, -1]]
- Adding two lists?

LISTS

- A list is a collection of objects of any data type
- List of strings: [“Hello”, “ “, “World”, “!”]
- List of integers: [1, 2, 3, -1]
- List of lists: [[“Hello”, “ “, “World”, “!”], [1, 2, 3, -1]]
- Adding two lists puts all the elements in one list, so adding the first two strings in this list would give you [“Hello”, “ “, “World”, “!”, 1, 2, 3, -1]

INITIALIZING DATA TYPES IN PYTHON

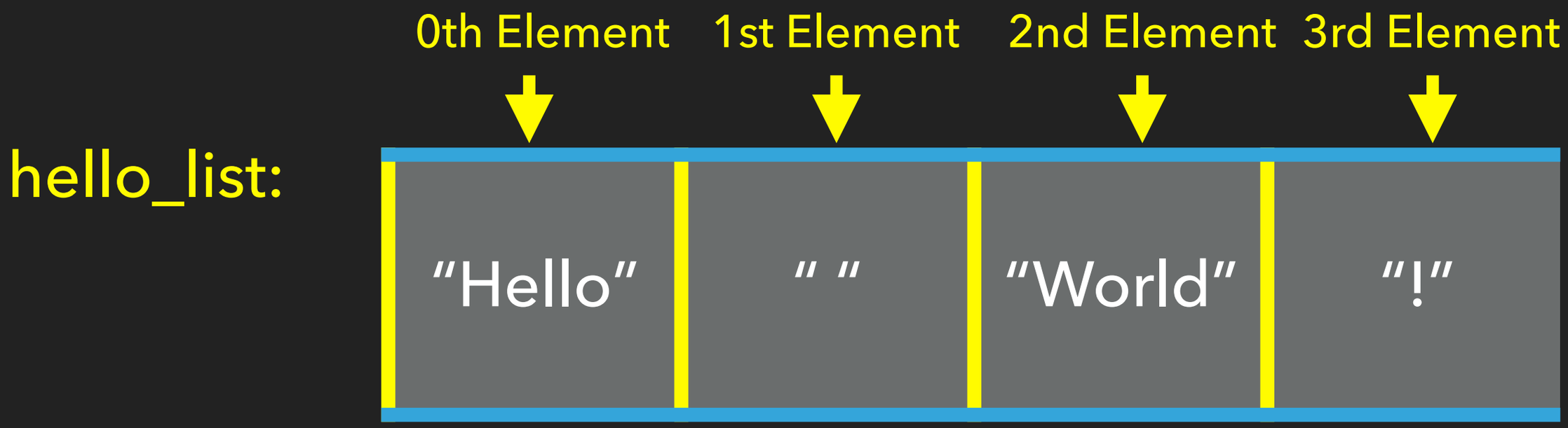
- String: Enclosed in quotes
  - my_string = “123”
- Integer:
  - my_int = 123
- Boolean: evaluates to True (1) or False (0).
  - my_boolean = True
- List: A collection of objects of any data type
  - my_list = [1, 2, 3]

A QUICK DETOUR INTO PROGRAMMER COUNTING

- Starts at 0
- So we look at this list:
  - hello_list = [“Hello”, “ “, “World”, “!”]

A QUICK DETOUR INTO PROGRAMMER COUNTING

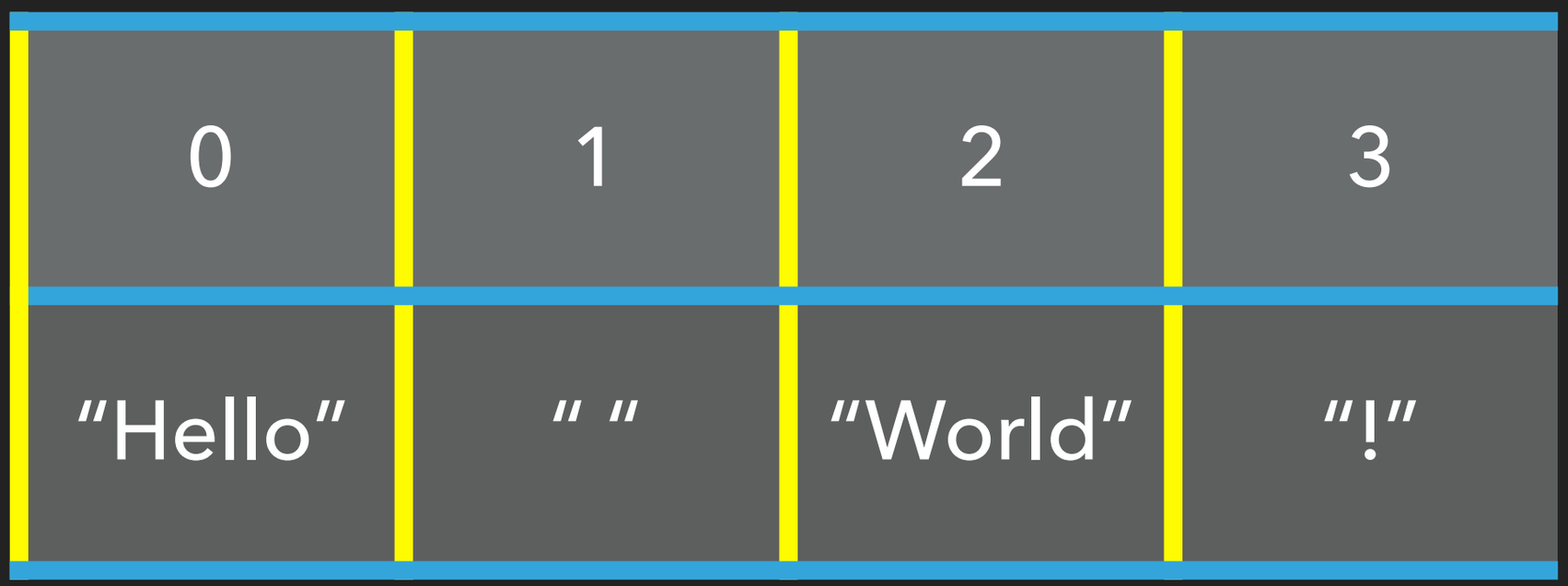
- Starts at 0
- So we look at this list:
  - hello_list = [“Hello”, “ “, “World”, “!”]
- Like this:



A QUICK DETOUR INTO PROGRAMMER COUNTING

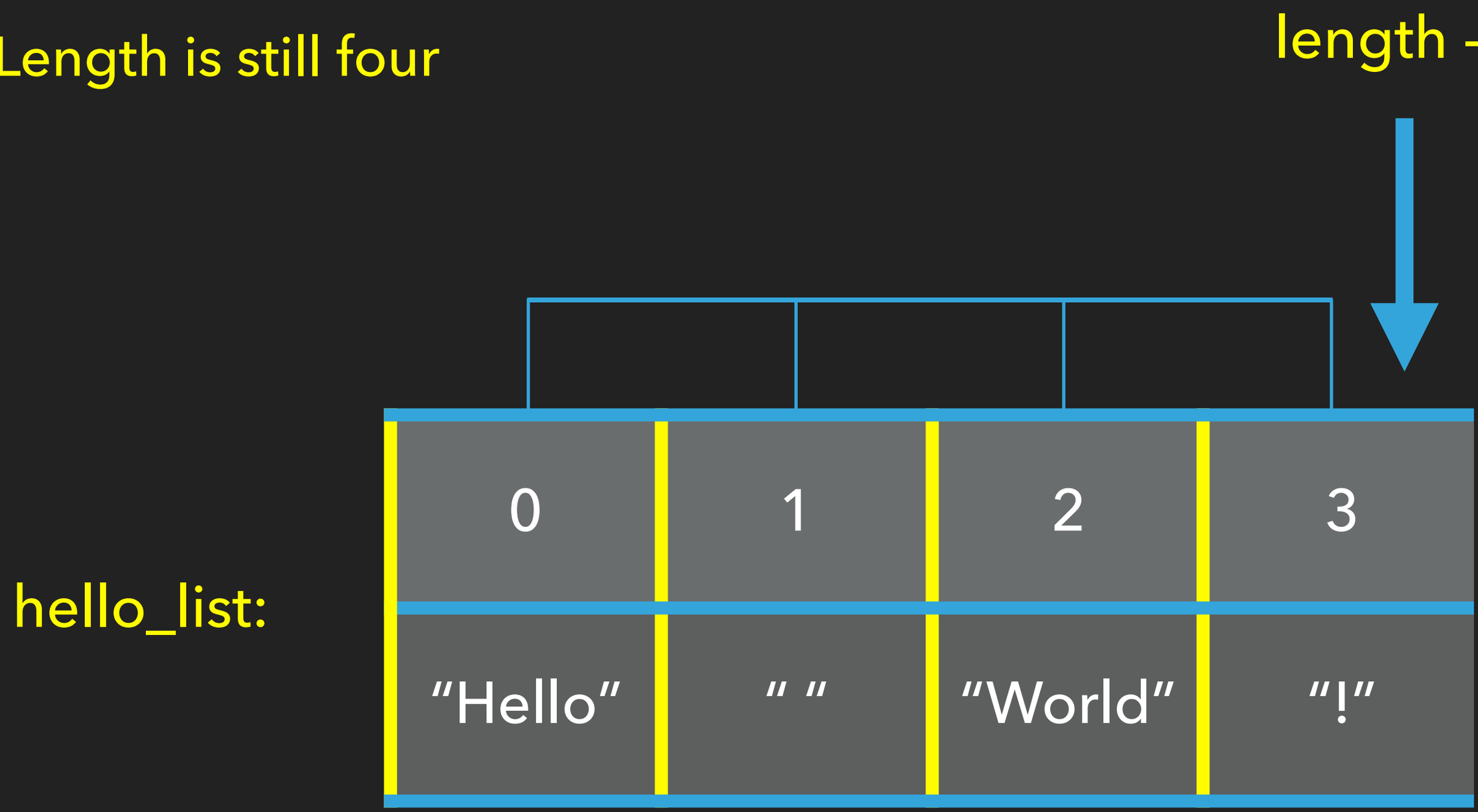
- Starts at 0
- So we look at this list:
  - hello_list = [“Hello”, “ “, “World”, “!”]
- Like this:

