Argentina programa 4.0

Ministerio de Desarrollo Productivo Argentina

Secretaría de Economía del Conocimiento

PROGRAMADOR JUNIOR EN MACHINE LEARNING I

UNIVERSIDAD NACIONAL DE MISIONES

FACULTAD DE CIENCIAS EXACTAS QUÍMICAS Y NATURALES

CURSO 1: Fundamentos de Machine Learning (ML) Python - Nivel 1 -

FUNDAMENTACIÓN

Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes.

Python es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto y, por lo tanto, gratuito, lo que permite desarrollar software sin límites. Python puede utilizarse en proyectos de inteligencia artificial, para crear sitios web escalables, realizar cálculos estructurales complejos con elementos finitos, y diseñar videojuegos, entre otras muchas aplicaciones

Objetivos

- Familiarizar al estudiante con el entorno y las herramientas de programación de Python.
- Desarrollar las capacidades de plasmar una propuesta de resolución a un problema sencillo planteado utilizando las herramientas planteadas.
- Identificar las ventajas del trabajo colaborativo y las herramientas disponibles.
- Promover la indagación sobre los temas planteados.

Tabla de contenidos

Conceptos básicos de Python

Introducción a Python

¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado y multiparadigma. Se caracteriza por su sintaxis clara y sencilla, que favorece la legibilidad del código. Python permite trabajar con diferentes paradigmas de programación, como el imperativo, el orientado a objetos, el funcional o el declarativo. Además, cuenta con una amplia biblioteca estándar que ofrece múltiples módulos y paquetes para facilitar el desarrollo de diversas aplicaciones.

Python es uno de los lenguajes más utilizados en el campo de la inteligencia artificial (IA), debido a sus ventajas y características. Algunas de ellas son:

- Es un lenguaje fácil de aprender y usar, lo que reduce la curva de aprendizaje y el tiempo de desarrollo.
- Es un lenguaje flexible y dinámico, que permite modificar el código en tiempo de ejecución y adaptarse a los cambios y requerimientos del problema.
- Es un lenguaje portable y multiplataforma, que se puede ejecutar en diferentes sistemas operativos y entornos.
- Es un lenguaje con una gran comunidad de desarrolladores y usuarios, que ofrecen soporte, documentación y recursos para mejorar el código y resolver dudas.
- Es un lenguaje con una gran variedad de librerías y frameworks especializados en IA, que facilitan la implementación de algoritmos y técnicas avanzadas, como el aprendizaje automático, el procesamiento del lenguaje natural, la visión artificial o el análisis de datos.

En conclusión, Python es un lenguaje de programación muy adecuado para la introducción a la IA, ya que permite crear soluciones eficientes, robustas y versátiles con un código claro y conciso.

¿Por qué Python es un lenguaje de programación popular?

Python es un lenguaje de programación popular por varias razones. Una de ellas es su simplicidad y legibilidad, que facilita el aprendizaje y la escritura de código. Otra razón es su versatilidad y portabilidad, que permite ejecutar programas en diferentes plataformas y sistemas operativos. Además, Python cuenta con una gran comunidad de desarrolladores y usuarios que contribuyen con librerías y módulos para diversas aplicaciones, especialmente en el campo de la inteligencia artificial. Algunas de las librerías más usadas para la IA son NumPy, SciPy, TensorFlow, PyTorch y scikit-learn, que ofrecen herramientas para el análisis de datos, el cálculo numérico, el aprendizaje automático y el procesamiento de imágenes, entre otras. Por estas razones, Python es un lenguaje de programación muy adecuado para introducirse en el mundo de la IA.

Sintaxis y estructura básica de Python

Comentarios y espaciado en Python

Python es un lenguaje de programación fácil de aprender y potente, que tiene una sintaxis clara y simple. En este texto, vamos a ver algunos aspectos básicos de la sintaxis y la estructura de Python, como los comentarios, el espaciado y la identificación.

