Lecture 2: Scheme

Kaleel Mahmood
Department of Computer Science and Engineering
University of Connecticut

Teaching Assistant Office Hours

	Monday	Tuesday	Wednesday	Thursday	Friday
9	Michael 9-11		Michael 9-11		
10				Jacob 10-12	Andy 10-12
11				Akul 11-2	
12				Samantha 12-2	Akul 12-2
1	Marquis, Kaustubh 1-4		Andy 1-3	Adrienne 1-3	Michael 2-3
2					
3			Adrienne 3-5	Marquis 3-5	Kaustubh 3-4
4	Samantha, Rachel 4-6		Rachel 4-6		
5					
6	Matthew, Jacob 6-8	Matthew 6-8			
7					
8					

Where are the office hours? 1st floor of ITE.

Lecture Overview

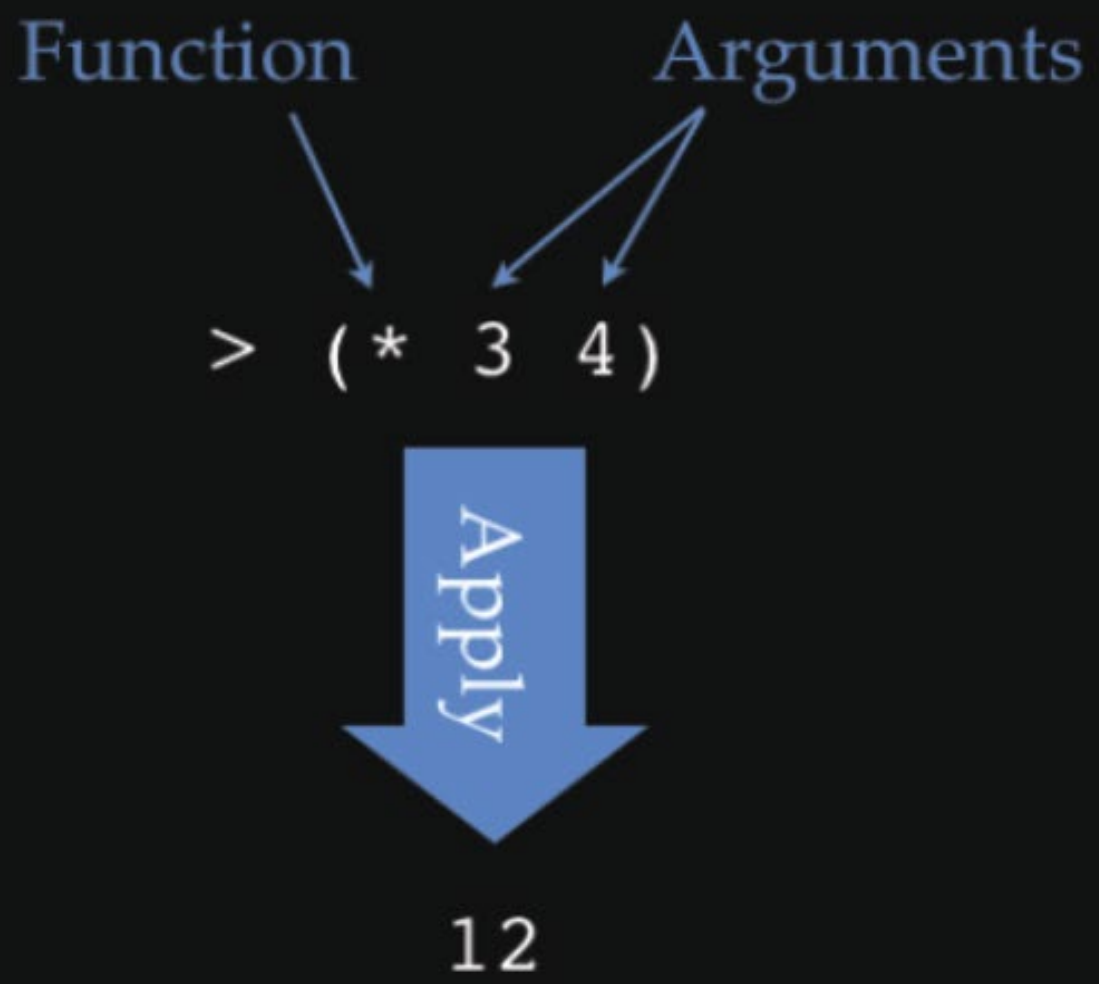
Basic Scheme Syntax

Understanding the Scheme Interpreter

Variables and the Environment

Basic Scheme Syntax

THE BASIC COMPUTATIONAL STEP: *APPLY* A FUNCTION TO ARGUMENTS



PREFIX NOTATION

- You are used to various notations for mathematical function application:
  - **prefix** notation: f(a, b) . The name of the function, f, appears *before* its arguments.
  - **infix** notation: a + b. The name of the function, +, appears *between* the arguments.
  - **postfix** notation (a la HP calculators). Here the name of the function appears *after* the arguments.
- Note that prefix & postfix are naturally more suited for functions that do not take two arguments.

SCHEME uses prefix notation

SCHEME IS AN *INTERPRETED* LANGUAGE

- Scheme programs are built during a "dialog" with the *interpreter*.