hello_list:



BUT SOME THINGS STAY THE SAME…

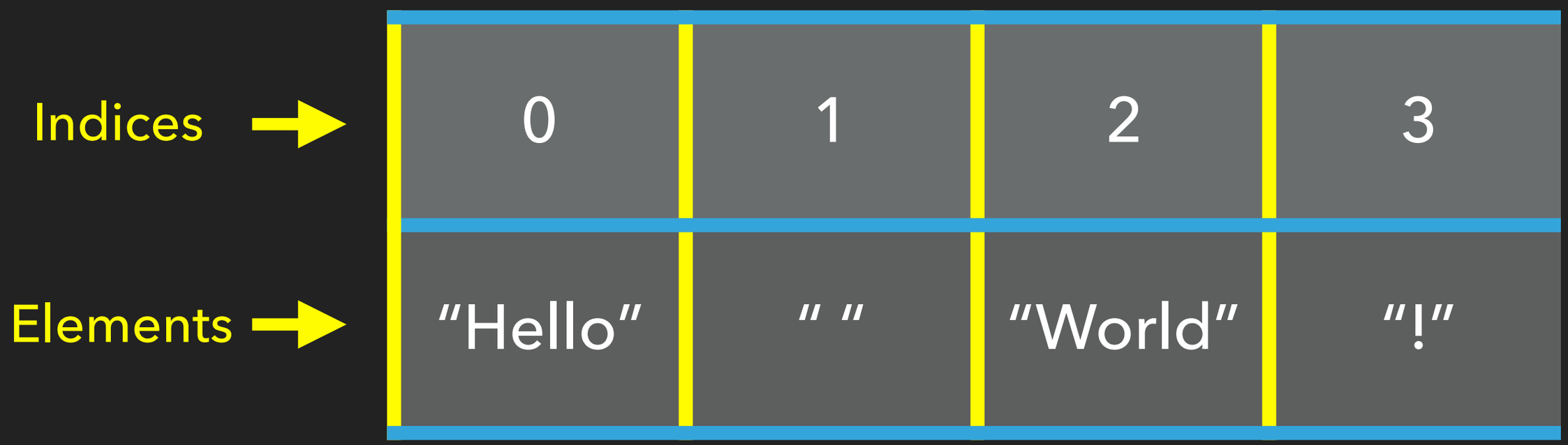
- Length is still four



WORKING WITH LISTS

- To get the values of the elements in the list, you refer to them by their position in the list.
- The positions are called ***indices***,

and using them is called ***indexing*** into the list



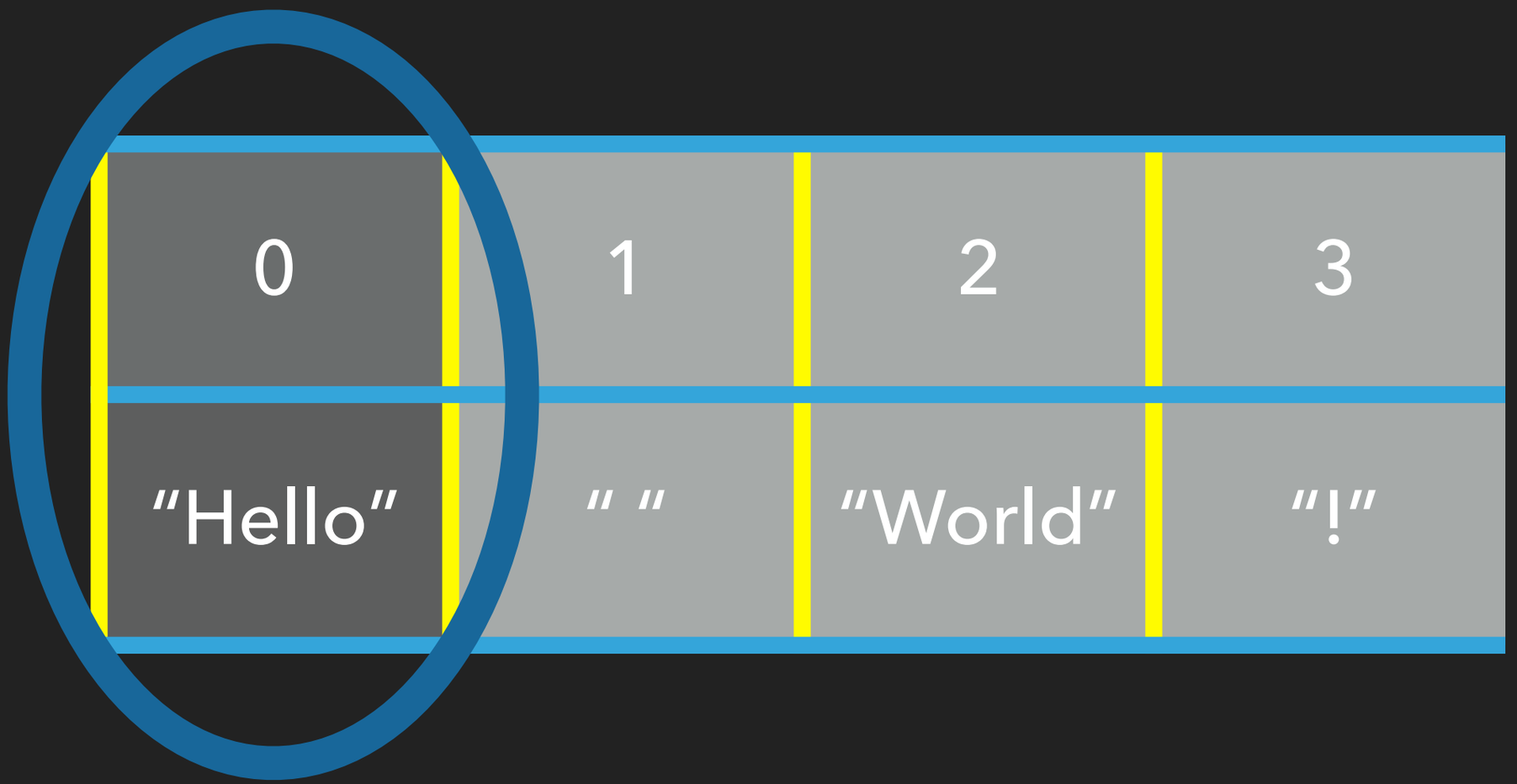
WORKING WITH LISTS

- So if we want to access “Hello”, we would index the list with

0	1	2	3
“Hello”	“ ”	“World”	“!”

