

APRENDIZAJE AUTOMÁTICO

Grado en Ingeniería Informática

Grupos 2 y 3



UNIVERSIDAD
DE GRANADA

CONTACTO

OFELIA RETAMERO PASCUAL

Correo: oretamero@decsai.ugr.es

Tutorías: **Jueves 11:00 - 12:00**

- ▶ Disponibilidad en otros horarios.
- ▶ Confirmar (al menos) el día de antes para concretar lugar.
- ▶ Los dos días anteriores a las entregas de prácticas no se harán tutorías.

GUÍA DE PRÁCTICAS

Toda la información y documentos se encontraran en PRADO.

- ▶ Sesiones 1 y 2: Introducción a Python
+ Ejercicio adicional (5 puntos). Entrega 1/2 marzo.
 - ▶ Sesiones 3 - 5: Práctica 1 (Conceptos y algoritmos básicos)
+ Ejercicio primero (12.5 puntos). Entrega 28 marzo.
-
- ▶ Sesiones 6 - 8: Práctica 2 (Modelos lineales)
 - ▶ Sesiones 9 - 13: Práctica 3 (Boosting, RRNN, FBR)
 - ▶ Sesiones 14 y 15: Desarrollo de proyecto

APRENDIZAJE AUTOMÁTICO

Sesión 1: **Introducción a Python**

ÍNDICE

¿Qué es el Aprendizaje Automático?

¿Por qué estudiar Aprendizaje Automático?

Python

Spyder & Anaconda

Scikit-Learn

Uso básico de Anaconda

Instalación de Scikit-Learn

Familiarización con Spyder & Python

- Partes de Spyder

- Ayuda Spyder

- Primeros pasos

Listas, tuplas y diccionarios

Condicionales y bucles

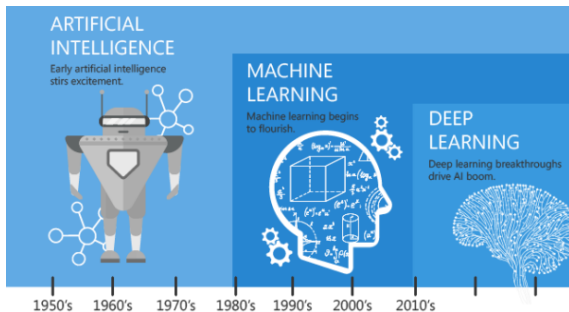
Funciones

Clases

Práctica 0

¿QUÉ ES EL APRENDIZAJE AUTOMÁTICO?

¿QUÉ ES EL APRENDIZAJE AUTOMÁTICO?



DATOS + ALGORITMOS DE APRENDIZAJE

¿POR QUÉ ESTUDIAR APRENDIZAJE AUTOMÁTICO?

¿POR QUÉ ESTUDIAR APRENDIZAJE AUTOMÁTICO?

- ▶ Infinidad de aplicaciones
- ▶ Rama de la IA más desarrollada en la actualidad
- ▶ Gran demanda del perfil (y buen sueldo)
- ▶ Trabajo interesante y no monótono
- ▶ Oportunidad de investigación en la universidad
- ▶ ...

PYTHON

Descarga versión 3.7.6: [Aquí](#)



PYTHON



Descarga versión 3.7.6: [Aquí](#)

► **¿Qué es?** Lenguaje de programación:

- interpretado,
- multiparadigma (orientación a objetos principalmente)
- multiplataforma,
- y de licencia libre.

PYTHON



Descarga versión 3.7.6: [Aquí](#)

► **¿Qué es?** Lenguaje de programación:

- interpretado,
- multiparadigma (orientación a objetos principalmente),
- multiplataforma,
- y de licencia libre.

► **¿Por qué Python?**

- Sencillo de escribir y aprender.
- Gran cantidad de recursos.
- Soporte de distinto tipos de aplicaciones.
- Demanda, moda y comunidad que lo apoya.

PYTHON



Descarga versión 3.7.6: [Aquí](#)

► **¿Qué es?** Lenguaje de programación:

- interpretado,
- multiparadigma (orientación a objetos principalmente),
- multiplataforma,
- y de licencia libre.

► **¿Por qué Python?**

- Sencillo de escribir y aprender.
- Gran cantidad de recursos.
- Soporte de distinto tipos de aplicaciones.
- Demanda, moda y comunidad que lo apoya.

