INTRODUCTION TO PYTHON

- **Python** is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.[33]
- Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.[34][35]
- Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0.[36] Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2.[37]
- Python consistently ranks as one of the most popular programming languages, and has gained widespread use in the machine learning community.[38][39][40][41]

History

- The designer of Python, Guido van Rossum, at OSCON 2006
- *Main article: History of Python*
- Python was invented in the late 1980s[42] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL,[43] capable of exception handling and interfacing with the Amoeba operating system.[12] Its implementation began in December 1989.[44] Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life" (BDFL), a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker[45] (he's since come out of retirement and is self-titled "BDFL-emeritus"). In January 2019, active Python core developers elected a five-member Steering Council to lead the project.[46][47]

- Python 3.12 adds syntax (and in fact every Python since at least 3.5 adds some syntax) to the language, the new (soft) keyword `type` (recent releases have added a lot of typing support e.g. new type union operator in 3.10), and 3.11 for exception handling, and 3.10 the `match` and `case` (soft) keywords, for structural pattern matching statements. Python 3.12 also drops outdated modules and functionality, and future versions will too, see below in Development section.
- Python 3.11 claims to be between 10 and 60% faster than Python 3.10, and Python 3.12 adds another 5% on top of that. It also has improved error messages, and many other changes.
- Since 27 June 2023, Python 3.8 is the oldest supported version of Python (albeit in the 'security support' phase), due to Python 3.7 reaching end-of-life.[66]
- Python 3.13 introduced an *incremental* (shorter pauses) garbage collector, an experimental JIT compiler;[67], and removals from the C API. Some standard library modules, 19 *dead batteries*, and many deprecated classes, functions and methods, and more will be removed in Python 3.15 and or 3.16.[68] Starting with 3.13, it and later versions have 2 years of full support (up from one and a half); followed by 3 years of security support (for same total support as before).

Acknowledgment

- I would like to express my deepest gratitude to our professor **DR.Gaurav Vinchurkar** who contributed to the successful completion of this Python board game project.
- First and foremost, I extend my sincere thanks to [Instructor's Name] for their invaluable guidance, support, and encouragement throughout this project. Their insights and expertise were instrumental in navigating the complexities of game development.
- I am also grateful to my peers and colleagues who provided helpful feedback and suggestions during the development process. Their constructive criticism and collaborative spirit significantly enhanced the quality of the final product.
- A special thank you goes to the open-source community for the wealth of resources, libraries, and tools available for Python development. Projects such as Pygame, NumPy, and many others provided the essential building blocks for this game.
- Additionally, I appreciate the support of my family and friends, who were understanding of the time and effort required to bring this project to fruition.
- Finally, I would like to acknowledge the role of the various online forums and communities where I found solutions to numerous challenges and where the collaborative spirit of programming truly shines.
- Thank you all for your support and contributions.

WHY I CHOSE THIS PROJECT?

- **1. Personal Passion and Enjoyment**
- My favorite board game has always provided me with immense enjoyment and countless hours of fun. The idea of recreating this beloved game in a digital format was incredibly appealing. It allowed me to blend my personal passion for the game with my enthusiasm for programming, making the project both enjoyable and fulfilling.
- **2. Deepening Understanding of Game Mechanics**
- Recreating a game I am deeply familiar with allowed me to delve into its mechanics, rules, and strategies on a deeper level. This provided a unique opportunity to understand and appreciate the intricacies of game design, from balancing gameplay to ensuring fair play and user engagement.
- **3. Python's Versatility and Ease of Use**
- Python is renowned for its simplicity, readability, and versatility, making it an ideal choice for developing a wide range of applications, including games. Its straightforward syntax and extensive libraries, such as Pygame, made it easier to implement complex game mechanics and create a visually appealing interface.

