INTRO TO PYTHON

PROGRAMMING

ROB CARRINGTON

- 4 years as a Python developer
- Worked in FinTech (@Q2eBanking)

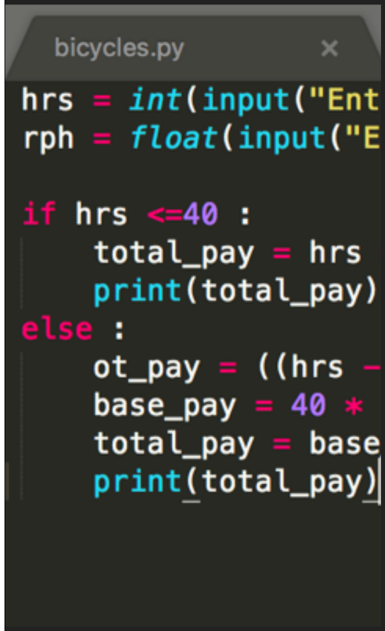
Q2

First Republic
It's a privilege to serve you®

Apple Bank

Hanmi Bank

WHY PYTHON?



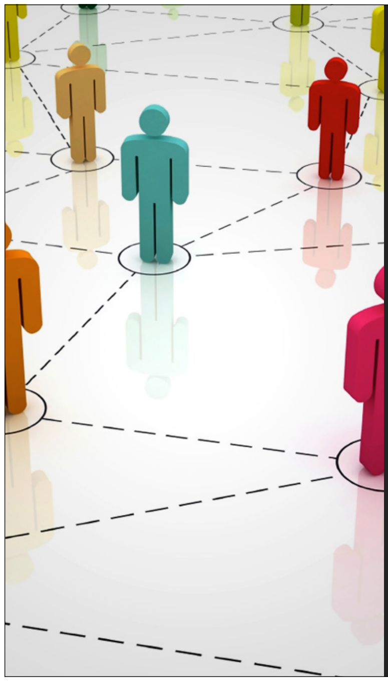
```
hrs = int(input("Ent
rph = float(input("E

if hrs <=40 :
    total_pay = hrs
    print(total_pay)
else :
    ot_pay = ((hrs -
    base_pay = 40 *
    total_pay = base
    print(total_pay)
```

SHORT, READABLE

CODE

MODERN SOFTWARE

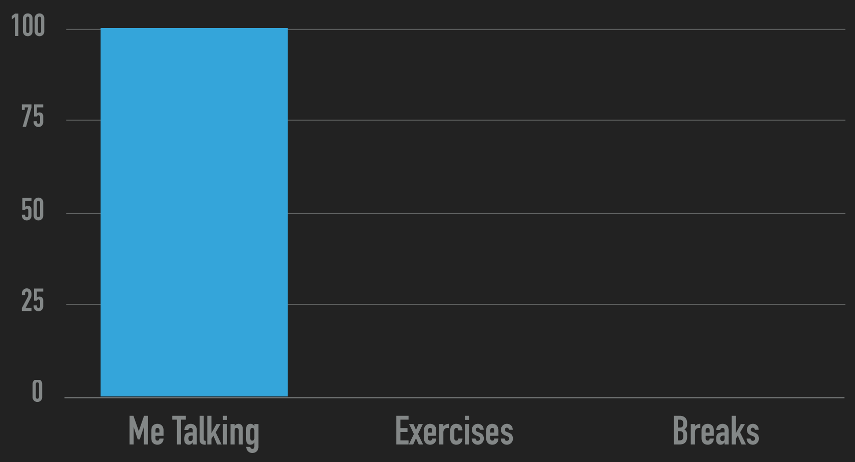
INFRASTRUCTURE



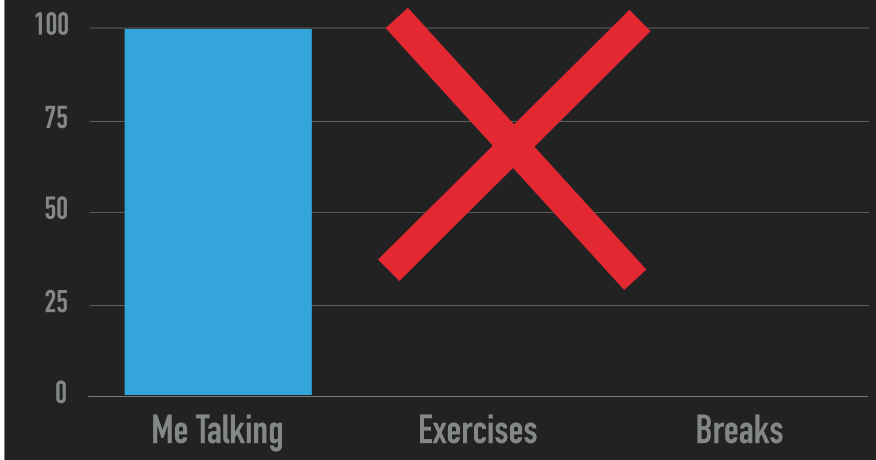
BROAD OPEN-SOURCE

COMMUNITY

CLASS FORMAT



CLASS FORMAT



CLASS FORMAT

- 10 minutes for lesson & demo
- 15 minutes for exercise
- Pause to review & ask questions
- Rinse and repeat

FUNDAMENTALS

HOW TO THINK ABOUT PYTHON

- You are giving orders to your computer
- Each order is called a "statement"

ASSIGNMENT STATEMENT

English:

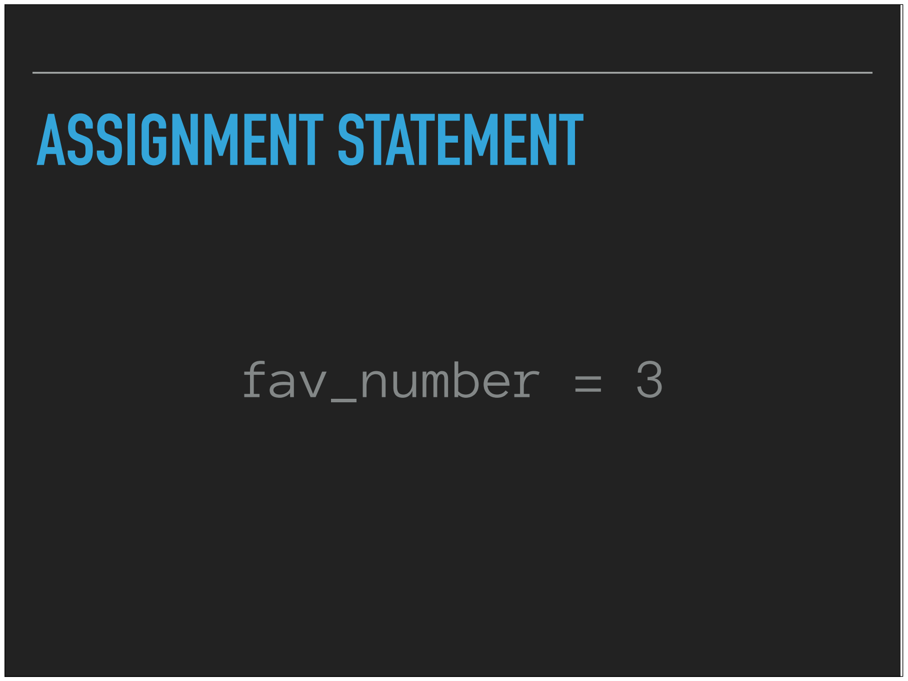
remember "fav_number" means 3

ASSIGNMENT STATEMENT

English:

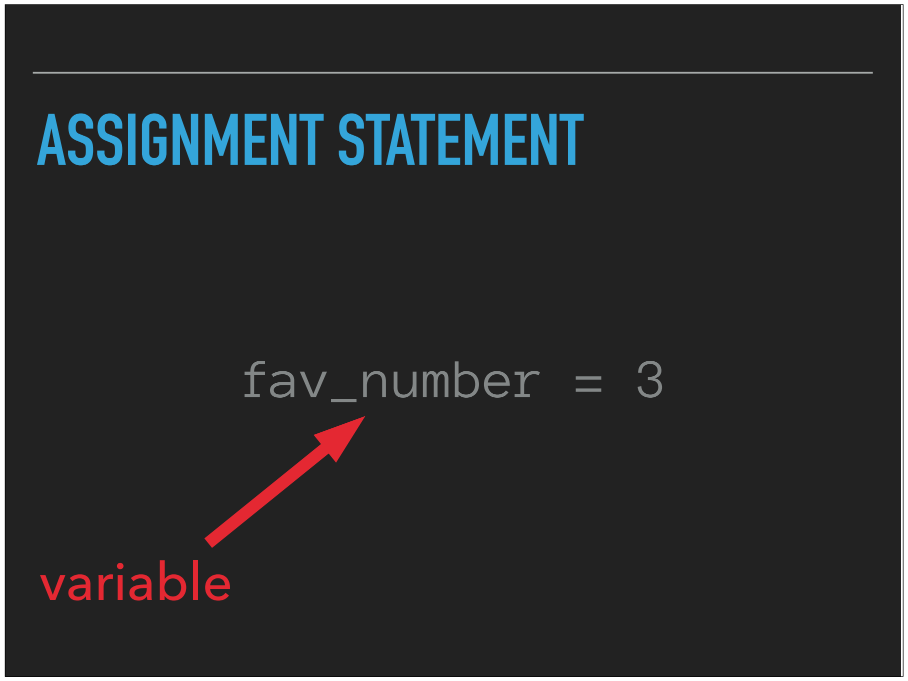
remember "fav_number" means 3

arbitrary label



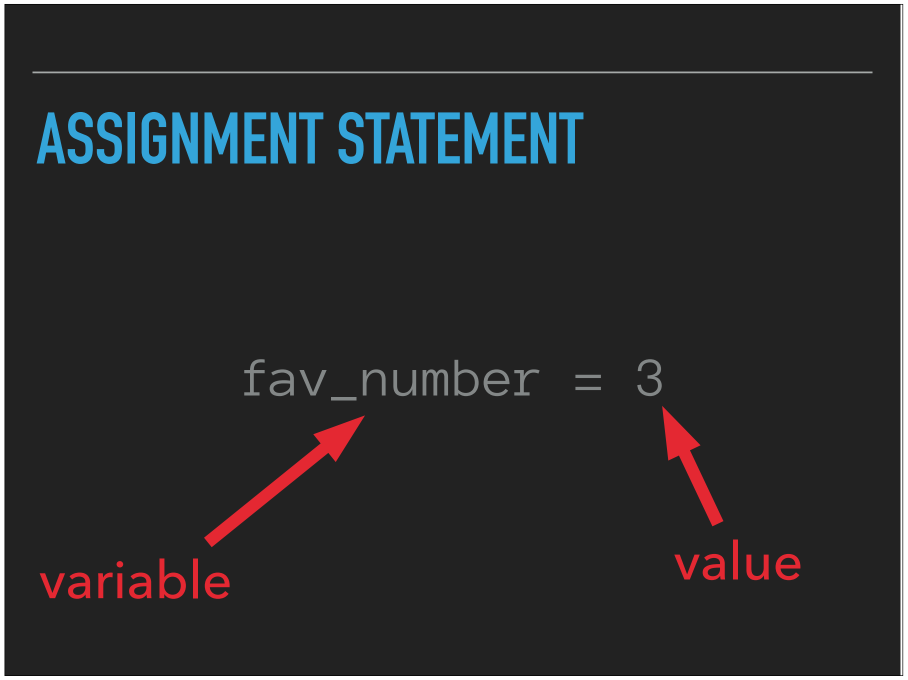
ASSIGNMENT STATEMENT

```
fav_number = 3
```



