

# CS Bridge, Lecture 8

## Functions



1



# Learn How To:

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



# Calling functions

```
turn_right()
```

```
move()
```

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

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

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

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



# Defining a function

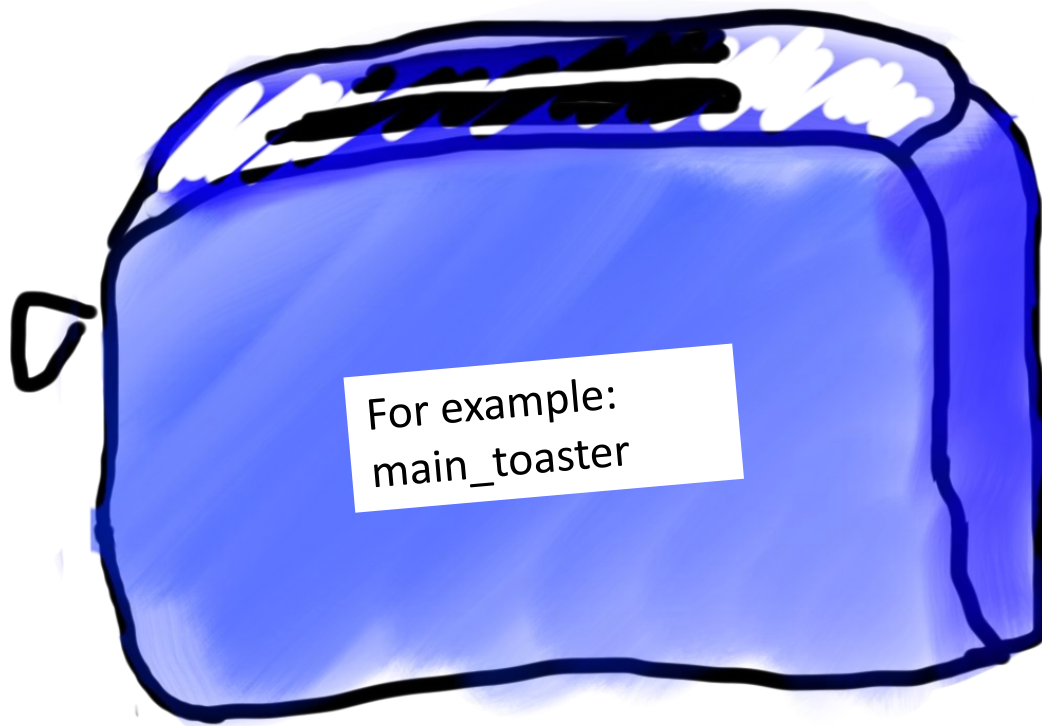
```
def turn_right():  
    turn_left()  
    turn_left()  
    turn_left()
```



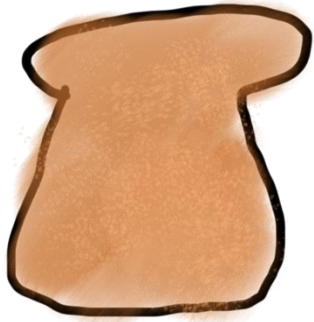
Big difference with python functions:  
Python functions can **take in data** and can **return data!**



