Lenguaje de marcas Documentation

Release 2017.0

José Domingo Muñoz

Apr 26, 2018

Contents

El módulo profesional de **Lenguajes de Marcas** se imparte durante el primer curso del Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos en Red (ASIR).

De acuerdo a la normativa reguladora del ciclo formativo, el módulo profesional de Lenguajes de Marcas se imparte durante el primer curso y tiene asignadas un total de 128 horas, a razón de 4 horas semanales.

El índice de contenidos que vamos a estudiar será:

CHAPTER 1

Unidades

1.1 Introducción a la programación. Python

1.1.1 Programación intuitiva

Enlaces

- code.org
- El País - Aprender a programar como se aprende a leer
- BBC - Free tool offers 'easy' coding
- Scratch
- vimeo.com - Historia de los videojuegos
- programamos.es
- Tutorial de Scratch

Prácticas

Ejercicio 1 Scratch

Estructura secuencial

1. Dibujar un cuadrado:

2. Dibuja un triangulo y un rectangulo.
3. Haz que el gato diga "Hola Mundo", espere 3 segundo y luego diga "Hasta luego".
4. Movimientos:
   - Mueve al gato a la derecha e izquierda, según pulsemos las teclas del cursor. El gato se debe mover 10 pasos, debe apuntar en la dirección deseada, y por último debe cambiar el disfraz para simular que está andando. (Nota: El estilo de rotación del objeto debe ser izquierda - derecha)
   - Cuando pulsemos el espacio el gato debe maullar.
   - Introduce la posibilidad de mover arriba y abajo el gato. En este caso el gato no debe girar.

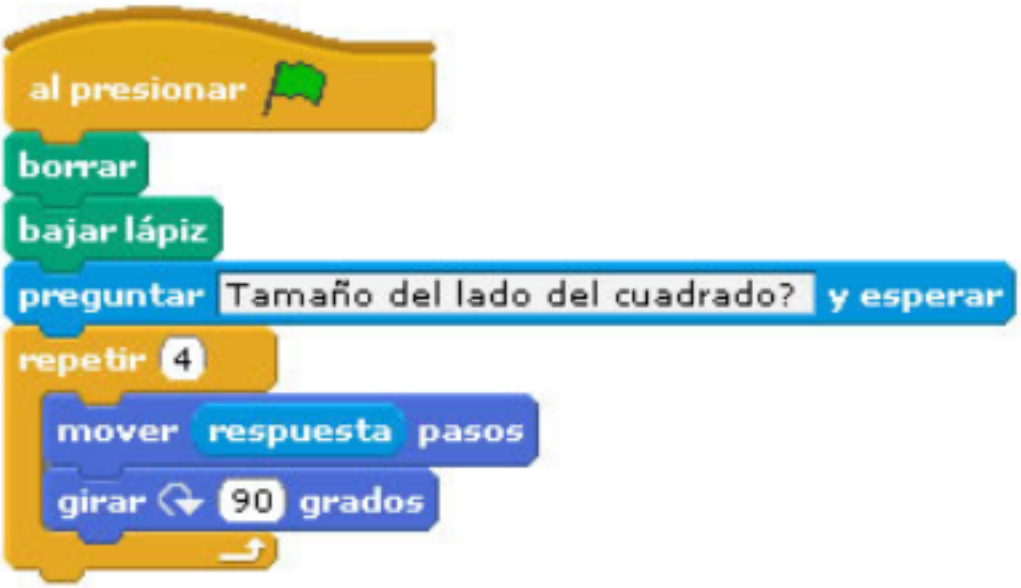
Variables

1. Pedir un nombre por teclado, y hacer que el gato diga "Hola Nombre".
2. Calcular el perímetro y área de un rectángulo dada su base y su altura.
3. Dados dos números, mostrar la suma, resta, división y multiplicación de ambos.

Ejercicio 2 Scratch

Estructura repetitiva

1. Dibujar un cuadrado, el tamaño del lado se pide por teclado:



2. Haz que el gato diga 5 veces "Hola, que pasa!!!"
3. Uso de un contador. Haz que el gato cuente del 1 al 10.
4. Haz que el gato ande indefinidamente de izquierda a derecha. Cambia el disfraz para simular que está andando. Haz que rebote en las paredes y gire al sentido contrario.
5. El gato te debe pedir un número, y decirte la tabla de multiplicar de ese número.
6. Uso de un acumulador. El gato calcula una potencia. Te pide la base y el exponente y te da el resultado de la potencia.

Ejercicio 3 Scratch

Estructura condicional

1. Realiza un programa donde el gato te pida la nota que has sacado, si es mayor o igual que cinco, te dice "Aprobado", sino te dirá "Suspenso".
2. Ahora el gato te pide dos números, y te dice cuál es el mayor.
3. Mejora el ejercicio 4 del boletín anterioro para que cuando pulsemos la tecla espacio, al gato nos diga cuantas veces ha chocado con la pared.
4. ¿Hay alguna diferencia entre estos dos programas?


al presionar
preguntar Ingrese el 1er ángulo del triángulo y esperar
fijar ángulo1 a respuesta
preguntar Ingrese el 2do ángulo del triángulo y esperar
fijar ángulo2 a respuesta
fijar ángulo3 a 180 - ángulo1 + ángulo2
fijar esRectángulo a NO
si ángulo1 = 90
fijar esRectángulo a SI
si ángulo2 = 90
fijar esRectángulo a SI
si ángulo3 = 90
fijar esRectángulo a SI
si esRectángulo = SI
decir SI es un triángulo rectángulo!!!!
esperar 5 segundos
si no
decir NO es un triángulo rectángulo :(
esperar 5 segundos

5. Juego: Adivina el número. Vamos a hacer que el gato piense un número aleatorio, entre el 1 y el 100. Nos va piediendo que digamos un número hasta que lo acertemos. Si el número que introducimos es menor al que él ha pensado, nos dice "El número que yo he pensado es mayor", en el caso en que el número es mayor nos dirá "El número que yo he pensado es menor". Cuando digamos el número pensado, dirá "Muy bien, has acertado!!!".
6. Modifica el ejercicio anterior, para que nos diga en cuantos intentos hemos acertado el número.

1.1.2 Introducción a la programación

- Introducción a los lenguajes de programación

Enlaces

- levenez.com - Computer Language History [pdf]
- Wikipedia - Lenguaje de programación interpretado
- Wikipedia - Lenguaje compilado
- Índice TIOBE - Septiembre 2017

Prácticas

Compilación y ejecución de un lenguaje compilado: C

Todas las distribuciones GNU/Linux incluyen alguna versión del GNU Project C and C++ Compiler (gcc), vamos a utilizarlo.

1. Crea el fichero `helloworld.c` con el siguiente contenido:

```
#include <stdio.h>
int main(void)
{
        printf("Hello world!\n");
        return 0;
}
```

2. Compílalo con gcc:

```
gcc helloworld.c
```

3. Ejecuta la aplicación:

```
./a.out
```

4. Verifica que `a.out` es un fichero binario para linux 64-bit:

```
file a.out
```

5. Realmente la compilación incluye varios pasos, el más importante de ellos es la creación de un fichero objeto intermedio, vamos a repetir el proceso en dos pasos:

```
gcc -c helloworld.c
```

que genera el fichero `helloworld.o`, del tipo:

```
helloworld.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

Para enlazarlo y producir el fichero de salida a.out:

```
gcc -Wl helloworld.o
```

¿Qué sentido tiene compilar por partes? Cuando el código es grande no hay un solo fichero fuente sino muchos, compilar individualmente estos "módulos" permite, por ejemplo, ahorrar mucho tiempo en la modificación y compilación de un solo componente.

Compilación y ejecución de C en Windows

Los sistemas Windows no incluyen inicialmente ningún compilador de C, pero hay muchos compiladores que funcionan en sistemas Windows, en particular dev-c++

1. Copia el fichero binario `a.out` compilado en GNU/Linux con gcc en la partición Windows y reinicia el equipo con este sistema
2. Ejecuta la aplicación `a.out` en Windows, ¿qué ocurre?
3. Instala el IDE dev-c++ en windows
4. Crea un fichero `hola.c` con el programa hola mundo, compílalo y ejecútalo
5. ¿Entiendes que teniendo el mismo código fuente los binarios son diferentes?

Bibliotecas estáticas o dinámicas

Un conjunto de ficheros objeto que se enlazan con la parte principal de un programa para producir un binario o ejecutable reciben el nombre de biblioteca (library en inglés). Si el enlace se realiza durante la compilación, las

bibliotecas se denominan estáticas o de enlace estático, mientras que si el enlace se realiza durante la ejecución las bibliotecas se denominan dinámicas o de enlace dinámico.

La principal ventaja de la compilación con bibliotecas de enlace dinámico es que el tamaño de los ficheros binarios de las aplicaciones es mucho menor y se optimiza el uso de la memoria porque una biblioteca dinámica puede ser utilizada por diferentes aplicaciones.

En sistemas windows las bibliotecas dinámicas se distribuyen en ficheros con extensión `.dll` (dynamic link library) y en sistemas GNU/Linux lo hacen en los paquetes `lib*` que incluyen ficheros con extensión `.so` (shared object).

Compilación e interpretación de un programa Java

Instalación y utilización de JRE

Java Runtime Environment (JRE) son el conjunto de aplicaciones que se instalan en un equipo para que puedan ejecutarse en él aplicaciones java. Los dos componentes principales de un JRE son:

- Java Virtual Machine: Aplicación que ejecuta el código java en bytecode y que está adaptada a la plataforma sobre la que opera.
- Bibliotecas Java

Existen diferentes implantaciones de JRE, siendo la última versión estable la 1.8, conocida como Java 8. Actualmente el propietario de Java es la empresa Oracle y ha modificado la antigua licencia libre de Java, por lo que ya no es posible que se distribuya legalmente en las distribuciones de software libre. Nosotros optaremos por utilizar OpenJDK, que es una implementación libre de Java.

Instalación de openjdk jre 8 en Debian Jessie:

- Busca el paquete openjdk-8-jre
- Si no lo tienes instalado en tu equipo, instálalo
- Como hay varios paquetes alternativos que implementan jre, hay que elegir el que queremos utilizar en nuestro equipo, para ello hay que hacer:

```
update-alternatives --config java
```

y elegir la opción de openjdk8.

Instalación de sun jre 8 en Windows:

- Comprueba si tienes instalada la versión 8 de JRE en tu equipo, si no es así, entra en el sitio de descargas, descárgate la versión de JRE para Windows e instálala.
- Ejecución de la misma aplicación en las dos plataformas
- Entra en el sitio http://www.jedit.org y descárgate la versión multiplataforma de este editor, no la de linux ni windows.
- Comprueba que puedes ejecutar esa misma aplicación en las dos plataformas.

Instalación y utilización de JDK

Java Development Kit (JDK) son el conjunto de programas para desarrollar aplicaciones y entre otros incluye el compilador javac que convierte un programa fuente java a bytecode.

- Instala el paquete openjdk-7-jdk
- Ejecuta la instrucción update-alternatives –config javac y elige openjdk 7 si hubiese más de una opción

- Crea un fichero HelloWorld.java con el siguiente contenido:

```
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello world!");
 }
}
```

- Compila la aplicación:

```
javac HelloWorld.java
```

- que creará el bytecode `Helloworld.class`, ejecuta el código java:

```
java HelloWorld
```

- Coge el fichero `HelloWorld.class` y ejecútalo en cualquier otro equipo con java.

Ejecución de programas interpretados

Lenguaje interpretado: Es el lenguaje cuyo código no necesita ser preprocesado mediante un compilador, eso significa que el ordenador es capaz de ejecutar la sucesión de instrucciones dadas por el programador sin necesidad de leer y traducir exhaustivamente todo el código.

Bash

El lenguaje Bash nos permite escribir programas utilizando las instrucciones que usamos en el terminal de linux. Este lenguaje está pensado para hacer pequeños programas (scripts) que nos facilitan hacer alguna tarea. El lenguaje Bash es interpretado.

Veamos un ejemplo, crea un fichero ejemplo.sh con el siguiente contenido:

```
#!/bin/bash
a=$((RANDOM%100))
intentos=1
read -p "¿Que numero crees que es? " b
while [ $b -ne $a ]
do
        if [ $b -lt $a ]
        then
                echo "El numero que has introducido es menor"
        else
                echo "El numero que has introducido es mayor"
        fi
        read -p "Has fallado, introduce otro numero " b
        intentos=$((intentos+1))
done
echo "Has acertado"
echo "Has necesitado" $intentos "intentos"
```

Para ejecutarlo, le tenemos que dar permiso de ejecución:

```
# chmod 755 ejemplo.sh#
./ejemplo.sh
```

Python

Python también es un lenguaje interpretado, veamos un ejemplo: crea un fichero ejemplo.py con el siguiente contenido:

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

a=random.randrange(0, 100)
intentos=1
b=int(input("Introduce un número:"))
while a!=b:
        if b>a:
                print("El número introducido es mayor")
        else:
                print("El número introducido es menor")
        intentos=intentos+1
        b=int(input("Introduce un número:"))
print("Has acertado en %d intentos." % intentos)
```

Para ejecutar el programa usamos el interprete python:

```
python3 ejemplo.py
```

1.1.3 Introducción a python3

Teoría

Introducción a python

Presentación del tema

Características de Python

Python es un lenguaje:

- Interpretado
- Alto nivel
- Multiparadigma, ya que soporta **orientación a objetos**, **programación imperativa** y **programación funcional**.
- Multipltaforma
- Libre

¿Por qué elegir python?

- Porque es fácil de aprender
- Sintaxis muy limpia y sencilla
- Hay que escribir menos
- Obtienes resultados muy rápido
- Puedes programar con distintos paradígmas:

  - Programación imperativa
  - Orientación a objetos
  - Programación funcional
- Puedes programas distintos tipos de aplicaciones:
  - Aplicaciones de escritorio
  - Aplicaciones web
  - Scripts
- Muchos usan Python (Google, Nokia, IBM). Es demandado.
- Gran cantidad de módulos, muchísimas funcionalidades.
- Una gran comunidad que apoya el proyecto.
- Viene preinstalado en la mayoría de sistemas

Python2 us Python3

python 2.x y python 3.x

La ultima versión 2.x fue la 2.7 de 2010, contando con soporte hasta el final de su vida útil. No está en desarrollo. La versión 3.x está bajo desarrollo activo, la última versión 3.6 salió el 23 de diciembre de 2016. Las modificaciones que se han incluido en python 3.x en sintaxis y módulos claves han hecho que no sea compatible con python 2.x.

En el post: What's New In Python 3.0 escrito por Guido van Rossum podemos encontrar los cambios introducidos en la versión 3.x. En la documentación podéis encontrar la página What's New in Python donde podéis estudiar las mejoras de cada una de las versión que van saliendo.

Entonces, ¿Qué versión debería utilizar?

Te debes asegurar si las bibliotecas que vas a utilizar son compatibles con la versión que vas a utilizar. El problema en los últimos años ha sido que no todas las librerías se habían exportado a la versión 3. En los últimos tiempo la versión 3 es suficientemente madura para se utilizada y muchos de las librerías y software más utilizado ya está exportado. Puedes ver la lista de los paquetes exportados a la versión 3 en la página http://python3wos.appspot.com/.

Si es totalmente necesario, porque la librería que necesito no está portada tendríamos que usar la versión 2, pero hay que tener en cuenta que python 2.x es un lenguaje antiguo con errores, por lo tanto merece la pena hacer un esfuerzo y buscad alternativas para usar la versión 3. Si tienes código en la antigua versión, también existen herramientas para realizar la portabilidad: Porting Python Code to 3.x.

Las principales difreencia entre python 2.x y 3.x

Print es una función en python3

En python2:

```
print "hola mundo"
```

En python3:

```
print ("Hola mundo")
```

División de números enteros

En python 2 al dividir enteros, siempre el resultado era un entero, en python3 el resultado es un número real.

