Dictionary objects

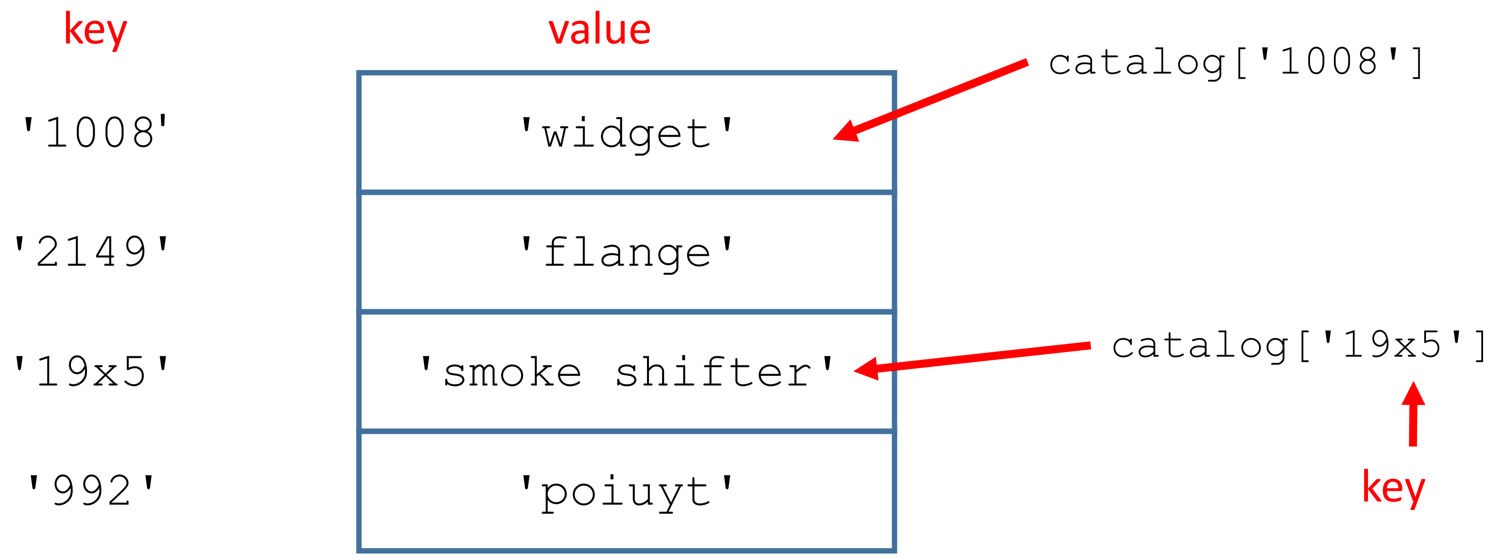
DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Dictionaries

- Dictionaries are an unordered data structure.
- They're defined using curly brackets: **{ }**
- Values are identified by keys. In this example, the keys are identifiers for the values
- We "look up" values in the dictionary using the keys.

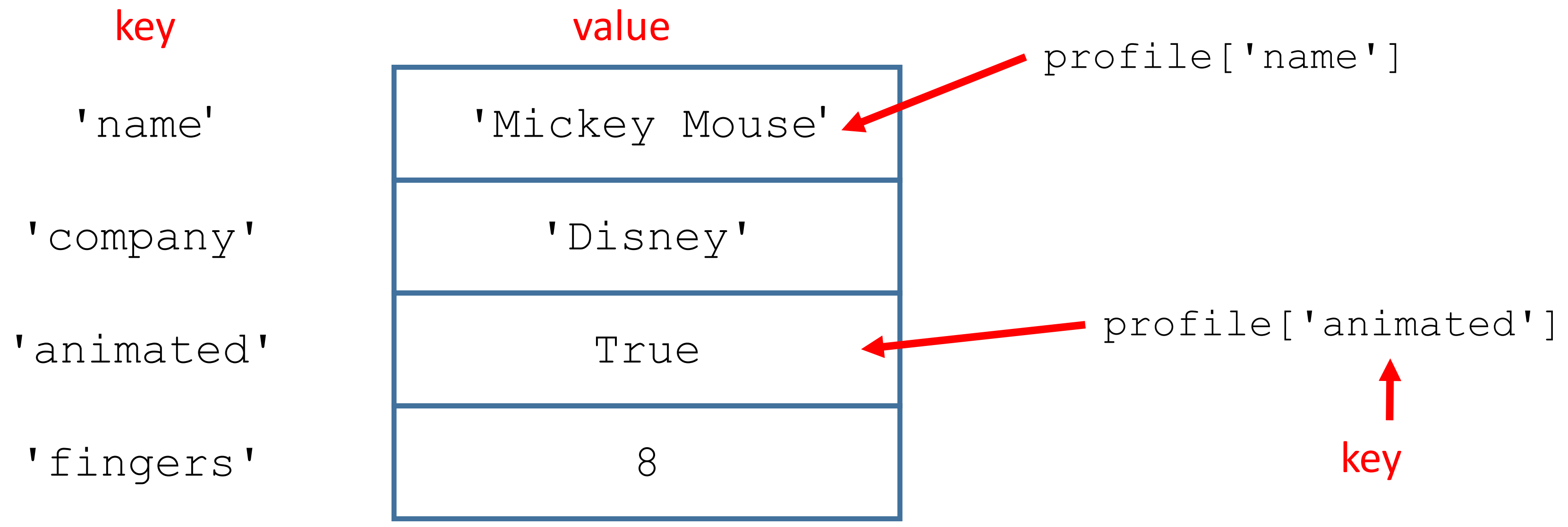
```
catalog = {'1008':'widget', '2149':'flange', '19x5':'smoke
shifter', '992':'poiuyt'}
```



