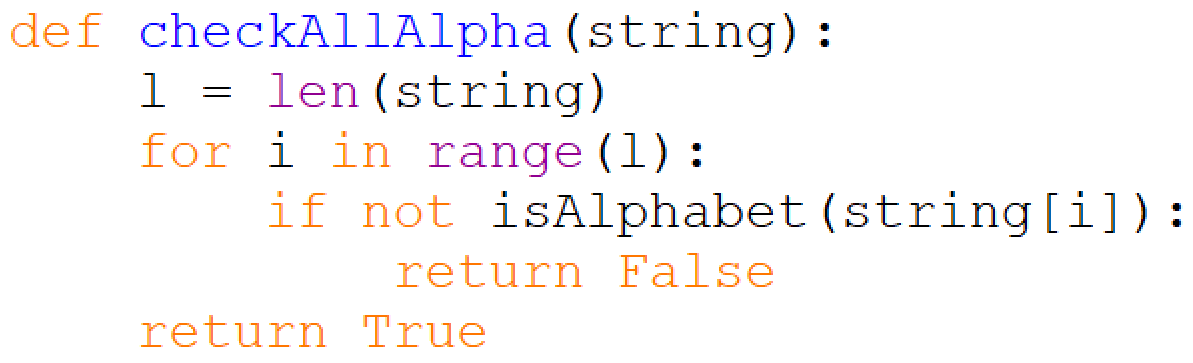
Checking Prime Numbers

More about Repetitions and Selections

Recap: Check if a String is all alphabet

- Combining
  - You check the character one-by-one
    - If the current one is NOT alphabet, return “No”!
  - Until finishing all character all checked, return “Yes”



```python
def checkAllAlpha(string):
    l = len(string)
    for i in range(l):
        if not isAlphabet(string[i]):
            return False
    return True
```

How to check if a number n is prime?

- First, in real life, how to we check?
    - Try writing down in English?

- For all the numbers i from 2 to n-1
    - We check if n is divisible by i
        - If divisible? Or not? What should we do?

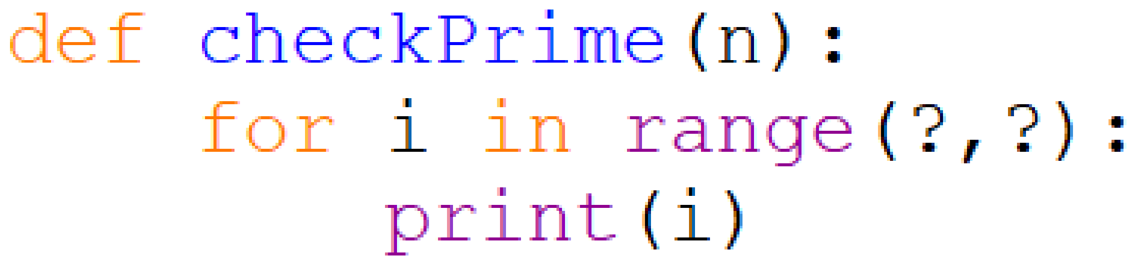
How to check if a number n is prime?

- First, in real life, how to we check?
    - Try writing down in English?

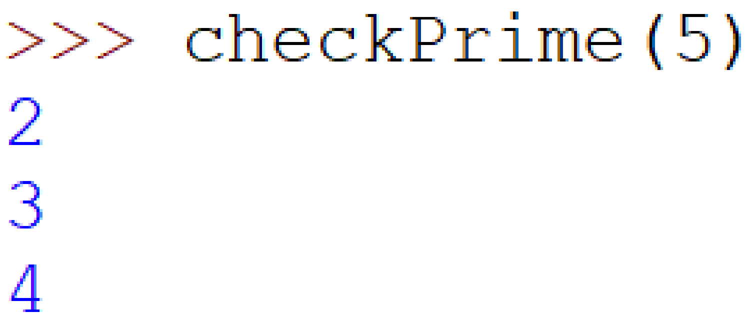
- For all the numbers i from 2 to n-1
    - We check if n is divisible by i
        - If divisible
            - Not prime! (Can we stop now?)
        - If not divisible
            - Continue to the next number i
- Finally, if all numbers i are not divisible, then n is prime!

How to check if a number n is prime?

- Plan to work out the program?
  - If you are ready, go ahead and program
  - But for safety? Let’s try something easier first, maybe….
    - **Printing out all the numbers from i to n-1**
    - Write a function divisible(n,m) to check if n is divisible by m



```
def checkPrime(n):
    for i in range(?,?):
        print(i)
```

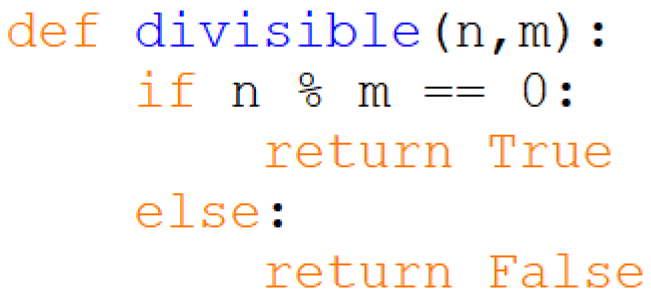


```
>>> checkPrime(5)
2
3
4
```

- What are the two “?”?

How to check if a number n is prime?

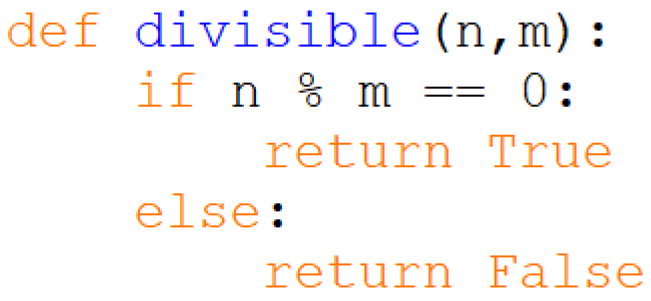
- Plan to work out the program?
  - If you are ready, go ahead and program
  - But for safety? Let’s try something easier first, maybe….
    - Printing out all the numbers from i to n-1
    - **Write a function divisible(n,m) to check if n is divisible by m**



```
def divisible(n,m):
    if n % m == 0:
        return True
    else:
        return False
```

How to check if a number n is prime?

