Iniciación a la programación con Python

Objetivos del curso

- Pensar cómo resolver problemas con ayuda del ordenador
- Dejar de ser usuarios para ser creadores
- Crear nuestras propias soluciones para resolver problemas de *data science* con el ordenador

Objetivo principal

- Aprender a programar, es decir, aprender a escribir *software*

¿Pero qué es un programa?

¿Qué es un programa?

- Las personas interactuamos con el ordenador a través de programas.
- Algunos programas interactúan también con otros programas, en vez de personas.

Un programa es un procedimiento para resolver un problema con un ordenador.

Ese procedimiento consiste en una serie de instrucciones

Ordenadores y programas

El ordenador no puede funcionar sin programas. El ordenador no es más que un conjunto de dispositivos organizados según una *arquitectura*.

- Procesador (CPU)
- Memoria RAM (volátil, se pierde al apagar)
- Almacenamiento (disco duro, permanente, no se pierde al apagar)
- Otros dispositivos

Tipos de programas

Hay muchos tipos de programas:

- Sistema operativo: organiza la comunicación entre los dispositivos
- Programas de usuario: ofimática, navegador web
- Programas que trabajan con programas: por ejemplo, el servidor web

¿Cómo se hace un programa?

Lenguajes de programación

- Cualquier persona puede hacer un programa. Hay que escribir una lista de instrucciones, codificadas en un **lenguaje de programación**.

- Existen muchos lenguajes de programación
  - http://es.wikipedia.org/wiki/Lista_de_lenguajes_de_programaci%C3%B3n

¿Qué lenguaje vamos a estudiar?

Lenguaje muy utilizado en ciencias, ingeniería, desarrollo web y ***data science***

Lenguajes de programación

Tipos de lenguajes

- El ordenador solo entiende un tipo de lenguaje, el **lenguaje máquina** o **lenguaje ensamblador**
- Nosotros escribiremos en un **lenguaje de alto nivel**.
- La traducción de un lenguaje a otro la realizan los programas denominados **compiladores** o **intérpretes**.

Lenguajes de alto y bajo nivel

Ejemplos de lenguajes de alto y bajo nivel

Lenguaje de bajo nivel

```
pseudocode
.model small
.stack
.data
Cadena1 DB 'Hola mundo.'
.code

programa:
    MOV AX, @data
    MOV DS, AX
    MOV DX, offset Cadena1
    MOV AH, 9
    INT 21h
    INT 20h
end programa
```

Lenguaje de alto nivel

```
print('Hola mundo')
```

Ventajas e inconvenientes

Tiempo de ejecución y desarrollo

Tiempo	Bajo nivel	Alto nivel	
Ejecución	Rápido	Lento	
Desarrollo	Largo	Corto	

Lenguaje del ordenador

El ordenador solo entiende el lenguaje máquina. La traducción la realizan:

- Compiladores
  - Producen un programa ejecutable independiente.
- Intérpretes
  - Interpretan el programa línea tras línea. Más sencillos para realizar pruebas y modificar programas sobre la marcha.

Por qué Python

[![Python](https://imgs.xkcd.com/comics/python.png)](https://xkcd.com/353/)

Básicamente, es el lenguaje perfecto para empezar

Características de Python

Interpretado

Tipado **fuerte** pero **dinámico**

Orientado a la **legibilidad**

The zen of Python

In [1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python en Data Science

Entornos de programación

IDEs

El compilador o el intérprete no se usa de manera aislada. Suele incluirse en *Entornos Integrados de Desarrollo* (IDE, por sus siglas en inglés).

Una IDE normalmente cuenta con:

- Editor de código fuente
- Explorador de variables
- Línea de comandos del intérprete
- Historial de comandos
- Explorador de ficheros
- Depurador
- Otras herramientas

Por ejemplo, Spyder

Los notebooks

Las IDEs en programación científica están siendo ampliamente reemplazadas por *notebooks*

- Nosotros vamos a usar *notebooks*, que permiten ir ejecutando los programas *celda* a *celda*
- Más detalles, enseguida en cuanto empecemos :)

Nuestro intérprete

- Python es un intérprete, que facilita las tareas de programación.
- En realidad, compila a *bytecode*, por lo que no es un intérprete *puro*.
  - Detalles en https://en.wikipedia.org/wiki/Python_(programming_language)

Ejemplo de programa

Nuestro primer programa

Cálculo del área de un círculo

```
# Calcula el area de un circulo

# Entrada de datos
rs = input('Introduce el radio del circulo: ')

# Algoritmo
r = float(rs)
pi = 3.14159
A = pi*r**2;

# Salida de datos
print('El area del circulo con radio %f es %f' % (r,A))
```

¿Cómo funciona?

