The Assignment

Tips:

- Stuck? Don’t worry! Learning takes time, and everyone goes at their own pace.
- Work together! Ask your neighbor for help.
- Can AI help you? Ask ChatGPT, but be mindful: don’t just copy the answer. Make sure you understand it too.

Basic version of the game

Steps:

1. Write code that welcomes the user with: “Welcome to our number guessing game”
2. Write code that asks the user: “Do you want to play a game? y/n”
3. Ask for the user's response — this should be “y” or “n”
4. If the answer is “y”, start the game
5. Have the computer choose a random number between 1 and 5
6. Ask the user to guess a number between 1 and 5
7. Store the user’s answer as a number
8. Compare the two answers:
    > a. If the user’s answer <u>does not match</u> the computer’s answer, tell them to try again
    > b. If it <u>does match</u>, congratulate them on winning the game!

Try it at: https://www.jdoodle.com/execute-ruby-online

Expand the game 🚀

Stappen:

1. At the start of the game, let the player set the range of numbers
2. Check if the answer to the first question is “y” or “yes” — do not make it case insensitive (e.g. “Yes” or “YES”)
3. Add a 2-second pause between the message “Welcome to our number guessing game” and the question “Do you want to play a game? y/n”
4. Check if the user already guessed a number before — let them know and ask for a new guess
5. Use methods to keep your code organized. For example, move the range setup to a separate method. See: https://launchschool.com/books/ruby/read/methods

Try it at: https://www.jdoodle.com/execute-ruby-online

Reverse the game 🔄

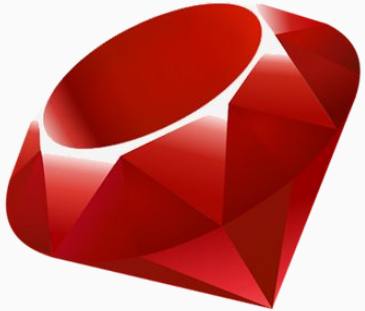
Instead of your guessing the number of your program, the program will guess your number.

Steps:

1. At the start of the game, let the player set the range of numbers
2. Check if the answer to the first question is “y” or “yes” — do not make it case insensitive (e.g. “Yes” or “YES”)
3. Ask the player for the range in which the number falls..
4. Let the program guess a number. If it was correct, answer y (yes), if not, answer h (higher) or l (lower).
5. The program continues adjusting its guesses until it finds the correct number.
6. Use methods to keep your code organized. For example, move the range setup to a separate method. See: https://launchschool.com/books/ruby/read/methods

Try it at: https://www.jdoodle.com/execute-ruby-online

Ruby Basics



Ruby

Yukihiro “Matz” Matsumoto

Created Ruby in 1993

Designed to make programming more fun and meant to be read like natural language

The Basis

- Programming = giving instructions to a computer to do things
- A computer is essentially dumb... → you must be very specific about what it should do
- Code is executed line by line

The Basis

Today we use two data types

Data type:

1. Integer (42, 1, 382) → numbers
2. String (“yes”, “no”, “weird things”, “sbeiuwei”, “I am a sentence”) → Letters en words

Storing Data

Variables

Storing data

Assign

my_answer = 42
my_name = “Michele”
my_street = “Hellostreet 21”

Retrieve

my_name # “Michele”

Data can be manipulated and stored:

box_one = 1
box_two = 2

box_three = box_one + box_two

→ box_three will return the number **3**!

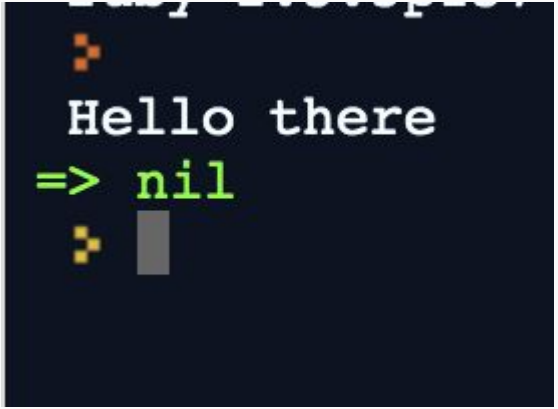
Language

Puts (put string)

puts “Hello there”

puts **“**Congratulations, you won!**”**
puts **“**Try again!**”**

In Ruby, we use `puts` to display text on the screen. It stands for “put string” — meaning: show (text) on the screen..



Language

Ask for Data

your_answer = **gets.chomp**

Important: Everything you type in a terminal is treated as a string (text). Use gets.chomp.to_i to store it as a whole number (integer).

```
What is your guess?
 42
=> "42"
```

Language

Computer answer

Choose a random number within range:

computer_guess = **rand(1..10)** → 1 to 10 inclusive

computer_guess = **rand(1...10)** → 1 to 10

Language

Comparing

box_one = 1
box_two = 10

box_one > box_two

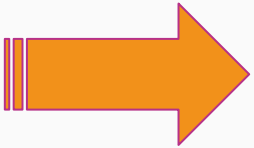
box_one < box_two

Ask the computer whether the integer in box one is greater than, less than, or equal to the integer in box two.

box_one <= box_two

box_one >= box_two

The computer tells whether it is true or not!



true

false

your_guess == computer_answer
your_guess != computer_answer

You can do the same with strings!
== → are the same
!= → they are not the same

Language

Conditions

```
if  Add a question condition (comparison)
  #do something
else
  #do something else
end
```

```
if  question condition
  #do something
elsif another question
  #do something
else
  #do something else
end
```

examples:

```
if  box_one == box_two
    puts "They are the same!"
else
    puts "They are different!"
end
```

OR

```
if  box_one != box_two
    puts "They are different!"
end
```

Language

While loop

Keeps asking the question until the answer is True

```
while  [insert a condition]
  #do something
end
```