

Lecture 6:

# MATHEMATICAL OPERATORS AND FORMATTING

---

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

# Announcements 1/2

**::MassMutual**



Information Session  
Weds September  
19<sup>th</sup> 5pm to 6pm  
CC103/104

**Data Science  
Development Program**

# Announcements 2/2

FROM CLASSROOM TO CONFERENCE ROOM:

## THE NUTS AND BOLTS OF A DATA SCIENCE CAREER



**Emily Dodwell (Smith '11)** a Senior Inventive Scientist in the Data Science and AI Research organization at AT&T Labs, will provide suggestions for pursuing a career in data science, informed by lessons she learned along the way. She will highlight steps that undergraduate students can take to build a professional network, tips to evaluate and apply for data science jobs, and considerations to translate coursework to an industry environment. She will also introduce a current project at AT&T Labs to demonstrate a data science workflow and the importance of ongoing professional development in the field.

Monday September 24, 12:10-1:10 pm Ford Hall 240.

**Pizza lunch will be served. Sponsored by Smith College SDS program**

# About the assignment rubric

- Assignments are generally worth 10 points in this course
- For assignment 2, these points fall into 3 categories:

## *Style points (3)*

The submission:

- includes a header including both partners' names, the filename, and a description of the program
- uses appropriate, informative variable names
- runs without syntax errors

# About the assignment rubric

- Assignments are generally worth 10 points in this course
- For assignment 2, these points fall into 3 categories:

## *First level (5 points)*

The calculator correctly performs the following mathematical operations:

- addition
- subtraction
- multiplication
- division
- exponentiation

# About the assignment rubric

- Assignments are generally worth 10 points in this course
- For assignment 2, these points fall into 3 categories:

## *Second level (2 points)*

The calculator correctly parses the user's input, including:

- successful interpretation of the two numeric values
- correctly parsing the selected mathematical operation, differentiating between integers and floats when appropriate

# About the assignment rubric

- Assignments are generally worth 10 points in this course
- For assignment 2, these points fall into 3 categories
- Recommendation: **go in order**
  - Start with first level and style points
  - *Then* go on to second level points
  - And *then* attempt bonus points (if applicable)

# Overview of the week

- ✓ More about variables
- ✓ Numeric values and basic operations
- More mathematical operators
  - Revisiting ints and floats
  - The `math` module
  - Formatting
- Lab: Cash Machine
- Conditional (“if”) statements

# (RECAP) Core concept 2: numeric values

- Two kinds of **numbers** in CS:
  - integers (“whole numbers”)
  - floats (“decimals” or “floating point numbers”)
- **Basic operators:**
  - addition: `+`
  - subtraction: `-`
  - multiplication: `*`
  - division: `/`
  - integer division: `//`
  - exponentiation: `**` (power)
  - modular arithmetic: `%` (modulo)

# (RECAP) Core concept 2: numeric values

- Two kinds of **numbers** in CS:
  - integers (“whole numbers”)
  - floats (“decimals” or “floating point numbers”)
- **Basic operators:**
  - addition: `+`
  - subtraction: `-`
  - multiplication: `*`
  - division: `/`
  - integer division: `//`
  - exponentiation: `**` (power)
  - modular arithmetic: `%` (modulo)

# Reviewing integer operators: // and %

What is the result of the following operations?

21 // 5	# 4
21 % 5	# 1
9 // 3	# 3
9 % 3	# 0
13 // 5	# 2
13 % 5	# 3
139 // 20	# 6
139 % 20	# 19

# Built-in functions that work on numbers

- `abs(x)` # return the absolute value of x
- `float(x)` # return x parsed as a float
- `int(x)` # return x parsed as an int
- `max(...)` # return the largest of a list of numbers
- `min(...)` # return the smallest of a list of numbers
- `round(x[, n])` # return x rounded to  $n$  digits after the  
# decimal point. If  $n$  is omitted, it  
# returns the nearest integer value
- `sum(...)` # return the sum of a list of numbers

