INTRO TO PYTHON

PROGRAMMING

DEFINING YOUR OWN

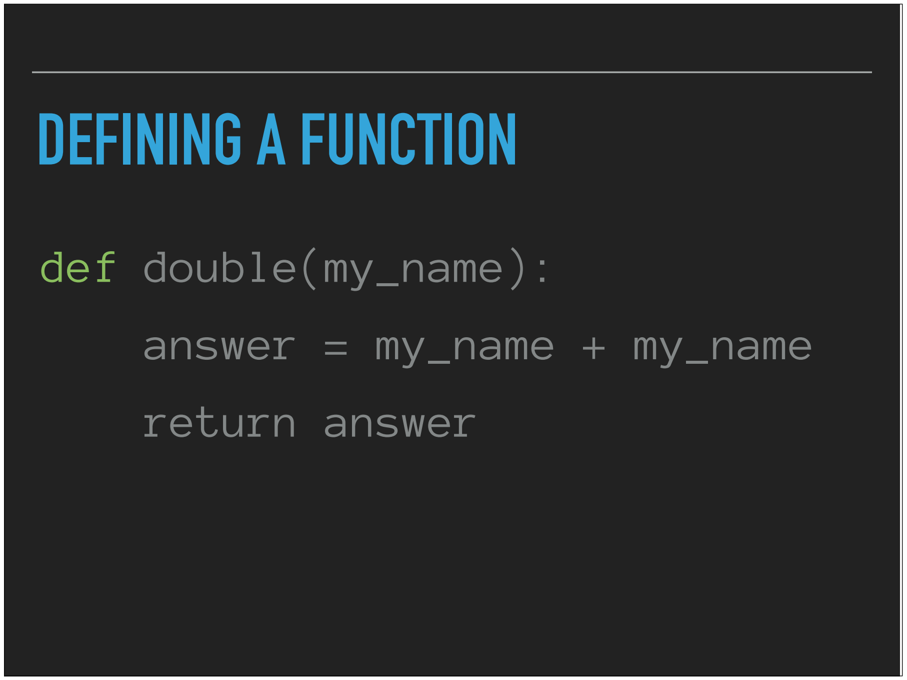
FUNCTIONS

DEFINING A FUNCTION

- Reusable chunk of code

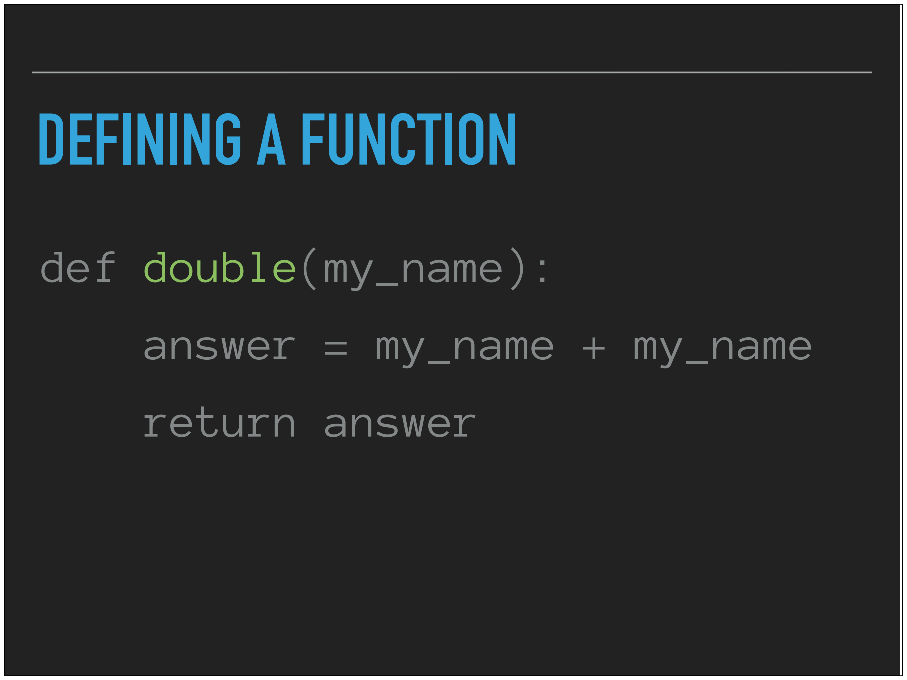
DEFINING A FUNCTION

- Reusable chunk of code
- Build action from basic functions



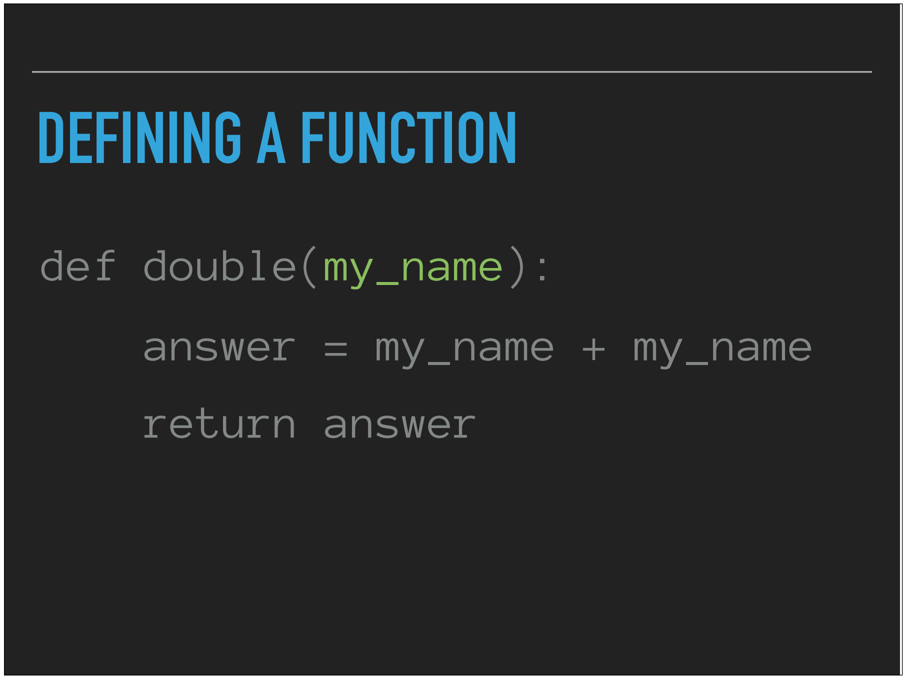
DEFINING A FUNCTION

```
def double(my_name):
    answer = my_name + my_name
    return answer
```



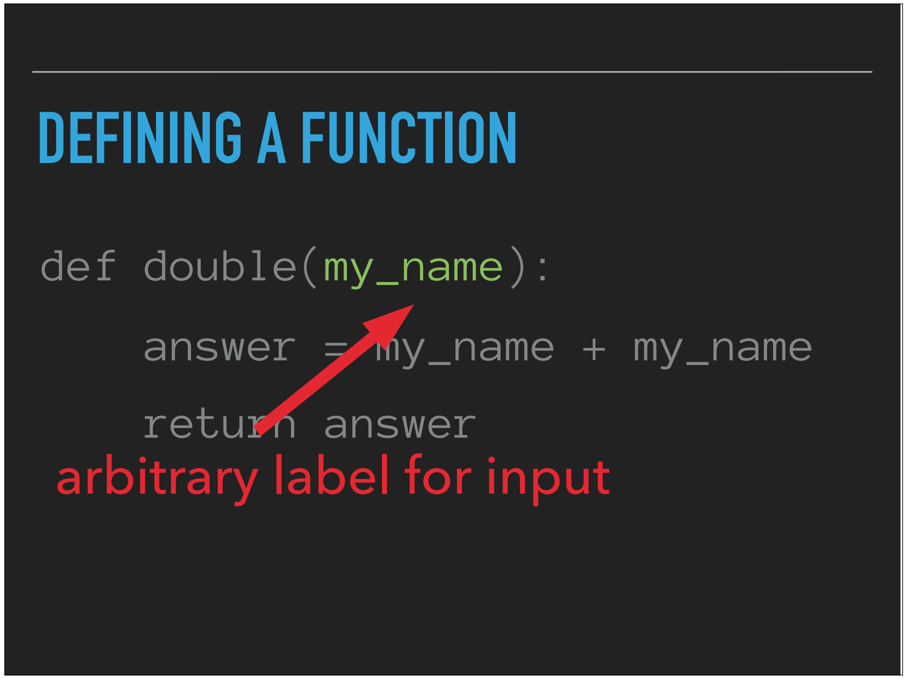
DEFINING A FUNCTION

```
def double(my_name):
    answer = my_name + my_name
    return answer
```



DEFINING A FUNCTION

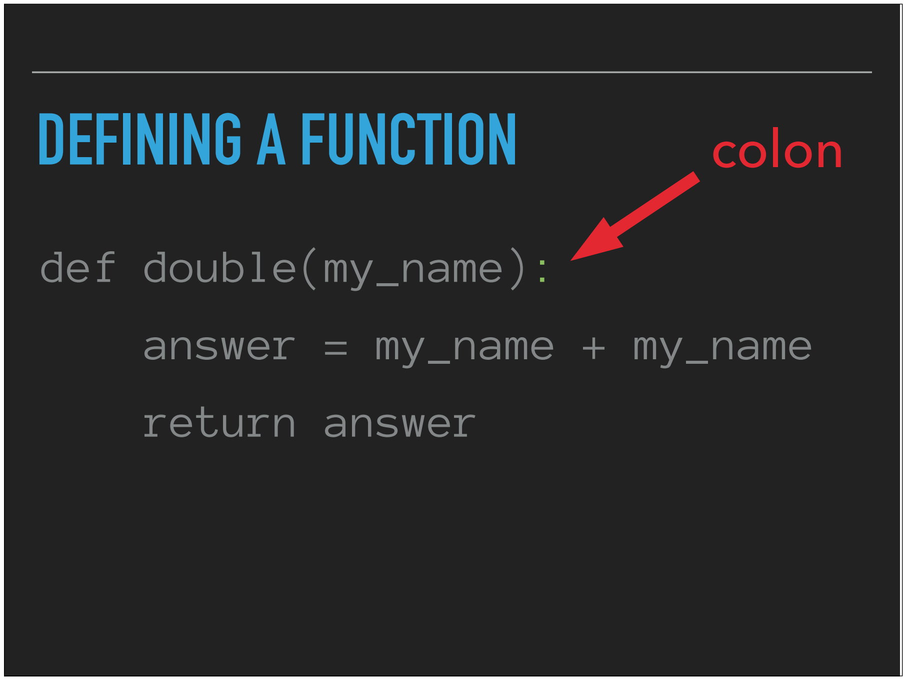
```
def double(my_name):
    answer = my_name + my_name
    return answer
```



