



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
```



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

```
i j
0 0
0 1
0 2
1 0
1 1
1 2
```



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

i j

i

j



There is an *i* in for loop

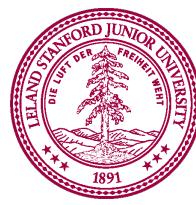
```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

```
i j
```

i 0

j



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

```
i j
```

i	0
j	0



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

```
i j
0 0
```

i	0
j	0



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

```
i j
0 0
```

i	0
j	1



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i, j)
```

terminal

```
i j
0 0
0 1
```

i	0
j	1



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

```
i j
0 0
0 1
```

i	0
j	2



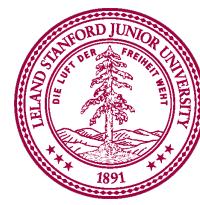
There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i, j)
```

terminal

```
i j
0 0
0 1
0 2
```

i	0
j	2



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

```
i j
0 0
0 1
0 2
```

i	0
j	2



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```



terminal

```
i j
0 0
0 1
0 2
```

i	0
j	2



There is an *i* in for loop

```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i, j)
```

terminal

```
i j
0 0
0 1
0 2
```

i 1

j



There is an *i* in for loop

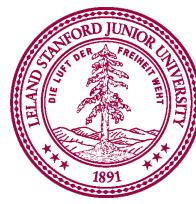
```
print('i', 'j')
for i in range(2):
    for j in range(3):
        print(i,j)
```

terminal

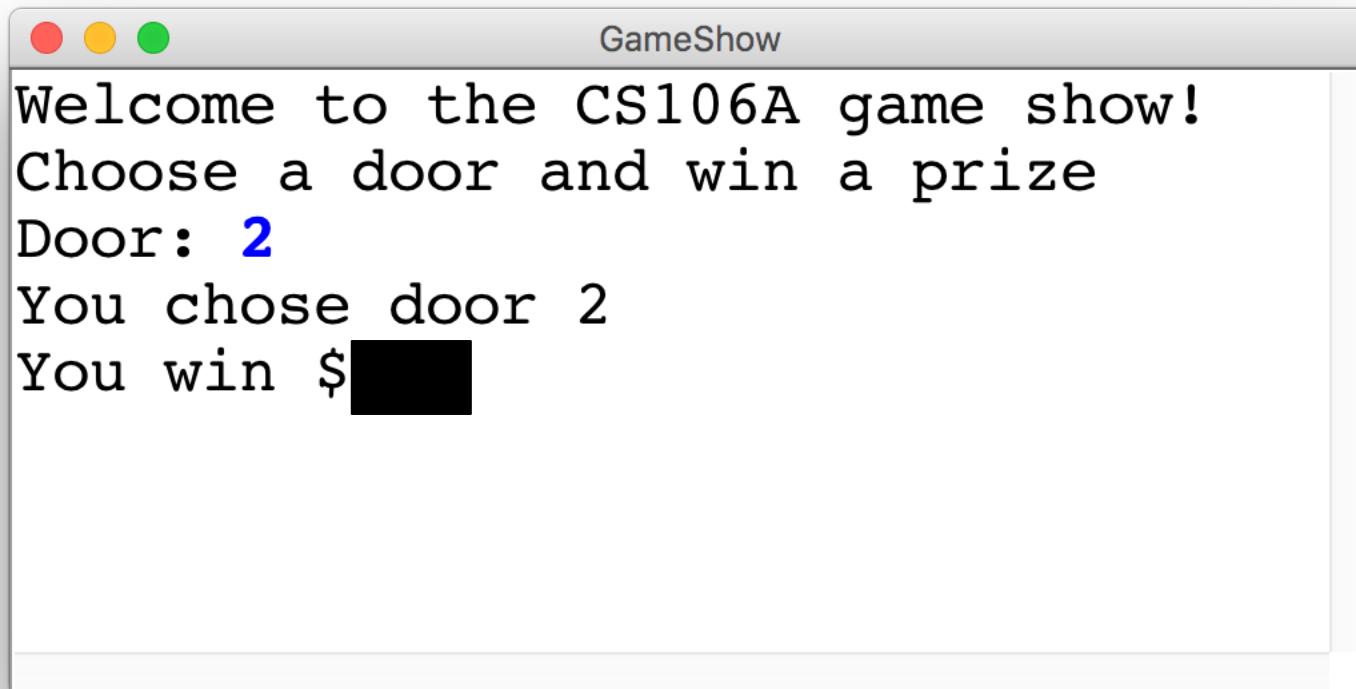
```
i j
0 0
0 1
0 2
1 0
1 1
1 2
```

i 1

j



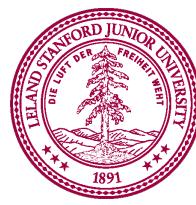
Game Show



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
```

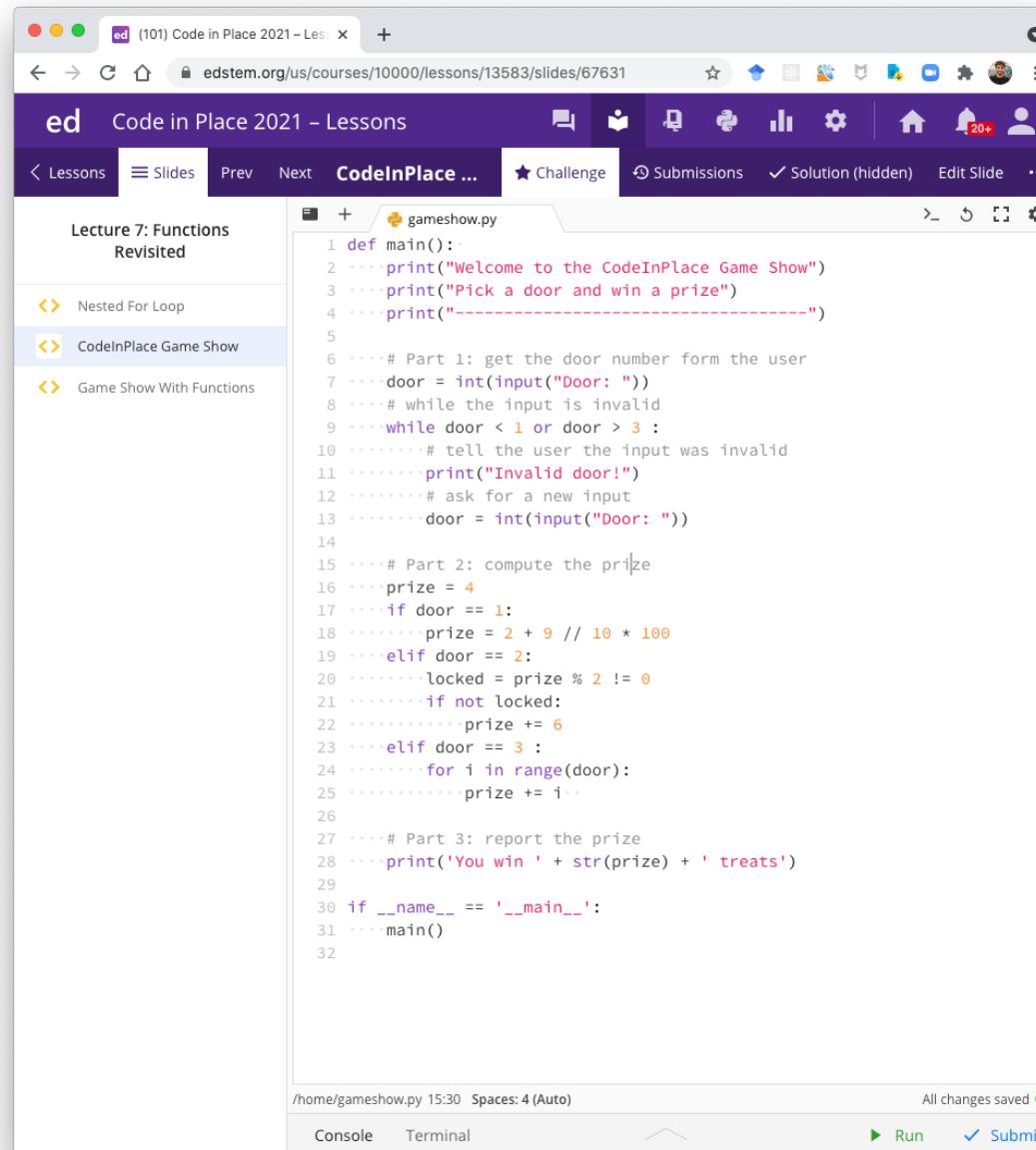


Learn How To:

1. Write a function that takes in information
2. Write a function that gives back information



Why can't we just add functions?



The screenshot shows a web-based code editor interface for a lesson titled "Lecture 7: Functions Revisited". The sidebar lists several sections: "Nested For Loop", "CodeInPlace Game Show" (which is currently selected), and "Game Show With Functions". The main area displays a Python script named "gameshow.py". The code implements a game show logic where the user inputs a door number (1-3) and receives a prize based on the number of doors chosen. The script uses nested loops and conditional statements to calculate the final prize value.

