

CS-121:

Week 3

Still in the fog

Unkown
Area

Foggy
Area

Fully Visible
Area

The 15 weeks

- ➡️ **Weeks 1-5:** Learn the basics of program flow, input and output, variables, conditionals, loops and functions
- **Weeks 5-10:** Practice the basics.
- **Weeks 11-15:** Choose your own adventure.

The 15 weeks

- **Weeks 1-5:** Learn the basics. Program flow, input and output, variables, conditionals, loops and functions
-  **Weeks 5-10:** Practice the basics with projects + learn additional topics (data structures, algorithms, file I/O)
- **Weeks 11-15:** Choose your own adventure.

The 15 weeks

- **Weeks 1-5:** Learn the basics. Program flow, input and output, variables, conditionals, loops and functions
- **Weeks 5-10:** Practice the basics with projects + learn additional topics (data structures, algorithms, file I/O)
-  **Weeks 11-15:** Choose your own adventure final project and review for future exams and courses.



Upcoming

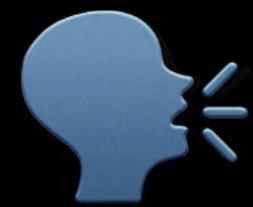
- Project 2: Due Thursday by 3pm (before class)
- Lab 3: Functions. Tomorrow 2-4pm
- Project 3: Starts Thursday

Project Submission Reminders

- Fill out your project reports (report.md) before submitting
- If you use outside help (including permitted AI), document it in your report
- Make sure you can explain all parts of your code

In the beginning...

Humans ❤️ Words
(language)



Linguistic Determinism

- Language shapes (and limits) the way we think
- You can't think about things you don't have words for
- Culture and language are intertwined



OATMEAL. *n. f.* [oat and meal.] Flower made by grinding oats.

Oatmeal and butter, outwardly applied, dry the scab on the head.
Arbuthnot on Aliment.

Our neighbours tell me oft, in joking talk,
Of ashes, leather, *oatmeal*, bran, and chalk. *Gay.*

OATMEAL. *n. f.* An herb. *Ainsworth.*

OATS. *n. f.* [aten, Saxon.] A grain, which in England is generally given to horses, but in Scotland supports the people.

It is of the grass leaved tribe ; the flowers have no petals, and are disposed in a loose panicle : the grain is eatable. The meal makes tolerable good bread. *Miller.*

The *oats* have eaten the horses. *Shakespeare.*

It is bare mechanism, no otherwise produced than the turning of a wild *oatbeard*, by the insinuation of the particles of moisture. *Locke.*

For your lean cattle, fodder them with barley straw first, and the *oat* straw last. *Mortimer's Husbandry.*

His horse's allowance of *oats* and beans, was greater than the journey required. *Swift.*

OATTHISTLE. *n. f.* [oat and *thistle*.] An herb. *Ains.*

Hallelujah

Francesco Parrino
Piano Arrangement

Words and Music by
Leonard Cohen

Moderato



6 Nf3

Kieseritzky's more pleasant experience with this line continued 6 Nc3 Ng4 7 Nh3 Nc6 8 Nd5 Nd4 9 Nxc7+ Kd8 10 Nxa8 f3 11 d3 f6 12 Bc4 d5 13 Bxd5 Bd6? 14 Qe1? fxg2+ 15 Kxg2 Qxh3+!! 16 Kxh3 Ne3+ 17 Kh4 Nf3+ 18 Kh5 Bg4# (0-1)
Schulten-Kieseritzky, Paris 1844.

On this occasion the boot was firmly on the other foot!

$$\zeta(s) = \sum *n = 1^\infty \frac{1}{n^s} = \prod *p \text{ prime} \left(1 - \frac{1}{p^s}\right)^{-1}$$

Messages

Mom

Edit

Call

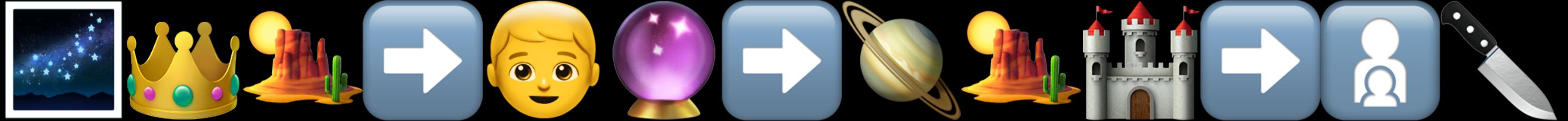
FaceTime

Contact >

"What does IDK, LY &
TTYL mean?"

I don't know, love you,
talk to you later.

OK, I will ask your sister.



```
98  ---
99
100 $$
101 \zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p \text{ prime}} \left(1 - \frac{1}{p^s}\right)^{-1}
102 $$
103
104 ^ Math notation is a language for numbers and relationships
105
106 ---
107
108 ! [inline] (images/text.png)
109
110 ^ Internet/texting slang is a language for the digital age
111
112 ---
113
114 # 🎉🎊➡️🥳🎈➡️🥳🎈➡️🎉🎊➡️🥳🎈➡️💦🚫➡️🥳🎈➡️🎉🎊➡️🥳🎈➡️🔥🎉🎊➡️🥳🎈
115
116 ^ Emoji language for the digital age (can you decode it? what movie is it?) (answer: Dune)
117
118 ---
119
120 ! [inline] (images/deckset.png)
121
```



james yu ✅
@jamesjyu

humans love language so much we taught sand how to speak

7:28 PM · Oct 17, 2022

...