Los comentarios son líneas de texto que se usan para explicar el código o añadir información relevante. Los comentarios no se ejecutan como código, sino que se ignoran por el intérprete de Python. Para escribir un comentario, se usa el símbolo # al principio de la línea. Por ejemplo:

```
# Esto es un comentario

print("Hola mundo") # Esto también es un comentario
```

El espaciado se refiere a los espacios en blanco que se usan para separar los elementos del código, como las palabras clave, los operadores, las variables, etc. El espaciado ayuda a mejorar la legibilidad y el estilo del código. En general, se recomienda usar un espacio entre los operadores y sus operandos, y entre las comas y los elementos de una lista o una tupla. Por ejemplo:

```
x = 3 + 5 # Buen espaciado

y=3+5 # Mal espaciado

z = (1, 2, 3) # Buen espaciado

w = (1,2,3) # Mal espaciado
```

La identación se refiere al sangrado que se usa para delimitar los bloques de código que pertenecen a una misma estructura de control, como un if, un for o una función. La identación es muy importante en Python, ya que define la jerarquía y el orden de ejecución del código. A diferencia de otros lenguajes, Python no usa llaves {} ni puntos y coma ; para marcar el inicio y el fin de un bloque de código, sino que usa la identación como criterio. En general, se recomienda usar cuatro espacios para cada nivel de identación. Por ejemplo:

```
def suma(a, b):
    # Esta es una función que suma dos números
    resultado = a + b
    return resultado


if x > y:
    # Este es un bloque que se ejecuta si x es mayor que y
    print("x es mayor que y")
```

```
else:
    # Este es otro bloque que se ejecuta si x no es mayor que y
    print("x no es mayor que y")
```

Creación de variables en Python

Python tiene una serie de reglas y convenciones que facilitan la lectura y escritura del código. Una de las primeras cosas que debemos aprender a programar en Python es cómo crear variables. Las variables son nombres que se usan para almacenar y referenciar valores en el código. En Python, no es necesario declarar el tipo de las variables, ya que el intérprete lo infiere automáticamente según el valor que se le asigne. Para crear una variable en Python, basta con usar el signo igual (=) seguido del valor que queremos asignarle. Por ejemplo:

```
nombre = "Ana"
edad = 25
pi = 3.1416
```

En este caso, hemos creado tres variables: nombre, edad y pi, y les hemos asignado los valores "Ana", 25 y 3.1416 respectivamente. Podemos usar estas variables en cualquier parte del código donde necesitemos sus valores. Por ejemplo:

```
print("Hola, me llamo", nombre, "y tengo", edad, "años.")
print("El valor de pi es aproximadamente", pi)
```

La función `print()` se usa para mostrar mensajes por pantalla. Como podemos ver, podemos pasarle varias variables o valores separados por comas, y la función los mostrará separados por espacios.

Es importante respetar las mayúsculas y minúsculas al escribir los nombres de las variables, ya que Python las distingue. Por ejemplo, las variables nombre y Nombre son diferentes y pueden tener valores distintos. También debemos evitar usar palabras reservadas del lenguaje como nombres de variables, ya que pueden causar errores o confusiones. Algunas palabras reservadas son: if, for, while, def, class, etc.

En resumen, la sintaxis y estructura básica de Python se basa en el uso de la indentación para definir los bloques de código, el signo igual (=) para crear variables y asignarles valores, y la función print() para mostrar mensajes por pantalla.

Estructuras de control en Python: condicionales y ciclos

En este texto, vamos a ver algunos aspectos básicos de la sintaxis y la estructura de Python, así como algunas de las estructuras de control más comunes que se usan para crear programas: condicionales y ciclos.

La sintaxis de Python se basa en el uso de la indentación para delimitar los bloques de código. Esto significa que hay que respetar los espacios en blanco que se usan al comienzo de cada línea para indicar el nivel de anidación de las instrucciones. Por ejemplo, si queremos definir una función que imprima un mensaje, tenemos que escribir algo así:

```
def saludo():
    print("Hola, mundo")
```

Como vemos, la definición de la función empieza con la palabra reservada `def`, seguida del nombre de la función y los paréntesis. Después, hay que dejar una línea en blanco y escribir el código de la función con una indentación de cuatro espacios. Esta indentación indica que el código pertenece al bloque de la función y se ejecutará cuando se llame a la función.

