Bootcamp Inteligencia Artificial
Nivel Explorador

Master Class 1: Python Para Inteligencia Artificial

Principios Python

Agenda

1. Variables y Asignaciones
2. Aritmética y Cadenas
3. Sentencias de control

1.1¿Qué puede hacer Python?

- Python puede utilizarse en un servidor para crear aplicaciones web.
- Python se puede utilizar junto con el software para crear flujos de trabajo.
- Python puede conectarse a sistemas de bases de datos. También puede leer y modificar archivos.
- Python puede utilizarse para manejar big data y realizar matemáticas complejas.
- Python puede utilizarse para la creación rápida de prototipos o para el desarrollo de software listo para la producción.

1.2¿Por qué Python?

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

1.3 Es bueno saber

- La versión principal más reciente de Python es Python 3. Sin embargo, Python 2, a pesar de que no se actualiza más que las actualizaciones de seguridad, sigue siendo bastante popular.
- En este curso se escribirá Python en una plataforma on line (Colab de Google).
- Es posible escribir Python en un Entorno de Desarrollo Integrado, como Thonny, Pycharm, Netbeans o Eclipse, que son particularmente útiles cuando se manejan grandes colecciones de archivos Python.

1.4 Sintaxis

- Python fue diseñado para ser legible, y tiene algunas similitudes con el idioma inglés con influencia de las matemáticas.
- Python utiliza nuevas líneas para completar un comando, a diferencia de otros lenguajes de programación que suelen utilizar punto y coma o paréntesis.
- Python se basa en la indentación, utilizando espacios en blanco, para definir el ámbito de aplicación, como el de los bucles, las funciones y las clases. Otros lenguajes de programación suelen utilizar corchetes para este fin.

```
print("Hello, World!")
```

1.5 Entornos de desarrollo en línea

Un entorno de desarrollo para Python es un ambiente que ofrece herramientas y funcionalidades para facilitar la escritura, depuración y ejecución de código Python. Estos entornos suelen incluir un editor de texto o código fuente, un intérprete de Python, herramientas de depuración, gestión de archivos y a menudo integración con sistemas de control de versiones.

1.6 Entornos de desarrollo en línea

Hay varios entornos de desarrollo en línea para Python que permiten a los usuarios escribir, ejecutar y compartir código sin la necesidad de instalar software en su computadora. Algunos de ellos son:

- **Google Colab**: es una plataforma gratuita basada en Jupyter Notebooks que se ejecuta en la nube de Google. Permite escribir y ejecutar código Python, además de integrarse con servicios de almacenamiento de Google. (https://colab.research.google.com/)
- **Deepnote**: es un entorno de desarrollo en línea que se centra en la colaboración y la ciencia de datos. Proporciona un entorno de cuaderno similar a Jupyter que se ejecuta en la nube. Está diseñado para ser intuitivo y permite a los usuarios escribir código en Python (u otros lenguajes) en celdas individuales, visualizar datos, crear gráficos y compartir proyectos fácilmente. (https://deepnote.com/)
- **Repl.it**: ofrece un entorno en línea para escribir y ejecutar código en Python y otros lenguajes. Proporciona colaboración en tiempo real, integración con GitHub y permite ejecutar proyectos directamente desde el navegador. (https://replit.com/)

1.7 Aspectos básicos de un notebook en Colab

La acción anterior insertará una celda en la cual podrás escribir y ejecutar código, de la siguiente manera:

1.8 Indentación en Python

- La sangría se refiere a los espacios al principio de una línea de código.
- Mientras que en otros lenguajes de programación la sangría en el código es sólo para la legibilidad, la sangría en Python es muy importante.
- Python utiliza la sangría para indicar un bloque de código.

```
if 5 > 2:
  print("Cinco es mayor a 2!")
```

- Python te dará un error si te saltas la sangría:

```
if 5 > 2:
print("Cinco es mayor a 2!")

  File "<ipython-input-5-1cf66c84ec30>", line 2
    print("Cinco es mayor a 2!")
        ^
IndentationError: expected an indented block
```

1.9 Variables en Python

Una variable puede tener un nombre corto (como x e y) o un nombre más descriptivo (edad, nombre del coche, volumen_total).

Reglas para las variables de Python:

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

1.10 Variables en Python

Muchos valores a múltiples variables

Python permite asignar valores a múltiples variables en una sola línea:

```
x, y, z = "Azul", "Rojo", "Verde"
print(x)
print(y)
print(z)
```

```
Azul
Rojo
Verde
```

1.11 Variables en Python

Salida de variables

La función print() de Python se utiliza a menudo para dar salida a las variables.

```
x = "Python es un lenguaje"
print(x)
```

```
Python es un lenguaje
```

```
x = "Python"
y = "es un"
z = "lenguaje"
print(x, y, z)
```

```
Python es un lenguaje
```

```
x = "Python "
y = "es un"
z = " lenguaje"
print(x+y+z)
```

```
Python es un lenguaje
```

1.12 Tipos de datos en Python

- En programación, el tipo de datos es un concepto importante.
- Las variables pueden almacenar datos de diferentes tipos, y diferentes tipos pueden hacer diferentes cosas.
- Python tiene los siguientes tipos de datos incorporados por defecto, en estas categorías:

Tipos de texto	`str`
Tipos numéricos	`int`, `float`, `complex`
Tipos de secuencia	`list`, `tuple`, `range`
Tipo de mapeo	`dict`
Tipos de conjuntos	`set`, `frozenset`
Tipo booleano	`bool`
Tipos binarios	`bytes`, `bytearray`, `memoryview`

1.13 Tipos de datos en Python

A continuación, se describen alguos tipos de datos que se manejan en Python:

```
#Tipos de datos
a=1
b=1.2
c='Hola'
d=True
e=[1,2,3,4,5]
```

```
print(a, type(a))
print(b, type(b))
print(c, type(c))
print(d, type(d))
print(e, type(e))
```

```
1 <class 'int'>
1.2 <class 'float'>
Hola <class 'str'>
True <class 'bool'>
[1, 2, 3, 4, 5] <class 'list'>
```

1.14 Números en Python

A continuación, se describen alguos tipos de datos que se manejan en Python:

```
x = 1     # int
y = 2.8   # float
z = 1j    # complex
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

```
x = 35e3
y = 12E4
z = -87.7e100

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

```
<class 'float'>
<class 'float'>
<class 'float'>
```

```
x = 3+5j
y = 5j
z = -5j
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

1.15 Números en Python

Conversión de tipo

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
#convertir a float:
a = float(x)
#convertir de float a int:
b = int(y)
#convertir de int a complejo:
c = complex(x)
```

```
print(a)
print(b)
print(c)
print(type(a))
print(type(b))
print(type(c))
```

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

1.16 Booleanos en Python

- En programación a menudo necesitas saber si una expresión es Verdadera o Falsa.
- Puedes evaluar cualquier expresión en Python, y obtener una de las dos respuestas, Verdadero o Falso.
- Cuando comparas dos valores, la expresión se evalúa y Python devuelve la respuesta booleana:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

```
print(bool("Hola"))
print(bool(15))
```

```
True
True
```

```
x = "Hello"
y = 15

print(bool(x))
print(bool(y))
```

```
True
True
```

1.17 Características lenguajes de programación

- Vocabulario o léxico: Es el conjunto de palabras válidas o reservadas en un lenguaje. Por ejemplo, *float, int, else, char, while*, cada una de ellas tiene un significado predeterminado en el lenguaje, es decir, son palabras reservadas.

- Gramática: es el conjunto de lineamientos que se deben seguir para construir las instrucciones.

1.18 Palabras reservadas

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

2.1 Operadores en Python

Los operadores se utilizan para realizar operaciones con variables y valores.
En el siguiente ejemplo, utilizamos el operador + para sumar dos valores:

```
print(10 + 5)
```

```
15
```

2.2 Operadores en Python

Python divide los operadores en los siguientes grupos:

- Operadores aritméticos
- Operadores de asignación
- Operadores de comparación
- Operadores lógicos
- Operadores de identidad
- Operadores de pertenencia
- Operadores a nivel de bits

2.3 Operadores aritméticos en Python

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

2.4 Operadores aritméticos en Python

Ejemplos

```
x = 5
y = 3
print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
print(x ** y)
print(x // y)
```

```
8
2
15
1.6666666666666667
2
125
1
```

2.5 Operadores de asignación en Python

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

2.6 Operadores de asignación en Python

```
x = 5
print(x)
```

```
5
```

```
x += 3
print(x)
```

```
8
```

```
x -= 3
print(x)
```

```
5
```

```
x *= 3
print(x)
```

```
15
```

```
x /= 3
print(x)
```

```
5.0
```

```
x %= 3
print(x)
```

```
2.0
```

```
x //= 3
print(x)
```

```
0.0
```

```
x //= 3
print(x)
```

```
0.0
```

```
x = 5
x &= 3
print(x)
```

```
1
```

2.7 Operadores de asignación en Python

```
x = 5
x |= 3
print(x)
```

```
7
```

```
x = 5
x ^= 3
print(x)
```

```
6
```

```
x = 5
x >>= 3
print(x)
```

```
0
```

```
x = 5
x <<= 3
print(x)
```

```
40
```

2.8 Operadores de comparación en Python

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

2.9 Operadores de comparación en Python

```
x = 5
y = 3
print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

```
False
True
True
False
True
False
```

1.10 Operadores lógicos en Python

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

2.11 Operadores lógicos en Python

```
x = 5
print(x > 3 and x < 10)
```

```
True
```

```
x = 5
print(x > 3 or x < 4)
```

```
True
```

```
x = 5
print(not(x > 3 and x < 10))
```

```
False
```

2.12 Operadores de identidad en Python

Los operadores de identidad se utilizan para comparar los objetos, no si son iguales, sino si son realmente el mismo objeto, con la misma ubicación de memoria.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

2.13 Operadores de identidad en Python

```
x = ["manzana", "limon"]
y = ["manzana", "limon"]
z = x
print(x is z)
print(x is y)
print(x == y)
```

```
True
False
True
```

2.14 Operadores de pertenencia en Python

Los operadores de pertenencia se utilizan para comprobar si una secuencia se presenta en un objeto:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

2.15 Matemáticas de Python

Python cuenta con un conjunto de funciones matemáticas incorporadas, incluyendo un extenso módulo matemático, que permite realizar tareas matemáticas con números.

Las funciones min() y max() se pueden utilizar para encontrar el valor más bajo o más alto de un iterable

```
x = min(5, 10, 25)
y = max(5, 10, 25)
print(x)
print(y)
```

```
5
25
```

2.16 Matemáticas de Python

La función abs() devuelve el valor absoluto (positivo) del número especificado:

```
x = abs(-7.25)
print(x)
```

```
7.25
```

La función pow(x, y) devuelve el valor de x a la potencia de y

```
x = pow(4, 3)
print(x)
```

```
64
```

2.17 El módulo Math

Python tiene también un módulo incorporado llamado math, que amplía la lista de funciones matemáticas.
Para utilizarlo, debes importar el módulo math.
Cuando hayas importado el módulo de matemáticas, puedes empezar a utilizar los métodos y constantes del módulo.

El método math.sqrt(), por ejemplo, devuelve la raíz cuadrada de un número

```
import math
x = math.sqrt(64)
print(x)
```

```
8.0
```

2.18 El módulo Math

El método math.ceil() redondea un número hacia arriba a su número entero más cercano, y el método math.floor() redondea un número hacia abajo a su número entero más cercano, y devuelve el resultado.

```
import math
x = math.ceil(1.4)
y = math.floor(1.4)
print(x)
print(y)
```

```
2
1
```

2.19 El módulo Math

La constante math.pi, devuelve el valor de PI (3,14...)

```
import math
x = math.pi
print(x)
```

```
3.141592653589793
```

Para ver los metodos del módulo math, se puede dirigir al siguiente enlace:
Enalce

2.20 Cadenas

- Las cadenas (o strings) son un tipo de datos compuestos por secuencias de caracteres que representan texto. Estas cadenas de texto son de tipo str y se delimitan mediante el uso de comillas simples o dobles.

- En el caso que queramos usar comillas (o un apóstrofo) dentro de una cadena tenemos distintas opciones. La más simple es encerrar nuestra cadena mediante un tipo de comillas (simples o dobles) y usar el otro tipo dentro de la cadena. Otra opción es usar en todo momento el mismo tipo de comillas, pero usando la barra invertida (\) como carácter de escape en las comillas del interior de la cadena para indicar que esos caracteres forman parte de la cadena.

2.21 Operaciones con String

- **Comparación**:

- Se usan los operadores convencionales (<, <=, >, >=, ==, !=) para comparar cadenas usando el orden lexicográfico.
- En el orden lexicográfico, se comparan de izquierda a derecha uno a uno los caracteres, mientras sean iguales.
- En el caso que no sean iguales, si el carácter de la primera cadena es menor que el de la segunda a la primera cadena se le considera menor, pero si es mayor, a la primera cadena se le considera mayor.
- Si todos los caracteres son iguales, se considera que las cadenas son iguales

2.22 Métodos de String

- Longitud (len): longitud de una cadena
- Subcadena (slice): obtiene una cadena de otra cadena
- Contando (count): cuenta cuantos caracteres hay en una cadena
- Buscando (find, rfind): obtiene la primera y última ocurrencia en una cadena
- Mayúscula y minúsculas: s.lower() s.upper()
- Removiendo caracteres (strip, lstrip, rstrip)
- Dividiendo cadenas (split)
- Remplazando (replace)

3.1 Sentencias de control

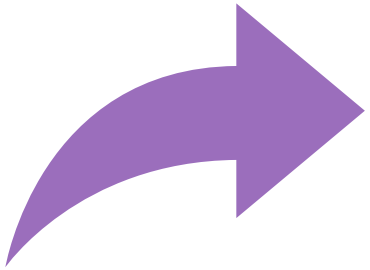
La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. Una estructura secuencial se representa de la siguiente forma:

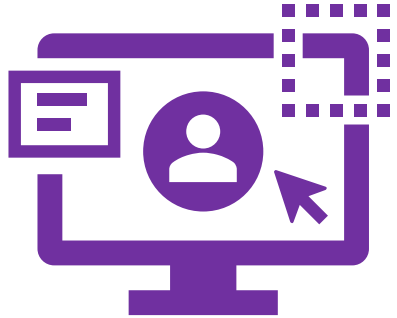
3.2 Sentencias de control

Inicio

Accion1
Accion2
.
.
.
AccionN

Fin





3.3 Estructura Condicional. Simple

Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que con base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen dos tipos básicos, las simples y las múltiples.

Las estructuras condicionales simples se les conoce como, “Tomas de decisión”.

Estas tomas de decisión tienen la siguiente forma:

Si <condición> entonces
Acción(es)
Fin-si

3.4 Estructura Condicional. Simple. Ejemplo

Un hombre desea saber cuánto dinero se genera por concepto de intereses sobre la cantidad que tiene en inversión en el banco. El decidirá reinvertir los intereses siempre y cuando estos excedan a $7000, y en ese caso desea saber cuánto dinero tendrá finalmente en su cuenta.

```
Inicio
        Leer p_int, cap
         int = cap * p_int
         si (int > 7000 ) entonces
                  capf = cap + int
          fin-si
          Imprimir capf
Fin
```

3.5 Estructura Condicional. Doble

Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:

```
Si <condición> entonces
        Acción(es)
Si no
        Acción(es)
Fin si
```

3.6 Estructura Condicional. Doble. Ejemplo

Determinar si un alumno aprueba o reprueba un curso, sabiendo que aprobara si su promedio de tres calificaciones es mayor o igual a 70; reprueba en caso contrario.

```
Inicio
    Leer calif1, calif2, calif3
    prom = (calif1 + calif2 + calif3)/3
    Si( prom >= 70 )entonces
            Imprimir “alumno aprobado”
     Si no
            Imprimir “alumno reprobado”
    Fin-si
Fin
```

3.7 Estructura Condicional. Anidado

Las estructuras condicionales anidadas se dan, cuando si el bloque de código verdadero o el bloque de código falso, contiene otra sentencia condicional.

```
Si <condición> entonces
        Acción(es)
Si no
        Si <condición> entonces
            Acción(es)
        Si no
                Si <condición> entonces
                        Acción(es)
                Si no
                        Acción(es)
                Fin si
        Fin si
Fin si
```

3.8 Estructura Condicional. Anidado. Ejemplo

Leer 2 números; si son iguales que los multiplique, si el primero es mayor que el segundo que los reste y si no que los sume.

```
Inicio
        Leer num1, num2
        Si (num1 = num2) entonces
                resul = num1 * num2
        Si no
                Si (num1 > num2) entonces
                        resul = num1 - num2
                Si no
                        resul = num1 + num2
                fin-si
        fin-si
Fin
```

3.9 Estructura de repetición. While

Un bucle while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True).

