

- Creado a principios de los 90 por Guido van Rossum (el nombre procede del programa de la BBC “Monty Python’s Flying Circus”)
- Es un “poderoso” lenguaje de programación “fácil” de aprender
- Interpretado, tipado dinámico y multiplataforma
- Cuenta con una amplia biblioteca estándar, y con una extensísima colección de aplicaciones y librerías desarrolladas
- Uno de los lenguajes predominantes en ciencia de datos
- Web oficial de Python: <http://www.python.org>
- Paradigmas de programación:
 - Programación orientada a objetos
 - Programación imperativa
 - Programación funcional

Python 3.x

- Eficiencia (C/C++)
- Desarrollo y mantenimiento de grandes proyectos (Java, C#)
- Enfocados a la web (Javascript)
- **Python es un lenguaje de propósito general que permite escribir código compacto y sencillo de leer.**



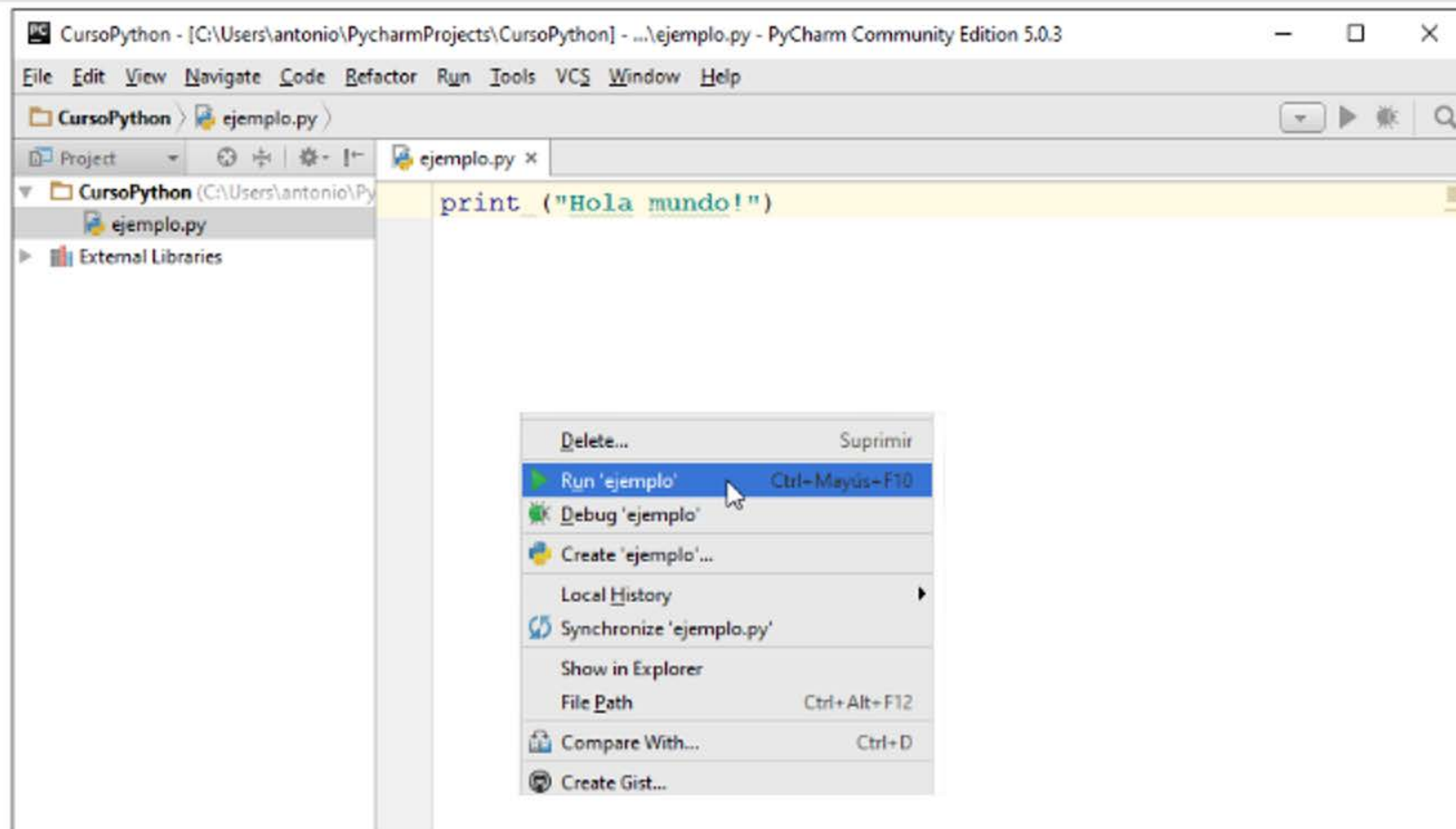
Curso de Formación en Informática

Introducción a la Programación en Python

```
# Calcula el área de un cuadrado a partir de la longitud de su lado.  
lado = float(input("Introduce la longitud del lado: "))  
area = lado * lado  
print("El área del cuadrado es", area)
```