# Aside: what does **parsed** mean?

- `abs(x)` # return the absolute value of x
- `float(x)` # return x **parsed** as a float
- `int(x)` # return x **parsed** as an int
- `max(...)` # return the largest of a list of numbers
- `min(...)` # return the smallest of a list of numbers
- `round(x[, n])` # return x rounded to  $n$  digits after the  
# decimal point. If  $n$  is omitted, it  
# returns the nearest integer value
- `sum(...)` # return the sum of a list of numbers

# Aside: what does **return** mean?

- `abs(x)` # **return** the absolute value of `x`
- `float(x)` # **return** `x` parsed as a float
- `int(x)` # **return** `x` parsed as an int
- `max(...)` # **return** the largest of a list of numbers
- `min(...)` # **return** the smallest of a list of numbers
- `round(x[, n])` # **return** `x` rounded to `n` digits after the  
# decimal point. If `n` is omitted, it  
# returns the nearest integer value
- `sum(...)` # **return** the sum of a list of numbers

# RECAP: Keywords

- Some words in Python\* are reserved as keywords, and cannot be used as a variable name:

and as assert break class continue def del elif else  
except exec finally for from global if import in is lambda  
not or pass raise **return** try while with yield

 a reserved keyword

- We saw a few in lab yesterday:



lab1.py - /Users/jcrouser/Google Drive/Teaching/C...  
for name in [ "Aleah", "Belle", "Chenwei" ]:  
 print( name )  
 print( '-' \* len( name ) )

Ln: 3 Col: 0

\* other languages have their own set of reserved words

# Foreshadowing: “functions”

```
● ● ● demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

y = do_something()
print(y) # y = 5
```

Ln: 9 Col: 16

# The math module

- Lots of other things we might want to do with numerical values are available as functions in the **math** module

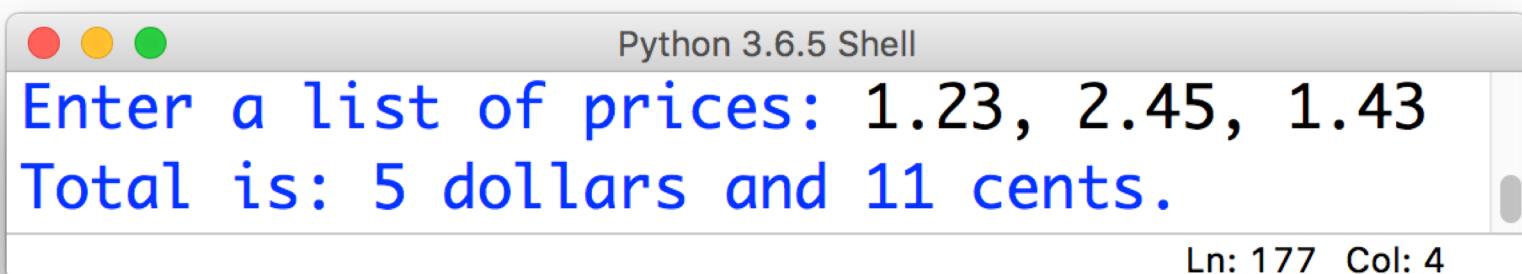
- In Python, modules are just files containing Python definitions and statements (ex. **name.py**)
- These can be imported using **import name**
- To access **name**'s functions, type **name.function()**

- **import math**
  - **math.floor(f)** # round float f down
  - **math.ceil(f)** # round float f up
  - **math.sqrt(x)** # take the square root of x

And more! Check out: <https://docs.python.org/2/library/math.html>

# 15-minute exercise: dollars and cents

Use **built-in functions** and functions from the **math module** to take a list of prices, calculate their sum, and output their total formatted like this:

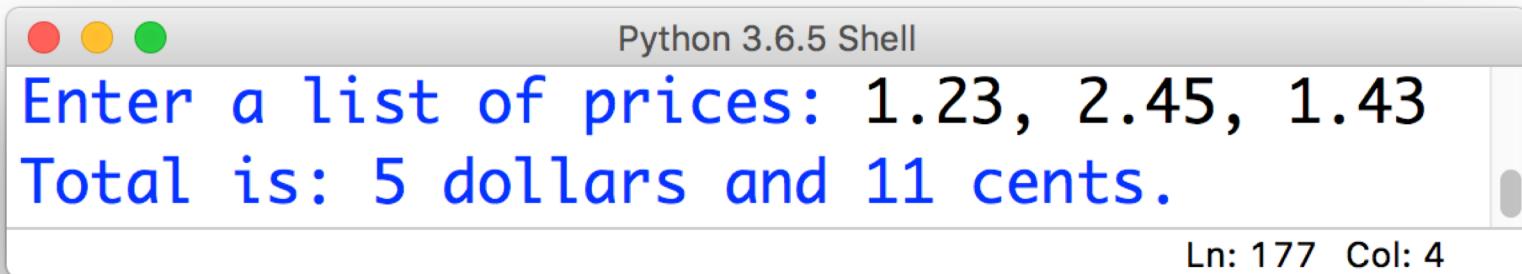


The screenshot shows a Mac OS X style window titled "Python 3.6.5 Shell". Inside the window, the text "Enter a list of prices: 1.23, 2.45, 1.43" is displayed in blue, followed by "Total is: 5 dollars and 11 cents." also in blue. In the bottom right corner of the window, there is a status bar with the text "Ln: 177 Col: 4".

```
Python 3.6.5 Shell
Enter a list of prices: 1.23, 2.45, 1.43
Total is: 5 dollars and 11 cents.
Ln: 177 Col: 4
```

# Finishing touches...

- What we have now:

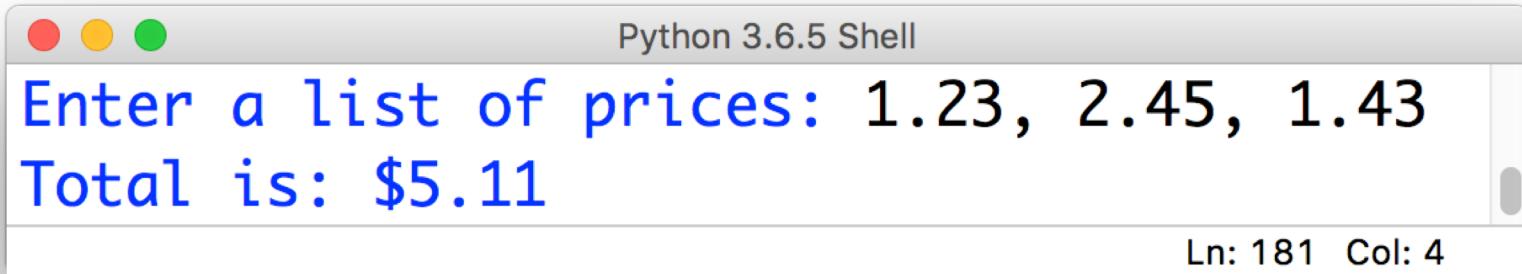


Python 3.6.5 Shell

```
Enter a list of prices: 1.23, 2.45, 1.43
Total is: 5 dollars and 11 cents.
```

Ln: 177 Col: 4

- What we probably want:



Python 3.6.5 Shell

```
Enter a list of prices: 1.23, 2.45, 1.43
Total is: $5.11
```

Ln: 181 Col: 4

# Discussion

Ideas?