Python también usa los dos puntos (:) para introducir los bloques de código que dependen de una condición o un ciclo. Por ejemplo, si queremos escribir una sentencia if que compruebe si un número es par o impar, tenemos que hacer algo así:

```
numero = int(input("Ingrese un número: "))
if numero % 2 == 0:
    print("El número es par")
else:
    print("El número es impar")
```

Como vemos, después de la condición del `if` y del `else` hay que poner dos puntos y dejar una línea en blanco. Luego, hay que escribir el código que se ejecutará si la condición se cumple o no con una indentación de cuatro espacios. Esta indentación indica que el código pertenece al bloque del `if` o del `else` y se ejecutará según el resultado de la evaluación.

Los ciclos son estructuras de control que permiten repetir un bloque de código mientras se cumpla una condición o mientras se recorre una secuencia de elementos. En Python hay dos tipos de ciclos: `while` y `for`. El ciclo while se usa para repetir un bloque de código mientras una condición sea verdadera. Por ejemplo, si queremos escribir un programa que cuente desde el uno hasta el diez, podemos hacer algo así:

```
contador = 1
while contador <= 10:
    print(contador)
    contador = contador + 1
```

Como vemos, después de la condición del `while` hay que poner dos puntos y dejar una línea en blanco. Luego, hay que escribir el código que se ejecutará en cada iteración con una indentación de cuatro espacios. Esta indentación indica que el código pertenece al bloque del while y se ejecutará mientras la condición sea verdadera. En este caso, el código imprime el valor del contador y lo incrementa en uno.

El ciclo `for` se usa para recorrer una secuencia de elementos, como una lista, una tupla o un rango. Por ejemplo, si queremos escribir un programa que imprima los números pares del uno al diez, podemos hacer algo así:

```
for numero in range(1, 11):
    if numero % 2 == 0:
        print(numero)
```

Como vemos, después del for hay que poner el nombre de la variable que tomará el valor de cada elemento de la secuencia, seguido de la palabra reservada `in` y la secuencia a recorrer. Después hay que poner dos puntos y dejar una línea en blanco. Luego, hay que escribir el código que se ejecutará en cada iteración con una indentación de cuatro espacios. Esta indentación indica que el código pertenece al bloque del `for` y se ejecutará por cada elemento de la secuencia. En este caso, el código comprueba si el número es par y lo imprime si lo es.

Tipos de datos en Python

Tipos de datos básicos en Python: enteros, flotantes, booleanos y cadenas de texto

Los tipos de datos básicos en Python son aquellos que definen un conjunto de valores con ciertas características y propiedades. Entre ellos se encuentran los enteros, los flotantes, los booleanos y las cadenas de texto.

Los enteros son números sin parte decimal, como 1, -5 o 42. Se pueden realizar operaciones aritméticas con ellos, como suma, resta, multiplicación o división. En Python, los enteros no tienen límite de tamaño, salvo por la memoria disponible. Un ejemplo de código que usa enteros es:

```
a = 10
b = -3
c = a + b
print(c) # imprime 7
```

Los flotantes son números con parte decimal, como 3.14, -2.5 o 6.0. También se pueden realizar operaciones aritméticas con ellos, pero hay que tener en cuenta que pueden tener errores de redondeo debido a la forma en que se almacenan internamente. Un ejemplo de código que usa flotantes es:

```
x = 2.5
y = -1.2
z = x * y
print(z) # imprime -3.0
```

Los booleanos son valores lógicos que pueden ser `True` (verdadero) o `False` (falso). Se usan para expresar condiciones o comparaciones, y se pueden combinar con operadores lógicos como `and`, `or` o `not`. Un ejemplo de código que usa booleanos es:

```
a = True
b = False
c = a and b
print(c) # imprime False
```

