PYTHON

Advanced OOP

Object Persistence

OBJECT PERSISTENCE

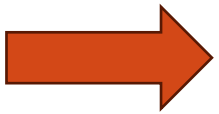
Shallow and deep copy operations

OBJECT PERSISTENCE

- La función id()
  - Proporciona la 'identidad' de un objeto.
    - Un número entero que se asigna durante toda la vida de un objeto.
    - Si las vidas de dos objetos no se solapan, pueden tener el mismo id.
    - **Atención**: no crear variables (incluye funciones) con el nombre.

```
a_string = '10 days to departure'
b_string = '20 days to departure'

print('a_string identity:', id(a_string))
print('b_string identity:', id(b_string))
```



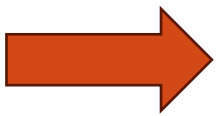
```
a_string identity: 1414357578800
b_string identity: 1414357579248
```

OBJECT PERSISTENCE

- La función id()
  - Dos variables que hacen referencia al mismo objeto tienen el mismo id.

```
a_string = '10 days to departure'
b_string = a_string

print('a_string identity:', id(a_string))
print('b_string identity:', id(b_string))
```



```
a_string identity: 2320672748592
b_string identity: 2320672748592
```

OBJECT PERSISTENCE

- Diferencias entre == y el operador **is**
  - == compara valores
  - **is** compara objetos (será True si tienen la misma identidad)

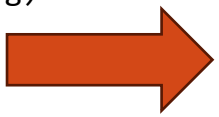
```
a_string = ['10', 'days', 'to', 'departure']
b_string = a_string

print('a_string identity:', id(a_string))
print('b_string identity:', id(b_string))
print('The result of the value comparison:', a_string == b_string)
print('The result of the identity comparison:', a_string is b_string)

print()

a_string = ['10', 'days', 'to', 'departure']
b_string = ['10', 'days', 'to', 'departure']

print('a_string identity:', id(a_string))
print('b_string identity:', id(b_string))
print('The result of the value comparison:', a_string == b_string)
print('The result of the identity comparison:', a_string is b_string)
```



```
a_string identity: 1920628019584
b_string identity: 1920628019584
The result of the value comparison: True
The result of the identity comparison: True

a_string identity: 1920628021440
b_string identity: 1920628339456
The result of the value comparison: True
The result of the identity comparison: False
```

OBJECT PERSISTENCE

- La copia mediante la asignación de slicing (*shallow copy* o copia superficial), genera un nuevo objeto, pero hay que prestar atención al contenido.

```
print("Part 1")
print("Let's make a copy")
a_list = [10, "banana", [997, 123]]
b_list = a_list[:]
print("a_list contents:", a_list)
print("b_list contents:", b_list)
print("Is it the same object?", a_list is b_list)
print("Part 2")
print("Let's modify b_list[2]")
b_list[0] = 33
b_list[2][0] = 112
print("a_list contents:", a_list)
print("b_list contents:", b_list)
print("Is it the same object?", a_list is b_list)
```

```
Part 1
Let's make a copy
a_list contents: [10, 'banana', [997, 123]]
b_list contents: [10, 'banana', [997, 123]]
Is it the same object? False
Part 2
Let's modify b_list[2]
a_list contents: [10, 'banana', [112, 123]]
b_list contents: [33, 'banana', [112, 123]]
Is it the same object? False
```

OBJECT PERSISTENCE

- *Shallow copy* o copia superficial sólo copia con un nivel de profundidad utilizando para el resto de los niveles referencias a los objetos existentes.

```
print("Part 1")
print("Let's make a copy")
a_list = [10, "banana", [997, 123]]
b_list = a_list[:]
print("a_list contents:", a_list)
print("b_list contents:", b_list)
print("Is it the same object?", a_list is b_list)
print("Part 2")
print("Let's modify b_list[2]")
b_list[0] = 33
b_list[2][0] = 112
print("a_list contents:", a_list)
print("b_list contents:", b_list)
print("Is it the same object?", a_list is b_list)
```

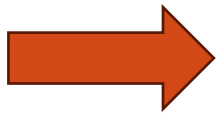
```
Part 1
Let's make a copy
a_list contents: [10, 'banana', [997, 123]]
b_list contents: [10, 'banana', [997, 123]]
Is it the same object? False
Part 2
Let's modify b_list[2]
a_list contents: [10, 'banana', [112, 123]]
b_list contents: [33, 'banana', [112, 123]]
Is it the same object? False
```

OBJECT PERSISTENCE

- *Deep copy* o copia profunda construye un nuevo objeto a partir del primero de forma recursiva.
- Requiere importar el módulo **copy** y utilizar la función **deepcopy**.

```
import copy

print("Part 1")
print("Let's make a copy")
a_list = [10, "banana", [997, 123]]
b_list = copy.deepcopy(a_list)
print("a_list contents:", a_list)
print("b_list contents:", b_list)
print("Is it the same object?", a_list is b_list)
print("Part 2")
print("Let's modify b_list[2]")
b_list[0] = 33
b_list[2][0] = 112
print("a_list contents:", a_list)
print("b_list contents:", b_list)
print("Is it the same object?", a_list is b_list)
```



```
Part 1
Let's make a copy
a_list contents: [10, 'banana', [997, 123]]
b_list contents: [10, 'banana', [997, 123]]
Is it the same object? False
Part 2
Let's modify b_list[2]
a_list contents: [10, 'banana', [997, 123]]
b_list contents: [33, 'banana', [112, 123]]
Is it the same object? False
```

