Fundamentos de programación en Python © EDICIONES ROBLE, S.L.

campusformacion.imf.com © EDICIONES ROBLE, S.L.
IVAN GARCIA GARCIA
campusformacion.imf.com © EDICIONES ROBLE, S.L.
IVAN GARCIA GARCIA

Índice

I. Introducción

Un elemento básico para cualquier científico de datos es el conocimiento de algún lenguaje de programación que le permita realizar programas con los que llevar a cabo el procesamiento de la información. Aunque sería posible utilizar cualquier lenguaje de propósito general, hay algunos que incluyen la funcionalidad específica para realizar las operaciones más comunes en este ámbito. En este sentido, existen varias alternativas.

Actualmente, hay dos lenguajes de programación que cubren las necesidades de procesamiento requeridas para realizar un proyecto de análisis de datos, que son el lenguaje R y el lenguaje Python.

En esta unidad, se va a estudiar el lenguaje Python. Se trata de un lenguaje de propósito general al que se le ha proporcionado, en forma de librerías, toda la funcionalidad necesaria para llevar a cabo análisis de datos de cualquier complejidad. Presenta funciones y estructura de datos semejantes al lenguaje R. Asimismo, este lenguaje se ha popularizado por varios motivos, entre los que destacan su rápido aprendizaje y su uso intensivo en la industria para este tipo de tareas.

En la presente unidad, se estudiará, en primer lugar, uno de los entornos de desarrollo más utilizados, denominado Jupyter Notebook, y, a continuación, se revisarán los principales elementos del lenguaje de programación, tales como los tipos de datos básicos, estructuras de control, estructuras de datos, funciones y gestión de archivos.

II. Objetivos

Los objetivos que los alumnos alcanzarán tras el estudio de esta unidad son:

- Conocer el entorno JupyterNotebook para el desarrollo de programas en Python.
- Conocer los elementos básicos del lenguaje Python.
- Conocer las estructuras de control y las principales estructuras de datos del lenguaje Python.
- Saber realizar programas de complejidad media y simple con el lenguaje Python.
- Entender lo que hace una parte de código Python de complejidad media y simple.

III. El lenguaje Python y el entorno Jupyter Notebook

3.1. El lenguaje de programación Python

Python es un lenguaje de programación de alto nivel creado por Guido van Rossum. Se desarrolla como un proyecto de código libre, de manera que existe una comunidad de desarrolladores que mantienen el lenguaje, gestionan las versiones del mismo y crean librerías para aumentar su funcionalidad.

Algunas de sus características son:

- **Es un lenguaje interpretado**, por lo que no se debe compilar el código antes de su ejecución.
- **Es multiparadigma**. En este sentido, permite varios estilos de programación: imperativo, orientado a objetos y funcional.
- **Es multiplataforma**. Python es un lenguaje disponible en los principales Sistemas Operativos (Windows, Linux y Mac).
- **Posee un tipado dinámico**. El tipo de los datos es inferido en tiempo de ejecución, de manera que no es necesario declarar el tipo de sus variables y permite conversiones dinámicas de los tipos de los datos.

En comparación con otros lenguajes de programación, Python es un lenguaje simple, fácil de leer y escribir y simple de depurar. Por estas razones es fácil de aprender, de manera que la curva de aprendizaje es corta.

Asimismo, Python cuenta con una gran cantidad de librerías, tipos de datos y funciones incorporadas en el propio lenguaje, lo que le dota de una gran capacidad de procesamiento. En particular, es ampliamente utilizado en el ámbito del análisis de datos y, en general, en tareas de procesamiento de ciencias e ingeniería.

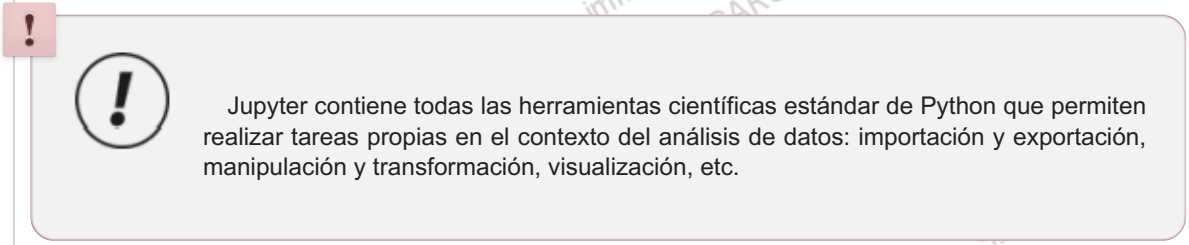
Desde el punto de vista del análisis de datos, dispone de una amplia variedad de librerías y herramientas, tales como Numpy, Pandas, Matplotlib, Scipy, Scikit-learn, Theano, TensorFlow, etc

3.2. El entorno Jupyter notebook

Entornos de desarrollo

En cualquier lenguaje de programación, las herramientas de edición constituyen un elemento esencial. En general, este tipo de herramientas implementan servicios tales como el autocompletado de palabras del lenguaje programación, ayuda interactiva, coloreado de las estructuras sintácticas del lenguaje, depuración de errores, la compilación o interpretación y la de ejecución del programa.

Se trata de un entorno de edición y ejecución visual que permite integrar trozos de códigos con contenidos multimedia o textuales que ayudan a documentar y facilitar la comprensión de los programas realizados. Los documentos generados se visualizan con un navegador (Explorer, Chrome, Firefox...) y pueden incluir cualquier elemento accesible a una página web, además de permitir la ejecución de código escrito en el lenguaje de programación Python.



Jupyter contiene todas las herramientas científicas estándar de Python que permiten realizar tareas propias en el contexto del análisis de datos: importación y exportación, manipulación y transformación, visualización, etc.

Descarga

Para instalarlo, lo mejor es el entorno Anaconda, el cual incluye un conjunto de herramientas de desarrollo para Python, entre las que se encuentra el Jupyter Notebook. Anaconda se puede descargar desde la página web de Anaconda.[1] En la zona de descargas, aparecen dos versiones. Se debe bajar la versión 3.x (figura 2.1.).

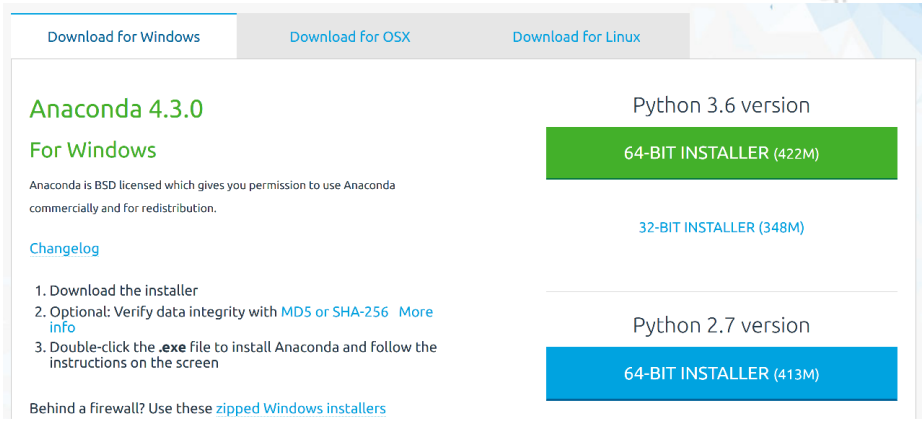


Figura 2.1. Zona de descargas de Anaconda.

[1] Página web de Anaconda. [En línea] URL disponible en http://continuum.io/downloads

Instalación

Una vez descargado el software, para su instalación, se pueden seguir las instrucciones de para cada sistema operativo que aparecen en la misma página de descargas. [2]

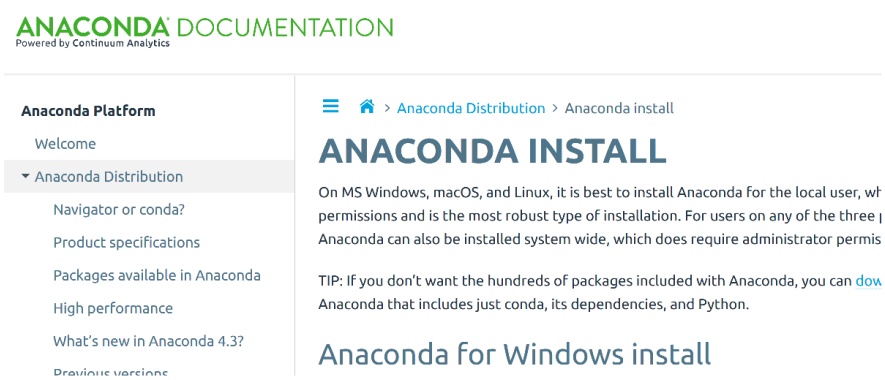


Figura 2.2. Instrucciones de instalación de Anaconda.

[2] Página web de Anaconda. Installation. [En línea] URL disponible en https://docs.continuum.io/anaconda/install

Ejecución

Para ejecutar Jupyter, es preciso buscar en la instalación realizada el icono de "Jupyter Notebook" y pulsar sobre el mismo (figura 2.3.).

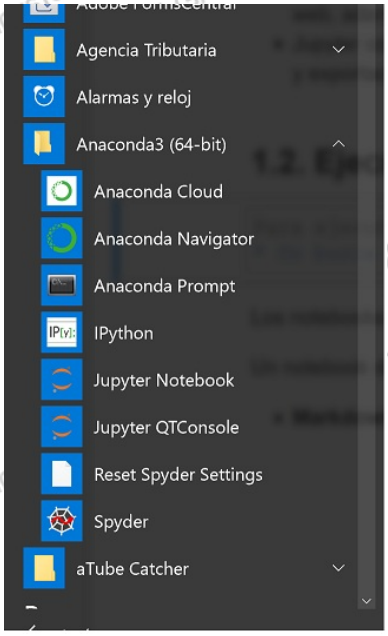


Figura 2.3. Ejecución del Jupyter Notebook.

Nueva terminal

Aparece una nueva terminal donde se ejecuta la herramienta (figura 2.4.). Este terminal no debe cerrarse mientras se esté trabajando con el Jupyter Notebook.

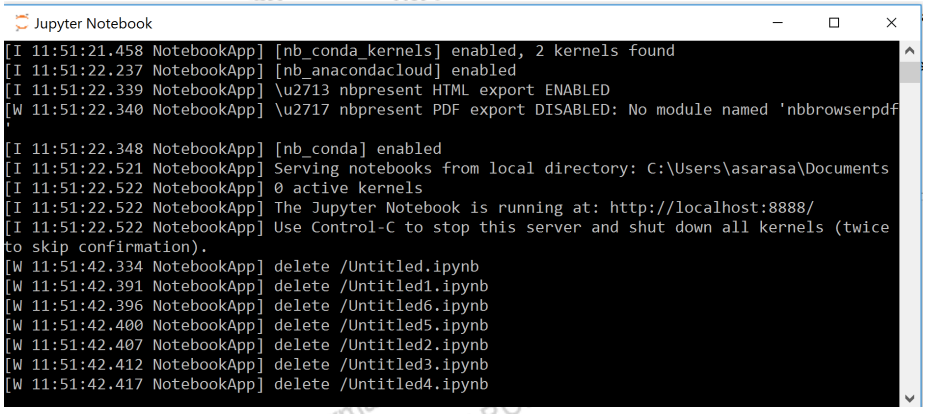


Figura 2.4. Terminal de ejecución de Jupyter Notebook.

Interface en el navegador

A la vez, se lanza un navegador donde se puede acceder a la interface principal del Jupyter Notebook (figura 2.5.).

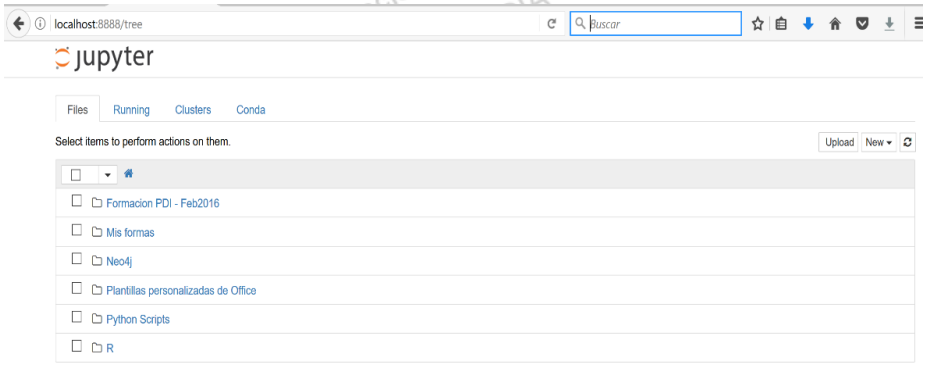


Figura 2.5. Interface principal del Jupyter Notebook.

Esta interface actúa como un navegador de archivos y es posible moverse entre distintas carpetas —basta pinchar sobre la carpeta correspondiente y se accede al contenido de dicha carpeta—, es posible crear nuevos ficheros y carpetas. Para ello se pulsa sobre el desplegable que aparece en la parte derecha, denominado **New,** y allí se selecciona **Text File** o **Folder,** dependiendo de lo que se quiera crear. También es posible renombrar y eliminar las carpetas y archivos. Para ello basta seleccionar la correspondiente carpeta o archivo. Una vez seleccionado, en la parte superior aparecen las opciones de **Rename** y el icono de la papelera, que permiten renombrar y eliminar respectivamente.

Creación de un notebook

Los notebooks de Jupyter son unos archivos con extensión .ipyb, en los que se puede escribir código python ejecutable, texto, dibujar gráficas y otras operaciones más. Para crearse un nuevo notebook basta con pulsar sobre el desplegable que aparece en la parte superior derecha, denominado **New,** y seleccionar en el mismo cualquiera de las opciones etiquetadas como Python [conda root] o Python [default] que se muestran debajo de Notebooks (figura 2.6.).

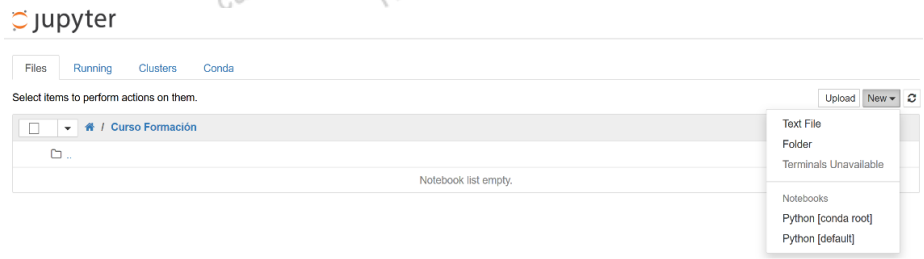


Figura 2.6. Creación de un notebook del Jupyter Notebook.

Notebook del Jupyter

Una vez pulsado, se carga el nuevo notebook creado (figura 2.7.).

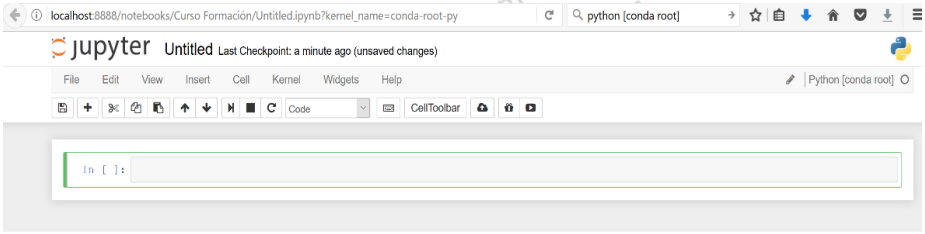


Figura 2.7. Notebook del Jupyter.

Elementos esenciales

Los elementos esenciales de un notebook son (figura 2.8.):

- **Título del notebook.** Cada notebook tiene un título asociado. Por defecto, aparece con el nombre Untitled. Para modificarlo, basta pulsar sobre Untitled y aparece una ventana en la que se puede modificar el nombre.
- **Menús y Barra de herramientas.** En la parte superior de la interface del notebook, aparece un conjunto de menús con diferentes opciones para gestionar los archivos, para editar, visionar, etc.
- **Iconos de acceso rápido.** Iconos que permiten realizar las acciones más comunes sobre los elementos del notebook.
- **Celdas.** La unidad de edición de un notebook son las celdas. Cada celda contiene código Python o la información que documenta dicho código. En este sentido, un notebook es una secuencia de celdas. Las celdas pueden ser de diferentes tipos según el contenido que almacenan:
  - **Markdown**: permite escribir texto formateado con el objetivo de documentar. Se usa el lenguaje de marcas Markdown.
  - **Raw NBConvert**: son celdas que permiten escribir fragmentos de código sin que sean ejecutados.
  - **Heading**: permite embeber código html.
  - **Code**: sirven para escribir código Python ejecutable. Están marcadas por la palabra **In [n]** y están numeradas. El resultado de la ejecución de cada celda se muestra en el campo **Out[n],** también numerado.

Para elegir el tipo de celda a utilizar, se selecciona en un desplegable que aparece en la fila superior junto a los iconos:

Figura 2.8. Elementos esenciales de un Notebook.

Ejecución de una celda

Todas las celdas son susceptibles de ser ejecutadas. La ejecución de una celda de código Python ejecutará el código escrito en la celda y producirá una salida. La ejecución de celdas de tipo Markdown dará formato al texto. Para ejecutar una celda, hay que colocarse en la celda y, posteriormente, pulsar el botón cuyo icono es un triángulo mirando a la derecha (figura 2.9.).

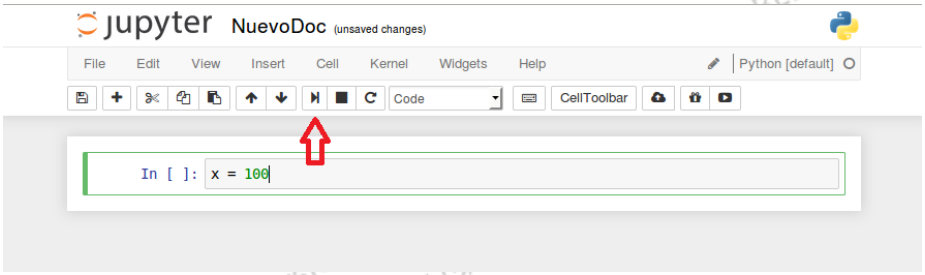


Figura 2.9. Ejecución de una celda.

De igual forma para interrumpir la ejecución, se pulsa sobre el cuadrado. Para crear celdas nuevas, se pulsa sobre el botón con el icono del signo +.

El flujo normal de edición de una celda consiste en:

- Elegir el tipo de celda. Por defecto, son de tipo Code. Dependiendo del tipo de celda elegido, Jupyter lo interpretará de diferente manera.
- Una vez introducido el código o texto en la celda, se debe ejecutar. Para ello, se pulsa sobre el icono en forma de flecha.
- A continuación, se genera una nueva celda para editar.

Ejemplo de función

Por ejemplo, si se quiere crear una celda que contenga la función: def multiplica(a,b): return a*b y después invocarla con los valores 3 y 4, se haría como se muestra en la figura 2.10.

Figura 2.10. Ejemplo de función.

IV. Elementos básicos de Python

4.1. Variables

Una variable es un nombre que referencia un valor.

Por ejemplo:

```
titulo=" Cálculo del área de un círculo"
pi=3.14159
radio=5
area= pi*(radio**2)
```

Una sentencia de asignación crea variables nuevas y las asocia a valores.

Por ejemplo:

```
mensaje="Esto es un mensaje de prueba"
n=17
pi=3.1415926535897931
```

Para mostrar el valor de una variable, se puede usar la sentencia print.

Por ejemplo:

```
print (n) #17
print (pi) # 3.1415926535897931
```

```
17
3.141592653589793
```

Las variables son de un tipo que coincide con el tipo del valor que referencian. El método **type ()** indica el tipo de una variable.

Por ejemplo:

```
type (mensaje) #str
```

```
str
```

```
type (n) #int
```

```
int
```

Algunos de los tipos más usados son:

- int: enteros.
- float: números reales.
- bool: valores booleanos: cierto y falso.
- str: cadenas.
- None: corresponde al valor nulo.

Cabe señalar que existen unas reglas de construcción de los nombres de las variables:

- Pueden ser arbitrariamente largos.
- Pueden contener tanto letras como números.
- Deben empezar con letras.
- Pueden aparecer subrayados para unir múltiples palabras.
- No pueden ser palabras reservadas de Python.

Es necesario llamar la atención sobre el hecho de que, antes de poder actualizar una variable, se debe inicializar mediante una asignación. A continuación, se puede actualizar la variable aumentándola (incrementar) o disminuyendo (decrementar).

Por ejemplo:

```
x=0
x=x+1
```

4.2. Palabras reservadas

Python reserva 31 palabras clave para su propio uso:

```
import keyword
print(keyword.kwlist)
```

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

4.3. Operadores

Los operadores son símbolos especiales que representan cálculos, como la suma o la multiplicación. Los valores a los cuales se aplican esos operadores reciben el nombre de operandos. Los principales operadores sobre los tipos int y float son:

- i+j suma
- i-j resta
- i*j multiplicación
- i/j división de dos números. Si son enteros, el resultado es un entero, y si son reales, el resultado es un real
- i//j cociente de la división entera
- i%j resto de la división entera
- i**j que representa i elevado a la potencia j
- i==j que representa i igual que j
- i!=j que representa i distinto que j
- i>j que representa i mayor que j, y de forma similar: >=, <, <=

Se pueden usar los operadores con las cadenas.

Por ejemplo:

```
3*"b"  # 'aaa'
```

```
'bbb'
```

```
"b"+"a"  # 'aa'
```

```
'ba'
```

Pero existen algunas particularidades cuando se usan los operadores sobre las cadenas.

Por ejemplo:

```
"c" * "c" #TypeError
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-10d4eab10195> in <module>()
----> 1 "c" * "c" #TypeError

TypeError: can't multiply sequence by non-int of type 'str'
```

```
len ("Psicología") # 10
```

```
10
```

```
"Psicología" [5] # 'l'
```

```
'l'
```

```
"Psicología" [12] # IndexError
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-19-751a6f6d8af6> in <module>()
----> 1 "Psicología" [12] # IndexError

IndexError: string index out of range
```

```
"Psicología" [0:5] # 'Psico'
```

```
'Psico'
```

```
"Psicología" [5: len("Psicología")] # 'logía'
```

```
'logía'
```

4.4. Expresiones

- Una expresión es una combinación de valores, variables y operadores.
- Un valor, por sí mismo, se considera una expresión y también lo es una variable.
- Las expresiones tienen un tipo. Así, por ejemplo, 6 + 7 es una expresión que representa un entero. Cuando en una expresión aparece más de un operador, el orden de evaluación depende de las reglas de precedencia. Para los operadores matemáticos, Python sigue las convenciones matemáticas:
  - El orden de los operadores es: paréntesis, exponenciales, multiplicación/división, suma/resta.
  - Cuando existe la misma precedencia, se evalúa de izquierda a derecha.

Por ejemplo:

```
4**1 +1      # 5
```

```
5
```

```
6*1**2       # 6
```

```
6
```

```
(-1)**3*3 # -3
```

```
-3
```

```
6*3/2          # 9
```

```
9.0
```

```
3/2*6          # 6
```

```
9.0
```

4.5. Comentarios

En Python comienzan con el símbolo #, de forma que todo lo que va desde # hasta el final de la línea es ignorado y no afecta al programa.

Por ejemplo:

```
#Calcula el porcentaje de hora transcurrido
minuto=300
porcentaje = (minuto*100)/60
porcentaje
```

```
500.0
```

En el ejemplo anterior, el comentario aparece como una línea completa, pero también pueden ponerse comentarios al final de una línea.

Por ejemplo:

```
porcentaje = (minuto*100)/60 #Calcula el porcentaje de hora transcurrido
porcentaje
```

```
500.0
```

4.6. Entrada de información

Python proporciona una función llamada input que recibe la entrada desde el teclado, de forma que cuando se llama el programa se detiene y espera a que el usuario escriba algo. Cuando el usuario pulsa “Intro”, el programa continúa y la función devuelve como una cadena aquello que el usuario escribió.

```
entrada = input ()
print ("Mi entrada es ", entrada)
```

```
Coche
Mi entrada es  Coche
```

Antes de recibir cualquier dato desde el usuario, es mejor escribir un mensaje explicando qué debe introducir. Se puede pasar una cadena a input, que será mostrada al usuario antes de que el programa se detenga para recibir su entrada.

```
nombre= input ("¿Cómo te llamas?\n")
print ("Mi nombre es", nombre)
```

```
¿Cómo te llamas?
Juan
Mi nombre es Juan
```

La secuencia \n al final del mensaje representa un newline, que es un carácter especial que provoca un salto de línea. Por eso, la entrada del usuario aparece debajo del mensaje. Si se espera que el usuario escriba un entero, puedes intentar convertir el valor de retorno a int usando la función int (), pero si el usuario escribe algo que no sea una cadena de dígitos, obtiene un error.

Por ejemplo:

```
deuda= input ("Deuda\n")
int (deuda)
```

```
Deuda
34

34
```

4.6. Expresiones booleanas

Una expresión booleana es aquella que puede ser verdadera (True) o falsa (False). True y False son valores especiales que pertenecen al tipo bool (booleano).

Por ejemplo:

```
type (True) # bool
```

```
bool
```

Los ejemplos siguientes usan el operador ==, que compara dos operandos y devuelve True si son iguales y False en caso contrario.

```
5==5 # True
```

```
True
```

Los principales operadores booleanos son:

- x == y # x es igual que y.
- x != y # x es distinto de y.
- x > y # x es mayor que y.
- x < y # x es menor que y.
- x >= y # x es mayor o igual que y.
- x <= y # x es menor o igual que y.
- x is y # x es lo mismo que y.
- x is not y # x no es lo mismo que y.
- not representa la negación.
- and cierto si las dos expresiones que relaciona son ciertas y falso en caso contrario.
- or falso si las dos expresiones que relaciona son falsas y cierto en caso contrario.

Por ejemplo:

- x >0 and x <10 es verdadero solo cuando x es mayor que 0 y menor que 10.
- n%2 == 0 or n%3 == 0 es verdadero si el número es divisible por 2 o por 3.
- not (x>y) es verdadero si x es menor o igual que y.

Hay que tener en cuenta que cualquier número distinto de cero se interpreta como “verdadero”.

Por ejemplo:

```
23 and True
```

```
True
```

V. Estructuras de control

5.1. Condicionales

Las expresiones condicionales facilitan la codificación de estructuras que bifurcan la ejecución del código en varias ramas o caminos de ejecución. Existen varias formas.

5.1.1. If simple

Tiene la estructura:

if expresión booleana:

ejecutar código

Por ejemplo:

```
x=5
if  x>0:
     print ("x es positivo")
```

Ahora bien, es preciso observar que:

- La expresión booleana después de la sentencia if recibe el nombre de condición. La sentencia if se finaliza con un carácter de dos puntos (:) y la/s línea/s que van detrás de la sentencia if van indentadas. Este código se denomina bloque.
- Si la condición lógica es verdadera, la sentencia indentada será ejecutada. Si la condición es falsa, la sentencia indentada será omitida.
- No hay límite en el número de sentencias que pueden aparecer en el cuerpo, pero debe haber al menos una. A veces puede resultar útil tener un cuerpo sin sentencias, usándose en este caso la sentencia pass, que no hace nada.

5.1.2. If-else

La segunda forma de la sentencia if es la ejecución alternativa, en la cual existen dos posibilidades y la condición determina cuál de ellas sería ejecutada:

if expresión booleana:

ejecutar código1

else:

ejecutar código2

Por ejemplo:

```
x=-5
if x>0:
  print ("x es positivo")
else:
  print ("x es negativo")
```

Dado que la condición debe ser obligatoriamente verdadera o falsa, solamente una de las alternativas será ejecutada. Las alternativas reciben el nombre de ramas, ya que se trata de ramificaciones en el flujo de la ejecución.

5.1.3. If-elif-else

La tercera forma de la sentencia if es elcondicional encadenado que permite que haya más de dos posibilidades o ramas:

if expresión booleana:

ejecutar código 1

elif:

ejecutar código 2

else:

ejecutar por defecto

Por ejemplo:

```
x=1
y=2
z=3
if x < y and x < z:
   print ("x es el más pequeño")
elif y < z:
   print ("y es el más pequeño")
else:
   print ("z es el más pequeño")
```

Se puede observar que:

- No hay un límite para el número de sentencias elif. Si hay una cláusula else, debe ir al final, pero tampoco es obligatorio que esta exista.
- Cada condición es comprobada en orden. Si la primera es falsa, se comprueba la siguiente y así sucesivamente. Si una de ellas es verdadera, se ejecuta la rama correspondiente y la sentencia termina. Incluso si hay más de una condición que sea verdadera, solo se ejecuta la primera que se encuentra.

Un condicional puede también estar anidado dentro de otro. Sin embargo, estos pueden ser difíciles de leer, por lo que deben evitarse y tratar de usar operadores lógicos que permitan simplificar las sentencias condicionales anidadas.

5.2. Bucles

Los bucles permiten la repetición de acciones y generalmente se construyen así:

- Se inicializan una o más variables antes de que elbucle comience.
- Se realiza alguna operación con cada elemento en el cuerpo del bucle, posiblemente cambiando las variables dentro de ese cuerpo.
- Se revisan las variables resultantes cuando el bucle se completa.

Existen varias formas:

5.2.1. WHILE

El primer tipo de bucle es el while. El cuerpo del bucle debe cambiar el valor de una o más variables, de modo que la condición pueda en algún momento evaluarse como falsa y el bucle termine. La variable que cambia cada vez que el bucle se ejecuta y controla cuándo termina este, recibe el nombre de variable de iteración. Si no hay variable de iteración, el bucle se repetirá para siempre, resultando así un bucle infinito. Cada vez que se ejecuta el cuerpo del bucle se dice que se realiza una iteración. Tiene la siguiente estructura:

while (expresión booleana):

código

Por ejemplo:

```
x=3
ans=0
bucle=x
while (bucle !=0):
    ans=ans+x
    bucle= bucle-1
print (str(x) + "*" + str(x) + "=" + str(ans))
```

```
3*3=9
```

En este ejemplo, el bucle nunca se ejecuta cuando x=0 y nunca terminará si empieza con x<0.

Es preciso observar que, a veces, no se sabe si hay que terminar un bucle hasta que se ha recorrido la mitad del cuerpo del mismo. En ese caso, se puede crear un bucle infinito a propósito y usar la sentencia break para salir explícitamente cuando se haya alcanzado la condición de salida.

Por ejemplo:

```
while True:
    linea= input ("Introducir fin para finalizar:")
    if linea=="fin":
        break
    print (linea)
```

```
Introducir fin para finalizar:fin
```

Algunas veces, estando dentro de un bucle se necesita terminar con la iteración actual y saltar a la siguiente de forma inmediata. En ese caso se puede utilizar la sentencia continue para pasar a la siguiente iteración sin terminar la ejecución del cuerpo del bucle para la actual.

5.2.2. FOR

El siguiente tipo de bucle es el for. Se repite a través de un conjunto conocido de elementos, de modo que ejecuta tantas iteraciones como elementos hay en el conjunto. Es útil utilizar la función range para crear una secuencia. Range puede tomar uno o dos valores:

- Si toma dos valores, genera todos los enteros desde la primera entrada hasta la segunda entrada-1. Por ejemplo: range (2, 5) = (2, 3, 4).
- Y si toma un solo parámetro, entonces range(x) = range(0,x).

Tiene la siguiente estructura:

for variable in secuencia:

código

Por ejemplo:

```
x=5
for i in range(x):
    print (i)
```

Téngase en cuenta que los bucles pueden estar anidados.

VI. Estructuras de datos

6.1. Tuplas

Una tupla es una secuencia de valores de cualquier tipo indexada por enteros. Las tuplas son inmutables —tienen una longitud fija y no pueden cambiarse sus elementos— y son comparables. Sintácticamente, una tupla es una lista de valores separados por comas y encerradas entre paréntesis.

Por ejemplo:

```
t=("a", "b", "c", "d", "e")
```

Para crear una tupla con un único elemento, es necesario incluir una coma al final.

```
(4,)
```

Otra forma de construir una tupla es usar la función interna tuple que crea una tupla vacía si se invoca sin argumentos, y si se le proporciona como argumento una secuencia (cadena, lista o tulpa) genera una tupla con los elementos de la secuencia.

Por ejemplo:

```
t=tuple ()
print (t)
```

```
()
```

```
t=tuple("supercalifrastilisticoespidalidoso")
print (t)
```

```
('s', 'u', 'p', 'e', 'r', 'c', 'a', 'l', 'i', 'f', 'r', 'a', 's', 't', 'i', 'l', 'i', 's', 't
', 'i', 'c', 'o', 'e', 's', 'p', 'i', 'd', 'a', 'l', 'i', 'd', 'o', 's', 'o')
```

Los principales operadores sobre tuplas son:

El operador corchete indexa un elemento.

```
t=(3, 5, "c", "d", "e")
print (t[0])
```

```
3
```

El operador slice selecciona un rango de elementos.

```
print (t[1:3])
```

```
(5, 'c')
```

Es posible reemplazar dos tuplas.

No se pueden modificar los elementos de una tupla, pero se puede reemplazar una tupla por otra.

```
t[1]=45
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-61-05771ac3f9cb> in <module>()
----> 1 t[1]=45

TypeError: 'tuple' object does not support item assignment
```

Se pueden comparar dos tuplas.

Se comienza comparando el primer elemento de cada secuencia. Si es igual en ambas, pasa al siguiente elemento, y así sucesivamente, hasta que encuentra uno que es diferente. A partir de ese momento, los elementos siguientes ya no son tenidos en cuenta.

```
(0, 1, 2) < (0, 3, 4)
```

```
True
```

6.2. Listas

Una lista es una secuencia de valores de cualquier tipo que reciben el nombre de elementos.

Creación

El método más simple para crear una lista es encerrar los elementos entre corchetes.

Por ejemplo:

```
t=[10,20,30,40]
```

Asignación

La asignación de valores a una lista no retorna nada, sin embargo, si se usa el nombre de la lista, es posible ver el contenido de la variable.

Por ejemplo:

Por ejemplo:

```
t
```

```
(3, 5, 'c', 'd', 'e')
```

Listas vacías

Una lista que no contiene elementos recibe el nombre de lista vacía —se crea con unos corchetes vacíos []—.

Por ejemplo:

```
t=[]
t
```

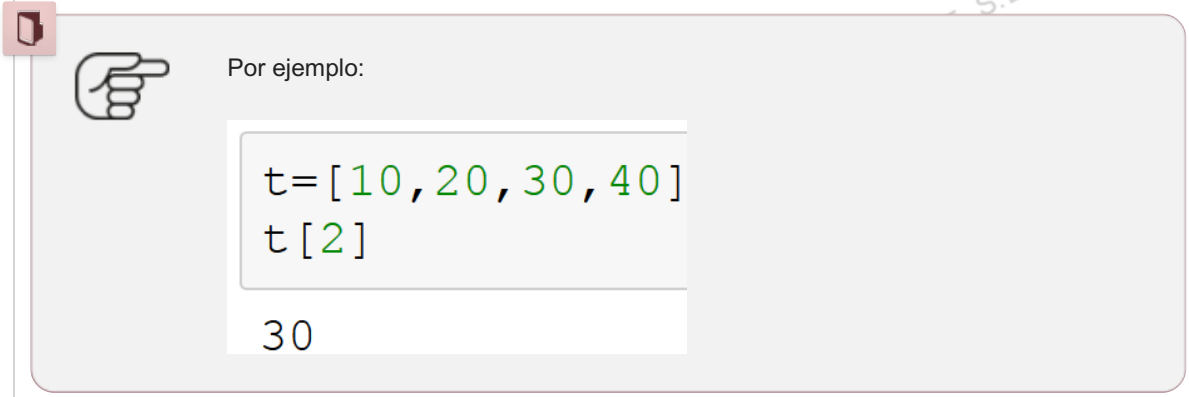
```
[]
```

Acceso a elementos

Para acceder a los elementos de una lista, se usa el operador corchete que contiene una expresión que especifica el índice —los índices comienzan por 0—. Los índices de una lista se caracterizan por:

- Cualquier expresión entera puede ser utilizada como índice.
- Si se intenta leer o escribir un elemento que no existe, se obtiene un IndexError.
- Si un índice tiene un valor negativo, se cuenta hacia atrás desde el final de la lista.

Por ejemplo:



Por ejemplo:

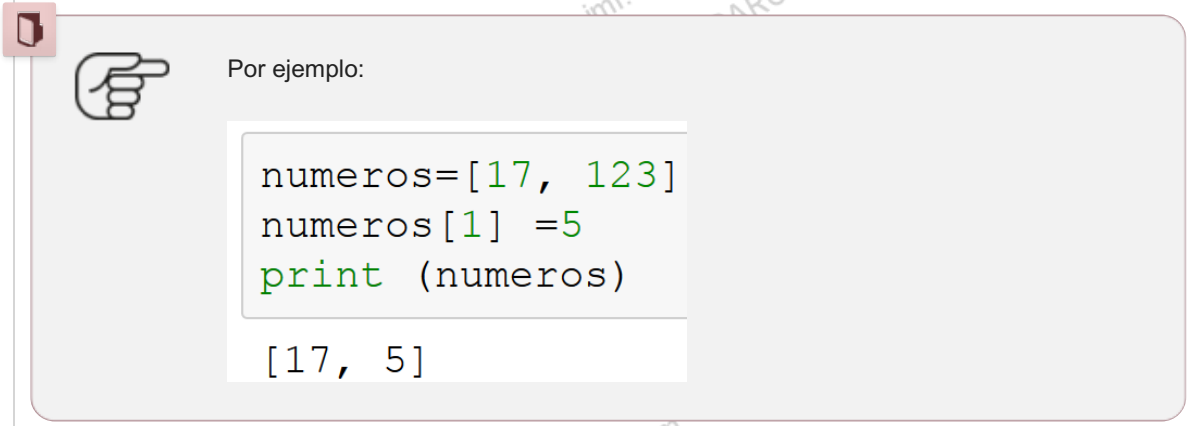
```
t=[10,20,30,40]
t[2]
```

```
30
```

Listas mutables

Las listas son mutables, puesto que su estructura puede ser cambiada después de ser creadas.

Por ejemplo:



Por ejemplo:

```
numeros=[17, 123]
numeros[1] =5
print (numeros)
```

```
[17, 5]
```

Tipo de elementos

Los elementos en una lista no tienen por qué ser todos del mismo tipo.

Por ejemplo:

Por ejemplo:

```
["casa",3.0,5,[11,20]]
```

```
['casa', 3.0, 5, [11, 20]]
```

Anidación

Cuando una lista está dentro de otra, se dice que está anidada. En una lista anidada, cada lista interna solo cuenta como un único elemento.

Por ejemplo:

```
t=[1,2,[4,5,6]]
t
```

```
[1, 2, [4, 5, 6]]
```

Indexación con números negativos

Soporta indexación con números negativos que permite seleccionar por el final de la lista.

Por ejemplo:

Por ejemplo:

```
t=[1,2,[4,5,6]]
t[-1]
```

```
[4, 5, 6]
```

6.2.1. Operadores y funciones

Las listas tienen definidos los siguientes operadores y funciones:

Concatenación

El operador + concatena listas.

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a+b
print (c)
```

```
[1, 2, 3, 4, 5, 6]
```

Pertenencia de un elemento

El operador in permite preguntar la pertenencia de un elemento a una lista.

```
a = [1, 2, 3]
2 in a
```

```
True
```

Recorrer elementos de una lista

El operador in se puede usar para recorrer los elementos de una lista usando un bucle for, como por ejemplo:

```
a = [1, 2, 3]
for i in a:
    print (i)
```

```
1
2
3
```

Repetición

El operador * repite una lista el número especificado de veces.

```
[1, 2, 3] *  3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Selección de secciones

El operador (slice) cuya sintaxis es [inicio:final:salto] permite seleccionar secciones de una lista:

```
t=[1,2,[4,5,6]]
t[1:2]
```

```
[2]
```

```
t[:]
```

```
[1, 2, [4, 5, 6]]
```

```
t[1:]
```

```
[2, [4, 5, 6]]
```

Eliminación de elementos

El operador del elimina un elemento de la lista referenciado en forma de índice.

```
t=["d", "c", "e", "b", "a"]
del t[1]
t
```

```
['d', 'e', 'b', 'a']
```

Suma de listas

La función sum () permite realizar la suma de una lista de números.

```
t=[1,2,3,4]
sum(t)
```

```
10
```

Elementos máximos y mínimos

Las funciones max () y min () proporcionan el elemento máximo/mínimo de una lista.

```
t=[1,2,3,4]
print (max(t),min(t))
```

```
4 1
```

Longitud de la lista

La función len () proporciona la longitud de una lista.

```
t=[1,2,3,4]
len(t)
```

```
4
```

Creación de una secuencia

La función range () crea una secuencia de valores a partir del dado como parámetro. Es útil para los bucles de tipo for. Por ejemplo:

```
lista=range(-3,3)
for i in lista:
    print(i)
```

```
-3
-2
-1
0
1
2
```

Ver contenido generado por range()

Para ver el contenido generado por range (), se debe usar el constructor list.

```
lista=range(-3,3)
list(lista)
```

```
[-3, -2, -1, 0, 1, 2]
```

Añadir elementos

append añade un nuevo elemento al final de una lista.

```
t=[3,4,5]
t.append("d")
t
```

```
[3, 4, 5, 'd']
```

Inserción de elementos al final

extend toma una lista como argumento y añade al final de la actual todos sus elementos.

```
t1=[3,4,5]
t2=[6,7]
t1.extend(t2)
t1
```

```
[3, 4, 5, 6, 7]
```

Ordenación de elementos

sort ordena los elementos de una lista de menor a mayor.

```
t=["d", "c", "e", "b", "a"]
t.sort()
t
```

```
['a', 'b', 'c', 'd', 'e']
```

Extracción de elementos

El método pop elimina un elemento de la lista referenciado en forma de índice. Devuelve el elemento que ha sido eliminado. Si no se proporciona un índice, borra y devuelve el último elemento.

```
t=["d", "c", "e", "b", "a"]
x= t.pop(1)
print(x, t)
```

```
c ['d', 'e', 'b', 'a']
```

Eliminar elementos

El método remove permite eliminar un elemento de la lista referenciándolo por su valor.

```
t=["d", "c", "e", "b", "a"]
t.remove("e")
t
```

```
['d', 'c', 'b', 'a']
```

6.2.2. Equivalencia en las listas

En Python dos listas son equivalentes si tienen los mismos elementos, pero no son idénticas. Sin embargo, si dos listas son idénticas, también son equivalentes, es decir, la equivalencia no implica que sean idénticas. Para comprobar si dos variables son idénticas, se puede usar el operador **is**.

En este ejemplo a y b son equivalentes pero no idénticas.

```
a=[1, 2, 3]
b=[1, 2, 3]
a is b
```

```
False
```

En este ejemplo a y b son idénticas.

```
a=[1, 2, 3]
b=a
a is b
```

```
True
```

Si a y b son idénticas significa que la lista tiene dos referencias o nombres diferentes. Así, los cambios que se hagan usando cualquiera de los nombres afectan a la misma lista.

```
a=[1, 2, 3]
b=a
b[0]=45
a
```

```
[45, 2, 3]
```

Hay que tener en cuenta que:

- En las operaciones que se realizan sobre las listas existen aquellas que modifican listas y otras que crean listas nuevas. Por ejemplo, el método append modifica una lista, pero el operador + crea una lista nueva.

```
t1=[2,3]
t2=[5]
t1.append(4)
print (t1, t1+t2)
```

```
[2, 3, 4] [2, 3, 4, 5]
```

- La mayoría de los métodos modifican la lista y devuelven el valor None.

6.2.3. Cadenas

Una cadena es una secuencia de caracteres y una lista es una secuencia de valores, pero una lista de caracteres no es lo mismo que una cadena. Para convertir desde una cadena a una lista de caracteres, se puede usar la función list, que divide una cadena en letras individuales.

```
s= "casa"
t= list(s)
print (t)
```

```
['c', 'a', 's', 'a']
```

Si se quiere dividir una cadena en palabras, puedes usar el método split.

Por ejemplo:

```
s = "El camión rojo de Juan"
t= s.split()
print (t)
```

```
['El', 'camión', 'rojo', 'de', 'Juan']
```

Una vez usado split, se puede utilizar el operador índice (corchetes) para buscar una palabra concreta en la lista.

```
print (t[2])
```

```
rojo
```

Se puede llamar a split con un argumento opcional denominado delimitador, que especifica qué caracteres se deben usar como delimitadores de palabras.

```
s="El-camión-rojo-de-Juan"
delimitador = "-"
s.split(delimitador)
```

```
['El', 'camión', 'rojo', 'de', 'Juan']
```

Join es la inversa de split y toma una lista de cadenas y concatena sus elementos. Al ser un método de cadena, debe invocarse sobre el delimitador y pasarle la lista como un parámetro.

Por ejemplo:

```
t=["El","camión","rojo","de","Juan"]
delimitador = " "
delimitador.join(t)
```

```
'El camión rojo de Juan'
```

6.2.4. Listas por comprensión

Una lista por comprensión es una expresión compacta para definir listas, conjuntos y diccionarios en Python. Se trata de definir cada uno de los elementos sin tener que nombrar cada uno de ellos. La forma general es:

[exp for val in <coleccion> if <condicion>]

Ejemplos:

```
lista=[x for x in [3,4,5]]
lista
```

```
[3, 4, 5]
```

```
lista=[x+5 for x in [1,2,3,4,5] if x>3]
lista
```

```
[9, 10]
```

```
lista = [ x for x,y in [(1,2), (3,4), (5,6)] ]
lista
```

```
[1, 3, 5]
```

```
letras = ['a', 'b', 'g', 'h', 'n' ]
mayusculas = [a.upper() for a in letras ]
mayusculas
```

```
['A', 'B', 'G', 'H', 'N']
```

6.3. Diccionarios

Un diccionario es una colección **no ordenada** de pares **clave-valor** donde la clave y el valor son objetos que pueden ser de (casi) cualquier tipo. La función dict () crea un diccionario nuevo sin elementos.

```
ejemplo= dict()
print (ejemplo)
```

```
{}
```

Las llaves {} representan un diccionario vacío.

Para añadir elementos al diccionario, se pueden usar corchetes y usar acceso indexado a través de la clave.

Por ejemplo:

```
ejemplo["primero"] = "Libro"
print (ejemplo)
```

```
{'primero': 'Libro'}
```

Otra forma de crear un diccionario es mediante una secuencia de pares clave-valor separados por comas y encerrados entre llaves.

Por ejemplo:

```
ejemplo2={"primero": "libro", "segundo":34, "tercero":(3,4)}
print (ejemplo2)
```

```
{'primero': 'libro', 'segundo': 34, 'tercero': (3, 4)}
```

El orden de los elementos en un diccionario es impredecible, pero eso no es importante, ya que se usan las claves para buscar los valores correspondientes. En este sentido, si la clave especificada no está en el diccionario se obtiene una excepción.

Algunos métodos:

El método len()

La función len devuelve el número de parejas clave-valo

```
len(ejemplo2)
```

```
3
```

El método in()

El operador in dice si algo aparece como clave en el diccionario.

```
"primero" in ejemplo2
```

```
True
```

El método values()

Para ver si algo aparece como valor en un diccionario, se puede usar el método values, que devuelve los valores como una lista, y después usar el operador in sobre esa lista.

```
valores= ejemplo2.values()
"uno" in valores
```

```
False
```

El método get ()

Toma una clave y un valor por defecto. Si la clave aparece en el diccionario get, devuelve el valor correspondiente. En caso contrario, devuelve el valor por defecto.

```
contadores={"naranjas": 1, "limones": 42, "peras": 100}
print (contadores.get("uvas", 0))
```

```
0
```

El método keys()

Crea una lista con las claves de un diccionario.

```
contadores.keys()
```

```
dict_keys(['naranjas', 'limones', 'peras'])
```

El método ítems()

Devuelve una lista de tuplas, cada una de las cuales es una pareja clave-valor sin ningún orden definido.

```
t= contadores.items()
print (t)
```

```
dict_items([('naranjas', 1), ('limones', 42), ('peras', 100)])
```

Téngase en cuenta lo siguiente:

- Se puede utilizar un diccionario como una secuencia en una sentencia for, de manera que se recorren todas las claves del diccionario.

Por ejemplo:

```
diccionario={1: "hola", 2: 42, 3: 100}
for clave in diccionario:
      print (clave, diccionario[clave])
```

```
1 hola
2 42
3 100
```

- El ejemplo anterior se podría haber realizado de una manera equivalente utilizando el método ítems ().

```
diccionario={1: "hola", 2: 42, 3: 100}
for clave,valor in diccionario.items():
      print (clave, valor)
```

```
1 hola
2 42
3 100
```

VII. Funciones

Una función es una secuencia de sentencias que realizan una operación y que reciben un nombre. Sus principales características son:

- Cuando se define una función, se especifica el nombre y la secuencia de sentencias.
- Una vez que se ha definido una función, se puede llamar a la función por ese nombre y reutilizarla a lo largo del programa.
- El resultado de la función se llama valor de retorno.

Creación

Para crear una función se utiliza la palabra reservada**def**. A continuación, aparece el nombre de la función, entre paréntesis los parámetros, y finaliza con :. Esta línea se denomina cabecera de la función. Después de los : aparece el código que se ejecuta cuando se llama a la función. Este trozo de código, se denomina cuerpo de la función y debe estar indentado. El cuerpo puede contener cualquier número de sentencias. Para devolver el valor se usa la palabra reservada return.

Por ejemplo:

```
# función que suma 3 números y devuelve el resultado
def suma_tres(x, y, z):    # 3 argumentos posicionales.
    m1 = x + y
    m2 = m1 + z
    return m2
```

Definición y llamada

Las reglas para los nombres de las funciones son los mismos que para las variables: se pueden usar letras, números y algunos signos de puntuación, pero el primer carácter no puede ser un número. No se puede usar una palabra clave como nombre de una función y se debería evitar también tener una variable y una función con el mismo nombre. Las funciones con paréntesis vacíos después del nombre indican que esta función no toma ningún argumento.

La sintaxis para llamar a una función definida consiste en indicar el nombre de la función junto a una expresión entre paréntesis denominados argumentos de la función. El argumento es un valor o variable que se pasa a la función como parámetro de entrada.

```
# invocación de la función
r = suma_tres(1, 2, 3)
r
```

Algunas características

- La definición de una función debe ser ejecutada antes de que la función se llame por primera vez, y no generan ninguna salida. Sin embargo, las sentencias dentro de cada función son ejecutadas solamente cuando se llama a esa función.
- En las funciones no se especifica el tipo de parámetro ni lo que se retorna.
- Las definiciones de funciones no alteran el flujo de la ejecución de un programa debido a que las sentencias dentro de una función no son ejecutadas. Sin embargo, una llamada a una función es como un desvío en el flujo de la ejecución. En vez de pasar a la siguiente sentencia, el flujo salta al cuerpo de la función, ejecuta todas las sentencias que hay allí y después vuelve al punto donde lo dejó.
- Las funciones que disponen de argumentos son asignadas a variables llamadas parámetros. Se puede usar cualquier tipo de expresión como argumento, la cual será evaluada antes de que la función sea llamada. El nombre de la variable que se pasa como argumento no tiene nada que ver con el nombre del parámetro, de manera que dentro de la función recibirá el nombre del parámetro.

```
def sumados(a,b):
    suma= a+b
    return suma
c=5
x= sumados (c, c+7)
print(x)
```

```
17
```

- Cuando se definen los argumentos de una función, estos pueden tener valores por defecto.

Por ejemplo:

```
def ejemplo(a=3):
    print(a)

ejemplo()
```

```
3
```

- Una vez que se ha definido una función puede usarse dentro de otra, facilitando de esta manera la descomposición de un problema, y resolverlo mediante una combinación de llamadas a funciones.

Por ejemplo:

```
def ejemplo1(a,b):
    return a+b

def ejemplo2(a,b,c):
    return c+ejemplo1(a,b)
ejemplo2(3,4,5)
```

```
12
```

Hay dos tipos de funciones:

Funciones con retorno

Aquellas que producen resultados, con los que se querrá hacer algo, como asignárselo a una variable.

```
def ejemplo1(a):
    return 3*a
b=ejemplo1(4)
print (b)
```

```
12
```

Funciones sin retorno

Aquellas que realizan alguna acción, pero no devuelven un valor y, sin embargo, pueden mostrar algo por pantalla. Si se asigna el resultado a una variable, se obtiene el valor None.

```
def ejemplo1(a):
    print (3*a)
b=ejemplo1(4)
print (b)
```

```
12
None
```

Funciones internas de Python

Python proporciona un número importante de funciones internas que pueden ser usadas sin necesidad de tener que definirlas previamente:

- Las funciones max y min dan respectivamente el valor mayor y menor de una lista.
- La función len devuelve cuantos elementos hay en su argumento. Si el argumento es una cadena devuelve el número de caracteres que hay en la cadena.
- Funciones que permiten convertir valores de un tipo a otro: int(), float(), y str().

En Python el paso de argumentos a una función se hace por referencia, de manera que las modificaciones que se hagan sobre los argumentos se mantienen después de la llamada y ejecución de la función.

VIII. Importación de módulos

Python dispone de una amplia variedad de módulos o librerías. Los módulos son programas que amplían las funciones y clases de Python para realizar tareas específicas. Los módulos tienen extensión py.

En la página web de Python (https://docs.python.org/3/py-modindex.html), se puede encontrar el índice de módulos de Python.

Para poder utilizarlas, hay que importarlas previamente, lo cual se puede hacer de varias formas:

Importar todo el módulo mediante la palabra reservada import.

De esta manera para utilizar un elemento hay que usar el nombre del módulo seguido de un punto (.) y el nombre del elemento que se desee obtener.

```
import random
for i in range (4):
        x= random.random ()
        print (x)
```

Importar solo algunos elementos del módulo.

Mediante la estructura from nombre_modulo import lista_elementos, los elementos importados se usan directamente por su nombre.

```
from random import randint
for i in range(4):
    x=randint (1,10)
    print(x)
```

Importar y definir un alias.

Importar todo el módulo mediante la palabra reservada import y definir un alias mediante la palabra reservada as, de manera que para usar un elemento hay que utilizar el nombre del módulo seguido de un punto (.) y el nombre del elemento que se desee obtener.

```
import random as rd
for i in range (4):
        x= rd.random ()
        print (x)
```

Para conocer las operaciones disponibles de un módulo, se puede usar el comando **dir**.

```
import math
dir(math)
```

IX. Gestión de archivos

Apertura

Para abrir un archivo en Python se usará la función **open**, que recibe el nombre del archivo a abrir. Por defecto, si no se indica nada, el archivo se abre en modo lectura.

La función open abrirá el archivo con el nombre indicado. Si no tiene éxito, se lanzará una excepción. Si se ha podido abrir el archivo correctamente, la variable asignada a la apertura permitirá manipularlo.