- The interpreter's job is to
  - ***evaluate scheme expressions***, and
  - ***maintain an "environment"*** of bindings of variable to values.

EVALUATION IN DETAIL: ATOMIC OBJECTS

- A *number*, like 7, evaluates to...itself!
- A *function*, like +, evaluates to...itself!

So...

EXAMPLES

```
> 7
7
```

expression
value

```
> 11
11
```

expression
value

```
> +
<procedure:+>
```

expression
value

```
> *
#<procedure:*>
```

expression
value

EVALUATION IN DETAIL: COMPOUND OBJECTS

- In SCHEME, "compound" expressions have the form

```
(<f> <a1> <a2> ... <an>)
```

- To evaluate a compound expression, the interpreter
  - evaluates each sub-expression, and, then
  - applies the first, *which must be a function*, to the rest *which must be arguments to the function*

EXAMPLES

```
> 7
7
```

expression
value

```
> 11
11
```

expression
value

```
> +
<procedure:+>
```

expression
value

```
> *
#<procedure:*>
```

expression
value

```
> (+ 7 8)
15
```

expression
value

```
> (* 18 11)
198
```

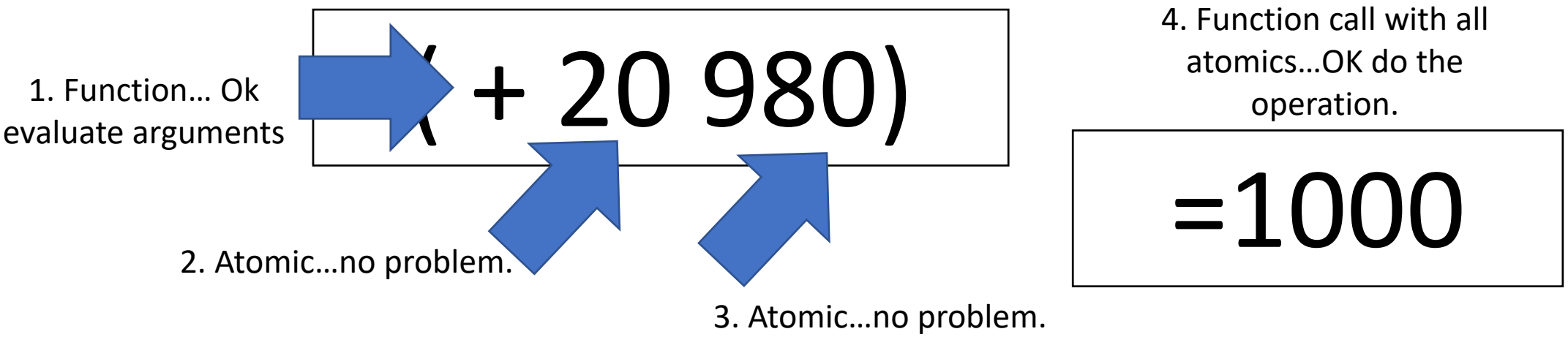
expression
value

This is the new part!
Argument *evaluation*
followed by
Function *application*

Understanding the Scheme Interpreter

Understand How the Scheme Interpreter Works

- Simplest case: One function call, all arguments to the function are atomics.
- Think like reading an English book, *read left to right.*