ASSIGNMENT STATEMENT

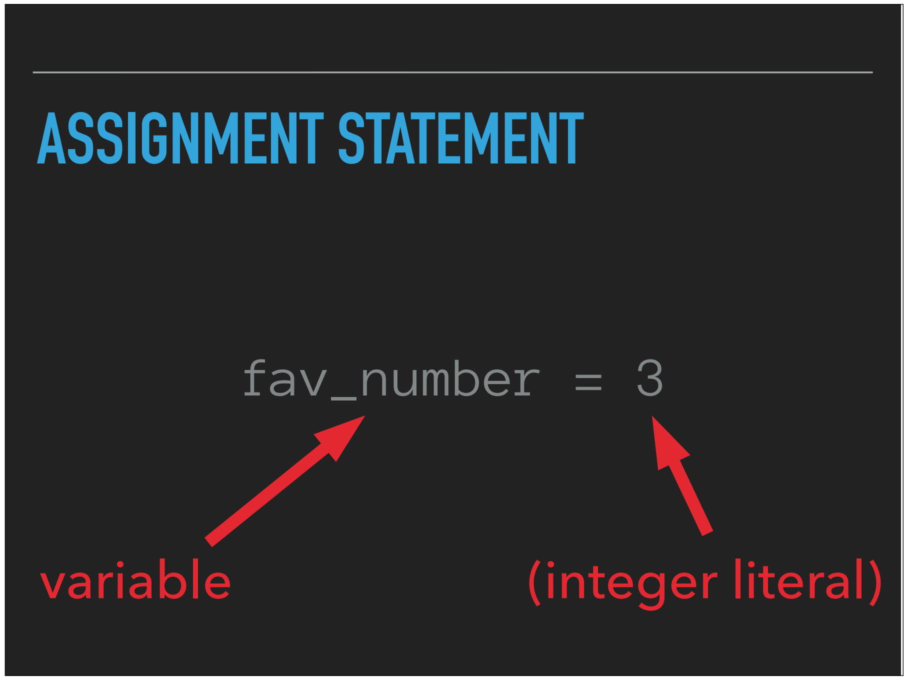
```
fav_number = 3
```



ASSIGNMENT STATEMENT

```
fav_number = 3
```

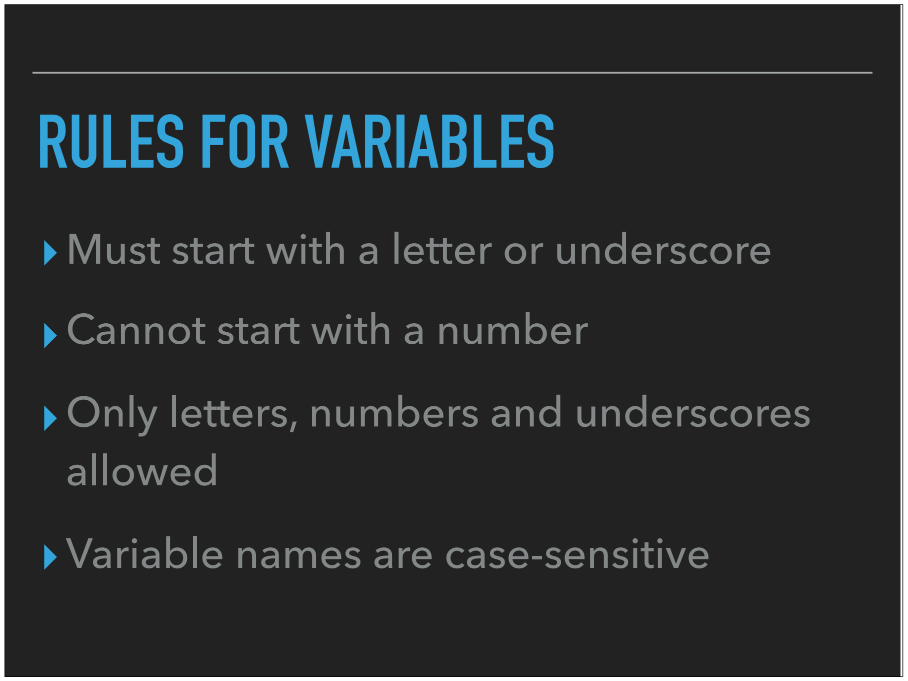
variable
value



ASSIGNMENT STATEMENT

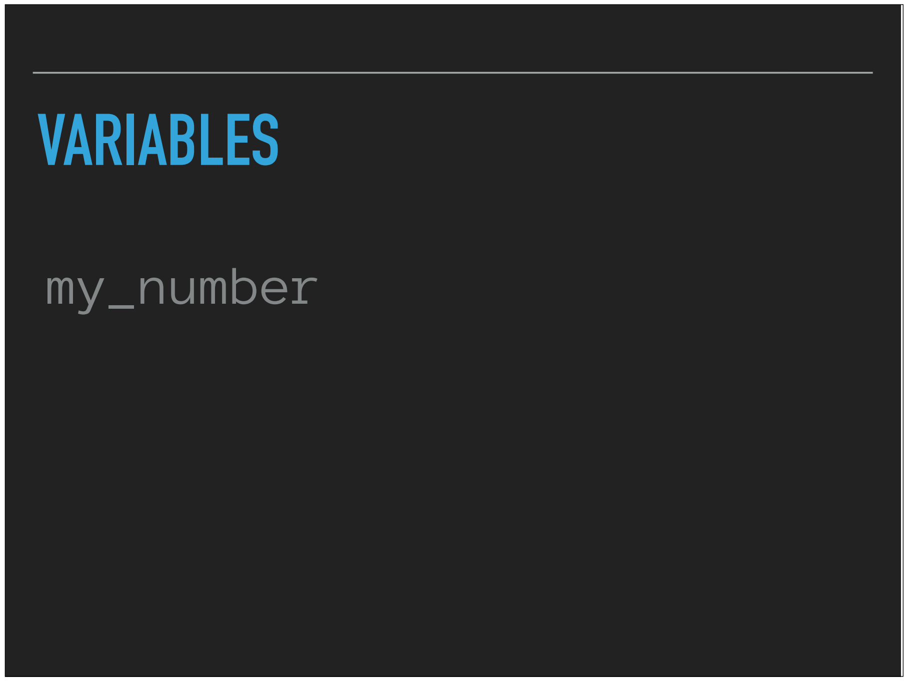
```
fav_number = 3
```

variable
(integer literal)



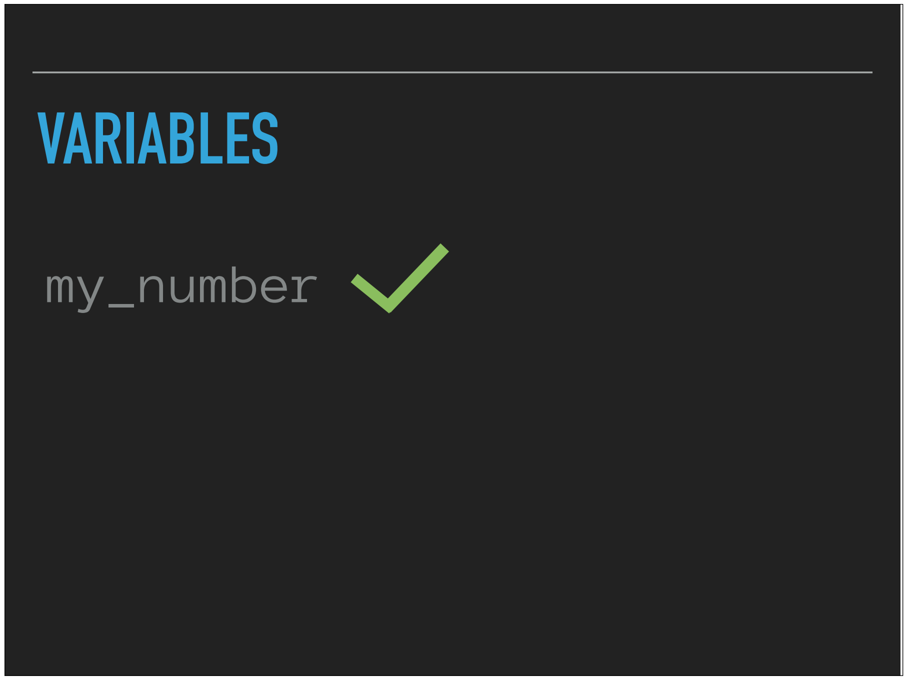
RULES FOR VARIABLES

- Must start with a letter or underscore
- Cannot start with a number
- Only letters, numbers and underscores allowed
- Variable names are case-sensitive



