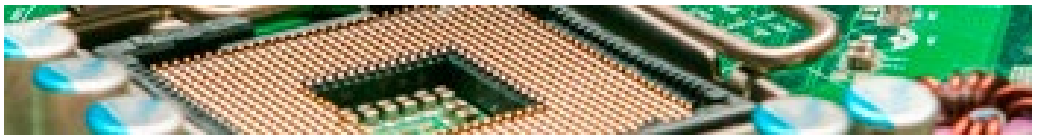
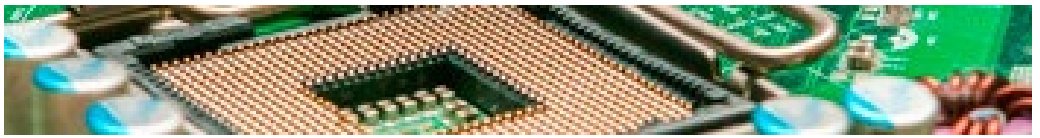
Tema 2: Planificación inteligente

Seminario 1. Introducción a Pyhop



Contenido

1. Introducción
2. Fases en la resolución de un problema
3. Componentes de un problema
   1. Context Model
   2. Conocimiento experto
      1. Tasks
      2. Methods
      3. Funciones auxiliares
   3. Estado inicial
   4. Objetivos
4. Resolución del problema
5. Ejercicios



Introducción

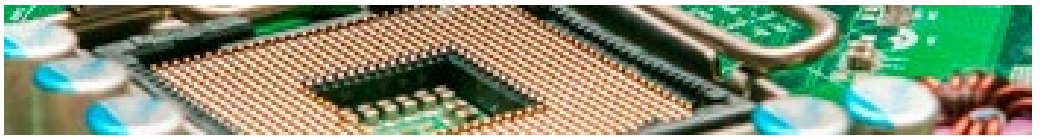
Pyhop es un planificador HTN desarrollado en Python

http://bitbucket.org/dananau/pyhop

- Funciona en Python 2.7 y 3.2
- El algoritmo es el mismo que el de SHOP (Simple Hierarchical Ordered Planner).
  - Planificador HTN utilizado en cientos de proyectos en el mundo, para organizaciones gubernamentales, industria y academia (http://www.cs.umd.edu/projects/shop).

Principales diferencias entre Pyhop y SHOP:

- Los operadores HTN son funciones Python, lo que simplifica el aprendizaje, ya que no es necesario usar el lenguaje propio de SHOP.
- El estado actual se especifica por medio de asignaciones de variables, en lugar de descripciones de hechos.

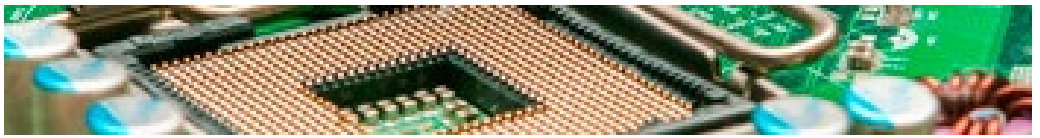


Pyhop

- Diseñado para la ayuda a la toma de decisiones, capaz de generar un plan compuesto por la **secuencia de acciones necesarias para satisfacer un objetivo dado.**

- Cada acción del plan puede indicar múltiple información en forma de **parámetros**: **quién** debe realizar la actividad, qué **recursos** están implicados, el **instante de tiempo** en el que se debe ejecutar y las **dependencias** con otras actividades.

