Functions

CS Bridge 2020

Learn How To:

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

Lecture Plan

- Functions: From Karel to Python
- **Example**: Factorial
- More About Functions
- **Example**: Graphics

Lecture Plan

- **Functions: From Karel to Python**
- **Example**: Factorial
- More About Functions
- **Example**: Graphics

Calling functions

```
turn_right()
```

```
move()
```

```
input("string please! ")
```

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

```
float("0.42")
```

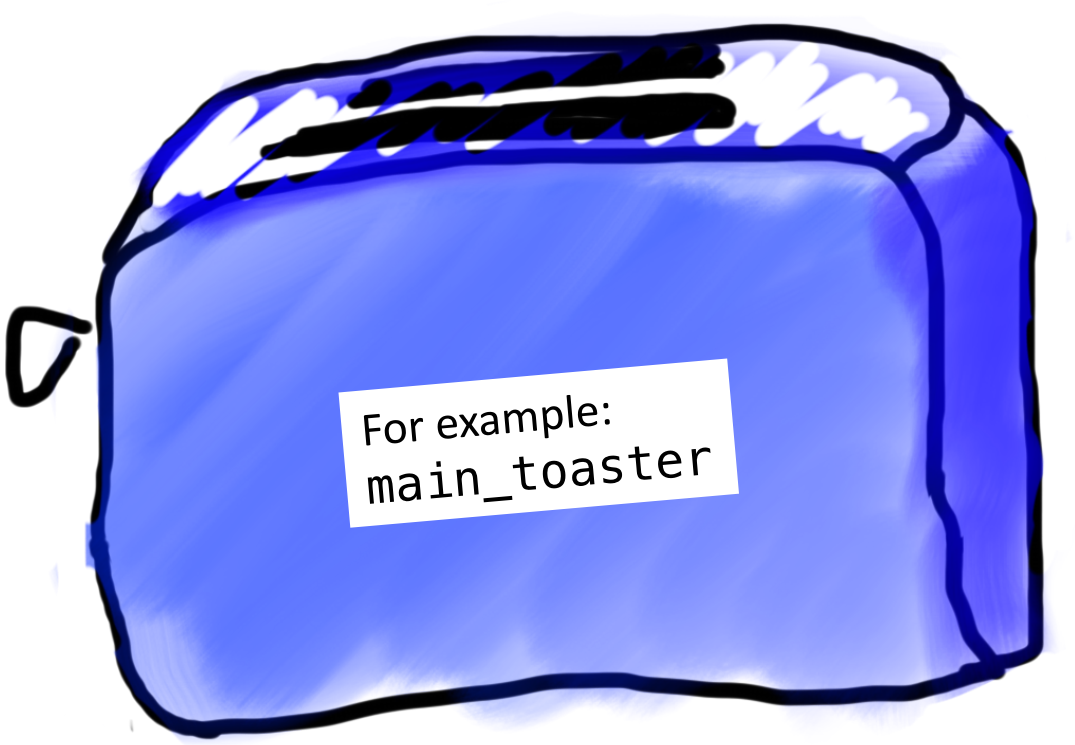
```
math.sqrt(25)
```


Defining a function

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

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

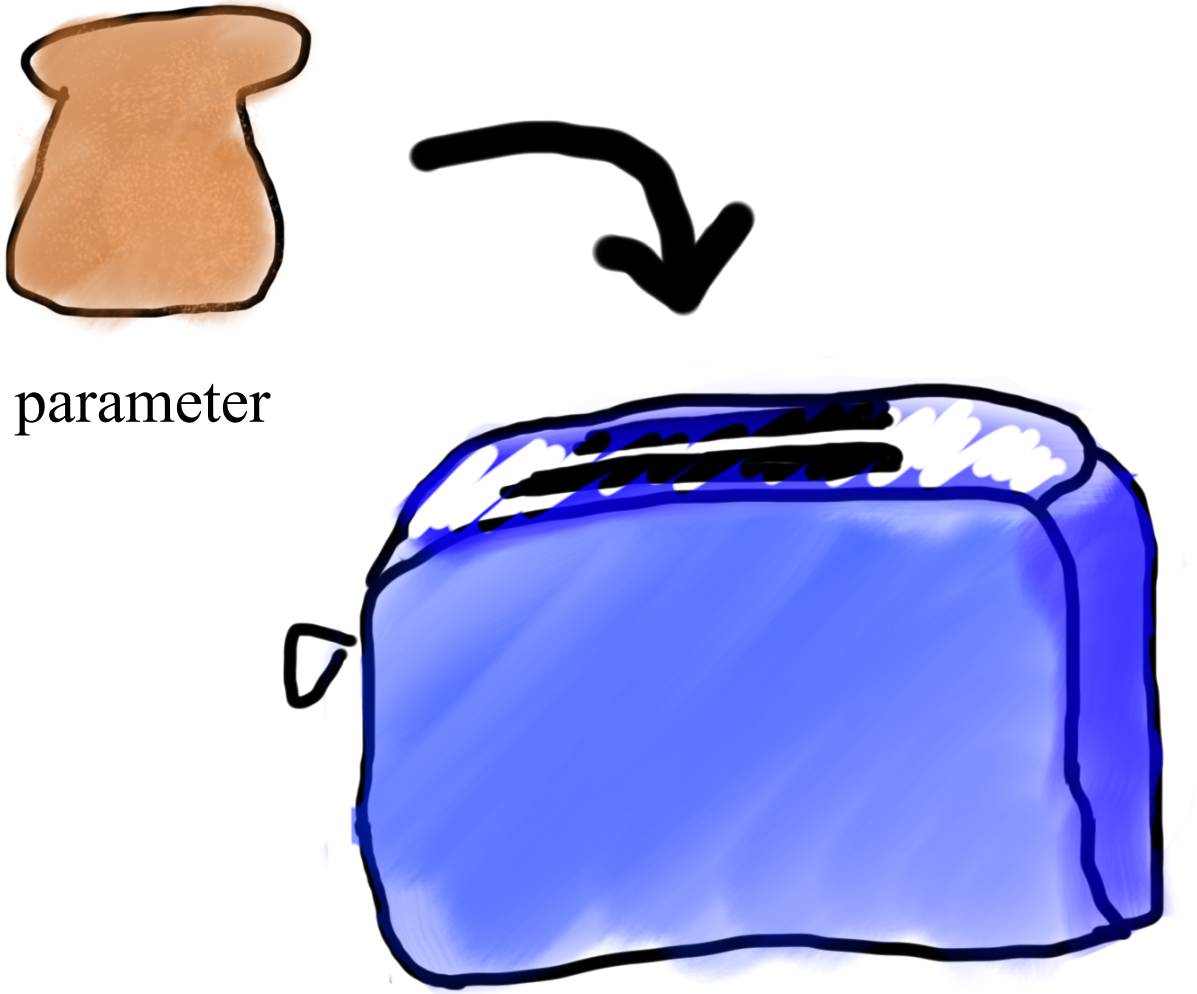
Toasters are functions



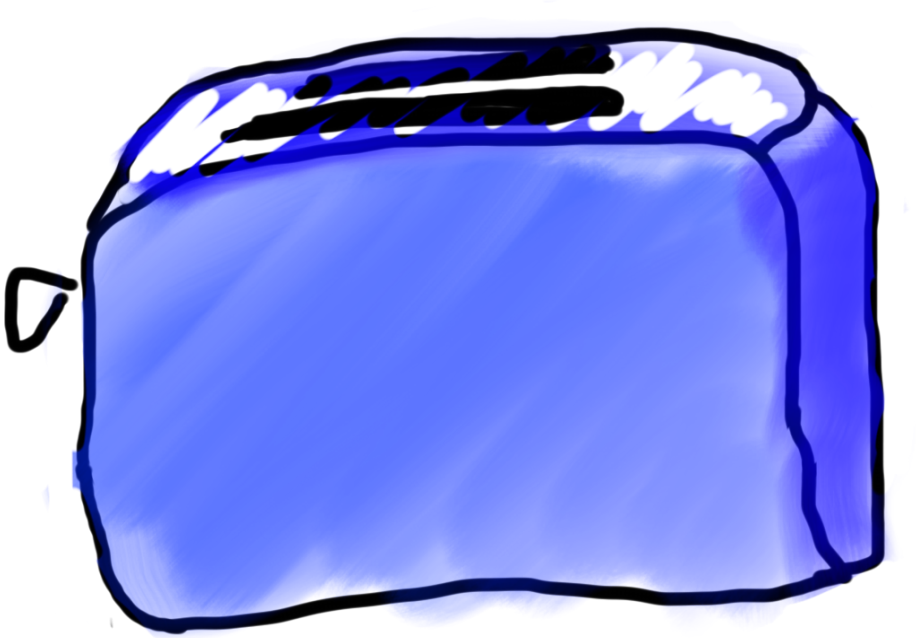
Toasters are functions

parameter

