Fundamentos de Python para chatbots
© ADR Infor SL

eclap.adrformacion.com © ADR Infor SL
JUAN ALBERTO AZEVEDO IBAÑEZ
eclap.adrformacion.com © ADR Infor SL
JUAN ALBERTO AZEVEDO IBAÑEZ

Indice

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:
Diseñar, implementar y gestionar chatbots inteligentes e interactivos en aplicaciones web utilizando Django y Python, integrando habilidades de procesamiento de lenguaje natural para mejorar la experiencia del usuario y optimizar la interacción entre usuarios y chatbots.

Resultados de Aprendizaje:

- Utilizar variables, estructuras de control, y funciones en Python para desarrollar la lógica básica de un chatbot.
- Aplicar el manejo de excepciones y la comprensión de listas para optimizar el código del chatbot.

Fundamentos de Python para chatbots

Vamos a ver de forma muy rápida algunos de los fundamentos básicos de Python que no serán muy útiles para trabajar con chatbots

Variables

Variables

En programación, una variable está formada por un espacio en el sistema de almacenaje y un nombre simbólico que está asociado a dicho espacio. Ese espacio contiene una cantidad de información conocida o desconocida; es decir, un valor.

Puedes imaginar una variable como una caja. En dicha caja puedes guardar cosas. Estas cosas son datos. Para poder "buscar" las cajas (variables) en el almacén, les pones nombre (identificador).

En otros lenguajes, además del identificador (nombre), debemos decirle a la "caja" qué tipo de dato vamos a guardar, pero Python esto lo reconoce de manera dinámica y no es necesario especificarlo. El lenguaje lo reconoce automáticamente. No obstante, siempre lo puedes especificar.

De forma genérica puedes definir una **asignación de variable en Pyhon** como sigue:

Identificador = Dato

Cómo asignar un valor a una variable en Python, dentro de la consola de Python:
>>>>numero = 2
>>>numero
2

Se puede reasignar el valor de una misma variable todas las veces que se quiera, pero solo almacena el último valor asignado... hasta que se le asigne otro.

Reglas y convención de nombres

Algunas reglas y convenciones de nombres para las variables y constantes:

- **No** puedes usar símbolos especiales como **!, @, #, $, %, etc.**
- El **primer carácter no** puede ser un **número** o dígito.
- El nombre de una variable debería tener la combinación de **letras en minúsculas** (de a a la z) o MAYÚSCULAS (de la A a la Z) o **dígitos** (del 0 al 9) o un **guion bajo** (_). Por ejemplo: variable_tres_palabras.
- Los nombres que comienzan con guion bajo (simple _ o doble __) se reservan para variables con significado especial.
- No pueden usarse como identificadores, las palabras reservadas: ***and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while y yield***.

Tipos de datos

En Python puedes encontrar los siguientes **tipos de datos: texto, numéricos y *booleanos***;

aunque existen otros como:

- **Secuencias**: Como listas, tuplas y rangos.
- **Diccionarios**: Representados por el tipo dict.
- **Conjuntos**: Definidos por el tipo set.
- **Iteradores, Clases, Instancias y Excepciones**: Otros tipos de datos presentes en Python

Cómo se define cada tipo de dato

Cada tipo de dato determina qué operaciones van a soportar las variables a las que se las estás asignando y, por lo tanto, qué operaciones se van a poder realizar con esos valores de datos, qué atributos tienen, etc.

Resumen de cómo se definen y se utilizan cada uno de los tipos de datos mencionados en Python:

Texto (str)

Texto (str): Se define utilizando comillas simples (') o dobles ("") alrededor del texto. Por ejemplo:

```
texto = "Hola, mundo!"
```

Numéricos:

Numéricos: Incluyen enteros (int), flotantes (float) y números complejos (complex). Se pueden definir de la siguiente manera:

```
entero = 10 flotante = 3.14 complejo = 2 + 3j
```

Booleanos:

Booleanos: Representan dos valores: Verdadero (True) y Falso (False). Se pueden definir directamente o como resultado de expresiones lógicas:

```
es_verdadero = True es_falso = False
```

Secuencias

Listas (list)

Listas (list): Se definen utilizando corchetes [] y pueden contener elementos de diferentes tipos.

```
lista = [1, 2, 3, "cuatro"]
```

Tuplas (tuple)

Tuplas (tuple): Se definen utilizando paréntesis () y son inmutables, es decir, no se pueden modificar después de su creación.

```
tupla = (1, 2, 3)
```