En python2:

```
>>> 4/3
1
```

En python3:

```
>>> 3/2
1.5

>>> num = 3/2
>>> type(num)
<class 'float'>
>>> num = 4/2
>>> type(num)
<class 'float'>
```

Las "cadenas" (strings) son Unicode de forma predeterminada en python 3

En python2 existe dos tipos diferenciados de cadenas: str (ascii) y unicode, en python 3 todas las cadenas son unicodes.

En python2:

```
>>> cad = "piña"
>>> cad
'pi\xc3\xb1a'
```

En python3:

```
>>> cad = "piña"
>>> cad
'piña'
```

Generación de listas de número

En python2 teníamos dos funciones parecidas: range que generaba una lista de números, y xrange que era una función que devolvía un objeto de tipo xrange. La diferencia entre ambas era que utilizar esta última era mucho más eficiente. E python3 sólo tenemos range que ha pasado a ser un tipo de datos.

En python2:

```
>>> range(1,10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> xrange(1,10)
xrange(1, 10)
>>> type(xrange(1,10))
<type 'xrange'>
```

En pyton3:

```
>>> range(1,10)
range(1, 10)
>>> type(range(1,10))
<class 'range'>
```

Input es una cadena de texto en python 3

En python 2 habían dos funciones para ingresar datos por un teclado raw_input() en que lo ingresado se trataba como una cadena de texto e input() en lo que se ingresaba se evaluaba y se trataba por su tipo. En python 3, se eliminó el input() de python 2 quedando el raw_input() como el nuevo input(). O sea el input() de python 3 siempre devuelve una cadena de texto.

En python2:

```
>>> cad=raw_input()
123
>>> type(cad)
<type 'str'>
>>> num=input()
123
>>> type(num)
<type 'int'>
```

En python3:

```
>>> num=input()
123
>>> type(num)
<class 'str'>
```

Comparando tipos

Python 3 nos indica un error cuando intentamos comparar tipos de datos diferentes.

En python2:

```
>>> [1,2] > "hola"
False
```

En python3:

```
>>> [1,2] > "hola"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: list() > str()
```

Instalación de python3

La página principal para descargar las distintas versiones es:www.python.org/downloads/.

Instalación en Linux

Python viene preinstalado de serie en la mayoría de las distribuciones GNU/Linux. Si no tenemos a nuestra disposición paquetes para instalarlo.

- En Debian 8 Jessie, la versión por defecto es la 3.4.2
- En Debian 9 Stretch, la versión es la 3.5.3-1
- En Ububtu 16.04 Xenial, la versión es la 3.5.3-3

Para más información

Instalación en Windows

Podemos descargarnos un instalador (paquete MSI) de las distintas versión de python.

Para más información

Instalación en Mac OS

Mac OS X 10.8 viene con Python 2.7 preinstalado. Si desas instalar Python3 puedes encontrar el paquete de la versión deseada en la página de desrcarga: www.python.org/downloads/.

Para más información

Entornos de desarrollos y editores de texto

Una decisión importante que debes tomar cuando empiezas a trabajar en informática es acerca del editor o editores de texto que vas a utilizar. Hay muchas opciones y aspectos a considerar. Además en determinadas entornos es posible que no sea suficiente con utilizar un simple editor de texto y sea el necesario el uso de un IDE (entorno de desarrollo integrado), que además de la posibilidad de editar el código, nos ofrezca otras herramientas: depuración de código, generación automático de código, ayuda integrada, manejo del proyecto, gestión de los sistemas de control de versiones,...

A La hora de decidir en qué entorno trabajar, nos deberíamos hacer las siguientes preguntas:

- ¿Editor gráfico o en consola? Para tareas de administración lo ideal sería saber manejar un editor de texto en consola, ya que en muchas ocasiones tendrás que hacerlo en un equipo remoto sin entorno gráfico. Las tres opciones más habituales son vim, nano y emacs-nox Sin embargo, para tareas de programación, es habitual es utilizar un editor gráfico con más funcionalidades: emacs, atom, sublime text, notepad++
- ¿Editor simple o IDE?. Habría que considerar que aprender a manejar un IDE lleva más tiempo que un simple editor y no es adecuado para aplicaciones sencillas como las que vamos a utilizar nosotros en este curso. Evidentemente el uso de un IDE se hace imprescindible en un entorno profesional.
- ¿Qué funcionalidades básicas debe tener el editor?: resaltado de sintaxis, numeración de líneas, control de sangrado (indentación), manejo completo desde teclado Soporte para python.
- ¿Es multiplataforma?. Permite que el mismo editor de texto se utilice en diferentes sistemas operativos y puede ser un aspecto determinante para nosotros.

IDE para python

- Entornos de desarrollo para python

Editores de texto para python

- Editores de texto para python

Enlaces

- Los 5 mejores editores Python

Mi primer programa en python3

La documentación de este curso esta escrita usando la distribución GNU/Linux Debian Jessie. Algunas particularidades pueden cambiar en otras versiones, distribuciones o sistemas operativos.

Uso del interprete

Al instalar python3 el ejecutable del interprete 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 ejecutaríamos de la siguiente manera.

```
$ python3 programa.py
```

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 interprete 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 caracter 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 empiza en la primera posición, si tenemos instrucciones dentro de un bloque de una estrucura de control de flujo habra 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 punto.

Comentarios

Se utiliza el caracter # 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
```

Ejemplo

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

# Sangrado con 4 espacios

edad = 23
if edad > =18:
   print('Es mayor de edad')
else:
   print('Es menor de edad')

# Cada bloque de instrucciones dentro de una estructura de control
# debe estar tabulada

if num >=0:
    while num<10:
        print (num)
        num = num +1

# El punto y coma ";" se puede usar para separar varias sentencias
# en una misma línea, pero no se aconseja su uso.

edad = 15; print(edad)

# Cuando el bloque a sangrar sólo ocupa una línea ésta puede
# escribirse después de los dos puntos:

if azul: print('Cielo')

# La barra invertida "\" permite escribir una línea de
# código demasiado extensa en varias líneas:

if condicion1 and condicion2 and condicion3 and \
    condicion4 and condicion5:

# Las expresiones entre paréntesis, llaves o corchetes pueden
# ocupar varias líneas:

dias = ['lunes', 'martes', 'miércoles', 'jueves',
        'viernes', 'sábado', 'domingo']
```

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 elmentos 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 estudianda cada una de las funciones en las unidades correspondientes.

Constantes predefinidas

En el módulo builtins se definen las siguientes constantes:

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

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

Ayuda en python

Un 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 termino 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.

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'''
```

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

```
\n  ASCII Linefeed (LF)
\t  ASCII Horizontal Tab (TAB)
```

Variables

Una variables 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 cuanta 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. Concretaremos 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:

- Operadores aritméticos (`+ - * ** / // %`)
- Operadores de cadenas (`+ *`)
- Operadores de asignación (`= -= *= /= //= %=`)
- Operadores de comparación (`< > <= >= == !=`)
- Operadores lógicos (`and, or, not`)

La procedencia de operadores es la siguiente:

1. Los paréntesis rompen la procedencia.
2. La potencia (**)
3. Operadores unarios (+ -)
4. Multiplicar, dividir, módulo y división entera (* /% //)
5. Suma y resta (+ -)
6. Operadores de comparación (<= < > >=)
7. Operadores de igualdad (<> == !=)
8. Operadores de asignación (= %= /= //= -= += *= **=)
9. Operadores lógicos (not, or, and)

Trabajando con variables

Las variables en python no se declaran, se determina su tipo en tiempo de ejecución empleando una técnica que se lama **tipado dinámico**.

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"
```

Tipo de datos numéricos

Python3 trabaja con dos 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.

Ejemplos

```
>>> entero = 7
>>> type(entero)
<class 'int'>
>>> real = 7.2
>>> type (real)
<class 'float'
```

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.
- `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 elevedao al exponete 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
>>> hex(255)
'0xff'
>>> oct(255)
'0o377'
>>> pow(2,3)
8
>>> round(7.567,1)
7.6
```

Conversión de tipos

- `int(x)`: Convierte el valor a entero.

- `float(x)`: Convierte el valor a float.

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'
```

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

No permite escribir en la salida estándar. Podemos indicar varios datos a imprimir, que por defecto serán separado 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

Existe 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 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 evalua una expresión, hay determinados valores que se interpretan como False:

- `None`
- `False`
- Cualquier número 0. (0, 0.0)
- Cualquier secuencia vacía ([], (), '')
- Cualquir 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 evalua el segundo argumento si el primero es False.
- `x and y`: Si x es falso entonces x, sino y. Este operados sólo evalua el segundo argumento si el primero es True.
- `not x`: Si x es falso entonces True, sino False.

Operadores de comparación

```
== != >= > <= <
```

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'
```

Estructura de control: Repetitivas

while

La estructura `while` nos permite repetir un bloque de instrucciones mientras al evaluar una expresión lógica nos devuelve True.

Ejemplo

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

for

La estructura `for` nos permite iterar los elementos de una secuencia (lista, rango, tupla, diccionario, cadena de caracteres,...).

Ejemplo

```
for i in range(1,10):
    print (i)
```

Instrucciones break y continue

break

Termina la ejecución del bucle.

continue

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

Tipo de datos secuencia: Listas

Las listas (`list`) me permiten guardar un conjunto de datos que se pueden repetir y que pueden ser de distintos tipos.

Construcción de una lista

Para crear una lista puedo usar varias formas:

- Con los caracteres `[` y `]`:

```
>>> lista1 = []
>>> lista2 = ["a",1,True]
```

- Utilizando el constructor `list`, que toma como parámetro un dato de algún tipo secuencia.

```
>>> lista3 = list()
>>> lista4 = list("hola")
>>> lista4
['h', 'o', 'l', 'a']
```

Operaciones básicas con listas

Vamos a ver distintos ejemplos partiendo de una lista, que es una secuencia mutable.

```
lista = [1,2,3,4,5,6]
```

- Las secuencias se pueden recorrer.

```
>>> for num in lista:
...   print(num,end="")
123456
```

- Operadores de pertenencia: Se puede comprobar si un elemento pertenece o no a una secuencia con los operadores `in` y `not in`.

```
>>> 2 in lista
True
>>> 8 not in lista
True
```

- Concatenación: +: El operador + me permite unir datos de tipos listas.

```
>>> lista + [7,8,9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Repetición: `*`: El operador `*` me permite repetir un dato de un tipo lista.

```
>>> lista * 2
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

- Indexación: Cada elemento tiene un índice, empezamos a contar por el elemento en el índice 0. Si intento acceder a un índice que corresponda a un elemento que no existe obtenemos una excepción `IndexError`.

```
>>> lista1[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module
IndexError: list index out of range
```

  Se pueden utilizar índices negativos:

```
>>> lista[-1]
6
```

- Slice: Veamos como se puede utilizar
  - `lista[start:end]` # Elementos desde la posición start hasta end-1
  - `lista[start:]` # Elementos desde la posición start hasta el final
  - `lista[:end]` # Elementos desde el principio hata la posición end-1
  - `lista[:]` # Todos Los elementos
  - `lista[start:end:step]` # Igual que el anterior pero dando step saltos.

  Se pueden utilizar también índices negativos, por ejemplo: `lista[::-1]`

```
>>> lista[2:4]
[3, 4]
>>> lista[1:4:2]
[2, 4]
```

Funciones predefinidas que trabajan con listas

```
>>> lista1 = [20,40,10,40,50]
>>> len(lista1)
5
>>> max(lista1)
50
>>> min(lista1)
10
>>> sum(lista1)
150
>>> sorted(lista1)
[10, 20, 30, 40, 50]
>>> sorted(lista1,reverse=True)
[50, 40, 30, 20, 10]
```

Listas multidimensionales

A la hora de definir las listas hemos indicado que podemos guardar en ellas datos de cualquier tipo, y evidentemente podemos guardar listas dentro de listas.

```
>>> tabla = [[1,2,3],[4,5,6],[7,8,9]]
>>> tabla[1][1]
5

>>> for fila in tabla:
...   for elem in fila:
...      print(elem,end="")
...   print()

123
456
789
```

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
```

Métodos principales de listas

Cuando creamos una lista estamos creando un objeto de la clase `list`, que tiene definido un conjunto de métodos:

```
lista.append   lista.copy     lista.extend   lista.insert   lista.remove   lista.sort
lista.clear    lista.count    lista.index    lista.pop      lista.reverse
```

Métodos de inserción: append, extend, insert

```
>>> lista = [1,2,3]
>>> lista.append(4)
>>> lista
[1, 2, 3, 4]

>>> lista2 = [5,6]
>>> lista.extend(lista2)
```

```
>>> lista
[1, 2, 3, 4, 5, 6]

>>> lista.insert(1,100)
>>> lista
[1, 100, 2, 3, 4, 5, 6]
```

Métodos de eliminación: pop, remove

```
>>> lista.pop()
6
>>> lista
[1, 100, 2, 3, 4, 5]

>>> lista.pop(1)
100
>>> lista
[1, 2, 3, 4, 5]

>>> lista.remove(3)
>>> lista
[1, 2, 4, 5]
```

Métodos de ordenación: reverse, sort,

```
>>> lista.reverse()
>>> lista
[5, 4, 2, 1]

>>> lista.sort()
>>> lista
[1, 2, 4, 5]

>>> lista.sort(reverse=True)
>>> lista
[5, 4, 2, 1]

>>> lista=["hola","que","tal","Hola","Que","Tal"]
>>> lista.sort()
>>> lista
['Hola', 'Que', 'Tal', 'hola', 'que', 'tal']
>>> lista=["hola","que","tal","Hola","Que","Tal"]
>>> lista.sort(key=str.lower)
>>> lista
['hola', 'Hola', 'que', 'Que', 'tal', 'Tal']
```

Métodos de búsqueda: count, index

```
>>> lista.count(5)
1
```

```
>>> lista.append(5)
>>> lista
[5, 4, 2, 1, 5]
>>> lista.index(5)
0
>>> lista.index(5,1)
4
>>> lista.index(5,1,4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 5 is not in list
```

Método de copia: copy

```
>>> lista2 = lista1.copy()
```

Tipo de datos secuencia: Tuplas

Las tuplas (`tuple`): Sirven para los mismo que las listas (me permiten guardar un conjunto de datos que se pueden repetir y que pueden ser de distintos tipos), pero en este caso es un tipo inmutable.

Construcción de una tupla

Para crear una lista puedo usar varias formas:

- Con los caracteres (y):

```
>>> tupla1 = ()
>>> tupla2 = ("a",1,True)
```

- Utilizando el constructor tuple, que toma como parámetro un dato de algún tipo secuencia.

```
>>> tupla3=tuple()
>>> tuple4=tuple([1,2,3])
```

Empaquetado y desempaquetado de tuplas

Si a una variable se le asigna una secuencia de valores separados por comas, el valor de esa variable será la tupla formada por todos los valores asignados.

```
>>> tuple = 1,2,3
>>> tuple
(1, 2, 3)
```

Si se tiene una tupla de longitud k, se puede asignar la tupla a k variables distintas y en cada variable quedará una de las componentes de la tupla.

```
>>> a,b,c=tuple
>>> a
1
```

Operaciones básicas con tuplas

En las tuplas se pueden realizar las siguientes operaciones:

- Las tuplas se pueden recorrer.
- Operadores de pertenencia: `in` y `not in`.
- Concatenación: `+`
- Repetición: `*`
- Indexación
- Slice

Entre las funciones definidas podemos usar: `len`, `max`, `min`, `sum`, `sorted`.

Las tuplas son inmutables

```
>>> tupla = (1,2,3)
>>> tupla[1]=5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Métodos principales

Métodos de búsqueda: `count`, `index`

```
>>> tupla = (1,2,3,4,1,2,3)
>>> tupla.count(1)
2

>>> tupla.index(2)
1
>>> tupla.index(2,2)
5
```

Tipo de datos cadenas de caracteres

- Las cadenas de caracteres (`str`): Me permiten guardar secuencias de caracteres. Es un tipo inmutable. Las cadenas de caracteres en python3 están codificada con Unicode.

Definición de cadenas. Constructor str

Podemos definir una cadena de caracteres de distintas formas:

```
>>> cad1 = "Hola"
>>> cad2 = '¿Qué tal?'
>>> cad3 = '''Hola,
    que tal?'''
```

También podemos crear cadenas con el constructor `str` a partir de otros tipos de datos.

