Computational Thinking

Lecture 1: Introduction

University of Engineering and Technology

VIETNAM NATIONAL UNIVERSITY HANOI

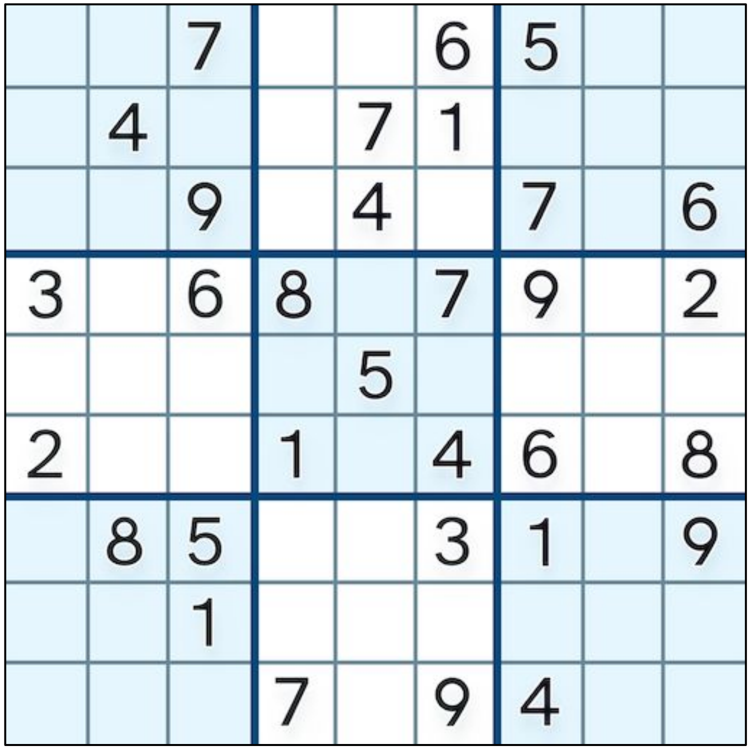
Outline

- Real-life Examples
- Problem-solving Thinking
- What is Computational Thinking?
- What is Programming?
- Getting Started with Python

Real-life Examples

Single-Player Games

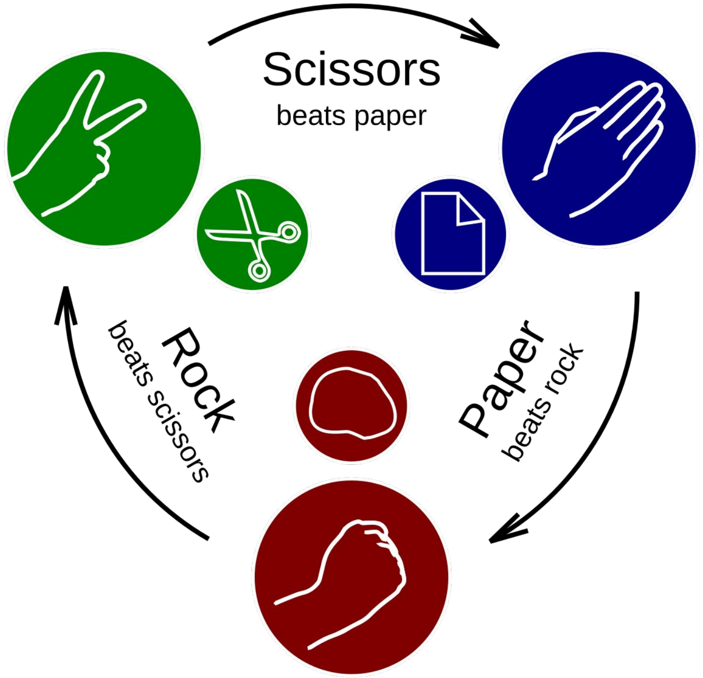
Tetris



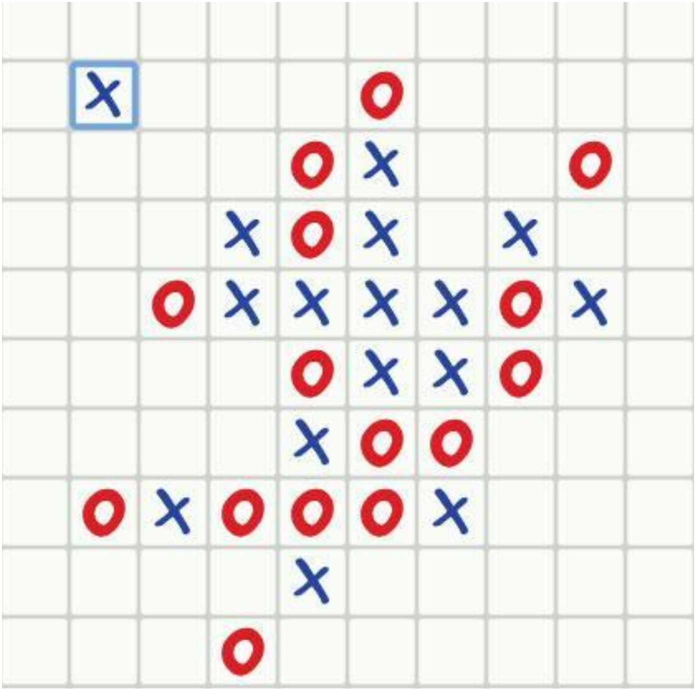
		7			6	5		
	4			7	1			
		9		4		7		6
3		6	8		7	9		2
				5				
2			1		4	6		8
	8	5			3	1		9
		1						
			7		9	4		

Sudoku

Multiplayer Games



Rock Paper Scissors

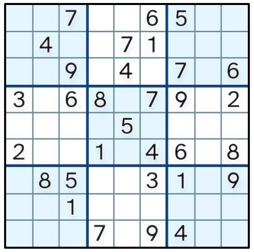


Tic Tac Toe

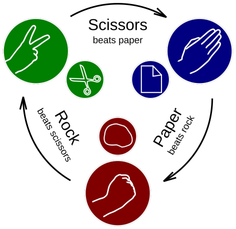
The notion of ‘Problem’

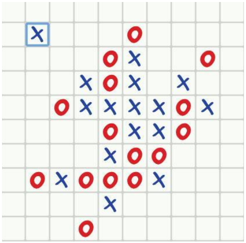
Problem

(to be solved)



		7			6	5		
	4			7	1			
		9		4		7		6
3		6	8		7	9		2
				5				
2			1		4	6		8
	8	5			3	1		9
		1						
			7		9	4		





	X				O				
				O	X			O	
			X	O	X		X		
		O	X	X	X	X	O	X	
				O	X	X	O		
				X	O	O			
	O	X	O	O	O	X			
				X					
			O						

Win the game!

(Optimal solution)

Thinking

(Decomposition, Pattern Recognition, Abstraction, Algorithm, etc.)

Real-life Problem – Expense Management

Tiền điện
Tiền ăn
Du lịch
Tiền nhà
Tiền nước
Sách
Học phí
Mua sắm

Problem: You have **X** million VND/month to cover your living cost in Hanoi.

Factors: Unexpected expenses, necessity.

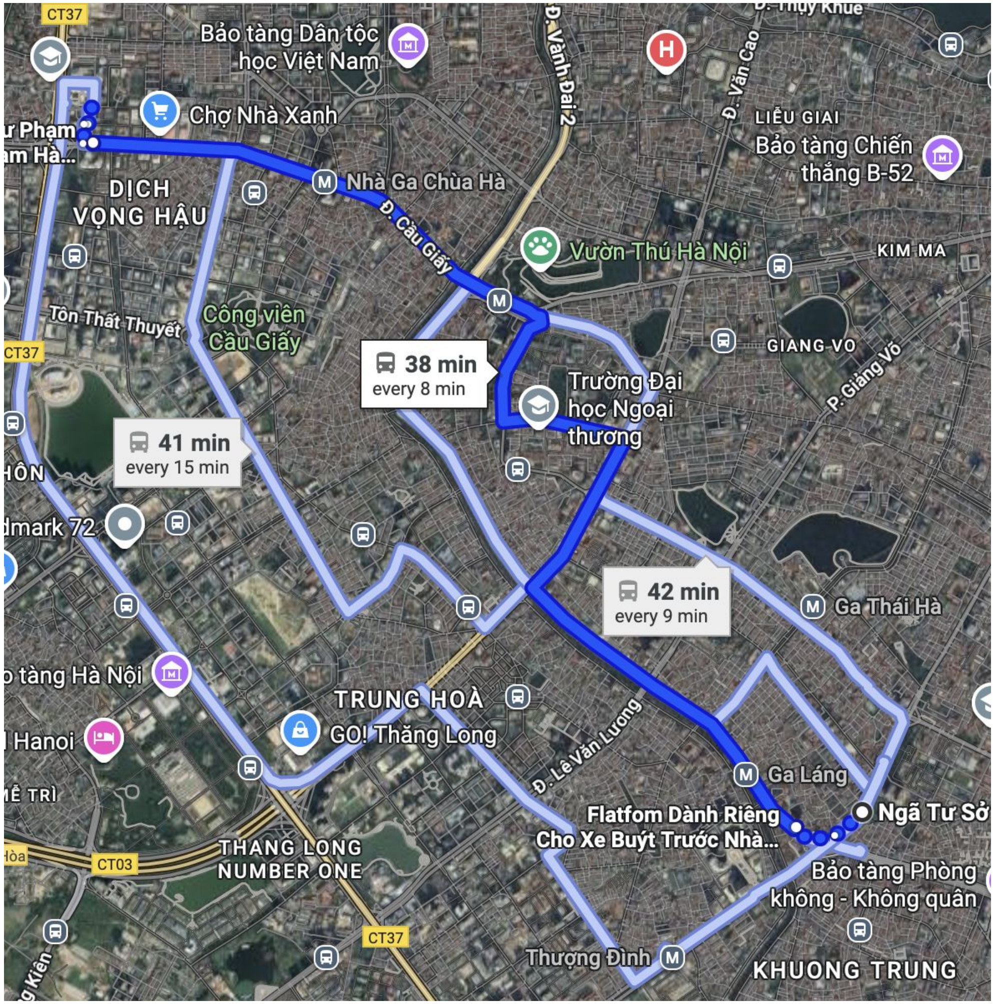
Algorithm?

Real-life Problem – Optimal Route Finding

Problem: Find an optimal route from location A (e.g.,"Nga Tu So") to location B (e.g., VNU-UET)

Factors: Traffic jams, weather, transportation means.

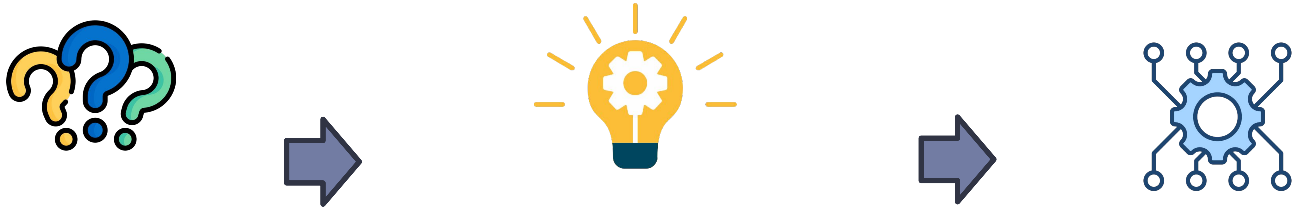
Algorithm?



Problem-solving Thinking

Problem-solving Thinking

Problem-solving thinking is the process of **understanding a problem**, exploring **possible solutions**, and **designing a clear, step-by-step method (algorithm)** to solve it.



Problem

A question or situation that needs to be solved.

Solution Idea

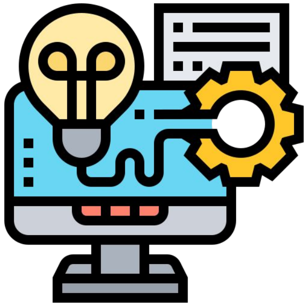
A general idea to reach the goal.

Algorithm

A clear, step-by-step procedure or instructions

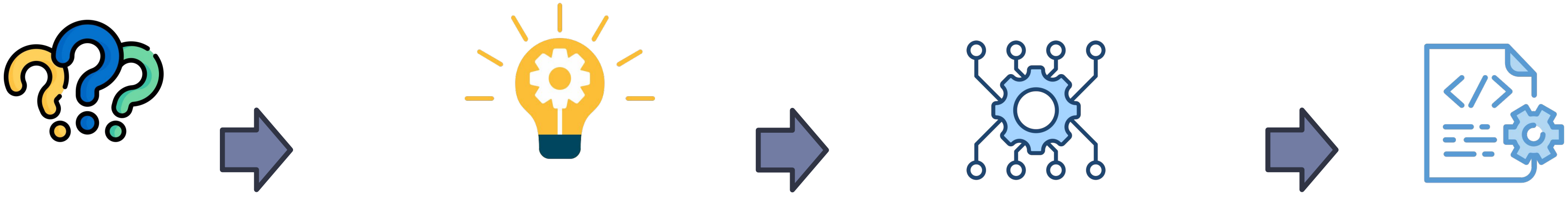
The Power of Computers in Supporting Problem-solving

Vs.



Aspect	Human Brain	Computer
Creativity	Creative, intuitive, can imagine new ideas	No creativity, only follows instructions
Problem Size	Handles only small-scale or simple problems effectively	Can solve very large, complex problems with big data
Speed	Slow with large-scale calculations	Extremely fast with millions of operations
Accuracy	Prone to errors, distraction, fatigue	Always precise, consistent, no fatigue

Problem-solving with Computational Thinking



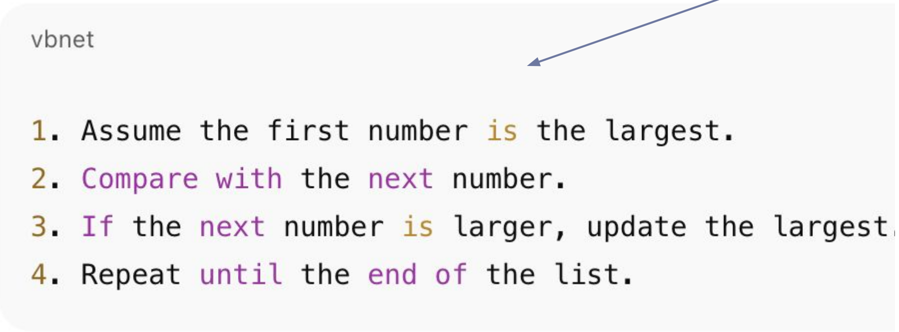
Problem → **Solution Idea** → **Algorithm** → **Code**

Problem: Given a list of numbers, find the largest one.