- Instalación interprete phyton (version 3.x)
- Modo interactivo en la ventana del intérprete
- Editor de texto plano (bloc de notas)
- Editor especializado [PyCharm](#) (entorno de programación)
- ejemplo.py



```
C:\Users\user>python
```

```
Python 3.7.2
```

```
>>> exec(open("prueba.py").read())
```

```
11
```

```
hola
```

```
>>> exit()
```

```
>>> print ("hola")
```

```
hola
```

```
C:\Users\user>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open("prueba.py").read())
```

Ejecutar archivos .py

```
C:\Users\user>py prueba.py
```

```
11
```

```
hola
```

```
C:\Users\user>
```

```

17 # Escribir una función cuadrados(l) que recibiendo una secuencia l de números,
18 # devuelve la lista de los cuadrados de esos números, en el mismo orden.
19
20
21 # Por ejemplo:
22 #
23 # >>> cuadrados([4,1,5.2,3,8])
24 # [16, 1, 27.040000000000003, 9, 64]
25
26 # Hacer dos versiones: una usando un bucle explícito, y la otra mediante
27 # definición de listas por comprensión.
28 # -----
29
30 # Por comprensión:
31
32 def cuadrados1(l):
33     a = [a*a for a in l]
34     return a
35
36 # Usando bucle:
37
38 def cuadrados2(l):
39     a = []
40     for i in l:
41         a.append(i*i)
42         print(a)
43     return a
44
45 cuadrados2([3,2,1])
46
47
48
49

```

Run: practica-01.phyton

C:\Users\user\venv\Scripts\python.exe "C:/Datos/Clases/IA/IAIC/1920/Material-Apuntes-IA/Apuntes-IA con phyton/Solucion practicas Phyton/practica-01.phyton.py"

[9]

[9, 4]

[9, 4, 1]

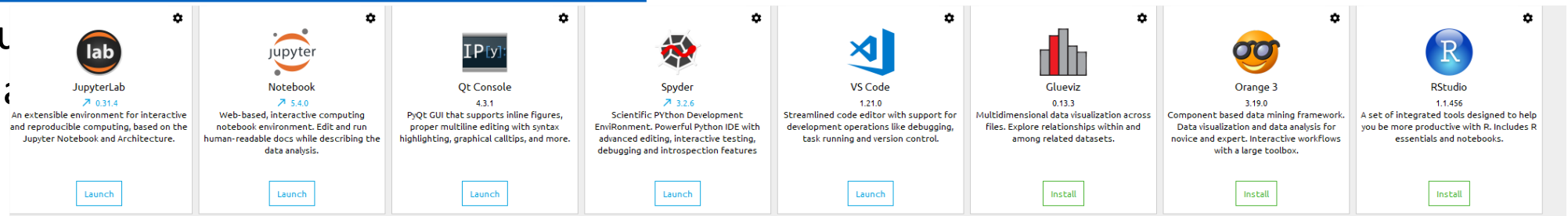
Process finished with exit code 0



Jupyter Notebook
Aplicación

- Es una aplicación web que permite crear documentos que contienen código vivo, texto, fórmulas, figuras, y medios audiovisuales.
Podemos escribir código python ejecutable, texto, dibujar gráficas
- Archivos con la extensión .ipynb
- Se visualizan con el navegador (Explorer, Chrome, Firefox, ...).
- La ejecución de Jupyter Notebook se puede realizar a través de la aplicación Anaconda Navigator o desde el **intérprete de comandos y ejecutar jupyter notebook en el directorio en el que quieras trabajar.**
- Anaconda es una plataforma para aplicaciones de análisis de datos en Python <https://www.anaconda.com/download/>

Anaconda es una
plataforma para aplicaciones de análisis de datos en Python



- Librerías de cálculo científico Numpy, Pandas y Matplotlib

Símbolo del sistema

```
Microsoft Windows [Versión 10.0.18362.295]  
(c) 2019 Microsoft Corporation. Todos los derechos reservados.  
  
C:\Users\user>cd C:\Datos\Clases\IA\IAIC\1920\notebooks-20190521  
  
C:\Datos\Clases\IA\IAIC\1920\notebooks-20190521>jupyter notebook_
```

localhost:8888/tree

Aplicaciones Bookmarks W3Schools Online... Stanford Large Net... UCM-Universidad C... Análisis de redes so... Facultad de Inform... Descarg...



Files

Running

Clusters

Select items to perform actions on them.

