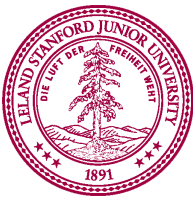


Review!

Boolean variables

```
karel_is_awesome = True
```

```
my_bool = 1 < 2
```



Boolean operations

```
p = True
```

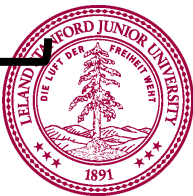
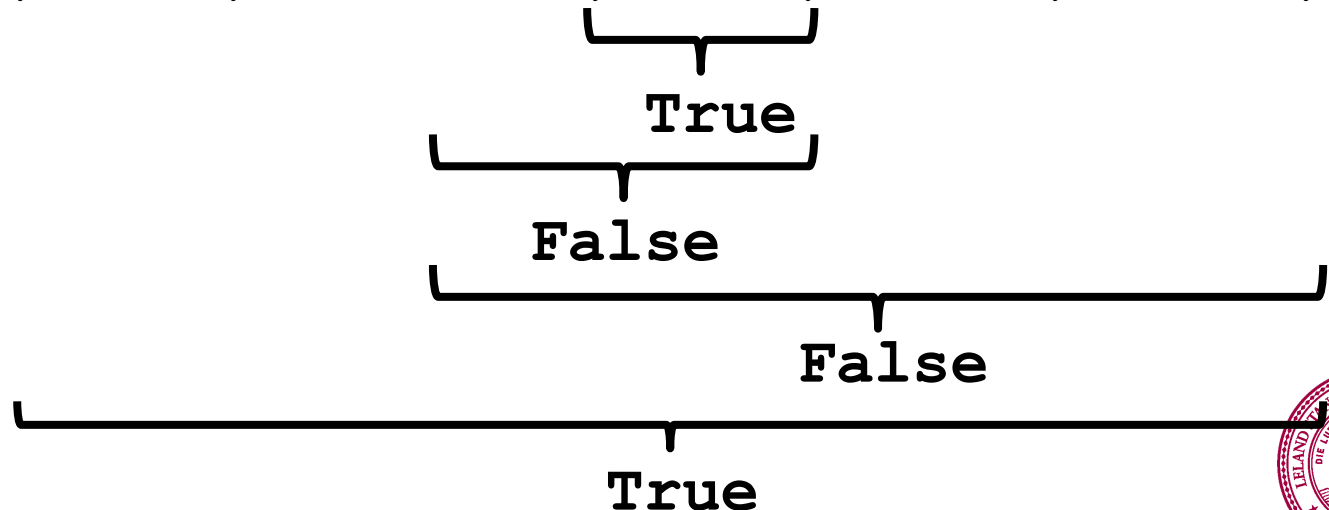
```
q = False
```

```
opposite = not p      # False
```

```
both_true = p and q   # False
```

```
either_true = p or q  # True
```

```
funky = (5 > 3) or not (1 < 2) and (6 == 6)
```



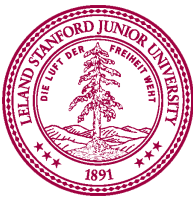
Count starting From 0

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

0
1
2
3
4
5
6
7
8
9

Index starts counting from 0

Counts up to, but not including the number in range



Review: `guessnumber.py`

Learn How To:

1. Write a function that takes in values
2. Write a function that returns a value
3. Trace function calls using stacks



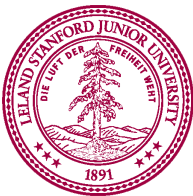
Recall, calling functions

```
print("hello world")
```

```
num = int(input("Enter number: "))
```

```
secret_number = random.randint(1, 100)
```

```
root = math.sqrt(num)
```

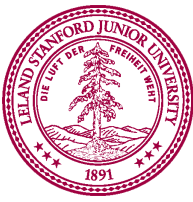


Defining a function

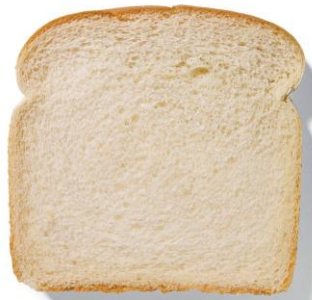
```
def turn_right():  
    turn_left()  
    turn_left()  
    turn_left()
```



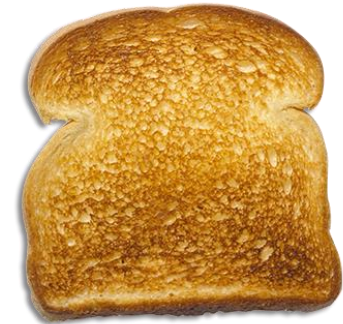
Big difference with python functions:
Python functions can **take in data**, and can **return data**!



A toaster is a function?!



parameter



return

A toaster is a function?!

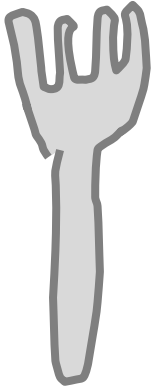


parameter



return

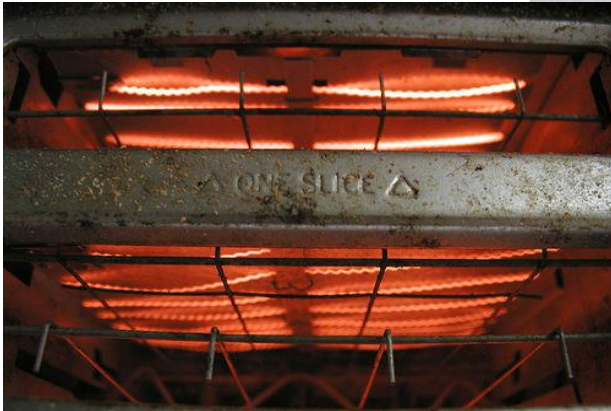
A toaster is a function?!



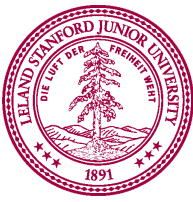
parameter



A toaster is a function?!



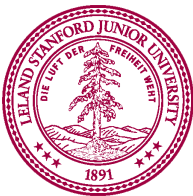
implementation



A toaster is a function?!



implementation



A toaster is a function?!



implementation



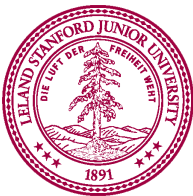
Abstraction



Civilization advances by extending the number of operations we can perform without thinking about them.

--Alfred North Whitehead

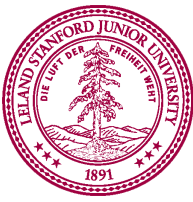
Abstraction is one of the most critical concept in computing



Anatomy of a function

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

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

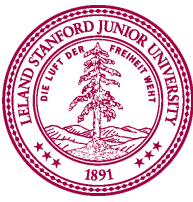


Anatomy of a function

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

function "definition"

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

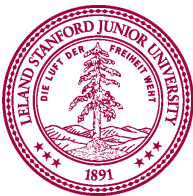


Anatomy of a function

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

function
name

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



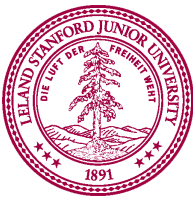
Anatomy of a function

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

input given

input expected

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



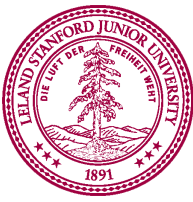
Anatomy of a function

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

arguments

parameters

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

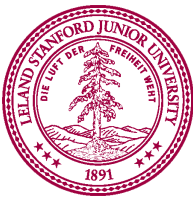


Anatomy of a function

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

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

function
body



Anatomy of a function

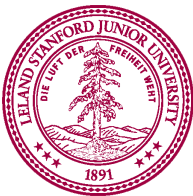
def main(): This call “evaluates” to the value returned

```
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

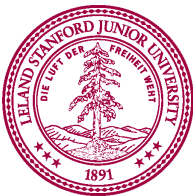


Anatomy of a function

Also a function call

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

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

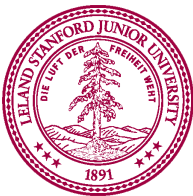


Anatomy of a function

No parameters (expects no input)

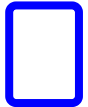
```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