Rangos (range)

Rangos (range): Se utilizan para representar una secuencia de números y se definen utilizando la función range().

```
rango = range(0, 10)
```

Diccionarios (dict)

Diccionarios (dict): Se definen utilizando llaves {} y contienen pares clave-valor.

diccionario = {"clave": "valor", "nombre": "Juan"}

Conjuntos (set)

Conjuntos (set): Se definen utilizando llaves {} o la función set() y contienen elementos únicos sin un orden definido.

conjunto = {1, 2, 3, 4}

Iteradores, Clases, Instancias y Excepciones: Estos son tipos de datos más avanzados y pueden ser definidos por el usuario en Python para diferentes propósitos, como la iteración sobre secuencias, la definición de estructuras de datos personalizadas, la creación de instancias de objetos y el manejo de excepciones.

En Python, los tipos de datos son dinámicos, lo que significa que el intérprete infiere el tipo de dato de una variable según el valor que se le asigna. Además, Python es un lenguaje fuertemente tipado, lo que significa que no se permiten ciertas operaciones entre tipos de datos incompatibles. Todos los tipos de datos en Python son clases, lo que significa que los datos son instancias de estas clases y pueden acceder a métodos y atributos asociados a ellas.

Esto simplemente es un recordatorio, no vamos a entrar en como se utilizan ni en métodos específicos para cada tipo de dato

Estructuras de control

Dentro de las estructuras de control tenemos fundamentalmente de dos tipos, las de control de flujo y los bucles.

Estructuras de control de flujo: Estas estructuras permiten alterar el flujo de ejecución del programa basándose en ciertas condiciones. Las estructuras de control de flujo más comunes son:

Condicionales

Permiten ejecutar cierto bloque de código si se cumple una condición.

Ejemplo en Python con la estructura if-elif-else:

if condicion1: # bloque de código si se cumple la condición 1 elif condicion2: # bloque de código si se cumple la condición 2 else: # bloque de código si no se cumple ninguna de las condiciones anteriores

Bifurcaciones:

Son estructuras más complejas que combinan múltiples condiciones y acciones en función de estas condiciones.

Bucles (loops)

Estas estructuras permiten ejecutar un bloque de código repetidamente mientras se cumpla una determinada condición. Los bucles más comunes son:

Bucle while:

Ejecuta un bloque de código mientras una condición sea verdadera.

while condicion: # bloque de código

Bucle for:

Itera sobre una secuencia (como una lista o un rango) y ejecuta un bloque de código para cada elemento de la secuencia.

for elemento in secuencia: # bloque de código

Estas estructuras de control son fundamentales en la programación ya que la programación ya que permiten crear programas más dinámicos y adaptables, permitiendo tomar decisiones y repetir acciones según lo requiera la lógica del programa.

Funciones y manejo de excepciones

Debemos recordar también que las funciones y el manejo de excepciones son aspectos fundamentales en la programación, ya que permiten modularizar el código y manejar situaciones inesperadas de manera controlada.

Funciones

Las funciones son bloques de código reutilizables que realizan una tarea específica, permitiendo organizar el código en secciones más pequeñas, lo que facilita la depuración y el mantenimiento del código. En Python, las funciones se definen con la palabra clave def, seguida del nombre de la función y, opcionalmente, parámetros entre paréntesis. El cuerpo de la función se define con sangría.

Las funciones pueden devolver un valor utilizando la palabra clave return.
Ejemplo de definición de una función en Python:

def suma(a, b): return a + b

Manejo de excepciones

Las excepciones son eventos que ocurren durante la ejecución de un programa y que interrumpen el flujo normal de ejecución. El manejo de excepciones permite controlar estas situaciones excepcionales y tomar acciones adecuadas en caso de que ocurran.

En Python, el manejo de excepciones se realiza utilizando bloques try, except y, opcionalmente, finally.

1. El bloque try se utiliza para envolver el código que podría generar una excepción. Si se produce una excepción dentro del bloque try, el control del programa se transfiere al bloque except.
2. El bloque except especifica cómo manejar la excepción. Puede haber múltiples bloques except para manejar diferentes tipos de excepciones.
3. El bloque finally se ejecuta siempre, independientemente de si se produce una excepción o no, y se utiliza para realizar limpieza de recursos, como cerrar archivos o conexiones de red.