```
>>> cad1=str(1)
>>> cad2=str(2.45)
>>> cad3=str([1,2,3])
```

Operaciones básicas con cadenas de caracteres

Podemos realizar las siguientes operaciones:

- Las cadenas se pueden recorrer.
- Operadores de pertenencia: `in` y `not in`.
- Concatenación: `+`
- Repetición: `*`
- Indexación
- Slice

Entre las funciones definidas podemos usar: `len`, `max`, `min`, `sorted`.

Las cadenas son inmutables

```
>>> cad = "Hola que tal?"
>>> cad[4]="."
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Comparación de cadenas

Las cadenas se comparan carácter a carácter, en el momento en que dos caracteres no son iguales se compara alfabéticamente (es decir, se convierte a código unicode y se comparan).

Ejemplos

```
>>> "a">"A"
True
>>> ord("a")
97
>>> ord("A")
65

>>> "informatica">"informacion"
True

>>> "abcde">"abcdef"
False
```

Métodos principales de cadenas

Cuando creamos una cadena de caracteres estamos creando un objeto de la clase `str`, que tiene definido un conjunto de métodos:

```
cadena.capitalize    cadena.isalnum       cadena.join          cadena.rsplit
cadena.casefold      cadena.isalpha       cadena.ljust         cadena.rstrip
cadena.center        cadena.isdecimal     cadena.lower         cadena.split
cadena.count         cadena.isdigit       cadena.lstrip        cadena.splitlines
cadena.encode        cadena.isidentifier  cadena.maketrans     cadena.startswith
cadena.endswith      cadena.islower       cadena.partition     cadena.strip
cadena.expandtabs    cadena.isnumeric     cadena.replace       cadena.swapcase
cadena.find          cadena.isprintable   cadena.rfind         cadena.title
cadena.format        cadena.isspace       cadena.rindex        cadena.translate
cadena.format_map    cadena.istitle       cadena.rjust         cadena.upper
cadena.index         cadena.isupper       cadena.rpartition    cadena.zfill
```

Métodos de formato

```
>>> cad = "hola, como estás?"
>>> print(cad.capitalize())
Hola, como estás?

>>> cad = "Hola Mundo"
>>> print(cad.lower())
hola mundo

>>> cad = "hola mundo"
>>> print(cad.upper())
HOLA MUNDO

>>> cad = "Hola Mundo"
>>> print(cad.swapcase())
hOLA mUNDO

>>> cad = "hola mundo"
>>> print(cad.title())
Hola Mundo

>>> print(cad.center(50))
                    hola mundo
>>> print(cad.center(50,"="))
====================hola mundo====================

>>> print(cad.ljust(50,"="))
hola mundo========================================
>>> print(cad.rjust(50,"="))
========================================hola mundo

>>> num = 123
>>> print(str(num).zfill(12))
000000000123
```

Métodos de búsqueda

```
>>> cad = "bienvenido a mi aplicación"
>>> cad.count("a")
3
>>> cad.count("a",16)
2
>>> cad.count("a",10,16)
1

>>> cad.find("mi")
13
>>> cad.find("hola")
-1

>>> cad.rfind("a")
21
```

El método `index()` y `rindex()` son similares a los anteriores pero provocan una excepción `ValueError` cuando no encuentra la subcadena.

Métodos de validación

```
>>> cad.startswith("b")
True
>>> cad.startswith("m")
False
>>> cad.startswith("m",13)
True

>>> cad.endswith("ción")
True
>>> cad.endswith("ción",0,10)
False
>>> cad.endswith("nido",0,10)
True
```

Otras funciones de validación: `isalnum()`, `isalpha()`, `isdigit()`, `islower()`, `isupper()`, `isspace()`, `istitle()`,...

Métodos de sustitución

format

En la unidad **"Entrada y salida estándar"** ya estuvimos introduciendo el concepto de formateo de la cadenas. Estuvimos viendo que hay dos métodos y vimos algunos ejemplos del *nuevo estilo* con la función predefinida `format()`.

El uso del estilo nuevo es actualmente el recomendado (puedes obtener más información y ejemplos en algunos de estos enlaces: enlace1 y enlace2) y obtiene toda su potencialidad usando el método `format()` de las cadenas. Veamos algunos ejemplos:

```
>>> '{} {}'.format("a", "b")
'a b'
>>> '{1} {0}'.format("a", "b")
```

```
'b a'
>>> 'Coordenadas: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.
↪81W')
'Coordenadas: 37.24N, -115.81W'
>>> '{0:b} {1:x} {2:.2f}'.format(123, 223,12.2345)
'1111011 df 12.23'
>>> '{:^10}'.format('test')
'   test   '
```

Otros métodos de sustitución

```
>>> buscar = "nombre apellido"
>>> reemplazar_por = "Juan Pérez"
>>> print ("Estimado Sr. nombre apellido:".replace(buscar, reemplazar_por))
Estimado Sr. Juan Pérez:

>>> cadena = "   www.eugeniabahit.com   "
>>> print(cadena.strip())
www.eugeniabahit.com
>>> cadena="000000001230000000000"
>>> print(cadena.strip("0"))
123
```

De forma similar `lstrip(["caracter"])` y `rstrip(["caracter"])`.

Métodos de unión y división

```
>>> formato_numero_factura = ("N° 0000-0", "-0000 (ID: ", ")"
>>> print("275".join(formato_numero_factura))
N° 0000-0275-0000 (ID: 275)

>>> hora = "12:23"
>>> print(hora.rpartition(":"))
('12', ':', '23')
>>> print(hora.partition(":"))
('12', ':', '23')
>>> hora = "12:23:12"
>>> print(hora.partition(":"))
('12', ':', '23:12')
>>> print(hora.split(":"))
['12', '23', '12']
>>> print(hora.rpartition(":"))
('12:23', ':', '12')
>>> print(hora.rsplit(":",1))
['12:23', '12']


>>> texto = "Linea 1\nLinea 2\nLinea 3"
>>> print(texto.splitlines())
['Linea 1', 'Linea 2', 'Linea 3']
```

Tipo de datos mapa: diccionario

Los diccionarios son tipos de datos que nos permiten guardar valores, a los que se puede acceder por medio de una clave. Son tipos de datos mutables y los campos no tienen asignado orden.

Definición de diccionarios. Constructor dict

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
```

Si tenemos un diccionario vacío, al ser un objeto mutable, también podemos construir el diccionario de la siguiente manera.

```
>>> dict1 = {}
>>> dict1["one"]=1
>>> dict1["two"]=2
>>> dict1["three"]=3
```

Operaciones básicas con diccionarios

```
>>> a = dict(one=1, two=2, three=3)
```

- `len()`: Devuelve número de elementos del diccionario.

  ```
  >>> len(a)
  3
  ```

- Indexación: Podemos obtener el valor de un campo o cambiarlo (si no existe el campo nos da una excepción `KeyError`):

  ```
  >>> a["one"]
  1
  >>> a["one"]+=1
  >>> a
  {'three': 3, 'one': 2, 'two': 2}
  ```

- `del()`:Podemos eliminar un elemento, si no existe el campo nos da una excepción `KeyError`:

  ```
  >>> del(a["one"])
  >>> a
  {'three': 3, 'two': 2}
  ```

- Operadores de pertenencia: `key in d` y `key not in d`.

  ```
  >>> "two" in a
  True
  ```

Los diccionarios son tipos mutables

Los diccionarios, al igual que las litas, son tipos de datos mutable. Por lo tanto podemos encontrar situaciones similares a las que explicamos en su momentos con las listas.

```
>>> a = dict(one=1, two=2, three=3)
>>> a["one"]=2
>>> del(a["three"])
>>> a
{'one': 2, 'two': 2}


>>> a = dict(one=1, two=2, three=3)
>>> b = a
>>> del(a["one"])
>>> b
{'three': 3, 'two': 2}
```

En este caso para copiar diccionarios vamos a usar el método `copy()`:

```
>>> a = dict(one=1, two=2, three=3)
>>> b = a.copy()
>>> a["one"]=1000
>>> b
{'three': 3, 'one': 1, 'two': 2}
```

Métodos principales de diccionarios

```
dict1.clear       dict1.get         dict1.pop         dict1.update
dict1.copy        dict1.items       dict1.popitem     dict1.values
dict1.fromkeys    dict1.keys        dict1.setdefault
```

Métodos de eliminación: clear

```
>>> dict1 = dict(one=1, two=2, three=3)
>>> dict1.clear()
>>> dict1
{}
```

Métodos de agregado y creación: copy, dict.fromkeys, update, setdefault

```
>>> dict1 = dict(one=1, two=2, three=3)
>>> dict2 = dict1.copy()

>>> dict.fromkeys(["one","two","three"])
{'one': None, 'two': None, 'three': None}
>>> dict.fromkeys(["one","two","three"],100)
{'one': 100, 'two': 100, 'three': 100}

>>> dict1 = dict(one=1, two=2, three=3)
>>> dict2 = {'four':4,'five':5}
>>> dict1.update(dict2)
>>> dict1
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}

>>> dict1 = dict(one=1, two=2, three=3)
```

```
>>> dict1.setdefault("four",4)
4
>>> dict1
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> dict1.setdefault("one",-1)
1
>>> dict1
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

Métodos de retorno: get, pop, popitem, items, keys, values

```
>>> dict1 = dict(one=1, two=2, three=3)
>>> dict1.get("one")
1
>>> dict1.get("four")
>>> dict1.get("four","no existe")
'no existe'

>>> dict1.pop("one")
1
>>> dict1
{'two': 2, 'three': 3}
>>> dict1.pop("four")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'four'
>>> dict1.pop("four","no existe")
'no existe'

>>> dict1 = dict(one=1, two=2, three=3)
>>> dict1.popitem()
('one', 1)
>>> dict1
{'two': 2, 'three': 3}

>>> dict1 = dict(one=1, two=2, three=3)
>>> dict1.items()
dict_items([('one', 1), ('two', 2), ('three', 3)])

>>> dict1.keys()
dict_keys(['one', 'two', 'three'])
```

Recorrido de diccionarios

Podemos recorrer las claves:

```
>>> for clave in dict1.keys():
...     print(clave)
one
two
three
```

Podemos recorrer los valores:

```
>>> for valor in dict1.values():
...     print(valor)
1
2
3
```

O podemos recorrer ambos:

```
>>> for clave,valor in dict1.items():
...   print(clave,"->",valor)
one -> 1
two -> 2
three -> 3
```

Lectura y escritura de ficheros de textos

Función open()

La función open() se utiliza normalmente con dos parámetros (fichero con el que vamos a trabajar y modo de acceso) y nos devuelve un objeto de tipo fichero.

```
>>> f = open("ejemplo.txt","w")
>>> type(f)
<class '_io.TextIOWrapper'>
>>> f.close()
```

Modos de acceso

Los modos que podemos indicar son los siguientes:

Como podemos comprobar podemos trabajar con ficheros binarios y con ficheros de textos.

Objeto fichero

Al abrir un fichero con un determinado modo de acceso con la función `open()` se nos devuelve un objeto fichero. El fichero abierto siempre hay que cerrarlo con el método `close()`:

```
>>> f = open("ejemplo.txt","w")
>>> type(f)
<class '_io.TextIOWrapper'>
>>> f.close()
```

Se pueden acceder a las siguientes propiedades del objeto file:

- `closed`: retorna `True` si el archivo se ha cerrado. De lo contrario, `False`.
- `mode`: retorna el modo de apertura.
- `name`: retorna el nombre del archivo
- `encoding`: retorna la codificación de caracteres de un archivo de texto

Podemos abrirlo y cerrarlo en la misma instrucción con la siguiente estructura:

```
>>> with open("ejemplo.txt", "r") as archivo:
...     contenido = archivo.read()
>>> archivo.closed
True
```

Métodos principales

Métodos de lectura

```
>>> f = open("ejemplo.txt","r")
>>> f.read()
'Hola que tal\n'

>>> f = open("ejemplo.txt","r")
>>> f.read(4)
'Hola'
>>> f.read(4)
' que'
>>> f.tell()
8
>>> f.seek(0)
>>> f.read()
'Hola que tal\n'

>>> f = open("ejemplo2.txt","r")
>>> f.readline()
'Línea 1\n'
>>> f.readline()
'Línea 2\n'
>>> f.seek(0)
0
>>> f.readlines()
['Línea 1\n', 'Línea 2\n']
```

Métodos de escritura

```
>>> f = open("ejemplo3.txt","w")
>>> f.write("Prueba 1\n")
9
>>> print("Prueba 2\n",file=f)
>>> f.writelines(["Prueba 3","Prueba 4"])
>>> f.close()
>>> f = open("ejemplo3.txt","r")
>>> f.read()
'Prueba 1\nPrueba 2\n\nPrueba 3Prueba 4'
```

Recorrido de ficheros

```
>>> with open("ejemplo3.txt","r") as fichero:
...     for linea in fichero:
...         print(linea)
```

Excepciones

Errores sintácticos y errores de ejecución

Veamos un ejemplo de error sintáctico:

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

Una excepción o un error de ejecución se produce durante la ejecución del programa. Las excepciones se puede manejar para que no termine el programa. Veamos algunos ejemplos de excepciones:

```
>>> 4/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

>>> a+4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined

>>> "2"+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Hemos obtenido varias excpciones: ZeroDivisionError, NameError y TypeError. Puedes ver la lista de excepciones y su significado.

Manejando excepciones. try, except, else, finally

Veamos un ejemplo simple como podemos tratar una excepción:

```
>>> while True:
...   try:
...     x = int(input("Introduce un número:"))
...     break
...   except ValueError:
...     print ("Debes introducir un número")
```

1. Se ejecuta el bloque de instrucciones de `try`.
2. Si no se produce la excepción, el bloque de `except` no se ejecuta y continúa la ejecución secuencia.
3. Si se produce una excepción, el resto del bloque `try` no se ejecuta, si la excepción que se ha produce corresponde con la indicada en `except` se salta a ejecutar el bloque de instrucciones `except`.
4. Si la excepción producida no se corresponde con las indicadas en `except` se pasa a otra instrucción `try`, si finalmente no hay un manejador nos dará un error y el programa terminará.

Un bloque `except` puede manejar varios tipos de excepciones:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

Si quiero controlar varios tipos de excepciones puedo poner varios bloques `except`. Teniendo en cuenta que en el último, si quiero no indico el tipo de excepción:

```
>>> try:
...   print (10/int(cad))
... except ValueError:
...   print("No se puede converir a entero")
... except ZeroDivisionError:
...   print("No se puede divir por cero")
... except:
...   print("Otro error")
```

Se puede utilizar también la clausula `else`:

```
>>> try:
...   print (10/int(cad))
... except ValueError:
...   print("No se puede converir a entero")
... except ZeroDivisionError:
...   print("No se puede divir por cero")
... else:
...   print("Otro error")
```

Por último indicar que podemos indicar una clausula `finally` para indicar un bloque de instrucciones que siempre se debe ejecutar, independientemente de la excepción se haya producido o no.

```
>>> try:
...     result = x / y
... except ZeroDivisionError:
...     print("División por cero!")
... else:
...     print("El resultado es", result)
... finally:
...     print("Terminamos el programa")
```

Introducción a las funciones

Introducción a la programación estructurada y modular

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de ordenador, utilizando únicamente subrutinas (funciones o procedimientos) y tres estructuras: secuencia, alternativas y repetitivas.

La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación (divide y vencerás).

La programación estructural y modular se lleva a cabo en python3 con la definición de funciones.

Definición de funciones

Veamos un ejemplo de definición de función:

```
>>> def factorial(n):
...     """Calcula el factorial de un número"""
...     resultado = 1
...     for i in range(1,n+1):
...       resultado*=i
...     return resultado
```

Podemos obtener información de la función:

```
>>> help(factorial)
Help on function factorial in module __main__:
factorial(n)
    Calcula el factorial de un número
```

Y para utilizar la función:

```
>>> factorial(5)
120
```

Ámbito de variables.

Una variable local se declara en su ámbito de uso (en el programa principal y dentro de una función).

