Smith College

Computer Science

Week 12

CSC111 — Spring 2018

Dominique Thiébaut
dthiebaut@smith.edu

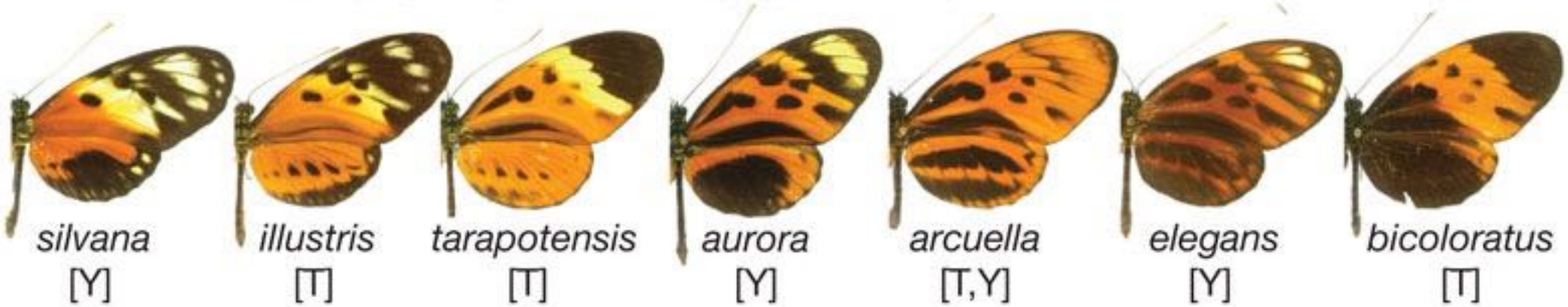
http://www.science.smith.edu/dftwiki/index.php/Nice_Wrapping_Paper...

Polymorphism Dictionaries & Recursion

http://www.nature.com/nature/journal/v477/n7363/fig_tab/nature10341_F1.html

Melinaea : models

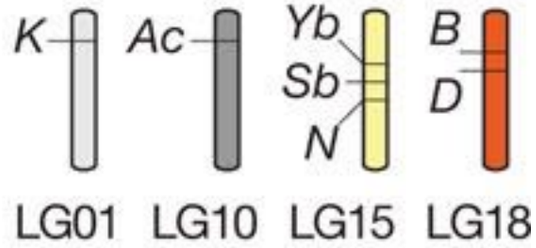
Heliconius numata : polymorphic mimic, sympatric morphs



Supergene

Heliconius melpomene : monomorphic, parapatric subspecies

Multilocus toolkit



Polymorphism

Poly = Many

Morph = Form, Shape

Poly = Many
Morph = Form, Shape

Image taken from http://stackoverflow.com/questions/3322318/explain-polymorphism

Poly = Many
Morph = Form, Shape

Image taken from http://stackoverflow.com/questions/3322318/explain-polymorphism

Poly = Many
Morph = Form, Shape

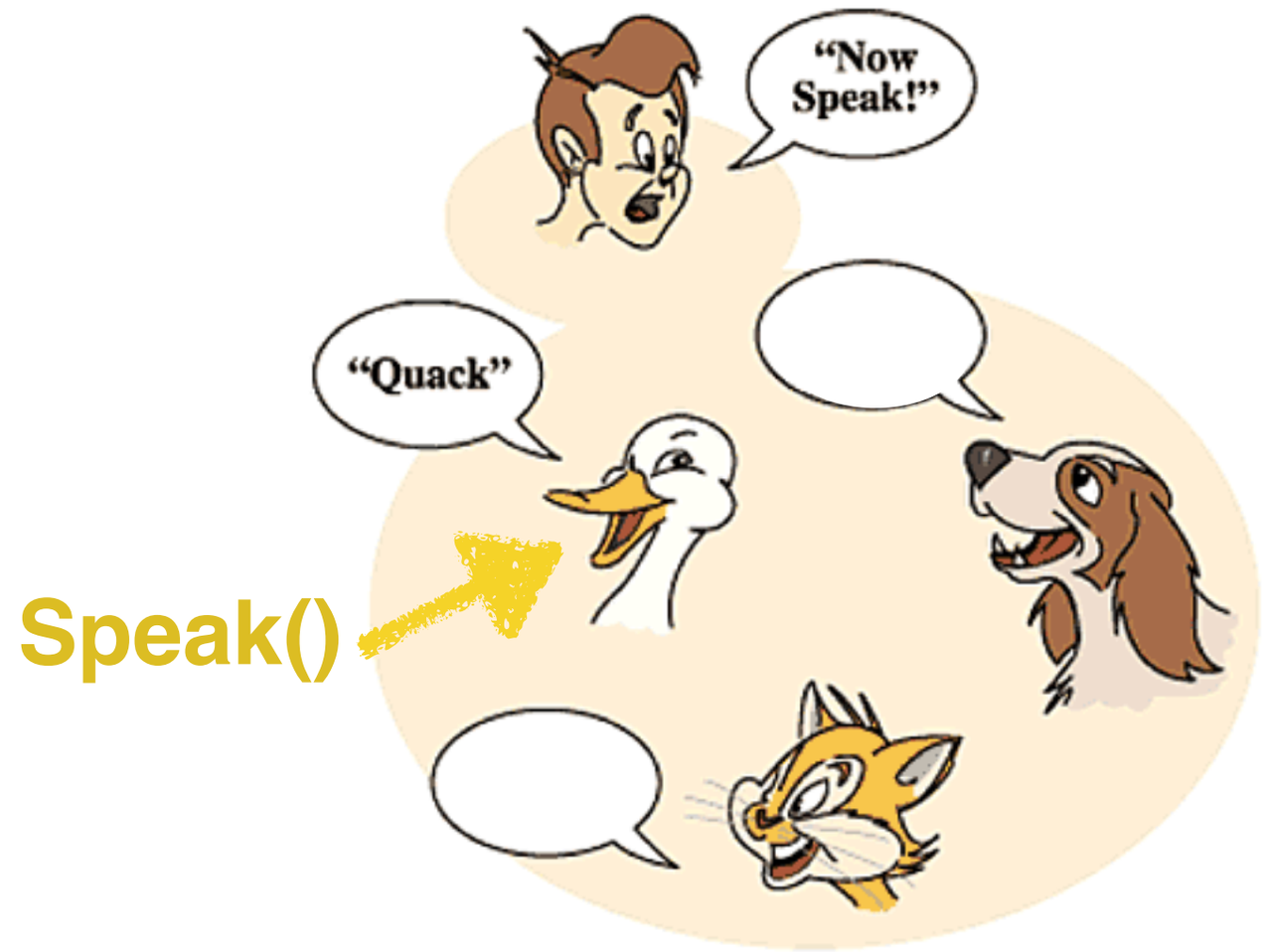


Image taken from http://stackoverflow.com/questions/3322318/explain-polymorphism

Poly = Many
Morph = Form, Shape

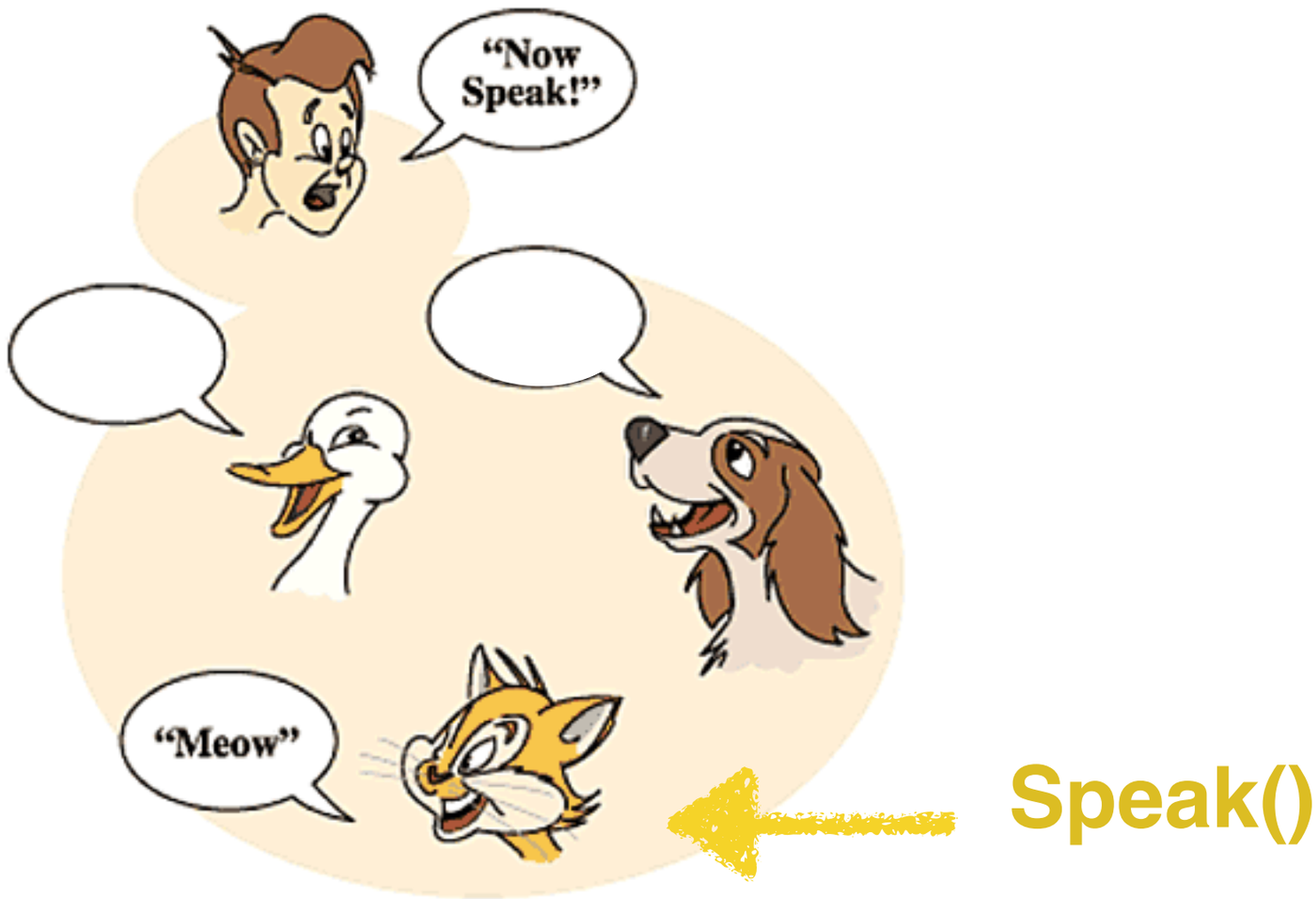


Image taken from http://stackoverflow.com/questions/3322318/explain-polymorphism

Poly = Many
Morph = Form, Shape

Polymorphism is the ability for a function, or method, to behave differently depending on which class it belongs to, or what type of parameter is given to it.

Image taken from http://stackoverflow.com/questions/3322318/explain-polymorphism

Let's Code this!

animalsTalk.py

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/111/animalsTalk

```
class Animal:
    def whatAreYou( self ):
        return
    def sleep( self ):
        print( "Zzzzzz...\n" )
    def speak( self ):
        return

class Dog( Animal ):
    def whatAreYou( self ):
        print( "I'm a dog!" )
    def speak( self ):
        print( "Woof, woof!" )

class Cat( Animal ):
    def whatAreYou( self ):
        print( "Why, I'm a cat.  Leave me alone!" )
    def speak( self ):
        print( "Meow!" )

class Duck( Animal ):
    def whatAreYou( self ):
        print( "Donald" )
    def speak( self ):
        print( "Quack, quack!" )
```

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/1...

```
def main():
    garfield = Cat()
    rex      = Dog()
    ducky    = Duck()

    garfield.whatAreYou()
    garfield.speak()
    garfield.sleep()

    rex.whatAreYou()
    rex.speak()
    rex.sleep()

    ducky.whatAreYou()
    ducky.speak()
    ducky.sleep()

main()
```

Ln: 58 Col: 4

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/111/animalsTalk

```
class Animal:
    def whatAreYou( self ):
        return
    def sleep( self ):
        print( "Zzzzzz...\n" )
    def speak( self ):
        return

class Dog( Animal ):
    def whatAreYou( self ):
        print( "I'm a dog!" )
    def speak( self ):
        print( "Woof, woof!" )

class Cat( Animal ):
    def whatAreYou( self ):
        print( "Why, I'm a cat.  Leave me alone!" )
    def speak( self ):
        print( "Meow!" )

class Duck( Animal ):
    def whatAreYou( self ):
        print( "Donald" )
    def speak( self ):
        print( "Quack, quack!" )
```

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/1...

```
def main():
    garfield = Cat()
    rex      = Dog()
    ducky    = Duck()

    garfield.whatAreYou()
    garfield.speak()
    garfield.sleep()

    rex.whatAreYou()
    rex.speak()
    rex.sleep()

    ducky.whatAreYou()
    ducky.speak()
    ducky.sleep()

main()
```

Ln: 58 Col: 4

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/111/animalsTalk.py

```
class Animal:
    def whatAreYou( self ):
        return
    def sleep( self ):
        print( "Zzzzzz...\n" )
    def speak( self ):
        return

class Dog( Animal ):
    def whatAreYou( self ):
        print( "I'm a dog!" )
    def speak( self ):
        print( "Woof, woof!" )
```

Python 3.5.4 Shell

```
========= RESTART: /Users/thiebaut/Deskt
op/Dropbox/111/animalsTalk.py ========
Why, I'm a cat.  Leave me alone!
Meow!
Zzzzzz...
I'm a dog!
Woof, woof!
Zzzzzz...
Donald
Quack, quack!
Zzzzzz...
>>>
```

Ln: 23 Col: 13


```
garfield.whatAreYou()
garfield.speak()
garfield.sleep()

rex.whatAreYou()
rex.speak()
rex.sleep()

ducky.whatAreYou()
ducky.speak()
ducky.sleep()

main()
```

Ln: 58 Col: 4

Polymorphism

Polymorphism

Polymorphism

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/111/animalsTalk.py

```
class Animal:
    def whatAreYou( self ):
        return
    def sleep( self ):
        print( "Zzzzzz...\n" )
    def speak( self ):
        return

class Dog( Animal ):
    def whatAreYou( self ):
        print( "I'm a dog!" )
    def speak( self ):
        print( "Woof, woof!" )
```

Python 3.5.4 Shell

```
======== RESTART: /Users/thiebaut/Deskt
op/Dropbox/111/animalsTalk.py ========
Why, I'm a cat.  Leave me alone!
Zzzzzz...
I'm a dog!
of!
Zzzzzz...
Donald
<!
Zzzzzz...
>>>
```

Ln: 23 Col: 13

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/1...

```
def main():
    garfield = Cat()
    rex      = Dog()
    ducky    = Duck()

    garfield.whatAreYou()
    garfield.speak()
    garfield.sleep()

    rex.whatAreYou()
    rex.speak()
    rex.sleep()

    ducky.whatAreYou()
    ducky.speak()
    ducky.sleep()

main()
```

Ln: 58 Col: 4

*animalsTalk.py - /Users/thiebaut/Desktop/Dropbox/111/animalsTalk.py

```
class Animal:
    def whatAreYou( self ):
        return
    def sleep( self ):
        print( "Zzzzzz...\n" )
    def speak( self ):
        return

class Dog( Animal ):
    def whatAreYou( self ):
        print( "I'm a dog!" )
    def speak( self ):
        print( "Woof, woof!" )
```

Overloading

Python 3.5.4 Shell

```
========= RESTART: /Users/thiebaut/Deskt
op/Dropbox/111/animalsTalk.py =========
Why, I'm a cat.  Leave me alone!
Meow!
Zzzzzz...
I'm a dog!
Woof, woof!
Zzzzzz...
Donald
Quack, quack!
Zzzzzz...
>>>
```

Ln: 23 Col: 13

Another Example of Polymorphism: the + operator

Another Example of Polymorphism: the + operator

Python 3.5.0b1 Shell

```
Python 3.5.0b1 (v3.5.0b1:071fefbb5e3d, May 23 2015, 18:22:54)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
>>> 1 + 10
11
>>>
>>> "Smith" + " " + "College"
'Smith College'
>>>
>>> ["Doc", "Grumpy", "Happy"] + ["Sleepy", "Bashful", "Sneezy", "Dopey" ]
['Doc', 'Grumpy', 'Happy', 'Sleepy', 'Bashful', 'Sneezy', 'Dopey']
>>>
>>> ("Joe", 3) + ("Marie", 4)
('Joe', 3, 'Marie', 4)
>>>
```

Another Example of Polymorphism: the + operator

Python 3.5.0b1 Shell

```
Python 3.5.0b1 (v3.5.0b1:071fefbb5e3d, May 23 2015, 18:22:54)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
>>> 1 + 10
11
>>>
>>> "Smith" + " " + "College"
'Smith College'
>>>
>>> ["Doc", "Grumpy", "Happy"] + ["Sleepy", "Bashful", "Sneezy", "Dopey" ]
['Doc', 'Grumpy', 'Happy', 'Sleepy', 'Bashful', 'Sneezy', 'Dopey']
>>>
>>> ("Joe", 3) + ("Marie", 4)
('Joe', 3, 'Marie', 4)
>>>
```

Overloading

> “Abundant syntax brings more burden than help.”
>
> –Guido van Rossum

“Abundant syntax brings more burden than help.