WORKING WITH LISTS

- So if we want to access “Hello”, we would index the list with **hello_list[0]**

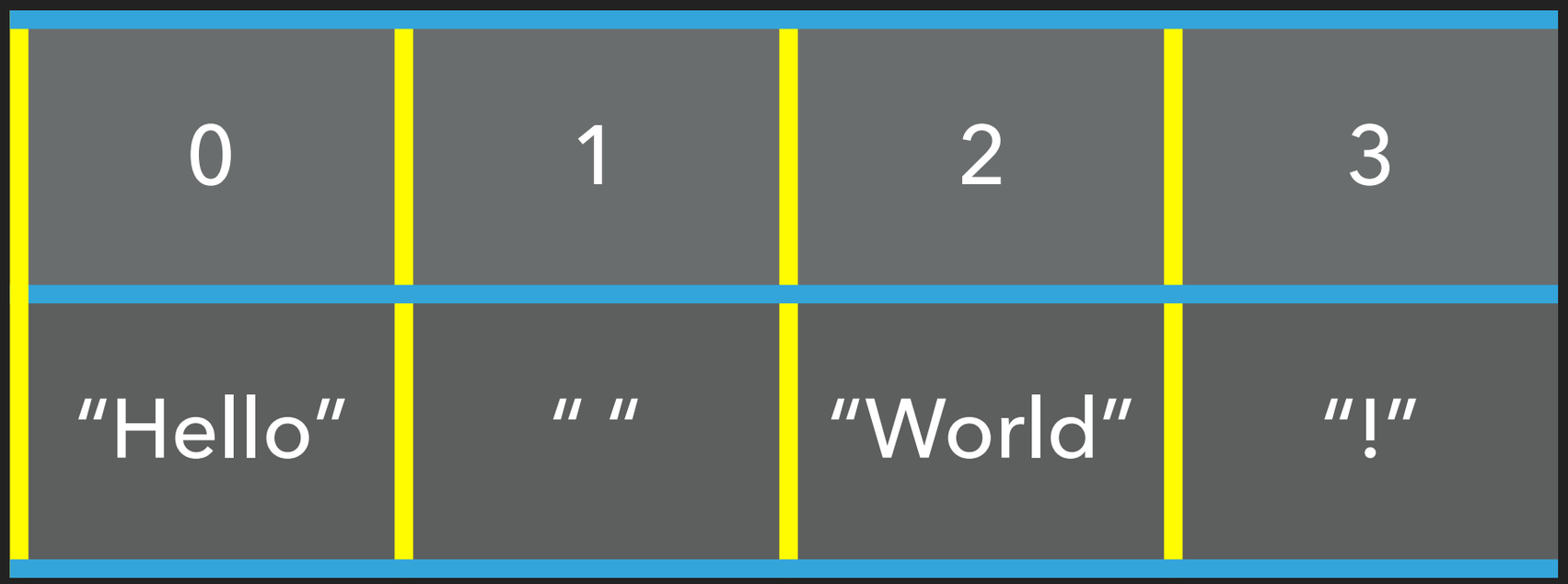


WORKING WITH LISTS

- If we want to access the first three elements:

  “Hello” and “ ” and “World”

  We would index with

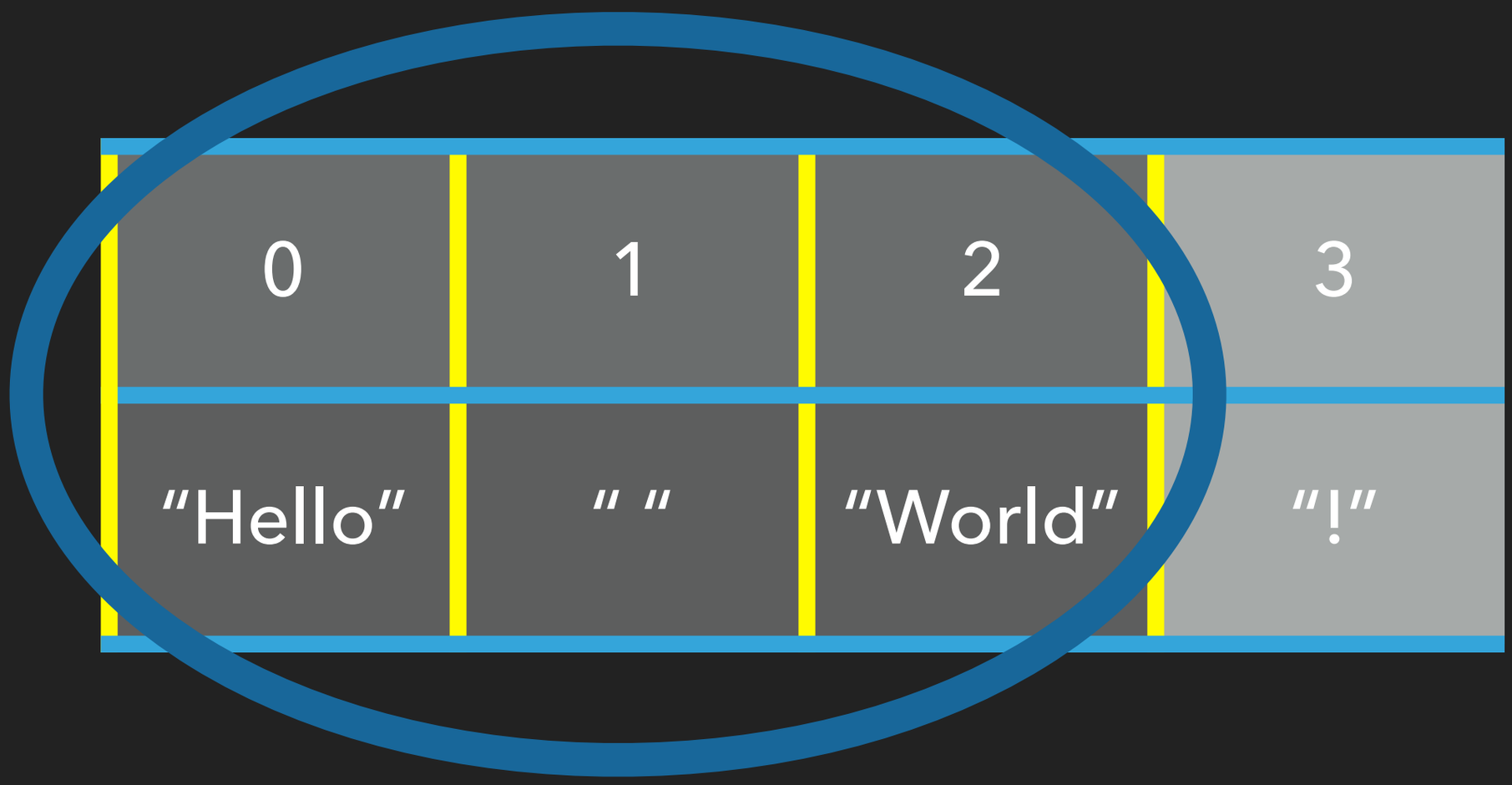


WORKING WITH LISTS

- If we want to access the first three elements:

  “Hello” and “ ” and “World”

