# Mi primer programa en python3

## Uso del intérprete

Al instalar python3 el ejecutable del intérprete lo podemos encontrar en /usr/bin/python3. Este directorio por defecto está en el PATH, por lo tanto lo podemos ejecutar directamente en el terminal. Por lo tanto para entrar en el modo interactivo, donde podemos ejecutar instrucción por instrucción interactivamente, ejecutamos:

```
$ python3
Python 3.4.2 (default, Oct  8 2014, 10:45:20)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

En el modo interactivo, la última expresión impresa es asignada a la variable _.

```
>>> 4 +3
7
>>> 3 + _
10
```

Si tenemos nuestro programa en un fichero fuente (suele tener extensión py), por ejemplo programa.py,lo ejecutaremos de la siguiente manera.

```
$ python3 programa.py
```

Por defecto la codificación de nuestro código fuente es UTF-8, por lo que no debemos tener ningún problema con los caracteres utilizados en nuestro programa. Si por cualquier motivo necesitamos cambiar la codificación de los caracteres, en la primera línea de nuestro programa necesitamos poner:

```
# -*- coding: encoding -
```

Por ejemplo:

```
# -*- coding: cp-1252 -*-
```

# Escribimos un programa

Un ejemplo de nuestro primer programa, podría ser este "hola mundo" un poco modificado:

```
numero = 5
if numero == 5:
        print ("Hola mundo!!!")
```

La indentación de la última línea es importante (se puede hacer con espacios o con tabulador), en python se utiliza para indicar bloques de instrucciones definidas por las estructuras de control (if, while, for, ...).

Para ejecutar este programa (guardado en hola.py):

```
$ python3 hola.py
$ Hola mundo!!!
```

# Ejecución de programas usando shebang

Podemos ejecutar directamente el fichero utilizando en la primera línea el shebang, donde se indica el ejecutable que vamos a utilizar.

```
#!/usr/bin/python3
```

También podemos usar el programa env para preguntar al sistema por la ruta el intérprete de python:

```
#!/usr/bin/env python
```

Por supuesto tenemos que dar permisos de ejecución al fichero.

```
$ chmod +x hola.py
```

```
$ ./hola.py
$ Hola mundo!!!
```

# Guía de estilo

Puede encontrar la guía de estilos para escribir código python en Style Guide for Python Code.

# Estructura del programa

- Un programa python está formado por instrucciones que acaban en un carácter de "salto de línea".
- El punto y coma “;” se puede usar para separar varias sentencias en una misma línea, pero no se aconseja su uso.
- Una línea empieza en la primera posición, si tenemos instrucciones dentro de un bloque de una estructura de control de flujo habrá que hacer una identación.
- La identación se puede hacer con espacios y tabulaciones pero ambos tipos no se pueden mezclar. Se recomienda usar 4 espacios.
- La barra invertida "\" al final de línea se emplea para dividir una línea muy larga en dos o más líneas.
- Las expresiones entre paréntesis "()", llaves "{}" y corchetes "[]" separadas por comas "," se pueden escribir ocupando varias líneas.
- Cuando el bloque a sangrar sólo ocupa una línea ésta puede escribirse después de los dos puntos.

## Comentarios

Se utiliza el carácter # para indicar los comentarios.

## Palabras reservadas

| | | | | |
|---|---|---|---|---|
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass | |
| break | except | in | raise | |

# Funciones y constantes predefinidas

## Funciones predefinidas

Tenemos una serie de funciones predefinidas en python3:

| | | | | |
|---|---|---|---|---|
| abs() | dict() | help() | min() | setattr() |
| all() | dir() | hex() | next() | slice() |
| any() | divmod() | id() | object() | sorted() |
| ascii() | enumerate() | input() | oct() | staticmethod() |
| bin() | eval() | int() | open() | str() |
| bool() | exec() | isinstance() | ord() | sum() |
| bytearray() | filter() | issubclass() | pow() | super() |
| bytes() | float() | iter() | print() | tuple() |
| callable() | format() | len() | property() | type() |

| | | | | |
|---|---|---|---|---|
| chr() | frozenset() | list() | range() | vars() |
| classmethod() | getattr() | locals() | repr() | zip() |
| compile() | globals() | map() | reversed() | __import__() |
| complex() | hasattr() | max() | round() | |
| delattr() | hash() | memoryview() | set() | |

Todas estas funciones y algunos elementos comunes del lenguaje están definidas en el módulo builtins.

### Algunos ejemplos de funciones

- La entrada y salida de información se hacen con la función print y la función input:
- Tenemos algunas funciones matemáticas como: abs, divmod, hex, max, min,...
- Hay funciones que trabajan con caracteres y cadenas: ascii, chr, format, repr,...
- Además tenemos funciones que crean o convierten a determinados tipos de datos: int, float, str, bool, range, dict, list, ...

Iremos estudiando cada una de las funciones en las unidades correspondientes.

## Constantes predefinidas

En el módulo builtins se definen las siguientes constantes:

- True y False: Valores booleanos
- None especifica que alguna variables u objeto no tiene asignado ningún tipo.

Hay alguna constante más que veremos a lo largo del curso si es necesario.

## Ayuda en python

Una función fundamental cuando queremos obtener información sobre los distintos aspectos del lenguaje es help. Podemos usarla entrar en una sesión interactiva:

```
>>> help()

Welcome to Python 3.4's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.4/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".
```

help>

O pidiendo ayuda de una término determinado, por ejemplo:

>>> help(print)

# Datos

## Literales, variables y expresiones

### Literales

Los literales nos permiten representar valores. Estos valores pueden ser de diferentes tipos, de esta manera tenemos diferentes tipos de literales:

#### Literales numéricos

- Para representar números enteros utilizamos cifras enteras (Ejemplos: 3, 12, -23). Si queremos representarlos de forma binaria comenzaremos por la secuencia 0b (Ejemplos: 0b10101, 0b1100). La representación octal la hacemos comenzando por 0o (Ejemplos: 0o377, 0o7) y por último, la representación hexadecimal se comienza por 0x (Ejemplos: 0xdeadbeef, 0xfff).

- Para los números reales utilizamos un punto para separar la parte entera de la decimal (12.3, 45.6). Podemos indicar que la parte decimal es 0, por ejemplo 10., o la parte entera es 0, por ejemplo .001, Para la representación de números muy grandes o muy pequeños podemos usar la representación exponencial (Ejemplos: 3.14e-10,1e100).

- Por último, también podemos representar números complejos, con una parte real y otra imaginaria (Ejemplo: 1+2j)

#### Literales cadenas

Nos permiten representar cadenas de caracteres. Para delimitar las cadenas podemos usar el carácter ' o el carácter ". También podemos utilizar la combinación "' cuando la cadena ocupa más de una línea. Ejemplos.

'hola que tal!'
"Muy bien"
"'Podemos \n
ir al cine"'

Las cadenas anteriores son del tipo str, si queremos representar una cadena de tipo byte podremos hacerlo de la siguiente manera:

b'Hola'
B"Muy bien"

Con el carácter /, podemos escapar algunos caracteres, veamos algunos ejemplos:

| | |
|---|---|
| \\ | Backslash (\) |
| \' | Single quote (') |
| \" | Double quote (") |
| \a | ASCII Bell (BEL) |
| \b | ASCII Backspace (BS) |
| \f | ASCII Formfeed (FF) |
| \n | ASCII Linefeed (LF) |
| \r | ASCII Carriage Return (CR) |
| \t | ASCII Horizontal Tab (TAB) |
| \v | ASCII Vertical Tab (VT) |

## Variables

Una variable es un identificador que referencia a un valor. Estudiaremos más adelante que python utiliza tipado dinámico, por lo tanto no se usa el concepto de variable como almacén de información. Para que una variable referencie a un valor se utiliza el operador de asignación =.

El nombre de una variable, ha de empezar por una letra o por el carácter guión bajo, seguido de letras, números o guiones bajos. No hay que declarar la variable antes de usarla, el tipo de la variable será el mismo que el del valor al que hace referencia. Por lo tanto su tipo puede cambiar en cualquier momento:

```
>>> var = 5
>>> type(var)
<class 'int'>
>>> var = "hola"
>>> type(var)
<class 'str'>
```

Hay que tener en cuenta que python distingue entre mayúsculas y minúsculas en el nombre de una variable, pero se recomienda usar sólo minúsculas.

**Definición, borrado y ámbito de variables**

Como hemos comentado anteriormente para crear una variable simplemente tenemos que utilizar un operador de asignación, el más utilizado = para que referencia un valor. Si queremos borrar la variable utilizamos la instrucción del. Por ejemplo:

```
>>> a = 5
>>> a
```

```
5
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

Podemos tener también variables que no tengan asignado ningún tipo de datos:

```
>>> a = None
>>> type(a)
<class 'NoneType'>
```

El ámbito de una variable se refiere a la zona del programa donde se ha definido y existe esa variable. Como primera aproximación las variables creadas dentro de funciones o clases tienen un ámbito local, es decir no existen fuera de la función o clase. Concretamos cuando estudiamos estos aspectos más profundamente.

## Expresiones

Una expresión es una combinación de variables, literales, operadores, funciones y expresiones, que tras su evaluación o cálculo nos devuelven un valor de un determinado tipo.

Veamos ejemplos de expresiones:

```
a + 7
(a ** 2) + b
```

## Operadores. Precedencia de operadores en python

Los operadores que podemos utilizar se clasifican según el tipo de datos con los que trabajen y podemos poner algunos ejemplos:

```
+     -     *     **     /     //     %
<<     >>     &     |     ^     ~
<     >     <=     >=     ==     !=
-=     *=     /=     //=     %=
```

Podemos clasificarlos en varios tipos:

- Operadores aritméticos
- Operadores de cadenas
- Operadores de asignación
- Operadores de comparación
- Operadores lógicos
- Operadores a nivel de bit
- Operadores de pertenencia

- Operadores de identidad

La precedencia de operadores es la siguiente:

1. Los paréntesis rompen la precedencia.
2. La potencia (**)
3. Operadores unarios (~ + -)
4. Multiplicar, dividir, módulo y división entera (* /% // )
5. Suma y resta (+ -)
6. Desplazamiento nivel de bit (>> <<)
7. Operador binario AND (&)
8. Operadores binario OR y XOR (^ |)
9. Operadores de comparación (<= < > >=)
10. Operadores de igualdad (<> == !=)
11. Operadores de asignación (= %= /= //= -= += *= **=)
12. Operadores de identidad (is, is not)
13. Operadores de pertenencia (in, in not)
14. Operadores lógicos (not, or, and)

## Función eval()

La función eval() recibe una expresión como una cadena y la ejecuta.

```
>>> x=1
>>> eval("x + 1")
2
```

# Tipos de datos

Podemos concretar aún más los tipos de datos (o clases) de los objetos que manejamos en el lenguaje:

- Tipos numéricos
  - Tipo entero (int)
  - Tipo real (float)
  - Tipo numérico (complex)
- Tipos booleanos (bool)
- Tipo de datos secuencia
  - Tipo lista (list)
  - Tipo tuplas (tuple)
  - Tipo rango (range)
- Tipo de datos cadenas de caracteres
  - Tipo cadena (str)
- Tipo de datos binarios
  - Tipo byte (bytes)
  - tipo bytearray (bytearray)
- Tipo de datos conjuntos
  - Tipo conjunto (set)
  - Tipo conjunto inmutable (frozenset)

- Tipo de datos iterador y generador (iter)
- Tipo de datos mapas o diccionario (dict)

En realidad todo tiene definido su tipo o clase:

- Ficheros
- Módulos
- Funciones
- Excepciones
- Clases

## Función type()

La función type nos devuelve el tipo de dato de un objeto dado. Por ejemplo:

```
>>> type(5)
<class 'int'>
>>> type(5.5)
<class 'float'>
>>> type([1,2])
<class 'list'>
>>> type(int)
<class 'type'>
```

## Función isinstance()

Esta función devuelve True si el objeto indicado es del tipo indicado, en caso contrario devuelve False.

```
>>> isinstance(5,int)
True
>>> isinstance(5.5,float)
True
>>> isinstance(5,list)
False
```

# Trabajando con variables

Como hemos indicado anteriormente las variables en python no se declaran, se determina su tipo en tiempo de ejecución empleando una técnica que se llama **tipado dinámico**.

# ¿Qué es el tipado dinámico?

En python cuando asignamos una variable, se crea una referencia (puntero) al objeto creado, en ese momento se determina el tipo de la variable. Por lo tanto cada vez que asignamos de nuevo la variable puede cambiar el tipo en tiempo de ejecución.

```
>>> var = 3
>>> type(var)
<class 'int'>
>>> var = "hola"
>>> type(var)
<class 'str'>
```

# Objetos inmutables y mutables

## Objetos inmutables

Python procura no consumir más memoria que la necesaria. Ciertos objetos son **inmutables**, es decir, no pueden modificar su valor. El número 2 es siempre el número 2. Es un objeto inmutable. Python procura almacenar en memoria una sola vez cada valor inmutable. Si dos o más variables contienen ese valor, sus referencias apuntan a la misma zona de memoria.

**Ejemplo**

Para comprobar esto, vamos a utilizar la función id, que nos devuelve el identificador de la variable o el objeto en memoria.

Veamos el siguiente código:

```
>>> a = 5
```

Podemos comprobar que a hace referencia al objeto 5.

```
>>> id(5)
10771648
>>> id(a)
10771648
```

Esto es muy distinto a otros lenguajes de programación, donde una variable ocupa un espacio de memoria que almacena un valor. Desde este punto cuando asigno otro número a la variable estoy cambiando la referencia.

```
>>> a = 6
>>> id(6)
10771680
>>> id(a)
10771680
```

Las cadenas también son un objeto **inmutable**, que lo sean tiene efectos sobre las operaciones que podemos efectuar con ellas. La asignación a un elemento de una cadena, por ejemplo está prohibida:

```
>>> a = "Hola"
>>> a[0]="h"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

De los tipos de datos principales, hay que recordar que son inmutables son los números, las cadenas o las tuplas.

### Objetos mutables

El caso contrario lo tenemos por ejemplo en los objetos de tipo listas, en este caso las listas son mutables. Se puede modificar un elemento de una lista.

**Ejemplo**

```
>>> a = [1,2]
>>> b = a
>>> id(a)
140052934508488
>>> id(b)
140052934508488
```

Como anteriormente vimos, vemos que dos variables referencia a la misma lista en memoria. Pero aquí viene la diferencia, al poder ser modificada podemos encontrar situaciones como la siguiente:

```
>>> a[0] = 5
>>> b
[5, 2]
```

Podemos ver con el id, que sigue siendo el mismo.

Cuando estudiemos las listas abordaremos este compartiendo de manera completa. De los tipos de datos principales, hay que recordar que son mutables son las listas y los diccionarios.

# Operadores de identidad

Para probar esto de otra forma podemos usar los operadores de identidad:

- is: Devuelve True si dos variables u objetos están referenciando la misma posición de memoria. En caso contrario devuelve False.

- is not: Devuelve True si dos variables u objetos **no** están referenciando la misma posición de memoria. En caso contrario devuelve False.

**Ejemplo**

```
>>> a = 5
>>> b = a
>>> a is b
True
>>> b = b +1
>>> a is b
False
>>> b is 6
True
```

# Operadores de asignación

Me permiten asignar una valor a una variable, o mejor dicho: me permiten cambiar la referencia a un nuevo objeto.

El operador principal es =:

```
>>> a = 7
>>> a
7
```

Podemos hacer diferentes operaciones con la variable y luego asignar, por ejemplo sumar y luego asignar.