```
>>> def operar(a,b):
...     suma = a + b
...     resta = a - b
...     print(suma,resta)
...
>>> operar(4,5)
9 -1
>>> resta
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'resta' is not defined
```

Podemos definir variables globales, que serán visibles en todo el módulo. Se recomienda declararlas en mayúsculas:

```
>>> PI = 3.1415
>>> def area(radio):
...     return PI*radio**2
...
>>> area(2)
12.566
```

Parámetros formales y reales

- Parámetros formales: Son las variables que recibe la función, se crean al definir la función. Su contenido lo recibe al realizar la llamada a la función de los parámetro reales. Los parámetros formales son variables locales dentro de la función.

- Parámetros reales: Son la expresiones que se utilizan en la llamada de la función, sus valores se copiarán en los parámetros formales.

Llamadas a una función

Cuando se llama a una función se tienen que indicar los parámetros reales que se van a pasar. La llamada a una función se puede considerar una expresión cuyo valor y tipo es el retornado por la función. Si la función no tiene una instrucción `return` el tipo de la llamada sera `None`.

```
>>> def cuadrado(n):
...   return n*n

>>> a=cuadrado(2)
>>> cuadrado(3)+1
10
>>> cuadrado(cuadrado(4))
256
>>> type(cuadrado(2))
<class 'int'>
```

Cuando estamos definiendo una función estamos creando un objeto de tipo `function`.

```
>>> type(cuadrado)
<class 'function'>
```

Y por lo tanto puedo guardar el objeto función en otra variable:

```
>>> c=cuadrado
>>> c(4)
16
```

Conceptos avanzados sobre funciones

Tipos de argumentos: posicionales o keyword

Tenemos dos tipos de parámetros: los posiciónales donde el parámetro real debe coincidir en posición con el parámetro formal:

```
>>> def sumar(n1,n2):
...   return n1+n2
...
>>> sumar(5,7)
12
>>> sumar(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sumar() missing 1 required positional argument: 'n2'
```

Además podemos tener parámetros con valores por defecto:

```
>>> def operar(n1,n2,operador='+',respuesta='El resultado es '):
...   if operador=="+":
...     return respuesta+str(n1+n2)
...   elif operador=="-":
```

```
...     return respuesta+str(n1-n2)
...   else:
...     return "Error"
...
>>> operar(5,7)
'El resultado es 12'
>>> operar(5,7,"-")
'El resultado es -2'
>>> operar(5,7,"-","La resta es ")
'La resta es -2'
```

Los parámetros keyword son aquellos donde se indican el nombre del parámetro formal y su valor, por lo tanto no es necesario que tengan la misma posición. Al definir una función o al llamarla, hay que indicar primero los argumentos posicionales y a continuación los argumentos con valor por defecto (keyword).

```
>>> operar(5,7) # dos parámetros posicionales
>>> operar(n1=4,n2=6)   # dos parámetros keyword
>>> operar(4,6,respuesta="La suma es")  # dos parámetros posicionales y uno keyword
>>> operar(4,6,respuesta="La resta es",operador="-")    # dos parámetros posicionales␣
↪y dos keyword
```

Argumentos arbitrarios (*args y **kwargs)

Para indicar un número indefinido de argumentos posicionales al definir una función, utilizamos el símbolo $\star$:

```
>>> def sumar(n,*args):
...   resultado=n
...   for i in args:
...     resultado+=i
...   return resultado
...
>>> sumar(2)
2
>>> sumar(2,3,4)
9
```

Para indicar un número indefinido de argumentos keyword al definir una función, utilizamos el símbolo $\star\star$:

```
>>> def saludar(nombre="pepe",**kwargs):
...   cadena=nombre
...   for valor in kwargs.values():
...    cadena=cadena+" "+valor
...   return "Hola "+cadena
...
>>> saludar()
'Hola pepe'
>>> saludar("juan")
'Hola juan'
>>> saludar(nombre="juan",nombre2="pepe")
'Hola juan pepe'
>>> saludar(nombre="juan",nombre2="pepe",nombre3="maria")
'Hola juan maria pepe'
```

Por lo tanto podríamos tener definiciones de funciones del tipo:

```
>>> def f()
>>> def f(a,b=1)
>>> def f(a,*args,b=1)
>>> def f(*args,b=1)
>>> def f(*args,b=1,*kwargs)
>>> def f(*args,*kwargs)
>>> def f(*args)
>>> def f(*kwargs)
```

Desempaquetar argumentos: pasar listas y diccionarios

En caso contrario es cuando tenemos que pasar parámetros que lo tenemos guardados en una lista o en un diccionario.

Para pasar listas utilizamos el símbolo $\star$:

```
>>> lista=[1,2,3]
>>> sumar(*lista)
6
>>> sumar(2,*lista)
8
>>> sumar(2,3,*lista)
11
```

Podemos tener parámetros keyword guardados en un diccionario, para enviar un diccionario utilizamos el símbolo $\star\star$:

```
>>> datos={"nombre":"jose","nombre2":"pepe","nombre3":"maria"}
>>> saludar(**datos)
'Hola jose maria pepe'
```

Devolver múltiples resultados

La instrucción `return` puede devolver cualquier tipo de resultados, por lo tanto es fácil devolver múltiples datos guardados en una lista o en un diccionario. Veamos un ejemplo en que devolvemos los datos en una tupla:

```
>>> def operar(n1,n2):
...   return (n1+n2,n1-n2,n1*n2)

>>> suma,resta,producto = operar(5,2)
>>> suma
7
>>> resta
3
>>> producto
10
```

Funciones recursivas

Una función recursiva es aquella que al ejecutarse hace llamadas a ella misma. Por lo tento tenemos que tener "un caso base" que hace terminar el bucle de llamadas. Veamos un ejemplo:

```
>>> def factorial(numero):
...     if(numero == 0 or numero == 1):
...         return 1
```

```
...     else:
...         return numero * factorial(numero-1)
...
>>> factorial(5)
120
```

Módulos y paquetes

- Módulo: Cada uno de los ficheros `.py` que nosotros creamos se llama módulo. Los elementos creados en un módulo (funciones, clases, ...) se pueden importar para ser utilizados en otro módulo. El nombre que vamos a utilizar para importar un módulo es el nombre del fichero.
- Paquete: Para estructurar nuestros módulos podemos crear paquetes. Un paquete, es una carpeta que contiene archivos `.py`. Pero, para que una carpeta pueda ser considerada un paquete, debe contener un archivo de inicio llamado `__init__.py`. Este archivo, no necesita contener ninguna instrucción. Los paquetes, a la vez, también pueden contener otros sub-paquetes.

Ejecutando módulos como scripts

Si hemos creado un módulo, donde hemos definido dos funciones y hemos hecho un programa principal donde se utilizan dichas funciones, tenemos dos opciones: ejecutar ese módulo como un script o importar ese módulo desde otro, para utilizar sus funciones. Por ejemplo, si tenemos un fichero llamado `potencias.py`:

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

def cuadrado(n):
    return(n**2)
def cubo(n):
    return(n**3)
if __name__ == "__main__":
    print(cuadrado(3))
    print(cubo(3))
```

En este caso, cuando lo ejecuto como un script:

```
$ python3 potencias.py
```

El nombre que tiene el módulo es `__main__`, por lo tanto se ejecutará el programa principal.

Además este módulo se podrá importar (como veremos en el siguiente apartado) y el programa principal no se tendrá en cuenta.

Importando módulos: import, from

Para importar un módulo completo tenemos que utilizar las instrucción `import`. lo podemos importar de la siguiente manera:

```
>>> import potencias
>>> potencias.cuadrado(3)
9
>>> potencias.cubo(3)
27
```

Namespace y alias

Para acceder (desde el módulo donde se realizó la importación), a cualquier elemento del módulo importado, se realiza mediante el **namespace**, seguido de un punto (.) y el nombre del elemento que se desee obtener. En Python, un **namespace**, es el nombre que se ha indicado luego de la palabra import, es decir la ruta (namespace) del módulo.

Es posible también, abreviar los **namespaces** mediante un **alias**. Para ello, durante la importación, se asigna la palabra clave `as` seguida del alias con el cuál nos referiremos en el futuro a ese namespace importado:

```
>>> import potencias as p
>>> p.cuadrado(3)
9
```

Importando elementos de un módulo: from...import

Para no utilizar el **namespace** podemos indicar los elementos concretos que queremos importar de un módulo:

```
>>> from potencias import cubo
>>> cubo(3)
27
```

Podemos importar varios elementos separándolos con comas:

```
>>> from potencias import cubo,cuadrado
```

Podemos tener un problema al importar dos elementos de dos módulos que se llamen igual. En este caso tengo que utilizar **alias**:

```
>>> from potencias import cuadrado as pc
>>> from dibujos import cuadrado as dc
>>> pc(3)
9
>>> dc()
Esto es un cuadrado
```

Importando módulos desde paquetes

Si tenemos un módulo dentro de un paquete la importación se haría de forma similar. tenemos un paquete llamado `operaciones`:

```
$ cd operaciones
$ ls
__init.py__  potencias.py
```

Para importarlo:

```
>>> import operaciones.potencias
>>> operaciones.potencias.cubo(3)
27

>>> from operaciones.potencias import cubo
>>> cubo(3)
27
```

Ejercicios

Boletín 1: Ejercicios sencillos

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 los catetos de un triángulo rectángulo, calcular su hipotenusa.
5. Dados dos números, mostrar la suma, resta, división y multiplicación de ambos.
6. Escribir un programa que le pida una palabra al usuario, para luego imprimirla 1000 veces, con espacios intermedios.
7. Escribir un programa que le pregunte al usuario una cantidad de euros, una tasa de interés y un número de años y muestre como resultado la cantidad final a pagar. La fórmula a utilizar es:

```
Cn = C * (1 + x/100) ^ n
```

   Donde C es el capital inicial, x es la tasa de interés y n es el número de años a calcular.

8. Escribir un programa que convierta un valor dado en grados Fahrenheit a grados Celsius. Recordar que la fórmula para la conversión es:

```
F = 9/5 * C + 32.
```

9. Calcular la media de tres números pedidos por teclado.
10. Realiza un programa que reciba una cantidad de minutos y muestre por pantalla a cuantas horas y minutos corresponde.

    Por ejemplo: 1000 minutos son 16 horas y 40 minutos.

Solución boletín 1: Ejercicios sencillos

1. Escribir un programa que pregunte al usuario su nombre, y luego lo salude.

```
nombre=input("Dime tu nombre:")
print("Hola %s" % nombre)
```

2. Calcular el perímetro y área de un rectángulo dada su base y su altura.

```
base=float(input("Dime la base:"))
altura=float(input("Dime la altura:"))
perimetro = 2*base + 2*altura
area = base * altura
print("Resultado: Area=%.2f Perimetro=%.2f" % (area,perimetro))
```

3. Calcular el perímetro y área de un círculo dado su radio.

```
import math
radio=float(input("Dime el radio:"))
print("Resultado: Area=%.2f Perimetro=%.2f" % (math.pi*radio**2,2*math.pi*radio))
```

4. Dados los catetos de un triángulo rectángulo, calcular su hipotenusa.

```
import math
cateto1=float(input("Dime el cateto1:"))
cateto2=float(input("Dime el cateto2:"))
print("Hipotenusa=%.2f" % math.sqrt(cateto1**2+cateto2**2))
```

5. Dados dos números, mostrar la suma, resta, división y multiplicación de ambos.

```
num1=float(input("Numero1:"))
num2=float(input("Numero2:"))
print("Suma:%d,resta:%d,multiplicacion:%d,division:%.2f"%(num1+num2,num1-num2,
↪num1*num2,num1/num2))
```

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

```
palabra=input("Dime una palabra:")
print((palabra+" ")*1000)
```

7. Escribir un programa que le pregunte al usuario una cantidad de euros, una tasa de interés y un número de años y muestre como resultado la cantidad final a pagar. La fórmula a utilizar es:

```
Cn = C * (1 + x/100) ^ n
```

Donde C es el capital inicial, x es la tasa de interés y n es el número de años a calcular.

```
cant=float(input("Euros:"))
interes=float(input("Interes:"))
year=int(input("Years:"))
a_pagar=cant*(1+interes/100)**year
print("A pagar %.2f euros." % a_pagar)
```

8. Escribir un programa que convierta un valor dado en grados Fahrenheit a grados Celsius. Recordar que la fórmula para la conversión es:

```
F = 9/5 * C + 32.

gf=float(input("Grados Fahrenheit:"))
gc=(gf*5/9)-32
print("Grados C:%.2f" % gc)
```

9. Calcular la media de tres números pedidos por teclado.

```
num1=float(input("Numero1:"))
num2=float(input("Numero2:"))
num3=float(input("Numero3:"))
print("Media:%.2f" % ((num1+num2+num3)/3))
```

10. Realiza un programa que reciba una cantidad de minutos y muestre por pantalla a cuantas horas y minutos corresponde.

    Por ejemplo: 1000 minutos son 16 horas y 40 minutos.

```
minutos=int(input("Minutos:"))
print("Horas:%d - Minutos:%d" % (minutos/60,minutos%60))
```

Boletín 2: Ejercicios alternativas

1. Realiza un programa que pida dos números 'a' y 'b' e indique si su suma es positiva, negativa o cero.
2. Realiza un programa que pida una nota numéricas enteras e imprima sus equivalentes en texto (0-2 => MD, 3-4 => I, 5 => Suf, 6 => B, 7-8 => Not, 9-10 => Sob, otro => Error)
3. Escribe un programa que lea un número e indique si es par o impar.
4. Escribe un programa que pida un número entero entre uno y doce e imprima el número de días que tiene el mes correspondiente.
5. 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.
6. Algoritmo que pida tres números y los muestre ordenados.
7. Realiza un programa en python que pida por teclado el resultado (dato entero) obtenido al lanzar un dado de seis caras y muestre por pantalla el número en letras (dato cadena) de la cara opuesta al resultado obtenido.
   - Nota 1: En las caras opuestas de un dado de seis caras están los números: 1-6, 2-5 y 3-4.
   - Nota 2: Si el número del dado introducido es menor que 1 ó mayor que 6, se mostrará el mensaje: "ERROR: número incorrecto.".

   Ejemplo:

   Introduzca número del dado: 5

   En la cara opuesta está el "dos".
8. Programa que lea 3 datos de entrada A, B y C. Estos corresponden a las dimensiones de los lados de un triángulo. El programa debe determinar que tipo de triangulo es, teniedno en cuenta los siguiente:
   - Si se cumple Pitágoras entonces es triángulo rectángulo
   - Si sólo dos lados del triángulo son iguales entonces es isósceles.
   - Si los 3 lados son iguales entonces es equilátero.
   - Si no se cumple ninguna de las condiciones anteriores, es escaleno.
9. 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.
10. Programa que lea un carácter por teclado y compruebe si es una letra mayúscula.

