Programación en Python
TEMA 2
Formato de salida

Universidad Politécnica de Valencia

2021-2022

Índice

1. Formateando la salida: la función *format*

Aprender a mostrar la información con formato es esencial para que el usuario encuentre atractiva la forma en que ve los resultados. El método format de las cadenas nos proporciona una herramienta fundamental, aunque cuesta un poco dominarlo.

Python tiene 2 maneras de formatear cadenas: (1) con el operador %, y (2) *interpolar* valores usando la función *format*.

En este práctica describimos más detallada como *interpolar* valores en una cadena, es decir, sustituir una marca especial por el valor de una expresión. Fíjate en esta sentencia:

```
>> "El número {0} ha sido interpolado".format(1.23)
El número 1.23 ha sido interpolados
```

Los caracteres 0 han sido reemplazados por los caracteres 1.23.

Podemos interpolar más de un valor con una sola llamada a format:

```
>> "Los números {0} y {1} han sido interpolado".format(1.23,9.9999)
Los números 1.23 y 9.9999 han sido interpolados
```

Cada marca de la forma n , donde n es un número entero, se sustituye por un argumento de format: la marca 0 se ha sustituido por el primer argumento y la marca 1 por el segundo. En general, la marca n se sustituye por el argumento (n+1)-ésimo. Ya te acostumbrarás a un principio básico de Python: todas las secuencias empiezan en cero. (Y no solo de Python: también C, Java y la mayoría de los lenguajes de uso común comparten ese principio).

Las marcas pueden disponerse dentro de la cadena en el orden que desees:

```
>> "Los números {1} y {0} han sido interpolado".format(1.23,9.9999)
Los números 9.9999  y 1.23 han sido interpolados
```

Las marcas pueden modificarse para controlar el aspecto de la información. Imaginemos que deseamos mostrar los valores flotantes redondeados con un solo decimal:

```
>> "Los números {0:.1f} y {1:.1f} han sido interpolado".format(1.23,9.9999)
Los números 1.2 y 10.0 han sido interpolados
```

Donde nos encontramos esta estructura:

```
{[flags][position]:[minimumwidth][.precision][código de tipo]}
```

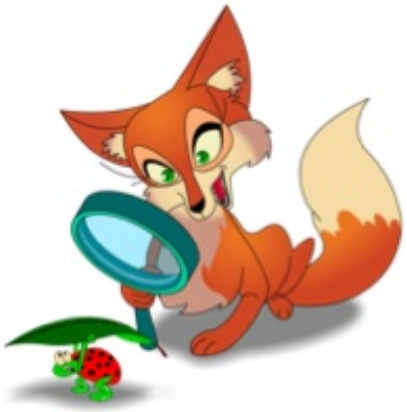
Donde

- poticion: el argumento que hay que sustituir 0 se ha sustituido por el primer argumento y la marca 1 por el segundo, etc
- minimumwidth: el tamaño minimo que puede tener
- .precisión: número de decimales con que queremos representar un número flotante.
- código de tipo: carácter que indica el tipo de representación que se desea. Es diferente según el tipo de datos del valor. He aquí algunos de sus posibles valores:
  * números enteros:
    - carácter b : en binario.
    - carácter c: como carácter Unicode.
    - carácter d: en base diez (es el valor por defecto).
    - carácter o: en octal.
    - carácter x: en hexadecimal.
    - carácter n: igual que d , pero como número adaptado a la cultura local (en español, por ejemplo, el punto decimal se muestra como una coma).
  * números flotantes:
    - carácter e: notación exponente.
    - carácter f: notación de punto fijo.
    - carácter g: notación en formato general, que solo es exponente para números grandes (es el valor por defecto).
    - carácter n: igual que g, pero como número adaptado a la cultura local.
    - carácter %: muestra el número multiplicado por 100, en formato f y seguido de un símbolo de porcentaje.

2. Ejercicios

Ejercicio 1. Mad Libs es un juego de palabras con plantilla de frase donde un jugador pide a otros una lista de palabras para sustituir los espacios en blanco en una historia, antes de leer en voz alta la historia, a menudo cómica o sin sentido. Vamos a hacer un pequeño Mad Libs.

