Functions

Chris Piech and Mehran Sahami
CS106A, Stanford University
Boolean Variable

karel_is_awesome = True

my_bool = 1 < 2
a = True

b = False

both_true = a and b

either_true = a or b

opposite = not a
Game Show

Welcome to the CS106A game show!
Choose a door and win a prize
Door: 2
You chose door 2
You win $[blackacted]
Choose a Door

door = int(input("Door: "))
# while the input is invalid
while door < 1 or door > 3 :
    # tell the user the input was invalid
    print("Invalid door!")
    # ask for a new input
    door = int(input("Door: "))
The Door Logic

prize = 4

if door == 1:
    prize = 2 + 9 // 10 * 100

elif door == 2:
    locked = prize % 2 != 0
    if not locked:
        prize += 6

elif door == 3:
    for i in range(door):
        prize += i
Civilization advances by extending the number of operations we can perform without thinking about them.

-Alfred North Whitehead
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)
def turn_right():
    turn_left()
    turn_left()
    turn_left()
Toasters are functions

For example:
main_toaster
Toasters are functions

parameter
Toasters are functions
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

```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
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
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

    Ends the function and gives back a value

Anatomy of a function

This call “evaluates” to the value returned
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)

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
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()
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()
Parameter Example

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

def main():
    print_opinion(5)

terminal
> python opinion.py
```python
def print_opinion(num):
    if (num == 5):
        print("I love 5!"")
    else:
        print("Whatever")

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

```
No variables

> 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)
```

![Diagram showing main memory with "No variables" and print_opinion memory with empty boxes. Terminal shows > python opinion.py]
```python
def print_opinion(num):
    if (num == 5):
        print(“I love 5!”)
    else:
        print(“Whatever”)

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

```
> python opinion.py
```

Parameter Example

main memory

| No variables |

print_opinion memory

| num |

terminal

> python opinion.py
def print_opinion(num):
    if (num == 5):
        print("I love 5!")
    else:
        print("Whatever")

def main():
    print_opinion(5)
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

main memory

No variables

print_opinion memory

num 5

terminal

> python opinion.py
I love 5!
**Parameter Example**

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

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

```bash
$ python opinion.py
I love 5!
```
Parameter Example

main memory

No variables

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

def main():
    print_opinion(5)

terminal

> python opinion.py
I love 5!
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!
```
```python
def meters_to_cm(meters):
    return 100 * meters

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

```
> python m2cm.py
```

Parameter and Return Example
def meters_to_cm(meters):
    return 100 * meters

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

> python m2cm.py
**Parameter and Return Example**

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

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

main memory

`No variables`

terminal

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

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

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

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

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

Parameter and Return Example

main memory

meteresToCm memory

terminal

> python m2cm.py
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

result 520.0

terminal

> python m2cm.py

520.0
def meters_to_cm(meters):
    return 100 * meters

def main():
    result = meters_to_cm(5.2)
    print(result)
```python
# Parameter and Return Example

def meters_to_cm(meters):
    return 100 * meters

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

terminal

```
> python m2cm.py
```

Piech + Sahami, CS106A, Stanford University
def meters_to_cm(meters):
    return 100 * meters

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

terminal
> python m2cm.py
Parameter and Return Example

```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
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))
```

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

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

**Parameter and Return Example**

```
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))

Parameter and Return Example

terminal
> python 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
> python 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
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

\[ \text{No variables} \]

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)
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)

Multiple Return Statements

main memory

max memory

No variables

num1 5

num2

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)
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)
```

**main memory**

| No variables |

**max memory**

<table>
<thead>
<tr>
<th>num1</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>num2</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

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

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

> python maxmax.py
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
> python maxmax.py
```

main memory

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

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

main memory

No variables

```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)
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))

        i  0
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
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
```python
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

n | 0 | result | 1 | i | 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))

print(0, factorial(0))

i  0

0  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  1

0  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))

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

    return result

0    1
1    result 2  i 3

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

```
0   1
1   1
```

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

```
0 1
```

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

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))

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

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

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

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

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

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

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”
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?
Boolean Variable

karelIsAwesome = true

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

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