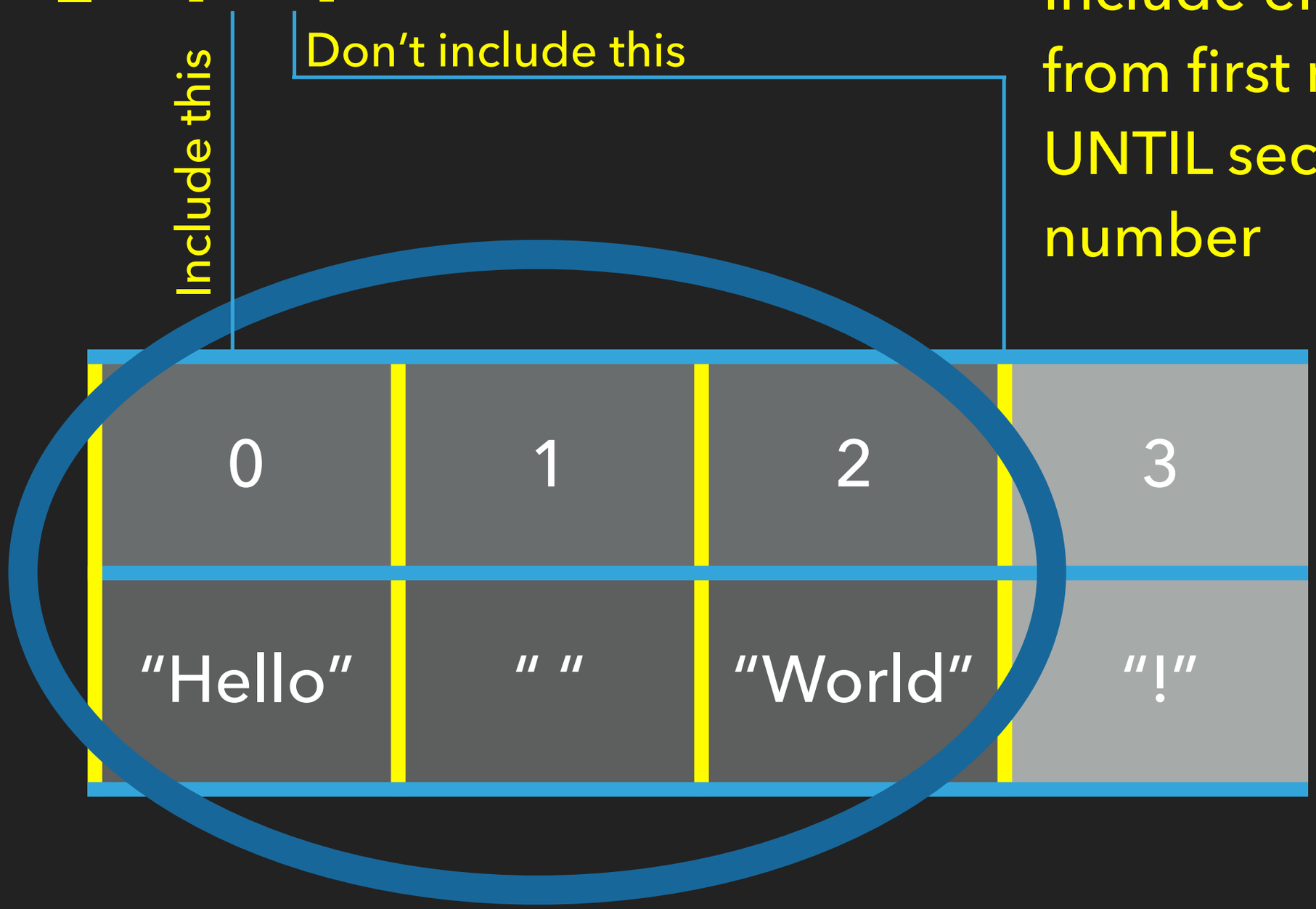
  We would index with **hello_list[0:3]**



WORKING WITH LISTS

hello_list[0:3]

Include elements from first number UP UNTIL second number



FOR LOOPS

- Two types of For Loops:
  - for number from 0 to 5:
    - print(number)

For Each Loop

- for number in a list of numbers:
  - print(number)

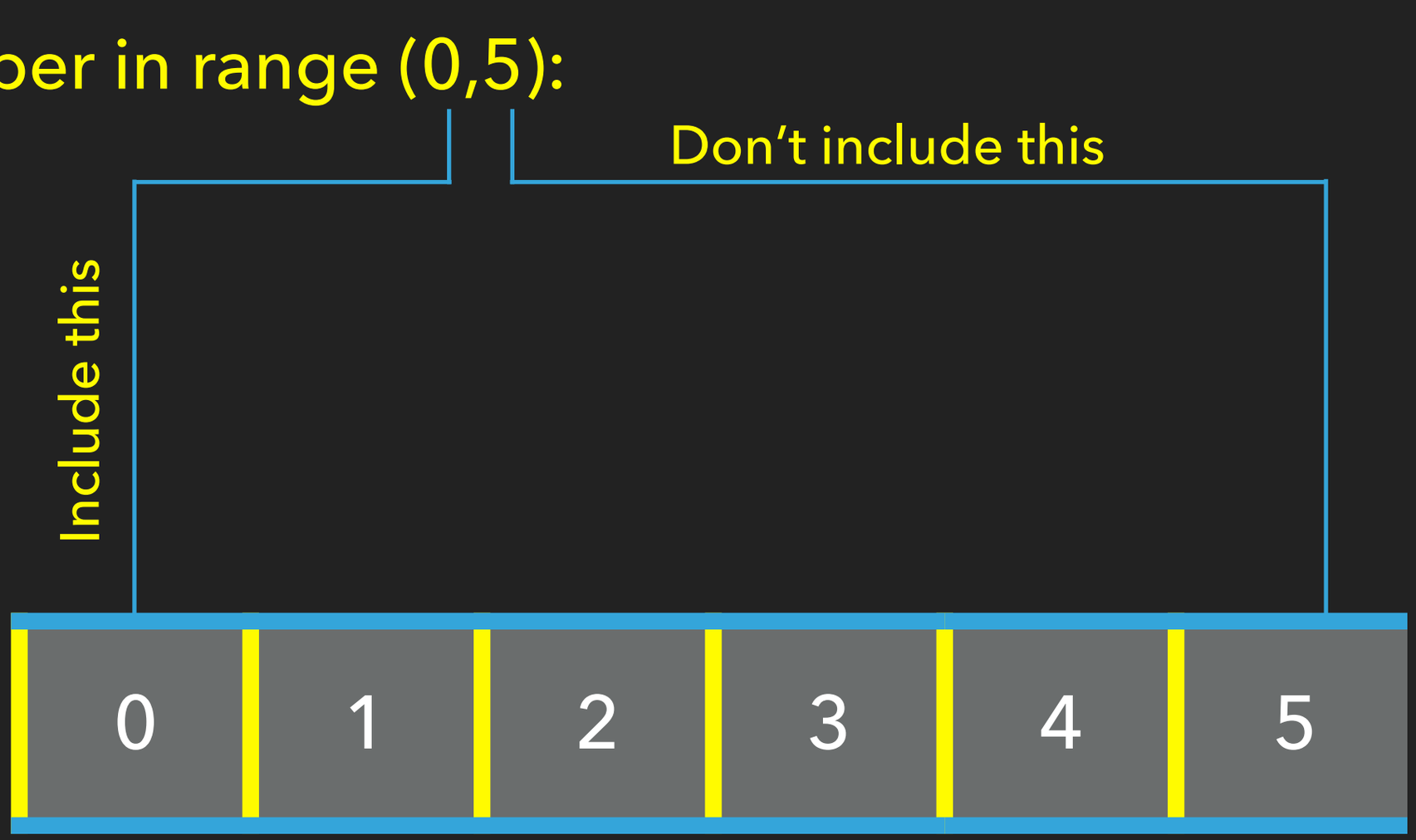
FOR LOOPS IN PYTHON

- In python, for loops and for each loops look basically the same:
- for number in range (0,5):

print(number)

FOR LOOPS IN PYTHON

- In python, for loops and for each loops look basically the same:
- for number in range (0,5):



FOR LOOPS IN PYTHON

- In python, for loops and for each loops look basically the same:
- for number in range (0,5):

    print(number)
- for number in [0,1,2,3,4]:

    print(number)

WHILE LOOPS IN PYTHON

- A while loop executes code *while* some expression evaluates to True
- while <expression is True>:
  - # Execute some code
  - # Do something that will change the expression at some point

A # denotes a comment in python.

Lines that begin with # will not be executed.

WHILE LOOPS IN PYTHON

```
base = 5

num = 1

power = 3

n = 1

while n <= power:

    num = num * base

    n = n + 1
```

FOR LOOP WITH LIST EXAMPLE

- Let's do an example where we combine all of the elements in our hello_list to create the full sentence, "Hello World!"

- First step: initialize an empty string variable that will contain the sentence

FOR LOOP WITH LIST EXAMPLE

```
new_string = ""

for element in hello_list:
        new_string = new_string + element

print(new_string)
```

WHILE LOOP WITH LIST EXAMPLE

```
new_string = ""
my_index = 0
while my_index < len(hello_list):
    new_string = new_string +
                        hello_list[my_index]
    my_index = my_index + 1
print(new_string)
```