Ejemplo de manejo de excepciones en Python:

try: resultado = 10 / 0 except ZeroDivisionError: print("Error: división por cero") finally: print("Finalizando programa")

Comprensión de listas

La comprensión de listas es una característica avanzada en Python que permite crear listas de manera concisa y legible en una sola línea de código. Es una técnica elegante para generar listas basadas en iteraciones o transformaciones de otras secuencias.

Sintaxis básica:

- La sintaxis general de la comprensión de listas es [expresión for elemento in iterable].
- Esta sintaxis puede incluir un if opcional para filtrar elementos: [expresión for elemento in iterable if condición].
- La expresión se evalúa para cada elemento del iterable y se agrega a la lista resultante si la condición (si está presente) se evalúa como verdadera.

Uso de iterables

Podemos utilizar cualquier iterable, como listas, tuplas, rangos, cadenas, conjuntos, etc.
Por ejemplo, crear una lista de los cuadrados de los números del 1 al 5:

cuadrados = [x ** 2 for x in range(1, 6)]

Transformaciones y operaciones

Podemos realizar transformaciones o aplicar operaciones a los elementos. Por ejemplo, podemos crear una lista de cadenas convertidas a mayúsculas:

palabras = ['hola', 'mundo', 'python'] mayusculas = [palabra.upper() for palabra in palabras]

Comprensión de listas anidadas

Es posible anidar comprensiones de listas para realizar operaciones más complejas o para trabajar con estructuras de datos anidadas. Por ejemplo, podemos crear una lista de tuplas donde cada tupla contiene el índice y el valor de una lista original:

lista = ['a', 'b', 'c'] lista_tuplas = [(indice, valor) for indice, valor in enumerate(lista)]

La comprensión de listas nos permite simplificar significativamente el código y hacerlo más legible. Sin embargo, es importante usarla con moderación y mantener la claridad del código. En algunos casos, un bucle for convencional puede ser más apropiado si la comprensión de listas resulta difícil de leer o comprende múltiples niveles de anidación.

Manejo de archivos

El manejo de archivos en Python es una habilidad esencial para muchos tipos de aplicaciones, desde procesamiento de datos hasta creación de archivos de configuración y registro de eventos. Es importante comprender cómo abrir, leer, escribir y cerrar archivos de manera adecuada para garantizar un funcionamiento correcto y eficiente de tus programas. El manejo de archivos se realiza utilizando las funciones y métodos proporcionados por el módulo open() y los objetos de archivo.

No vamos a entrar en exhaustividad en el manejo de archivos de momento pero aquí vemos las formas básicas de apertura, lectura, escritura y cierre de archivos con Python:

Apertura

Para abrir un archivo en Python, se utiliza la función open() que toma al menos un argumento, el nombre del archivo que se desea abrir, y devuelve un objeto de archivo. La función open() también acepta argumentos adicionales para especificar el modo de apertura del archivo, como 'r' para lectura, 'w' para escritura, 'a' para agregar contenido al final del archivo, 'b' para modo binario, entre otros.
Ejemplo:

archivo = open('archivo.txt', 'r')

Lectura y escritura

Una vez abierto un archivo, se pueden realizar diversas operaciones como lectura, escritura y manipulación de su contenido utilizando métodos proporcionados por el objeto de archivo. Para leer el contenido de un archivo, se utiliza el método read() que devuelve una cadena con todo el contenido del archivo. Para escribir en un archivo, se utiliza el método write() que permite escribir una cadena en el archivo.
Ejemplo:

contenido = archivo.read() # Lectura archivo.write('Hola, mundo!') # Escritura

Cierre

Después de terminar de trabajar con un archivo, es importante cerrarlo utilizando el método close() para liberar los recursos del sistema asociados con el archivo. Es una buena práctica cerrar los archivos después de su uso para evitar problemas de manejo de recursos y asegurar la integridad de los datos.
Ejemplo:

archivo.close()

Contextos

La forma más segura y conveniente de manejar archivos en Python es utilizando contextos de archivos, que garantizan que el archivo se cierre correctamente después de su uso, incluso si ocurre una excepción durante la ejecución.
Ejemplo:

with open('archivo.txt', 'r') as archivo: contenido = archivo.read()