```
def main():
    print("Welcome to the CodeInPlace Game Show")
    print("Pick a door and win a prize")
    print("-----")

    door = int(input("Door: "))
    while door < 1 or door > 3 :
        print("Invalid door!")
        door = int(input("Door: "))

    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

    print('You win ' + str(prize) + ' treats')

if __name__ == '__main__':
    main()
```



Calling functions

`turn_right()`

`move()`

gives back the text the user entered
`input("string please! ")`

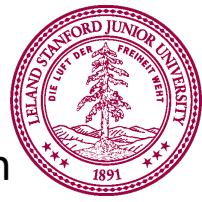
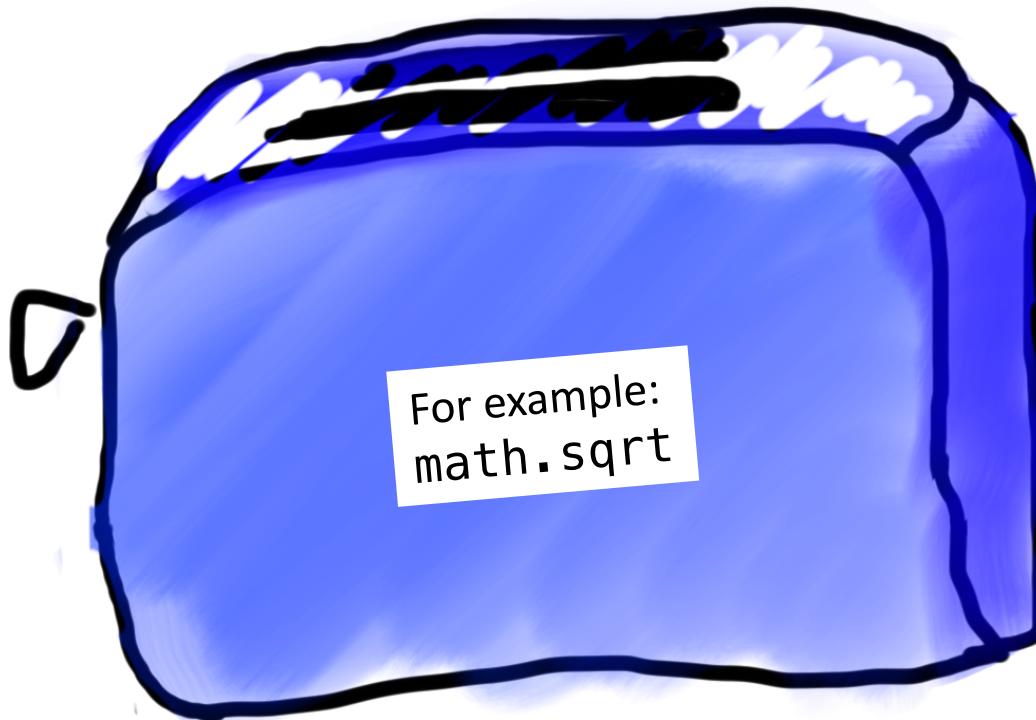
`print("hello world")`