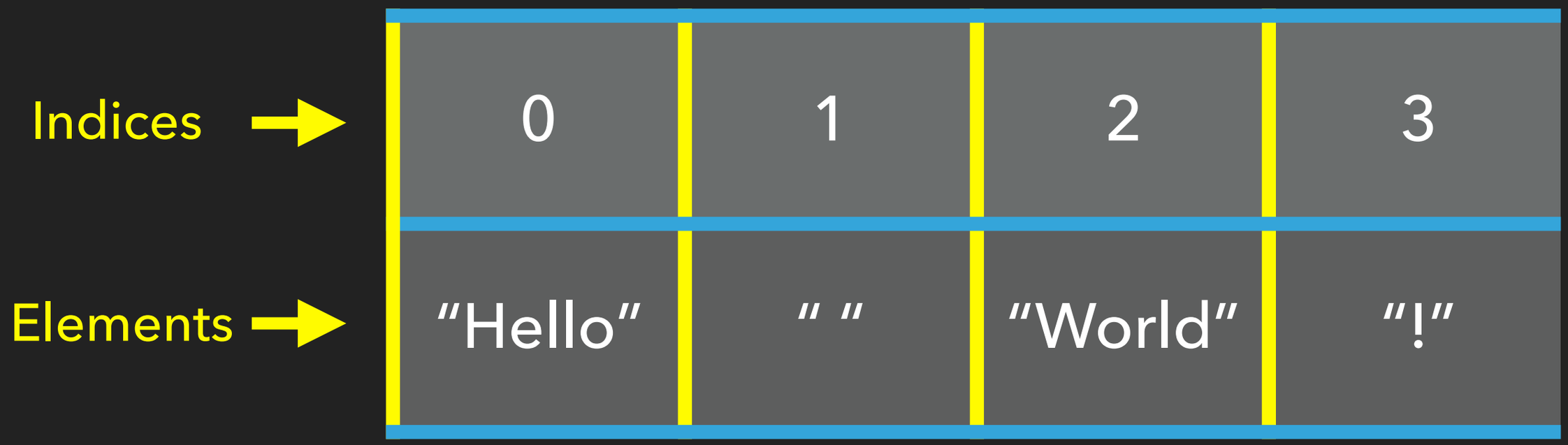
REMEMBER OUR LIST EXAMPLE

- To get the values of the elements in the list, you refer to them by …

0	1	2	3
“Hello”	“ ”	“World”	“!”

REMEMBER OUR LIST EXAMPLE

- To get the values of the elements in the list, you refer to them by their position in the list.



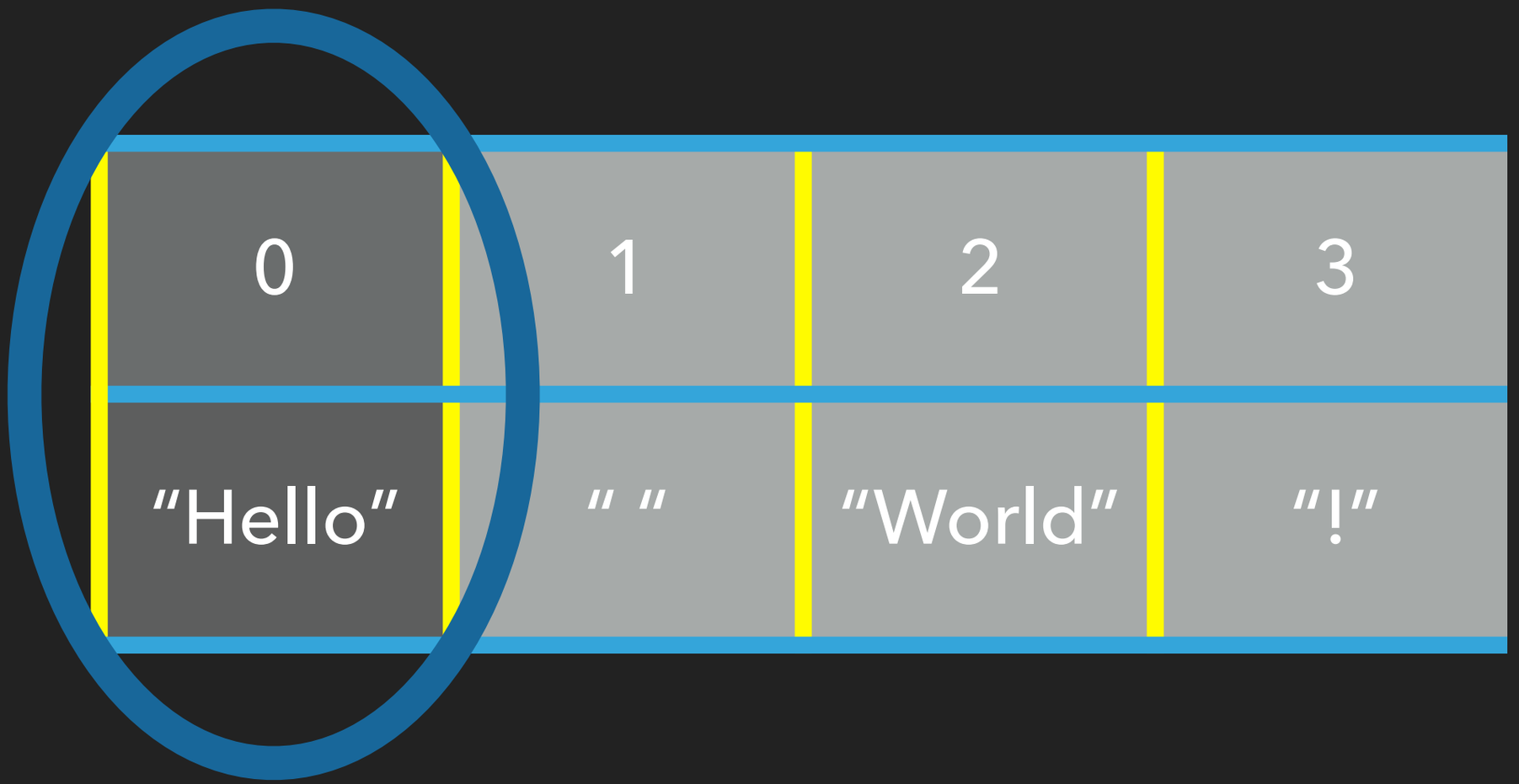
REMEMBER OUR LIST EXAMPLE

- If we want to access “Hello”, we would index the hello_list with

0	1	2	3
“Hello”	“ ”	“World”	“!”

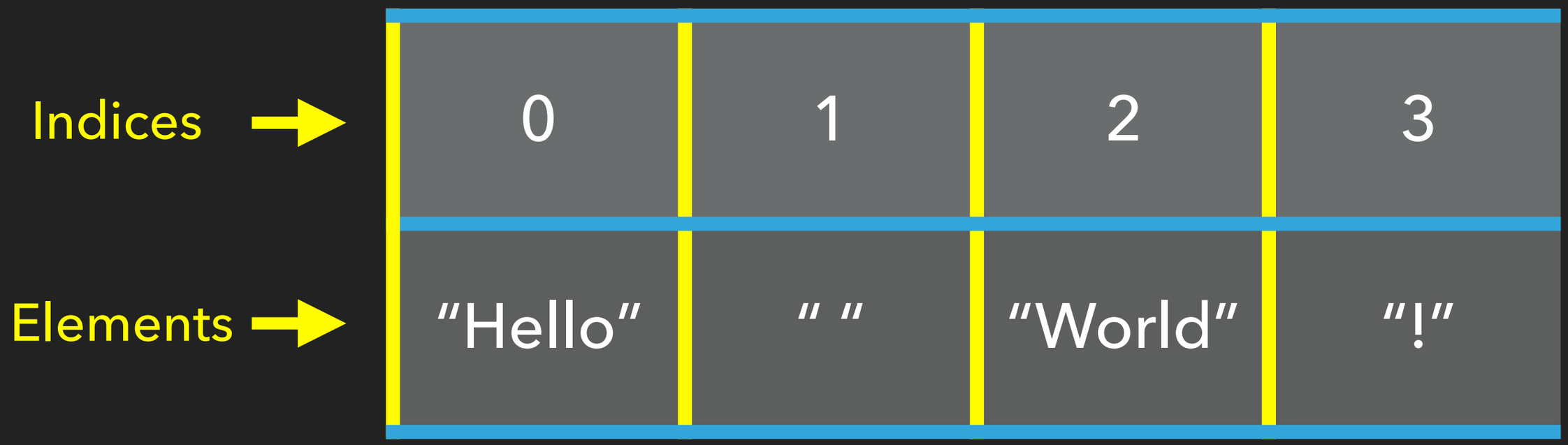
REMEMBER OUR LIST EXAMPLE

- If we want to access “Hello”, we would index the hello_list with **hello_list[0]**



