PYTHON FOR BIG DATA DEVELOPERS

Ana Soto Jiménez

ÍNDICE

6.3 Carga y lectura de ficheros

- 6.3.1 Lectura y escritura de ficheros: CSV, facta
- 6.3.2 Carga y lectura de CVS en Databricks
- 6.3.3 Leer y escribir JSON y comprimirlos
- 6.3.4 Leer y escribir ORC y comprimirlos
- 6.3.5 Leer y escribir PARQUET y comprimirlo

6.4 Estructura de Datos

- 6.4.1 Dataframes
- 6.4.2 Matrices
- 6.4.3 Listas
- 6.4.4 RDD
  - 6.4.4.1 Creación de RDD
    - 6.4.4.1.1 A partir de colecciones de datos
    - 6.4.4.1.2 A partir de diccionarios
    - 6.4.4.1.3 A partir de lectura de ficheros
  - 6.4.4.2 Almacenamiento RDD
    - 6.4.4.2.1 Escritura de RDD en fichero
  - 6.4.4.3 Operaciones con RDD
    - 6.4.4.3.1 Operaciones básicas con listas RDD
    - 6.4.4.3.2 Operaciones básicas con diccionarios
    - 6.4.4.3.3 Ordenar RDD
    - 6.4.4.3.4 Filtrado de RDD
    - 6.4.4.3.5 Muestreo de datos RDD
    - 6.4.4.3.6 Operaciones con conjuntos
    - 6.4.4.3.7 Operaciones Map/Reduce

6.5 Procesamiento de datos

- 6.5.1 Por lotes (BATCH)
  - 6.5.1.1 Definir un dataframe “pyspark”
  - 6.5.1.2 Realizar agrupaciones sobre datos
  - 6.5.1.3 Realizar consultas SQL
- 6.5.2 En tiempo real (Streaming)
  - 6.5.2.1 Preparación de datos que simulan tiempo real
  - 6.5.2.2 Realizar agrupaciones sobre datos
  - 6.5.2.3 Establecer la configuración de flujo de datos del clúster
  - 6.5.2.4 Consultas SQL

6.6 Tuberías (Pipelines)

- 6.6.1 Uso de pipelines
- 6.6.2 Uso de pipelines en árboles de decisión
- 6.6.3 Uso de pipelines en k-means

6.7 Machine Learning

- 6.7.1 Árboles de decisión (simples)
- 6.7.2 Regresión lineal simple
- 6.7.3 Regresión lineal siempre – spark
- 6.7.4 Regresión logistics
- 6.7.5 K-means spark

1. INTRODUCCIÓN

1.1 - ¿Qué es Python?

Python es un lenguaje de programación popular. Fue creado en 1991 por Guido van Rossum. Se utiliza para: desarrollo web, desarrollo de software, matemáticas y sistema de scripting.

1.2 - ¿Qué puede hacer?

Python se puede utilizar en un servidor para crear aplicaciones web. Python se puede utilizar junto con el software para crear flujos de trabajo, además puede conectarse a sistemas de bases de datos.

También puede leer y modificar archivos, Python se puede utilizar para manejar grandes volúmenes de datos y realizar matemáticas complejas.

Python se puede utilizar para la creación rápida de prototipos o para el desarrollo de software listo para producción.

1.3 - ¿Por qué Python?

- Python funciona en diferentes plataformas (Windows, Mac, Linux, Raspberry Pi, etc.).
- Tiene una sintaxis simple similar al idioma inglés.
- Sintaxis que permite a los desarrolladores escribir programas con menos líneas que otros lenguajes de programación.
- Se ejecuta en un sistema de intérpretes, lo que significa que el código se puede ejecutar tan pronto como se escribe. Esto significa que la creación de prototipos puede ser muy rápida.
- Python puede tratarse de forma procesal, orientada a objetos o funcional.

1.4 – Sintaxis de Python en comparación con otros lenguajes de programación

- Fue diseñado para facilitar la lectura, y tiene algunas similitudes con el idioma inglés con influencia de las matemáticas.
- Python usa nuevas líneas para completar un comando, a diferencia de otros lenguajes de programación que a menudo usan punto y coma o paréntesis.

- Se basa en la sangría, utilizando espacios en blanco, para definir el alcance; Como el alcance de los bucles, funciones y clases. Otros lenguajes de programación a menudo utilizan rizos para este propósito.

2. EMPEZAR CON PYTHON

2.1 Instalar

Algunos pc y Mac tendrán Python ya instalado.

Para comprobar si tiene Python instalado en una PC con Windows, busque Python en la barra de inicio o ejecute lo siguiente en la línea de comandos (cmd.exe):

C:\Uuarios\tu nombre>python --version

Para verificar si tienes Python instalado en Linux o Mac, en linux abre la línea de comandos o en Mac abre la Terminal y escribe:

python --version

Si descubre que no tiene Python instalado en su computadora, puede descargarlo gratuitamente del siguiente sitio web: https://www.python.org/

2.2 Primer programa con Python

Python es un lenguaje de programación interpretado, esto significa que, como desarrollador, usted escribe archivos Python (.py) en un editor de texto y luego los coloca en el intérprete de Python para su ejecución.

La forma de ejecutar un archivo python es la siguiente en la línea de comandos:

C:\Uuarios\tu nombre >python helloworld.py

Donde «**voyaaprenderpython.py**» es el nombre de tu archivo python.
Escribamos nuestro primer archivo de Python, llamado **voyaaprenderpython.py** , que se puede hacer en cualquier editor de texto.

```
print("!Estoy aprendiendo python!")
```

voyaaprenderpython.py

Simple como eso. Guarda tu archivo. Abra su línea de comando, navegue hasta el directorio donde guardó su archivo y ejecute:

C:\Usuarios\tu nombre >python helloworld.py

La salida debe leer:

!Estoy aprendiendo python!
Enhorabuena, has escrito y ejecutado tu primer programa de Python.

2.3 – Comandos
Para probar una pequeña cantidad de código en Python, a veces es más rápido y más fácil no escribir el código en un archivo. Esto es posible porque Python se puede ejecutar como una línea de comandos.

Escriba lo siguiente en la línea de comando de Windows, Mac o Linux:

C:\Usuarios\tu nombre>python
C:\Uuarios\tu nombre>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("!Estoy aprendiendo python!")

¡Que escribirá «! Estoy aprendiendo python! » en la línea de comando:

C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("!Estoy aprendiendo python!")
!Estoy aprendiendo python!

! Estoy aprendiendo python!

Cuando termines en la línea de comandos de python, puedes simplemente escribir lo siguiente para salir de la interfaz de línea de comandos de python:

exit()

3. PYTHON

3.1 Sintaxis

La sintaxis de Python se rige por una serie de reglas generales que sirven para dar orden, coherencia y limpieza al lenguaje.