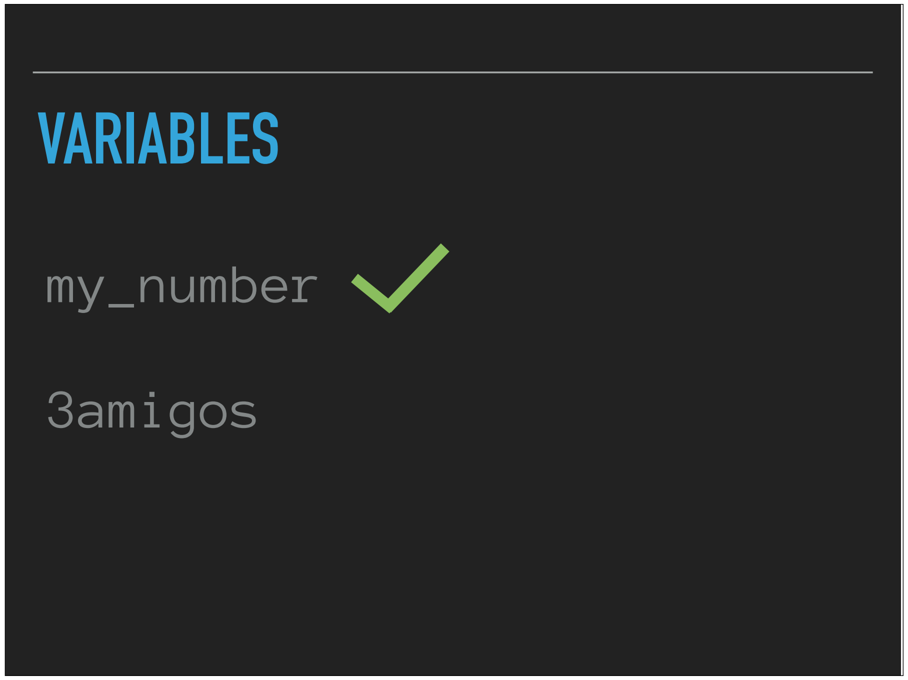
VARIABLES

`my_number`



VARIABLES

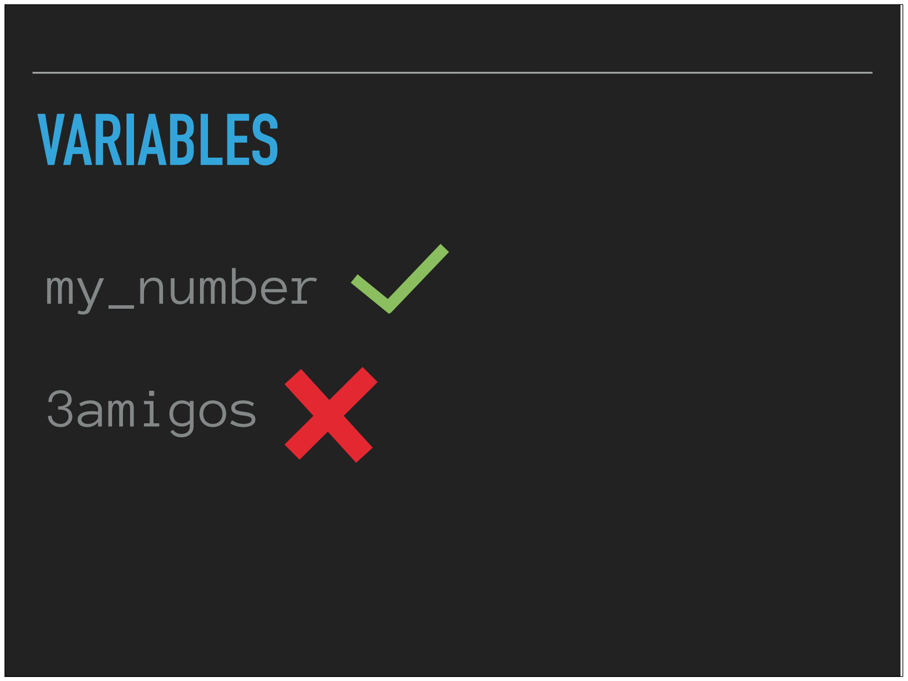
`my_number` ✓



VARIABLES

`my_number` ✓

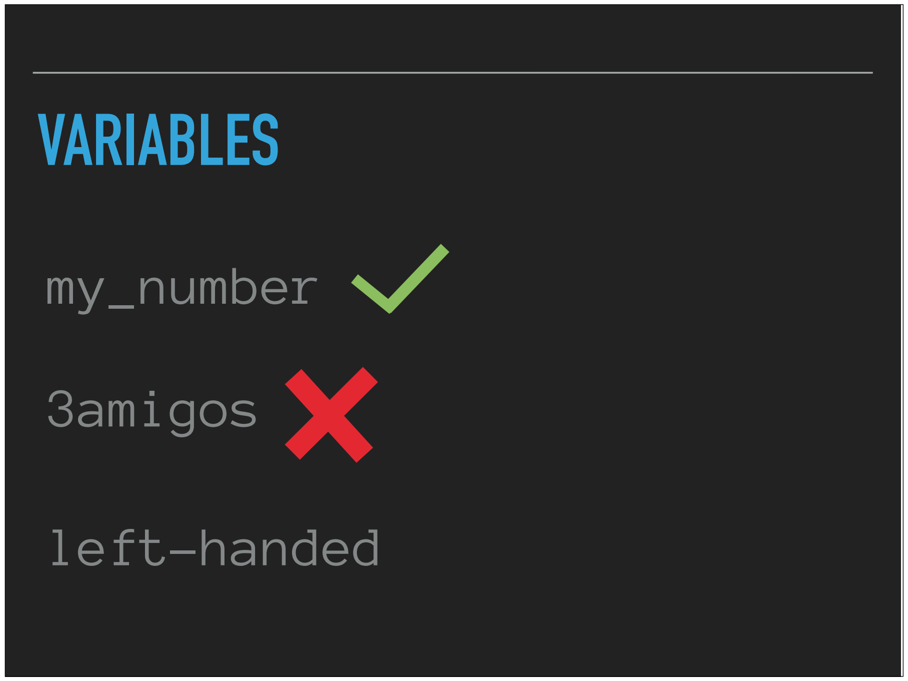
`3amigos`



VARIABLES

`my_number` ✓

`3amigos` ✗

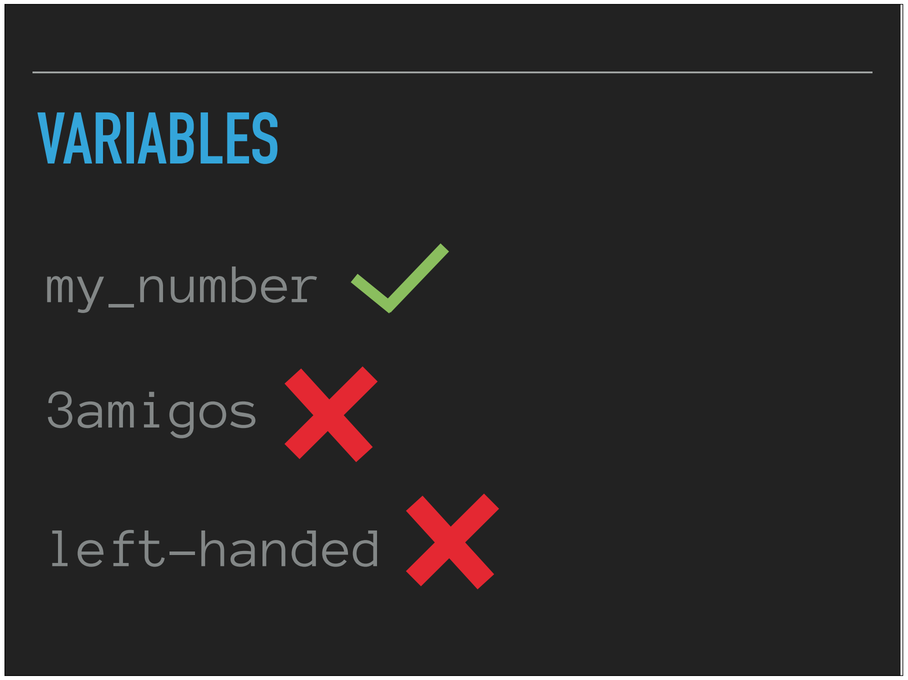


VARIABLES

`my_number` ✓

`3amigos` ✗

`left-handed`

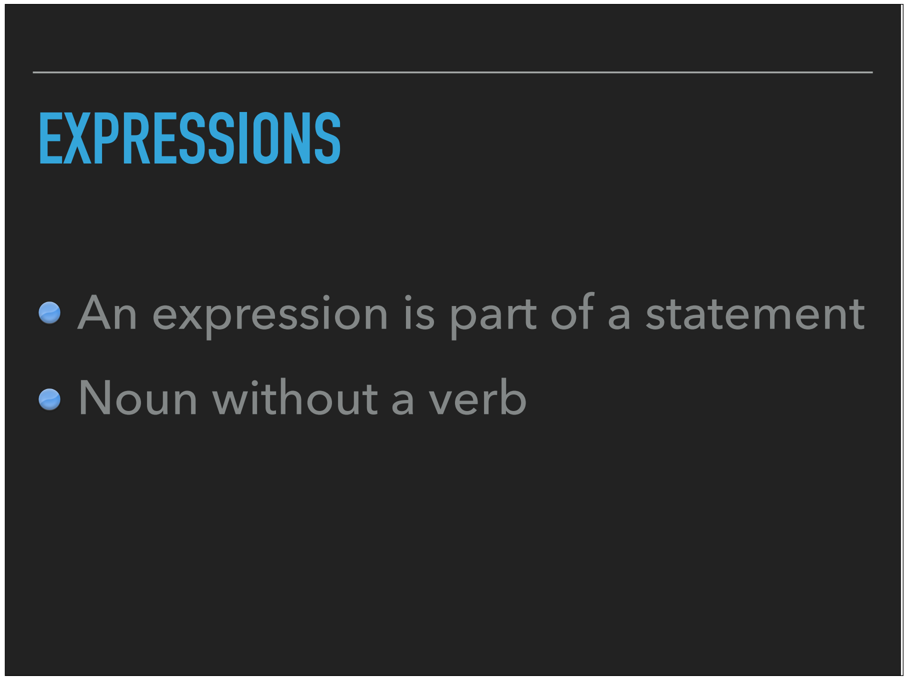


VARIABLES

`my_number` ✓

`3amigos` ✗

`left-handed` ✗



EXPRESSIONS

- An expression is part of a statement
- Noun without a verb

EXPRESSION EXAMPLES

- Variables (like fav_number)
- Values (like 3)

PYTHON THINKS LIKE YOUR CALCULATOR

- If you enter 7 + 9, this is still an expression.
- Equivalent to 16.

