



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

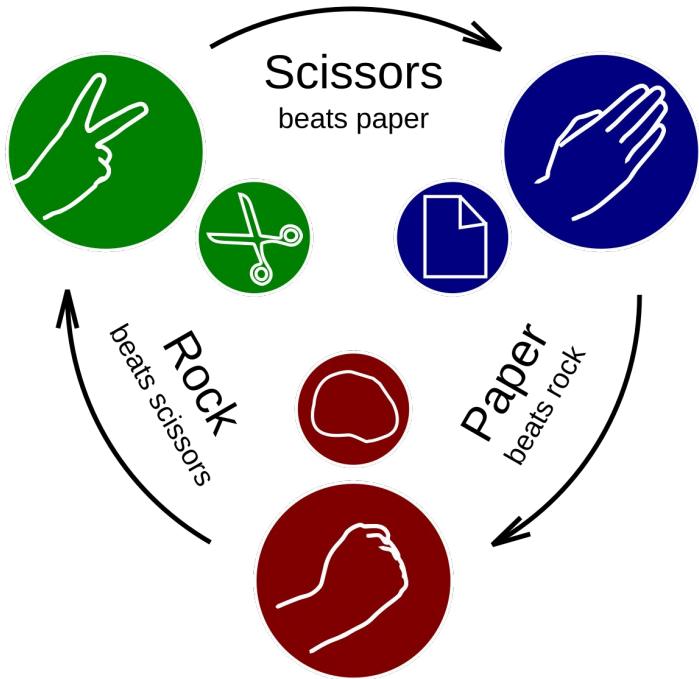


Tetris

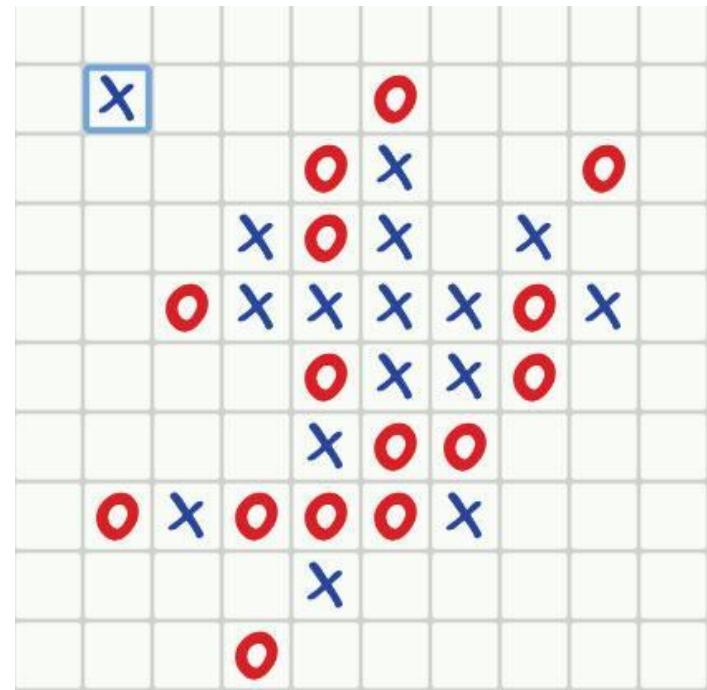
	7		6	5	
4		7	1		
	9	4	7	6	
3	6	8	7	9	2
		5			
2		1	4	6	8
	8	5	3	1	9
	1				
		7	9	4	

Sudoku

Multiplayer Games



Rock Paper Scissors



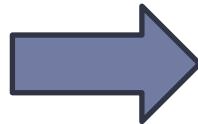
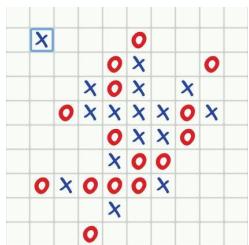
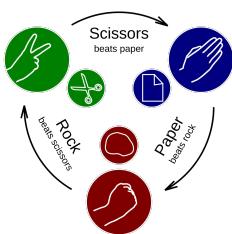
Tic Tac Toe

The notion of ‘Problem’

Problem
(to be solved)



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



Win the game!
(Optimal solution)

Thinking

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

Real-life Problem – Expense Management



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

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

Given a list of numbers, find the largest one.



Solution Idea

Value comparison



Algorithm

Pseudo code

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

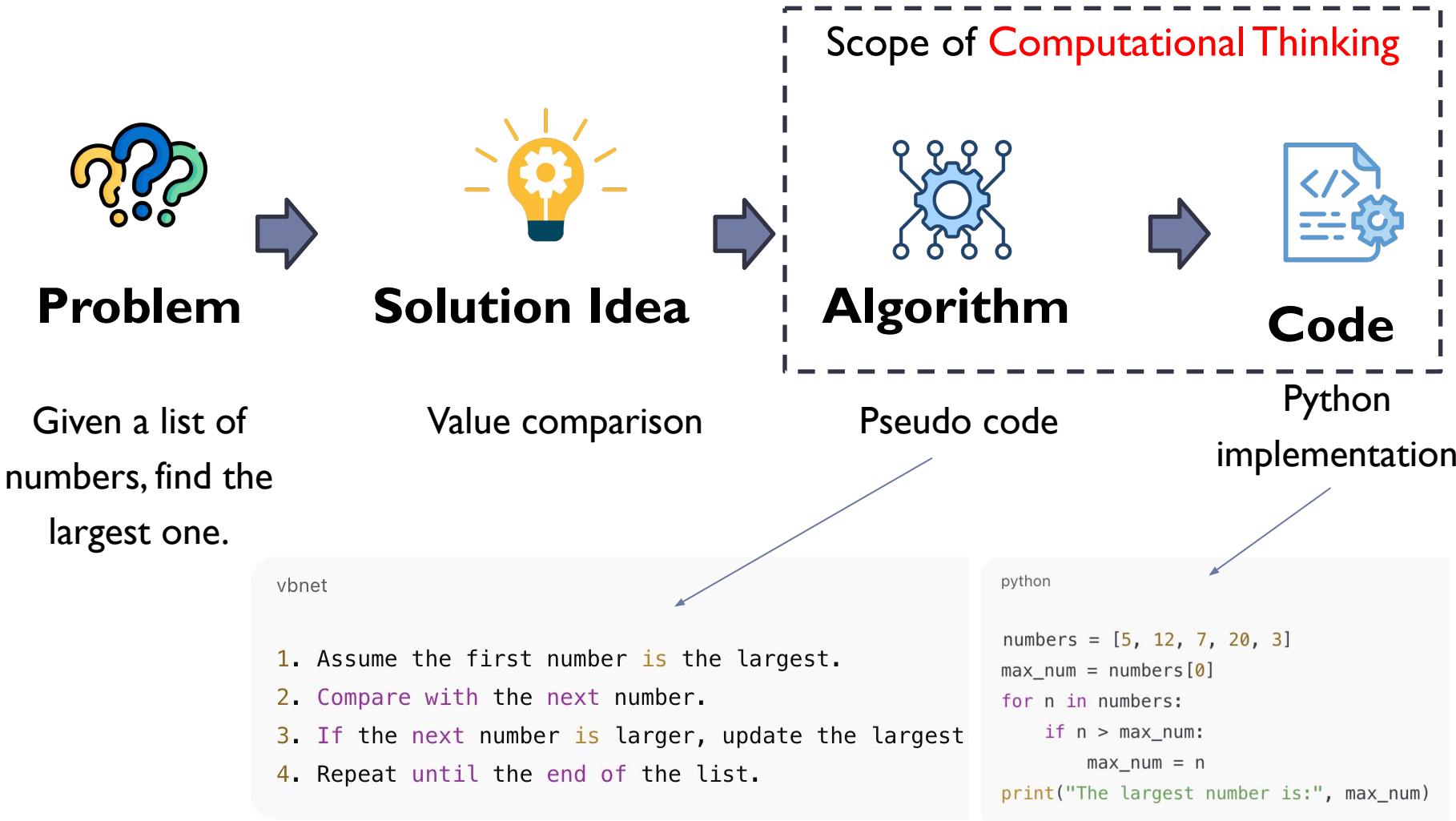
```
numbers = [5, 12, 7, 20, 3]
max_num = numbers[0]
for n in numbers:
    if n > max_num:
        max_num = n
print("The largest number is:", max_num)
```



