Building complex data objects

Presenter: Steve Baskauf
steve.baskauf@vanderbilt.edu

DISC DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

CodeGraf landing page

- vanderbi.lt/codegraf

List of lists

DISC DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Lists can contain any kind of object

- Lists can also contain other lists:

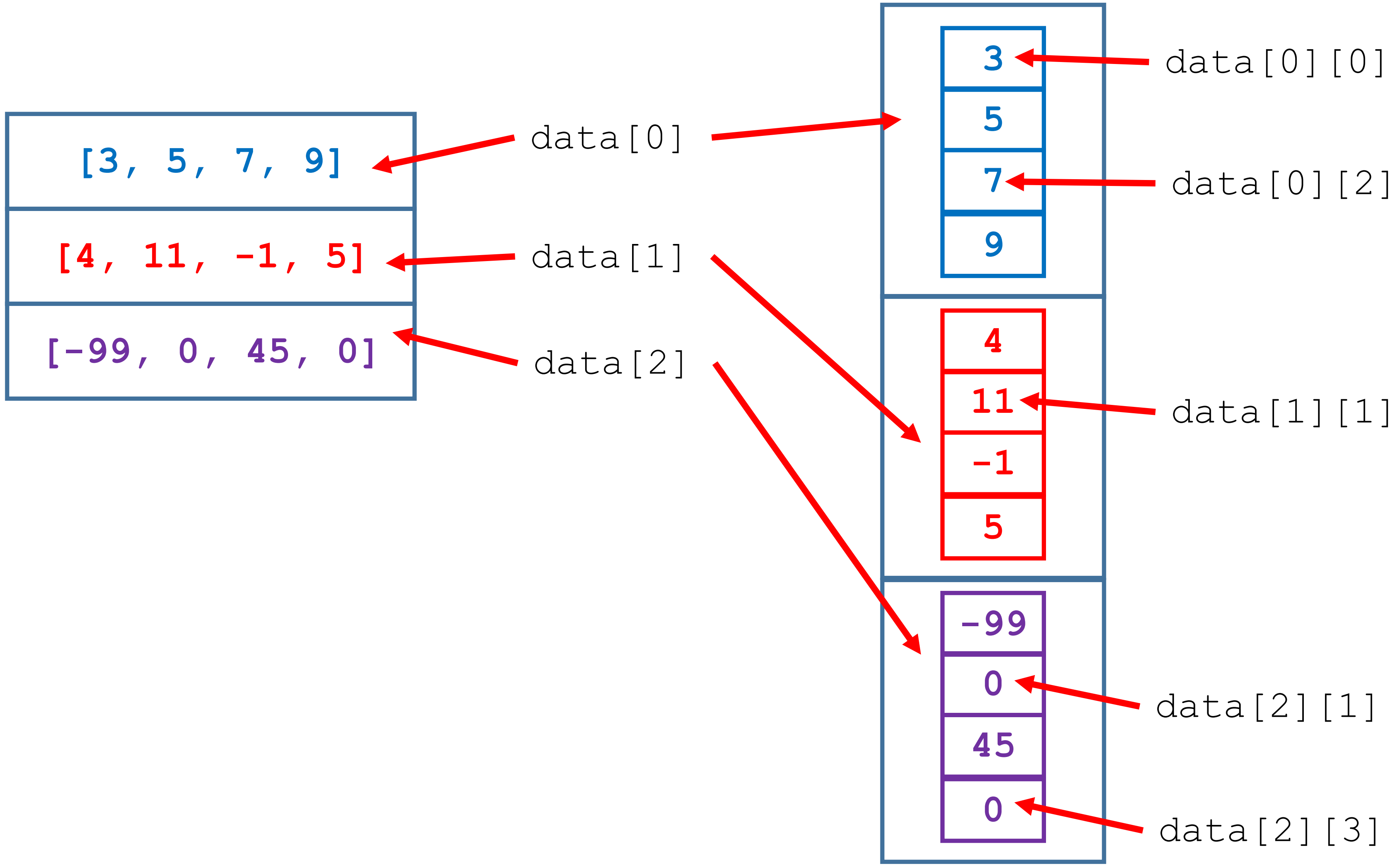
```
first_row = [3, 5, 7, 9]
second_row = [4, 11, -1, 5]
third_row = [-99, 0, 45, 0]
data = [first_row, second_row, third_row]
```