ChatGPT 4 >

james yu ✅
@jamesjyu

humans love language so much we taught sand how to speak

7:28 PM · Oct 17, 2022

ELI5

The tweet by James Yu, "humans love language so much we taught sand how to speak," is a clever reflection on the development of computer technology. Sand is a primary component of silicon, which is used to make the semiconductors at the heart of microchips and computer processors. By programming these silicon-based chips, we essentially teach them to process and "speak" languages—computer languages like Python, JavaScript, and many others. This illustrates not only our capacity to manipulate natural materials into complex technology but also our deep fascination with communication and language.

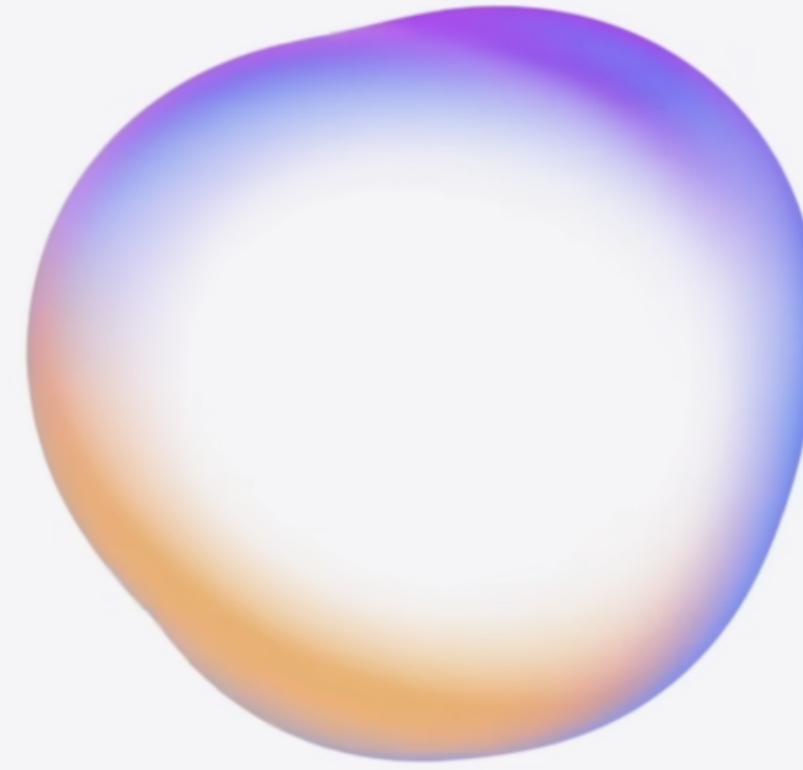
write a nice haiku about this concept

Silicon whispers,
Language birthed from grains of sand,
Words in circuits dance.

Message ChatGPT

Speaking Typing
things into
existence

AKA: Prompt Engineering



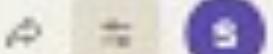
Shrimp with a cowboy hat



Shrimp with a cowboy hat

A +

8-Bit Crab Art



Certainly! I'll create a simple 8-bit style crab for you using SVG.



8-bit style crab SVG
Click to open image

Here's a simple 8-bit style crab created using SVG. The crab features:

1. A rectangular red body
2. Two small black square eyes
3. Two red square claws
4. Four rectangular legs

The crab is designed in a minimalist 8-bit style, using only basic geometric shapes and a limited color palette (red and black). This style is reminiscent of early video game graphics.

Let me know if you'd like any modifications or if you have any questions about the SVG code!

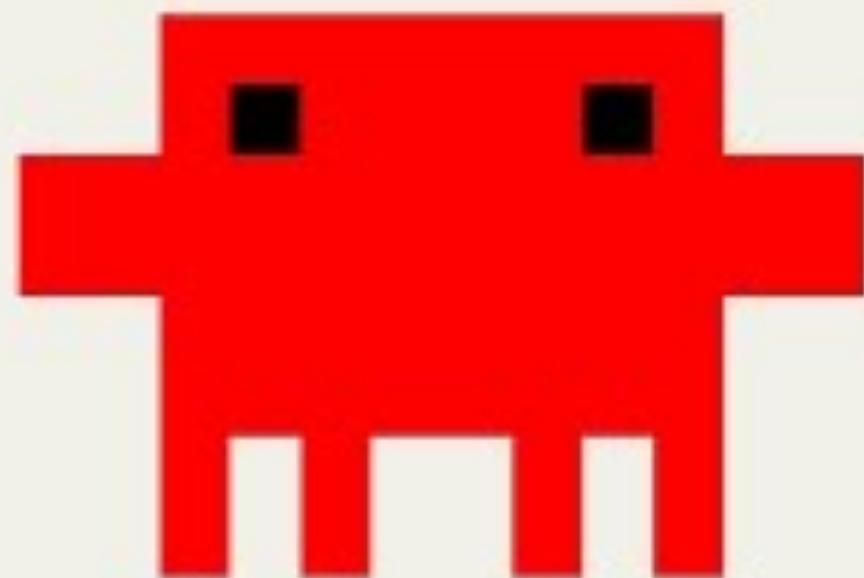
Copy Reply 10



Claude can make mistakes. Please double-check responses.