- Resolución de problemas cuyos procedimientos de actuación están **orientados a objetivos.**



Fases en la resolución de un problema

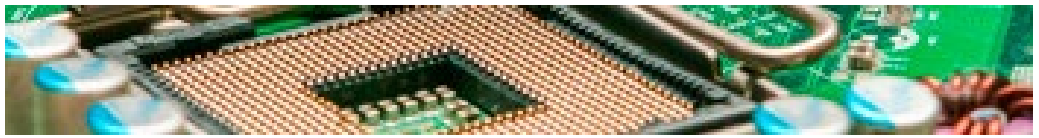
Modelado
- Definir información de contexto
- Definir el conocimiento experto

Planificación
- Definir el estado del mundo
- Especificar objetivo
- Generar solución

Ejecución
- Ejecutar el plan obtenido

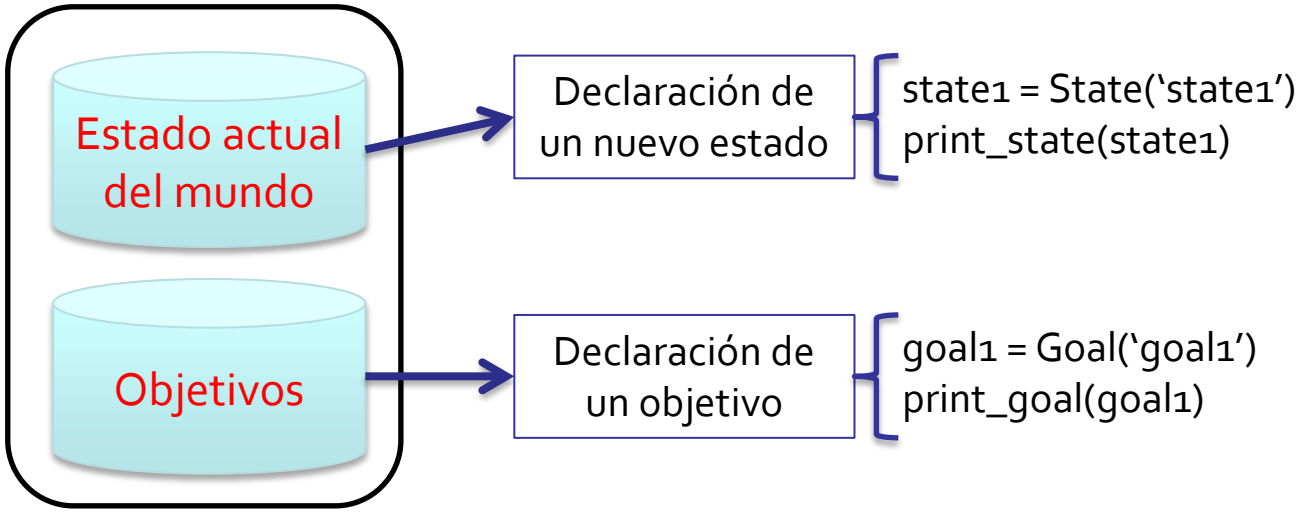
En esta fase se define la información de contexto donde se produce el problema. Posteriormente se define el conocimiento experto, es decir, los diferentes operadores, tareas y métodos.

El usuario especificará el estado del mundo inicial, seleccionará el objetivo que desea alcanzar y el planificador se encargará de generar un plan teniendo en cuenta el conocimiento experto definido y toda la información de contexto que haya introducido el usuario.

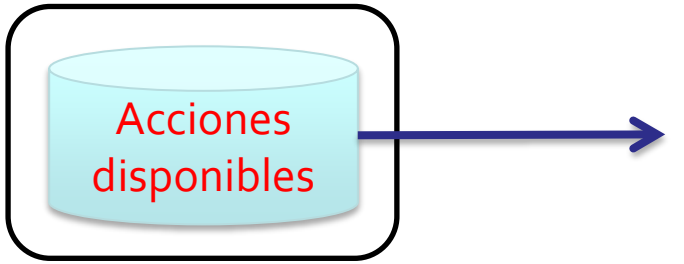


Componentes de un problema

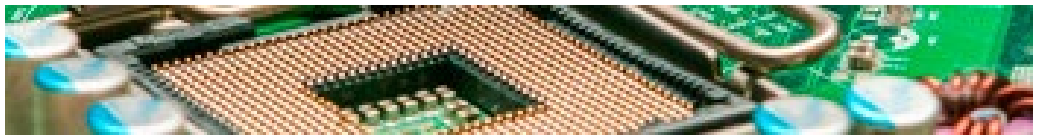
PROBLEMA



DOMINIO



Las acciones disponibles se estructuran en dos niveles: tareas/operadores y métodos.