EXPRESSION VS STATEMENT

- Equal to a single value –> expression

EXPRESSION VS STATEMENT

- Equal to a single value –> expression
- Important to know when you're giving an order vs noun

WHY DO I CARE ABOUT CALCULATORS?

- Compare to Excel formulas
- Find compound interest, etc

OLD FRIENDS

- Integers (called “int”)
- Decimals (called “float”)

OLD FRIENDS

- Addition (+)
- Multiplication (*)
- Division (/)
- Exponents (**)

A NEW FRIEND

- Modulo (%)
- Finds the remainder

CONCEPTS COVERED SO FAR

- Statement
- Expression
- Variable
- Int, Float

EXERCISES

NEW DATA TYPE: STRING

- Text wrapped in quotes
- Only meaningful to humans

STRING EXAMPLE

```
greeting = “hello world!”
```

STRING EXAMPLE

```
greeting = “hello world!”
```

value

STRING EXAMPLE

```
greeting = “hello world!”
```

(string literal)

what they hear
blah blah GINGER blah blah blah blah blah blah blah blah GINGER blah blah blah blah blah...
Larson

INDEXING SYNTAX

English:

get the first letter in my_string

INDEXING SYNTAX

```
my_string[0]
```

INDEXING SYNTAX

English translation:

get the seventh letter in my_string

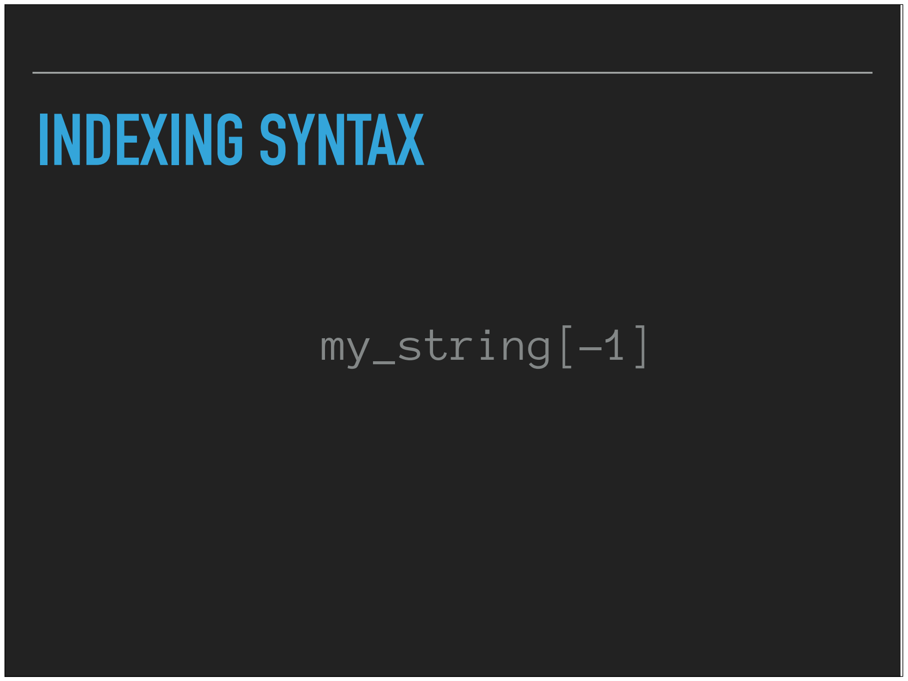
INDEXING SYNTAX

```
my_string[6]
```

INDEXING SYNTAX

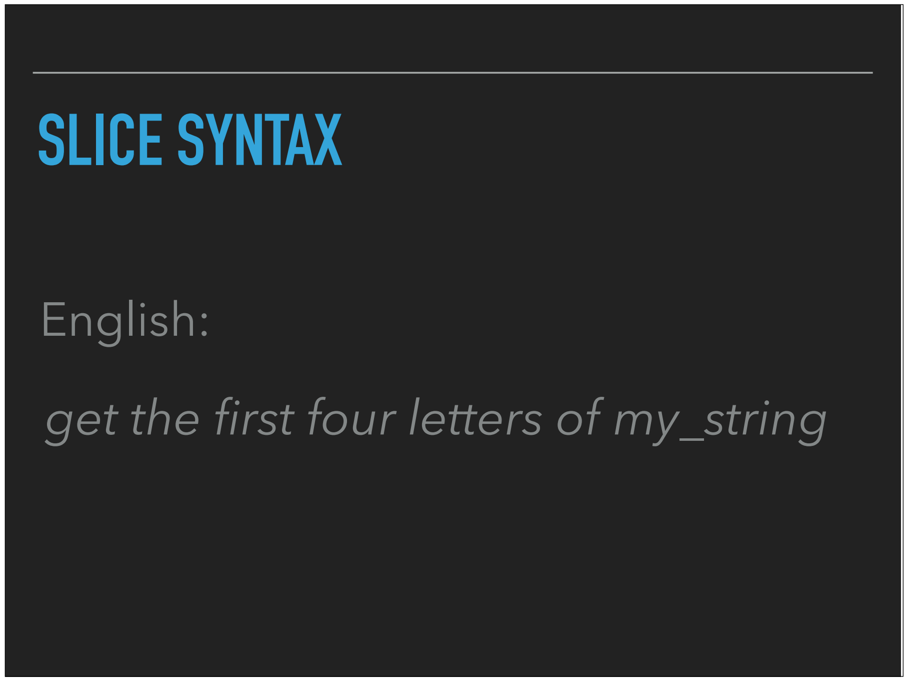
English:

get the last letter in my_string



INDEXING SYNTAX

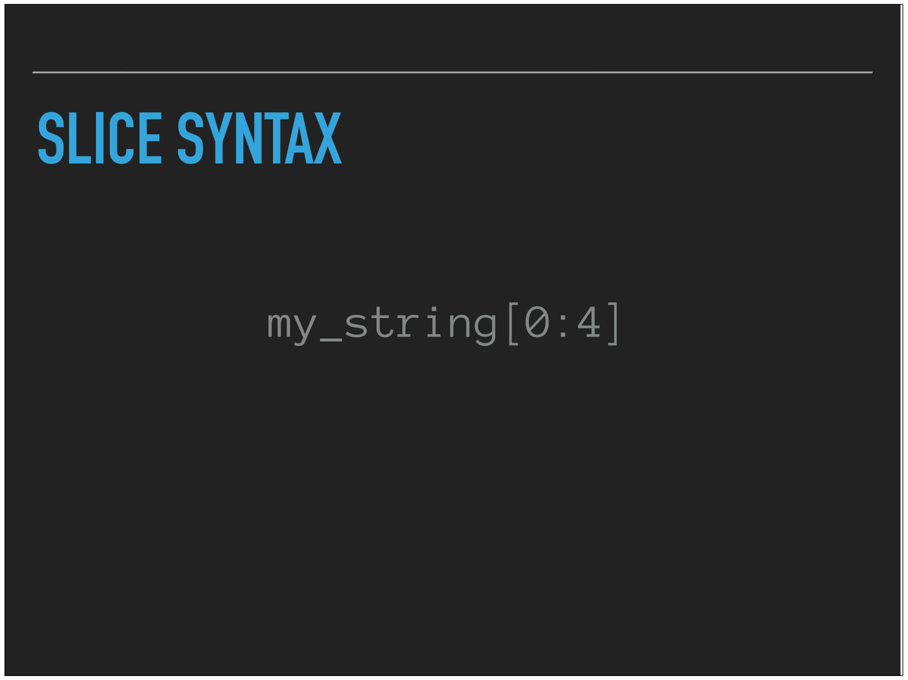
```
my_string[-1]
```



SLICE SYNTAX

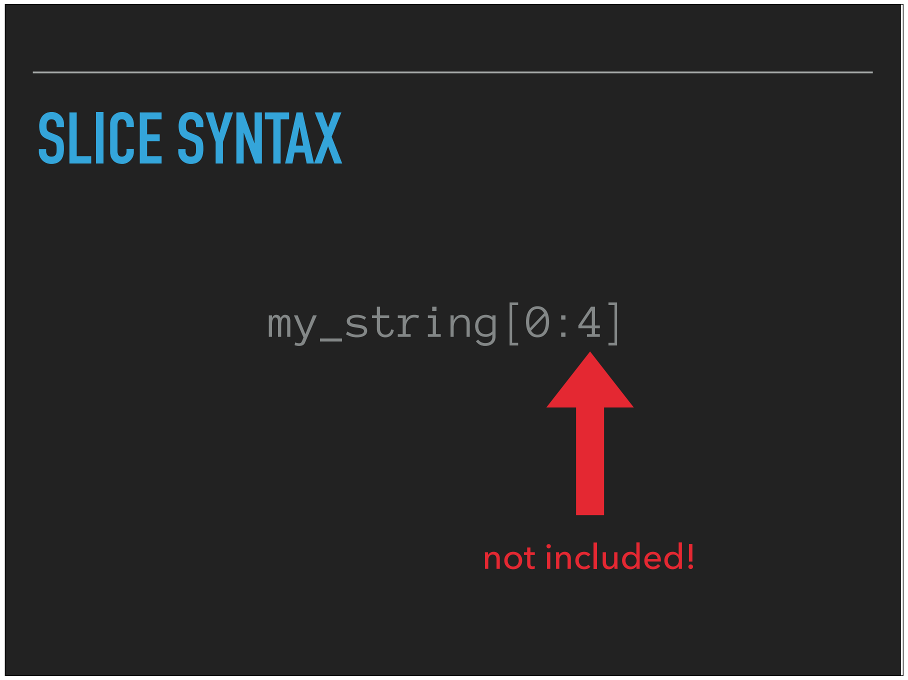
English:

get the first four letters of my_string



SLICE SYNTAX

```
my_string[0:4]
```



SLICE SYNTAX

```
my_string[0:4]
```

not included!

SLICE SYNTAX

English:

get all the letters up to the third

SLICE SYNTAX

```
my_string[:3]
```

SLICE SYNTAX

English:

grab from the third element through until the end

SLICE SYNTAX

```
my_string[2:]
```

SLICE: STEP

```
[start_index:end_index]
```

SLICE: STEP

```
[start_index:end_index:step]
```

SLICE: STEP

```
my_string[::2]
```

SLICE: STEP

```
my_string[::2]
```

→ even # items in list

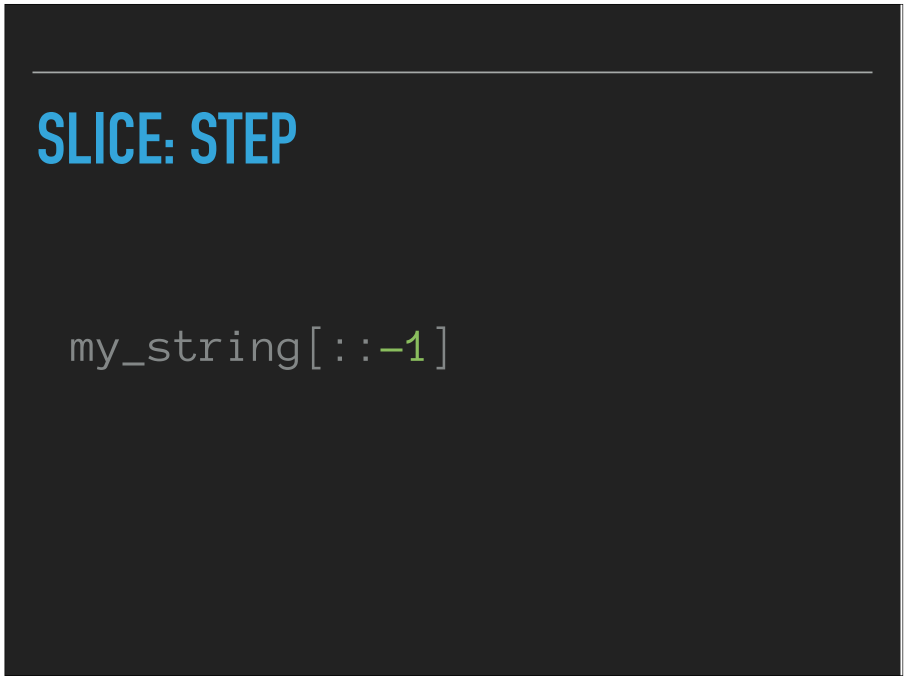
SLICE: STEP

```
my_string[1::2]
```

SLICE: STEP

```
my_string[1::2]
```

→ odd # items in list



SLICE: STEP

```
my_string[::-1]
```

SLICE: STEP

```
my_string[::-1]
```

→ reverse list

EXERCISES

VALUES: WHAT KINDS ARE THERE?

- Int
- Float
- String

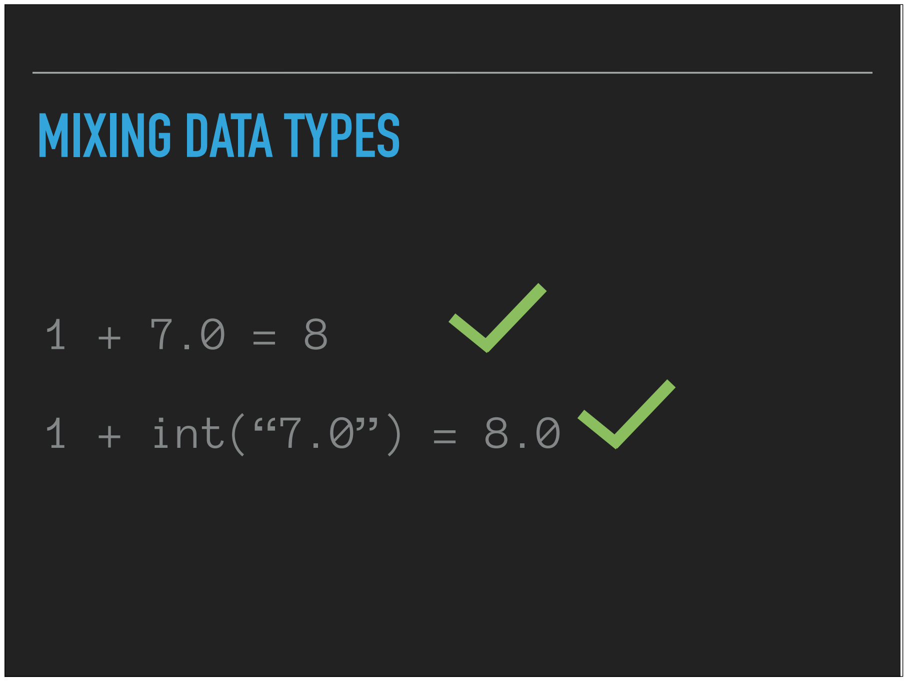
MIXING DATA TYPES

```
1 + 7.0 = 8.0
```
✓