Reply to Claude...

Claude 1.5 Soraat -



Last edited just now

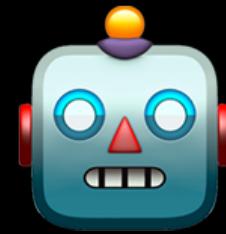
Copy

"out of the dust
from the ground"





```
def greet(name):  
    return f"Hello, {name}!"  
  
print(greet("Alice"))
```



Time to be the
computer

👉 # get ready!

do_clap(3)

do_slam(2)

do_clap(3)

```
# get ready!
👉 do_clap(3)
do_slam(2)
do_clap(3)
```

```
# get ready!
do_clap(3)
👉 do_slam(2)
do_clap(3)
```

```
# get ready!
do_clap(3)
do_slam(2)
👉 do_clap(3)
```

Let's define a function: jamslam

```
def jamslam():  
    do_slam(2)  
    do_clap(1)
```

```
def jamslam():
    do_slam(2)
    do_clap(1)
```

👉 # get ready!

```
jamslam()
```

```
jamslam()
```

```
jamslam()
```

done!

```
def jamslam():
    do_slam(2)
    do_clap(1)
```

get ready!

👉 jamslam()

jamslam()

jamslam()

done!

```
def jamslam():
    ➤ do_slam(2)
    do_clap(1)
```

```
# get ready!
jamslam()
jamslam()
jamslam()
# done!
```

```
def jamslam():
    do_slam(2)
    ➤ do_clap(1)
```

```
# get ready!
jamslam()
jamslam()
jamslam()
# done!
```

```
def jamslam():
    do_slam(2)
    do_clap(1)
```

get ready!

jamslam()

👉 jamslam()

jamslam()

done!

```
def jamslam():
    ➤ do_slam(2)
    do_clap(1)
```

```
# get ready!
jamslam()
jamslam()
jamslam()
# done!
```

```
def jamslam():
    do_slam(2)
    ➤ do_clap(1)
```

```
# get ready!
jamslam()
jamslam()
jamslam()
# done!
```

```
def jamslam():
    do_slam(2)
    do_clap(1)
```

get ready!

```
jamslam()
```

```
jamslam()
```

👉 jamslam()

done!

```
def jamslam():
    ➤ do_slam(2)
    do_clap(1)
```

```
# get ready!
jamslam()
jamslam()
jamslam()
# done!
```

```
def jamslam():
    do_slam(2)
    ➤ do_clap(1)
```

```
# get ready!
jamslam()
jamslam()
jamslam()
# done!
```

```
def jamslam():
    do_slam(2)
    do_clap(1)
```

get ready!

```
jamslam()
```

```
jamslam()
```

```
jamslam()
```

👉 # done!

More on Functions

What is a Function?

- A function is a reusable block of code
- It performs a specific task
- Think of it as a mini-program within your program

Why Use Functions?

- Reusability: Write once, use many times
- Organization: Break complex problems into smaller parts
- Readability: Makes code easier to understand

Anatomy of a Function

```
def greet(name):  
    return f"Hello, {name}!"
```

- def: Keyword to define a function
- greet: Function name
- (name): Parameter(s)
- return: Sends a result back

How to Use a Function

- Define the function
- Call the function

```
# Defining the function
def add(a, b):
    return a + b

# Calling the function
result = add(3, 5)
print(result) # Output: 8
```

Function Parameters

- Inputs that a function can accept
- Allow functions to work with different data
- Like variables that store values for just that function

```
def multiply(x, y):  
    return x * y  
  
print(multiply(4, 7))      # Output: 28  
print(multiply(2, 3))      # Output: 6
```

Return Values

- The result a function gives back
- Can be any data type (int, float, str, etc.)
- Use the return keyword
- A function can return one value or multiple values

```
def divide(a, b):  
    return a / b  
  
result = divide(10, 2)  
print(result) # Output: 5.0
```

Functions without Return

- Not all functions need to return a value
- These functions perform actions
- No return statement needed

```
def greet(name):  
    print(f"Hello, {name}!")
```

```
greet("Alice") # Output: Hello, Alice!
```

Benefits of Using Functions

1. Code reuse
2. Easier debugging
3. Better organization
4. Improved readability
5. Easier collaboration with other programmers

Open your code notebook

Challenge: Mystery Function

```
def mystery(x, y):  
    return # your solution here, a single line formula, no conditionals  
  
print(mystery(2, 3)) # 4  
print(mystery(5, 2)) # 10  
print(mystery(3, 3)) # 6  
print(mystery(1, 1)) # 2
```

Go Time



- Focus on Code
- Ask for Help/Feedback

