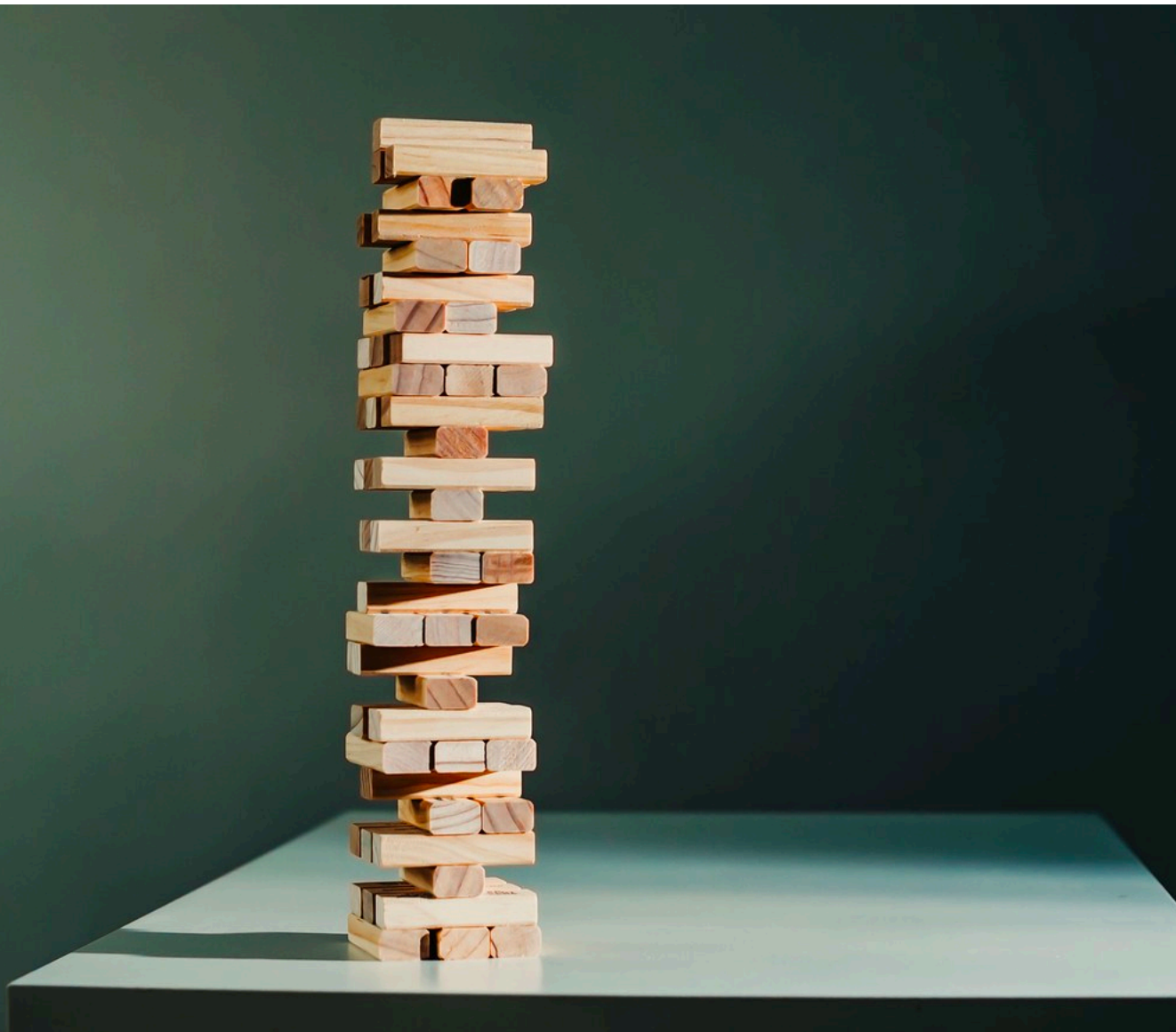# A ROBOT TEACHES ITSELF TO PLAY JENGA. BUT THIS IS NO GAME
www.wired.com/story/a-robot-teaches-itself-to-play-jenga

"Researchers at MIT report today in Science Robotics that they've engineered a robot to teach itself the complex physics of Jenga. This, though, is no game—it's a big step in the daunting quest to get robots to manipulate objects in the real world.

The researchers equipped an industrial robot arm with a force sensor in its wrist and a two-pronged manipulator, and sat it down in front of a Jenga tower. The robot got its sense of sight from a camera trained on the tower.