0



- ☐ Árboles de decisión.ipynb
- ☐ Carga y visualización de datos.ipynb
- ☐ Clustering.ipynb
- ☐ Intro Python.ipynb
- ☐ k-NN y Validación cruzada.ipynb
- ☐ MLP Regresion.ipynb

- Para aprender usaremos el notebook Intro-python.ipynb

1. Tipos numéricos

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

```
Out[1]: 4
```

```
In [2]: (50-5*6)/4
```

```
Out[2]: 5.0
```

```
In [3]: (2+3)**4
```

```
Out[3]: 625
```

```
In [4]: (1+2j)/(1+1j)
```

```
Out[4]: (1.5+0.5j)
```


2. Variables

Variables en python: símbolos en los que "almacenamos" datos, para referenciarlos durante un programa. *Las variables en Python no hay que declararlas*

En la práctica, la definición anterior nos sirve en la mayoría de las situaciones, pero siendo más precisos, una variable es una **referencia** a una posición de memoria, en la que está almacenada el dato.

Las asignaciones se realizan con el símbolo =

```
In [5]: ancho = 20  
alto = 5*9  
area = ancho * alto
```

Los nombres de variables en Python sólo pueden comenzar con una letra o un subrayado _, y sólo pueden contener letras, números o guiones bajos.

```
In [6]: area
```

```
Out[6]: 900
```

```
In [7]: ancho, alto, area
```

```
Out[7]: (20, 45, 900)
```

```
In [8]: # Asignaciones a varias variables en una línea <-- Comentario en Python  
x,y=2,3
```

En python, los números son **inmutables**, lo que quiere decir que una vez se crea el dato, no puede ser cambiado. Más adelante veremos más tipos de datos inmutables.

Expresiones y variables

```
In [1]: x = 1
        y = 2
        print('suma:', x + y)
        print('resta:', x - y)
        print('multiplicación:', x * y)
        print('division:', x / y)
        print('division entera:', x // y)
        print('potencia:', y ** 10)
```

```
suma: 3
resta: -1
multiplicación: 2
division: 0.5
division entera: 0
potencia: 1024
```

```
In [2]: x = 5
        x += 2
        x # El último valor de la celda se imprime por defecto
```

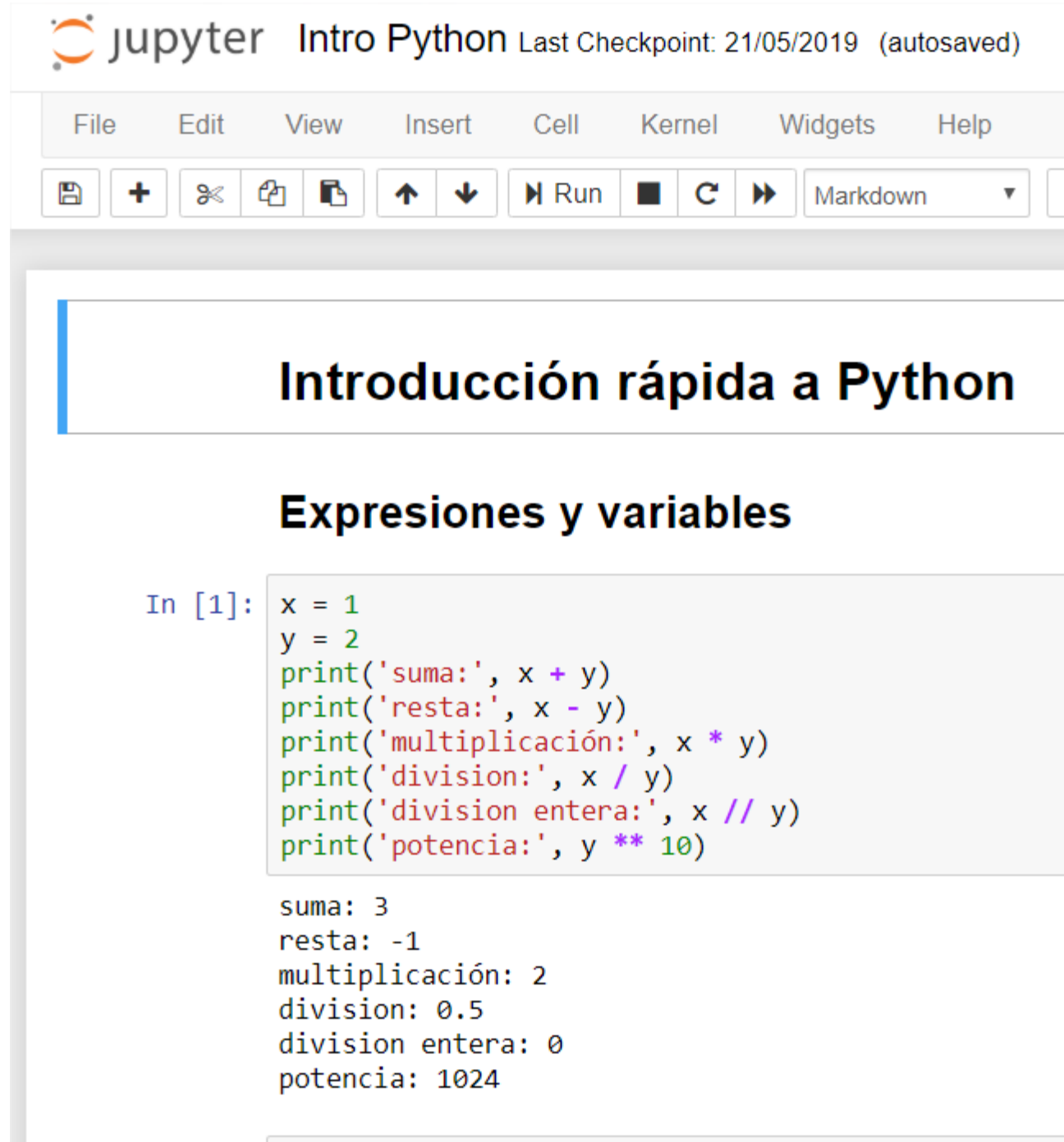
Out[2]: 7

```
In [3]: # operaciones con cadenas
        mayusculas = 'antonio'.upper()
        print(mayusculas)
        cadena_datos = '{} tiene {} años'.format('juan', 21)
        print(cadena_datos)
        concatenar = 'hola' + ' ' + 'mundo!'
        print(concatenar)
```

```
ANTONIO
juan tiene 21 años
hola mundo!
```