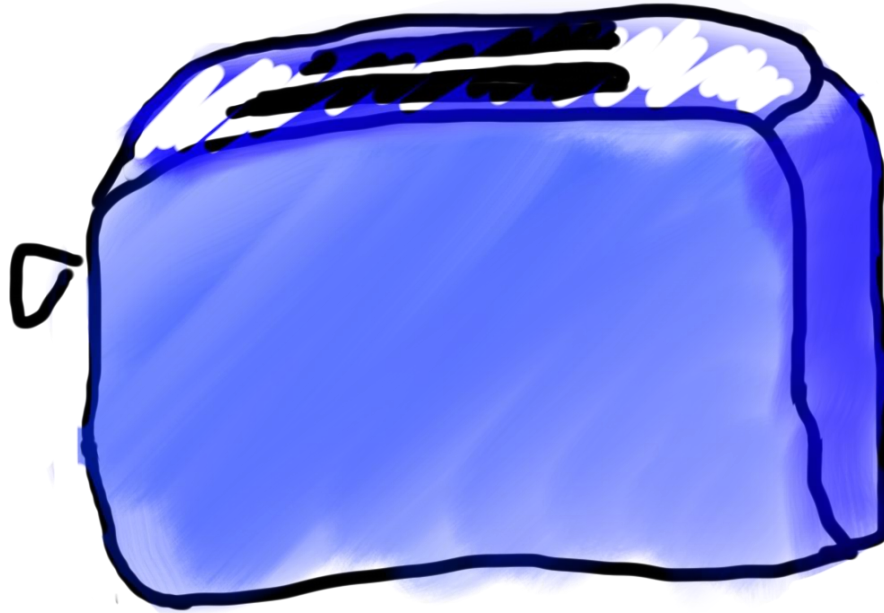
# Toasters are functions



# Toasters are functions



parameter



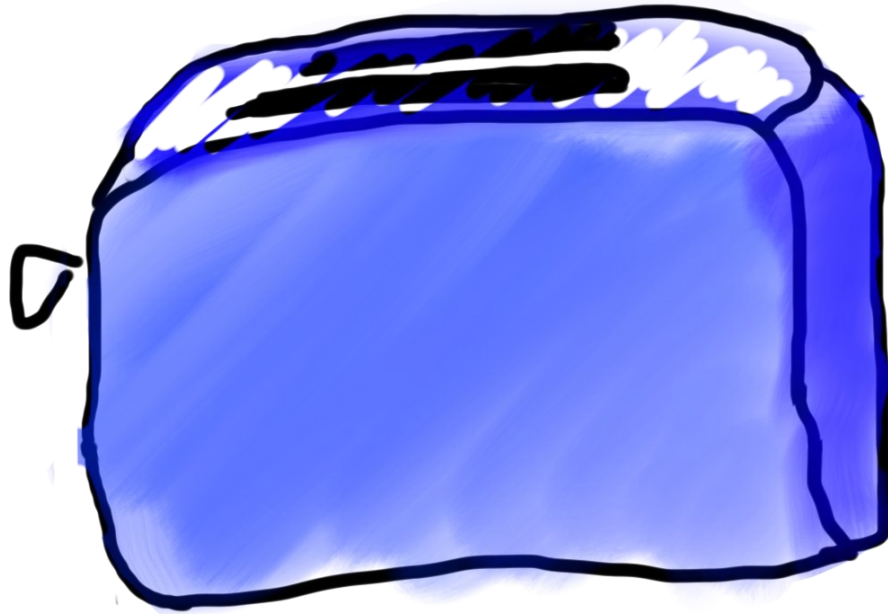