Solution Idea: Value comparison

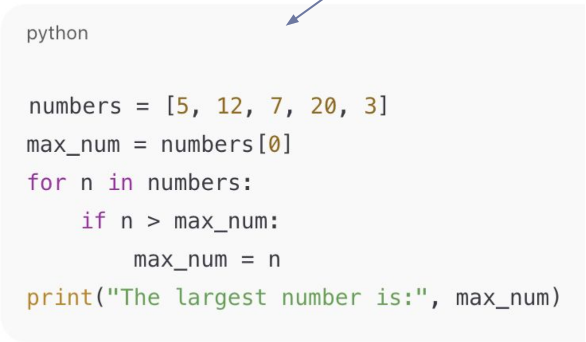
Algorithm: Instructions via Pseudo code

Code: Python implementation



vbnet

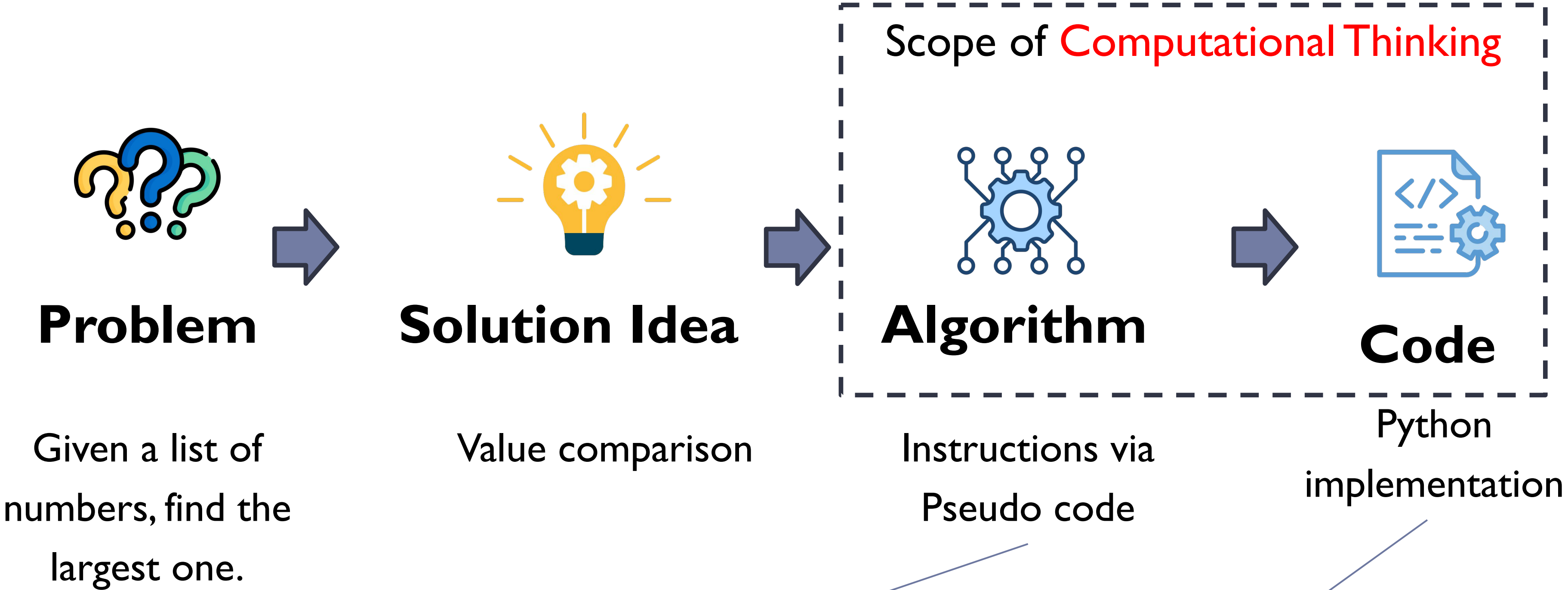
```
1. Assume the first number is the largest.
2. Compare with the next number.
3. If the next number is larger, update the largest
4. Repeat until the end of the list.
```



python

```python
numbers = [5, 12, 7, 20, 3]
max_num = numbers[0]
for n in numbers:
    if n > max_num:
        max_num = n
print("The largest number is:", max_num)
```

Problem-solving with Computational Thinking



Scope of Computational Thinking

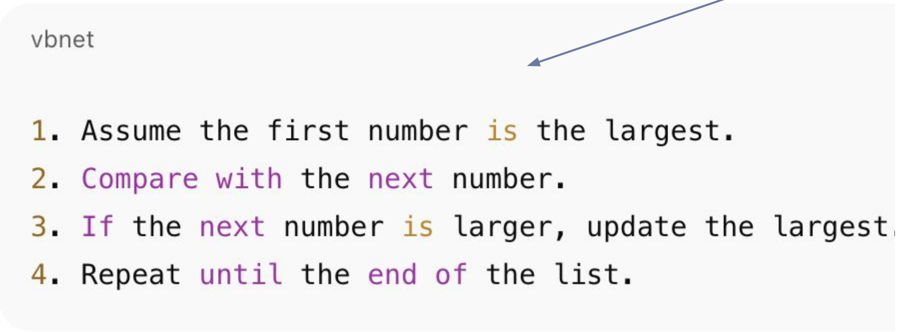
Problem → **Solution Idea** → **Algorithm** → **Code**

Given a list of numbers, find the largest one.

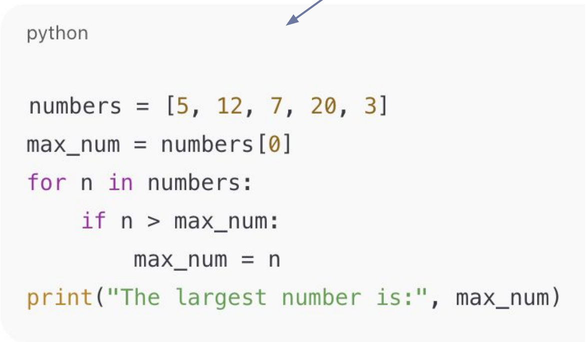
Value comparison

Instructions via Pseudo code

Python implementation



```
1. Assume the first number is the largest.
2. Compare with the next number.
3. If the next number is larger, update the largest.
4. Repeat until the end of the list.
```

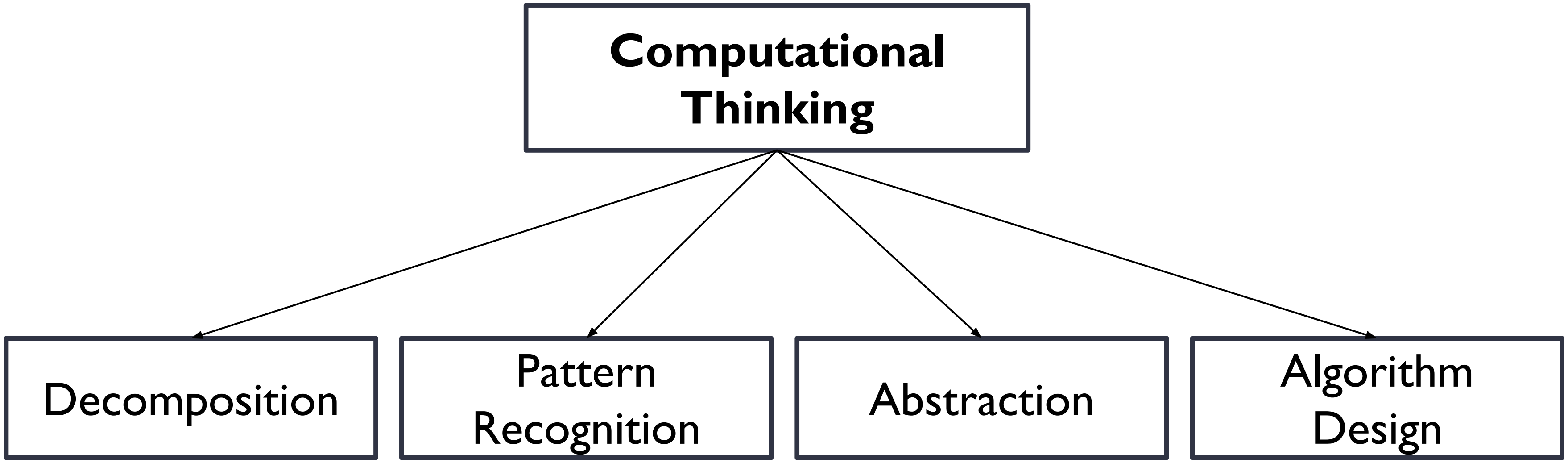


```python
numbers = [5, 12, 7, 20, 3]
max_num = numbers[0]
for n in numbers:
    if n > max_num:
        max_num = n
print("The largest number is:", max_num)
```

What is Computational Thinking?

Computational Thinking

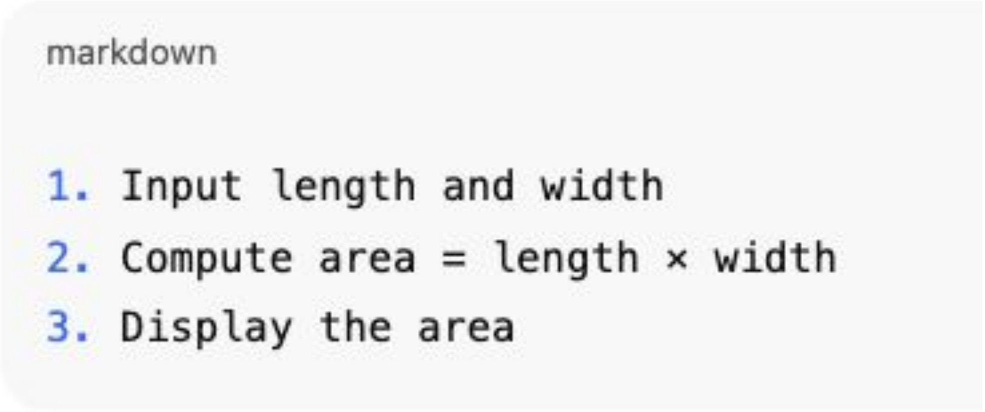
Computational Thinking (CT) is a problem-solving approach that uses concepts from computer science to design solutions which can be carried out by humans or computers.



Decomposition

Break a big problem into smaller, manageable parts.

Example: Write a program to calculate area of a rectangle



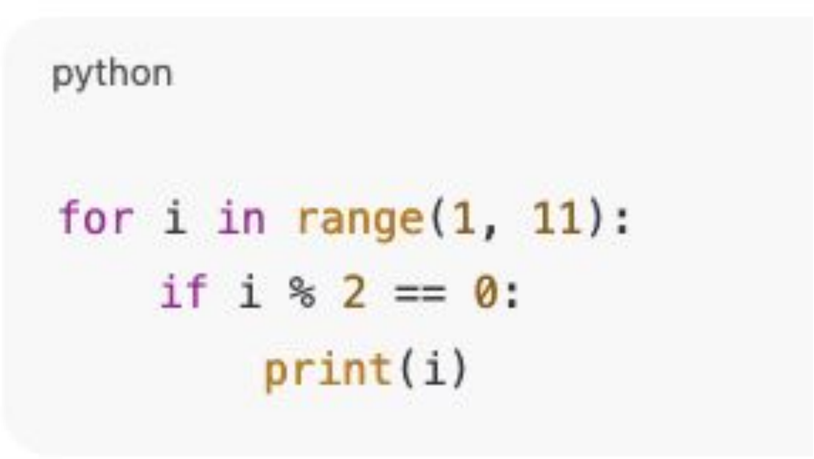
```
1. Input length and width
2. Compute area = length × width
3. Display the area
```

Decompose the big problem → smaller steps are easier to solve.

Pattern Recognition

Find similarities or repeated elements.

Example: Print even numbers from 1–10



python

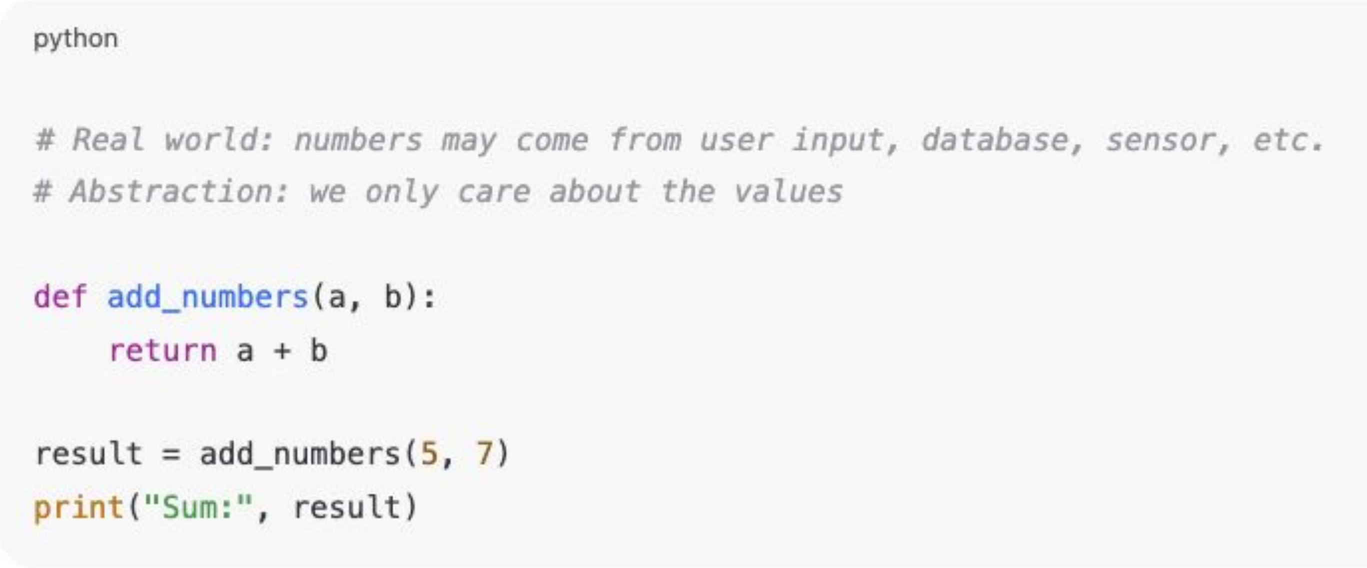
```python
for i in range(1, 11):
    if i % 2 == 0:
        print(i)
```

Patterns help us create general rules for many cases.

Abstraction

Focus on important details, ignore the irrelevant

Example: Write a function: Add two integer numbers.



python

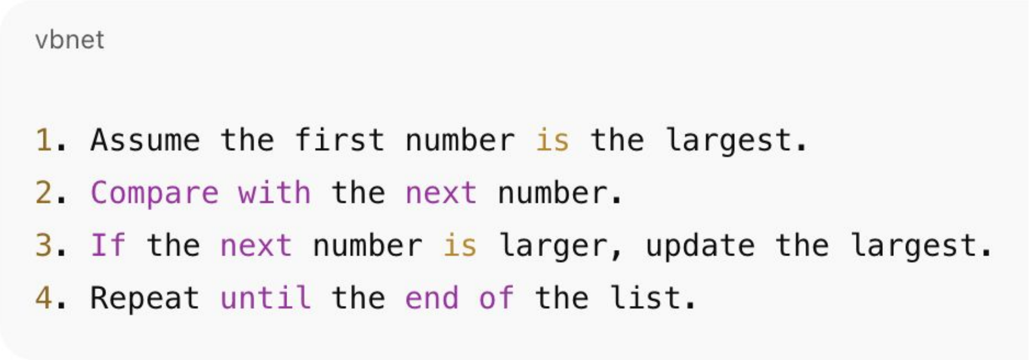
```python
# Real world: numbers may come from user input, database, sensor, etc.
# Abstraction: we only care about the values

def add_numbers(a, b):
    return a + b

result = add_numbers(5, 7)
print("Sum:", result)
```

Simplify complexity by showing only the useful details.

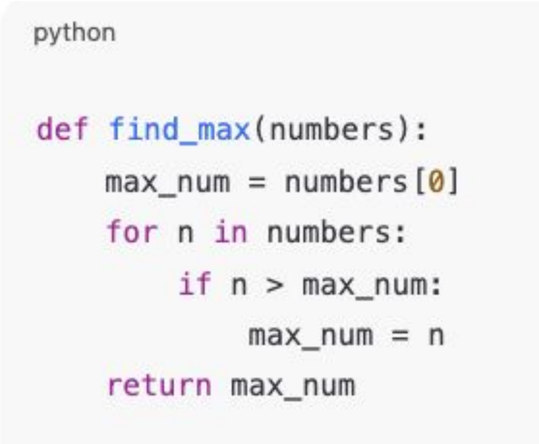
Algorithm Design

Create step-by-step instructions to solve the problem.

Example: Find the largest number in a given list



```
1. Assume the first number is the largest.
2. Compare with the next number.
3. If the next number is larger, update the largest.
4. Repeat until the end of the list.
```



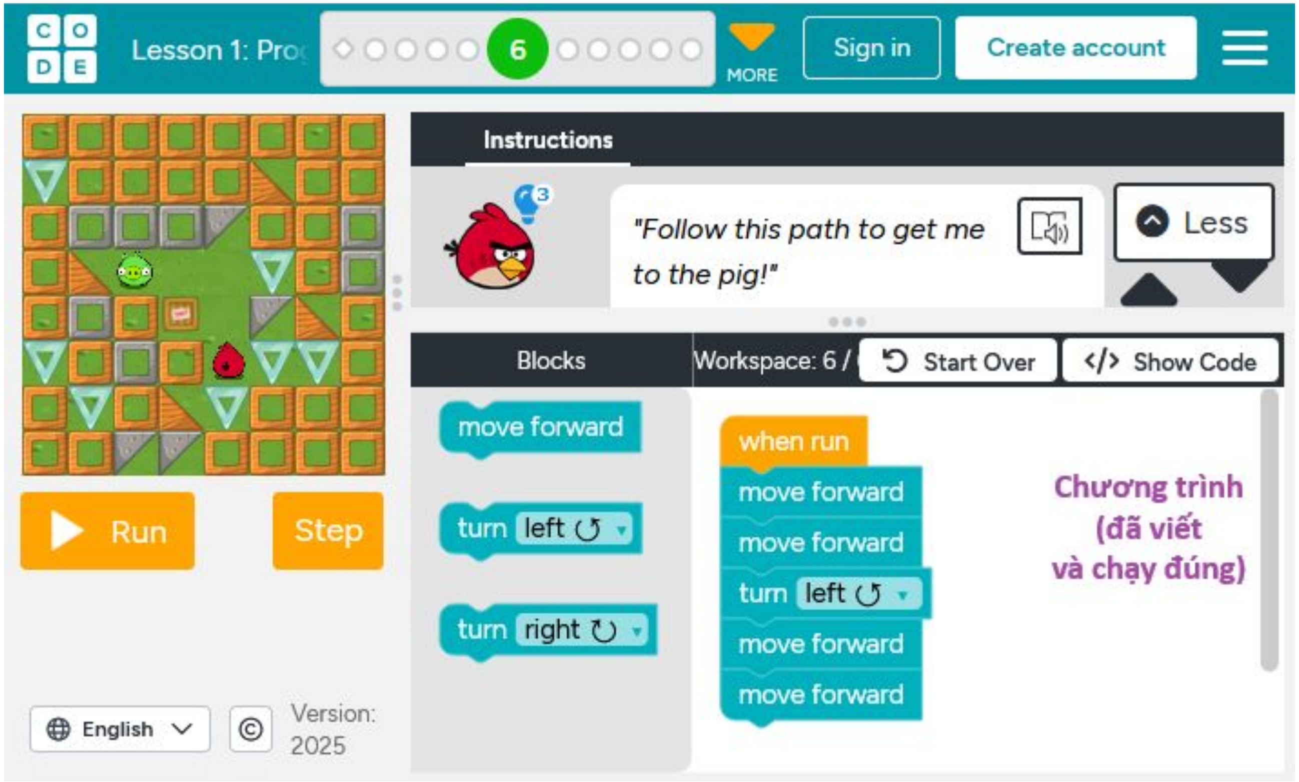
```python
def find_max(numbers):
    max_num = numbers[0]
    for n in numbers:
        if n > max_num:
            max_num = n
    return max_num
```

Algorithms are precise recipes to solve problems.