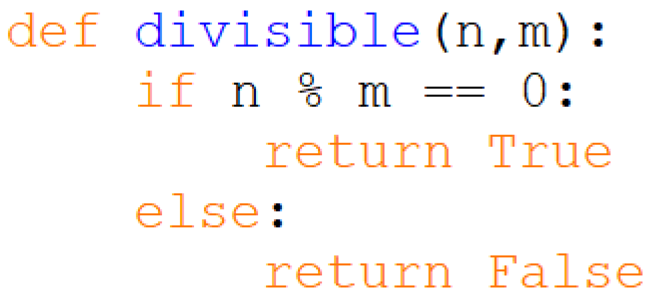
- Plan to work out the program?
  - If you are ready, go ahead and program
  - But for safety? Let’s try something easier first, maybe….
    - Printing out all the numbers from i to n-1
    - **Write a function divisible(n,m) to check if n is divisible by m**



```
def divisible(n,m):
    if n % m == 0:
        return True
    else:
        return False
```

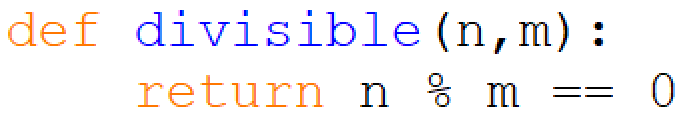
How to check if a number n is prime?

- Plan to work out the program?
  - If you are ready, go ahead and program
  - But for safety? Let’s try something easier first, maybe….
    - Printing out all the numbers from i to n-1
    - **Write a function divisible(n,m) to check if n is divisible by m**



```
def divisible(n,m):
    if n % m == 0:
        return True
    else:
        return False
```

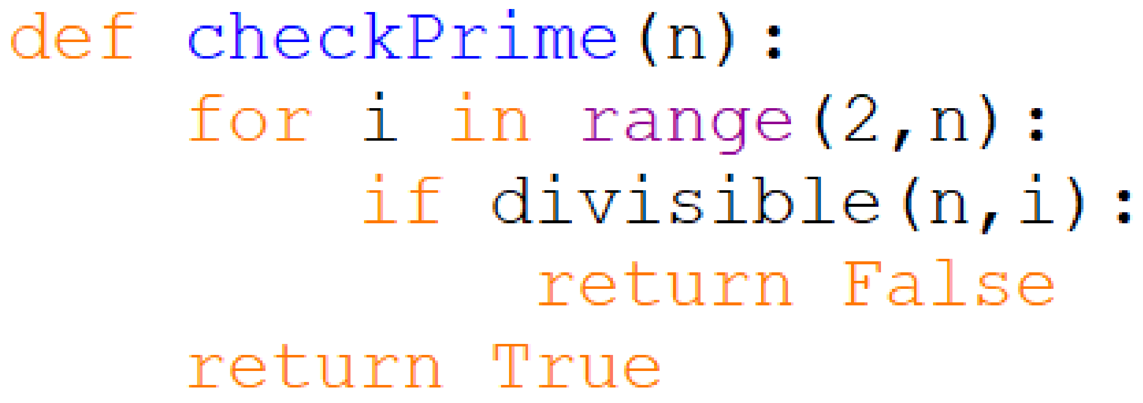
- Improvement?



```
def divisible(n,m):
    return n % m == 0
```

How to check if a number n is prime?

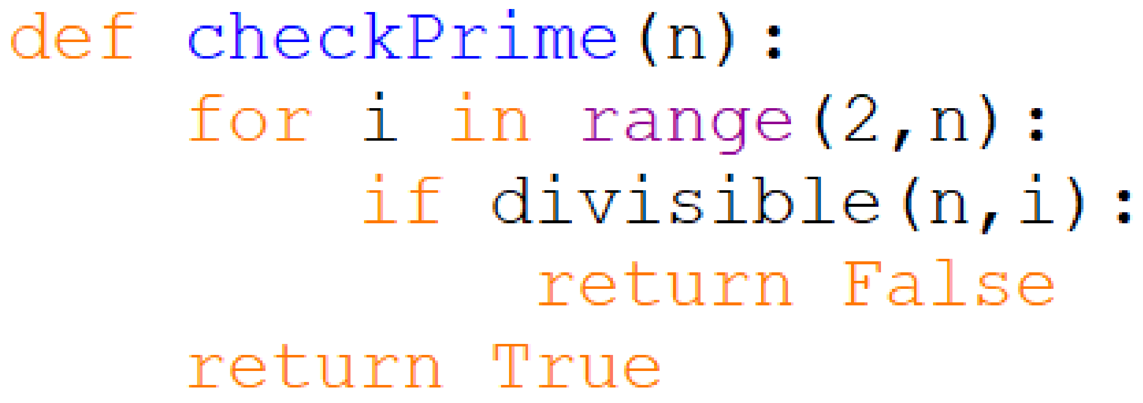
- For all the numbers i from 2 to n-1
  - We check if n is divisible by i
    - If divisible
      - Not prime! (Can we stop now?)
    - If not divisible
      - Continue to the next number i
- Finally, if all numbers i are not divisible, then n is prime!



```
def checkPrime(n):
    for i in range(2,n):
        if divisible(n,i):
            return False
    return True
```

Improvement?