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

–Guido van Rossum

Dictionaries

Review Lists

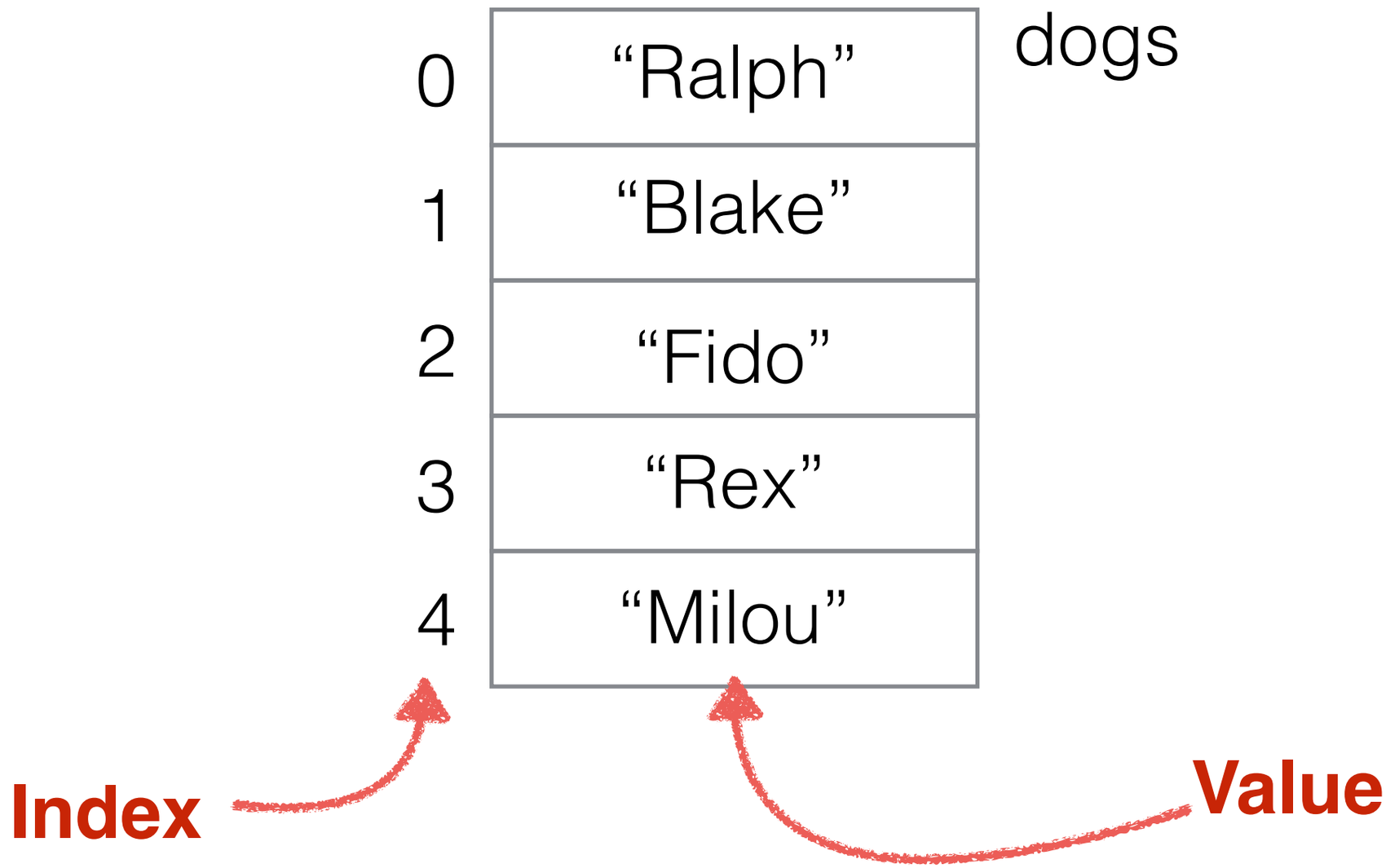
dogs = [“Ralph”, “Blake”, “Fido”, “Rex”, “Milou”]

Review Lists

dogs = [“Ralph”, “Blake”, “Fido”, “Rex”, “Milou”]

Review Lists

dogs = [“Ralph”, “Blake”, “Fido”, “Rex”, “Milou”]



Dictionaries: Indexing Data by Name

```
"Apple"      : 95,
"Apricot"    : 29,
"Avocado"  : 270,
"Banana"    : 95,
```

What we'd like:

```
"Apple"     : 95,
"Apricot"   : 29,
"Avocado"   : 270,
"Banana"    : 95,
```

What we'd like:

	calories
“Apple”	95
“Apricot”	29
“Avocado”	270
“Banana”	95

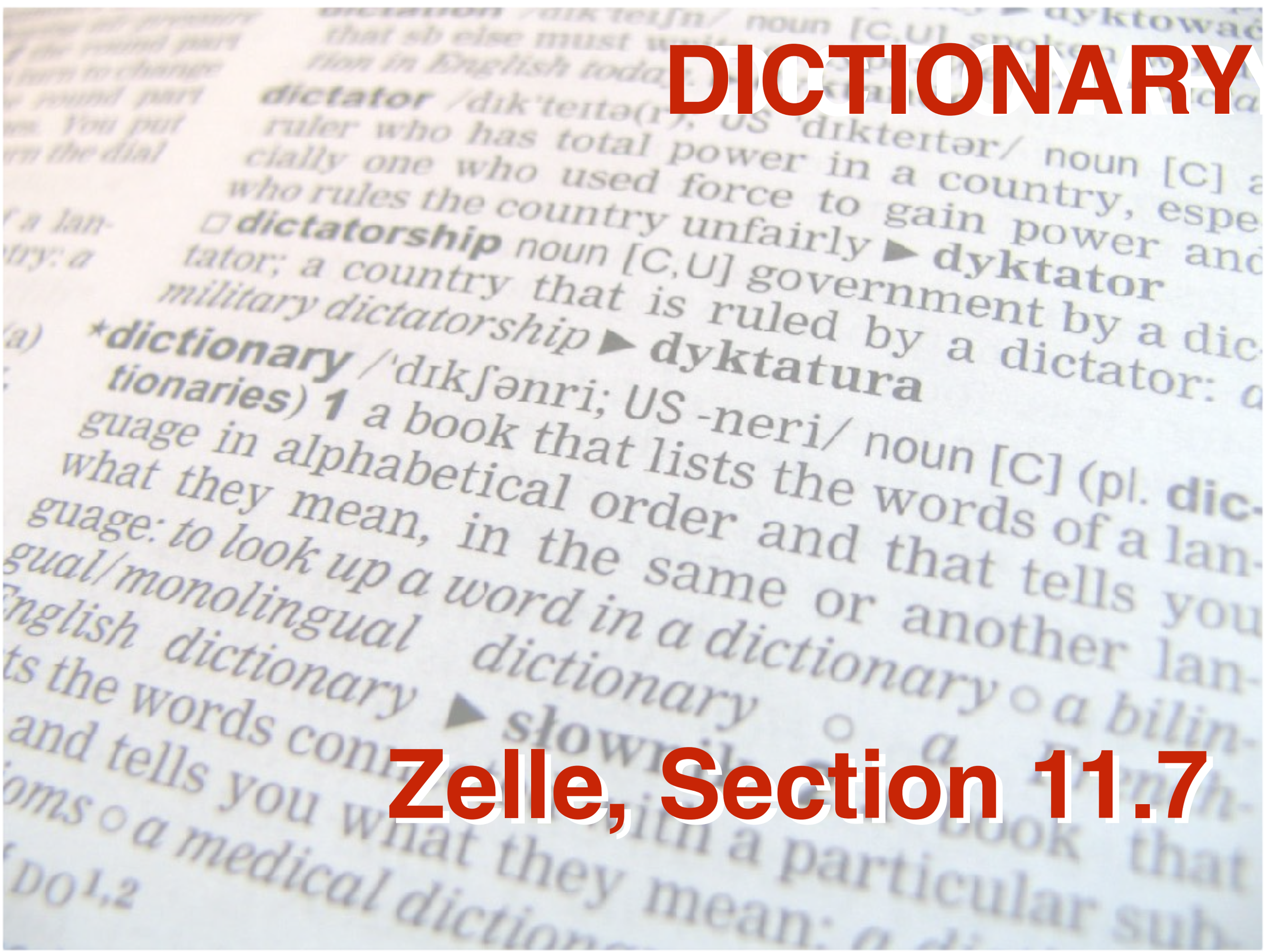
What we'd like:

```
"Apple"     : 95,
"Apricot"   : 29,
"Avocado"   : 270,
"Banana"    : 95,
```

	calories
“Apple”	95
“Apricot”	29
“Avocado”	270
“Banana”	95

Key

Value


DICTIONARY
dictator /dɪk'teɪtə(r); US 'dɪktertər/ noun [C] a
ruler who has total power in a country, espe
cially one who used force to gain power and
who rules the country unfairly ► dyktator
dictatorship noun [C,U] government by a dic
tator; a country that is ruled by a dictator:
military dictatorship ► dyktatura
*dictionary /'dɪkʃənri; US -neri/ noun [C] (pl. dic-
tionaries) 1 a book that lists the words of a lan-
guage in alphabetical order and that tells you
what they mean, in the same or another lan-
guage: to look up a word in a dictionary ○ a bilin-
gual/monolingual dictionary ○ a
English dictionary ► słownik
Zelle, Section 11.7

Dictionaries

- *Dictionaries* are **Data Structures**
- The item used as an index is called the **key**
- The item associated with the key is the **value**
- A dictionary is a container, or a **collection of (key, value) pairs**
- **Searching** for a key in a dictionary is a very **fast** operation.

searchListAndDictionary.py

Example

Python 3.5.4 Shell

```
Python 3.5.4 (v3.5.4:3f56838976, Aug  7 2017, 12:56:33)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> stocks = {
        "DJIA": 24733.38,
        "S&P":  2698.45,
        "NASDAQ": 7261.41,
        "CNBC IQ" : 269.97,
        "RUSS 2K" : 1580.33,
        "FTSE" : 7318.42 }
>>> stocks[ "DJIA" ]
24733.38
>>> stocks[ "FTSE" ]
7318.42
>>> for stock in stocks:
        print( stock )


FTSE
CNBC IQ
NASDAQ
S&P
DJIA
RUSS 2K
>>>
```

Ln: 25 Col: 4

Example

Python 3.5.4 Shell

```
>>> for st in stocks:
        print( st, "-->", stocks[st] )


FTSE --> 7318.42
CNBC IQ --> 269.97
NASDAQ --> 7261.41
S&P --> 2698.45
DJIA --> 24733.38
RUSS 2K --> 1580.33
>>> stocks[ "DAX" ] = 12558.91
>>> for st in stocks:
        print( st, "-->", stocks[st] )


FTSE --> 7318.42
CNBC IQ --> 269.97
NASDAQ --> 7261.41
S&P --> 2698.45
DJIA --> 24733.38
DAX --> 12558.91
RUSS 2K --> 1580.33
>>>
```

Ln: 61 Col: 4

Example

Python 3.5.4 Shell

```
>>> stocks[ "DAX" ] = 12558.91
>>> for st in stocks:
        print( st, "-->", stocks[st] )


FTSE --> 7318.42
CNBC IQ --> 269.97
NASDAQ --> 7261.41
S&P --> 2698.45
DJIA --> 24733.38
DAX --> 12558.91
RUSS 2K --> 1580.33
>>>
>>>
>>>
>>>
>>> # add 10 to S&P
>>> stocks[ "S&P" ] += 10
>>> stocks[ "S&P" ]
2708.45
>>>
```

Ln: 59 Col: 4

Example

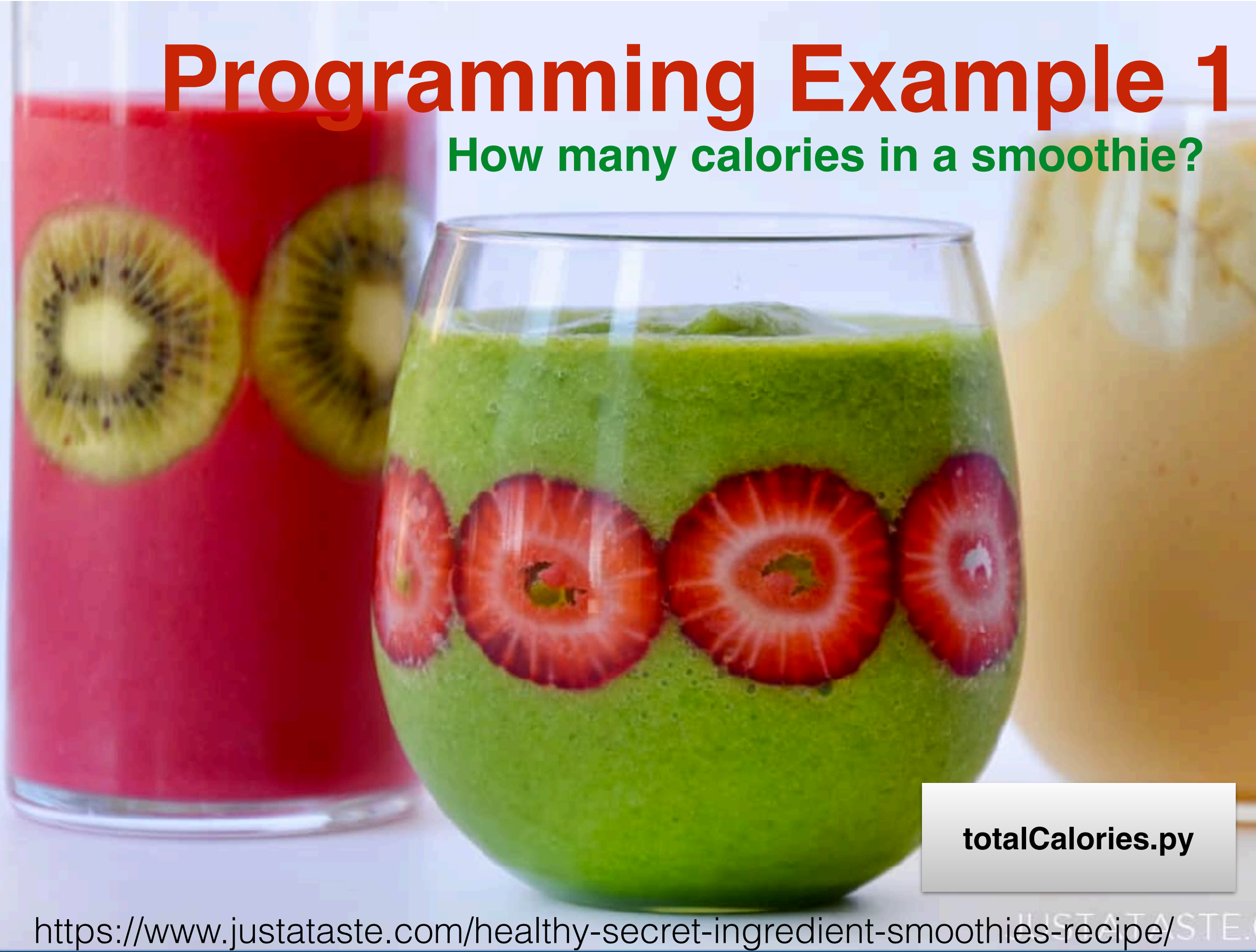
Python 3.5.4 Shell

```
>>>
>>>
>>>
>>> # add 10 to S&P
>>> stocks[ "S&P" ] += 10
>>> stocks[ "S&P" ]
2708.45
>>> # test if key is in dictionary
>>> "DAX" in stocks
True
>>> "dax" in stocks
False
>>> "DJIA" in stocks
True
>>> "Djia" in stocks
False
>>> "RUSS 2K" in stocks
True
>>>
```

Ln: 60 Col: 4

Programming Example 1

How many calories in a smoothie?



totalCalories.py

https://www.justataste.com/healthy-secret-ingredient-smoothies-recipe/

totalCalories.py - /Users/thiebaut/Desktop/Dropbox/111/totalCalories.py (3.5.4)

```
# totalCalories.py
# D. Thiebaut
# A program illustrating the use of a dictionary
# to compute the total amount of calories in a
# smoothy containing several different fruit.
# Additional fruit and calorie data can be found
# at this URL: http://www.lasting-weight-loss.com/calories-in-fruit.html
#
caloriesDict = {
    "Apple"          : 95,
    "Apricot"        : 29,
    "Avocado"        : 270,
    "Banana"         : 95,
    "Blackberry"     : 25,
    "Blackcurrant"   : 28,
    "Blueberry"      : 30,
    "Boysenberry"    : 66,
    "Cherimoya"      : 115,
    "Coconut"        : 351,
    "Cranberry"      : 15,
    "Date"           : 107,
    "Fig"            : 43,
    "Grapefruit"     : 48 }
```

totalCalories.py

totalCalories.py - /Users/thiebaut/Desktop/Dropbox/111/totalCalories.py (3.5.4)

```
def main():
    # define a list of smoothies and their contents
    Smoothies = [ ( "Blueberry", "Banana", "Apple" ),
                  ( "fig", "grapefruit", "avocado" ) ]

    # for each smoothie, compute the total number of calories
    for smoothy in Smoothies:
        totalCal = 0
        # display smoothie nicely formatted
        print( "Smoothy:", ", ".join( smoothy ) )

        # for each fruit in this smoothie, add its calorie count
        # to total
        for fruit in smoothy:
            fruit = fruit.capitalize()
            if fruit in caloriesDict:
                print( "{0:20}-->{1:5}".format( fruit, caloriesDict[ fruit ] ) )
                totalCal = totalCal + caloriesDict[ fruit ]

        # print total calories
        print( "{0:20}   {1:5} calories".format( "Total calories", totalCal ) )
        print()

main()
```

Ln: 44 Col: 30

Programming Example 2

```
Adelyn@smith.edu      MTH
Adison@mtholyoke.edu ARH
Aisha@smith.edu       MTH
Alana@smith.edu       MTH
Aleena@smith.edu      CSC
.
.
.
Taniya@smith.edu      SPN
Tara@smith.edu        CSC
Tatum@smith.edu       MTH
Tianna@smith.edu      ARH
Tiara@smith.edu       CSC
Tori@smith.edu        MTH
Violet@smith.edu      ARH
Wendy@smith.edu       ECO
Yareli@smith.edu      FRN
Yaritza@smith.edu     ARH
Yuliana@smith.edu     ECO
smithy@umass.edu      ARH
```

Programming Example 2

```
Adelyn@smith.edu       MTH
Adison@mtholyoke.edu   ARH
Aisha@smith.edu        MTH
Alana@smith.edu        MTH
Aleena@smith.edu       CSC
.
.
.
Taniya@smith.edu       SPN
Tara@smith.edu         CSC
Tatum@smith.edu        MTH
Tianna@smith.edu       ARH
Tiara@smith.edu        CSC
Tori@smith.edu         MTH
Violet@smith.edu       ARH
Wendy@smith.edu        ECO
Yareli@smith.edu       FRN
Yaritza@smith.edu      ARH
Yuliana@smith.edu      ECO
smithy@umass.edu       ARH
```

countMajors.py - /Users/thiebaut/Desktop/Dropbox/111/countMajors.py (3.6.4)

```
# create a dictionary for the majors
majorsDict = {}

# process each line...
for line in lines:

    # split each line into 2 fields
    try:
        email, major = line.strip().split()
    except ValueError:
        continue

    # if major not a key in dictionary,
    # add it...
    if not major in majorsDict:
        majorsDict[ major ] = 0

    # increment count for the major just found
    majorsDict[ major ] += 1

# display the majors and the number of students
for major in majorsDict:
    print( major, "-->", majorsDict[major] )

main()
```

countMajors.py

Programming Example 2

```
Adelyn@smith.edu       MTH
Adison@mtholyoke.edu   ARH
Aisha@smith.edu        MTH
Alana@smith.edu        MTH
Aleena@smith.edu       CSC
.
.
.
Taniya@smith.edu       SPN
Tara@smith.edu         CSC
Tatum@smith.edu        MTH
Tianna@smith.edu       ARH
Tiara@smith.edu        CSC
Tori@smith.edu         MTH
Violet@smith.edu       ARH
Wendy@smith.edu        ECO
Yareli@smith.edu       FRN
Yaritza@smith.edu      ARH
Yuliana@smith.edu      ECO
smithy@umass.edu       ARH
```

countMajors.py - /Users/thiebaut/Desktop/Dropbox/111/countMajors.py (3.5.4)

