05 - Entrada i Sortida

October 5, 2015

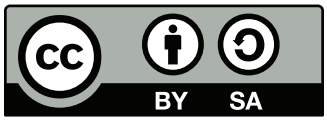

Figure 1: BY-SA

Authors : *Sonia Estradé*
José M. Gómez
Ricardo Graciani
Frank Güell
Manuel López
Xavier Luri
Josep Sabater

Els programes han d'interaccionar amb l'usuari per obtenir les entrades que necessita i retornar els resultats. Per aquest motiu, els diferents llenguatges de programació proporcionen métodes per poder intercanviar informació amb l'usuari. En aquest tema analitzarem les diferents alternatives qeu ens dona Python per intercanviar informació per consola o, en el nostre cas, el ipython notebook.

1 Sortida per pantalla i entrada per teclat

El contingut de les variables, literals o el resultat d'operacions poden ser visualitzats per pantalla amb la funció `print()`. Ens centrarem en la versió de Python 3.x.

Comencem per alguns exemples amb cadenes de text:

```
In [1]: # Definim variables i els hi assignem valors del tipus cadena de text
        string_1 = "Aquest es un text estrany: "
        string_2 = "La vida es tant bonica..."

        # Impressio bàsica i concatenació
        print(string_1)
        print(string_2)
        print('-'*10)
```

```
Aquest es un text estrany:
La vida es tant bonica...
----------
```

També podem mostrar dos o més cadenes de text concatenant-les:

```
In [2]: print('Resultat concatenació: ' + string_1 + string_2)
```

```
Resultat concatenació: Aquest es un text estrany: La vida es tant bonica...
```

O podem indicar un nombre de cadenes de text separades per`,`. En aquest cas també podem indicar el caracter/s separador/s a mostrar per pantalla entre els diferents strings especificats. Per defecte, aquest és l'espai en blanc:

```
In [3]: #On a sep indiquem el caracter separador entre els diferents arguments del print
        print('Resultat sense concat.: ', string_1, string_2, sep='')
```

```
Resultat sense concat.: Aquest es un text estrany: La vida es tant bonica...
```

O també:

```
In [4]: print('Resultat sense concat.: ', string_1, string_2, sep='\n-----\n')
```

```
Resultat sense concat.:
-----
Aquest es un text estrany:
-----
La vida es tant bonica...
```

Podem combinar strings i valors numèrics de dues formes diferents:

1. `print(<text literal>, <valor numèric>)`
2. `contatenant dos strings: print (<text literal> + str(<valor numèric>))`.

En el segon cas, cal convertir prèviament el nombre a string mitjançant `str()`:

```
In [5]: sqrt_2_value = 2**(1/2)

        # Es pot fer utilitzant la següent sintaxi del print ja
        # vista anteriorment.
        # Si no especifiquem separador aquest serà l'espai en blanc.
        print("[0] 2**(1/2)", sqrt_2_value)

        # Podem fer-ho també de la següent manera,
        # però necessitarem fer una conversió de tipus numèric a
        # string amb 'str'
        print("[1] 2**(1/2) " + str(sqrt_2_value))
```

```
[0] 2**(1/2) 1.4142135623730951
[1] 2**(1/2) 1.4142135623730951
```

El valor de $\sqrt{2}$ es mostra amb un nombre de decimals per defecte. Més endavant veurem com modificar el comportament del **print** per tal d'especificar altres opcions.

Podem mostrar també strings previament compostos mitjançant la concatenació:

```
In [6]: x = 10 * 3.25
        y = 200 ** 2
        s = 'El resultat de x és ' + str(x) + ' i y és ' + str(y) + '...'
        print(s)
```

```
El resultat de x és 32.5 i y és 40000...
```

També podem mostrar més d'un valor de diferents tipus per pantalla amb un únic `print()` i sense fer les conversions corresponents. Com podeu veure, és possible combinar variables amb literals.