- **4. Educational Value**
- Developing a board game involves numerous programming concepts and skills, such as object-oriented programming, event handling, and graphical user interface (GUI) design. This project provided a comprehensive learning experience, allowing me to apply these concepts in a practical context and enhance my overall programming proficiency.
- **5. Leveraging Community Resources**
- The Python community offers a wealth of resources, including tutorials, documentation, and forums. Libraries like Pygame are specifically designed for game development, providing powerful tools for creating graphics, managing user input, and handling game logic. These resources significantly streamlined the development process and made Python an attractive choice for this project.
- **6. Creative and Problem-Solving Challenges**
- Recreating my favorite board game presented both creative and technical challenges. From designing the user interface to implementing game rules and ensuring an intuitive user experience, the project required innovative problem-solving and creativity. These challenges were intellectually stimulating and rewarding to overcome.

- **7. Showcasing Skills and Portfolio Development**
- Successfully completing this project not only showcases my programming skills but also my ability to manage and execute a complex project. Including a fully functional game in my portfolio demonstrates my proficiency in software development and game design, which can be advantageous for career opportunities in these fields.
- **8. Nostalgia and Sharing with Others**
- Recreating a game that holds sentimental value allowed me to experience a sense of nostalgia and personal satisfaction. Moreover, sharing the digital version of my favorite board game with friends, family, and the wider community brought joy and fostered a sense of connection through a shared appreciation of the game.
- In summary, choosing to recreate my favorite board game in Python was driven by a combination of personal passion, the desire to deepen my understanding of game mechanics, the educational benefits of using Python, the support of a rich community, the creative and problem-solving challenges, and the opportunity to showcase my skills and share my work with others.

LIBRARIES USED:-

Pygame

- **Pygame**
- Pygame is a set of Python modules specifically designed for writing video games. It provides essential functionalities that simplify the development process, allowing developers to focus more on game design and logic. Here are some key features of Pygame and how they were utilized in creating the board game:
- **1. Graphics Handling**
- Pygame makes it easy to create and manipulate graphics, which is essential for designing the visual components of the game such as the board, pieces, and background. Key functions include:
- **pygame.image.load()**: Loads images from files, enabling the use of custom graphics.
- **pygame.Surface.blit()**: Draws one image onto another, crucial for rendering the game scene.
- **pygame.draw.rect()**, **pygame.draw.circle()**: Draws shapes directly onto surfaces, useful for creating game elements dynamically.

- **2. Event Handling**
- Managing user inputs is crucial for any interactive game. Pygame's event handling system manages keyboard and mouse inputs efficiently:
- **pygame.event.get()**: Retrieves a list of all events, allowing the game to respond to user actions.
- **pygame.KEYDOWN**, **pygame.KEYUP**: Detects key presses and releases, enabling keyboard control.
- **pygame.MOUSEBUTTONDOWN**, **pygame.MOUSEBUTTONUP**: Det ects mouse button events, allowing for mouse interaction.
- **3. Sound and Music**
- Adding audio elements like sound effects and background music enhances the gaming experience. Pygame's audio module supports loading and playing sound files:
- `pygame.mixer.Sound()
- `: Loads and plays sound effects, essential for adding auditory feedback to user actions.
- **pygame.mixer.music.load()**: Loads background music tracks.
- **pygame.mixer.music.play()**: Plays the loaded music, adding ambiance to the game.

- **4. Sprites and Groups**
- Pygame's sprite classes simplify the management of game objects:
- **pygame.sprite.Sprite**: The base class for all game objects.
- **pygame.sprite.Group**: A container class to manage multiple sprites, allowing collective operations like updates and collision detection.
- **5. Collision Detection**
- Efficient collision detection is crucial for gameplay mechanics:
- **pygame.sprite.collide_rect()**: Checks for rectangle collisions between sprites.
- **pygame.sprite.collide_circle()**: Checks for circular collisions.
- **pygame.sprite.groupcollide()**: Checks for collisions between groups of sprites.

NumPy

NumPy is a powerful library for numerical computing in Python. It provides support for arrays and matrices, along with a collection of mathematical functions to operate on these data structures. In the context of developing a board game, NumPy can be used for various purposes:

- **1. Efficient Data Management**
- NumPy arrays provide an efficient way to handle and manipulate large
- datasets, such as the game board or state configurations. They are more efficient and faster than Python lists, which is crucial for performance in games:
- **numpy.array()**: Creates an array, allowing for efficient storage and manipulation of game data.
- **numpy.reshape()**: Reshapes arrays, useful for managing multi-dimensional game boards.
- **2. Mathematical Operations**
- NumPy provides a wide range of mathematical functions that can be used to implement game logic and mechanics:
- **numpy.sum()**, **numpy.mean()**: Perform quick calculations on arrays, useful for evaluating game states or scoring.
- **numpy.random.choice()**: Generates random selections, which can be used for random events or AI moves in the game.