```
>>> a+=2
>>> a
9
```

Otros operadores de asignación: +=, -=, *=, /=, %=, **=, //=

# Asignación múltiple

En python se permiten asignaciones múltiples de esta manera:

```
>>> a, b, c = 1, 2, "hola"
```

# Entrada y salida estándar

## Función input

No permite leer por teclado información. Devuelve una cadena de caracteres y puede tener como argumento una cadena que se muestra en pantalla.

*Ejemplos*

```
>>> nombre=input("Nombre:")
Nombre:jose
>>> nombre
'jose'
>>> edad=int(input("Edad:"))
Edad:23
>>> edad
23
```

## Función print

Nos permite escribir en la salida estándar. Podemos indicar varios datos a imprimir, que por defecto serán separados por un espacio (se puede indicar el separador) y por defecto se termina con un carácter salto de línea \n (también podemos indicar el carácter final). Podemos también imprimir varias cadenas de texto utilizando la concatenación.

*Ejemplos*

```
>>> print(1,2,3)
1 2 3
>>> print(1,2,3,sep="-")
1-2-3
>>> print(1,2,3,sep="-",end=".")
1-2-3.>>>

>>> print("Hola son las",6,"de la tarde")
Hola son las 6 de la tarde
>>> print("Hola son las "+str(6)+" de la tarde")
Hola son las 6 de la tarde
```

## Formateando cadenas de caracteres

Existen dos formas de indicar el formato de impresión de las cadenas. En la documentación encontramos el estilo antiguo y el estilo nuevo.

*Ejemplos del estilo antiguo*

```
>>> print("%d %f %s" % (2.5,2.5,2.5))
2 2.500000 2.5

>>> print("%s %o %x"%(bin(31),31,31))
0b11111 37 1f

>>> print("El producto %s cantidad=%d precio=%.2f"%("cesta",23,13.456))
El producto cesta cantidad=23 precio=13.46
```

## Función format()

Para utilizar el nuevo estilo en python3 tenemos una función format y un método format en la clase str. Vamos a ver algunos ejemplos utilizando la función format, cuando estudiemos los métodos de str lo estudiaremos con más detenimiento.

*Ejemplos*

```
>>> print(format(31,"b"),format(31,"o"),format(31,"x"))
11111 37 1f

>>> print(format(2.345,".2f"))
2.35
```

# Tipo de datos numéricos

Python3 trabaja con tres tipos numéricos:

- Enteros (int): Representan todos los números enteros (positivos, negativos y 0), sin parte decimal. En python3 este tipo no tiene limitación de espacio.
- Reales (float): Sirve para representar los números reales, tienen una parte decimal y otra decimal. Normalmente se utiliza para su implementación un tipo double de C.
- Complejos (complex): Nos sirven para representar números complejos, con una parte real y otra imaginaria.

Como hemos visto en la unidad anterior son tipos de datos inmutables.

*Ejemplos*

```
>>> entero = 7
>>> type(entero)
<class 'int'>
>>> real = 7.2
>>> type (real)
<class 'float'
>>> complejo = 1+2j
>>> type(complejo)
```

<class 'complex'>

# Operadores aritméticos

- +: Suma dos números
- -: Resta dos números
- *: Multiplica dos números
- /: Divide dos números, el resultado es float.
- //: División entera
- %: Módulo o resto de la división
- **: Potencia
- +, -: Operadores unarios positivo y negativo

# Funciones predefinidas que trabajan con números:

- abs(x): Devuelve al valor absoluto de un número.
- divmod(x,y): Toma como parámetro dos números, y devuelve una tupla con dos valores, la división entera, y el módulo o resto de la división.
- hex(x): Devuelve una cadena con la representación hexadecimal del número que recibe como parámetro.
- oct(x): Devuelve una cadena con la representación octal del número que recibe como parámetro.
- bin(x): Devuelve una cadena con la representación binaria del número que recibe como parámetro.
- pow(x,y): Devuelve la potencia de la base x elevado al exponente y. Es similar al operador **`.
- round(x,[y]): Devuelve un número real (float) que es el redondeo del número recibido como parámetro, podemos indicar un parámetro opcional que indica el número de decimales en el redondeo.

*Ejemplos*