MIXING DATA TYPES

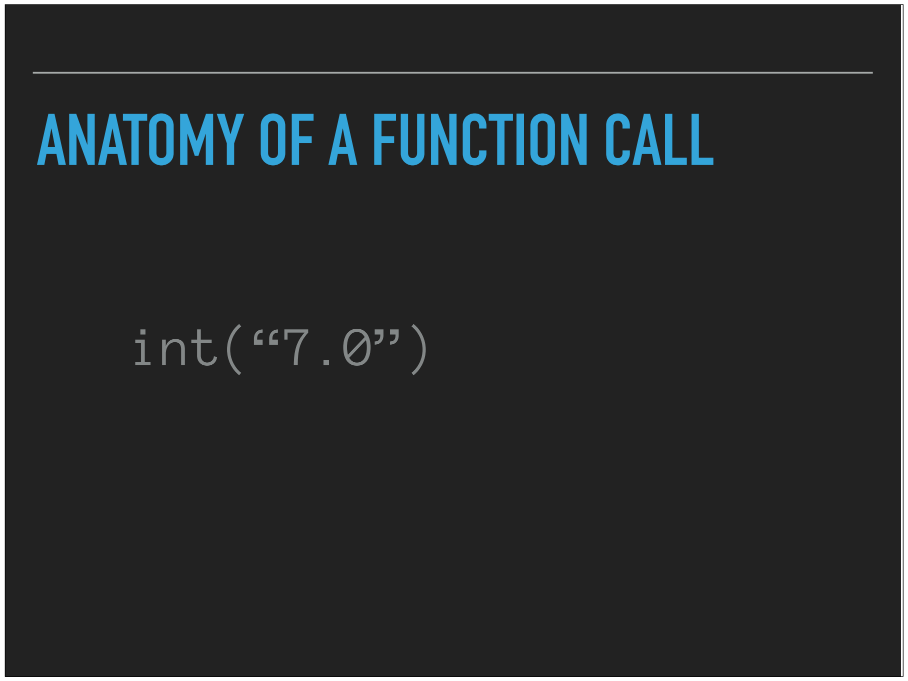
```
1 + 7.0 = 8.0
```
✓

```
1 + “7.0” = error
```
✗



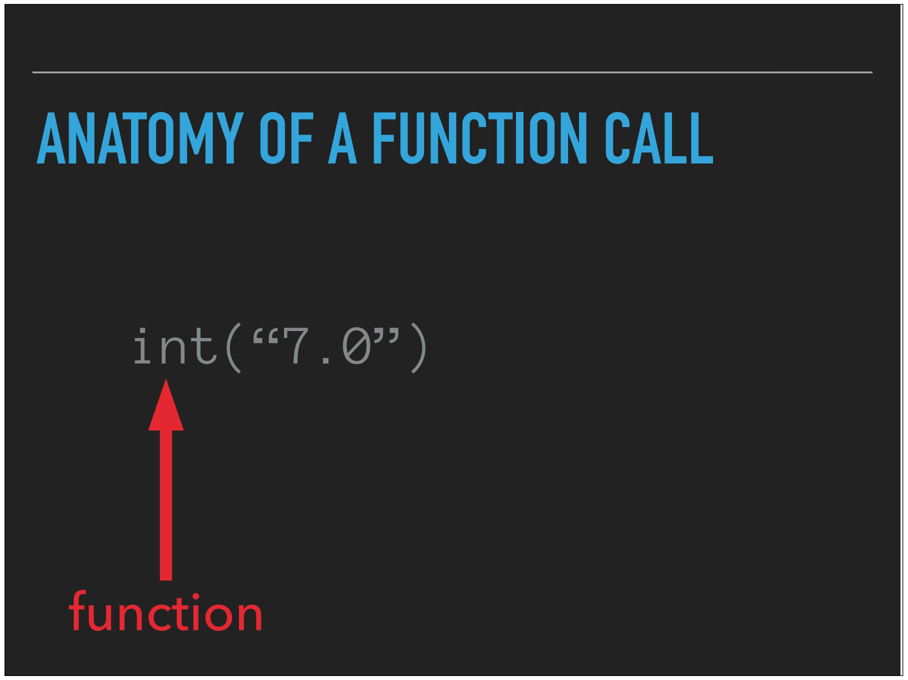
MIXING DATA TYPES

```
1 + 7.0 = 8  ✓
1 + int("7.0") = 8.0 ✓
```



ANATOMY OF A FUNCTION CALL

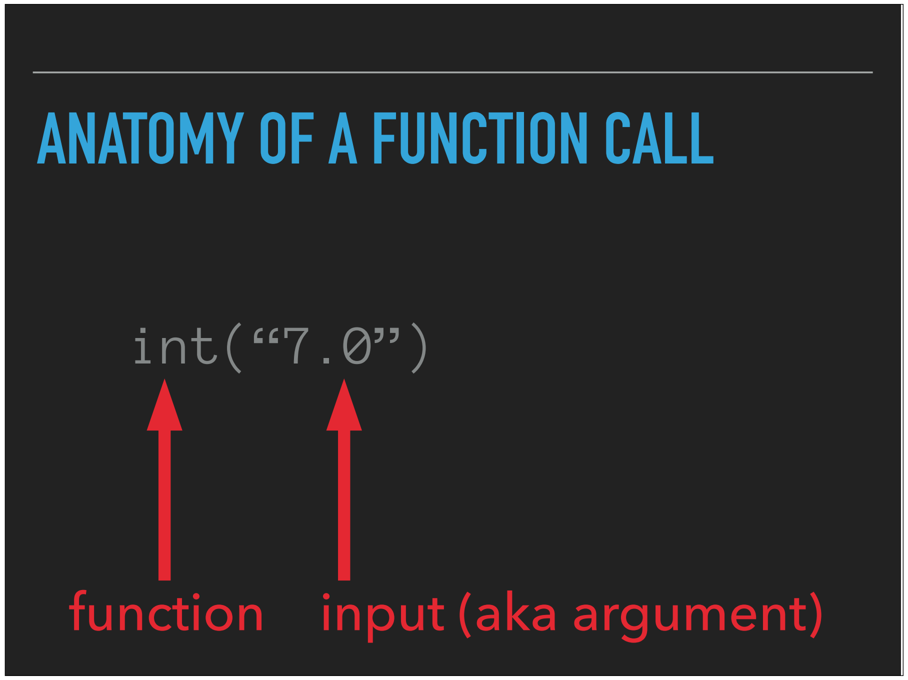
```
int("7.0")
```



ANATOMY OF A FUNCTION CALL

```
int("7.0")
```

function



ANATOMY OF A FUNCTION CALL

```
int("7.0")
```

function input (aka argument)

FUNCTION

- Performs an action
- Action in example: cast to integer

FUNCTION VS METHOD

- Same: performs an "action" with input
- Different: where input comes from

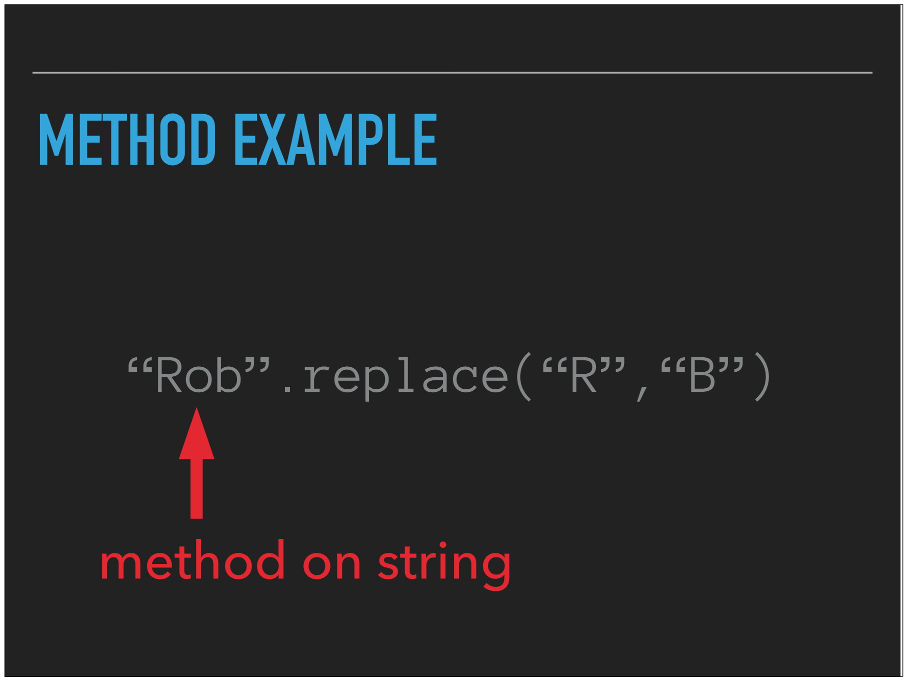
METHOD EXAMPLE

```
"Rob".replace("R","B")
```