Toasters are functions

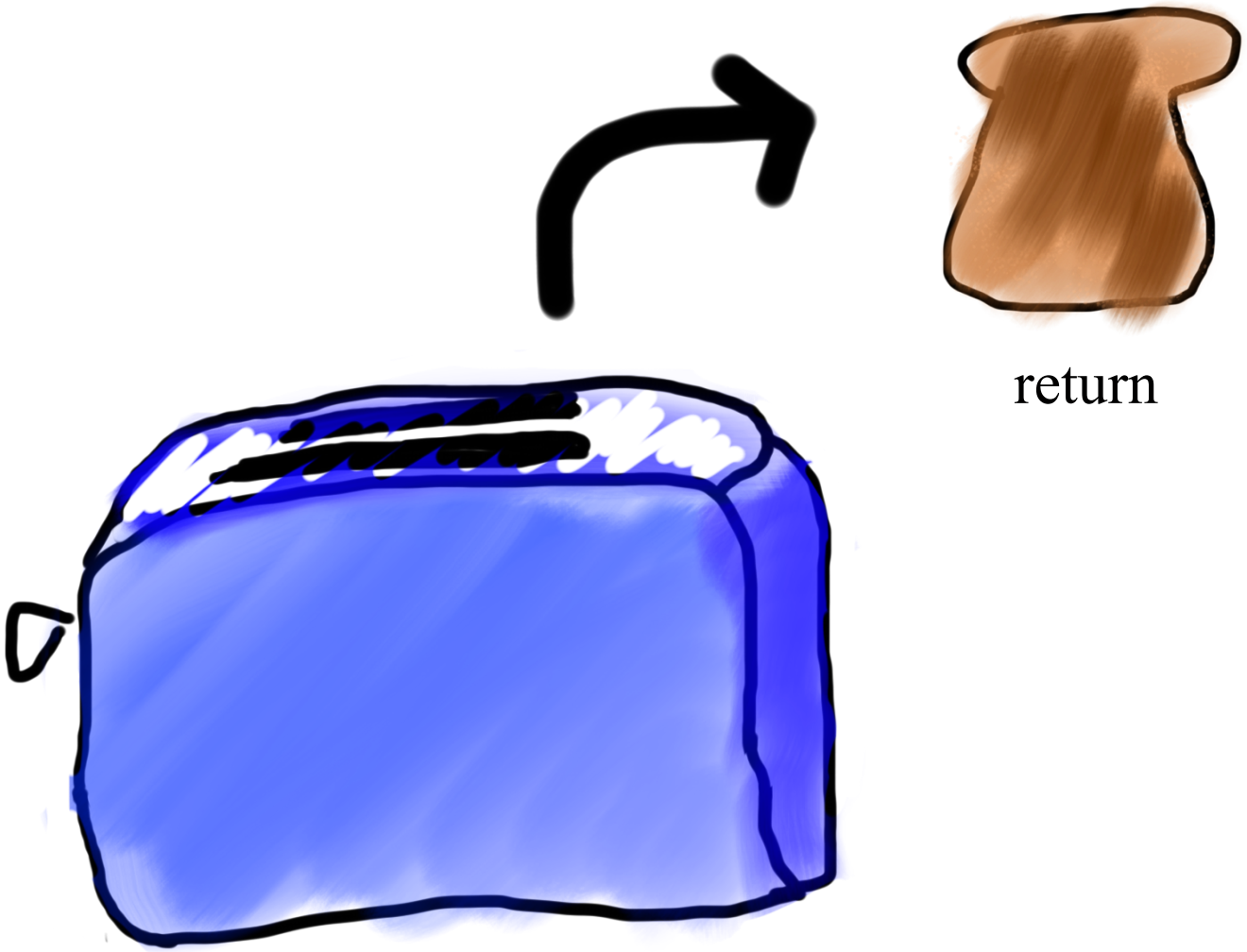


Toasters are functions

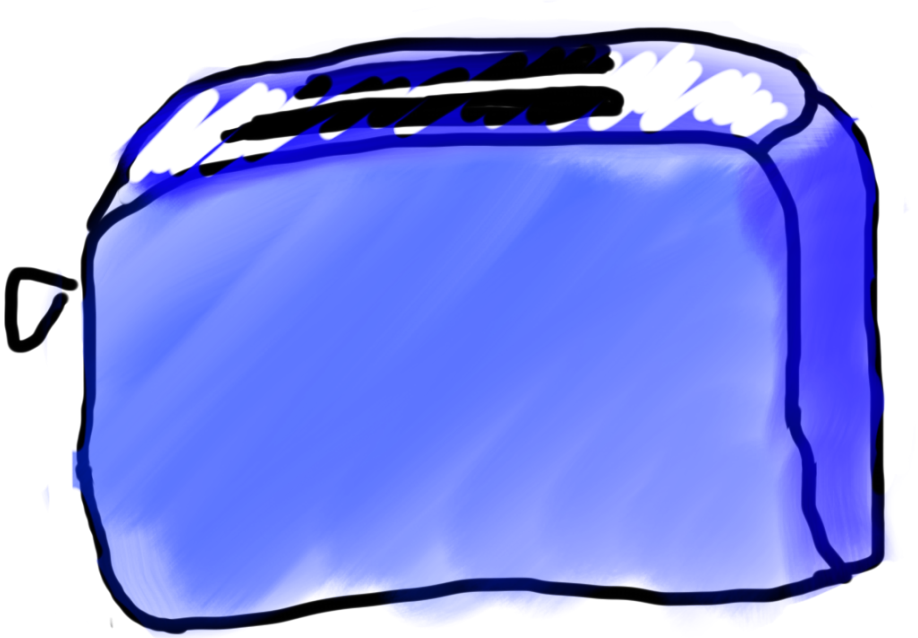


Toasters are functions

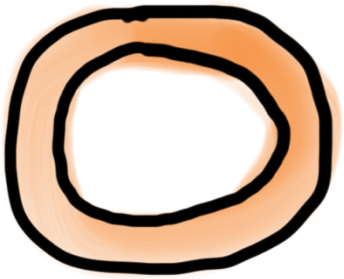
Toasters are functions



Toasters are functions

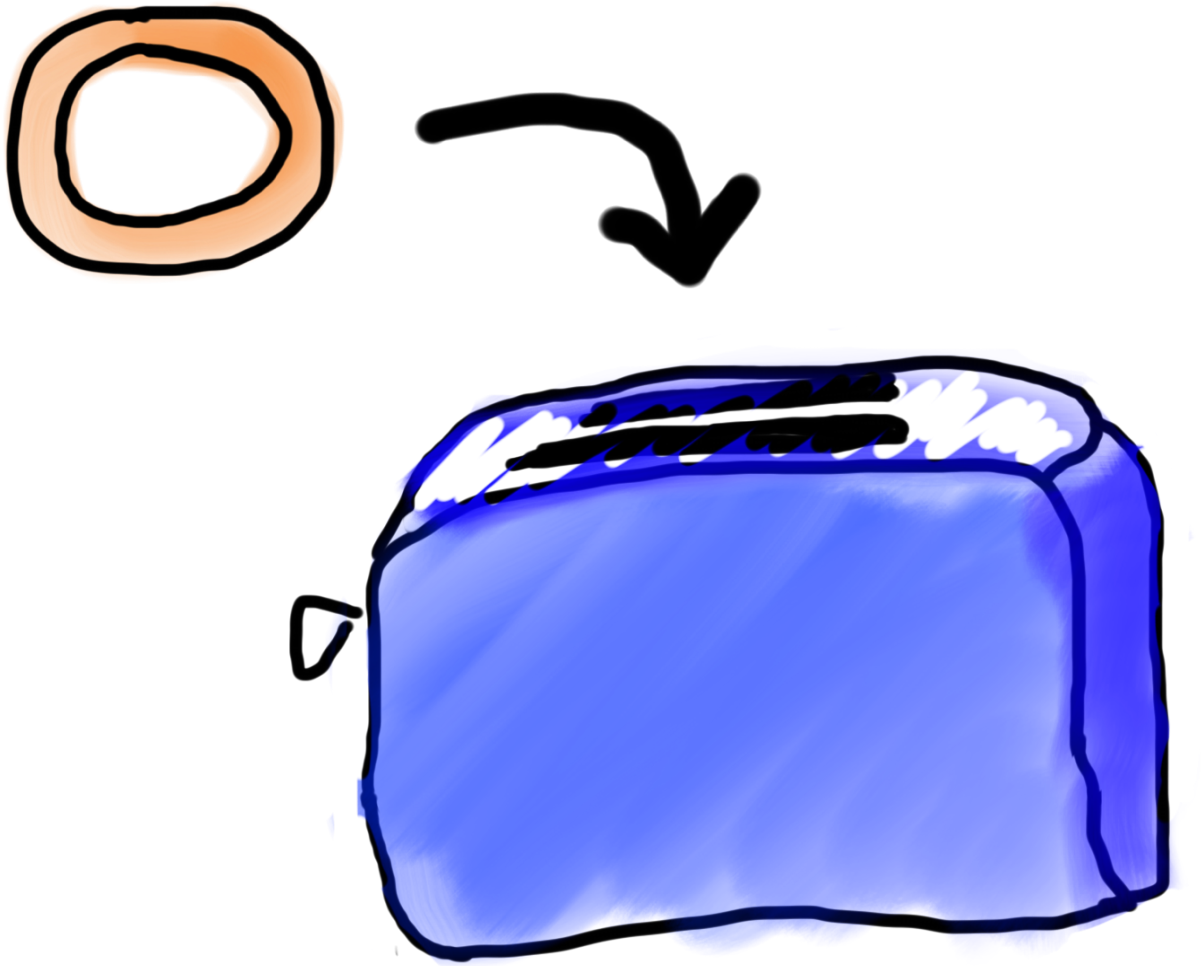


Toasters are functions

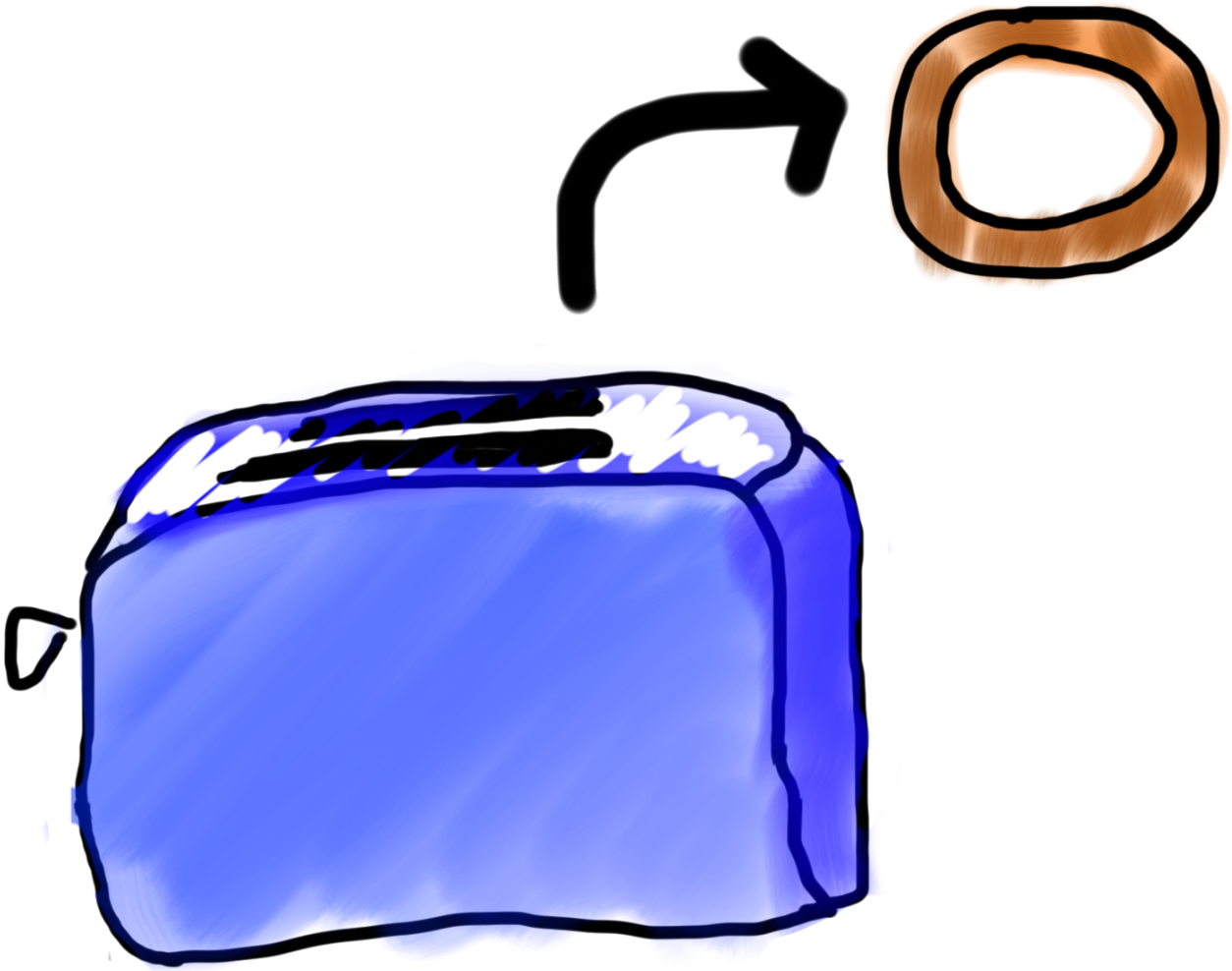


* You don’t need a second toaster if you want to toast bagels. Use the same one.

