**# Desarrollo pensamiento computacional. Dos ejemplos de aplicación**

**## Introducción**

En el campo de la educación y la investigación, el desarrollo del pensamiento computacional ha demostrado ser una herramienta poderosa para incentivar la resolución de problemas a través de modelos computacionales. En este artículo, exploraremos dos ejemplos de cómo el lenguaje de programación Python puede fomentar esta competencia y aplicarse en situaciones prácticas.

**## Ejemplo 1: Criptografía - Protección de la confidencialidad de las comunicaciones**

En nuestra experiencia docente, hemos observado cómo la computación puede utilizarse para resolver problemas de protección de la comunicación, específicamente en el campo de la criptografía. Supongamos que necesitamos enviar información confidencial por correo electrónico y deseamos codificarla para garantizar que nadie más pueda leer el mensaje en caso de acceso no autorizado a nuestro correo.

**### El método de sustitución simple**

Una de las técnicas más básicas de protección de las comunicaciones es el método de sustitución simple. Consiste en reemplazar cada letra de un mensaje por la siguiente letra del alfabeto. Por ejemplo, la letra 'A' se reemplazaría por 'B', 'B' por 'C', y así sucesivamente.

Para implementar este método, python tiene la librería `string` experta en la manipulación de las letras de nuestro alfabeto. La importación de esta librería se realiza de la siguiente manera:

```python

import string

```

Luego, debemos obtener el alfabeto con la función `ascii\_uppercase` para obtener el alfabeto en mayúsculas. Esto siguiendo la didáctica del autor Nielsen en su libro de criptografía

```python

ALFABETO = string.ascii\_uppercase

print(ALFABETO)

```

Así, podemos ver que en Python, el módulo `string` nos permite obtener el alfabeto en mayúsculas. Pero, ¿cómo lo ordena una computadora? En realidad, asigna números enteros a cada letra, y en Python, la asignación de enteros comienza en cero. Por ejemplo, con el siguiente código, podemos obtener la primera latra del alfabeto, observe que se utilizó en esta código, al número entero cero.

```python

ALFABETO[0] # Accedemos a la primera letra usando el índice 0

```

Por supuesto, aquí tienes el código para implementar la codificación y decodificación de mensajes utilizando el método de sustitución simple:

```python

import string

def sustitucion\_simple(texto, desplazamiento = 1):

# Definimos el alfabeto en mayúsculas

ALFABETO = string.ascii\_uppercase

# Creamos un diccionario que mapea cada letra original a su letra sustituta

mapeo = {}

for i, letra in enumerate(ALFABETO):

indice\_sustituto = (i + desplazamiento) % len(ALFABETO)

mapeo[letra] = ALFABETO[indice\_sustituto]

# Codificamos el mensaje letra por letra

mensaje\_codificado = ""

for letra in texto:

if letra.upper() in mapeo:

letra\_codificada = mapeo[letra.upper()]

if letra.isupper():

mensaje\_codificado += letra\_codificada

else:

mensaje\_codificado += letra\_codificada.lower()

else:

mensaje\_codificado += letra

return mensaje\_codificado

def decodificar\_sustitucion\_simple(texto\_codificado, desplazamiento):

# Invertimos el desplazamiento para obtener el desplazamiento inverso

desplazamiento\_inverso = len(string.ascii\_uppercase) - desplazamiento

return sustitucion\_simple(texto\_codificado, desplazamiento\_inverso)

# Ejemplo de uso:

mensaje\_original = "ESTE ES UN MENSAJE SECRETO"

desplazamiento = 3

# Codificar el mensaje

mensaje\_codificado = sustitucion\_simple(mensaje\_original, desplazamiento)

print("Mensaje codificado:", mensaje\_codificado)

# Decodificar el mensaje

mensaje\_decodificado = decodificar\_sustitucion\_simple(mensaje\_codificado, desplazamiento)

print("Mensaje decodificado:", mensaje\_decodificado)

```

Este código implementa las funciones `sustitucion\_simple` y `decodificar\_sustitucion\_simple`. La función `sustitucion\_simple` toma un texto y un desplazamiento como entrada y devuelve el texto codificado utilizando el método de sustitución simple. La función `decodificar\_sustitucion\_simple` realiza el proceso inverso y decodifica el mensaje dado el texto codificado y el desplazamiento original.

**## Ejemplo 2: Clasificación de plantas con una red neuronal**

Otra aplicación interesante del pensamiento computacional es la construcción de modelos para la clasificación de datos. En este caso, utilizaremos una red neuronal para clasificar diferentes especies de plantas a partir de características específicas.