METHOD EXAMPLE

```
"Rob".replace("R","B")
```

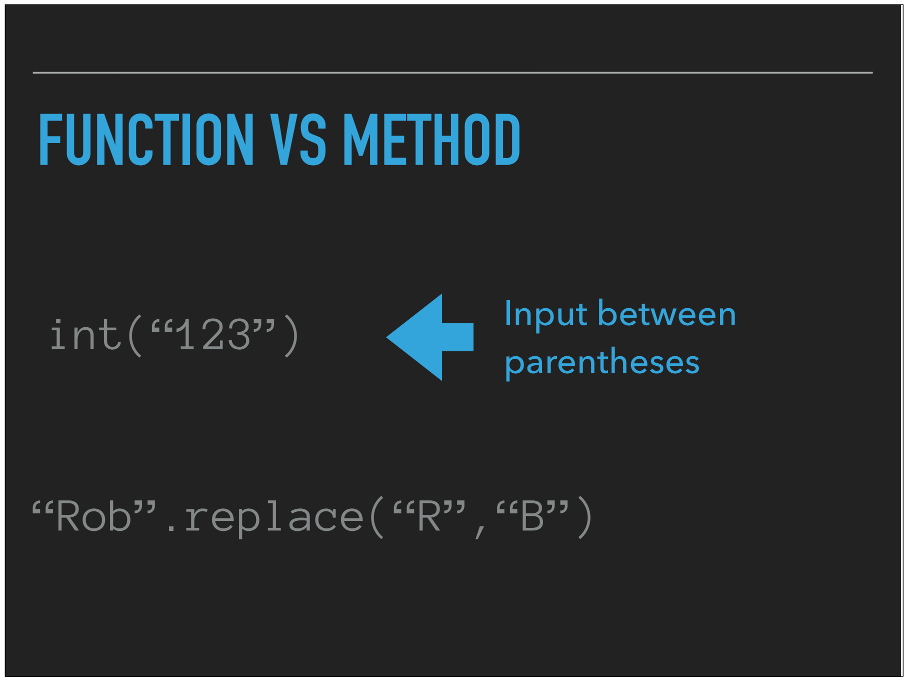
$\longrightarrow$ "Bob"



METHOD EXAMPLE

```
“Rob”.replace(“R”,“B”)
```

method on string

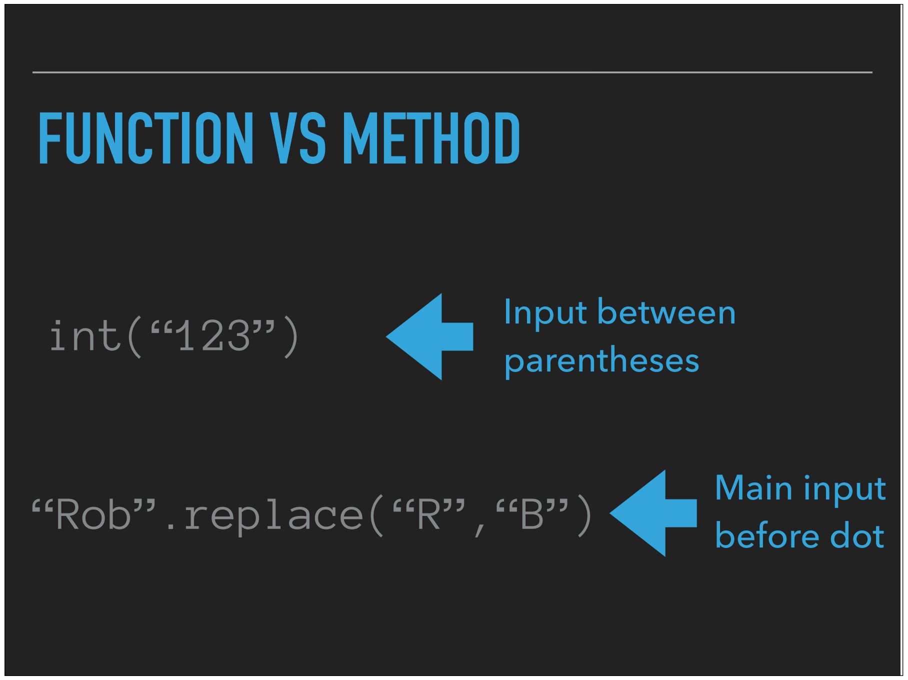


FUNCTION VS METHOD

```
int(“123”)
```

Input between parentheses

```
“Rob”.replace(“R”,“B”)
```



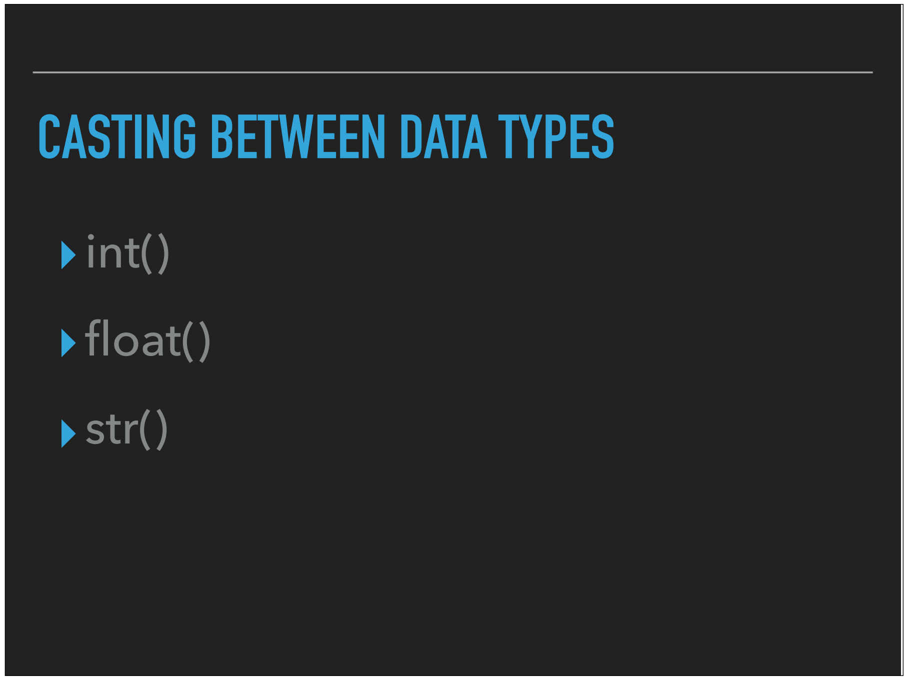
FUNCTION VS METHOD

```
int(“123”)
```

Input between parentheses

```
“Rob”.replace(“R”,“B”)
```

Main input before dot



CASTING BETWEEN DATA TYPES

- int()
- float()
- str()

CONCEPTS COVERED SO FAR

- Function
- Method
- Casting

EXERCISES