DEFINING A FUNCTION

```
def double(my_name):
    answer = my_name + my_name
    return answer
```

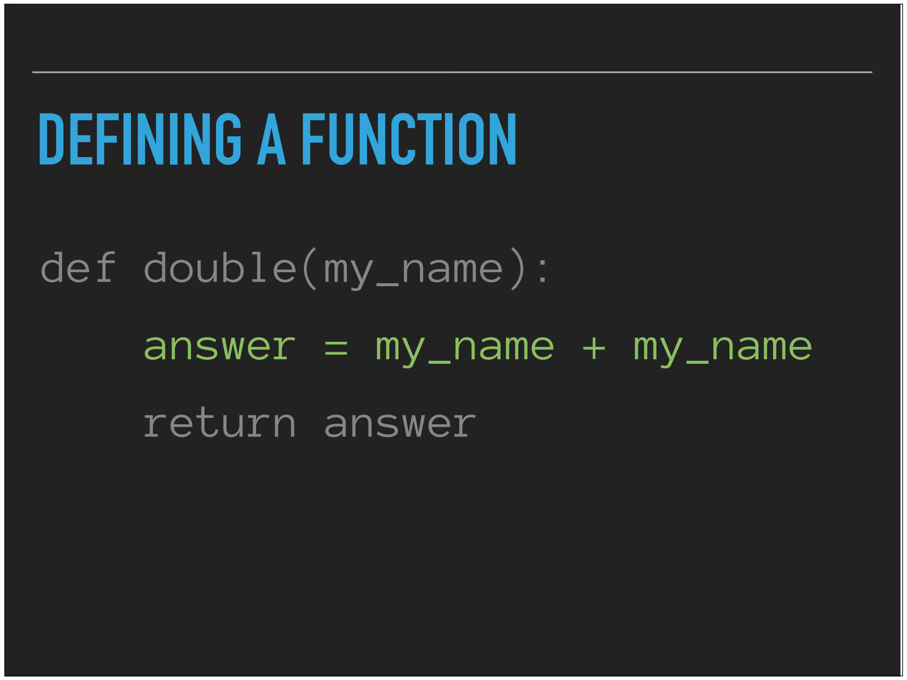
arbitrary label for input



DEFINING A FUNCTION

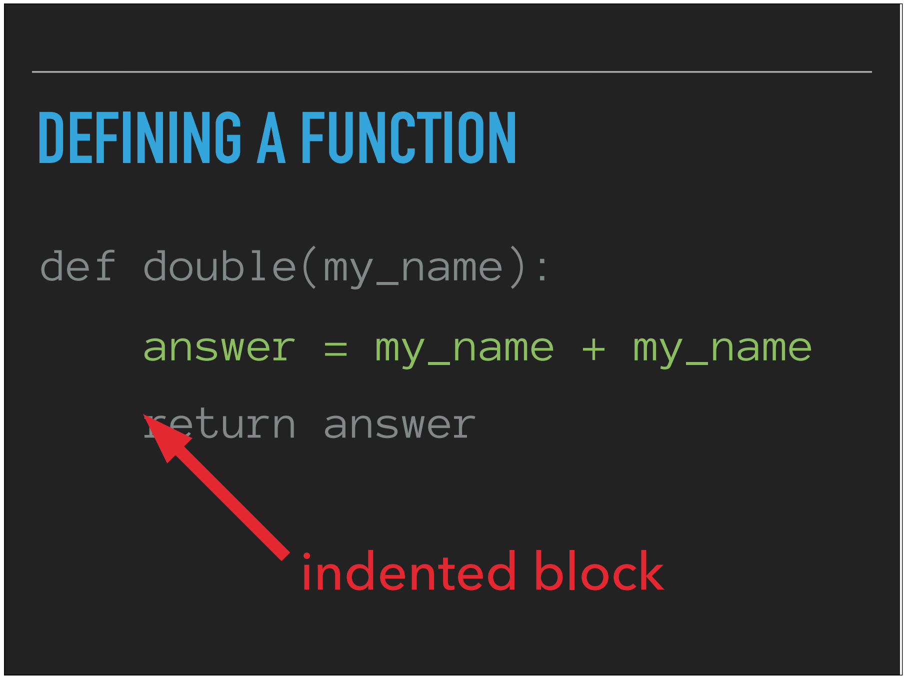
colon

```
def double(my_name):
    answer = my_name + my_name
    return answer
```



DEFINING A FUNCTION

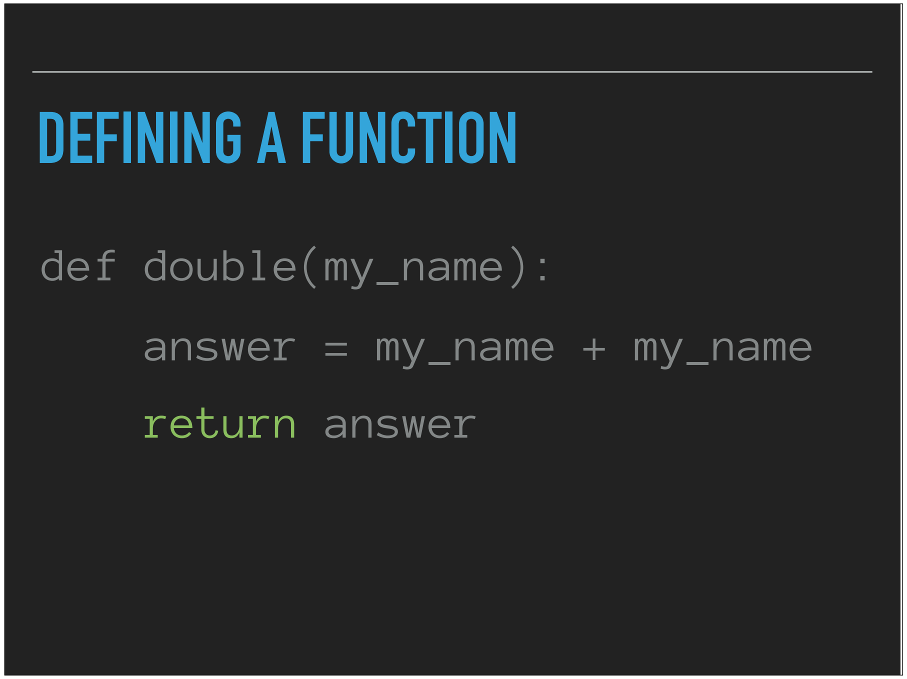
```
def double(my_name):
    answer = my_name + my_name
    return answer
```



DEFINING A FUNCTION

```
def double(my_name):
    answer = my_name + my_name
    return answer
```

indented block



DEFINING A FUNCTION

```
def double(my_name):
    answer = my_name + my_name
    return answer
```

CALLING A FUNCTION

```
double(“Rob”)
```

CALLING A FUNCTION

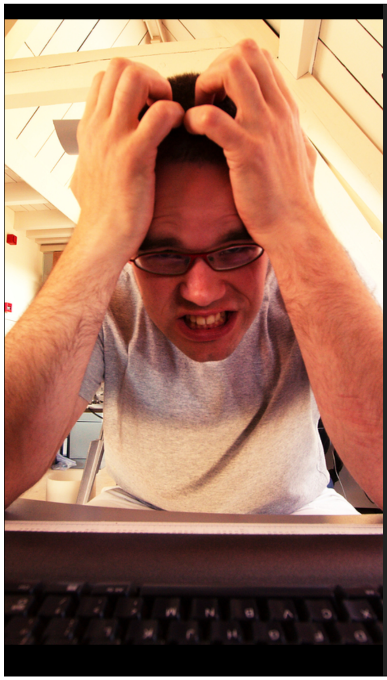
```
def double(“Rob”):
    answer = “Rob” + “Rob”
    return answer
```

CALLING A FUNCTION

```
double(“Rob”)
```

no space

EXERCISES



LET'S TALK ABOUT

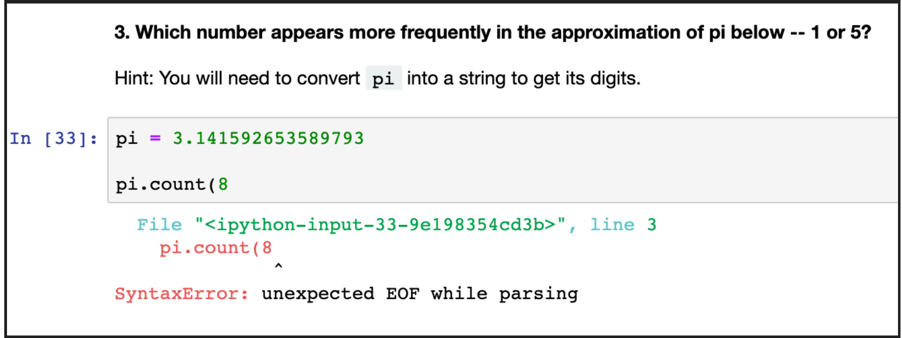
DEBUGGING

DEFAULT DEBUGGING STRATEGY

THREE TYPES OF BUGS

- Syntactic: Code won't run
- Runtime: Code stops with error
- Semantic: Runs, but wrong output

SOLVING SYNTACTIC BUGS



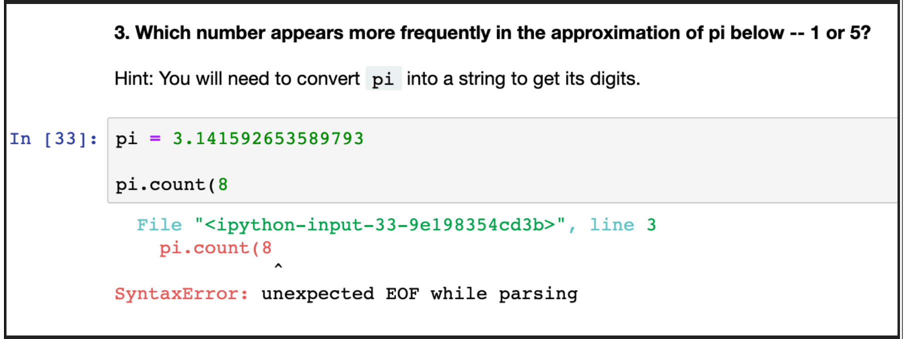
3. Which number appears more frequently in the approximation of pi below -- 1 or 5?