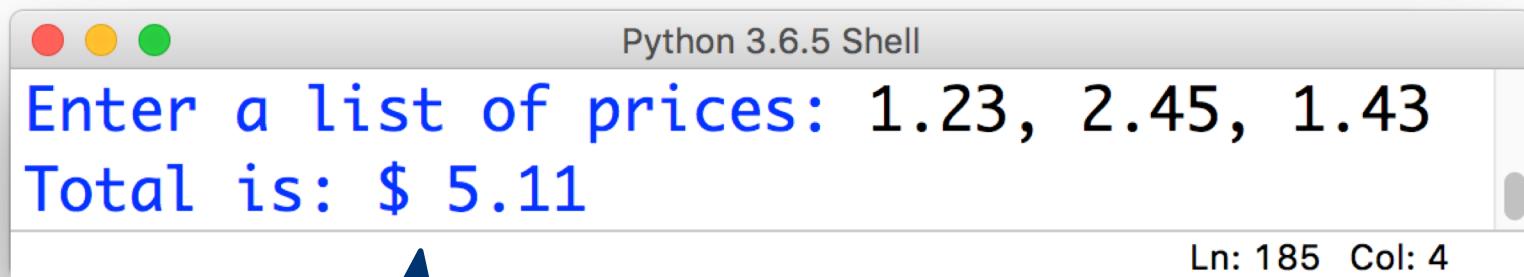
What tools do you have?



# Close, but not quite:

- Just using concatenation:

```
print("Total is: $", sum(x))
```

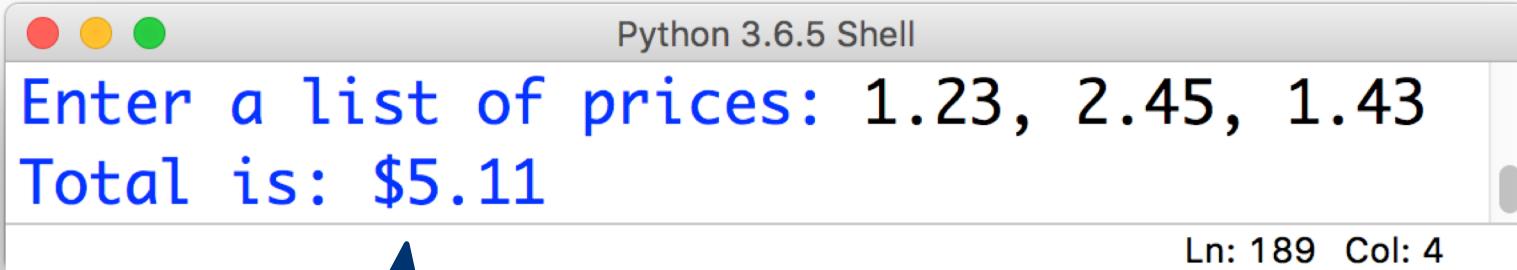


annoying space

# Closer, but unsatisfying (and fragile)

- Using concatenation and casting to string:

```
print("Total is: $", str(sum(x)))
```



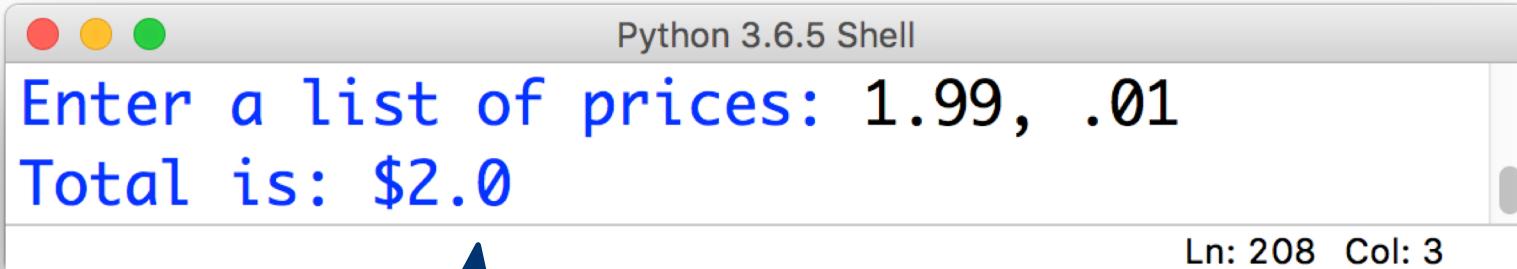
The screenshot shows a Mac OS X window titled "Python 3.6.5 Shell". Inside, the text "Enter a list of prices: 1.23, 2.45, 1.43" is displayed in blue, followed by "Total is: \$5.11" in black. In the bottom right corner, it says "Ln: 189 Col: 4". A blue arrow points from the text "no annoying space" below to the space character between "\$" and "5.11" in the output.