```
>>> abs(-7)
7
>>> divmod(7,2)
(3, 1)
>>> hex(255)
'0xff'
>>> oct(255)
'0o377'
>>> pow(2,3)
8
>>> round(7.567,1)
7.6
```

# Operadores a nivel de bit

- x | y: x OR y

- x ^ y: x XOR y
- x & y: a AND y
- x << n: Desplazamiento a la izquierda **n** bits.
- x >> n: Desplazamiento a la derecha **n** bits.
- ~x: Devuelve los bits invertidos.

# Conversión de tipos

- int(x): Convierte el valor a entero.
- float(x): Convierte el valor a float.
- complex(x): Convierte el valor a un complejo sin parte imaginaria.
- complex(x,y): Convierta el valor a un complejo, cuya parte real es x y la parte imaginaria y.

Los valores que se reciben también pueden ser cadenas de caracteres (str).

*Ejemplos*

```
>>> a=int(7.2)
>>> a
7
>>> type(a)
<class 'int'>
>>> a=int("345")
>>> a
345
>>> type(a)
<class 'int'>
>>> b=float(1)
>>> b
1.0
>>> type(b)
<class 'float'>
>>> b=float("1.234")
>>> b
1.234
>>> type(b)
<class 'float'>
```

Por último si queremos convertir una cadena a entero, la cadena debe estar formada por caracteres numéricos, sino es así, obtenemos un error:

```
a=int("123.3")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '123.3'
```

# Tipo de datos booleanos

## Tipo booleano

El tipo booleano o lógico se considera en python3 como un subtipo del tipo entero. Se puede representar dos valores: verdadero o false (True, False).

## ¿Qué valores se interpretan como FALSO?

Cuando se evalúa una expresión, hay determinados valores que se interpretan como False:

- None
- False
- Cualquier número 0. (0, 0.0, 0j)
- Cualquier secuencia vacía ([], (), '')
- Cualquier diccionario vacío ({})

## Operadores booleanos

Los operadores booleanos se utilizan para operar sobre expresiones booleanas y se suelen utilizar en las estructuras de control alternativas (if, while):

- x or y: Si x es falso entonces y, sino x. Este operados sólo evalúa el segundo argumento si el primero es False.
- x and y: Si x es falso entonces x, sino y. Este operados sólo evalúa el segundo argumento si el primero es True.
- not x: Si x es falso entonces True, sino False.

## Operadores de comparación

**== != >= > <= <**

## Funciones all() y any()

- all(iterador): Recibe un iterador, por ejemplo una lista, y devuelve True si todos los elementos son verdaderos o el iterador está vacío.
- any(iterador): Recibe un iterador, por ejemplo una lista, y devuelve True si alguno de sus elemento es verdadero, sino devuelve False.

# Boletín 1

1. Escribir un programa que pregunte al usuario su nombre, y luego lo salude.
2. Calcular el perímetro y área de un rectángulo dada su base y su altura.
3. Calcular el perímetro y área de un círculo dado su radio.
4. Dados dos números, mostrar la suma, resta, división y multiplicación de ambos.

5. Escribir un programa que le pida una palabra al usuario, para luego imprimirla 1000 veces, con espacios intermedios.
6. Escribir un programa que le pida una palabra al usuario, para luego imprimirla 1000 veces, con espacios intermedios.

# Estructura de control: Alternativas

Si al evaluar la expresión lógica obtenemos el resultado True ejecuta un bloque de instrucciones, en otro caso ejecuta otro bloque.

## Alternativas simples