Un programa escrito en Python se divide en un número determinado de líneas lógicas, cuyo final está determinado por el token o señal de nueva línea (new line).
De esta forma, se va organizando el código siguiendo reglas sintácticas fundamentales que constituyen la estructura lexicográfica del lenguaje en la que escribes un programa python.

- Líneas Físicas VS Líneas Lógicas

  Las líneas físicas son secuencias de caracteres que concluyen con el carácter de fin de línea (\n en sistemas Unix-Like o \r\n en Windows).

  Nota: Las líneas físicas en la práctica no son más que las líneas que enumera tu editor de código.

  Las líneas lógicas, por su parte, son componentes lógicos de la sintaxis de Python, cuyo final está determinado por el token o señal de nueva línea (new line). Este token determina el fin de una línea lógica y da el comienzo de otra.

  Una línea lógica puede estar compuesta por una o varias líneas físicas de un archivo de texto plano.

  Las líneas físicas se pueden unir para formar una única línea lógica para lo cual se puede emplear el carácter de barra invertida \ colocado como carácter de escape justo antes de la secuencia de fin de línea. Este proceder se denomina Unión Explícita de Líneas y es considerado como una mala práctica y generalmente debe evitarse, pero aún así, es sintácticamente legal hacerlo.

  Uno de los riesgos que implica este tipo de unión de líneas físicas, es que por error se incluya, por ejemplo, un carácter de espacio inmediatamente detrás del carácter \, lo cual constituye un error sintáctico que puede ser difícil de encontrar. Por suerte, la mayoría de los editores e IDEs actuales incluyen la opción de eliminar automáticamente los caracteres de espacio al final de las líneas físicas. Un ejemplo es:

  ```
  # Una línea lógica muuuuuy larga
  circle_area = math.pi * radius ** 2.0 if radius > 0.0 and radius != float ('inf')\
  else None
  ```

  Las líneas físicas también se pueden unir en una única línea lógica, empleado los pares de caracteres (), [] y {}. Esto se conoce como Unión Implícita de Líneas.

Si una línea lógica se inicia con un paréntesis (, se extenderá por tantas líneas físicas como sea necesario y solo terminará con el carácter de cierre, es decir,). Esto también se cumple con los corchetes y llaves, [], {}.

La unión implícita es la forma recomendada y generalmente empleada por la mayoría de los programadores con experiencia en Python. Un ejemplo de este tipo de unión de líneas físicas podría ser:

>>>print (
..... 'Hello',
..... 'World!'
……)
Hello world!

- Sentencias Simples y Compuestas
  Existen dos tipos de sentencias en Python:
  - Las sentencias simples, que deben completarse en una única línea lógica, como, por ejemplo:
    >>>from sys import platform
  - Las sentencias compuestas, que deben comenzar con una cláusula de sentencia compuesta y deben contener sentencias simples y/o compuestas indentadas, a las cuales se les suele llamar cuerpo o bloque. La cláusula inicial o encabezado de una sentencia compuesta inicia siempre con una palabra clave (keyword) y termina con el carácter de dos puntos : , Por ejemplo:
    >>> if a > b
    ……. print (a, 'is greater than' , b)

  A diferencia de otros lenguajes, Python no riene declaraciones u otros elementos sintácticos de alto nivel, solo sentencias, que generalmente ocupan una o varias líneas físicas en tu editor.

  El fin de una línea física, generalmente determina el fin de la mayoría de las sentencias. Las líneas físicas terminan con la secuencia de fin de línea \n (o \r\n en Windows).
  >>>> var = 'Welcome to Python Scouts!' #Línea física que termina con la secuencia de fin de línea \n
  >>>>>>>
  Además, es válido emplear el carácter punto y coma ; para terminar las sentencias.
  >>>>> print (var); #Sentencia que termina con ;
  Welcome to Python Scouts!
  O para incluir varias sentencias simples en una misma línea física:

>>>>var1 = 0; var2 = 1 #Empleo del ; para separar dos sentencias en una misma línea física
>>>>var1
0
>>>>var2
1
El empleo del ; para separar varias sentencias en una misma línea física es considerado una mala práctica, pues atenta contra la legibilidad del código y no se apega al estilo de codificación aceptado en Python, por tanto, se recomienda evitarlo.

- Indentación

  Python, a diferencia de otros lenguajes, no emplea llaves {} o estructuras begin…end para definir bloques de código. Para esto, el lenguaje se vale de lo que se conoce como indentación, que no es más que la inclusión de espacios o caracteres de tabulación al inicio de las líneas lógicas.

  La indentación del código tiene su origen en la necesidad de hacer que el código sea más legible y comprensible. Esta es la razón fundamental por la cual Python la hace parte de su sintaxis, esencialmente para mejorar la legibilidad , comprensión y sencillez del código que es, sin duda, uno de los rasgos identificativos y más valorados del lenguaje.

  Las sentencias pueden ser agrupadas dentro de una cláusula o cabecera (header) de sentencia compuesta mediante la indentación.

  Python usa la indentación de las líneas lógicas, para determinar la agrupación de sentencias y su pertenencia a determinado bloque o cuerpo de sentencia compuesta. Por ejemplo:

```
def printer ();
        #Los espacios al inicio de estas líneas forman la
        #indentación
print (`Hello world!')
print ('Welcome to Python Scouts')
        #Las sentencias anteriores forman el bloque o cuerpo de
        #la función printer()
printer() #El llamado a printer() queda fuera del bloque, pues
#ya no hay indentación
```

Como ya sabes, la indentación debe ser la misma (igual cantidad de espacios), al menos para las líneas que componen un mismo bloque de código y la primera sentencia de un archivo de código.

La indentación es un componente sumamente importante en la sintaxis de Python.

- Comentarios
  Un comentario es una secuencia de caracteres que comienza con el carácter de numeral # y continúa hasta el final de la línea física.

  Las líneas físicas que comienzan con # son ignoradas completamente por el intérprete de Python. En realidad, estas líneas están dirigidas a los programadores/mantenedores/clientes del código, que en muchas ocasiones serás tú mismo.

