



Functions

Chris Piech and Mehran Sahami
CS106A, Stanford University

Boolean Variable

```
karel_is_awesome = True
```

```
my_bool = 1 < 2
```



Boolean Operations

`a = True`

`b = False`

`both_true = a and b`

`either_true = a or b`

`opposite = not a`



There is an *i* in for loop

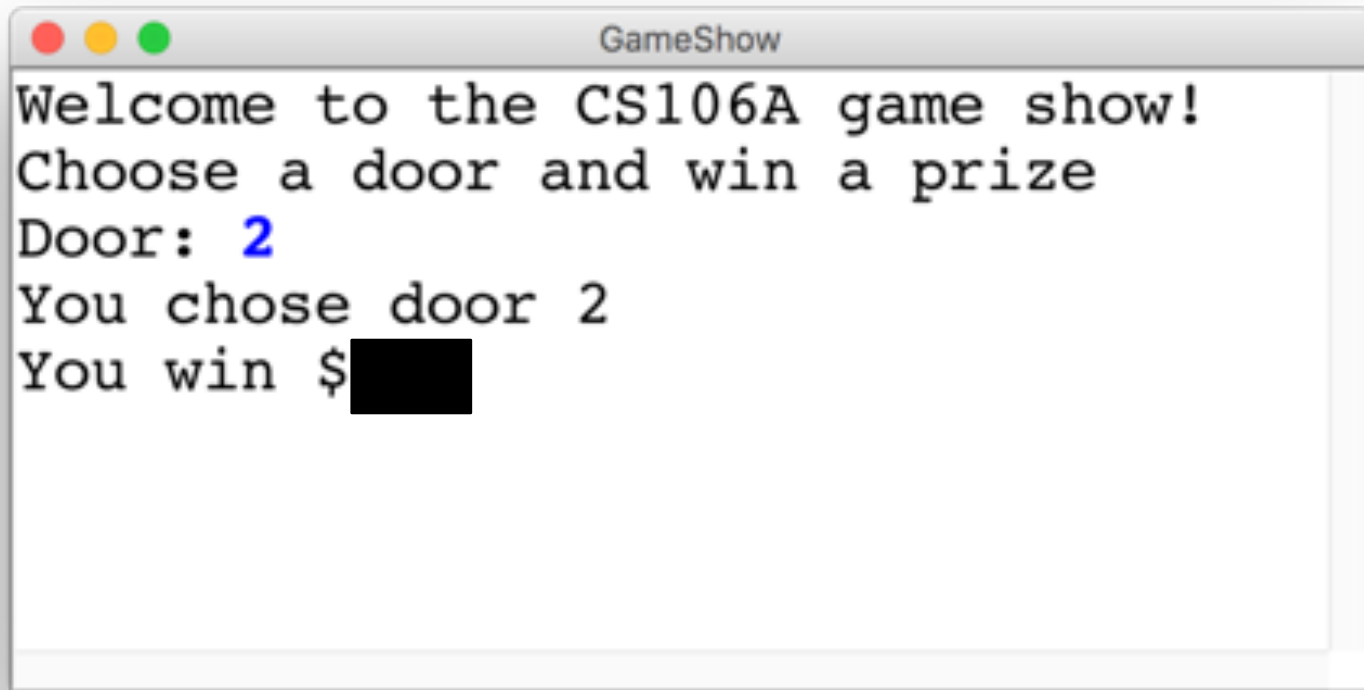
```
for i in range(10):  
    print(i)
```

terminal

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```



Game Show



Choose a Door

```
door = int(input("Door: "))  
# while the input is invalid  
while door < 1 or door > 3 :  
    # tell the user the input was invalid  
    print("Invalid door!")  
    # ask for a new input  
    door = int(input("Door: "))
```

or
and



The Door Logic

```
prize = 4
```

```
if door == 1:  
    prize = 2 + 9 // 10 * 100
```

```
elif door == 2:  
    locked = prize % 2 != 0  
    if not locked:  
        prize += 6
```

```
elif door == 3 :  
    for i in range(door):  
        prize += i
```



Learn How To:

1. Write a function that takes in input
2. Write a function that gives back output
3. Trace function calls using stacks



Calling functions

`turn_right()`

`move()`

`input("string please! ")`

`print("hello world")`

`float("0.42")`

`math.sqrt(25)`



Defining a function

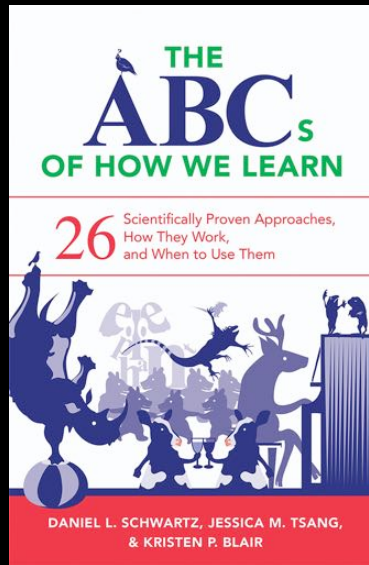
```
def turn_right():  
    turn_left()  
    turn_left()  
    turn_left()
```



Big difference with python functions:
Python functions can **take in data**, and can **return data**!



Contrasting Case



Thanks Dan Schwartz

Defining a function

```
def turn_right():  
    turn_left()  
    turn_left()  
    turn_left()
```

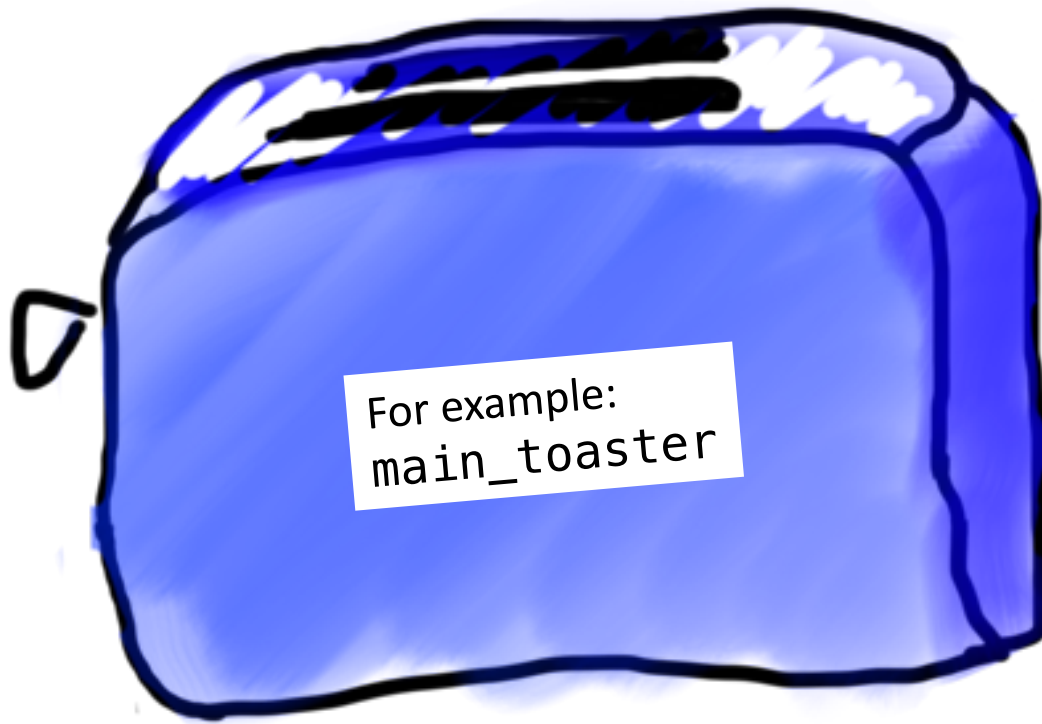
```
def move_n(n_moves):  
    for i in range(n_moves):  
        move()
```

```
def main():  
    turn_right()  
    move_n(10)
```

Big difference with python functions:
Python functions can **take in data**, and can **return data**!



