

Diseño y Análisis de Algoritmos

INTRODUCCIÓN A PYTHON



Universidad
Rey Juan Carlos

Motivación

¿Por qué Python?

- **Simplicidad:** los códigos son compactos y limpios
- **IDE:** Pycharm es un IDE completo para el desarrollo y depuración de código Python
- **Multiparadigma:** puede utilizarse tanto de manera imperativa (*scripts*) como orientada a objetos
- **Estructuras de datos:** tiene implementadas una colección de las estructuras más utilizadas

Motivación

¿Por qué Python?

- **Prototipado rápido:** es muy similar al pseudocódigo, permitiéndonos probar ideas en poco tiempo
- **Extensiones:** existe un gran número de módulos que amplían la funcionalidad del lenguaje
- **Multiplataforma:** Windows, Mac OS, Linux...
- **Muy extendido:** es uno de los lenguajes más solicitados por las empresas

Hello World!

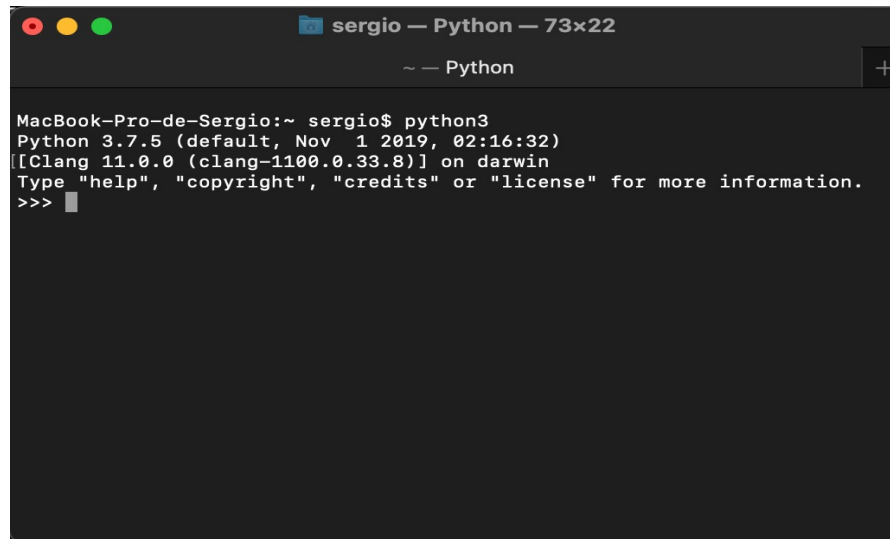
- ¿Cómo implemento el clásico Hello World en Python?

```
print("Hello World!")
```

- Nada más, ni imports, ni creación de un método main, ni una clase, ni nada similar, sólo imprimir la cadena por pantalla

¿Dónde programo?

- Con la consola interactiva (Shell o prompt) podemos directamente escribir y ejecutar instrucciones en Python
- Para ello, abrimos un terminal y escribimos `python3` para arrancar el intérprete

