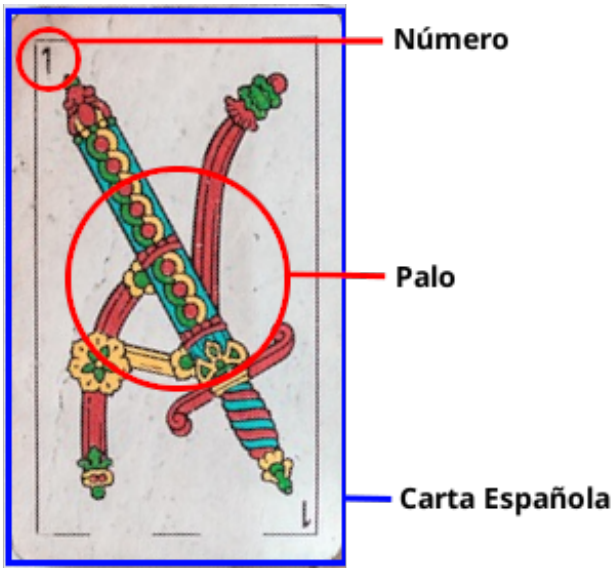
1. Clases

Como se ha mencionado en el *Apunte Nº2*, `Python` es un lenguaje orientado a objetos. Si bien no se abordará este tema en este momento, podemos decir que una **clase** es un modelo que caracteriza objetos. En este momento será utilizado simplemente para **estructurar datos**. Es decir, como un contenedor de datos.

Los registros son justamente eso, un *modelo de datos*. Por ejemplo, imagine que se quisiera programar un juego que utilice **cartas españolas**. Se podría vincular en arreglos separados (uno para el nombre del palo y otro para los números), pero esto terminaría siendo muy engorroso. Es posible utilizar una clase para agrupar estos datos de la siguiente manera:

```
class CartaEspanyola:
  palo   : str
  numero : int
```



Aquí se utiliza la palabra reservada (*keyword*) `class` para definir la clase y luego dentro de este ámbito escribimos las variables correspondientes y el tipo de dato que se almacenará en cada uno de los **atributos** de la clase. En `Python` es habitual nombrar clases utilizando *UpperCamelCase*[1].

1.1. Utilización

Una vez definida nuestra clase, podremos declararla como si se tratara de una variable:

```
mi_carta = CartaEspanyola()
```

Luego para acceder a cada variable miembro interna de la clase debemos utilizar el operador *punto* (`'.'`). Tanto para leer como para escribir valores en las mismas.

[1] `https://es.wikipedia.org/wiki/Camel_case`

Matías S. Ávalos© CC BY NC SA 4.0

```
mi_carta.palo   = "Basto"
mi_carta.numero = 4

print(f"{mi_carta.palo} - {mi_carta.numero}")
```

La salida generada por este fragmento de código sería:

```
Basto - 4
```

1.2. Funciones y clases

Como ya se ha analizado anteriormente, las variables en **Python** contienen referencia en memoria a los objetos. Es por ello que cuando pasamos una clase como argumento a un parámetro de alguna función, ésta será capaz de modificar sus variables internas. Es decir que las clases son **mutables**.

Programa 1: Funciones con clases

```
1  class CartaEspanyola:
2    palo   : str
3    numero : int
4
5  def mostrar_carta(c : CartaEspanyola):
6    print(f"{c.numero} de {c.palo}")
7
8  def establecer_carta(c : CartaEspanyola, palo :str, num :int):
9    c.palo   = palo
10   c.numero = num
11
12 def main():
13   mi_carta = CartaEspanyola()
14   establecer_carta(mi_carta, "Espada", 1)
15   mostrar_carta(mi_carta);
16
17   establecer_carta(mi_carta, "Basto", 4)
18   mostrar_carta(mi_carta);
19
20 main()
```