What is programming?

What is programming?

Given a set of instructions and a task, write a sequence of instructions that do the task.



"Follow this path to get me to the pig!"

```
when run
move forward
move forward
turn left ↺
move forward
move forward
```

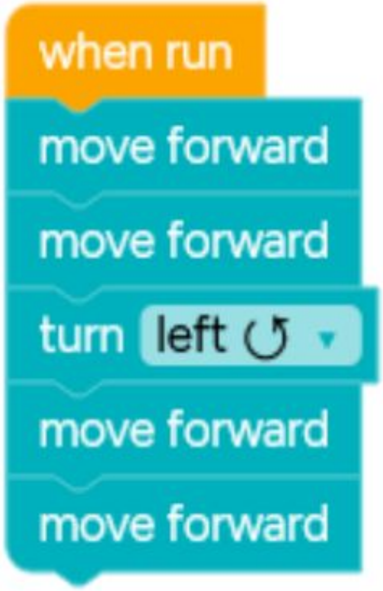
Chương trình (đã viết và chạy đúng)

This is Scratch at code.org Kids' games, actually

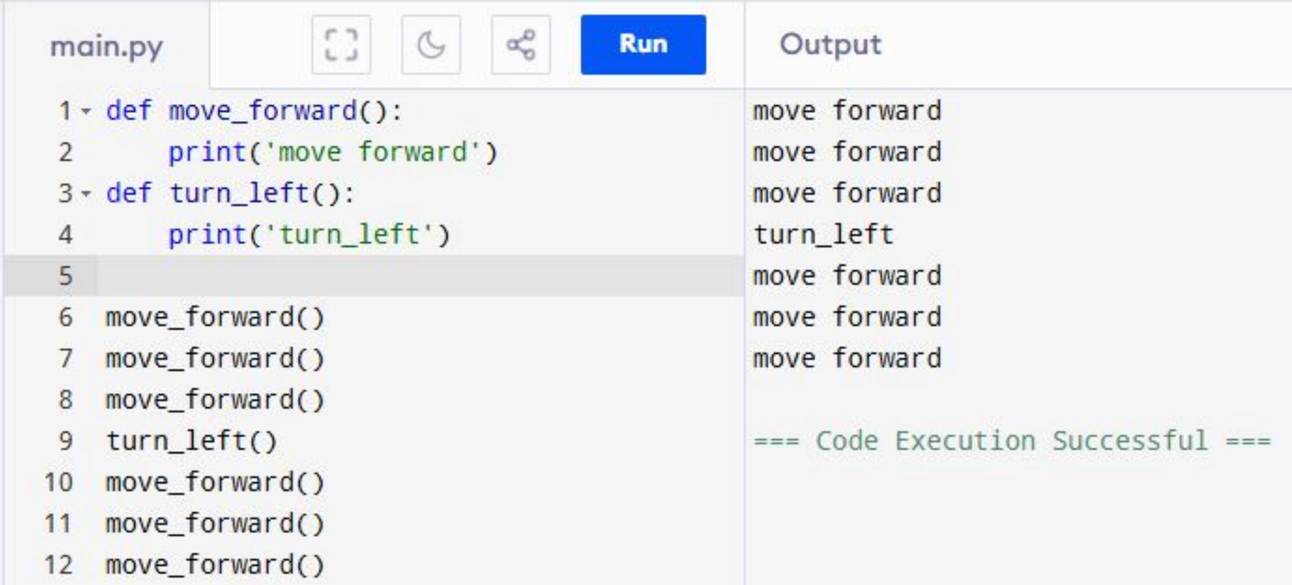
https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/6

What about programming in Python?

Same process: same task,
a different set of instructions



when run
move forward
move forward
turn left ↺
move forward
move forward



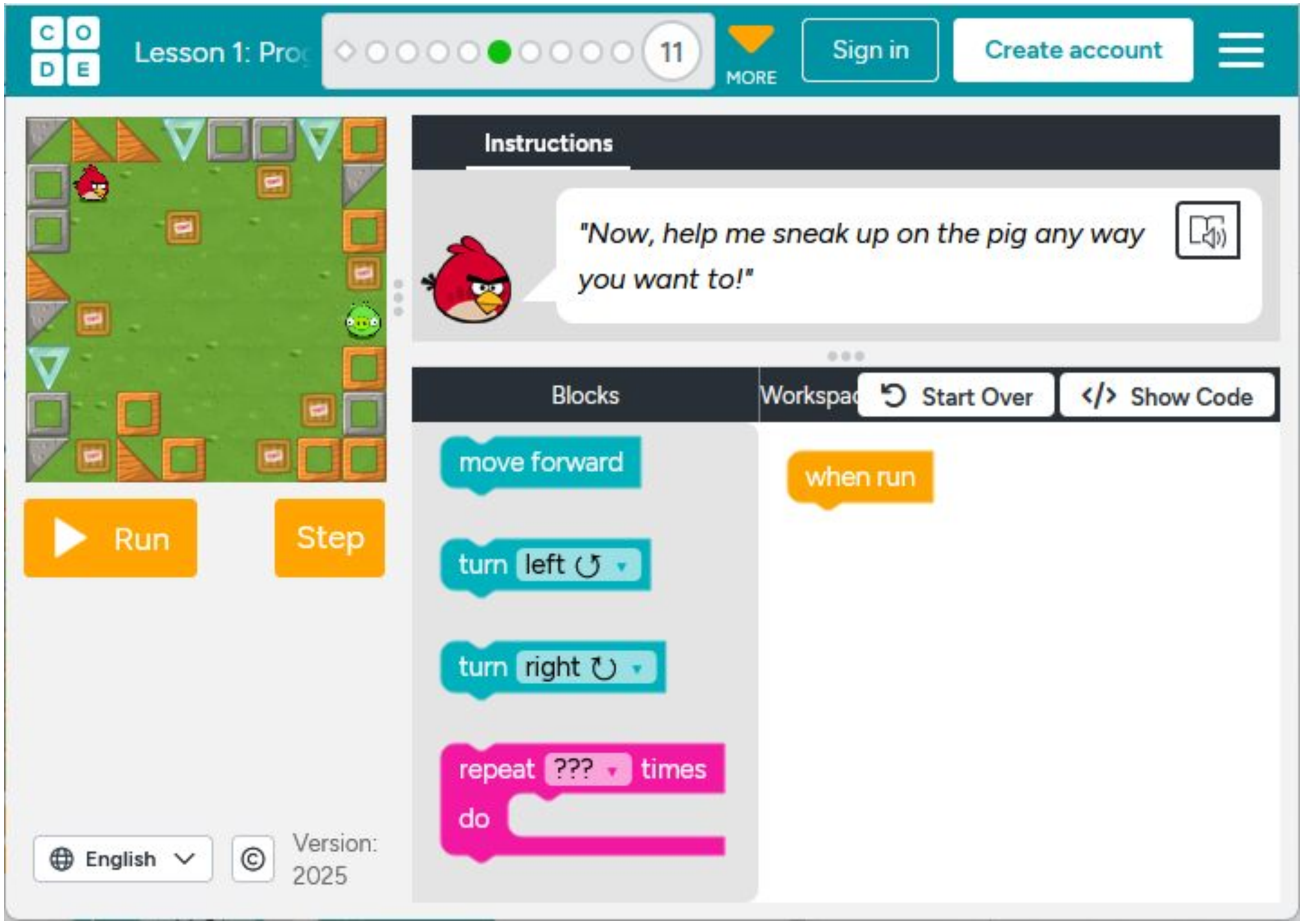
```python
def move_forward():
    print('move forward')
def turn_left():
    print('turn_left')

move_forward()
move_forward()
move_forward()
turn_left()
move_forward()
move_forward()
move_forward()
```

```
move forward
move forward
move forward
turn_left
move forward
move forward
move forward


=== Code Execution Successful ===
```

https://www.programiz.com/python-programming/online-compiler/

More complicated tasks?

Your turn.



"Now, help me sneak up on the pig any way you want to!"

https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/11

A little bit about variables, values, and expressions

- Values/Objects
  - Numbers 1 -2.5
  - Logical values True False
  - Strings 'Hello' "Hello" "It's a good day, today"
  - List [1,2,3] ["it's", "a", "good", "day"]
  - Tuple ('Math', 8.4) (‘John', ‘English', 84)
  - Dictionary {'Math': 8.4, 'English': 9.0, 'Physics': 6.5}

https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/11

Expressions

- An expression represents something
  - Python evaluates it (turns it into a value)
  - Similar to a calculator
- Examples:
  - `2.3`
  - `(3 * 7 + 2) * 0.1`

Storing and Computing Data

What data might we want to work with?

(What's on your computer?)

- Values/Objects
  - Numbers 1 -2.5
  - Logical values True False
  - Strings 'Hello' "Hello" "It's a good day, today"
  - List [1,2,3] ["it's", "a", "good", "day"]
  - Tuple ('Math', 8.4) ('John', 'English', 84)
  - Dictionary {'Math': 8.4, 'English': 9.0, 'Physics': 6.5}

Variables

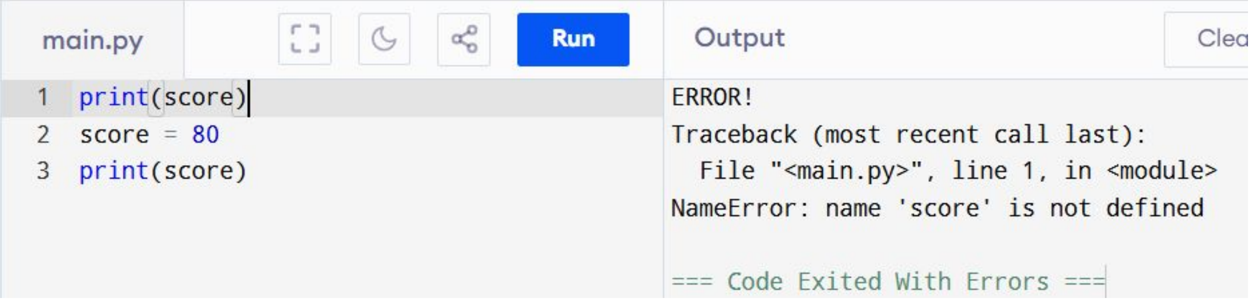
We need names to refer to pieces of data

- Variables: names of objects
  - `x = 5`
  - `x = 10 * 2`
  - `numbers = [1,2,3]`
    - numbers is now a name of the list [1,2,3].
  - `numbers = [1.2, 354.2, 7.3]`
    - numbers is now a name of the second list

Variables…

We need to define a name before it can be used

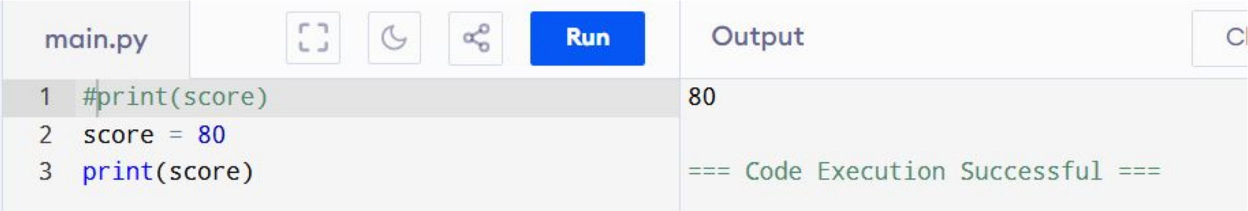
score is used before being defined



```
print(score)
score = 80
print(score)
```

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 1, in <module>
NameError: name 'score' is not defined

=== Code Exited With Errors ===
```

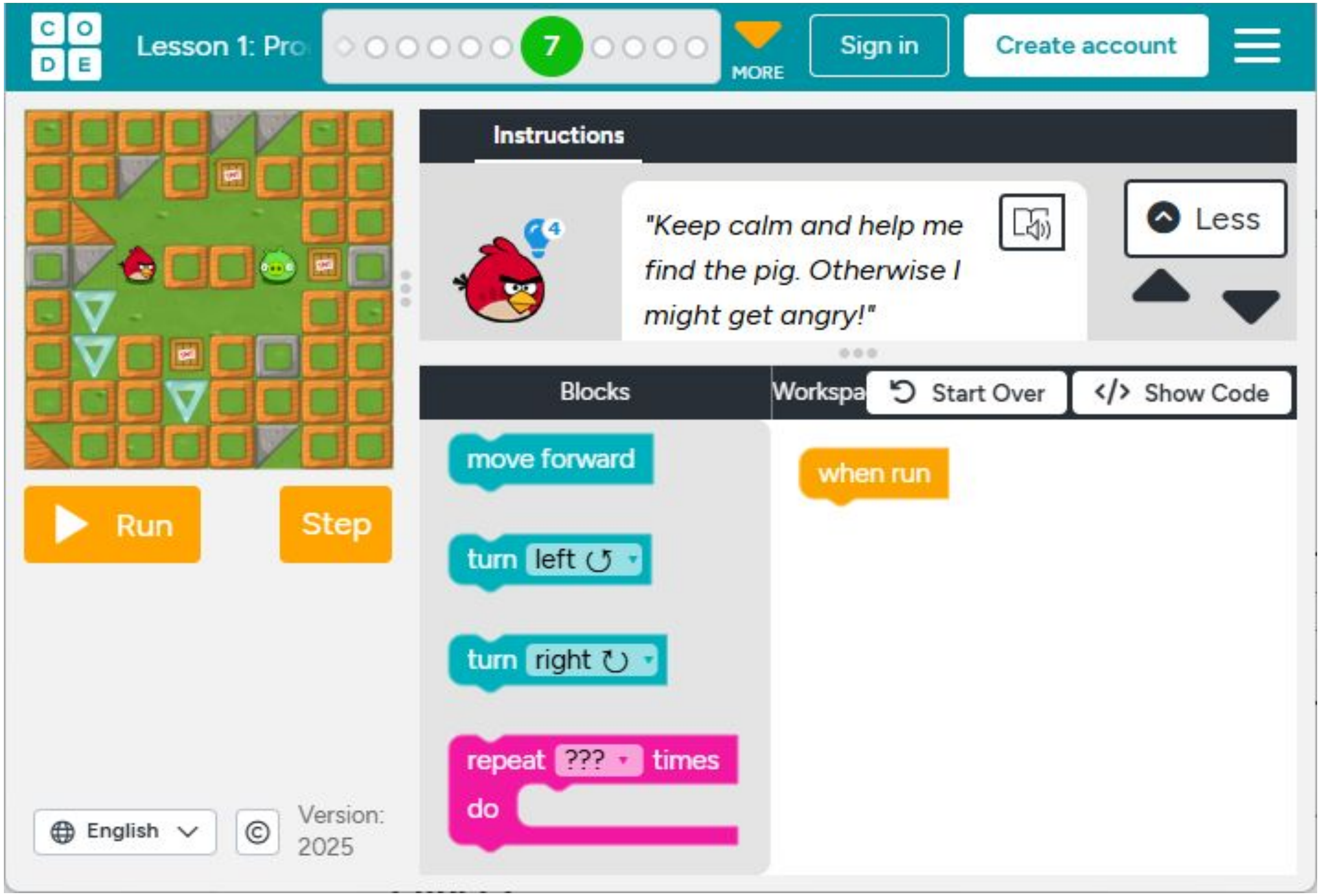
score is used after having been defined