Solución boletín 2: Ejercicios alternativas

1. Realiza un programa que pida dos números 'a' y 'b' e indique si su suma es positiva, negativa o cero.

```
num1=int(input("Número 1:"))
num2=int(input("Número 2:"))

if num1+num2>0:
    print("Suma positiva")
elif num1+num2<0:
    print("Suma negativa")
else:
    print("Suma es 0")
```

2. Realiza un programa que pida una nota numéricas enteras e imprima sus equivalentes en texto (0-2 => MD, 3-4 => I, 5 => Suf, 6 => B, 7-8 => Not, 9-10 => Sob, otro => Error)

```
nota=int(input("Nota:"))

if nota>=0 and nota<=2:
    print("MD")
elif nota==3 or nota==4:
    print("I")
elif nota==5:
    print("Suf")
elif nota==6:
    print("B")
elif nota==7 or nota==8:
    print("Not")
elif nota==9 or nota==10:
    print("Sob")
else:
    print("Error")
```

3. Escribe un programa que lea un número e indique si es par o impar.

```
num=int(input("Número:"))

if num%2==0:
    print("Número par")
else:
    print("Número impar")
```

4. Escribe un programa que pida un número entero entre uno y doce e imprima el número de días que tiene el mes correspondiente.

```
mes=int(input("Mes:"))

if mes==1 or mes==3 or mes==5 or mes==7 or mes==8 or mes==10 or mes==12:
    print("31 días")
elif mes==4 or mes==6 or mes==9 or mes==11 :
    print("30 días")
elif mes==2:
    print("28 o 29 días")
else:
    print("Mes incorrecto")
```

1. 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.

```
usuario=input("Usuario:")
clave=input("Contraseña:")

if usuario=="pepe" and clave=="asdasd":
    print("Has entrado en el sistema")
else:
    print("Error")
```

2. Algoritmo que pida tres números y los muestre ordenados.

```
num1=int(input("Número 1:"))
num2=int(input("Número 2:"))
```

```
num3=int(input("Número 3:"))

if num1>=num2 and num2>=num3:
    print(num1,num2,num3)
if num1>=num3 and num3>num2:
    print(num1,num3,num2)
if num2>num1 and num1>=num3:
    print(num2,num1,num3)
if num2>=num3 and num3>num1:
    print(num2,num3,num1)
if num3>num1 and num1>=num2:
    print(num3,num1,num2)
if num3>num2 and num2>num1:
    print(num3,num2,num1)
```

3. Realiza un programa en python que pida por teclado el resultado (dato entero) obtenido al lanzar un dado de seis caras y muestre por pantalla el número en letras (dato cadena) de la cara opuesta al resultado obtenido.
   - Nota 1: En las caras opuestas de un dado de seis caras están los números: 1-6, 2-5 y 3-4.
   - Nota 2: Si el número del dado introducido es menor que 1 ó mayor que 6, se mostrará el mensaje: "ERROR: número incorrecto.".

   Ejemplo:

   Introduzca número del dado: 5 En la cara opuesta está el "dos".

```
dado=int(input("Número del dado:"))

if dado==1:
    print("Seis")
elif dado==2:
    print("Cinco")
elif dado==3:
    print("Cuatro")
elif dado==4:
    print("Tres")
elif dado==5:
    print("Dos")
elif dado==6:
    print("Uno")
else:
    print("Error: número incorrecto")
```

4. Programa que lea 3 datos de entrada A, B y C. Estos corresponden a las dimensiones de los lados de un triángulo. El programa debe determinar que tipo de triangulo es, teniedno en cuenta los siguiente:
   - Si se cumple Pitágoras entonces es triángulo rectángulo
   - Si sólo dos lados del triángulo son iguales entonces es isósceles.
   - Si los 3 lados son iguales entonces es equilátero.
   - Si no se cumple ninguna de las condiciones anteriores, es escaleno.

```
Solución:

    lado1=float(input("A:"))
    lado2=float(input("B:"))
    lado3=float(input("C:"))
```

```
if lado1**2==(lado2**2+lado3**2):
    print("Rectangulo")
if lado1==lado2 and lado2==lado3:
    print("Equilátero")
elif lado1==lado2 or lado1==lado3 or lado2==lado3:
    print("Isósceles")
else:
    print("Escaleno")
```

1. 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.

```
year=int(input("Año:"))

if year%4==0 and year%100!=0 or year%400==0:
    print("Bisiesto")
else:
    print("No bisiesto")
```

2. Programa que lea un carácter por teclado y compruebe si es una letra mayúscula.

```
letra=input("Letra:")

if letra>="A" and letra<="Z":
    print("Mayuscula")
else:
    print("No mayuscula")
```

Boletín 3: Ejercicios bucles

1. Realiza un programa en python que muestre la tabla de multiplicar, convierte este pseudocódigo en el programa python:

```
INICIO
MIENTRAS numero < 0 O numero > 9
  ESCRIBIR "Dame un numero entre 0 y 9"
  LEER numero
  SI numero > 9 ENTONCES
    ESCRIBIR "Numero demasido alto"
  SINO
    SI numero < 0 ENTONCES
      ESCRIBIR "Numero demasiado bajo"
    FINSI
  FIN_SI
FIN_MIENTRAS
PARA i=0 HASTA 15 con INCREMENTO 1
  ESCRIBIR numero "X" i "=" numero*i
FIN_PARA
FIN
```

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. Algoritmo que pida números hasta que se introduzca un cero. Debe imprimir la suma y la media de todos los números introducidos.
5. Algoritmo que pida caracteres e imprima 'VOCAL' si son vocales y 'CONSONANTE' si no, el programa termina cuando se introduce un *espacio*.
6. Escribir un programa que imprima todos los números pares entre dos números que se le pidan al usuario.
7. Algoritmo que muestre la tabla de multiplicar de los números 1,2,3,4 y 5.
8. Escribe un programa que lea una lista de diez números y determine cuántos son positivos, y cuántos son negativos.
9. Escribe un programa que dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla el resultado de la potencia. No se puede utilizar el operador de potencia (**).
10. 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.

Solución boletín 3: Ejercicios bucles

1. Realiza un programa en python que muestre la tabla de multiplicar, convierte este pseudocódigo en el programa python:

```
INICIO
MIENTRAS numero < 0 O numero > 9
  ESCRIBIR "Dame un numero entre 0 y 9"
  LEER numero
  SI numero > 9 ENTONCES
    ESCRIBIR "Numero demasido alto"
  SINO
    SI numero < 0 ENTONCES
      ESCRIBIR "Numero demasiado bajo"
    FINSI
  FIN_SI
FIN_MIENTRAS
PARA i=0 HASTA 15 con INCREMENTO 1
  ESCRIBIR numero"X"i"="numero*i
FIN_PARA
FIN
```

```
num=int(input("Dame un numero entre 0 y 9:"))

while num<0 or num>9:
    if num>9:
        print("Numero demasido alto")
    else:
        print("Numero demasido bajo")
    num=int(input("Dame un numero entre 0 y 9:"))
for i in xrange(1,16):
    print("%d*%d=%d"%(num,i,i*num))
```

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),

```
num=int(input("Número:"))
fact=1
for i in xrange(2,num+1):
    fact=fact*i
print(fact)
```

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.

```
secreto=int(input("Número secreto:"))
num=int(input("Número:"))
while num!=secreto:
    if num>secreto:
        print("El número es menor")
    else:
        print("El número es mayor")
    num=int(input("Número:"))
print("Has acertado")
```

4. Algoritmo que pida números hasta que se introduzca un cero. Debe imprimir la suma y la media de todos los números introducidos.

```
suma=0
cont=0
num=int(input("Número:"))
while num!=0:
    cont=cont+1
    suma=suma+num
    num=int(input("Número:"))
media=float(suma)/cont
print("La suma es %d y la media es %f"%(suma,media))
```

5. Algoritmo que pida caracteres e imprima 'VOCAL' si son vocales y 'CONSONANTE' si no, el programa termina cuando se introduce un *espacio*.

```
letra=input("Letra:")
while letra!=" ":
    if letra=="a" or letra=="e" or letra=="i" or letra=="o" or letra=="u":
        print("Vocal")
    else:
        print("Consonante")
    letra=input("Letra:")
```

6. Escribir un programa que imprima todos los números pares entre dos números que se le pidan al usuario.

```
num1=int(input("Número:"))
num2=int(input("Número:"))
for i in range(num1,num2+1):
    if i%2==0:
        print(i,)
```

7. Algoritmo que muestre la tabla de multiplicar de los números 1,2,3,4 y 5.

```
for num1 in range(1,6):
    for num2 in range(1,11):
        print("%d*%d=%d"%(num1,num2,num1*num2))
```

8. Escribe un programa que lea una lista de diez números y determine cuántos son positivos, y cuántos son negativos.

```
cont_pos=0
cont_neg=0;
for cont in range(1,11):
    num=int(input("Número:"))
    if num>=0:
        cont_pos=cont_pos+1
    else:
        cont_neg=cont_neg+1
print("%d positivos,%d negativos"%(cont_pos,cont_neg)    )
```

9. Escribe un programa que dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla el resultado de la potencia. No se puede utilizar el operador de potencia (**).

```
base=float(input("Base:"))
exp=int(input("Exponente:"))
potencia=1
for cont in range(1,exp+1):
    potencia=potencia*base
print("Potencia=%f"%potencia)
```

10. 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.

```
num=int(input("Número:"))
primo = True
for cont in range(2,num):
    if num%cont==0:
        primo=False
if primo:
    print("Es primo")
else:
    print("No es primo")
```

Boletín 4: Ejercicios listas

1. Lee por teclado números y guardalo en una lista, el proceso finaliza cuando metamos un número negativo. Muestra el máximo de los números guardado en la lista, muestra los números pares.
2. Escribe un programa que permita crear una lista de palabras. Para ello, el programa tiene que pedir un número y luego solicitar ese número de palabras para crear la lista. Por último, el programa tiene que escribir la lista.
3. Dada una lista de números enteros (guarda la lista en una variable) y un entero *k*, escribir un programa que:
   - Cree tres listas listas, una con los menores, otra con los mayores y otra con los iguales a *k*.
   - Crea otra lista lista con aquellos que son múltiplos de *k*.
4. Realizar un programa que, dada una lista, devuelva una nueva lista cuyo contenido sea igual a la original pero invertida. Así, dada la lista ['Di', 'buen', 'día', 'a', 'papa'], deberá devolver ['papa', 'a', 'día', 'buen', 'Di'].
5. Escribe un programa que permita crear una lista de palabras y que, a continuación, pida una palabra y diga cuántas veces aparece esa palabra en la lista.
6. Escriba un programa que permita crear una lista de palabras y que, a continuación, pida dos palabras y sustituya la primera por la segunda en la lista.

7. Escriba un programa que permita crear una lista de palabras y que, a continuación, pida una palabra y elimine esa palabra de la lista.
8. Escriba un programa que permita crear dos listas de palabras y que, a continuación, elimine de la primera lista los nombres de la segunda lista.
9. Escriba un programa que permita crear una lista de palabras y que, a continuación, elimine los elementos repetidos (dejando únicamente el primero de los elementos repetidos).
10. Escribir una función que reciba una lista de elementos e indique si se encuentran ordenados de menor a mayor o no.

Solución boletín 4: Ejercicios listas

1. Lee por teclado números y guardalo en una lista, el proceso finaliza cuando metamos un número negativo. Muestra el máximo de los números guardado en la lista, muestra los números pares.

```
num=int(input("Número:"))
lista=[]
while num>0:
    lista.append(num)
    num=int(input("Número:"))

print(max(lista))
for n in lista:
    if n % 2 ==0:
        print(n)
```

2. Escribe un programa que permita crear una lista de palabras. Para ello, el programa tiene que pedir un número y luego solicitar ese número de palabras para crear la lista. Por último, el programa tiene que escribir la lista.

```
num=int(input("Número de palabras:"))
lista=[]
for i in range(1,num+1):
    lista.append(input("Palabra:"))

print(lista)
```

3. Dada una lista de números enteros (guarda la lista en una variable) y un entero *k*, escribir un programa que:
   - Cree tres listas listas, una con los menores, otra con los mayores y otra con los iguales a *k*.
   - Crea otra lista lista con aquellos que son múltiplos de *k*.

```
lista=[2,4,6,1,3,4,5,7,8]
k=4
lmenor=[]
ligual=[]
lmayor=[]
lmultiplo=[]
for num in lista:
    if num<k:
        lmenor.append(num)
    if num>k:
        lmayor.append(num)
    if num==k:
        ligual.append(num)
    if num%k==0:
```

```
            lmultiplo.append(num)
    print(lmayor)
    print(lmenor)
    print(igual)
    print(lmultiplo)
```

4. Realizar un programa que, dada una lista, devuelva una nueva lista cuyo contenido sea igual a la original pero invertida. Así, dada la lista ['Di', 'buen', 'día', 'a', 'papa'], deberá devolver ['papa', 'a', 'día', 'buen', 'Di'].

```
lista=['Di', 'buen', 'dia', 'a', 'papa']
print(lista[::-1])
```

5. Escribe un programa que permita crear una lista de palabras y que, a continuación, pida una palabra y diga cuántas veces aparece esa palabra en la lista.

```
palabra=input("Palabra:")
lista=[]
while palabra != " ":
    lista.append(palabra)
    palabra=input("Palabra:")

buscar=input("Palabra a buscar:")
print("La he encontrado %d veces"% lista.count(buscar))
```

6. Escriba un programa que permita crear una lista de palabras y que, a continuación, pida dos palabras y sustituya la primera por la segunda en la lista.

```
palabra=input("Palabra:")
lista=[]
while palabra != " ":
    lista.append(palabra)
    palabra=input("Palabra:")

buscar=input("Palabra a buscar:")
sustituir=input("Palabra a sustituir:")

cont=0
for cad in lista:
    if cad==buscar:
        lista[cont]=sustituir
    cont=cont+1

print(lista)
```

7. Escriba un programa que permita crear una lista de palabras y que, a continuación, pida una palabra y elimine esa palabra de la lista.

```
palabra=input("Palabra:")
lista=[]
while palabra != " ":
    lista.append(palabra)
    palabra=input("Palabra:")

eliminar=input("Palabra a eliminar:")

while eliminar in lista:
    lista.remove(eliminar)
```

```
print(lista)
```

8. Escriba un programa que permita crear dos listas de palabras y que, a continuación, elimine de la primera lista los nombres de la segunda lista.

```
palabra=input("Palabra lista 1:")
lista1=[]
while palabra != " ":
    lista1.append(palabra)
    palabra=input("Palabra lista 1:")

palabra=input("Palabra lista 2:")
lista2=[]
while palabra != " ":
    lista2.append(palabra)
    palabra=input("Palabra lista 2:")

for cad in lista2:
    while cad in lista1:
        lista1.remove(cad)

print(lista1)
```

9. Escriba un programa que permita crear una lista de palabras y que, a continuación, elimine los elementos repetidos (dejando únicamente el primero de los elementos repetidos).

```
palabra=input("Palabra lista 1:")
lista1=[]
while palabra != " ":
    lista1.append(palabra)
    palabra=input("Palabra lista 1:")

lista2=[]
for cad in lista1:
    if not cad in lista2:
        lista2.append(cad)
lista1=lista2[:]
print(lista1)
```

10. Escribir una función que reciba una lista de elementos e indique si se encuentran ordenados de menor a mayor o no.

```
num=int(input("Número:"))
lista=[]
while num>0:
    lista.append(num)
    num=int(input("Número:"))
lista2=lista[:]
lista.sort()
if lista==lista2:
    print("Ordenada")
else:
    print("No ordenada")
```

Boletín 5: Ejercicios cadenas

1. Crear un programa que lea por teclado una cadena, y muestre la siguiente información:
   - Imprima los dos primeros caracteres.
   - Imprima los tres últimos caracteres.
   - Imprima dicha cadena cada dos caracteres. Ej.: recta debería imprimir rca
   - Dicha cadena en sentido inverso. Ej.: hola mundo! debe imprimir !odnum aloh
   - Imprima la cadena en un sentido y en sentido inverso. Ej: reflejo imprime reflejoojelfer.
2. Crear un programa que lea por teclado una cadena y un carácter, e inserte el carácter entre cada letra de la cadena. Ej: separar y , debería devolver s,e,p,a,r,a,r
3. Crear un programa que lea por teclado una cadena y un carácter, y reemplace todos los espacios por el carácter. Ej: `mi archivo de texto.txt` y _ debería devolver `mi\_archivo\_de\_texto.txt`
4. Crear un programa que lea por teclado una cadena y un carácter, y reemplace todos los dígitos en la cadena por el carácter. Ej: su clave es: 1540 y X debería devolver su clave es: XXXX
5. Crear un programa que lea por teclado una cadena y un carácter, e inserte el caracter cada 3 dígitos en la cadena. Ej. 2552552550 y . debería devolver 255.255.255.0
6. Escribir una función que reciba una cadena que contiene un número entero y devuelva una cadena con el número y las separaciones de miles. Por ejemplo, si recibe 1234567890, debe devolver 1.234.567.890.
7. Crea un programa python que lea una cadena de caracteres y muestre la siguiente información:
   - La primera letra de cada palabra. Por ejemplo, si recibe Universal Serial Bus debe devolver USB.
   - Dicha cadena con la primera letra de cada palabra en mayúsculas. Por ejemplo, si recibe república argentina debe devolver República Argentina.
   - Las palabras que comiencen con la letra A. Por ejemplo, si recibe "Antes de ayer" debe devolver "Antes ayer".
8. Escribir funciones que dadas dos cadenas de caracteres:
   - Indique si la segunda cadena es una subcadena de la primera. Por ejemplo, cadena es una subcadena de subcadena.
   - Devuelva la que sea anterior en orden alfábetico. Por ejemplo, si recibe kde y gnome debe devolver gnome.
9. Escribir una función que reciba una cadena de unos y ceros (es decir, un número en representación binaria) y devuelva el valor decimal correspondiente.