Hint: You will need to convert `pi` into a string to get its digits.

```
In [33]: pi = 3.141592653589793

         pi.count(8
```

```
  File "<ipython-input-33-9e198354cd3b>", line 3
    pi.count(8
              ^
SyntaxError: unexpected EOF while parsing
```

SOLVING SYNTACTIC BUGS



3. Which number appears more frequently in the approximation of pi below -- 1 or 5?

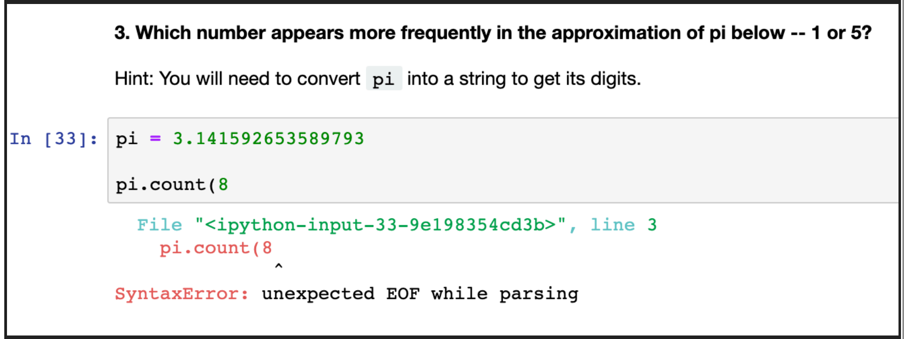
Hint: You will need to convert `pi` into a string to get its digits.

```
In [33]: pi = 3.141592653589793

         pi.count(8
```

```
  File "<ipython-input-33-9e198354cd3b>", line 3
    pi.count(8
              ^
SyntaxError: unexpected EOF while parsing
```

1. Find line number in error

SOLVING SYNTACTIC BUGS



3. Which number appears more frequently in the approximation of pi below -- 1 or 5?

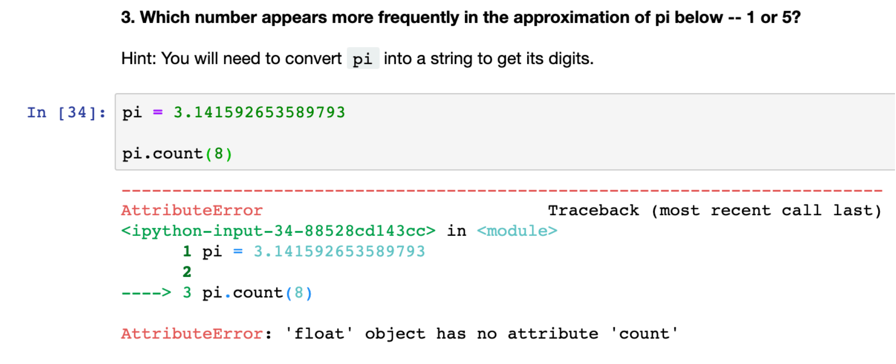
Hint: You will need to convert `pi` into a string to get its digits.

```
In [33]: pi = 3.141592653589793

         pi.count(8
```

```
  File "<ipython-input-33-9e198354cd3b>", line 3
    pi.count(8
              ^
SyntaxError: unexpected EOF while parsing
```

1. Find line number in error
2. Compare your code with recipe

SOLVING RUNTIME BUGS



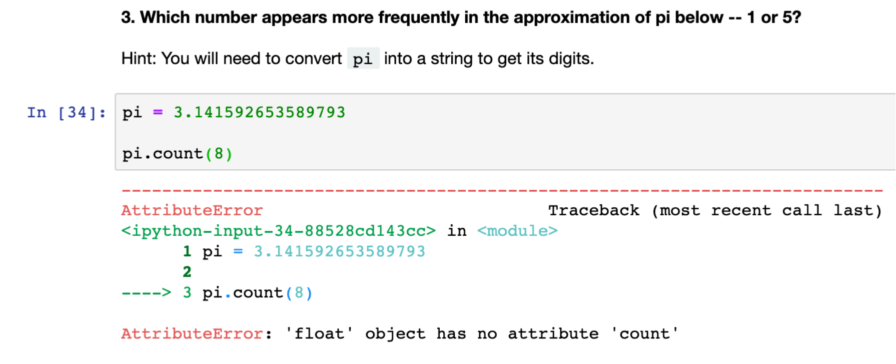
3. Which number appears more frequently in the approximation of pi below -- 1 or 5?

Hint: You will need to convert `pi` into a string to get its digits.

```
In [34]: pi = 3.141592653589793

         pi.count(8)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-34-88528cd143cc> in <module>
      1 pi = 3.141592653589793
      2 
----> 3 pi.count(8)

AttributeError: 'float' object has no attribute 'count'
```

SOLVING RUNTIME BUGS



3. Which number appears more frequently in the approximation of pi below -- 1 or 5?

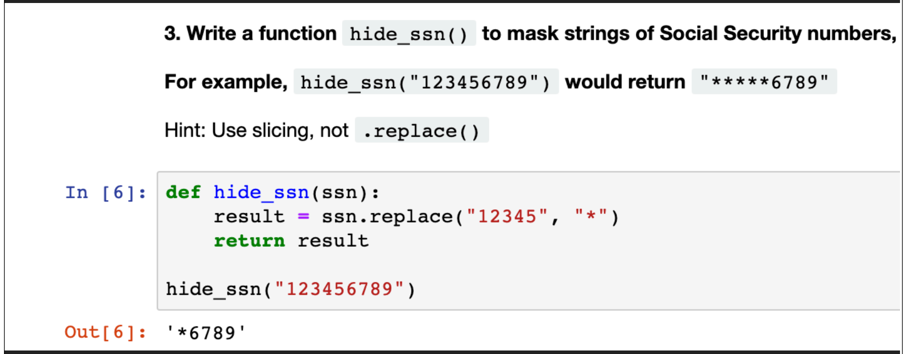
Hint: You will need to convert `pi` into a string to get its digits.

```
In [34]: pi = 3.141592653589793

         pi.count(8)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-34-88528cd143cc> in <module>
      1 pi = 3.141592653589793
      2 
----> 3 pi.count(8)

AttributeError: 'float' object has no attribute 'count'
```

Double check your data types

SOLVING SEMANTIC BUGS



3. Write a function `hide_ssn()` to mask strings of Social Security numbers,

For example, `hide_ssn("123456789")` would return `"*****6789"`

Hint: Use slicing, not `.replace()`

```
In [6]: def hide_ssn(ssn):
            result = ssn.replace("12345", "*")
            return result

        hide_ssn("123456789")

Out[6]: '*6789'
```

PRINT FUNCTION

English:

Print value of my_name on the screen

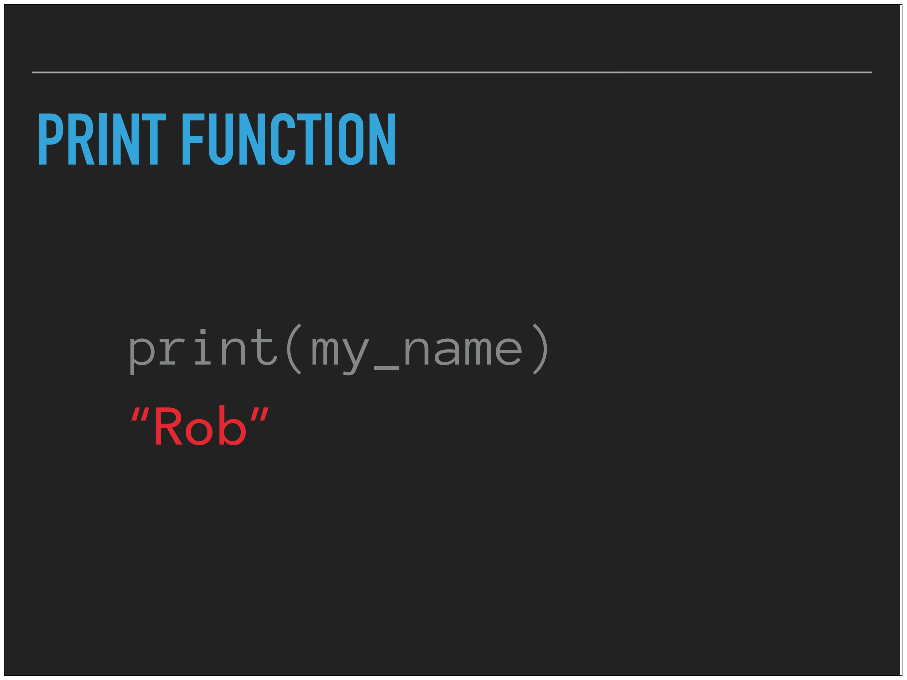
PRINT FUNCTION

English:

Print my_name on the screen

PRINT FUNCTION

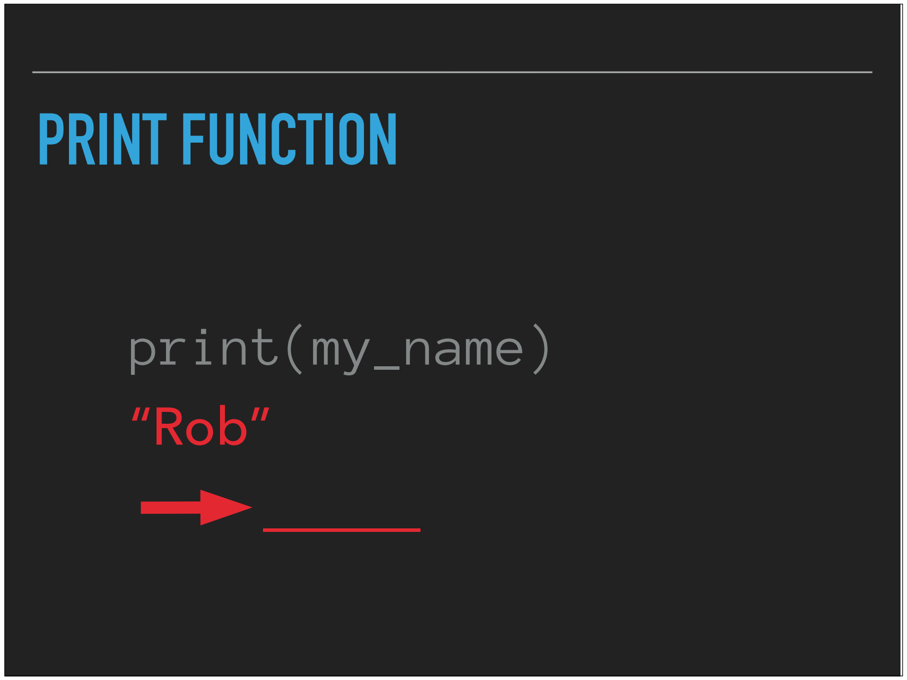
```
print(my_name)
```



PRINT FUNCTION

```
print(my_name)
```

“Rob”

