LUCIANO GABBANELLI

GAMES IN PYTHON

1. Introduction

The main objective of this project is the creation of 3 very simple and well-known games. They are totally solvable with the content seen in this module of Python. The main tools for creating these games are:

- Random libraries to generate random variables that correspond to the algorithm's choices to compete with the player
- Built-in Python objects to store information and player movements
- Control flow statements to decide who wins
- Think and design a clean code syntax
- Encapsulate your code! For instance, if you choose a segmentation with functions, break down the game into multiple, well-defined functions rather than trying to minimize the number of functions used

2. Objectives

- Create a Jupyter Notebook from scratch called "***Project_games.ipynb***."
- Its content must have the algorithms to play the 3 proposed games.
- The games to be created are the following:

    1) **Guess a number from one to ten!** The user tries numbers from one to ten and the program tells him if he got it right or not. The program ends when the user guesses.

        a) **Rock paper scissors.** A digital version of the famous hand game originating from China, usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand.

    2) **Rock paper scissors lizard Spock.** Five-weapon expansion of rock paper scissors invented by Sam Kass and Karen Bryla.

        Documentation: Rock Paper Scissors Lizard Spock

3. General analysis

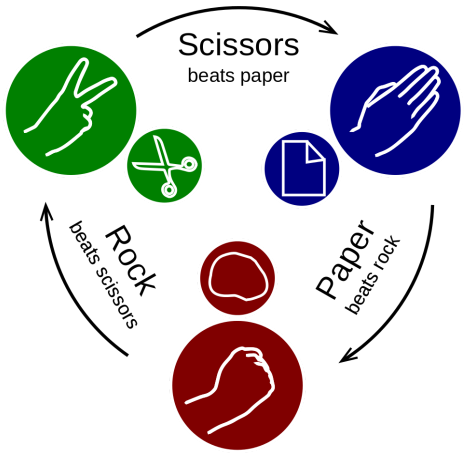
Read carefully the rules of each game. The rules must be strictly respected.

1) **Guess a number from one to ten!**

   The program explains the rules of the game, chooses a number and asks the user for a number from 1 to 10. If the user guesses right, he wins. Otherwise, the program reports that the number is not correct and asks for a new number. The user tries again until successful. A warning message should be printed after the fifth attempt encouraging the user to think more carefully. Once the number is guessed, the program tells the user how many attempts he has made until guessing.

2) **Rock paper scissors**

   The program asks the user to choose between rock, paper, scissors and chooses one of them to play with the user. The rules of who wins once the elections are made can be found in the following figure:

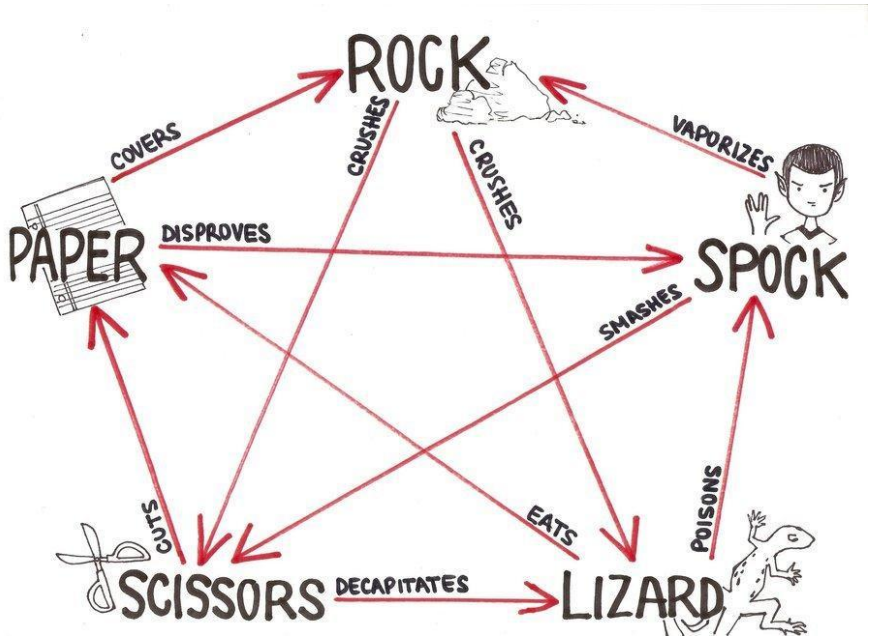


   The program must determine who wins. The user and the computer play 3 times; whoever wins the most number of games wins.

3) **Rock paper scissors lizard Spock**

   The rules goes as follows: Rock Paper Scissors Spock Lizard

   Information is summarized in the following figure:



At the end of the game, the algorithm must ask if the user wants to play again. If the user answers yes, the game restarts; if not, then the algorithm ends.