En aquest cas, aprofitarem per canviar el separador entre els valors a mostrar per `' - '`:

```
In [7]: value_a = 2.5
        value_b = 90
        value_c = "Això es el qeu vols"
        print(value_a, value_b, 'valor de la cadena:', 'es',
              value_c, 15, sep=' - ')
```

```
2.5 - 90 - valor de la cadena: - es - Això es el qeu vols - 15
```

En un string podem incloure una sèrie de caràcters especials. Aquestes es distingueixen perquè comencen pel caràcter \. N'hi ha de diversos entre els quals destaquem el salt de línia `\n` i el tabulador `\t`. Més endavant tornaran a apareixer.

Si directament imprimim el contingut d'una variable string via `print()`, aquest interpretarà els caràcters especials i mostrarà el resultat de la interpretació. En canvi si el que desitjem és mostrar exactament els caràcters que conformen una seqüència de text sense que els caràcters especials siguin interpretats ho podem fer mitjançant la funció `repr()`:

```
In [8]: s = 'Hola, benvolgut: \n\t Alexandre.'
        #Treiem per pantalla la interperetació del car. especial:
        print(s)
        #Treiem per pantalla el contingut literal:
        print(repr(s))
```

```
Hola, benvolgut:
         Alexandre.
'Hola, benvolgut: \n\t Alexandre.'
```

1.1 Donant format a les dades a treure per pantalla

Aquí veurem alguns exemples de com donar format especial al text que ha d'apareixer per pantalla amb el `print()`. En la versió *3.4* de Python es pot fer de dues maneres:

- **Seguint les especificacions del llenguatge C**: de forma similar a com es fa quan s'utilitza la funció `sprintf()` del llenguatge de programació C. Usant el signe **%** seguit de nombres i/o lletres.
- **A la Python**: usant la funció `format()` i usant {}.

Podeu trobar més informació a: www.python-course.eu
O bé a la pàgina de l'especificació de Python 3.4: docs.python.org

1.1.1 Donant format segons l'estil C:

Tot seguit veurem uns exemples de com donar format segons la vella escola, usant l'estil del llenguatge C.

Cal tenir en compte que els tipus de la dada que volem mostrar ve indicada pels següents caràcters: Tot seguit veurem uns exemples de com donar format segons la vella escola.

Cal tenir en compte que els tipus de la dada que volem mostrar ve indicada pels següents caràcters

Tipus	Especificador
Cadena de text	%s
Enter	%d
Real	%f, %e, %g

On el format és el següent:

`<string>%(value_1, value_2,..., value_n)`

Quan la funció `print()` troba dins d'un string algun dels codis mostrats a la taula anterior, cerca el valor corresponent dins del parèntesi. Allí on aparegui la el primer codi, aquest serà reemplaçat pel primer valor, el segon codi pel segon valor, ...

```
In [9]: 'El meu nom es %s, i tinc %d anys, amg una alçada de %.2f m' % \
        ('Joan', 22, 1.75)

Out[9]: 'El meu nom es Joan, i tinc 22 anys, amg una alçada de 1.75 m'
```

El primer codi indicar una cadena, i fa servir la primera cadena, el segon espera un enter i el darrer un float.

Especificador de cadenes de text %s Podem incloure el contingut d'una variable de cadena mitjançant aquest especificador.

De forma ordenada, es mostrarà el resultat per pantalla, substituint cadascun d'aquests especificadors per la variable o literal de tipus string corresponent que apareix a dins dels parentesi de la dreta.

Veiem-ne un exemple:

```
In [10]: Text_1 = "Lorem"
         Text_2 = "Ipsum"

         print("El primer text és %s i el segón és %s" % (Text_1,Text_2))

El primer text és Lorem i el segón és Ipsum

In [11]: Number_1 = 3
         Number_2 = 2**(1/2)

         print("El primer text és %s i el segón és %s" % (Number_1, Number_2))

El primer text és 3 i el segón és 1.4142135623730951
```

Especificador d'enters %d Permet imprimir el valor d'un enter. Es pot controlar el nombre de dígits, espais i si cal incloure zeros al davant.