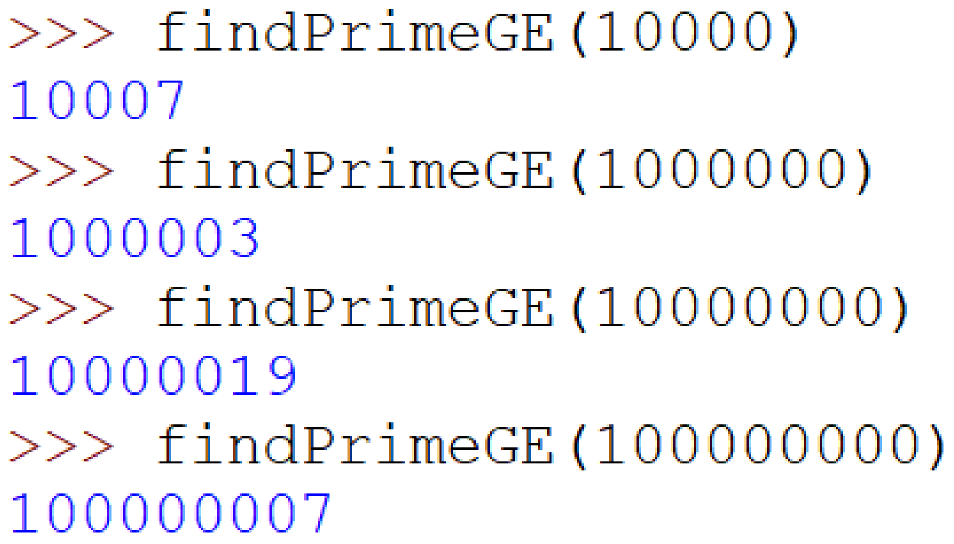
- The range of 2 to n?
  - Is it necessary?
  - We only need to check from 2 to `int(sqrt(n))`
    - why?!
  - or `int(sqrt(n))+1`?
- Finally, if all numbers i are not divisible, then n is prime!



```
def checkPrime(n):
    for i in range(2,n):
        if divisible(n,i):
            return False
    return True
```

Exercise (5 min)

- Given a number n, find a prime number that is greater than or equal to n



```
>>> findPrimeGE(10000)
10007
>>> findPrimeGE(1000000)
1000003
>>> findPrimeGE(10000000)
10000019
>>> findPrimeGE(100000000)
100000007
```

- Home exercise: find a Fibonacci number >= n

Short Circuit Logic

What will be the output?

```
def f1():
    return True

def foo():
    a = 3
    if a > 1 or f1():
        return "yes"
    return "no"

print(foo())
```

```
def f1():
    print("haha")
    return True

def foo():
    a = 3
    if a > 1 or f1():
        return "yes"
    return "no"

print(foo())
```

What will be the output?

```python
def f1():
    return True

def foo():
    a = 3
    if a > 1 or f1():
        return "yes"
    return "no"

print(foo())
```

“yes”

```python
def f1():
    print("haha")
    return True

def foo():
    a = 3
    if a > 1 or f1():
        return "yes"
    return "no"

print(foo())
```

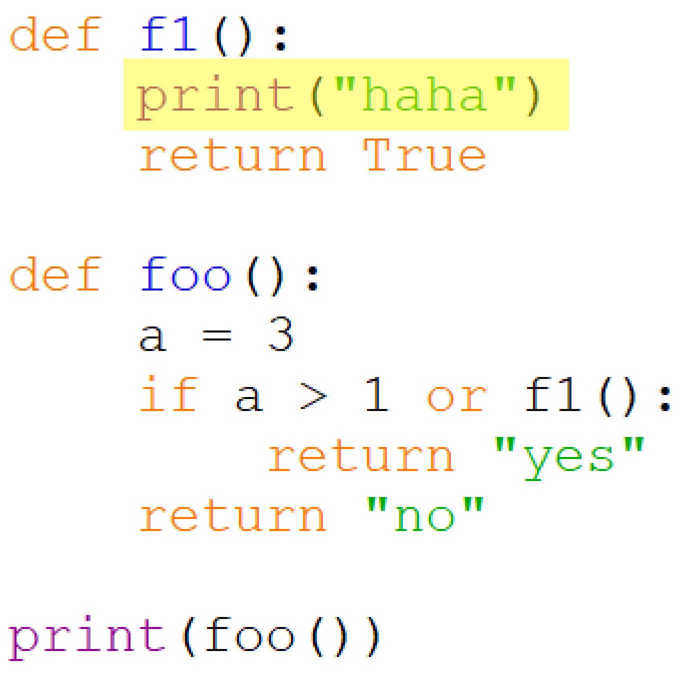
“yes”

What will be the output?

- Isn’t it supposed to be… ?

  “haha”

  “yes”
- The function f1() is “skipped”
  - Why?
- Conclusion:
  - If the left side of the “or” can decide the output already, the right side of the “or” will be “skipped”
  - Short Circuit Logic!

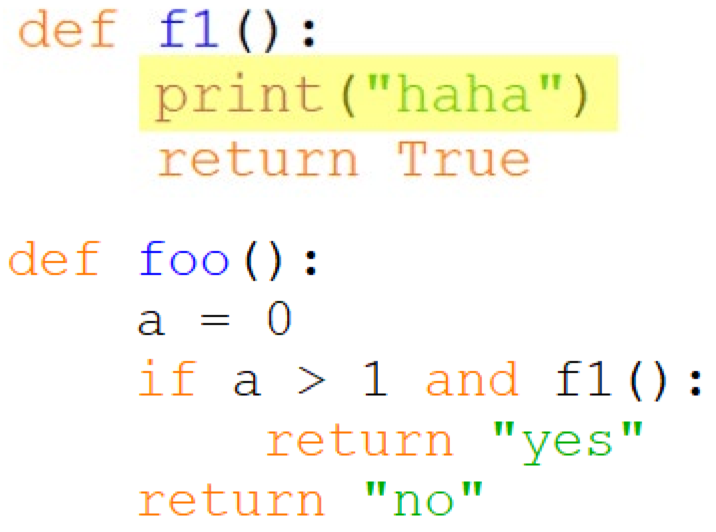


```
def f1():
    print("haha")
    return True

def foo():
    a = 3
    if a > 1 or f1():
        return "yes"
    return "no"

print(foo())
```

“yes”

How about this?