- **3. Logical Operations**
- NumPy supports logical operations that are useful for checking game rules and conditions:
- **numpy.where()**: Locates elements that satisfy a given condition, helpful for finding empty spaces on the board or checking for winning conditions.
- **numpy.logical_and()**, **numpy.logical_or()**: Combine multiple conditions, useful for complex game logic.
- **4. Performance**
- NumPy's operations are implemented in C, making them much faster than standard Python loops. This performance boost is crucial for real-time game processing where speed is essential:
- Vectorized operations: Perform batch operations on arrays without explicit loops, significantly improving execution speed.

SOURCE CODE :-

```
#Snake and ladder game using Python


#importing all the required modules
import time
import random
import sys
Import Pygame
Import Numpy



#function for printing the text for player turn
text_for_plr_turn = [
    "Its your turn. Hit enter to roll the dice.",
    "Are you prepared?",
    "Lets Go.",
    "Please move along.",
    "You are doing great.",
    "Ready to be a champion.",
    "",
]
```

```
#function for printing the text for snake bite
text_for_snake_bite = [
    "boom!",
    "bang!",
    "snake bite",
    "oops!",
    "dang",
    "oh shit",
    "alas!"
]

#function for printing the text for ladder jump
text_for_ladder_jump = [
    "yipee!",
    "wahoo!",
    "hurrah!",
    "oh my Goodness...",
    "you are on fire",
    "you are a champion",
    "you are going to win this"
]
```

```
# Dictionary containing snake bite positions
#Snake Postions where key is the head of snake and value is the tail of snake
snake_position = {
    12: 4,
    18: 6,
    22: 11,
    36: 7,
    42: 8,
    53: 31,
    67: 36,
    73: 28,
    80: 41,
    84: 53,
    90: 48,
    94: 65,
    96: 80,
    99: 2
}
```

- #Ladder Positions where key is the bottom of ladder and value is the top of ladder
- ladders_position = {
- 3: 26,
- 5: 15,
- 13: 44,
- 25: 51,
- 29: 74,
- 36: 57,
- 42: 72,
- 49: 86,
- 57: 76,
- 61: 93,
- 77: 86,
- 81: 98,
- 88: 91
- }

```
#function for printing the rules of the game
def first_msg():
    msg = """"""

    print(msg)

# Delay of 1 second between each action
SLEEP_BETWEEN_ACTIONS = 1
MAX_VAL = 100
DICE_FACE = 6
#function define for taking input (Name of player) from user
def get_player_names():
    p1_name = None
    while not p1_name:
        p1_name = input("Name of first player: ").strip()

    p2_name = None
    while not p2_name:
        p2_name = input("Name of second player: ").strip()

    print("\n" + p1_name + " and " + p2_name + " You will play against each other'\n")
    return p1_name, p2_name
```

```
#function define for snake bite
    def got_snake_bite(old_value, current_value, player_name):
        print("\n" + random.choice(text_for_snake_bite).upper() + " ~~~~~~~~>")
        print("\n"" + player_name + " got a bite from snake. Going down from " + str(old_value) + " to " + str(current_value))
    #function define for ladder jump
    def got_ladder_jump(old_value, current_value, player_name):
        print("\n" + random.choice(text_for_ladder_jump).upper() + " ########")
        print("\n" + player_name + " is clibing the ladder from " + str(old_value) + " to " + str(current_value))
    #function define for snake and ladder
    def snake_ladder(player_name, current_value, dice_value):
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        old_value = current_value
        current_value = current_value + dice_value
        if current_value > MAX_VAL:
            print("You need " + str(MAX_VAL - old_value) + " to win this game. Keep trying.")
            return old_value
    )
```

```
    print("\n" + player_name + " moved from " + str(old_value) + " to "
+ str(current_value))
    if current_value in snake_position:
        final_value = snake_position.get(current_value)
        got_snake_bite(current_value, final_value, player_name)
    elif current_value in ladders_position:
        final_value = ladders_position.get(current_value)
        got_ladder_jump(current_value, final_value, player_name)
    else:
        final_value = current_value
    return final_value


#function define for checking the winner
def check_win(player_name, position):
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    if MAX_VAL == position:
        print("\n" + player_name + "has  won the game.")
        print("Congratulations" + player_name +"You are the winner")
        sys.exit(1)
```

```
    #function define for playing the game
def start():
    first_msg()
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    p1_name, p2_name = get_player_names()
    time.sleep(SLEEP_BETWEEN_ACTIONS)


    p1_current_position = 0
    p2_current_position = 0
 while True:
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        input_1 = input("\n" + p1_name + ": "
+ random.choice(text_for_plr_turn) + " Press enter for rolling the dice:
")
        print("\n d\Dice is being rolled...")
        dice_value = get_dice_value()
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        print(p1_name + " moving....")
        p1_current_position = snake_ladder(p1_name,
p1_current_position, dice_value)
```

```
        check_win(p1_name, p1_current_position)


        input_2 = input("\n" + p2_name + ": "
+ random.choice(text_for_plr_turn) + " Press enter for rolling the dice: ")
        print("\n Dice is being rolled...")
        dice_value = get_dice_value()
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        print(p2_name + " moving....")
        p2_current_position = snake_ladder(p2_name,
p2_current_position, dice_value)


        check_win(p2_name, p2_current_position)



if __name__ == "__main__":
    start()
```