  Los comentarios son útiles para indicar lo que estás haciendo y sobre todo, por qué lo haces, siempre y cuando no sea algo obvio, en cuyo caso resultan redundantes y superfluos. Por ejemplo:

  >>>> #Esto es un comentario que comienza con # y es ignorado por Python

  >>>>

  >>>> # Le asigno el valor 0 a count <=Esto es un comentario innecesario

  >>>>count = 0

  >>>> # Inicializo count a 0 para contar las veces que…. <= MEJOR

  >>>> count = 0

  Existe un tipo especial de comentario denominado declaración de codificación (encoding declaration), que se incluye en la primera o segunda línea de un módulo o script de Python para declarar explícitamente la codificación del texto contenido en el archivo. Un ejemplo de este tipo de declaración es:

  # -- coding: utf-8 -*-*

  # La declaración de codificación se coloca en la primera o segunda línea del módulo.

  En la mayoría de los casos, este tipo de declaración no es necesaria, pues casi todos los IDEs y editores de texto manejan esto de forma transparente. En el caso de Python 3.X, el estándar de codificación es

el UTF-8, mientras que la rama 2.X del lenguaje, el estándar suele ser ASCII.
Finalmente, en muchas ocasiones incluirás en tu código un tipo de comentario que se denomina comentarios en línea (in-line comments). Se trata de un comentario que escribes a continuación de una sentencia o expresión. Por ejemplo:

> if temperatura <= 273: #Validar la temperatura <= Comentario en línea

> raise ValueError('Wrong temperatura value')

Cuando unes las líneas físicas mediante el mecanismo de unión implícita que viste con anterioridad, puedes incluir comentarios en línea, como, por ejemplo:

>>> month_names = ['January', 'Frebuary', March', #Primer trimestre

'April', 'May', 'June', #Segundo trimestre

'July', 'August', 'September', #Tercer trimestre

'October', 'November', 'December'] #Cuarto trimestre

- Espacios y líneas en blanco

  Los espacios en blanco pueden ser empleados libremente en el interior de las sentencias (entre tokens). A excepción del inicio de línea, donde los espacios se interpretan como indentación y determinan la pertenencia de una sentencia simple a una compuesta.

  Las líneas en blanco contienen solo caracteres de espacio, tabulación y fin de línea. Su empleo está estrechamente ligado a la legibilidad del código.

  Por otro lado, si estás en una sesión interactiva del intérprete o Ciclo de Lectura, evaluación, impresión (REPL), una línea en blanco representa la conclusión de una sentencia compuesta.

  Existen recomendaciones bien establecidas en el PEP 8 con relación al empleo de espacios y líneas en blanco y es aconsejable qye las sigas a fin de que tu código presente la apariencia de un código Python bien escrito.

- Tokens del lenguaje

  Los tokens son componentes lexicográficos elementales que forman cada una de las líneas lógicas. En las sintaxis de Python se reconocen los tokens o señales siguientes:

  - *NewLine:* determina el fin de una línea lógica y el comienzo de otra.

  - *Indent:* indentación de las sentencias dentro de una sentencia compuesta.

  - *Dedent:* fin de la indentación que determina el fin de una sentencia compuesta.

  - *Identifiers:* nombres que identifican a variables, funciones, clases, métodos, constantes, módulos, paquetes… Los identificadores comienzan con las letras (A – Z, a – z) o con guiones bajos _ seguidos de cero o más letras, guiones bajos o dígitos (0-9). Python es un lenguaje Case Sensitive, lo que significa que las letras mayúsculas y minúsculas son distintas.

  - *Palabras clave o reservadas (keywords):* palabras con significado especial para el lenguaje, que no pueden ser empeladas como identificadores. Algunas palabras clave son sentencia simple, (Ej. *break, continue).* Otros, cláusulas de sentencias compuestas (Ej. *def, class, for, while)*. Mientras que otras son operadores (Ej. *and, or, is, in).* Las palabras reservadas de Python se pueden consultar tecleando en el prompt intérprete, las sentencias siguientes:

    >>>>improt keyword

    >>>> keyword.kwlist

    ['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

  - Literales (literals): valores numéricos o de cadena de caracteres que aparecen directamente escritos en el código. Por ejemplo:

>>>'Hello world' #Literal de string

'Hello World'

>>>>1452.25 #Literal de float

1452.25

>>>> 15 #Literal de int

15

>>>1_000_000 #Literal de int con guion bajo de agrupación

1000000

- Operadores (operators): caracteres empleados para denotar operaciones diversas tales como: aritméticas, lógicas, de asignación… Los operadores actuales del lenguaje son:

  + - * ** / // % @ << >>

  & | ^ ~ < > <= >= == !=

- Delimitadores (delimiters): caracteres empleados para delimitar literales, líneas lógicas, entre otras. Python incluye los siguientes:

  ( ) [ ] { }

  , : . ; @

  = -> += -= *=

  /= //= %= @= &=

  |= ^= >>= <<= **=

3.2 Variables

A diferencia de otros lenguajes de programación, Python no tiene comandos para declarar una variable. Una variable se crea en el momento en que primero le asigna un valor.

```
x = 15
y = "Ana"
print(x)
print(y)
```

Las variables no necesitan ser declaradas con ningún tipo en particular e incluso pueden cambiar el tipo después de que se hayan establecido.

Ejemplo

```
x = 6 # x is of type int
x = "Ramón" # x es de tipo str
print(x)
```

- Reglas de las variables en Python
  - ✓ Un nombre de variable debe comenzar con una letra o el carácter de subrayado
  - ✓ Un nombre de variable no puede comenzar con un número
  - ✓ Un nombre de variable solo puede contener caracteres alfanuméricos y guiones bajos (A-z, 0-9 y _).
  - ✓ Los nombres de las variables distinguen entre mayúsculas y minúsculas (age, Age y AGE son tres variables diferentes)

- Variables de salida

  La declaración de impresión de Python se usa a menudo para generar variables.

  Para combinar texto y una variable, Python usa el carácter +:

  Ejemplo

  ```
  x = "inteligente"
  print("Ana is " + x)
  ```

  También puede usar el carácter + para agregar una variable a otra variable:

  Ejemplo

  ```
  x = "Ana es "
  y = "inteligente"
  z =  x + y
  print(z)
  ```

Para los números, el carácter + funciona como un operador matemático:

Ejemplo

```
x = 2
y = 8
print(x + y)
```

Si intentas combinar una cadena y un número, Python te dará un error:

```
x = 3
y = "Ana"
print(x + y)
```

3.3 Números

Hay tres tipos numéricos en Python:

int
float
complex

Las variables de tipos numéricos se crean cuando se les asigna un valor:

```
x = 6    # int
y = 4.5  # float
z = 3m  # complex
```

Para verificar el tipo de cualquier objeto en Python, use la función type ():

Ejemplo

```
print(type(x))
print(type(y))
print(type(z))
```

- Enteros o int
  Es un número entero, positivo o negativo, sin decimales, de longitud ilimitada.
  Ejemplo:

```
x = 6
y = 12345678910
z = -7654321

print(type(x))
print(type(y))
print(type(z))
```

- Float

  Float, o «número de punto flotante» es un número, positivo o negativo, que contiene uno o más decimales.

  Ejemplo

  Floats:

  x = 5.50
  y = 5.0
  z = -40.35

  print(type(x))
  print(type(y))
  print(type(z))

  El float también puede ser un número científico con una «u» para indicar el valor de 5.

  Ejemplo

  x = 13u6
  y = 43u2
  z = -40.6u89

  print(type(x))
  print(type(y))
  print(type(z))

- Complex

  Los números complex se escriben con una «j» como parte imaginaria:

  Ejemplo

  x = 2+3j
  y = 3j
  z = -3j

  print(type(x))
  print(type(y))
  print(type(z))

3.4 Castings

Puede haber ocasiones en las que desee especificar un tipo en una variable. Esto se puede hacer con el casting.