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



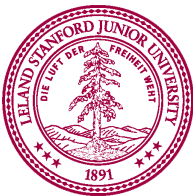
Anatomy of a function

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



When a function ends it “returns”

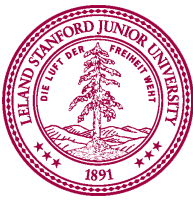
```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Formally

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

- ***name***: name of function (in **snake_case**)
- ***parameters***: information passed into function
- ***return***: information given back from the function



Parameters



Parameters let you provide a function with some information when you call it.

Parameters allow a function to be reused with different input information.



Program
user



Uses the
terminal
(or user
interface)

Coder:
function
caller



Uses helper
functions

Coder:
function
author



Writes helper
functions
other coders
can use



Program
user



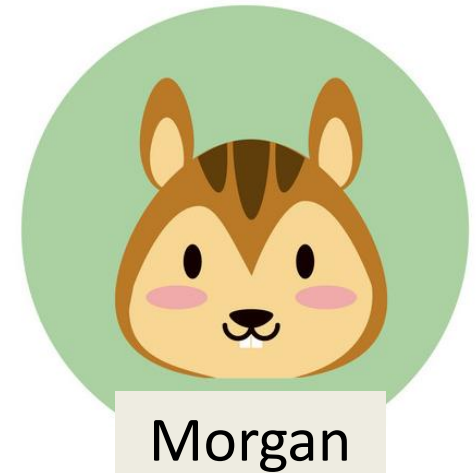
Uses the
terminal
(or user
interface)

Coder:
function
caller

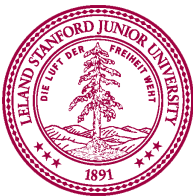


Uses helper
functions

Coder:
function
author



Writes helper
functions
other coders
can use



Anatomy of a function

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

Function caller



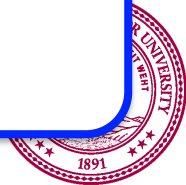
```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

Function author



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

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



Learn by Example



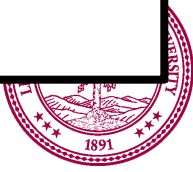
No Parameter, No Return

```
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.")
```

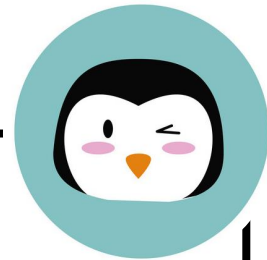
```
def main():  
    print_intro()
```



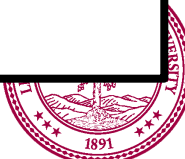
caller

terminal

```
> python intro.py
```



user



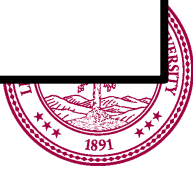
No Parameter, No Return

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

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

terminal

```
> python intro.py
```



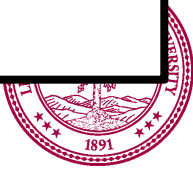
No Parameter, No Return

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

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

terminal

```
> python intro.py
```



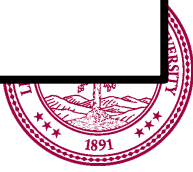
No Parameter, No Return

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

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

terminal

```
> python intro.py
```



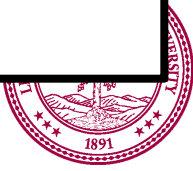
No Parameter, No Return

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

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

terminal

```
> python intro.py
```



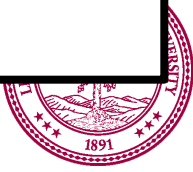
No Parameter, No Return

```
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
```



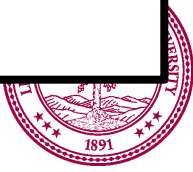
No Parameter, No Return

```
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

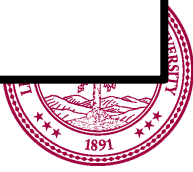
```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```



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

terminal

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



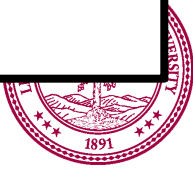
No Parameter, No Return

```
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

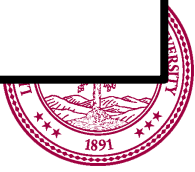
```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

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



terminal

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



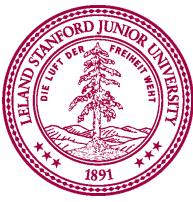
OMG! Let's do another one!

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

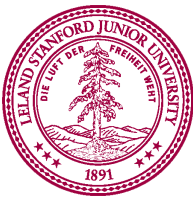
No variables

```
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

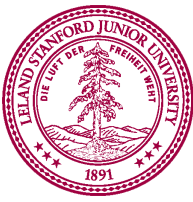
No variables

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

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

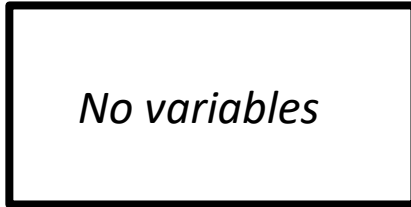
terminal

```
> python opinion.py
```



Parameter Example

main memory



print_opinion memory

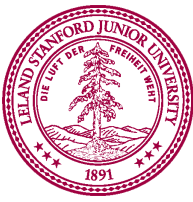


terminal