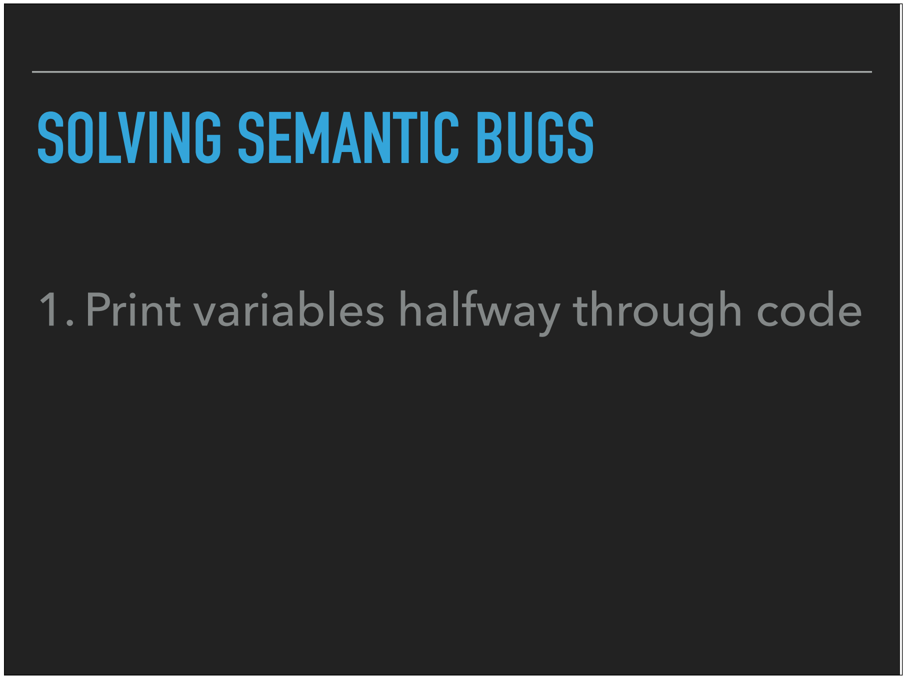


PRINT FUNCTION

```
print(my_name)
```

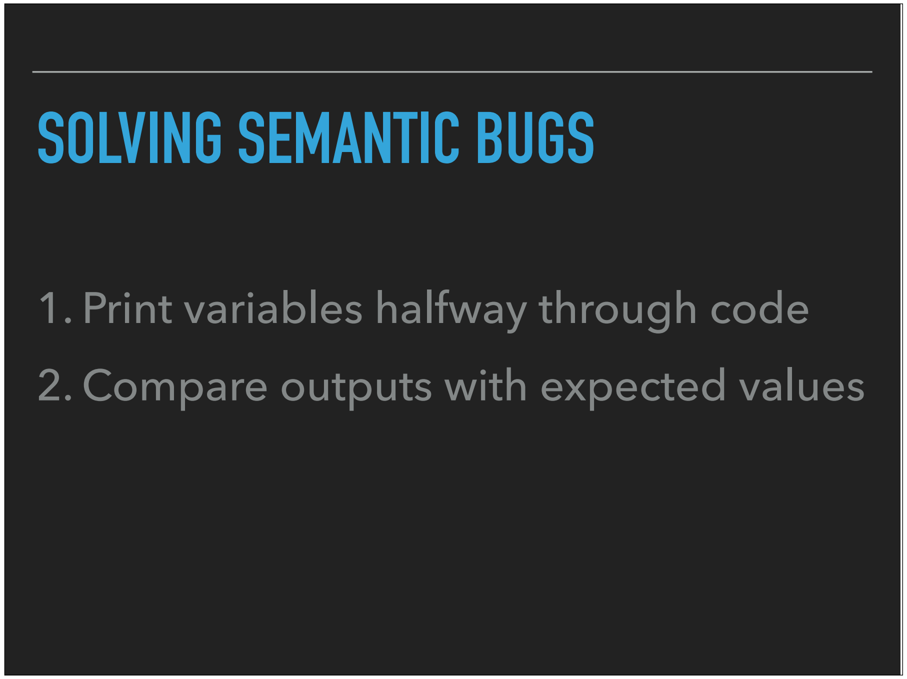
“Rob”

→ ______



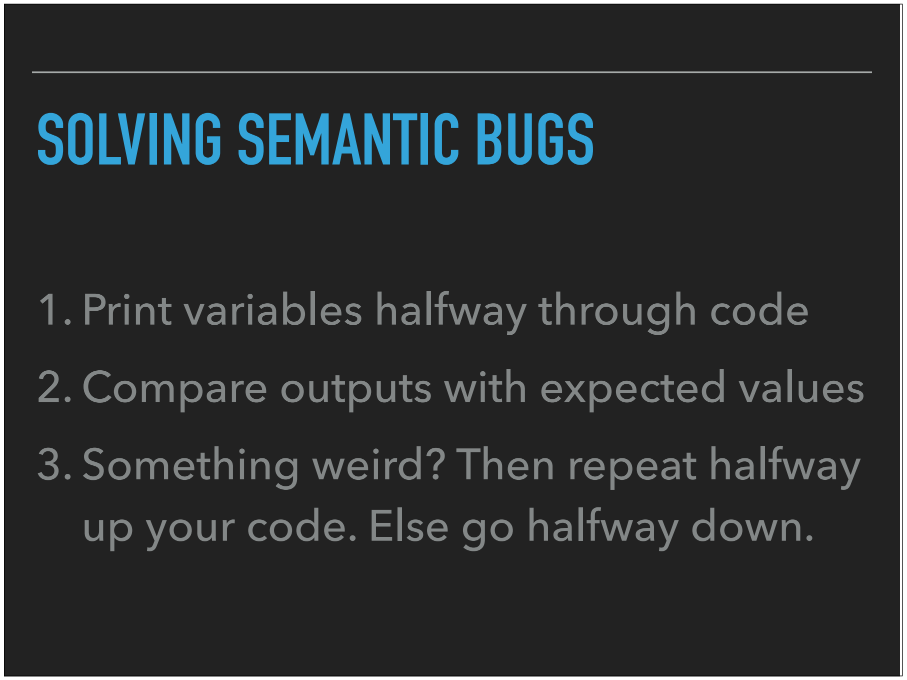
SOLVING SEMANTIC BUGS

1. Print variables halfway through code



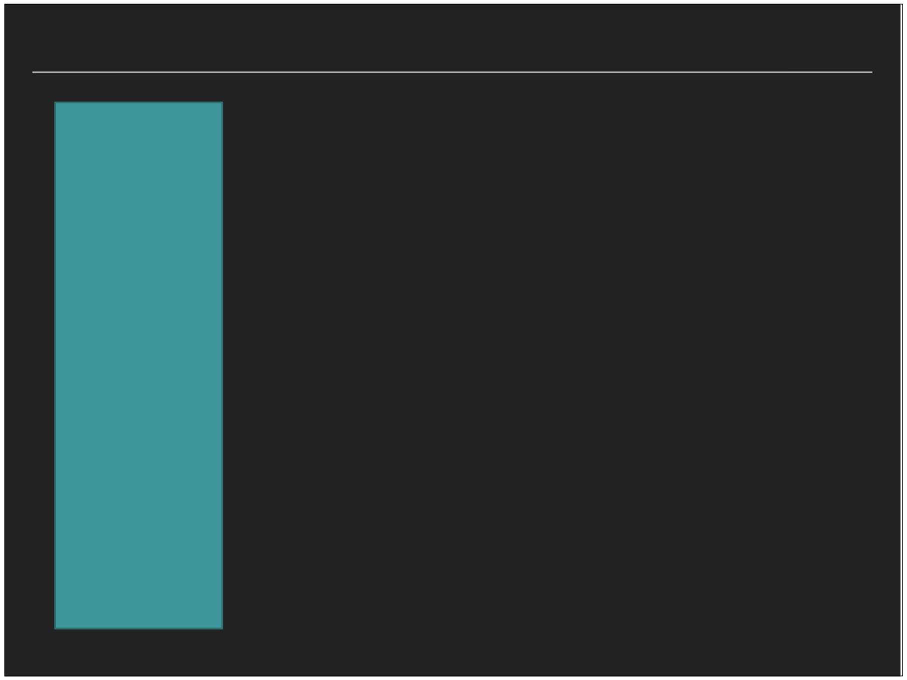
SOLVING SEMANTIC BUGS

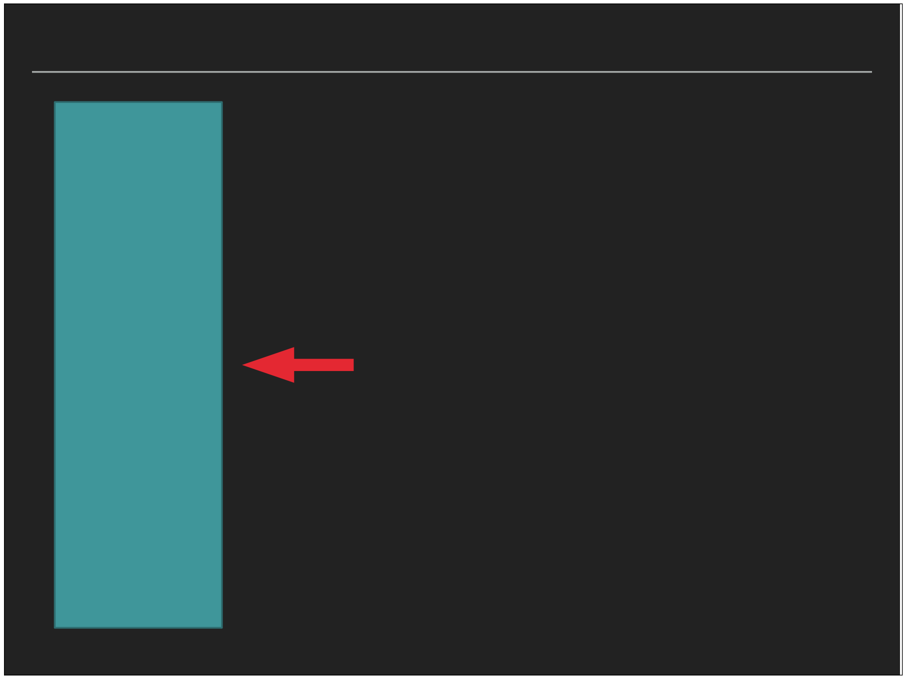
1. Print variables halfway through code
2. Compare outputs with expected values