Explicación del código

- Los colores se conocen como *resaltado de sintaxis*, y se usan para distinguir más fácilmente los elementos del programa.

- Las líneas que comienzan por # son comentarios. El ordenador las ignora, pero son importantes para entender qué hace el programa.

- Pide un número al usuario y almacena su valor en la variable *rs*

```
rs = input('Introduce el radio del circulo: ')
```

- El resultado se devuelve como una cadena de texto, pero nosotros necesitamos un número con decimales (*número en coma flotante*)

```
r = float(rs)
```

- Además, necesitamos el valor de π, por lo que definimos una variable nueva. La variable hace más legible el programa:

```
pi = 3.14159
```

- Calcula el área como πr^2

```
A = pi*r**2;
```

- Muestra un mensaje en pantalla con el resultado, usando dos números en *coma flotante*

```
print('El area del circulo con radio %f es %f' % (r,A))
```

Estructura general de un programa

Los programas que vamos a desarrollar son más complejos que este ejemplo. Pero la tarea de escribir el programa se puede facilitar si diseñamos el programa para que siga esta estructura:

Entrada de datos: Recibe toda la información necesaria para el programa al comienzo del programa. Ya no vuelve a recibir información en ninguna otra parte del programa.

Algoritmo: Realiza los cálculos que transforman los datos de las entradas a las salidas, todo en la misma parte del programa, sin mezclar con entrada o con salida de datos.

Salida de datos: Devuelve toda la información calculada en el algoritmo, sin realizar cálculos (ya se han realizado previamente) ni pedir información nueva.

¿Qué ocurre por debajo?

¿Cómo funciona el ordenador?

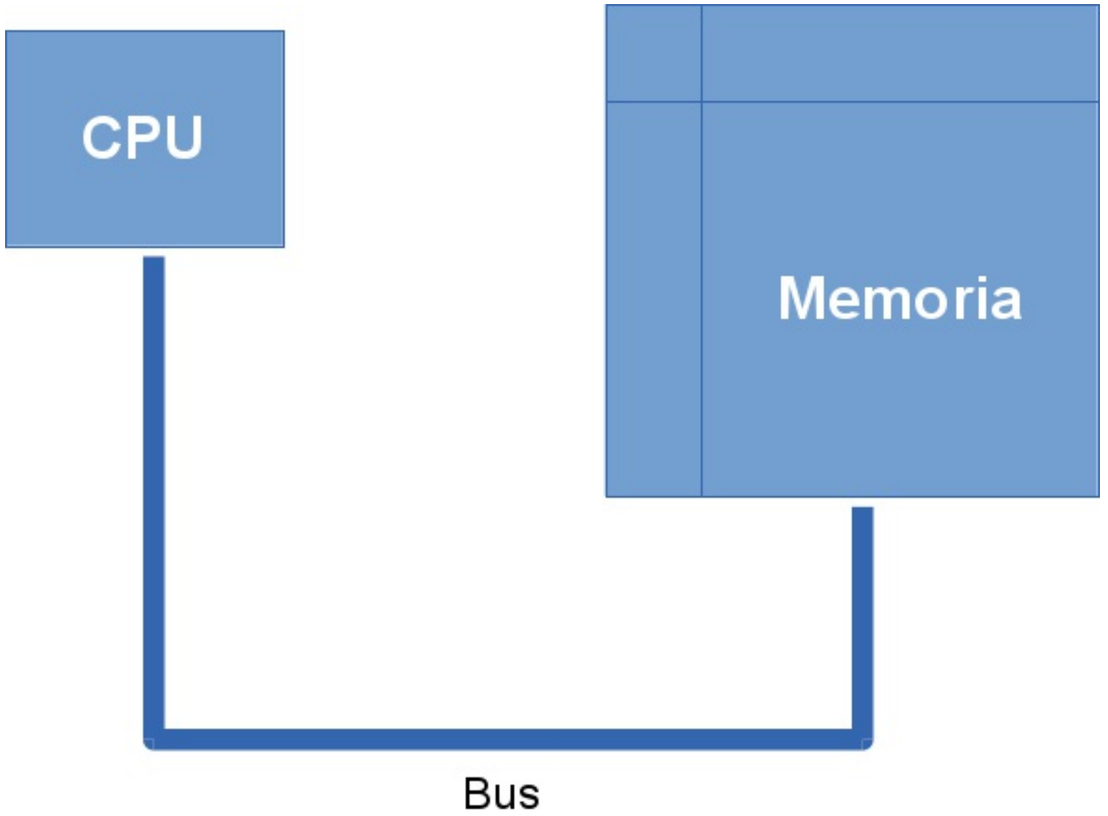
- ¿Dónde se guarda el programa que escribimos?
- ¿Y los valores de las variables que creamos?
- ¿Quién y cómo realiza los cálculos del ordenador?