4. Project organization

It is important to document your own work in each project you do. In this way you can easily after a time read your own resume of what you did in each project.

The project needs to be solved in a single Jupyter notebook called "***Project_games.ipynb***" which you need to create yourself.

Your code **MUST** include the following:

- **Markdown** cells will have the instructions to play the games. If a user opens the notebook, the Markdown cells should have enough information for the user to have fun.
- **Comprehensive docstrings and type hints are essential.** Explore documentation and resources to familiarize with these concepts. You'll probably need something like this: `from typing import Dict, List`
- **Comments in code cells** (remember that code comments start with a hashtag, i.e. #) should have information about how the game works internally. If a programmer opens your code, he must understand how the program flows by reading your comments.

5. Requirements

- To initiate the game, simply execute a one-line-command cell. If your code is structured using functions, the gameplay cell should only contain a command such as: `play_rpsls()`

- Ensure you structure your code with an appropriate number of functions for clarity and maintainability. **Relying solely on one or two functions will not suffice.**

- It is essential that you add comments on the most important elements during the development of the project. Markdown cells or comments in code cells.

- All code, including comments, must be written in English.

- All files that were used in the creation of the project as well as its solution must be attached.

- Delete files that are not used or are not necessary to evaluate the project.

6. Deliverables

To have your project evaluated, please submit the following files to your Google Classroom assignment.

Each file contributes to the final grade!

- A Jupyter notebook called "***Project_games.ipynb***".

- An **.html** version of the Jupyter notebook for the project documentation.

- A "***Readme.md***" file is crucial, offering insights into your project's development and guiding users on its usage. Learn how to create one!

  Please, add at the end a **short record of incidents** that were detected during project execution:

  - Explain what you have learned during this project.
  - Conclusions you have obtained from the analyses carried out.
  - What problems have you encountered when developing this project?

- A **video** in which you must clearly explain the items described below.

  If working individually, cover only one of the games; however, if they share code, address all three to the extent they are interconnected.

  - How did you organize the project and why.
  - What did you find most difficult about this project.
  - What have you learned in this project.
  - You must explain your code, how you have structured it (for example if you chose to encapsulate its parts in functions, how do they work together) and what is the advantage of the chosen structure.
  - Do not discuss the rules or any details that waste time!

  Requirements:

  - The video should last around **5 minutes**.
  - Think first about what you want to say, make sure to get to the point since it's a short video.
  - You have to be visible in the video.

The project ends here. If you have time, you can find a surprise below!

7. Extension for PROS!

Modularizing Code with `.py` Files:

Objective: Master the art of organizing and structuring your Python code by isolating preparatory game logic into modular `.py` files. Then, initiate the game(s) in your primary Jupyter Notebook or other Python scripts using just a single function call to play: `play()`

Understanding Modularization:

- Why? Modularizing code makes it cleaner, more manageable, and reusable.
- How? By separating different parts of your code into functions and classes, and storing them in a `.py` file.

Best Practices:

- **Proper Usage:** Consider which file you should import libraries into: the `.py` or the `.ipynb`? Think on scopes!
- **Encapsulation:** Aim to encapsulate distinct functionalities into distinct functions or classes.
- **Descriptive Naming:** Ensure that your files, functions, and classes have descriptive names that clearly convey their purpose.
- **Documentation:** Incorporate comments and docstrings in your `.py` file to explain the purpose and usage of functions or classes.

Benefits:

- **Cleaner Notebooks:** By offloading the bulk of the code to `.py` files, your notebooks become more focused and readable.
- **Reusability:** Once written in a .py file, functions and classes can be easily reused in multiple notebooks or scripts.

If you still have time, you can find another surprise below!

8. Extension for SUPER PROS!

The Mysterious `__init__.py` File

Have you ever stumbled upon some directories in a repository that contain a seemingly blank file named `__init__.py`? 🤔

You might wonder, "Why would anyone put an empty file in a directory on purpose? Are they just trying to look cool?" Well, time to unveil the mystery!

This file, although often empty, has a very important role: It's like putting up a sign on a door that says, "Hey Python, there's something cool in here!"

Your Quest:

1. **Discovery Phase 🔮 :** Structure your project in a way that demands the presence of `__init__.py`.

2. **Experiment Phase 👾 :** Remove the `__init__.py` file temporarily and try to import your package/module. Witness the power of the file by observing the chaos that ensues in its absence!

3. **Mastery Phase 🎩 :** Even though it can be empty, sometimes developers add initialization code or metadata inside. Try putting some greeting messages or version info in your `__init__.py`, and see if you can print them when the package is imported.

4. **Epic Codemancer's Riddle Phase 🐇 :** For those seeking the ultimate challenge, find out the other powers of `__init__.py`. Can you control what gets imported with the `from package import *` statement? Dive deep, and find out!