Code

Python implementation

Problem-solving with Computational Thinking



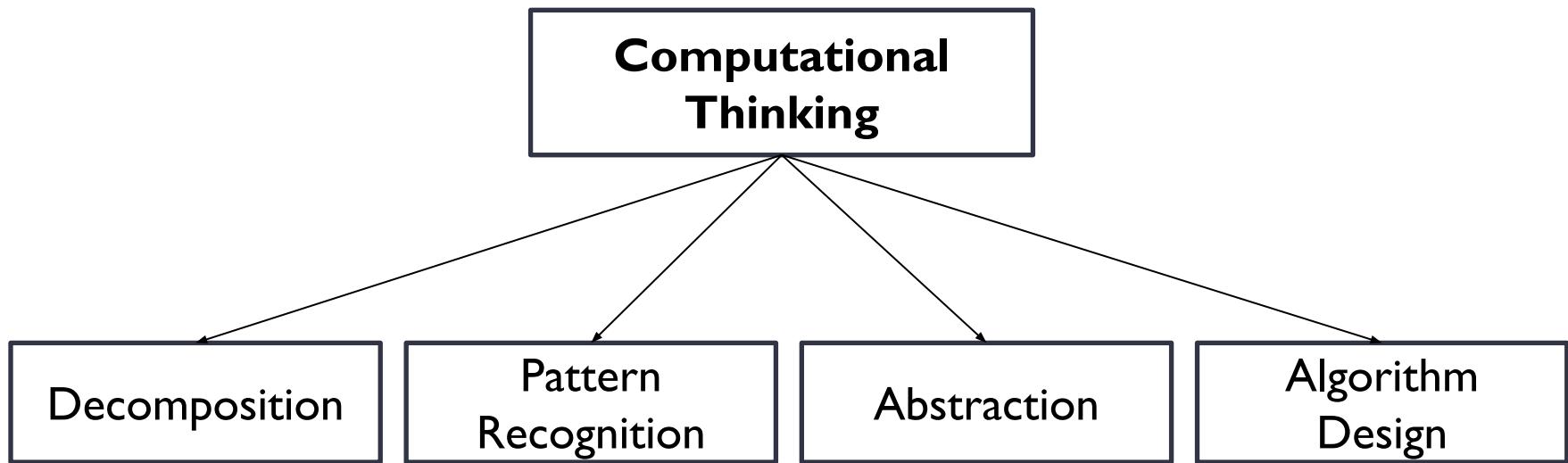


What is Computational Thinking?



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

markdown

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

```
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

```
# 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: Write a function: Find the largest number of a given list

```
python

def find_max(numbers):
    max_num = numbers[0]          # Step 1: assume first is largest
    for n in numbers:             # Step 2: check each number
        if n > max_num:          # Step 3: update if larger found
            max_num = n
    return max_num                # Step 4: return result

nums = [5, 12, 7, 20, 3]
print("Largest number is:", find_max(nums))
```

Algorithms are precise recipes to solve problems.

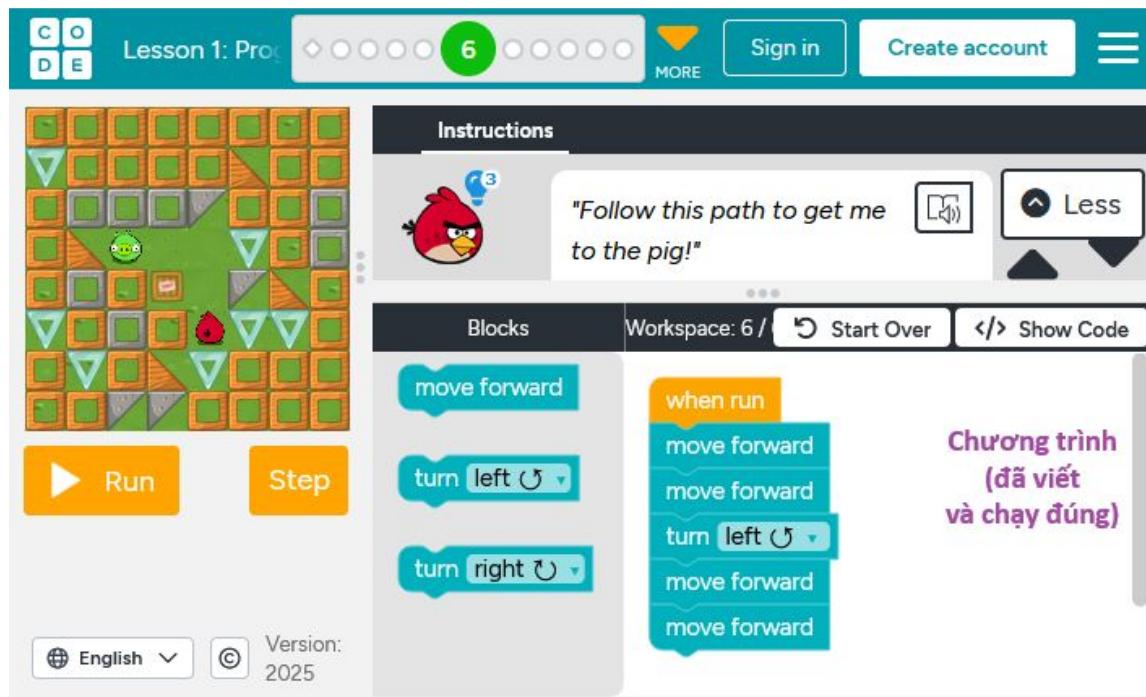


What is programming?