La forma més senzilla és:

```
In [12]: Var_1 = 12
         Var_2 = 1234
         Var_3 = 1234567890

         # Sense format
         print("Var_1 =", Var_1)
         print("Var_2 =", Var_2)
         print("Var_3 =", Var_3)

Var_1 = 12
Var_2 = 1234
Var_3 = 1234567890
```

Una altra opció és aliniar a la dreta el valor especificant el nombre de digits total a mostrar per pantalla. En aquest cas, el nombre de dígits s'ha d'indicar entre els caracters `%` i `d`. En el següent exemple usarem com minim 8 dígits:

```
In [13]: # Format fixant la longitud: Davant del %d especifiquem
         # el nombre de caracters amb els qual s'ha de mostrar
         print("Var_1 = %8d" % Var_1)
         print("Var_2 = %8d" % Var_2)
         print("Var_3 = %8d" % Var_3)

Var_1 =       12
Var_2 =     1234
Var_3 = 1234567890
```

En els primers dos casos els nombres s'alinien de forma correcta ja que els valors corresponents a mostrar contenen menys de 8 dígits. En el tercer, el valor del nombre consta de 10 dígits. En aquest cas especial, per tal de no perdre precisió, no es té en compte l'aliniació.

També és possible inserir zeros a l'esquerra d'un valor en comptes d'espais en blanc. Aquest vegada cal inserir un 0 just abans de %, tot seguit el nombre total de dígit a mostra i finalment d.

```
In [14]: # Format fixant la longitud i completant amb zeros
         print("Var_1 = %010d" % Var_1)
         print("Var_2 = %010d" % Var_2)
         print("Var_3 = %010d" % Var_3)

Var_1 = 0000000012
Var_2 = 0000001234
Var_3 = 1234567890
```

En aquest cas el nombre de dígits es 10 i a les hores tots el nombres estan correctament aliniats.

Especificador de reals %f, %e i %g Permet imprimir un real. Es pot controlar el número de dígits davant i darrera la coma i si cal incloure zeros al davant.

La forma més senzilla és:

```
In [15]: sqrt_2 = 2**(1/2)

         print('Valor de 2**(1/2) es %f' % (sqrt_2))

Valor de 2**(1/2) es 1.414214
```

I també podem controlar el nombre de digits (abans i després del punt) a mostrar per pantalla, així com especificar el caracter de signe:

```
In [16]: print("Valor de 2**(1/2) es %+15.10f" % (sqrt_2))
         print("Valor de 2**(1/2) es %15.4f" % (sqrt_2))

Valor de 2**(1/2) es   +1.4142135624
Valor de 2**(1/2) es          1.4142
```

En el primer cas, la aliniació a la dreta té en compte tots els digits, incloent el punt i el signe.

El format **%e** mostra sempre l'exponent, tot seguint les mateixes regles pel que fa a nombre de decimals i aliniació:

```
In [17]: avogadro = 6.023e23

         print("Un altre valor es %+15.4e" % (avogadro))
         print("Un altra opció es %+15.4g" % (avogadro))
         print("Un altre cas   %+15.4g" % (sqrt_2))
```

```
Un altre valor es     +6.0230e+23
Un altra opció es      +6.023e+23
Un altre cas            +1.414
```

El comportament del format `%g` és similar al del `%e` si l'exponent és menor de -4 o major que la precissió, altrament es comporta com `%f`. Podem apreciar com el nombre de dígits el en cas de `%g` es correspon amb la precisió.

Altres formats: octal (%o) i hexadecimal (%x) Permeten escriure valors enters en format octal o hexadecimal

```
In [18]: Val_1 = 2**8
         Val_2 = 2**10
         Val_3 = 242942

         print("[Val_1] enter:%d octal:%o  hexa:0x%x" % (Val_1, Val_1, Val_1))
         print("[Val_2] enter:%d octal:%o  hexa:0x%x" % (Val_2, Val_2, Val_2))
         print("[Val_3] enter:%d octal:%o  hexa:0x%x" % (Val_3, Val_3, Val_3))
```

```
[Val_1] enter:256 octal:400  hexa:0x100
[Val_2] enter:1024 octal:2000  hexa:0x400
[Val_3] enter:242942 octal:732376  hexa:0x3b4fe
```

Caràcters especials Es poden imprimir o incloure en cadenes de text alguns caràcters especials. Els més usats, com ja hem vist en alguns exemples anteriors, són:

Descripció	Caràcter especial
Salt de línia	\n
Tabulador	\t
	\
Per Cent	%

```
In [19]: print("Això és un text\n\tpartit en dues línies\n\n")
         print("Tab0\tTab1\tTab2")
         print("A\tB\tC")
```

```
Això és un text
        partit en dues línies


Tab0        Tab1        Tab2
A        B        C
```

I ara un exemple on combinem diferents tipus de dades:

```
In [20]: # Un altre exemple en el que combinen tot
         v1 = 100
         v2 = 123456789.123456789
         s = '-'*10 + "\n"

         print("v1:\n\tdec (ent) - %d\n\tdec (real) - %f\n\thexa - 0x%X\n%s" %(v1, v1, v1, s))
         print("Limitant el nombre de dígits als decimals d'un float: %.3f\n" %v2)
```

```
        print("Afegint zeros a dreta si no arribem a num xifres determinat: %.5f\n" % 123.123)
        print("Imprimint un real com un sencer: %d\n" % v2)
        print("Afegint zeros a esquerra si no arribem a num. xifres deter.: %05d\n" % 123)

v1:
        dec (ent) - 100
        dec (real) - 100.000000
        hexa - 0x64
----------

Limitant el nombre de dígits als decimals d'un float: 123456789.123

Afegint zeros a dreta si no arribem a num xifres determinat: 123.12300

Imprimint un real com un sencer: 123456789

Afegint zeros a esquerra si no arribem a num. xifres deter.: 00123
```

1.1.2 Donant format a la Python:

Especifiquem un string dins del qual apareixen parelles de {} i a continuació la funció `format()` dins de la qual adjuntem tants valors (o variables) com conjunts {} tinguem a l'string.

A l'hora de mostrat per pantalla dins de l'string cadascun de les parelles {} es substitueix, per ordre, amb els valors dels objectes de dins del parentesi que acompanya al format.

Cal tenir en compte que, a l'igual que donant format seguint la vella escola, els tipus de la dada que volem mostrar ve indicada pels següents caràcters:

Tipus	Especificador
Cadena de text	s
Enter	d
Real	f, e, g
Locale	n

```
In [21]: nom = 'Pere'
         v = 10.0

         # Deixem que format determini automàticament el tipus dels
         # valors a mostrar.
         s = "Lorem Ipsum:\n\t1r val: {}\n\t2n val: {}\n\t3r val: {}".format(nom, v, 12+12)
         print(s)

Lorem Ipsum:
        1r val: Pere
        2n val: 10.0
        3r val: 24
```

In this case, the data is shown the same way as using the `str()` function.

Podem alterar l'ordre d'aparició del valors a mostrar, tot indicant explicitament dins del **{}** l'index del valor que volem que aparegui:

```
In [22]: nom = 'Pere'
         separa = '-'*10 + "\n"
```

```
        v = 10.0

        s = "Lorem Ipsum:\n\t1r val: {1}\n\t2n val: {0}\n\t3r val: {2}".format(nom, v, 12+12, 35)
        print(s)
        print(separa)

        print('{} and {}'.format('dogs', 'cats'))
        print('{0} and {1}'.format('dogs', 'cats'))
        print('{1} and {0}'.format('dogs', 'cats'))
```

```
Lorem Ipsum:
        1r val: 10.0
        2n val: Pere
        3r val: 24
----------

dogs and cats
dogs and cats
cats and dogs
```

Podem usar paraules clau dins del **format()** per tal de referir-nos als diferents valors:

```
In [23]: print('En {person} és {adjective}.'.format(person='Pau', adjective='molt divertit'))
```

```
En Pau és molt divertit.
```

Podem expressar valors en forma de taula:

```
In [24]: print('{0:2d} {1:10d} {2:3d}'.format(1, 1024, 2389))
         print('{0:2d} {1:10d} {2:3d}'.format(12, 1234567890, 123))
```

```
1       1024 2389
12 1234567890 123
```

El nombre que apareix entre claus i abans dels dos punts indica la posició del valor que es troba dins dels parèntesi que és mostrarà en cada cas. El que hi ha darrera dels dos punts indica el comportament de forma similar a l'estil C.