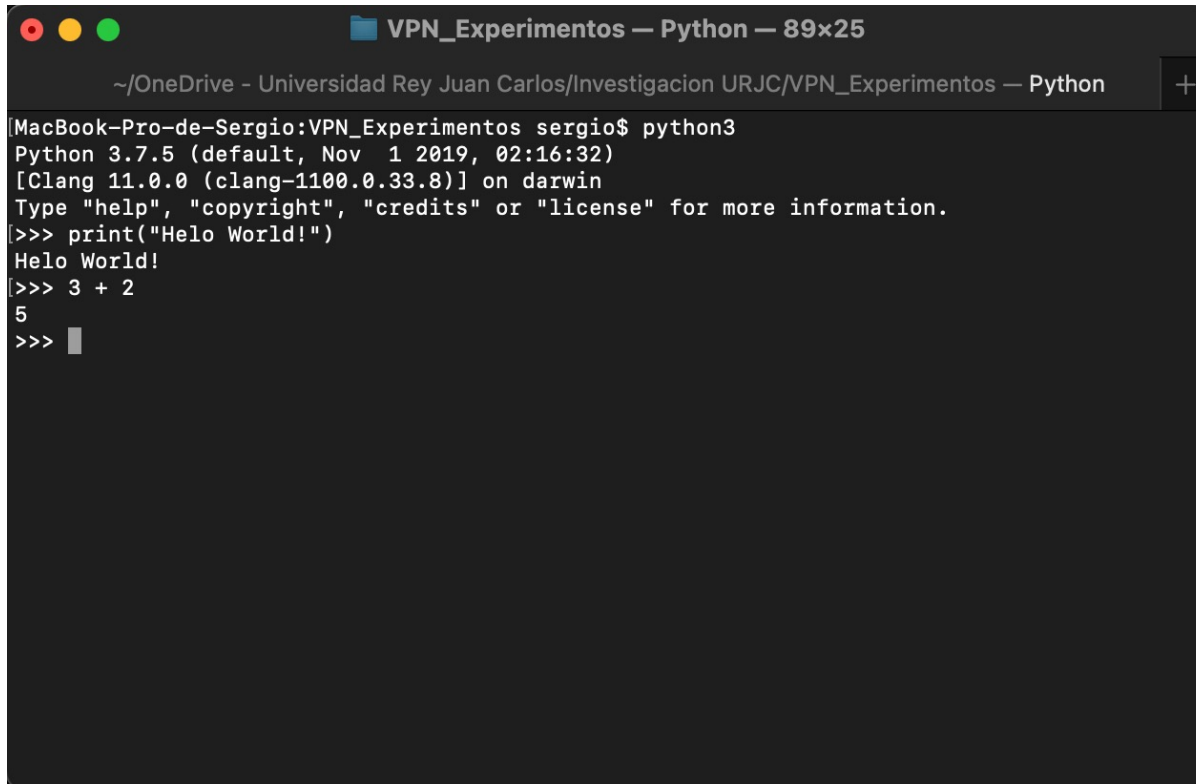


```
sergio — Python — 73x22
~ — Python +

MacBook-Pro-de-Sergio:~ sergio$ python3
Python 3.7.5 (default, Nov 1 2019, 02:16:32)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

¿Dónde programo?

- Ahí ya podemos empezar a escribir instrucciones, operaciones matemáticas, etc.



