T05_Entrada_Salida_Archivos

January 29, 2021

1 Python de cero a experto

Autor: Luis Miguel de la Cruz Salas

1.1 Pythonico es más bonito

1.1.1 Entrada y salida estándar

La entrada estándar para proporcionar información a un programa se realiza mediante la función *input*.

[1]:
```
input()
```

```
 dato
```

[1]:
```
'dato'
```

[2]:
```
entrada = input()
```

```
 otro
```

[3]:
```
print(entrada)
```

```
otro
```

[4]:
```
entrada = input('Teclea un valor ')
```

```
Teclea un valor  3.1416
```

[5]:
```
print(entrada, type(entrada))
```

```
3.1416 <class 'str'>
```

[6]:
```
entrada = int(input('Teclea un valor '))
```

```
Teclea un valor  2.1345
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-51eb3e1713d8> in <module>
----> 1 entrada = int(input('Teclea un valor '))

ValueError: invalid literal for int() with base 10: '2.1345'
```

[7]:
```
print(entrada, type(entrada))
```

```
3.1416 <class 'str'>
```

Existen varias formas de presentar la salida de un programa al usuario: en la pantalla (generalmente la salida estándar)

En ambos casos se desea un control adecuado sobre el formato de la salida. Para ello se tienen varias maneras de controlar la salida:

- Con el uso de cadenas con formato, las cuales comienzan con f o F:

[8]:
```
nombre = 'LUIS MIGUEL'
edad = 25
f'Hola mi nombre es {nombre} y tengo {edad} años'
```

[8]:
```
'Hola mi nombre es LUIS MIGUEL y tengo 25 años'
```

[9]:
```
import math
print(f'El valor de PI es aproximadamente {math.pi:.10f}.') # {valor:formato}
```

```
El valor de PI es aproximadamente 3.1415926536.
```

[10]:
```
na1 = 'Fulano'; n1 = 55_21345678
na2 = 'Sutano'; n2 = 77_12932143
print(f'{na1:10} ==> {n1:15d}') # alineación del texto
print(f'{na2:10} ==> {n2:15d}') # alineación del texto
```

```
Fulano     ==>      5521345678
Sutano     ==>      7712932143
```

Véase Format Specification Mini-Language

- Con el uso del método format

[11]:
```
print('El curso se llama "{}" y tenemos {} alumnos'.format('Python de cero a␣
↪experto', 100))
```

```
El curso se llama "Python de cero a experto" y tenemos 100 alumnos
```

[12]:
```
votos_a_favor = 42_572_654
votos_en_contra = 43_132_495
```

```
total_de_votos = votos_a_favor + votos_en_contra
porcentaje = votos_a_favor / total_de_votos
'{:>10} votos a favor ({:2.2%})'.format(votos_a_favor, porcentaje)
```

[12]:
```
'  42572654 votos a favor (49.67%)'
```

[13]:
```
print('{0} y {1}'.format('el huevo', 'la gallina'))
print('{1} y {0}'.format('el huevo', 'la gallina'))
```

```
el huevo y la gallina
la gallina y el huevo
```

[14]:
```
print('Esta {sustantivo} es {adjetivo}.'.format(adjetivo='exquisita',␣
↪sustantivo='comida'))
```

```
Esta comida es exquisita.
```

[15]:
```
print('El {0}, el {1}, y el {otro}.'.format('Bueno', 'Malo',
                                            otro='Feo'))
```

```
El Bueno, el Malo, y el Feo.
```

[16]:
```
dicc = {'Siamés': 5, 'Siberiano': 4, 'Sphynx': 0}
print('Sphynx: {0[Sphynx]:d}; Siamés: {0[Siamés]:d}; Siberiano: {0[Siberiano]:
↪d}'.format(dicc))
```

```
Sphynx: 0; Siamés: 5; Siberiano: 4
```

[17]:
```
print('Sphynx: {Sphynx:d}; Siamés: {Siamés:d}; Siberiano: {Siberiano:d}'.
↪format(**dicc))
```

```
Sphynx: 0; Siamés: 5; Siberiano: 4
```

[18]:
```
for x in range(1, 11):
    print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

```
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

Véase Format String Syntax

Forma antigua de formatear la salida

[19]:
```
import math
print('El valor aproximado de pi es %5.6f.' % math.pi)
```

```
El valor aproximado de pi es 3.141593.
```

1.1.2 Gestión de archivos

La función para gestionar archivos es `open()`. Toma dos parámetros: el nombre del archivo y el modo. Existen cuatro diferentes modos:

- "r" - Read - Default value. Abre el archivo para lectura. Se produce un error si el archvio no existe.
- "a" - Append - Abre el archivo para agregar información. Si el archivo no existe, lo crea.
- "w" - Write - Abre el archivo para escritura. Si el archivo no existe, lo crea. Si el archivo existe, lo sobreescribe.
- "x" - Create - Crea el archivo, regresa un error si el archivo existe.

Adicionalmente se puede especificar si el archivo se abre en modo texto o binario:

- "t" - Text - Valor por omisión.
- "b" - Binary

[20]:
```
f = open('gatos.txt', 'w')
```

[21]:
```
gatos = ['Persa', 'Sphynx', 'Ragdoll','Siamés']
peso_minimo = [2.3, 3.5, 5.4, 2.5]
dicc = dict(zip(gatos,peso_minimo))
```

[22]:
```
for i in dicc:
    f.write('{:>10} {:>10} \n'.format(i,dicc[i]))
```

[23]:
```
f.close()
```

[24]:
```
f = open('gatos.txt','r')
for i in f:
    print(i)
f.close()
```

```
     Persa        2.3

    Sphynx        3.5

   Ragdoll        5.4

    Siamés        2.5
```

Gestores de contexto Permiten asignar y liberar recursos justo en el momento requerido. Se usa mayormente con `with`, veamos un ejemplo:

[25]:
```
with open('contexto_ejemplo', 'w') as archivo_abierto:
    archivo_abierto.write('En este ejemplo, todo se realiza con un gestor de␣
↪contexto.')
```

Lo que realiza la operación anterior es: 1. abre el archivo `contexto_ejemplo`, 2. Escribe algo en el archivo, 3. Cierra el archivo. Si ocurre algún error mientras se escribe en el archivo, entonces se intenta cerrar el archivo. El código es equivalente a:

```
archivo_abierto = open('contexto_ejemplo', 'w')
try:
    archivo_abierto.write('En este ... contexto!')
finally:
    archivo_abierto.close()
```

Es posible implemetar nuestros propios gestores de contexto. Para ello se requiere un conocimiento más profundo de Progrmación Orientada a Objetos.

[ ]:
```

```