- The inner lists can be nested directly inside the outer list:

```
data = [[3, 5, 7, 9], [4, 11, -1, 5], [-99, 0, 45, 0]]
```

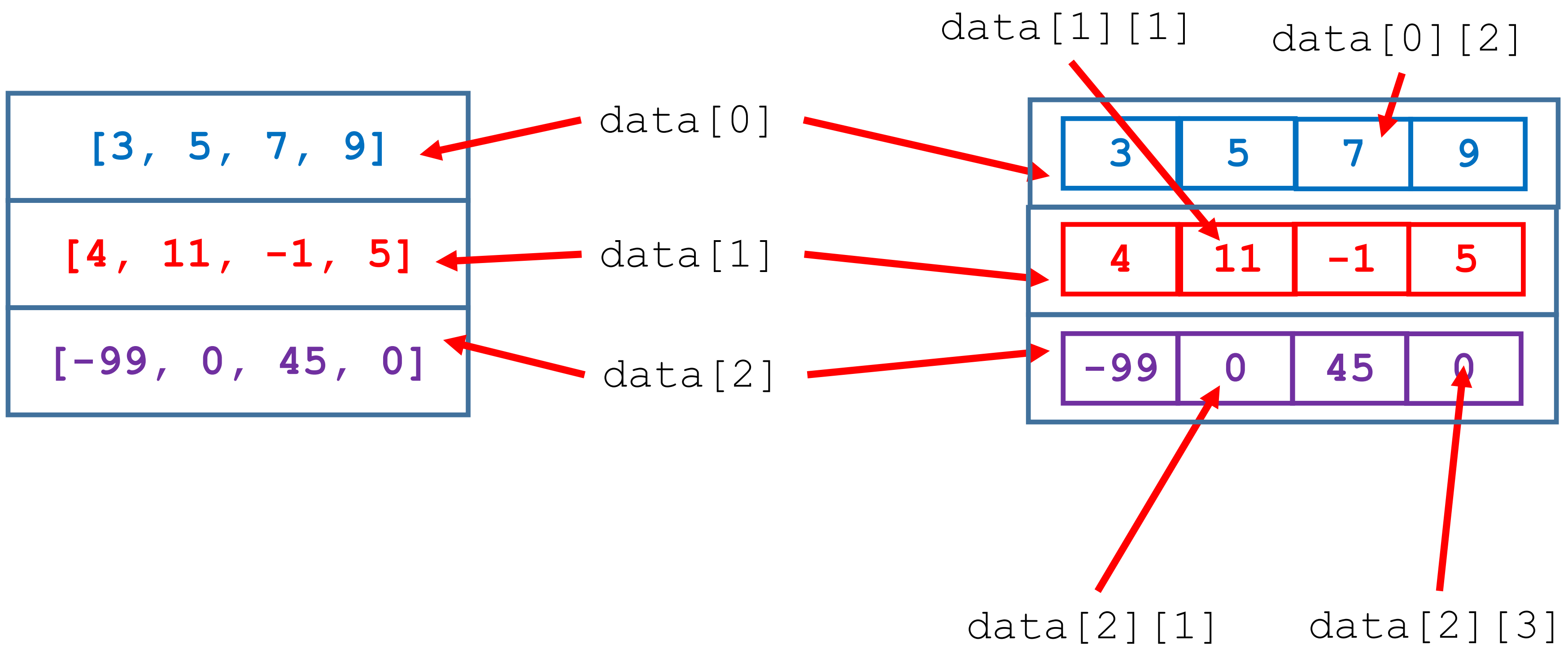
Lists of lists

```
data = [[3, 5, 7, 9], [4, 11, -1, 5], [-99, 0, 45, 0]]
```



Lists of lists

```
data = [[3, 5, 7, 9], [4, 11, -1, 5], [-99, 0, 45, 0]]
```



You can think of this like:

`data[row][column]`

where the indices refer to parts of a table.
A list of lists is similar to an array in other programming languages