Arquitectura del ordenador

Componentes principales del ordenador:

- Procesador (CPU)
- Memoria RAM
- Disco duro

Esquema de la arquitectura de un ordenador



¿Qué ocurre cuando programamos?

Pasos cuando estamos programando:

- Escribimos el programa en el entorno, y guardamos el fichero en el **disco duro**.
- Al ejecutar, el programa se lee del disco duro, y se transfiere a la **memoria RAM**.
- Los comandos se transfieren uno a uno a la **CPU**, a través de un **bus**.
- La **CPU** pide los datos de las variables necesarias para el cálculo a la **memoria RAM**, y los guarda en los **registros**.
- La **CPU** realiza la operación, y guarda en los **registros** el resultado de la operación.
- El resultado se transfiere a través del **bus** a la **memoria RAM**, donde podrá ser reutilizado para las operaciones que vengan después.
- Al terminar la ejecución, el código del programa se elimina de la **memoria RAM** (pero obviamente continúa en el disco duro, y se puede ejecutar de nuevo).

Pregunta

Para mejorar la experiencia de usuario en un ordenador... Estamos comprando un ordenador y tenemos el presupuesto algo limitado. Tenemos que elegir entre poner más memoria, más disco duro o un procesador más rápido.

- ¿En cuál de los tres invertimos para mejorar la experiencia de uso del ordenado?
- ¿Cuál es la influencia de cada uno de estos tres componentes en el funcionamiento del ordenador?

Para llevar

Resumen del tema

- En este curso, el objetivo es que aprendamos a escribir nuestros propios programas. En el **Máster en *Data Science*** los usaremos para manejar datos en un ordenador, y usar el resultado de nuestros análisis para tomar decisiones dirigidas por datos.

- Hay muchos lenguajes de programación, de alto nivel y bajo nivel.
    - Los de bajo nivel se ejecutan rápido, pero son difícil de entender y se tarda más tiempo en programar.
    - Los de alto nivel se entienden de manera más sencilla y se tarda menos en programar, pero se ejecutan más lentamente.

- Para programar usaremos *notebooks*, que permiten experimentar con nuestros programas de manera cómoda, y ver los resultados intermedios de manera inmediata.

Resumen del tema

- Intentaremos siempre estructurar nuestros programas en
  - Entrada de datos
  - Algoritmo
  - Salida de datos

- Las partes principales del ordenador son la memoria RAM, la CPU y el disco duro.
  - El texto que escribimos como código fuente se guarda en el disco duro.
  - El programa se carga en memoria, y se va transfiriendo a la CPU. Los resultados se guardan en la memoria en forma de variables.
  - Al terminar, el programa se borra de la memoria RAM, pero podemos ejecutarlo otra, cargándolo desde el disco duro.
  - Toda la comunicación se realiza siempre a través del bus.

Para saber más

- https://es.wikipedia.org/wiki/Programaci%C3%B3n
- https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n
- https://es.wikipedia.org/wiki/Arquitectura_de_von_Neumann