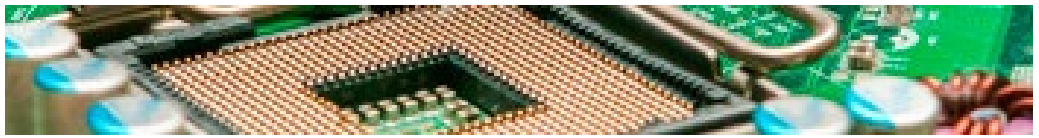


Context model

Define la estructura del estado a representar, es decir, qué información es necesaria para resolver el problema.

Se divide en:

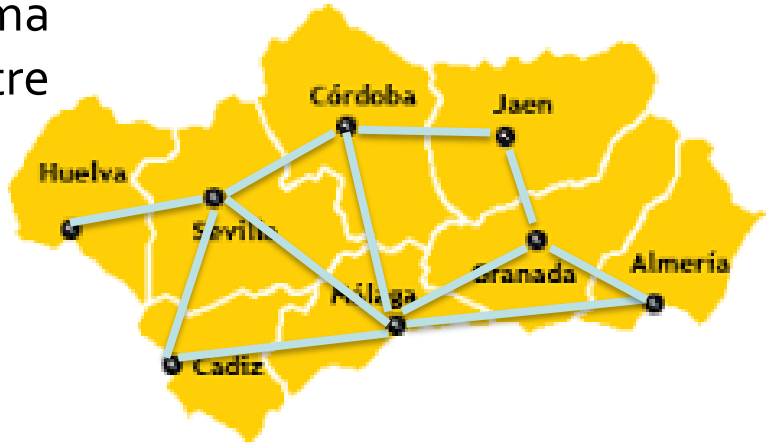
- Información estática: no cambia durante la ejecución del problema y que sirve de soporte. Por ejemplo: distancias entre ciudades.
- Información dinámica: representa el estado actual del mundo. Por ejemplo: cantidad de dinero disponible por el agente en un momento determinado.



Context model

Ejemplo. Se trata de un problema de cálculo de rutas para viajar entre diferentes ciudades

Ciudad	X	Y
Huelva	25	275
Cádiz	200	50
Sevilla	250	325
Córdoba	475	450
Málaga	550	100
Jaén	750	425
Granada	800	250
Almería	1000	150



Representa las conexiones entre las ciudades

Representa la posición de las ciudades

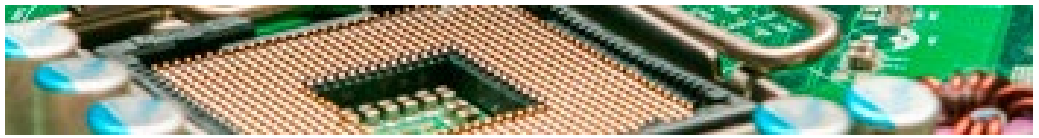
Información a representar en el estado:

- coordinates
- connection

Estática

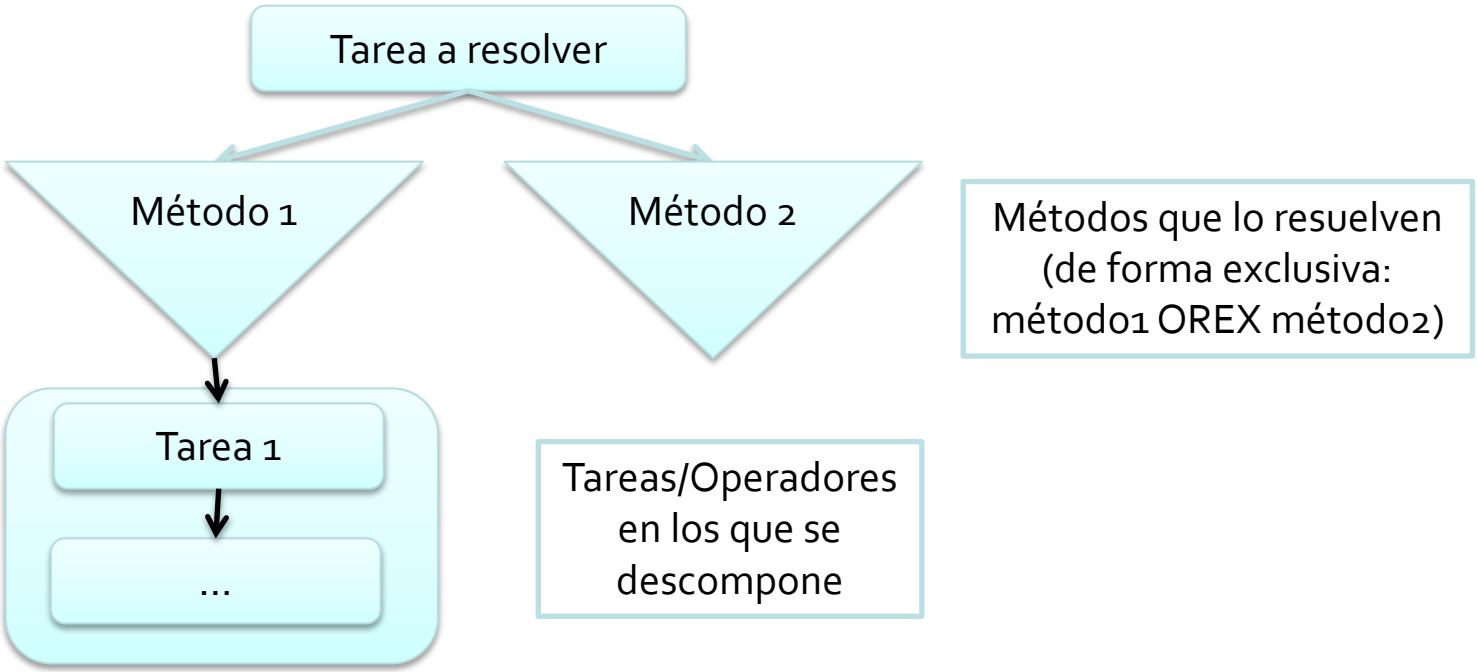
- location
- path
- cost

Dinámica



Conocimiento experto

- Para representar el conocimiento experto debe estar orientado a objetivos.
- El conjunto de estos objetivos definirá un procedimiento de actuación ante problemas.



Operadores

Representan las **acciones ejecutables** en el dominio

```
def nombre_operador(state,otros_parametros):
    <extracción valores estado>
    if condicion1 and condicion2 and … :
        state.variable = nuevo_valor
        …
        state.variable = nuevo_valor
        return state
    else: return False
```

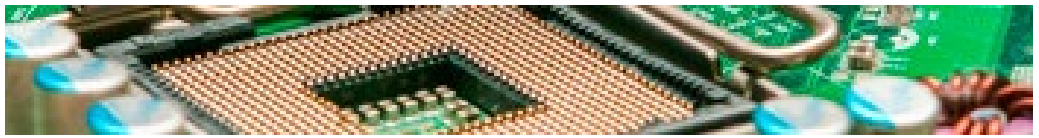
Condiciones

Efectos

Se devuelve el nuevo estado tras aplicarse el operador o False si el operador no es aplicable