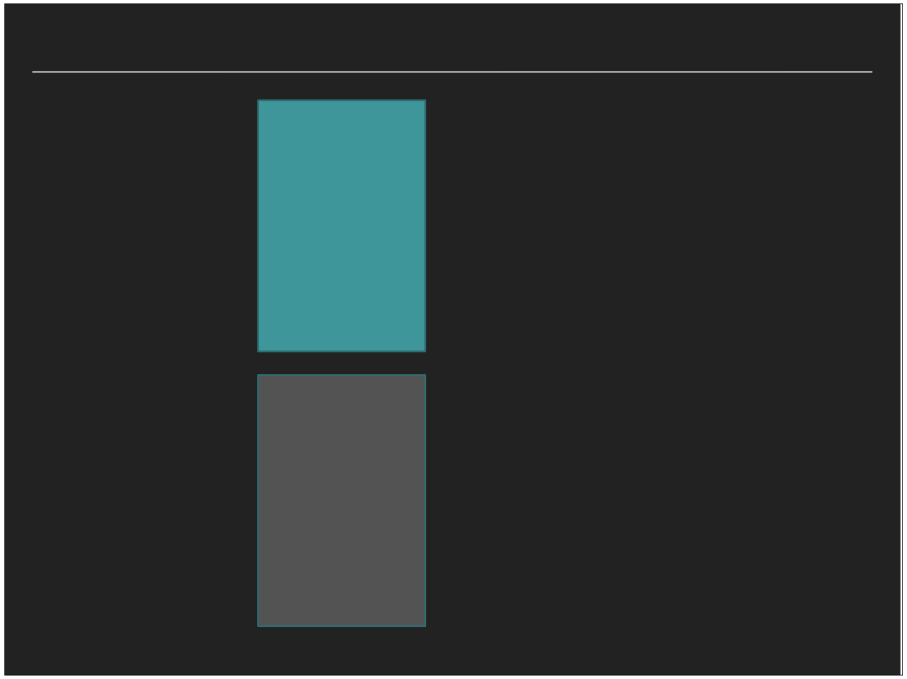


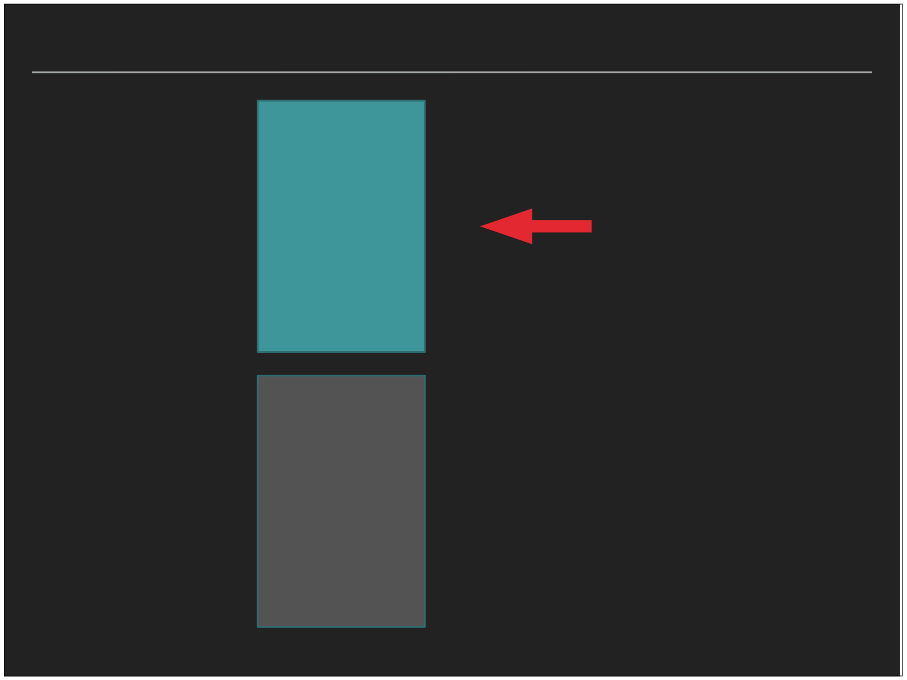
SOLVING SEMANTIC BUGS

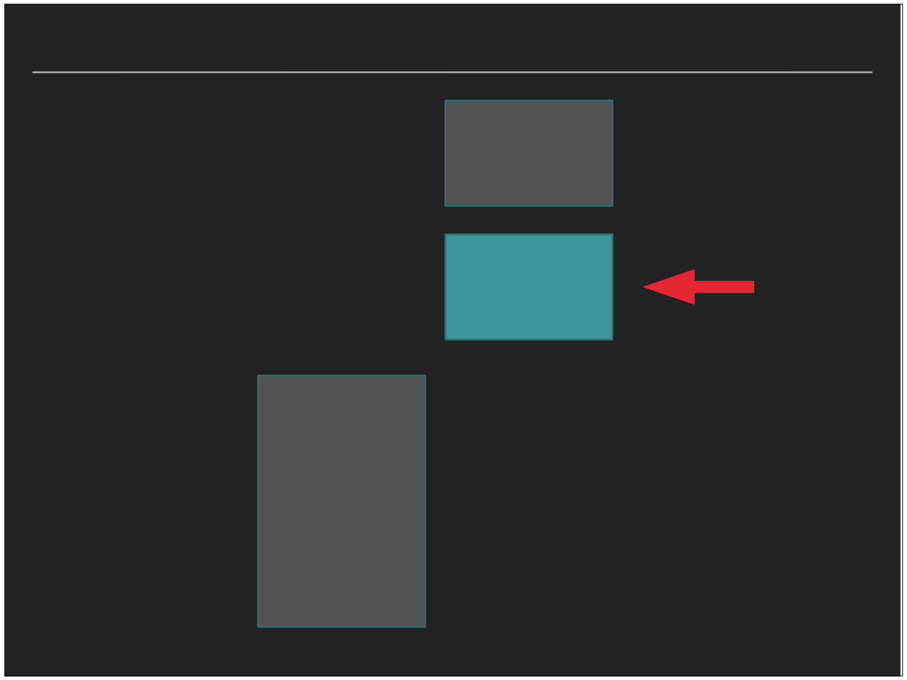
1. Print variables halfway through code
2. Compare outputs with expected values
3. Something weird? Then repeat halfway up your code. Else go halfway down.

