Guía 2: Entrada / Salida y Strings

Profesor: John Serrano - john.serrano@usach.cl

1 Entradas en Python

En la guía anterior se mencionó como un extra, pero ahora ya es importante conocer las entradas en Python. Una entrada (input) es una forma en la que el usuario puede proporcionar datos mediante el teclado durante la ejecución de un programa. Es decir, un programa podría solicitarle a un usuario que ingrese su nombre y esto es posible gracias a la función **input()** en Python. Como ya conocimos el concepto de **variable** en Python, podemos (y siempre es recomendable!) guardar la entrada del usuario en una palabra cuyo nombre sea representativo de lo que se ingresa.

Dentro de los paréntesis de **input()** podemos colocar un mensaje en formato String para indicarle al usuario que es lo que se espera que el usuario ingrese. Si bien **input()** funcionará de igual forma si es que no colocamos ningún mensaje dentro de los paréntesis, no es recomendable, dado que sin alguna indicación, el usuario no sabrá qué es lo que debe ingresar, lo que podría llevar a errores dentro del programa.

[1]:
```
# Se solicita el nombre del usuario
nombre = input("Ingrese su nombre: ")
# Se imprime el nombre ingresado.
print("Tu nombre es:", nombre)
```

```
Ingrese su nombre:  John
```

```
Tu nombre es: John
```

NOTA: Siempre es recomendable dejar un espacio luego de los dos puntos y ahí colocar las comillas de cierre. El motivo es porque de lo contrario, el nombre a ingresar estará inmediatamente pegado de los dos puntos, lo cual no es muy estéticamente agradable.

Para cada dato que se necesite que el usuario ingrese, se debe llamar a la función **input()**.

[2]:
```
# Se solicita el nombre del usuario
nombre = input("Ingrese su nombre: ")
# Se solicita la edad del usuario
edad = input("Ingrese su edad: ")
# Se solicita la fecha de nacimiento del usuario (13/10/2000)
fecha_nacimiento = input("Ingrese su fecha de nacimiento en formato dd/mm/aaaa:␣
↪")
# Se imprime el mensaje que utiliza las entradas ingresadas.
print("Tu nombre es", nombre, "tienes", edad, "anios y naciste el",␣
↪fecha_nacimiento + ".")
```

```
Ingrese su nombre:  John
Ingrese su edad:  23
Ingrese su fecha de nacimiento en formato dd/mm/aaaa:  13/10/2000
```

```
Tu nombre es John tienes 23 anios y naciste el 13/10/2000.
```

NOTA: Siempre se debe evitar el uso de la ñ y tildes en Python a menos que sea solicitado por el ejercicio.

NOTA 2: Más adelante se explica nuevamente print() y porque a veces se utiliza , y otras veces se usa + para los mensajes.

¿Qué pasa si un usuario es porfiado y no ingresa lo solicitado? En algunos casos, eso podría provocar un resultado incorrecto, inesperado o incluso un error total del programa. En las próximas clases (y guías) ya iremos viendo formas de ver esos casos específicos. Por ahora, lo que si podemos hacer es comprobar y modificar el tipo de dato de la entrada. Por ejemplo, quizás se requiere que el tipo de dato de la edad sea de tipo int, ya que es un número entero, pero si utilizamos la función **type()** de la clase / guía 1, veremos que no es así.

[3]:
```
# Se comprueba el tipo de dato de edad
print(type(edad))
```

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

Lo anterior se debe a que todo lo ingresado mediante **input()** es transformado a un String, independiente incluso del formato que se utilice para la entrada. Pero si utilizamos las funciones de transformación vistas en la clase anterior, podemos transformar correctamente el tipo de dato de la entrada. Por ejemplo, para el caso de la edad:

[4]:
```
# Se solicita la edad del usuario. Se transforma automáticamente a int.
edad = int(input("Ingrese su edad: "))
print(type(edad))
```

```
Ingrese su edad:  23
```

```
<class 'int'>
```

Las mismas reglas de transformaciones de datos aplican y nuevamente, hay que tener cuidada, ya que por ejemplo, si se ingresa un caracter o elemento que no puede ser transformado a int, el programa lanzará un error.