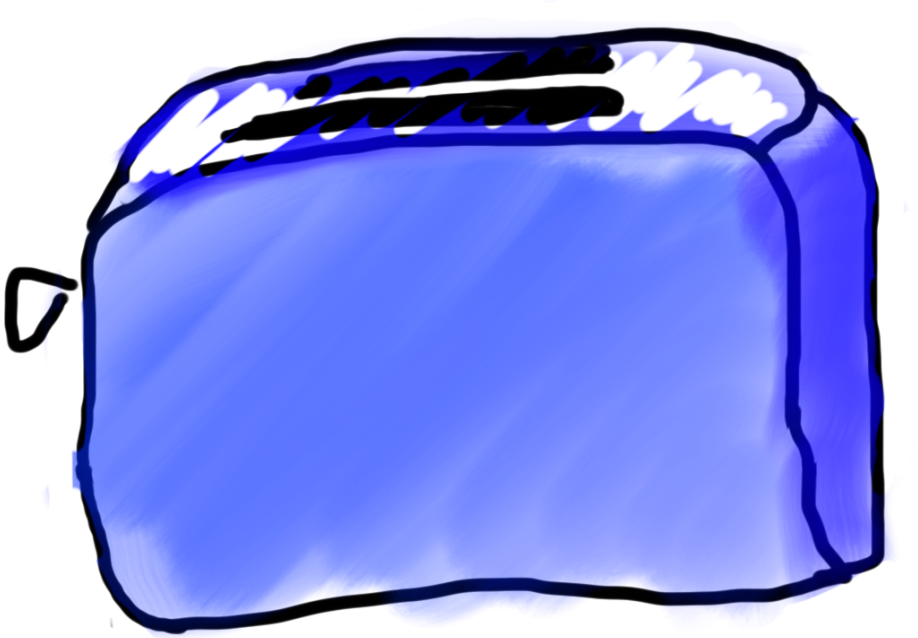
Toasters are functions



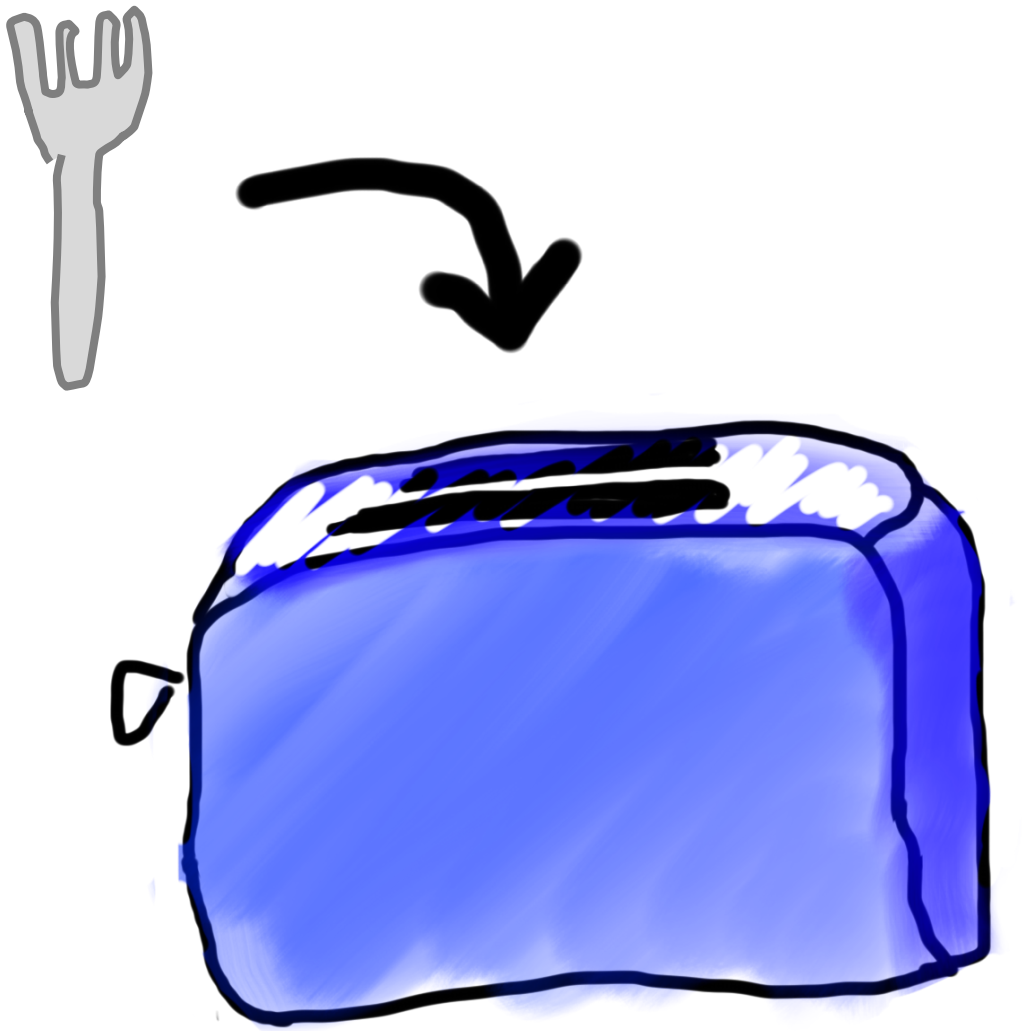
Toasters are functions



Toasters are functions



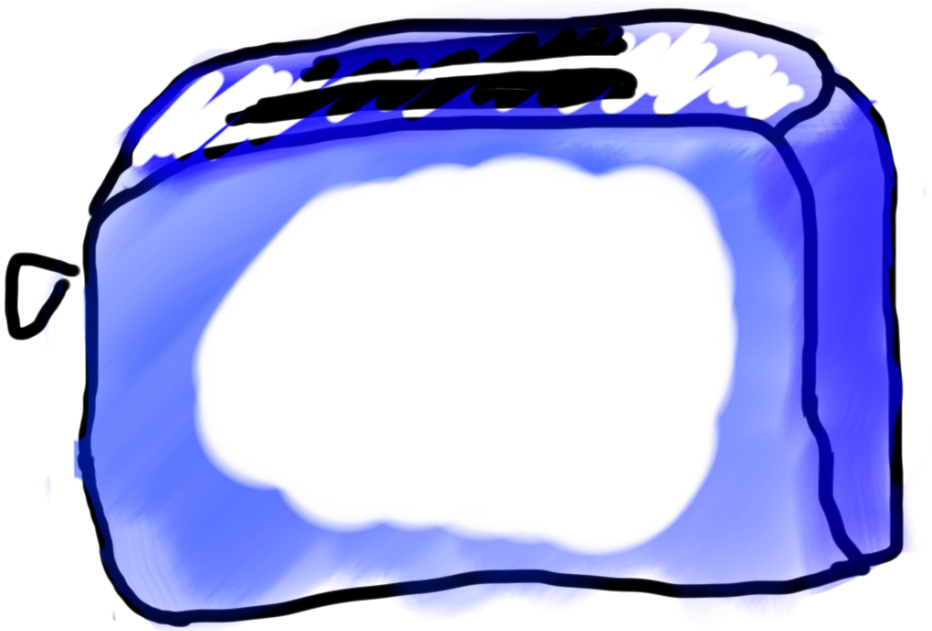
Toasters are functions



Toasters are functions

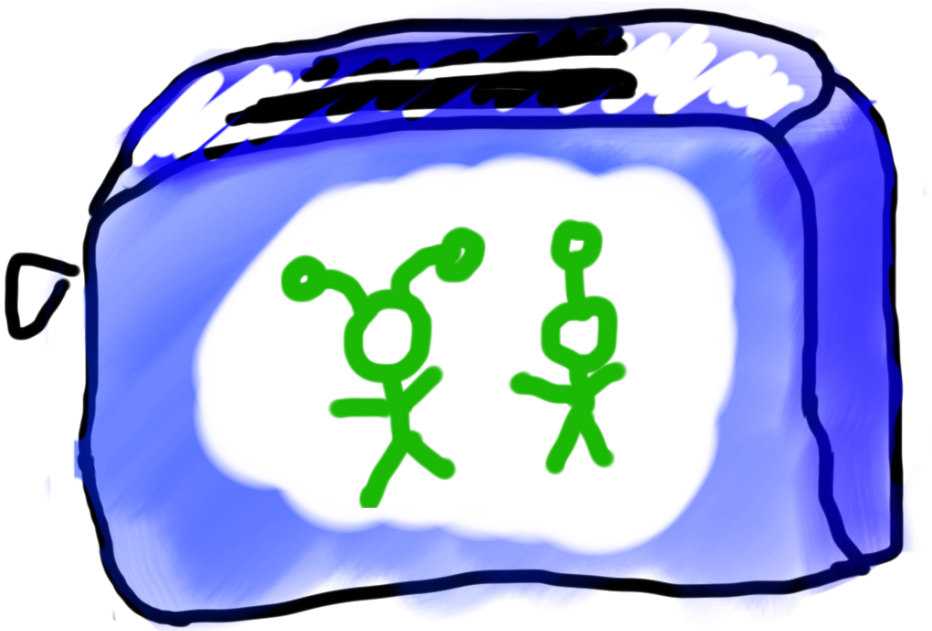
functions are Like Toasters

functions are Like Toasters

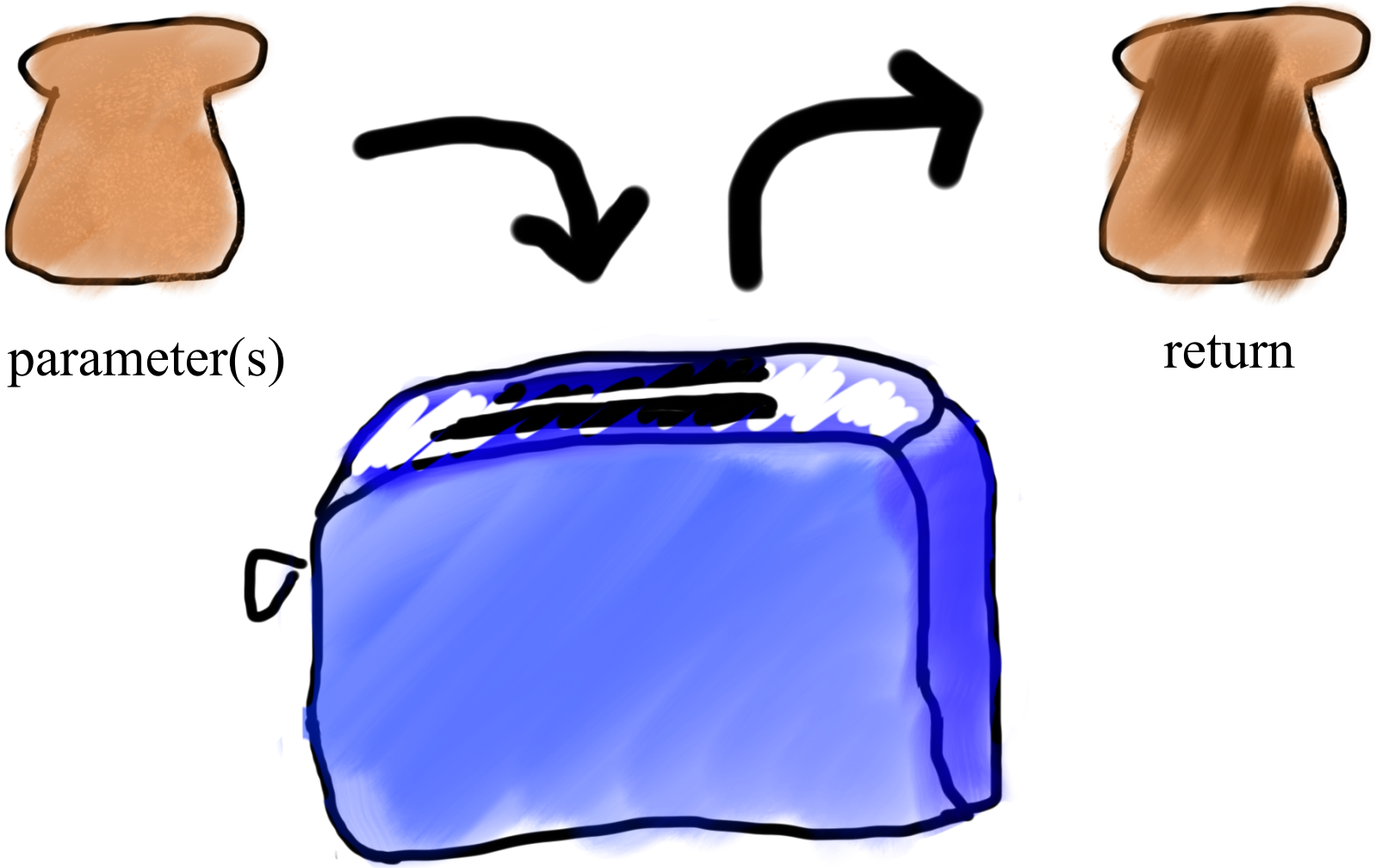


functions are Like Toasters

functions are Like Toasters



functions are Like Toasters



Formally

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

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

Classic Challenge

Perhaps the most underrated concept by students

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

function "call"

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)
```

name

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

Anatomy of a function

Input given

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

Input expected

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


Anatomy of a function

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

Arguments

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

Parameters

Anatomy of a function

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

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

body

Anatomy of a function

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

This call "evaluates" to the value returned

```
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 definition

```
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
```

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

Learn by Example

Lecture Plan

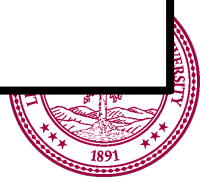
- Functions: From Karel to Python
  - **no parameter, no return**
  - parameter
  - parameter and return
- **Example**: Factorial
- More About Functions
- **Example**: Graphics

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

```
> python3 intro.py
```



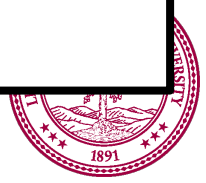
No Parameter, No Return

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

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

terminal

```
> python3 intro.py
```



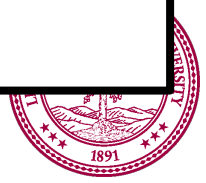
No Parameter, No Return

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

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

terminal

```
> python3 intro.py
```

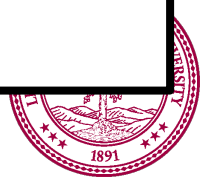


No Parameter, No Return

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


def main():
    print_intro()
```

terminal

```
> python3 intro.py
```

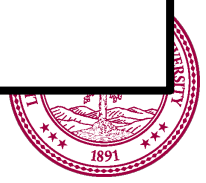


No Parameter, No Return

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


def main():
    print_intro()
```

terminal

```
> python3 intro.py
Welcome to class
```



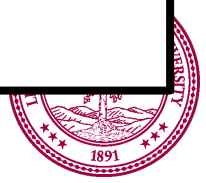
No Parameter, No Return

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

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

terminal

```
> python3 intro.py
Welcome to class
It’s the best part of my day
```

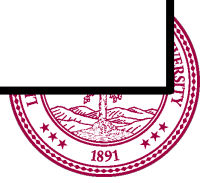


No Parameter, No Return

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


def main():
    print_intro()
```

terminal

```
> python3 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

```
> python3 intro.py
Welcome to class