```
#print(score)
score = 80
print(score)
```

```
80

=== Code Execution Successful ===
```

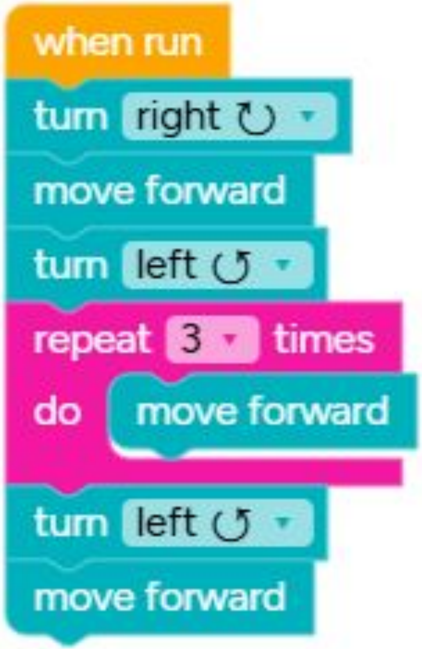
More games



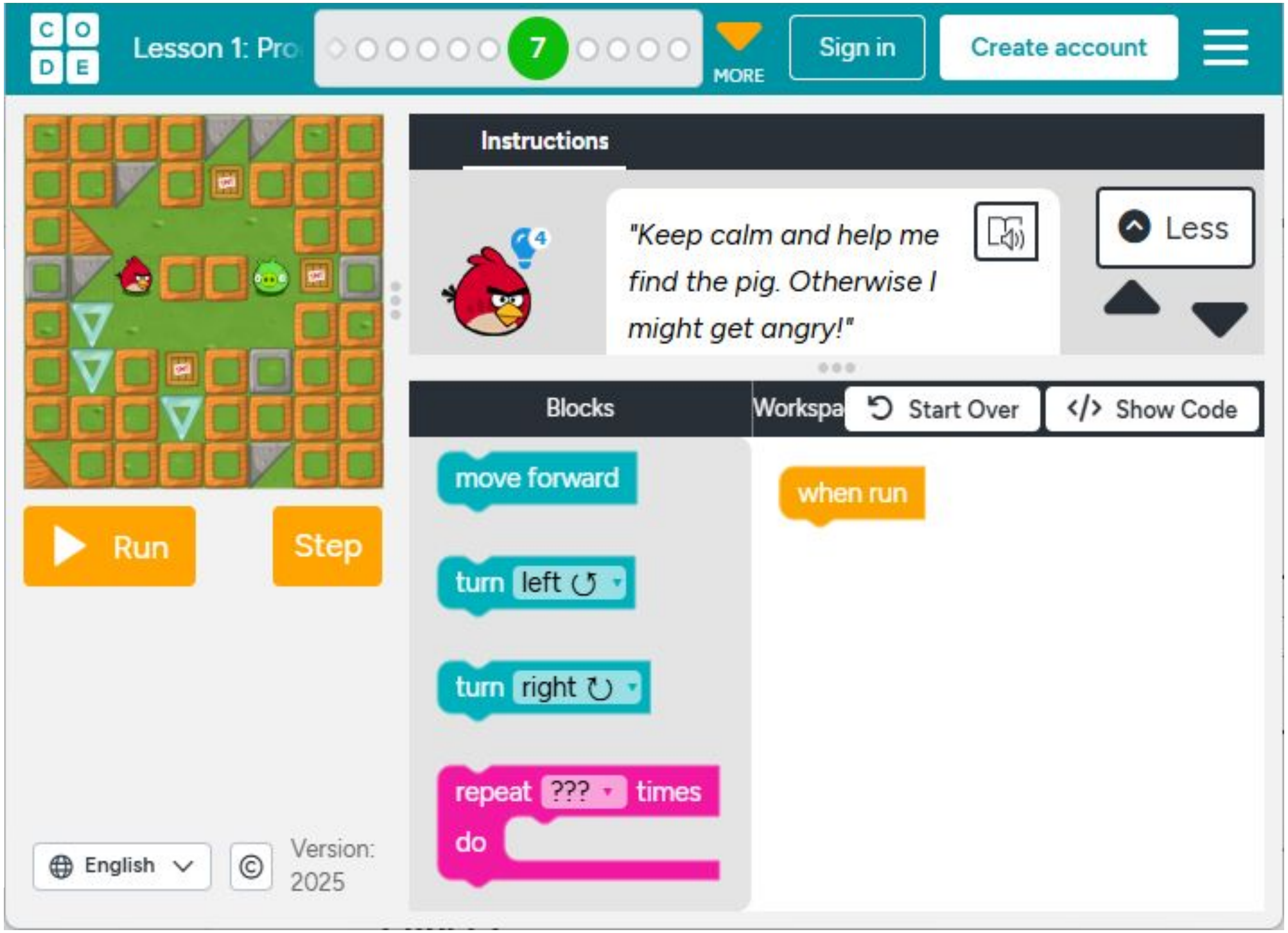
"Keep calm and help me find the pig. Otherwise I might get angry!"

https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/7

Loops



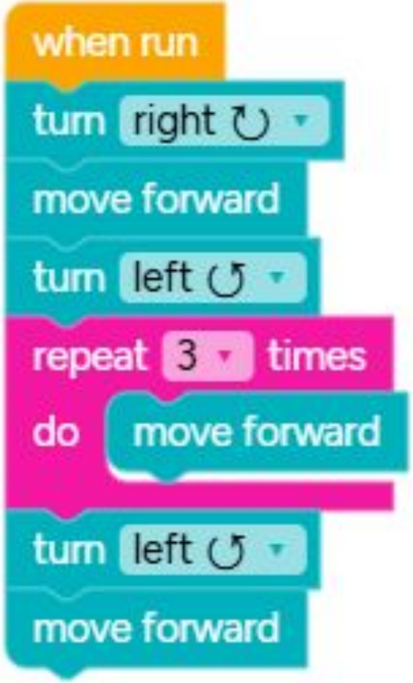
```
when run
turn right ↻
move forward
turn left ↺
repeat 3 times
do  move forward
turn left ↺
move forward
```

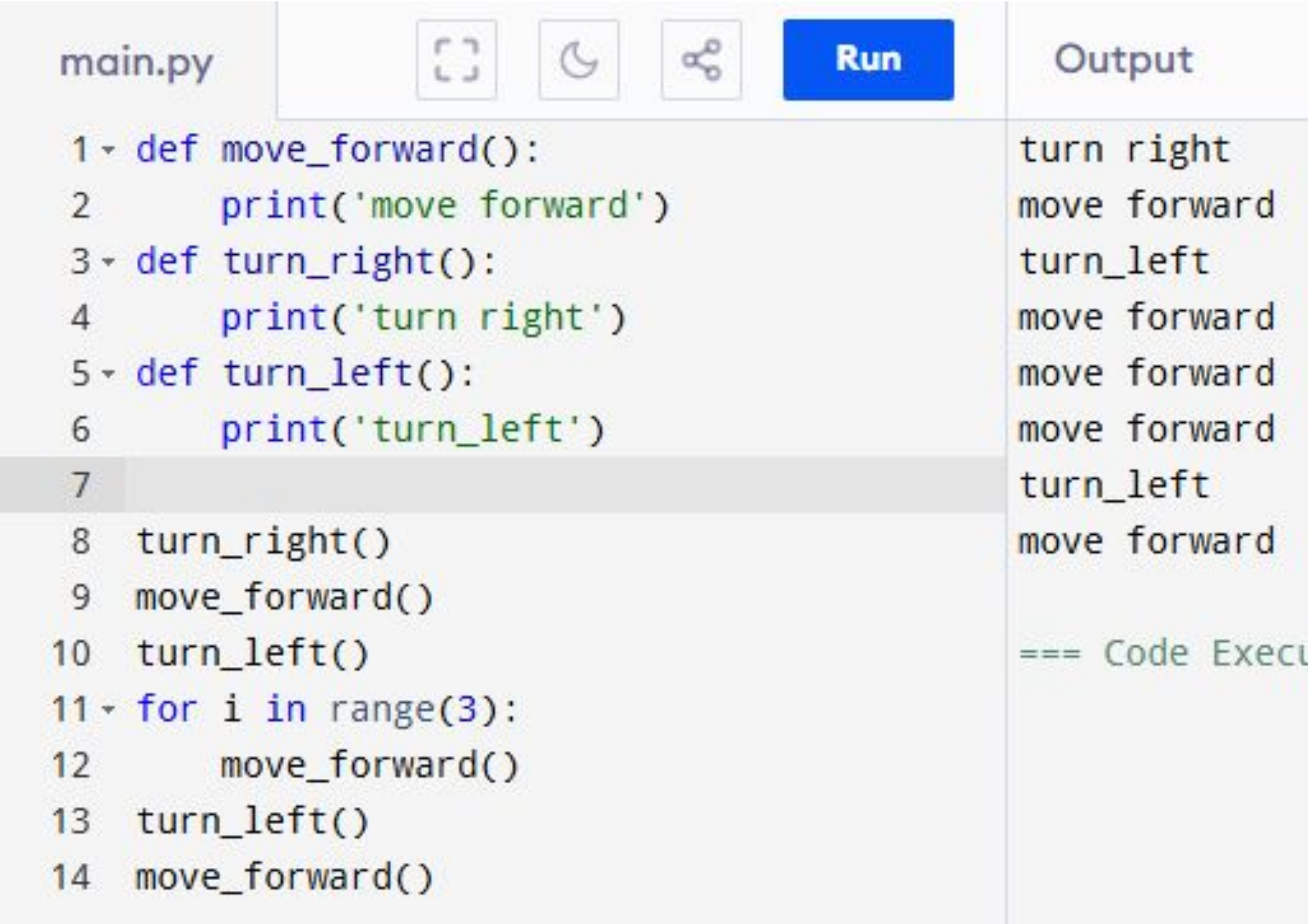


"Keep calm and help me find the pig. Otherwise I might get angry!"

https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/7

Loops





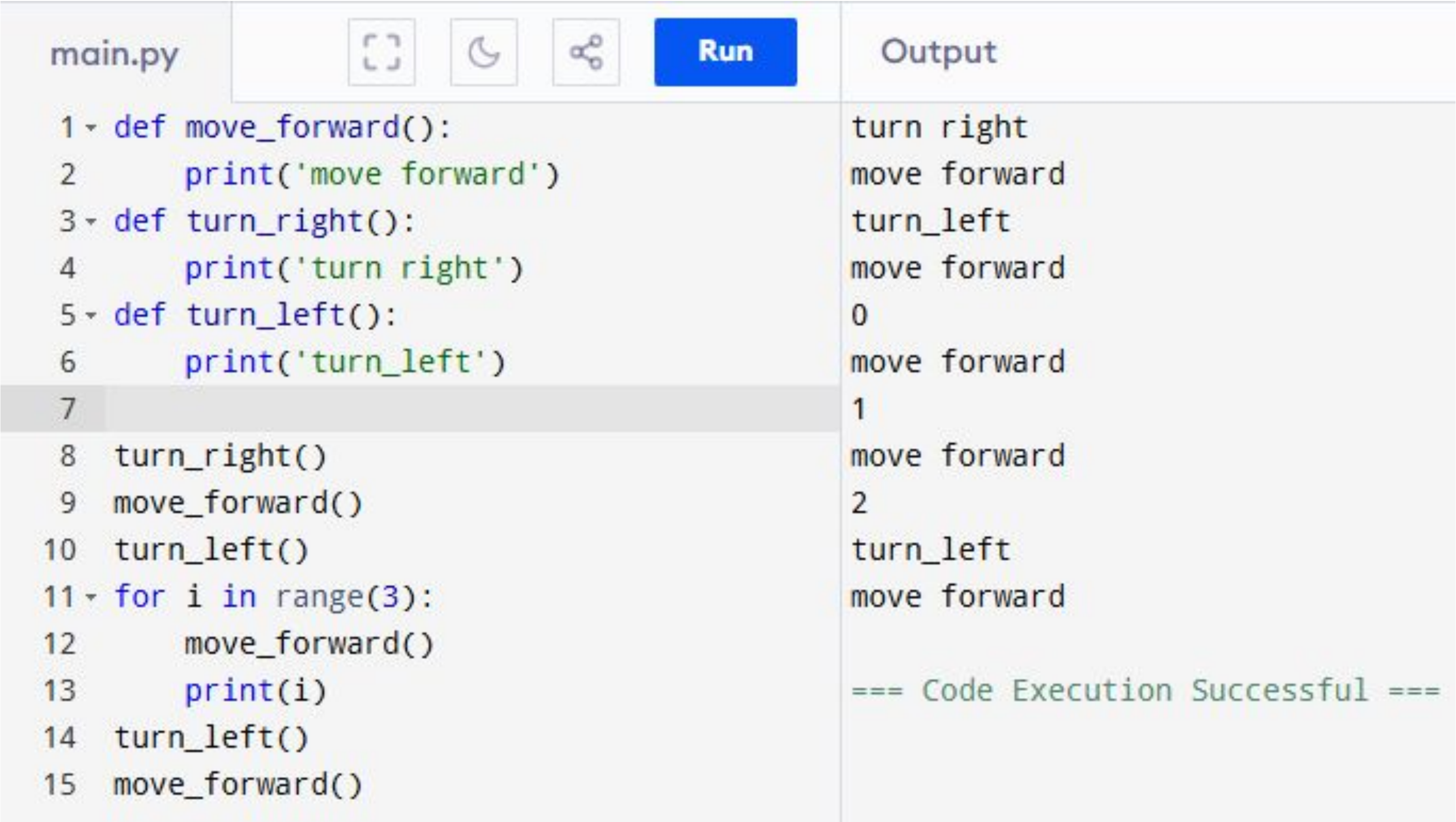
```
def move_forward():
    print('move forward')
def turn_right():
    print('turn right')
def turn_left():
    print('turn_left')

turn_right()
move_forward()
turn_left()
for i in range(3):
    move_forward()
turn_left()
move_forward()
```

```
turn right
move forward
turn_left
move forward
move forward
move forward
turn_left
move forward


=== Code Execu
```

https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/7

Repetition In Python

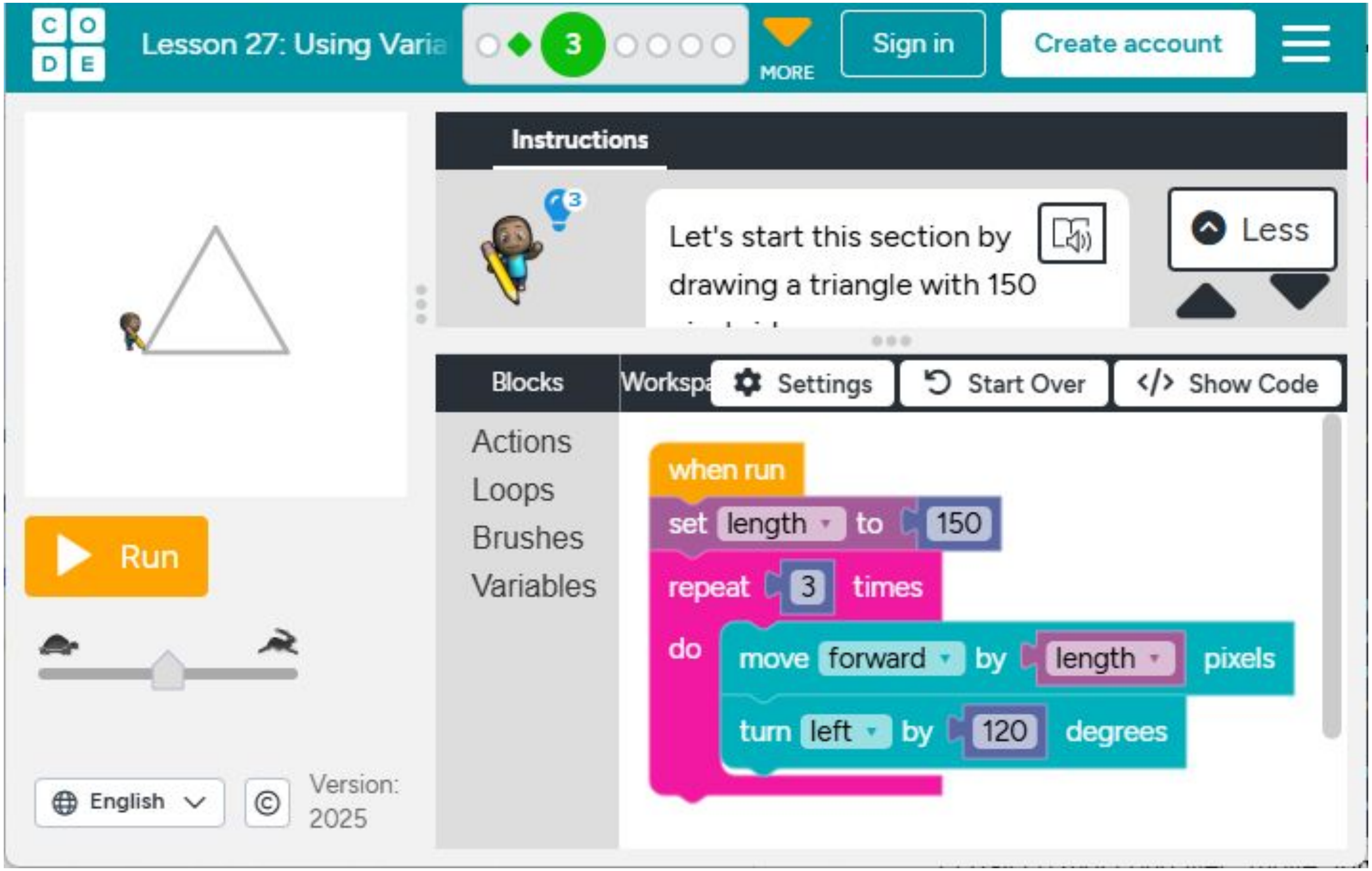
You can do other things in a for loop



```python
def move_forward():
    print('move forward')
def turn_right():
    print('turn right')
def turn_left():
    print('turn_left')

turn_right()
move_forward()
turn_left()
for i in range(3):
    move_forward()
    print(i)
turn_left()
move_forward()
```

```
turn right
move forward
turn_left
move forward
0
move forward
1
move forward
2
turn_left
move forward