► **Inconvenientes** 🤔

SPYDER Y ANACONDA



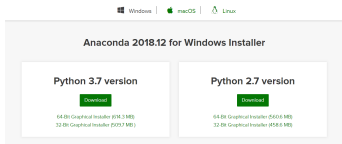
Descarga: [Aquí](#)



SPYDER Y ANACONDA



Descarga: [Aquí](#)



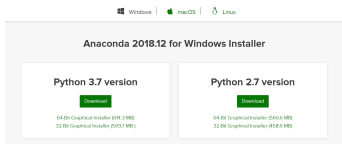
► ¿Qué es?

- Anaconda: Distribución para los lenguajes R y Python ampliamente usada en Ciencia de Datos y Aprendizaje Automático.
- Spyder: Entorno de desarrollo para Python.

SPYDER Y ANACONDA



Descarga: [Aquí](#)



► ¿Qué es?

- Anaconda: Distribución para los lenguajes R y Python ampliamente usada en Ciencia de Datos y Aprendizaje Automático.
- Spyder: Entorno de desarrollo para Python.

► ¿Por qué usarlos?

- Cantidad de funcionalidades.
- Facilidad de uso.
- Herramienta libre.

SCIKIT-LEARN



Documentación: [Aquí](#)

Cheat-sheet: [Aquí](#)

- ▶ **¿Qué es?** Paquete en Python para análisis de datos, construido sobre NumPy, SciPy y matplotlib.



Documentación: [Aquí](#)

Cheat-sheet: [Aquí](#)

- ▶ **¿Qué es?** Paquete en Python para análisis de datos, construido sobre NumPy, SciPy y matplotlib.
- ▶ **¿Por qué Scikit-Learn?**
 - Cantidad de algoritmos y funciones para el tratamiento y análisis de datos, con interfaz común y fácil uso.
 - Muchas librerías compatibles (entre ellas Deep learning).
 - Puede usarse en aplicaciones comerciales.

USO BÁSICO DE ANACONDA

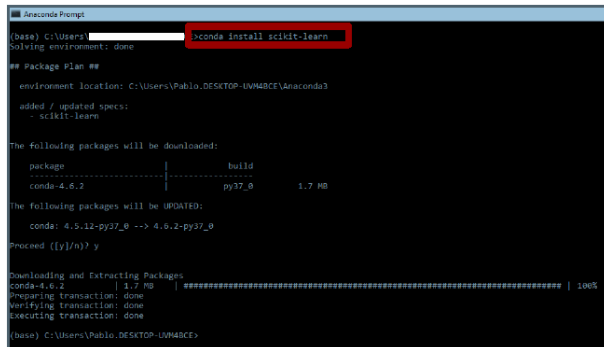
- ▶ Ayuda conda: `conda -h`
- ▶ Crear entorno: `conda create -n <name>`
- ▶ Eliminar entorno: `conda-env remove <name>`
- ▶ Ayuda sobre comando (relacionado con conda):
`conda env <command> -h`
- ▶ Listar entornos: `conda env list`
- ▶ Activar/Desactivar entornos:
 - Windows: `activate <name>` / `deactivate <name>`
 - Linux: `source activate <name>` / `source deactivate <name>`
- ▶ Instalar/Desinstalar/Actualizar paquete:
`conda install <name>` / `conda remove <name>` /
`conda update <name>`

INSTALACIÓN DE SCIKIT-LEARN

1. Abrir Anaconda Prompt

INSTALACIÓN DE SCIKIT-LEARN

1. Abrir Anaconda Prompt
2. Escribir `conda install scikit-learn`



```

Anaconda Prompt
(base) C:\Users\ [redacted] >conda install scikit-learn
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Pablo.DESKTOP-UVM48CE\Anaconda3

added / updated specs:
- scikit-learn

The following packages will be downloaded:

package-----|-----build-----|-----
conda-4.6.2-----|-----py37_0-----| 1.7 MB

The following packages will be UPDATED:

conda: 4.5.12-py37_0 --> 4.6.2-py37_0

Proceed ([y]/n)? y

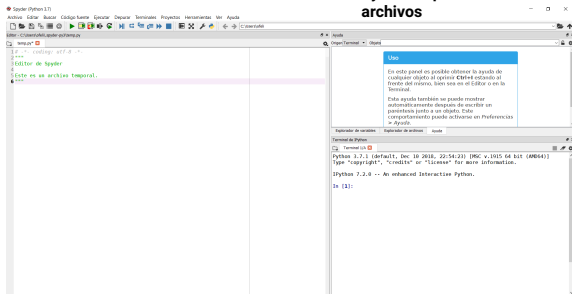
Downloading and Extracting Packages
conda-4.6.2 | 1.7 MB | ***** | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) C:\Users\Pablo.DESKTOP-UVM48CE>
```