```
while <condición>:
    cuerpo del bucle
```

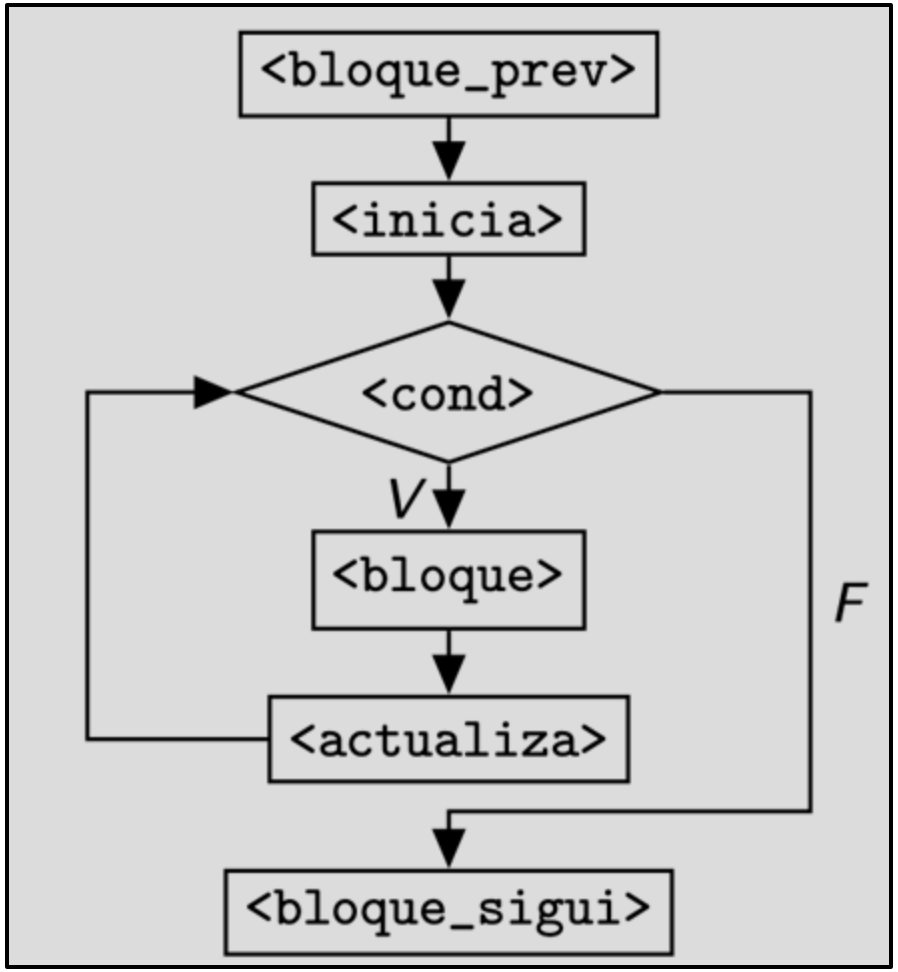
3.10 Estructura de repetición. While

- El ciclo mientras (while) permite ejecutar un bloque de instrucciones mientras que una expresión booleana dada se cumpla, es decir, mientras su evaluación dé como resultado verdadero.

- La expresión booleana se denomina condición de parada y siempre se evalúa antes de ejecutar el bloque de instrucciones; tras esto se pueden presentar dos casos:

    - Si la condición no se cumple, el bloque no se ejecuta.
    - Si la condición se cumple, el bloque se ejecuta, después de lo cual la instrucción vuelve a empezar, es decir, la condición se vuelve a evaluar.

3.11 Estructura de repetición. While



3.12 Estructura de repetición. For

Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

for <elem> **in** <iterable>:
    cuerpo del bucle

3.13 Estructura de repetición. For

- Esta estructura de control tiene dos propósitos primordiales que no siempre son soportados por todo lenguaje de programación:

    1. Como una forma compacta de escribir un ciclo mientras (while).
    2. Para iterar sobre los elementos de una colección de elementos.

- Esta estructura es usualmente utilizada cuando se conocen los valores inicial y final de la variable que es utilizada en la condición de parada.

3.14 Estructura de repetición. For

- Un ciclo para (for) puede ser usado para obtener uno a uno los elementos de una colección de elementos y poder realizar con cada uno de ellos el mismo bloque de operaciones.
- Un esquema textual que en Python representa dicho ciclo (for) es el que se da en el siguiente fragmento de código.

```
<bloque_prev>
for <elemento> in <coleccion>:
    <bloque>
<bloque_sigui>
```

3.15 Estructura de repetición. For

- El fragmento <bloque_prev> es el bloque de instrucciones previas que han sido ejecutadas antes del ciclo.
- El fragmento <elemento> es la variable que se usa para ir recorriendo (iterando) sobre los elementos de la colección.
- El fragmento <coleccion> es la colección de elementos que ser´a recorrida (iterada) con el ciclo.
- El fragmento <bloque> es el bloque de instrucciones principal del ciclo que se ejecuta con cada uno de los elementos de la colección.
- El fragmento <bloque_sigui> es el bloque de instrucciones que se ejecutan después de terminar de ejecutar el ciclo.

3.16 Colección Range.

- Existen varias colecciones que se pueden iterar en Python, una de ellas es la colección (range).
- Una colección (range) es una colección de números en un intervalo, definido por valor inicial, un valor final y un valor de incremento/decremento usado a partir del valor inicial para determinar que valores quedan en el rango.
- Si no se da el valor de inicio, éste se fija en cero (0) y si no se da valor de incremento/decremento, este se fija en uno (1).

3.17 Colección Range.

Los rangos se combinan perfectamente con la instruccion (for):

- Se define una variable que se utilizará para recorrer cada número en el rango.

```
for i in range(-5, 6, 1):
    print(i)

for I in range(10,0,-1):
    print(i)

for i in range(0, 55, 5):
  print(i)
```

Preguntas