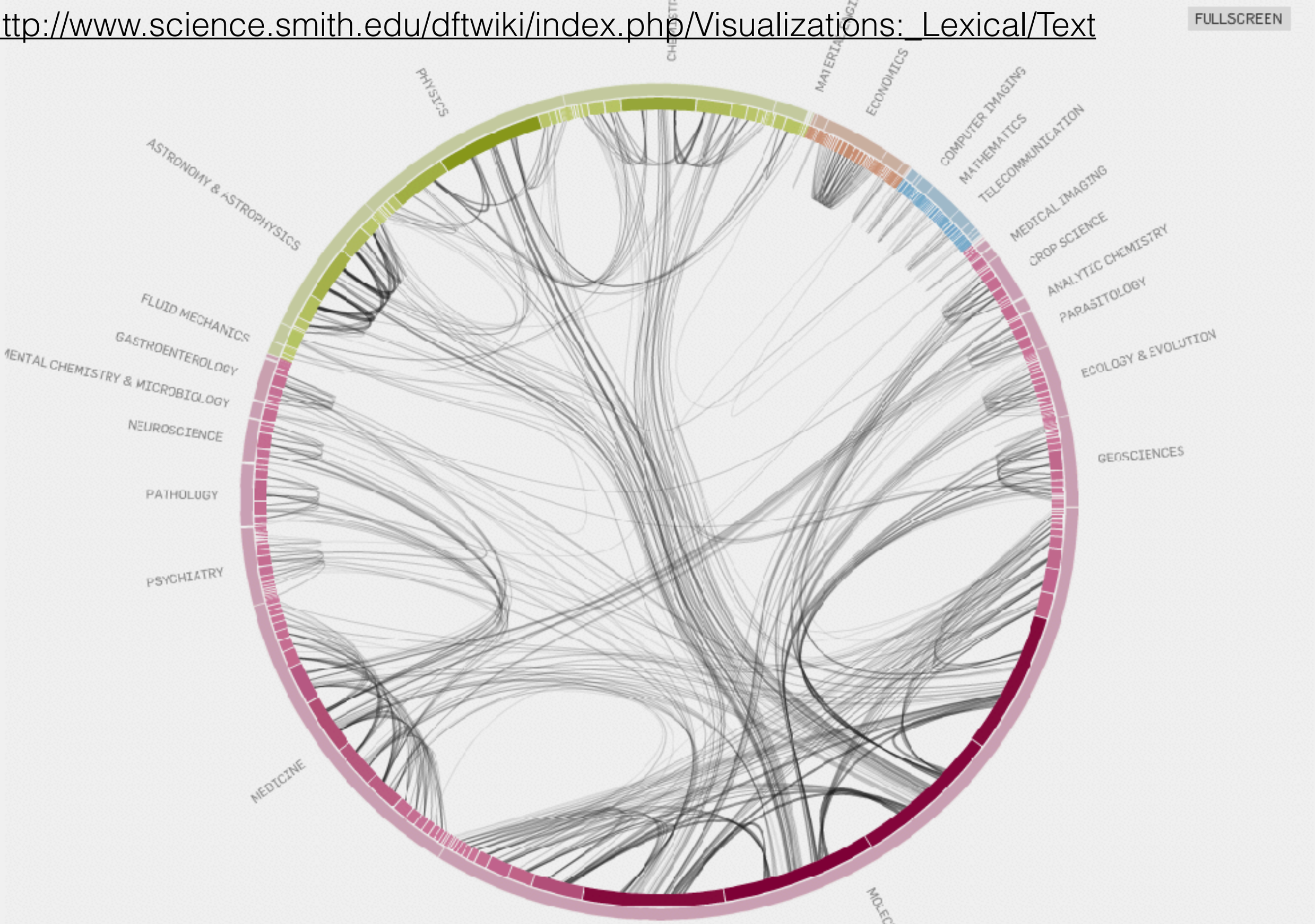
```
# create a dictionary for the majors
majorsDict = {}

# process each line...
for line in lines:

    # split each line into 2 fields
    try:
        email, major = line.strip().split()
    except ValueError:
        continue

    # if major not a key in dictionary,
    # add it...
    if not major in majorsDict:
        majorsDict[ major ] = 0

    # increment count for the major just found
    majorsDict[ major ] += 1

# display the majors and the number of students
for major in majorsDict:
    print( major, "-->", majorsDict[major] )

main()
```

countMajors.py

Programming Example 3

CREDIT: MOVIESTORE/REX/SHUTTERSTOCK

http://www.science.smith.edu/dftwiki/index.php/Visualizations:_Lexical/Text



http://well-formed.eigenfactor.org/index.html

streetCarWithDictionaries.py - /Users/thiebaut/Desktop/Dropbox/111/streetCarWithDictionaries.py (3.5.4)

```
# StreetCarWithDictionaries.py
# D. Thiebaut
# This program uses a dictionary of dictionaries
# to count how many times the three characters
# Stanley, Stella, and Blanche refer to each other
# in the play Street Car Named Desire, by Tennessy
# Williams

PLAY = "streetCar.txt"
characterCounters =   { "STANLEY": {"BLANCHE":0, "STELLA":0, "STANLEY":0 },
                        "BLANCHE": {"STANLEY":0, "STELLA":0, "BLANCHE":0 },
                        "STELLA":  {"STANLEY":0, "BLANCHE":0,"STELLA":0  }
                      }

def getLinesFromPlay( fileName ):
    '''opens the file given by its name, and
    returns all the text lines it contains.'''
    file = open( fileName, 'r' )
    lines = file.read()
    lines = lines.split( '\n' )
    file.close()
    return lines
```

streetCarWithDictionaries.py

streetCarWithDictionaries.py - /Users/thiebaut/Desktop/Dropbox/111/streetCarWithDictionaries.py (3.5.4)

```
def main():
    global PLAY, characterCounters

    lines = getLinesFromPlay( PLAY )
    for line in lines:
        # find lines that start with uppercase and contain a ':'
        if len(line) > 0 and 'A' <= line[0] <= 'Z' and line.find( ':' ) != -1:
            # split the line at the first ':'
            character, tirade = line.split( ':', 1 )

            # skip lines where character is not a key in dictionary
            if character not in characterCounters:
                continue

            # make tirade uppercase so all character names are uppercase
            tirade = tirade.upper().strip()

            # for each of the characters who are keys in dictionary...
            for otherCharacter in characterCounters:

                # count how many times each appears in tirade
                count = tirade.count( otherCharacter )
                if count != 0:
                    # update counter for character mentionning otherCharacter
                    characterCounters[ character ][ otherCharacter ] += count
```

Ln: 51 Col: 3

streetCarWithDictionaries.py - /Users/thiebaut/Desktop/Dropbox/111/streetCarWithDictionaries.py (3.5.4)

```
    # display the resulting counts
    for character in characterCounters:
        print( character )

        # get the dictionary of counters for that character
        dico = characterCounters[ character ]

        # go through each other-character in the dictionary, and display
        # associated counts
        for otherCharacter in dico:
            print( "   refers to", otherCharacter,
                   dico[otherCharacter ], "times" )


main()
```

Ln: 57 Col: 8

Python 3.5.4 Shell

```
= RESTART: /Users/thiebaut/Desktop/Dropbox/111/streetCarWithDictionaries.py =
STANLEY
   refers to STANLEY 0 times
   refers to BLANCHE 18 times
   refers to STELLA 16 times
BLANCHE
   refers to STANLEY 15 times
   refers to BLANCHE 5 times
   refers to STELLA 63 times
STELLA
   refers to STANLEY 26 times
   refers to BLANCHE 58 times
   refers to STELLA 0 times
>>> characterCounters[ "STANLEY" ][ "STELLA" ]
16
>>> characterCounters
{'STANLEY': {'STANLEY': 0, 'BLANCHE': 18, 'STELLA': 16}, 'BLANCHE': {'STANLEY': 15,
'BLANCHE': 5, 'STELLA': 63}, 'STELLA': {'STANLEY': 26, 'BLANCHE': 58, 'STELLA': 0}}
>>>
```

Ln: 48 Col: 4

Recursion

Tough problems, simple solutions

Recursion & Recursive Functions

Finding the Largest in a List

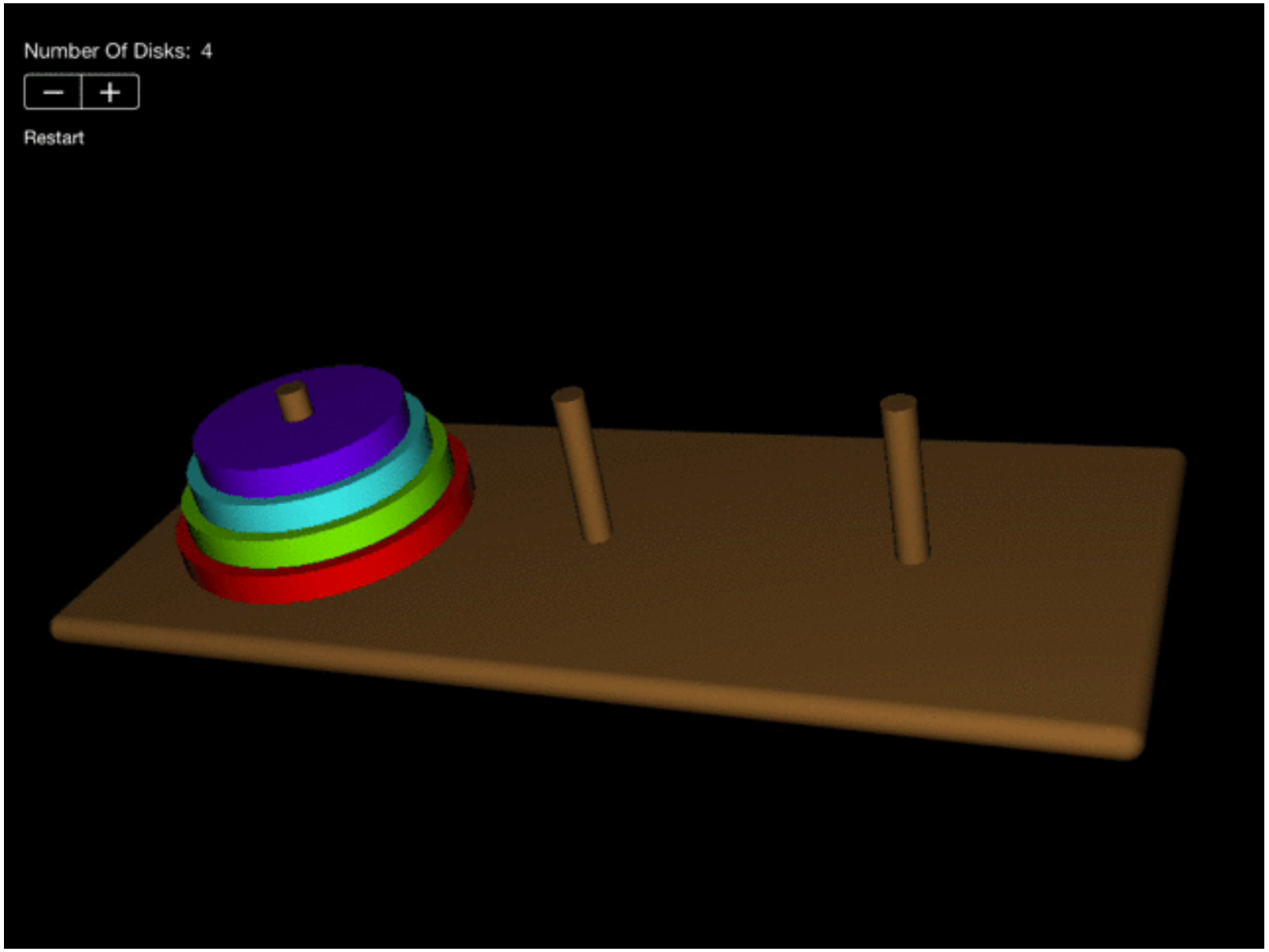
Finding the Smallest in a List

Factorial

Traversing a Maze

Fractal Trees

Towers of Hanoi



http://weheartswift.com

A B C

http://en.wikipedia.org

A B
A C
B C
A B
C A
...

A B C

http://en.wikipedia.org

Some Problems Can be Solved With *Little* Coding using Special Programming Techniques

Recursive Functions

Solving Problems Recursively:

Class Exercises

Largest in a list

Smallest in a list

Find item in a list

Wrap text

Factorial

Draw a Chessboard

bunch of bananas

Find the largest in a list…

bunch of bananas

Find the largest in a list…

(Lazy, but clever.
Has lots of friends)

Python

```
def minion( bunch ):
    """this minion is given a bunch of bananas.
    if there's only 1 banana in the bunch, it will
    return that banana as the largest in the bunch.

    If there's more than 1 banana in the bunch, this
    minion will keep one banana (the first one it finds)
    and pass the remainder of the bananas to another
    minion.  That new minion will follow the *exact*
    same script in order to find the largest banana.
    At some point that second minion will return the
    largest of the bananas in the remainder.  Our first
    minion will compare that to the one it kept, and
    return the largest of the two.
    """

    # if minion gets a bunch of 1 banana, that banana
    # is the largest of what it got.
    if len( bunch ) == 1:
        banana = bunch[0]
        return banana

    # if we're here, it's because the bunch has more
    # than 1 banana.
    # keep the first one:
    kept = bunch[0]

    # create a smaller bunch: the remaining bananas
    remainingBananas = bunch[1: ]

    # pass that to another minion
    largestOfRemainingBananas = minion( remainingBananas )

    # figure out which of the 2 bananas is largest: the
    # one kept, or the one just returned
    if kept > largestOfRemainingBananas:
        return kept
    else:
        return largestOfRemainingBananas

def main():
    bananas = [10, 2, 3, 20, -1, 5, 7, 0, 3, 6]
    largest = minion( bananas )

    print( "bunch = ", bananas )
    print( "largest banana found = ", largest )

main()
```

Python

```python
def minion( bunch ):
    """this minion is given a bunch of bananas.
    if there's only 1 banana in the bunch, it will
    return that banana as the largest in the bunch.

    If there's more than 1 banana in the bunch, this
    minion will keep one banana (the first one it finds)
    and pass the remainder of the bananas to another
    minion.  That new minion will follow the *exact*
    same script in order to find the largest banana.
    At some point that second minion will return the
    largest of the bananas in the remainder.  Our first
    minion will compare that to the one it kept, and
    return the largest of the two.
    """

    # if minion gets a bunch of 1 banana, that banana
    # is the largest of what it got.
    if len( bunch ) == 1:
        banana = bunch[0]
        return banana

    # if we're here, it's because the bunch has more
    # than 1 banana.
    # keep the first one:
    kept = bunch[0]

    # create a smaller bunch: the remaining bananas
    remainingBananas = bunch[1: ]

    # pass that to another minion
    largestOfRemainingBananas = minion( remainingBananas )

    # figure out which of the 2 bananas is largest: the
    # one kept, or the one just returned
    if kept > largestOfRemainingBananas:
        return kept
    else:
        return largestOfRemainingBananas

def main():
    bananas = [10, 2, 3, 20, -1, 5, 7, 0, 3, 6]
    largest = minion( bananas )

    print( "bunch = ", bananas )
    print( "largest banana found = ", largest )

main()
```

Stopping Condition

Recursive Step

Animation

Animation

Animation

Animation

Animation

Animation

Animation

Animation

Animation

Animation

Once Again!
With this Simplified Version

```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Once Again!
With this Simplified Version

```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Stopping Condintion

Recursive Step

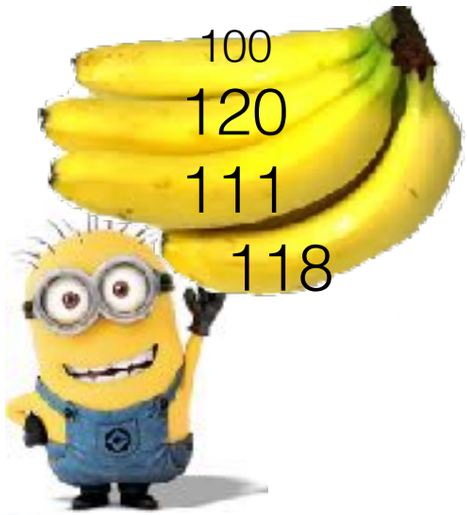
Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas ) )

main()
```

Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation




```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

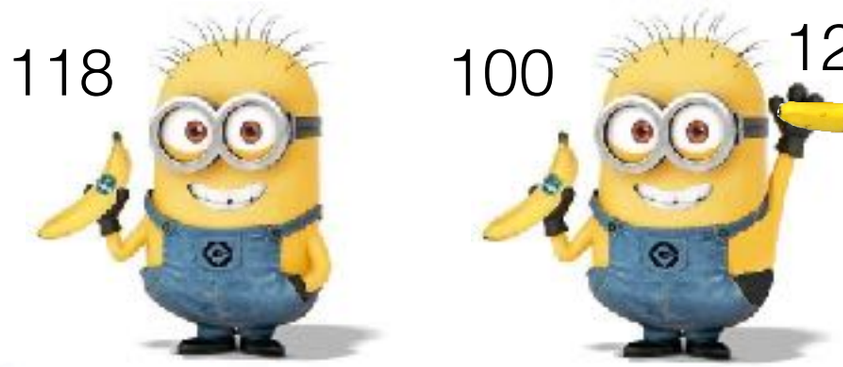
Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation




```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas )

main()
```

Animation


```
def minion( bunch ):
    if len( bunch )==1:
        return bunch[0]

    kept = bunch[0]
    remaining = bunch[1: ]

    biggestRemaining = minion( remaining )

    if biggestRemaining > kept:
        return biggestRemaining
    else:
        return kept

def main():
    bananas = [100, 120, 111, 118]
    print( "biggest = ", minion( bananas ) )
main()
```

Minions: Can you tell me if there's a banana that weighs 118 grams?

Minions: Here's a bunch of bananas. Return to me the smallest and the largest.

Tough problems, simple solutions

Recursive Functions

Finding the Largest in a List

Finding the Smallest in a List

Factorial

Traversing a Maze

Towers of Hanoi

Standard CS Example: *Factorial*

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!***

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!*** =

$$\begin{cases} 1 & \text{if } n == 1 \\ \boldsymbol{n} \times (\boldsymbol{n-1})\boldsymbol{!} & \text{otherwise} \end{cases}$$

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!*** =

$$n! = \begin{cases} 1 & \text{if } n == 1 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!*** =

$$\begin{cases} 1 & \text{if } n == 1 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

$5! = 5 \times 4!$

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!*** =

$$n! = \begin{cases} 1 & \text{if } n == 1 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

$5! = 5 \times 4!$

$= 5 \times 4 \times 3!$

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!*** =

$$n! = \begin{cases} 1 & \text{if } n == 1 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

$5! = 5 \times 4!$

$= 5 \times 4 \times 3!$

$= 5 \times 4 \times 3 \times 2!$

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!*** =

$$n! = \begin{cases} 1 & \text{if } n == 1 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

$$\begin{aligned} 5! &= 5 \times 4! \\ &= 5 \times 4 \times 3! \\ &= 5 \times 4 \times 3 \times 2! \\ &= 5 \times 4 \times 3 \times 2 \times 1! \end{aligned}$$

Factorial

- **given**: positive integer ***n***
- **wanted**: factorial of ***n*** = ***n!*** =

$$n! = \begin{cases} 1 & \text{if } n == 1 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

5! = 5 x 4!
= 5 x 4 x 3!
= 5 x 4 x 3 x 2!
= 5 x 4 x 3 x 2 x 1!
= 5 x 4 x 3 x 2 x 1 = 120


factorial.py - /Users/thiebaut/Desktop/111b/factorial.py
def factorial(n):
# stopping condition
if n==1:
return 1
# recursive step
result = n * factorial(n-1)
return result
def main():
while True:
n = int(input("> "))
if n<=0:
break
print("{0:1}! = {1:1}".format(n, factorial(n)))
main()
Ln: 18 Col: 61

Demo Time!