no annoying space

# Closer, but unsatisfying (and fragile)

- Using concatenation and casting to string:

```
print("Total is: $", str(sum(x)))
```



A screenshot of a Mac OS X window titled "Python 3.6.5 Shell". The window contains the following text:  
Enter a list of prices: 1.99, .01  
Total is: \$2.0

The output "Total is: \$2.0" is incorrect because it only displays one decimal place, while the input contained two decimal places. A blue arrow points from the text "wrong number of decimal places" to the output line.

wrong number  
of decimal places

# Solution: formatting with `.format()`

- The `.format()` method (which gets called on a `string`) might be helpful here!
- How it works:

The diagram illustrates the process of string formatting. A blue curved arrow labeled "insert" points from the placeholder `{0:10}` in the code to the corresponding output in the result. Another blue curved arrow labeled "print (at least)" points from the `print` statement in the code to the resulting printed string.

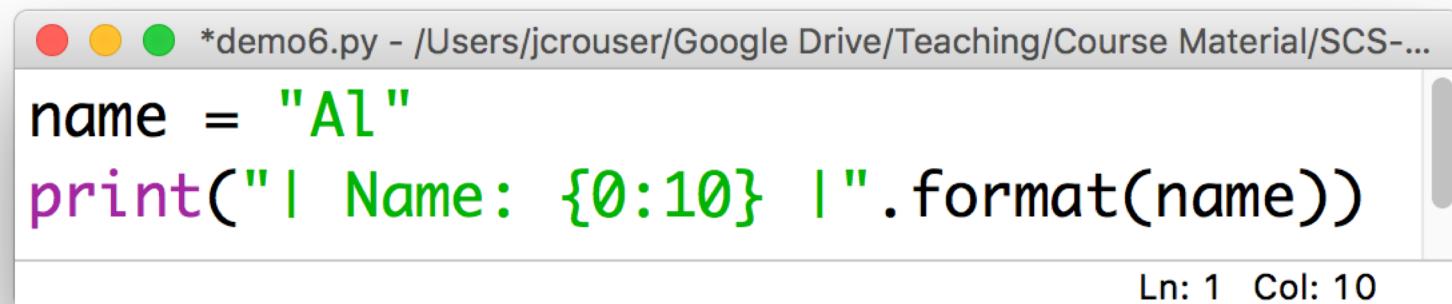
```
*demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-...
name = "Jordan"
print("| Name: {0:10} |".format(name))
```

Result: | Name: Jordan |

Ln: 1 Col: 16

# Solution: formatting with `.format()`

- The `.format()` method (which gets called on a `string`) might be helpful here!
- How it works:



A screenshot of a terminal window titled "\*demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-...". The window contains the following Python code:

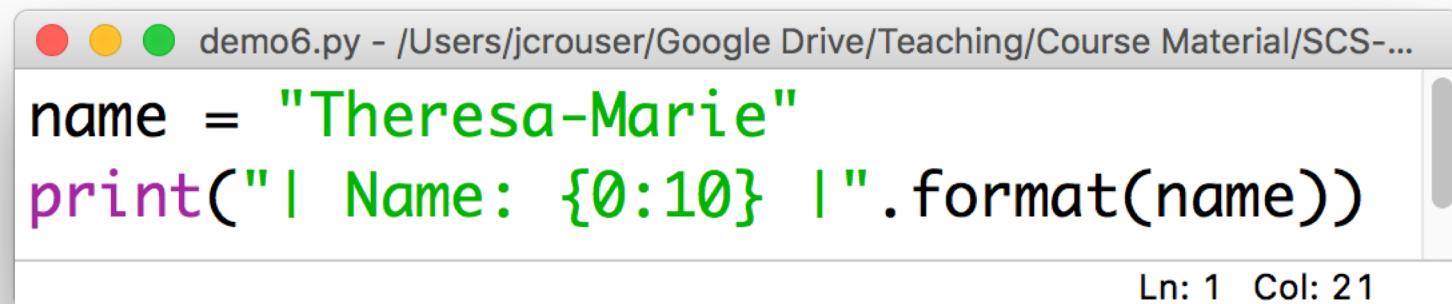
```
name = "Al"
print("| Name: {0:10} |".format(name))
```

The terminal shows the command prompt and the output of the print statement. The status bar at the bottom right indicates "Ln: 1 Col: 10".