Python es un lenguaje orientado a objetos, y como tal utiliza clases para definir tipos de datos, incluidos sus tipos primitivos. Por lo tanto, la conversión en python se realiza mediante funciones de constructor:

- Integers (int):
  Construye un número entero a partir de un literal entero, un literal flotante (redondeando hacia abajo al número entero anterior), o un literal de cadena (siempre que la cadena represente un número entero)

  ```
  x = int(3) # x obtendrá 3
  y = int(3.5) # y obtendrá 3
  z = int("3") # z obtendrá 3
  ```

- Float (float())

  Construye un número flotante a partir de un literal entero, un literal flotante o un literal de cadena (siempre que la cadena represente un flotante o un entero)

  ```
  x = float (3) # obtendrás 3.0
  y = float (3.1) # obtendrás 3.1
  z = float ("3") # obtendrás 3.0
  w = float ("3.5") # obtendrás 3.5
  ```

- Strings (str())

  Construye una cadena a partir de una amplia variedad de tipos de datos, incluidas cadenas, literales enteros y literales flotantes

  ```
  x = str ("d4") #obtendrás 'd4'
  y = str (3) # obtendrás '3'
  z = str (5.0) # obtendrás '5.0'
  ```

3.5 Strings

- Literales de cuerda

Los literales de cadena en python están rodeados por comillas simples o comillas dobles. ‘hola’ es lo mismo que «hola».
Las cadenas se pueden enviar a la pantalla utilizando la función de impresión.
Por ejemplo*: print(«hello»).*
Como muchos otros lenguajes de programación populares, las cadenas en Python son matrices de bytes que representan caracteres Unicode.
Sin embargo, Python no tiene un tipo de datos de caracteres, un solo carácter es simplemente una cadena con una longitud.
Se pueden usar corchetes para acceder a los elementos de la cadena.
Ejemplo
Obtenga el carácter en la posición 1 (recuerde que el primer carácter tiene la posición 0):

```
a = "Hola Mundo"
print(a[1])
```

Ejemplo

Subcadena - Obtenga los caracteres de la posición 4 a la posición 6 (no incluidos):

```
b = "Hola Mundo"
print(b[4:6])
```

Ejemplo

El método strip () elimina cualquier espacio en blanco desde el principio o el final:

```
a = "Hola Mundo"
print(a.strip()) # returns "Hola Mundo"
```

Ejemplo

El método len () devuelve la longitud de una cadena:

```
a = "Hola Mundo"
print(len(a))
```

Ejemplo

El método lower () devuelve la cadena en minúsculas:

```
a = "Hola Mundo"
print(a.lower())
```

Ejemplo

El método upper () devuelve la cadena en mayúsculas:

```
a = "Hola Mundo"

print(a.upper())
```

Ejemplo

El método replace () reemplaza una cadena con otra cadena:

```
a = "Hola Mundo"

print(a.replace("O", "M"))
```

Ejemplo

El método split () divide la cadena en subcadenas si encuentra instancias del separador:

```
a = "Hola Mundo"

print(a.split(",")) # returns ['Hola', ' Mundo']
```

- Comandos

Python permite la entrada de línea de comandos. Eso significa que podemos pedirle al usuario una entrada.

El siguiente ejemplo solicita el nombre del usuario, luego, al utilizar el método input ()**,** el programa imprime el nombre en la pantalla:
Ejemplo

ejemplo_input.py

```
print("tu palabra")
x = input()
print("Hola" + x)
```

Guarde este archivo como ejemplo_input.py y cárguelo a través de la línea de comando:

```
:\Users\tu nombre>python
```

3.6 Operadores

Los operadores se utilizan para realizar operaciones en variables y valores. Python divide los operadores en los siguientes grupos:

- Operadores aritméticos
- Operadores de Asignación
- Operadores de comparación
- Operadores lógicos
- Operadores de identidad
- Operadores de membresía
- Operadores bitwise
- Operadores de aritmética de Python

Los operadores aritméticos se utilizan con valores numéricos para realizar operaciones matemáticas comunes:

- Operadores de asignación
  Los operadores de asignación se utilizan para asignar valores a las variables:

Operador	Ejemplo	Igual a:
=	x = 2	x = 2
+=	x += 4	x = x + 4
-=	x -= 5	x = x – 5
*=	x *= 2	x = x * 2
/=	x /= 4	x = x / 4
%=	x %= 6	x = x % 6
//=	x //= 7	x = x // 7
**=	x **= 7	x = x ** 7
&=	x &= 7	x = x & 7
|=	x |= 7	x = x | 7
^=	x ^= 9	x = x ^ 9
>>=	x >>= 9	x = x >> 9
<<=	x <<= 9	x = x << 9

- Operadores aritméticos

Operador	Nombre	Ejemplo
+	Suma	x + y
–	Resta	x – y
*	Multiplicación	x * y
/	División	x / y
%	Módulos	x % y
**	Exponencial	x ** y
//	Floor division	x // y

- Operaciones de comparación

Operador	Name	Example
==	Igual	x == y
!=	Distinto	x != y
>	Mayor que	x > y
<	Menor que	x < y
>=	Mayor o igual que	x >= y
<=	Menor o igual que	x <= y

- Operadores lógicos

Operador	Descripción	Ejemplo
and	Devuelve True si ambas afirmaciones son verdaderas	x < 3 and x < 20
or	Devuelve True si una de las afirmaciones es verdadera	x < 3 or x < 5

not	Invertir el resultado, devuelve False si el resultado es verdadero	not(x < 10 and x < 20)

- Operadores de identidad
  Los operadores de identidad se utilizan para comparar los objetos, no si son iguales, pero si en realidad son el mismo objeto, con la misma ubicación de memoria:

Operador	Descripción	Ejemplo
is	Devuelve verdadero si ambas variables son el mismo objeto	x is y
is not	Devuelve verdadero si ambas variables no son el mismo objeto	x is not y

- Operadores de membresía
  Los operadores de membresía se utilizan para probar si una secuencia se presenta en un objeto:

Operador	Descripción	Ejemplo	
in	Devuelve True si una secuencia con el valor especificado está presente en el objeto	x in y	
not in	Devuelve True si una secuencia con el valor especificado no está presente en el objeto	x not in y	

- Operadores de Bitwise
  Los operadores bitwise se utilizan para comparar números (binarios):

Operador	Name	Description
&	AND	Establece cada bit a 1 si ambos bits son 1
|	OR	Establece cada bit a 1 si uno de dos bits es 1
^	XOR	Establece cada bit a 1 si solo uno de dos bits es 1

~	NOT	Invierte todos los bits.
<<	Desplazar cero a la izquierda	Gire a la izquierda presionando ceros desde la derecha y deje que los bits más a la izquierda caigan.
>>	Desplazar cero a la derecha	Gire a la derecha presionando las copias del bit de la izquierda desde la izquierda y deje que los bits de la derecha caigan.