gives back a float version
`float("0.42")`

gives back the square root
`math.sqrt(25)`



Toasters are functions



Toasters are functions



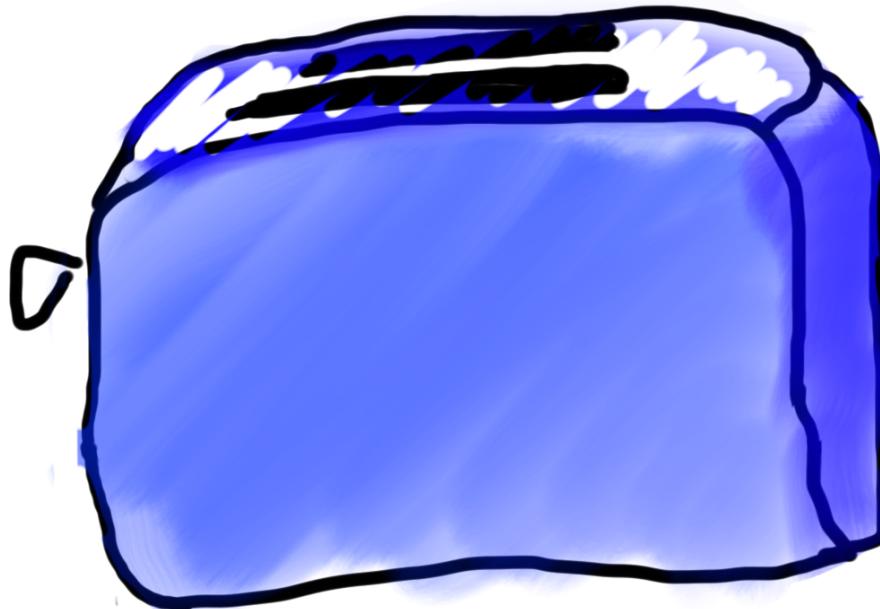
parameter



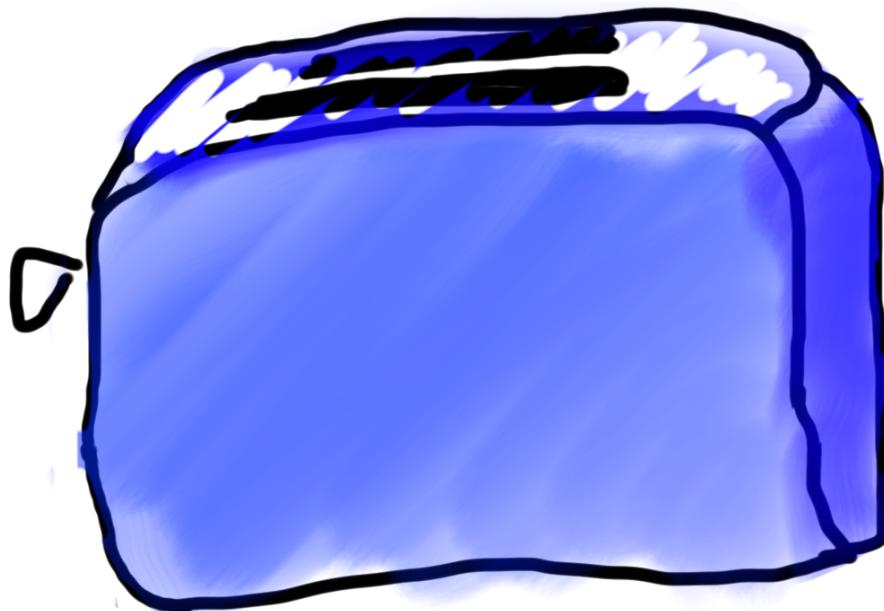
Toasters are functions



parameter



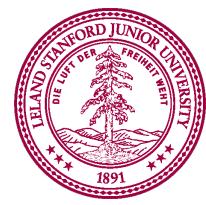
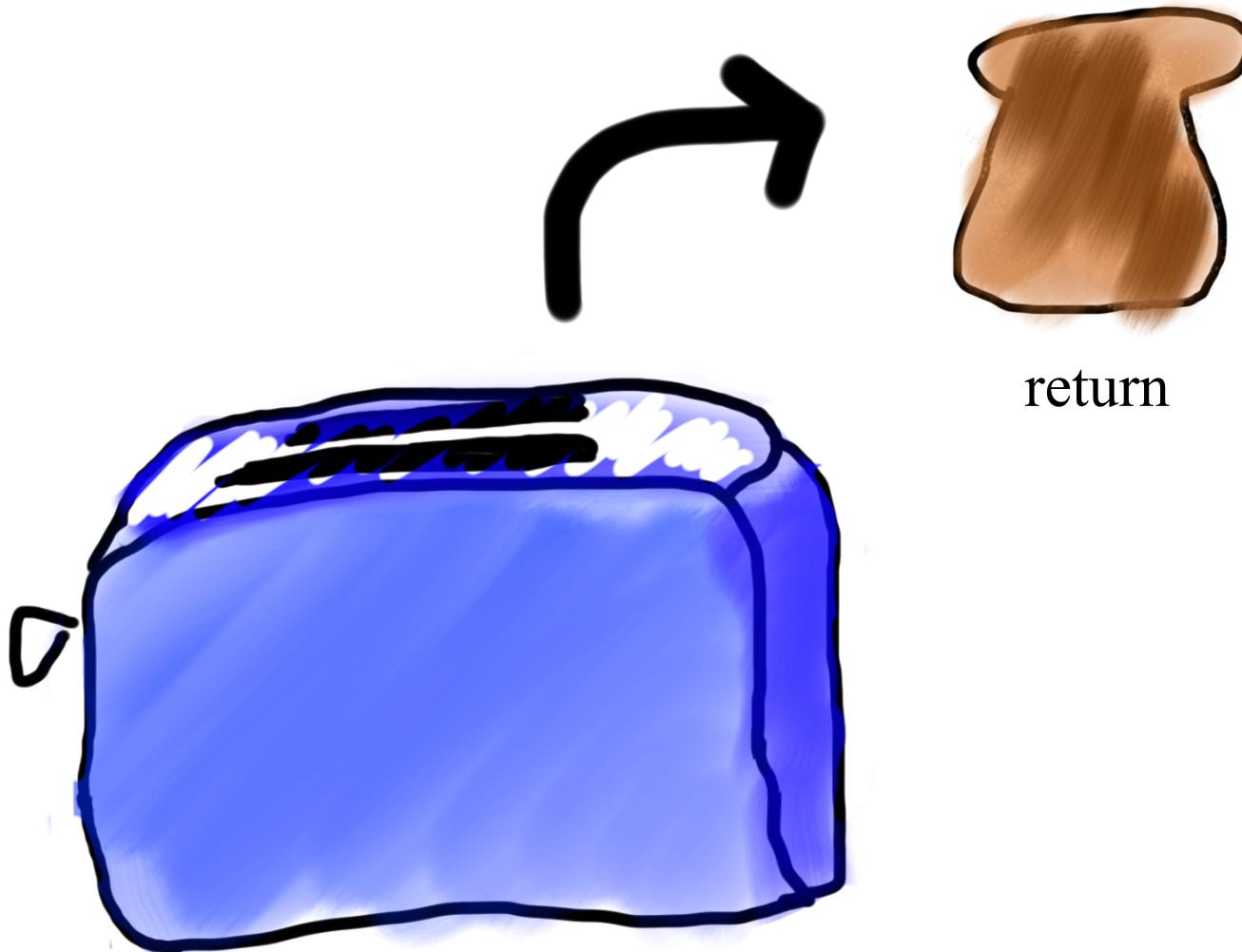
Toasters are functions



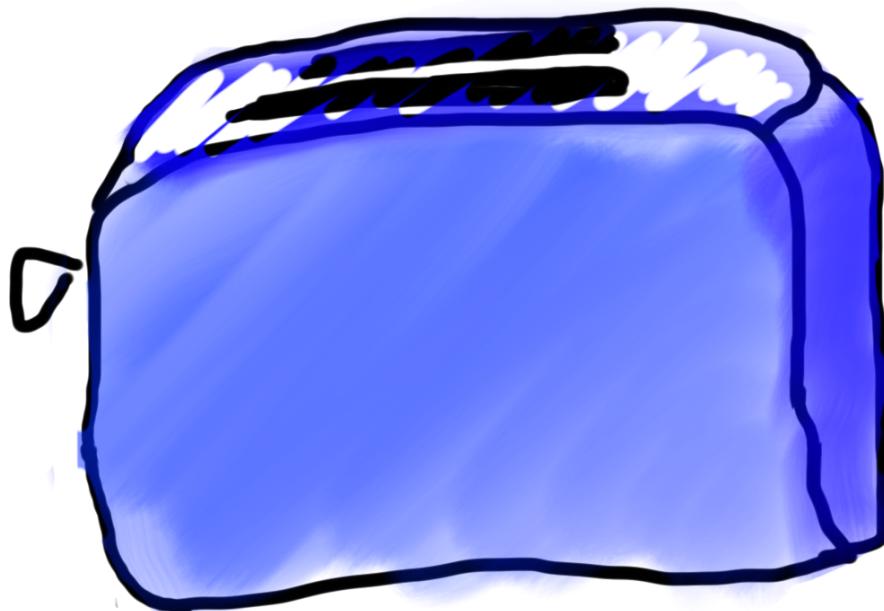
Toasters are functions



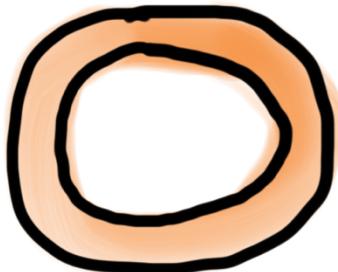
Toasters are functions



Toasters are functions



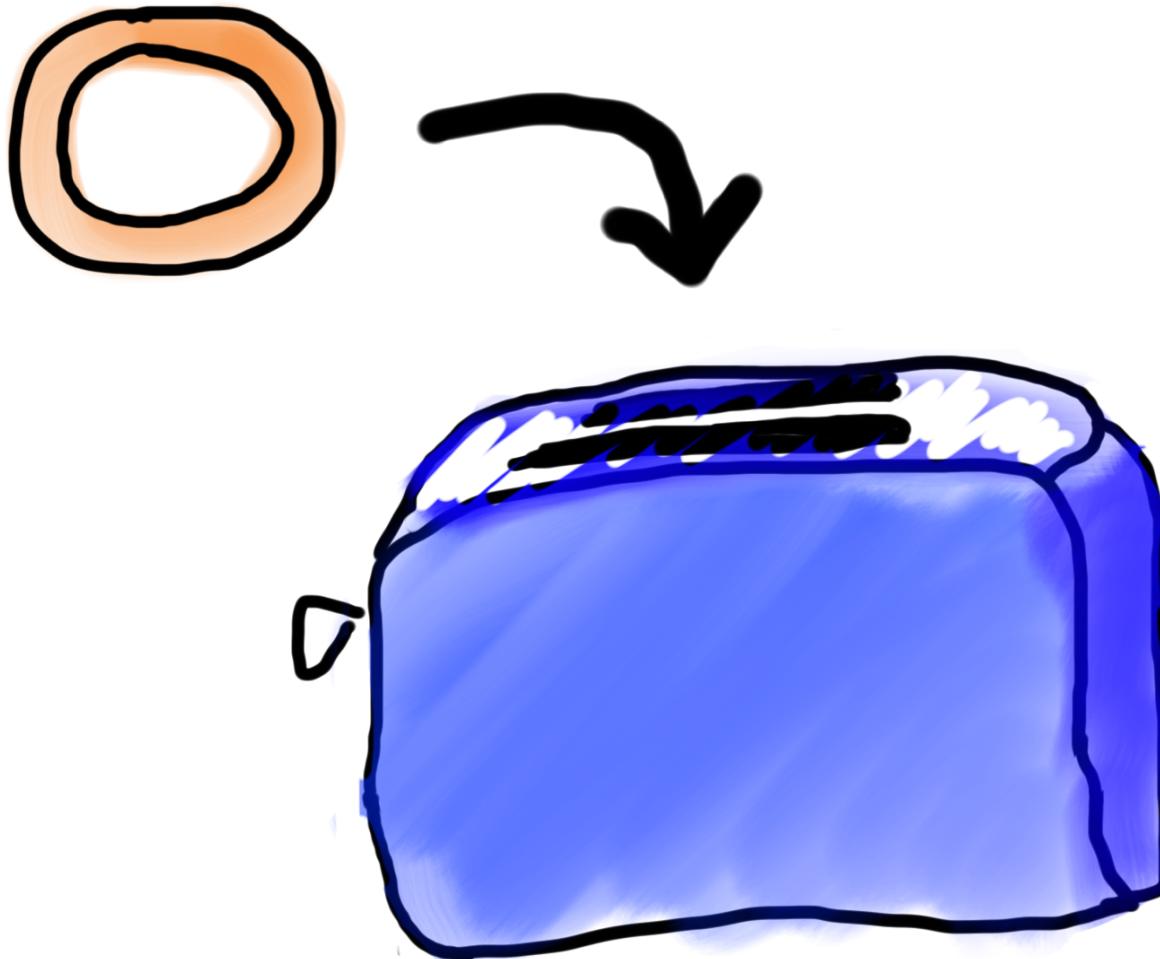
Toasters are functions