Under the Hood:

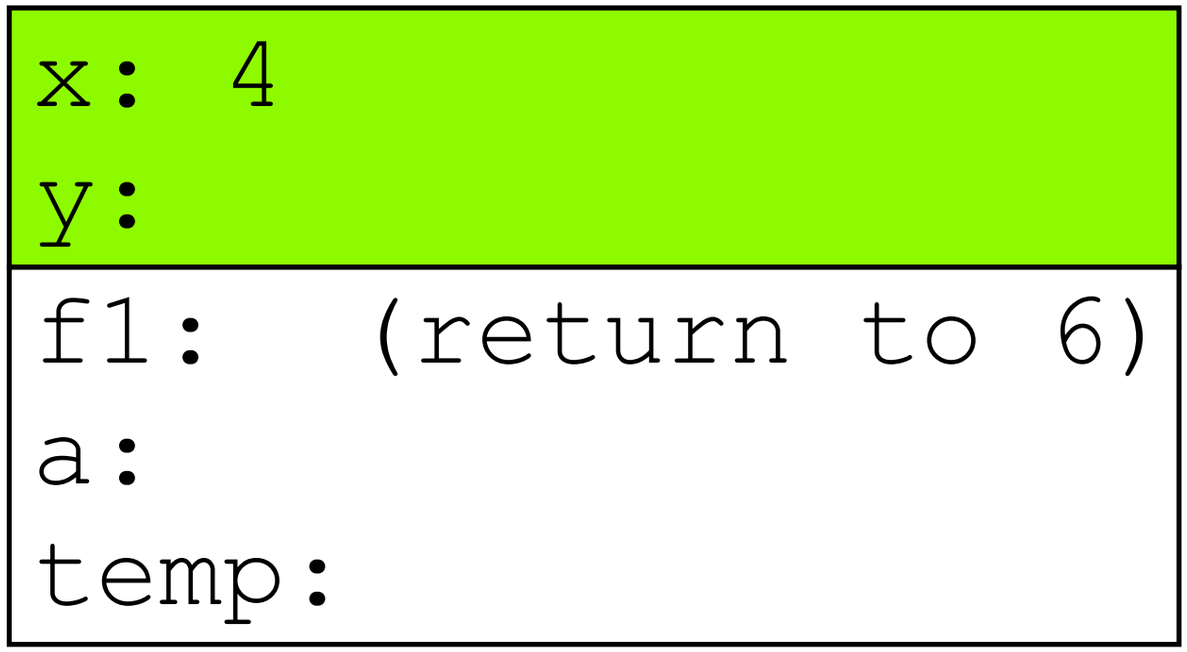
Animated Python Functions

```
1  def f1( a ):
2      temp = a*2
3      return temp
4
5  x = 4
6  y = f1( x )
7  print( y )
```

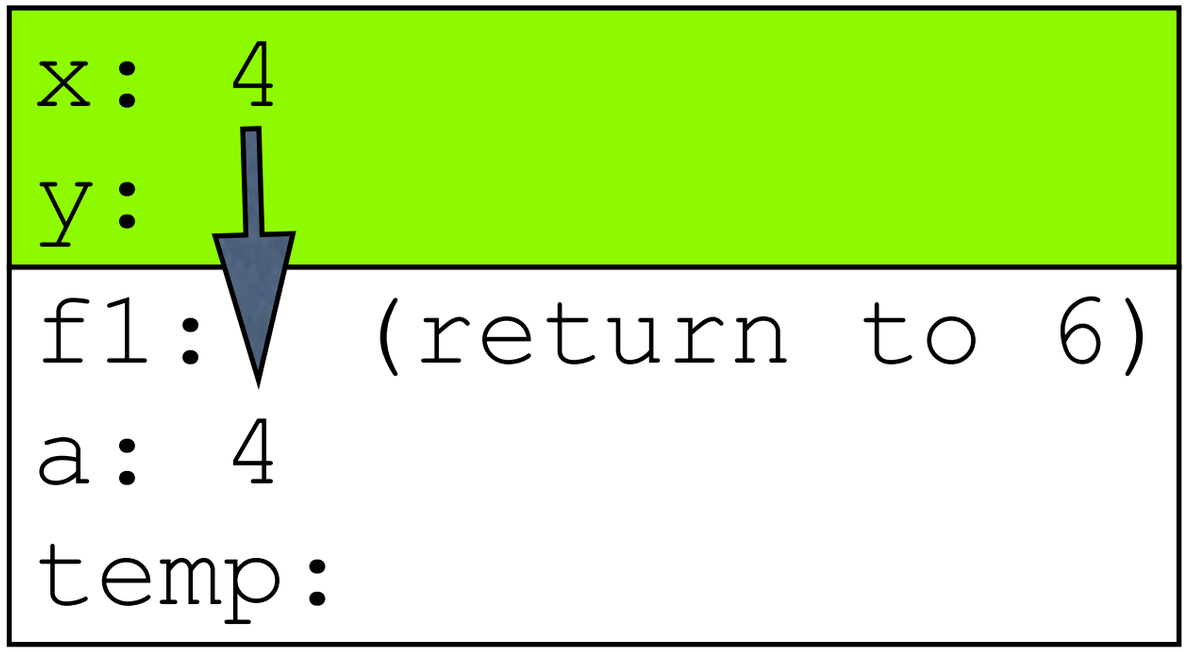
```
1   def f1( a ):
2       temp = a*2
3       return temp
4
5   x = 4
6→  y = f1( x )
7   print( y )
```

```
x:  4
y:
```

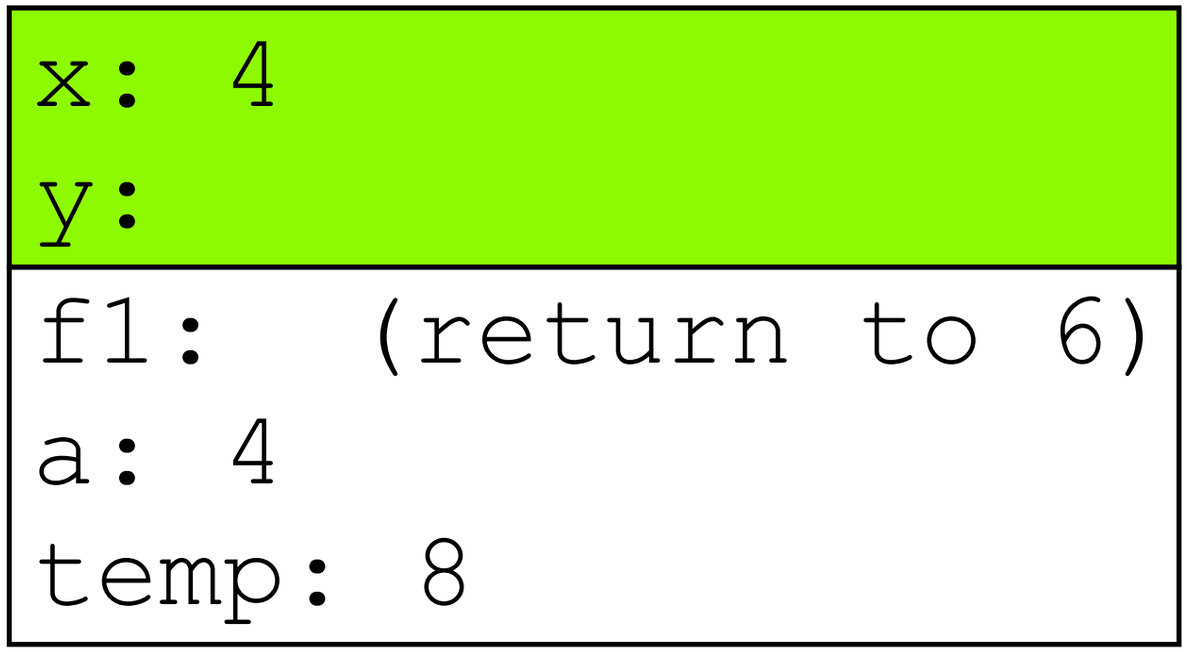
```
→ def f1( a ):
2       temp = a*2
3       return temp
4
5   x = 4
6   y = f1( x )
7   print( y )
```



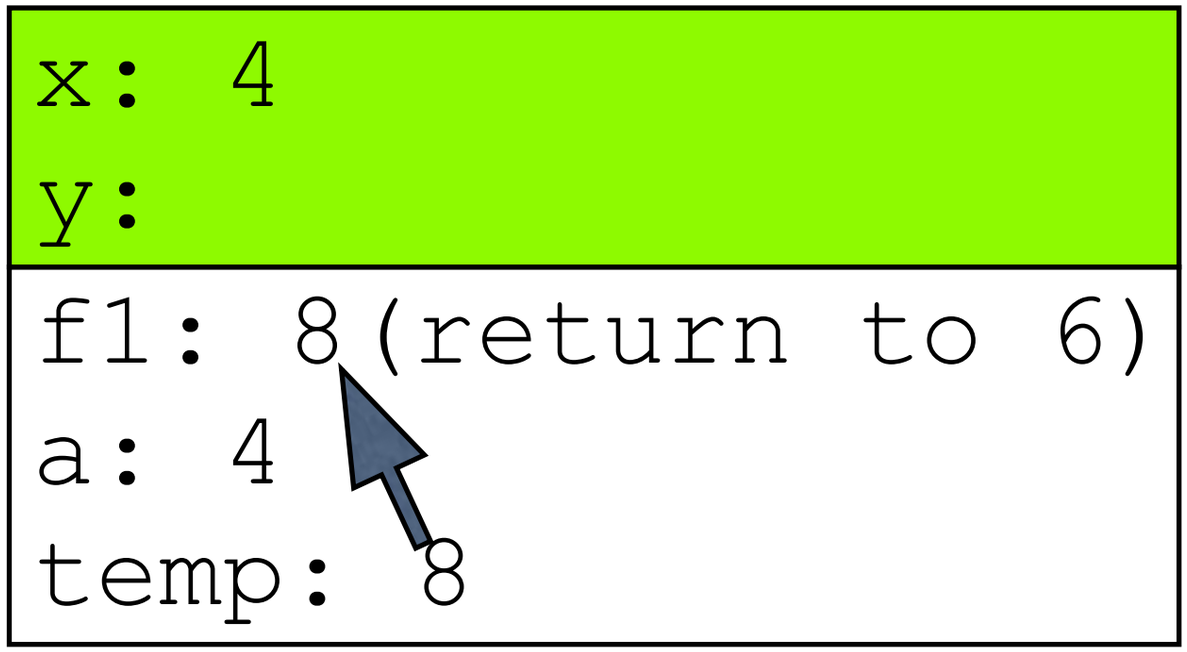
```
1 def f1( a ):
2     temp = a*2
3     return temp
4
5 x = 4
6 y = f1( x )
7 print( y )
```



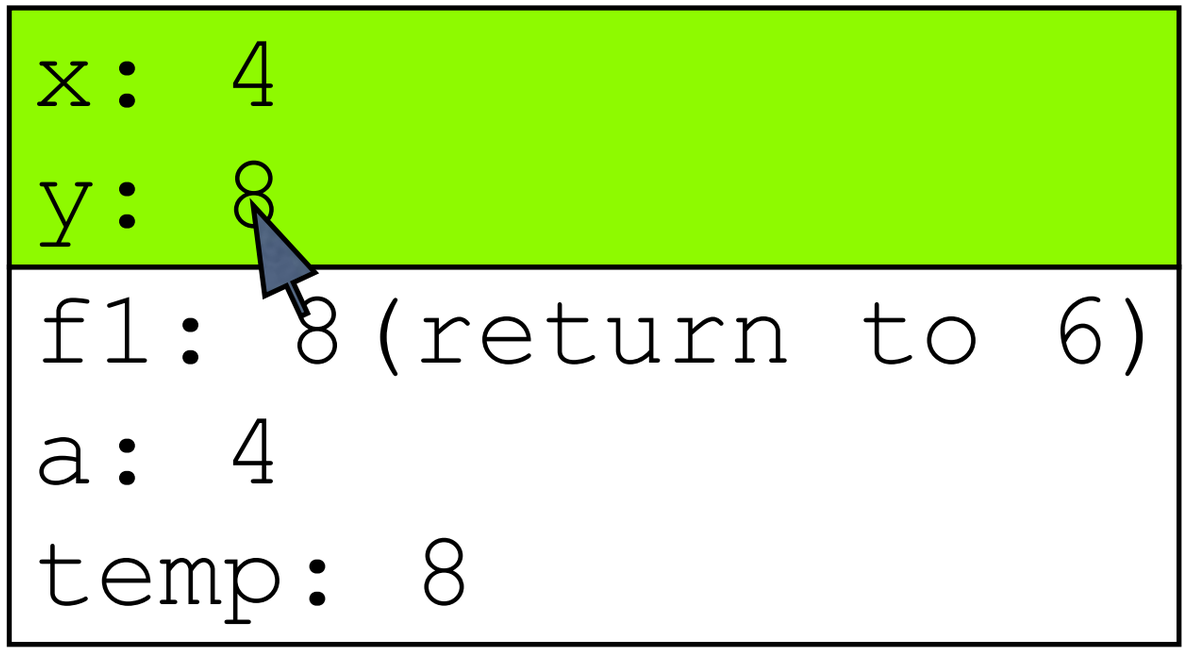
```
1   def f1( a ):
2    → temp = a*2
3       return temp
4
5   x = 4
6   y = f1( x )
7   print( y )
```



```
1   def f1( a ):
2       temp = a*2
3    → return temp
4
5   x = 4
6   y = f1( x )
7   print( y )
```



```
1   def f1( a ):
2       temp = a*2
3       return temp
4
5   x = 4
6→  y = f1( x )
7   print( y )
```



```
1   def f1( a ):
2       temp = a*2
3       return temp
4
5   x = 4
6→  y = f1( x )
7   print( y )
```

```
x: 4
y: 8
```

```
1   def f1( a ):
2       temp = a*2
3       return temp
4
5   x = 4
6   y = f1( x )
7→  print( y )
```

```
x: 4
y: 8
```

```
1   def f1( a ):
2       temp = a*2
3       return temp
4
5   x = 4
6   y = f1( x )
7→  print( y )
```

```
x:  4
y:  8
```

8

We stopped here last time…

Recursion: Factorial

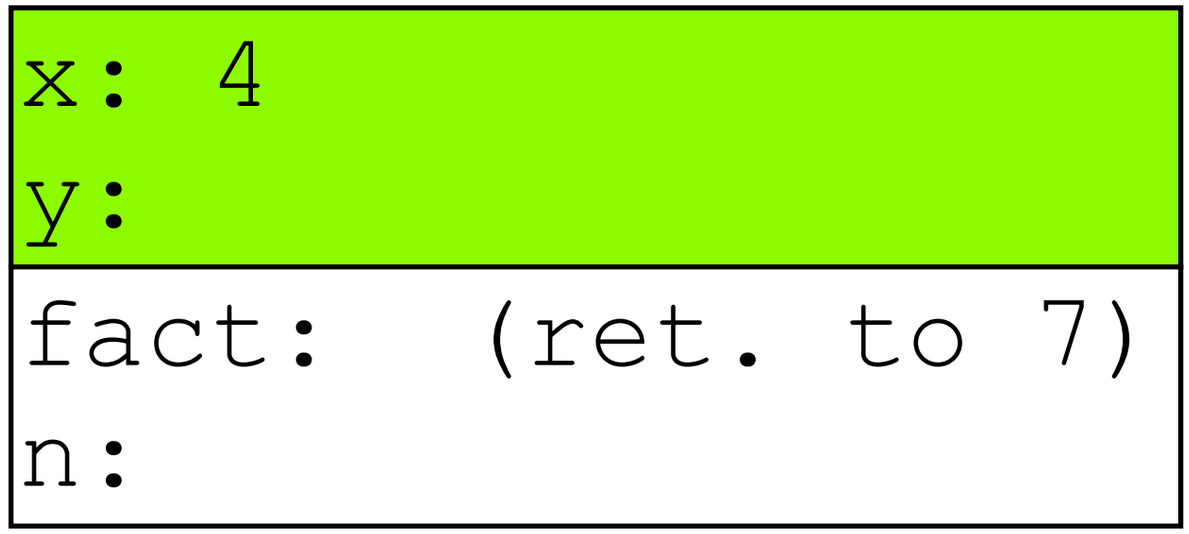
```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```

```
x:  4
y:
```

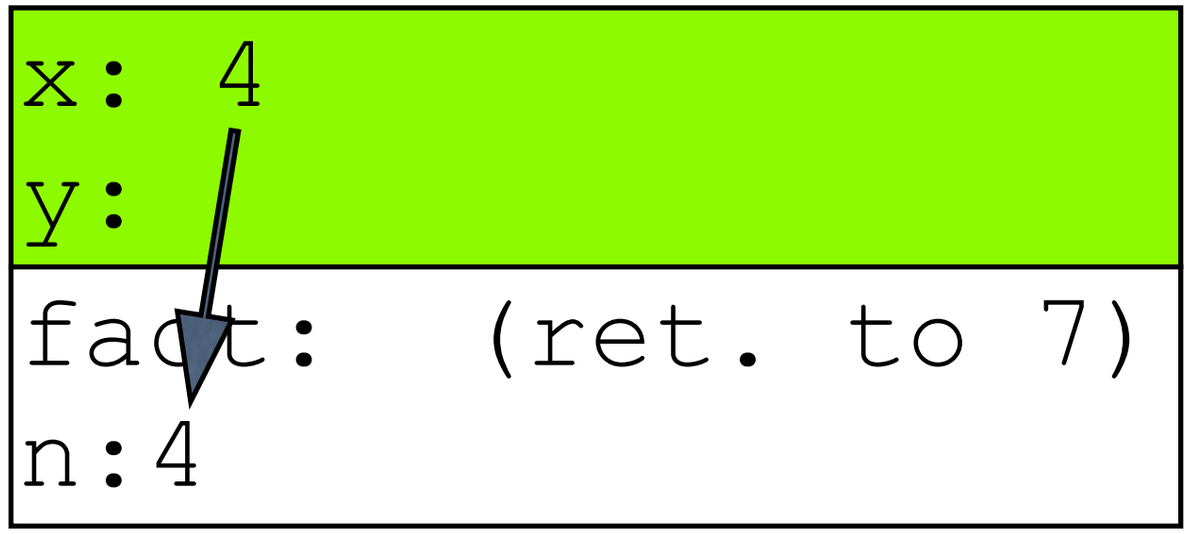
```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```

```
x: 4
y:
```