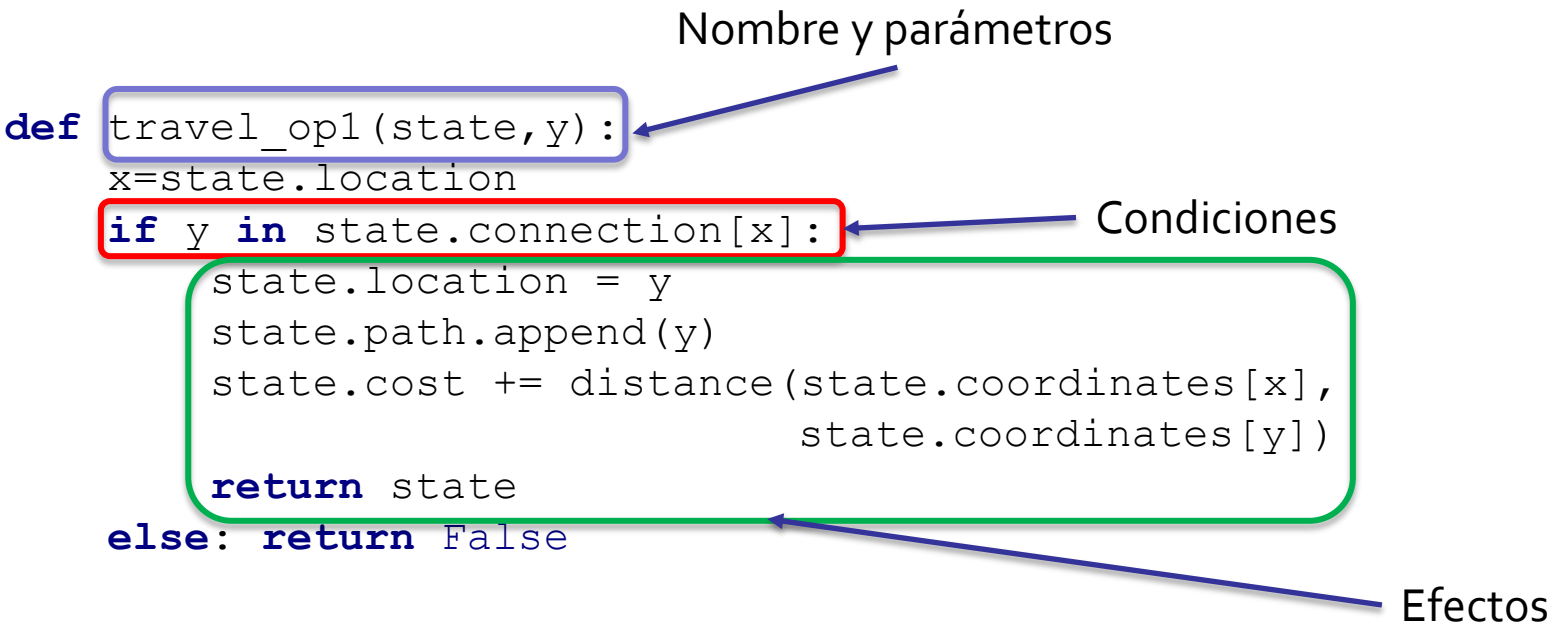
```
pyhop.declare_operators(nombre_todos_operadores)
```

Muy importante: un operador no puede descomponerse (es ejecutable)



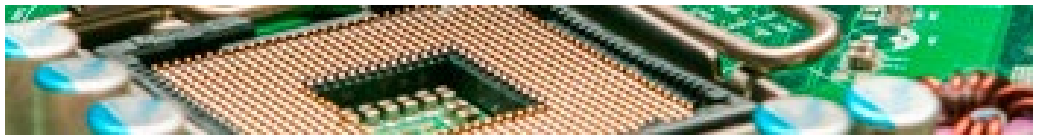
Operadores

Ejemplo:



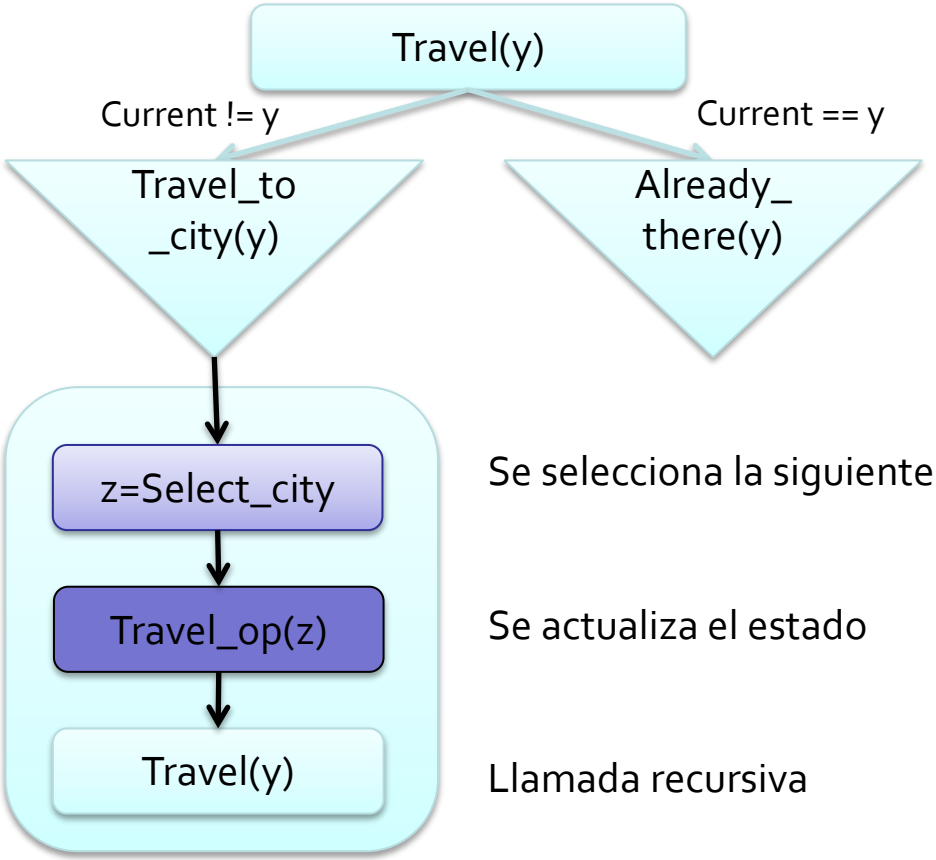

```
def travel_op1(state,y):
    x=state.location
    if y in state.connection[x]:
        state.location = y
        state.path.append(y)
        state.cost += distance(state.coordinates[x],
                               state.coordinates[y])
        return state
    else: return False
```

```
pyhop.declare_operators(travel_op1, travel_op2…)
```

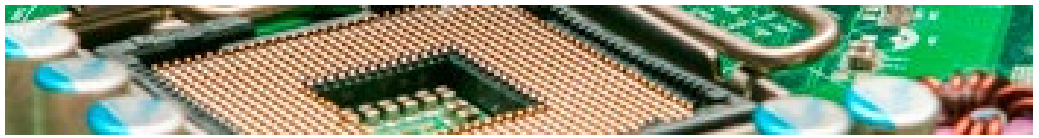



Métodos de una tarea

Representan los métodos del dominio



Función auxiliar	Se selecciona la siguiente ciudad a viajar
Operador	Se actualiza el estado
Tarea	Llamada recursiva



Métodos de una tarea

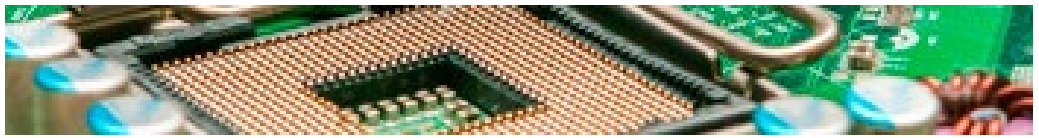
```
def travel_to_city(state,goal):
    x=state.location
    y=goal.final
    if x!=y:
        z=select_new_city(state,y)
        g=pyhop.Goal('g')
        g.final=y
        return [('travel_op',z), ('travel',g)]
    return False

def already_there(state,goal):
    x=state.location
    y=goal.final
    if x==y:
        return []
    return False

pyhop.declare_methods('travel',travel_to_city,already_there)
```