It’s the best part of my day
```

Lecture Plan

- Functions: From Karel to Python
  - no parameter, no return
  - **parameter**
  - parameter and return
- **Example**: Factorial
- More About Functions
- **Example**: Graphics

Parameter Example

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


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

terminal

```
> python3 opinion.py
```

Parameter Example

main memory

No variables

terminal

> python3 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

> python3 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

terminal

```
> python3 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

```
> python3 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

> python3 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

> python3 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

```
> python3 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

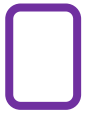
print_opinion memory

num 5

terminal

```
> python3 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

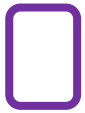
main memory

No variables

terminal

```
> python3 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

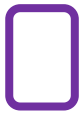
main memory

No variables

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


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

terminal

```
> python3 opinion.py
I love 5!
```



Lecture Plan

- Functions: From Karel to Python
  - no parameter, no return
  - parameter
  - **parameter and return**
- **Example**: Factorial
- More About Functions
- **Example**: Graphics

Parameter and Return Example

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


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

terminal

```
> python3 m2cm.py
```

Parameter and Return Example

main memory

No variables

terminal

```
> python3 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

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


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

terminal

```
> python3 m2cm.py
```

Parameter and Return Example

main memory

No variables

meters_to_cm memory

terminal

```
> python3 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

meters_to_cm memory

meters 5.2

terminal

```
> python3 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

meters_to_cm memory

meters | 5.2

terminal

> python3 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

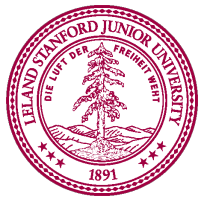
No variables

terminal

> python3 m2cm.py

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


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

520.0



Parameter and Return Example

main memory

terminal

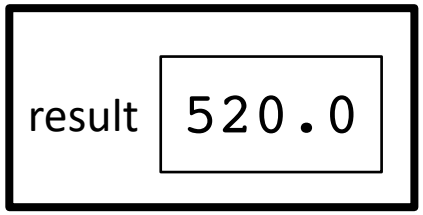
```
> python3 m2cm.py
```

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



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

520.0

Parameter and Return Example

main memory



terminal

```
> python3 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

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


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

terminal

```
> python3 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

```
> python3 m2cm.py
```

If a method returns something, you can use it directly in an expression!