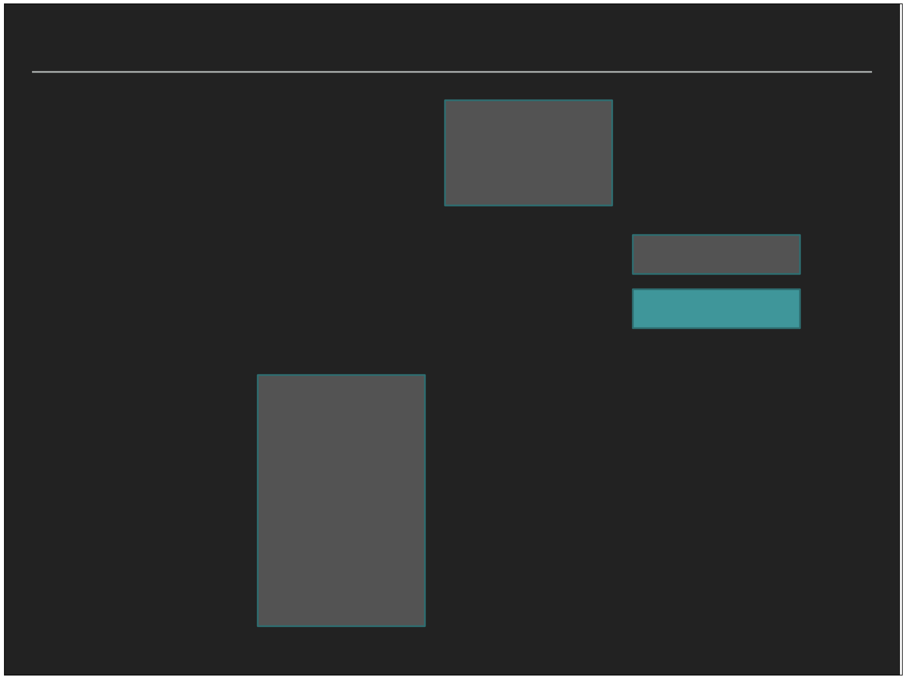


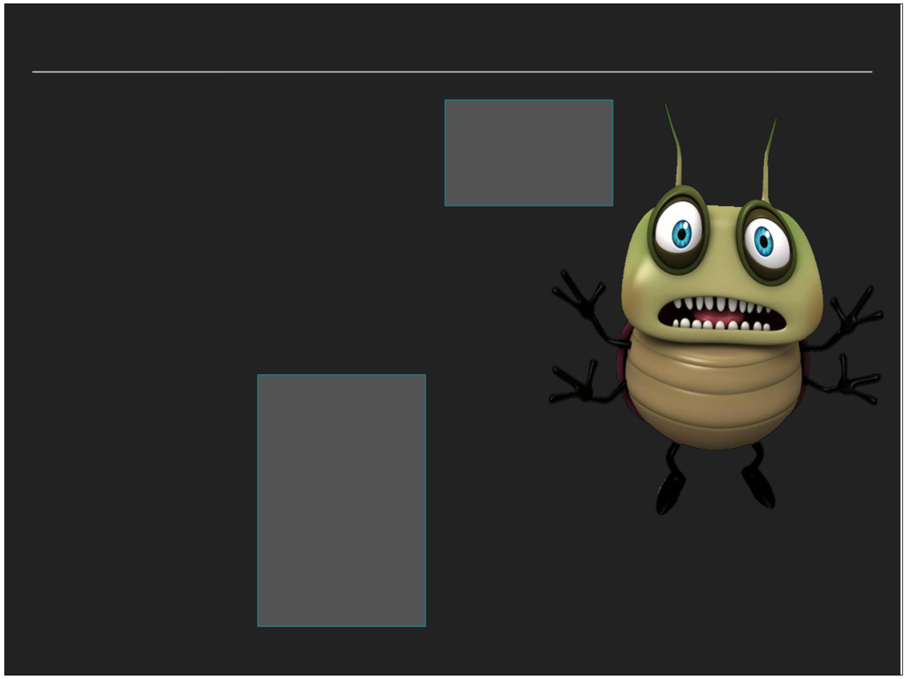


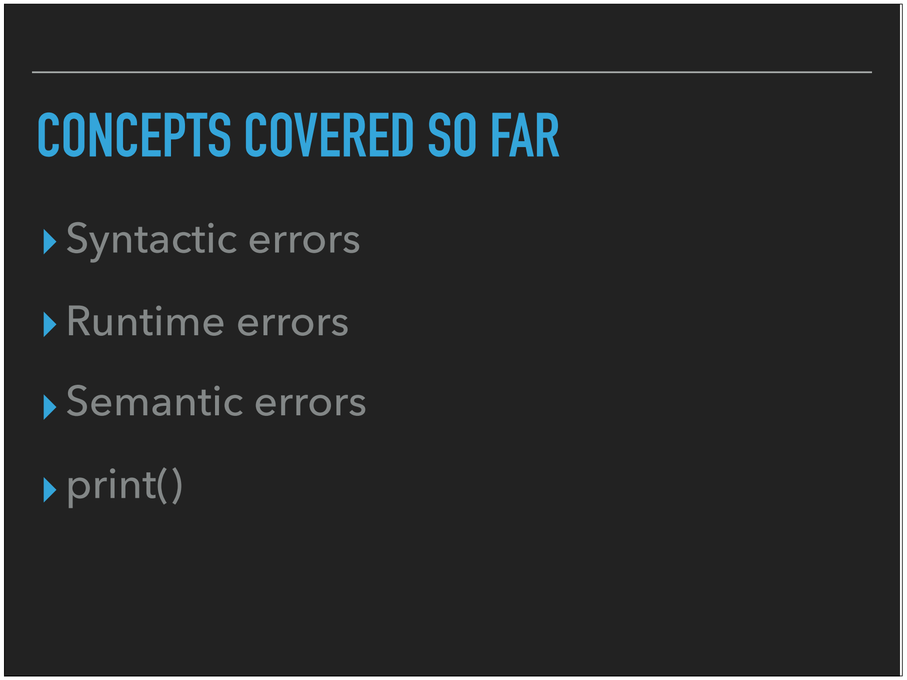








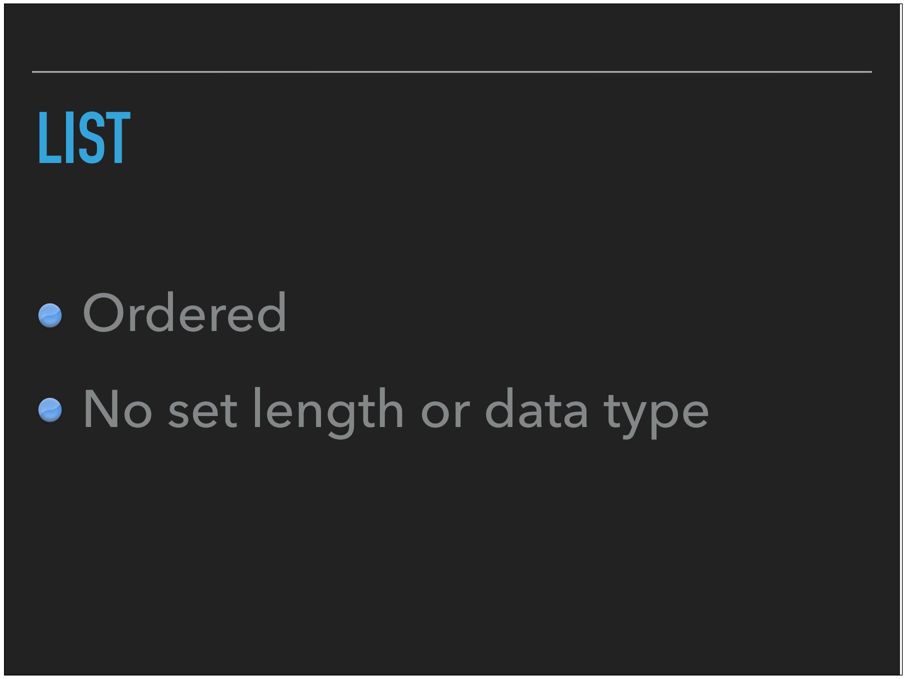




CONCEPTS COVERED SO FAR

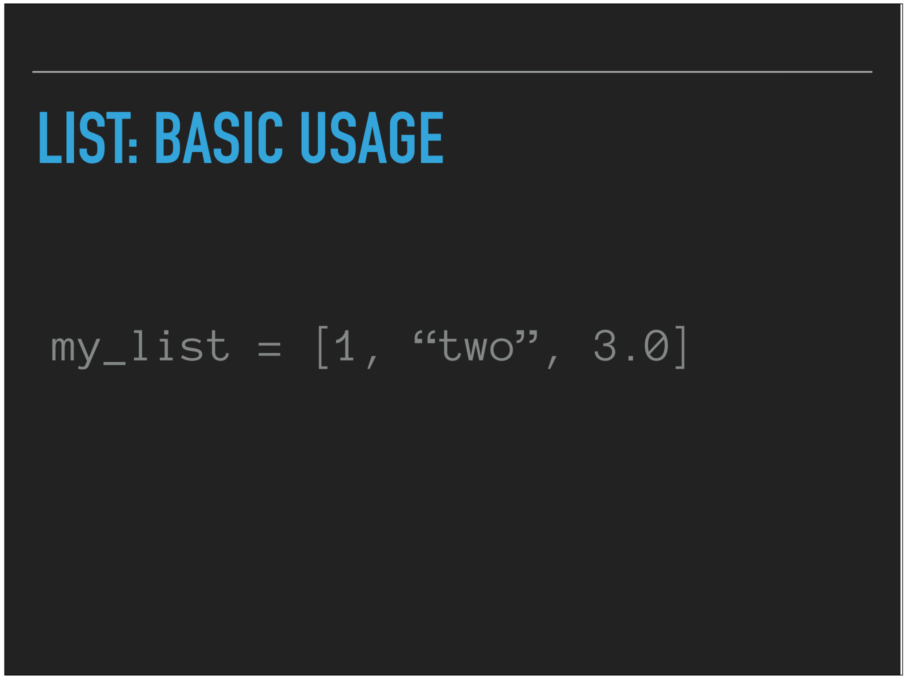
- Syntactic errors
- Runtime errors
- Semantic errors
- print()

