**Title of script:** Emoji and Characters-based Password Generator: Python Script with Unicode and UTF-8 Explored

**Author:** Aniska Chatterjee- 21BCE5468

**Keywords:** Emoji password generator, Python, password security, UTF-8, Unicode, random password, tutorial, argparse, random module.

**Outline**

**Outline**

1. Introduction to password security and complexity.
2. Explanation of **Unicode** and **UTF-8** encoding.
3. How emojis are represented in Unicode and their relevance in password generation.
4. Building the Emoji and Characters-based Password Generator.
   * Importing necessary modules.
   * Building the password generator class with emoji support.
   * Explanation of Unicode representation and UTF-8 encoding for emojis.
5. Testing the password generator script with various configurations.
6. Adding flexibility using **argparse** for user-defined passwords.
7. Conclusion and best practices for secure password generation.

|  |  |
| --- | --- |
| Visual Cue | Narration |
| **Show slide:**  Welcome | Welcome to the Spoken Tutorial on **Emoji and Characters-based Password Generator using python** |
| **Show slide:**  Learning Objectives | In this tutorial, we will learn to   * Understand the fundamentals of **Unicode** and **UTF-8** encoding. * Learn how to represent emojis in passwords using **Unicode**. * Build a Python-based password generator that supports a mix of uppercase, lowercase letters, digits, special characters, and emojis. |
| **Show slide:**  System Requirements  **Python 3.12.3** | To record this tutorial, I am using   * **Python 3.12.3** |
| **Show slide:**  Prerequisite  [https:/spoken-tutorial.org](https://www.spoken-tutorial.org/) | To follow this tutorial:   * You must have basic knowledge of using **Python** * For pre-requisite **Python Tutorials**, please visit this website * **Python** libraries required for **automation** must be installed |
| **Show slide:**  Code files | ●      The files used in this tutorial are provided in the **Code**  **files** link.  ●      Please download and extract the files.  ●      Make a copy and then use them while practicing. |
| **Show Slide:**  Emoji and Characters-based Password Generator  (pause at this page) | In this tutorial we will build the password **using emojis, special characters, alphabets, digits** as shown here. |
| **Slide:**  Why Secure Passwords Matter | Importance:   * Let’s start by understanding why secure passwords are important. * Hackers are continuously evolving their techniques to crack passwords through methods like brute-force attacks, dictionary attacks, and social engineering. * One way to enhance the complexity of passwords is by introducing special characters and emojis, which are less common and harder to predict. |
| **Show Slide:**  Emoji and Characters-based Password Generator – Unicode and UTF-8 | Now, before we dive into the code, it’s crucial to understand **Unicode** and **UTF-8**. Unicode is a universal character set that assigns a unique number, called a **code point**, to every character, letter, and symbol across all languages and platforms. |
| **Show Slide:**  Emoji and Characters-based Password Generator - Code Point Example  **Show Slide:**  Emoji and Characters-based Password Generator - Code Point Example | * For example, the Unicode code point for the letter **A** is U+0041 * and for the smiling face emoji 😊, it’s U+1F60A. * These code points allow computers to store and manage characters from different languages consistently. |
| **Show Slide:**  Emoji and Characters-based Password Generator – UTF-8 | **UTF-8** is a specific encoding system for **Unicode**. It represents characters using 1 to 4 bytes, making it both efficient and compatible with a wide range of systems. UTF-8 is widely used across the internet because it is backward compatible with ASCII and can represent all Unicode characters.   * **1-byte characters**: For ASCII characters (like A, B, 1, etc.). * **2-byte or 3-byte characters**: For common characters in many languages. * **4-byte characters**: For emojis and symbols from other writing systems. |
| **Show Slide:**  How Emojis are Represented in Unicode | Emojis are represented as 4-byte Unicode characters. For example, the Unicode for 😂 (the "Face with Tears of Joy" emoji) is U+1F602, while ❤️ (Red Heart emoji) is U+2764 with an additional variation selector.  By using Unicode in password generation, we can embed these emoji characters in the password, increasing its complexity. |
| **Slide:**  **Install python using the following tutorial** | Note that **python** must be installed for this tutorial.  If not, please install using the below commands. |
| **Point to the file in downloads folder** | I have created the source code **generator.py** for this demonstration. |
| **Open generator.py file** | Now, let us go through the code in the text editor. |
| **Highlight:**  import random  import string  import argparse | First we need to **import** the necessary **modules**.   * The random module helps in selecting random characters, * string provides pre-built character sets, * and argparse allows us to accept user input from the command line. |
| **Highlight:** **Defining the Password Generator Class**  class PasswordGenerator: | Next, we define the **PasswordGenerator** class, which handles generating passwords with a mix of upper and lowercase letters, digits, special characters, and emojis. We also include parameters to control how many of each character type should appear. |