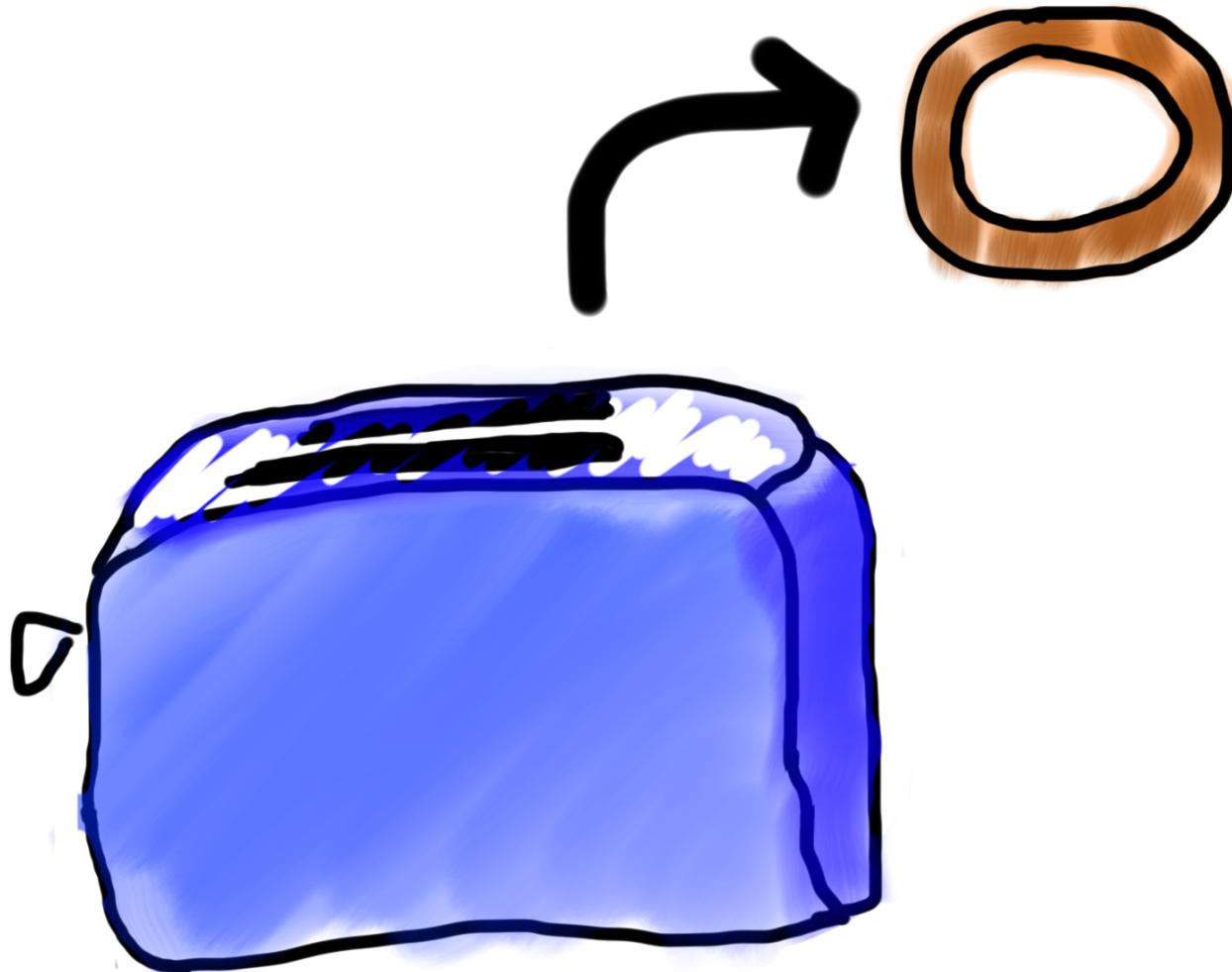
* You don't need a second toaster if you want to toast bagels. Use the same one.



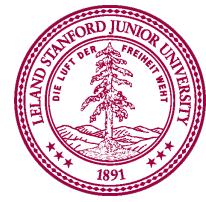
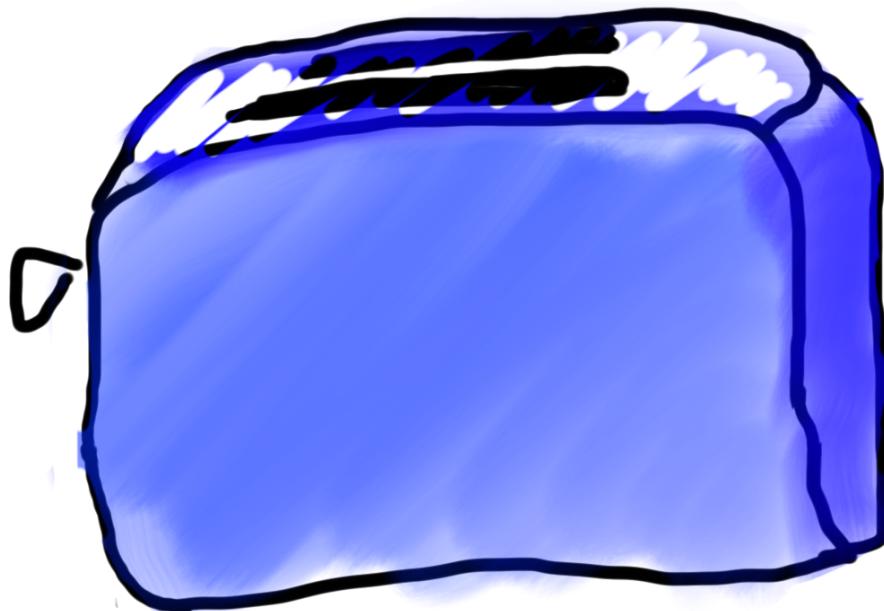
Toasters are functions



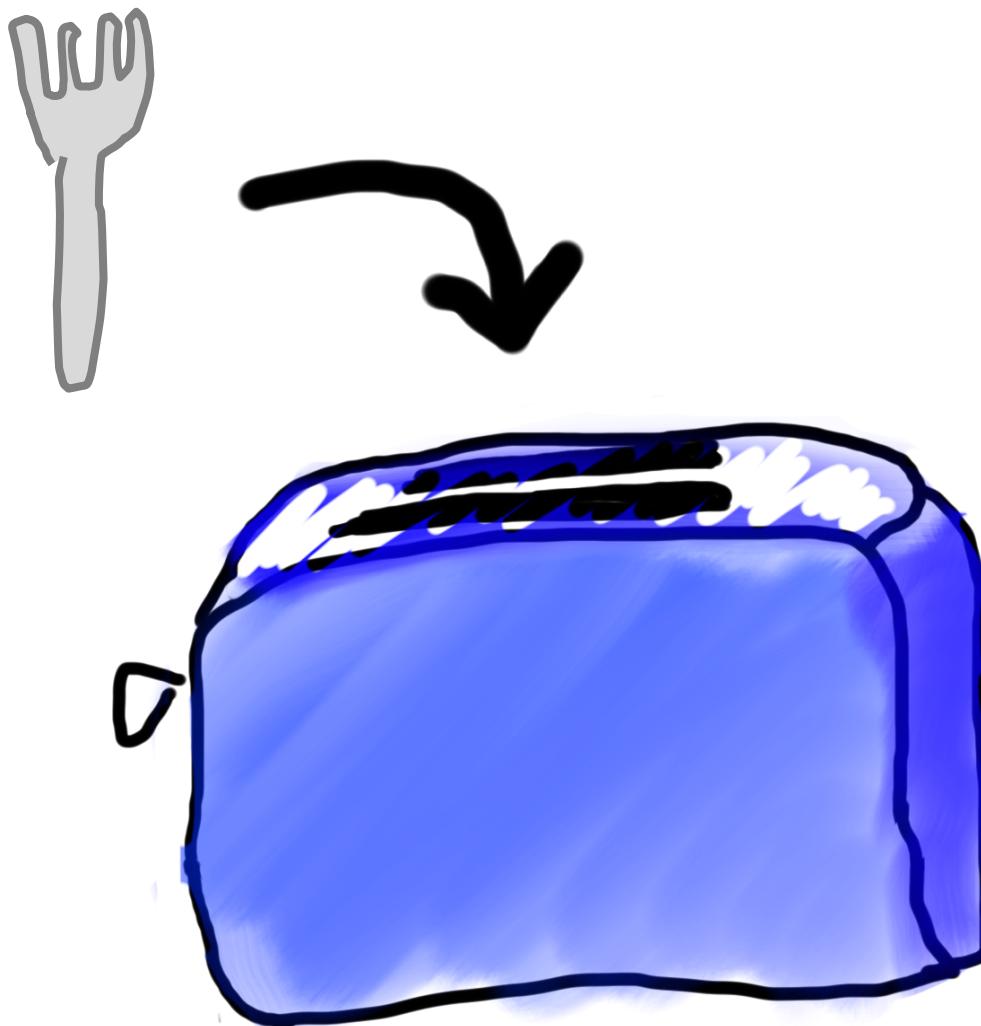
Toasters are functions



Toasters are functions



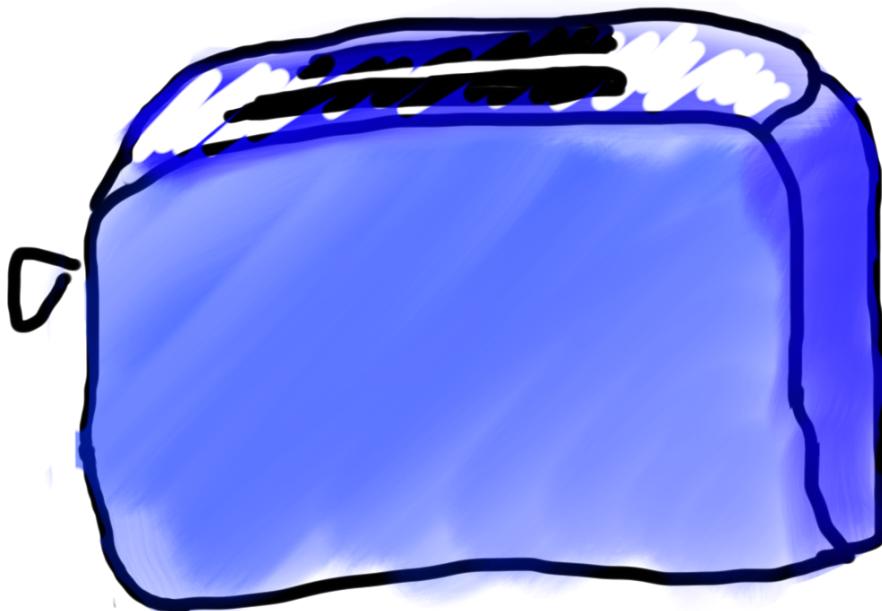
Toasters are functions



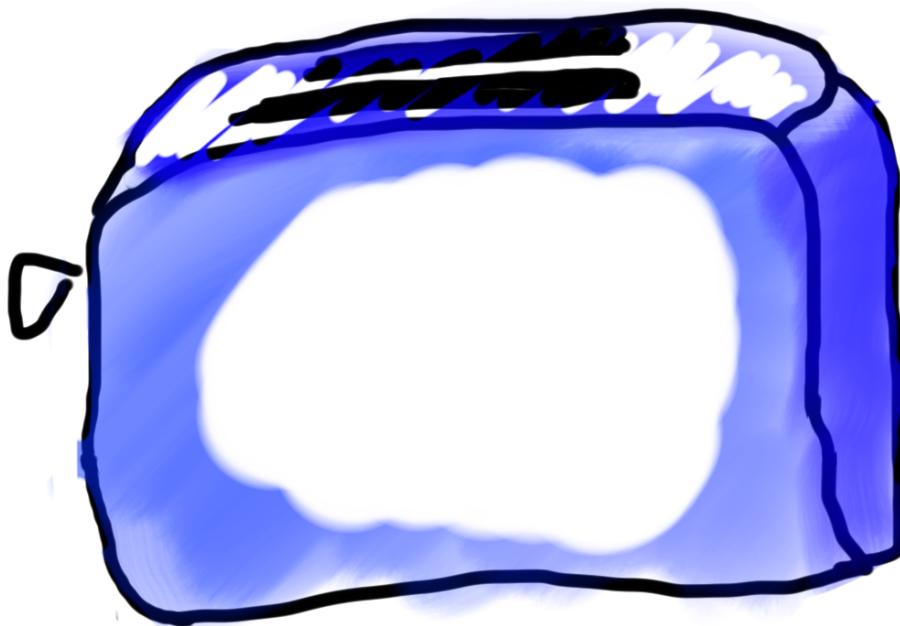
Toasters are functions



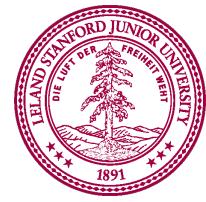
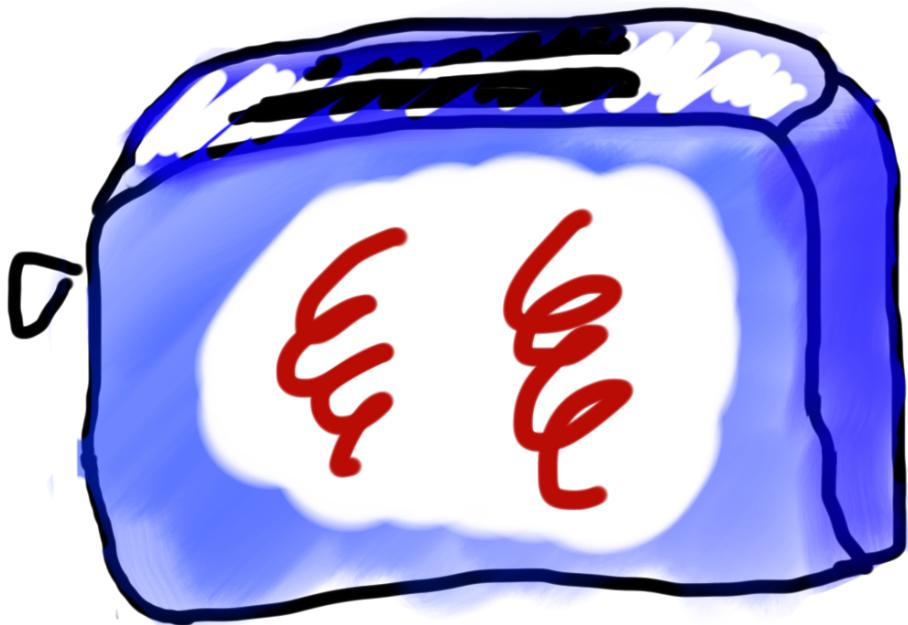
functions are Like Toasters



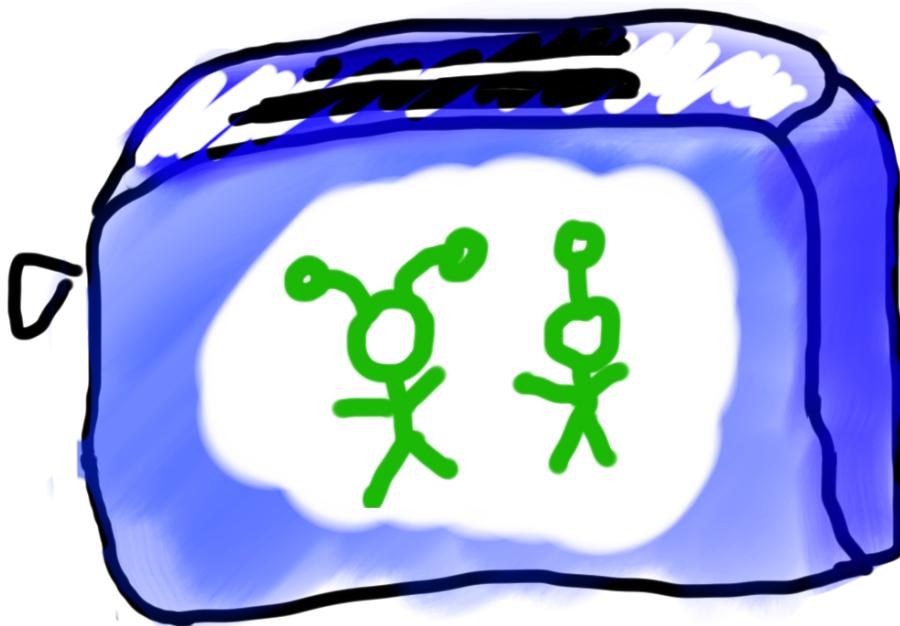
functions are Like Toasters