EXERCISES



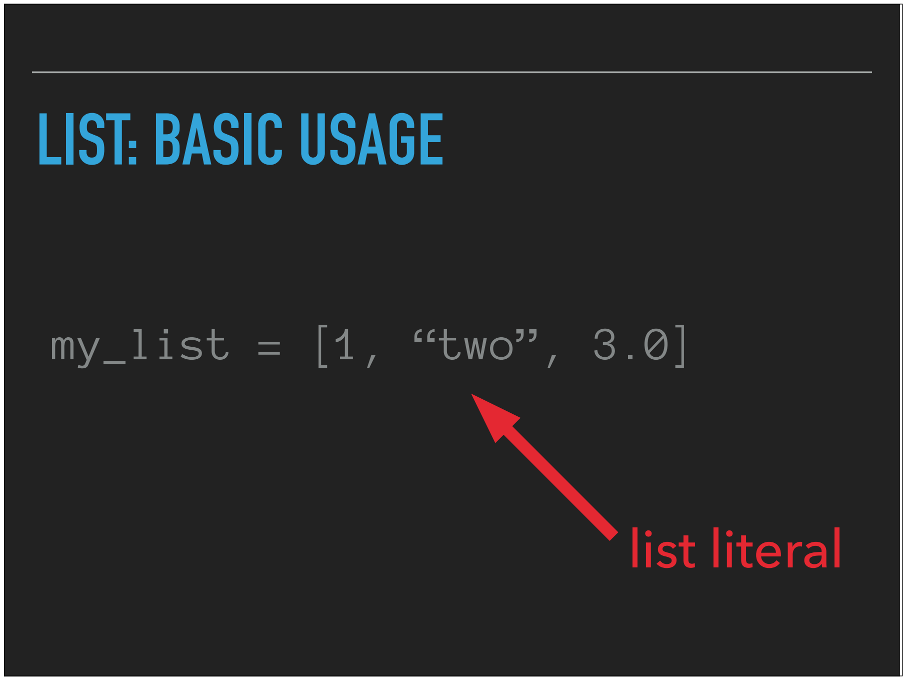
LIST

- Ordered
- No set length or data type



LIST: BASIC USAGE

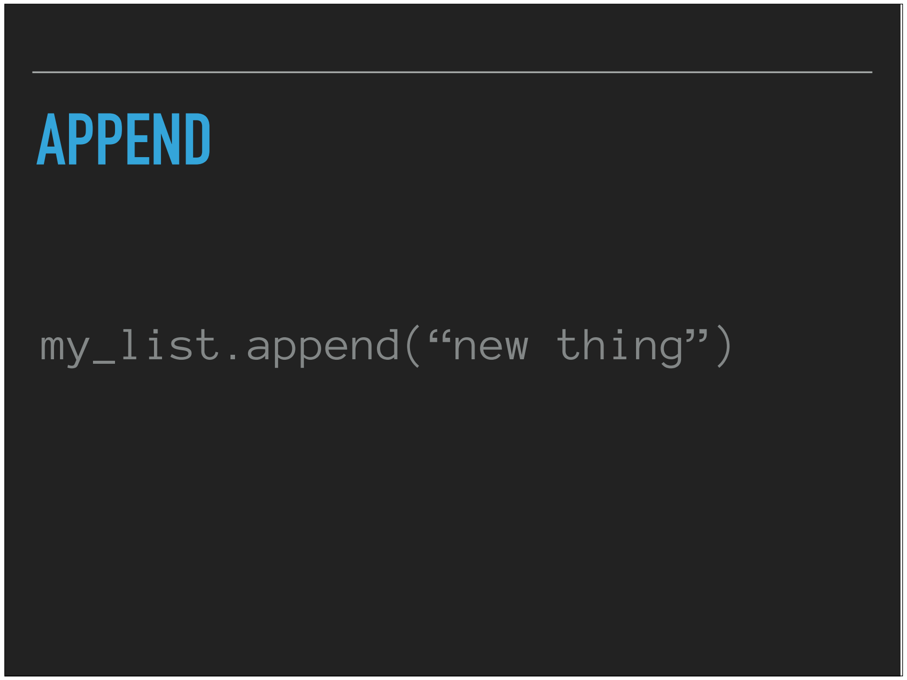
```
my_list = [1, “two”, 3.0]
```



LIST: BASIC USAGE

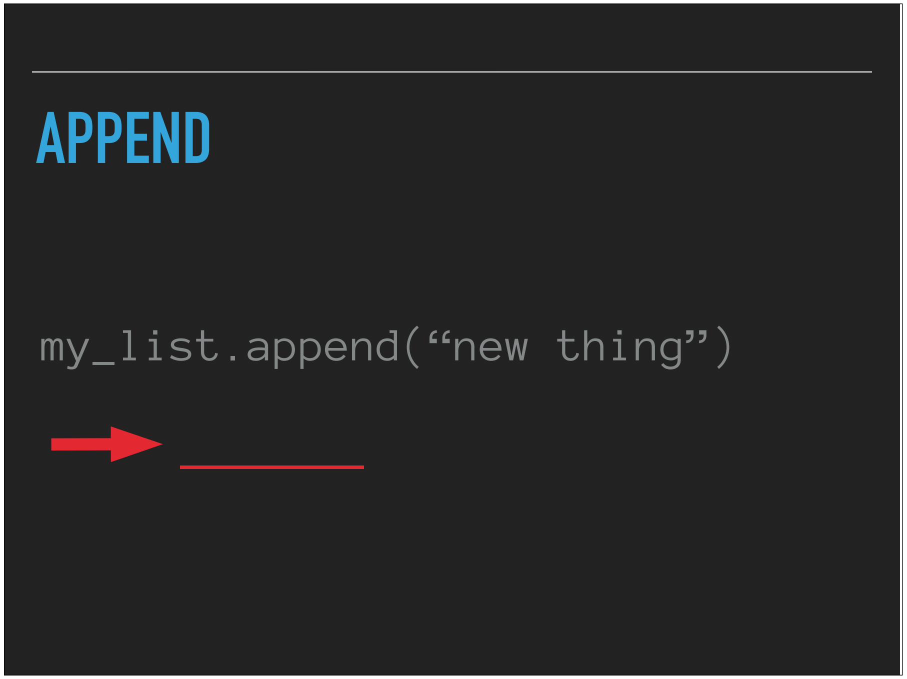
```
my_list = [1, “two”, 3.0]
```

list literal



APPEND

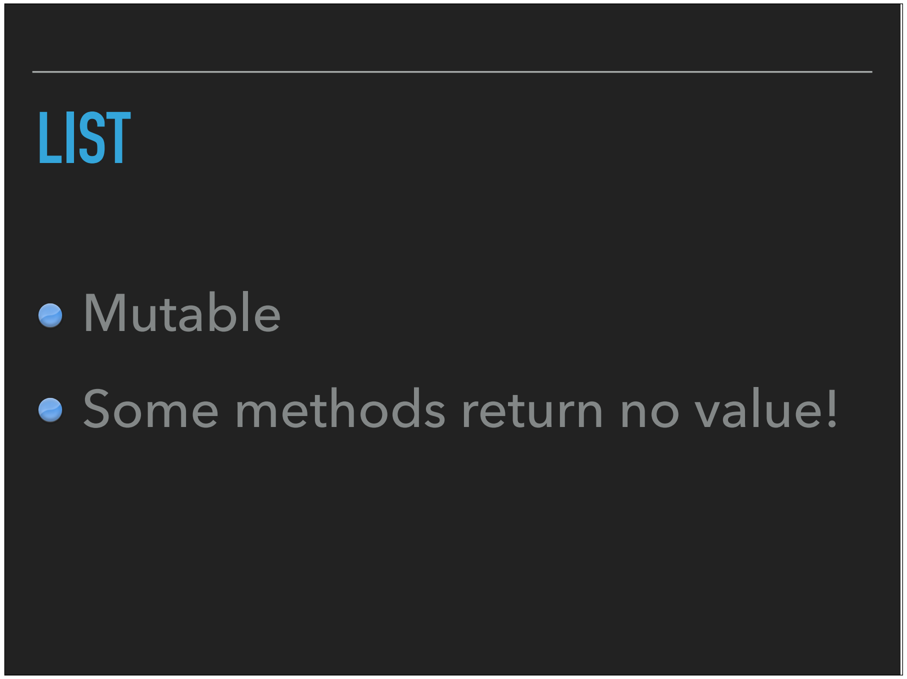
```
my_list.append(“new thing”)
```



APPEND

```
my_list.append(“new thing”)
```

➡ ______



LIST

- Mutable
- Some methods return no value!

STRINGS

- Immutable
- Methods always return new string

NUMBERS

- Immutable
- Methods always return new string

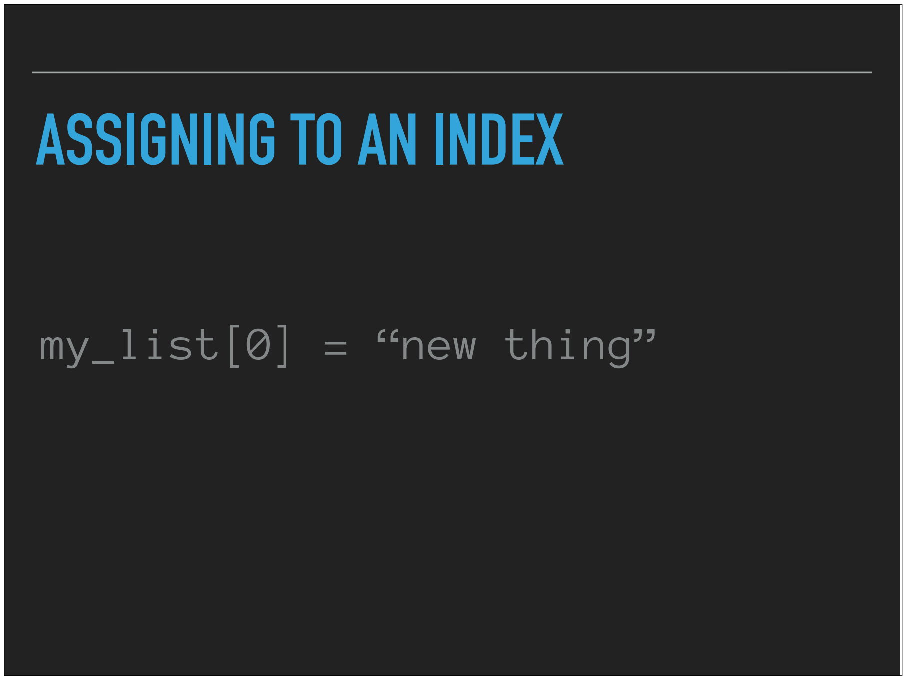
LIST: BASIC USAGE

```
my_list.append(“new thing”)
print(my_list)
```

LIST: BASIC USAGE

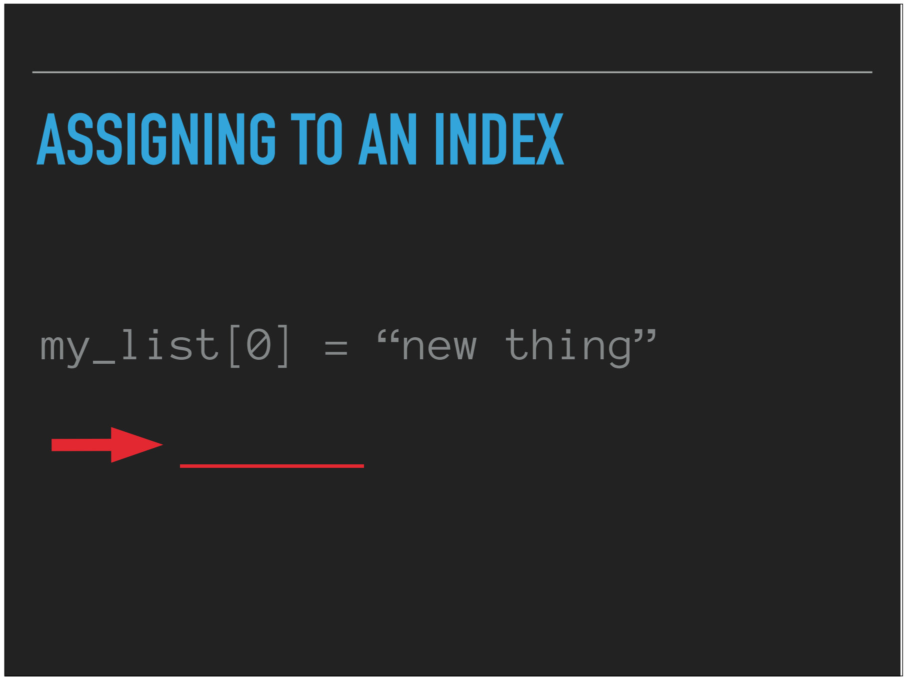
```
my_list.append(“new thing”)
print(my_list)
```

→ [1, “two”, 3.0, “new thing”]



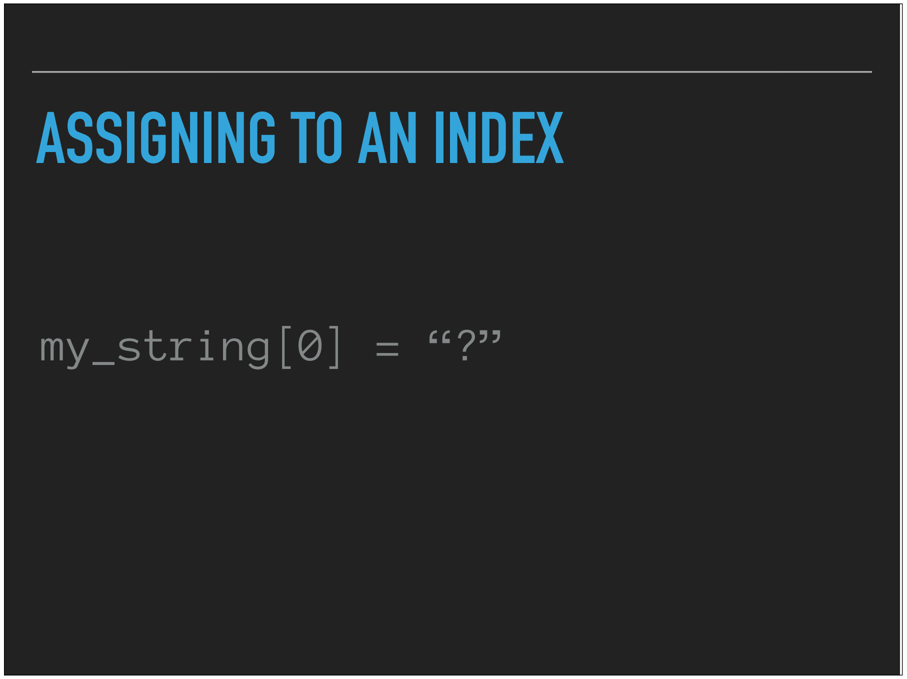
ASSIGNING TO AN INDEX

```
my_list[0] = “new thing”
```