What is programming?

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

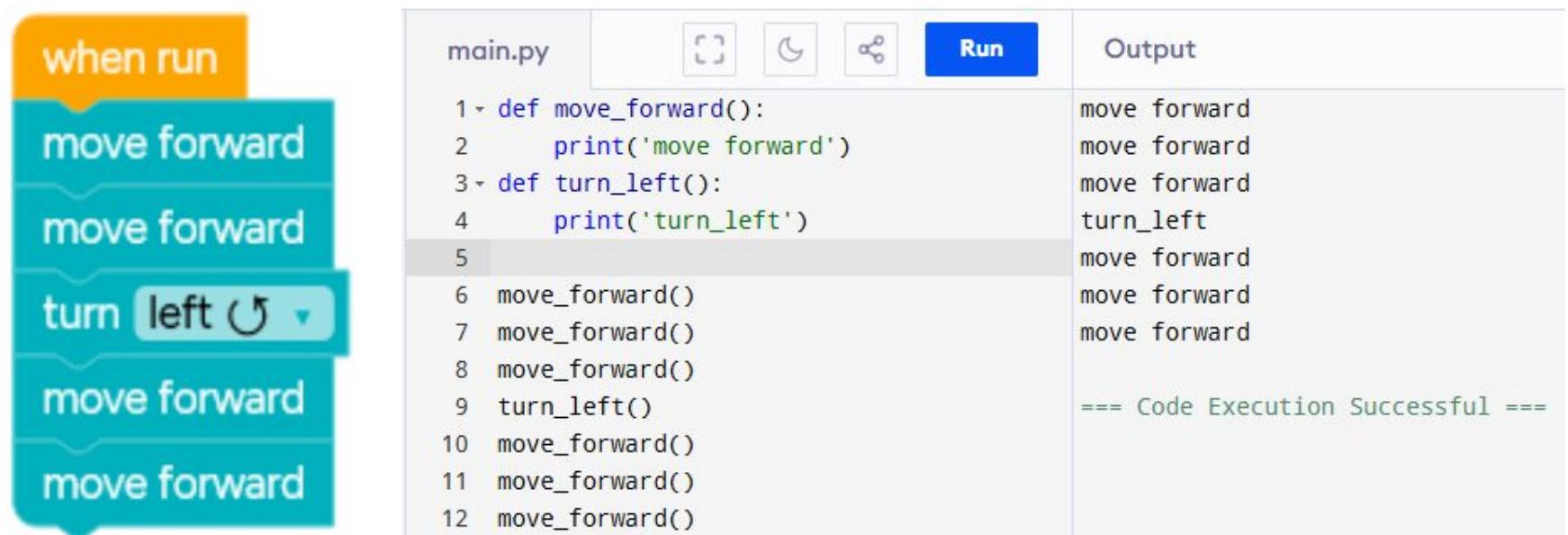


This is Scratch
at code.org
Kids' games,
actually

<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



The image shows a Scratch script on the left and its corresponding Python code and execution output on the right.

Scratch Script:

- when green flag clicked
- move forward
- move forward
- turn left
- move forward
- move forward

Python Code (main.py):

```
1 def move_forward():
2     print('move forward')
3 def turn_left():
4     print('turn_left')
5
6 move_forward()
7 move_forward()
8 move_forward()
9 turn_left()
10 move_forward()
11 move_forward()
12 move_forward()
```