# Toasters are functions



parameter

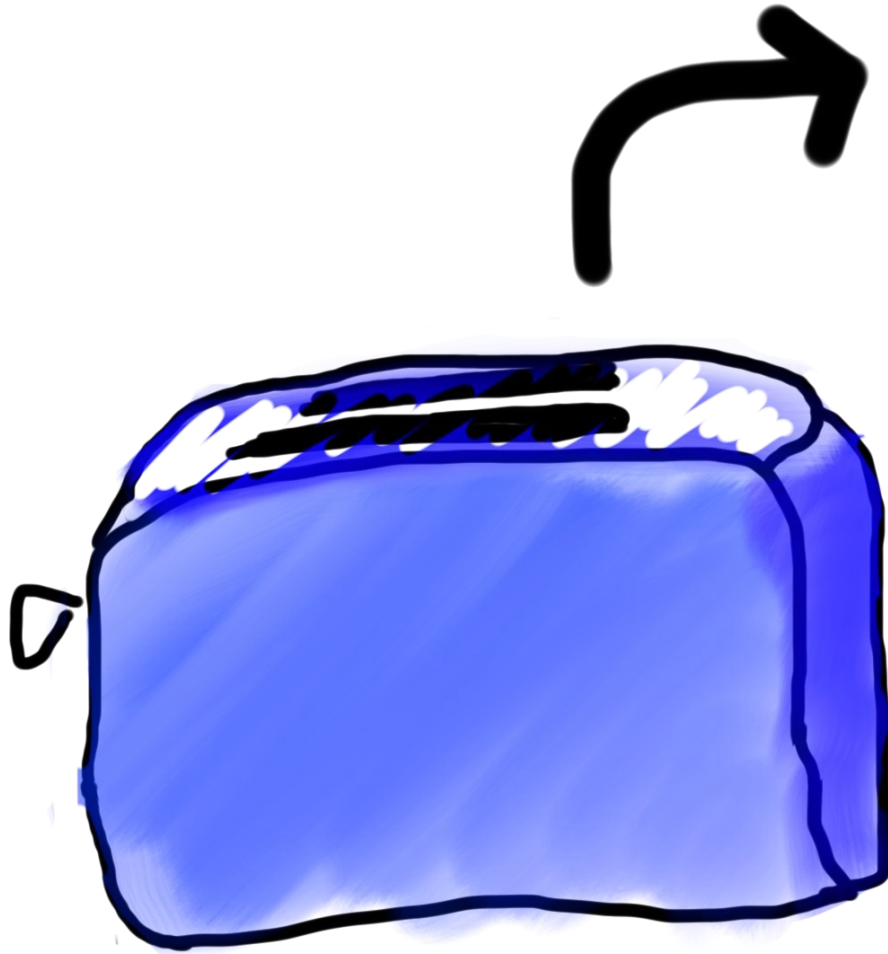


# Toasters are functions

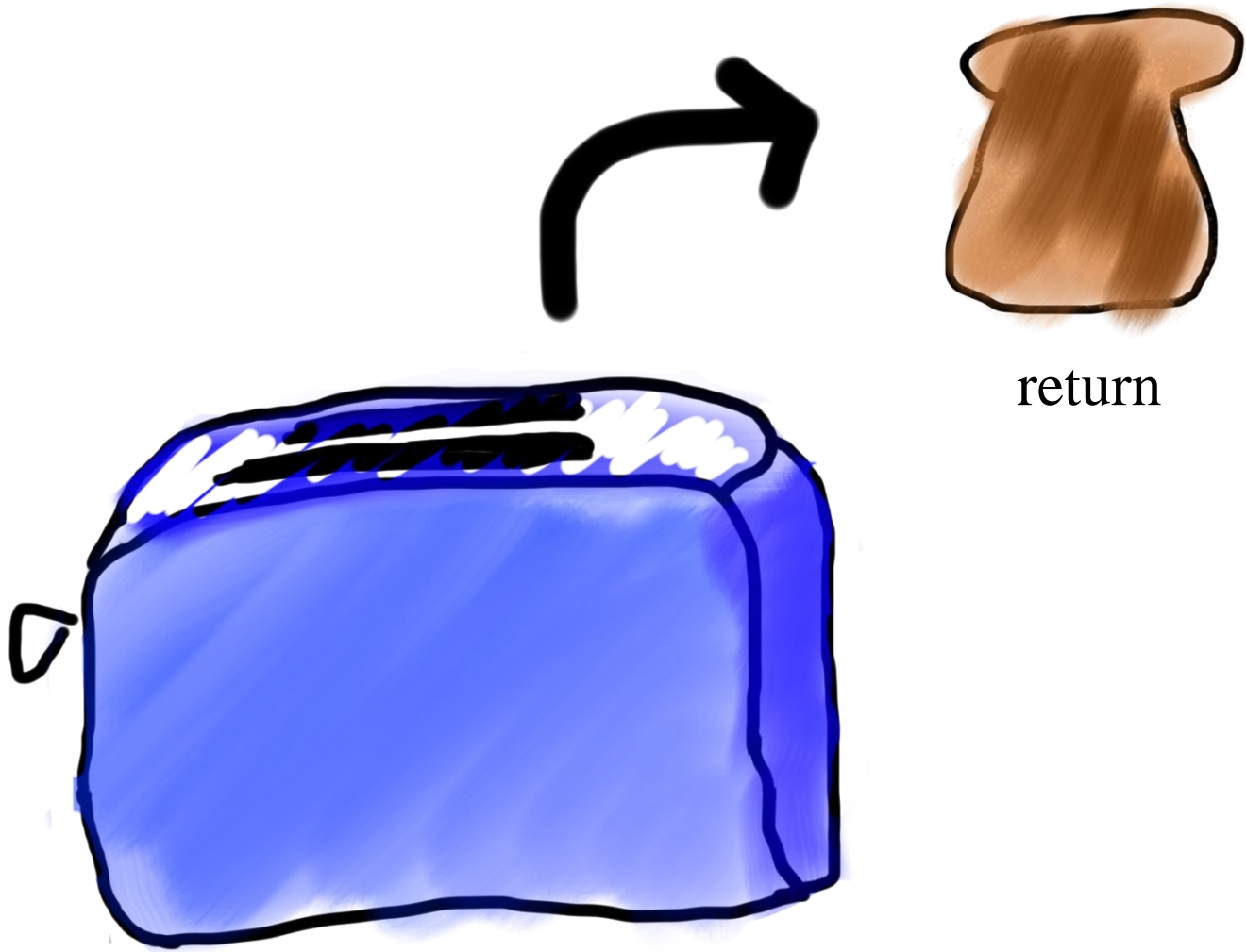