=== Code Execution Successful ===
```

More games

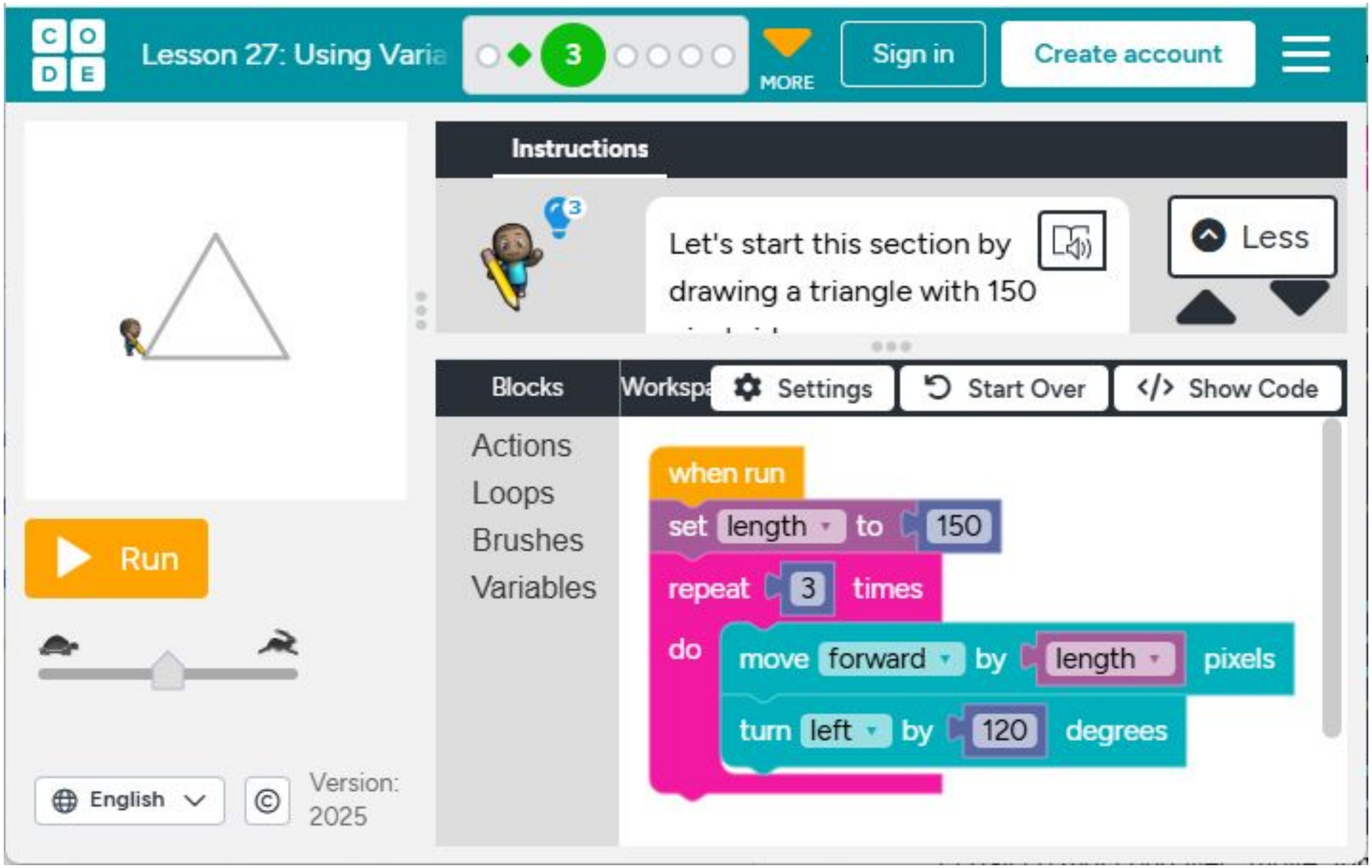


Let's start this section by drawing a triangle with 150

```
when run
set length to 150
repeat 3 times
do  move forward by length pixels
    turn left by 120 degrees
```

https://studio.code.org/courses/express-2025/units/1/lessons/27/levels/3

More games



Let's start this section by drawing a triangle with 150

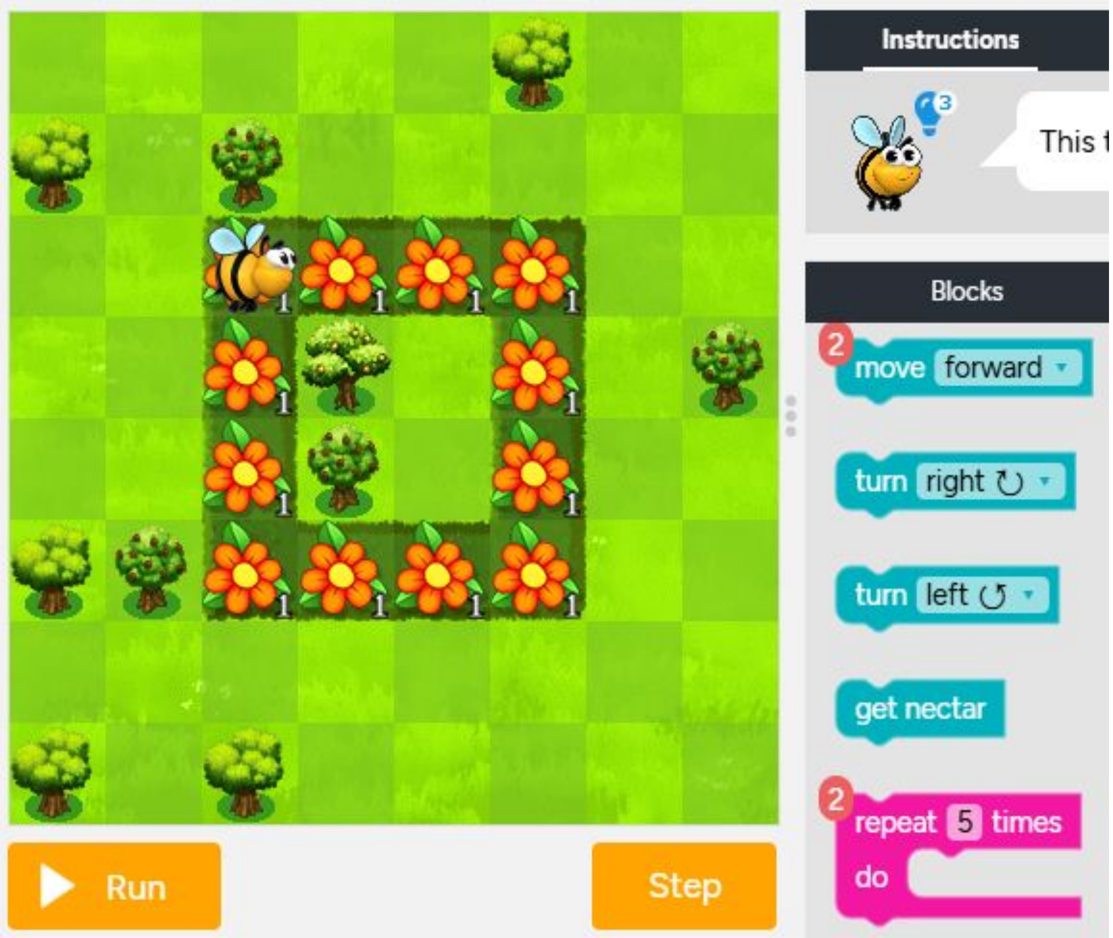
```
when run
set length to 150
repeat 3 times
do
    move forward by length pixels
    turn left by 120 degrees
```

https://studio.code.org/courses/express-2025/units/1/lessons/27/levels/3

More complicated logic

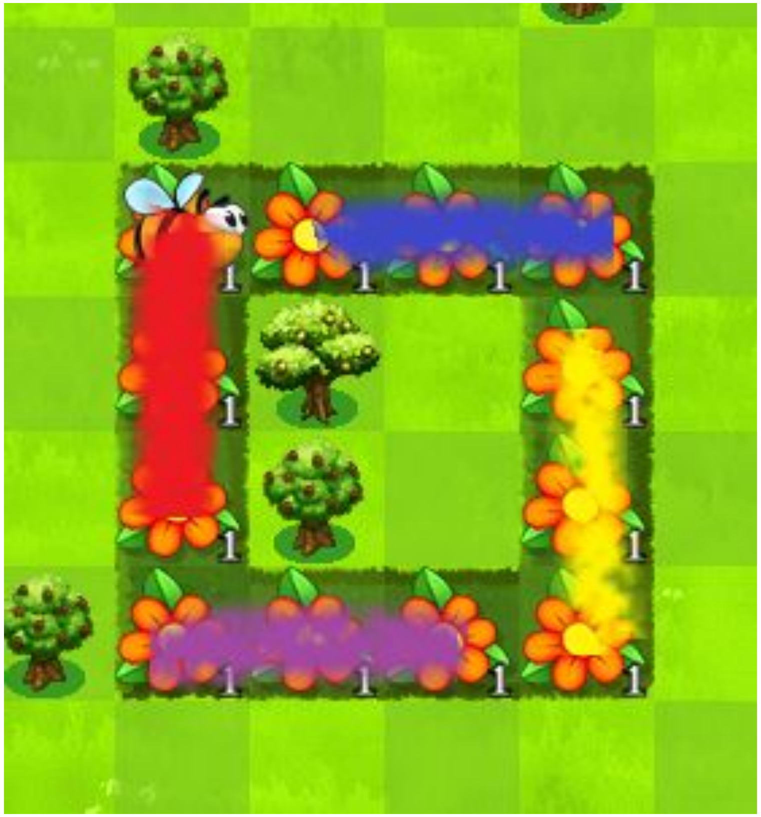
- Using as few blocks as possible to get the bee to take all the flower's nectar

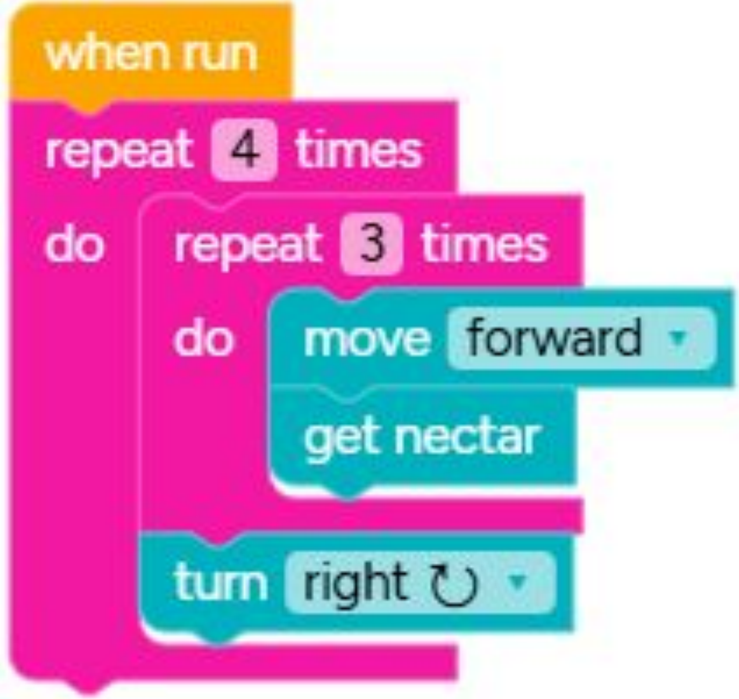
- What are the tasks that are pretty much the same?



https://studio.code.org/courses/express-2025/units/1/lessons/14/levels/5

More complicated logic



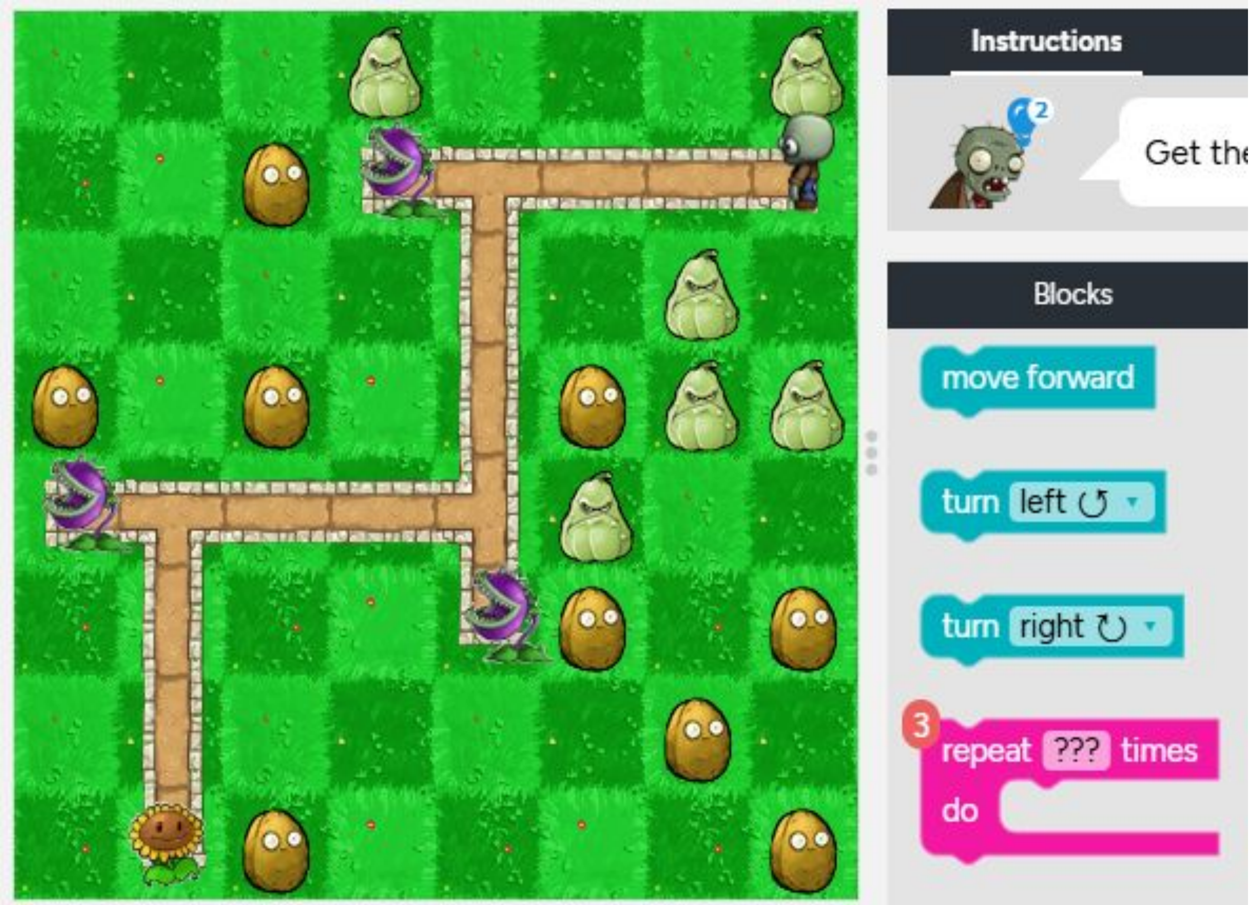