```
VPN_Experimentos — Python — 89x25
~/OneDrive - Universidad Rey Juan Carlos/Investigacion URJC/VPN_Experimentos — Python
[MacBook-Pro-de-Sergio:VPN_Experimentos sergio$ python3
Python 3.7.5 (default, Nov 1 2019, 02:16:32)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> print("Helo World!")
Helo World!
[>>> 3 + 2
5
[>>> ]
```

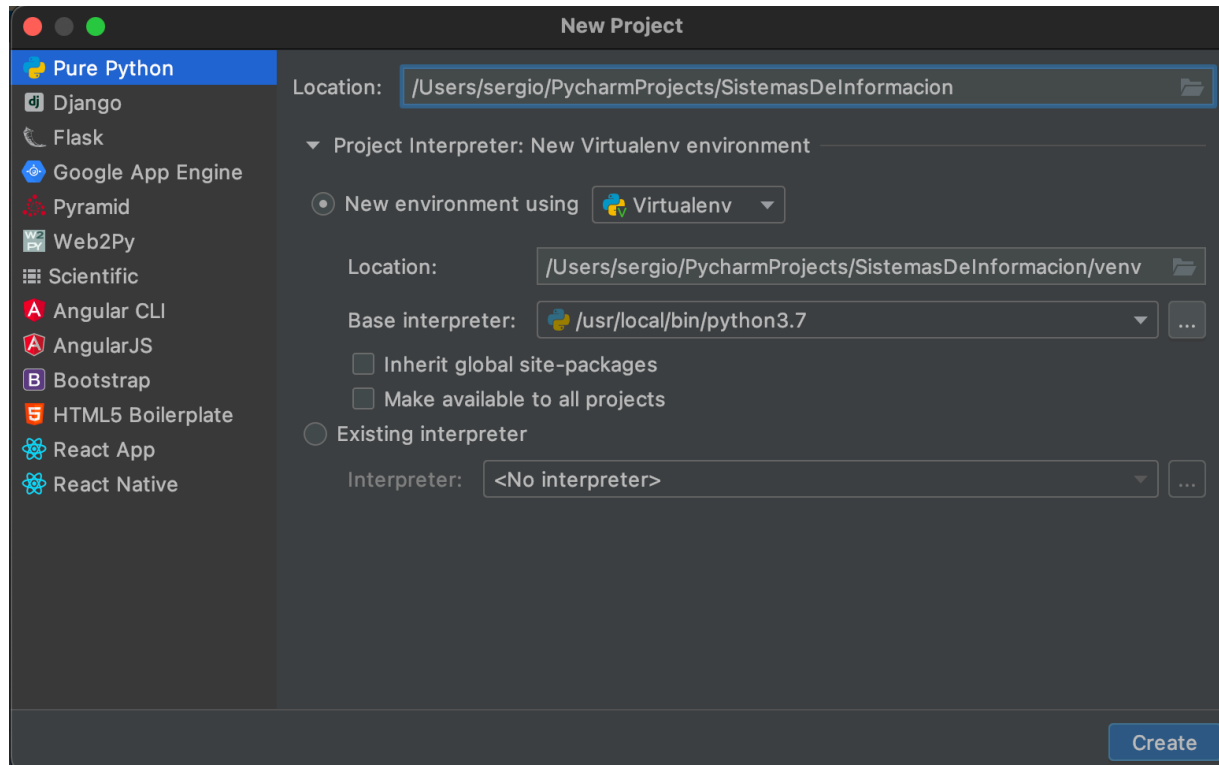
- Pycharm es un IDE desarrollado por JetBrains, disponible en su versión completa de forma gratuita con la cuenta de alumno URJC.
- Está instalado en el Escritorio de Desarrollo en MyApps

IDE - Pycharm

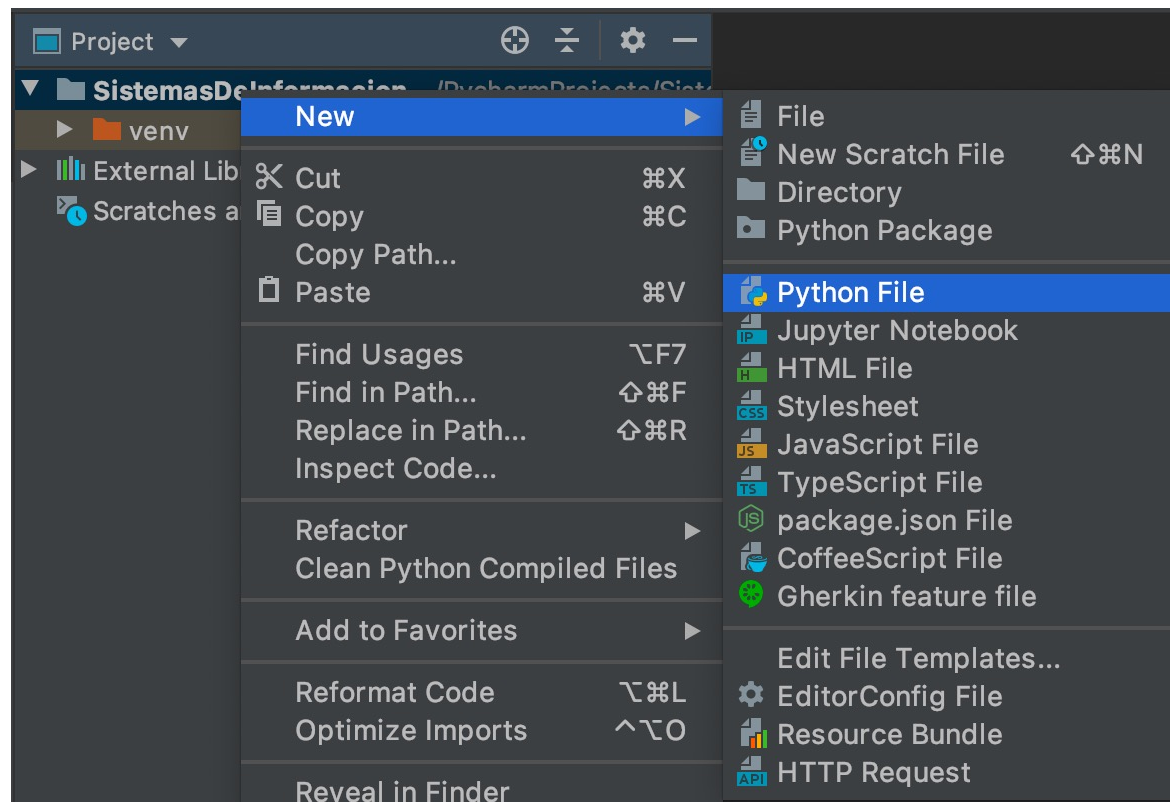
- Al arrancar Pycharm, nos ofrece la opción de crear un nuevo proyecto



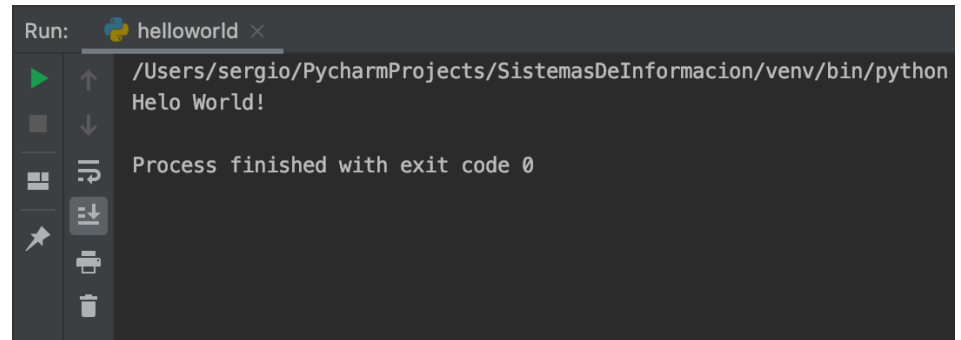
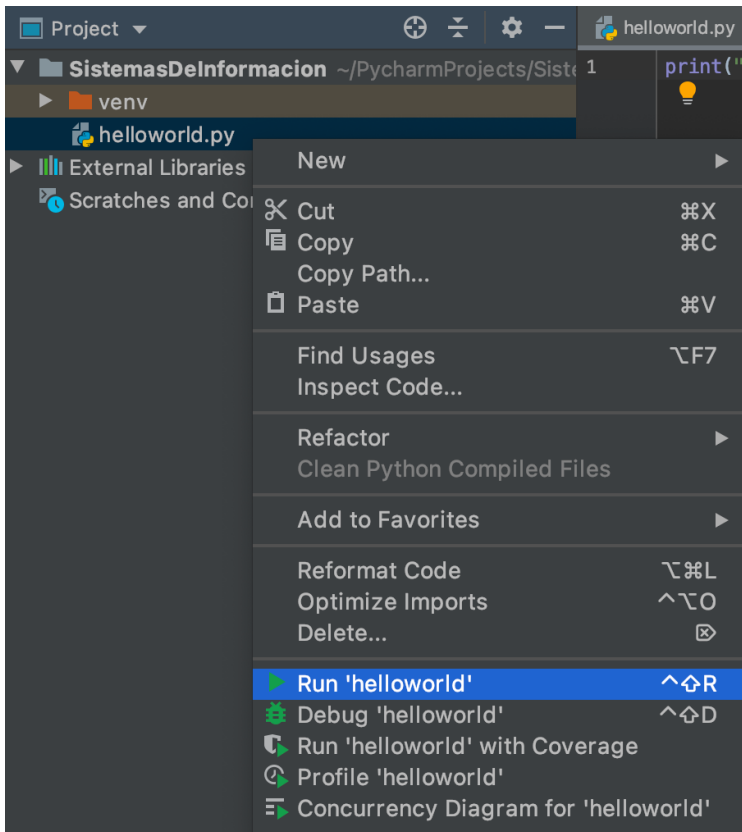
- Elegimos Python 3.9 (11, o la versión que tengamos) como intérprete y dejamos el resto por defecto
 - Creará un entorno virtual asociado al proyecto



- Para crear un nuevo fichero, click derecho en el proyecto, y elegir un nuevo fichero Python



- Para ejecutar un fichero, click derecho en el fichero →
Run nombrefichero.py



Operadores y expresiones

Operación	Operador	Aridad	Asociatividad	Preced.
Exponenciación	**	Binario	Por la dcha	1
Identidad, cambio de signo	+, -	Unario	-	2
Multip, div	*, /	Binario	Por la izq	3
Div ent, mód	//, %	Binario	Por la izq	3
Suma, resta	+, -	Binario	Por la izq	4
Distinto, igual que	!=, ==	Binario	-	5
Menor, menor o igual que	<, <=	Binario	-	5
Mayor, mayor o igual que	>, >=	Binario	-	5
Negación	not	Unario	-	6
Conjunción	and	Binario	Por la izq	7
Disyunción	or	Binario	Por la izq	8

Tipos de datos

- Enteros
 - Ocupan menos espacio en memoria
 - Es más rápido operar con enteros que con reales
- Reales
 - Se pueden utilizar diferentes notaciones: $3.2\text{E}-3$, $.01$, $2.$
- Booleano
 - Puede tomar valor true o false
 - Operadores `and`, `or` y `not`

Tipos de datos

- Operadores de relación

operador	comparación
!=	distinto que
==	igual que
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

Tipos de datos

- Las cadenas de caracteres se escriben entre comillas simples o dobles
- Operadores
 - Concatenación: +
 - Repetición: *