| **Highlight:**  class PasswordGenerator:  def \_\_init\_\_(self, length=12, uppercase=True, lowercase=True, digits=True, special=True, emojis=True, unicode\_emojis=False):  self.length = length  self.uppercase = uppercase  self.lowercase = lowercase  self.digits = digits  self.special = special  self.emojis = emojis  self.unicode\_emojis = unicode\_emojis | * Here’s the class definition. Notice how we have a boolean flag for each character set, such as uppercase, lowercase, digits, special, and emojis. * We also introduce a unicode\_emojis flag. If it’s set to True, we’ll convert the emojis into their **Unicode** equivalents, ensuring they are compatible with websites or platforms that may not display emojis natively but can handle Unicode characters. |
| **Highlight:**  def generate\_password(self): | Now, let’s look at the generate\_password function, which handles the core functionality of building the password from the available character sets. |
| **Highlight:**  character\_pool = []  password = [] | **Initialize character pools:** The function starts by initializing two empty lists:   * character\_pool: This list will hold all the potential characters that can be included in the password, depending on user preferences (uppercase letters, lowercase letters, digits, special characters, and emojis). * password: Initially, this list is empty and will later store the actual characters that make up the generated password. |
| **Highlight:**  def add\_characters(char\_set, max\_count): | **Helper function add\_characters:** This function takes two inputs:   * char\_set: A set of characters (like uppercase letters, digits, or emojis). * max\_count: The maximum number of characters that can be added from that set. |
| **Highlight:**  (char\_set, max\_count): | * The function checks if max\_count is provided. * If it is, it selects a random sample of characters from char\_set up to the max\_count. * Otherwise, it converts the entire char\_set into a list. * This ensures that only the allowed number of characters from each character type are included. |
| **Highlight:**  return [random.choice(char\_set) for \_ in range(max\_count)] if max\_count is not None else list(char\_set) | * Randomly choose characters from char\_set if max\_count is specified, otherwise use the entire set. |
| **Highlight:**  if self.uppercase:  uppercase\_chars = add\_characters(string.ascii\_uppercase, self.max\_uppercase or self.length) character\_pool.extend(uppercase\_chars)  if self.lowercase:  lowercase\_chars = add\_characters(string.ascii\_lowercase, self.max\_lowercase or self.length) character\_pool.extend(lowercase\_chars)  if self.digits:  digit\_chars = add\_characters(string.digits, self.max\_digits or self.length) character\_pool.extend(digit\_chars)  if self.special:  special\_chars = add\_characters(string.punctuation, self.max\_special or self.length) character\_pool.extend(special\_chars)  if self.emojis:  emoji\_str = "😂❤️🤣👍😭🙏😘🥰😍😊" emoji\_chars = add\_characters(emoji\_str, self.max\_emojis or self.length) character\_pool.extend(emoji\_chars) | This whole block focuses on Adding character sets based on user options.   * Now, the function checks which character types the user has chosen (uppercase, lowercase, digits, special characters, and emojis) and adds them to the character\_pool accordingly. |
| **Highlight:**  if self.uppercase:  uppercase\_chars = add\_characters(string.ascii\_uppercase, self.max\_uppercase or self.length) character\_pool.extend(uppercase\_chars) | **Uppercase Letters:** If the self.uppercase flag is true, the function uses the helper function add\_characters to add uppercase letters (A-Z) from string.ascii\_uppercase to the character\_pool. The number of characters added depends on the max\_uppercase value or the total password length. |
| **Highlight:**  if self.lowercase:  lowercase\_chars = add\_characters(string.ascii\_lowercase, self.max\_lowercase or self.length) character\_pool.extend(lowercase\_chars) | **Lowercase Letters:** Similarly, if self.lowercase is true, it adds lowercase letters (a-z) from string.ascii\_lowercase to the character\_pool. |
| **Highlight:**  if self.digits:  digit\_chars = add\_characters(string.digits, self.max\_digits or self.length) character\_pool.extend(digit\_chars) | **Digits:** If self.digits is true, digits (0-9) from string.digits are added. |
| **Highlight:**  if self.special:  special\_chars = add\_characters(string.punctuation, self.max\_special or self.length) character\_pool.extend(special\_chars) | **Special Characters:** If self.special is true, special characters (such as @, #, !) from string.punctuation are added. |
| **Highlight:**  if self.emojis:  emoji\_str = "😂❤️🤣👍😭🙏😘🥰😍😊…" | **Emojis:** If self.emojis is true, a predefined string of emojis (in this case, a small sample including 😂, ❤️, 🤣, etc.) |
| **Highlight:**  if self.unicode\_emojis:  emoji\_chars = [f"U+{ord(emoji):X}" for emoji in random.sample(emoji\_str, min(self.max\_emojis or self.length, len(emoji\_str)))] | In this code we see If unicode\_emojis is True, convert emojis to Unicode representation.  By using f"U+{ord(emoji):X}" we can randomly choose emojis and convert them to Unicode (e.g., U+1F602 for 😂) |
| **Highlight:**  else:  emoji\_chars = add\_characters(emoji\_str, self.max\_emojis or self.length)  character\_pool.extend(emoji\_chars) | Add actual emojis if unicode\_emojis is False  Add to the character pool |