Toma de contacto con el entorno

- Abre el archivo notebook intro-python.ipynb
- Familiarízate con el lenguaje y con la ejecución de instrucciones sencillas.
- Haz cambios en las celdas de input `in[]:` y ejecútalas para ver la salida



The screenshot shows a Jupyter Notebook interface. At the top, the Jupyter logo is followed by "Intro Python" and "Last Checkpoint: 21/05/2019 (autosaved)". Below this is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Under the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, running, and a dropdown menu currently set to "Markdown".

The main content area has a title "Introducción rápida a Python" in a blue-bordered box. Below this is a section header "Expresiones y variables".

A code cell is shown with the prompt "In [1]:" followed by a Python script:

```
x = 1
y = 2
print('suma:', x + y)
print('resta:', x - y)
print('multiplicación:', x * y)
print('division:', x / y)
print('division entera:', x // y)
print('potencia:', y ** 10)
```

Below the code, the output of the script is displayed:

```
suma: 3
resta: -1
multiplicación: 2
division: 0.5
division entera: 0
potencia: 1024
```

Pedir datos por consola

```
In [4]: nombre = input('Dime tu nombre:') # input siempre devuelve una cadena str()
numero_entero = int(input('Dime un número entero:'))
numero_real = float(input('Dime un número real:'))
print('Nombre:', nombre, 'Entero:', numero_entero, 'Real:', numero_real)
```

```
Dime tu nombre:antonio
Dime un número entero:6
Dime un número real:3.14159
Nombre: antonio Entero: 6 Real: 3.14159
```

Condicionales

```
In [5]: x = 25
if x % 2 == 0:
    print('Divisible por 2')
elif x % 3 == 0:
    print('Divisible por 3')
elif x % 5 == 0:
    print('Divisible por 5')
else:
    print('Ni idea')
```

```
Divisible por 5
```

```
In [6]: # condiciones compuestas
year = int(input('Introduce un año:'))
if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    print('bisiesto')
else:
    print('no es bisiesto')
```

```
Introduce un año:1919
no es bisiesto
```

Bucles

```
In [7]: suma = 0
        i = 1
        while i <= 10:
            suma += i
            i += 1
        suma
```

Out[7]: 55

Tuplas

```
In [8]: # El operador coma crea tuplas. Se pueden escribir o no entre paréntes.
        a = 1, 2, 3
        print(a)
        b = (4, 5, 6)
        print(b)
```

(1, 2, 3)
(4, 5, 6)

```
In [9]: # Se pueden leer sus componentes
        a[0]
```

Out[9]: 1

```
In [10]: # Las tuplas son inmutables.
          # a[0] = 10 # ERROR!
```

```
In [11]: # La asignación entre tuplas asigna campo a campo.
          # (parece asignación múltiple pero es asignación entre tuplas)
          a, b, c = 10, 20, 30
          print(a)
          print(b)
          print(c)
```

10
20
30

Rangos

In [12]: `range(1, 10)` # Representa los números del 1 al 9 ≥ 1 y < 9

Out[12]: `range(1, 10)`

In [13]: `# Los bucles for permiten recorrer estructuras de datos`
`for x in range(1, 10):`
 `print(x)`