```
> python opinion.py
```

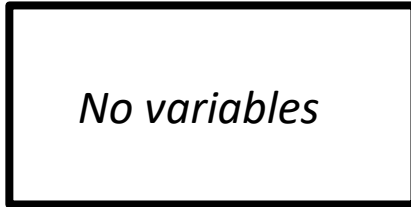
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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

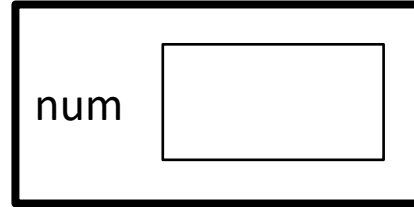


Parameter Example

main memory



print_opinion memory

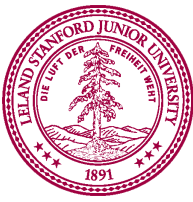


terminal

> python opinion.py

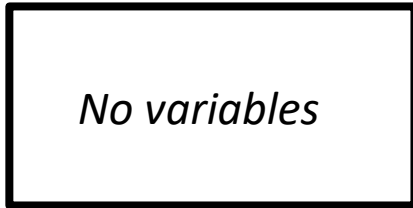
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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

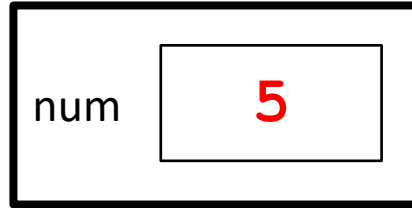


Parameter Example

main memory



print_opinion memory

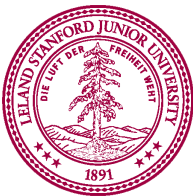


terminal

> python opinion.py

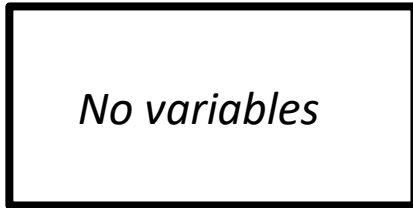
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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

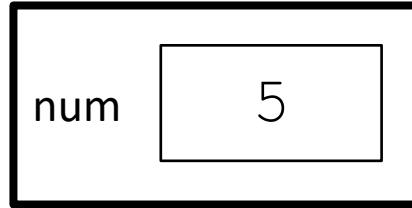


Parameter Example

main memory



print_opinion memory

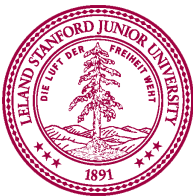


terminal

> python opinion.py

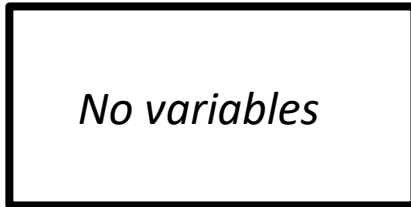
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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

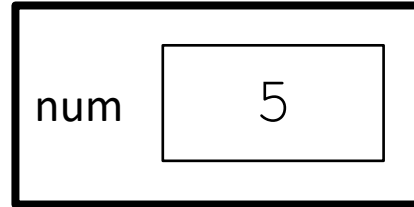


Parameter Example

main memory



print_opinion memory

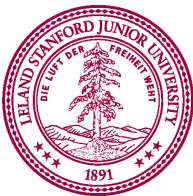


terminal

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

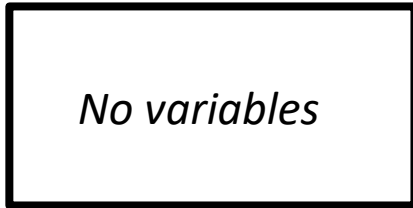
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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

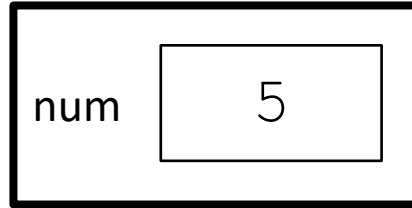


Parameter Example

main memory



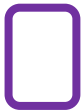
print_opinion memory



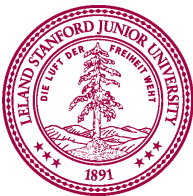
terminal

```
> 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)
```

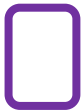


Parameter Example

main memory

No variables

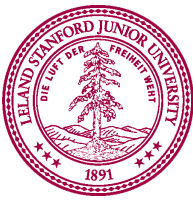
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```



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

terminal

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



Parameter Example

main memory

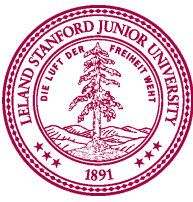
No variables

```
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!
```



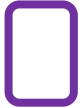
Parameter Example

main memory

No variables

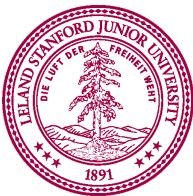
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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



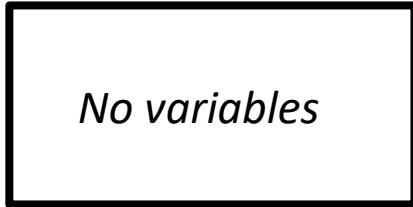
terminal

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



Parameter Example

main memory



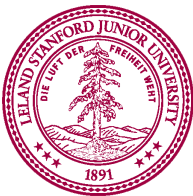
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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



terminal

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



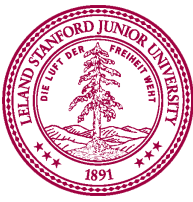
One more! Pleeeeeeze!

Parameter and Return Example

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

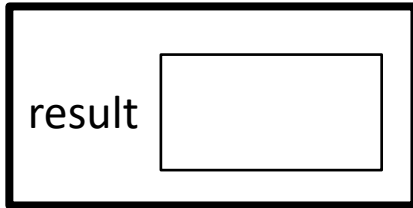
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory

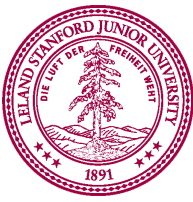


```
def meters_to_cm(meters):  
    return 100 * meters
```

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

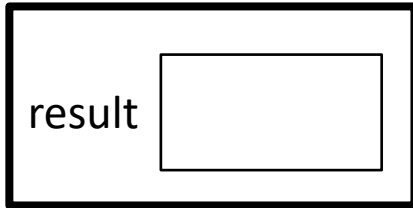
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory

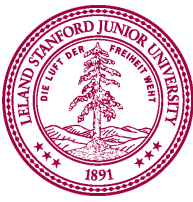


```
def meters_to_cm(meters):  
    return 100 * meters
```

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

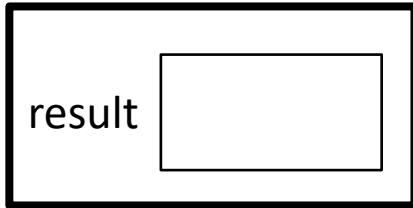
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory



metresToCm memory

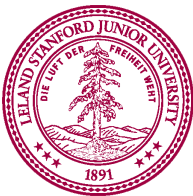


```
def metres_to_cm(meters):  
    return 100 * meters
```

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

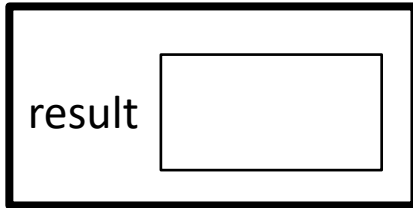
terminal

```
> python m2cm.py
```

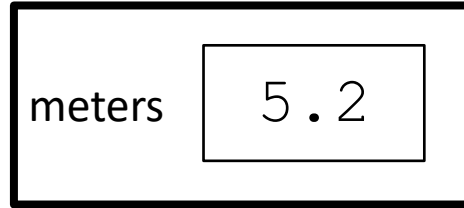


Parameter and Return Example

main memory



metersToCm memory

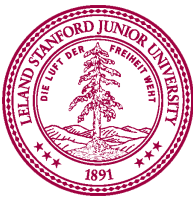


terminal

```
> python m2cm.py
```

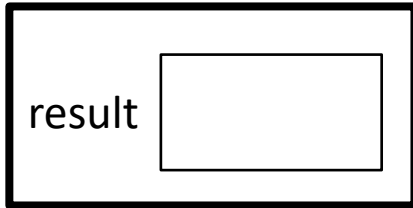
```
def meters_to_cm(meters):  
    return 100 * meters
```

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

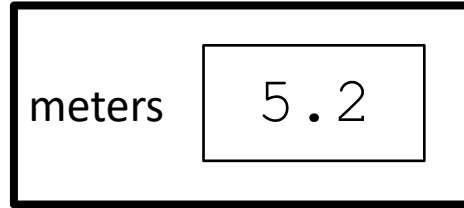


Parameter and Return Example

main memory



metersToCm memory



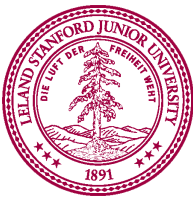
terminal