OBJECT PERSISTENCE

- El módulo **copy** dispone de la función **copy** para realizar *shallow copy*.
- Se pueden hacer copias *shallow* utilizando slicing.
- Se pueden hacer copias *shallow* utilizando la función *list* o *dict*:
  - l1 = list(l2)
  - d1 = dict(d2)

OBJECT PERSISTENCE

- Respecto del rendimiento de la copia en función de su tipo:
  - Asignación < Copia superficial < Copia profunda.

```
import copy
import time

a_list = [(1,2,3) for x in range(1_000_000)]

print('Single reference copy')
time_start = time.time()
b_list = a_list
print('Execution time:', round(time.time() - time_start, 3))
print('Memory chunks:', id(a_list), id(b_list))
print('Same memory chunk?', a_list is b_list)

print()

print('Shallow copy')
time_start = time.time()
b_list = a_list[:]
print('Execution time:', round(time.time() - time_start, 3))
print('Memory chunks:', id(a_list), id(b_list))
print('Same memory chunk?', a_list is b_list)

print()

print('Deep copy')
time_start = time.time()
b_list = copy.deepcopy(a_list)
print('Execution time:', round(time.time() - time_start, 3))
print('Memory chunks:', id(a_list), id(b_list))
print('Same memory chunk?', a_list is b_list)
```

OBJECT PERSISTENCE

- **copy.deepcopy** también permite copiar diccionarios y objetos propios.

```
import copy

class Example:
    def __init__(self):
        self.properties = ["112", "997"]
        print("Hello from __init__()")

a_example = Example()
b_example = copy.deepcopy(a_example)
print("Memory chunks:", id(a_example), id(b_example))
print("Same memory chunk?", a_example is b_example)
print()
print("Let's modify the movies list")
b_example.properties.append("911")
print('a_example.properties:', a_example.properties)
print('b_example.properties:', b_example.properties)
```

OBJECT PERSISTENCE

Serialization of Python objects using the pickle module

OBJECT PERSISTENCE

- Módulo pickle
  - Permite la serialización y deserialización
  - ¿Qué se puede 'picklear' (serializar)?
    - None, booleans;
    - integers, floating-point numbers, complex numbers;
    - strings, bytes, bytearrays;
    - tuples, lists, sets, and dictionaries containing pickleable objects;
    - objects, including objects with references to other objects (remember to avoid cycles!)
    - references to functions and classes, but not their definitions.

OBJECT PERSISTENCE

- import pickle
  - Función dump→vuelca un objeto en un fichero)
- Importante manejar los archivos en modo **binary**.

OBJECT PERSISTENCE

- Módulo pickle.
  - Serialización y almacenamiento en fichero
  - Función **dump**

```
import pickle

a_dict = dict()
a_dict['EUR'] = {'code':'Euro', 'symbol': '€'}
a_dict['GBP'] = {'code':'Pounds sterling', 'symbol': '£'}
a_dict['USD'] = {'code':'US dollar', 'symbol': '$'}
a_dict['JPY'] = {'code':'Japanese yen', 'symbol': '¥'}

a_list = ['a', 123, [10, 100, 1000]]

with open('multidata.pckl', 'wb') as file_out:
    pickle.dump(a_dict, file_out)
    pickle.dump(a_list, file_out)
```

OBJECT PERSISTENCE

- Módulo pickle
  - Carga desde fichero y deserialización
  - Función **load**

```
import pickle

with open('multidata.pckl', 'rb') as file_in:
    data1 = pickle.load(file_in)
    data2 = pickle.load(file_in)

print(type(data1))
print(data1)
print(type(data2))
print(data2)
```

OBJECT PERSISTENCE

- Módulo pickle
  - Para poder transmitir por una red o almacenar en una base de datos se debe trabajar con bytes.
  - Función **dumps** → Convierte un objeto 'normal' a un objeto 'bytes'
  - Función **loads** → Convierte un objeto 'bytes' a un objeto 'normal'

```
import pickle

a_list = ['a', 123, [10, 100, 1000]]
bytes = pickle.dumps(a_list)
print('Intermediate object type, used to preserve data:', type(bytes))

# now pass 'bytes' to appropriate driver

# therefore when you receive a bytes object from an appropriate driver you can
deserialize it
b_list = pickle.loads(bytes)
print('A type of deserialized object:', type(b_list))
print('Contents:', b_list)
```

OBJECT PERSISTENCE

- Módulo pickle
  - Intentar 'picklear' un objeto que no lo permita **genera PicklingError**.
  - Intentar 'picklear' una estructura recursiva **puede** generar **Recursion Error** (si la recursión es muy profunda)
  - Intentar 'despickear' una función o clase si esta no está en el espacio de nombres **genera** AttributeError.
  - NOTA: Las funciones y las clases se 'picklean' sólo por su nombre, sin incluir atributos ni código. De esta manera se puede reconstruir una función o clase aunque haya cambiado internamente.