FAMILIARIZACIÓN CON SPYDER & PYTHON

Partes de Spyder

Inspector de variables/
Ayuda/ Explorador de
archivos

Editor de
archivos .py
Donde
desarrollaremos
nuestros
programas y
trabajo



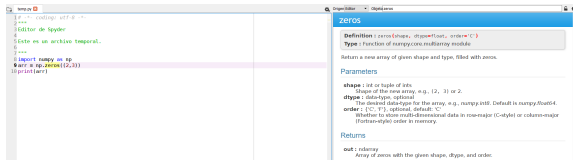
Consola de IPython

FAMILIARIZACIÓN CON SPYDER & PYTHON

Ayuda Spyder

- ▶ Ctrl + i

Ayuda sobre la función en la que el cursor se encuentra.



- ▶ Ctrl + botón izq / Ctrl + G

Consulta de código fuente

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (1)

1. Abrir Spyder

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (1)

1. Abrir Spyder
2. Primera línea del Editor `# -*- coding: utf-8 -*-`
necesaria para escribir con acentos y/o letra ñ

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (1)

1. Abrir Spyder
2. Primera línea del Editor `# -*- coding: utf-8 -*-`
necesaria para escribir con acentos y/o letra ñ
3. Comentarios en el Editor `""" <texto> """` o `#`

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (1)

1. Abrir Spyder
2. Primera línea del Editor `# -*- coding: utf-8 -*-`
necesaria para escribir con acentos y/o letra ñ
3. Comentarios en el Editor `""" <texto> """` o `#`
4. Escribir código con operaciones básicas en la línea de comandos (Terminal)

```
IPython 7.2.0 -- An enhanced Interactive Python.
```

```
In [1]: 1+2  
Out[1]: 3
```

```
In [2]: 5-3  
Out[2]: 2
```

```
In [3]: 5/3  
Out[3]: 1.6666666666666667
```

```
In [4]: 5//3  
Out[4]: 1
```

```
In [5]: 2**3  
Out[5]: 8
```

```
In [6]: 21%8  
Out[6]: 5
```

```
In [7]: |
```

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (2)

5. Importar un paquete

```
In [7]: import numpy as np
```

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (2)

5. Importar un paquete

```
In [7]: import numpy as np
```

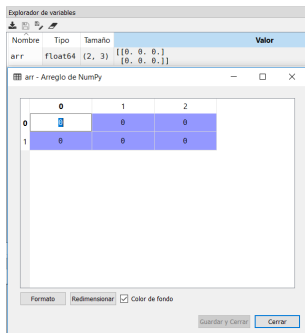
6. Asignación y declaración

```
arr = np.zeros((2,3))
```

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (2)

5. Importar un paquete `In [7]: import numpy as np`
6. Asignación y declaración `arr = np.zeros((2,3))`
7. Arrays en el Explorador de variables



FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (3)

#Declaración var y asignación

```
entero1 = 5 #Int  
entero2 = 505 #Int  
flotante = 50.5 #Float  
boolean_t = True #Boolean  
boolean_f = False #Boolean  
string1 = 'String1' #String  
string2 = 'String2' #String
```

#Operaciones aritméticas básicas

```
suma = entero1 + flotante  
resta = entero1 - flotante  
producto = entero1 * flotante  
division = entero1 / flotante  
division_entera = entero2 // entero1  
resto = entero2 % entero1
```

FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (3)

```
#Operaciones lógicas
igual = entero2==(500 + 5)
no_igual = entero1 != suma
mayor = entero2 > entero1 #>=
menor = entero1 < entero2 # <=
and_logico = igual and mayor
or_logico = igual or no_igual

#Cambiar tipos
entero2flotante = float(entero1)
flotante2entero = int(flotante)
astring = str(entero2)
abool = bool(entero1)

#Strings
formatear = 'String con entero %d, flotante %f y string %s' % (entero1,
                                                                flotante,
                                                                string1)

concatenar = string1 + str(entero1)

#Mostrar por pantalla
print('Dos de los strings:', string1, string2)
print('String y entero:', concatenar, entero1)
```


FAMILIARIZACIÓN CON SPYDER & PYTHON

Primeros pasos (4)

```
# Python es un lenguaje fuertemente tipado  
strint = string1 + enterol  
# y dinámicamente tipado  
string1 = 'Cambio'  
print(string1)
```

LISTAS, TUPLAS Y DICCIONARIOS

1. ¿Qué son?

- ▶ **Listas** Almacenan datos de cualquier tipo y se acceden mediante índices enteros. Son dinámicas, es decir, pueden modificarse sus elementos, añadir, eliminar, ...
- ▶ **Tuplas** Almacenan datos de cualquier tipo y se acceden mediante índices enteros. No pueden modificarse, solo consultarse (son estáticas).