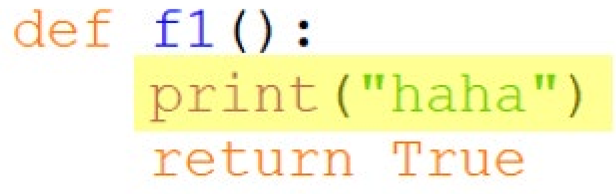
- Will f1() be called?



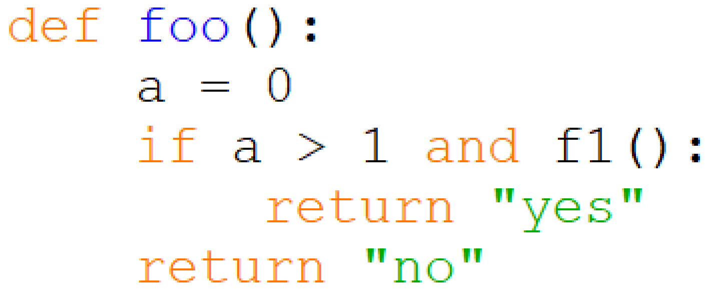
```
def f1():
    print("haha")
    return True

def foo():
    a = 0
    if a > 1 and f1():
        return "yes"
    return "no"
```

How about this?

- Will f1() be called?
    - (The final output is only a “no”)
- Why?
- Same rule!
- Conclusion:
    - If the left side of the “and” can decide the output already, the right side of the “and” will be “skipped”
    - Short Circuit Logic!



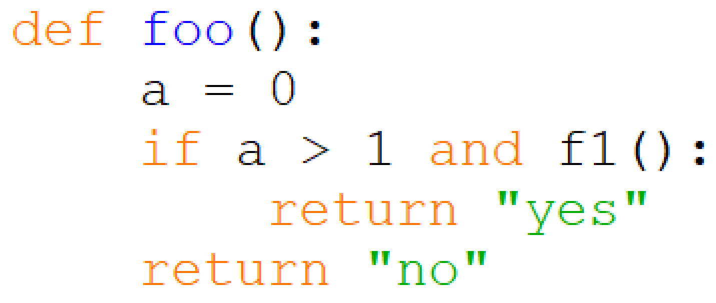
```
def f1():
    print("haha")
    return True
```



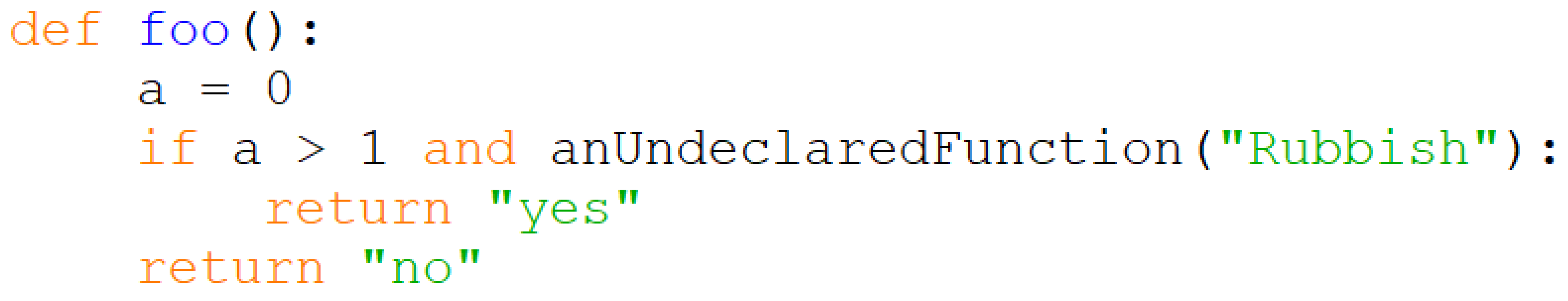
```
def foo():
    a = 0
    if a > 1 and f1():
        return "yes"
    return "no"
```

Short Circuit Evaluation

- Conclusion:
  - If the left side of the “logical operator” can decide the output already, the right side of the “logical operator” will be “skipped”



```
def foo():
    a = 0
    if a > 1 and f1():
        return "yes"
    return "no"
```



```
def foo():
    a = 0
    if a > 1 and anUndeclaredFunction("Rubbish"):
        return "yes"
    return "no"
```

Short Circuit Evaluation

- Conclusion:
  - If the left side of the "logical operator" can decide the output already, the right side of the "logical operator" will be "skipped"

```
def foo():
    a = 0
    if a > 1 and f1():
        return "yes"
    return "no"
```

- And it can even **dodge** errors!
  - (The "anUndeclaredFunction()" is not declared in the file.)

```
def foo():
    a = 0
    if a > 1 and anUndeclaredFunction("Rubbish"):
        return "yes"
    return "no"
```

Function

Scope and Recursion

Scope

Quick Scope Exercise

Code

```
x = 0

def foo_printx():
    print(x)


foo_printx()
print(x)
```

Output

```
0
0
```

Quick Scope Exercise

Code

```
x = 0

def foo_printx():
    print(x)


foo_printx()
print(x)
```

Output

Quick Scope Exercise

Code

```
x = 0
y = 999
def foo_printx(y):
    print(y)


foo_printx(x)
print(x)
```

Output

Quick Scope Exercise

Code **Output**

```
x = 0
y = 999
def foo_printx(y):
    print(y)


foo_printx(x)                  0
print(x)                       0
```

Quick Scope Exercise

Code

```
x = 0

def foo_printx():
    x = 999
    print(x)

foo_printx()
print(x)
```

Output

Quick Scope Exercise

Code

```
x = 0

def foo_printx():
    x = 999
    print(x)

foo_printx()
print(x)
```

Output

```
999
0
```

Quick Scope Exercise

Why?