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

```
> python3 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

```
> python3 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

```
> python3 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

```
> python3 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

```
> python3 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

```
> python3 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

```
> python3 m2cm.py
520.0
910.0
```

Contrasting Case:

```
# How is this function
def meters_to_cm_case1(meters):
    return 100 * meters
```

```
# Different than this function?
def meters_to_cm_case2(meters):
    print(100 * meters)
```


Is returning
the same as printing?

Is returning
the same as printing?

NO

Multiple Return Statements

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

    return num2


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

terminal

```
> python3 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

```
> python3 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

```
> python3 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

```
> python3 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

```
> python3 maxmax.py
```


Multiple Return Statements

main memory

No variables

max memory

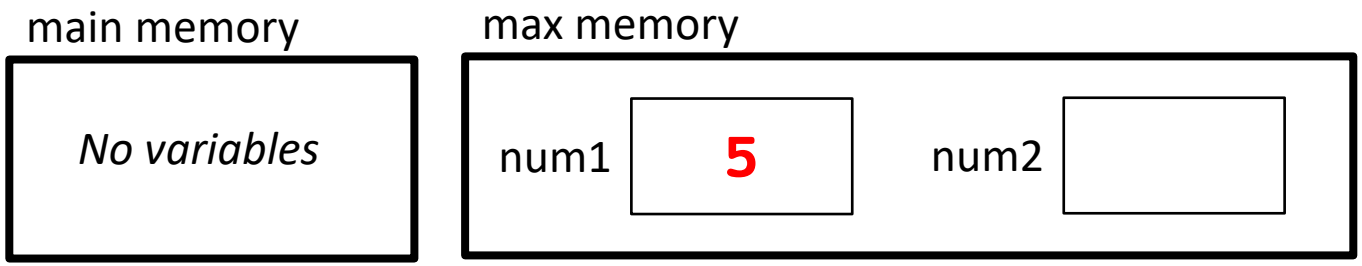
num1 num2

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

    return num2



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

terminal

```
> python3 maxmax.py
```

Multiple Return Statements




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

    return num2



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

terminal

```
> python3 maxmax.py
```

Multiple Return Statements

main memory

No variables

max memory

num1 **5**

num2 **1**

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

    return num2



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

terminal

```
> python3 maxmax.py
```

Multiple Return Statements

main memory

No variables

max memory

num1 | 5 | num2 | 1

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

    return num2


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

terminal

```
> python3 maxmax.py
```

Multiple Return Statements

main memory

No variables

max memory

num1 | 5 | num2 | 1

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

    return num2



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

terminal

```
> python3 maxmax.py
```

Multiple Return Statements

main memory

No variables

max memory

num1 | 5 | num2 | 1

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

    return num2


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

terminal

```
> python3 maxmax.py
```

Multiple Return Statements

main memory

No variables

max memory

num1 | 5 | num2 | 1

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

    return num2


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

5

terminal

```
> python3 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)
```

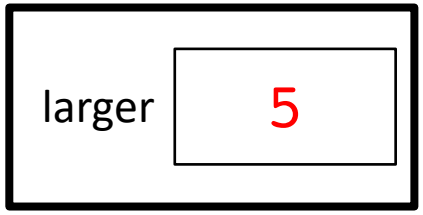
5

terminal

```
> python3 maxmax.py
```

Multiple Return Statements

main memory




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

    return num2



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

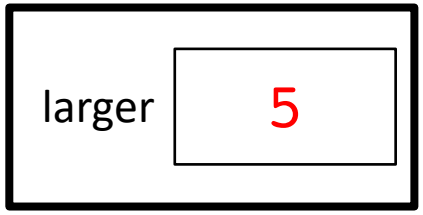
5

terminal

```
> python3 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

```
> python3 maxmax.py
```

Multiple Return Statements

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

    return num2


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


Multiple Return Statements

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

    return num2


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


Multiple Return Statements

main memory

No variables

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

    return num2
```

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

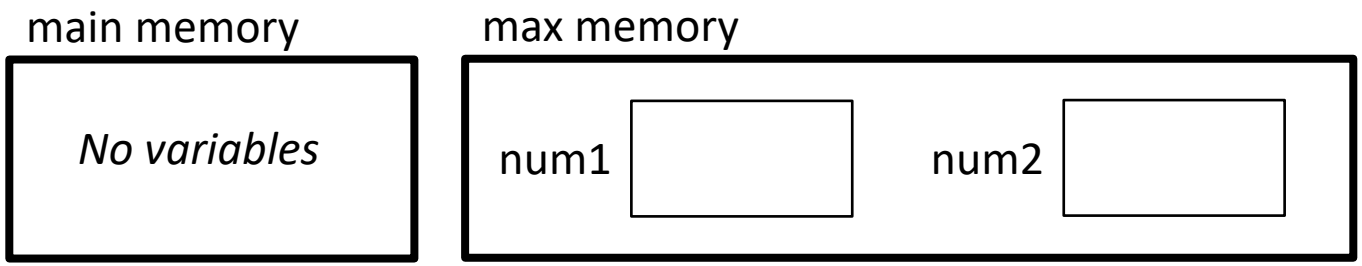

Multiple Return Statements

main memory

No variables

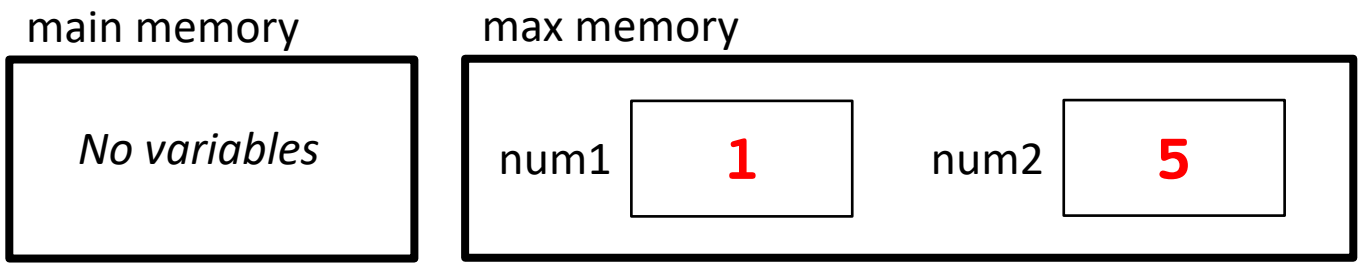
```
def max(num1, num2):
    if num1 >= num2:
        return num1

    return num2



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


Multiple Return Statements



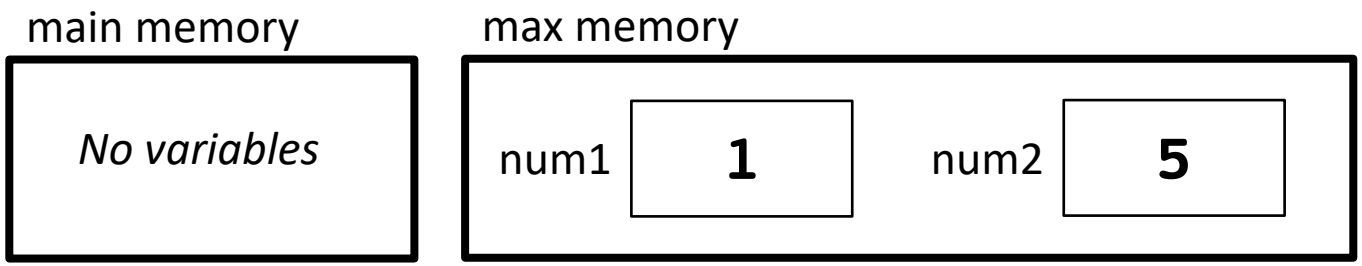