Para ello, emplearemos el famoso dataset de flores de iris de Fisher y la librería `scikit-learn`, asociada a Python, que proporciona herramientas eficientes para el aprendizaje automático y la minería de datos.

```python

# Aquí importaríamos las librerías necesarias y cargaríamos el dataset de iris

# Luego, crearíamos y entrenaríamos la red neuronal para la clasificación de plantas

>> import tensorflow as tf

>>> from tensorflow import keras

>>> tf.\_\_version\_\_

'2.0.0'

>>> keras.\_\_version\_\_

'2.2.4-tf'

from sklearn.datasets import load\_iris

# Cargamos el conjunto de datos de iris

iris = load\_iris()

# Las características (features) de las plantas se encuentran en iris.data

# Las etiquetas de las especies de las plantas se encuentran en iris.target

# Puedes acceder a la información sobre el conjunto de datos utilizando iris.DESCR

```

Si deseas utilizar TensorFlow para el análisis posterior o la construcción de modelos de aprendizaje automático con este conjunto de datos, puedes convertirlo en un DataFrame de pandas y luego trabajar con él:

```python

import pandas as pd

# Convertimos el conjunto de datos de iris en un DataFrame

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['target'] = iris.target

# Ahora puedes utilizar el DataFrame df con TensorFlow para entrenar modelos, análisis, etc.

```

En siguiente código, primero carga el conjunto de datos de iris y divide los datos en conjuntos de entrenamiento y prueba. Luego, escalamos las características para asegurarnos de que tengan una misma escala y esto ayude al rendimiento del modelo.

El modelo de la red neuronal se define utilizando tf.keras.models.Sequential, que es una pila lineal de capas. Tenemos una capa oculta con 10 neuronas y función de activación 'relu', y una capa de salida con 3 neuronas (correspondientes a las 3 clases de iris) y función de activación 'softmax', que nos dará la probabilidad de que una instancia pertenezca a cada clase.

Luego, compilamos el modelo con el optimizador 'adam' y la función de pérdida 'sparse\_categorical\_crossentropy', que es adecuada para problemas de clasificación con etiquetas enteras.

Finalmente, entrenamos el modelo con los datos de entrenamiento, validamos durante el entrenamiento y evaluamos su precisión en el conjunto de prueba. Al final, imprimimos la precisión del modelo en el conjunto de prueba.

Este es solo un ejemplo básico de cómo construir un modelo de clasificación con TensorFlow. Puedes experimentar con diferentes arquitecturas de red, hiperparámetros, funciones de activación y optimizadores para mejorar el rendimiento del modelo.

```python

import tensorflow as tf

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Cargar el conjunto de datos de iris

iris = load\_iris()

X, y = iris.data, iris.target

# Dividir el conjunto de datos en entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Escalar las características (features) para mejorar el rendimiento del modelo

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Construir el modelo de la red neuronal

model = tf.keras.models.Sequential([

tf.keras.layers.Dense(10, activation='relu', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(3, activation='softmax')

])

# Compilar el modelo

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Entrenar el modelo

model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.1)

# Evaluar el modelo en el conjunto de prueba

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Precisión del modelo en el conjunto de prueba: {accuracy \* 100:.2f}%')

```

La red neuronal entrenada podrá clasificar una nueva planta desconocida en una de las especies de iris conocidas, lo cual resulta una herramienta útil en la botánica y otras áreas de estudio.

**## Conclusiones**

El desarrollo del pensamiento computacional a través del lenguaje Python ofrece una gran variedad de aplicaciones prácticas. En este artículo, hemos visto cómo puede utilizarse en el campo de la criptografía para proteger la confidencialidad de las comunicaciones, así como en la clasificación de datos con redes neuronales para identificar especies de plantas.

Estos ejemplos demuestran cómo el aprendizaje de la programación y el enfoque en el pensamiento computacional pueden mejorar nuestra capacidad para resolver problemas y aplicar soluciones en diversas áreas de estudio y trabajo. Al dominar estas habilidades, no solo reducimos el estrés al enfrentar desafíos, sino que nos convertimos en mejores seres humanos al contribuir positivamente a la sociedad y al desarrollo de soluciones innovadoras.

**## Referentes**

\* Nielson, S. J., & Monson, C. K. (2019). Practical Cryptography in Python: Learning Correct Cryptography by Example. Apress.

\* Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.".