Global Variables

Code

```
x = 0

def foo_printx():
    print(x)


foo_printx()
print(x)
```

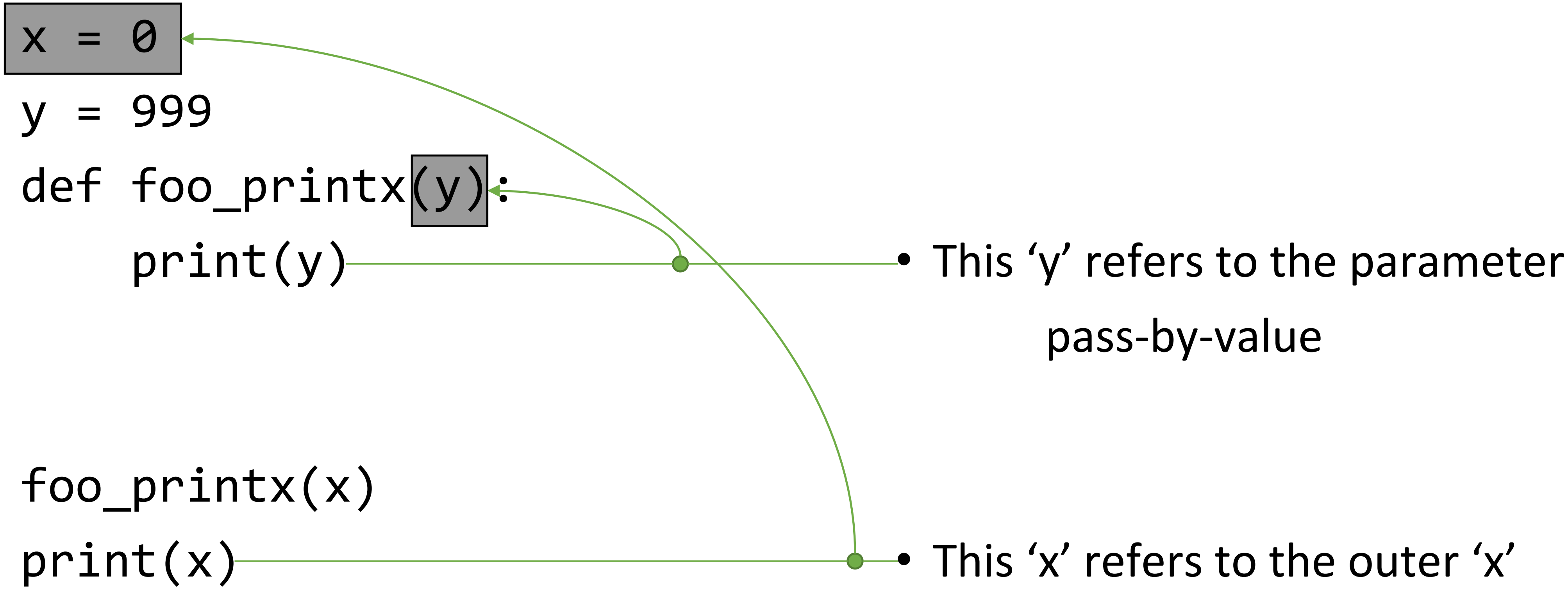
Explanation

- This ‘x’ refers to the outer ‘x’
- This ‘x’ also refers to the outer ‘x’

Global vs Local Variables

Code

Explanation



```
x = 0
y = 999
def foo_printx(y):
    print(y)


foo_printx(x)
print(x)
```

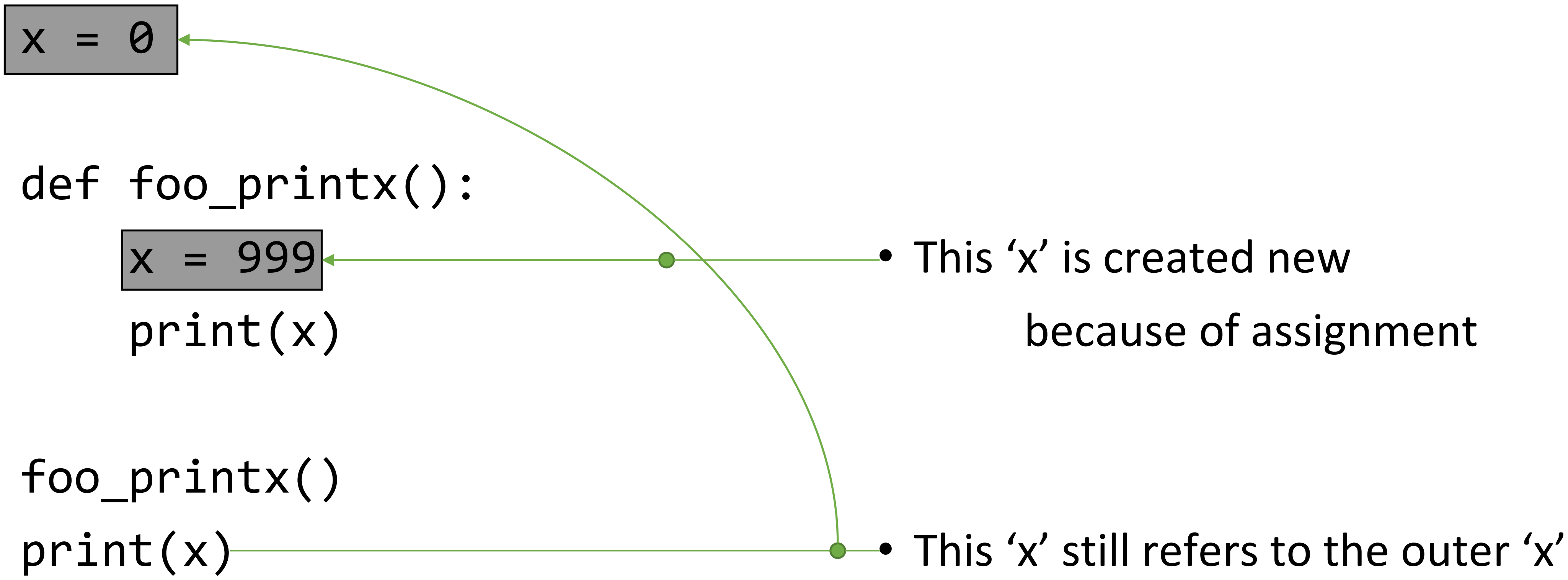
- This ‘y’ refers to the parameter

  pass-by-value

- This ‘x’ refers to the outer ‘x’