Toasters are functions



Toasters are functions



parameter



Toasters are functions



parameter



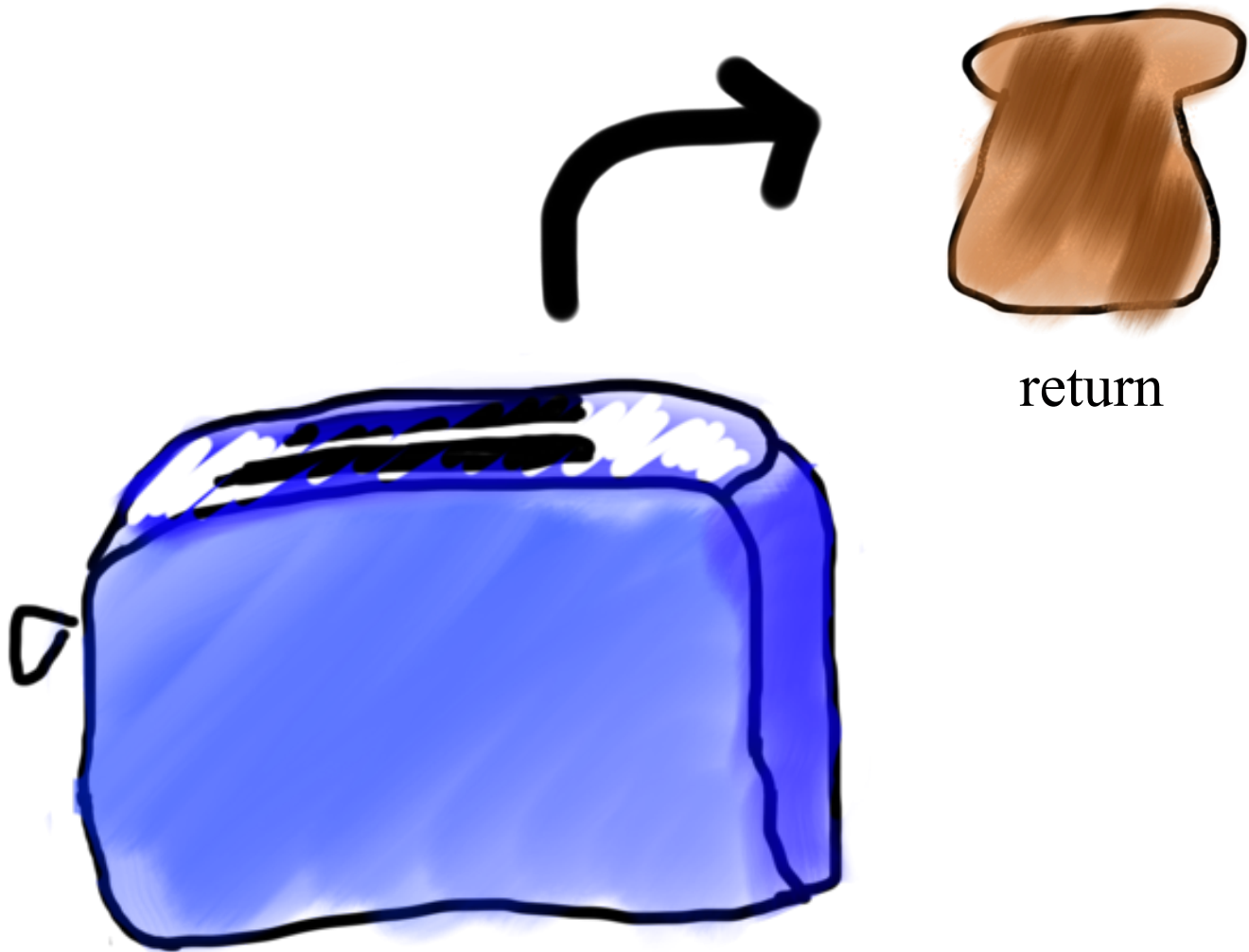
Toasters are functions



Toasters are functions



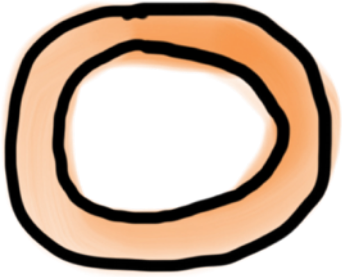
Toasters are functions



Toasters are functions



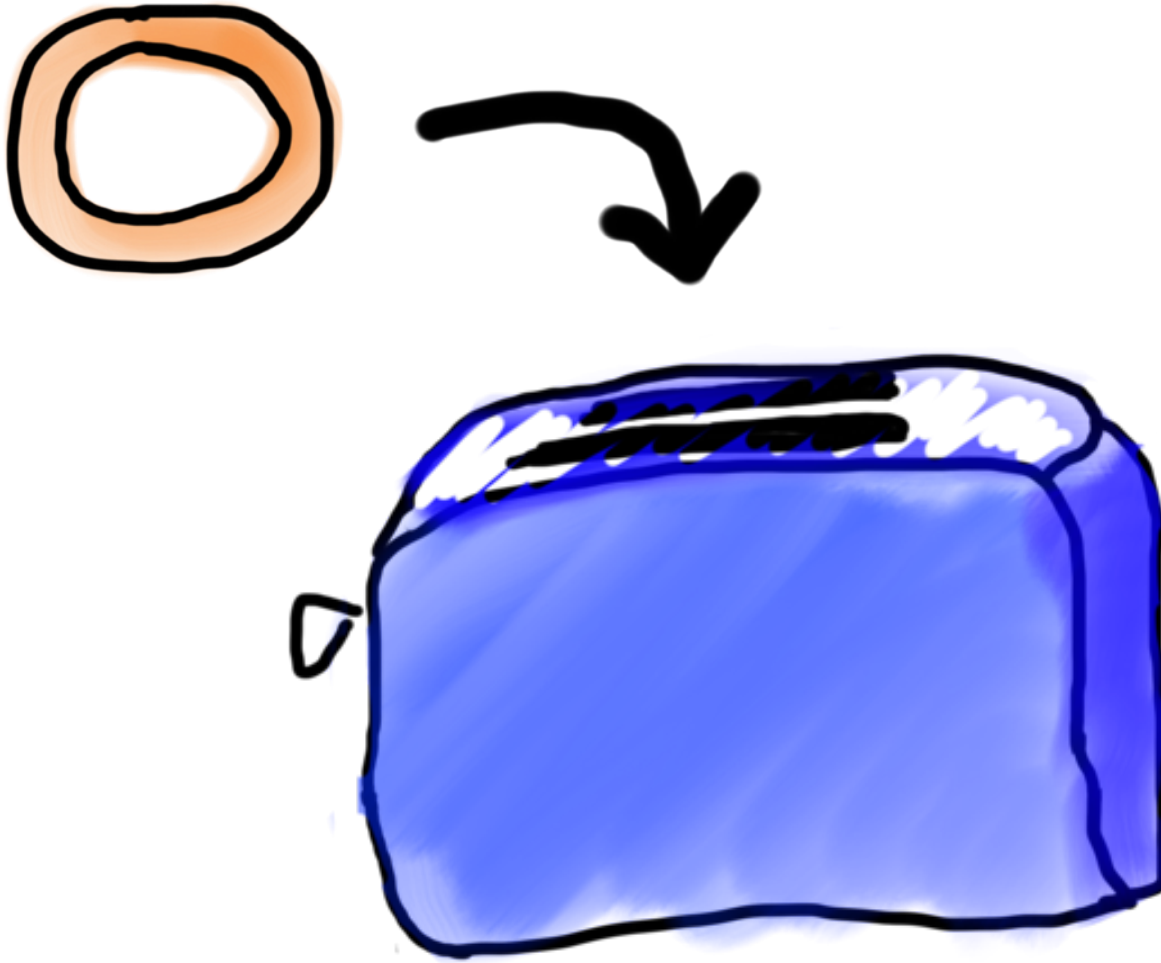
Toasters are functions



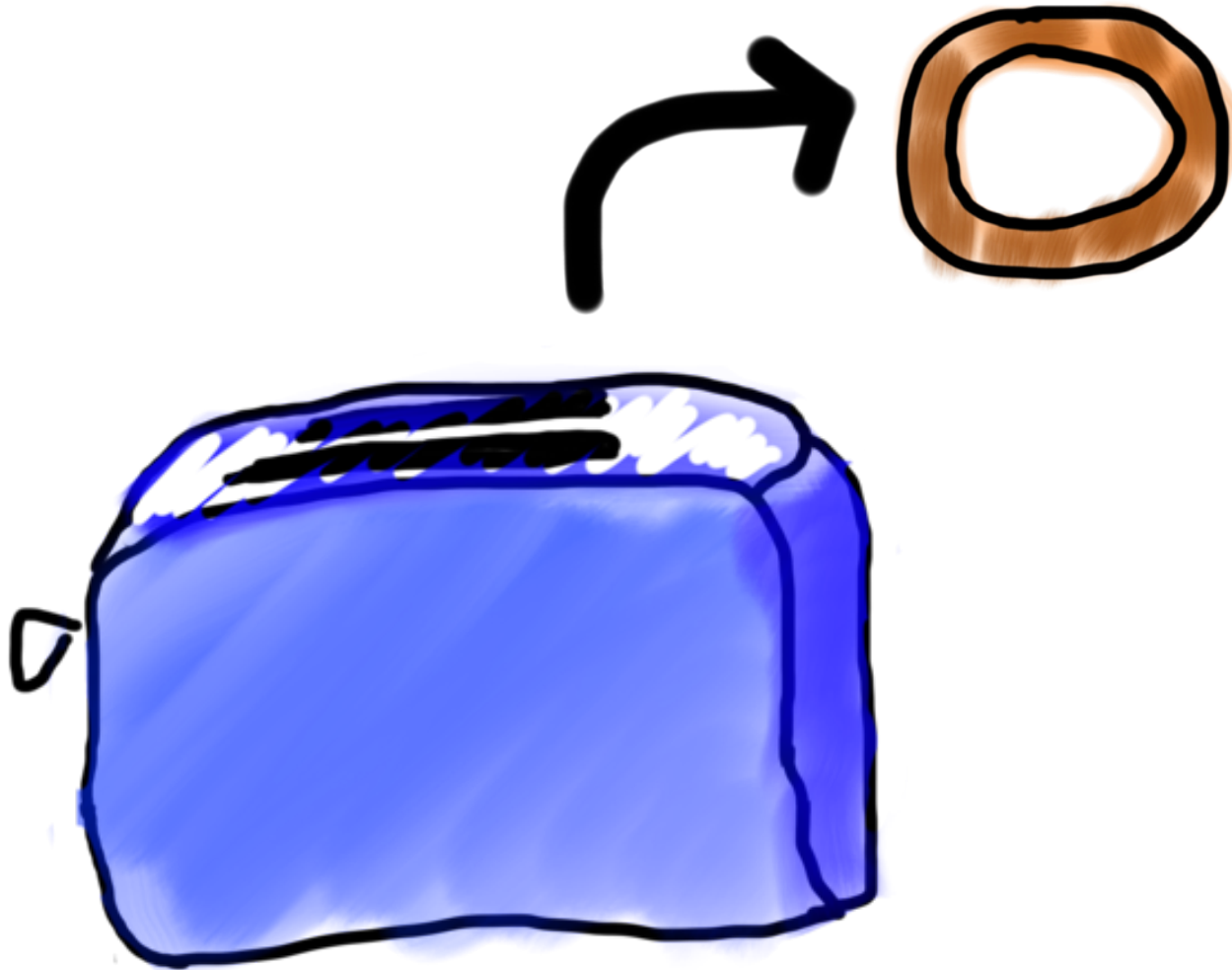
* You don't need a second toaster if you want to toast bagels. Use the same one.