Dictionaries

- Keys can also represent characteristics of an object
- Keys are always strings, values can be any object type
- "dict" is Python slang for "dictionary"

```
profile = {'name':'Mickey Mouse', 'company':'Disney', 'animated':True,
'fingers':8}
```



Commands for editing dictionaries

- An **empty dictionary** can be created using

```
traits = {}
```

- Both **creating** and **changing a value** in the dictionary are done by assigning a value by designated key

```
traits['height'] = 12
```

- An item can be **removed** using the **`del`** command

```
del traits['eye color']
```

List of lists

Lists can contain any kind of object

- Lists can also contain other lists:

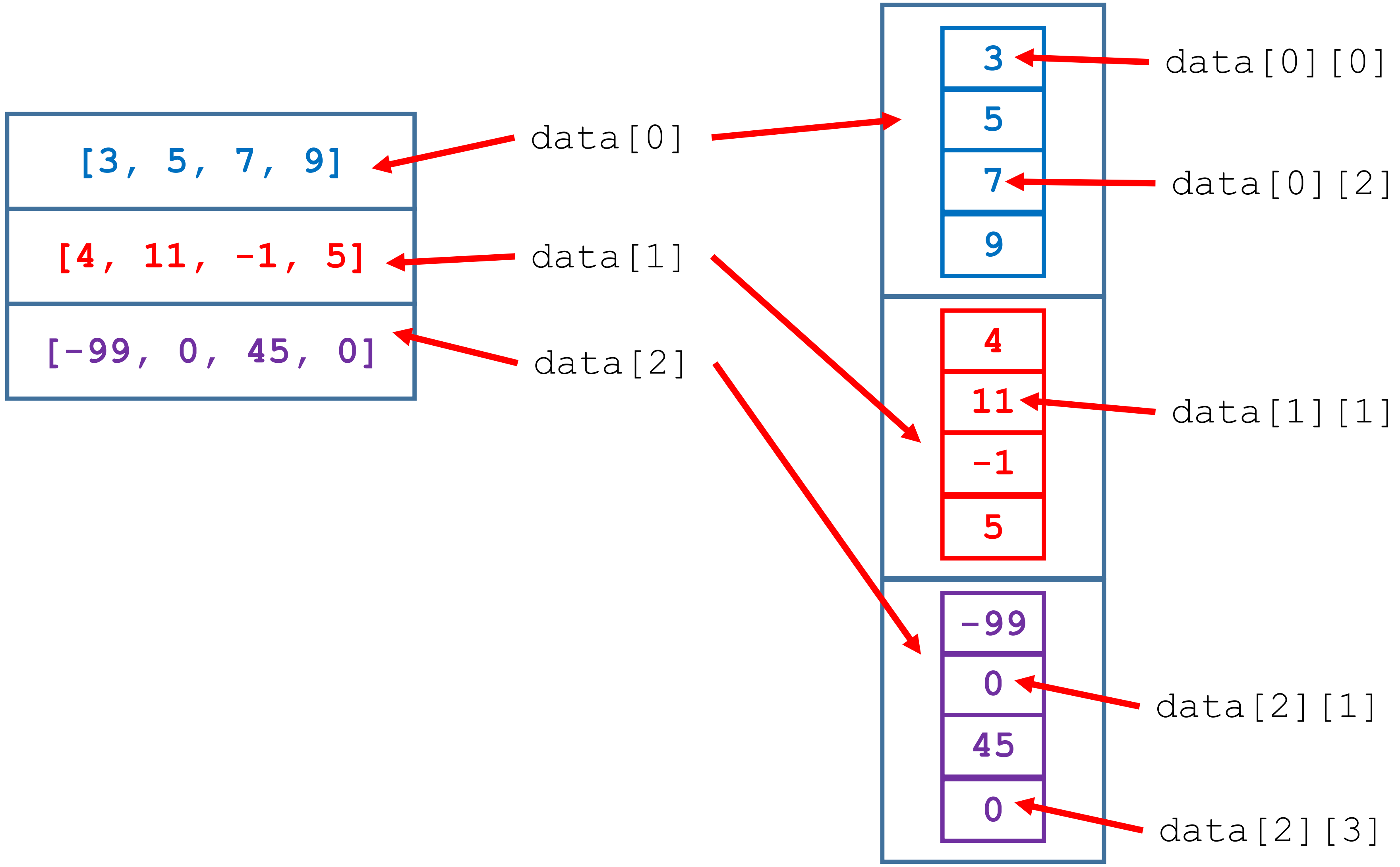
```
first_row = [3, 5, 7, 9]
second_row = [4, 11, -1, 5]
third_row = [-99, 0, 45, 0]
data = [first_row, second_row, third_row]
```