Output:

```
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.



The image shows a Scratch game interface. At the top, there's a navigation bar with the 'CODE' logo, a progress bar showing level 11, and buttons for 'Sign in' and 'Create account'. Below the progress bar is a green play area featuring a red Angry Bird character and a green pig character. A speech bubble from the bird says, "Now, help me sneak up on the pig any way you want to!" To the right of the play area is an 'Instructions' panel. Below the play area are three buttons: 'Run', 'Step', and 'Stop'. On the right side of the screen is a workspace containing several Scratch script blocks:

- A yellow 'when run' hat block.
- A teal 'move forward' block.
- A teal 'turn left' block.
- A teal 'turn right' block.
- A pink 'repeat [??? times]' control block with a 'do' slot below it.

At the bottom left, there are language and version selection buttons: 'English' and 'Version: 2025'. At the bottom right, there are buttons for 'Blocks', 'Workspace', 'Start Over', and 'Show Code'.

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

A little bit about variables, values, and expressions

- Values/Objects
 - Numbers | -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 | -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

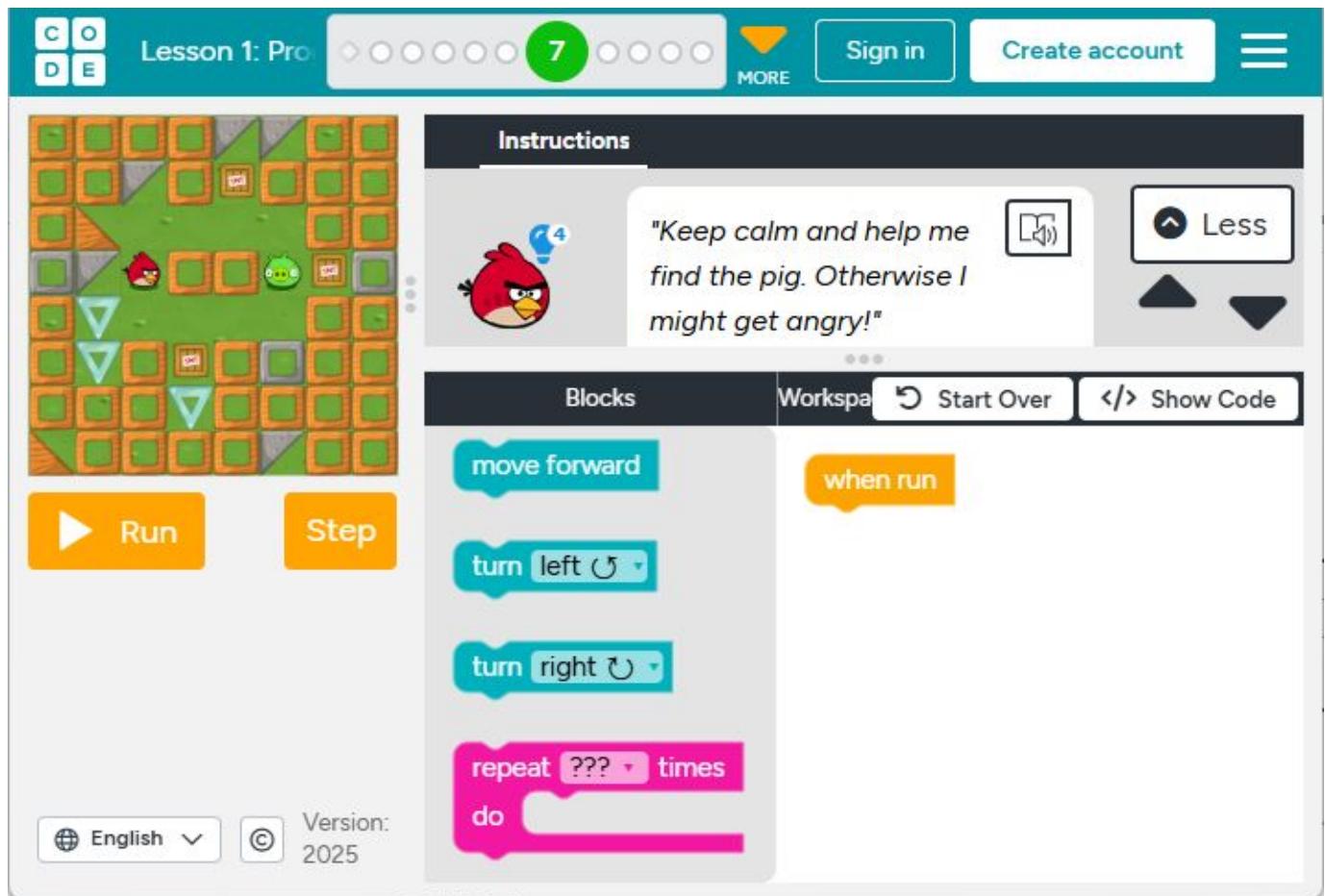
main.py	Run	Output	Clea
1 print(score) 2 score = 80 3 print(score)		ERROR! Traceback (most recent call last): File "<main.py>", line 1, in <module> NameError: name 'score' is not defined == Code Exited With Errors ==	

score is used after
having been defined

main.py	Run	Output	Cle
1 #print(score) 2 score = 80 3 print(score)		80 == Code Execution Successful ==	



More games



The image shows a Scratch game interface. At the top, there's a navigation bar with 'CODE' (in a 3x2 grid), 'Lesson 1: Pro' (with a progress bar at 7), 'Sign in', 'Create account', and a menu icon. Below the bar is a green stage featuring two Angry Birds characters: a red one on the left and a green one on the right, both facing towards the center. The stage has a grid of orange and green tiles. On the bottom left is an orange 'Run' button with a play icon, and on the bottom right is an orange 'Step' button. To the right of the stage is an 'Instructions' panel with a speech bubble containing the text: "Keep calm and help me find the pig. Otherwise I might get angry!" followed by a speaker icon and a 'Less' button with up and down arrows. Below the instructions are several Scratch script blocks:

- A teal 'move forward' block.
- A teal 'turn left' block.
- A teal 'turn right' block.
- A pink 'repeat [??? times] do' block.
- An orange 'when run' block.

At the bottom left of the stage area are language and version settings: 'English' and 'Version: 2025'. On the bottom right are buttons for 'Blocks', 'Workspace', 'Start Over', and 'Show Code'.

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



Loops

CODE Lesson 1: Pro 7 MORE Sign in Create account ⏓



Instructions

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

Blocks Workspace Start Over Show Code