```
> python m2cm.py
```

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

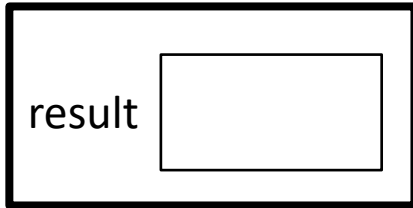
520.0

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



Parameter and Return Example

main memory



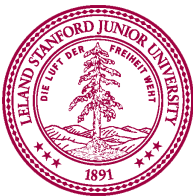
```
def meters_to_cm(meters):  
    return 100 * meters
```

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

520.0

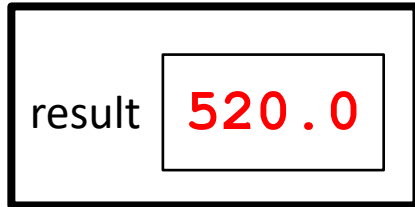
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory

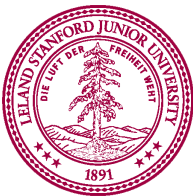


```
def meters_to_cm(meters):  
    return 100 * meters
```

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

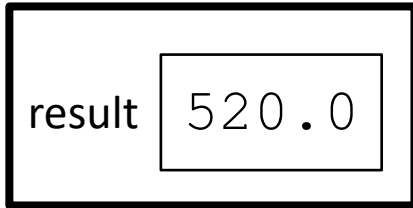
terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory



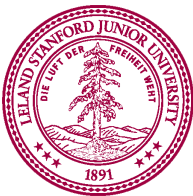
```
def meters_to_cm(meters):  
    return 100 * meters
```

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



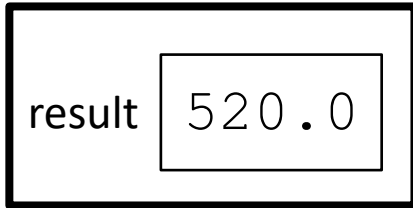
terminal

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



Parameter and Return Example

main memory



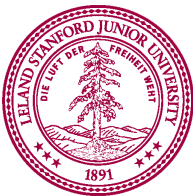
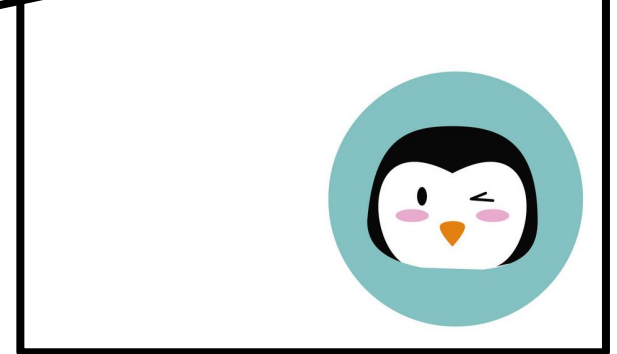
terminal

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

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

This is our favorite paradigm: function author returns value, function caller prints it.
It's the most flexible, modular. Its elegance is unparalleled.

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



More!

More!

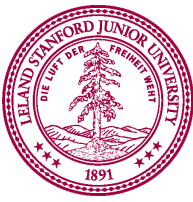
More!

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
```



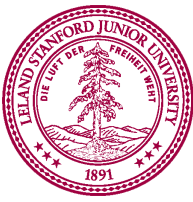
Parameter and Return Example

```
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
```



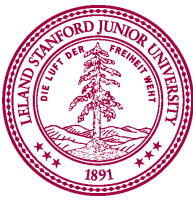
Parameter and Return Example

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

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

terminal

```
> python m2cm.py
```



Parameter and Return Example

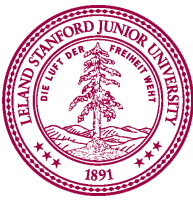
```
def meters_to_cm(meters):  
    return 100 * meters
```

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

520.0

terminal

```
> python m2cm.py
```



Parameter and Return Example

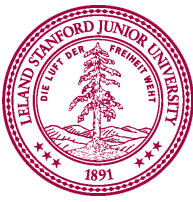
```
def meters_to_cm(meters):  
    return 100 * meters
```

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

520.0

terminal

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



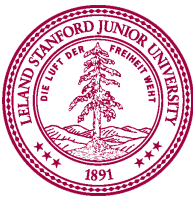
Parameter and Return Example

```
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

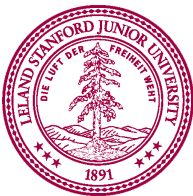
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
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

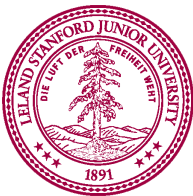
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
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

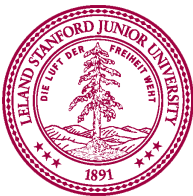


```
def meters_to_cm(meters):  
    return 100 * meters
```

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

terminal

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



Is returning
the same as printing?

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)
```



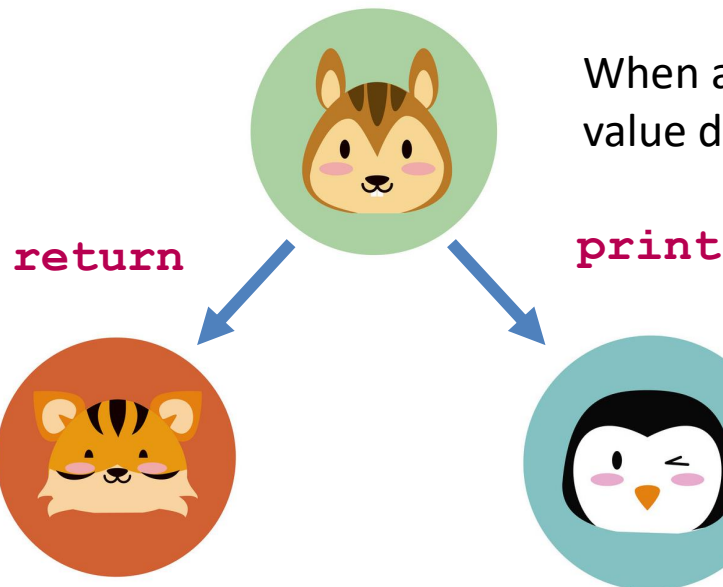
How is this function

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



Different than this function?

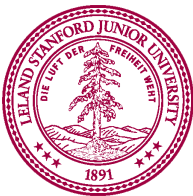
```
def meters_to_cm_case2(meters):  
    print(100 * meters)
```



When a function **produces** a value does it *print* or *return*?

Caller receives value.
Can store it and/or print it.

User sees value on the terminal.



Is returning
the same as printing?

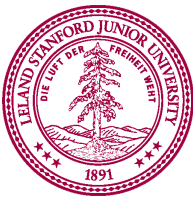
NO

Multiple Return Statements

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

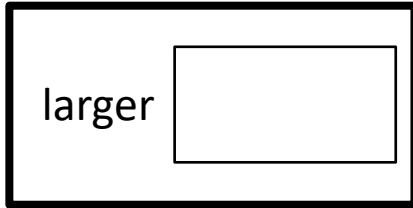
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

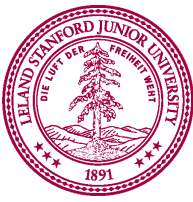


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

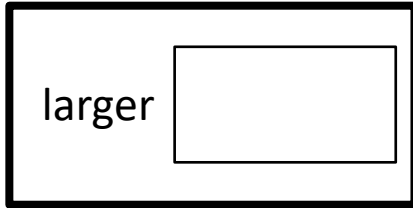
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

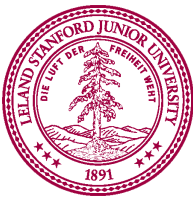


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

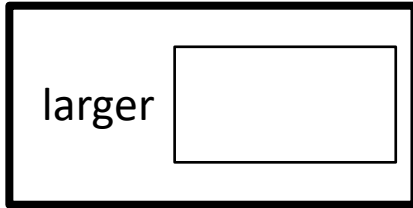
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