Nested for loops

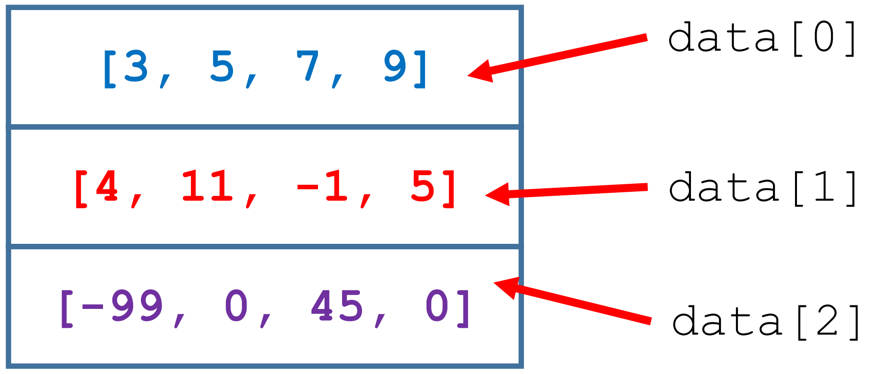
DISC DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Nested **for** loops

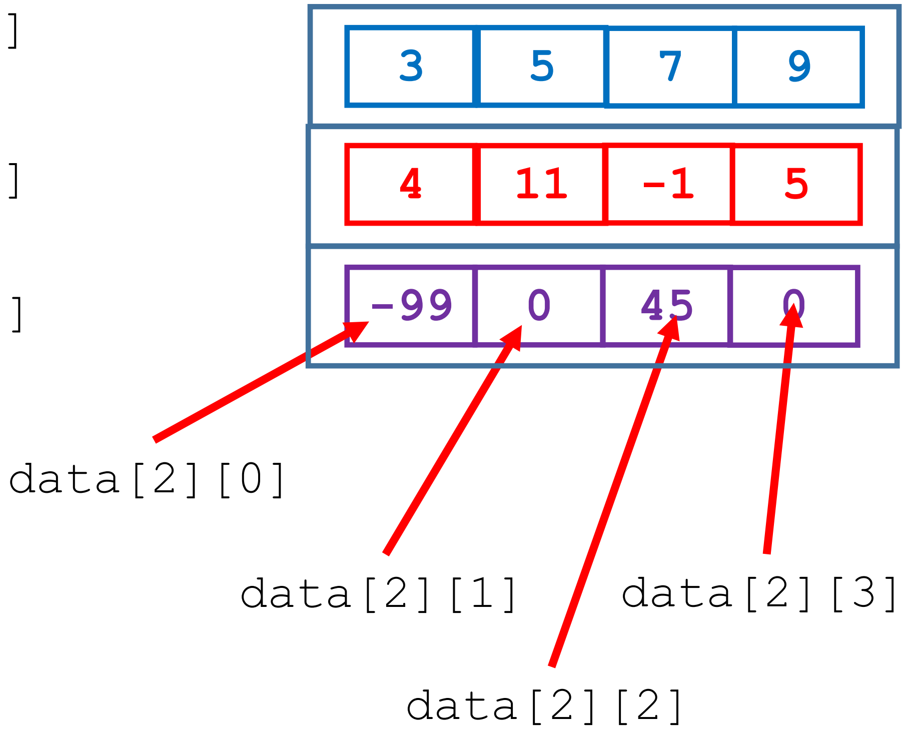
"outer" loop

```
for row in data:
    print(row)
```



nested loops

```
for row in data:
    for column in row:
        print(column)
```



data[row][column]