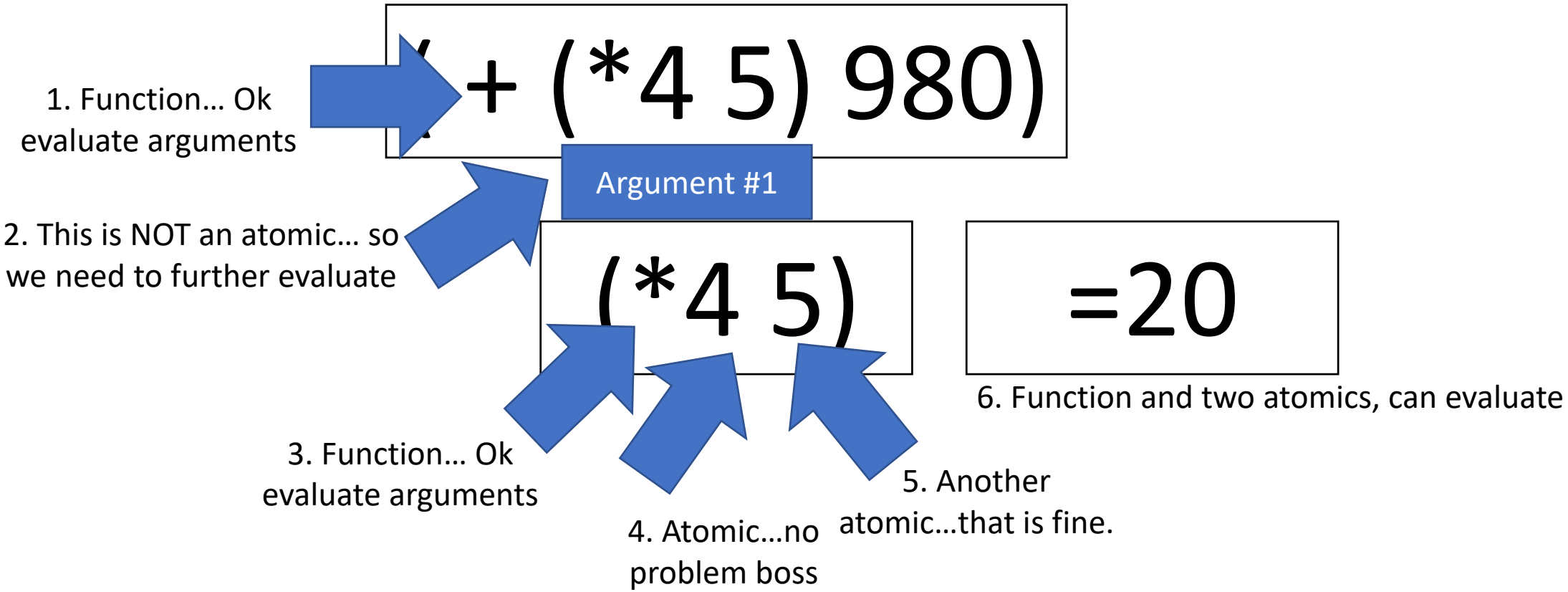
Understand how the Scheme Interpreter Works

~~(+ 20 980)~~

(+ (*4 5) 980)

Think like reading an English book, *read left to right.*
Until you have to recur…then take a step down and keep reading left to right.

Understand how the Scheme Interpreter Works



Understand how the Scheme Interpreter Works

(+ (*4 5) 980)

Argument #1

=20

7. Go to next argument. Its an atomic. So no need to recurse.

6. We have evaluated argument 1 and gotten an atomic. Great.

Understand how the Scheme Interpreter Works

(+ (*4 5) 980)

=1000

Argument #1

=20

8. Two atomics for the argument so we can evaluate!

7. Go to next argument. Its an atomic. So no need to recurse.

6. We have evaluated argument 1 and gotten an atomic. Great.

Take a bite of victory…

Some Questions to Ponder…

- What if instead of this: (+ (*4 5) 980)

- We have this: (+ (*4 5) (* 4 245))

Question 1: Will the answer be the same? Can you analyze it recursively?