```
def max(num1, num2):
    if num1 >= num2:
        return num1

    return num2



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

Multiple Return Statements



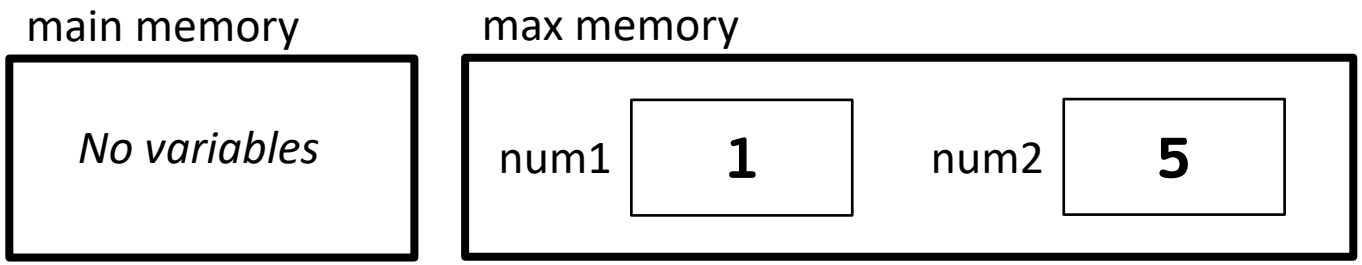

```
def max(num1, num2):
    if num1 >= num2:
        return num1

    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

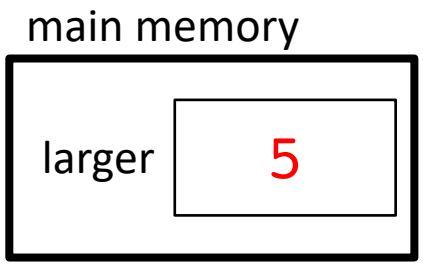
main memory

No variables

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

    return num2


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

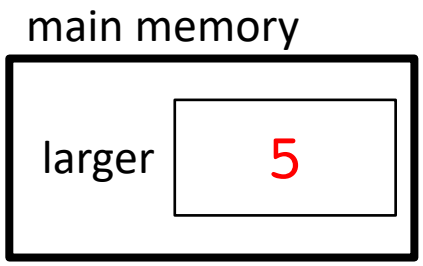
5

Multiple Return Statements




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

    return num2



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

Multiple Return Statements



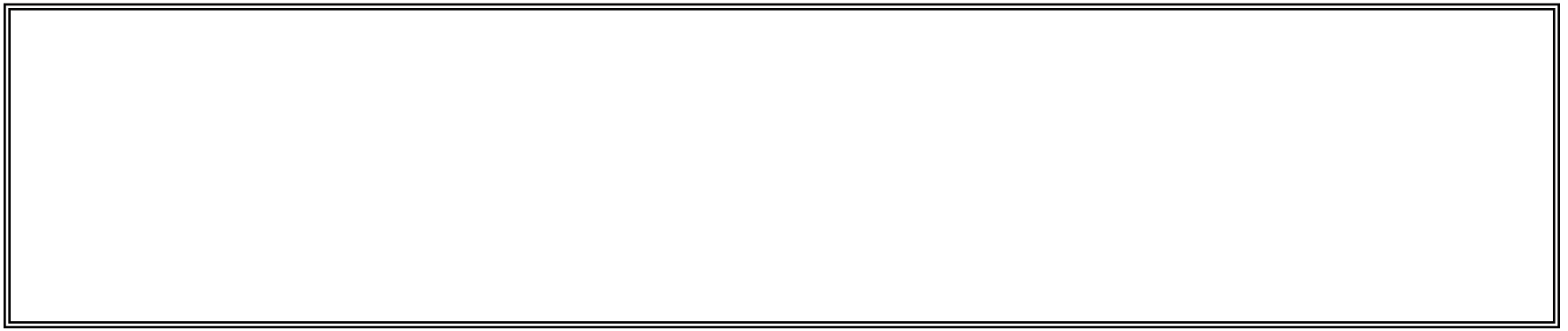

```
def max(num1, num2):
    if num1 >= num2:
        return num1

    return num2



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

Lecture Plan

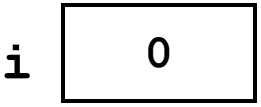
- Functions: From Karel to Python
- **Example: Factorial**
- More About Functions
- **Example**: Graphics

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

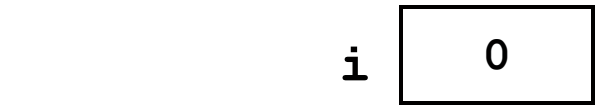
i 



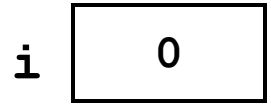
```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```



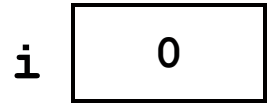
```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```



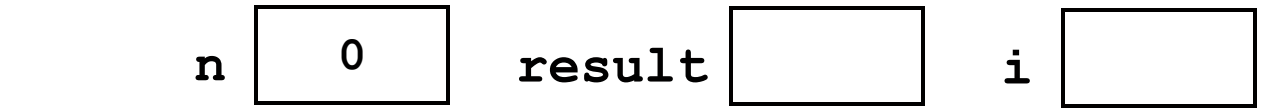
```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```



```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```



```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```



```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```

n | 0 | result | 1 | i |

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```

n	0	result	1	i	0

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```

n 0 result 1 i 0

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

1

i 0

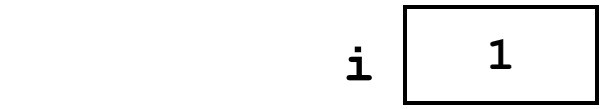
```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

1

i | 0

```
0   1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```



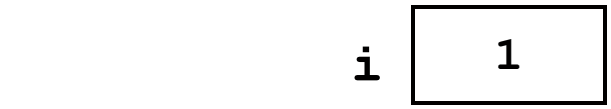

```
0    1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i | 1

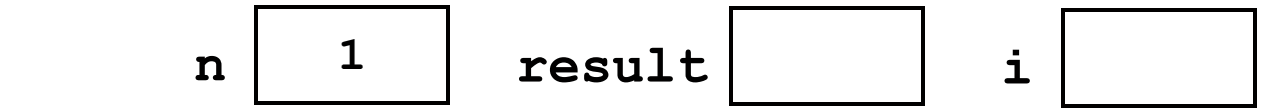
```
0    1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```




```
0    1
```

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```




```
0    1
```

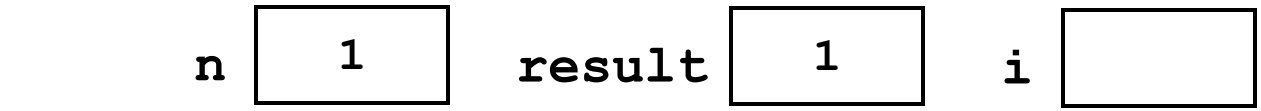
Parameters

Every time a function is called, new memory is created for that call.

Parameter values are passed in.

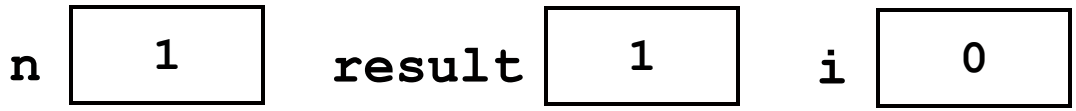
All *local* variables start fresh (no old values)

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```



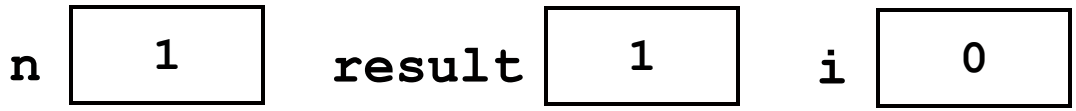

```
0    1
```

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```



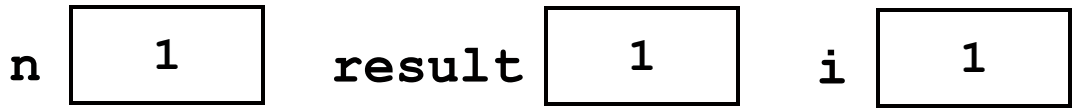

```
0    1
```

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```




```
0    1
```

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```




```
0    1
```

```
def factorial(n):
    result = 1
    for i in range(n):
        result *= (i + 1)

    return result
```

n 1 result 1 i 1

```
0    1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

1

i 1

```
0    1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

1

i 1

```
0     1
1     1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i | 2

```
0     1
1     1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i | 2

```
0     1
1     1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i | 2

```
0     1
1     1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i | 2

```
0     1
1     1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

2

i 2

```
0     1
1     1
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

2

i 2

```
0     1
1     1
2     2
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i 3

```
0    1
1    1
2    2
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i | 3

```
0    1
1    1
2    2
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i 3

```
0    1
1    1
2    2
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i 3

```
0    1
1    1
2    2
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

6

i 3

```
0    1
1    1
2    2
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

6

i 3

```
0     1
1     1
2     2
3     6
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i 4

```
0    1
1    1
2    2
3    6
```

```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

i | 4

```
0    1
1    1
2    2
3    6
```

Parameters

Every time a function is called, new memory is created for the call.