```
name = "Sergio"  
print("Hello, "+name+"!")  
print(("Hello, "+name+"!")*4)
```

Identificadores

- Los nombres de variables, funciones, etc. Deben cumplir ciertas normas:
 - Combinación de letras, dígitos y/o el carácter subrayado `' '`
`—`
 - El primer carácter no puede ser un dígito
 - No puede coincidir con una palabra reservada del lenguaje
 - Distingue mayúsculas y minúsculas

Funciones

- Python tiene algunas funciones predefinidas
 - `abs()` : valor absoluto
 - `float()` : devuelve el valor real de un entero o cadena de caracteres
 - `int()` : devuelve el valor entero de un real (truncando) o cadena de caracteres
 - `str()` : devuelve la cadena de caracteres de un entero o real
 - `round()` : puede usarse con uno o dos argumentos
 - Con 1 redondea al entero más cercano
 - Con 2 el segundo indica cuántos decimales queremos conservar

- La importación del módulo completo no está recomendada, aunque en ocasiones la utilizaremos
 - Si importamos elemento a elemento, sabemos su procedencia, aumentando la legibilidad
 - Si definimos una variable con un nombre que coincide con el de una función del módulo, puede haber conflictos

Módulos

- Para evitar estos problemas se puede importar de una de las siguientes maneras:
 - `import MODULO`
 - `import MODULO as ALIAS`
- Así, para utilizar una función del módulo, deberíamos usar el propio nombre del módulo (o el alias definido):