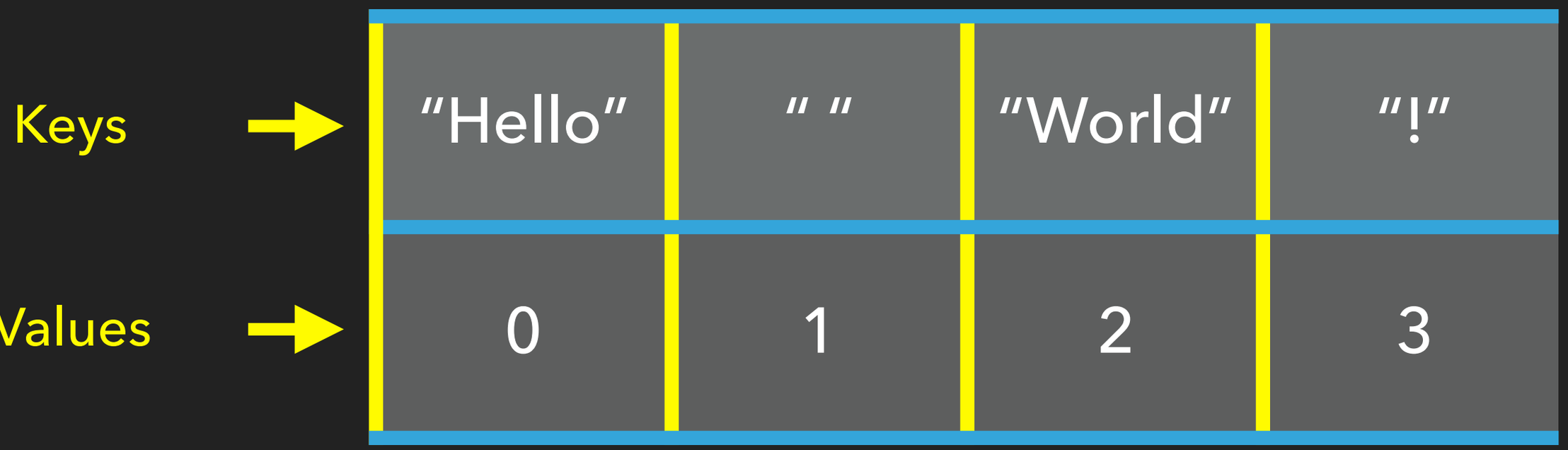
BACK TO DATA TYPES

- But what if we knew the words, and we wanted to ask what the corresponding number is?



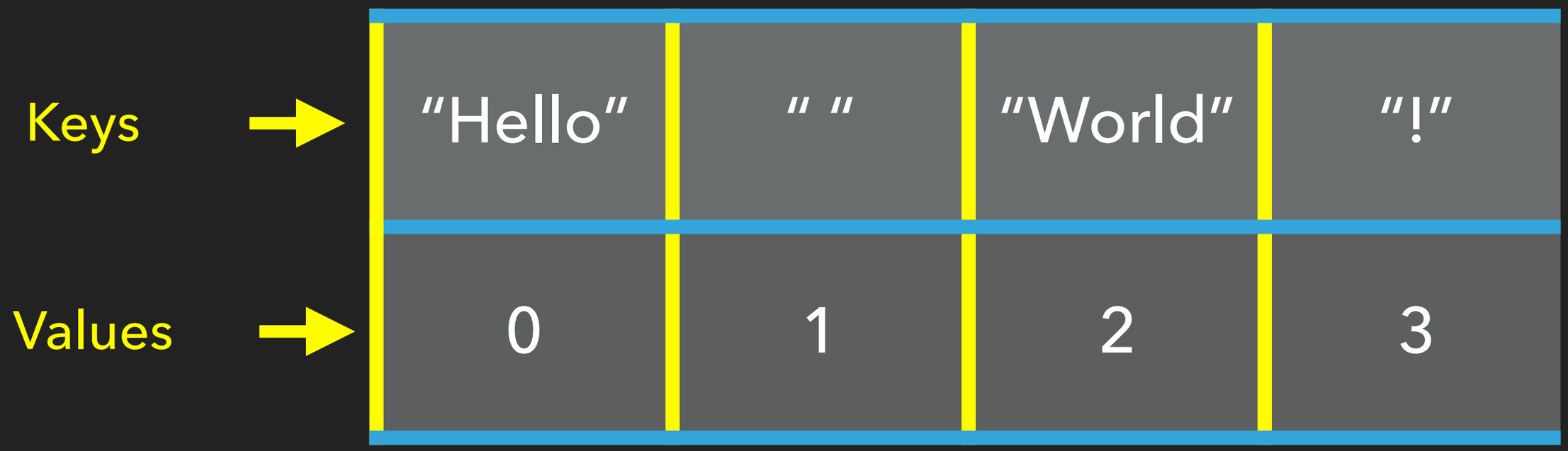
INTRODUCTION TO DICTIONARIES

hello_dictionary:



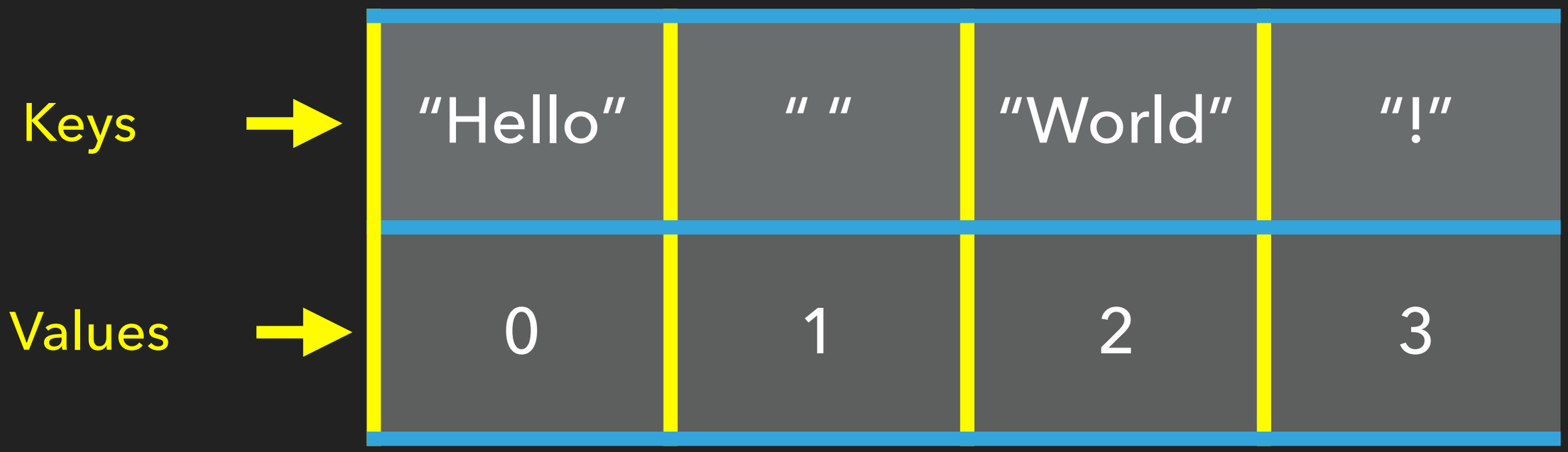
WORKING WITH DICTIONARIES

- To get the values of the elements in the list, you refer to them by their *key*.



