Curso Básico de Python

7. Entrada y salida

Contenido

Python ofrece diversas formas de presentar la salida de un programa en un formato fácilmente legible. También es posible grabar y leer desde archivos. En este capítulo examinamos estas técnicas.

Lecturas recomendadas:

1. Entrada y salida en el Tutorial de Python.

Referencias para consulta:

1. Python Docs. Tipos integrados, Cadenas de caracteres - `str`, el tipo `str` y sus métodos.
2. Python Docs. `string` - operaciones comunes de cadenas de caracteres, sintaxis de `format` para presentar las cadenas de caracteres.
3. Python Docs. Funciones incorporadas, `open`, abrir archivos para lectura y escritura.
4. Python Docs. `pickle`, serialización de objetos en Python.

1 Salida simple

Hasta ahora hemos la función `print` usado para mostrar la salida :

```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

La función `print` imprime el o los valores indicados como argumentos; convierte distintos tipos de datos a `string`. El separador `sep` es por defecto un espacio, pero puede indicarse otro caracter. La impresión termina con el caracter indicado en `end`, por defecto un cambio de línea, pero puede indicarse un espacio u otro caracter. La salida es el flujo de salida del sistema `sys.stdout`, normalmente asignado a la pantalla, pero puede indicarse otro. La opción `flush` fuerza la descarga hacia el flujo de salida. Algunos ejemplos:

```
>>> print("cadena caracteres", 3.33, 42, 1+3j, True, False, None, sep="; ")
cadena caracteres; 3.33; 42; (1+3j); True; False; None
>>> for i in range(2,10,2):
    print(i, end=", ")

2, 4, 6, 8,
```

La función `repr` devuelve una cadena de caracteres como representación canónica de un objeto; la *representación canónica* es una representación estándar vinculada a la forma trabajo interno del

sistema. En lo siguientes ejemplos, `print` muestra un formato legible, `repr` destaca el carácter de cadena de caracteres incluyendo comillas y mantiene \n sin interpretar:

```
>>> print(0.23)
0.23
>>> repr(0.23)
'0.23'
>>> print("cadena de caracteres\nen dos líneas")
cadena de caracteres
en dos líneas
>>> repr("cadena de caracteres\nen dos líneas")
"'cadena de caracteres\\nen dos líneas'"
```

Las funciones de formato de cadenas permiten presentar salidas prolijas, como una tabla con ancho de columna fijo:

```
>>> for x in range(0,10):
    print(str(x).rjust(10), str(x*x).rjust(10), str(x*x*x).rjust(10))

         0          0          0
         1          1          1
         2          4          8
         3          9         27
       ...
         8         64        512
         9         81        729
```

2 El método `format`

El método `str.format` permite presentar cadenas de caracteres a partir de parámetros con variedad de opciones para los distintos tipos de datos.

Parámetros posicionales:

```
>>> "{} de {} formateada con {}".format("cadena", "caracteres", "format")
'cadena de caracteres formateada con format'
```

Parámetros nominados por posición:

```
>>> "el {0} {1} al {2} con un {3}".\
    format('policía','atrapó','ladrón','telescopio')
'el policía atrapó al ladrón con un telescopio'
>>> "el {0} {1} al {2} con un {3}".\
    format('Cholo','corrió','hermano','palo')
'el Cholo corrió al hermano con un palo'
```

Parámetros nominados por posición y palabra clave:

```
>>> "la {0} {1} de {material} es {color}".\
    format('casa','nueva',material='madera',color='blanca')
'la casa nueva de madera es blanca'
>>> "la {material} de la {1} {0} es {color}".\
    format('casa','nueva',material='madera',color='blanca')
'la madera de la nueva casa es blanca'
```

El formato numérico permite indicar el ancho total y la cantidad de decimales, así como si el número es entero 'd' o de punto flotante 'f':

