
Universidad
Tecmilenio®

Universidad Tecmilenio

Fundamentos y aplicaciones de la inteligencia artificial

Estructuras de datos en el lenguaje de programación multiplataforma

Sentencias condicionales y de control de flujo (bucles) en Python

Introducción

Las estructuras de datos son las bases de cualquier programa de computación. El lenguaje de programación Python cuenta con un extenso conjunto de estructuras de datos dentro de su biblioteca estándar.

Por otro lado, y de acuerdo con Matthes (2019), en la programación a menudo es necesario examinar un conjunto de condiciones y decidir qué acción tomar con base en ellas; y en ocasiones, también es necesario repetir lo que hace un programa más de una vez. En programación, a eso se le conoce como bucle.

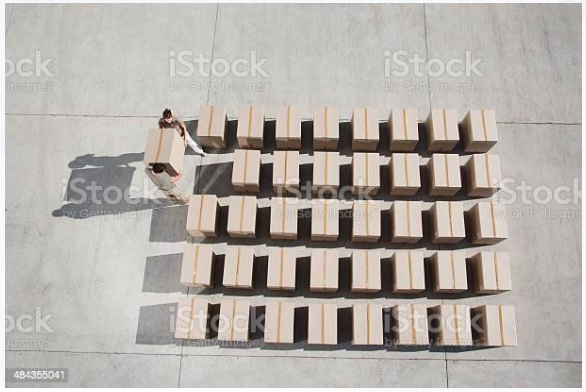
Durante el desarrollo de este tema aprenderás sobre los diferentes tipos de datos abstractos que posee Python, la forma de declararlos y utilizarlos en nuestro código. Además, revisarás varios ejemplos que nos van a permitir asimilar de una forma más práctica todos estos conceptos.

También aprenderás a escribir pruebas condicionales que permiten comprobar cualquier situación de interés y a ejecutar bucles cuando sea necesario.

Estructuras de datos en Python

Las matrices son una de las estructuras de datos fundamentales, estas tienen múltiples utilidades al momento de implementar diferentes algoritmos. Consisten en registros de datos de tamaño fijo que permiten que cada elemento se ubique de manera eficiente en función de su índice.

Las matrices almacenan la información en bloques de memoria adyacentes, por lo cual son consideradas estructuras de datos contiguas; en contraposición a otras, como las listas enlazadas que vinculan las diferentes posiciones de los espacios de memoria que utilizan (Real Python, 2020).



484355041

Listas

Son parte del lenguaje central de Python y, a pesar de su nombre, en el fondo se comportan como si fueran **matrices dinámicas**, es decir, una lista nos da la posibilidad de agregar o eliminar elementos, ajustando automáticamente el espacio que tiene asignado al almacenamiento de estos mediante la asignación o liberación de memoria.

Además, pueden contener cualquier tipo de elementos, ya que todos estos se consideran como objetos, incluidas las funciones. Por lo tanto, se pueden mezclar y combinar diferentes tipos de datos y almacenarlos todos en una sola lista. La desventaja es que ocupa más espacio.

```
In [2]:    datos_estudiante = ["Raul", "Mexico", 176, 98.6, True]
```

Declaración de una lista de elementos variados.

Operaciones con listas

• Para acceder a un elemento dentro de una lista se indica la posición de este mediante el uso de un índice: `datos_estudiante[0]`.

Python considera que el primer elemento de la lista se encuentra en la posición 0, y no en la 1.

También implementa varios métodos para realizar diversas operaciones con las listas.

Método append(x): este método agrega un elemento al final de la lista: `estudiantes.append("Maria")`.

Método insert(i, x): este método agrega un elemento en un índice dado; el primer argumento indica el valor del índice.

```
datos_estudiante.insert(2,"Nuevo Leon")
  print(datos_estudiante)
```

La salida sería: `['Raul', 'Mexico', 'Nuevo Leon', 176, 98.6, True]`.

Método remove(x): elimina el valor de la lista que coincide con el elemento indicado por ***x***. En caso de no encontrarlo, devuelve un error.

Arreglos y tuplas

Los arreglos son matrices con características mutables y que se comportan de manera similar a las listas, excepto por una diferencia importante: están restringidas a un solo tipo de datos. Una forma de almacenar variables de tipo numérico, como los enteros o los números de punto flotante de una manera eficiente, es mediante el uso de arreglos.

Cuando contienen muchos elementos son más eficientes que las listas y las tuplas en cuanto al espacio, de hecho, se definen igual que las listas.

Una función muy útil que nos permite determinar rápidamente la longitud de un arreglo es ***len()***. Funciones como len() o print() son internas de Python y se conocen como funciones propias del lenguaje.

```
len(colores)
```

Por otro lado, las tuplas se utilizan cuando queremos crear listas que sabemos que no sufrirán ningún tipo de modificación. Además, los elementos no se pueden agregar ni eliminar de forma dinámica. Todos los elementos de una tupla deben definirse en el momento de su creación.

```
datos_pelicula = ("Tom_Cruise", "Misión_Imposible", 110, 1996, 4.7)
```

Cadenas de caracteres

Python utiliza un objeto especial para almacenar datos de texto en forma de secuencia inmutable que se llama cadena de caracteres. Se puede considerar como una matriz inmutable de caracteres. Los objetos de cadena ahorran espacio porque están empaquetados de forma compacta y se especializan en un solo tipo de datos, según Python Software Foundation (s.f.).

