2.3 Entrada/Salida

- 2.1 Abstracción de problemas para su programación. Conceptos fundamentales
- 2.2 Variables, expresiones, asignación
- 2.3 Uso de entrada/salida por consola
- 2.4 Manejo de estructuras básicas de control de flujo: secuencial, alternativa y repetitiva
- 2.5 Definición y uso de subprogramas y funciones. Ámbito de variables
- 2.6 Entrada/salida a ficheros
- 2.7 Tipos y estructuras de datos básicas: arrays

Entrada estándar

Para introducir datos por la entrada estándar (teclado) usaremos la función ***input***

- Devuelve una cadena con los caracteres introducidos por el usuario mediante el teclado.

```
entrada= input()
```

- Permite sacar un mensaje indicando al usuario lo que debe introducir.

```
entrada= input("Dame tu nombre:")
```

- La cadena leída se puede convertir a otro tipo de dato.

```
entero =  int     (input("Dame un entero:"))
Flotante =  float  (input("Dame un real:"))
```

Pueden producir un error de conversión

Salida estándar

Para mostrar datos por la salida estándar (pantalla) usaremos la función ***print,*** que tiene el formato:

print(expr1, expr2, ..., sep=" ", end="\n")

- *expr :* puede ser de tipo cadena, entero, flotante o booleano, una variable o cualquier expresión de Python
- *sep, end :* son opcionales (pueden no darse, ya que toman valores por defecto)

```
A=30
print()
print("Hola mundo")
print(25)
print(3.14)
print(2+2)
print(5*A)
```

```
Hola mundo
25
3.14
4
150
```

- Las comillas no se muestran en la salida
- Por defecto se introduce un salto de línea al final → end="\n"

Salida estándar

- Si no se quiere introducir un salto de línea, hay que dar un valor al argumento 'end'.
- Para imprimir en la misma línea es habitual utilizar el valor end=" ".

```
print("Adios",end=" ")
print("mundo",end=" ")
print("cruel")
print("Es broma!")
```

```
Adios mundo cruel
Es broma!
```

Salida estándar

- Pueden imprimirse varios valores con la misma función ***print*** (separados mediante comas)

```
print("Te costará",30,"euros")
```

```
Te costará 30 euros
```

- En la salida las comas se sustituyen por el separador por defecto, un espacio en blanco → sep=“ “

- Si queremos cambiar el separador, hay que dar un valor al argumento ‘sep’.

```
print(3,23,48,sep=" < ")
print(192,168,178,42,sep=".")
```

```
3 < 23 < 48
192.168.178.42
```

Salida estándar

- Las cadenas de caracteres del ***print*** pueden contener caracteres especiales → precedidos de \

```
print("\nTe costará\t", 30, "euros")
```

```
Te costará    30 euros
```

\n: introduce un salto de línea
\t: introduce una tabulación

Salida estándar

Los strings permiten técnicas avanzadas de formato mediante la función `.format()` y campos de reemplazo:

- `{}`: reemplaza el campo por el parámetro en `.format()`, secuencialmente.
- `{i}`: reemplaza el campo por el iº parámetro en `.format()`.
- `{i:.mf}`: reemplaza el campo por el iº parámetro en `.format()` como un valor decimal con m posiciones decimales.

Admite muchas otras posibilidades: hexadecimal, binario, notación científica, tabulación…

```
print("Tengo {} años".format(25))
```

```
Tengo 25 años
```

```
print("Tengo {0} años, mido {1:.2f} metros de
altura y soy de {2}.".format(25, 1.8, "Tehran"))
```

```
Tengo 25 años, mido 1.80 metros de altura, y soy de
Tehran.
```

Salida estándar

Alternativamente, Python 3 incluye f-strings.

- Sintaxis similar a `.format()`, pero los strings se definen con una `f` delante.
- Básicamente, cualquier campo `{}` se interpreta en tiempo de ejecución.
- Puede formatear valores como float al igual que en `.format()`.

```
print(f"Tengo {6 * 5} años")
```

```
Tengo 30 años
```

```
age = 25
height = 1.8
birth_place = "Tehran"
print(f"Tengo {age} años, mido {height:.2f} metros
de altura y soy de {birth_place}.")
```

```
Tengo 25 años, mido 1.80 metros de altura y soy de
Tehran.
```