"outer" structure

"inner" structure

```
for column in row:
    print(column)
```

"inner" loop

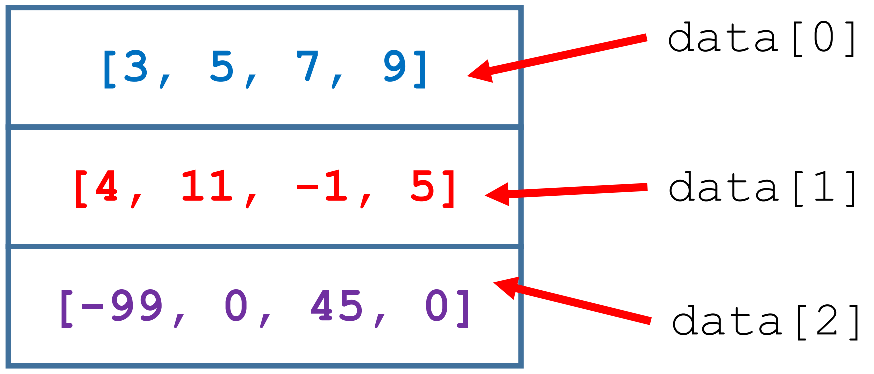
Nested **for** loops using **range()**

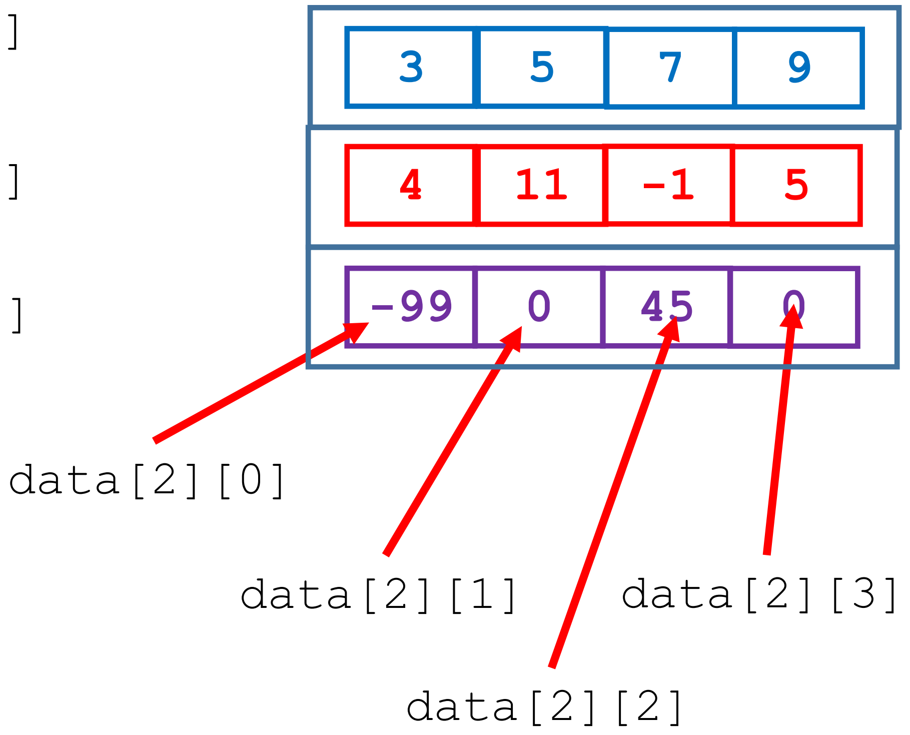
"outer" loop

```
for row in range(len(data)):
    print(row)
    print(data[row])
```

nested loops

```
for row in range(len(data)):
    print(row)
    for column in range(len(data[row])):
        print(data[row][column])
```





`data[row][column]`

"outer" structure

"inner" structure

```
for column in range(len(data[2])):
    print(data[2][column])
```