```
>>> '{0} {1:10d} {2:10.2f}'.format("anchos 20-10-10 --->", 2367, 123.4567)
'anchos 20-10-10 --->       2367     123.46'
```

Tabla con anchos de columna fijos y formato numérico:

```
>>> factura = [ ['clavo', 0.50, 10], ['tornillo',1.50, 5] ]
```

```
>>>
print('{0}{1}{2}{3}'.format('item'.rjust(10),'p/unit'.rjust(10),'cant'.rjust
(10),'subtotal'.rjust(10)))
      item    p/unit      cant  subtotal
>>> for lin in factura:
   print('{0}{1:10}{2:10.2f}
{3:10.2f}'.format(lin[0].rjust(10),lin[1],lin[2], lin[1]*lin[2]))

     clavo       0.5     10.00      5.00
  tornillo       1.5      5.00      7.50
```

Acceso a los atributos de los argumentos:

```
>>> z = 3.11 + 2.214j
>>> 'complejo {0:13.3f}, real={0.real:6.3f}, imag={0.imag:6.3f}j'.format(z)
'complejo  3.110+2.214j, real= 3.110, imag= 2.214j'
```

También es posible dar formato anteponiendo 'f' o 'F', con la misma sintaxis de `format`:

```
>>> import math
>>> print(f'El valor aproximado de pi es {math.pi:.3f}.')
El valor aproximado de pi es 3.142.
>>> telefonos = { 'Hugo':123, 'Paco':124, 'Luis':125}
>>> for nombre, interno in telefonos.items():
   print(f'{nombre:10} ==> {interno:10d}')

Hugo       ==>        123
Paco       ==>        124
Luis       ==>        125
```

La sintaxis completa para formatos se puede ver en la documentación de Python, `string` - operaciones comunes de cadenas de caracteres.

3 Manejo de archivos

El comando `open` permite crear o abrir un archivo para lectura o escritura. Con `help("open")` podemos ver la sintaxis completa. En su forma más simple:

```
open("archivo", modo='r')
```

Abre un archivo y devuelve un flujo en el cual es posible escribir o leer. En caso de error levanta una excepción `Raise OSError`. Los modos más comunes son:

- `'r'` abre para lectura (opción por defecto).
- `'w'` abre para escritura, trunca el archivo si existe.
- `'x'` crea un nuevo archivo y lo abre para escritura.
- `'a'` abre para escritura agregando al final si el archivo ya existe.

El modo por defecto es `'rt'`, lectura de texto.

Crear un archivo nuevo, agregar algunas líneas, y cerrarlo.

```
>>> zen_de_python = ["=== Zen de Python ===", "Bello es mejor que feo",
"Explícito es mejor que implícito", "Simple es mejor que complejo"]
>>> f = open("f_zen", "x")     # archivo nuevo para escritura
>>> for lin in zen_de_python:
   f.write(lin + "\n")         # escribe línea, agrega cambio de línea

22
23
33
29
>>> f.close()                  # cierra el archivo
```

Abrir el archivo y agregar una línea al final; cerrar.

```
>>> f = open("f_zen", "a")      # abre archivo para agregar
>>> f.write("Complejo es mejor que complicado" + "\n")  # agrega al final
33
>>> f.close()
```

Abrir el archivo para lectura, leer todo su contenido en un texto único, cerrar.

```
>>> f = open("f_zen", "r")      # abre archivo para lectura
>>> txt = f.read()              # lee todo el archivo
>>> f.close()
>>> print(txt)
=== Zen de Python ===
Bello es mejor que feo
Explícito es mejor que implícito
Simple es mejor que complejo
Complejo es mejor que complicado
```

Abrir el archivo para lectura, leer por líneas generando una lista, cerrar.