But the researchers didn't teach it how to win against a human. Instead, the researchers asked the robot to do some exploring, probing blocks at random.

This Jenga bot is also signaling a shift in how some robots learn. "

Working with Lists

```
>>> digits = [1, 8, 2, 8]
```
The number of elements

```
>>> len(digits)
4
```
An element selected by its index
```
>>> digits[3]
8
```

Concatenation and repetition

```
>>> [2, 7] + digits * 2
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]
```

Nested lists

```
>>> pairs = [[10, 20], [30, 40]]
>>> pairs[1]
[30, 40]
>>> pairs[1][0]
30
```

```
>>> digits = [2//2, 2+2+2+2, 2, 2*2*2]
```

```
>>> getitem(digits, 3)
8
```

```
>>> add([2, 7], mul(digits, 2))
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]
```

Containers

Built-in operators for testing whether an element appears in a compound value

```
>>> digits = [1, 8, 2, 8]
>>> 1 in digits
True
>>> 8 in digits
True
>>> 5 not in digits
True
>>> not(5 in digits)
True
```

(Demo)

For Statements

（Demo）

Sequence Iteration

```
def count(s, value):
    total = 0
    for element in s:




        if element == value:
            total = total + 1
    return total
```

> Name bound in the first frame
> of the current environment
> (not a new frame)

For Statement Execution Procedure

```
for <name> in <expression>:

        <suite>
```

1. Evaluate the header `<expression>`, which must yield an iterable value (a sequence)

2. For each element in that sequence, in order:

   A. Bind `<name>` to that element in the current frame

   B. Execute the `<suite>`

Sequence Unpacking in For Statements

> A sequence of
> fixed-length sequences

```
>>> pairs = [[1, 2], [2, 2], [3, 2], [4, 4]]

>>> same_count = 0
```

> A name for each element in a
> fixed-length sequence

Each name is bound to a value, as in
multiple assignment

```
>>> for x, y in pairs:
...         if x == y:
...             same_count = same_count + 1

>>> same_count
2
```
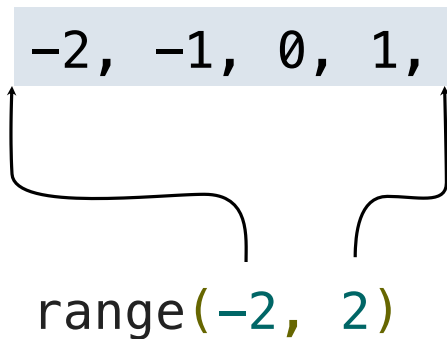
Ranges

The Range Type

A range is a sequence of consecutive integers.*

$$..., -5, -4, -3, \boxed{-2, -1, 0, 1,} 2, 3, 4, 5, ...$$

range(-2, 2)

Length: ending value – starting value

(Demo)

Element selection: starting value + index

```
>>> list(range(-2, 2))
[-2, -1, 0, 1]
```
List constructor

```
>>> list(range(4))
[0, 1, 2, 3]
```
Range with a 0 starting value

* Ranges can actually represent more general integer sequences.

List Comprehensions

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p']
>>> [letters[i] for i in [3, 4, 6, 8]]
```

['d', 'e', 'm', 'o']

List Comprehensions

[`<map exp>` `for` `<name>` `in` `<iter exp>` `if` `<filter exp>`]

Short version: [`<map exp>` `for` `<name>` `in` `<iter exp>`]

A combined expression that evaluates to a list using this evaluation procedure:

1. Add a new frame with the current frame as its parent

2. Create an empty *result list* that is the value of the expression

3. For each element in the iterable value of `<iter exp>`:

   A. Bind `<name>` to that element in the new frame from step 1

   B. If `<filter exp>` evaluates to a true value, then add the value of `<map exp>`
      to the result list

Strings

Strings are an Abstraction

Representing data:

      `'200'`   `'1.2e-5'`   `'False'`   `'[1, 2]'`

Representing language:

```
"""And, as imagination bodies forth
The forms of things unknown, the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""
```

Representing programs:

      `'curry = lambda f: lambda x: lambda y: f(x, y)'`

(Demo)

String Literals Have Three Forms

```
>>> 'I am string!'
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> '您好'
'您好'

>>> """The Zen of Python
claims, Readability counts.
Read more: import this."""
'The Zen of Python\nclaims, Readability counts.\nRead more: import this.'
```

Single-quoted and double-quoted strings are equivalent

A backslash "escapes" the following character

"Line feed" character represents a new line

Dictionaries

```
{'Dem': 0}
```

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)

- Two **keys cannot be equal;** There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value