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

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

terminal

```
0
1
2
3
4
5
6
7
8
9
```
Welcome to the CS106A game show! Choose a door and win a prize
Door: 2
You chose door 2
You win $\$\ldots\$
```
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
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 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")

math.sqrt(25)

float("0.42")
```
Defining a function

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

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

Thanks Dan Schwartz
def turn_right():
    turn_left()
    turn_left()
    turn_left()

def move_n(n_moves):
    for i in range(n_moves):
        move()

def main():
    turn_right()
    move_n(10)

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

For example:

main_toaster
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

Piech + Sahami, CS106A, Stanford University
functions are Like Toasters
functions are Like Toasters
functions are Like Toasters
functions are Like Toasters
functions are Like Toasters
functions are Like Toasters

parameter(s) → return
Classic Challenge for CS106A

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
Anatomy of a function

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

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

Piech + Sahami, CS106A, Stanford University
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

This call "evaluates" to the value returned

Ends the function and gives back a value

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
Anatomy of a function

No parameters (expects no input)

```python
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
Formally

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

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

Function author’s contract - average:
If you call this function you must provide two params. The function will give back a return value
User: Uses the terminal (later UI)

Function Caller (Coder): Uses helper functions

Function Author (Coder): Writes helper functions others can use
User

- Waddle
  - Uses the terminal (later UI)

Function Caller (Coder)

- Rawr
  - Uses helper functions

Function Author (Coder)

- Meerkat
  - Writes helper functions others can use

---

Piech + Sahami, CS106A, Stanford University
Learn by Example
def print_intro():
    print("Welcome to class")
    print("It's the best part of my day.")

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

def main():
    print_intro()
```

```bash
> python intro.py
```

No Parameter, No Return
def print_intro():
    print("Welcome to class")
    print("It's the best part of my day.")

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

def main():
    print_intro()
```

terminal

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

def main():
    print_opinion(5)
**Parameter Example**

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

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

---

**Terminal**

```
> python opinion.py
```

---

*No variables*
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)
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)
```
```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)
Parameter Example

```python
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
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 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)
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

terminal

> python m2cm.py

Piech + Sahami, CS106A, Stanford University
```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

```
> python m2cm.py
```

Piech + Sahami, CS106A, Stanford University
**Parameter and Return Example**

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

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

<table>
<thead>
<tr>
<th>main memory</th>
<th>metersToCm memory</th>
<th>terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>No variables</td>
<td>meters: 5.2</td>
<td>&gt; python m2cm.py</td>
</tr>
</tbody>
</table>

Piech + Sahami, CS106A, Stanford University
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)
```

```
Parameter and Return Example

main memory

No variables

terminal

> python m2cm.py

520.0

Piech + Sahami, CS106A, Stanford University
```
Parameter and Return Example

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

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

Terminal:
```
> python m2cm.py
520.0
```
```
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)

This is our favorite paradigm (fn author returns, fn caller print).
It’s the most flexible, modular. Its elegance is unparalleled.
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
> python m2cm.py
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
```python
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
```
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
> python 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))
def meters_to_cm(meters):
    return 100 * meters

def main():
    print(meters_to_cm(5.2))
    print(meters_to_cm(9.1))
# How is this function
```python
def meters_to_cm_case1(meters):
    return 100 * meters
```

# Different than this function?
```python
def meters_to_cm_case2(meters):
    print(100 * meters)
```
# How is this function
```python
def meters_to_cm_case1(meters):
    return 100 * meters
```

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

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

- **Return**
  - Caller receives value.
  - Can store it and/or print it

- **Print**
  - User receives value on the console
Is returning the same as printing?
Is returning the same as printing?

No
# How is this function
```python
def meters_to_cm_case1():
    meters = float(input("M: "))
    return 100 * meters
```

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

When a function requires a value does it call `input` itself?

- **parameter**: Fn author receives value. Can store it and/or print it
- **input**: User inputs value within the function. Seems nice, but hard to build upon!
Ideal flow of information: Input

```python
def example_caller():
    data = float(input("enter:"))
    call_the_author(data)
```
Ideal flow of information: Input

def example_caller():
    data = float(input("enter:"))
    result = call_the_author(data)
    print(result)
def main():
    meters = float(input("meters: "))
    cms = meters_to_cm(meters)
    print(cms + " cm")
As “input” the same as parameters?
As “input” the same as parameters? NO
Multiple Return Statements

```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)

main memory
No variables

> 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

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

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

Terminal:
```
> python maxmax.py
```
Multiple Return Statements

```python
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
```

---

Piech + Sahami, CS106A, Stanford University
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

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

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

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

**Multiple Return Statements**

main memory

No variables

max memory

<table>
<thead>
<tr>
<th>num1</th>
<th>num2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

terminal

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

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

terminal
> python maxmax.py

main memory

No variables
Multiple Return Statements

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

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

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

Terminal output:
```
$ python maxmax.py
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)
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)
Multiple Return Statements

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

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

- **main memory**: No variables
- **max memory**
  - num1: 1
  - num2: 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

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

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

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

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

Piech + Sahami, CS106A, Stanford University
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2

def main():
    larger = max(1, 5)
if extra_time:
Function for IO

What functions do you define?