El darrer exemple de en **estil C**, el podem reproduir ara tot seguint el format de **a la Python**. Fixeu-vos que podem especificar dins de les parelles **{}** el format específic amb el qual volem que surti per pantalla el valor corresponent:

```
In [25]: v1 = 100
         v2 = 123456789.123456789
         s = '-'*10 + "\n"

         # Composem els strings als quals volem donar format
         str_format_1 = "v1:\n\tdec (ent) - {0:d}\n\tdec (real) - {1:f}\n\thexa - 0x{2:X}\n{3}"
         str_format_2 = "Limitant el nombre de dígits als decimals d'un float: {0:.3f}\n"
         str_format_3 = "Afegint zeros a dreta si no arribem a num xifres determinat: {0:.5f}\n"
         str_format_4 = "Imprimint un real com un sencer: {0}\n"
         str_format_5 = "Afegint zeros a esquerra si no arribem a num. xifres deter.: {0:05d}\n"

         #A pliquem format i treiem per pantalla
         print(str_format_1.format(v1, v1, v1, s))
         print(str_format_2.format(v2))
         print(str_format_3.format(123.123))
         print(str_format_4.format(int(v2)))
         print(str_format_5.format(123))
```

```
v1:
        dec (ent) - 100
        dec (real) - 100.000000
        hexa - 0x64
----------

Limitant el nombre de dígits als decimals d'un float: 123456789.123

Afegint zeros a dreta si no arribem a num xifres determinat: 123.12300

Imprimint un real com un sencer: 123456789

Afegint zeros a esquerra si no arribem a num. xifres deter.: 00123
```

Finalment, introduirem un nou component: el **locale**.

Primerament cal importar el que hi ha configurat per defecte (en un futur veurem l'ús específic de la paraula **import**):

```
In [26]: import locale

         locale.setlocale(locale.LC_ALL, 'ES_es')
         locale.localeconv()

Out[26]: {'currency_symbol': 'Eu',
          'decimal_point': ',',
          'frac_digits': 2,
          'grouping': [127],
          'int_curr_symbol': 'EUR ',
          'int_frac_digits': 2,
          'mon_decimal_point': ',',
          'mon_grouping': [3, 3, 0],
          'mon_thousands_sep': '.',
          'n_cs_precedes': 0,
          'n_sep_by_space': 1,
          'n_sign_posn': 1,
          'negative_sign': '-',
          'p_cs_precedes': 0,
          'p_sep_by_space': 1,
          'p_sign_posn': 1,
          'positive_sign': '',
          'thousands_sep': ''}

In [27]: help(locale.setlocale)

Help on function setlocale in module locale:

setlocale(category, locale=None)
    Set the locale for the given category.  The locale can be
    a string, an iterable of two strings (language code and encoding),
    or None.

    Iterables are converted to strings using the locale aliasing
    engine.  Locale strings are passed directly to the C lib.

    category may be given as one of the LC_* values.
```

```
In [28]: print("{:n}".format(v2))
```

```
1,23457e+08
```

En aquest cas, el separador decimal és **‘,’** (veure camp `decimal_point`) ja que hem definit que vollem treballar amb la cofiguració espanyola.

1.2 Entrada per teclat

- `input():` retorna una **cadena de text** introduïda per l’usuari a través del teclat (o consola).

```
In [29]: nom = input("Introdueix el teu nom: ")
         print('Hola {}!'.format(nom) )
```

```
Introdueix el teu nom: Joan
Hola Joan!
```

Podem demanar més dades cridant `input():` diversas vegadas:

```
In [30]: nom = input("Com et dius? ")
         edat = input("Quants anys tens? ")
         pes = input("Quant peses? ")

         print(type(nom), type(edat), type(pes))

         print("Et dius {} tens {} anys i peses {}kg.".format(nom, edat, pes))
```

```
Com et dius? Joan
Quants anys tens? 21
Quant peses? 80
<class 'str'> <class 'str'> <class 'str'>
Et dius Joan tens 21 anys i peses 80kg.
```

Si volem obtenir l’edat i el pes com a valors numèrics (fixeu-vos que són de tipus cadena de text), cal convertir-los.