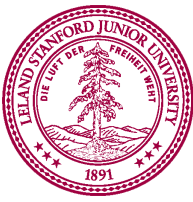


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

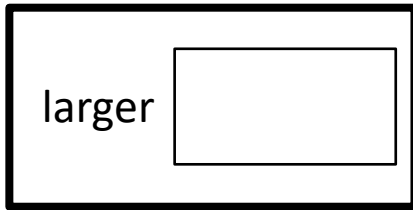
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

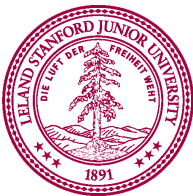


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

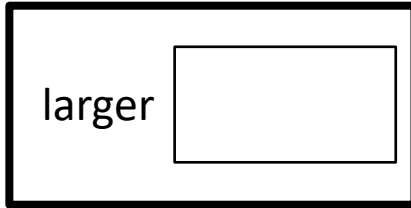
terminal

```
> python maxmax.py
```

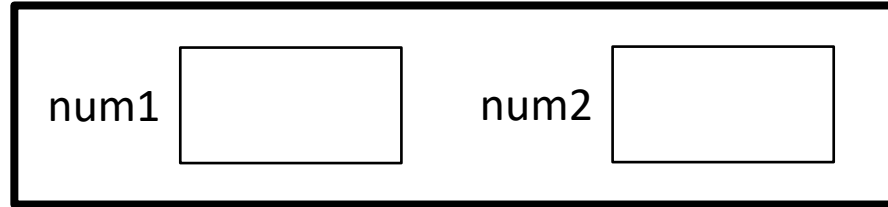


Multiple Return Statements

main memory



max memory

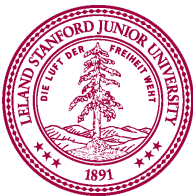


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

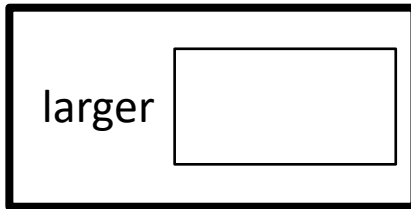
terminal

```
> python maxmax.py
```

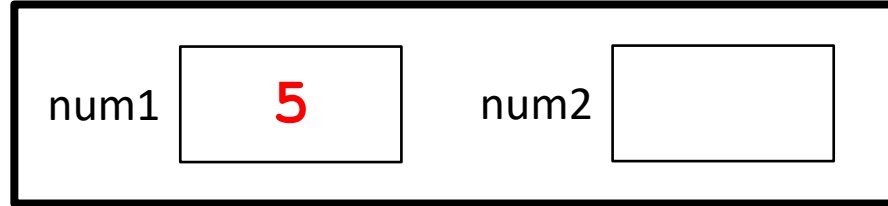


Multiple Return Statements

main memory



max memory

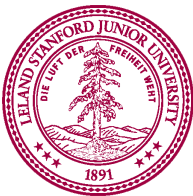


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

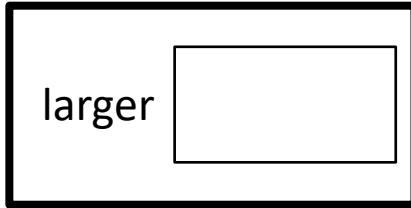
terminal

```
> python maxmax.py
```

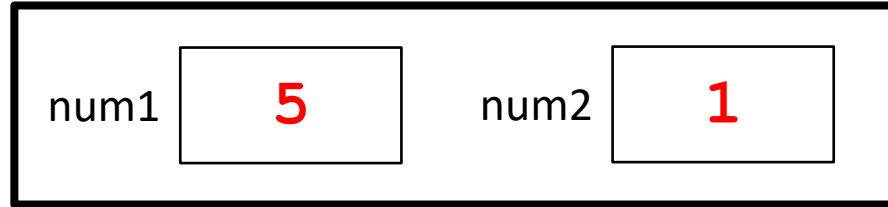


Multiple Return Statements

main memory



max memory

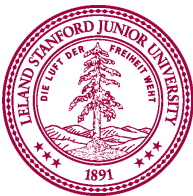


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

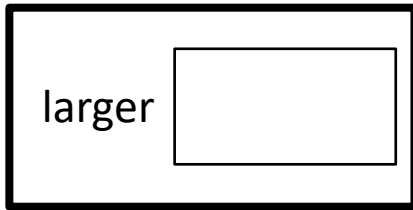
terminal

```
> python maxmax.py
```

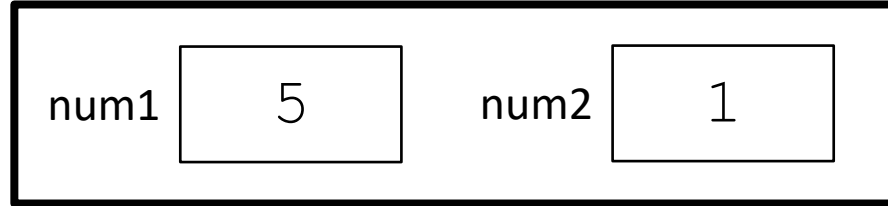


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):
```

```
    if num1 >= num2:
```

```
        return num1
```

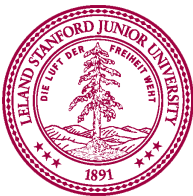
```
    return num2
```

```
def main():
```

```
    larger = max(5, 1)
```

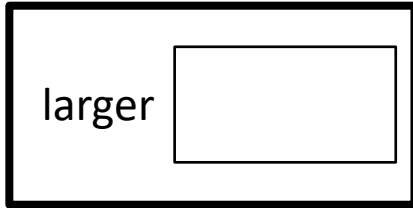
terminal

```
> python maxmax.py
```

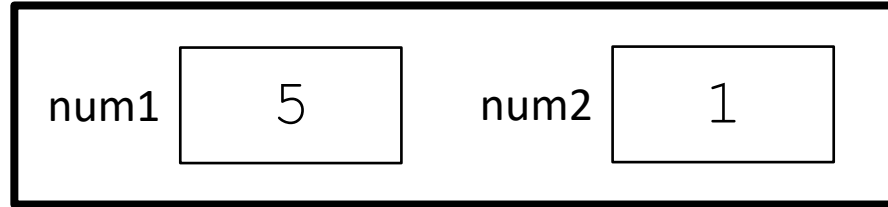


Multiple Return Statements

main memory



max memory

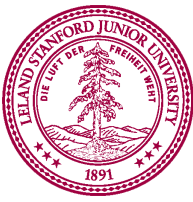


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

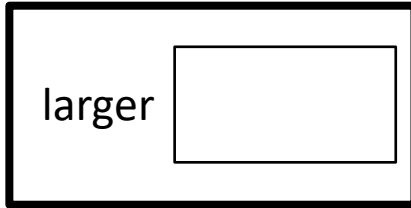
terminal