Importante: descomposición en operadores o tareas (no métodos)
Se pasan los parámetros pero no el estado (que es algo dinámico)

Se devuelve la descomposición (puede ser vacía)
o False si no es aplicable

Task

Métodos que resuelven la tarea (**uno u otro, OR-exclusiva**)

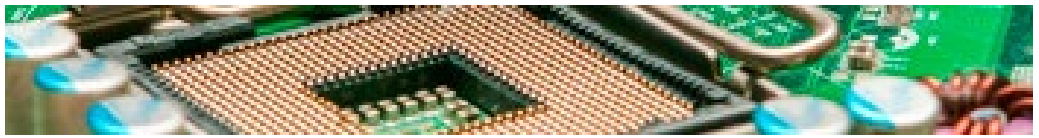


Funciones auxiliares

Es posible añadir tantas funciones para realizar cálculos auxiliares como sea necesario.

Tienen el formato de funciones Python.

```
def distance(c1, c2):
    x=pow(c1['X']-c2['X'],2)
    y=pow(c1['Y']-c2['Y'],2)
    return sqrt(x+y)
```

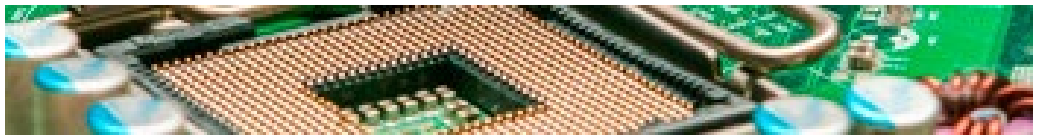
Estado inicial

Información estática (invariable en **nuestro** dominio)

```
state1 = pyhop.State('state1')
state1.coordinates = {'Huelva':{'X':25,'Y':275},
                      'Cadiz':{'X':200,'Y':50}, ...}
state1.connection = {'Huelva':{'Sevilla'},
                     'Cadiz':{'Sevilla','Malaga'}, …}
```

Información dinámica

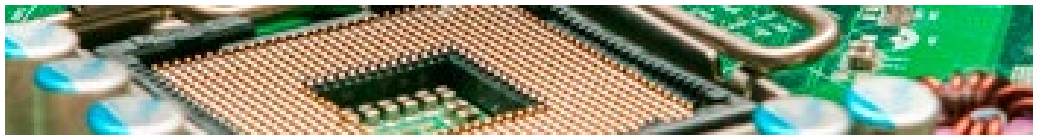
```
state1.location = 'Huelva'
state1.path = ['Huelva']
state1.cost = 0
```

Especificación de objetivos

```
nombre_goal = pyhop.Goal('etiqueta_goal')
nombre_goal.variable = valor_final
…
```

```
goal1 = pyhop.Goal('goal1')
goal1.final = 'Almeria'
```

Resolución del problema

Se invoca a pyhop con el estado inicial y la tarea a resolver junto con sus parámetros

```
pyhop.pyhop(state, [objetivos_a_resolver], verbose=…)
```

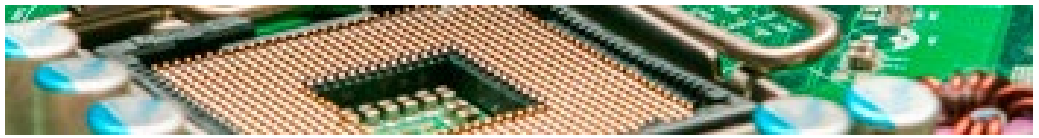
```
pyhop.pyhop(state1,[('travel',goal1)],verbose=3)
```