Global vs Local Variables

Code **Explanation**



```
x = 0

def foo_printx():
    x = 999
    print(x)

foo_printx()
print(x)
```

- This ‘x’ is created new because of assignment
- This ‘x’ still refers to the outer ‘x’

Global vs Local Variables

Code

```
x = 0

def foo_printx():
    x = 999
    print()

foo_printx()
print(x)
```

Explanation

- Global scope
- Local scope
  - Local ‘x’ is ***born*** here
  - *Will **be deleted*** when the function ends here
- The two ‘x’ will be different ‘x’
  - ‘999’ will only be available within function

Rule of Thumb

Code

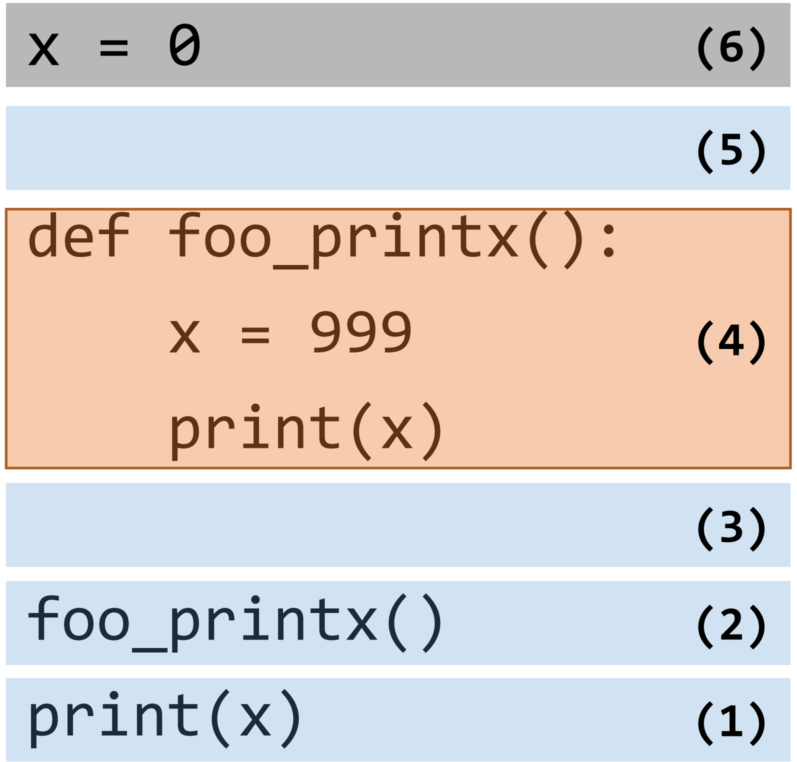
```
x = 0

def foo_printx():
    x = 999          (2)
    print(x)         (1)

foo_printx()
print(x)
```

Go up and go out cannot go in

- Simple case: x within function
    1. Start here
    2. Go up
        - Found!

Rule of Thumb

Code



```
x = 0                      (6)
                           (5)
def foo_printx():
    x = 999                (4)
    print(x)
                           (3)
foo_printx()               (2)
print(x)                   (1)
```

Go up and go out cannot go in

- Harder case: x outside function
  1. Start here
  2. Go up
  3. Go up
  4. Cannot go in
  5. Go up
  6. Go up
     - Found!

Global vs Local Variables

- A variable which is defined in the main body of a file is called a ***global*** variable. It will be **visible throughout the file**, and also inside any file which imports that file. EXCEPT…
- A variable which is defined inside a function is ***local*** to that function. It is accessible **from the point at which it is defined until the end of the function**, and exists for as long as the function is executing.
- The parameter names in the function definition behave like local variables, but they contain the values that we pass into the function when we call it.

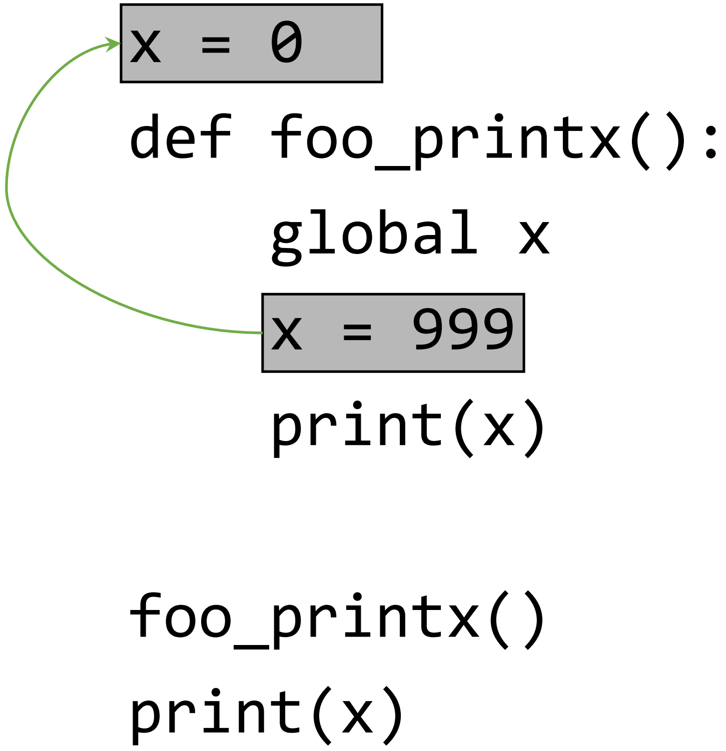
Crossing Boundaries

Problem

- What if we want to modify variables from outside within the function?
  - Use “global” keyword
  - No local variables ‘x’ is created

```
Output:
```

