Game Programming with Python

Marcus Birkenkrahe, Lyon College

March 1, 2025

Lyon College Discovery Day 2025 STEM workshop

- Link: `https://tinyurl.com/LyonPython2025`
- Includes: Script + Sample code + Game extension

Goals

- Understand basic game programming concepts
- Learn how to use JupyterLab Lite (Notebook IDE)
- Learn just enough of Python to build a game
- Build a simple "Guess the Number" game
- Hear about our new exciting programs at Lyon College
- Hear my views on Artificial Intelligence (AI)

Introduction - You and the Machine

- Today we build a simple game in Python using JupyterLab Lite - no setup required (other than a browser and Internet access).
- We're actually crossing disciplinary boundaries here: A "discipline" in science is a field of inquiry or a field of study like computer science.
- When you do anything in computer science, you're immediately in the middle of a huge ocean, in a tiny boat, with no paddle, and you have to make your own wind.

Figure 1: Computer science: Alone on the ocean.

- This is partly because computer science is so young - it's barely a science after fewer than 100 years of existence. By comparison, we've done physics for (at least) 2,500 years, and mathematics for even longer than that.
- But being in the middle of an ocean with nothing but your clothes on and no plan is also incredibly exciting, of course. To make it here, you have to draw on everything you've got.
- I forgot one thing: You've got a companion in the boat with you. The machine, the computer. If it's your friend depends entirely on your skill.

Figure 2: You're not alone if you have a machine

- That's what we teach at Lyon College: These survival skills to befriend and use a machine to be your trusty helper and solve amazing problems.
- The problem: The computer is a dumb dog that needs to be brought to life, and when it is alive, it will never be smart, never understand anything, and you have to treat it like a dumb dog forever.

Games were Midwives to Computers

Figure 3: Dennis Ritchie and Ken Thompson, creators of C and Unix (1970)

- So how do games come into that? Well, it turns out that games stood at the very beginning of computer science: Both the programming language C and the operating system Unix, two technologies that lead, among other things to Python (written in C), and to Linux, were created so that their creators could play games.
- There are two important principles here already:

1. **Portability:** You program a game on one computer but you want to play it on many computers.
2. **Modularity:** When you program a movement, or a transaction, you want to reuse it many times over.
3. **Efficiency**: You don't want to play your game alone, and you want to play games that are as close to real life as possible. To pull this off, the machine needs to be fast.
4. **Open source:** The more developers you have, people who help you improve your game, the faster you can build better games. This is easier if the source code of your programs is open to everyone.

- To make realistic games, you need a lot more than a language and an operating system: You need a story, design, a knowledge of physics, and for larger games, you need to manage a lot of people.
- Today, however, we're only going to put our toe in the water. Let's start by exploring our computing environment.

Interactive Development Environments (IDE)

- To program, you usually need a few things: An editor to write your source code, a compiler to turn your source code into machine code that the computer can understand, and an operating system to run all of these and then show you the results.
- For a short workshop with beginners, this so-called "computing infrastructure" is a tall order. Fortunately, there is an alternative: Interactive Development Environments (IDEs) allow you to write and evaluate your code in one place.
- For some languages like Python or R (one of my favorites), it gets even better: You can use interactive notebooks to write and run your code line-by-line rather than having to create a program first.
- This is a little like learning to write a formal letter if you've never done it before: It is easier to write one sentence and then ask your dad or your mom for input than writing the whole letter in one go.
- In computing, both is done and done often: Python is a language that allows you to write code line by line, while C is a language that forces

you to write a whole program. C (in the form of C++) is much, more relevant for professional game programming than Python, but it's also much harder.

- At Lyon, if you study either computer or data science, you learn a lot of different languages, including Python, C, C++, and R, but also SQL for database management, and shell languages for command-line programming that you need for cybersecurity.
- To circumvent having to create an account, we use an open and free environment called Google Colaboratory or "Colab". To use it you do need a Google account, alas.
- Log into `https://colab.research.google.com` for a short demonstration.