Toasters are functions



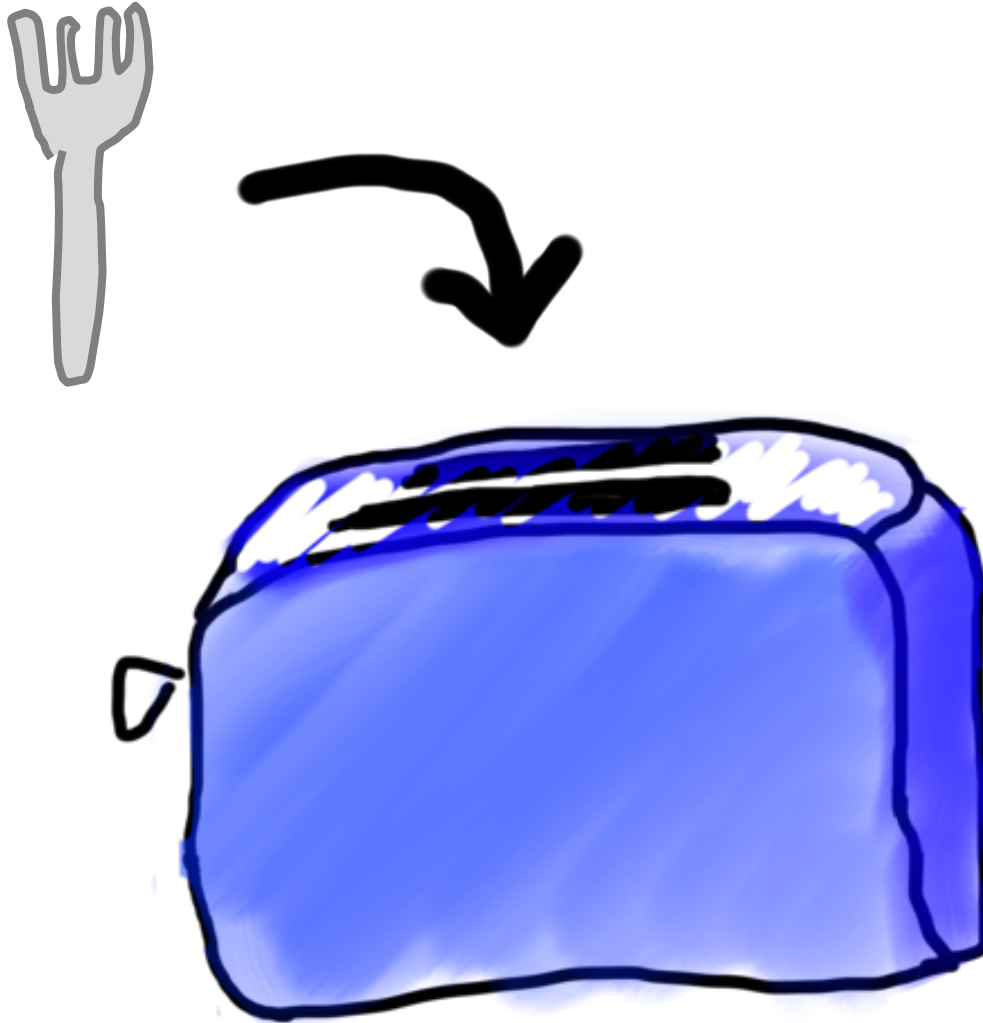
Toasters are functions



Toasters are functions



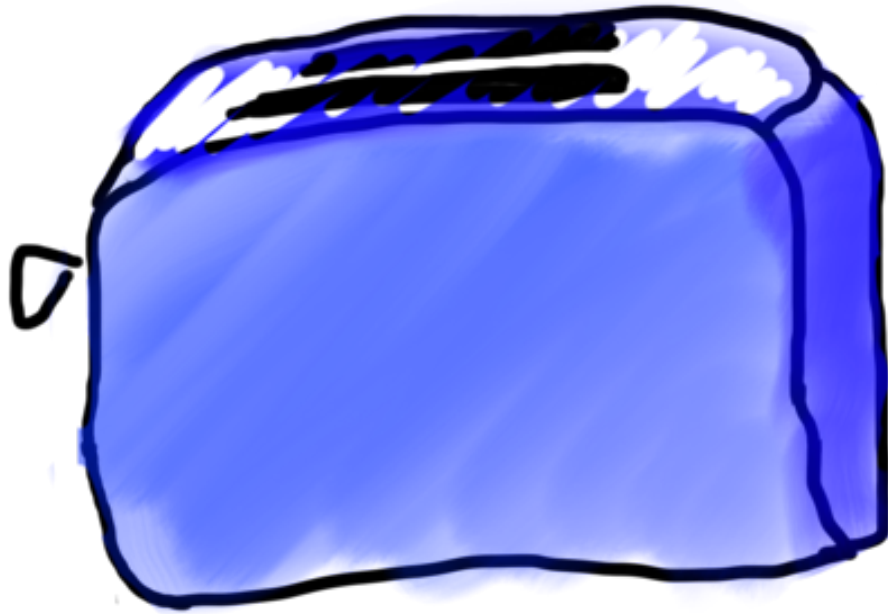
Toasters are functions



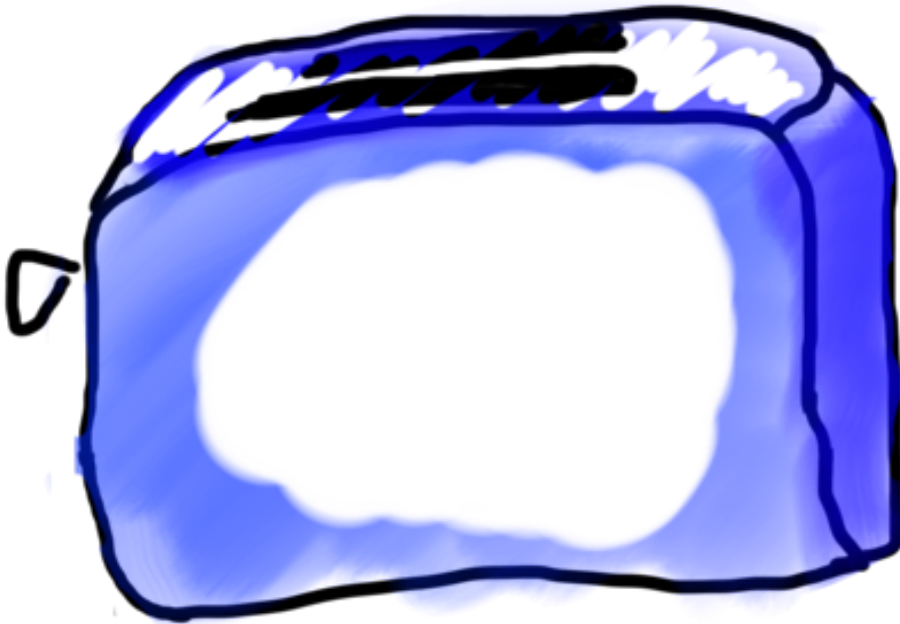
Toasters are functions



functions are Like Toasters



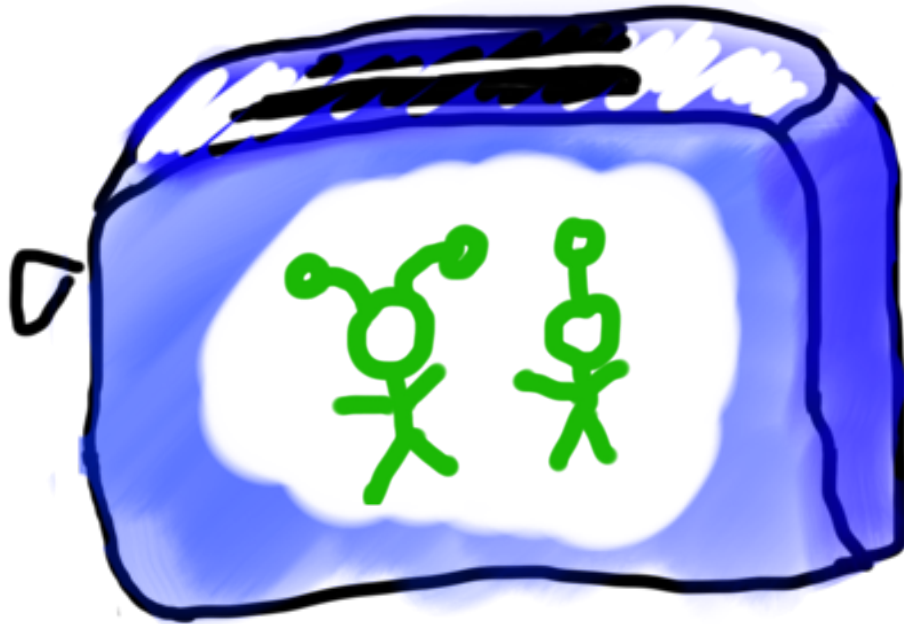
functions are Like Toasters



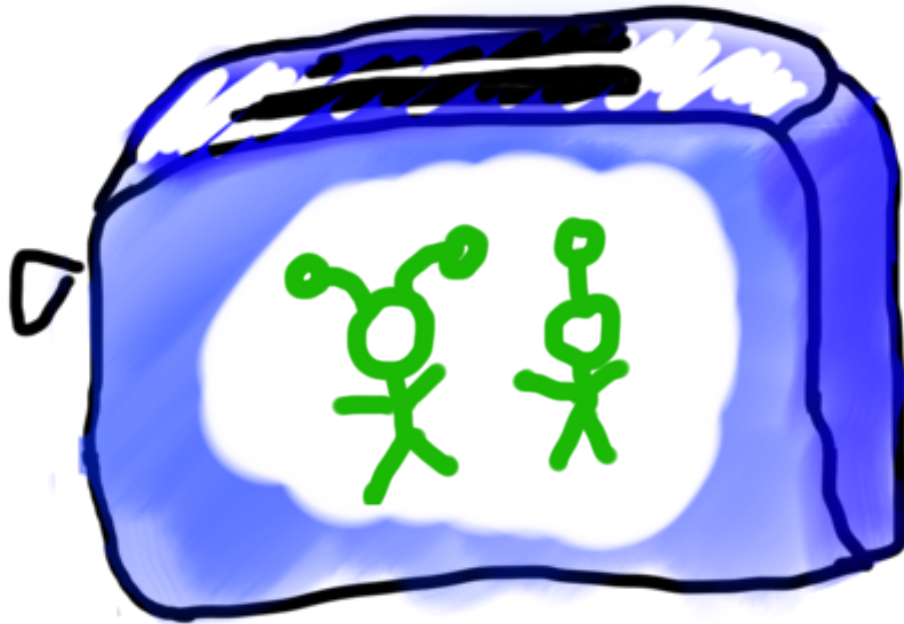
functions are Like Toasters



functions are Like Toasters



functions are Like Toasters



functions are Like Toasters



functions are Like Toasters



parameter(s)



return



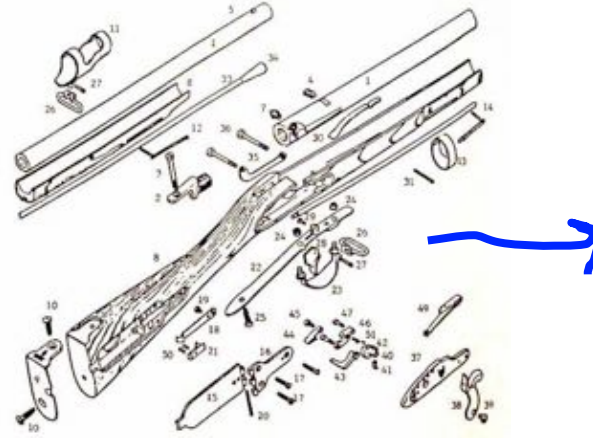
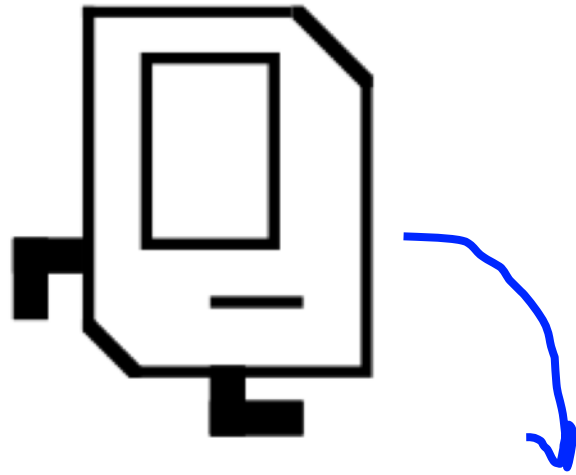
Classic Challenge for CS106A



Perhaps the
most
underrated
concept by
students



Historical Aside



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Anatomy of a function

```
def main():    function “call”  
    mid = average(5.0, 10.2)  
    print(mid)
```

function “definition”

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def name average(a, b):  
    sum = a + b  
    return sum / 2
```



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

Input given

Input expected

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

Arguments

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

Parameters



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):
```

```
    sum = a + b  
    return sum / 2
```

body



Anatomy of a function

```
def main():    This call “evaluates” to the value returned  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

Ends the function and gives
back a value



Anatomy of a function

Also a function call

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Anatomy of a function

No parameters (expects no input)

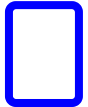
```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```



When a function ends it “returns”

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Formally

```
def name_of_function (parameters) :  
    statements  
    # optionally  
    return value
```

- *name*: information passed into function
- *parameters*: information passed into function
- *return*: information given back from the function



Parameters



Parameters let
you provide a
function some
information
when you are
calling it.



Is returning
the same as printing?

Is returning
the same as printing?