1
2
3
4
5
6
7
8
9

In [14]: `# del 1 al 9 de 3 en 3`
`for x in range(1, 10, 3):`
 `print(x)`

1
4
7

In [15]: `# del 10 al 2 hacia abajo`
`for x in range(10, 1, -1):`
 `print(x)`

10
9
8

EJEMPLOS

```
numeros_pequeños = list(range(0, 10))  
print("Pequeños:", numeros_pequeños)  
numeros_grandes = list(range(1000000, 1000010))  
print("Grandes:", numeros_grandes)  
numeros_impares = list(range(1, 10, 2))  
print("Impares:", numeros_impares)  
numeros_pares = list(range(2, 10, 2))  
print("Pares:", numeros_pares)  
numeros_decrecientes = list(range(10, 0, -1))  
print("Decrecientes:", numeros_decrecientes)
```

- Python dispone de funciones integradas al lenguaje y que se agrupan en módulos que se pueden importar

print imprime mensajes en la consola.

input solicita un dato al usuario y lo devuelve como cadena de texto.

int, float, str convierten respectivamente a entero, real y cadena de texto.

range devuelve objetos iterables que se pueden recorrer como listas.

list, set, dict permiten crear listas, conjuntos y diccionarios vacíos o a partir de otras estructuras de datos.

- y permite crear funciones definidas por el usuario con **def**

```
>>> def hola(arg):  
...     """El docstring de la función"""  
...     print "Hola ", arg, "!"  
...  
>>> hola("Belen")  
Hola Belen !
```

```
nombre = input("Introduce tu nombre: ")
edad = int(input("Introduce tu edad: "))
persona = { "nombre" : nombre, "edad" : edad }
print("Datos:", persona)
# Pueden devolver más de un valor
# Cociente y resto de 10 entre 3
cociente, resto = divmod(10, 3)
print("cociente:", cociente)
print("resto:", resto)
```


- El módulo math contiene las funciones matemáticas y podemos importarlo usando la instrucción import:

```
import math  
raiz = math.sqrt(16)    # raiz cuadrada  
print(raiz)
```

```
import math  
  
print(math.sqrt(16))    # Raiz cuadrada de 16  
print(math.cos(0))      # Coseno de un ángulo de 0 radianes  
print(math.radians(90)) # Radianes equivalentes a 90 grados  
print(math.log10(1000)) # Logaritmo en base 10 de 1000  
print(math.pow(5, 3))   # 5 elevado a 3
```

```
from math import sqrt, pow  
  
print(sqrt(25))  
print(pow(2,10))
```

Ejercicio 1

- Escribe un programa que calcule el volumen de un cono a partir del *radio* de la base y su *altura*. Usa la siguiente fórmula:

$$\text{volumen} = \pi * \text{radio} * \text{radio} * \text{altura} / 3$$

Solicita los valores de *radio* y *altura* al usuario. Recuerda que el valor aproximado de *pi* es 3.14159.

Ejercicio 2

- Escribe un programa que clasifique los elementos de una lista en positivos y negativos. Los elementos positivos deben añadirse a una lista y los negativos a otra. Si la lista original contiene ceros, se deben ignorar.

```
# La lista original  
lista_original = [7, 6, -9, 234, -4, 0, -7]
```

```
# Listas que recibirán el contenido  
positivos = []  
negativos = []
```

```
....
```

Ejercicio 3

- Escribir una función `cuadrados(l)` que recibiendo una secuencia `l` de números, devuelve la lista de los cuadrados de esos números, en el mismo orden. Por ejemplo:
- `cuadrados([4,1,5.2,3,8])`
- `[16, 1, 27.040000000000003, 9, 64]`
- Hacer dos versiones: una usando un bucle explícito, y la otra mediante definición de listas por comprensión.

- Python viene con multitud de tipos de datos integrados como diccionarios, listas, set...
- Los diccionarios son estructuras de datos que almacenan pares clave : valor y nos permiten recuperar los valores asociados a las claves de manera muy rápida.

```
dic_vacio = {}  
persona = { "nombre" : "juan", "edad" : 25, "profesión" : "informático"}  
print(persona)
```

```
{'nombre': 'juan', 'edad': 25, 'profesión': 'informático'}
```

```
print(persona["nombre"])  
print(persona["edad"])  
print(persona["profesión"])
```

```
persona["nombre"] = "pedro"  
print(persona)
```

- Para añadir elementos a un diccionario basta con asignar un valor a una clave que aún no existe:

```
persona["nacionalidad"] = "española"
print(persona) → {'nombre': 'pedro', 'edad': 25, 'profesión':
'informático', 'nacionalidad': 'española'}
```