[5]:
```
# Se solicita la edad del usuario. Se transforma automáticamente a int.
edad = int(input("Ingrese su edad: "))
```

```
Ingrese su edad:  H
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[5], line 2
      1 # Se solicita la edad del usuario. Se transforma automáticamente a int.
----> 2 edad = int(input("Ingrese su edad: "))

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

2 Salidas en Python

Ya conocimos a la función **print()** (imprimir), la cual muestra un mensaje por consola. Por lo tanto, dentro de los paréntesis debe estar el mensaje que se desea mostrar al usuario.

[6]:
```
# Se imprime el mensaje "Hola Mundo!". Nótese que al imprimir Strings, Python no␣
↪muestra las comillas del string.
print("Hola Mundo!")
```

```
Hola Mundo!
```

[7]:
```
# Se imprime el numero int 2.
print(2)
```

```
2
```

[8]:
```
# Se imprime el numero float -323232.8888
print(-323232.8888)
```

```
-323232.8888
```

Si tenemos variables, también podemos imprimirlas. Incluso, podemos juntar nuestras variables con un mensaje, como se vio en el uso de prints más arriba. Hay que tener dos cosas en consideración:

- La separación de elementos se hace con una **coma (,)**, lo cual provoca que automáticamente se agregue un espacio luego del elemento antes de la coma.
- También se puede utilizar el símbolo **más (+)** si necesitamos que dos elementos estén juntos sin un espacio entre medio.

Lo anterior se puede ver en el uso de **print()** en el segundo bloque de código más arriba.

Algo importante a considerar es que print() siempre imprime en una nueva línea, pero si queremos tener múltiples prints que impriman en una sola línea, debemos incluir **, end** = ” “ luego de colocar el texto a imprimir, dentro de los paréntesis).

[9]:
```
print("Esto es un print", end = " ")
print("en multiples", end = " ")
print("lineas.")
```

```
Esto es un print en multiples lineas.
```

3 Strings en Python

Los Strings corresponden a caracteres o cadenas de textos. En palabras simples, son letras o palabras. Se escriben utilizando comillas dobles o comillas simples (NO PUEDE SER UNA COMBINACIÓN DE AMBAS!). Sus elementos se cuentan desde 0 hasta **n-1** elementos y podemos obtener el largo del string utilizando la función **len(nombre_string)**, donde **nombre_string** es el nombre de la variable que contiene al string.

NOTA: En la computación siempre se cuenta desde el 0, NO desde el 1 como estamos acostumbrados.

String	"	h	o	l	a	"
Posicion		0	1	2	3	

Por lo tanto y del ejemplo anterior, h es el primer elemento cuya posición es 0, o es el elemento de la posición 1, l es el elemento de la posición 2 y a es el elemento de la posición 3. El String "hola" tiene 4 elementos. Si hay un espacio entre medio de los elementos y dentro de las comillas, también se cuenta como otro elemento más.

Podemos utilizar la suma para unir dos strings. Los strings se unen exactamente en la última y primera posición del primer y segundo string respectivamente.

[10]:
```
string1 = "hola "                          # Se crea el string "hola "
print("El primer string es: ", string1)
string2 = "mundo"                          # Se crea el string "mundo"
print("El segundo string es: ", string2)
union = string1 + string2                  # Se crea la union de ambos strings␣
 ↪("hola mundo")
print("La union de ambos strings es: ", union)
```

```
El primer string es:  hola
El segundo string es:  mundo
La union de ambos strings es:  hola mundo
```

NOTA: en el primer String el último elemento es un espacio para que así haya una separación entre hola y mundo. De lo contrario, el resultado hubiera sido holamundo (Sin el espacio entremedio)

Utilizando el simbolo de la multiplicación (*), podemos repetir un string x veces.

[11]:
```
string1 = "hola!"                  # Se crea el string "hola!"
string2 = string1 * 5              # Se repite el string "hola!" 5 veces.
print("El string original es: ", string1)
print("El string repetido 5 veces es: ", string2)
```