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

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

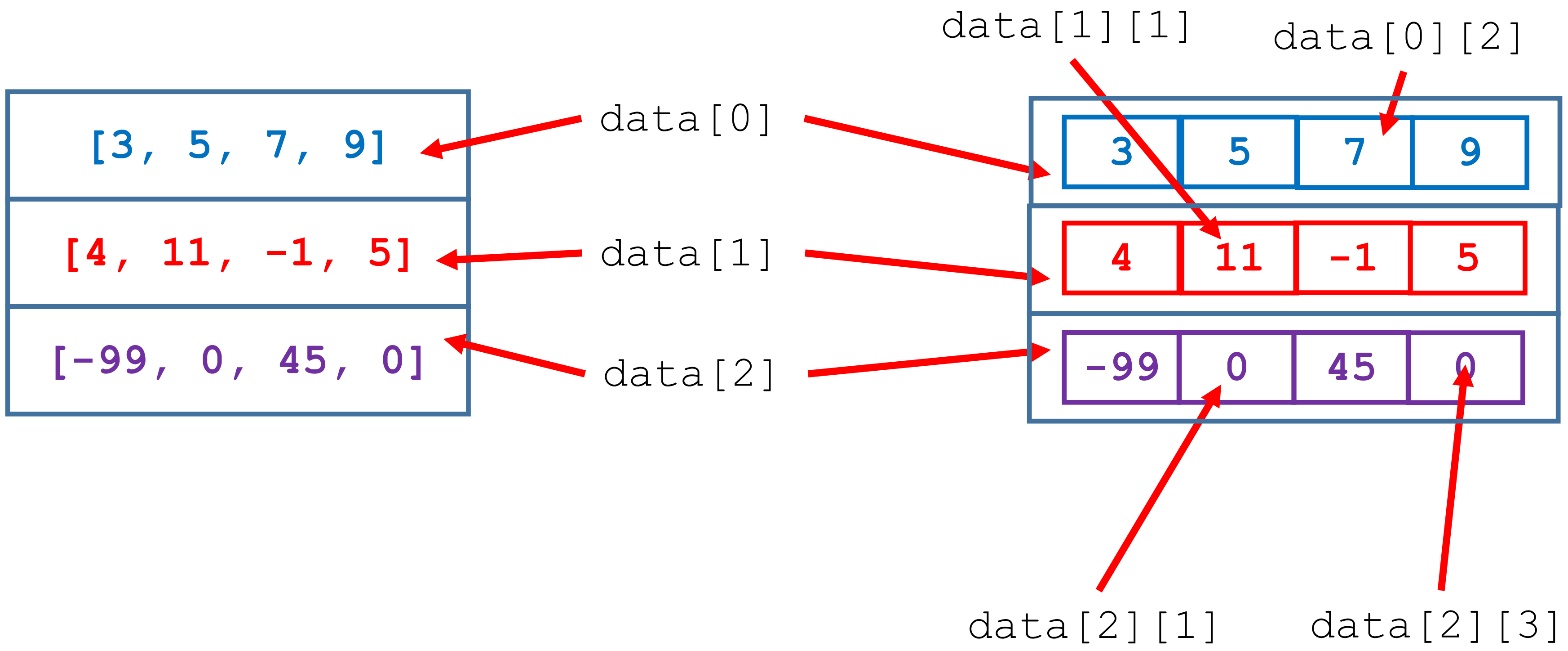
Lists of lists

```
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:

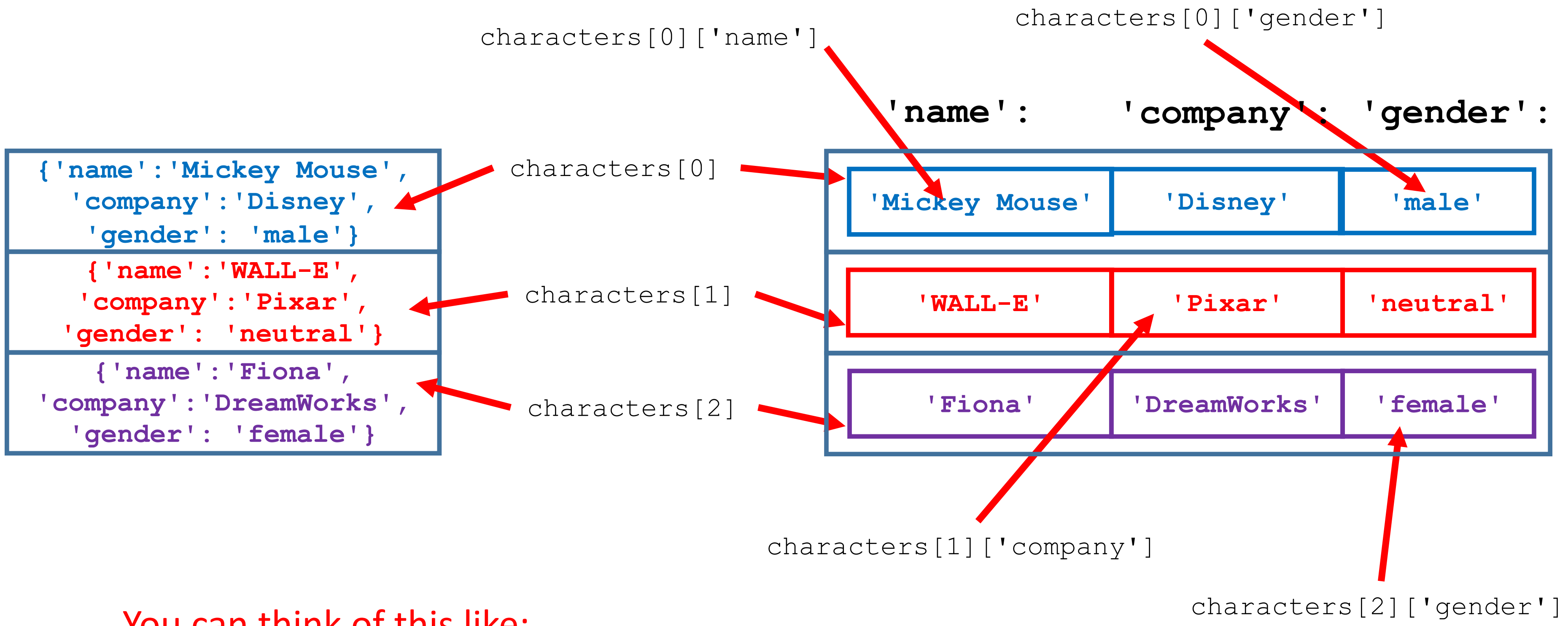