- Para eliminar una clave del diccionario y su valor asociado usamos la palabra reservada del:

```
del persona["nacionalidad"]
print(persona)
{'nombre': 'pedro', 'edad': 25, 'profesión': 'informático'}
```

- Podemos recorrer las claves almacenadas en un diccionario con un bucle:

```
for clave in persona:    # Otra forma: for clave in persona.keys():
    print(clave)
```

- Para recorrer los valores almacenados en el diccionario usamos el método values():

```
for valor in persona.values():
    print(valor)
```

- Para recorrer explícitamente los pares clave : valor usamos el método items:

```
for clave, valor in persona.items():
    print(clave, valor)
```

Uso de diccionarios para representar estados

""" [Figure 4.9]

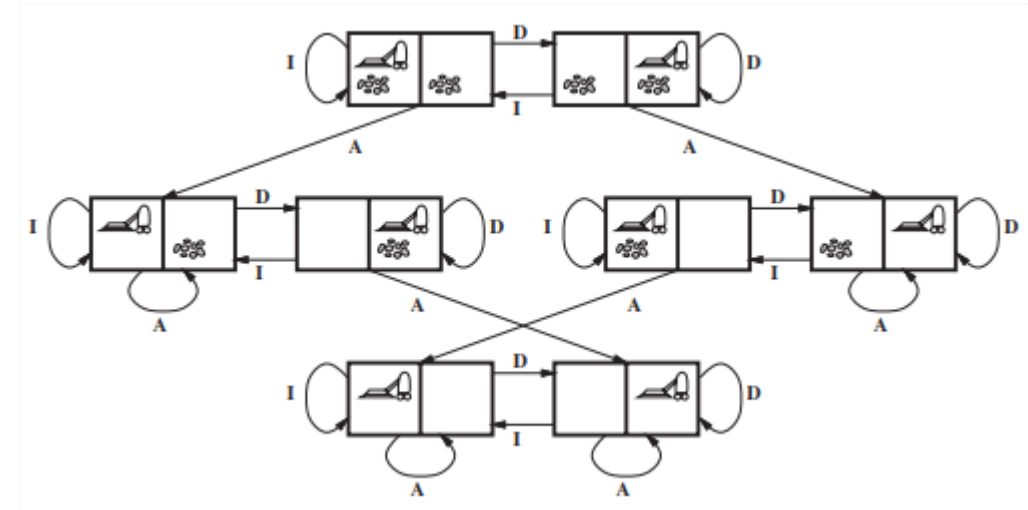
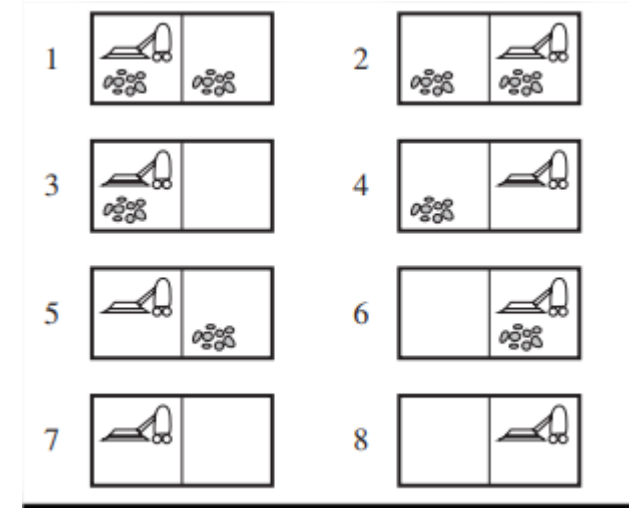
Eight possible states of the vacuum world

Each state is represented as

*	"State of the left room"	"State of the right room"	"Room in which the agent is present"
1 - DDL	Dirty	Dirty	Left
2 - DDR	Dirty	Dirty	Right
3 - DCL	Dirty	Clean	Left
4 - DCR	Dirty	Clean	Right
5 - CDL	Clean	Dirty	Left
6 - CDR	Clean	Dirty	Right
7 - CCL	Clean	Clean	Left
8 - CCR	Clean	Clean	Right

"""

```
vacuum_world = Graph(dict(
    State_1=dict(Suck=['State_5'], Right=['State_2']),
    State_2=dict(Suck=['State_4'], Left=['State_2']),
    State_3=dict(Suck=['State_7'], Right=['State_4']),
    State_4=dict(Suck=['State_4'], Left=['State_3']),
    State_5=dict(Suck=['State_5'], Right=['State_6']),
    State_6=dict(Suck=['State_8'], Left=['State_5']),
    State_7=dict(Suck=['State_7'], Right=['State_8']),
    State_8=dict(Suck=['State_8'], Left=['State_7'])
))
```



```
class Persona:
```

```
    name = ""
```

```
    school = ""
```

```
juan = Person()
```

```
juan.name = 'Juan'
```

```
juan.school = 'Universidad Complutense de Madrid'
```

```
Class Persona:
```

```
    pass
```

```
# crear una instancia (objeto) de esta clase
```

```
p = Persona()
```

```
# Hemos creado un nuevo objeto del tipo Persona.
```

```
# Podemos saber la clase de un objeto y qué espacio ocupa
```

```
print p
```

```
< __main__.Person instance at 0x109a1cb48 >
```