OBJECT PERSISTENCE

- Módulo pickle:
  - Almacenamiento y recuperación de una función.

```
import pickle

def f1():
    print('Hello from the jar!')

#Serializando y almacenando la función
with open('function.pckl', 'wb') as file_out:
    pickle.dump(f1, file_out)

#Leyendo y deserializando la función
with open('function.pckl', 'rb') as file_in:
    data = pickle.load(file_in)

print(type(data))
print(data)
data() #Invocando a la función deserializada
```

OBJECT PERSISTENCE

- Módulo pickle:
  - Almacenamiento y recuperación de un objeto.

```
import pickle

#Definición de la clase
class Cucumber:
    def __init__(self):
        self.size = 'small'

    def get_size(self):
        return self.size

#Instanciación del objeto
cucu = Cucumber()
```

```
#Serialización y almacenamiento del objeto
with open('cucumber.pckl', 'wb') as file_out:
    pickle.dump(cucu, file_out)

#Lectura y deserialización del objeto
with open('cucumber.pckl', 'rb') as file_in:
    data = pickle.load(file_in)

print(type(data))
print(data)
print(data.size)
print(data.get_size())
```

OBJECT PERSISTENCE

- Módulo pickle. Consideraciones.
  - El proceso de serialización de Python es incompatible con otros lenguajes (Java o C++ por ejemplo). Para lograr dicha compatibilidad hay desarrollar soluciones basadas en JSON o XML.
  - El formato binario generado por el módulo pickle puede cambiar entre versiones de Python.
  - El módulo pickle no está protegido frente a datos erróneos. Hay que tener precaución con la fuente de los datos serializados.

OBJECT PERSISTENCE

- Módulo shelve.
  - Construido sobre **pickle**.
  - Implementa un diccionario de serialización.
  - Proceso de escritura:
    - Importación del módulo: **import shelve**
    - A través de **shelve**, apertura del fichero con el diccionario en el modo adecuado.

```
shelve_name = 'first_shelve.shlv'
my_shelve = shelve.open(shelve_name, flag='c')
```

    - Escritura y cierre.

```
my_shelve['USD'] = {'code':'US dollar', 'symbol': '$'}
my_shelve['JPY'] = {'code':'Japanese yen', 'symbol': '¥'}
my_shelve.close()
```

OBJECT PERSISTENCE

- Módulo shelve.
  - Proceso de lectura:
    - Importación del módulo: **import shelve**
    - A través de **shelve**, apertura del fichero con el diccionario en el modo adecuado.

```
shelve_name = 'first_shelve.shlv'
new_shelve = shelve.open(shelve_name)
```

    - Lectura y cierre.

```
print(new_shelve['USD'])
new_shelve.close()
```

OBJECT PERSISTENCE

- Módulo shelve.
  - Ejemplo completo:

```
import shelve

shelve_name = 'first_shelve.shlv'
my_shelve = shelve.open(shelve_name, flag='c')

my_shelve['EUR'] = {'code':'Euro', 'symbol': '€'}
my_shelve['GBP'] = {'code':'Pounds sterling', 'symbol': '£'}
my_shelve['USD'] = {'code':'US dollar', 'symbol': '$'}
my_shelve['JPY'] = {'code':'Japanese yen', 'symbol': '¥'}
my_shelve.close()

new_shelve = shelve.open(shelve_name)
print(new_shelve['USD'])
new_shelve.close()
```

OBJECT PERSISTENCE

- Módulo shelve.
  - Modos de apertura del fichero del diccionario:

The meaning of the optional flag parameter:

Value	Meaning
'r'	Open existing database for reading only
'w'	Open existing database for reading and writing
'c'	Open database for reading and writing, creating it if it doesn't exist (this is a default value)
'n'	Always create a new, empty database, open for reading and writing

OBJECT PERSISTENCE

- Módulo shelve.
  - Consideraciones sobre shelve:
    - Las claves deben ser cadenas.
    - Utiliza un buffer que puede ser vaciado de manera forzada mediante el método **sync** de la clase **shelve**.
    - El método **close** vacía el buffer.
    - Se pueden utilizar las siguientes utilidades de diccionario sobre el objeto **shelve**:
      - Función **len()**
      - Operador **in**
      - Métodos **items()** y **keys()**
      - Métodos **update()**
      - La palabra clave **del**

OBJECT PERSISTENCE

- Módulo shelve.
  - Consideraciones sobre shelve:
    - No hay que modificar los archivos creados por shelve.
    - Es más eficiente que pickle.
    - No es seguro cargar datos desde fuentes no confiables.