Las cadenas de texto son secuencias de caracteres, como "Hola", "Python" o "Tipos de datos". Se pueden crear con comillas simples o dobles, y se pueden manipular con métodos o funciones como `len`, `upper` o `format`. Un ejemplo de código que usa cadenas de texto es:

```
s = "Tipos de datos"

t = s.upper()

print(t) # imprime TIPOS DE DATOS
```

Listas, tuplas y diccionarios en Python

Los tipos de datos básicos en Python son los que permiten almacenar valores simples, como números, booleanos o cadenas de caracteres. Sin embargo, Python también ofrece otros tipos de datos que pueden contener múltiples valores en una sola variable, como las listas, las tuplas y los diccionarios.

Las listas son secuencias ordenadas y mutables de elementos, que pueden ser de cualquier tipo. Se definen con corchetes y se separan los elementos con comas. Por ejemplo:

```
numeros = [1, 2, 3, 4, 5]

nombres = ["Ana", "Juan", "Sofía", "Pablo"]

mezcla = [True, 10.5, "abc", [0, 1, 1]]
```

Las listas se pueden modificar añadiendo o eliminando elementos con métodos como `append` o `remove`. También se puede acceder a los elementos por su índice, que empieza en 0. Por ejemplo:

```
numeros.append(6) # Añade el 6 al final de la lista

nombres.remove("Ana") # Elimina el elemento "Ana" de la lista

print(mezcla[2]) # Imprime el tercer elemento de la lista: "abc"
```

Las tuplas son secuencias ordenadas e inmutables de elementos, que también pueden ser de cualquier tipo. Se definen con paréntesis y se separan los elementos con comas. Por ejemplo:

```
colores = ("Azul", "Verde", "Rojo", "Amarillo")

coordenadas = (10, 20)

vacio = ()
```

Las tuplas no se pueden modificar una vez creadas, por lo que no tienen métodos como `append` o `remove`. Sin embargo, se puede acceder a los elementos por su índice, igual que en las listas. Por ejemplo:

```
print(colores[0]) # Imprime el primer elemento de la tupla: "Azul"

print(coordenadas[-1]) # Imprime el último elemento de la tupla: 20
```

Los diccionarios son colecciones no ordenadas y mutables de pares clave-valor, donde las claves deben ser objetos inmutables y únicos, y los valores pueden ser de cualquier tipo. Se definen con llaves y se separan los pares con comas. Por ejemplo:

```
edades = {"Ana": 25, "David": 18, "Lucas": 35}

productos = {1: "Lápiz", 2: "Goma", 3: "Regla"}

vacio = {}
```

Los diccionarios se pueden modificar añadiendo o eliminando pares con la notación de corchetes o con métodos como `pop` o `update`. También se puede acceder a los valores por su clave con la notación de corchetes o con el método `get`. Por ejemplo:

```
edades["Ximena"] = 30 # Añade un nuevo par al diccionario

productos.pop(2) # Elimina el par con la clave 2 del diccionario

print(vacio.get("clave")) # Imprime el valor asociado a la clave "clave" o None si no existe
```

Conversión entre tipos de datos en Python

La conversión entre tipos de datos es útil cuando queremos manipular los datos de forma diferente según el contexto. Por ejemplo, si queremos concatenar un número con una cadena, debemos convertir el número en una cadena usando la función `str()`. O si queremos obtener la parte entera de un número decimal, debemos convertirlo en un entero usando la función `int()`.

Python tiene funciones integradas para convertir entre los tipos de datos básicos. Estas funciones son:

- `bool()`: convierte un valor en un booleano (`True` o `False`).
- `int()`: convierte un valor en un entero (un número sin decimales).
- `float()`: convierte un valor en un flotante (un número con decimales).
- `complex()`: convierte un valor en un complejo (un número con parte real e imaginaria).
- `str()`: convierte un valor en una cadena (una secuencia de caracteres).