**Result:** | Name: Al |

# Solution: formatting with `.format()`

- The `.format()` method (which gets called on a `string`) might be helpful here!
- How it works:



A screenshot of a terminal window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-...". The window contains the following Python code:

```
name = "Theresa-Marie"
print("| Name: {0:10} |".format(name))
```

The code defines a variable `name` with the value "Theresa-Marie" and prints it using the `format` method with a width of 10. The terminal shows the output "Ln: 1 Col: 21".

**Result:** | Name: Theresa-Marie |

doesn't truncate 

# Calling `.format()` with multiple inputs

- Can also handle multiple inputs, e.g.

```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan-CSC/labs/demo6.py (3.6.5)
first = "Jordan"
last = "Crouser"
print("| Name: {0:10} {1:10} |".format(first, last))
0th      1st
Ln: 3 Col: 24
```

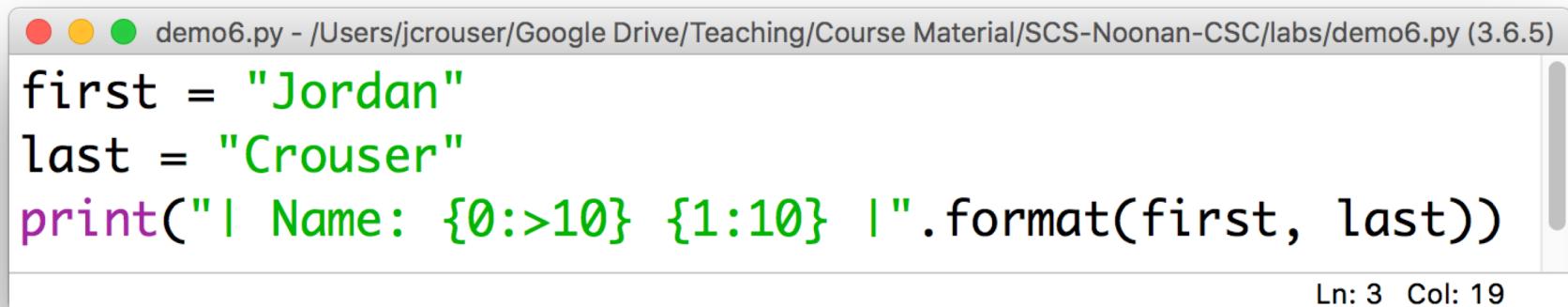
put the  
1<sup>st</sup> thing here

put the  
0<sup>th</sup> thing here

**Result:** | Name: Jordan Crouser |

# Right-justification with `.format()`

- To align the format to the right instead of to the left, use `>`



A screenshot of a terminal window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan-CSC/labs/demo6.py (3.6.5)". The window contains the following Python code:

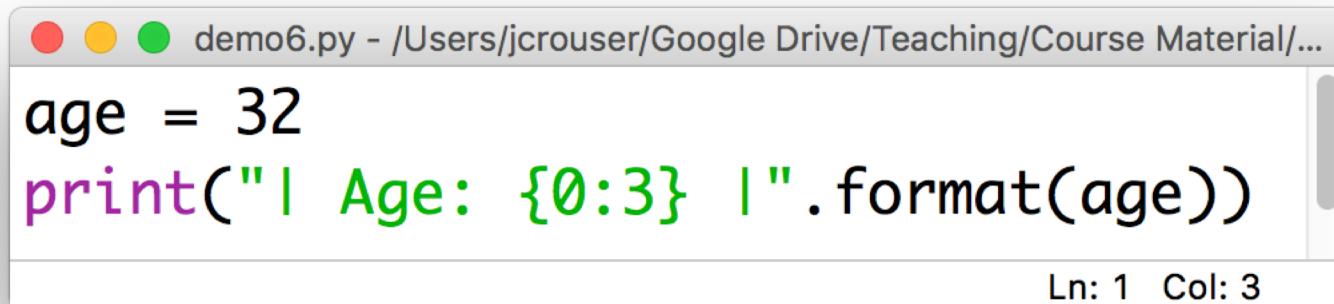
```
first = "Jordan"
last = "Crouser"
print("| Name: {0:>10} {1:10} |".format(first, last))
```