`data[row][column]`

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

Lists of dictionaries

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.

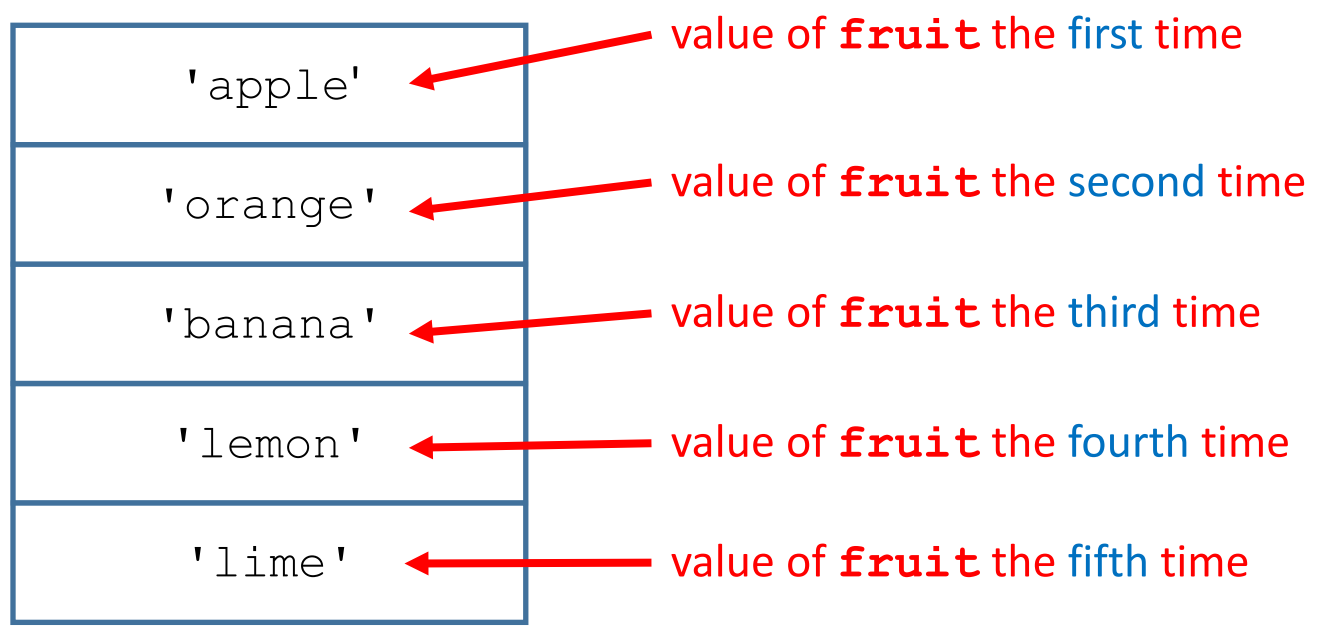
`for` loop

DISC DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Iterating with `for`