Code



```
x = 0
def foo_printx():
    global x
    x = 999
    print(x)


foo_printx()
print(x)
```

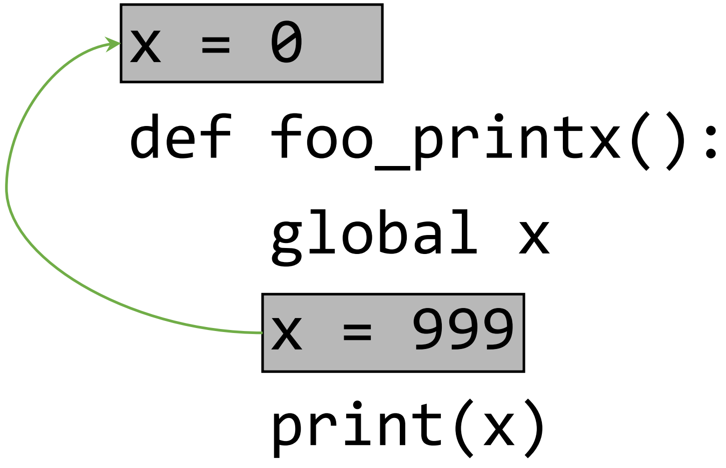
Crossing Boundaries

Problem

- What if we want to modify variables from outside within the function?
  - Use “global” keyword
  - No local variables ‘x’ is created

```
Output:
999
999
```

Code



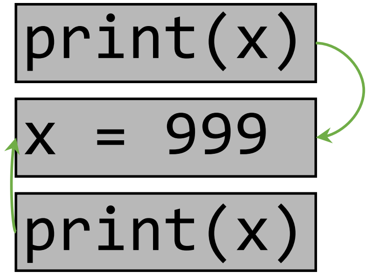
```
x = 0
def foo_printx():
    global x
    x = 999
    print(x)


foo_printx()
print(x)
```

To Cross or Not to Cross

Problem

- Consider the following code
- What is happening?
    - Second print(x) refers to x = 999
    - What about the first print(x)?

Code



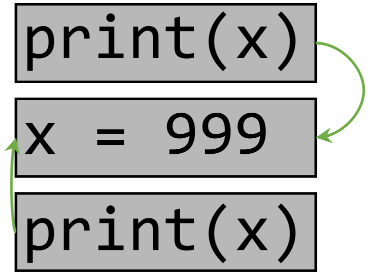
```
x = 0
def foo_printx():
    print(x)
    x = 999
    print(x)


foo_printx()
print(x)
```

To Cross or Not to Cross

Problem

- Consider the following code
- What is happening?
  - Second print(x) refers to x = 999
  - What about the first print(x)?
    - Also to x = 999
    - But it comes after!
    - This is an error
    - It has no value

Code



```
x = 0
def foo_printx():
    print(x)
    x = 999
    print(x)


foo_printx()
print(x)
```

Parameters are Local

Code

```
def foo(x):            (1)
    bar(x + 1)         (A)
    print(x)           (3)
def bar(x):            (A)
(C) x = x + x          (B)
    print(x)           (D)


print(foo(3))          (1)
```

Explanation

1. Pass 3 to x in foo
2. Evaluate x + 1
   A. Pass 4 to x in bar
   B. Evaluate x + x
   C. Assign to x
   D. print(x) in bar
      - print 8
3. print(x) in foo
   - print 3

Parameters are Local

Code

```
def foo(x):
    bar(x + 1)
    print(x)
def bar(x):
    x = x + x
    print(x)

print(foo(3))
```

Explanation

- The ‘x’ in bar is different from the ‘x’ in foo

Recursion

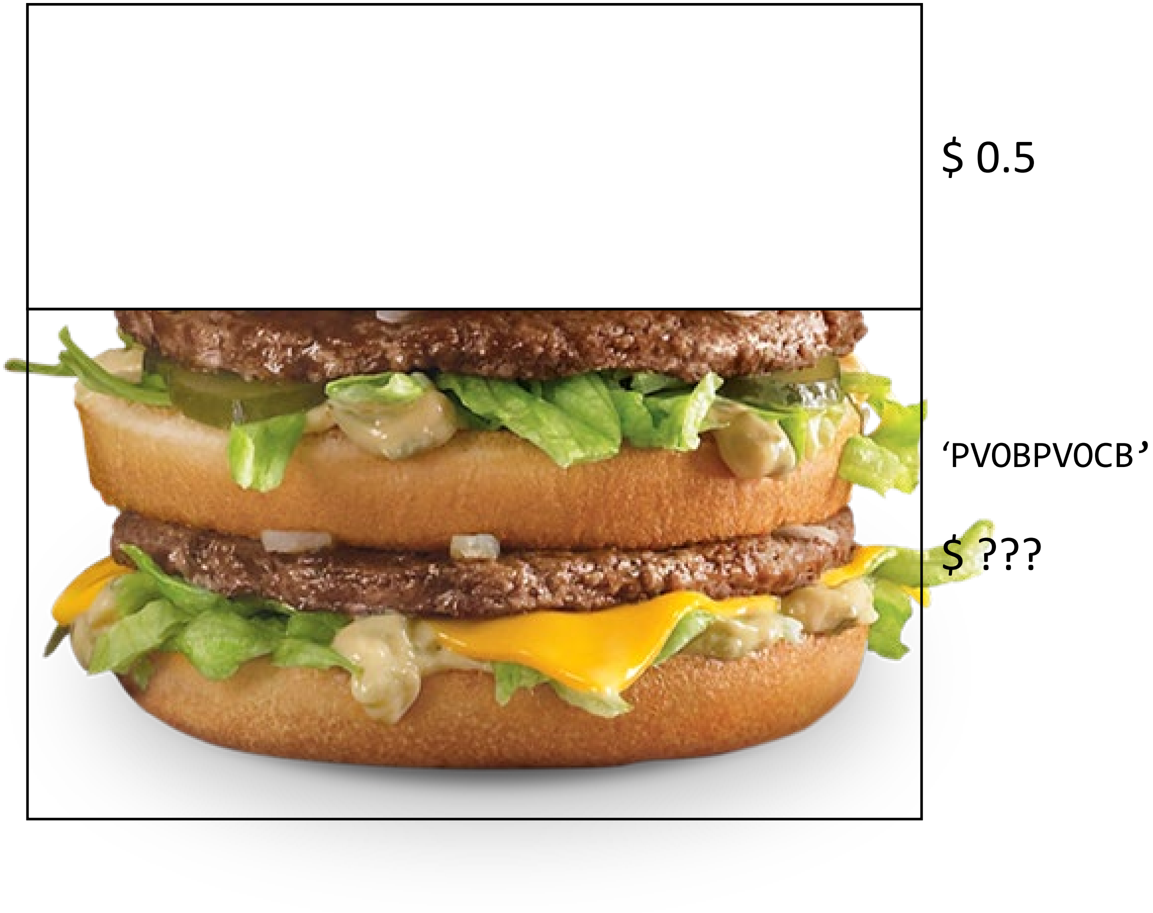
Converting a function from iteration to recursion