The status bar at the bottom right shows "Ln: 3 Col: 19".

**Result:** | Name: Jordan Crouser |

# .format( ) on integers

- Calling .format( ) on an integer works just like with strings, but they're automatically right-aligned



A screenshot of a Mac OS X terminal window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/...". The window contains the following text:

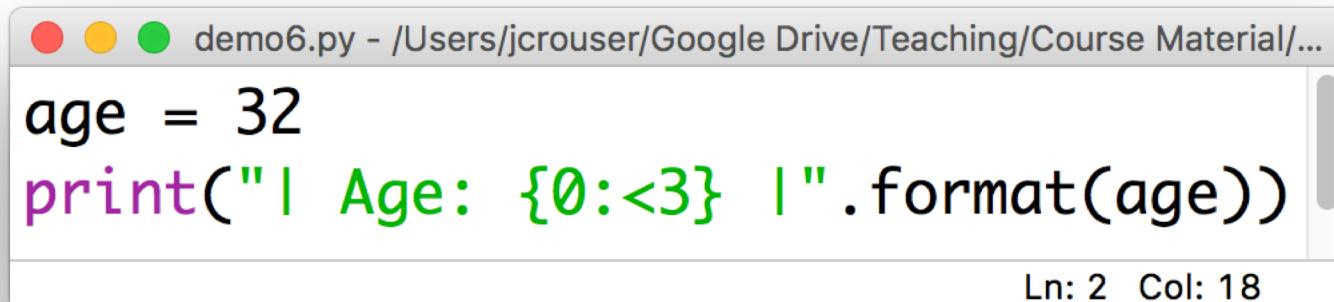
```
age = 32
print("I Age: {0:3} | ".format(age))
```

The cursor is positioned after the closing parenthesis of the print statement. In the bottom right corner of the terminal window, it says "Ln: 1 Col: 3".

**Result:** | Age: 32 |

# .format( ) on integers

- Use < to left-align:



A screenshot of a Mac OS X terminal window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/...". The window contains the following text:

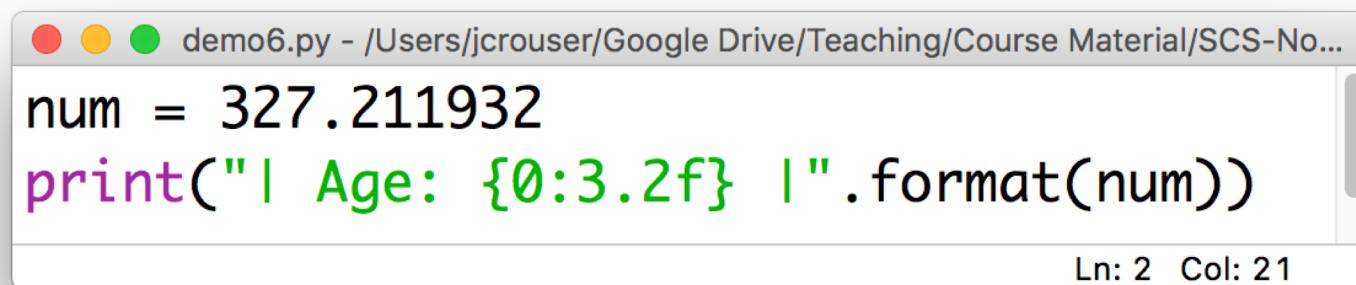
```
age = 32
print("I Age: {0:<3} ".format(age))
```

The status bar at the bottom right shows "Ln: 2 Col: 18".