```

when run
  turn right
  move forward
  turn left
  repeat (3) times
    do [move forward v]
    turn left
    move forward
end
  
```

English Version: 2025

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



Loops



main.py	Run	Output
<pre> 1 def move_forward(): 2 print('move forward') 3 def turn_right(): 4 print('turn right') 5 def turn_left(): 6 print('turn_left') 7 8 turn_right() 9 move_forward() 10 turn_left() 11 for i in range(3): 12 move_forward() 13 turn_left() 14 move_forward() </pre>		
		turn right
		move forward
		turn_left
		move forward
		move forward
		move forward
		turn_left
		move forward
		==== Code Execu

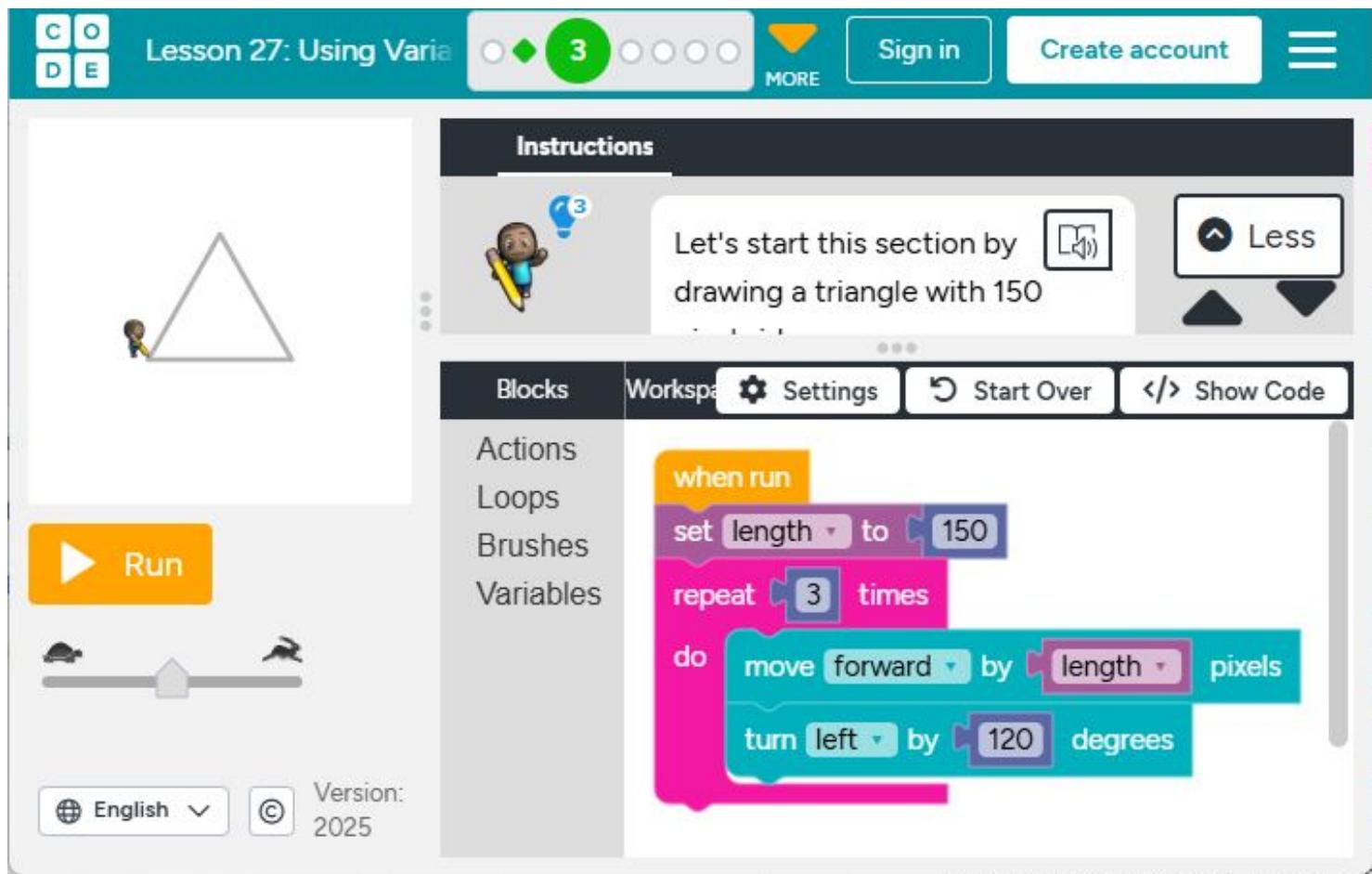
<https://studio.code.org/courses/express-2025/units/1/lessons/1/levels/7>

Repetition In Python

You can do other things in a for loop

main.py	Run	Output
<pre>1 def move_forward(): 2 print('move forward') 3 def turn_right(): 4 print('turn right') 5 def turn_left(): 6 print('turn_left') 7 8 turn_right() 9 move_forward() 10 turn_left() 11 for i in range(3): 12 move_forward() 13 print(i) 14 turn_left() 15 move_forward()</pre>		turn right move forward turn_left move forward 0 move forward 1 move forward 2 turn_left move forward *** Code Execution Successful ***

More games



The image shows a Scratch game interface titled "Lesson 27: Using Variables". At the top, there is a toolbar with icons for CO, O, D, E, a green circle with the number 3, and a "MORE" button. To the right are "Sign in" and "Create account" buttons, and a menu icon.