```
import math  
  
print(math.sin(0))
```

```
import math as mth  
  
print(mth.sin(0))
```

Entrada / Salida

- La sentencia `input` recibe datos por teclado como una cadena de caracteres
- Podemos convertirlo al tipo de dato que necesitemos

```
lado = int(input("Introduce un valor para el lado:"))  
print("El área del cuadrado de lado "+str(lado)+" es de "+str(lado*lado))
```

Entrada / Salida

- La sentencia `print` imprime cadenas de caracteres por consola

```
print("El área del cuadrado de lado "+str(lado)+" es de "+str(lado*lado))  
print("El área del cuadrado de lado", lado, "es de", lado*lado)
```

- Por defecto añade un salto de línea al final, pero podemos modificarlo o incluso eliminarlo

```
print("El área del cuadrado de lado", lado, "es de", lado*lado, end='')
```

Entrada / Salida con ficheros

- Para leer/escribir un fichero usamos la sentencia `open`
- Recibe dos parámetros: el nombre del fichero y el modo en que lo queremos abrir

```
f = open("nombre_fichero", "modo")
```

- Estos modos pueden ser lectura (r), escritura (w), o escritura añadiendo al final o append (a)

Entrada / Salida con ficheros

- Para leer todo el contenido de un fichero de una sola vez, utilizamos el método `read()`.

```
f = open("nombre_fichero", "modo")
contenido = f.read()
```

- Si queremos leer un fichero línea a línea, utilizamos el método `readlines()`, que almacenará una colección con todas las líneas del fichero

```
f = open("nombre_fichero", "modo")
lineas_fichero = f.readlines()
for line in lineas_fichero:
    print(line)
```

Entrada / Salida con ficheros

- Para escribir en un fichero, lo abriremos en modo escritura

```
f = open("nombre_fichero", "w+")
```

- Utilizaremos el método `write()` para escribir en él

```
f = open("nombre_fichero", "w+")  
lineas_a_escribir = ["Hola", "qué tal"]  
for line in lineas_a_escribir:  
    f.write(line+"\n")
```

- O el método `writelines()` para escribir todo de golpe

```
f = open("nombre_fichero", "w+")  
lineas_a_escribir = ["Hola", "qué tal"]  
f.writelines(lineas_a_escribir)
```


Sentencia condicional – if