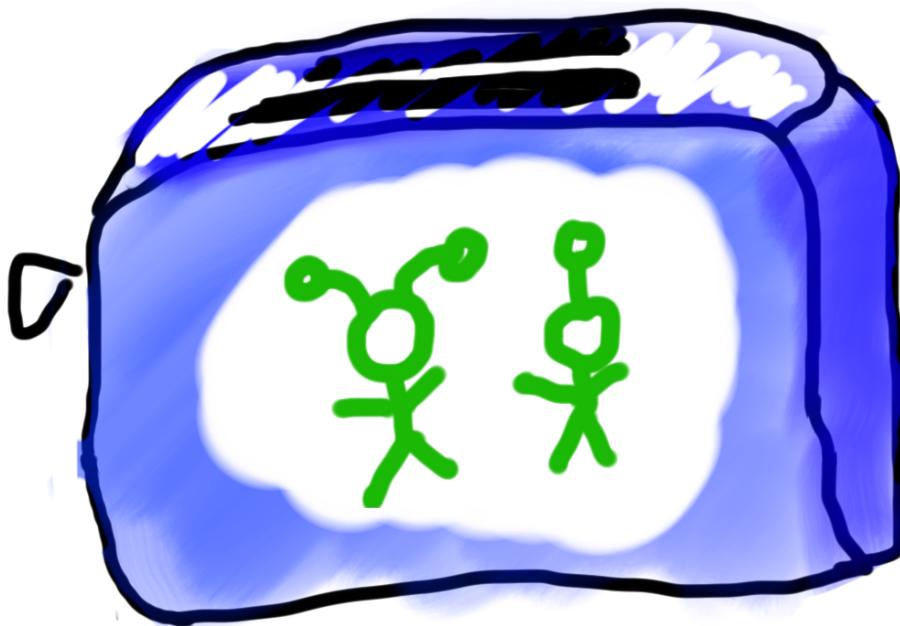
functions are Like Toasters



functions are Like Toasters



functions are Like Toasters



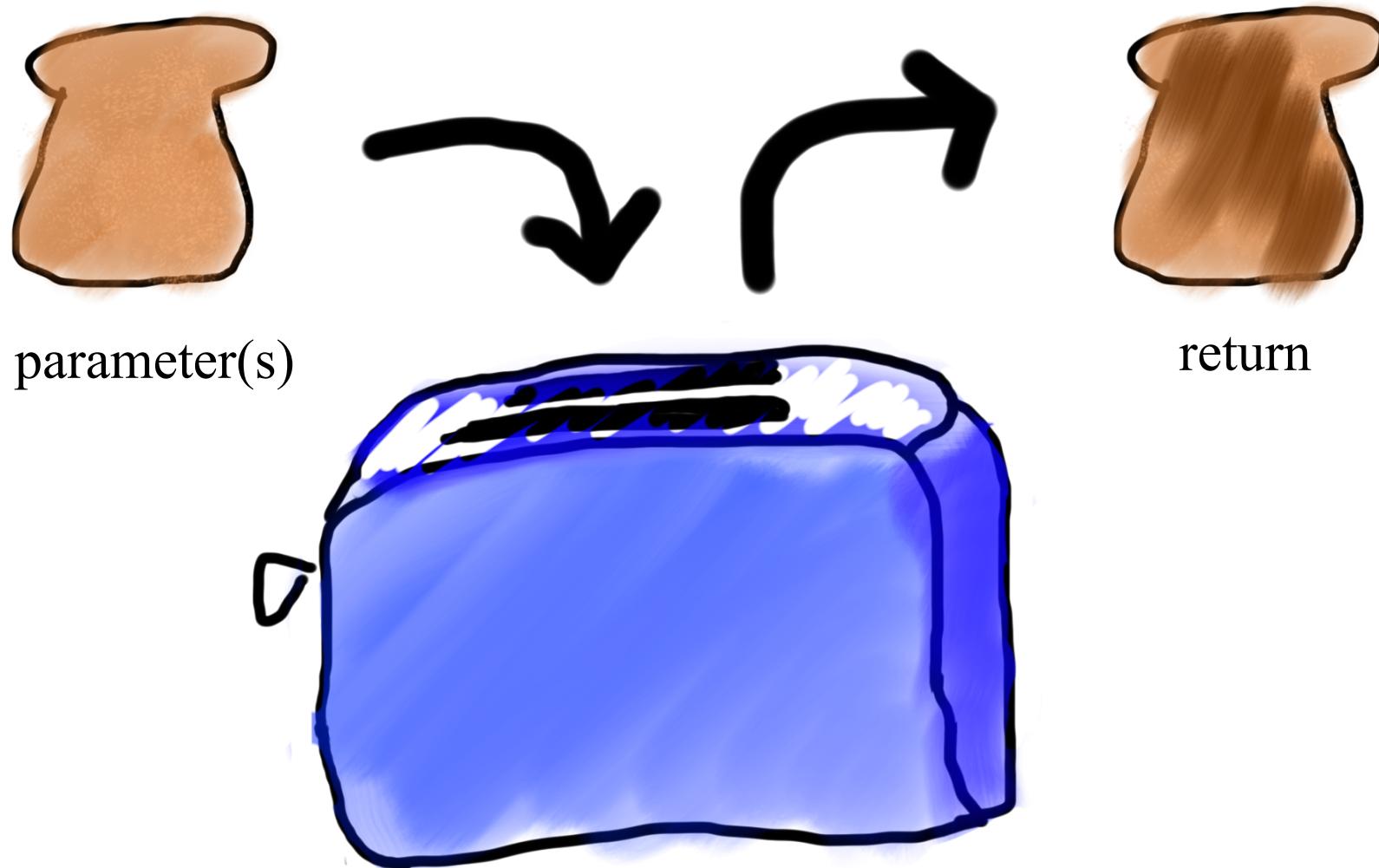
functions are Like Toasters



Piech + Sahami, CS106A, Stanford University



functions are Like Toasters



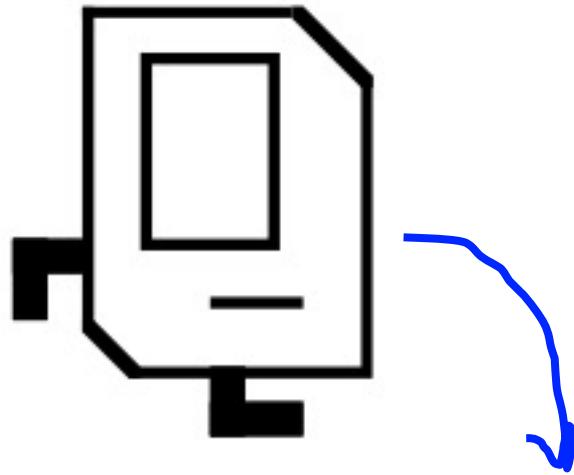
Classic Challenge for CS106A



Perhaps the
most
underrated
concept by
students



Historical Aside

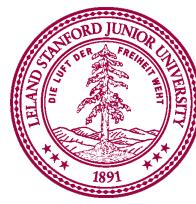


Defining a function

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



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



Intuition

Example 1