The main area is titled "Instructions". It features a cartoon character holding a pencil and a lightbulb icon with the number 3 above it. The text says: "Let's start this section by drawing a triangle with 150". To the right are "Less" and "More" buttons.

The workspace includes tabs for "Blocks", "Workshop", "Settings", "Start Over", and "Show Code". On the left, there are categories: Actions, Loops, Brushes, and Variables. The "Variables" category is currently selected, showing a variable named "length" set to 150.

The script area contains the following Scratch script:

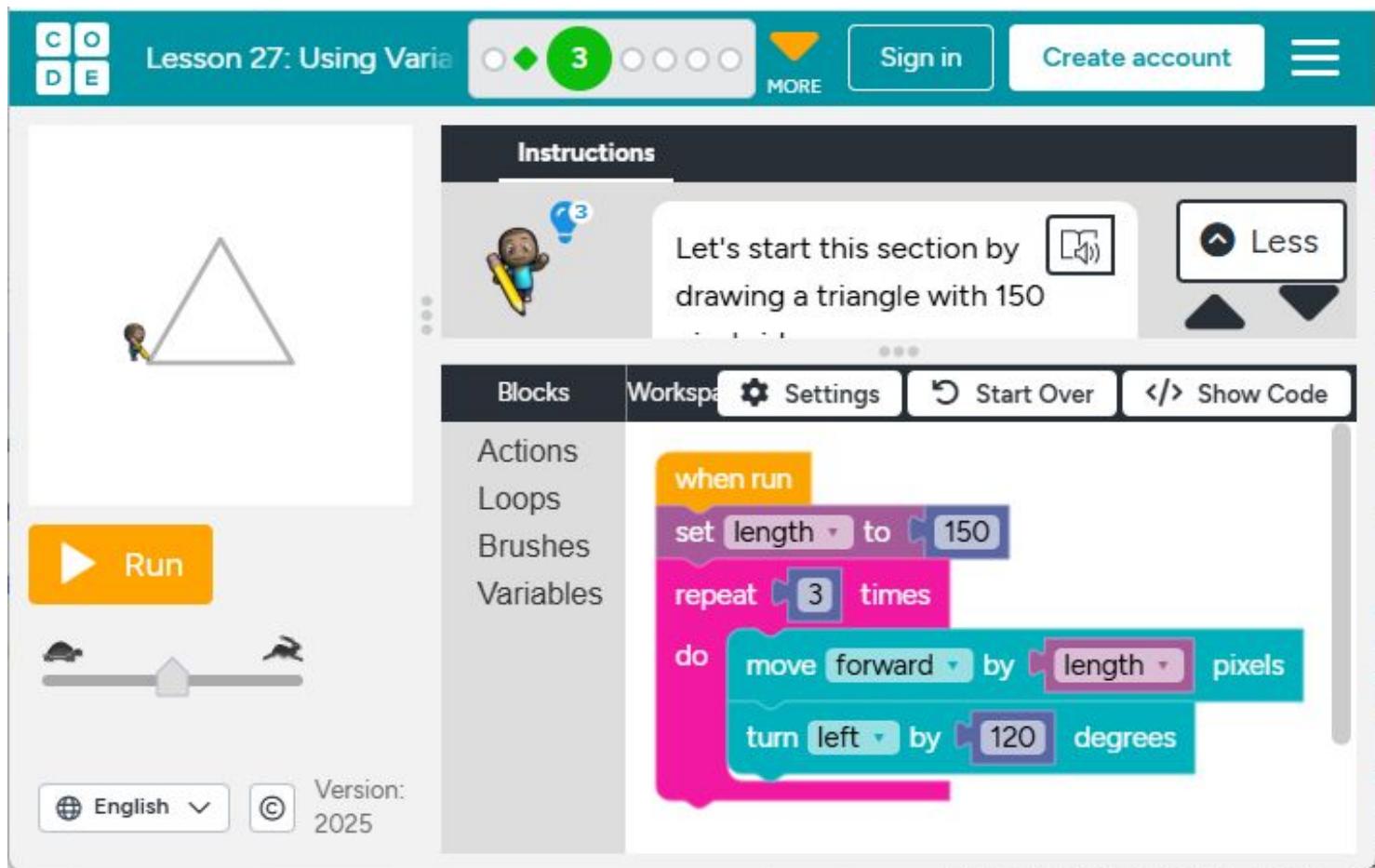
```
when run
  set [length v] to [150]
  repeat (3)
    do
      [move [forward v] by [length v] pixels]
      [turn [left v] by [120] degrees]
```

At the bottom left are language and version settings: "English" and "Version: 2025".

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



More games



The image shows a Scratch game interface titled "Lesson 27: Using Variables". At the top, there is a navigation bar with a "CODE" icon, the lesson title, a progress bar showing level 3, and buttons for "Sign in", "Create account", and a menu. On the left, there's a preview area showing a character drawing a triangle. Below it is a "Run" button and a slider. At the bottom left are language and version selection buttons ("English" and "Version: 2025").

Instructions:

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

Blocks:

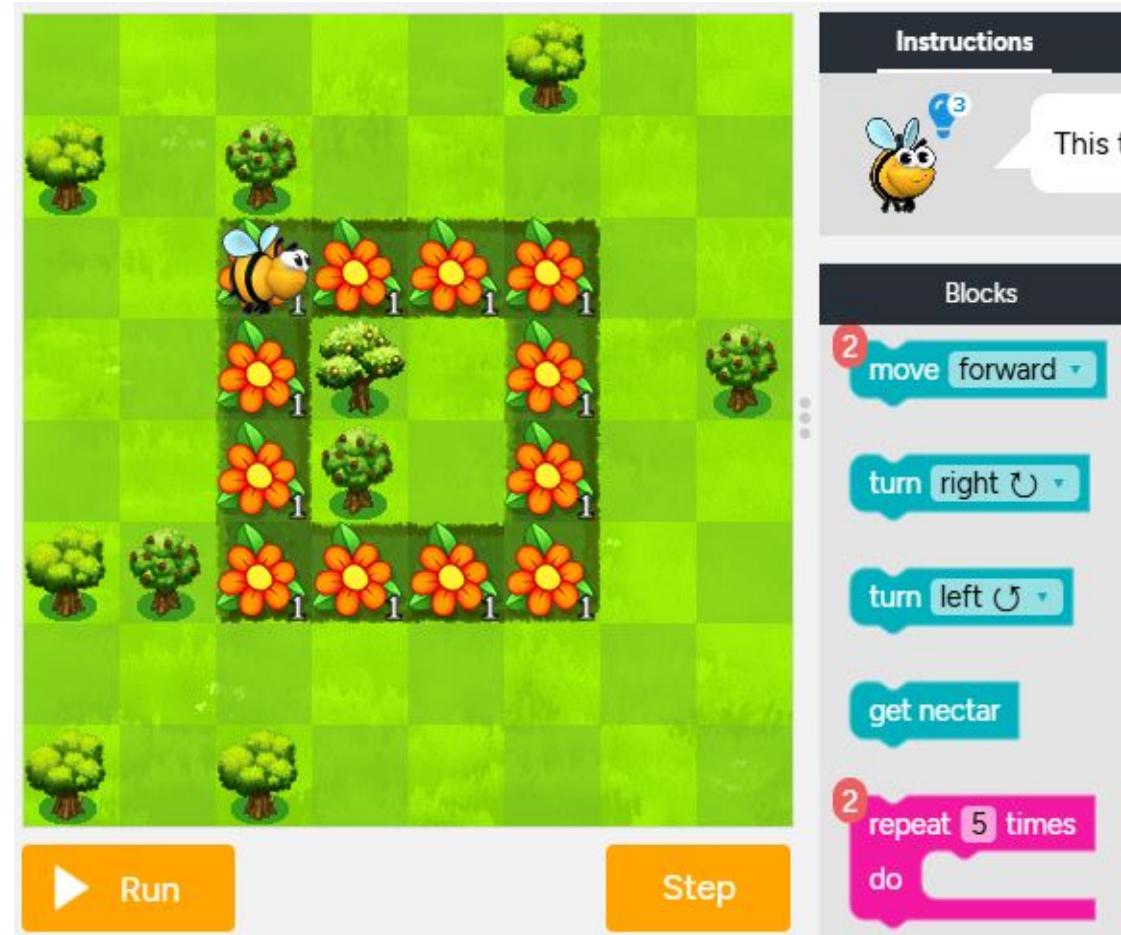
- when run
- set length to [150]
- repeat (3) times
 - do [move forward by <length> pixels]
 - turn left by [120] degrees

<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

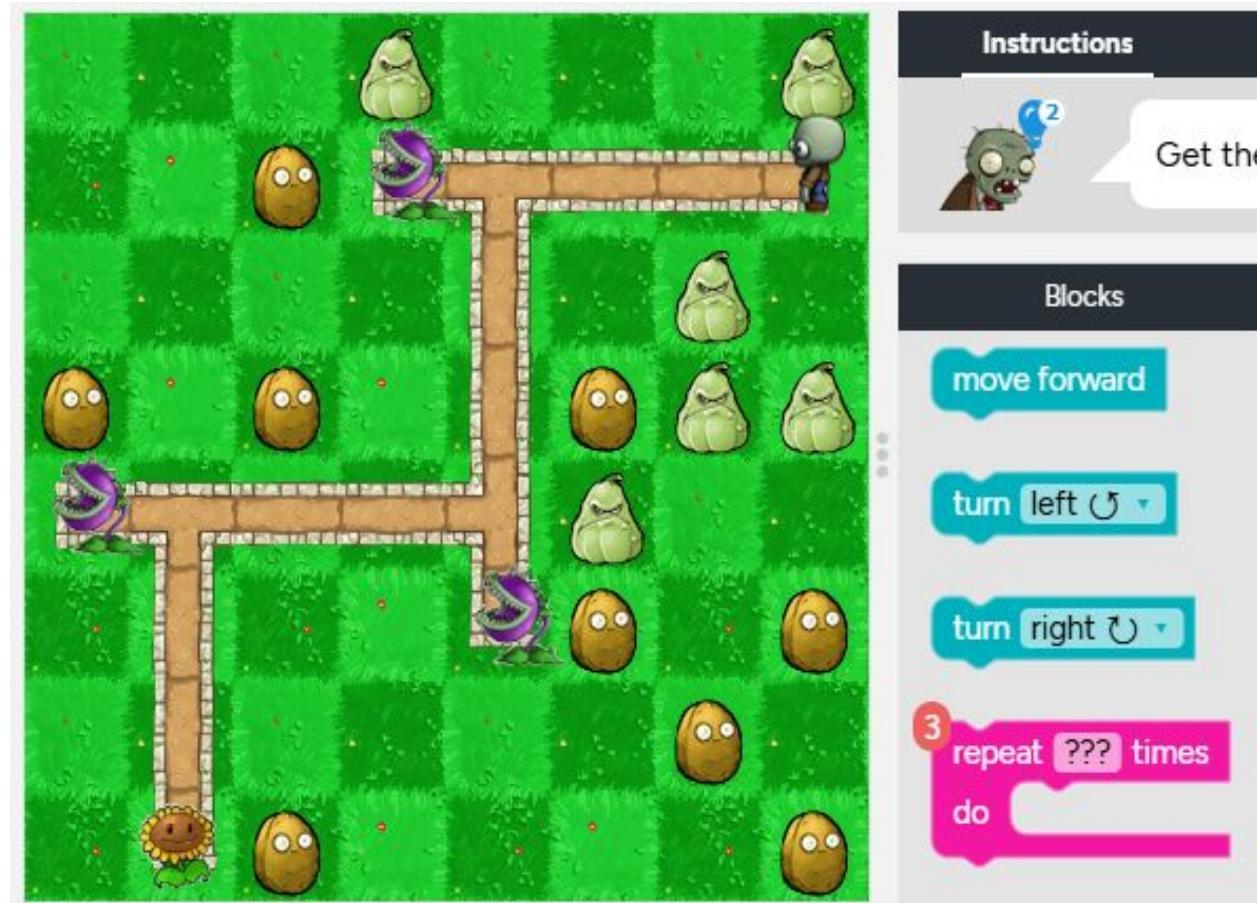