**Result:** | Age: 32 |

# .format() on floats

- We need to specify a number of digits **before** and **after** the decimal point:



A screenshot of a terminal window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-No...". The window contains the following Python code:

```
num = 327.211932
print("| Age: {0:3.2f} |".format(num))
```

The code defines a variable num with the value 327.211932, and then prints it using the print function and a formatted string. The formatted string uses the placeholder {0:3.2f} to format the number with three digits before the decimal point and two digits after. The output of the program is:

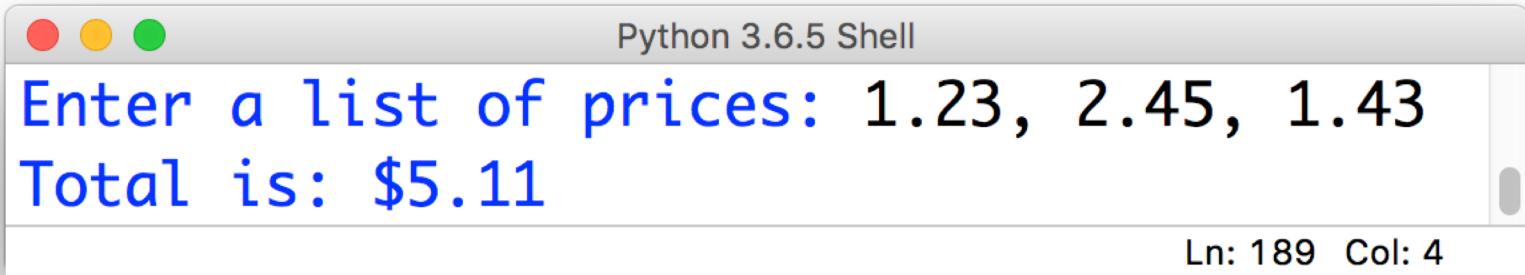
| Age: 327.21 |

Ln: 2 Col: 21

**Result:** | Age: 327.21 |

# Revisiting: dollars and cents

Modify your previous code to use the `.format()` method so that your output looks like this:



A screenshot of a Mac OS X window titled "Python 3.6.5 Shell". The window contains two lines of text: "Enter a list of prices: 1.23, 2.45, 1.43" and "Total is: \$5.11". The status bar at the bottom right shows "Ln: 189 Col: 4".

```
Enter a list of prices: 1.23, 2.45, 1.43
Total is: $5.11
Ln: 189 Col: 4
```

# Overview of the week

- ✓ More about variables
- ✓ Numeric values and basic operations
- ✓ More mathematical operators
  - ✓ Revisiting ints and floats
  - ✓ The `math` module
  - ✓ Formatting
- **Lab: Cash Machine**
- Conditional (“if”) statements

# Lab 2: Cash Machine

The screenshot shows a web browser window with the title bar "Mathematical Operations". The address bar contains the URL <https://jcrouser.github.io/CSC111/labs/lab-2-math.html>. Below the address bar, there is a message: "For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)". On the left side of the page, there is a sidebar with a blue header containing the text "Getting Started: understanding how Floats take over". Below this header are three items: "Cash Machine", "Submission of Lab 2", and "Lab 2". The main content area has a large blue header "Mathematical Operations" and a sub-section titled "Getting Started: understanding how Floats take over". Below this section, there is a text block: "In this opening section, we'll play around to get an understanding of important functions on numbers, and also how floating point numbers (numbers with a decimal point) *take over* expressions when they are present." Another text block below it says: "Open the Python shell and type the expressions below. Try to predict the output of the interpreter before pressing the ENTER key." At the bottom of the main content area, there is a code block:

```
a = 3
x = 1.5
type( a )
type( x )
type( a * x )
type( a * 100 )
type( 1 )
type( 1.0 )
type( "1" )
a / 3
a // 3
type( a/3 )
type( 5/4 )
type( 5//4 )
```