OUTPUT:-

```
Name of first player: anshul
Name of second player: gaurav sir

'anshul' and 'gaurav sir You will play against each other'


anshul: You are doing great. Press enter for rolling the dice:

 d\Dice is being rolled...
Dice value 5
anshul moving....

anshul moved from 0 to 5

YOU ARE A CHAMPION ########

anshul is clibing the ladder from 5 to 15

gaurav sir: You are doing great. Press enter for rolling the dice:

 Dice is being rolled...
Dice value 6
gaurav sir moving....

gaurav sir moved from 0 to 6
```

CONCLUSION

- The Snake and Ladder board game project in Python was an engaging and educational endeavor that successfully combined the use of Pygame for graphical interface and user interactions with NumPy for efficient data handling and game logic. The project involved creating a digital version of the classic board game, where players move their pieces across a grid based on dice rolls, navigating through snakes and ladders until they reach the final square.
- **Key Achievements**
- **Interactive Game Board**: The project featured a visually appealing and interactive game board, rendered using Pygame. The graphical representation of the board, along with player pieces, provided a clear and engaging user experience.
- **Smooth User Interaction**: The implementation of event handling allowed for smooth user interactions. Players could roll the dice and see their pieces move accordingly, with the game logic accurately reflecting the rules of Snake and Ladder.
- **Efficient Game Logic**: Using NumPy, the game logic was efficiently managed. This included handling the movement of pieces, detecting snakes and ladders, and ensuring players did not exceed the final square. The use of NumPy ensured the game operated quickly and reliably.
- **Random Dice Rolls**: The dice roll functionality, implemented with NumPy's random number generation, provided a fair and unpredictable element to the game, simulating the real-life experience of rolling a dice.
- **Learning Outcomes**
- **Integration of Libraries**: This project demonstrated the effective integration of Pygame and NumPy, showcasing how these libraries can be combined to create complex applications. Understanding the strengths of each library and how to leverage them was a key learning outcome.
- **Game Development Skills**: Developing this game enhanced skills in game development, including graphics rendering, event handling, and implementing game mechanics. These skills are transferable to other types of games and applications.
- **Problem-Solving and Debugging**: Throughout the development process, various challenges and bugs were encountered and resolved. This enhanced problem-solving abilities and provided a deeper understanding of debugging in Python.
- **Project Management**: Completing this project from start to finish involved planning, designing, coding, testing, and refining the game. These project management skills are valuable for future development projects.