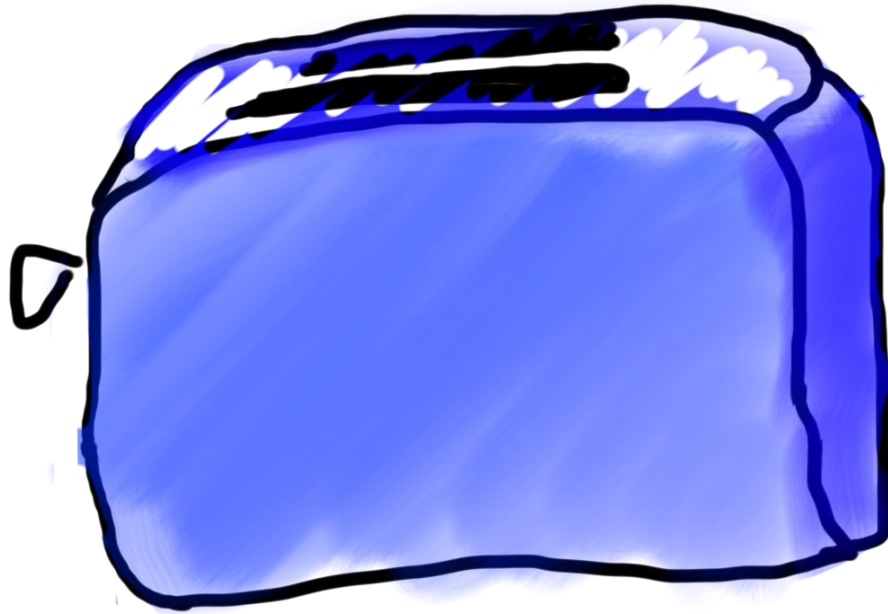
# Toasters are functions



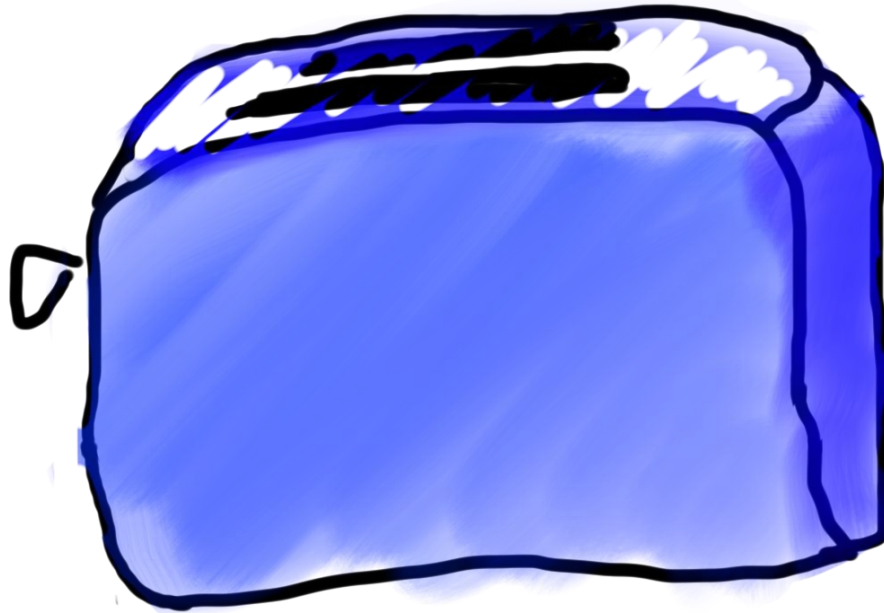
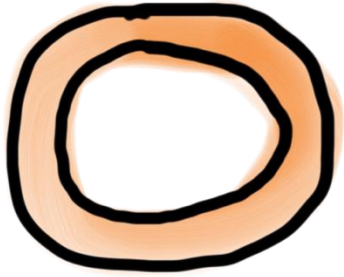
# Toasters are functions



# Toasters are functions



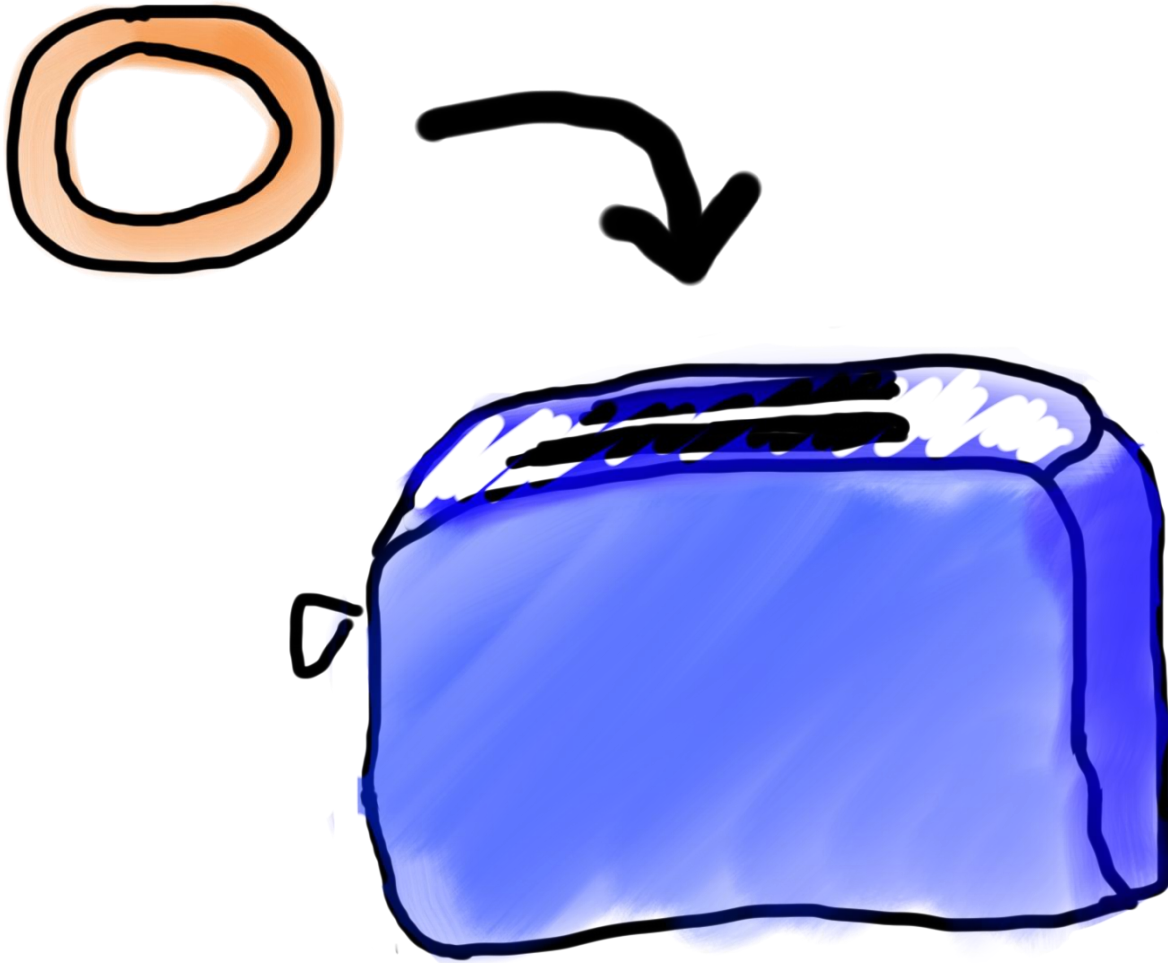
# Toasters are functions



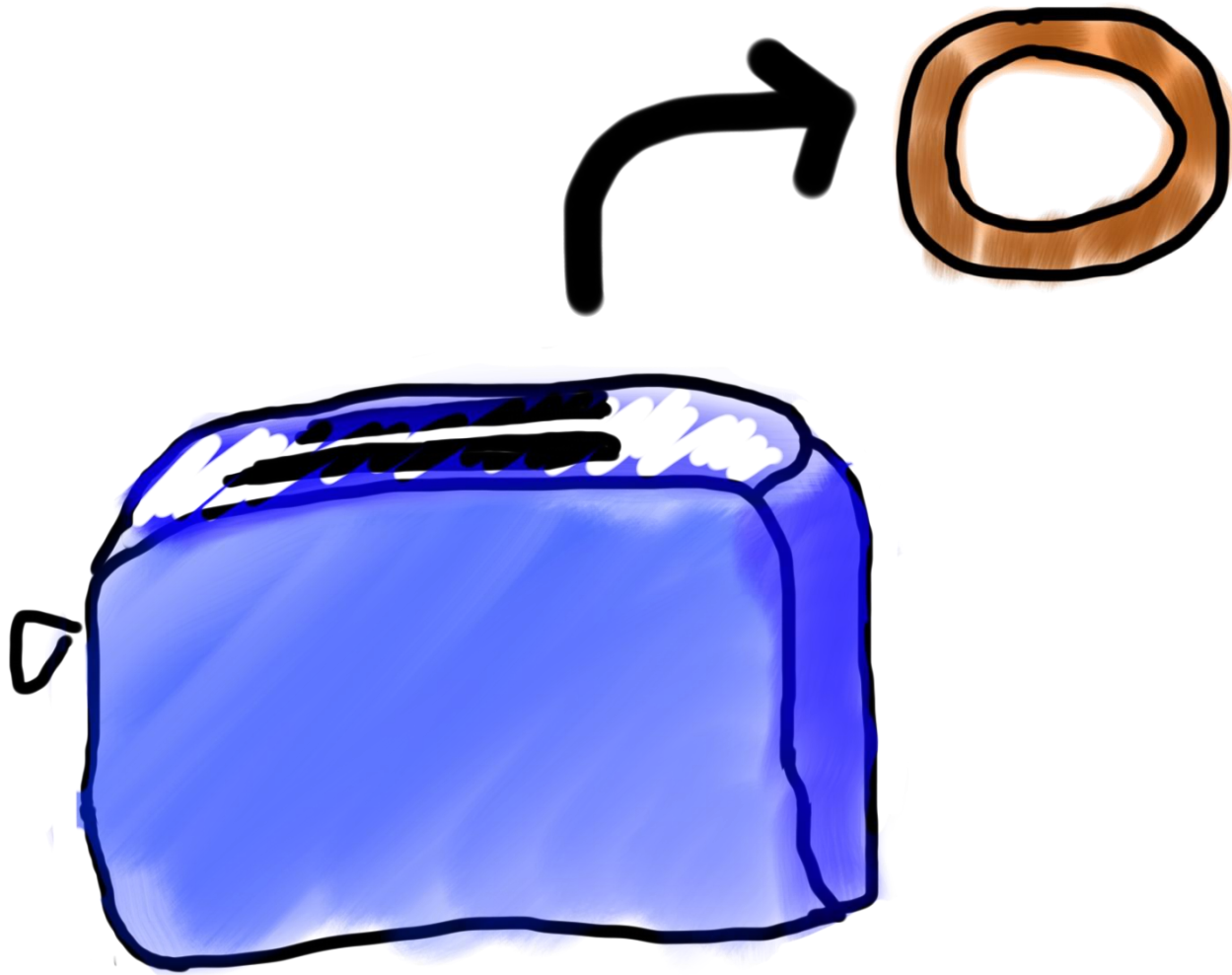
\* You don't need a second toaster if you want to toast bagels. Use the same one.



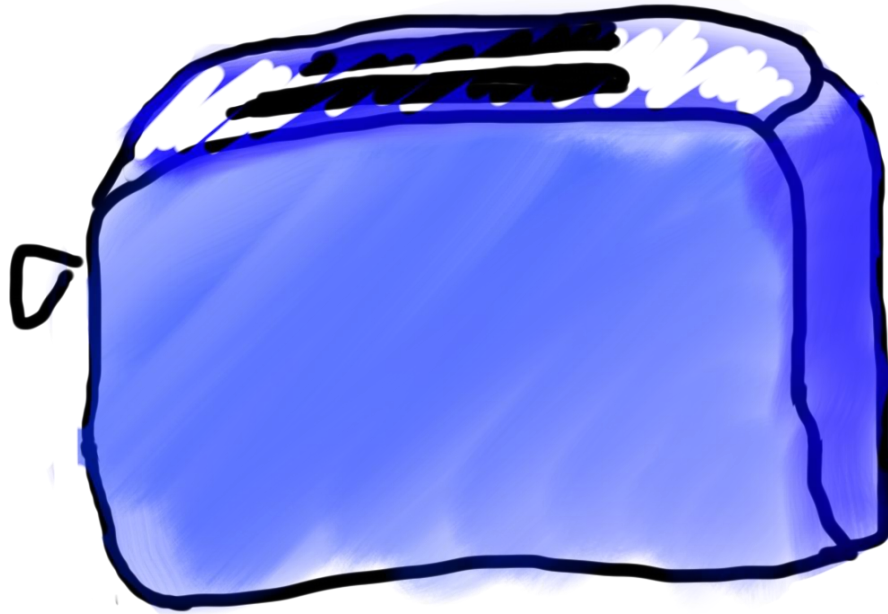
# Toasters are functions



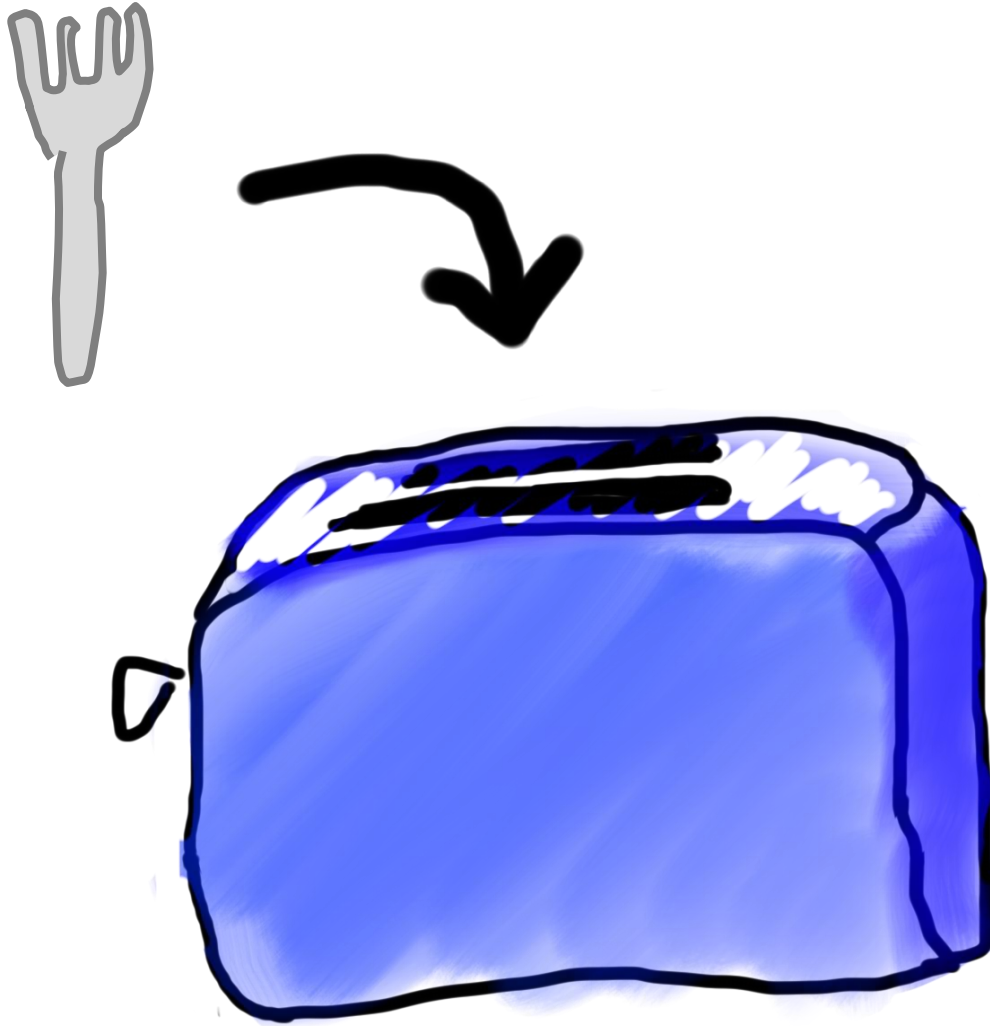
# Toasters are functions



# Toasters are functions



# Toasters are functions

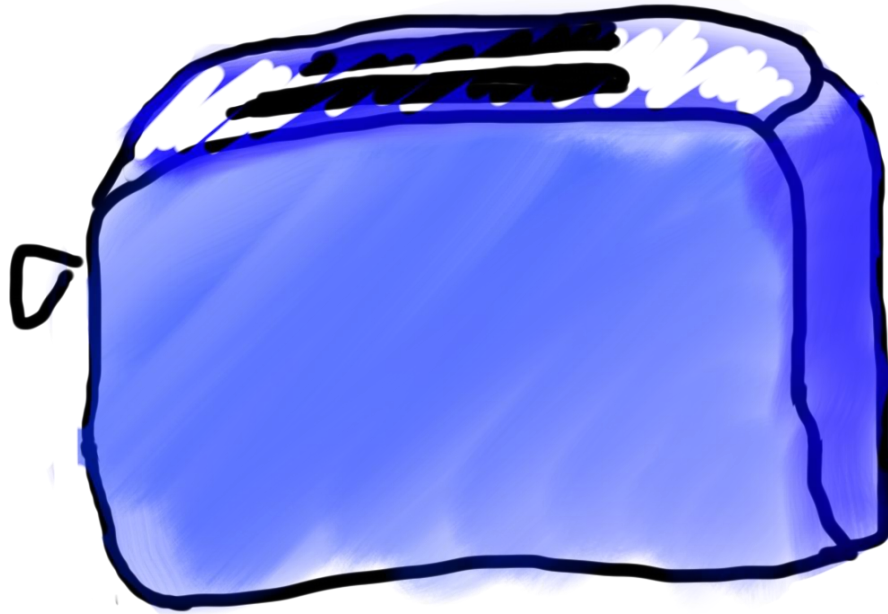




# Toasters are functions



# functions are Like Toasters



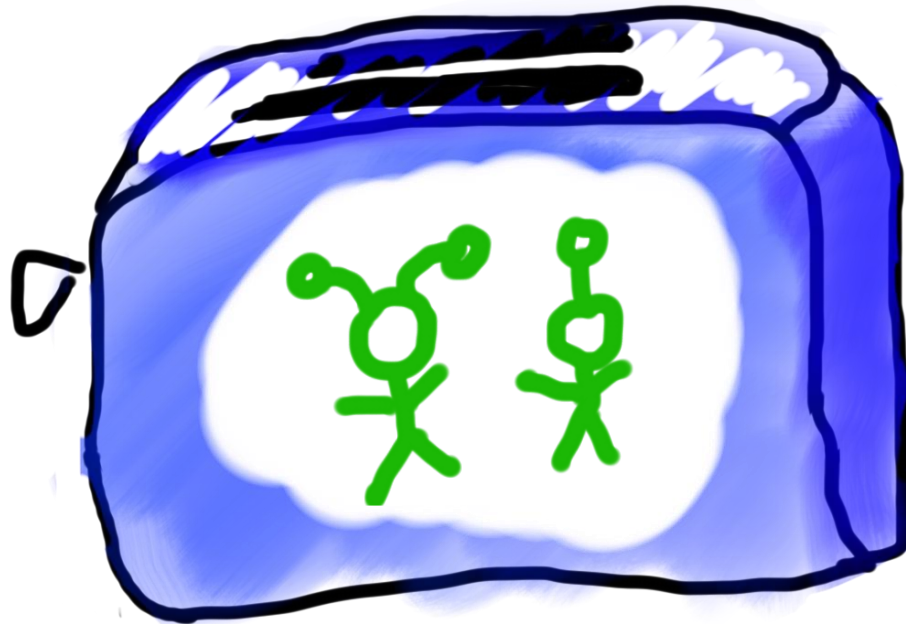
# functions are Like Toasters



# functions are Like Toasters



# functions are Like Toasters



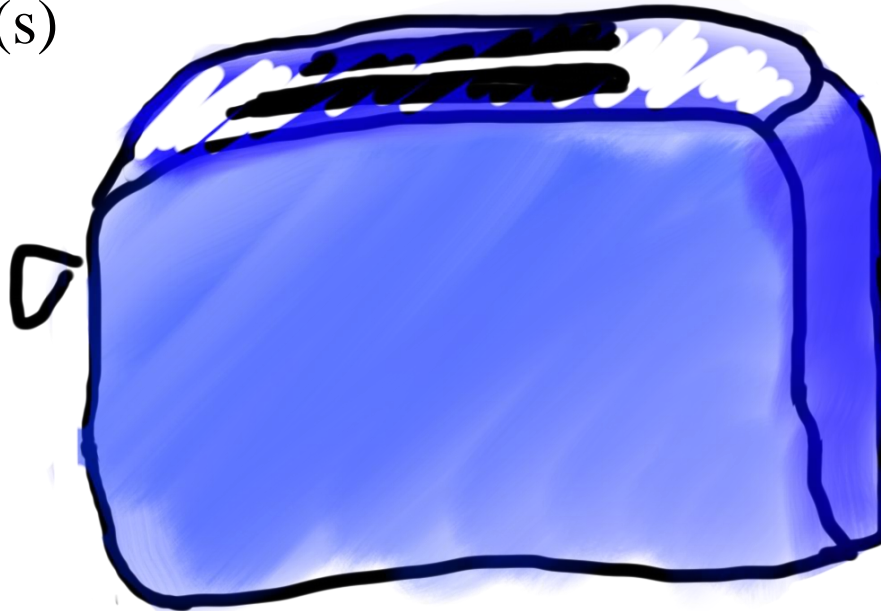
# functions are Like Toasters



parameter(s)



return



# 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



# Why use functions?

- Sometimes we require repeating the same task at different places of our programs.
- We then can write a code for doing that task and copy it to all the places that we require.





# Why use functions?

- But a better approach is to put that code into a function and call it at the places we need it.
- In that way, we can make our program shorter and more readable.
- Another advantage of it is its easier maintenance: If we need changes in our code, we just change the function. We do not need to change multiple copies of the same code.



# Divide and Conquer a Large Task

```
statement  
statement  
statement  
statement  
statement  
statement  
statement  
statement
```

```
def function1():  
    statement  
    statement  
  
def function2():  
    statement  
    statement  
    statement  
  
def function3():  
    statement  
    statement
```

```
function1()  
function2()  
function3()
```



# Classic Challenge



Perhaps the  
most  
underrated  
concept by  
students



# Function Naming Rules

- Every function should have a unique name.
- Function names follow the same naming rules as variable names.
  - Cannot use keywords as a function name
  - Cannot contain spaces
  - First character must be a letter or an underscore
  - All other characters must consist of letters, numerals and underscores
  - Names are case-sensitive
- Function names should better be descriptive of the task carried out by the function



- Here is a part of the lyrics of the song "Call me maybe":

Hey, I just met you  
And this is crazy

But here's my number  
So call me, maybe?

It's hard to look right  
At you baby

But here's my number  
So call me, maybe?

Hey, I just met you  
And this is crazy

But here's my number  
So call me, maybe?

And all the other boys  
Try to chase me

But here's my number  
So call me, maybe?

*Note that the marked  
text are chorus lines  
that keep repeating.*



# Functions

- Assume we want to write a program that types these lyrics into the command window.
- Remember that the repeating lines are a good application area for using functions.



```
def chorus():  
    print("But here's my number")  
    print("So call me, maybe?")  
    print()
```

```
print("Hey, I just met you")  
print("And this is crazy")  
chorus()  
print("It's hard to look right")  
print("At you baby")  
chorus()  
print("Hey, I just met you")  
print("And this is crazy")  
chorus()  
print("And all the other boys")  
print(" Try to chase me")  
chorus()
```

Hey, I just met you  
And this is crazy  
But here's my number  
So call me, maybe?

It's hard to look right  
At you baby  
But here's my number  
So call me, maybe?

Hey, I just met you  
And this is crazy  
But here's my number  
So call me, maybe?

And all the other boys  
Try to chase me  
But here's my number  
So call me, maybe?



```

def chorus():
    print("But here's my number")
    print("So call me, maybe?")
    print()

def hey_part():
    print("Hey, I just met you")
    print("And this is crazy")

hey_part()
chorus()
print("It's hard to look right")
print("At you baby")
chorus()
hey_part()
chorus()
print("And all the other boys")
print("Try to chase me")
chorus()

```

Hey, I just met you  
 And this is crazy  
 But here's my number  
 So call me, maybe?

It's hard to look right  
 At you baby  
 But here's my number  
 So call me, maybe?

Hey, I just met you  
 And this is crazy  
 But here's my number  
 So call me, maybe?

And all the other boys  
 Try to chase me  
 But here's my number  
 So call me, maybe?





# 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
```



Is returning  
the same as printing?

Is returning  
the same as printing?

NO

# Learn by Example



# Sum of Square of Two Numbers

Write a function that does the following jobs:

- Take two float inputs from the user.
- Take the square of both numbers and sum them.
- Print the result.



# Sum of Square of Two Numbers

```
Enter a number: 3
Enter a number: 4
Sum of squares is 25
```

Write your  
program



# Sum of Square of Two Numbers

# Version 1

```
def sum_squares():  
    a = float(input('Enter a number: '))  
    b = float(input('Enter a number: '))  
    result = a**2 + b**2  
    print(result)
```

```
sum_squares()
```





# General Definition of a Function

```
def function_name(parameter1, parameter2):  
    statement  
    statement  
    result = statement  
    return result
```



*Parameters which  
function expects as  
inputs*



*Output of the  
function*



# Sum of Square of Two Numbers

- Take two float inputs from the user.
- Take the square of both numbers and sum them in a function.
- Return the result to the caller.
- Print the result.



# Sum of Square of Two Numbers

```
Enter a number: 3
Enter a number: 4
Sum of squares is 25
```

Write your  
program



# Sum of Square of Two Numbers

# Version 2

```
def sum_squares():  
    a = float(input('Enter a number: '))  
    b = float(input('Enter a number: '))  
    result = a**2 + b**2  
    return result  
  
print(sum_squares())
```



# Sum of Square of Two Numbers

# Version 3

```
def sum_squares(a,b):  
    result = a**2 + b**2  
    return result  
  
a = float(input('Enter a number: '))  
b = float(input('Enter a number: '))  
print(sum_squares(a,b))
```



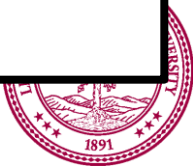
# No Parameter, No Return

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

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

terminal

```
> python3 intro.py
```



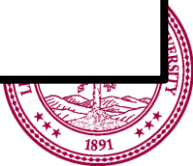
# No Parameter, No Return

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

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

terminal

```
> python3 intro.py
```



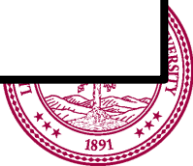
# No Parameter, No Return

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

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

terminal

```
> python3 intro.py
```





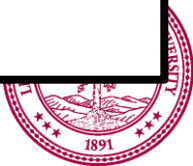
# No Parameter, No Return

```
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
```



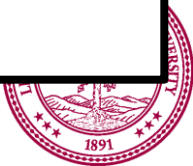
# No Parameter, No Return

```
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

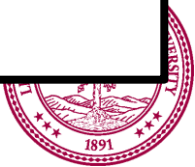
```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

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



terminal

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



# 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))
```



# 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

More Examples

# 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



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

terminal

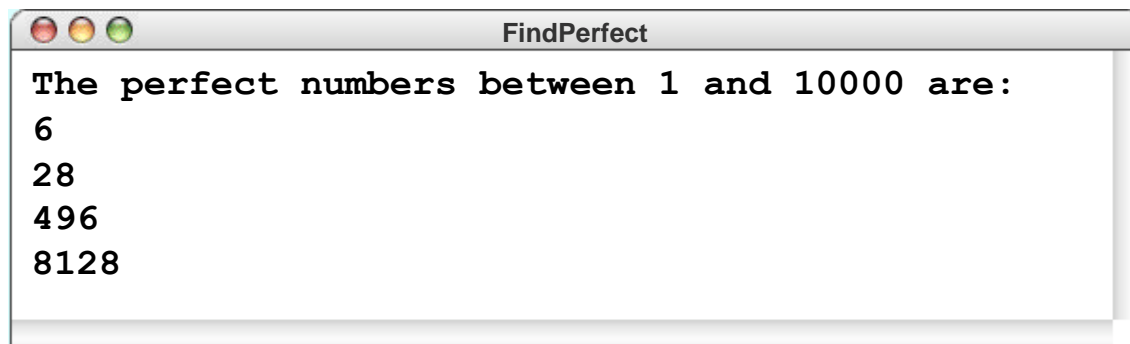
```
> python3 maxmax.py  
1 is smaller than 5
```

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



# 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:



```
FindPerfect
The perfect numbers between 1 and 10000 are:
6
28
496
8128
```