NO

Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

Function caller



```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

Function author



**Function author's
contract - average:**

If you call this
function you must
provide two **params**.
The function will give
back a **return** value

User



Uses the
terminal
(later UI)

Function
Caller (Coder)



Uses helper
functions

Function
Author (Coder)



Writes helper
functions
others can
use



User



Uses the
terminal
(later UI)

Function
Caller (Coder)



Uses helper
functions

Function
Author (Coder)



Writes helper
functions
others can
use



Learn by Example



No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



No Parameter, No Return



function author

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

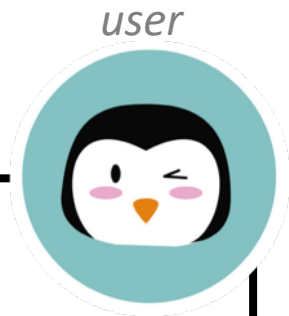
```
def main():  
    print_intro()
```



caller

terminal

```
> python intro.py
```



No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



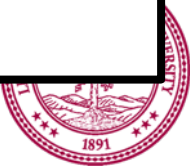
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



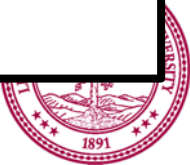
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



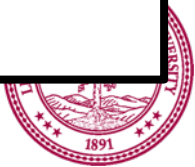
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



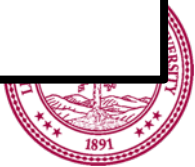
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class
```



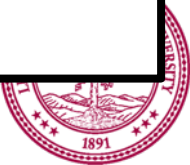
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```



No Parameter, No Return

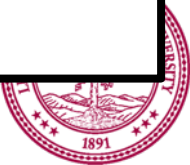
```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```



```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```



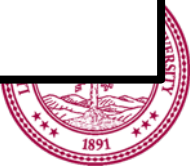
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```



No Parameter, No Return

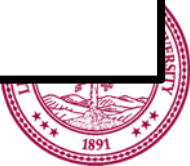
```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```



terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```



Parameter Example

```
def print_opinion(num) :  
    if (num == 5) :  
        print("I love 5!")  
    else :  
        print("Whatever")  
  
def main() :  
    print_opinion(5)
```

terminal

```
> python opinion.py
```



Parameter Example

main memory

No variables

```
def print_opinion(num):  
    if(num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

terminal

```
> python opinion.py
```



Parameter Example

main memory

No variables

```
def print_opinion(num):  
    if(num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

terminal

```
> python opinion.py
```



Parameter Example

main memory

No variables

print_opinion memory



terminal

> python opinion.py

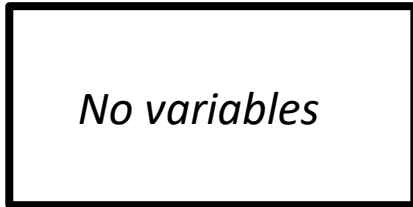
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

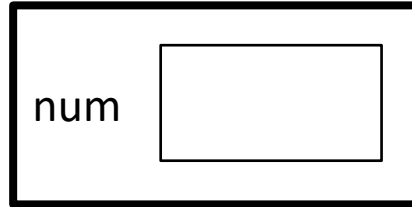


Parameter Example

main memory



print_opinion memory



terminal

```
> python opinion.py
```

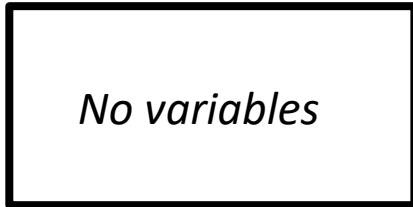
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

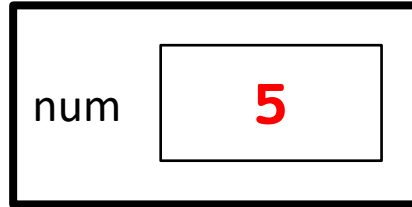


Parameter Example

main memory



print_opinion memory



terminal

```
> python opinion.py
```

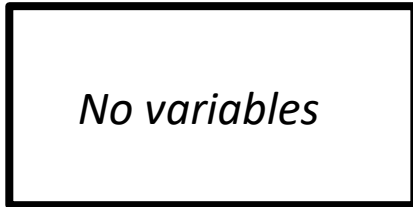
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

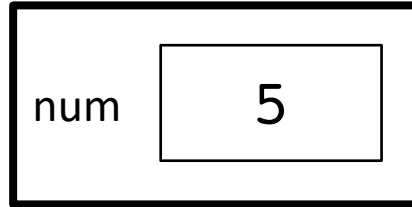


Parameter Example

main memory



print_opinion memory



terminal

```
> python opinion.py
```

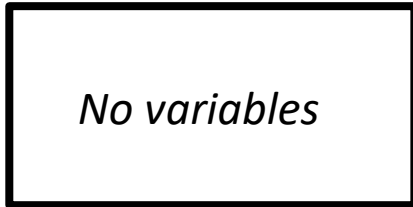
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

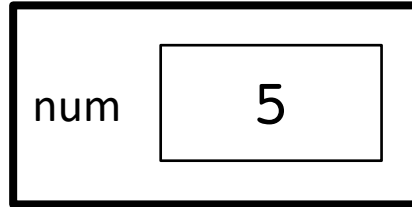


Parameter Example

main memory



print_opinion memory



terminal

```
> python opinion.py  
I love 5!
```

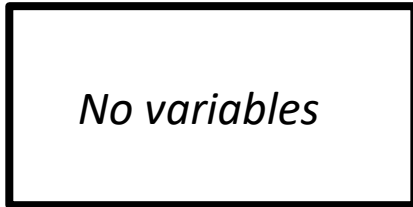
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

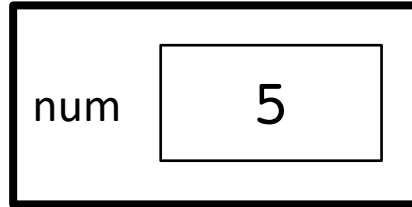


Parameter Example

main memory



print_opinion memory



terminal

```
> python opinion.py  
I love 5!
```

```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```



```
def main():  
    print_opinion(5)
```



Parameter Example

main memory

No variables

```
def print_opinion(num):  
    if(num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```



```
def main():  
    print_opinion(5)
```

terminal

```
> python opinion.py  
I love 5!
```



Parameter Example

main memory

No variables

```
def print_opinion(num):  
    if(num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

terminal

```
> python opinion.py  
I love 5!
```



Parameter Example

main memory

No variables

```
def print_opinion(num):  
    if(num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```



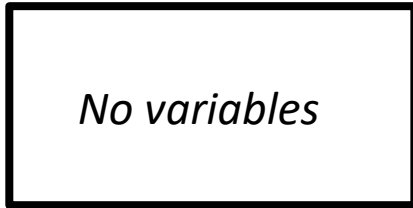
terminal

```
> python opinion.py  
I love 5!
```



Parameter Example

main memory



```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```



terminal

```
> python opinion.py  
I love 5!
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters  
  
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory

No variables

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory

No variables

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

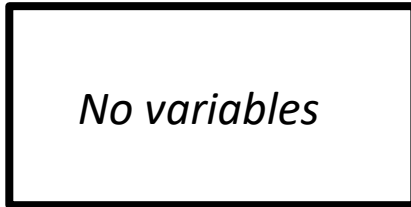
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory



metersToCm memory



```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

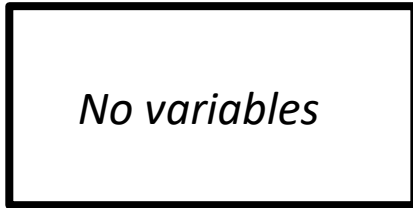
terminal

```
> python m2cm.py
```

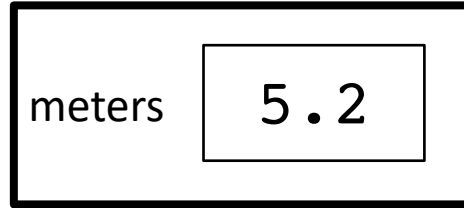


Parameter and Return Example

main memory



metersToCm memory



terminal

```
> python m2cm.py
```

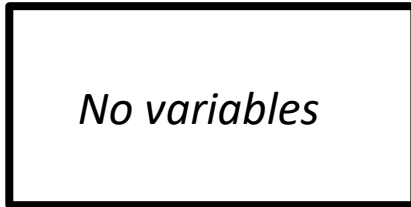
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```



Parameter and Return Example

main memory



metresToCm memory



terminal

```
> python m2cm.py
```

```
def meters_to_cm(meters):  
    return 100 * meters
```