| **Highlight:**  if not character\_pool:  raise ValueError("At least one character set must be selected") | In this line of code we ensure at least one character set is selected, otherwise raise an error |
| **Highlight:**  while len(password) < self.length: password.append(random.choice(character\_pool)) | Randomly select characters from the pool until the password reaches the specified length and then append random characters to the password list |
| **Highlight:**  password\_str = ' '.join(password)  print("Your Password:", password\_str)  return password\_str | Join the list of characters into a single string for the final password  Print the generated password  Return the generated password string |
| **Highlight:**  def main(): | Main function that sets up the command-line interface (CLI) for the password generator |
| **Highlight:**  Argparse for CLI Flexibility  parser =argparse.ArgumentParser(description='Password generator with optional emoji Unicode output') | Next, we’ll implement argparse to allow users to customize the password from the command line. The user can specify the length of the password and whether or not to include emojis, special characters, etc. |
| **Highlight:**  parser = argparse.ArgumentParser(description='Password generator with optional emoji Unicode output')  parser.add\_argument('length', type=int, default=12, help='Length of password')  parser.add\_argument('-u', '--uppercase', action='store\_true', help="Include uppercase letters")  parser.add\_argument('-l', '--lowercase', action='store\_true', help="Include lowercase letters")  parser.add\_argument('-d', '--digits', action='store\_true', help="Include digits")  parser.add\_argument('-s', '--special', action='store\_true', help="Include special characters")  parser.add\_argument('-e', '--emojis', action='store\_true', help="Include emojis") | This block of code allows us to use flags in the terminal to include or exclude character types in the password generation process.  Add argument for password length  Optional flags to include specific character sets |
| **Highlight:**  parser.add\_argument('--unicode-emojis', action='store\_true', help="Output Unicode values for emojis") | This line of code is to Flag to generate Unicode representations of emojis |
| **Highlight:**  args = parser.parse\_args() | Parse the arguments from the command line |
| **Highlight:**  password\_generator = PasswordGenerator(args.length, args.uppercase, args.lowercase, args.digits, args.special, args.emojis, args.max\_uppercase, args.max\_lowercase, args.max\_digits, args.max\_special, args.max\_emojis, args.unicode\_emojis) | Create an instance of PasswordGenerator with the parsed arguments |
| **Highlight:**  password\_generator.generate\_password() | Next, Call the password generator to create the password |
| **Highlight:**  if \_\_name\_\_ = = "\_\_main\_\_":  main() | If the script is executed directly, call the main() function |
| **Type in terminal:**  > cd Downloads  > Python3 password\_gen.py | In the terminal, type **cd Downloads** and press enter**.**  Next type **python3 password\_gen.py** and press enter. |
| **Show password\_gen terminal window:**  **password\_gen terminal** window opens up on running the code | (pause here)Wait for a moment as it may take a while for the **terminal** window to appear.  The **terminal window** will pop up once the code is executed.  Let us test out our **password generator application.** |
| **Type in tkinter window:**  python3 password\_generator.py 16 --uppercase --lowercase --digits --emojis --unicode\_emojis  **Click enter** | Let’s run the script with different configurations. For example: |
| **Highlight:**  **Result displayed** | This command generates a 16-character password with uppercase letters, lowercase letters, digits, emojis, and converts the emojis into Unicode. |
| **Close terminal** | We can terminate this program by closing the **terminal**. |
| **Show slide:**  Summary | This brings us to the end of the tutorial. Let us summarize.  In this tutorial, we have learnt to   * Create functions to generate password * Implement text processing techniques in Python * Use Unicode and emojis to create password |
| **Show slide:**  Assignment | As an assignment, do the following:   * Modify the **checker.py** andprint the total number of errors in the result area. * Hint:  Use the **len** function to calculate the number of errors. |
| **Show slide:**  About the Spoken Tutorial Project | The video at the following link summarizes the **Spoken Tutorial Project.** Please download and watch it |
| **Show Slide:**  Spoken Tutorial Workshops | The **Spoken Tutorial Project** team conducts workshops and gives certificates.  For more details, please write to us. |
| **Show Slide:** Answers for THIS Spoken Tutorial | Please post your timed queries in this forum. |
| **Show Slide:**  FOSSEE Forum | For any general or technical questions on **Python for Automation**, visit the **FOSSEE forum** and post your question. |
| **Show slide:**  Acknowledgement | **The Spoken Tutorial Project** was established by the **Ministry of Education, Government of India.** |
| **~~Show slide:~~**  ~~Acknowledgement~~ | ~~We thank~~ **~~Dr. Revathi, Dr. S.Brindha, and Dr. T. Subbulakshmi~~** ~~from~~ **~~VIT, Chennai~~** ~~for their support.~~ |
| **Show slide:**  Thank You | This is **Jasmine Tresa Jose**, a **FOSSEE Semester Long Intern 2024, IIT Bombay** signing off.  Thanks for joining. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_