Sintáxis Básica de Python

Python

Python es un lenguaje de programación poderoso y rápido, se lleva bien con otros lenguajes, corre en cualquier lugar, es amigable, fácil de aprender y es de software libre.

Sintáxis

En la gran mayoría de los lenguajes de programación el compilador o intérprete ignora los espacios que el usuario utilice para escribir su código. En Python, los espacios se utilizan para formar grupos de instrucciones y también para la sintaxis de algunas instrucciones.

If-Elif-Else

La sentencia de control `if-elif-else` tiene la siguiente sintaxis:

Sintaxis:

```
if condición A:
    instrucción A1
    instrucción A2
    ...
    instrucción An
elif condición B:
    instrucción B1
    instrucción B2
    ...
    instrucción Bn
else:
    instrucción C1
    instrucción C2
    ...
    instrucción Cn
```

Ejemplo 1. if-else

```
    if 2<3:
        print("verdadero")
    else:
        print("falso")
    print("fin")
```

Ejemplo 2. Solicite dos números x y `y`. Si `x` es positivo `y` se multiplica por dos, si `x` es cero `y` se multiplica por 3 y si `x` es negativo y se multiplica por cuatro.

```
x=int(input("Ingrese el valor de x "))
y=int(input("Ingrese el valor de y "))
if x>0:
    z=y*2
    print("Positivo")
elif x==0:
    z=y*3
    print("Cero")
else:
    z=y*4
    print("Negativo")
print(z)
```

For

El ciclo `for` tiene la siguiente sintaxis:

```
for iterador in iterando:
    instrucción 1
    instrucción 2
    ...
    instrucción n
```

Ejemplo 3. Un ciclo que imprime numeros enteros desde 0 hasta 9.

```
for i in range(10):
    print(i)
```

La función `range` genera una secuencia de números enteros comenzando desde 0, con un incremento unitario. Por esa razón es que se generan los números enteros desde 0 hasta 9. Es posible cambiar el valor inicial de la secuencia y el incremento.

Ejemplo 4. Considere los primeros 10 números naturales, los pimeros 5 urales se multiplican por 2 y los siguientes números naturales se multiplican por 3.

```
for x in range(1,11):
    if x<=5:
        print(f'{x} -> {x*2}')
    else:
        print(f'{x} -> {x*3}')
```

While

El ciclo `while` repite las instrucciones siempre y cuando la condición sea verdadera. La sintaxis del ciclo `while` es la siguiente:

```
while condición:
    instrucción 1
```

```
    instrucción 2
    ...
    instrucción n
```

Ejemplo 5. Se imprimen los primeros 10 numeros naturales.

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

Listas

Creación

Una lista es un conjunto de valores ordenados y modificables que permite repeticiones. Los elementos en la lista son numerados comenzando desde 0, de manera que la localidad donde se encuentra el último elemento para una lista de tamaño `n` será `n-1`.

```
lista = ["manzana", "plátano", "naranja", "mandarina", "maracuyá"]
print(lista)
```

Acceso

El acceso a algún elemento en particular de la lista se hace utilizando los corchetes, indicando dentro de ellos la posición que ocupa dentro de la lista.

```
lista[posición]
```

Índices negativos

Es posible usar numeración negativa para hacer referencia a los elementos dentro de una lista pero en orden reverso. Es decir, `-1` hace referencia al último elemento, `-2` al penúltimo, `-3` al antepenúltimo y así sucesivamente.

Rango de índices

Es posible seleccionar un subconjunto de elementos contiguos contenidos en la lista especificando el rango de posiciones que ocupan dentro de la lista.

```
lista = ["manzana", "plátano", "naranja", "mandarina", "maracuyá", "toronja", "mango", "
print(lista[2:6])
```

Por otro lado, si se omite el límite inferior del rango, python considerará 0 como posición inicial.

```
lista = ["manzana", "plátano", "naranja", "mandarina", "maracuyá", "toronja", "mango", "
print(lista[:6])
```

Finalmente, si se omite el límite superior del rango, python considerará el último elemento como tal límite.

```
lista = ["manzana", "plátano", "naranja", "mandarina", "maracuyá", "toronja", "mango", '
print(lista[2:])
```

Tamaño de una lista

Para determinar el tamaño de una lista se puede utilizar la función `len()`

```
print(len(lista))
```