#Se definen con la palabra clave def igual que las funciones.

```
class Persona:
```

```
    name = ""  
    school = ""
```

```
    def print_name(self):  
        print self.name
```

```
    def print_school(self):  
        print self.school
```

```
juan = Persona()  
juan.name = 'Jorge'  
juan.school = 'Universidad Politecnica de Madrid'  
juan.print_name()  
juan.print_school()
```

```
palabra_mayusculas = "python".upper()  
lista = []  
lista.append(3)
```

```
def print_information(self, name, school):  
    print name  
    print school
```

```
juan.print_information(juan.name, juan.school)
```

**self, es el objeto del método que está siendo llamado (es decir juan).
En las llamada al método NO necesitamos pasar self como un argumento**

Inicialización

El inicializador es un método especial, con nombre `__init__` (subrayados dobles al principio y al final).

```
class Persona:
    def __init__(self, n, s):
        self.name = n
        self.school = s

    def print_name(self):
        print self.name

    def print_school(self):
        print self.school
```

```
belen = Persona('Belen', 'Universidad Complutense de Madrid')
belen.print_name()
belen.print_school()
```

Hasta aquí hemos visto una toma de contacto con python y con el entorno jupyter y también sabemos ejecutar archivos .py desde el entorno pycharm

Para las prácticas vamos a usar el framework AIMA

[Artificial Intelligence: A Modern Approach \(Third edition\) by Stuart Russell and Peter Norvig](#)

<http://aima.cs.berkeley.edu/python/readme.html>

Dos opciones:

1. Se puede descargar directamente y copiar el zip en la carpeta de trabajo. Lo importante es que jupyter tenga acceso a los archivos y podamos importarlos.
2. O seguir las instrucciones en <https://github.com/aimacode/aima-python>

!git clone <https://github.com/aimacode/aima-python.git>

Lo descarga en el directorio de trabajo

```
➤ cd aima-python
```

```
C:\Users\user\practicas\aima-python
```

```
➤ from search import *
```

```
import sys  
sys.path.append('aima-python/')
```

<http://aima.cs.berkeley.edu/python/readme.html>

← → ↻ ⓘ No es seguro | aima.cs.berkeley.edu/python/readme.html

📁 Aplicaciones ★ Bookmarks 🌐 W3Schools Online... 📄 Stanford Large Net... 🌐 UCM-Universidad C... 📄 Análisis de redes so... 🌐 Facultad de Inform... ⬇ Descargas 🌐 Campus Virtual - U... 🌐 Google 🕒 https://calendar.go... 🌐 CoreW

[Artificial Intelligence: A Modern Approach](#)

AIMA Python Code

This file gives an overview of the Python code for the algorithms in the textbook [AI: A Modern Approach](#). The code is Copyright (c) 2002 by [Peter Norvig](#) and is offered free of charge [for your use](#). As you may know, the textbook presents provide this Python code as well as [Lisp code](#). The intent is to implement all the algorithms in both languages, so that you can choose whichever language you prefer. As yet neither implementation is complete, but the Lisp version is closer.

Installation Instructions

Here is how to download the files and make them ready for use. You only need to do this once, and if you are taking a course, your instructor may have set this up for you.

1. Create a directory where you want the code to reside on your local machine. You can call this whatever you want; we'll call it *home*.
2. Get the [data.zip](#), store it in *home* file and unzip it. Your browser may unzip automatically, or you can give the command "unzip aima-python.zip" or drag the file to your zip program icon. In the end, just make sure you have files in t
3. Download the file [aima-python.zip](#) into *home*.
4. Unzip it, creating files in *home/python*.
5. You must have Python (version 2.2 or later) installed. Python comes preinstalled on most versions of Linux and Mac OS. Versions are also available for Windows, Solaris, and other operating systems. If your system does not have Py
6. Make sure that *home/python* is in your [module search path](#). You do this either by always starting Python from the directory where you keep the files, or by editing the environment variable PYTHONPATH.
7. Test the code. There are unit tests interspersed in the code. They follow the Python [doctest](#) conventions and can be run with the command line "python doctests.py -v *.py". The "-v" is optional; it means "verbose". Various output i instances of the word "Failure", nor of a long line of "*****". If you do use the "-v" option, the last line printed should be "Test passed."