```
for fruit in basket:
        do this indented code block once for each fruit
then do this code block
```



'apple'
'orange'
'banana'
'lemon'
'lime'
value of fruit the first time
value of fruit the second time
value of fruit the third time
value of fruit the fourth time
value of fruit the fifth time

`basket`

(iterable list)

Example

notice this colon

```
basket = ['apple', 'orange', 'banana', 'lemon', 'lime']
for fruit in basket:
    print('I ate one ' + fruit)
print("I'm full now!")
```

- The indented code block can have more than one line.
- The upcoming code block is signaled by a colon (:)
- Strings are iterable by character.
- `for` is useful when there are a definite number of loops

`range()` as an iterable

- The range iterates from the first number to one step less than the second number:
  - `range(1, 11)` iterates from 1 to 10
- A step is optional:
  - `range(2, 10, 2)` iterates by twos from 2 to 8
- The step can be negative:
  - `range(10, 0, -1)` iterates from 10 to 1

Using the value of the range

```
for number in range(1, 11):
    the_square = number**2
    the_area = the_square * 3.14159
    print(number, '\t', the_area)
print("Those are the areas of all the circles!")
```

- The value of the iterated variable can be used anywhere in the indented code block.

Examples

- It's very common to use the length of a list as the end of a range (see last example).
    - Using the length of the list iterates through the whole list because counting is zero-based.

`while` loop

DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Looping with **`while`**

```
power = 0
exponent = 0
print('exponent\tpower')
while power < 100:
    power = 2**exponent
    exponent += 1
    print(exponent, '\t', power)
print("Those are the powers of two.")
```

- **`while`** loops are useful for an indefinite number of loops
- The test value must have an initial value.
- The test value must be able to meet the condition.
- The test is not made again until the loop end

Stringing together methods

DIGITAL SCHOLARSHIP AND COMMUNICATIONS

Jean & Alexander Heard LIBRARIES

Applying methods sequentially

- Recall that functions can be nested inside functions.
- A method can be added onto a method.
- The output of the first method must be the correct type for the second method.
- Example:

```
from datetime import date
this_day = date.today().weekday()
```

Example using strings, lists, and numbers

- Stringing together methods makes compact code
- It also makes less readable code.
- Similar problem to nesting many functions:

```
sqrt(int(input('How many? ')))
```