Solución boletín 5: Ejercicios cadenas

1. Crear un programa que lea por teclado una cadena, y muestre la siguiente información:
   - Imprima los dos primeros caracteres.
   - Imprima los tres últimos caracteres.
   - Imprima dicha cadena cada dos caracteres. Ej.: recta debería imprimir rca
   - Dicha cadena en sentido inverso. Ej.: hola mundo! debe imprimir !odnum aloh
   - Imprima la cadena en un sentido y en sentido inverso. Ej: reflejo imprime reflejoojelfer.

```
cad=input("Cadena:")
print(cad[:2])
print(cad[-2:])
print(cad[::2])
print(cad[::-1])
print(cad+cad[::-1])
```

2. Crear un programa que lea por teclado una cadena y un carácter, e inserte el carácter entre cada letra de la cadena. Ej: separar y , debería devolver s,e,p,a,r,a,r

```
cad=input("Cadena:")
caracter=input("Caracter:")
print(cad.replace("",caracter)[1:-1])
```

3. Crear un programa que lea por teclado una cadena y un carácter, y reemplace todos los espacios por el carácter. Ej: mi archivo de texto.txt y _ debería devolver mi_archivo_de_texto.txt

```
cad=input("Cadena:")
caracter=input("Caracter:")
print(cad.replace(" ",caracter))
```

4. Crear un programa que lea por teclado una cadena y un carácter, y reemplace todos los dígitos en la cadena por el carácter. Ej: su clave es: 1540 y X debería devolver su clave es: XXXX

```
cad=input("Cadena:")
caracter=input("Caracter:")
for i in range(0,9):
    cad=cad.replace(str(i),caracter)
print(cad)
```

5. Crear un programa que lea por teclado una cadena y un carácter, e inserte el caracter cada 3 dígitos en la cadena. Ej. 2552552550 y . debería devolver 255.255.255.0

```
cad=input("Número:")
car=input("Caracter:")
cont=0
cad2=""
for c in cad:
    if cont!=0 and cont%3==0:
        cad2=cad2+car
    cad2=cad2+c
    cont=cont+1
print(cad2)
```

6. Escribir una función que reciba una cadena que contiene un número entero y devuelva una cadena con el número y las separaciones de miles. Por ejemplo, si recibe 1234567890, debe devolver 1.234.567.890.

```
num=int(input("Número:"))
cad=str(num)
cad2=""
cont=1
for caracter in cad[::-1]:
    cad2=caracter+cad2
    if cont%3==0:
        cad2="."+cad2
    cont=cont+1
print(cad2)
```

7. Crea un programa python que lea una cadena de caracteres y muestre la siguiente información:
   - La primera letra de cada palabra. Por ejemplo, si recibe Universal Serial Bus debe devolver USB.
   - Dicha cadena con la primera letra de cada palabra en mayúsculas. Por ejemplo, si recibe república argentina debe devolver República Argentina.
   - Las palabras que comiencen con la letra A. Por ejemplo, si recibe Antes de ayer debe devolver Antes ayer.

```
cad=input("Cadena:")

# La primera letra de cada palabra. Por ejemplo, si recibe Universal Serial␣
↪Bus debe devolver USB.
lista=cad.split(" ")
for palabra in lista:
    print(palabra[0],)
print("")
# Dicha cadena con la primera letra de cada palabra en mayúsculas. Por␣
↪ejemplo, si recibe república argentina debe devolver República Argentina.
for palabra in lista:
    print(palabra.capitalize(),)
print(""            )

# Las palabras que comiencen con la letra A. Por ejemplo, si recibe Antes␣
↪de ayer debe devolver Antes ayer.
for palabra in lista:
    if palabra.startswith("a") or palabra.startswith("A"):
        print(palabra,)
```

8. Escribir funciones que dadas dos cadenas de caracteres:
   - Indique si la segunda cadena es una subcadena de la primera. Por ejemplo, cadena es una subcadena de subcadena.
   - Devuelva la que sea anterior en orden alfábetico. Por ejemplo, si recibe kde y gnome debe devolver gnome.

```
cad1=input("Cadena 1:")
cad2=input("Cadena 2:")
```

```
if cad1.find(cad2)>-1:
    print("cad2 es subcadena de cad1")
else:
    print("cad2 no es subcadena de cad1")

if cad1<cad2:
    print(cad1)
else:
    print(cad2)
```

1. Escribir una función que reciba una cadena de unos y ceros (es decir, un número en representación binaria) y devuelva el valor decimal correspondiente.

```
cad1=input("Numero binario:")

cont=0
decimal=0
for num in cad1[::-1]:
    if num=="1":
        decimal=decimal+(2**cont)
```

```
    cont=cont+1
print(decimal)
```

Boletín 6: Ejercicios variados

1. Crear un programa de ordenador para gestionar los resultados de la quiniela de fútbol. Para ello vamos a utilizar dos listas:

   - Equipos: Que es una lista cuyos elementos son una lista con el nombre de los equipos que juegan el partido. En la quiniela se indican 15 partidos. Ejemplo: `equipos = [["Sevilla","Betis"], ["Madrid","Barcelona"],...]`
   - Resultados: Es una lista de enteros donde se indica el resultado. También tiene dos columnas (cada elemento es una lista), en la primera se guarda el número de goles del equipo que está guardado en la primera columna de la tabla anterior, y en la segunda los goles del otro equipo. Ejemplo: `resultados=[[3, 0],[0,0],...]`

   El programa ira pidiendo los nombres de los equipos de cada partido y el resultado del partido, a continuación se imprimirá la quiniela de esa jornada.

2. La letra del DNI se calcula a partir de su número. Para ello se divide el número entre 23 y el resto (que tiene que ser un número entre 0 y 22 se sustituye por la letra correspondiente de la siguiente tabla:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
T R W A G M Y F P D X  B  N  J  Z  S  Q  V  H  L  C  K  E
```

   Escribe un programa que te pida un número de DNI y una letra y te diga si es correcto o no.

3. El módulo `random` incluye la función `random()` que genera un número seudo-aleatorio entre 0 y 1:

```
>>> from random import random
>>> random()
0.51605767814317494
```

   Crea un programa que pida al usuario un número n y genere una lista con n elementos con valores aleatorios y muestre como salida:

   - La lista de los n números aletaorios con una precisión de 3 decimales.
   - La suma de todos los elementos con una precisión de 2 decimales.

   Nota: Los valores deben redondearse a la precisión solicitada, no truncarse.

   Ejemplo

```
Dame un numero: 4
La lista de 4 numeros aleatorios es: (0.123, 0.432, 0.335, 0.456)
La suma de estos 4 elementos es: 1.3
```

4. Escribe un programa que pida al usuario su fecha de nacimiento y le responda el día de la semana correspondiente (para ello debes utilizar la función adecuada del módulo calendar). Ejemplo:

   Ejemplo

```
Introduce tu fecha de nacimiento (DD-MM-YYYY): 29-02-1992
Naciste en sabado
```

5. Una dirección 6to4 es una dirección IPv6 reservada para equipos que tienen actualmente una dirección IPv4 pública. Un ejemplo de dirección 6to4 sería:

```
2002:503b:198:0:219:66ff:fea8:db3
```

El campo 2002: es fijo y el bloque importante para esta discusión es el que determina la parte de red de la dirección, es decir, los campos `503b:198` que son la representación hexadecimal de la dirección IPv4 correspondiente:

```
80.59.1.152 = 0x50.0x3b.0x1.0x98 = 503b:198
```

el resto de campos se corresponden con la dirección de subred y la dirección de host, y no son relevantes para este ejercicio.

Escribe un programa que pida una dirección IPv4 pública y nos dé la parte de red correspondiente de la dirección 6to4 asociada:

Ejemplo:

```
Dame una dirección IPv4 publica: 85.135.34.12
La parte de red 6to4 correspondiente es: 2002:5587:220
```

6. El Código Cuenta Cliente (CCC) es el código que identifica en España las cuentas corrientes de los clientes de bancos. El CCC tiene 20 dígitos en formato AAA-BBBB-CC- DDDDDDDDDD.

   - AAAA son cuatro dígitos que identifican la entidad bancaria.
   - BBBB son cuatro dígitos que identifican la oficina.
   - CC se denomina dígito de control (DC).
   - DDDDDDDDDD son 10 dígitos de la cuenta del cliente en el banco.

Según la Wikipedia: Los dígitos situados en las posiciones novena y décima se generan a partir de los demás dígitos del CCC, permitiendo comprobar la validez del mismo y reducir la posibilidad de errores de manipulación. El primero de ellos valida conjuntamente los códigos de entidad y de oficina; el segundo, valida el número de cuenta. Cada uno de los dígitos del DC se calcula utilizando el mismo algoritmo, para lo que se complementa con dos ceros a la izquierda la entidad y oficina.

La siguiente función en Python calcula el DC correspondiente para una lista de 10 número enteros:

```
def calcula_dc(lista):
"""Calcula el dígito de control de una CCC.
Recibe una lista con 10 numeros enteros y devuelve el DC
correspondiente"""
pesos = [1, 2, 4, 8, 5, 10, 9, 7, 3, 6]
aux = []
for i in range(10):
    aux.append(lista[i]*pesos[i])
resto = 11 - sum(aux) %11
if resto == 10:
    return 1
elif resto == 11:
    return 0
else:
    return resto
```

Por ejemplo:

```
>>> lista = [1, 6, 7, 0, 0, 0, 0, 3, 3, 2]
>>> calcula_dc(lista)
5
```

Escribe un programa que pida al usuario un CCC en el formato arriba indicado y compruebe su validez.

7. Escribe un programa para jugar al ahorcado.
   - Un jugador introduce una palabra secreta y otro jugador tratará de adivinarla.
   - La pantalla se limpia y aparece la horca vacía, el número de intentos y la palabra a acertar, donde cada letra se sustituye por un asterisco.

```
EL JUEGO DEL AHORCADO

  +---+
  |   |
      |
      |
      |
      |
  ======

Palabra a acertar :********
Fallos : 0
Letras utilizadas :

Introduce una letra ( '*' para resolver ):
```

Reglas del juego:

- El jugador puede cometer como máximo 6 fallos. Por cada fallo aparecerá un elemento más en la horca: cabeza, tronco, brazo izquierdo, brazo derecho, pierna izquierda y pierna derecha.
- Cada letra acertada aparecerá en la lista de letras utilizadas y se sustituirá en la posición que corresponda en la palabra a acertar.
- Una letra ya utilizada contará siempre como fallo (Esté o no en la palabra a acertar)
- No se permite el uso de vocales
- El jugador puede intentar resolver la palabra a acertar en cualquier momento tecleando la tecla *, tras lo cual se solicitará la palabra.
- El juego termina cuando el número de fallos es igual a 6 o cuando el jugador acierta la palabra, solicitando la resolución de la misma.
- Cualquier otro carácter que se introduzca: numero o signo de puntuación, contará como fallo.
- En un momento cualquiera el programa mostrará:

```
EL JUEGO DEL AHORCADO

   +---+
   |   |
   o   |
  /|   |
       |
       |
   ======
```

```
Palabra a acertar :y**t*p***t*
Fallos : 3
Letras utilizadas : y n m p t b

Introduce una letra ( '*' para resolver ):
```

- Se obtendrá mayor puntuación en el ejercicio si se estructura adecuadamente el código mediante el uso de funciones.
- Para que no se desplacen los caracteres a posiciones no deseadas, utiliza el triple apóstrofe con el print, por ejemplo:

```
>>> print('''
  +---+
  |   |
  o   |
 /|   |
      |
      |
  ======
''')
```

- Para limpiar la pantalla se puede utilizar (en GNU/Linux):

```
import os
os.system ('clear')
```

Ejercicio de funciones

Queremos hacer una librería con funciones que nos ayuden a trabajar con fechas. Crea un fichero `fechas.py` con las siguientes funciones, debes pensar el número de parámetros que reciben y que valor devuelven:

- `es_bisiesto`: Función que recibe un año y te devuelve si el año es bisiesto o no.
- `dias_del_mes`: Función que recibe un mes y un año, y te devuelve el número de días que tiene dicho mes en ese año.
- `calcular_dia_juliano`: Función que recibe una fecha (un día, un mes y un año) y devuelve el día juliano (cantidad de días que han pasado desde el 1 de enero).

A partir de estas funciones realiza, en otro fichero, un programa. Para poder realizar el programa tenemos que importar las funciones que vamos a utilizar, para ello:

```
from fechas import calcular_dia_juliano
```

El programa haría lo siguiente:

Se pide por teclado una fecha (una cadena de caracteres con formato dd/mm/aaaa), y se muestra por pantalla el día juliano al que corresponde.

¿Se te ocurre hacer alguna otra función a partir del código que hemos desarrollado?

Boletín 7: Ejercicios de funciones

1. Escribir dos funciones que permitan calcular:
   - La cantidad de segundos en un tiempo dado en horas, minutos y segundos.

- La cantidad de horas, minutos y segundos de un tiempo dado en segundos.

2. Realiza una función que dependiendo de los parámetros que reciba: convierte a segundos o a horas:
   - Si recibe un argumento, supone que son segundos y convierte a horas, mintos y segundos.
   - Si recibe 3 argumentos, supone que son hora, minutos y segundos y los convierte a segundos.
3. Queremos hacer una función que añada a una lista los contactos de una agenda. Los contactos se van a guardar en un diccionario, y al menos debe tener el campo de nombre, el campo del teléfono, aunque puede tener más campos. Los datos se irán pidiendo por teclado, se pedirá de antemanos cuantos contactos se van a guardar. Si vamos a guardar más información en el contacto, se irán pidiendo introduciendo campos hasta que introduzcamos el `*`.
4. Amplía el programa anterior para hacer una función de búsqueda, que reciba un conjunto de parámetros keyword y devuelve los contactos (en una lista) que coincidan con los criterios de búsqueda.
5. Realizar una función recursiva que reciba una lista y que calcule el producto de los elementos de la lista.

Solución boletín 7: Ejercicios de funciones

1. Escribir dos funciones que permitan calcular:
   - La cantidad de segundos en un tiempo dado en horas, minutos y segundos.
   - La cantidad de horas, minutos y segundos de un tiempo dado en segundos.

```
def calcular_segundos(horas,minutos,segundos):
    return horas*3600+minutos*60+segundos

def calcular_horas(segundos):
    horas = segundos // 3600
    segundos-=horas*3600
    minutos = segundos // 60
    segundos-=minutos*60
    return horas,minutos,segundos
```

1. Realiza una función que dependiendo de los parámetros que reciba: convierte a segundos o a horas:
   - Si recibe un argumento, supone que son segundos y convierte a horas, minutos y segundos.
   - Si recibe 3 argumentos, supone que son hora, minutos y segundos y los convierte a segundos.

```
def calcular(*args):
if len(args)==1:
    return calcular_horas(args[0])
elif len(args)==3:
    return calcular_segundos(*args)
else:
    raise TypeError("Se espera 1 o 3 parámetros")
```

2. Queremos hacer una función que añada a una lista los contactos de una agenda. Los contactos se van a guardar en un diccionario, y al menos debe tener el campo de nombre, el campo del teléfono, aunque puede tener más campos. Los datos se irán pidiendo por teclado, se pedirá de antemanos cuantos contactos se van a guardar. Si vamos a guardar más información en el contacto, se irán pidiendo introduciendo campos hasta que introduzcamos el `*`.