3.7 Listas

Hay cuatro tipos de datos de recopilación en el lenguaje de programación Python:

- list - es una colección que está ordenada y cambiable. Permite duplicar miembros.
- Tuple - es una colección que está ordenada e inmutable. Permite duplicar miembros.
- Set - es una colección que no está ordenada ni indexada. No hay miembros duplicados.
- Dictionary - es una colección que no está ordenada, modificable e indexada. No hay miembros duplicados. Al elegir un tipo de colección, es útil comprender las propiedades de ese tipo. Elegir el tipo correcto para un conjunto de datos en particular podría significar la retención del significado, y podría significar un aumento en la eficiencia o la seguridad.

- Listas
  Una lista es una colección que está ordenada y cambiable. En Python las listas están escritas entre corchetes.
  Ejemplo

  Crear una lista
  ```
  thislist = ["Mexico", "Colombia", "Uruguay"]
  print(thislist)
  ```

- Artículos de acceso
  Puede acceder a los elementos de la lista haciendo referencia al número de índice:
  Ejemplo
  Imprime el segundo elemento de la lista:
  ```
  thislist = ["Mexico", "Colombia", "Uruguay"]
  ```

print(thislist[1])

- Cambiar el valor del elemento
  Para cambiar el valor de un artículo específico, consulte el número de índice:
  Ejemplo
  Cambia el segundo elemento:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
thislist[1] = "Uruguay"
print(thislist)
```

- Recorrer una lista o Loop through a list
  Puede recorrer los elementos de la lista utilizando un bucle for:
  Ejemplo
  Imprima todos los artículos en la lista, uno por uno:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
for x in thislist:
  print(x)
```

- Comprobar si el artículo existe
  Para determinar si un elemento especificado está presente en una lista, use la palabra clave in:
  Ejemplo

```
thislist = ["Mexico", "Colombia", "Uruguay"]
if "Mexico" in thislist:
  print("Sí, 'Mexico' está en la lista")
```

- Longitud de la lista
  Para determinar cuántos elementos tiene una lista, use el método len ():
  Ejemplo
  Imprima el número de artículos en la lista:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
print(len(thislist))
```

- Agregar artículos
  Para agregar un elemento al final de la lista, use el método append ():
  Ejemplo
  Usando el método append () para agregar un elemento:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
thislist.append("Puerto Rico")
print(thislist)
```

Para agregar un elemento en el índice especificado, use el método insert ().
Ejemplo
Insertar un elemento como la segunda posición:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
thislist.insert(1, "Cuba")
print(thislist)
```

- Eliminar artículo

Existen varios métodos para eliminar elementos de una lista:
Ejemplo
El método remove () elimina el elemento especificado:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
thislist.remove("Mexico")
print(thislist)
```

Ejemplo
El método pop () elimina el índice especificado, (o el último elemento si no se especifica el índice):

```
thislist = ["Mexico", "Colombia", "Uruguay"]
thislist.pop()
print(thislist)
```

Ejemplo
La palabra clave “del” elimina el índice especificado:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
del thislist[0]
print(thislist)
```

Ejemplo
La palabra clave del también puede eliminar la lista completamente:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
del thislist
```

Ejemplo
El método clear () vacía la lista:

```
thislist = ["Mexico", "Colombia", "Uruguay"]
thislist.clear()
print(thislist)
```

- List Constructor ()

También es posible usar el constructor list () para hacer una lista.
Ejemplo
Usando el constructor list () para hacer una lista:

thislist = list(("Mexico", "Colombia", "Uruguay"))
print(thislist)

- Métodos de lista

Método	Descripción
append()	Agrega un elemento al final de la lista
clear()	Elimina todos los elementos de la lista.
copy()	Devuelve una copia de la lista.
count()	Devuelve el número de elementos con el valor especificado.
extend()	Agregue los elementos de una lista (o cualquier otro iterable) al final de la lista actual
index()	Devuelve el índice del primer elemento con el valor especificado.
insert()	Agrega un elemento en la posición especificada
pop()	Elimina el elemento en la posición especificada.
remove()	Elimina el elemento con el valor especificado.
reverse()	Invierte el orden de la lista
sort()	Invierte el orden de la lista

3.8 Tuples

Una tuple es una colección que está ordenada e inmutable. En Python, los tuples están escritas con corchetes.

```
thistuple = ("Mexico", "Colombia", "Uruguay")
print(thistuple)
```

Acceder a los elementos de la tupla Puede acceder a los elementos de la tupla consultando el número de índice:

Ejemplo

Devuelve el artículo en la posición 1:

```
thistuple = ("Mexico", "Colombia", "Uruguay")
print(thistuple[1])
```

- Cambiar los valores de una tuple
  Una vez que se crea una tupla, no puede cambiar sus valores. Los tuples son inmutables.

  Ejemplo
  No puedes cambiar valores en una tuple:

  ```
  thistuple = ("Mexico", "Colombia", "Uruguay")
  thistuple[1] = "Argentina"
  #Los valores seguirán siendo los mismos:
  print(thistuple)
  ```

- Bucle a través de una tuple
  Puede recorrer los elementos de la tupla utilizando un bucle for.
  Ejemplo
  Iterar a través de los elementos e imprimir los valores:

  ```
  thistuple = ("Mexico", "Colombia", "Uruguay")
  for x in thistuple:
  print(x)
  ```

  Compruebe si el artículo existe Para determinar si un elemento específico está presente en una tupla, use la palabra clave in:
  Ejemplo
  Compruebe si «apple» está presente en el tuple:

  ```
  thistuple = ("Mexico", "Colombia", "Uruguay")
  if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
  ```

- Longitud de una tuple, len ()

Para determinar cuántos elementos tiene una lista, use el método len ():

Ejemplo

Imprime el número de artículos en la tupla:

```
thistuple = ("Mexico", "Colombia", "Uruguay")
print(len(thistuple))
```

- Agregar artículos

Una vez que se crea una tupla, no puede agregarle elementos. Las tuplas son inmutables.

Ejemplo

No puede agregar elementos a una tupla:

```
thistuple = ("Mexico", "Colombia", "Uruguay")
thistuple[3] = "Argentina" #Esto generará un error.
print(thistuple)
```

Nota: no puede eliminar elementos de una tupla. Las tuplas no se pueden cambiar, por lo que no puede eliminar elementos de ella, pero puede eliminar la tupla por completo:

Ejemplo

La palabra clave del puede eliminar la tupla por completo:

```
thistuple = ("Mexico", "Colombia", "Uruguay")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

- Tuple constructor ()

También es posible usar el constructor tuple () para hacer una tupla.

Ejemplo

Usando el método de la tupla () para hacer una tupla:

```
thistuple = tuple(("Mexico", "Colombia", "Uruguay"))
#Tenga en cuenta los dobles corchetes
print(thistuple)
```

- Métodos Tuples
- Python tiene dos métodos incorporados que puedes usar en las tuplas.

Método	Descripción
count()	Devuelve el número de veces que se produce un valor especificado en una tupla

index()	Busca en la tupla un valor específico y devuelve la posición donde se encontró

3.9 Sets

Un conjunto es una colección que no está ordenada ni indexada. En los conjuntos de Python están escritos con llaves.