```
def woot():
    for i in range(3):
        print('woot')
```

```
def main():
    woot()
```

Example 2

```
def woot(num):
    for i in range(num):
        print('woot')
```

```
def main():
    woot(10)
```

Big difference with python functions:

Python functions can **take in data**, and can **return data!**



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



Anatomy of a function

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

function caller



function "definition"

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

function author



Anatomy of a function

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

function caller



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

function author



Anatomy of a function

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

information given

function caller



information expected

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

function author



Anatomy of a function

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

Arguments

function caller



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

Parameters

function author



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
```

body

function author



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 information)

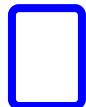
```
def main():
    mid = average(5.0, 10.2)
    print(mid)
```

```
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 at any point  
return value
```

- **name:** how you refer to the 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

Function
Caller (Coder)

Function
Author (Coder)



Uses the
terminal
(later UI)

Uses helper
functions

Writes helper
functions
others can
use

User

Function Caller (Coder)

Function Author (Coder)



Uses the terminal
(later UI)

Uses helper functions

Writes helper functions others can use

Learn by Example



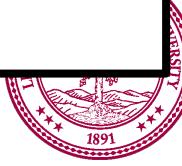
No Parameter, No Return

```
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
```



user



No Parameter, No Return



function author

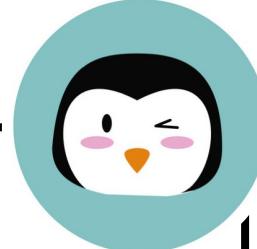
```
def print_intro():
    print("Welcome to class")
    print("It's the best part of my day.")
```

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

caller

terminal

```
> python intro.py
```



user



No Parameter, No Return



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



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



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



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
Welcome to class
```



user



No Parameter, No Return



function author

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

user

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

terminal

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



caller



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
Welcome to class
It's the best part of my day
```



user

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
Welcome to class
It's the best part of my day
```



user

No Parameter, No Return



function author

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



caller

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

terminal

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



user

Parameter Example

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



```
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

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

No variables

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

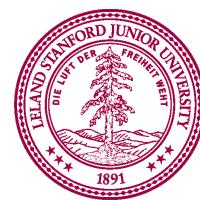
terminal

No variables

> 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

No variables

print_opinion memory

num

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

No variables

print_opinion memory

num 5

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

No variables

print_opinion memory

num 5

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

No variables

print_opinion memory

num 5

terminal

> python opinion.py
I love 5!