Recap: burgerPrice(burger)

```
def burgerPrice(burger):
    price = 0
    for char in burger:
        if char == ‘B’:
            price += 0.5
        : # code omitted
    return price
```

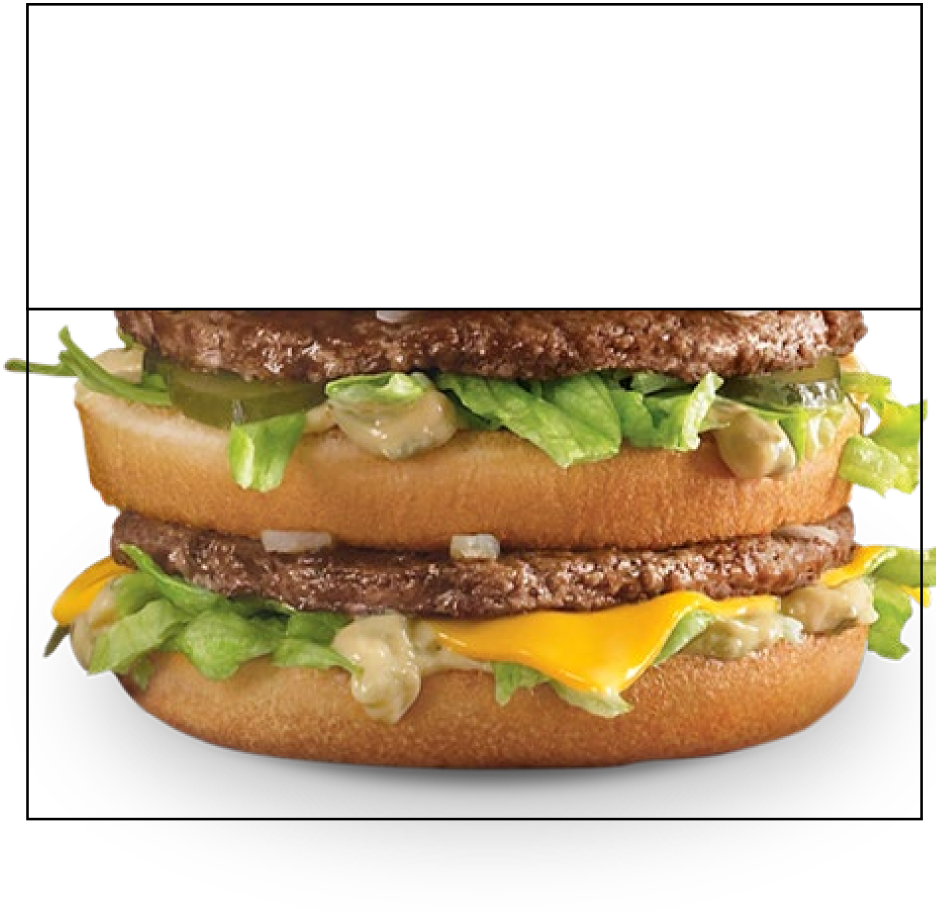
burgerPrice(burger) in recursion?

- Idea
  - `bigMac = ‘BPVOBPVOCB’`
  - What’s the price
    - `burgerPrice(bigMac) = 6.7`
  - How?
    - B is bun, costs 0.5
    - The rest is ‘PVOBPVOCB’
      - How to get?
        - String slicing: `theRest = bigMac[1:]`
      - How much?
        - Recursion: `burgerPrice(theRest)`
    - Total: `0.5 + burgerPrice(theRest)`



Try it yourself? (10 min)

- Idea
    - `bigMac = ‘BPVOBPVOCB’`
    - What’s the price
        - `burgerPrice(bigMac) = 6.7`
    - How?
        - B is bun, costs 0.5
        - The rest is ‘PVOBPVOCB’
            - How to get?
                - String slicing: `theRest = bigMac[1:]`
            - How much?
                - Recursion: `burgerPrice(theRest)`
        - Total: `0.5 + burgerPrice(theRest)`



$ 0.5

‘PVOBPVOCB’

$???

- burgerPrice(‘BPVOBPVOCB’)
- 0.5 + 1.0 + burgerPrice(‘VOBPVOCB’)
- 0.5 + 1.0 + 0.3 + burgerPrice(‘OBPVOCB’)
- 0.5 + 1.0 + 0.3 + 0.4 + burgerPrice(‘BPVOCB’)
- …
- 0.5 + …. + 0.4 + 0.5 + burgerPrice(‘’)
- 0.5 + …. + 0.4 + 0.5 + 0

burgerPrice(burger) in recursion?

Please do not show this code until students tried it for themselves