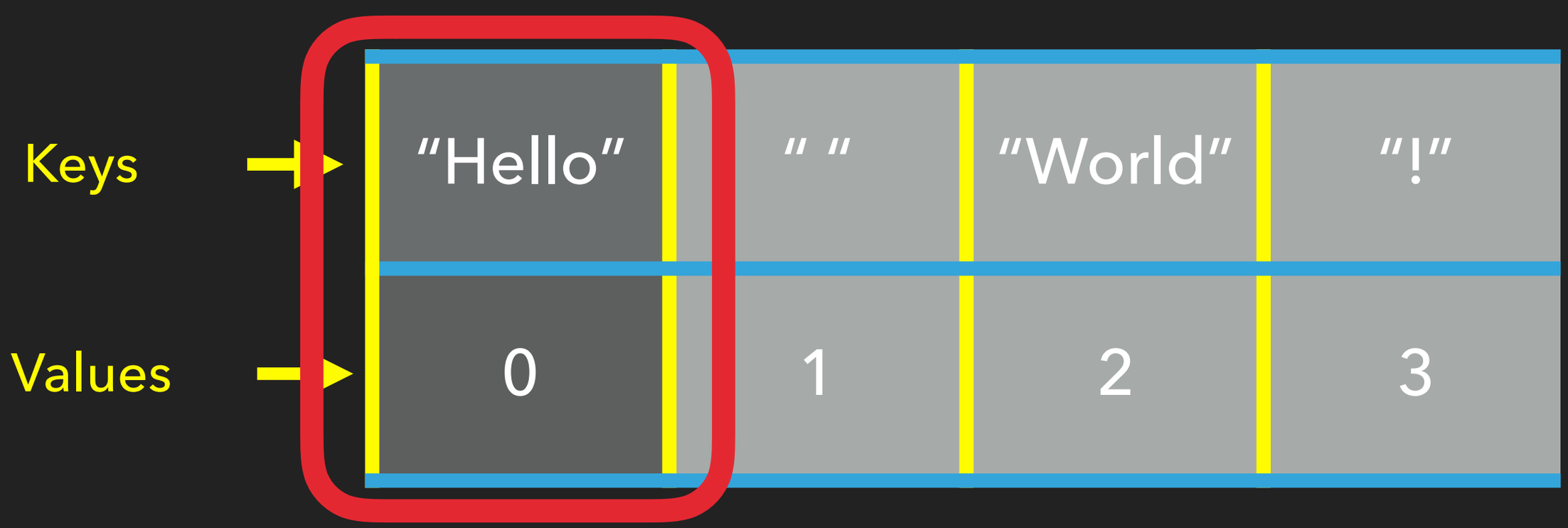
WORKING WITH DICTIONARIES

- So we would access the value of “Hello” with



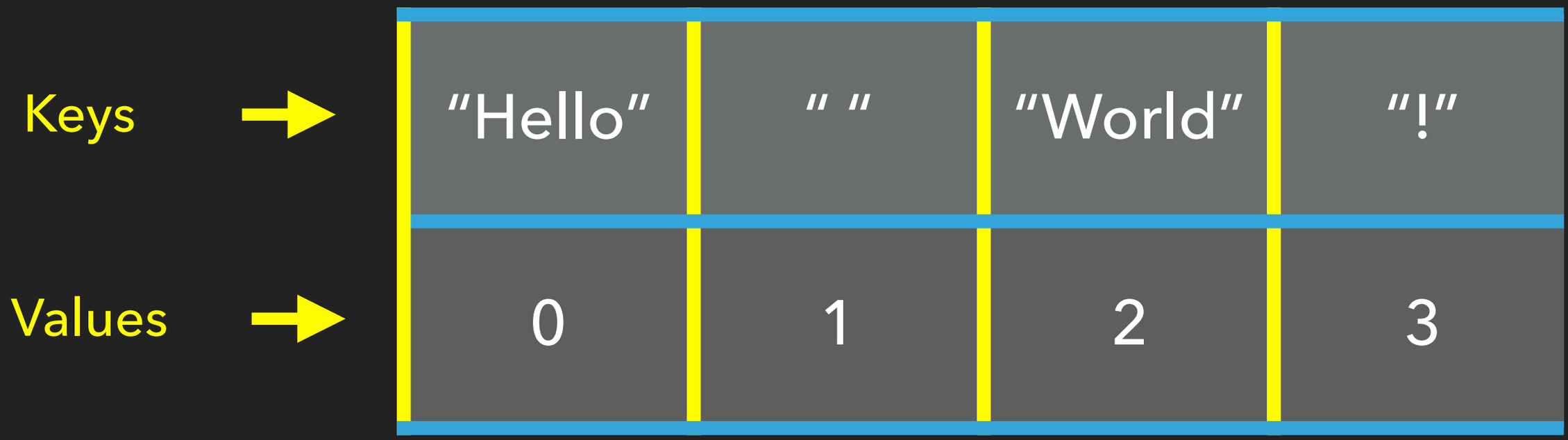
WORKING WITH DICTIONARIES

- So we would access the value of “Hello” with **hello_dictionary[“Hello”]**



WORKING WITH DICTIONARIES

- So can we access blocks of the dictionary like we did lists?



WORKING WITH DICTIONARIES

- So can we access blocks of the dictionary like we did lists? **No**

Unordered

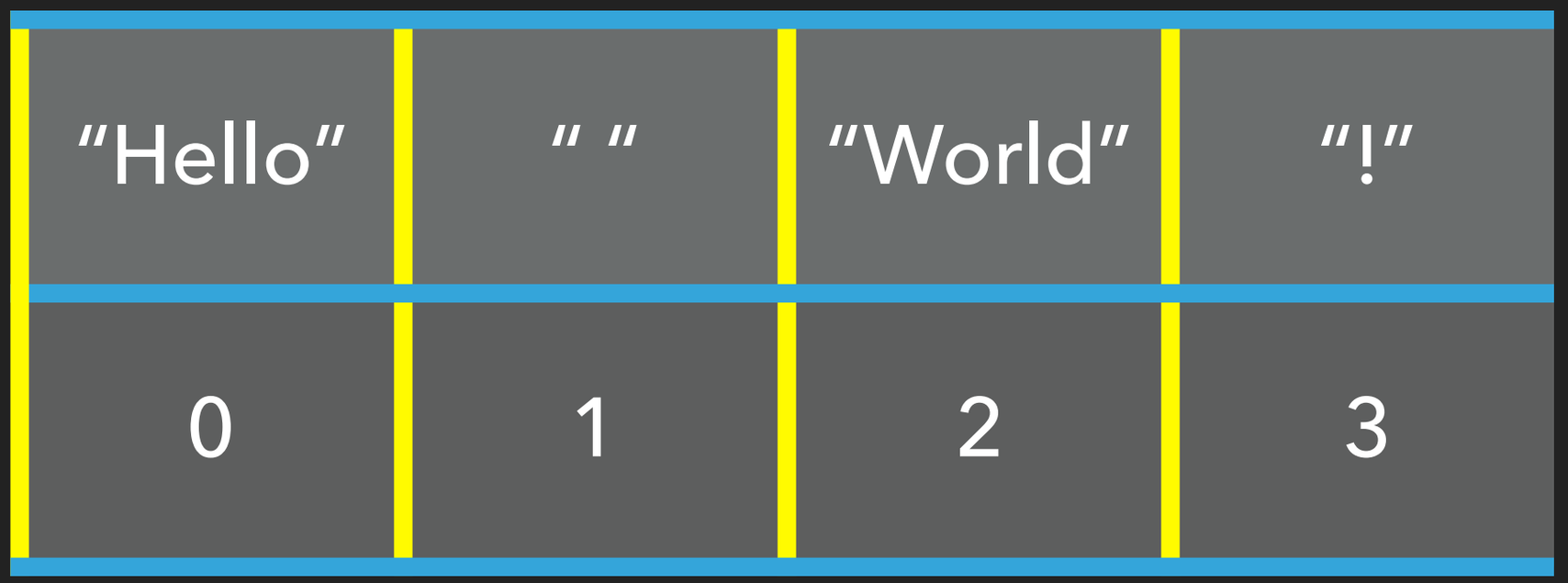
WORKING WITH DICTIONARIES

- Initializing Dictionaries in Python:
  - my_dictionary =

    {key: value, key: value, key: value, key: value}

WORKING WITH DICTIONARIES

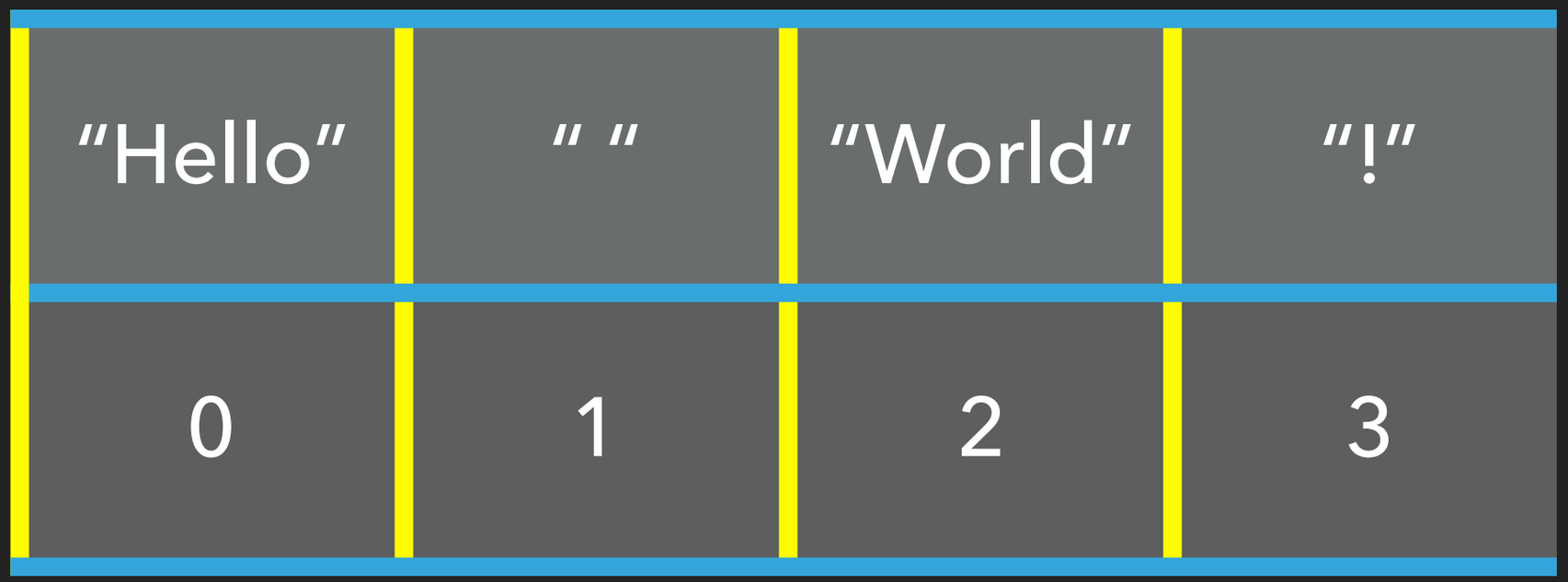
- Initializing Dictionaries in Python:
  - hello_dictionary =