Question 2: What would happen if I flip the order? (+ (* 4 245) (*4 5))

Even More Questions to Ponder…

Q3. Can we have multiple (more than two) arguments to a function? E.g.

(+ 5 6 7 8) Hint: The answer is yes for arithmetic functions.

Q4. Think about your old friend Python. If you coded 3*5+4 in Python and did the *exact* same operation in Scheme without respect to order of operations would they have the same result?

Hint: Here is the Scheme expression (* 3 (+ 5 4))

BINDING VARIABLES TO VALUES

- Our first example of abstraction.
- Involves a **special form**: define. (Called "special" because it breaks the regular evaluation rule.)
- The form

```
(define <variable> <value>)
```

- binds the variable <variable> to the value obtained by evaluating <expression>.

So...

BINDING: EXAMPLES

```
> (define a 6)
```

DEFINING NEW FUNCTIONS

- The define special form can also be used to define new *functions*.
- The syntax is the following:

```
(define (<function-name> <argument-var>)
        <body>)
```

```
(define (inc x) (+ x 1))

> (inc 1)
2
> (inc 2)
3
> (inc (inc 5))
```

Defines the "increment" function:

`inc(x) = x+1`

MORE EXAMPLES OF FUNCTION DEFINITION

```
(define (inc x) (+ x 1))
```

Increment. inc(x) = x+1.

```
(define (dec x) (- x 1))
```

Decrement. dec(x) = x-1.

```
(define (square x) (* x x))
```

Square. square(x) = x^2.

```
(define (sum-of-squares x y)
        (+ (square x) (square y)))
```

- This last function, sum-of-squares, is a function of two variables:

$$\texttt{sum-of-squares(x,y)} = x^2 + y^2.$$

Variables and the Environment

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
(define spiderman 3)


(define (kaleelFunction x)
  (define spiderman 6)
  (+ spiderman x)
  )

(kaleelFunction 2)

spiderman

```

First we define our variable "spiderman"

Next we write the "kaleelFunction" which is very complicated. It defines the variable spiderman (again) AND add spiderman to a number and returns it.

Call the kaleelFunction with 2

Check the value of spiderman

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
(define spiderman 3)


(define (kaleelFunction x)
  (define spiderman 6)
  (+ spiderman x)
  )

(kaleelFunction 2)

spiderman

```

Question: What is the value of the variable spiderman?

Answer:

Why does spiderman=3? For the same reason three DIFFERENT actors all play spiderman in the spiderman movies...

The spiderman (variable) is defined based on what movie (environment) he is in.

Spiderman can be different people in different movies…variables with the same name can have different values in different environments.

What is the value of Spiderman? Well it depends on the environment…

The technical answer: When you declare variables in Scheme they have a *Lexical* scope. When you call a method a new environment is created and if variables share a name…the variable is temporarily replaced. See next slide for an example.

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
(define spiderman 3)


(define (kaleelFunction x)
  (define spiderman 6)
  (+ spiderman x)
  )

(kaleelFunction 2)

spiderman
```

GLOBAL Environment

spiderman = 3

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
1   (define spiderman 3)
2
3
4   (define (kaleelFunction x)
5     (define spiderman 6)
6     (+ spiderman x)
7     )
8
9   (kaleelFunction 2)
10
11  spiderman
12
```

GLOBAL Environment

spiderman = 3

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
1  (define spiderman 3)
2
3
4  (define (kaleelFunction x)
5    (define spiderman 6)
6    (+ spiderman x)
7    )
8
9  (kaleelFunction 2)
10
11 spiderman
12
```

GLOBAL Environment

spiderman = 3

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
1   (define spiderman 3)
2
3
4   (define (kaleelFunction x)
5     (define spiderman 6)
6     (+ spiderman x)
7     )
8
9   (kaleelFunction 2)
10
11  spiderman
12
```