Lecture Plan

- Functions: From Karel to Python
- Example: Factorial
- **More About Functions**
- **Example**: Graphics

Bad Times With functions

```
// NOTE: This program is buggy!!

def add_five(x):
    x = x + 5



def main():
    x = 3
    add_five(x)
  print("x = " + x)
```

Bad Times With functions

If you change a parameter using =, the change does not persist outside the function!

Bad Times With functions

```
// NOTE: This program is buggy!!

def add_five(x):
    x = x + 5



def main():
    x = 3
    add_five(x)
  print("x = " + x)
```

Good Times With functions

```
// NOTE: This program is feeling just fine...

def add_five(x):
    x = x + 5
  return x


def main():
    x = 3
    x = add_five(x)
  print("x = " + x)
```


More Examples

Changed Name

```
def main():
    num = 5
    cow(num)


def cow(grass):
    print(grass)
```


Same Variable Name

```
def main():
    num = 5
    cow()
    print(num)


def cow():
    num = 10
    print(num)
```


No functions in functions

```
def main():
    print("hello world")
    def say_goodbye():
        print("goodbye!")
```

Technically legal, but often a sign at the start that you are confusing definition and calling

No functions in functions

```
def main():
    print("hello world")
    say_goodbye()


def say_goodbye():
    print("goodbye!")
```

Remember Booleans?

Boolean Variable

```
karel_is_awesome = True

my_bool = 1 < 2
```


Is Even

```
def main():
    for i in range(100):
        if is_even(i):
            print(i)
```


Boolean Return

```
def main():
    for i in range(100):
        if is_even(i):
            print(i)
```

```
def is_even(x):
    if x % 2 == 0:
        return True
    else:
        return False
```

Boolean Return

```
def main():
    for i in range(100):
        if is_even(i):
            print(i)
```

```
def is_even(x):
    return x % 2 == 0
```

Is Divisible By 7

```
def main():
    for i in range(100):
        if is_divisible_by(i, 7):
            print(i)
```


Boolean Return

```
def main():
    for i in range(100):
        if is_divisible_by(i, 7):
            print(i)
```

```
def is_divisible_by(num, divisor):
    if num % divisor == 0:
        return True
    else:
        return False
```

Boolean Return

```
def main():
    for i in range(100):
        if is_divisible_by(i, 7):
            print(i)



def is_divisible_by(num, divisor):
    return num % divisor == 0
```


Can a function return multiple values?

YES!

Multiple Return Values

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

    return num2
```

terminal

```
> python3 maxmax.py
5 is bigger.
```

```
def main():
    larger = max(5, 1)
    print(str(larger) + " is bigger.")
```


Multiple Return Values

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

    return num2, num1
```

terminal

```
> python3 maxmax.py
1 is smaller than 5
```

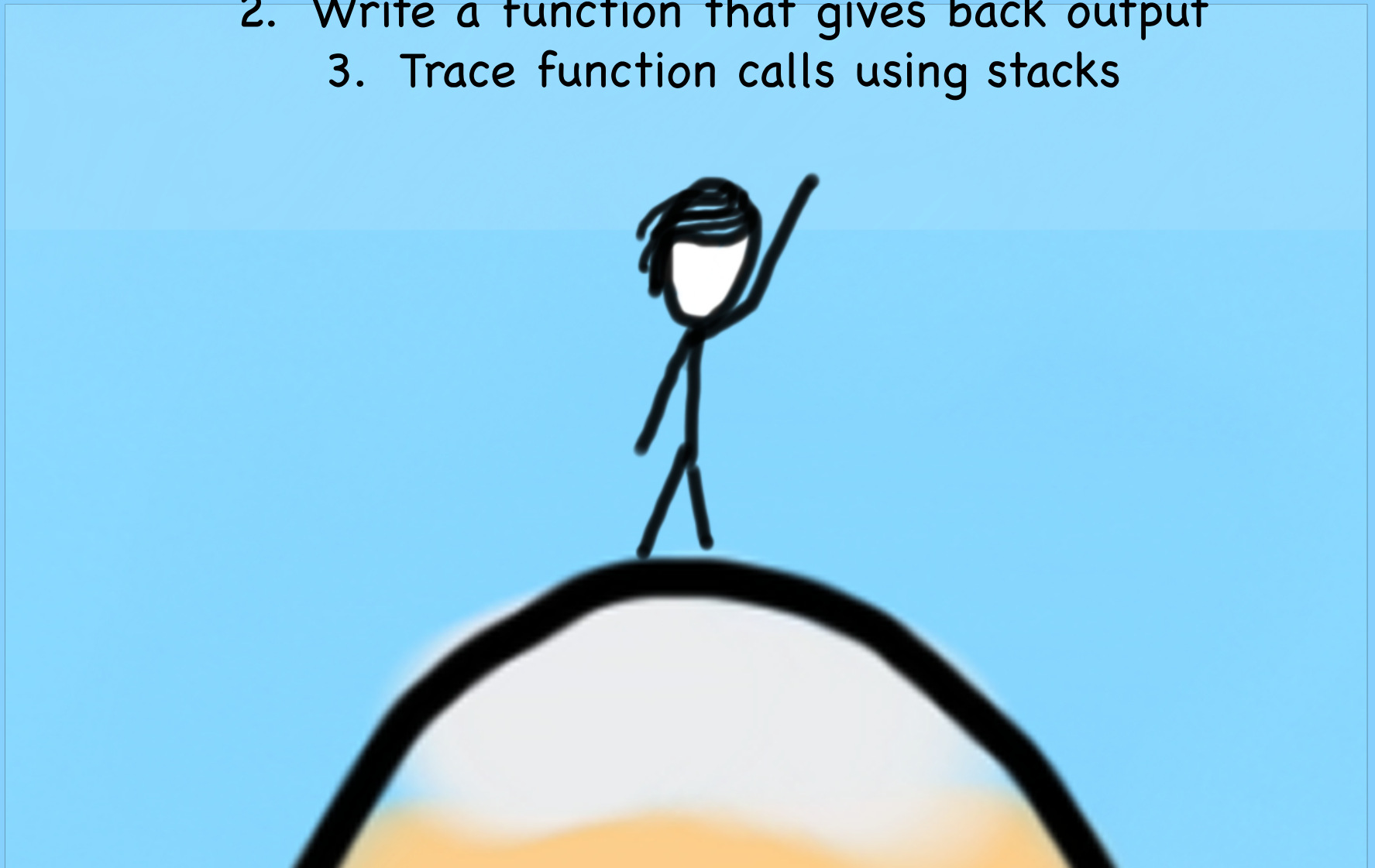
```
def main():
    smaller, larger = order(5, 1)
    print(str(smaller) + " is smaller than " + str(larger))
```

Lecture Plan

- Functions: From Karel to Python
- Example: Factorial
- More About Functions
- **Example: Graphics**

Learn How To:

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



Bonus Exercise

- Greek mathematicians took a special interest in numbers that are equal to the sum of their proper divisors (a proper divisor of *n* is any divisor less than *n* itself). They called such numbers ***perfect numbers***. For example, 6 is a perfect number because it is the sum of 1, 2, and 3, which are the integers less than 6 that divide evenly into 6. Similarly, 28 is a perfect number because it is the sum of 1, 2, 4, 7, and 14.
- Design and implement a Python program that finds all the perfect numbers between two limits. For example, if the limits are 1 and 10000, the output should look like this: