Functions

Chris Gregg
Based on slides by Chris Piech and Mehran Sahami
CS106A, Stanford University
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

For example:
main_toaster

- Thanks Mehran
Toasters are functions

parameter
Toasters are functions

parameter
Toasters are functions
Toasters are functions
Toasters are functions

return
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**: what you call the function
- **parameters**: information passed into function
- **return**: information given back from the function
Perhaps the most underrated concept by students
def main():
    mid = average(5.0, 10.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():  
    mid = average(5.0, 10.2) 
    print(mid)

def average(a, b):
    sum = a + b 
    return sum / 2
def main():
    mid = average(5.0, 10.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(5.0, 10.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(5.0, 10.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(5.0, 10.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(5.0, 10.2)
    print(mid)

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

Anatomy of a function

This call "evaluates" to the value returned

ends the function and gives back a value
def main():
    mid = average(5.0, 10.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(5.0, 10.2)
    print(mid)

def average(a, b):
    sum = a + b
    return sum / 2
def main():
    mid = average(5.0, 10.2)
    print(mid)

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

Anatomy of a function

When a function ends it “returns”
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
def print_intro():
    print("Welcome to class")
    print("It's the best part of my day.")

def main():
    print_intro()

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

def main():
    print_intro()

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

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

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

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

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

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

def main():
    print_intro()

terminal

> python intro.py
Welcome to class
It’s the best part of my day
def print_intro():
    print("Welcome to class")
    print("It's the best part of my day.")

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

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

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

def main():
    print_opinion(5)

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

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

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

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

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

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

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

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

def main():
    print_opinion(5)
def meters_to_cm(meters):
    return 100 * meters

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

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

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

```
> python3 m2cm.py
```

Parameter and Return Example

main memory

No variables

terminal

> python3 m2cm.py
```python
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

meteresToCm memory
terminal

> python3 m2cm.py
def meters_to_cm(meters):
    return 100 * meters

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

main memory

No variables

metersToCm memory

meters 5.2

terminal

> python3 m2cm.py
```python
def meters_to_cm(meters):
    return 100 * meters

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

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

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

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

terminal

> python3 m2cm.py

520.0
```python
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

> python3 m2cm.py

520.0
```python
def meters_to_cm(meters):
    return 100 * meters

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

Parameter and Return Example

```
> python3 m2cm.py
520.0
```
def meters_to_cm(meters):
    return 100 * meters

def main():
    print(meters_to_cm(5.2))
    print(meters_to_cm(9.1))
```python
def meters_to_cm(meters):
    return 100 * meters

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

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

def main():
    print(meters_to_cm(5.2))
    print(meters_to_cm(9.1))
def meters_to_cm(meters):
    return 100 * meters

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

> python3 m2cm.py
520.0
def meters_to_cm(meters):
    return 100 * meters

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

910.0
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
910.0

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)
Is returning the same as printing?
Is returning the same as printing?

NO
```python
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

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

terminal

```
> python maxmax.py
```

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

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

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

main memory

No variables

terminal

> python maxmax.py
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

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

Main Memory

No variables

Terminal

> python maxmax.py
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

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

def main():
    larger = max(5, 1)
Multiple Return Statements

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

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

Terminal output:

```
> python maxmax.py
```

Main memory:

- `No variables`

Max memory:

- `num1: 5`
- `num2: 1`
```python
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

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

```
$ python maxmax.py
```

<table>
<thead>
<tr>
<th>main memory</th>
<th>max memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>No variables</td>
<td>num1 5 num2 1</td>
</tr>
</tbody>
</table>

Terminal:

```
$ python maxmax.py
```
```python
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

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

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

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

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

taxtream
> python maxmax.py

No variables
num1 5
num2 1
Multiple Return Statements

main memory

No variables

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

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

> python maxmax.py
```

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

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

terminal
> python maxmax.py

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

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

terminal
> python maxmax.py
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

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

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

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

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

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

Multiple Return Statements

main memory

No variables
Multiple Return Statements

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

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

main memory

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

def main():
    larger = max(1, 5)
I give you

print_no_return

What functions do you define?
MAX_NUM = 4

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

def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
Understand the Mechanism
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))
```python
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i

    return result

n 0  result  i
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i

    return result

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

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))

0    1
```python
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))

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

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td></td>
</tr>
</tbody>
</table>
```python
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

```
0   1
1   1
```

```python
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(1, n+1):
        result *= i
    return result

0    1

1    1    1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

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

0    1
1
2
3
4
5
6
7
8
9
10
```python
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

```
0    1
1    1
2    2
```
```python
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

```
0    1
```

```
result  1  2
```
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))

1
1
0    1
1    1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0    1
1    1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
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))

0    1
1    1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0  1
1  1
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

    i  2
    0  1
    1  1
    2  2
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0    1
1    1
2    2
```python
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

The output is:

```
0 1
1 1
2 2
```
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

    # Example output
    print(i, 3)

    # Sample output
    0 1
    1 1
    2 2
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0    1
1    1
2    2
```python
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))
```

<table>
<thead>
<tr>
<th>i</th>
<th>factorial(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0    1
1    1
2    2
3    6
def main():
    for i in range(MAX_NUM):
        print(i, factorial(i))

0  1
1  1
2  2
3  6
Every time a function is called, new memory is created for the call.
// NOTE: This program is buggy!!

def add_five(x):
    x += 5

def main():
    x = 3
    add_five(x)
    print("x = " + x)
// NOTE: This program is feeling just fine...

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

def main():
    x = 3
    x = add_five(x)
    print("x = " + x)
For primitives: Variables are not passed when you use parameters. Values are passed.
Pass by “Value”

- Thanks Mehran
More Examples
def main():
    num = 5
    cow(num)

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

def cow():
    num = 10
    print(num)
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
def main():
    print("hello world")
    say_goodbye()

def say_goodbye():
    print("goodbye!")
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
Remember Booleans?
karelIsAwesome = true

myBool = 1 < 2
def main():
    for i in range(100):
        if is_square(i):
            print(i)
```python
def main():
    for i in range(100):
        if is_square(i):
            print(i)

def is_square(x):
    root = math.sqrt(x)
    if is_whole(root):
        return True
    else:
        return False
```

Boolean Return
def main():
    for i in range(100):
        if is_square(i):
            print(i)

def is_square(x):
    root = math.sqrt(x)
    return is_whole(x)
• 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
```