520.0

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```



Parameter and Return Example

main memory

No variables

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

520.0

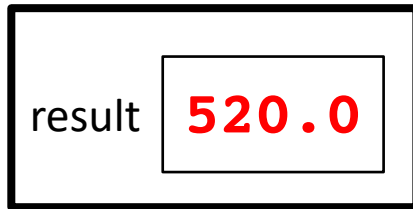
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory



```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

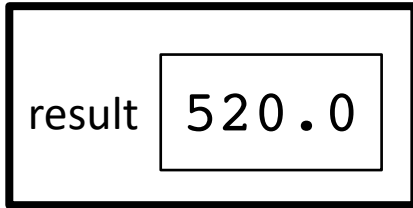
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory



```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```



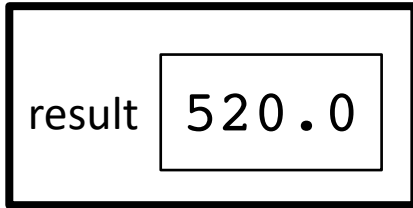
terminal

```
> python m2cm.py  
520.0
```



Parameter and Return Example

main memory



terminal

```
> python m2cm.py  
520.0
```

```
def meters_to_cm(meters):  
    return 100 * meters
```

This is our favorite paradigm (fn author returns, fn caller print).
It's the most flexible, modular. Its elegance is unparalleled.

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```



Parameter and Return Example

terminal

```
> python m2cm.py
```

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

terminal

```
> python m2cm.py
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

terminal

```
> python m2cm.py
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

520.0

terminal

```
> python m2cm.py
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

520.0

terminal

```
> python m2cm.py  
520.0
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

terminal

```
> python m2cm.py  
520.0
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

910.0

terminal

```
> python m2cm.py  
520.0
```



Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

910.0

terminal

```
> python m2cm.py  
520.0  
910.0
```



Parameter and Return Example



```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```



terminal

```
> python m2cm.py  
520.0  
910.0
```



How is this function

```
def meters_to_cm_case1(meters):  
    return 100 * meters
```

Different than this function?

```
def meters_to_cm_case2(meters):  
    print(100 * meters)
```



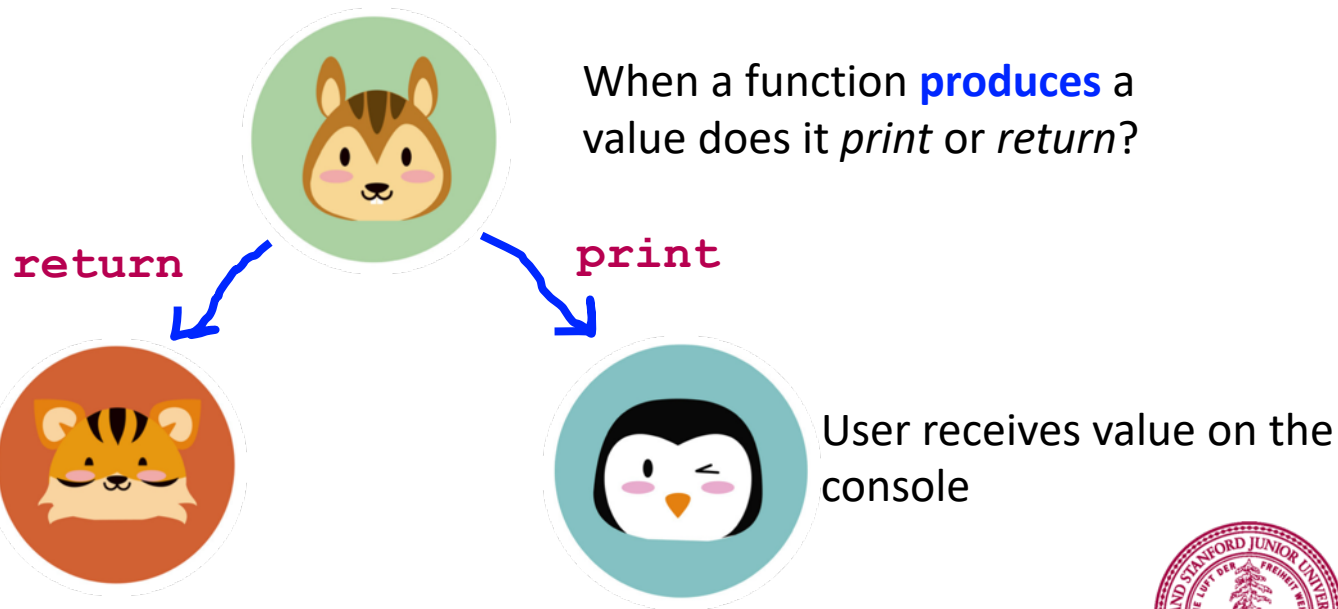
How is this function

```
def meters_to_cm_case1(meters):  
    return 100 * meters
```



Different than this function?

```
def meters_to_cm_case2(meters):  
    print(100 * meters)
```



Is returning
the same as printing?

Is returning
the same as printing?

NO

How is this function

```
def meters_to_cm_case1():  
    meters = float(input("M: "))  
    return 100 * meters
```



Different than this function?

```
def meters_to_cm_case2(meters):  
    return 100 * meters
```

When a function
requires a value does it
call *input* itself?



parameter

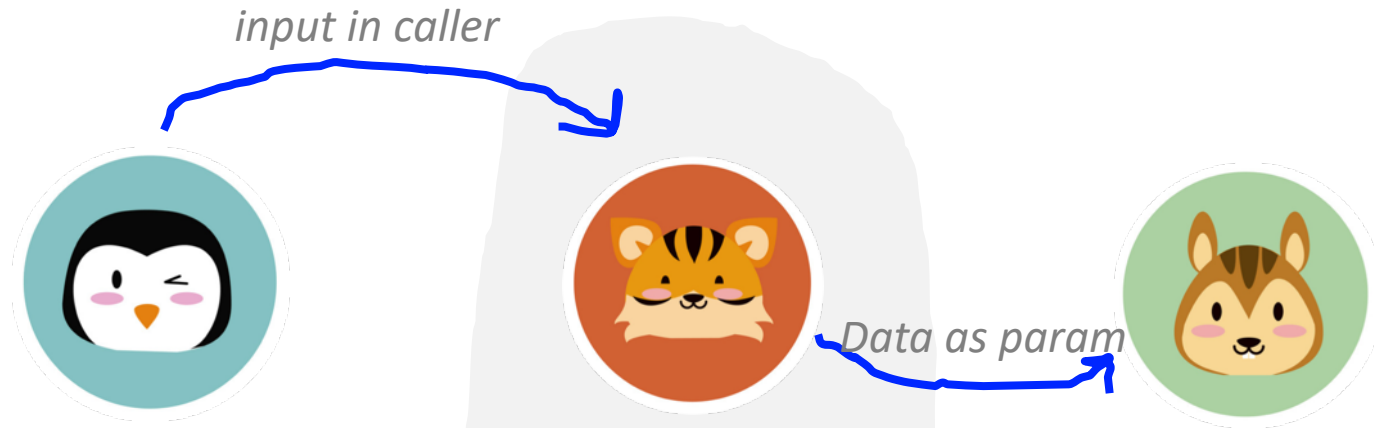
Fn author receives value.
Can store it and/or print it



input

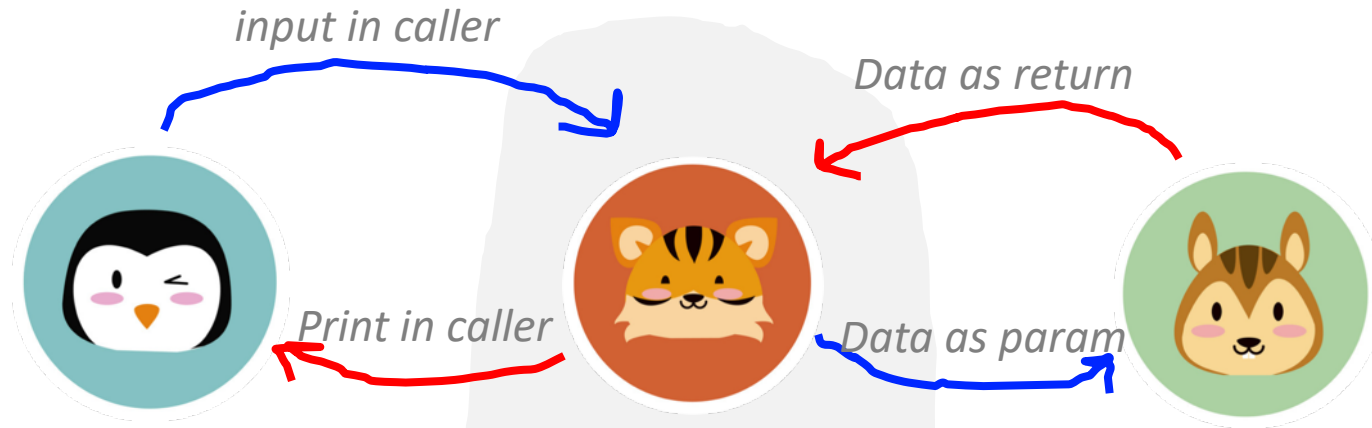
User inputs value within
the function. Seems nice,
but hard to build upon!