NOTA: Las tuplas pueden modificarse si contienen objetos que puedan hacerlo, como listas

- ▶ **Diccionarios** Almacenan datos de cualquier tipo y se acceden mediante palabras clave (keys). Pueden modificarse sus elementos, añadir, eliminar, ...

LISTAS, TUPLAS Y DICCIONARIOS

1. ¿Qué son?

- ▶ **Listas** Almacenan datos de cualquier tipo y se acceden mediante índices enteros. Son dinámicas, es decir, pueden modificarse sus elementos, añadir, eliminar, ...
- ▶ **Tuplas** Almacenan datos de cualquier tipo y se acceden mediante índices enteros. No pueden modificarse, solo consultarse (son estáticas).

NOTA: Las tuplas pueden modificarse si contienen objetos que puedan hacerlo, como listas

- ▶ **Diccionarios** Almacenan datos de cualquier tipo y se acceden mediante palabras clave (keys). Pueden modificarse sus elementos, añadir, eliminar, ...

2. Declaración

```
tupla = (5, 't1', True, 0.5)
lista = [5, 't1', True, 0.5]
diccionario = {'a':1, 'b':2.0}
```

LISTAS, TUPLAS Y DICCIONARIOS

3. Operaciones básicas

► Listas

- Devuelve el elemento en la posición `i` `lista[i]`
- Devuelve el elemento en la posición `i` y luego lo borra
`lista.pop(i)`
- Añade elemento al final `lista.append(elemento)`
- Inserta elemento en la posición `i` `lista.insert(i, elemento)`
- Fusiona lista con `lista2` `lista.extend(lista2)`
- Elimina elemento, la primera vez que aparece
`lista.remove(elemento)`
- Devuelve la longitud `len(lista)`
- Repetición `lista*i`
- Busca elemento y devuelve posición o error si no lo encuentra
`lista.index(elemento)`

► Tuplas (Igual que las listas, siempre y cuando no supongan una modificación de la tupla: indexado, longitud, repetición y búsqueda de posición)

LISTAS, TUPLAS Y DICCIONARIOS

3. Operaciones básicas

► Dicionarios

- Devuelve el valor que corresponde con la key introducida
`diccionario.get('key')`
- Devuelve el valor que corresponde con la key introducida, y borra la key y el valor `diccionario.pop('key')`
- Inserta una key o actualiza su valor si ya existiera
`diccionario.update('key':'valor')`
- Devuelve True o False si la key (no los valores) existe en el diccionario `"key" in diccionario`
- Devuelve True o False si definición existe en el diccionario (no como key) `"definicion" in diccionario.values()`
- Devuelve la longitud `len(diccionario)`
- Añadir elemento `diccionario['c'] = False`
- Eliminar elemento del `diccionario['c']`
- Devuelve las keys `diccionario.keys()`
- Devuelve los valores `diccionario.values()`

CONDICIONALES Y BUCLES

```
#Condicional
if condicion:
    #Hacer algo
elif otra_condicion:
    #Hacer algo
else:
    #Hacer algo

#Bucle for
for i in range(inicio, fin, paso):
    #Hacer algo

for elemento in lista:
    #Hacer algo

#Bucle while
while condicion:
    #Hacer algo
```

FUNCIONES

- ▶ Forma general

```
def nombre_funcion(par1, par2):  
    res = par1 + par2  
    return res
```

- ▶ Definiendo parámetro o parámetros por defecto

```
def funcion_valDef(par1=0, par2=1):  
    res = par1 + par2  
    return res
```

CLASSES

```
class Class():  
    def __init__(self, a):  
        self.a = a  
  
    def llamar(self, b):  
        return self.a*b
```

```
class Class2(Class):  
    def __init__(self, a, b=2.0):  
        super().__init__(a)  
        self.b = b  
  
    def llamar(self, c):  
        return self.a*self.b*c  
  
    def __call__(self, c):  
        return self.llamar(c)
```


PRÁCTICA 0

- ▶ Se trata de un ejercicio EXTRA, es decir que la entrega no es obligatoria pero sube nota.
- ▶ Puntuación máxima 5 puntos
- ▶ Será muy útil para familiarizarse con la asignatura, los conceptos básicos y el uso de Python.
- ▶ Todos los ejercicios contarán lo mismo.
- ▶ La fecha de entrega es definitiva, cuidado con la hora pues no se corregirán entregas fuera de fecha o enviadas mediante cualquier otra plataforma que no sea PRADO.

Al ataque!