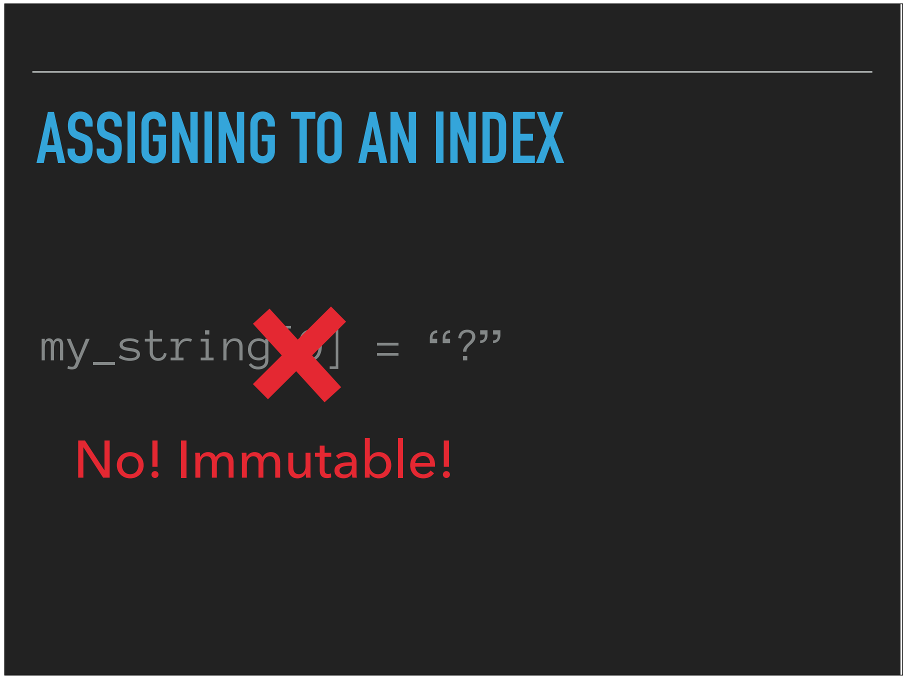
ASSIGNING TO AN INDEX

```
my_list[0] = “new thing”
```



ASSIGNING TO AN INDEX

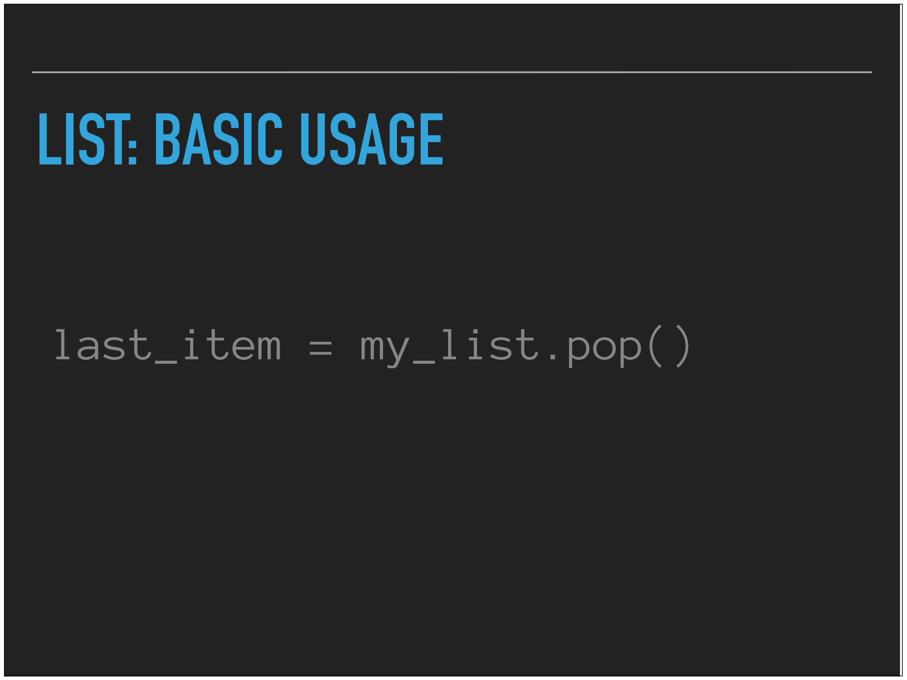
```
my_string[0] = “?”
```



ASSIGNING TO AN INDEX

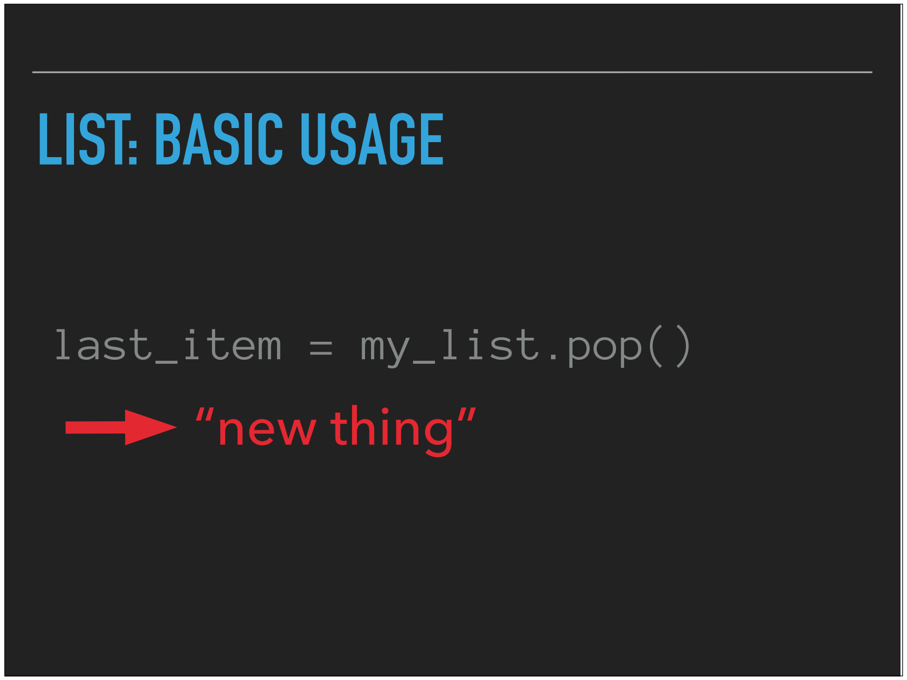
```
my_string[0] = “?”
```

No! Immutable!



LIST: BASIC USAGE

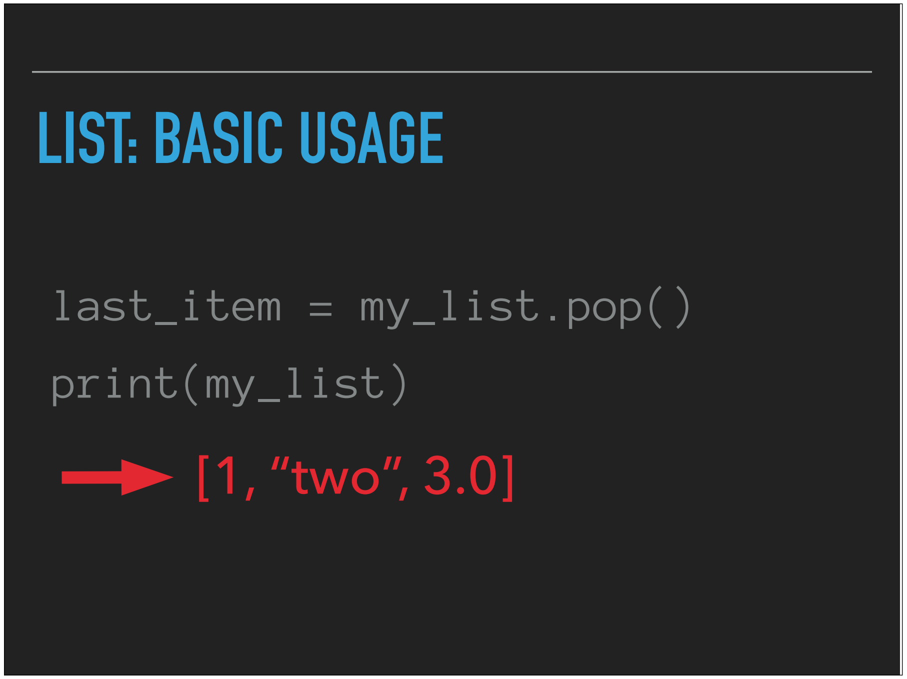
```
last_item = my_list.pop()
```



LIST: BASIC USAGE

```
last_item = my_list.pop()
```

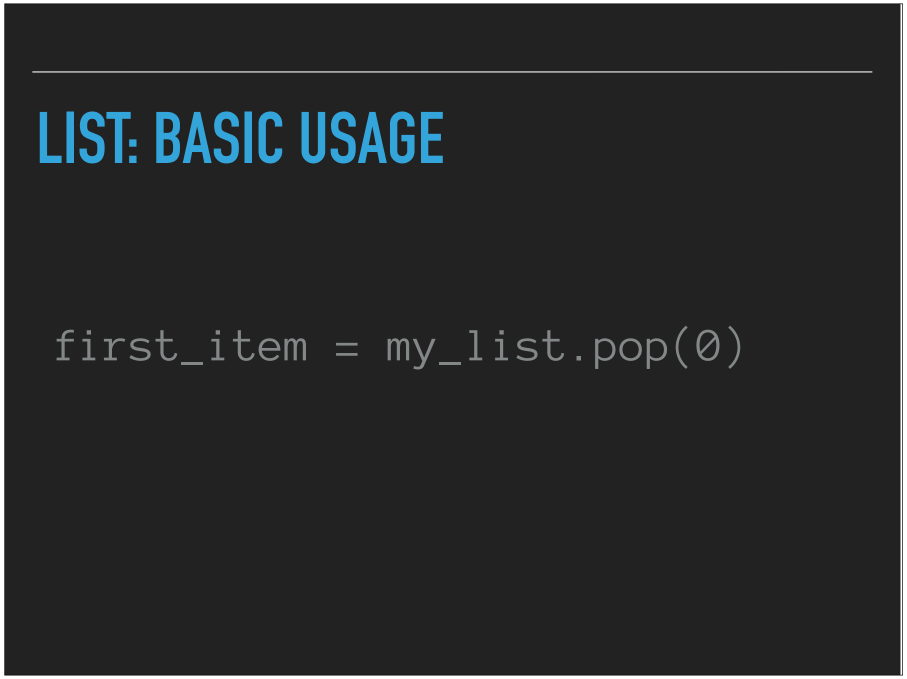
→ “new thing”



LIST: BASIC USAGE

```
last_item = my_list.pop()
print(my_list)
```

→ [1, “two”, 3.0]



LIST: BASIC USAGE

```
first_item = my_list.pop(0)
```

LIST: BASIC USAGE

```
first_item = my_list.pop(0)
```

→ 1

CONCEPTS COVERED SO FAR

- Lists
- Mutability
- No return value

SEQUENCES

STRINGS

LISTS

STRINGS: SAME

- Same index and slice syntax!

STRINGS: BUT DIFFERENT

- Cannot add or remove characters

EXERCISES