```
when run
repeat 4 times
do  repeat 3 times
    do  move forward
        get nectar
    turn right ↻
```

https://studio.code.org/courses/express-2025/units/1/lessons/14/levels/5

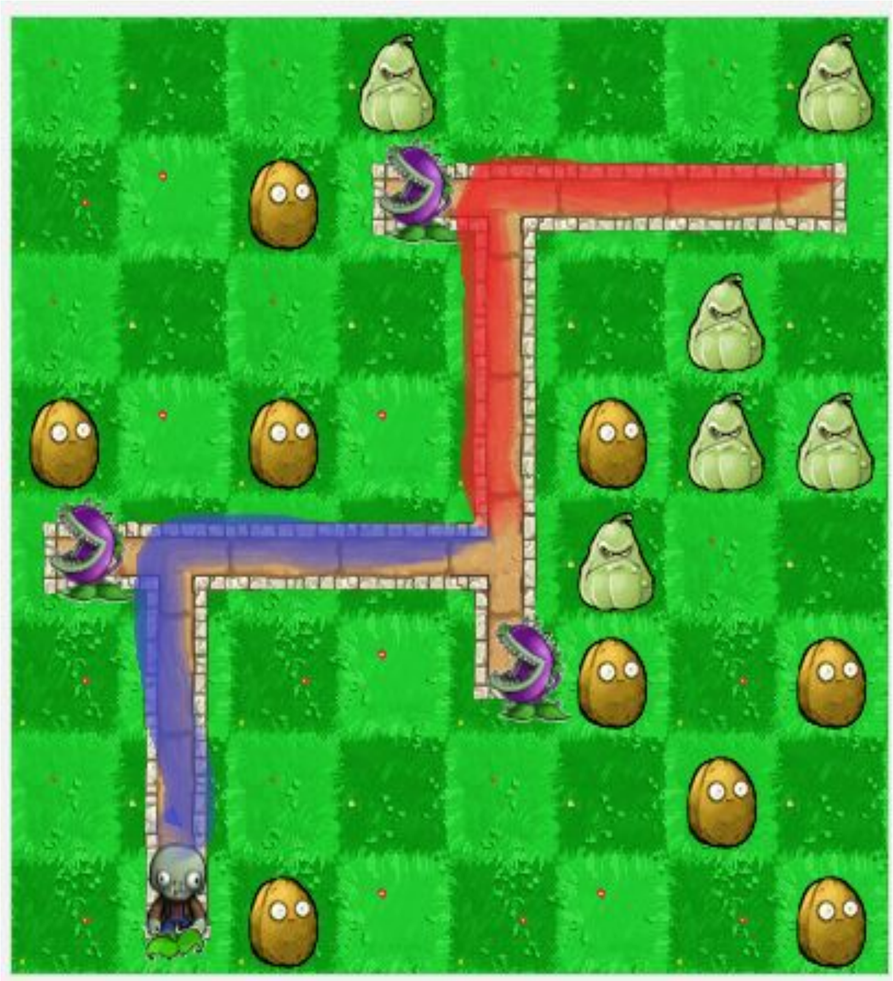
More complicated logic

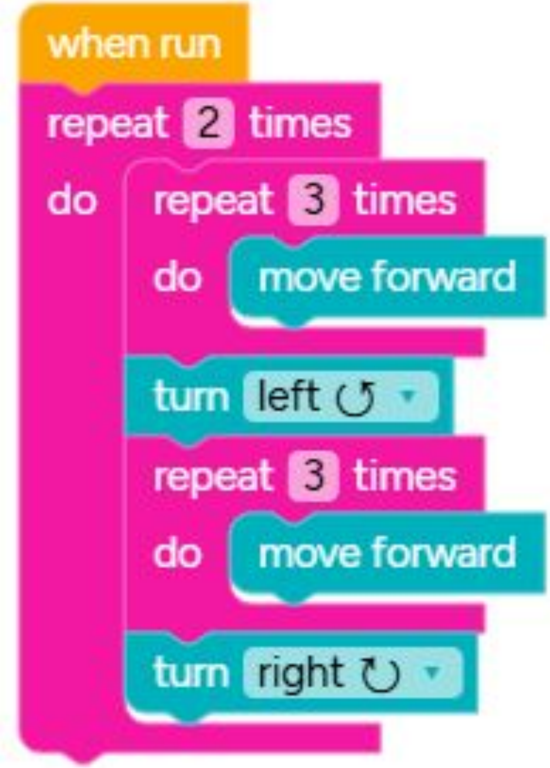
- Using as few blocks as possible to get the zombie to the flower
- What are the tasks that are pretty much the same?



https://studio.code.org/courses/express-2025/units/1/lessons/14/levels/8

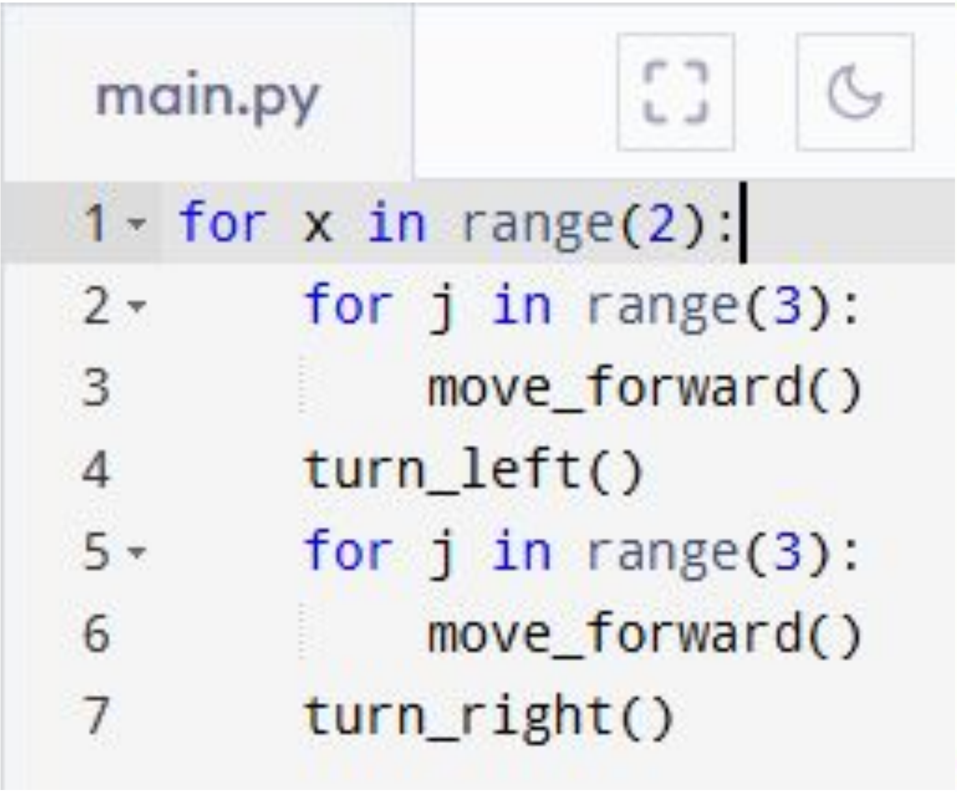
More complicated logic





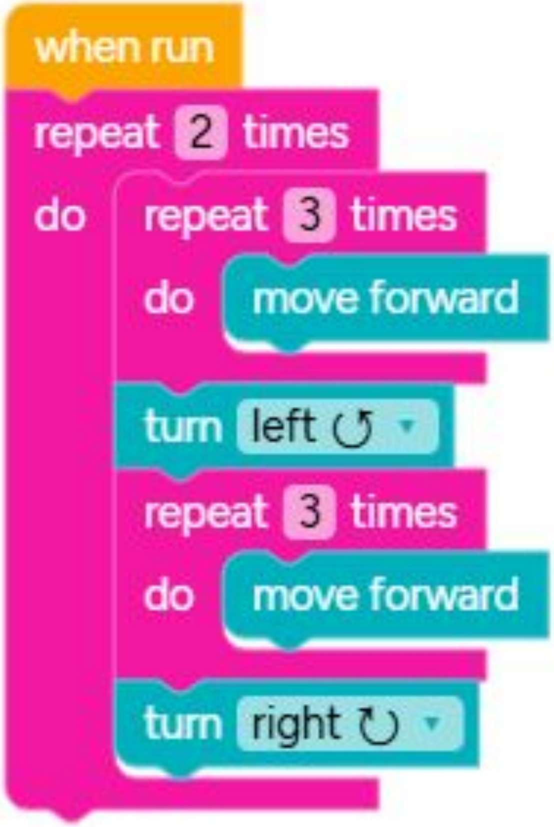
```
when run
repeat 2 times
do  repeat 3 times
    do  move forward
    turn left ↺
    repeat 3 times
    do  move forward
    turn right ↻
```

More complicated logic



main.py

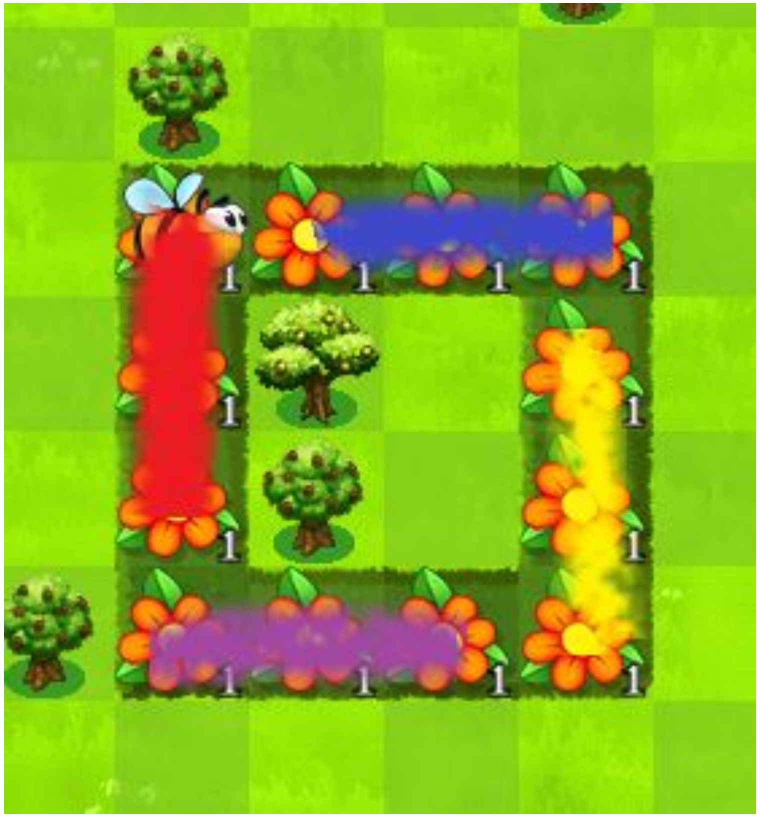
```python
for x in range(2):
    for j in range(3):
        move_forward()
    turn_left()
    for j in range(3):
        move_forward()
    turn_right()
```

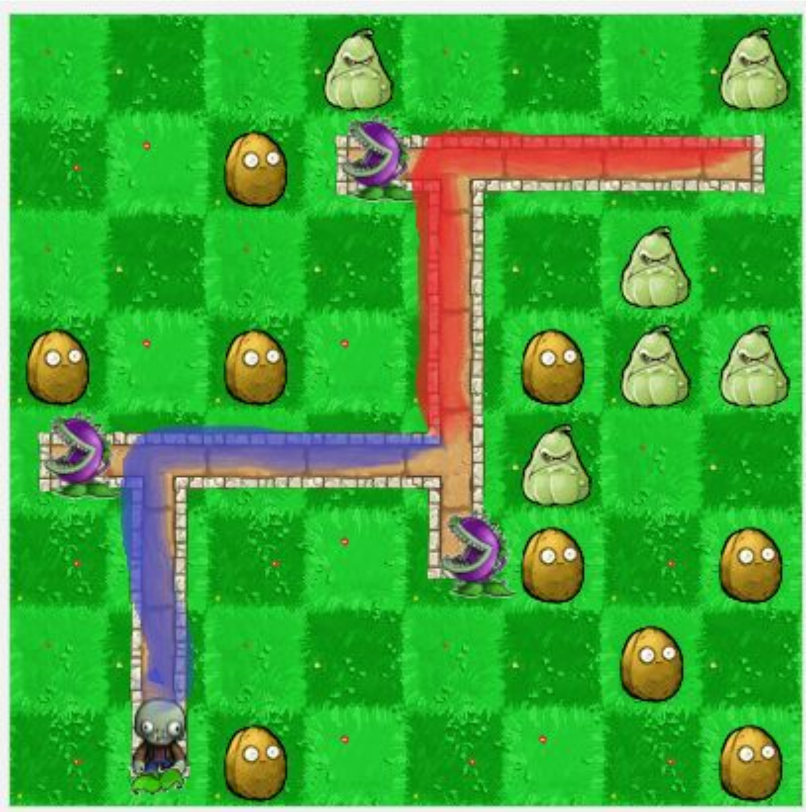