```
>>> f = open("f_zen", "r")
>>> lines = f.readlines()       # lee todas las líneas en una lista
>>> f.close()
>>> print(lines)                # imprime la lista
['=== Zen de Python ===\n', 'Bello es mejor que feo\n', 'Explícito es mejor
que implícito\n', 'Simple es mejor que complejo\n', 'Complejo es mejor que
complicado\n']
```

Para la descipción completa de `open`, ver en Documentaciòn Python, Funciones incorporadas, `open`.

El Zen de Python es un listado de los principios de diseño y la filosofía de Python; se puede ver escirbiendo `import this` en IDLE o en el intérprete de Python.

Los principales métodos de entrada y salida en archivo son:

- `read([size])` : lee del archivo y devuelve una cadena única de largo `size` bytes a lo sumo. Si `size` es negativo o `None` (se omite), lee hasta el final del archivo.
- `readline([size])` : lee del archivo hasta el caracter \n o el fin de archivo y devuelve una cadena. Si ya está al final del archivo devuelve una cadena nula. Si se indica `size`, lee como máximo esa cantidad.
- `readlines([hint])` : lee del archivo y devuelve una lista de líneas, cadenas terminadas en \n, tantas como indique `hint`; si este valor se omite, es negativo o cero, se lee todo el archivo.
- `tell()` : devuelve la posición actual en el archivo.
- `seek(offset[, whence])` : cambia la posición del archivo en la cantidad `offset` de bytes a partir de la posición `whence`: desde principio del archivo si es 0, desde la posición actual si es 1, desde el final del archivo si es 2. Devuelve la posición alcanzada.
- `write(cadena)` : escribe la cadena de caracteres al archivo.

4 Serialización de objetos con `pickle`

Python admite estructuras de datos complejas. Para asegurar la persistencia de esas estructuras, es preciso poder guardarlas en un archivo y luego recuperarlas exactamente iguales. Esto se logra convirtiendo las estructuras en una secuencia de bytes; este proceso se denomina *serialización*. El proceso inverso, convertir la secuencia de bytes en las estructuras complejas se denomina *deserialización*. El módulo `pickle` permite guardar y recuperar estructuras de Python hacia y desde un archivo. Sus principales funciones son:

```
dump(objeto, archivo)  : serializa el objeto y lo guarda en el archivo
load(archivo) : deserializa y recupera un objeto; se puede reiterar
```

En los ejemplos, el módulo `demo_pickle` muestra el uso de estas funciones.

5 Ejercicios

1. Probar en IDLE o en el intérprete de Python los ejemplos de este capítulo. Asegurarse de entender bien cómo se está dando el formato en cada caso.

2. Para dar formato a cadenas de caracteres son útiles las funciones `ljust(), rjust(), zfill()`. ¿Qué hace dada una? Sugerencia: `help("zfill")`, etc. Probar estas funciones ¿Qué otras funciones de `str` pueden ser útiles para el formato? Sugerencia: `help(str)`, o en la documentación de Python, Cadenas de caracteres - `str`, sobre el tipo `str` y sus métodos.

3. Ejecutar y examinar el código del módulo `demo_archivos`.

4. Grabar en el archivo `numerados.txt` la cadena
   `st = '0----+----1----+----2----+----3----+----4----+----5'`
   Abrir el archivo en modo lectura. Leer 10 caracteres. Verificar la posición del puntero. ¿Cómo desplazamos el puntero al principio del archivo? ¿Cómo desplazamos el puntero al final del archivo? ¿Cómo leemos los caracteres del 16 al 25 inclusive? Verificar.

5. Ejecutar y examinar el código del módulo `demo_pickle`.

6. En el ejemplo de recuperación de objetos `demo_pickle`, ¿qué pasa si se intenta leer un tercer objeto? ¿Puede leerse un archivo `pickle` con objetos guardados sin saber a priori la cantidad de objetos? ¿Cómo?

7. Los ejemplos `demo_pickle` y `demo_archivos` se invocan mediante opciones en la línea de comando, e.g. `$ python3 demo_pickle.py --pickdump`. ¿Cómo debería modificarse el código para presentar un menú de opciones? ¿Sería posible hacer coexistir las dos formas, invocar por opción o a través de un menú?