Para usar estas funciones, basta con pasar el valor que queremos convertir como argumento entre paréntesis. Por ejemplo:

```
# Convertir un entero en un flotante
x = 5
y = float(x)
print(y) # 5.0


# Convertir una cadena en un entero
s = "42"
```

```
n = int(s)
print(n) # 42


# Convertir un flotante en una cadena
z = 3.14
t = str(z)
print(t) # "3.14"
```

Algunas conversiones pueden provocar errores si el valor no es compatible con el tipo de dato al que se quiere convertir. Por ejemplo, si intentamos convertir una cadena que no representa un número en un entero, obtendremos una excepción `ValueError`:

```
# Intentar convertir una cadena no numérica en un entero
s = "hola"
n = int(s) # ValueError: invalid literal for int() with base 10: 'hola'
```

También hay que tener cuidado al convertir flotantes en enteros, ya que se pierde la parte decimal del número. Por ejemplo:

```
# Convertir un flotante en un entero
z = 3.14
n = int(z)
print(n) # 3
```

En este caso, el resultado es 3 y no 4, porque Python no redondea el número al entero más cercano, sino que lo trunca al número entero menor.

Operadores en Python

Operadores aritméticos, de asignación y de comparación en Python

Los operadores en Python son símbolos que permiten realizar diferentes tipos de operaciones sobre los datos, como aritméticas, de asignación y de comparación. Estos son algunos de los operadores más comunes en Python:

Operadores aritméticos: son los que realizan operaciones matemáticas sobre los números, como la suma (`+`), la resta (`-`), la multiplicación (`*`), la división (`/`), el módulo (`%`), la potencia (`**`), y la división entera (`//`). Por ejemplo:

```
a = 10
b = 3
c = a + b # Suma
d = a - b # Resta
e = a * b # Multiplicación
```

```
f = a / b # División

g = a % b # Módulo

h = a ** b # Potencia

i = a // b # División entera
```

Operadores de asignación: son los que asignan un valor a una variable, como el igual (=), el más igual (+=), el menos igual (-=), el por igual (*=), el entre igual (/=), el módulo igual (%=), el potencia igual (**=), y la división entera igual (//=). Por ejemplo:

```
a = 10 # Asigna el valor 10 a la variable a

a += 5 # Suma 5 al valor de a y lo asigna a la misma variable

a -= 2 # Resta 2 al valor de a y lo asigna a la misma variable

a *= 3 # Multiplica por 3 el valor de a y lo asigna a la misma variable

a /= 2 # Divide entre 2 el valor de a y lo asigna a la misma variable

a %= 4 # Calcula el módulo de 4 del valor de a y lo asigna a la misma variable

a **= 2 # Eleva al cuadrado el valor de a y lo asigna a la misma variable

a //= 3 # Calcula la división entera de 3 del valor de a y lo asigna a la misma variable
```

Operadores de comparación: son los que comparan dos valores y devuelven un valor booleano (True o False) según el resultado de la comparación, como el mayor que (>), el menor que (<), el igual que (==), el mayor o igual que (>=), el menor o igual que (<=), y el distinto que (!=). Por ejemplo:

```
a = 10

b = 3

c = a > b # True, porque 10 es mayor que 3

d = a < b # False, porque 10 es menor que 3

e = a == b # False, porque 10 no es igual que 3

f = a >= b # True, porque 10 es mayor o igual que 3

g = a <= b # False, porque 10 es menor o igual que 3

h = a != b # True, porque 10 es distinto que 3
```

Operadores lógicos en Python: and, or y not

Los operadores lógicos en Python son palabras clave que se utilizan para combinar o negar expresiones booleanas. Los operadores lógicos en Python son `and`, `or` y `not`. Estos operadores tienen la siguiente sintaxis y significado:

- `and`: devuelve `True` si ambos operandos son `True`, y `False` en cualquier otro caso. Por ejemplo: `True` and `False` devuelve `False`.

- `or`: devuelve `True` si al menos uno de los operandos es `True`, y `False` solo si ambos operandos son `False`. Por ejemplo: `True` or `False` devuelve `True`.

- `not`: devuelve el valor contrario del operando. Por ejemplo: `not True` devuelve `False`.