WORKING WITH DICTIONARIES

- Initializing Dictionaries in Python:
  - hello_dictionary =

{ “Hello”: 0, “ “: 1, “World”: 2, “!”: 3 }



ANATOMY OF A FUNCTION

```
def main(argument1, argument2):
    # Code to execute

main(argument1, argument2)
```

Hey Python, this is a function definition!

Name of the function

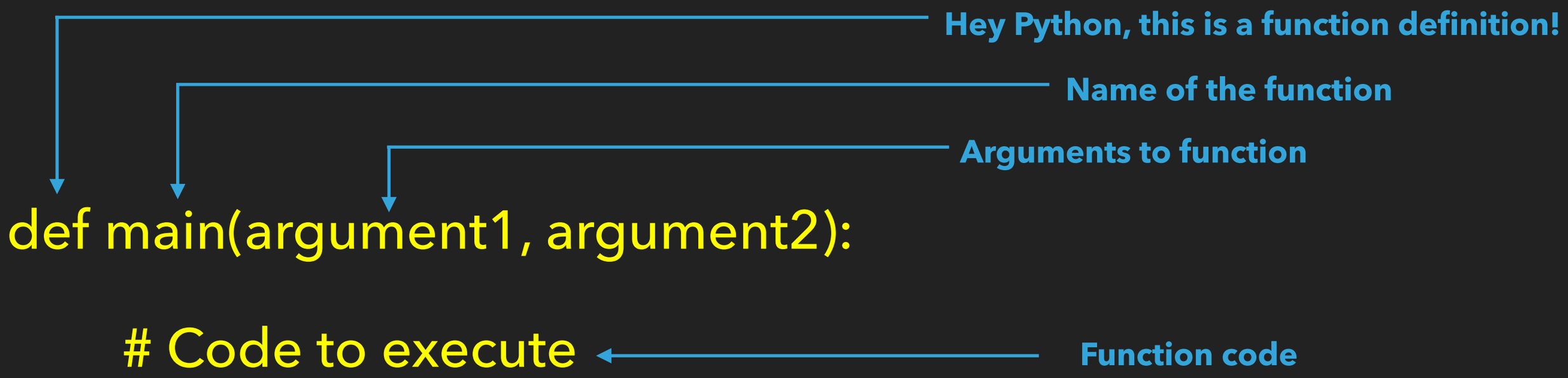
Arguments to function

Function code

Name of the function to call

Arguments to function

ANATOMY OF A FUNCTION



REMEMBER OUR WHILE EXAMPLE

```
base = 5

power = 3

num = 1

n = 1

while n <= power:

    num = num * base

    n = n + 1
```

REMEMBER OUR WHILE EXAMPLE

- But what if we wanted to do this more than once?

```
base = 5

power = 3

num = 1

n = 1

while n <= power:

    num = num * base

    n = n + 1
```

REMEMBER OUR WHILE EXAMPLE

base = 5

power = 3

Changes every time

```
num = 1

n = 1

while n <= power:

    num = num * base

    n = n + 1
```

REMEMBER OUR WHILE EXAMPLE

```
base = 5
power = 3
num = 1
n = 1
while n <= power:
    num = num * base
    n = n + 1
```

Changes every time

Repeats every time

REMEMBER OUR WHILE EXAMPLE

base = 5

power = 3

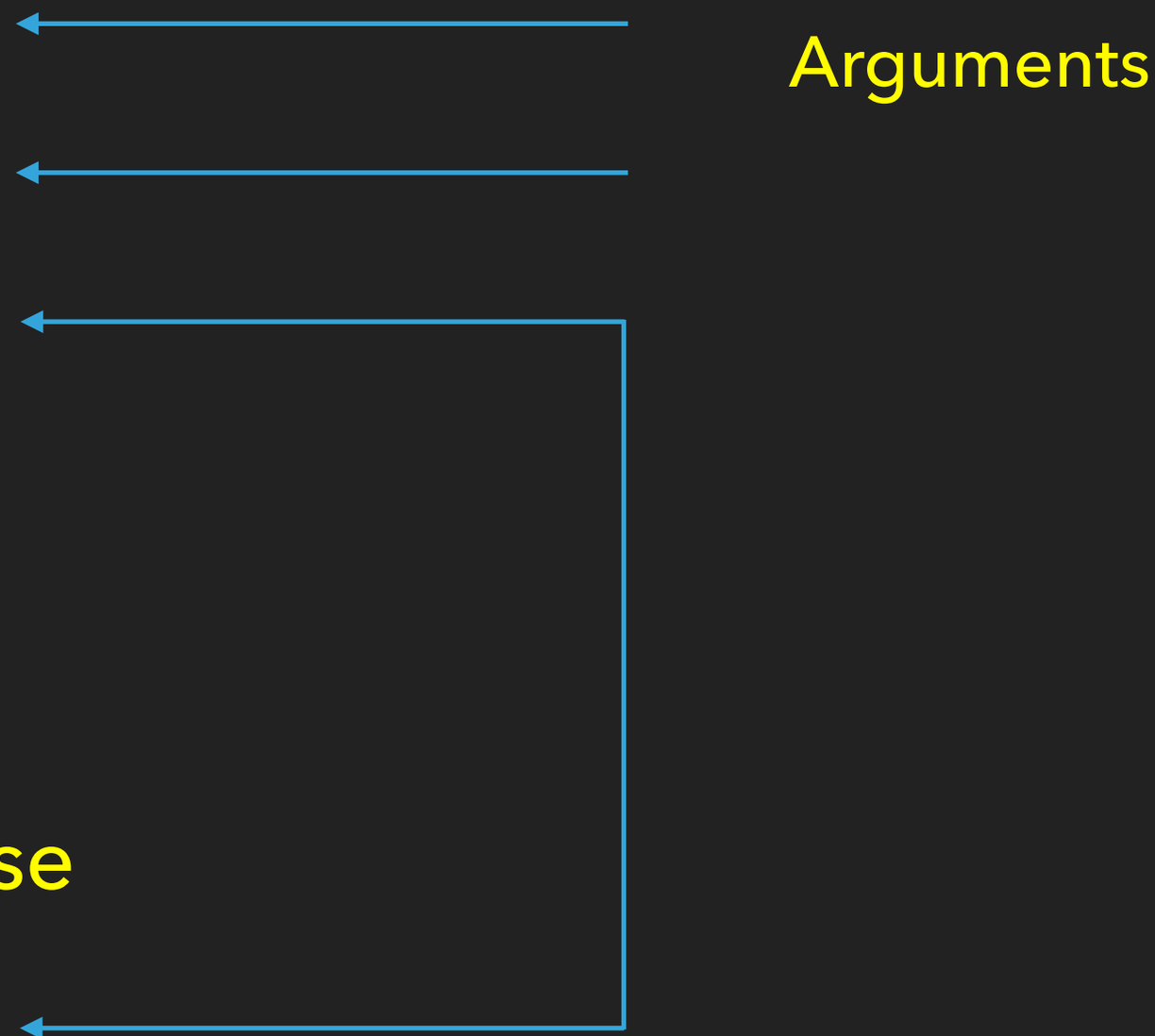
num = 1

n = 1

while n <= power:

    num = num * base

    n = n + 1



REMEMBER OUR WHILE EXAMPLE

```
base = 5
power = 3
num = 1
n = 1
while n <= power:
    num = num * base
    n = n + 1
```

Arguments

Code to execute