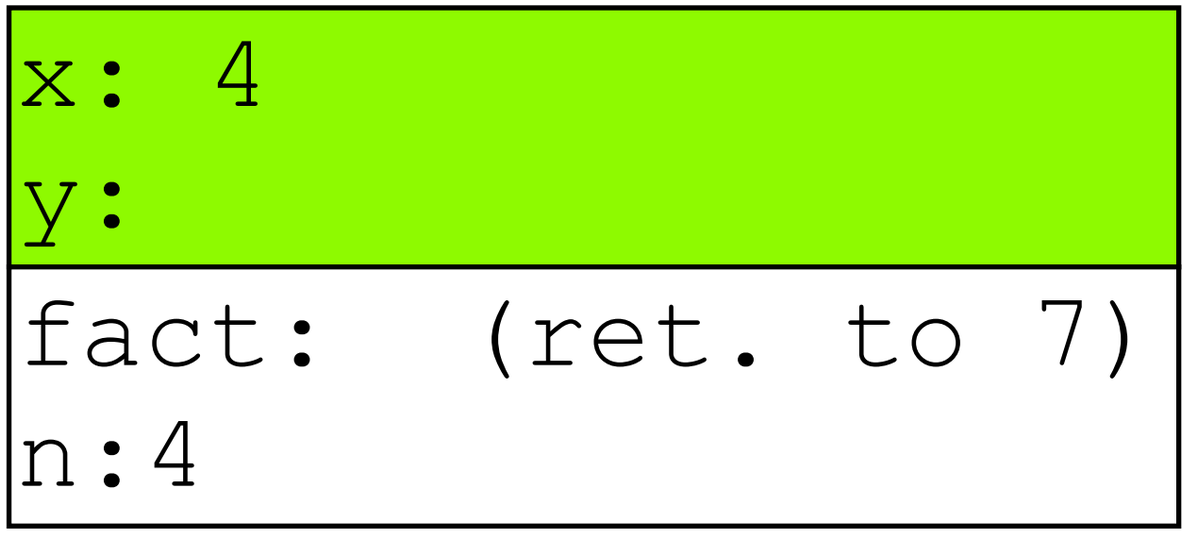
```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



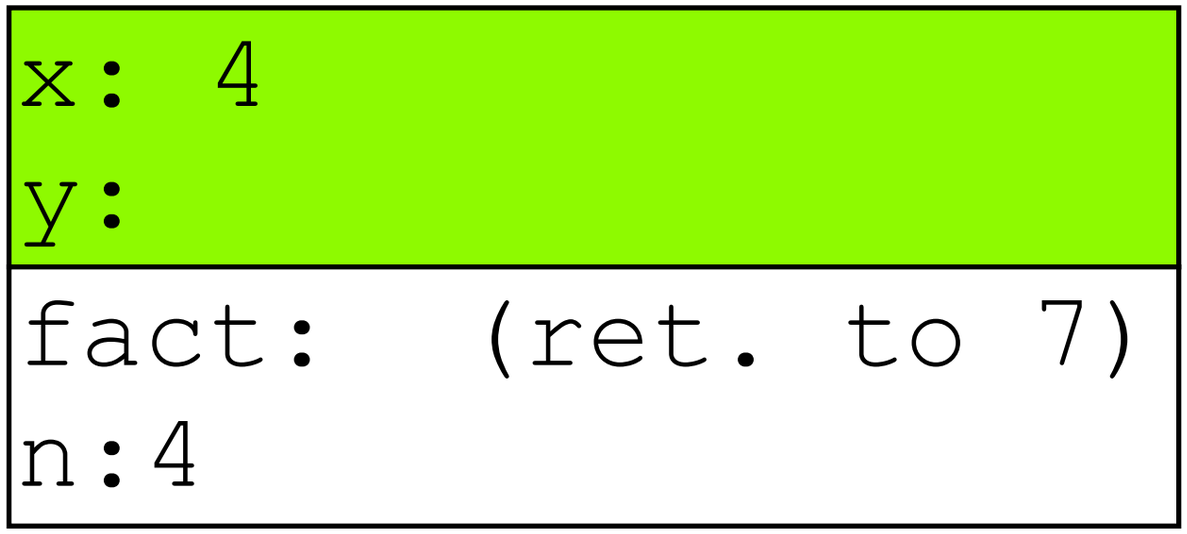
```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



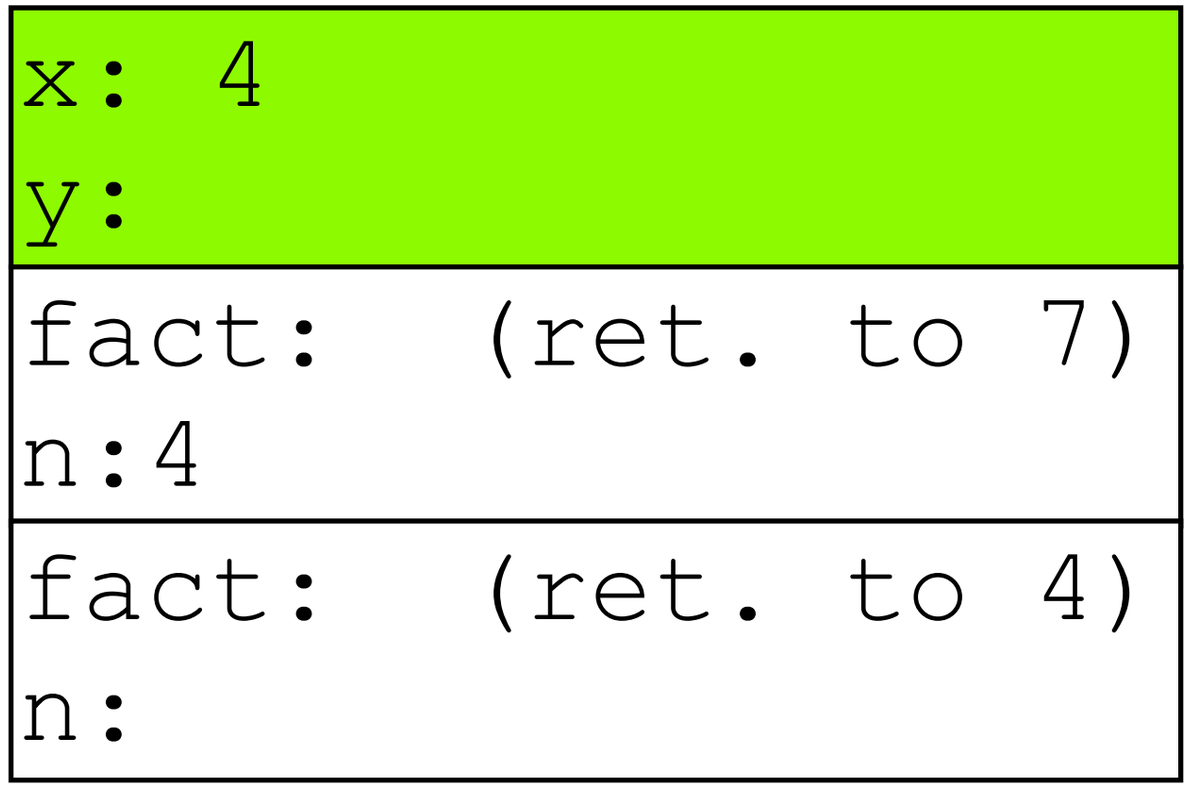
```
1 def fact( n ):
2  →if n<=1:
3        return 1
4    return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



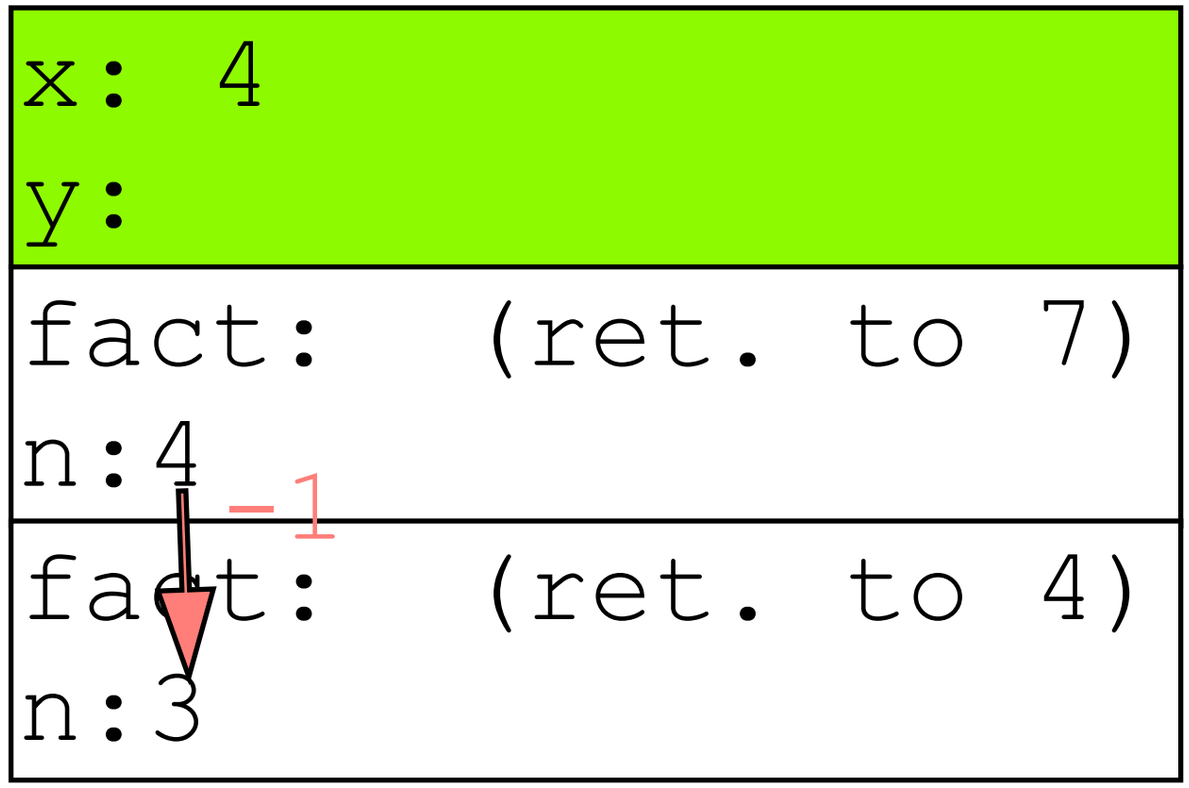
```
1 def fact( n ):
2     if n<=1:
3         return 1
4  →  return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



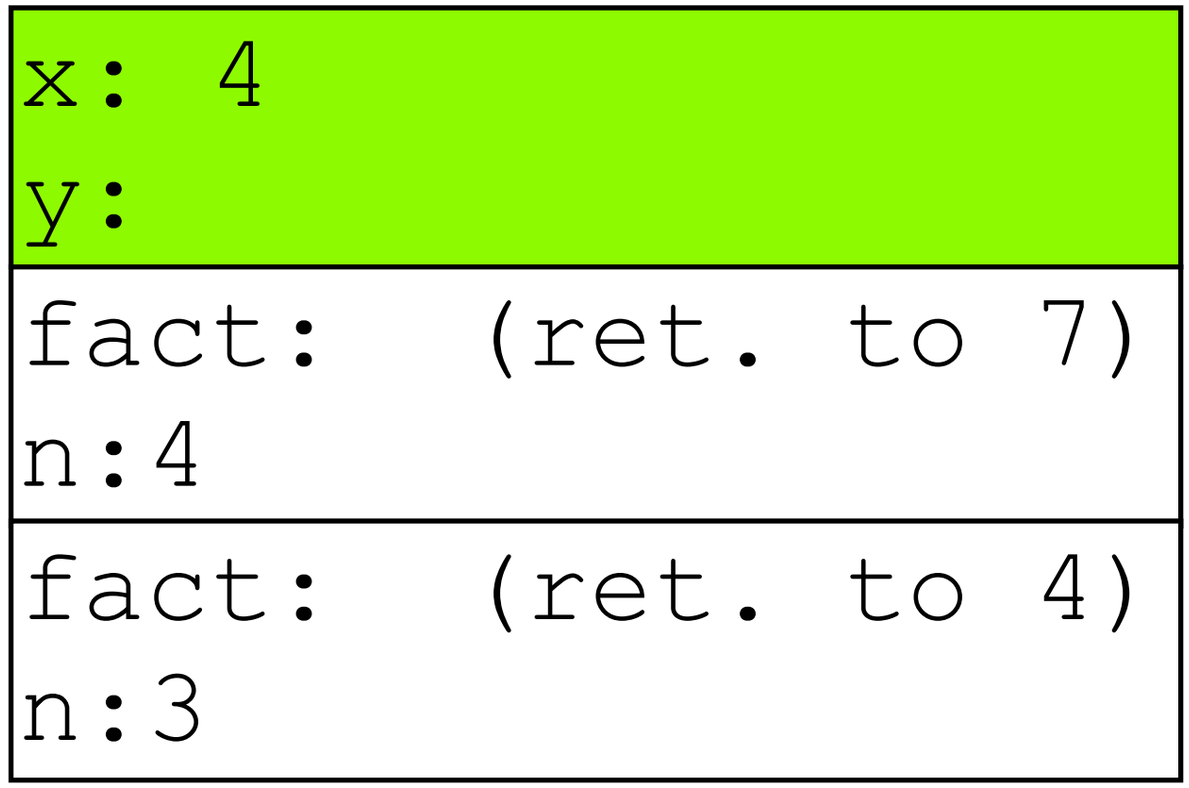
```
1 def fact( n ):
2     if n<=1:
3         return 1
4  →  return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



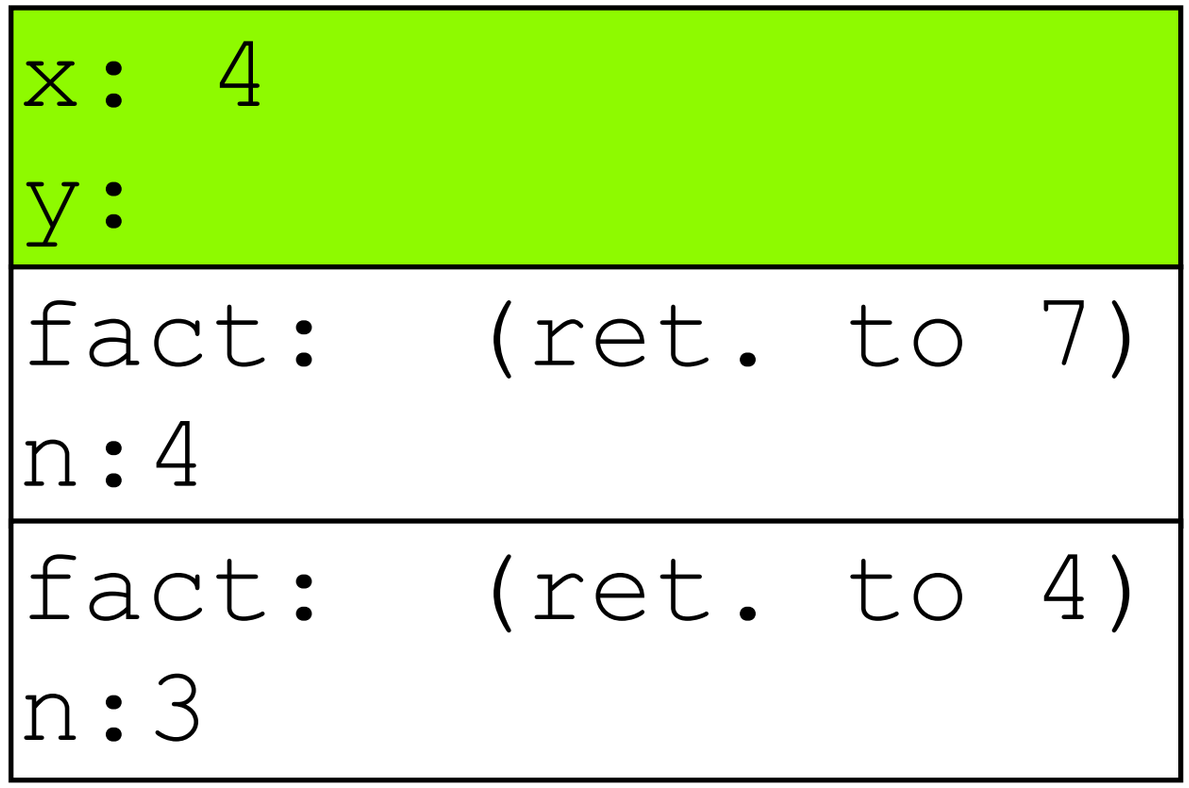
```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