Ideal flow of information: Input



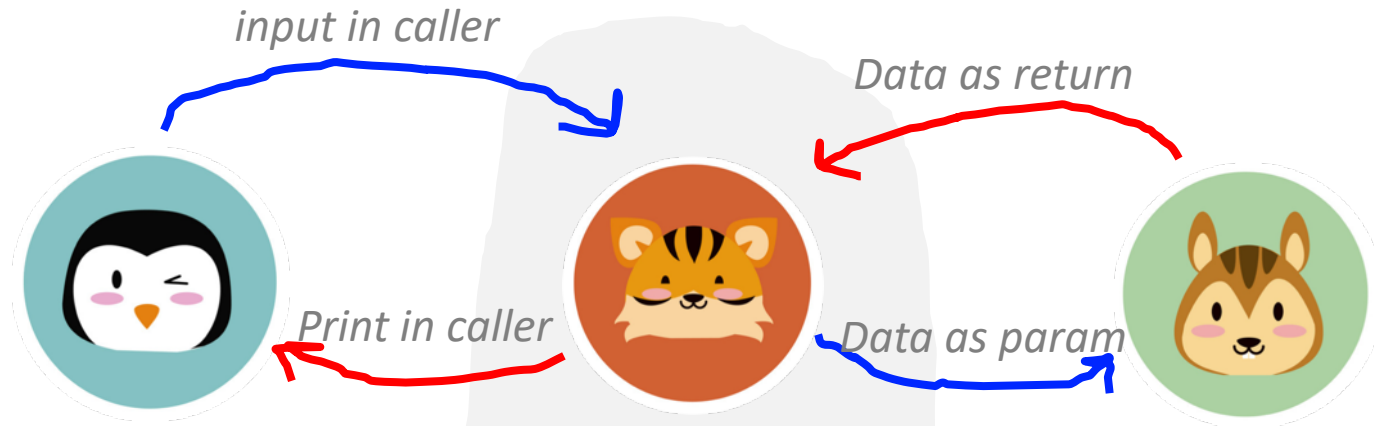
```
def example_caller():  
    data = float(input("enter:"))  
    call_the_author(data)
```

Ideal flow of information: Input



```
def example_caller():  
    data = float(input("enter:"))  
    result = call_the_author(data)  
    print(result)
```

Ideal flow of information: Input



```
def main():  
    meters = float(input("meters:"))  
    cms = meters_to_cm(meters)  
    print(cms + "cm")
```


As "input"
the same as parameters?

As "input"
the same as parameters?