GLOBAL Environment

spiderman = 3

Method Environment

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
1   (define spiderman 3)
2
3
4   (define (kaleelFunction x)
5     (define spiderman 6)
6     (+ spiderman x)
7     )
8
9   (kaleelFunction 2)
10
11  spiderman
12
```

GLOBAL Environment

spiderman = 3

Method Environment

x =2

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
(define spiderman 3)


(define (kaleelFunction x)
  (define spiderman 6)
  (+ spiderman x)
  )

(kaleelFunction 2)

spiderman
```

GLOBAL Environment

spiderman = 3

Method Environment

x =2

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
(define spiderman 3)


(define (kaleelFunction x)
  (define spiderman 6)
  (+ spiderman x)
  )

(kaleelFunction 2)

spiderman
```

GLOBAL Environment

spiderman = 3

Method Environment

x =2

spiderman = 6

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
1   (define spiderman 3)
2
3
4   (define (kaleelFunction x)
5     (define spiderman 6)
6     (+ spiderman x)
7     )
8
9   (kaleelFunction 2)
10
11  spiderman
12
```

GLOBAL Environment

spiderman = 3

Method Environment

x =2

spiderman = 6

return 8

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
1  (define spiderman 3)
2
3
4  (define (kaleelFunction x)
5    (define spiderman 6)
6    (+ spiderman x)
7    )
8
9  (kaleelFunction 2)
10
11 spiderman
12
```

GLOBAL Environment

spiderman = 3

Method Environment

x =2

spiderman = 6

return 8

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
1  (define spiderman 3)
2
3
4  (define (kaleelFunction x)
5    (define spiderman 6)
6    (+ spiderman x)
7    )
8
9  (kaleelFunction 2)
10
11 spiderman
12
```

GLOBAL Environment

spiderman = 3

Understanding Variables and the Environment

Alternative title: How I will explain environment variables to you using, "Spiderman No Way Home 3."

```
(define spiderman 3)


(define (kaleelFunction x)
  (define spiderman 6)
  (+ spiderman x)
  )

(kaleelFunction 2)

spiderman
```

GLOBAL Environment

spiderman = 3

Do you miss Python yet?

```python
spiderman = 3

def kaleelfunction(x):
    spiderman = 6
    return x + spiderman


kaleelfunction(2)
print(spiderman)
```

What would happen in Python?

FUNCTION DEFINITION: TERMINOLOGY

- Consider `(define (f x) <body>)`. Here
  - `f` is the function *name*,
  - `x` is the *formal parameter*, and
  - `<body>` is the *body* of the function, which usually contains references to the variable x (the parameter).
- When the function is called (e.g., `(f 8)`), the value it is called upon (8, in this case) is the *actual parameter*.
- The word *argument* is also used, for both the formal and actual parameter.

Figure Sources

- Professor Greg Johnson's CSE 1729 Lecture Slides
- https://linkslab.uconn.edu/wp-content/uploads/sites/246/2013/11/ITE.jpg
- https://imgflip.com/memetemplate/178495100/uh-oh-Gru
- http://c718858.r58.cf2.rackcdn.com/catalog/product/cache/6/image/9df78eab33525d08d6e5fb8d27136e95/i/s/iss_research_oh_yeah_victory_bar_12_box.jpg
- https://racket-lang.org/img/racket-logo.svg
- https://static.wikia.nocookie.net/batman/images/a/a4/The_Riddler_%28BF%29.jpg/revision/latest/scale-to-width-down/350?cb=20120718050811
- https://ichef.bbci.co.uk/news/800/cpsprodpb/5198/production/_108388802_spider-man.jpg
- https://i0.wp.com/screengeek.net/wp-content/uploads/2021/11/spider-man-no-way-home.jpg
- https://media.istockphoto.com/vectors/drop-icon-on-white-background-water-icon-vector-vector-id967219304?k=20&m=967219304&s=612x612&w=0&h=_rIZiXKXzqxfB9ULc9ArvTNsk9QnZSj3CSijPGoYAp8=
- https://miro.medium.com/max/2000/1*FZWJzoMSaaMsE2qEb-jT9A.jpeg