```
> python maxmax.py
```

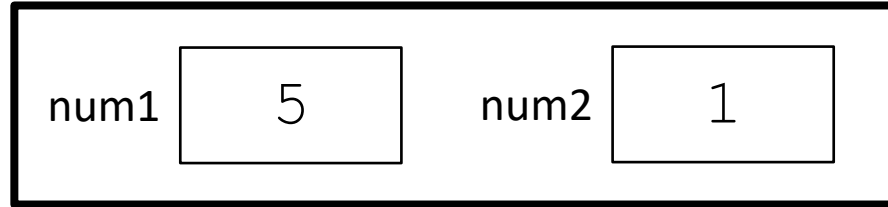


Multiple Return Statements

main memory



max memory

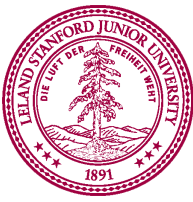


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

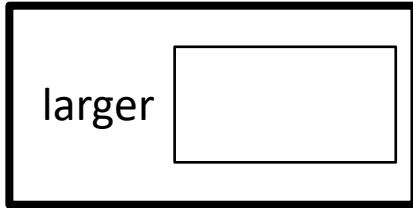
terminal

```
> python maxmax.py
```

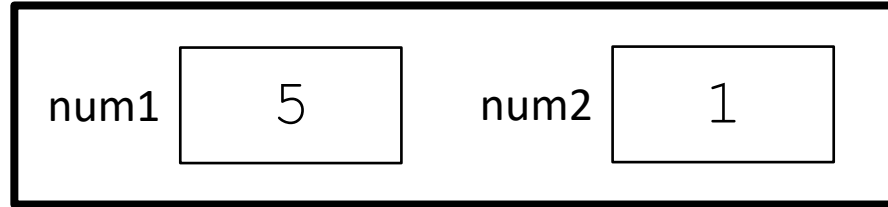


Multiple Return Statements

main memory



max memory

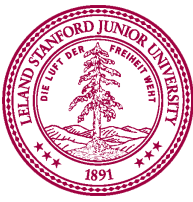


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

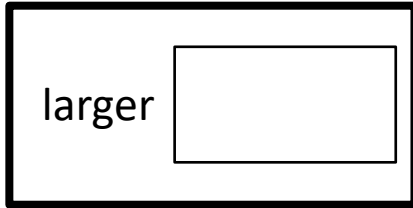
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

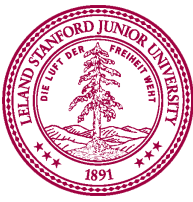


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

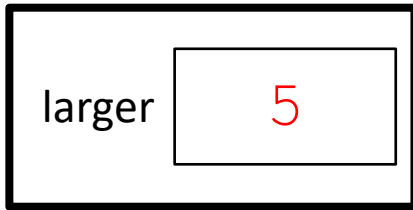
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

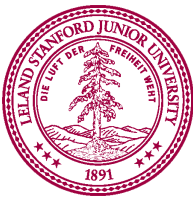


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

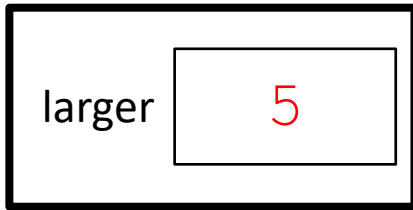
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory



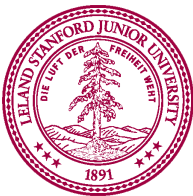
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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



terminal

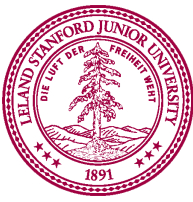
```
> python maxmax.py
```



What if we change the
order of arguments?

Multiple Return Statements

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

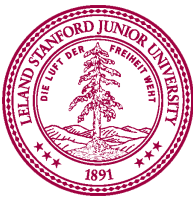


Multiple Return Statements

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

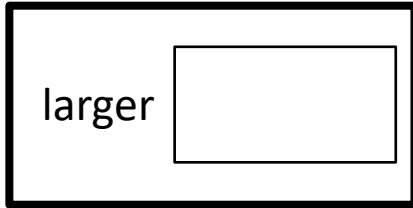
Changed the order of arguments

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



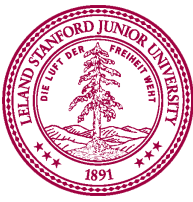
Multiple Return Statements

main memory



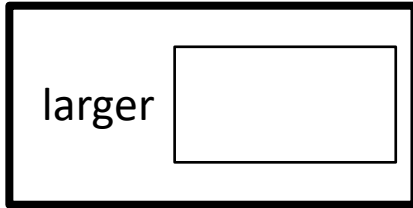
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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



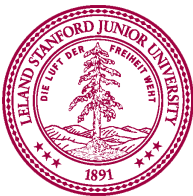
Multiple Return Statements

main memory

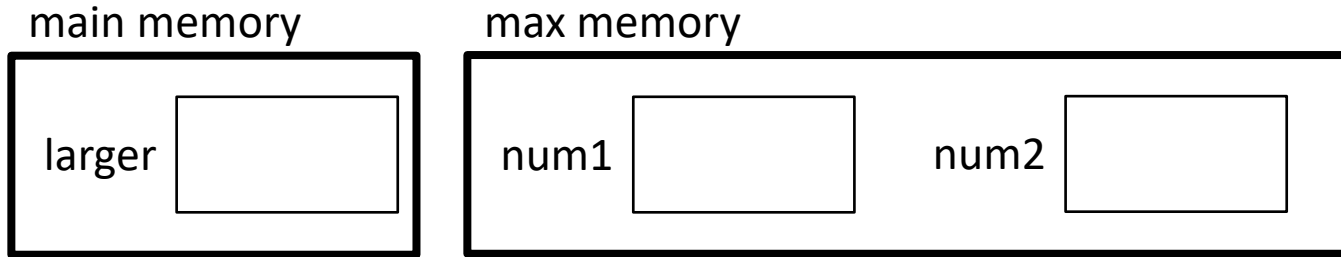


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

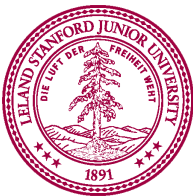


Multiple Return Statements

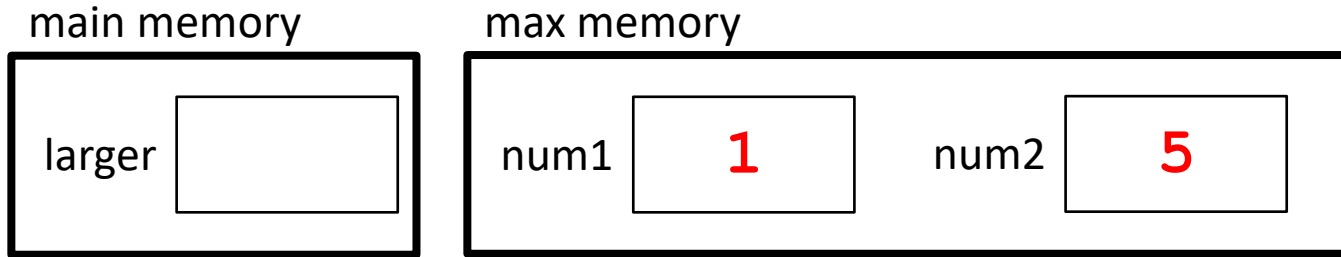


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

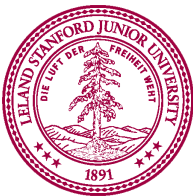


Multiple Return Statements

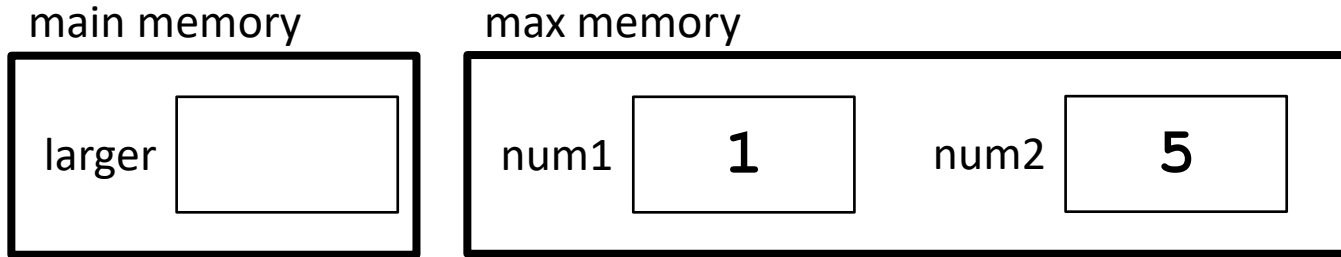


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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

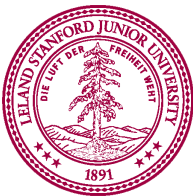


Multiple Return Statements

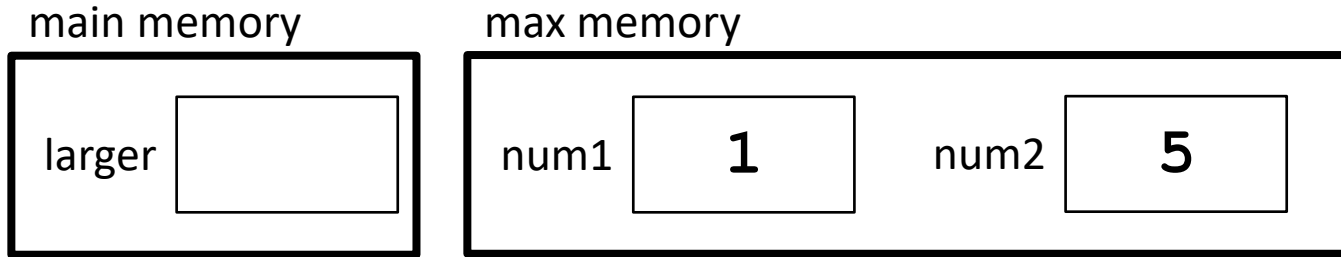


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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



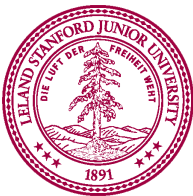
Multiple Return Statements



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

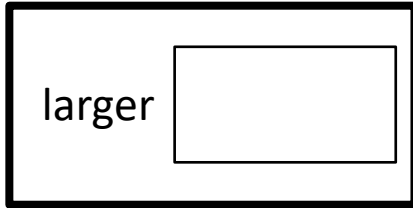
```
    return num2 5
```

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



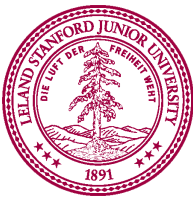
Multiple Return Statements

main memory



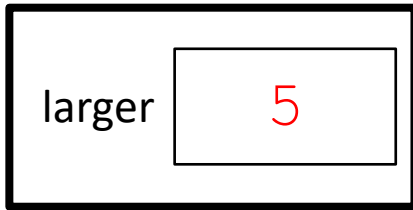
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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



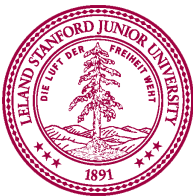
Multiple Return Statements

main memory



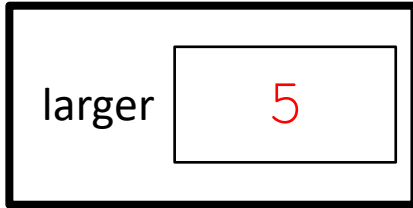
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

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



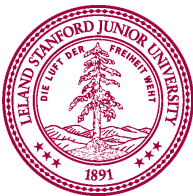
Multiple Return Statements

main memory



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

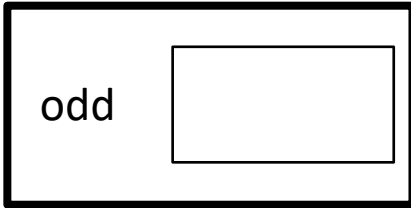
```
def main():  
    larger = max(1, 5)
```



Hey, can I return a bool?

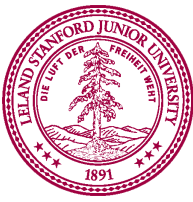
Predicate functions

main memory



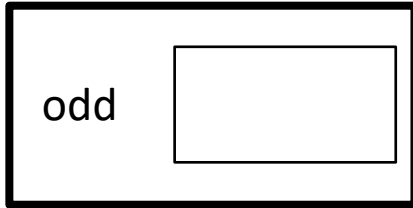
```
def is_odd(num) :  
    return (num % 2) == 1
```