```
if condicion:  
    instruccion 1  
    instruccion 2  
    ...  
    instruccion N
```

- Es muy importante respetar el sangrado, es lo que determina el contenido del if

Sentencia condicional – if-else

```
if condicion:  
    cosas  
else:  
    otras cosas
```

- En Python tenemos evaluación perezosa
 - Si evaluando la primera parte de la expresión sabemos el resultado, no se evalúa el resto

Sentencia condicional – if-else-if

```
if condicion:  
    cosas  
elif condicion2:  
    mas cosas  
else:  
    otras cosas
```

- Si tenemos más de una condición podemos utilizar un if-elif

Sentencia iterativa – while

```
while condicion:  
    instruccion 1  
    instruccion 2  
    ...  
    instruccion N
```

- Mientras se cumpla la condición, se ejecutan las instrucciones internas

Sentencia iterativa – for

```
for variable in serie_de_valores:  
    instruccion 1  
    instruccion 2  
    ...  
    instruccion N
```

- Para todos los elementos de la serie de valores se ejecuta el código

Función range

- A veces no podemos escribir el rango de valores
- range es una función que, dado el inicio y el final, nos da el rango de valores intermedio
 - El inicio es inclusivo y el final exclusivo
- Si la utilizamos con un argumento, sólo estamos especificando el final (comienza en 0)
- Si la utilizamos con tres, el tercero indica el incremento

```
for i in range(11):  
    print(i)  
for i in range(1,11):  
    print(i)  
for i in range(1,11,2):  
    print(i)
```

Función range

- Podemos generar series decrecientes especificando un valor negativo en el tercer argumento

```
for i in range(11,1,-1):  
    print(i)
```

Definición de funciones

- Para definir una función:

```
def nombreFuncion(a,b,c,...):  
    ...  
    ...  
    return ...
```

- `def` indica que vamos a definir una función
- `return` indica lo que devuelve la función

```
def cuadrado(a):  
    return a*a
```

- Si no ponemos `return`, tenemos un procedimiento

Listas

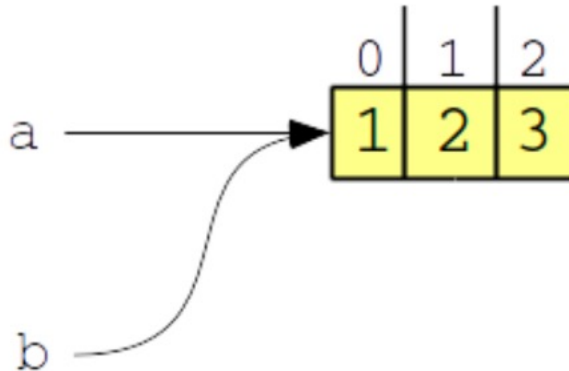
- Python permite definir listas de cualquier tipo de datos
- Los valores se escriben entre corchetes y separados por comas
- Por ejemplo, para definir una lista que contenga los tres primeros enteros:

```
a = [1,2,3]
```

Listas

- ¡¡OJO!! Las listas son punteros

```
a = [1,2,3]  
b = a
```



Listas

- Para saber la longitud de una lista, tenemos la función `len()`
- Para concatenar listas, tenemos el operador `+`
- Para repetir una lista, tenemos el operador `*`
- Para obtener un fragmento, podemos usar el operador de corte (slice):

```
a = [4,5,1,7,8]
b = a[2:4]
print(b)
```

Listas

- Para añadir elementos, podemos concatenar una lista o utilizar el método `append` sobre la lista

```
a = [4, 5, 1, 7, 8]
a.append(9)
```

- Para eliminar, utilizamos el operador `del`

```
a = [4, 5, 1, 7, 8]
del a[3]
```

- Para saber si una lista contiene un elemento, utilizamos el operador `in`

```
a = [4, 5, 1, 7, 8]
print(3 in a)
```

Matrices

- Las matrices se almacenan como listas de listas

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

- El acceso se realiza de la misma manera

```
m[0][1]
```

Diccionarios

- Se trata de una correspondencia entre claves y valores
- Creación

```
d = {}
```

- Añadir pares clave-valor

```
d['uno'] = 1  
d['dos'] = 2
```

- Similar a la lista, pero el acceso es con clave en lugar de con índice

Diseños y Análisis de Algoritmos

INTRODUCCIÓN A PYTHON



Universidad
Rey Juan Carlos