NO

Multiple Return Statements

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2  
  
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

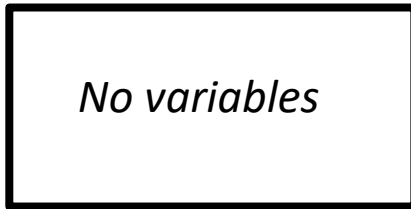
terminal

```
> python maxmax.py
```

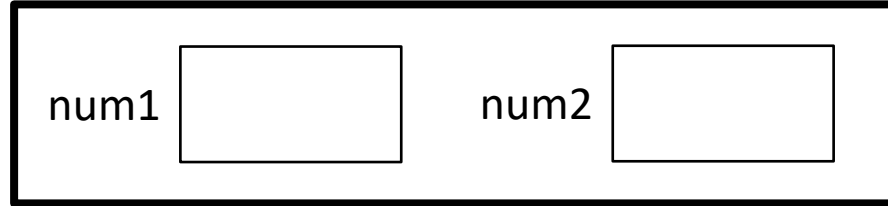


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

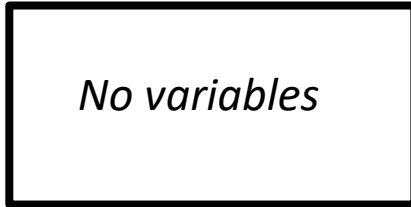
terminal

```
> python maxmax.py
```

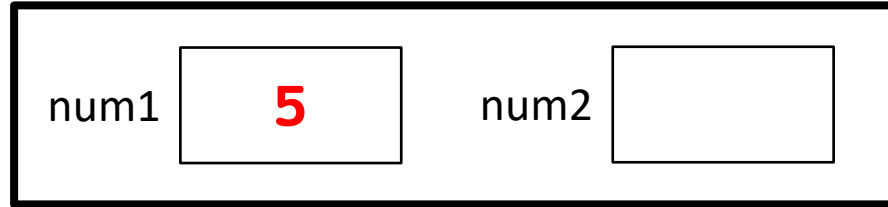


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

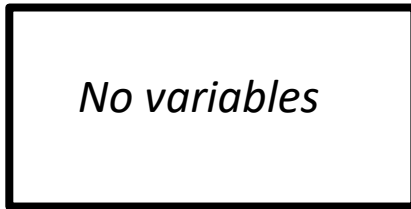
terminal

```
> python maxmax.py
```

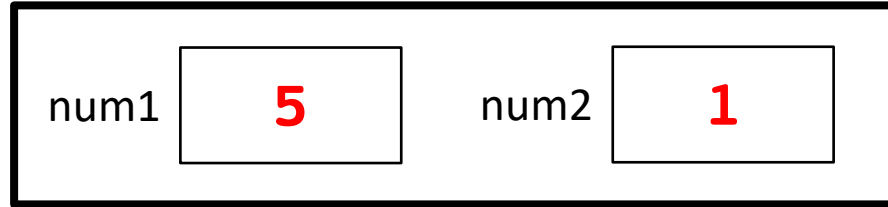


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

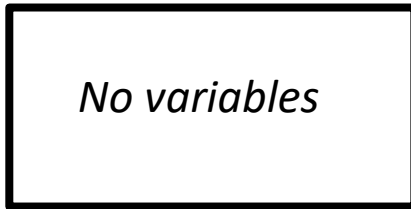
terminal

```
> python maxmax.py
```

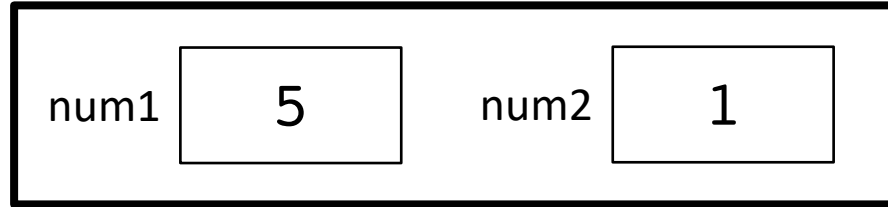


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

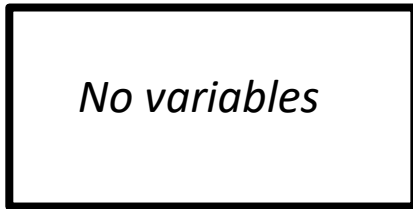
terminal

```
> python maxmax.py
```

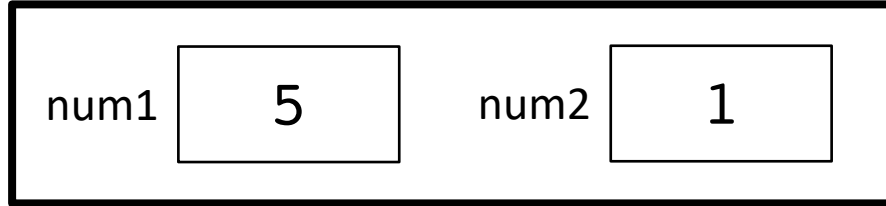


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

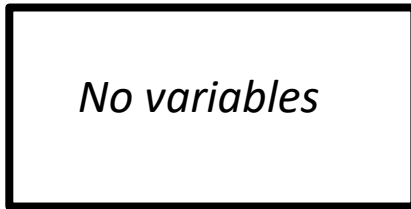
terminal

```
> python maxmax.py
```

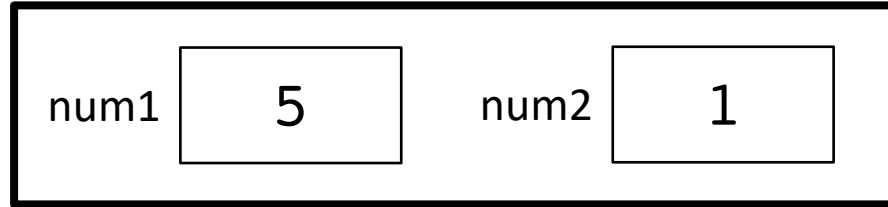


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

5

```
def main():  
    larger = max(5, 1)
```

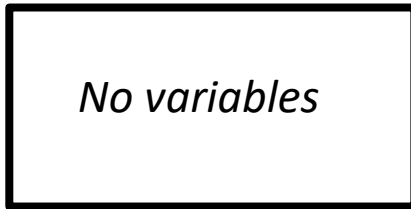
terminal

```
> python maxmax.py
```

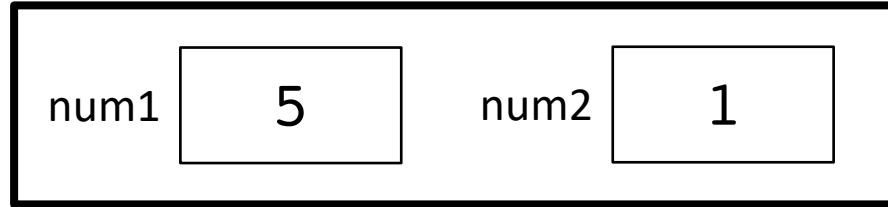


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

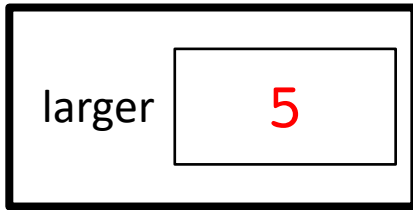
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

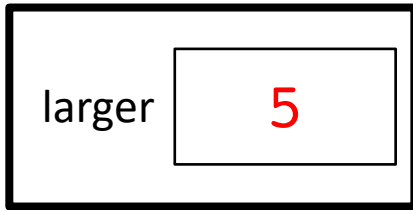
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```



terminal

```
> python maxmax.py
```



Multiple Return Statements

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2  
  
def main():  
    larger = max(5, 1)
```



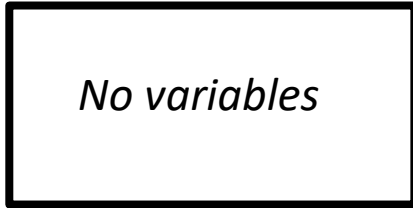
Multiple Return Statements

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2  
  
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

main memory



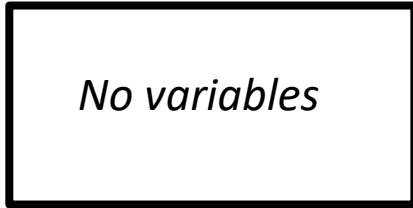
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

main memory



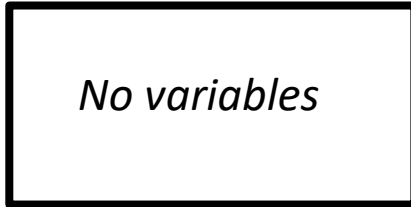
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

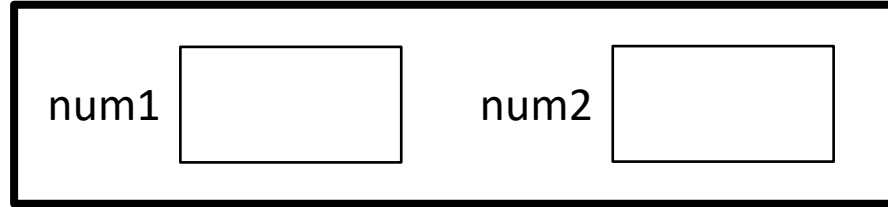


Multiple Return Statements

main memory



max memory



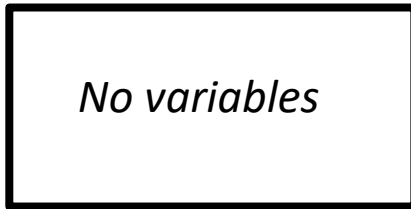
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

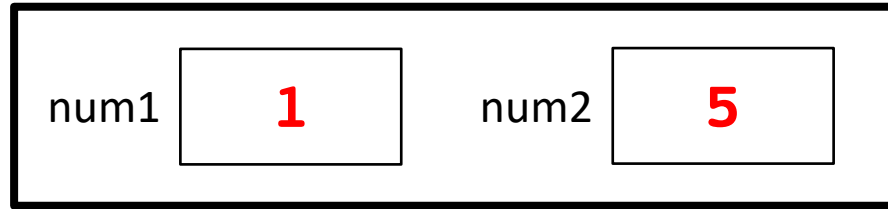


Multiple Return Statements

main memory



max memory



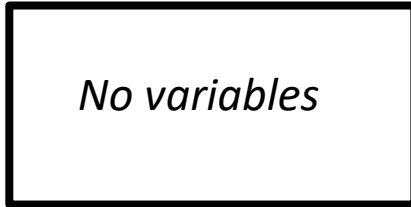
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

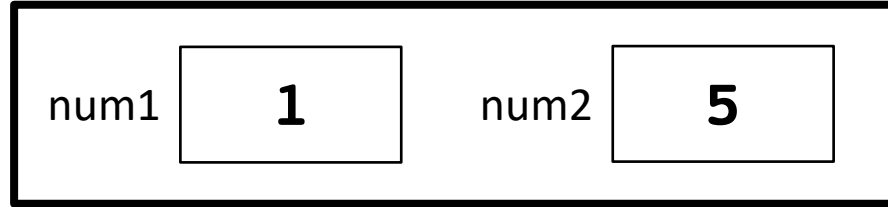


Multiple Return Statements

main memory



max memory



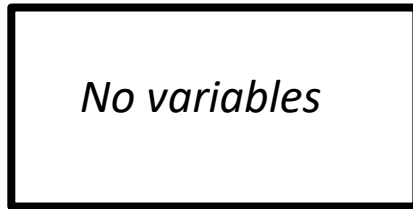
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

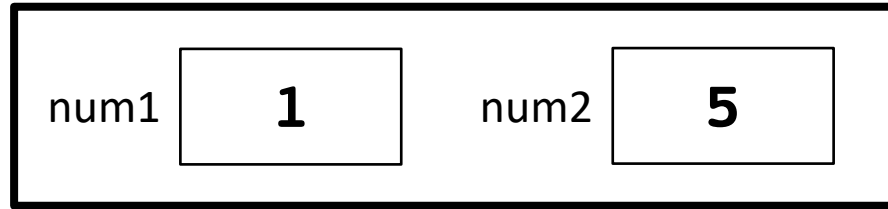


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1
```

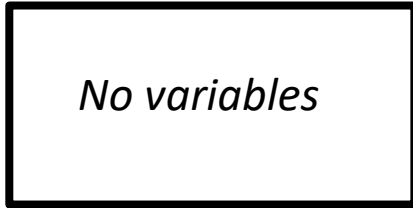
```
    return num2 5
```

```
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

main memory



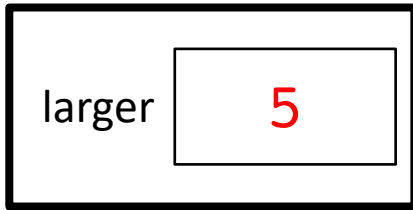
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

main memory



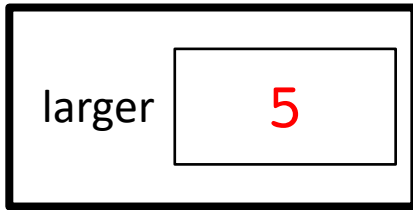
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

main memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```




```
if extra_time:
```

Function for IO

[illegible]

What functions do you define?