```
thisset = {"Mexico", "Venezuela", "Cuba"}
print(thisset)
```

Nota: los conjuntos no están ordenados, por lo que los elementos aparecerán en un orden aleatorio.

- Artículos de acceso
  No puede acceder a los elementos de un conjunto haciendo referencia a un índice, ya que los conjuntos no están ordenados, los elementos no tienen índice. Pero puede recorrer los elementos del conjunto usando un loop for, o preguntar si un valor específico está presente en un conjunto, usando la palabra clave in.
  Ejemplo
  Recorra el conjunto e imprima los valores:

```
thisset = {"Mexico", "Venezuela", "Cuba"}
for x in thisset:
  print(x)
```

- Cambiar artículos
  Una vez que se crea un conjunto, no puede cambiar sus elementos, pero puede agregar nuevos elementos. Agregar articulos Para agregar un elemento a un conjunto use el método add ().
  Para agregar más de un elemento a un conjunto, use el método update ().
  Ejemplo
  Agregue un elemento a un conjunto, usando el método add ():

```
thisset = {"Mexico", "Venezuela", "Cuba"}
 thisset.add("Colombia")
 print(thisset)
```

  Ejemplo
  Agregue varios elementos a un conjunto, usando el método update ():

```
thisset = {"Mexico", "Venezuela", "Cuba"}
```

```
thisset.update(["Colombia", "Puerto Rico", "Argentina"])

print(thisset)
```

- Obtener la longitud de sus caracteres

  Para determinar cuántos elementos tiene un conjunto, use el método len ().

  Ejemplo

  Obtener el número de artículos en un conjunto:

```
thisset = {"Mexico", "Venezuela", "Cuba"}
print(len(thisset))
```

- Borrar el artículo

  Para eliminar un elemento de un conjunto, use el método remove () o el método discard ().

  Ejemplo

  Elimine «Venezuela» utilizando el método remove ():

```
thisset = {"Mexico", "Venezuela", "Cuba"}
thisset.remove("Venezuela")
print(thisset)
```

  Nota: Si el elemento a eliminar no existe, remove () generará un error.

  Ejemplo

  Elimine «Venezuela» usando el método discard ():

```
thisset ={"Mexico", "Venezuela", "Cuba"}
thisset.discard("Venezuela")
print(thisset)
```

  Nota: Si el elemento a eliminar no existe, discard() NO generará un error.

También puede usar el método pop () para eliminar un elemento, pero este método eliminará el último elemento. Recuerde que los conjuntos no están ordenados, por lo que no sabrá qué elemento se elimina.
El valor de retorno del método pop () es el elemento eliminado.

```
thisset = {"Mexico", "Venezuela", "Colombia"}
x = thisset.pop()
print(x)
print(thisset)
```

- Constructor set

  También es posible usar el constructor set () para hacer un conjunto.

  thisset = set(("Mexico", "Venezuela", "Colombia"))
  print(thisset)

3.10 Diccionarios

Diccionarios en Python: Un diccionario es una colección que no está ordenada, que se puede cambiar e indexar. En Python, los diccionarios están escritos con corchetes, y tienen claves y valores.
Ejemplo

Crea e imprime un diccionario:

thisdict = {"vino": "Tinto",
"denominación origen": "Rioja",
"año": 2016 }
print (thisdict)

- Acceder a los artículos
  Puede acceder a los elementos de un diccionario haciendo referencia a su nombre clave:
  Ejemplo
  Obtener el valor de la clave «modelo»:

  x = thisdict ["crianza"]

  También hay un método llamado get () que le dará el mismo resultado:
  Ejemplo
  Obtener el valor de la clave «modelo»:

  x = thisdict.get ("crianza")

- Cambiar valores
  Puede cambiar el valor de un elemento específico refiriéndose a su nombre de clave:
  Ejemplo

  thisdict = { "vino": "Tinto",
  "denominación origen": "Rioja",

"año": 2016 }
thisdict ["año"] = 2019

Cambia el «año» a 2019:

- Bucle a través de un diccionario

Puede recorrer un diccionario utilizando un for loop.
Cuando recorre un diccionario, el valor de retorno son las claves del diccionario, pero también hay métodos para devolver los valores.
Ejemplo
Imprima todos los nombres clave en el diccionario, uno por uno: para x en este punto:

print(x)

Ejemplo

Imprima todos los valores en el diccionario, uno por uno:

for x in thisdict:
print(x)

Ejemplo
También puede usar la función values () para devolver los valores de un diccionario:

for x in thisdict.values():
print(x)

Ejemplo

Recorra ambas claves y valores, usando la función items ():

for x, y in thisdict.items():
print(x, y)

Comprobar si existe clave Para determinar si una clave especificada está presente en un diccionario, use la palabra clave in:

Ejemplo

Compruebe si «año» se encuentra actualmente en el diccionario:

thisdict = {
"vino": "Tinto",
"denominación origen": "Rioja",
"año": 2016
}
if "año" in thisdict:
print("Sí, el año está en el diccionario")

- Longitud del diccionario

  Para determinar cuántos elementos (pares clave-valor) tiene un diccionario, use el método len ().

  Ejemplo

  Imprima el número de artículos en el diccionario:

  ```
  print(len(thisdict))
  ```

- Añadiendo artículos

  La adición de un elemento al diccionario se realiza utilizando una nueva clave de índice y asignándole un valor:

  Ejemplo

  ```
  thisdict = {
  "vino": "Tinto",
  "denominación origen": "Rioja",
  "año": 2016
  } thisdict["tipo"] = "reserva"
  print(thisdict)
  ```

- Eliminando artículos

  Existen varios métodos para eliminar elementos de un diccionario:

  Ejemplo

  El método pop () elimina el elemento con el nombre de clave especificado:

  Ejemplo

  El método pop item () elimina el último elemento insertado (en las versiones anteriores a 3.7, en su lugar se elimina un elemento aleatorio):

  ```
  thisdict = {
  "vino": "Tinto",
  "denominación origen": "Rioja",
  "año": 2016
  }
  thisdict.popitem()
  print(thisdict)
  ```

  Ejemplo

  La palabra clave del elimina el elemento con el nombre de clave especificado:

  ```
  thisdict = {
  "vino": "Tinto",
  "denominación origen": "Rioja",
  "año": 2016
  }
  ```

del thisdict["denominación"]
print(thisdict)

Ejemplo

La palabra clave del también puede eliminar el diccionario completamente:

thisdict = { "vino": "Tinto",
"denominación origen": "Rioja",
"año": 2016
}
del thisdict
print(thisdict) #Dará error ya que no existe "thisdict"

Ejemplo

La palabra clave clear () vacía el diccionario:

thisdict = {
"vino": "Tinto",
"denominación origen": "Rioja",
"año": 2016 } thisdict.clear()
print(thisdict)

- El constructor dict ()

  También es posible usar el constructor dict () para hacer un diccionario:

  Ejemplo

  thisdict = dict(vino = "Tinto", año="2016", denominación origen = Rioja)
  print (thisdict)

- Métodos de diccionario

Método	Descripción
clear()	Elimina todos los elementos del diccionario.
copy()	Devuelve una copia del diccionario.
fromkeys()	Devuelve un diccionario con las claves y valores especificados.
get()	Devuelve el valor de la clave especificada.
items()	Devuelve una lista que contiene una tupla para cada par de valores clave
keys()	Devuelve una lista con las claves del diccionario.

pop()	Elimina el elemento con la clave especificada.
popitem()	Elimina el último par clave-valor insertado
setdefault()	Devuelve el valor de la clave especificada. Si la clave no existe: inserte la clave, con el valor especificado
update()	Actualiza el diccionario con los pares clave-valor especificados.
values()	Devuelve una lista de todos los valores del diccionario.

3.11 If Else

Condiciones Python y declaraciones If

Python soporta las condiciones lógicas usuales de las matemáticas:

Es igual a: a == b
No es igual a: a! = B
Menos que: a <b
Menor o igual a: a <= b
Mayor que: a> b
Mayor o igual que: a> = b
Estas condiciones se pueden usar de varias maneras, más comúnmente en «declaraciones if» y bucles. Se escribe una «sentencia if» usando la palabra clave if.

Ejemplo

Si declaración:

```
a = 50
b = 100
if b > a:
print("b mayor que a")
```

En este ejemplo, utilizamos dos variables, a y b, que se usan como parte de la sentencia if para comprobar si b es mayor que a. Como a es 50, y b es 100, sabemos que 100 es mayor que 50, y por eso imprimimos en la pantalla que «b es mayor que a».

- Sangría
  Python se basa en la sangría, usando espacios en blanco, para definir el alcance en el código. Otros lenguajes de programación a menudo utilizan rizos para este propósito.
  Ejemplo

  Si declaración, sin sangría (generará un error):

  ```
  a = 50
  b = 100
  if b > a:
  print("b es mayor que a") # esto dará error
  ```

- Elif
  La palabra clave elif es una manera de pythons de decir «si las condiciones anteriores no fueran ciertas, entonces intente esta condición».

  Ejemplo

  ```
  a = 50
  b = 50
  if b > a:
  print("b mayor que a")
  elif a == b:
  print("a y b son iguales")
  ```

  En este ejemplo, a es igual a b, por lo que la primera condición no es verdadera, pero la condición elif es verdadera, por lo que imprimimos en la pantalla que «a y b son iguales».

- Else
  La palabra clave else atrapa cualquier cosa que no esté atrapada por las condiciones anteriores.
  Ejemplo

  ```
  a = 100
  b = 50
  if b > a:
  print("b es mayor que a")
  elif a == b:
  print("a y b son igualesl")
  else: print("a es mayor que b")
  ```

  En este ejemplo, a es mayor que b, por lo que la primera condición no es verdadera, también la condición elif no es verdadera, por lo

que pasamos a la otra condición e imprimimos en la pantalla que «a es mayor que b». También puedes tener un else sin el elif:
Ejemplo

```
a = 100
b = 50 if b > a:
print("b es mayor que a")
else:
print("b no es mayor que  a")
```

- Sort Hand If…Else

  Si solo tiene que ejecutar una instrucción, puede colocarla en la misma línea que la instrucción if.

  Ejemplo

  Una línea si declaración:

  ```
  if a > b: print("a es mayor que b")
  ```

-

- Short Hand if
  Si solo tiene que ejecutar una instrucción, puede colocarla en la misma línea que la instrucción if.
  Ejemplo
  Una línea si declaración:

  ```
  if a > b: print("a es mayor que b")
  ```

- And
  La palabra clave y es un operador lógico, y se utiliza para combinar declaraciones condicionales:
  Ejemplo
  Prueba si a es mayor que b, Y si c es mayor que a:

  ```
  if a > b and c > a:
  print("Las dos condiciones son verdaderas")
  ```

- Or

  La palabra clave o es un operador lógico, y se utiliza para combinar declaraciones condicionales:

  Ejemplo

  Pruebe si a es mayor que b, O si a es mayor que c:

```
if a > b or a > c:
print("Al menos una de las condiciones son verdaderas")
```

3.12 Loops

- While Loops

Python tiene dos comandos de bucle primitivos:

- while loops
- for loops

```
i = 3
while i < 7:
print(i)
i += 3
```

Nota: recuerde incrementar i, o de lo contrario el ciclo continuará para siempre.

El while loop requiere que las variables relevantes estén listas, en este ejemplo necesitamos definir una variable de indexación, i, que establecemos en la primera línea.

- La declaración de break

  Con la instrucción break, podemos detener el bucle incluso si la condición while es verdadera:

  Ejemplo

  Salga del bucle cuando i es 3:

```
i = 3
while i < 8:
print(i)
if i == 6:
break
i += 3
```

La declaración de continuar

Con la instrucción continue podemos detener la iteración actual y continuar con la siguiente:
Ejemplo
Continuar con la siguiente iteración si i es 3:

```
i = 2
while i < 8:
 i += 2
 if i == 7:
 continue
 print(i)
```

- Python for Loops

  Un bucle for se usa para iterar sobre una secuencia (es decir, una lista, una tupla, un diccionario, un conjunto o una cadena).
  Esto se parece menos a la palabra clave para en otro lenguaje de programación, y funciona más como un método de iterador que se encuentra en otros lenguajes de programación orientados a objetos.
  Con el bucle podemos ejecutar un conjunto de sentencias, una vez para cada elemento en una lista, tupla, conjunto, etc.
  Ejemplo

  Imprime cada país en una lista de Países:

  ```
  countries = ["México", "Colombia", "Chile"]
  for x in countries:
  print(x)
  ```

  El bucle for no requiere una variable de indexación para establecer de antemano. Buceando a través de una cadena Incluso las cadenas son objetos iterables, contienen una secuencia de caracteres:

  Ejemplo

  Recorre las letras de la palabra «Colombia»:

  ```
  for x in "Colombia":
  print(x)
  ```

- Break Statement

  Con la instrucción break, podemos detener el bucle antes de que haya pasado por todos los elementos:
  Ejemplo
  Salga del bucle cuando x es «Chile»:

```
countries = ["México", "Colombia", "Chile"]
for x in countries:
print(x) if x == "Chile":
break
```

Ejemplo

Salga del bucle cuando x es «Mexico», pero esta vez la ruptura viene antes de la impresión:

```
countries = ["México", "Colombia", "Chile"]
for x in countries:
if x == "México":
break print(x)
print (x)
```

- Continue

  Con la instrucción continue podemos detener la iteración actual del bucle y continuar con la siguiente:

  Ejemplo

  No imprima Colombia:

  ```
  countries = ["México", "Colombia", "Chile"]
  for x in countries:
  if x == "Colombia":
  continue
  print(x)
  ```

- Range

  Para recorrer un conjunto de códigos un número específico de veces, podemos usar la función range (), La función range () devuelve una secuencia de números, comenzando desde 0 por defecto e incrementa en 1 (por defecto), y termina en un número específico.

  Ejemplo

  Usando la función range ():

  ```
  for x in range(6):
  print(x)
  ```

  Tenga en cuenta que el rango (8) no son los valores de 0 a 8, sino los valores de 0 a 4. La función range () por defecto es 0 como valor de inicio, sin embargo, es posible especificar el valor de inicio agregando un parámetro: range (2, 8), lo que significa valores de 2 a 8 (pero sin incluir 8):

  Ejemplo

  Usando el parámetro de inicio:

```
for x in range(2, 8):
print(x)
```

La función range () por defecto incrementa la secuencia en 1, sin embargo, es posible especificar el valor de incremento agregando un tercer parámetro: range (4, 20, 4):

Ejemplo

Incrementa la secuencia con 3 (el valor predeterminado es 1):

```
for x in range(4, 20, 4):
print(x)
```

- Lo demás en For Loop

  La palabra clave else en un bucle for especifica un bloque de código que se ejecutará cuando finalice el bucle:

  Ejemplo

  Imprima todos los números del 0 al 10 e imprima un mensaje cuando el ciclo haya terminado:

```
for x in range(4):
print(x)
else:
print("Terminado")
```

- Bucles anidados

  Un bucle anidado es un bucle dentro de un bucle. El «bucle interno» se ejecutará una vez para cada iteración del «bucle externo»:

  Ejemplo

  Imprime cada ciudad para cada país:

```
ciudad = ["Madrid", "Londres", "Nueva York"]
pais = [“España", "Inglaterra", "EEUU"]
for x in ciudad:
for y in país:
print(x, y)
```

3.13 Functions

Una función es un bloque de código que solo se ejecuta cuando se llama. Puede pasar datos, conocidos como parámetros, a una función. Una función puede devolver datos como resultado.

En Python, una función se define usando la palabra clave def:

Ejemplo

```
def my_function():
print("Mi primera función")
```

Llamando a una función Para llamar a una función, use el nombre de la función seguido de paréntesis:

Ejemplo

```
def my_function():
  print("Hello from a function")
 my_function()
```

- Parámetros
  La información se puede pasar a funciones como parámetro. Los parámetros se especifican después del nombre de la función, dentro de los paréntesis.
  Puede agregar tantos parámetros como desee, solo sepárelos con una coma.
  El siguiente ejemplo tiene una función con un parámetro (fname). Cuando se llama a la función, pasamos un nombre que se utiliza dentro de la función para imprimir el nombre completo:
  Ejemplo

```
  def my_function(fname):
 print(fname + " Python")
my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

- Valor de parámetro predeterminado
  El siguiente ejemplo muestra cómo usar un valor de parámetro predeterminado. Si llamamos a la función sin parámetro, utiliza el valor predeterminado:

Ejemplo

```
def my_function(nombre = "Manuel"):
print("Me llamo " + nombre)
my_function("Manuel")
my_function("Agustín")
my_function() my_function("Javier")
```

- Valores de retorno

Para permitir que una función devuelva un valor, use la instrucción return:

Ejemplo

```
def my_function(x):
return 10 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

- Recursión

Python también acepta la recursión de funciones, lo que significa que una función definida puede llamarse a sí misma. La recursión es un concepto matemático y de programación común. Significa que una función se llama a sí misma.

Esto tiene el beneficio de significar que puede recorrer los datos para alcanzar un resultado. El desarrollador debe tener mucho cuidado con la recursión, ya que puede ser bastante fácil escribir una función que nunca termina, o una que utiliza cantidades excesivas de memoria o potencia del procesador.

Sin embargo, cuando se escribe correctamente, la recursión puede ser un enfoque muy eficiente y matemáticamente elegante para la programación. En este ejemplo, tri_recursion () es una función que hemos definido para llamar a sí misma («recurse»).

Usamos la variable k como los datos, los cuales disminuyen (-1) cada vez que repetimos. La recursión finaliza cuando la condición no es mayor que 0 (es decir, cuando es 0). Para un desarrollador nuevo, puede llevarle tiempo descubrir cómo funciona exactamente esto, la mejor manera de averiguarlo es probándolo y modificándolo.

Ejemplo

```
def tri_recursion(k):
if(k>0):
result = k+tri_recursion(k-1)
print(result)
```

else: result = 0
return result
print("\D:\Ejercicio Recursion ejemplo\")
tri_recursion(8)

3.14 Lambda

Una función lambda es una pequeña función anónima. Una función lambda puede tomar cualquier número de argumentos, pero solo puede tener una expresión.

- Sintaxis
  Argumentos lambda: expresión
  La expresión se ejecuta y el resultado se devuelve:
  Ejemplo
  Una función lambda que agrega 20 al número pasado como argumento e imprime el resultado:

  x = lambda a : a + 20
  print(x(10))

  Las funciones Lambda pueden tomar cualquier número de argumentos:
  Ejemplo
  Una función lambda que multiplica el argumento a con el argumento b e imprime el resultado:

  *x = lambda a, b : a * b*
  print(x(2, 4))

  Ejemplo
  Una función lambda que suma los argumentos a, b y c e imprime el resultado:
  x = lambda a, b, c : a + b + c print(x(2, 4, 8))

- ¿Por qué usar funciones lambda en python?
  El poder de lambda se muestra mejor cuando se usan como una función anónima dentro de otra función. Digamos que tienes una definición de función que toma un argumento, y ese argumento se multiplicará con un número desconocido:
  def myfunc(n):
  *return lambda a : a * n*
  Use esa definición de función para hacer una función que siempre duplique el número que envía:

Ejemplo

```
def myfunc(n):
return lambda a : a * n
mydoubler = myfunc(2)
print(mydoubler(8))
```

O, use la misma definición de función para hacer una función que siempre triplique el número que envía:

Ejemplo

```
def myfunc(n):
return lambda a : a * n
mytripler = myfunc(3)
print(mytripler(8))
```

O, use la misma definición de función para hacer ambas funciones, en el mismo programa:

Ejemplo

```
def myfunc(n):
return lambda a : a * n
mydoubler = myfunc(2)
mytripler = myfunc(3)
print(mydoubler(8))
print(mytripler(8))
```

Use las funciones lambda cuando se requiera una función anónima por un corto período de tiempo.