Funciones

En python la declaración de funciones requiere muy poco código. Basta con utilizar la palabra clave `def` para comenzar la definición de la función.

```
def nombreFuncion(parámetroEntrada):
    instrucción 1
    instrucción 2
    ...
    instrucción n
    return parámetroSalida
```

Ejemplo 6. Escriba una función que calcule el factorial de una función. Verifique que el número ingresado por el usuario sea positivo y considere que por definición el factorial de cero es uno.

```
def factorial(x):
    if x>=0:
        f=1
    if x>0:
        f=1
        for i in range(1,x+1):
        f=f*i
        return f
    else:
        print("El factorial no está definido")

x = int(input("Introduzca un número: "))
print(f'{x}! = {factorial(x)}')
```

Ejemplo 7. Escriba una función que calcule el factorial de una función. Considere la defincón recursiva del factorial. Verifique que el número ingresado por el usuario sea positivo y considere que por definición el factorial de cero es uno.

```
def factorial(x):
    if x==0:
        f=1
```

```
    else:
        f=x*factorial(x-1)
    return(f)

x = int(input("Ingrese un número"))
if x>=0:
    fac=factorial(x)
    print(f'{x}! = {fac}')
else:
    print("El factorial no está definido en los negativos")
```

Ejemplo 8. Escriba una función recursiva que imprima los elementos de una lista anidada en varios niveles.

```
lista1 = ["manzana", "plátano", "naranja", "mandarina", "maracuyá", "toronja", "mango",
lista2 = [1, 2, 3, 4, 5]
lista3 = ["A", "B", "C", "D"]
lista4 = ["a", 1, "b", 2, "c", 3, "d", 4]
superLista = [lista1 + lista2 + lista3] + [[lista4]] + lista1

def impresionRecursiva(listaAnidada):
for elemento in listaAnidada:
    if isinstance(elemento, list):
        impresionRecursiva(elemento)
    else:
        print(elemento)

impresionRecursiva(superLista)
```

Arreglos

Python no tiene de forma nativa soporte para arreglos, en su lugar opta por usar listas anidadas. Sin embargo, es posible utilizar el paquete NumPy para utilizar los arreglos de manera semejante a la existente en otros lenguajes. **NumPy** además resulta ser más eficiente en el manejo de datos que su contraparte nativa de python mediante listas anidadas. Adicionalmente, **NumPy** incluye más herramientas que extienden la funcionalidad de Python.

Declaración

Sintaxis:

```
import numpy
arreglo = numpy.array([1, 2, 3, 4, 5])
print(arreglo)
```

Esta sintaxis puede resultar incómoda porque será necesario escribirla todas las veces que necesite declarar un arreglo. Una alternativa para simplificar un poco

esta declaración es mediante la creación de un alias, esto de la siguiente manera:

```
import numpy as np
arreglo = np.array([1, 2, 3, 4, 5])
print(arreglo)
```

Para declarar un arreglo bidimensional se utiliza la siguiente sintáxis:

```
import numpy as np
arreglo = np.array([[1, 2, 3], [4, 5, 6]])
print(arreglo)
```

El atributo `ndim` devuelve la cantidad de dimensiones que tiene un arreglo.

```
import numpy as np
arreglo = np.array([[1, 2, 3], [4, 5, 6]])
print(arreglo.ndim)
```

Acceso a elementos de un arreglo

El acceso a elementos dentro de un arreglo en `numpy` es similar a la forma que se utiliza para las listas. Recuerde que elos índice para los elementos dentro del arreglo comienza en 0.

```
import numpy as np
arreglo = np.array([1, 2, 3, 4, 5])
print(arreglo[0])
```

Para el caso de un arreglo bidimensional se utiliza una coma para separar la posición de las dimensiones.

```
import numpy as np
arreglo = np.array([[1, 2, 3], [4, 5, 6]])
print(arreglo[1, 2])
```

Si el arreglo tiene más dimensiones se utiliza la misma idea para cada una de ellas.

```
import numpy as np
arreglo = np.array([[[1, 2, 3], [4, 5, 6]],[[7, 8, 9], [10, 11, 12]])
print(arreglo[0, 1, 2])
```

De la misma forma que con las listas, también es posible utilizar índices negativos.

```
arreglo = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('El último elemento en el arreglo bidimensional', arreglo[1, -1])
```

Cortes de arreglos

Es posible *cortar* un subconjunto de un arreglo para definir uno nuevo. Esto es de especial utilidad para extraer vectores de una matriz existente, ya sea para definir un nuevo vector o bien realizar operaciones con el.