Mira el siguiente ejemplo:



Whether you ..... (verb)
computer programs to
solve ..... (plural noun) or just
for ....., (noun) it is very
important that you
ALWAYS TEST your code
for ..... bugs. (adjective)

Necesitamos pedir al jugador las siguientes palabras en ingles:

- verb, por ejemplo: `write`
- plural noun, por ejemplo: `problems`
- noun, por ejemplo: `fun`
- adjective, por ejemplo: `nasty`

Entonces, para estos ejemplos, nuestro programa devuelve:

```
Whether you write
computer programs to
solve problems or just
for fun, it is very
important that you
ALWAYS TEST your code
for nasty bugs.
```

Prueba con otros inputs y intenta sacar alguna frase graciosa.

Ejercicio 2. Imagina que necesitas testear un programa P que toma dos floats como entrada y produce un booleano como resultado. Un caso de test para probar este programa P consiste de 4 partes:

- un identificador
- dos inputs del tipo float
- una salida esperada del tipo bool
- el resultado del test: ”PASS” or ”FAIL”

Escribe otro programa en Python que primero pide al usuario los siguientes datos:

- el identificador, un integer (`i`)

- los dos inputs, floats (`in1` and `in2`)
- la salida esperada, un bool (`output`)
- el resultado del test, un string (`result`)

Tu programa tiene que generar un string que representa el caso de test usando los datos pedidos. Esta representación se puede usar como una descripción del caso de test. Por ejemplo, si:

```
i = 2

f1 = 123.456

f2 = 12345.67

output = True

result = "PASS"
```

tu programa tiene que producir la representación:

```
TEST_ID_00002 --- inputs: 1.23e+02, 1.23e+04 --- output: True --- result: PASS
```

La primera parte del string se usa para clasificar los casos de test: siempre tiene que empezar con `'test_ID_'` seguido por un identificador de mínimo 5 dígitos. Si `i` tiene menos digitos entonces hay que rellenarlo con ceros a la izquierda.

Los floats se tiene que presentar en formato científico.

Tienes que hacer 2 diferentes implementaciones de tu programa. Uno usando el modulo String operador % para formatear, y otro con el `str.format()`.

ejemplo de ejecución

```
>>> %Run
  Introduce los detalles de tu test case
  identificador (un int): 2
  primer input (float): 123.456
  segundo input (float): 12345.67
  el output esperado (True o False): True
  el resultado del test (PASS o FAIL): PASS

  La representación de tu test case es:
  TEST_ID_00002 --- inputs: 1.23e+02, 1.23e+04 --- output: True --- result: PASS

>>> %Run
  Introduce los detalles de tu test case
  identificador (un int): 3
  primer input (float): 2.8005
  segundo input (float): 341.98301
  el output esperado (True o False): False
  el resultado del test (PASS o FAIL): FAIL

  La representación de tu test case es:
  TEST_ID_00003 --- inputs: 2.80e+00, 3.42e+02 --- output: True --- result: FAIL
>>>
```

Ejercicio 3. Implementar un programa que lea tres valores enteros: día, mes y año de nacimiento de una persona. Utilizando estos datos, el programa debe mostrar como resultado un PIN de cuatro cifras asociado a la fecha de nacimiento. El PIN se calcula como:

1. p1 = (d1 + d2) % 10.
2. p2 = (m1 + m2) % 10.
3. p3 = (a1 + a4) % 10.
4. p4 = (a2 + a3) % 10.

Por ejemplo, si la fecha introducida es: 29 9 1975, el PIN sería 1 9 6 6:

1. p1 = (2 + 9) % 10 = 1.
2. p2 = (0 + 9) % 10 = 9.
3. p3 = (1 + 5) % 10 = 6.
4. p4 = (9 + 7) % 10 = 6.

ejemplo de ejecución

```
>>> %Run
  Introduce tu dia de nacimiento: 29
  Introduce tu mes de nacimiento: 9
  Introduce tu año de nacimiento: 1975
  Tu PIN es 1 9 6 6
```

test case ID	inputs			expected output (PIN)
	dia	mes	año	
1	10	12	101	1 3 1 1
2	1	1	1	1 1 1 0
3	27	3	1978	9 3 9 6
4	55	28	300	0 0 0 3
5	356	903	1568	1 3 9 1

Mira los casos de test 4 y 5. ¿Son validos? Los inputs 55 y 356 no son números validos para un día de nacimiento. Sin embargo, nuestro programa funciona y calcula un PIN. Python no sabe nada de fechas de nacimientos y sus valores validos. Para Python los 3 inputs simplemente son números enteros. Si queríamos que nuestro programa no calcula un PIN cuando la fecha no es valida, entonces deberíamos añadir condiciones en nuestro programa que verifican los inputs. Cómo lo podemos hacer, lo veremos en el siguiente tema con instrucciones de decisión como el `if - then - else`.