Python Coding Basics and Demo

- Since you've already got the notebook open you can code along with me if you're fast, or you can just watch and do it later.
- This is very much what you'd do if you came to Lyon to study with us: The classes are all very interactive, and the emphasis is on doing stuff, not on sitting and listening or talking.
- You've probably heard the saying "Give a man a fish and he eats for a day - teach a man to fish and he eats for a lifetime." What we do here is teach you to fish - but you also don't go hungry!
- What follows are the basic building blocks not just of game programs, but of all programs. I'll give them to you as an overview and so you can try them out, and then we'll put them in a program.

Python Basics (Live Demo)

In the workshop, I'll show these commands live for you to code along. In the script, I provide more technical background - this is what you'd learn if you took an introductory programming class.

- **Variables**: These are boxes to store stuff, like a number or a name.

```
box = 7                          # a numeric variable
secret_number = 9                # another numeric variable
box, secret_number               # priting two variables as a tuple
name = "Marcus"                  # defining a string variable
print(name)                      # printing a variable (reusable)
print("Marcus")                  # printing a string (immutable)
```

- **Input**: This lets players type something.

```
input()
input("Enter a number: ")          # returns string
int(input("Enter a number: "))     # returns integer
guess = int(input("Enter a number: "))  # store in variable ‘guess‘
print(guess)                       # print ‘guess‘
```

- **If-statements**: The computer decides what to do based on your guess.

```
secret_number = 7
guess = 7

if guess == secret_number:
    print("You win!")
else:
    print("Try again!")
```

- **Loops**: Keep asking until they guess right.

```
guess, secret_number  # for example (5,9)

while guess != secret_number:            # stops when 9 is entered
    guess = int(input("Guess again: "))
```

- **Live Demo**: Simple version of the game for demonstration.

```
secret_number = 5
guess = int(input("Guess a number between 1 and 10: "))
while guess != secret_number:
   if guess < secret_number:
```

```
        print("Too low!")
    else:
        print("Too high!")
    guess = int(input("Guess again: "))
print("You got it!")
```

Hands-On Coding (20 Minutes)

- Let's build the game together! You'll type code in Google Colab, run it, and play it.
- Instructions
  1. Open Google Colab: `https://colab.research.google.com`
  2. Start a new notebook (or copy from my shared link).
  3. Follow these steps to code the game.
- Steps to Build the Game
  1. Step 1: Set up the secret number
     - Explanation: We'll store a number the player has to guess.

     ```
     secret_number = 7  # You can change this later!
     ```

  2) Step 2: Ask for a guess

  - Explanation: Use `input()` to let the player type their guess.

    ```
    guess = int(input("Guess a number between 1 and 10: "))
    ```

  - Step 3: Keep asking until they get it
    - Explanation: Use a `while` loop and `if` statements to check the guess.

      ```
      while guess != secret_number:
          if guess < secret_number:
      print("Too low!")
          else:
      print("Too high!")
      guess = int(input("Guess again: "))
      ```

  - Step 4: Celebrate!
    - Explanation: Tell them they won when the guess matches.

```
print("Congrats! You guessed the secret number!")
```

- Full Code: Paste this into a Colab cell and run it (Shift+Enter):

```
# Guess the Number Game
secret_number = 7  # Change me!
guess = int(input("Guess a number between 1 and 10: "))
while guess != secret_number:
    if guess < secret_number:
print("Too low!")
    else:
print("Too high!")
    guess = int(input("Guess again: "))
print("Congrats! You guessed the secret number!")
```

- Tips
  - Indentation matters! Use 4 spaces under `while` and `if`.
  - Test it: Run the code and guess a few times.
  - If you get errors (e.g., "invalid literal"), check `int()` is around `input()`.
- Challenge (Optional)
  - Change the `secret_number` to a different value.
  - Edit the hints (e.g., "Way off!" instead of "Too low!").
  - Use `random` to present a random secret number (link)