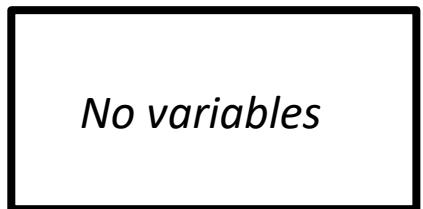
```
def print_opinion(num) :  
    if(num == 5) :  
        print("I love 5!")  
    else :  
        print("Whatever")
```

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

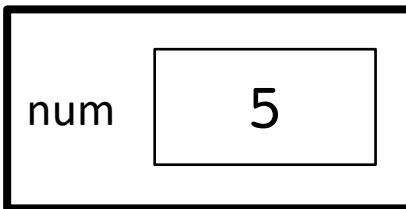


Parameter Example

main memory



print_opinion memory



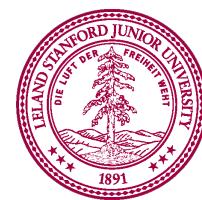
terminal

```
> 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

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

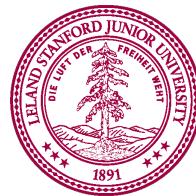
No variables

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 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

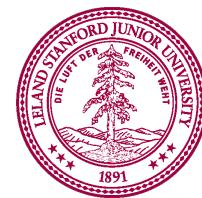
No variables

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

No variables

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

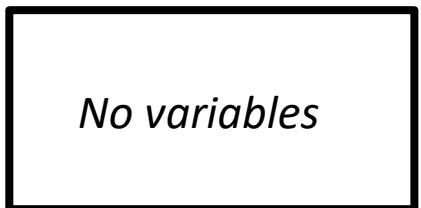
```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

> 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

No variables

metersToCm memory

meters 5 . 2

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)
```

terminal

```
> python m2cm.py
```



Parameter and Return Example

main memory

result **520.0**

terminal

> python m2cm.py

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

```
def main():      520.0  
    result = meters_to_cm(5.2)  
    print(result)
```



Parameter and Return Example

main memory

```
result 520.0
```



terminal

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

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

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



Parameter and Return Example

main memory

```
result 520.0
```



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

```
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))      520.0
    print(meters_to_cm(9.1))
```

terminal

```
> python m2cm.py
```



Parameter and Return Example

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

```
def main():      520.0  
    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))
```

terminal

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



Parameter and Return Example

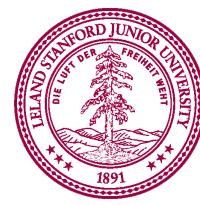
```
def meters_to_cm(meters):
    return 100 * meters

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)
```



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

return

print



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



User receives value on the console

Is returning
the same as printing?

Is returning
the same as printing?

NO

Is “input”
the same as parameters?

Is “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)
```