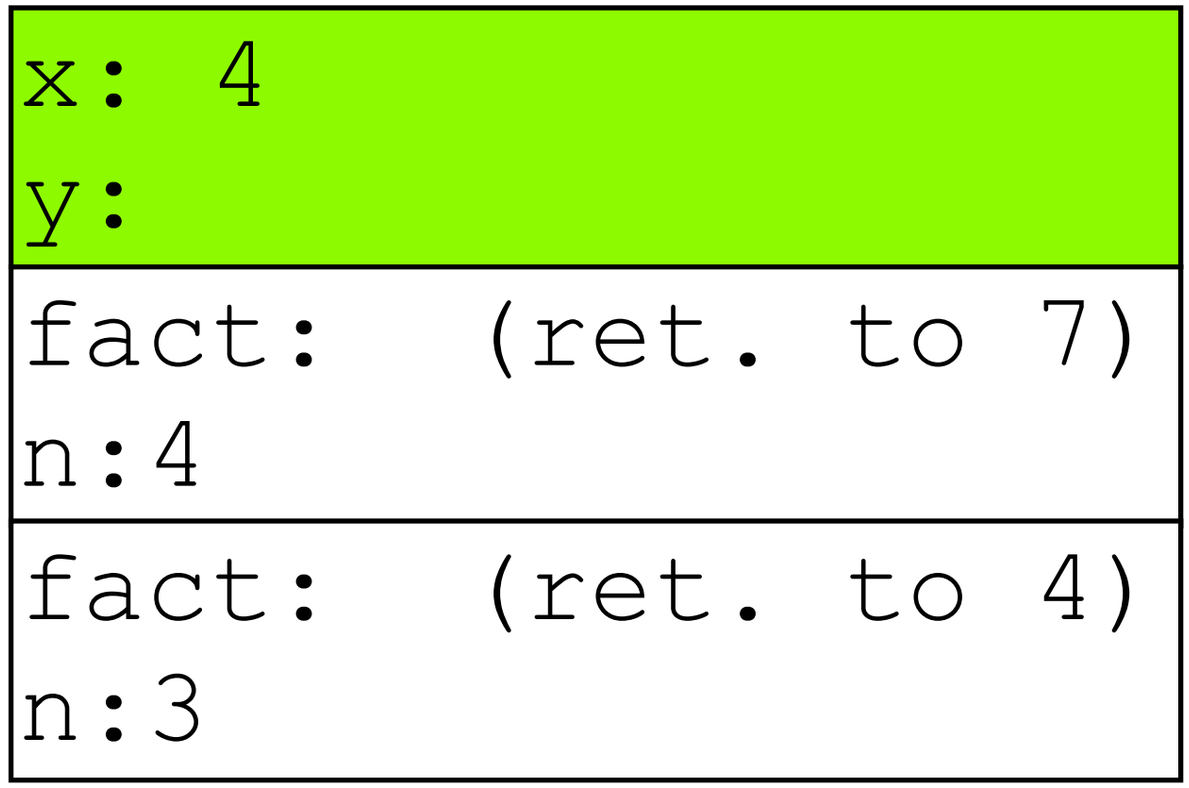
```
1 def fact( n ):
2 →   if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



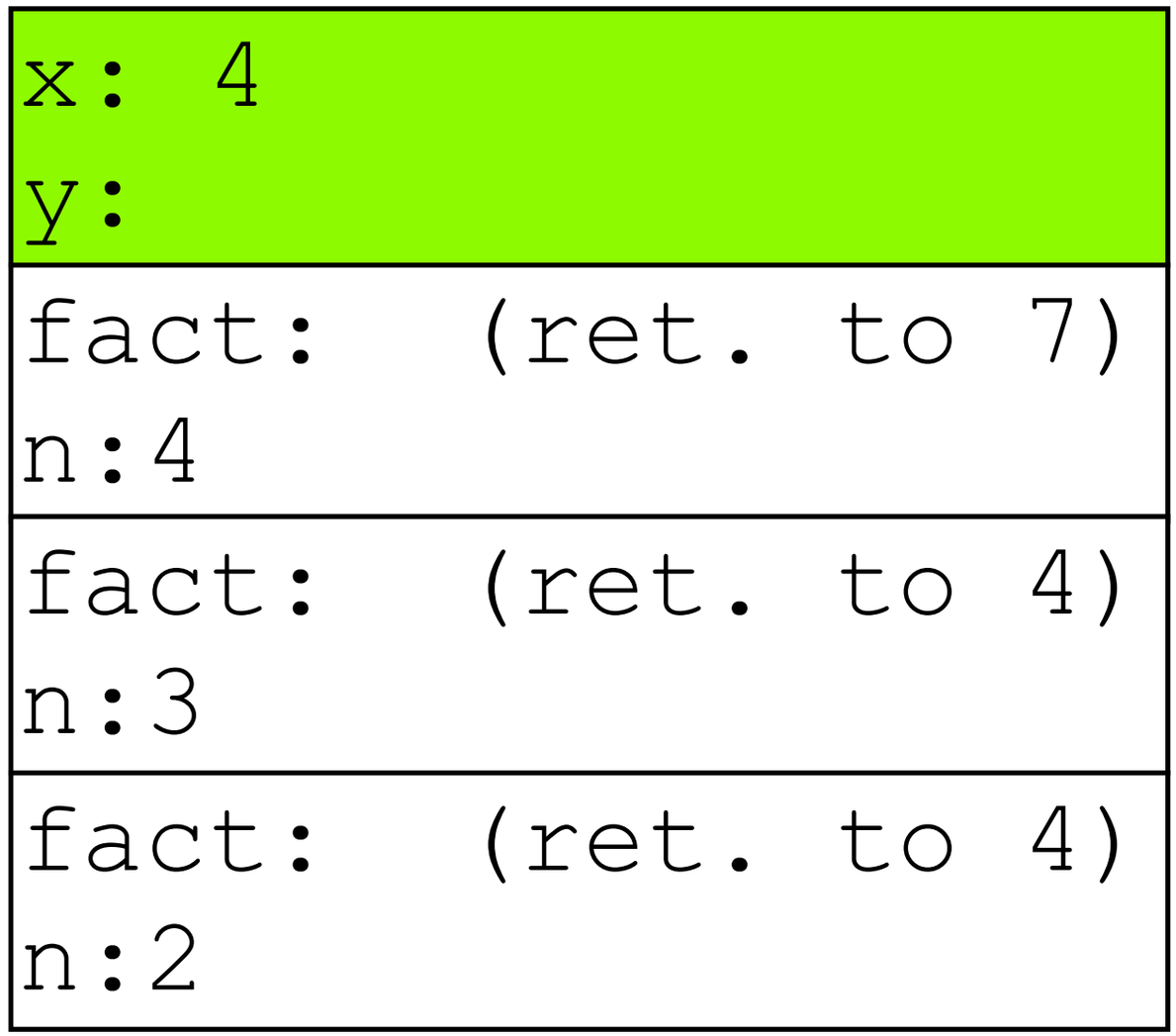
```
1 def fact( n ):
2 →   if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



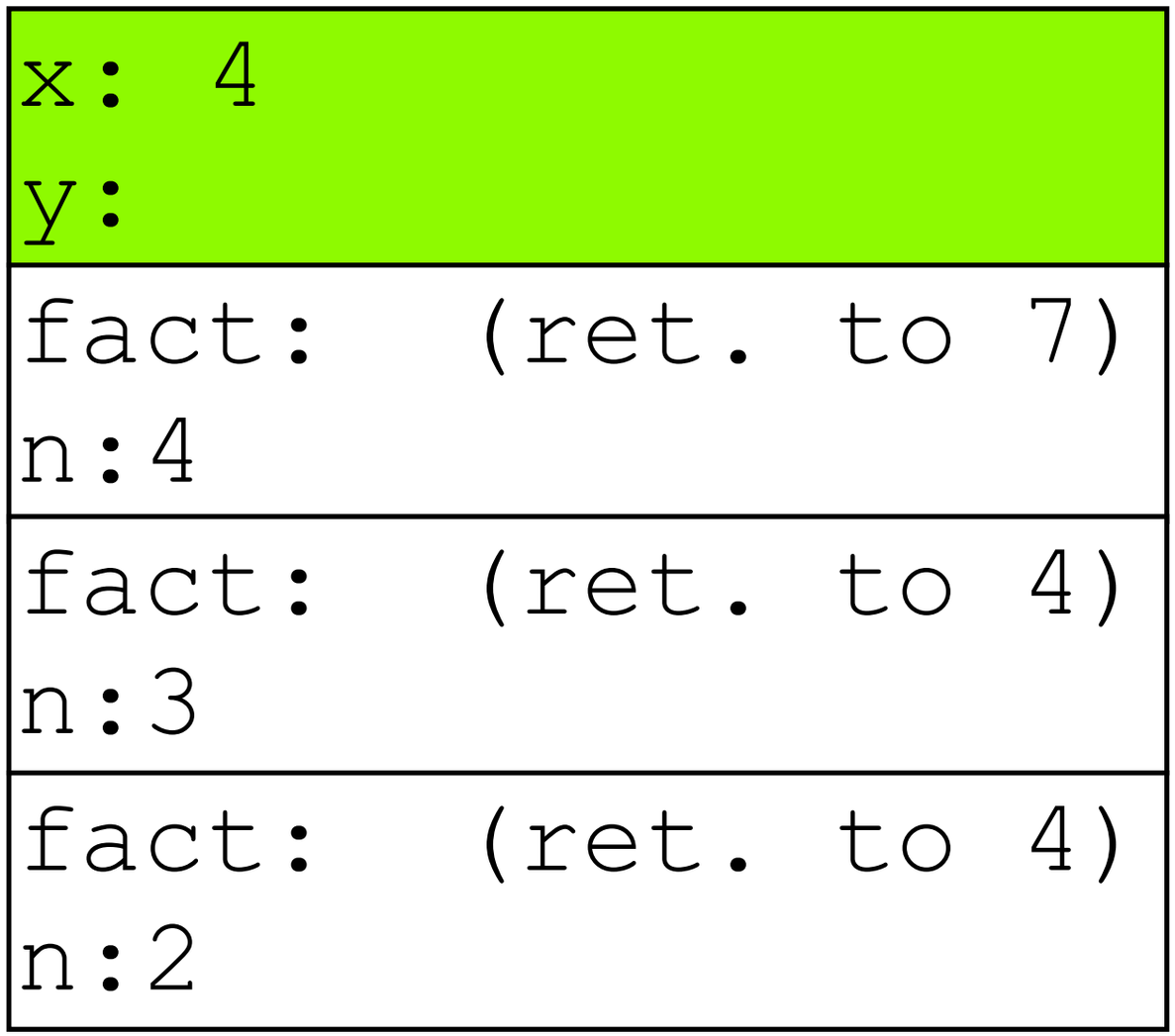
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 → return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



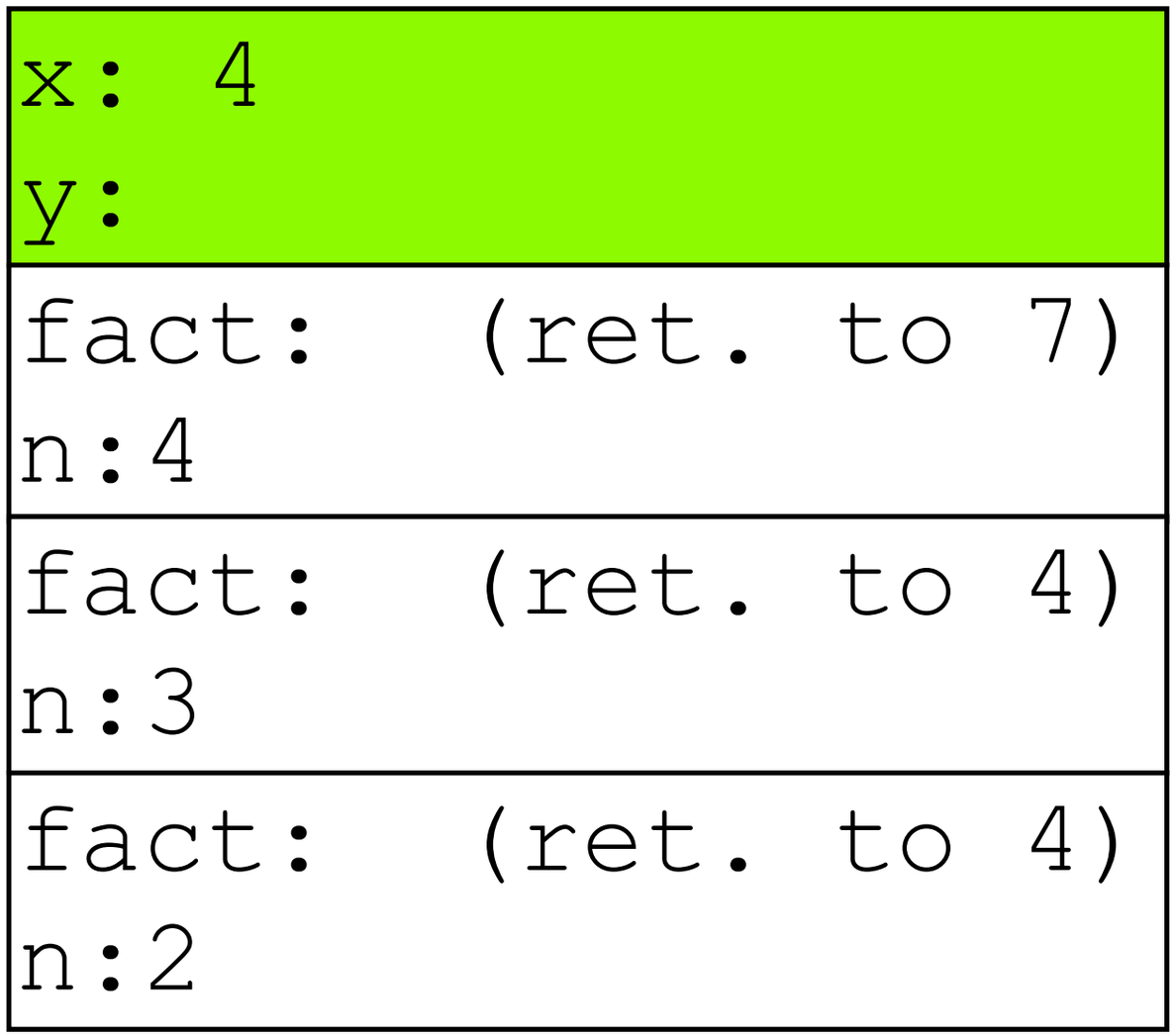
```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



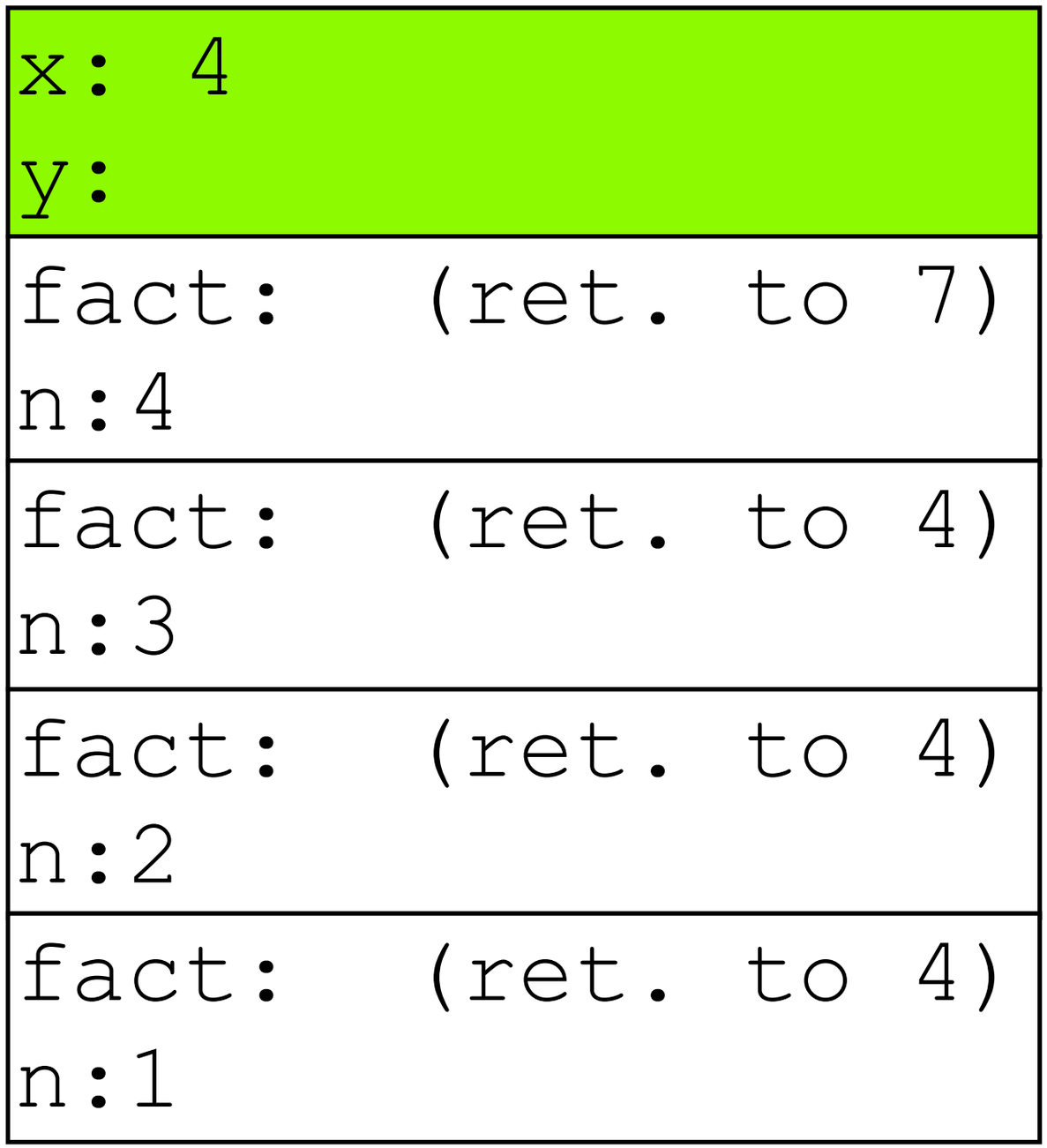
```
1 def fact( n ):
2  →  if n<=1:
3          return 1
4      return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



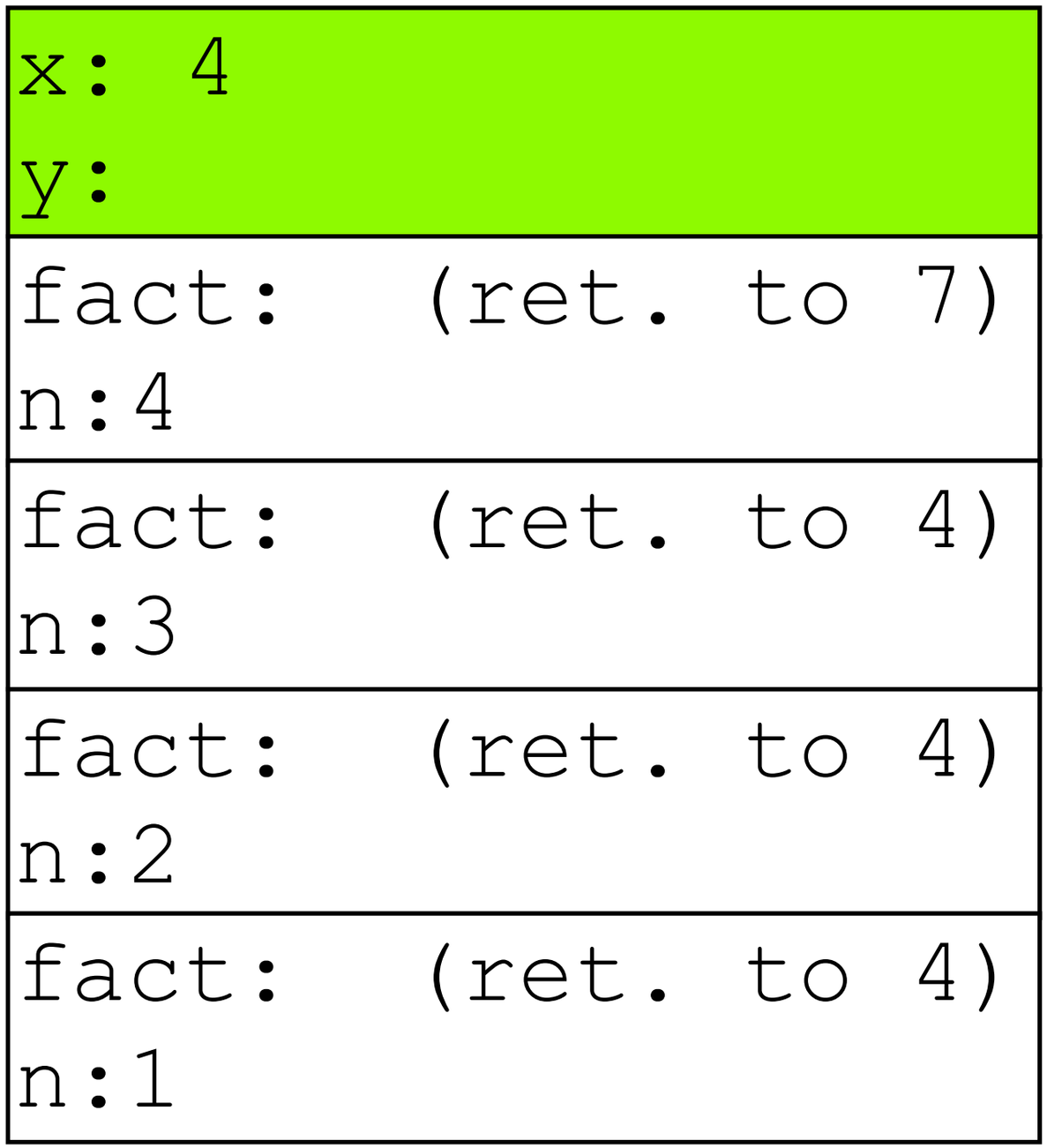
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 →   return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