```
when green flag clicked
repeat (4)
  repeat (3)
    move 10 steps forward
    get nectar
  turn right
end
```

<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
```

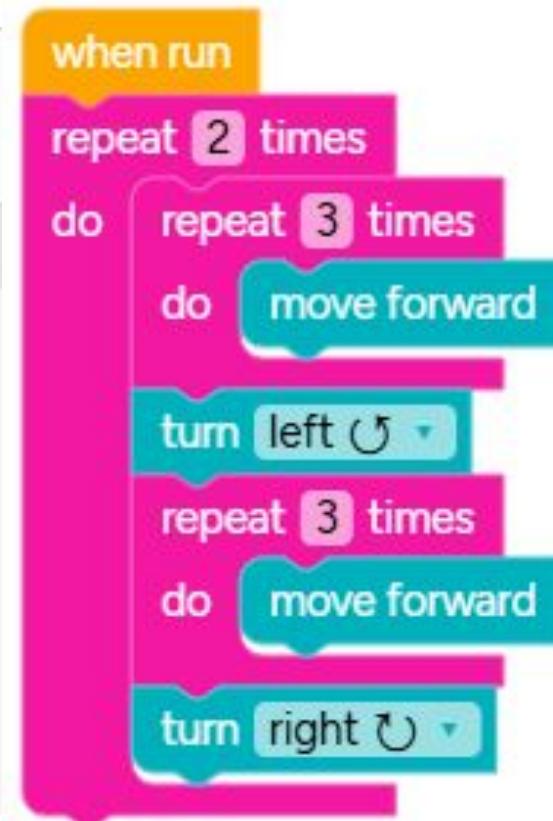
More complicated logic

main.py

```

1 for x in range(2):
2     for j in range(3):
3         move_forward()
4         turn_left()
5     for j in range(3):
6         move_forward()
7     turn_right()

```



Important point: Dividing into subtasks

- Divide main task into subtasks



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, 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.

The image shows a Scratch script for a bee character. The script starts with a 'when run' hat and a 'repeat (7)' control block. Inside the repeat loop, the bee moves forward and checks if it is at a flower. If yes, it gets nectar; otherwise, it makes honey. The background features a grid of trees and question mark clouds, with a row of flowers/honeycombs at the bottom.

```

when green flag clicked
repeat (7)
  move forward
  if at flower then
    get nectar
  else
    make honey
end

```

Available blocks on the right side:

- move forward
- turn right ↗
- turn left ↙
- get nectar
- make honey
- if at flower then
 do [] v
 else [] v
 end
- repeat (???) times
 do [] v

<https://studio.code.org/courses>

lessons/17/levels/12



Selection in Python

main.py	Run	Output
1 numbers = [10, 4, 3, 50]		even
2 for x in numbers:		even
3 if x % 2 == 0:		odd
4 print('even')		even
5 else:		
6 print('odd')		==== Code Execut

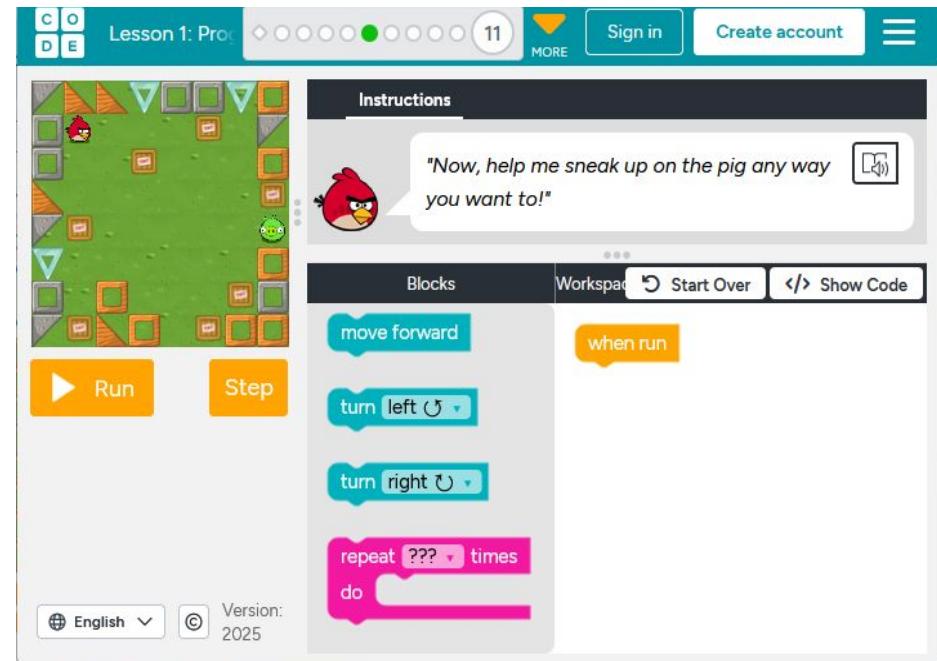
Scratch vs. Python

main.py	Run	Output	Scratch Script
1 numbers = [10, 4, 3, 50]		even	when run repeat (7) times do [move forward v. if at flower [get nectar v. else [make honey v.]])
2 for x in numbers:		even	
3 if x % 2 == 0:		odd	
4 print('even')		even	
5 else:			
6 print('odd')		==== Code	

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

So... what is programming?