"inner" loop

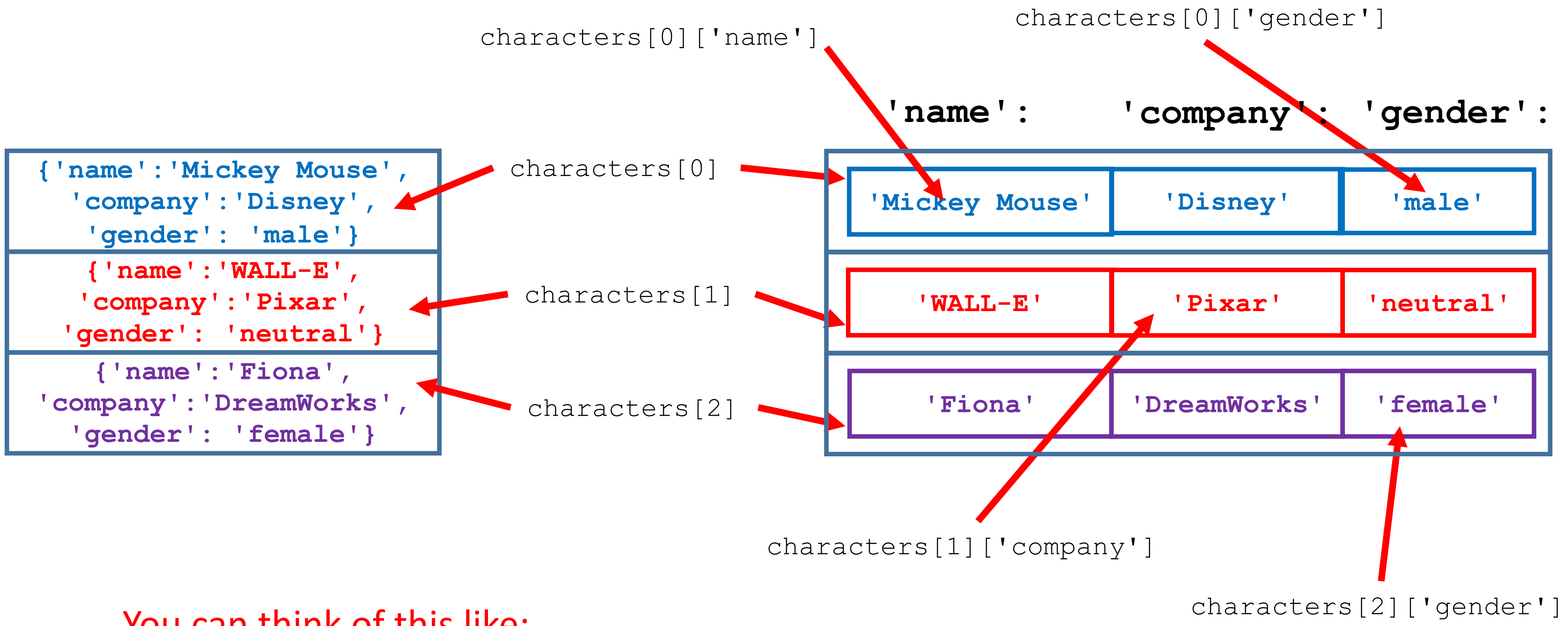
Lists of dictionaries

DISC DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Lists of dictionaries

```
characters = [{'name':'Mickey Mouse', 'company':'Disney', 'gender':
'male'}, {'name':'WALL-E', 'company':'Pixar', 'gender': 'neutral'},
{'name':'Fiona', 'company':'DreamWorks', 'gender': 'female'}]
```



You can think of this like:

```
data[row][key]
```

Since the keys aren't ordered, there is no significance to the order of the columns.

Lists of dictionaries (cont.)

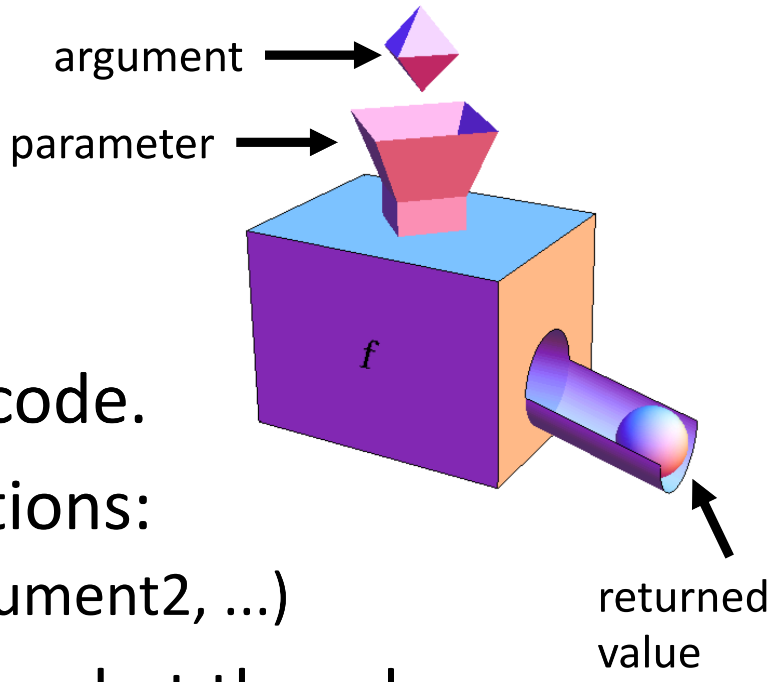
- Lists are iterable. Dictionaries aren't (they are unordered).
- It's common for each item on the list to represent an individual of some category of thing and each key:value pair in that individual's dictionary to represent a property of that individual.
- Stepping through the list processes each individual.