Lectura

La operación más sencilla que se debe realizar sobre un archivo es leer su contenido. Para procesarlo línea por línea, es posible hacerlo de la siguiente forma:

```
fichero= open("cuna.txt")
for linea in fichero:
    print(linea)
```

```
Ya te vemos dormida.

Tu barca es de madera por la orilla.

Blanca princesa de nunca.

Duerme por la noche oscura.

Cuerpo y tierra de nieve.

Duerme por el alba, duerme.

Ya te alejas dormida.
```

Además, usando la función **readlines** es posible recuperar de una sola vez todo el contenido del archivo estructurado en forma de líneas.

```
fichero = open("cuna.txt")
lineas = fichero.readlines()
lineas
```

```
['Ya te vemos dormida. \n',
 'Tu barca es de madera por la orilla. \n',
 'Blanca princesa de nunca. \n',
 'Duerme por la noche oscura.\n',
 'Cuerpo y tierra de nieve. \n',
 'Duerme por el alba, duerme.\n',
 'Ya te alejas dormida. ']
```

En este caso, la variable líneas tendrá una lista de cadenas con todas las líneas del archivo. Téngase en cuenta que es posible eliminar los saltos de línea

```
lineas[0].rstrip()
```

Apertura para escritura

Si se quiere abrir un archivo en modo escritura, hay que indicar una w como segundo parámetro de la función open. En caso de que no exista el archivo se crea y, si existe, se pierde la información que hubiera.

```
arc_write = open('nuevo.txt', 'w')
for i, line in enumerate(lineas):
    if i%2 == 0:
        arc_write.write(str(i) + ' ' + line)
    else:
        pass
```

Cierre

Al terminar de trabajar con un archivo, se debe cerrar, ya que lo que se haya escrito no se guardará realmente hasta no cerrar el archivo. Para ello, se usa close.

```
arc_write.close()
```

Apertura con posicionamiento

También es posible abrir un archivo en modo escritura posicionándose al final del mismo. Para ello, se usa la opción a con la función open. En este caso se crea el archivo si no existe, pero en caso de que exista se posiciona al final, manteniendo el contenido original.

```
open('nuevo.txt', 'a').write('\nEste es el final')
```

X. Resumen

En esta unidad, se ha introducido el lenguaje de programación Python. Se trata de un lenguaje de propósito general que dispone de potentes librerías para análisis de datos, lo que le convierte en uno de los lenguajes más utilizados en el ámbito del Big Data.

Uno de los entornos de trabajo más utilizados en Python es el Jupyter Notebook. Consiste en una herramienta que permite integrar código con otros recursos, como vídeo, imágenes o código html.

También se ha realizado un repaso de los elementos básicos del lenguaje Python, comenzando por las variables, expresiones y operadores. A continuación, se han revisado las principales estructuras de control, las estructuras de datos fundamentales (listas, tuplas, diccionarios y cadenas), las funciones como principal elemento de estructuración de los programas en Python y, por último, otros aspectos del lenguaje, como importación de módulos o apertura de ficheros.

XI. Caso práctico

Considera un sistema de cifrado en el que se sustituye cada letra en el texto original por otra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Por ejemplo, si el desplazamiento es 3 posiciones y se considera la letra A, entonces sería sustituida por la letra D, que se encuentra situada 3 lugares a la derecha de la A. Se considera que el alfabeto es circular por lo que a continuación de la Z comienza la letra A. Solo se codifican las letras, el resto de símbolos se mantienen.

Una vez cifrado el texto, si este contiene más de una palabra, se reordenan las palabras cifradas, moviendo cada palabra m posiciones hacia la derecha. Así, la palabra que ocupa la posición 1 se mueve a la posición m+1, y así sucesivamente —la palabra que ocupa la posición n se moverá a la posición m—.

Se pide implementar un programa en Python que solicite al usuario que introduzca por teclado un texto a codificar, dos números que representan el desplazamiento de letras y el desplazamiento de las palabras codificadas. Como resultado, el programa mostrará por pantalla el mensaje codificado. Se deben hacer las comprobaciones necesarias sobre la entrada, es decir, es una cadena y 2 números.

Solución:

```
def cifrar(texto,desp,desp2):
    muestra = ""
    texto = convertir(texto,desp2)
    palabras = texto.split(' ')
    copia = palabras[:]
    indices = []
    for i in range(0,len(palabras)):
        indices.append((i + int(desp)) % len(palabras))
        copia[indices[i]] = palabras[i]
    for j in range(0,len(copia)):
        muestra += "".join(copia[j])+" "
    print ("\nTexto cifrado: "+ muestra)


def convertir(text,desp2):
    resul = ""
    for car in text:
```

```
        if car.isalpha():
            if car.islower():
                resul += chr((ord(car) - 97 + int(desp2)) % 26 + 97)
            if car.isupper():
                resul += chr((ord(car) - 65 + int(desp2)) % 26 + 65)
        else:
            resul += car
    return resul


if __name__ == "__main__":
    texto = input("Texo a cifrar: ")
    desp2 = input("Desplazamiento letra: ")
    desp = input("Desplazamiento palabra: ")
    if desp.isdigit() and desp2.isdigit():
        cifrar(texto,desp,desp2)
    else:
        print ("El desplazamiento ha de ser un digito")
```

Recursos

Enlaces de Interés

https://docs.python.org/3/py-modindex.html
https://docs.python.org/3/py-modindex.html
Página web de Python

http://continuum.io/downloads
http://continuum.io/downloads
Página web de Anaconda.

https://docs.continuum.io/anaconda/install
https://docs.continuum.io/anaconda/install
Página web de Anaconda. Installation

https://docs.python.org/3/py-modindex.html
https://docs.python.org/3/py-modindex.html
Página web de Python.

Bibliografía

- **Bahit, Eugenia. *Python para principiantes*. 2011-2013. [En línea] URL disponible en http://librosweb.es/libro/python/** :
- **Marzal Varó, Andrés, Gracia Luengo, Isabel, y García Sevilla, Pedro.** : Marzal Varó, Andrés, Gracia Luengo, Isabel, y García Sevilla, Pedro. *Introducción a la programación con Python 3*. Castelló de la Plana: Publicacions de la Universitat Jaume I. Servei de Comunicació i Publicacions; 2014. [En línea] URL disponible en http://repositori.uji.es/xmlui/bitstream/handle/10234/102653/s93_impressora.pdf?sequence=2&isAllowed=y
- **Rossum, Guido van. *Guía de aprendizaje de Python. Release 2.0*. Fred L. Drake, Jr. (ed. orig.); 2000. [En línea] URL disponible en http://es.tldp.org/Tutoriales/Python/tut.pdf** :
- **Rossum, Guido van. *Tutorial de Python*. Fred L. Drake, Jr. (ed. orig.); 2009. [En línea] URL disponible en http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf** :

Glosario.

- **Bucle**: estructura de control de Python que permite repetir un conjunto de acciones un número de veces que depende de una condición determinada.
- **Condicional**: estructura de control de Python que permite ejecutar acciones alternativas de acuerdo con el valor de una expresión.
- **Diccionario**: estructura de datos de Python que permite almacenar la información en forma de parejas clave-valor. Es un tipo de datos mutable.
- **Entorno de desarrollo**: también denominado editor. Es una aplicación informática que permite crear y ejecutar programas para un determinado lenguaje de programación.

- **Estructura de control**: hace referencia a sentencias de un lenguaje que permiten estructurar el flujo de ejecución de un programa.

- **Estructura de datos**: hace referencia a una forma lógica determinada de almacenar la información.

- **Expresión**: conjunto de valores y/o variables combinadas mediante un conjunto de operadores que representan un valor de un tipo de datos.

- **Función**: es la unidad de estructuración de un programa en Python. Representa un conjunto de acciones que reciben un nombre y que pueden depender de un conjunto de parámetros y devolver como resultado uno o más valores.

- **Jupyter notebook**: entorno de desarrollo para Python.

- **Lista**: estructura de datos de Python que permite almacenar un conjunto de datos de diferentes tipos. Es un tipo de datos mutable.

- **Listas por comprensión**: es una forma de definir una lista en la que, en vez de indicar los valores concretos, se indica la expresión que permite generar dichos valores.

- **Módulo**: también denominado librería, se trata de un conjunto de funciones y/o métodos que añaden nueva funcionalidad a Python con respecto a la que dispone de manera estándar.

- **Mutabilidad**: es una propiedad de las estructuras de datos en Python que indica si los datos pueden modificar su estructura y contenido una vez que se han definido.

- **Notebook**: tipo de archivo utilizado por el Jupyter Notebook para editar código Python que permite integrar otros recursos como vídeos o imágenes.

- **Programa**: se trata de un conjunto de sentencias de un lenguaje de programación, cuya ejecución produce un resultado.

- **Tupla**: estructura de datos de Python que permite almacenar un conjunto de datos de diferentes tipos. Es similar a las listas, pero no mutable.

- **Variable**: nombre que referencia una posición de memoria.