```
def guardar_agenda(l_agenda,**kwargs):
    l_agenda.append(kwargs)
    return l_agenda
```

```
def main():
    agenda=[]
    cantidad = int(input("¿Cuántos contactos vas a introducir?"))
    for i in range(cantidad):
        contacto={}
        contacto["nombre"]=input("Indica el nombre:")
        contacto["telefono"]=input("Indica el teléfono:")
        campo=input("Introuzca otro campo:")
        while campo!="*":
            contacto[campo]=input("Introuzca valor:")
            campo=input("Introuzca otro campo:")
        agenda=guardar_agenda(agenda,**contacto)
    print(agenda)

if __name__ == '__main__':
     main()
```

3. Amplía el programa anterior para hacer una función de búsqueda, que reciba un conjunto de parámetros keyword y devuelve los contactos (en una lista) que coincidan con los criterios de búsqueda.

```
def buscar(l_agenda,**kwargs):
    lista_aciertos=[]
    for contacto in l_agenda:
        aciertos=0
        for campo,valor in kwargs.items():
            if campo in contacto and contacto[campo]==valor:
                aciertos+=1
        if aciertos==len(kwargs):
            lista_aciertos.append(contacto)
    return lista_aciertos

def main():
    ...

    ## Búsqueda
    filtro={}
    campo=input("Introduzca un campo para buscar:")
    while campo!="*":
        filtro[campo]=input("Introduzca valor a buscar:")
        campo=input("Introduzca otro campo a buscar:")
    print(buscar(agenda,**filtro))
```

4. Realizar una función recursiva que reciba una lista y que calcule el producto de los elementos de la lista.

```
def multiplicar(lista):
    if len(lista)==1:
        return lista[0]
    else:
        return lista.pop()*multiplicar(lista)

if __name__ == '__main__':
    print(multiplicar([3,4,5]))
```

Boletín 8: Ejercicios funciones (2)

Necesitamos que se facture el uso de un teléfono. Nos informarán de la tarifa por segundo (en céntimos), cuántas comunicaciones se realizaron, la duración de cada comunicación expresada en horas, minutos y segundos. Como resultado deberemos informar la duración en segundos de cada comunicación y su costo.

Vamos a dividir este problemas en problemas más pequeños:

- Cada comunicación se expresa en horas, minutos y segundos, la tarifa es € por segundos, por lo tanto lo primero que vamos a solucionar es convertir las horas, minutos y segundos en segundos. Para ello vamos a crear una función llamada **pasar_a_segundos**. Piensa los parámetros de entrada que tiene esta función y el valor que devuelve. ¿De qué tipo son?
- Una vez que sabemos los segundos que ha tardado una comunicación y la tarifa por segundos vamos a crear una función llamada **calcular_coste** que nos calcule cuanto cuesta, en céntimos, la llamada. Piensa los parámetros y el valor devuelto de la función.
- Por último vamos a crear una función para convertir el coste en céntimos, en una cantidad de dinero expresada en euros y céntimos. Para ello creamos la función **convertir_a_euros**. Piensa los parámetros de entrada y los valores devueltos.

Ejercicios

1. Crea un programa que te pregunte por teclado la tarifa por segundos en céntimos, el número de comunicaciones que se han realizado, y te vaya pidiendo horas, minutos y segundo que han durado cada una de las comunicaciones. Finalmente te mostrará cuanto ha costado cada una de las comunicaciones y el total de dinero de todas las comunicaciones.
2. Realiza un programa que te informe de cuanto vale cada comunicación y el total de dinero de todas las comunicaciones. En esta ocasión los datos de la duración de las comunicaciones y la tarifa por segundos se encuentran en este fichero donde en la primera línea te encuentras la tarifa, y en las restantes la duración de cada una de las comunicaciones expresadas en horas, minutos y segundos.

Boletín 9: Ejercicios funciones (3)

Vamos a crear un programa que lea los resultados de los partidos de la liga española en el año 2016-2017, y nos devuelva información sobre estos datos.

El programa leerá la información del siguiente fichero, que tiene la siguiente estructura: Fecha, Equipo que juega en casa, Equipo que juega fuera, resultado al final del partido y resultado en el intermedio.

El programa ofrecerá un menú, para seleccionar la información deseada:

1. **Estadística de un equipo**: Nos pide por teclado el nombre de un equipo y nos muestra el número de goles que ha metido, los paridos ganados, perdidos y empatados.
2. **Nombres de equipos**: Nos muestra la lista de equipos que juegan.
3. **Clasificación de la liga**: Nos muestra los tres primeros equipos de la liga.
4. **Quiniela por fecha**: Introducimos una fecha y nos dice los resultados de la quiniela de ese día.
5. **Salir**

Para realizar este programa podemos realizar las siguientes funciones:

- `menu()`: Muestra el menú y devuelve un entero con la opción escogida.
- `LeerPartidos()`: Función que lee el fichero y devuelve una lista con los partidos (cada partido se va a guardar en un diccionario).
- `SumarGoles(equipo)`: Función que recibe un nombre de un equipo y devuelve el total de goles metidos.

- `InfoEquipos(equipo)`: Función que recibe un nombre de un equipo y devuelve una lista con los paridos ganados, perdidos y empatados.
- `Equipos()`: Función que devuelve una lista con todos los equipos.
- `Quiniela(dia,mes,año)`: Función que recibe el día, el mes y el año. Y devuelve una lista con los partidos y resultados de la quiniela.

1.2 Lenguajes de marcas

1.2.1 Introducción a los lenguajes de marcas

Teoría

- Introducción a los lenguajes de marcas
- Wikipedia - Lenguajes de marcado

Ejercicios

- books.csv
- books.html
  - HTML viewer
- books.xml
  - XML viewer
- books.json
  - JSON viewer
- books.yaml
  - YAML validator

Recursos

Git y GitHub

Utilizaremos git para realizar el control de versiones de los proyectos que realicemos en la asignatura. Deberás ir realizando los proyectos de forma incremental, y registrar los cambios (commits) realizados.

Los cambios realizados en el código fuente del proyecto se guardan en el repositorio local y después se publicarán en GitHub, en un repositorio remoto propio. Allí se podrán consultar todos los cambios realizados, sus fechas, sus descripciones, y se podrán compartir con otros compañeros y el profesor.

Los repositorios subidos a la versión gratuita de GitHub son públicos y convierten la cuenta personal de GitHub en un perfecto escaparate y portafolio de proyectos desarrollados. Cada vez son más las empresas que piden un buen expediente en GitHub. La visibilidad y transparencia de GitHub lo convierten también en una herramienta fundamental para aprender.

Un objetivo de la asignatura es participar en este movimiento de código abierto y colaborar con nuestros proyectos para que otros puedan aprender de ellos (y, al mismo tiempo, mejorar el portafolio y el curriculum personal de cara a futuros trabajos).

Empecemos

1. Crea una cuenta en GitHub y envía un correo a pledin.jd@gmail.com con la dirección de tu página GitHub y tu nombre completo. La forma de acceder a los repositorios remotos de GitHub va a ser por SSH, por lo tanto debes copiar tu clave pública a GitHub, para ello:
   - Copia el contenido de tu fichero ~/.ssh/id_rsa.pub, para ello: añade una nueva clave SSH en el apartado "SSH keys" de tu perfil en GitHub y pega el contenido de tu clave pública.
   - Si no tienes ese fichero, puedes generar una nueva clave ssh pública: http://librosweb.es/pro_git/capitulo_4/generando_tu_clave_publica_ssh.html.
2. Crea en GitHub un repositorio con el nombre **prueba** (inicializa el repositorio con un fichero README) y la descripción **Repositorio de prueba**.
3. Instala git en tu ordenador.

```
apt-get install git
```

4. Copia la url SSH del repositorio (no copies la URL https) y vamos a clonar el repositorio en nuestro ordenador.

```
git clone git@github.com:josedom24/prueba.git
```

5. Entramos en el nuestro repositorio local y configuramos git:

```
cd prueba
git config --global user.name "Pepito Pérez"
git config --global user.email pepito.perez@gmail.com
git commit --amend --reset-author
```

   Comprueba que dentro del repositorio que hemos creado se encuentra el fichero README.md, en este fichero podemos poner la descripción del proyecto.

6. Vamos a crear un nuevo fichero, lo vamos a añadir a nuestro repositorio local y luego lo vamos a sincronizar con nuestro repositorio remoto de GitHub. Cada vez que hagamos una modificación en un fichero lo podemos señalar creando un commit. Los mensajes de los commits son fundamentales para explicar la evolución de un proyecto. Un commit debe ser un conjunto pequeño de cambios de los ficheros del proyecto con una cierta coherencia.

```
echo "Esto es una prueba">ejemplo.txt
git add ejemplo.txt
git commit -m "He creado el fichero ejemplo.txt"
git push
```

7. Si modificas un fichero en tu repositorio local, no tienes que volver a añadirlo a tu repositorio (**git add**). Pero tienes que usar la opción -a al hacer el commit.

```
git commit -am "He modificado el fichero ejemplo.txt"
git push
```

8. Si quieres cambiar el nombre de un fichero o directorio de tu repositorio:

```
git mv ejemplo.txt ejemplo2.txt
git commit -am "He cambiado el nombre del fichero"
git push
```

9. Si quieres borrar un fichero de tu repositorio:

```
git rm ejemplo2.txt
git commit -am "He borrado el fichero ejemplo2"
git push
```

10. Puedes clonar tu repositorio de GitHub en varios ordenadores (por ejemplo, si quieres trabajar en tu casa y en el instituto), por lo tanto antes de trabajar en un repositorio local tienes que sincronizar los posibles cambios que se hayan producido en el repositorio remoto, para ello:

```
git pull
```

11. Para comprobar el estado de mi repositorio local:

```
git status
```

Si te quieres hacerte un experto de Pro Git, el libro oficial de Git

1.2.2 Introducción a XML

Teoría

- Introducción a XML

Ejercicios

Inventar un lenguaje XML para gráficos vectoriales

Crea un documento XML que represente un gráfico vectorial con los siguientes elementos (La base de la imagen será el vértice inferior izquierda (coordenadas x=0,y=0))

1. Un rectángulo de altura 120 píxeles, anchura 150 píxeles, con su vértice inferior izquierda en la posición (150,400)
2. Una circunferencia con su centro en la posición (400,400) y un radio de 100 píxeles 3- Una línea recta que vaya desde el vértice superior derecho del rectángulo hasta el centro de la circunferencia

Dibuja esos mismos elementos con la aplicación inkscape y compara el fichero resultante (formato SVG) con el que tú has creado.

Inventar un lenguaje XML para paneles de información de autopistas

Crea un documento XML bien formado que pueda utilizarse en un panel de información de una autopista, en concreto crea los siguientes eventos:

- Evento 1:
  - Señal de accidente
  - Punto kilométrico 42,200
  - Nombre de la vía: A-49
  - Sentido: Huelva
  - Corte de la vía: 1 carril
  - Fecha y hora del evento

  - Retención: Sí
  - Kilómetros de retención: 8
- Evento 2:
  - Señal de retención
  - Punto kilométrico: 550
  - Nombre de la vía: A-4
  - Sentido: Sevilla
  - Fecha y hora del evento
  - Retención: Sí
  - Kilómetros de retención: 3

Inventar un lenguaje XML para almacenar información meteorológica

Escribe un documento XML que almacene la siguiente información sobre la predicción metorológica de Dos Hermanas para el día 25 de Octubre de 2011:

- Probabilidad de precipitación: 55%
- Estado del cielo: Intervalos nubosos
- Dirección del viento:Suroeste
- Velocidad del viento: 10 Km/h
- Temperatura máxima: 21ºC
- Temperatura mínima: 10ºC
- Sensación térmica máxima: 21ºC
- Sensación térmica mínima: 10ºC
- Humedad relativa máxima: 90%
- Humedad relativa mínima: 50%
- Índice Ultravioleta máximo: 3

Inventar un lenguaje XML para facturas

Queremos estructurar la información que genera un proceso de facturación de una empresa en un fichero XML. Para ello tenemos que tener en cuenta los siguientes aspectos:

- Cada factura tiene un código.
- La factura también necesita la fecha de emisión.
- En la factura aparecen los datos del cliente (dni, nombre, dirección, código postal, población).
- De cada producto que se ha comprado debe aparecer la cantidad de productos comprados, la denominación y el precio ubnitario.
- Se debe guardar el IVA de cada producto.
- Si es necesario se indicara un descuento al importe total de la factura.

Razona la siguiente pregunta: ¿Es necesario guradar el importe total por producto y el importe total de la factura?

Inventar un lenguaje XML para almacenar los datos de préstamo de los libros de una biblioteca

Se quiere guardar en un fichero XML la información generada por los prestamos de libros en una biblioteca. Para ello ten en cuenta los siguientes aspectos:

- De cada libro guardamos varios datos: código ISBN, nombre, editorial, año de publicación, autor.
- De cada libro podemos tener uno o varios ejemplares. Cada ejemplar se diferencia de otro por un código numérico.
- Se prestan los ejemplares, y de cada prestamos hay que indicar el ejemplar del libro, el socio al que se ha prestado y la fecha de prestamo.
- Del socio hay que guadar DNI, nombre y dirección.

Recursos

- World Wide Web Consortium (W3C)
- Wikipedia - XML

1.2.3 Python y XML

Teoría

- Trabajar con ficheros xml desde python (1ª parte)
- Trabajar con ficheros xml desde python (2ª parte)
- Tutorial de XPATH
  - Repositorio de github: xpath
  - Code Beauty - XPath Tester

Ejercicios

Ejercicio: Provincias y poblaciones

Utilizando el fichero provinciasypoblaciones.xml, crea distintas funciones en python, utilizando la libreria lxml, que realicen las siguientes tareas:

1. Función que devuelve una lista con los nombres de las provincias.
2. Función que devuelve una lista con todos las poblaciones.
3. Función que devuelve las provincias y el total de poblaciones que tiene cada uno. Piensa la estructura de datos que va a devolver la función.
4. Función que recibe el nombre de una provincia y devuelve la lista de poblaciones.
5. Función que recibe el nombre de una población y te devuelve la provincia donde se encuentra.
6. Función que recibe una lista distintos identificadores de provincias, y te devuelve las provincias que corresponden a cada identificador, y sus poblaciones.
7. Función que reciba el nombre de una provincia devuelva las "ciudades grandes" (atributo c="1").

8. Función que reciba una localidad y te devuelva si es "ciudad grande" (atributo c="1") o no de provincia. Si es "ciudad grande" de provincia te devuelve el nombre de la provincia.

Crea en otro fichero el programa principal que te muestre el siguiente menú:

1. Mostrar todas las provincias: Muestra por pantalla el nombre de todas las provincias.
2. Mostrar todas las poblaciones: Muestra por pantalla el nombre de todas las poblaciones.
3. Mostrar provincias y número de poblaciones: Muestra por pantalla el nombre de todas las provincias y la cantidad de poblaciones.
4. Mostrar las poblaciones: Lee una provincia por teclado y te muestra el nombre de las poblaciones. Si la provincia no existe te da un error.
5. Mostrar la provincia: Lee una población por teclado y te muestra el nombre de la provincia. Si la población no existe te da un error.
6. Información por identificador: Pide un conjunto de identificadores de provincias, y te muestra por pantalla las provincias correspondientes y sus poblaciones.
7. Ciudades grandes: Lee una provincia por teclado y te muestra las poblaciones grances ("ciudades grandes"). Si la provincia no existe te da un error.
8. Es ciudad grande: Lee una población y si es ciudad grande te muestra el nombre de la provincia, sino te dice que no es una ciudad grande.

Ejercicio: Usuarios de la moodle

Utilizando el fichero users.xml, realiza difierentes funciones python para obtener la la siguiente información:

1. Lista de usuarios (profesores y alumnos) matriculados en el módulo "Lenguaje de marcas".
2. Mostrar el nombre de usuario de los usarios que han cambiado su avatar en nuestra plataforma moodle (Elemento picture)
3. Mostrar los correos electrónicos de los usuarios que han accedido por última vez desde fuera del instituto (elemento lastip)
4. Pedir una cadena por teclado y mostrar todos los usarios cuyo nombre empieza por dicha cadena (Ejemplo: si meto la cadena "A" mostrará todos los usuarios cuyo nombre empeiza por A...)
5. Mostrar la lista de usuarios agrupados por localidades.