Debido a que las cadenas son inmutables en Python, modificar una cadena requiere de la creación de una copia modificada. El equivalente más cercano a una cadena mutable es almacenar caracteres individuales dentro de una lista, de acuerdo con Python Software Foundation (s.f.).

```
saludo = "Hola, mi nombre es:"
```

Operaciones con cadenas de caracteres

Concatenar cadenas es muy útil para combinar diferentes elementos en estructuras más complejas, por ejemplo: `saludo_con_nombre = saludo + " " + nombre.`

Método upper(): este método permite transformar toda la cadena en caracteres en mayúsculas.

Método capitalize(): es útil en una aplicación donde se solicita la introducción de un apellido y el usuario, por equivocación, lo escribe todo en minúsculas.

Explicación

Universidad Tecmilenio

Uso de las sentencias condicionales (*if*, *if-else* y *if-elif*)

***if*:** para evaluar si el valor de una expresión es *Verdadero (True)* o *Falso (False)* **==** operador de igualdad **!=** operador de desigualdad.

```
ingrediente = 'lechuga'
if ingrediente != 'tomate':
print("Sin tomate")
```

```
Sin tomate
```

Existe otro tipo de situaciones en las que se requiere que dos condiciones se cumplan para realizar una acción o que solo una de ellas sea suficiente: ***and*** y ***or***

```
edad_0 = 20
edad_1 = 17
edad_0 >= 18 or edad_1 >= 18
```

```
True
```

Explicación

Uso de las sentencias condicionales (*If, if-else* y *if-elif)*

Muchas veces se requiere tomar una acción cuando se cumple una condición y otra diferente para todos los demás casos. Entonces, se utiliza la sintaxis ***if-else*** de Python.

```
edad = 17
if edad >= 18:
print("Tienes la edad suficiente para votar")
else:
print("Lo sentimos, eres demasiado joven para votar")
```

```
Lo sentimos, eres demasiado joven para votar
```

A menudo se requieren probar más de dos situaciones posibles. Aquí es necesario usar la sintaxis ***if-elif-else***. Ejemplo: las tarifas en los museos para diferentes edades.

```
edad = 11
if edad < 5:
    precio = 0
elif edad < 18:
    precio = 95
elif edad < 60:
    precio = 80
else:
    precio = 40
print ("Su costo de admisión es {precio}.")
```

```
Su costo de admisión es 95.
```

Explicación

Uso de ciclos (*for* y *while*)

El iterador ***for*** funciona tomando una colección de elementos y ejecutando un bloque de código una vez para cada elemento de la colección. En contraste, el tipo de bucle ***while*** se ejecuta siempre que cierta condición sea verdadera.

A continuación, tenemos el mismo ejemplo codificado con ambos bucles:

```
conejos = ['Oddy', 'Tiky', 'Manchas', 'Nubito']
actual = 0
for conejo in conejos:
    print(conejo)
```

```
Oddy
Tiky
Manchas
Nubito
```

```
conejos = ['Oddy', 'Tiky', 'Manchas', 'Nubito']
actual = 0
while actual < len(conejos):
    print(conejos[actual])
    actual += 1
```

```
Oddy
Tiky
Manchas
Nubito
```

Explicación

Uso de ciclos (*for* y *while*)

La sentencia ***break*** en un bucle ***for*** le permite salir de este, de la misma manera en que lo hace para un bucle ***while***. Por otra parte, si se inserta un ***continue*** en un bucle ***for***, el programa salta a la siguiente iteración del bucle.

In [15]:
```
quesos = []
for queso in quesos:
    print('Esta tienda tiene un estupendo queso ', queso)
    break
else:  # si el break no se ejecutó significa que no hay queso
    print('Esta tienda no tiene ni un queso.')
```

```
Esta tienda no tiene ni un queso.
```

Actividad

Objetivo: analizar ejercicios que implementen las estructuras de datos, condiciones y bucles en Python.

1. Utilizando la herramienta Jupyter Notebook, programa lo siguiente:
- Escribe una lista que contenga un mínimo de cinco elementos.
- Agrega un nuevo elemento en el índice 2.

•Agrega un elemento al final de la lista.

2. Implementa un programa en Python con la solución a la siguiente petición de un cliente:

•Una sala de cine cobra diferentes precios de entrada según la edad de la persona. Si una persona es menor de tres años, la entrada es gratuita. Si son mayores de 12 años, el boleto cuesta 15 pesos.

•Escribe un bucle en el que se le solicite a los usuarios su edad y despliegue el costo de la entrada.

Cierre

Python es un lenguaje muy versátil, fácil de aprender y de utilizar. Está especialmente diseñado para simplificar el trabajo con múltiples estructuras de datos, condiciones y bucles, y de esa forma poder sacarle el mayor provecho posible.

A medida que vayas desarrollando y resolviendo problemas con él, tus habilidades se irán incrementando. Ser buen programador no es una meta imposible de alcanzar, pero no olvides que el camino a la excelencia solo se puede transitar si nos acompañamos de la perseverancia y la experiencia adquirida con la práctica de tus habilidades.

ID de la fotografía:1329825154

Bibliografía

Matthes, E. (2019). *Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming* (2ª ed.). Estados Unidos: No Starch Press.

Real Python. (2020). *Common Python Data Structures (Guide)*. Recuperado de https://realpython.com/python-data-structures/

Python Software Foundation. (s.f.). Recuperado de https://www.python.org/psf/