Resumen

- Las variables en Python deben seguir las reglas y convenciones de nombres, como comenzar con una letra o guion bajo, y no usar caracteres especiales ni palabras reservadas. Por ejemplo, un nombre válido sería mi_variable.
- Python ofrece diversos tipos de datos como enteros, flotantes, cadenas, y booleanos. Cada tipo tiene su propia forma de declaración. Por ejemplo Por ejemplo: una cadena se define entre comillas: mi_cadena = "Hola Mundo".
- Las secuencias en Python incluyen listas, tuplas y cadenas. Cada una tiene sus propias características; por ejemplo, las listas son mutables, permitiendo modificar sus elementos.
- Las estructuras de control en Python, como los bucles y condicionales, son fundamentales para gestionar el flujo de un programa, aunque no se explicitan en los títulos proporcionados.
- El manejo de excepciones es crucial para controlar errores en tiempo de ejecución y asegurar que el programa sigue funcionando sin interrupciones.
- La comprensión de listas es una herramienta poderosa en Python que permite crear nuevas listas aplicando expresiones a cada uno de los elementos de una lista existente. Por ejemplo: [x**2 for x in range(10)] crea una lista de los cuadrados de los números del 0 al 9.
- Aunque no se detalla, el manejo de archivos implica operaciones como lectura y escritura, esenciales para el procesamiento de datos en aplicaciones de chatbots.

eclap.adrformacion.com © ADR Infor SL
JUAN ALBERTO AZEVEDO IBAÑEZ

Actividades prácticas

Ejercicios básicos con Python

La actividad práctica titulada 'Ejercicios básicos con Python' está diseñada para evaluar tu comprensión de los fundamentos de Python, específicamente en relación con la creación y manipulación de variables, el manejo de distintos tipos de datos, la implementación de estructuras de control de flujo y la escritura de funciones simples. Tendrás que resolver tres ejercicios, asegurándote de aplicar correctamente las buenas prácticas de programación vistas en clase.

1. **Ejercicio 1:** Definir variables de diferentes tipos
   Crea las siguientes variables en Python y asigna el tipo de dato adecuado para cada una: un texto que contenga "Hola, Chatbot", un número entero que represente la cantidad de letras en ese texto, un valor booleano que indique si la cantidad de letras es mayor a 10, y una lista que contenga cada palabra del texto como un elemento separado.

2. **Ejercicio 2:** Control de flujo y bucles
   Usa una estructura condicional para comprobar si la lista de palabras creada anteriormente tiene más de dos elementos. Si es así, itera sobre la lista utilizando un bucle *for* y convierte cada palabra a mayúsculas, almacenando el resultado en una nueva lista.

3. **Ejercicio 3:** Funciones y manejo de excepciones
   Define una función que tome dos números como argumentos y devuelva su división. Asegúrate de manejar mediante *try-except* el caso en el que el divisor sea cero, retornando un mensaje apropiado.

Procesando respuesta, no cierres el navegador, este proceso podría tardar unos segundos

Recursos

Enlaces de Interés

SpaCy
https://es.wikipedia.org/wiki/SpaCy

Spacy es una biblioteca de procesamiento del lenguaje natural (NLP) en Python

NLTK
https://es.wikipedia.org/wiki/NLTK

Definición de NLTK que realiza la wikipedia

Glosario.

- **Corpus de texto**: Conjunto de documentos o textos escritos que se utilizan para análisis lingüísticos y entrenamiento de modelos de procesamiento del lenguaje natural (PLN).
- **Lematización**: El lema representa la forma canónica de una palabra y generalmente es el término que encontrarías en un diccionario. Por ejemplo, en español, el lema del verbo "corriendo" es "correr", y el lema del sustantivo "ratones" es "ratón".
- **NLTK**: El kit de herramientas de lenguaje natural, o más comúnmente NLTK, es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural (PLN) simbólico y estadístico para el lenguaje de programación Python. NLTK incluye demostraciones gráficas y datos de muestra. Se acompaña de un libro que explica los conceptos subyacentes a las tareas de procesamiento del lenguaje compatibles el toolkit,3 además de programas de ejemplo.4 NLTK está destinado a apoyar la investigación y la enseñanza en procesamiento de lenguaje natural (PLN) o áreas muy relacionadas, que incluyen la lingüística empírica, las ciencias cognitivas, la inteligencia artificial, la recuperación de información, y el aprendizaje de la máquina.5 NLTK se ha utilizado con éxito como herramienta de enseñanza, como una herramienta de estudio individual, y como plataforma para los sistemas de investigación de prototipos y construcción.
- **Tokenización**: Dividir texto en palabras o frases