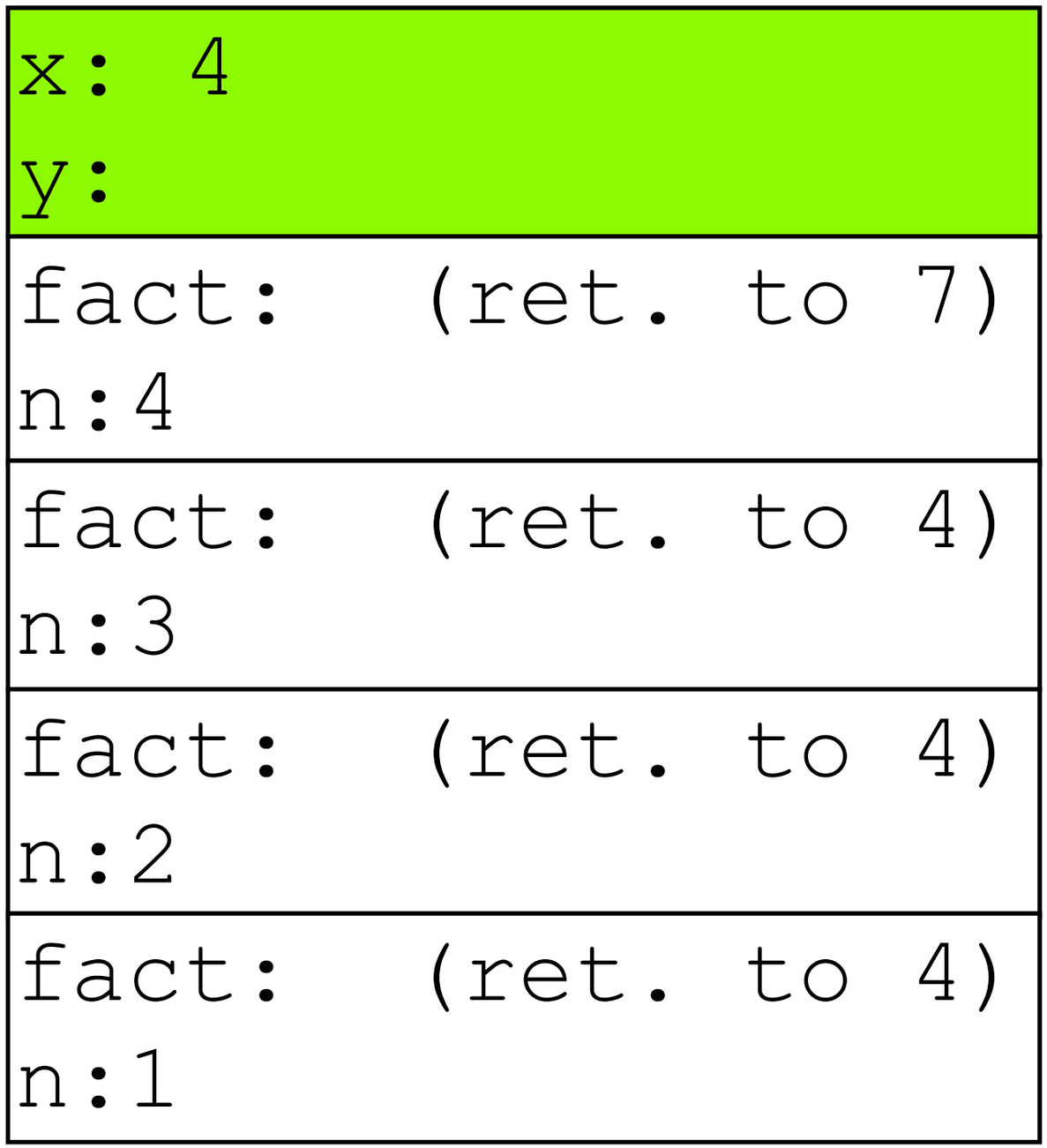
```
→1 def fact( n ):
 2     if n<=1:
 3         return 1
 4     return n*fact(n-1)
 5
 6 x = 4
 7 y = fact( x )
 8 print( y )
```



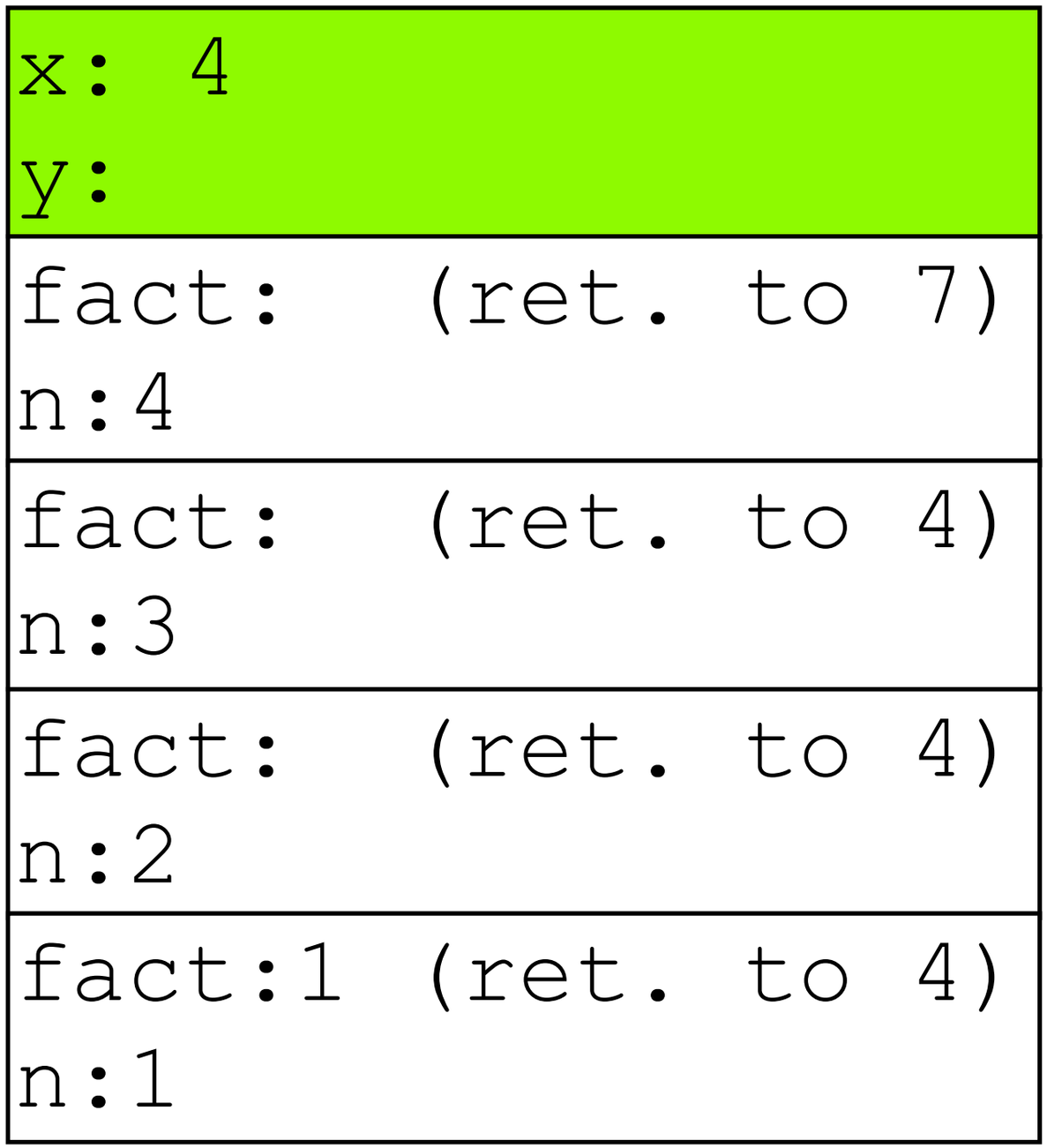
```
1 def fact( n ):
2 →   if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



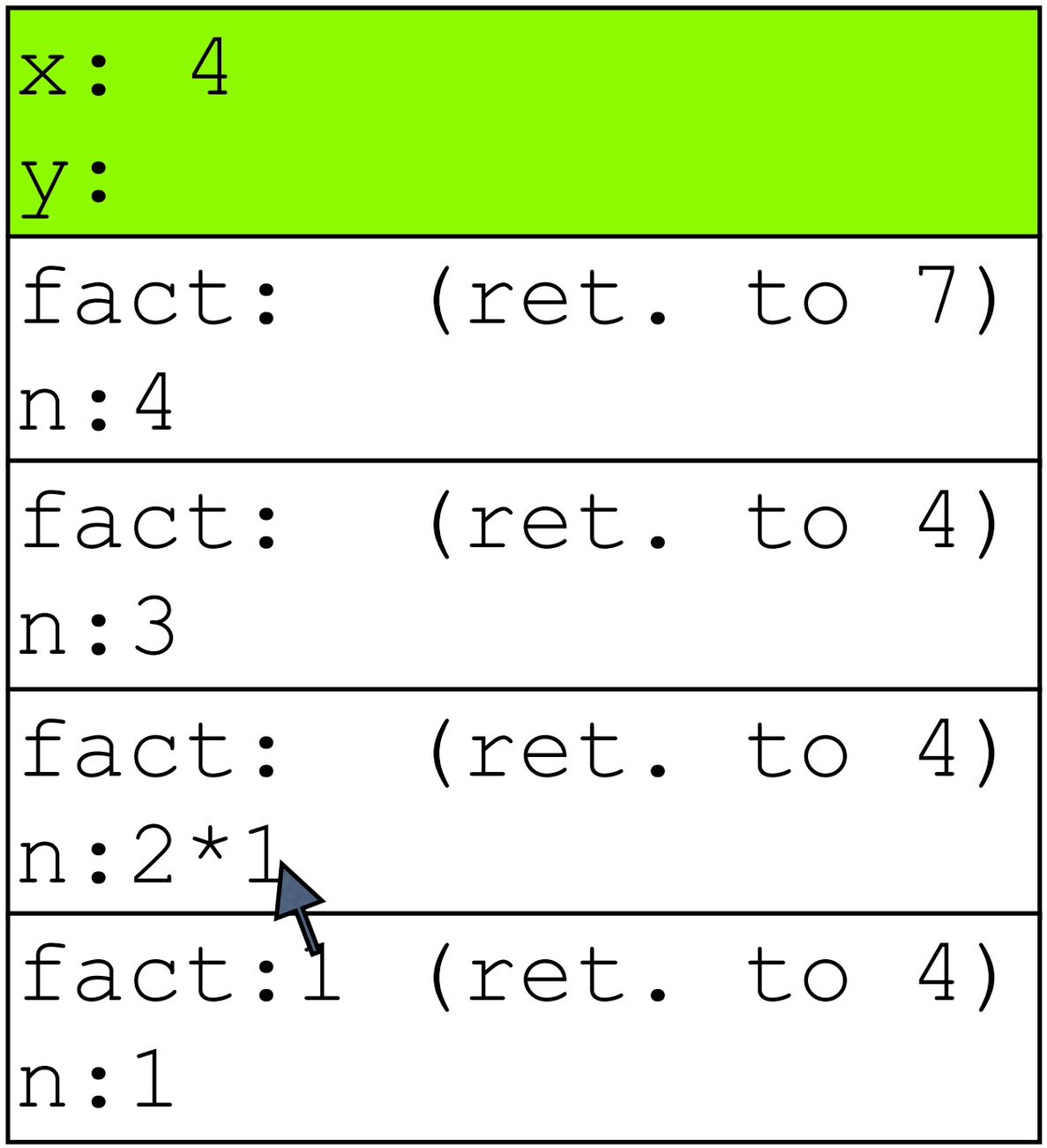
```
1 def fact( n ):
2     if n<=1:
3         → return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



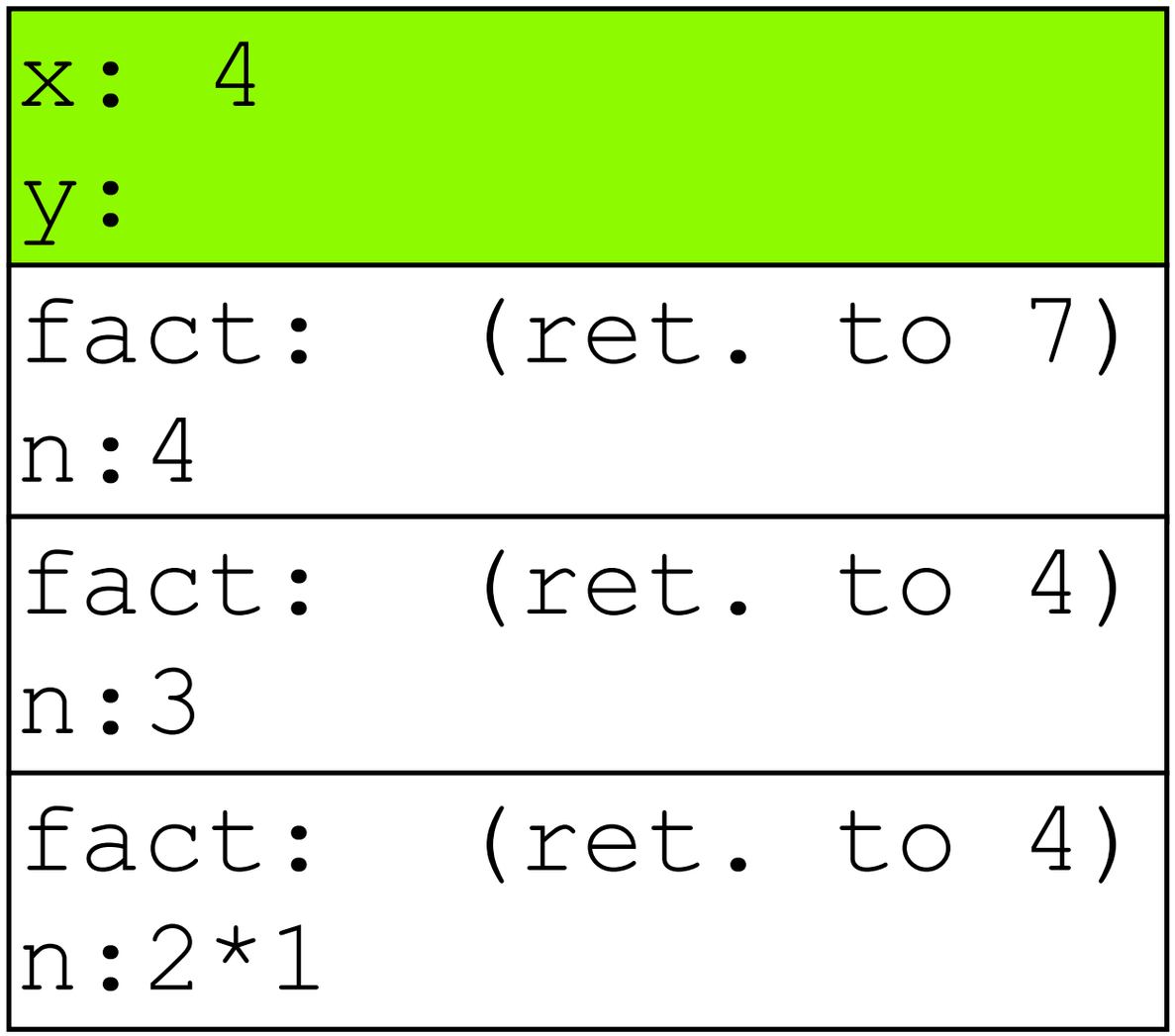
```
1 def fact( n ):
2     if n<=1:
3         → return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



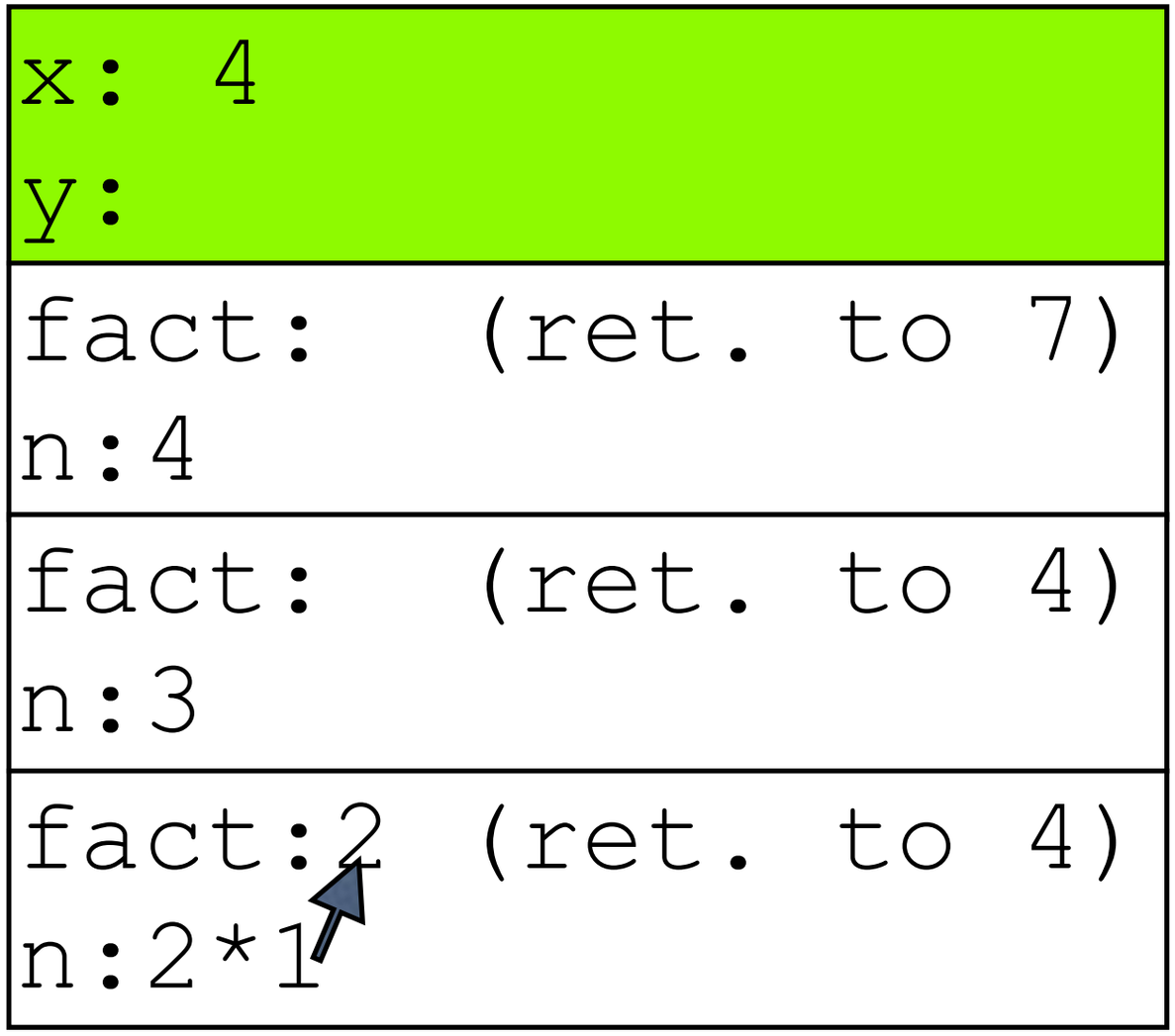
```
1 def fact( n ):
2     if n<=1:
3         return 1
4  →  return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