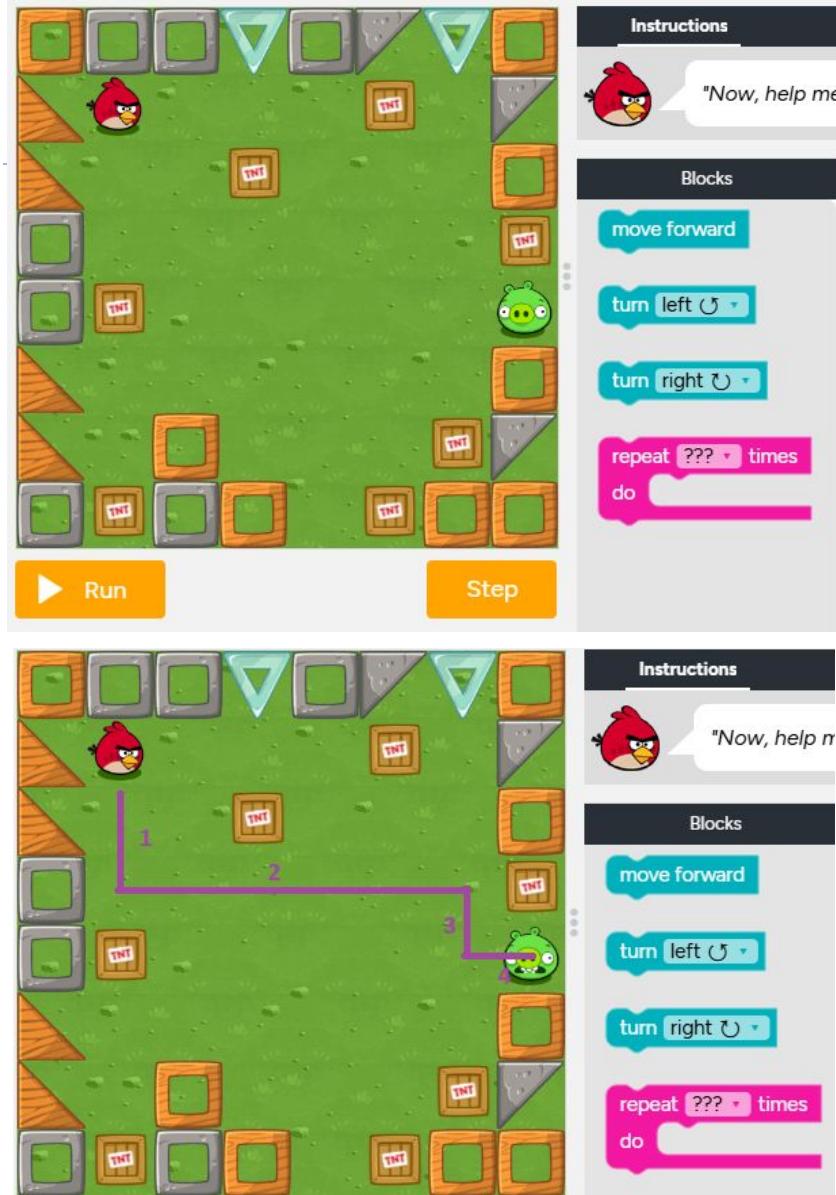
- In this game
 - Arrange the blocks in the right sequence
 - Run to see if it does the job correctly
 - See where it goes wrong
 - Fix the sequence
- In professional terms
 - Write some code
 - Test the code
 - Debug
 - Fix errors



<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.



<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!

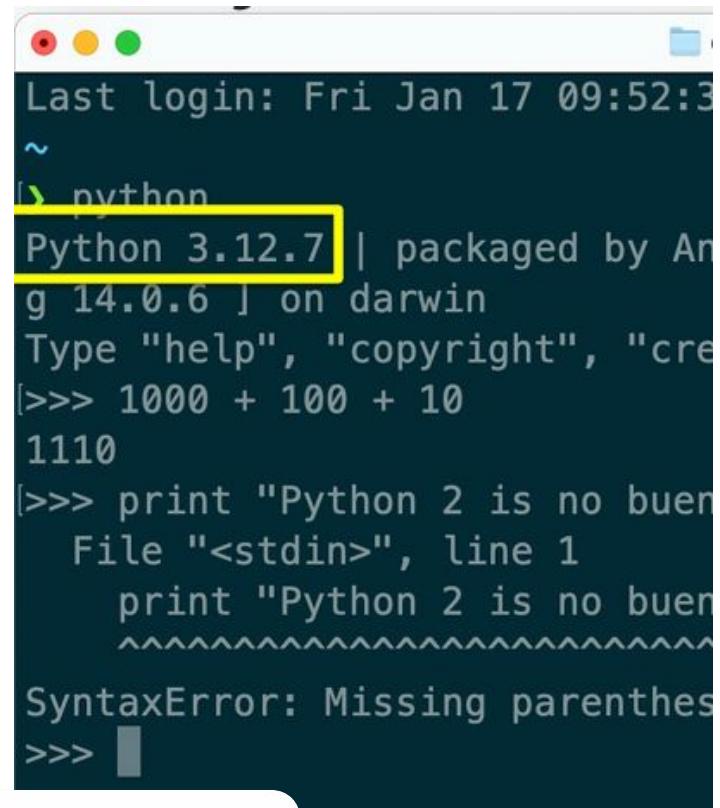


How to run a Python program



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 09:52:31 on ttys000
~
>>> python
Python 3.12.7 | packaged by Anaconda 14.0.6 | on darwin
Type "help", "copyright", "credits" or "license" for more information
[>>> 1000 + 100 + 10
1110
[>>> print "Python 2 is no bueno"
      File "<stdin>", line 1
        print "Python 2 is no bueno"
          ^
          ^
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean?
[>>> ]
```

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 + '.')
```

Command line:

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

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

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



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!

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