Aixo el podem fer de dues maneras. Les funcions `int()` o `float()` retornen el valor en format enter o real contingut en una cadena. En combinació amb `input()` permeten llegir dades numèriques introduïdes per l’usuari a través del teclat.

```
In [31]: s_oper1 =  input("Introdueix primer operant:")
         s_oper2 =  input("Introdueix segon operant:")
         text_oper1 = "Has entrat el nombre {} i és del tipus {}".format(s_oper1, type(s_oper1))
         text_oper2 = "Has entrat el nombre {} i és del tipus {}".format(s_oper2, type(s_oper2))
         separa = '-'*10 + "\n"

         print(separa+text_oper1+"\n"+text_oper2)

         # Abans de fer l'operació numèrica cal convertir els strings a floats
         f_oper1 = float(s_oper1)
         f_oper2 = float(s_oper2)
         print('El tipus de f_oper1 és {} i el de f_oper1 és {}'.format(type(f_oper1), type(f_oper2)))
         print(separa, f_oper1, '+', f_oper2, '=', f_oper1+f_oper2)
```

```
Introdueix primer operant:1.3
Introdueix segon operant:2.5
----------
Has entrat el nombre 1.3 i és del tipus <class 'str'>
Has entrat el nombre 2.5 i és del tipus <class 'str'>
El tipus de f_oper1 és <class 'float'> i el de f_oper1 és <class 'float'>
----------
 1.3 + 2.5 = 3.8
```

Podem també efectuar directament la conversió:

```
In [32]: # Conversió numèrica immediata

         x = int(input("Entra un nombre: "))
         y = int(input("Entra un segon nombre: "))
         print('La suma de ', x, ' i ', y, ' és ', x+y, '.', sep='')
```

```
Entra un nombre: 3
Entra un segon nombre: 4
La suma de 3 i 4 és 7.
```

Una altra opció és emprar el mòdul **locale** prèviament introduït. D'aquesta manera Python tindrà en compte el separador decimal amb el qual treballi el nostre sistema: **'.'** o **','**. És a dir, si el nostre sistema funciona en català o en espanyol, podrem introduir valors reals on aparegui una **','**.

En aquest cas farem servir les funcions `locale.atoi()` i `locale.atof()` segons es tracti d'un sencer o bé un valor real:

```
In [33]: import locale

         locale.setlocale(locale.LC_ALL, 'ES_es')

         s_oper1 =  input("Escriu el primer operand:")
         s_oper2 =  input("Escriu el segon operand:")
         text_oper1 = "El primer operand es {}"\
                         " i el seu tipus es {}".format(s_oper1,
                                                        type(s_oper1))
         text_oper2 = "El segon operand es {}"\
                         " i el seu tipus es {}".format(s_oper2,
                                                       type(s_oper2))
         separator = '-'*10 + "\n"

         print(separator+text_oper1+"\n"+text_oper2)

         f_oper1 = locale.atof(s_oper1)
         f_oper2 = locale.atof(s_oper2)

         print("El tipus de f_oper1 es {}"\
               " i el de f_oper1 es {}".format(type(f_oper1),
                                               type(f_oper2)))
         print(separator, "{:n} + {:n} = {:n}".format(f_oper1,
                                                      f_oper2,
                                                      f_oper1+f_oper2))
```

```
Escriu el primer operand:1,4
Escriu el segon operand:2,6
```

```
----------
El primer operand es 1,4 i el seu tipus es <class 'str'>
El segon operand es 2,6 i el seu tipus es <class 'str'>
El tipus de f_oper1 es <class 'float'> i el de f_oper1 es <class 'float'>
----------
 1,4 + 2,6 = 4
```

I ara amb enters:

```
In [34]: x = locale.atoi(input("Escriu un enter: "))
         y = locale.atoi(input("Escriu un segon enter: "))
         print('Sumant {:n} i {:n} = {:n}'.format(x, y, x+y))

Escriu un enter: 3
Escriu un segon enter: 5
Sumant 3 i 5 = 8
```

En aquest darrer exemple, el resultat és sempre el mateix ja que no s'usa el separador decimal, però aquesta solució mitjançant el locale ens permet que el nostre codi sigui portable a qualsevol sistema.