```
El string original es:  hola!
El string repetido 5 veces es:  hola!hola!hola!hola!hola!
```

Podemos acceder a un elemento especifico de un string utilizando la forma **nombre_string[x]**, donde **nombre_string** es el nombre de la variable que contiene al string y **x** es la posición del elemento al que se quiere acceder.

[12]:
```
string1 = "hola mundo"             # Se crea el string "hola mundo"
elemento = string1[5]              # Se accede al elemento de la posicion 5 del␣
 ↪string ("m")
```

```
print("El string original es: ", string1)
print("El elemento en la posicion 5 del string es: ", elemento)
```

```
El string original es:  hola mundo
El elemento en la posicion 5 del string es:  m
```

Un String también corresponde a un objeto. Un objeto nos permite ejecutar métodos, los cuales funcionan similar a una función, solo que están relacionados directamente con esa variable y no es necesario redefinirla. Los métodos funcionan de la siguiente manera:

objeto.metodo(algo), donde objeto puede corresponder a un String o una lista, metodo es algún método disponible para ese objeto (ya vienen los de los strings) y algo es lo que recibe el método para funcionar. A veces no es necesario colocar algo dentro de los paréntesis.

Usando el método **nombre_string.lower()** podemos transformar una entrada, variable o cualquier string en minúsculas.

[13]:
```
# Se define una variable con un string
string = "PYTHONIA"
# Se imprime el string usando el metodo lower() para transformar todo a␣
 ↪minusculas.
print(string.lower())
```

```
pythonia
```

Usando el método **nombre_string.upper()** podemos transformar una entrada, variable o cualquier string en minúsculas. Nótese que tanto lower() como upper() pueden ser utilizados al final de un input.

[14]:
```
mensaje = input("Ingrese un mensaje: ").upper()
print(mensaje)
```

```
Ingrese un mensaje:  los alumnos de fpi a-2 aprobaran si revisan todas las guias

LOS ALUMNOS DE FPI A-2 APROBARAN SI REVISAN TODAS LAS GUIAS
```

También existen:

- **nombre_string.islower()**: Devuelve True si todas las letras del String son en minúsculas.
- **nombre_string.isupper()**: Devuelve True si todas las letras del String son en mayusculas.
- **nombre_string.isdigit()**: Devuelve True si el string está compuesto solo por números y no hay espacios.
- **nombre_string.isalpha()**: Devuelve True si el string está compuesto solo por letras del abecedario y no hay espacios.

[15]:
```
mensaje = input("Ingrese un mensaje: ").islower()
print(mensaje)
```

```
Ingrese un mensaje:  esto es un mensaje

True
```

[16]:
```
mensaje = input("Ingrese un mensaje: ").isdigit()
print(mensaje)
```

```
Ingrese un mensaje:  011001100111000001101001

True
```

[17]:
```
mensaje = input("Ingrese un mensaje: ").isalpha()
print(mensaje)
```

```
Ingrese un mensaje:  FPI

True
```

Podemos utilizar el método **nombre_string.strip()** para eliminar los saltos de línea (****) o los espacios al inicio y al final de un string.

[18]:
```
string1 = "    USACH  en  Paro\n"
print("El string original es: ", string1)
string2 = string1.strip()
print("El string modificado con strip es:", string2)
```

```
El string original es:      USACH  en  Paro

El string modificado con strip es: USACH  en  Paro
```

Podemos utilizar el método **nombre_string.count(elemento)** para contar cuantas veces se repite un elemento dentro de un string.

[19]:
```
string1 = "esto es un texto en formato string"
# Se imprime cuantas veces se cuenta la letra e.
print(string1.count("e"))
```

```
4
```