```
when run
repeat 2 times
do  repeat 3 times
    do  move forward
    turn left ↺
    repeat 3 times
    do  move forward
    turn right ↻
```

Important point: Dividing into subtasks

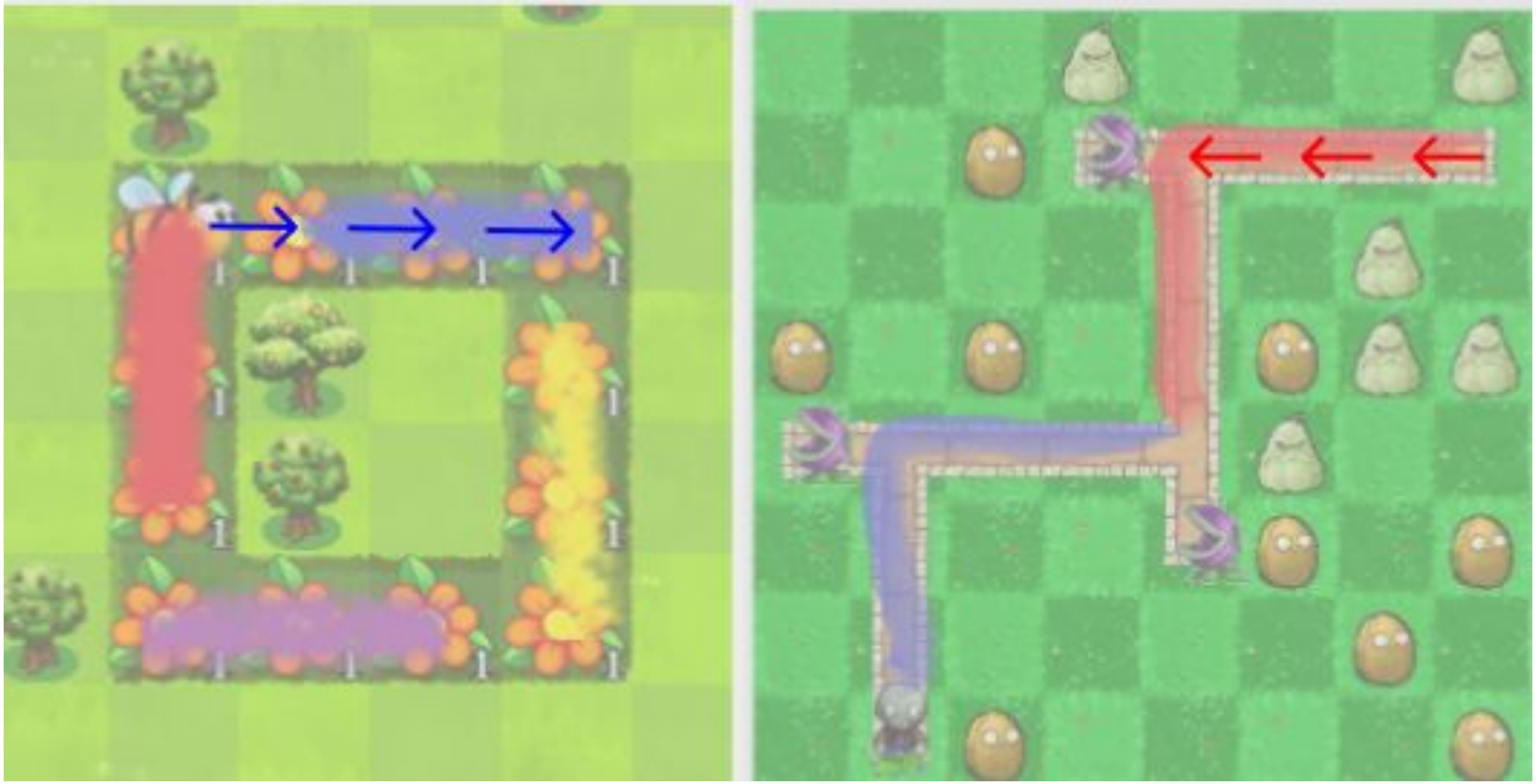
- Divide main task into subtasks, find patterns





Important point: Dividing into subtasks

- Divide further



More games?

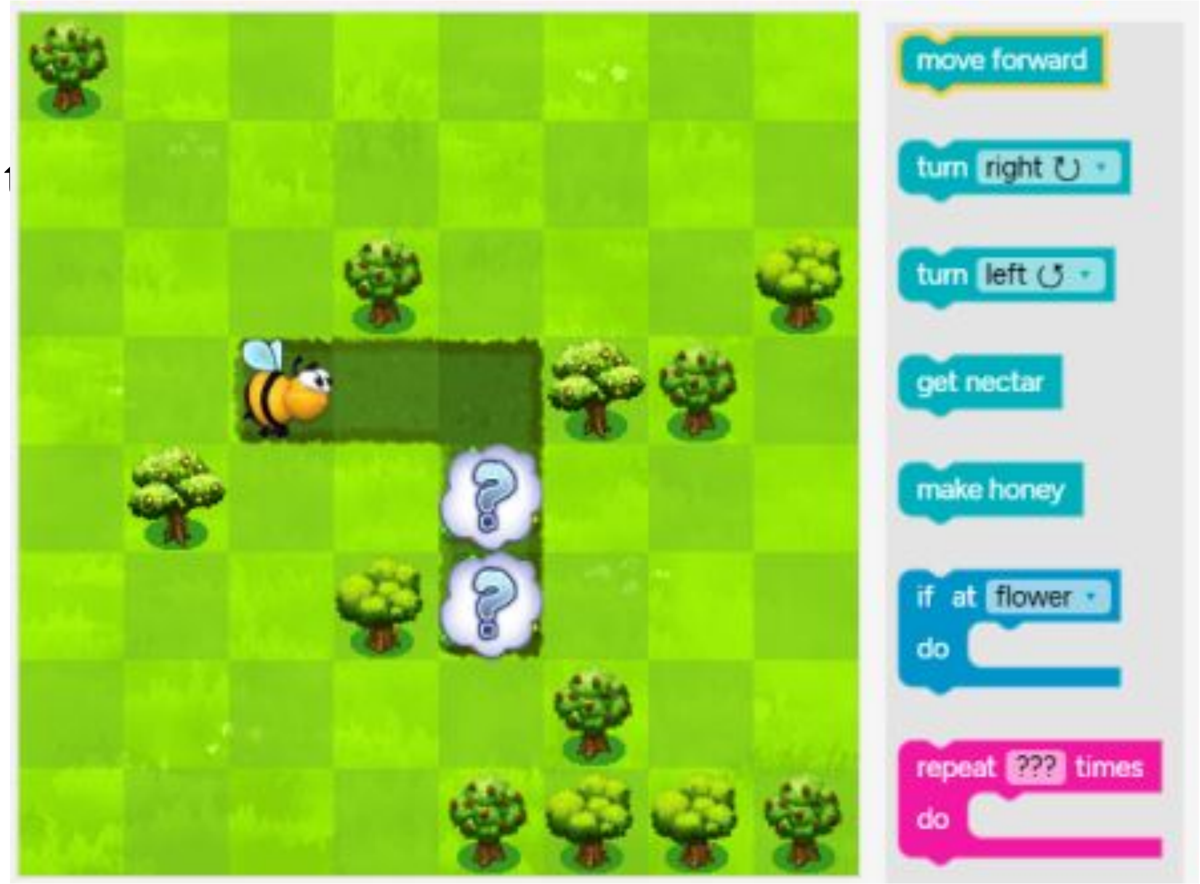
You can practice by playing games here

- https://studio.code.org/courses/express-2025/units/1/lessons/14/levels/1
- https://studio.code.org/courses/express-2025/units/1/lessons/15/levels/1

Selection

Check underneath every cloud to see if it is hiding a flower before you get nectar.

If there is a flower underneath the cloud, t need to get nectar *once*.



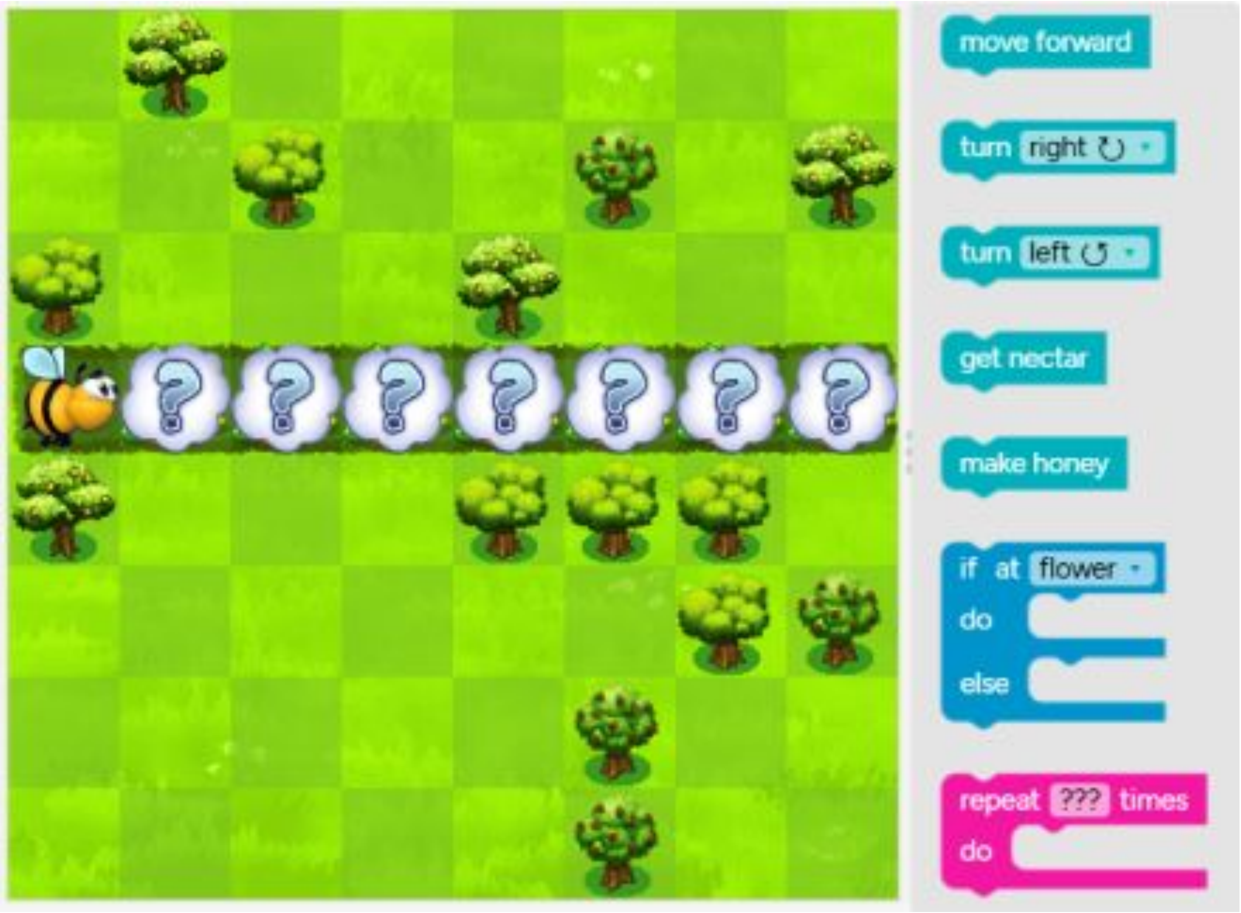
Remember:
Not all clouds hide the same thing!

https://studio.code.org/courses/express-2025/units/1/lessons/17/levels/3

Selection

Collect all of the nectar or make all the honey.

You can only collect nectar from flowers and make honey from honeycombs. Check any space to see if there is a flower or honeycomb. There will only ever be one flower or one honeycomb behind each cloud.

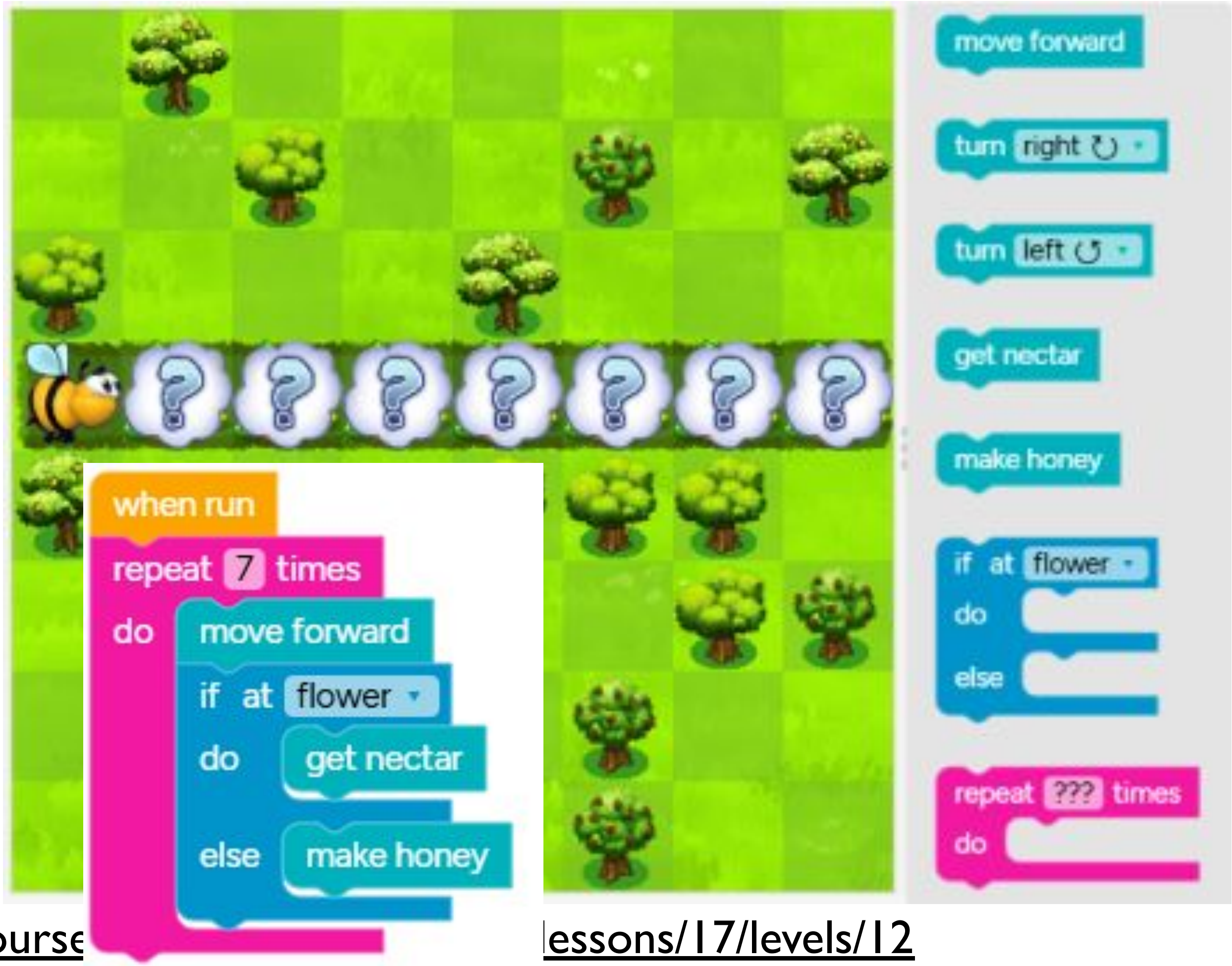


https://studio.code.org/courses/express-2025/units/1/lessons/17/levels/12

Selection

Collect all of the nectar or make all the honey.

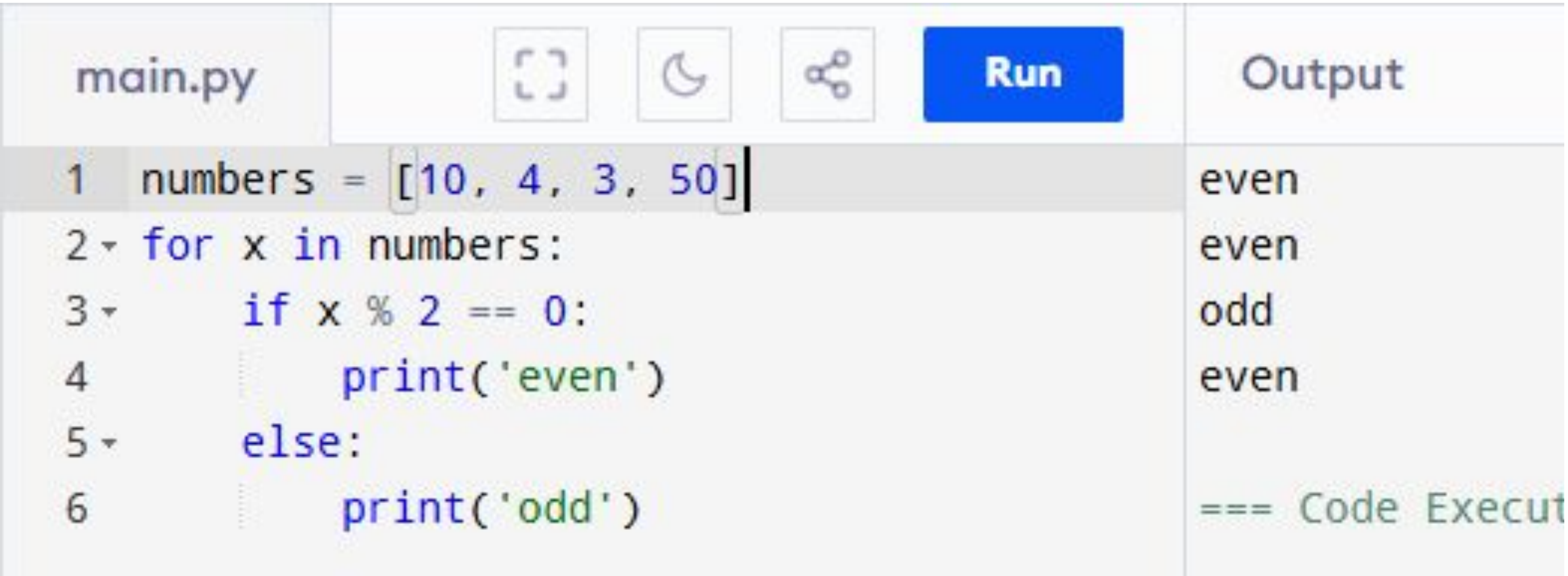
You can only collect nectar from flowers and make honey from honeycombs. Check any space to see if there is a flower or honeycomb. There will only ever be one flower or one honeycomb behind each cloud.



```
when run
repeat 7 times
do  move forward
    if at flower
    do   get nectar
    else make honey
```

https://studio.code.org/course lessons/17/levels/12

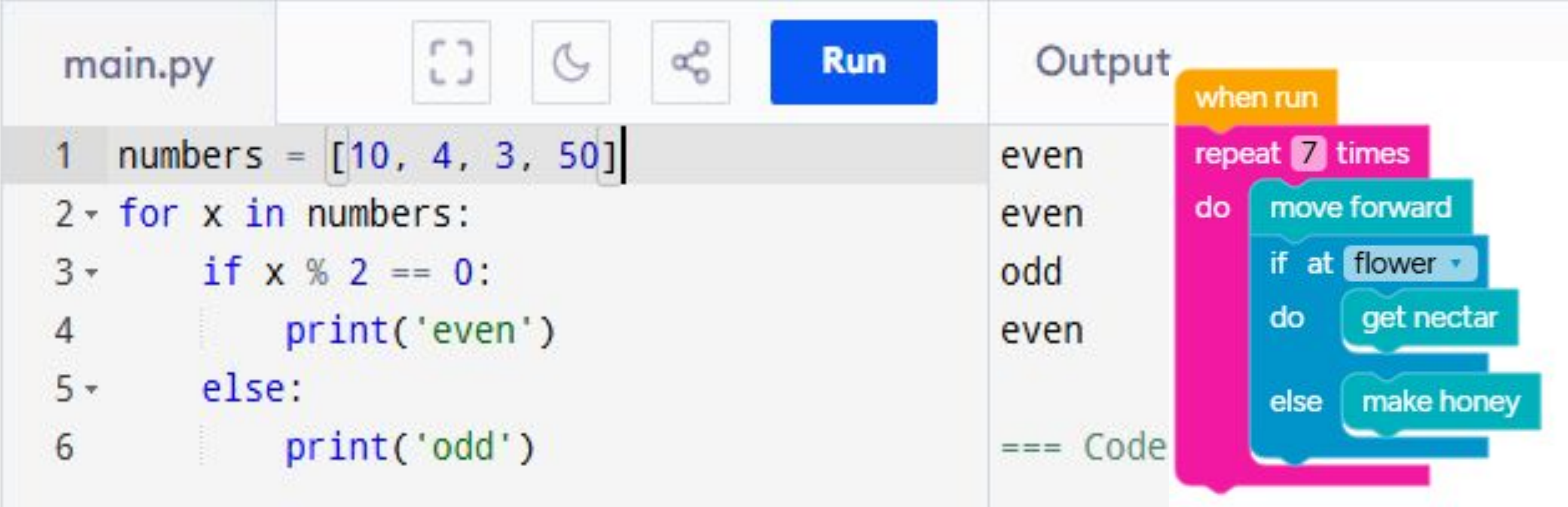
Selection in Python



```python
numbers = [10, 4, 3, 50]
for x in numbers:
    if x % 2 == 0:
        print('even')
    else:
        print('odd')
```

```
even
even
odd
even

=== Code Execut
```

Scratch vs. Python



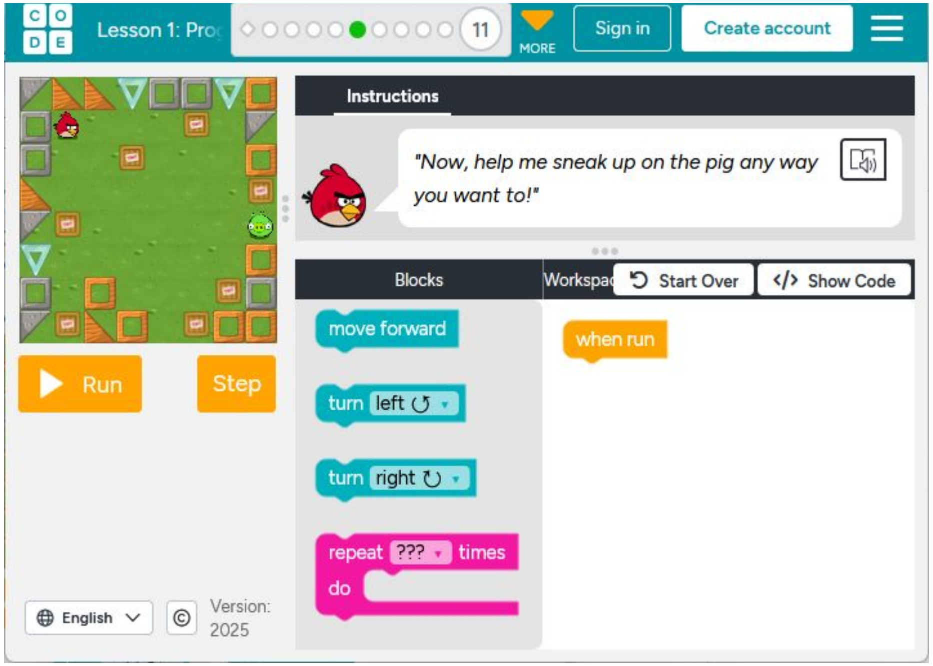
```
numbers = [10, 4, 3, 50]
for x in numbers:
    if x % 2 == 0:
        print('even')
    else:
        print('odd')
```