Making your own functions

DiSC DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Functions



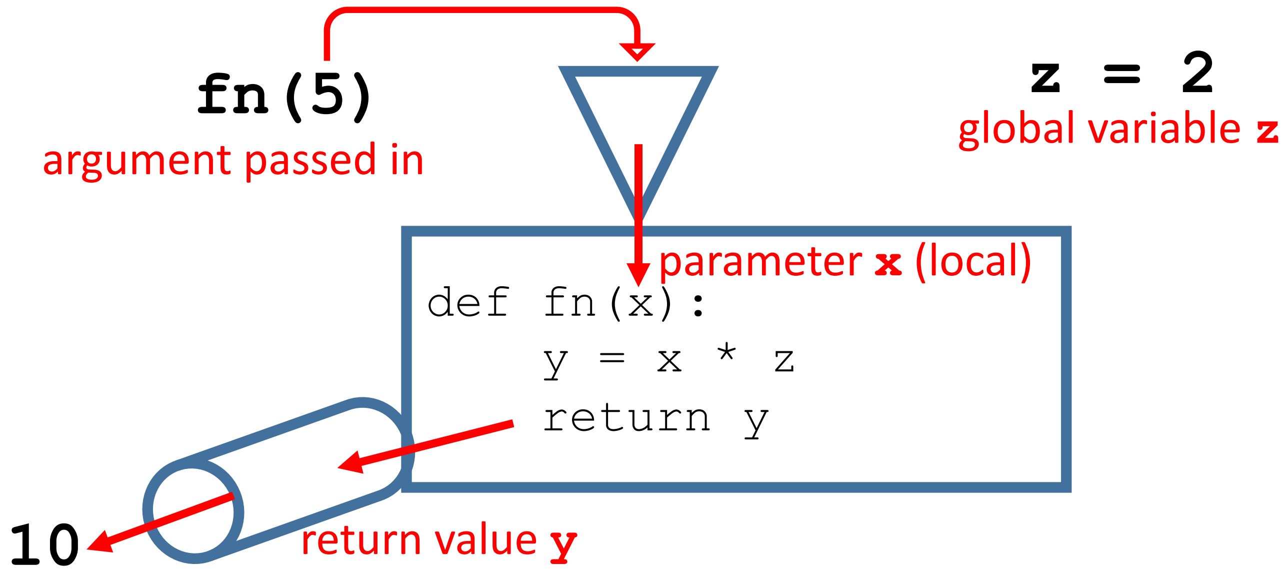
- A function defines a block of code.
- We pass arguments into functions:
    - functionName(argument1, argument2, ...)
- It's good to name functions by what they do.
  Example:

```
my_latte = make_latte(beans, milk, water)
```

- Functions can be:
    - built-in
    - **defined by you in your code ✓**
    - defined by somebody else in a module

Image: Nykamp DQ, "Function machine f." From Math Insight. http://mathinsight.org/image/function_machine_f CC BY-NC-SA

What's going on inside the function?



```
def fn(x):
    y = x * z
    return y
```

- **Parameters** are placeholder variables for arguments.
- The **scope** of variables assigned **inside** the function is **local**.
- Variables assigned in the **main script** are **global** and can be used in the function.
- The function can **return** one or more objects as a **return value**.

Defining a function

- Functions are defined using the **`def`** statement.
- The **`def`** statement ends with a colon.
- The function can have zero to many parameters.
- The function code is an indented code block.
- The function can return one or more values or nothing.

```
def fn(x):
    y = x * z
    return y
```

- It is safest to pass variables into the function as parameters to avoid having the function modify a global variable.

- Arguments can be literals or variables
- The name of the variable passed into the function as an argument can be different than the name used as the parameter for that argument.

```
# function to multiply two numbers
def multiplication(first_number, second_number):
    answer = first_number * second_number
    return answer

# main script
print(multiplication(3,5))

num1 = 3
num2 = 5
answer = multiplication(num1, num2)
print(answer)
```