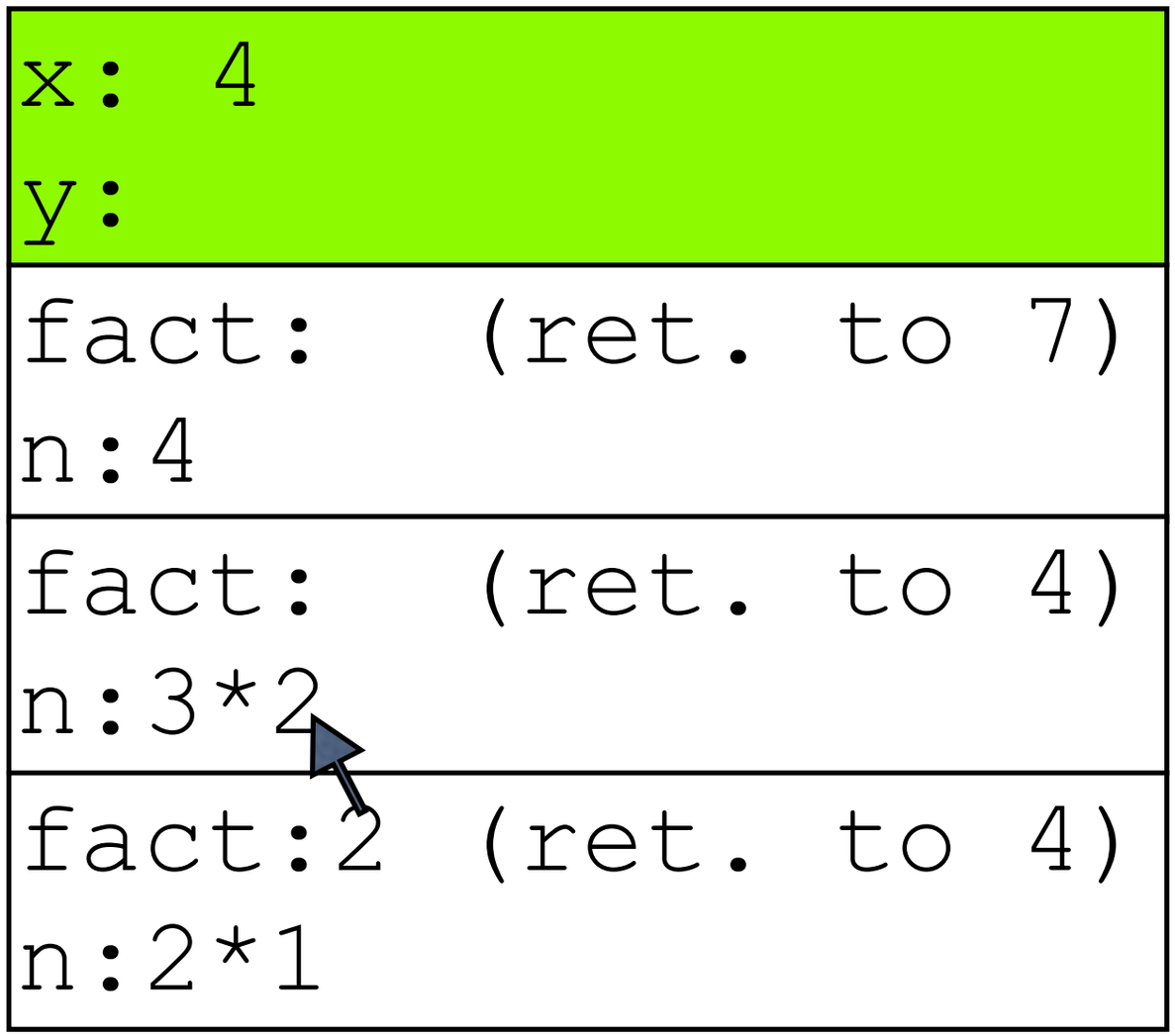
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 → return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



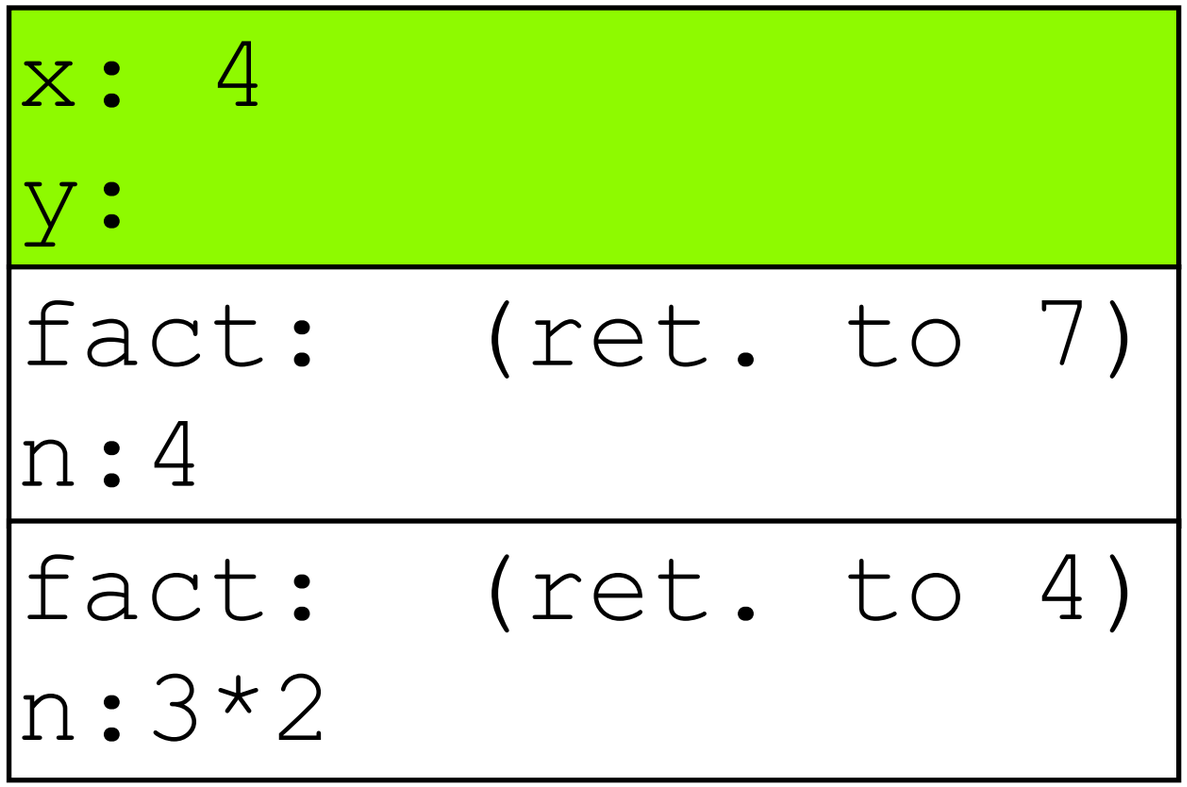
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 → return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



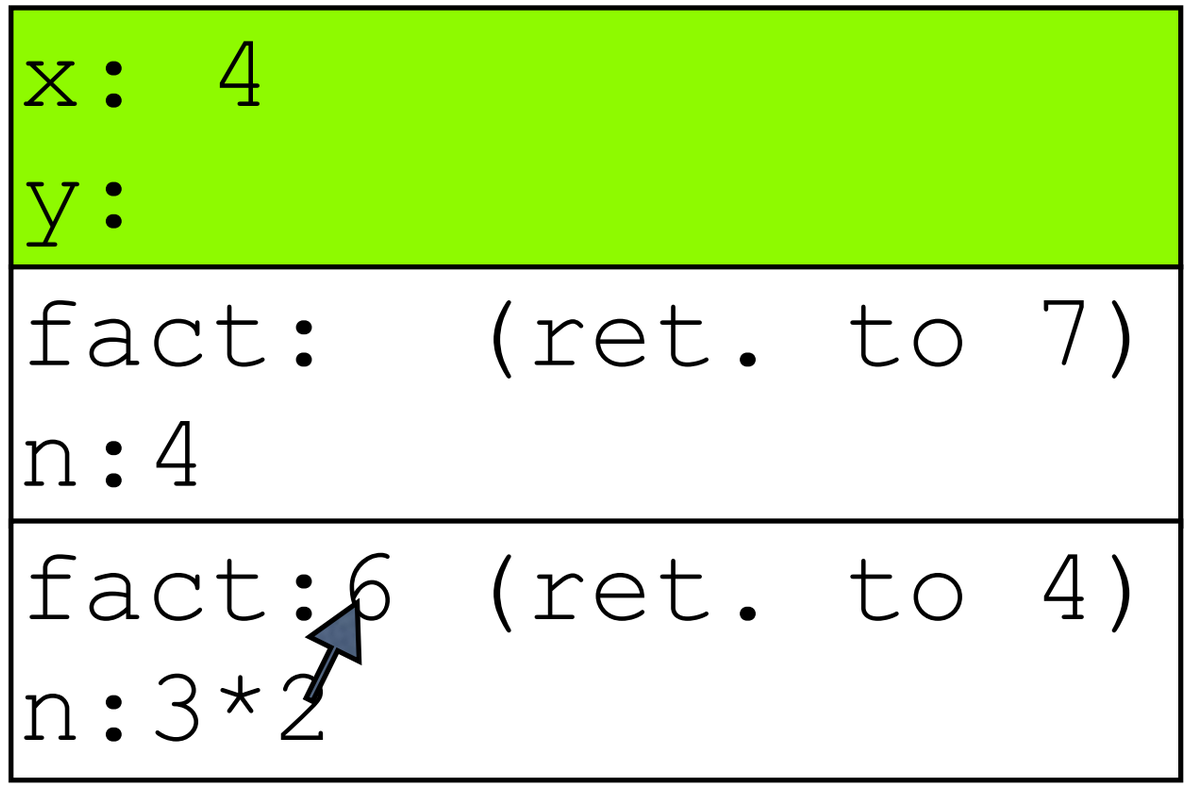
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 → return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



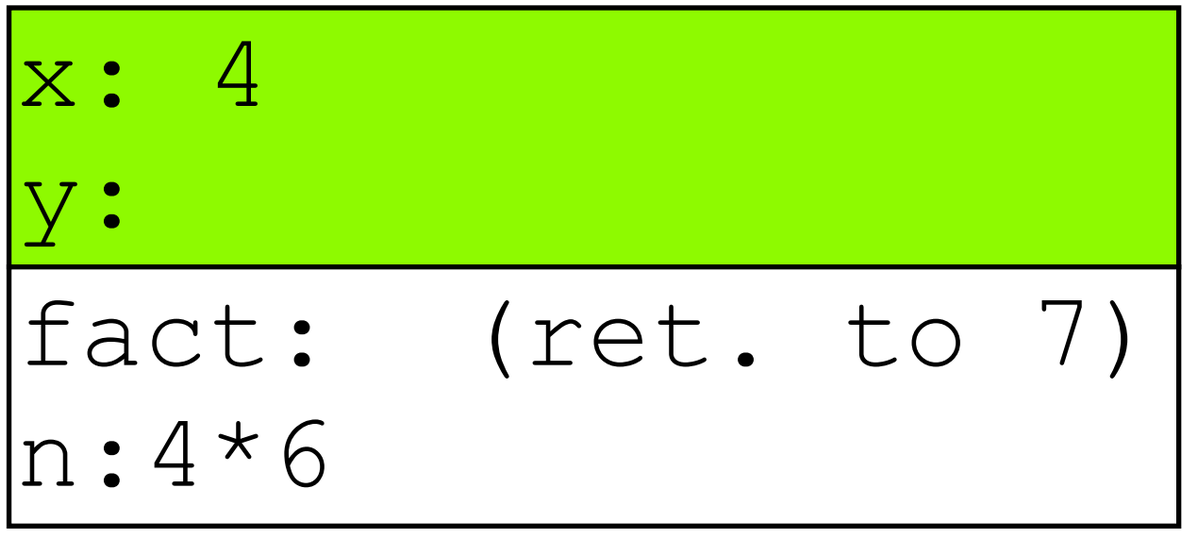
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 → return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



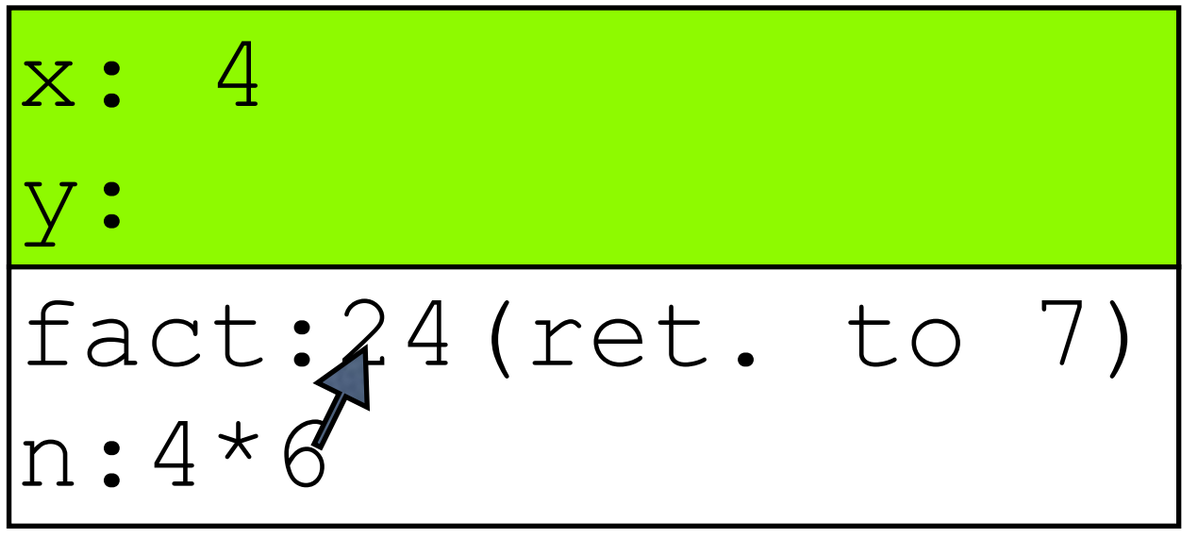
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 → return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



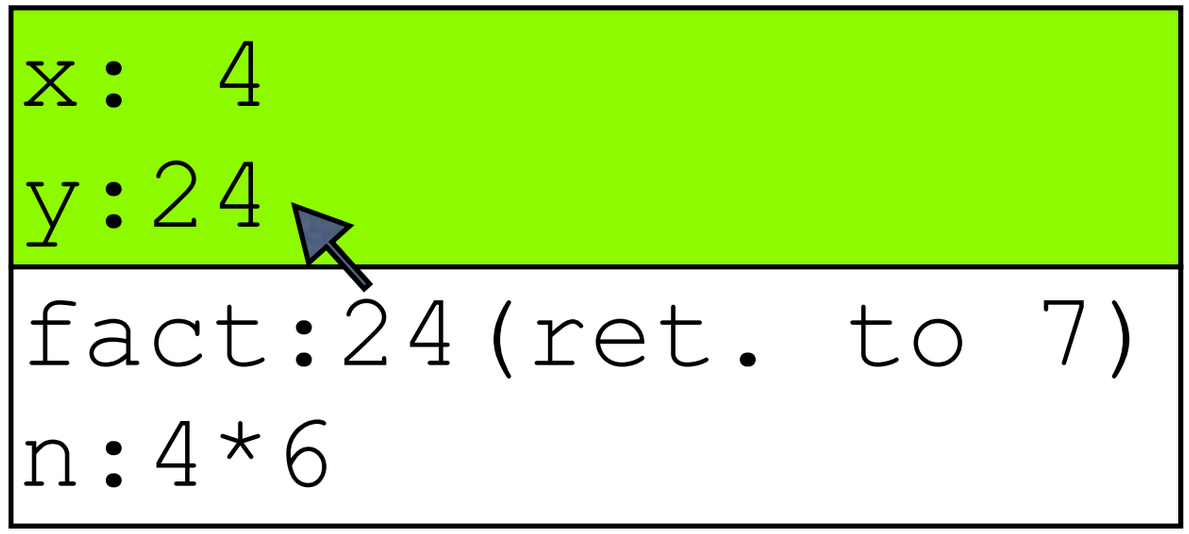
```
1 def fact( n ):
2     if n<=1:
3         return 1
4 → return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



```
1 def fact( n ):
2     if n<=1:
3         return 1
4 →   return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```



```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```

```
x: 4
y:24
```

```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
→8 print( y )
```

```
x: 4
y:24
```

```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
→8 print( y )
```

```
x: 4
y:24
```

24

```
1 def fact( n ):
2     if n<=1:
3         return 1
4     return n*fact(n-1)
5
6 x = 4
7 y = fact( x )
8 print( y )
```

24

Minions: Can you tell me if there's a banana that weighs 118 grams?

Minions: Here's a bunch of bananas. Return to me the smallest and the largest.

We stopped here last time…