En este caso el resultado (**secuencia de operadores**) es:

```
[('travel_op', 'Sevilla'), ('travel_op', 'Malaga'),
('travel_op', 'Almeria')]
```

```
    state1.cost = 1058.25786834
    state1.location = Almeria
    state1.path = ['Huelva', 'Sevilla', 'Malaga', 'Almeria']
```

Revisar salida por consola

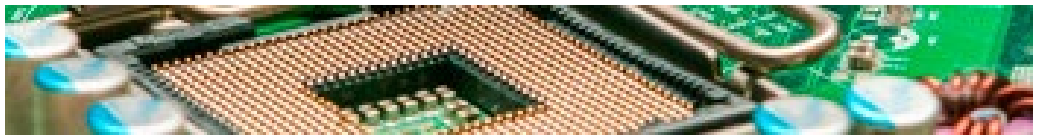


Ejercicio 1

Utilizando el fichero "ejercicio1.py", realizar las siguientes modificaciones en el estado inicial:

a) Añadir la ciudad de Madrid en la posición (750, 750) y establecer conexiones con Jaén y Sevilla.
b) Modificar el objetivo para viajar de Madrid a Málaga.

Ejecutar el planificador y visualizar el resultado.

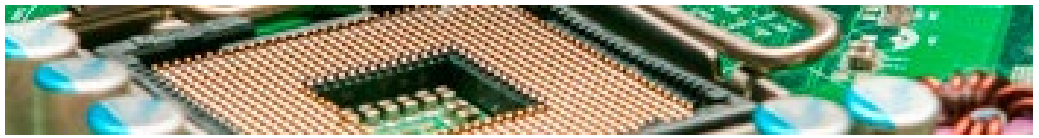


Ejercicio 2

Utilizando el resultado del ejercicio 1, completar el diseño del dominio **añadiendo los operadores** de **subir** y **bajar** de un coche, que trasladará a la persona desde la ciudad origen hasta la ciudad destino. Para ello:

- Se debe definir una nueva variable en el estado para indicar la localización del coche
- El operador de subir debe comprobar que la ciudad actual del coche debe ser la misma que la de la persona y su efecto será que la persona está dentro del coche. Análogamente, el operador de bajar comprobará que la persona está en un coche y su efecto será que la persona está en la ciudad en la que actualmente está el coche
- Se debe modificar el operador *travel_op* para que compruebe que la persona se encuentra en un coche y modifique la ubicación del coche y no de la persona

Nota. Podéis utilizar el archivo "ejercicio2 - Esquema inicial" como base

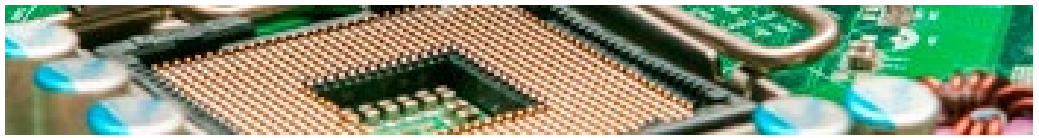


Ejercicio 3

¿Y si hubiera 3 coches distintos con los que podemos realizar el transporte, cada uno con una cantidad de combustible inicial?

Ejemplo:

```
state1.cars = {'c0':{'fuel':100, 'location':'Huelva'},
               'c1':{'fuel':500, 'location':'Huelva'},
               'c2':{'fuel':2000, 'location':'Huelva'}}
```

Otros ejercicios…

¿Y si antes de ejecutar el operador "travel_op" quisiéramos repostar combustible para ir **siempre** con el depósito lleno?

¿Y si realmente solo queremos repostar cuando el vehículo esté en reserva?

¿Y si antes de repostar tenemos que sacar dinero de un cajero electrónico porque no tenemos dinero suficiente en efectivo (no queremos pagar con tarjeta)?

Otras ampliaciones que elija el alumno/a (por ejemplo las gasolineras no están en todos los sitios, ni los cajeros, etc.)