El corte (o rebanada) de la matriz se hace indicando un rango de posiciones, es decir `[inicio : fin]`. Además se puede especificar un incremento `[inicio : fin : incremento]`. Si no se especifica un inicio, de asume como 0, y si no se especifica un final se asume el último elemento de la matriz. Si no se especifica un incremento, se asume como 1.

```
import numpy as np
arreglo = np.array([1, 2, 3, 4, 5])
arreglo2 = arreglo[1:4]
print(arreglo2)
```

En el siguiente ejemplo, se hace una rebanada de la matriz especificando un incremento diferente a uno:

```
import numpy as np
arreglo = np.array([1, 2, 3, 4, 5, 6, 7])
print(arreglo[1:5:2])
```

Cortes de arreglos bidimensionales

Es posible realizar rebanadas de arreglos de 2 ó más dimensiones, resultando un vector o una matriz según sea el caso.

El siguiente ejemplo realiza una rebanada de un arreglo bidimensional y el resultado es un vector. Observe que el vector es una rebanada de la segunda dimensión de la matriz.

```
import numpy as np
arreglo = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arreglo[1, 1:4])
```

En este ejemplo se hace una rebanada de un arreglo bidimensional y el resultado es nuevamente un vector. En esta ocasión el vector es una rebanada vertical por lo que el resultado contiene elementos de ambas dimensiones de la matriz.

```
import numpy as np
arreglo = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arreglo[0:2, 2])
```

En este último caso, se hace una rebanada que resulta una matriz que contiene elementos de ambas dimensiones de la matriz original.

```
import numpy as np
arreglo = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arreglo[0:2, 1:4])
```

Arreglos aleatorios

Es posible generar un arreglo o matriz lleno de números (pseudo) aleatorios, para ello se puede utilizar el método `rand`. Para la generación, basta con especificar las dimensiones del arreglo que se desea generar.

Ejemplo. Generación de un vector unidimensional de 20 elementos aleatorios.

```
import numpy as np
arreglo = np.random.rand(20)
print(arreglo)
```

Ejemplo. Generación de una matriz bidimensional de tamaño `5x3`, es decir 5 renglones y 3 columnas.

```
import numpy as np
arreglo = np.random.rand(5, 3)
print(arreglo)
```

Ejemplo. Generación de una matriz tridimensional de tamaño `5x3x2`, es decir 5 matrices de 3 renglones y 2 columnas cada una de ellas.

```
import numpy as np
arreglo = np.random.rand(5, 3, 2)
print(arreglo)
```

Añadir elementos a un arreglo

Para añadir elementos al final de un arreglo, se puede utilizar el método `append` contenido en la biblioteca `Numpy`. El argumento `axis` permite especificar el lugar donde serán añadidos los elementos al arreglo.

Ejemplo. Añadir una matriz al final de una matriz, justo debajo de la matriz inicial (`axis=0`).

```
import numpy as np
arreglo = np.append([[1, 2], [3, 4]], [[10, 20], [30, 40]], axis=0)
print(arreglo)
```

Ejemplo. Añadir una matriz al final de la matriz, a la derehca de la matriz inicial (`axis=1`).

```
import numpy as np
arreglo = np.append([[1, 2], [3, 4]], [[10, 20], [30, 40]], axis=1)
print(arreglo)
```

Formateo de impresión de un arreglo

Si se desea controlar la forma en la que los números contenidos en un arreglo serán impresos, se puede utilizar el método `printoptions` de la biblioteca `Numpy`.

Ejemplo. Generar una matriz bidimensional de tamaño 10x5 llena con números aleatorios no enteros con valores entre 0 y 1. Imprimir la matriz mostrando únicamente 4 cifras significativas no enteras.

```
import numpy as np
x = np.random.rand(10,5)
    with np.printoptions(precision=4, suppress=True):
```

```
np.set_printoptions(formatter={'float': '{: 0.4f}'.format})
print(x)
```

La función `with` permite especificar el formato mediante el cual serán impresos unicamente los números en el arreglo, dejando intactos los parámetros de los demás `print` que pudiesen existir. El parámetro `suppress=True` indica que los números deben ser impresos en la forma de punto flotante y evitando la notación científica. :::