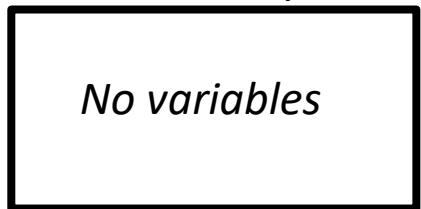
terminal

```
> python maxmax.py
```

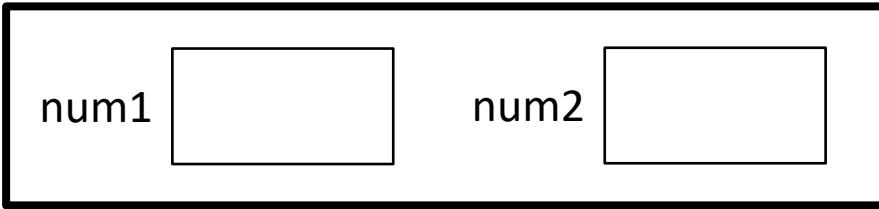


Multiple Return Statements

main memory



max memory

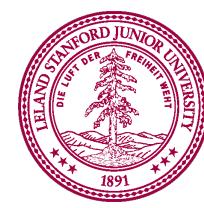


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

terminal

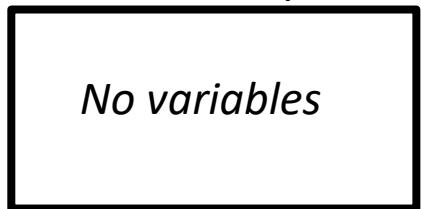
```
> python maxmax.py
```

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

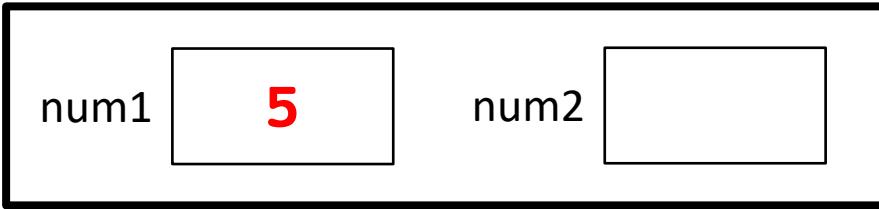


Multiple Return Statements

main memory



max memory



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

terminal

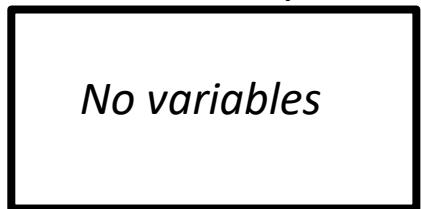
```
> python maxmax.py
```

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

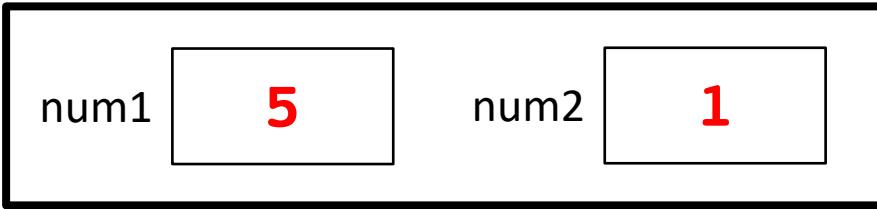


Multiple Return Statements

main memory



max memory

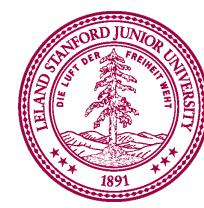


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

terminal

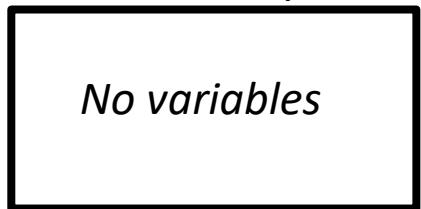
```
> python maxmax.py
```

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

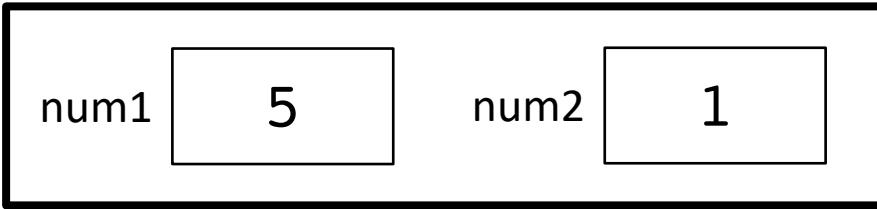


Multiple Return Statements

main memory



max memory



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

```
    return num2
```

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

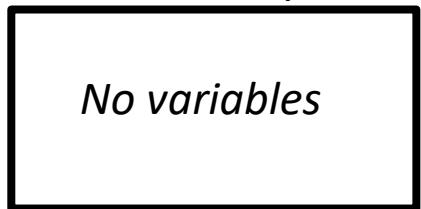
terminal

```
> python maxmax.py
```

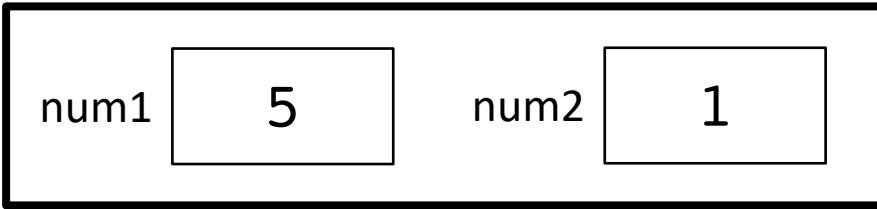


Multiple Return Statements

main memory



max memory



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

```
return num2
```

terminal

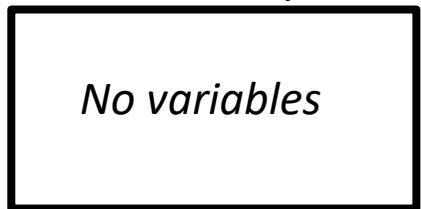
```
> python maxmax.py
```

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

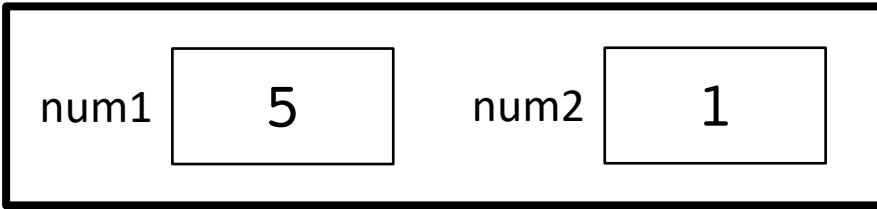


Multiple Return Statements

main memory



max memory

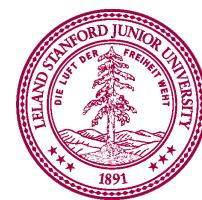


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
    5  
  
return num2
```

terminal

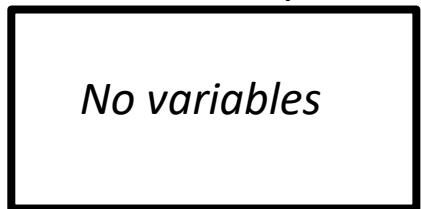
```
> python maxmax.py
```

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

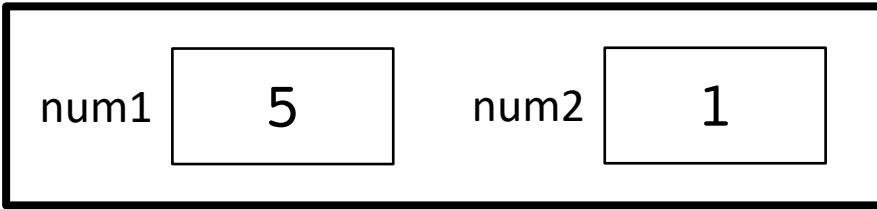


Multiple Return Statements

main memory



max memory



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

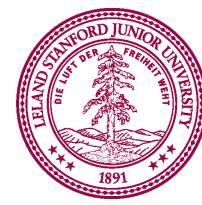
```
return num2
```

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

5

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

larger

5

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

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

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

larger

5

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

terminal

```
> python maxmax.py
```

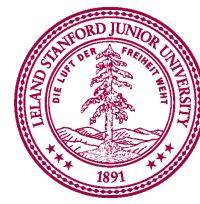
```
def main():  
    larger = max(5, 1)
```



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

No variables

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

```
    return num2
```

def main():

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



Multiple Return Statements

main memory

No variables

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

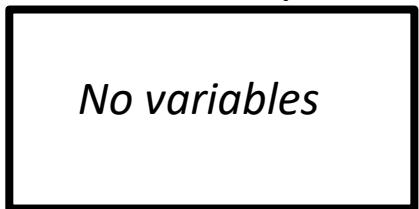
```
    return num2
```

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

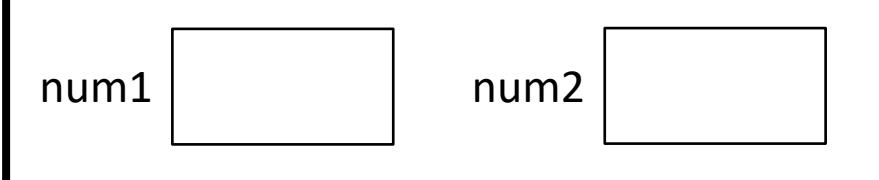


Multiple Return Statements

main memory



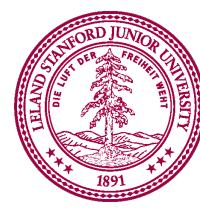
max memory



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

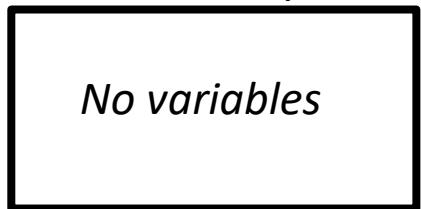
```
    return num2
```

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

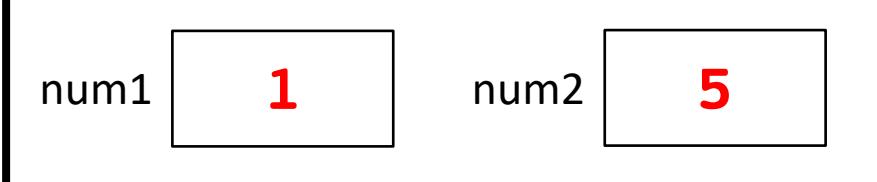


Multiple Return Statements

main memory



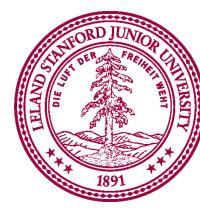
max memory



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

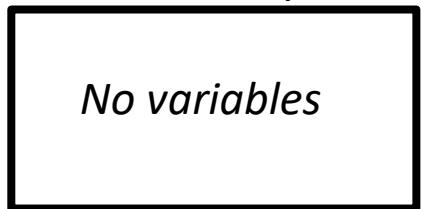
```
    return num2
```

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

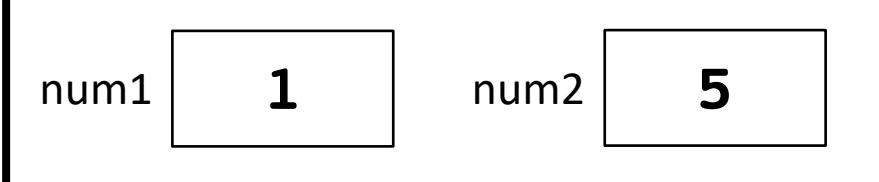


Multiple Return Statements

main memory



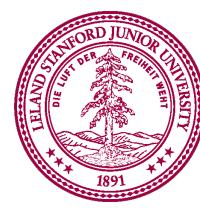
max memory



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

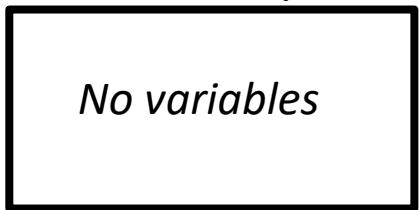
```
    return num2
```

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

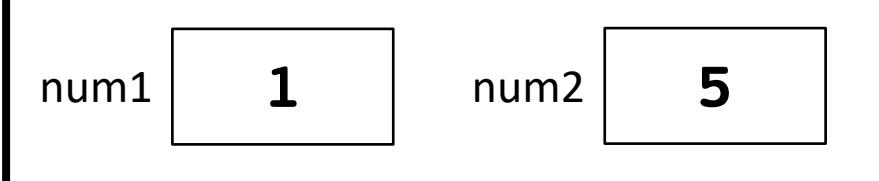


Multiple Return Statements

main memory



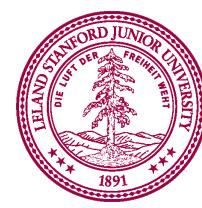
max memory



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

```
return num2 5
```

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



Multiple Return Statements

main memory

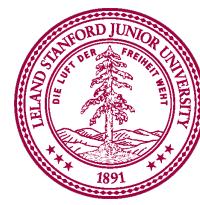
No variables

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

```
    return num2
```

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

5



Multiple Return Statements

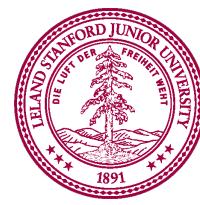
main memory



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

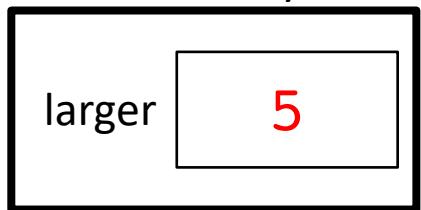
```
    return num2
```

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



Fix this program

The screenshot shows a web-based code editor interface. The title bar says "ed (101) Code in Place 2021 – Lessons". The main area displays a Python script named "gameshow.py". The code is as follows:

```
1 def main():
2     print("Welcome to the CodeInPlace Game Show")
3     print("Pick a door and win a prize")
4     print("-----")
5
6     # Part 1: get the door number from the user
7     door = int(input("Door: "))
8     # while the input is invalid
9     while door < 1 or door > 3:
10        # tell the user the input was invalid
11        print("Invalid door!")
12        # ask for a new input
13        door = int(input("Door: "))
14
15    # Part 2: compute the prize
16    prize = 4
17    if door == 1:
18        prize = 2 + 9 // 10 * 100
19    elif door == 2:
20        locked = prize % 2 != 0
21        if not locked:
22            prize += 6
23    elif door == 3:
24        for i in range(door):
25            prize += i
26
27    # Part 3: report the prize
28    print('You win ' + str(prize) + ' treats')
29
30 if __name__ == '__main__':
31     main()
32
```

At the bottom of the editor, it says "/home/gameshow.py 15:30 Spaces: 4 (Auto)" and "All changes saved". There are buttons for "Console", "Terminal", "Run", and "Submit".