```
def main() :  
    odd = is_odd(3)
```



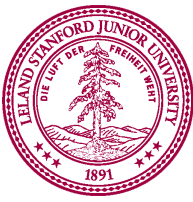
Predicate functions

main memory



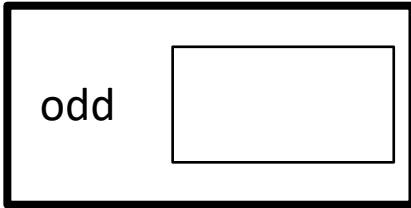
```
def is_odd(num) :  
    return (num % 2) == 1
```

```
def main() :  
    odd = is_odd(3)
```



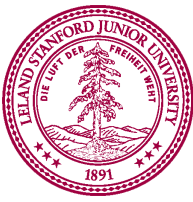
Predicate functions

main memory

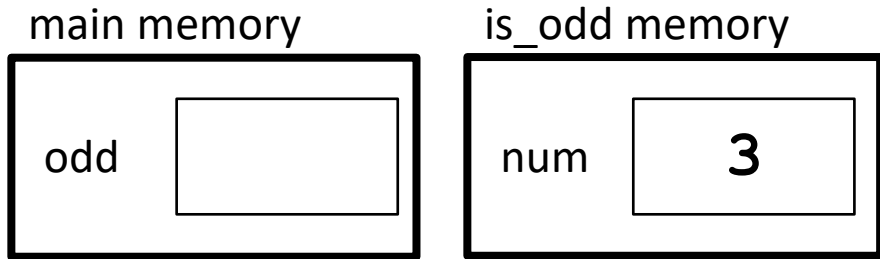


```
def is_odd(num) :  
    return (num % 2) == 1
```

```
def main() :  
    odd = is_odd(3)
```

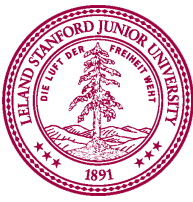


Predicate functions

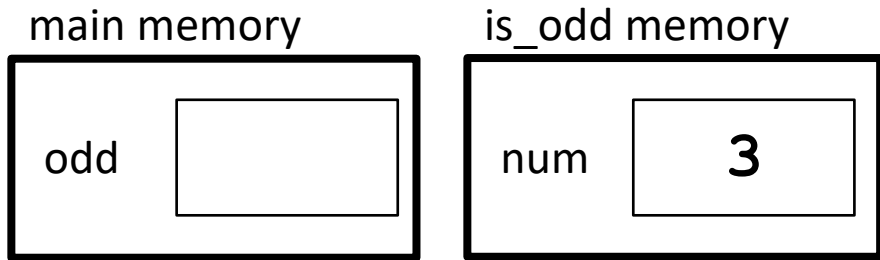