User's Guide

Once you have the files installed, you can use them in several ways.

- Read the code. This can enhance your understanding of the algorithms, and clarify parts that were not spelled out in the book's pseudo-code.
- Run the existing code on your own data. For the module(s) you want, do "import *module*" and then run the functions you want on the data you want.
- Experiment with extending the code.

Code File Summary

You can

Chapter	Module	Files	Lines	Description
1-2	agents	.py	532	Implement Agents and Environments (Chapters 1-2).
3-4	search	.py .txt	735	Search (Chapters 3-4)
5	csp	.py .txt	449	CSP (Constraint Satisfaction Problems) problems and solvers. (Chapter 5).
6	games	.py	285	Games, or Adversarial Search. (Chapters 6)
7-10	logic	.py .txt	887	Representations and Inference for Logic (Chapters 7-10)
11-12	planning	.py	6	Planning (Chapters 11-12)
13-15	probability	.py .txt	170	Probability models. (Chapter 13-15)
17	mdp	.py .txt	141	Markov Decision Processes (Chapter 17)
18-20	learning	.py	585	Learn to estimate functions from examples. (Chapters 18-20)
21	rl	.py	14	Reinforcement Learning (Chapter 21)
22	nlp	.py .txt	169	A chart parser and some grammars. (Chapter 22)
23	text	.py .txt	364	Statistical Language Processing tools. (Chapter 23)
	doctests	.py .txt	42	Run all doctests from modules on the command line. For each
	py2html	.py	109	Pretty-print Python code to colorized, hyperlinked html.
	utils	.py .txt	713	Provide some widely useful utilities. Safe for "from utils import *".
			5201	

Download the file [aima-python.zip](#)

- En clase hemos visto como representar los problemas y varios algoritmos de búsqueda.
- El framework AIMA nos proporciona una implementación de los algoritmos.
- <https://github.com/aimacode/aima-python>
- Hay muchos problemas de ejemplo:
 - Jarras
 - 8 puzzle
 - Misioneros
- Como práctica de este tema haremos alguno de los ejercicios de clase en formato práctico
- Se os pedirá entregar alguno de los problemas resuelto y medir experimentalmente las propiedades de los algoritmos.

search.py

from search import Problem

search.py forma parte de AIMA, se puede modificar y utilizar para resolver problemas según el paradigma de búsqueda en el espacio de estados.

Añadido

```
class Problem(object):

    """The abstract class for a formal problem. You should subclass
    this and implement the methods actions and result, and possibly
    __init__, goal_test, and path_cost. Then you will create instances
    of your subclass and solve them with the various search functions."""

    def __init__(self, initial, goal=None):
        """The constructor specifies the initial state, and possibly a goal
        state, if there is a unique goal. Your subclass's constructor can add
        other arguments."""
        self.initial = initial
        self.goal = goal

    def actions(self, state):
        """Return the actions that can be executed in the given
        state. The result would typically be a list, but if there are
        many actions, consider yielding them one at a time in an
        iterator, rather than building them all at once."""
        raise NotImplementedError

    def result(self, state, action):
        """Return the state that results from executing the given
        action in the given state. The action must be one of
        self.actions(state)."""
        raise NotImplementedError

    def goal_test(self, state):
        """Return True if the state is a goal. The default method compares the
        state to self.goal or checks for state in self.goal if it is a
        list, as specified in the constructor. Override this method if
        checking against a single self.goal is not enough."""
        if isinstance(self.goal, list):
            return is_in(state, self.goal)
        else:
            return state == self.goal

    def path_cost(self, c, state1, action, state2):
        """Return the cost of a solution path that arrives at state2 from
        state1 via action, assuming cost c to get up to state1. If the problem
        is such that the path doesn't matter, this function will only look at
        state2. If the path does matter, it will consider c and maybe state1
        and action. The default method costs 1 for every step in the path."""
        return c + 1

    def value(self, state):
        """For optimization problems, each state has a value. Hill-climbing
        and related algorithms try to maximize this value."""
        raise NotImplementedError

    def coste_de_aplicar_accion(self, estado, accion):
        """Hemos incluido esta función que devuelve el coste de un único operador (aplicar accion a estado). Por defecto, este
        coste es 1. Reimplementar si el problema define otro coste """
        return 1
```

Práctica 1

- Practica1ParteA.ipynb
 - Os damos hechos varios ejemplos de uso y resolución de problemas de clase tipo 8 puzzle y jarras
 - Puzle de 8 con huecos para terminar
 - Se definen heurísticas y generan estadísticas
 - Misioneros
- Ejercicio de cruzar el puente con linterna
- Torres de hanoi
- Grados de separación entre actores
- El juego de la vida
- Os diremos exactamente qué ejercicios entregar. Podéis practicar con cualquier problema de la selección de ejercicios