```
if numero<0:
        print("Número es negativo")
```

## Alternativas dobles

```
if numero<0:
        print("Número es negativo")
else:
        print("Número es positivo")
```

## Alternativas múltiples

```
if numero>0:
        print("Número es negativo")
elif numero<0:
        print("Número es positivo")
else:
        print("Número es cero")
```

## Expresión reducida del if

```
>>> lang="es"
>>> saludo = 'HOLA' if lang=='es' else 'HI'
>>> saludo
'HOLA'
```

# Boletín 2. Alternativas

1. Realiza un programa que pida dos números ‘a’ y ‘b’ e indique si su suma es positiva, negativa o cero.
2. Escribe un programa que lea un número e indique si es par o impar.

3. Escribe un programa que pida un número entero entre uno y doce e imprima el número de días que tiene el mes correspondiente.
4. Escribe un programa que pida un nombre de usuario y una contraseña y si se ha introducido “pepe” y “asdasd” se indica “Has entrado al sistema”, sino se da un error.
5. Escribir un programa que lea un año indicar si es bisiesto. Nota: un año es bisiesto si es un número divisible por 4, pero no si es divisible por 100, excepto que también sea divisible por 400.
6. Programa que lea un carácter por teclado y compruebe si es una letra mayúscula.

# Estructura de control: Repetitivas

## while

La estructura while nos permite repetir un bloque de instrucciones mientras al evaluar una expresión lógica nos devuelve True. Puede tener una estructura else que se ejecutará al terminar el bucle.

*Ejemplo*

```
año = 2001
while año <= 2017:
        print ("Informes del Año", año)
        año += 1
else:
        print ("Hemos terminado")
```

## for

La estructura for nos permite iterar los elementos de una secuencia (lista, rango, tupla, diccionario, cadena de caracteres,...). Puede tener una estructura else que se ejecutará al terminar el bucle.

*Ejemplo*

```
for i in range(1,10):
   print (i)
else:
   print ("Hemos terminado")
```

## Instrucciones break, continue y pass

### break

Termina la ejecución del bucle, además no ejecuta el bloque de instrucciones indicado por la parte else.

### continue

Deja de ejecutar las restantes instrucciones del bucle y vuelve a iterar.

### pass

Indica una instrucción nula, es decir no se ejecuta nada. Pero no tenemos errores de sintaxis.

## Recorriendo varias secuencias. Función zip()

Con la instrucción for podemos ejecutar más de una secuencia, utilizando la función zip. Esta función crea una secuencia donde cada elemento es una tupla de los elementos de cada secuencia que toma cómo parámetro.

*Ejemplo*

```
>>> list(zip(range(1,4),["ana","juan","pepe"]))
[(1, 'ana'), (2, 'juan'), (3, 'pepe')]
```

Para recorrerla:

```
>>> for x,y in zip(range(1,4),["ana","juan","pepe"]):
...     print(x,y)
1 ana
2 juan
3 pepe
```

# Boletín 3. Repetitivas

1. Pedir un número por teclado y mostrar la tabla de multiplicar. (hacer dos versiones una que utilice while y otra que utilice for)
2. Crea una aplicación que pida un número y calcule su factorial (El factorial de un número es el producto de todos los enteros entre 1 y el propio número y se representa por el número seguido de un signo de exclamación. Por ejemplo 5! = 1x2x3x4x5=120
3. Crea una aplicación que permita adivinar un número. En primer lugar la aplicación solicita un número entero por teclado. A continuación va pidiendo números y va

respondiendo si el número a adivinar es mayor o menor que el introducido. El programa termina cuando se acierta el número.

4. Programa que muestre la tabla de multiplicar de los números 1,2,3,4 y 5.
5. Escribe un programa que diga si un número introducido por teclado es o no primo. Un número primo es aquel que sólo es divisible entre él mismo y la unidad.