```
def is_odd(num):  
    return (num % 2) == 1
```

```
def main():  
    odd = is_odd(3)
```

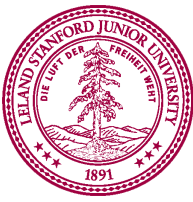


Predicate functions

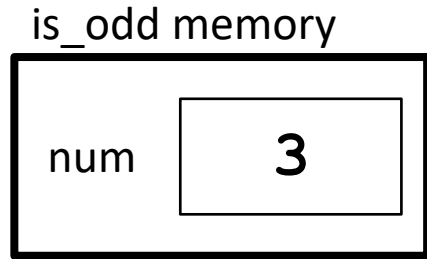
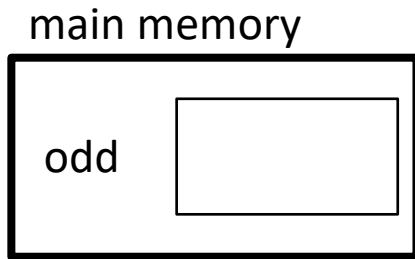


```
def is_odd(num):  
    return (num % 2) == 1
```

```
def main():  
    odd = is_odd(3)
```

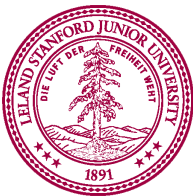


Predicate functions



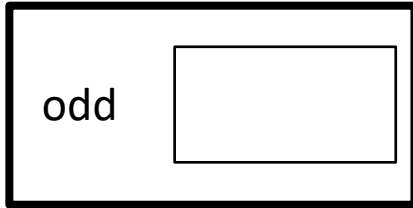
```
def is_odd(num):  
    return (num % 2) == 1 True
```

```
def main():  
    odd = is_odd(3)
```



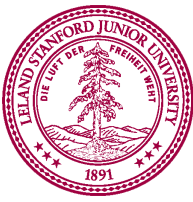
Predicate functions

main memory



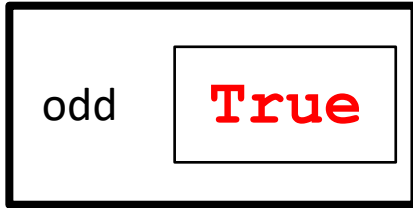
```
def is_odd(num) :  
    return (num % 2) == 1
```

```
def main() : True  
    odd = is_odd(3)
```



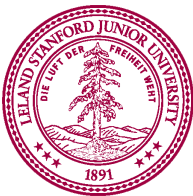
Predicate functions

main memory



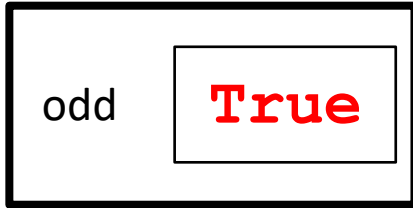
```
def is_odd(num) :  
    return (num % 2) == 1
```

```
def main() :  
    odd = is_odd(3)
```



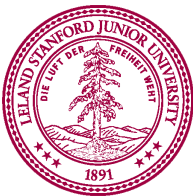
Predicate functions

main memory



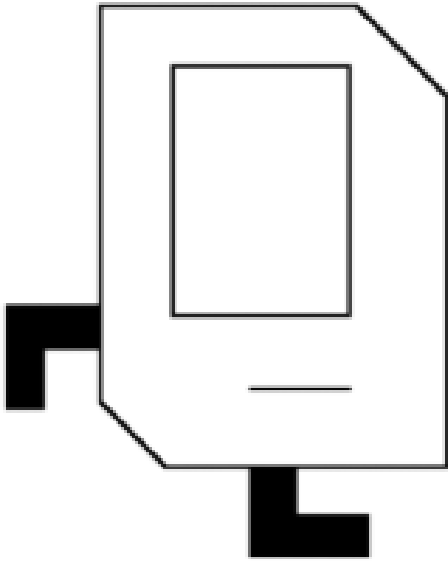
```
def is_odd(num) :  
    return (num % 2) == 1
```

```
def main() :  
    odd = is_odd(3)
```

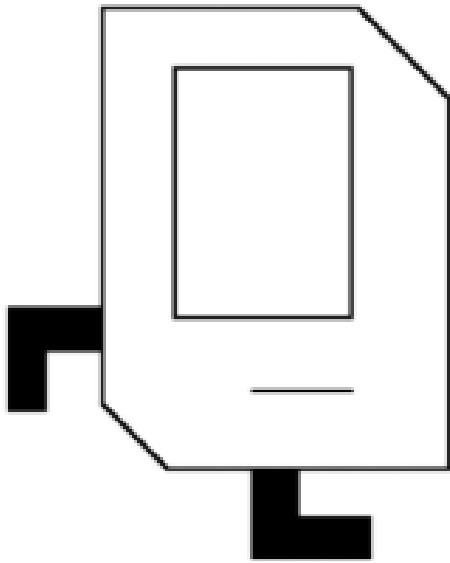


You're ready!

`are_we_related()`



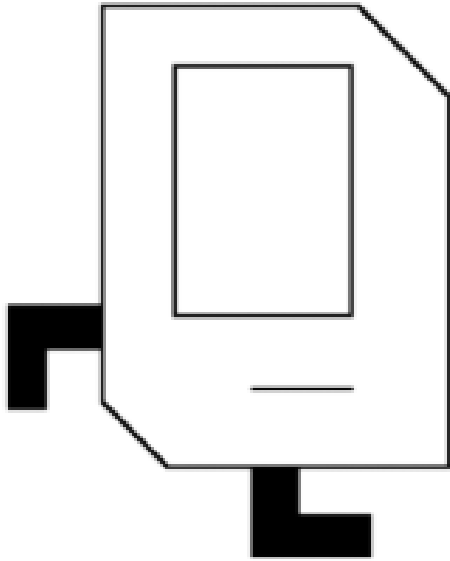
You're ready!



`i_am_your_father()`

You're ready!

noooooo ()



join_me()

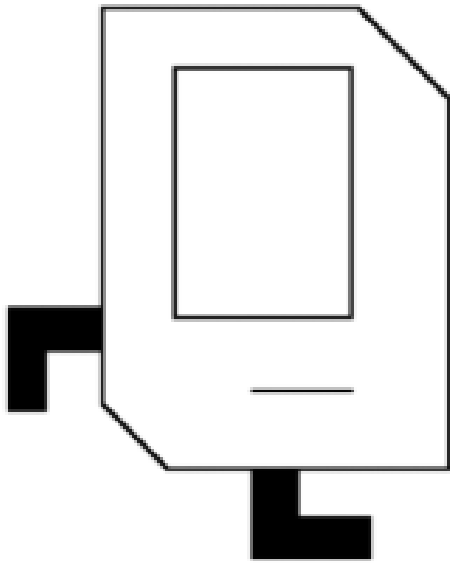


You're ready!



`together_we_can_rule_the_galaxy()`

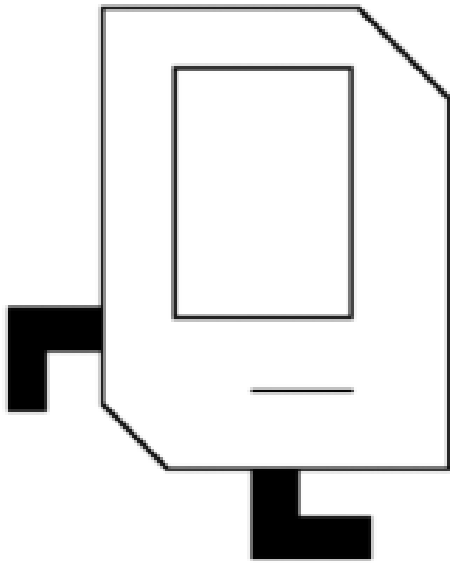
You're ready!



`as_function_and_robot()`

You're ready!

`im_just_here_for_the_beebers()`



```
if time_available():  
    superguessnumber.py
```