Los operadores lógicos se pueden utilizar para construir condiciones complejas a partir de expresiones simples. Por ejemplo, si queremos comprobar si una variable x está entre 0 y 10, podemos escribir:

```
if x > 0 and x < 10:
    print("x está entre 0 y 10")
```

Los operadores lógicos también se pueden combinar entre sí, usando paréntesis para indicar el orden de evaluación. Por ejemplo, si queremos comprobar si una variable y es par o múltiplo de 5, podemos escribir:

```
if (y % 2 == 0) or (y % 5 == 0):
    print("y es par o múltiplo de 5")
```

Los operadores lógicos tienen la siguiente precedencia: `not` > `and` > `or`. Es decir, el operador `not` se evalúa antes que el operador `and`, y este antes que el operador `or`. Si hay dudas sobre el orden de evaluación, se recomienda usar paréntesis para aclararlo.

Los operadores lógicos se pueden aplicar a cualquier tipo de dato en Python, no solo a valores booleanos. En general, cualquier valor que no sea cero, vacío o `None` se considera verdadero, y cualquier valor que sea cero, vacío o `None` se considera falso. Por ejemplo:

```
if "" or 0 or None:
    print("Esta línea no se ejecuta")
if "hola" and [1, 2, 3] and 42:
    print("Esta línea sí se ejecuta")
```

Los operadores lógicos devuelven el valor del último operando evaluado, no necesariamente un valor booleano. Por ejemplo:

```
a = 10
b = 20
c = a and b # c vale 20
d = a or b # d vale 10
e = not a # e vale False
```

Cuestionario guia de lectura del material

1. ¿Qué es python y qué ventajas tiene como lenguaje de programación?
2. ¿Cómo se define una función en python y cómo se le pasan argumentos?
3. ¿Qué son las listas y los diccionarios en python y cómo se acceden a sus elementos?
4. ¿Cómo se usa la sentencia if para controlar el flujo de ejecución en python?
5. ¿Qué es el bucle for y cómo se usa junto con la función range?

Bibliografía:

1. Bagnato, J. i., (2020). Aprende Machine Learning en Español: Teoría + Práctica Python. Editorial Leanpub.
2. Britos, P. V., & García Martínez, R. (2009). Propuesta de Procesos de Explotación de Información. In *XV Congreso Argentino de Ciencias de la Computación*.
3. Chazallet, S. (2016). Python 3: los fundamentos del lenguaje. Ediciones ENI.
4. Geron, A., (2020). Aprende Machine Learning con Scikit-Learn, Keras y Tensor Flow: Conceptos, herramientas y técnicas para construir sistemas inteligentes. Editorial O'Reilly y Anaya
5. Hilera, J. R. y Martinez, V. J. (2000) Redes Neuronales Artificiales. Fundamentos. modelos y aplicaciones. Alfaomega Ed
6. JIMÉNEZ, R. O., (2021). Python a fondo. Editorial Marcombo
7. Kuna, H. D., Caballero, S., Rambo, A., Meinl, E., Steinhilber, A., Pautsch, J., ... & Villatoro, F. (2010). Avance en procedimientos de la explotación de información para la identificación de datos faltantes, con ruido e inconsistentes. In *XII Workshop de Investigadores en Ciencias de la Computación*.
8. Kuna, H., Pautsch, G., Rey, M., Cuba, C., Rambo, A., ... & Villatoro, F. (2012). Obtenido de COMPARACIÓN DE LA EFECTIVIDAD DE PROCEDIMIENTOS DE LA EXPLOTACIÓN DE INFORMACIÓN PARA LA IDENTIFICACIÓN DE OUTLIERS EN BASES DE DATOS:
9. Matthes, E. (2021) Curso intensivo de Python, 2ª edición: Introducción práctica a la programación basada en proyectos. Editorial Anaya Multimedia
10. Ochoa, M. A. (2004). Herramientas inteligentes para explotación de información. Trabajo Final Especialidad en Ingeniería de Sistemas Expertos url: https://ri.itba.edu.ar/server/api/core/bitstreams/a848d640-0277-459d-9104-b37017309d31/content