```
Ejemplo:

Sevilla
-------
Alberto Molina
...
...
...

Utrera
------
José Domingo Muñoz
...
...

Dos Hermanas
------------
```

```
...
...
...
```

6. Mostrar la lista de ususarios ordenada por el último acceso (Elemento lastaccess)

Ejercicio: Información meteorológica de Sevilla

Utilizando el fichero sevilla.xml, realiza varias funciones en python que nos den la siguiente información:

- Latitud y longitud de Sevilla
- Temperatura, viento y humedad actuales en Sevilla
- Temperatura máxima y mínima de los próximos 7 días (pronostico) hay que indicar también la fecha.
- Una función que reciba una fecha y una hora y si existe en el pronostico devuelva temperatura, viento, precipitación, presión y humedad.

Realiza los mismos ejercicios utilizando expresiones XPath.

Ejercicio: Mapa de Utrera

Utiliza el fichero Utrera.xml, realiza varias funciones python que devuelvan la siguiente información:

- Calles de Utrera (elementos "way" con un elemento hijo "tag" que tenga el atributo k="highway").Mostrar el `id`.
- Calles de Utrera con nombre (elementos "way" con un elemento hijo "tag" que tenga el atributo `k="highway"` y el atributo `k="name"`). Mostrar el nombre.
- Nodos de Utrera (mostrar el `id` editados por el usuario "384182" y que tienen algún elemento hijo `tag`.
- Número de supermecados de Utrera (hay que buscar elementos del tipo `<tag k="shop" v="supermarket"/>`).

Realiza los mismos ejercicios utilizando expresiones XPath.

Ejercicios resueltos

- Ejercicio 1
- Ejercicio 1 xpath
- Ejercicio 2
- Ejercicio 2 xpath
- Ejercicio 3
- Ejercicio 3 xpath
- Ejercicio 4
- Ejercicio 4 xpath

Recursos

- lxml - XML and HTML with Python
- lxml.de - The lxml.etree Tutorial
- nmt.edu - Python XML processing with lxml

1.2.4 Esquemas en XML

Teoría

- Introducción al lenguaje XSD (XML Schema Definition)

Ejercicios

Ejercicio Esquemas XML

La herramienta de validación desde línea de comandos que utilizaremos se llama `xmllint`, que pertenece al paquete debian `libxml2-utils` Para validar un fichero xml con un fichero XSD (XML Schema Definition) ejecutamos el siguiente comando:

```
xmllint fichero.xml --schema fichero.xsd
```

Tambien podemos utilizar un programa gráfico que se llama xsddiagram.

Ejemplo 1: Tipos simples

ejemplo1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<nota>hola</nota>
```

ejemplo1.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="nota" type="xs:string"/>
</xs:schema>
```

Ejemplo 2: Tipos complejos

libro1.xml

```
<libro>
    <autor>Miguel de Cervantes Saavedra</autor>
    <titulo>El Quijote de la Mancha</titulo>
</libro>
```

libro1.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="libro">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="autor" type="xs:string" />
            <xs:element name="titulo" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
    </xs:element>
</xs:schema>
```

Ejemplo 3: Tipos complejos II

libro2.xml

```
<biblioteca>
    <libro>
        <autor>Miguel de Cervantes Saavedra</autor>
        <titulo>El Quijote de la Mancha</titulo>
    </libro>
    <libro>
        <autor>Pablo Neruda</autor>
        <titulo>Veinte poemas de amor y una canción desesperada</titulo>
    </libro>
</biblioteca>
```

libro2.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="biblioteca">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="autor" />
                        <xs:element name="titulo" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Ejemplo 4: Restricciones

libro3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
    <libro>
```

```
        <autor>Miguel de Cervantes Saavedra</autor>
        <titulo>El Quijote de la Mancha</titulo>
        <codigo>123</codigo>
    </libro>
    <libro>
        <autor>Pablo Neruda</autor>
        <titulo>Veinte poemas de amor y una canción desesperada</titulo>
        <codigo>124</codigo>
    </libro>
</biblioteca>
```

libro3.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="biblioteca">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="autor" />
                        <xs:element name="titulo" />
                        <xs:element name="codigo">
                            <xs:simpleType>
                                <xs:restriction base="xs:integer">
                                    <xs:minInclusive value="1"/>
                                    <xs:maxInclusive value="9999"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Ejemplo 5: Restricciones II

libro4.xml

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
    <libro>
        <autor>Miguel de Cervantes Saavedra</autor>
        <titulo>El Quijote de la Mancha</titulo>
        <codigo>123</codigo>
        <ubicacion>estantería 1</ubicacion>
    </libro>
    <libro>
        <autor>Pablo Neruda</autor>
        <titulo>Veinte poemas de amor y una canción desesperada</titulo>
        <codigo>124</codigo>
        <ubicacion>estantería 11</ubicacion>
```

```
    </libro>
</biblioteca>
```

libro4.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="biblioteca">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="autor" />
                        <xs:element name="titulo" />
                        <xs:element name="codigo">
                            <xs:simpleType>
                                <xs:restriction base="xs:integer">
                                    <xs:minInclusive value="1"/>
                                    <xs:maxInclusive value="9999"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="ubicacion">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="estantería 1"/>
                                    <xs:enumeration value="estantería 2"/>
                                    <xs:enumeration value="estantería 3"/>
                                    <xs:enumeration value="estantería 4"/>
                                    <xs:enumeration value="estantería 5"/>
                                    <xs:enumeration value="estantería 6"/>
                                    <xs:enumeration value="estantería 7"/>
                                    <xs:enumeration value="estantería 8"/>
                                    <xs:enumeration value="estantería 9"/>
                                    <xs:enumeration value="estantería 10"/>
                                    <xs:enumeration value="estantería 11"/>
                                    <xs:enumeration value="estantería 12"/>
                                    <xs:enumeration value="estantería 13"/>
                                    <xs:enumeration value="estantería 14"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Ejemplo 6: Atributos

libro5.xml

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
    <libro codigo="123" ubicacion="estantería 1">
        <autor>Miguel de Cervantes Saavedra</autor>
        <titulo>El Quijote de la Mancha</titulo>
    </libro>
    <libro codigo="1023" ubicacion="estantería 3">
        <autor>Pablo Neruda</autor>
        <titulo>Veinte poemas de amor y una canción desesperada</titulo>
    </libro>
</biblioteca>
```

libro5.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="biblioteca">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="autor" />
                        <xs:element name="titulo" />
                    </xs:sequence>
                    <xs:attribute name="codigo">
                        <xs:simpleType>
                            <xs:restriction base="xs:integer">
                                <xs:minInclusive value="1"/>
                                <xs:maxInclusive value="9999"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                    <xs:attribute name="ubicacion">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="estantería 1"/>
                                <xs:enumeration value="estantería 2"/>
                                <xs:enumeration value="estantería 3"/>
                                <xs:enumeration value="estantería 4"/>
                                <xs:enumeration value="estantería 5"/>
                                <xs:enumeration value="estantería 6"/>
                                <xs:enumeration value="estantería 7"/>
                                <xs:enumeration value="estantería 8"/>
                                <xs:enumeration value="estantería 9"/>
                                <xs:enumeration value="estantería 10"/>
                                <xs:enumeration value="estantería 11"/>
                                <xs:enumeration value="estantería 12"/>
                                <xs:enumeration value="estantería 13"/>
                                <xs:enumeration value="estantería 14"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
</xs:schema>
```

Ejercicio: esquema XML para facturas

Utilizando como base el fichero factura.xml que se te facilita, crea un esquema en formato XSD (XMLSchema) que permita validar facturas como la anterior y que tengan en cuenta también los siguientes aspectos:

1. Toda factura tiene un identificador obligatorio
2. Los elementos emision, vendedor, cliente y articulo son obligatorios, mientras que descuento es opcional
3. El elemento articulo puede aparecer varias veces, los demás sólo una vez
4. vendedor tiene el atributo obligatorio id y el elemento obligatorio nombre.
5. cliente tiene el atributo obligatorio id y los elementos nombre, direccion y teléfono.
6. nombre y direccion son obligatorios, mientras que telefono es opcional y además puede aparecer más de una vez
7. Cada articulo tiene los atributos obligatorios id e iva y los elementos obligatorios denominacion, precio y cantidad
8. Descuento puede contener sólo un numero entre 0 y 100

Tienes que incluir todas las restricciones posibles para que los valores permitidos se ajusten al máximo a los adecuados

Solución

- factura.sxd

Ejercicio: esquema XML para panales de autopistas

Implementa un esquema XML adecuado para almacenar información de paneles de autopista como el del fichero autopistas.xml con las siguientes características:

1. El elemento raíz se denomina "paneles"
2. "paneles" contiene un número indeterminado de elementos denominados "evento"
3. Cada "evento" tiene dos atributos obligatorios "tipo" y "fecha"
    - "tipo" puede tener sólo dos valores "Accidente" o "Retención"
    - "fecha" tiene que ser de tipo fecha normalizado
4. Cada "evento" tiene los elementos obligatorios "via", "pk" y "sentido" y los elementos opcionales "retencion" y "corte".
    - "via" tiene los elementos opcionales "nombre", "ref", "origen", "destino", "doble" y "carrilescortados".
        - "nombre" puede contener una cadena de texto de 50 caracteres como máximo
        - "ref" está compuesto por una cadena de 3 caracteres como máximo, un guión y un número comprendido entre 1 y 9999, por ejemplo SE-4100 o A-92.
        - "origen" y "destino" puede contener una cadena de texto de 30 caracteres como máximo.
        - "doble" no tiene contenido

    - "carriles" tiene que ser un número entre 1 y 8
- "pk" es un número con tres decimales.
- "sentido" puede tener los valores -1, 0 ó 1.
- "retencion" tiene el atributo opcional "long" que es del mismo tipo que pk
- "carrilescortados" tiene el atributo obligatorio "valor" que tiene que ser un número entre 1 y 8.

Recursos

- Eric Wilde. XML Foundations: XSD - Part I
- Eric Wilde. XML Foundations: XSD - Part II
- [Eric Wilde. XML Foundations: XSD - Part III]](http://dret.net/lectures/xml-fall10/xsd-3)
- openStreetMap API v0.6/XSD
- aemet - XSD

1.2.5 JSON

Teoría

- Gestionar ficheros json

Ejercicios

Ejercicio: Librería

Escribe distintas funciones en en python que lean el fichero json libreria.json con datos de nuestra librería y muestre la siguiente información:

- ¿Cuántos libros hay en la librería?
- Recibe un límite inferior y superior para el precio y muestra todos los libros cuyo precio esta en ese intervalo.
- Recibe una cadena por tecldo, y muestra el título y el año de publicación de los libros cuyo título empiece por la cadena introducida.
- Devuelve todos los títulos de los libros con la lista de sus autores.

Ejercicio: Pruebas nivel de idioma

Escribe distitnas funciones python que lea el siguiente fichero json sobre pruebas de nivel de idiomas y obtenga la siguiente información:

- ¿Cuantas pruebas de idiomas están descritas en el documento?
- Devuelve el título de las pruebas de nivel que van a durar más de dos horas.
- De las pruebas de tipo “No Presencial” devuelve la URL de información.
- Recibe el código de la prueba “ID” y muestra su título y profesores.
- Para cada uno de las pruebas, muestra su título y sus profesores.

Ejercicio: Provincias y municipios

Utilizando el fichero ej3.json de provincias y municipios de España, crea distintas funciones en python, utilizando la libreria json, que realicen las siguientes funciones:

- Función que devuelve todas las provincias.
- Función que devuelve todos los municipios.
- Función que devuelve la lista de provincias y el total de municipios que tiene cada una.
- Función que recibe el nombre de una provincia y devuelve sus municipios.
- Función que recibe el nombre de un municipio y te devuelve la provincia donde se encuentra.
- En una lista tenemos distintos identificadores de provincias, devolver el nombre de las provincias y todos los municipios correspondientes a los identificadores que se encuentran en la lista.

Recursos

- Introducing JSON
- XML2JSON
- Python: Leer un Json

1.2.6 YAML

Teoría

- Aprende yaml en Y minutos
- Trabajando con YAML desde python3: pyyaml

Ejercicios

Ejercicio: zips of USA

Descarga el fichero usa.yaml. Este fichero es similar al del ejercicio 5 de la entrega 6 pero en formato yaml. Se trata de un listado de los códigos postales de EEUU en formato YAML. Realiza las siguientes funciones en python:

- Cuenta el número de códigos postales que aparecen
- Cuenta el número de códigos postales de cada estado
- Obtén la URL del mapa de OpenStreetMap de la ciudad de "Akaska" en el estado de Dakota del Sur (SD). Nota: Las coordenadas que aparecen en el fichero zips.json vienen en formato [longitud,latitud] y la url genérica que utiliza OpenStreetMap es:

```
http://www.openstreetmap.org/#map=zoom/latitud/longitud
```

Ejercicio: Configuración de ansible

Ansible es un software que nos permite la configuración automática de nuestra infraestructura. Las tareas que se van a ejecutar de forma automática en nuestros servidores se escriben en playbook que son ficheros escrito en YAML.

El fichero ansible.yaml es un playbook para instalar mysql en un servidor. Crea distintas funciones en python que nos devuelvan la siguiente información:

- Cuantas tareas se ejecutan en el playbook.
- La lista de los nombres de las distintas tareas.
- En la primera tarea se realiza la instalación de paquetes. DEvuelve el nombre de los paquetes que se instalan.
- Devuelve la contraseña que se va a asignar al root de mysql.
- Devuelve si esta receta tiene handlers (manejadores para reiniciar los servicios).

Recursos

- YAML Ain't Markup Language (YAML™) Version 1.2
- json2yaml
- YAML Lint
- PyYAML Documentation

1.2.7 HTML5 y CSS

Teoría

- Índice
- Curso de HTML5 desde cero
- Tutorial HTML5 de w3c
- librosweb.es - Introducción a CSS
- Tutorial de css de w3c

Ejercicios

- Ejemplo de formulario HTML5
- Aprende a crear tu propio sitio Web con HTML5 y CSS3
  - Ficheros
- Proyecto HTML5 y CSS3

Recursos

- Ejemplos de formularios HTML5
- W3C validator

1.3 Proyecto: Servicios Web

1.3.1 Web Services

Teoría

- Presentación servicios web
- Requests: HTTP for Humans
- Instalación de módulos en entornos virtuales con python3

oauth

- Repositorio aplicación oauth
- Aplicación oauth

Ejercicios

- Ejemplos de peticiones a una API RESTful
- Ejercicios librería requests
  - Solución ejercicios librería requests
- SWAPI: The Star Wars API
- Ejercicio: Zomato API

Recursos

- Introducción a los servicios web RESTful
- Vídeo: ¿Qué es REST?
- ProgrammableWeb - APIs, Mashups and the Web as Platform
- Public APIs

1.3.2 Flask: (Miniframework python para desarrollar páginas web)

Teoría

- flask: Miniframework python para desarrollar páginas web (1ª parte)
- flask: Enrutamiento (2ª parte)
- flask: Trabajando con peticiones y respuestas (3ª parte)
- flask: Plantillas con jinja2 (4ª parte)

Heroku

- Despliegue de aplicación flask en un PaaS Heroku

Ejercicios

- Ejercicios con Flask

Recursos

- Documentación de flask
- Curso flask (openwebinars)

Vídeos

1.3.3 Proyecto final de curso

Proyecto 3ª evaluación: Utilización de API web

- Enunciado del proyecto
- Proyectos de años anteriores