```
even
even
odd
even

=== Code
```

Can you see how similar a "program in Scratch" to a program in Python?

So… what is programming?

- In this game
  - Arrange the blocks in the right sequence
  - Run to see if it does the job correctly
  - See where it goes wrong
  - Fix the sequence
- In professional terms
  - Write some code
  - Test the code
  - Debug
  - Fix errors

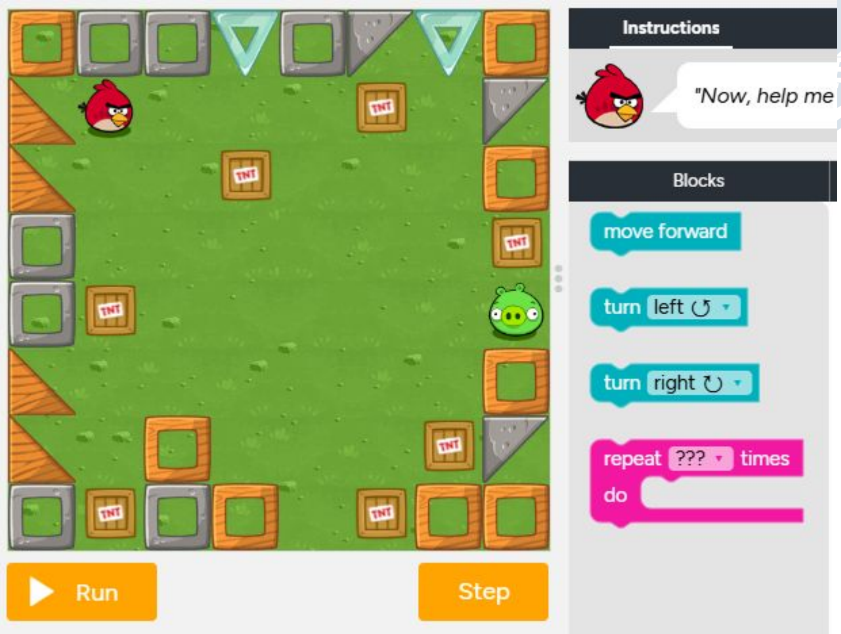


"Now, help me sneak up on the pig any way you want to!"

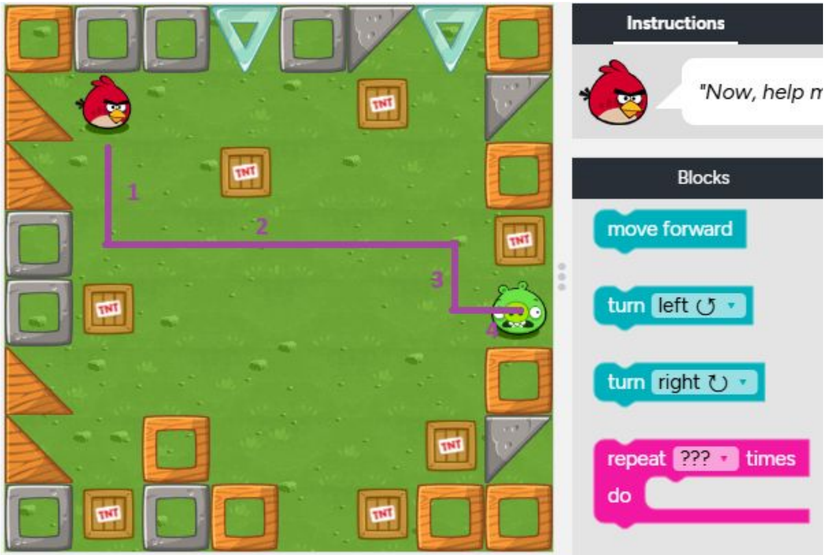
https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/11

But… how?

- Think in the given programming language
  - How to tell the machine to do the task using that language?
- Divide the task into smaller subtasks
  - Keep dividing subtasks into even smaller ones, until each task can be done by one instruction in the given set.



"Now, help me



"Now, help m

https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/11

Not sure what that means?

Don't worry.

Try playing with Scratch at code.org.

Practice solving problems by programming.

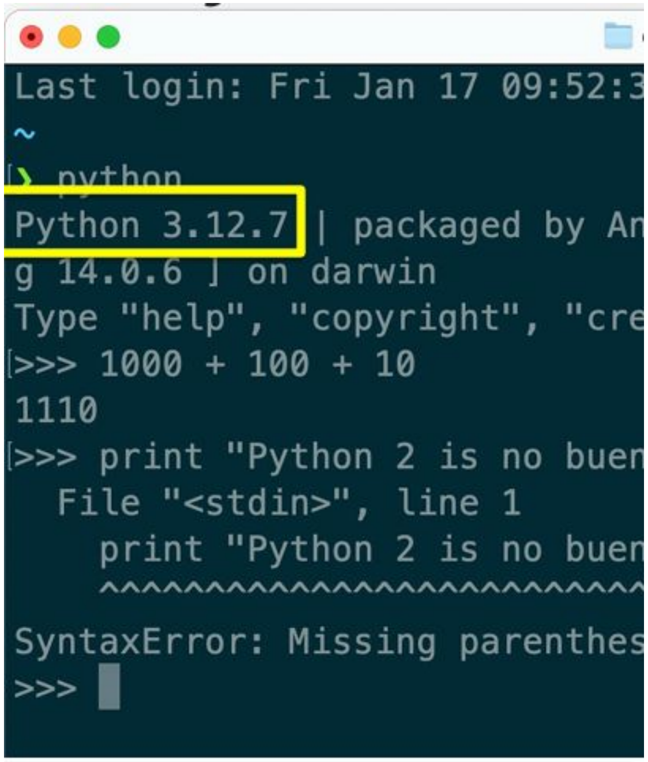
Bit by bit, you'll see!

Happy coding!

How to run a Python program

Getting Started with Python

- Designed to be used from the "command line"
  - OS X/Linux: **Terminal**
  - Windows: **PowerShell**
    - Preferred over **cmd**
    - See Lab instructions
- Install, then type "python"
  - Starts the interactive mode
  - Type commands at >>>
- Quit by typing quit() then pressing Return



```
Last login: Fri Jan 17 09:52:3
~
[❯ python
Python 3.12.7 | packaged by An
g 14.0.6 ] on darwin
Type "help", "copyright", "cre
[>>> 1000 + 100 + 10
1110
[>>> print "Python 2 is no buen
  File "<stdin>", line 1
    print "Python 2 is no buen
    ^^^^^^^^^^^^^^^^^^^^^^^^^^

SyntaxError: Missing parenthes
>>> █
```

This class uses **Python 3**
- Make sure you are, too!

Running a module

Module text file add.py

```
x = 'Hello'
y = x + ' World'
```

From the command line, type:

python <module filename>

Example:

```
C:\> python add.py
```

Nothing happen?
Actually, something did happen: Python executed all the commands in that file. They just don't do anything except assign some variables.

Running a module

Edit the file add.py

```
x = 'Hello'
y = x + ' World'
print("y = " + y)
```

Run it again:

```
C:\> python add.py
y = Hello  World
```

Now it showed something!
That's the result of the **print** statement.

Running a more interesting module

Module file guess.py

```
"""A really bad guessing game."""
user_guess = input('What word am I thinking of? ')
print('Wrong. I am not thinking of '
    + user_guess + '.')
```

The **input** function displays a prompt and waits for an input.

Here, we typed **cat** as an input

Command line:

```
C:\> python guess.py
What word am I thinking of? cat
Wrong. I am not thinking of cat.
```

Interactive mode _ typing code

```
C:\> python
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23...
Type "help", "copyright", "credits" or "license" for more ...
>>> x = 'Hello'
>>> y = x + ' World'
>>> y
'Hello World'
>>>
```

Interactive mode _ import a module

Import a .py file

```
C:\> python
>>> import add
y = Hello World
>>>
```

Remember the file **add.py**? The statement **import add** (no extension .py) runs the script add.py

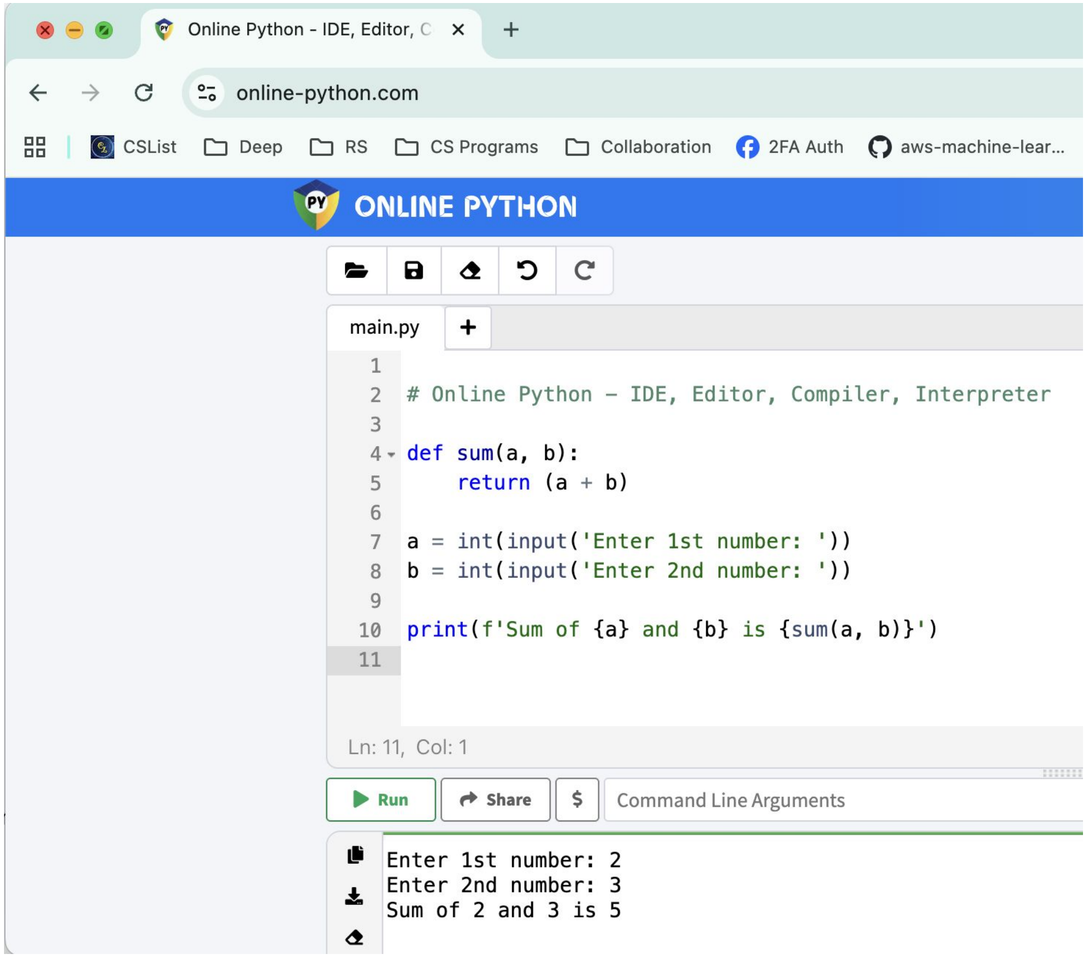
```
x = 'Hello'
y = x + ' World'
print("y = " + y)
```

Summary: Three ways to execute code

1. Typing code in interactive mode.
2. Importing a module.
3. Running a script.

Now you can go ahead installing Python in your computer and run all the sample codes.
Have fun!

If you do not want to install Python, how can we do?



```python

# Online Python - IDE, Editor, Compiler, Interpreter

def sum(a, b):
    return (a + b)

a = int(input('Enter 1st number: '))
b = int(input('Enter 2nd number: '))

print(f'Sum of {a} and {b} is {sum(a, b)}')

```

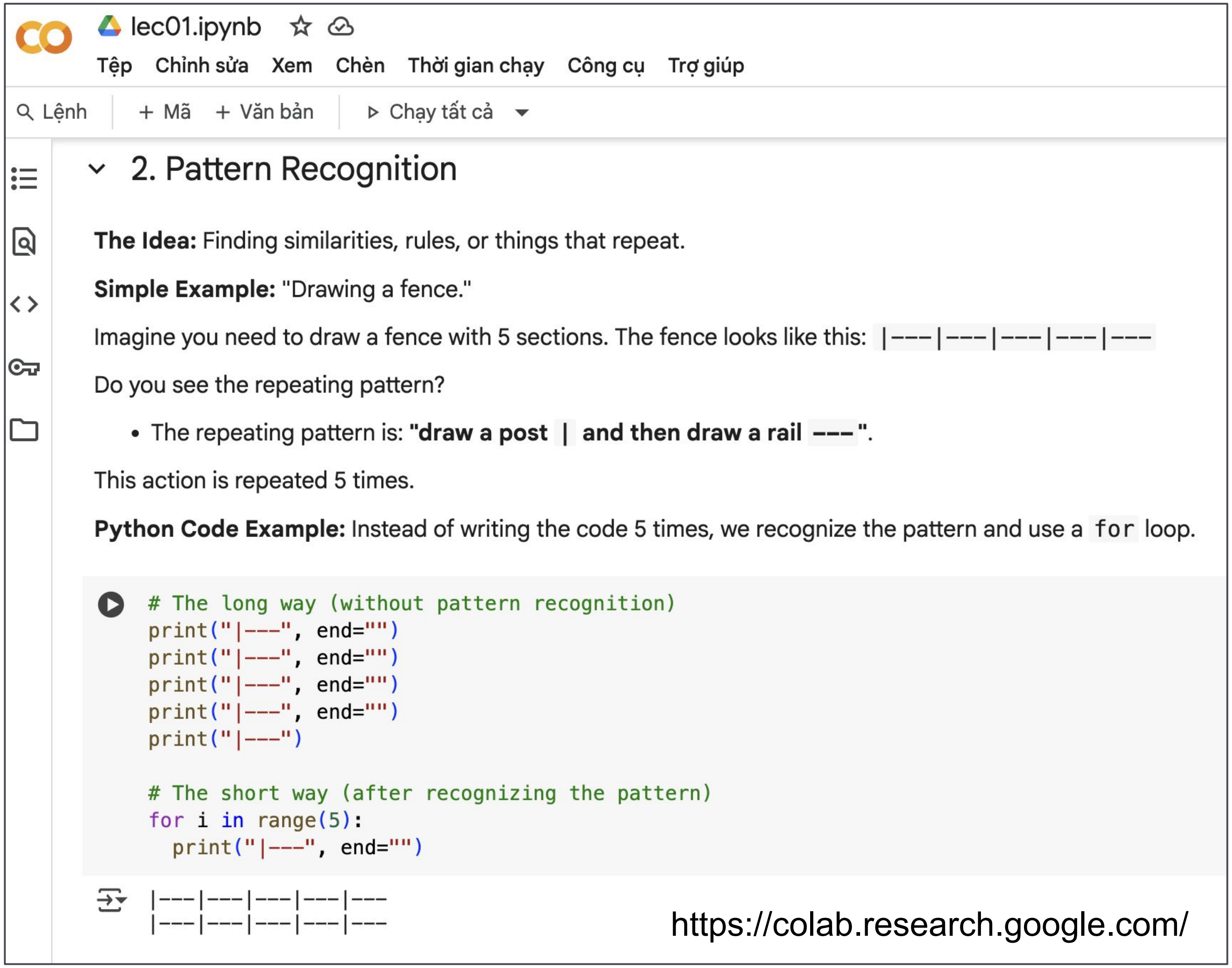
```
Enter 1st number: 2
Enter 2nd number: 3
Sum of 2 and 3 is 5
```

There are numerous Online IDE that support python programming:

online-python.com
pythononline.net
onlinegdb.com

…

Jupyter Notebook with Google Colab



2. Pattern Recognition

The Idea: Finding similarities, rules, or things that repeat.

Simple Example: "Drawing a fence."

Imagine you need to draw a fence with 5 sections. The fence looks like this: `|---|---|---|---|---`

Do you see the repeating pattern?

- The repeating pattern is: **"draw a post `|` and then draw a rail `---`"**.

This action is repeated 5 times.

Python Code Example: Instead of writing the code 5 times, we recognize the pattern and use a `for` loop.

```
# The long way (without pattern recognition)
print("|---", end="")
print("|---", end="")
print("|---", end="")
print("|---", end="")
print("|---")

# The short way (after recognizing the pattern)
for i in range(5):
  print("|---", end="")
```

```
|---|---|---|---|---
|---|---|---|---|---
```

https://colab.research.google.com/

Summary - Key Takeaways

- Problem-solving (with Computational) Thinking
  - Problem → Solution Idea → Algorithm → Code
- Computational Thinking
  - Decomposition, Pattern Recognition, Abstraction, Algorithm Design
- What is Programming
  - Programming = **writing instructions** so that a computer can perform a task.
- Getting Started with Python
  - Variable, Value, Expression, Loops, Selection
  - Learning by doing