2. Enumeradores

Los enumeradores son recursos que ayudan a mantener los límites y ayudar al programador a no cometer errores. En el ejemplo del Programa 1, solo existen 4 palos posibles: Basto, Copa, Espada y Oro. Por lo tanto limitar la variable a esos 4 únicos valores es algo deseable. Esto se hace asignando un valor numérico que representa dichos estados (de allí el nombre de *enumerar*). Esto en **Python** se hace de la siguiente manera:

Matías S. Ávalos© CC BY NC SA 4.0

```
import enum

class Palo(enum.Enum):
  BASTO  = 0
  COPA   = 1
  ESPADA = 2
  ORO    = 3
```

Note que `Palo` es también un clase, pero utiliza `enum.Enum` de base. Para ello es necesario importar el módulo estándar `enum`. La definición de los valores dentro de la clase dependerá de la implementación que se le dé al enumerador.

La decisión dependerá de las facilidades que otorgue este mecanismo. Se reescribirá el Programa 1 utilizando un enumerador:

Programa 2: Uso de enumeradores

```
1  import enum
2
3  class Palo(enum.Enum):
4    BASTO  = 0
5    COPA   = 1
6    ESPADA = 2
7    ORO    = 3
8
9  class CartaEspanyola:
10   palo   : Palo
11   numero : int
12
13 def mostrar_carta(c : CartaEspanyola):
14   print(f"{c.numero} de {c.palo.name}")
15
16 def establecer_carta(c : CartaEspanyola, p :Palo, num :int):
17   c.palo   = p
18   c.numero = num
19
20 def main():
21   mi_carta = CartaEspanyola()
22   establecer_carta(mi_carta, Palo.ESPADA, 1)
23   mostrar_carta(mi_carta);
24
25   establecer_carta(mi_carta, Palo.BASTO, 4)
26   mostrar_carta(mi_carta);
27
28 main()
```

Al ser una variable del tipo `Palo`, esta puede ser asignada únicamente a valores utilizando el operador *punto* (p.e. `Palo.ESPADA`), limitando solo a las opciones correctas.

Además los enumeradores nos proporcionan un `.name` que nos devuelve el nombre, y otro llamado `.value` que devuelve el valor numérico equivalente.

Matías S. Ávalos© CC BY NC SA 4.0

3. Arreglo de Registros

Una de las ventaja más evidente es el poder agrupar los datos en un solo arreglo de datos. Esto hace que no necesitemos tener dos arreglos independientes para datos relacionados. Es parecido a tener una tabla de datos, donde cada elemento del arreglo es una fila y cada miembro de la estructura una columna.

baraja		
índice	**palo**	**número**
0	BASTO	1
1	BASTO	2
...	...	...
47	ORO	12

Cada registro (fila), es una `CartaEspanyola`, y `baraja` es el arreglo de 48 cartas. Observe una posible implementación de esto modificado el Programa 2:

Programa 3: Arreglo de cartas

```
1  import enum
2  import random
3
4  class Palo(enum.Enum):
5    BASTO  = 0
6    COPA   = 1
7    ESPADA = 2
8    ORO    = 3
9
10 class CartaEspanyola:
11   palo   : Palo
12   numero : int
13
14 def inicializar_baraja(baraja : []):
15   for p in range(4):
16     for n in range(12):
17       # Se *castea* p de int a Palo: 0->Basto, 1->Copa, etc.
18       baraja[(12 * p) + n].palo   = Palo(p)
19       baraja[(12 * p) + n].numero = n + 1
20
21 def mostrar_carta(c : CartaEspanyola):
22   print(f"{c.numero} de {c.palo.name}")
23
24 def main():
25   # Se declara y carga el arreglo con las cartas:
26   CANT_CARTAS = 48
27   baraja = [CartaEspanyola() for i in range(CANT_CARTAS)]
28   inicializar_baraja(baraja)
29
30   # Se muestran ordenadas
31   print("Baraja creada:")
32   for i in range(CANT_CARTAS):
```


Matías S. Ávalos© CC BY NC SA 4.0

```
33      mostrar_carta(baraja[i])
34
35    # Se mezclan las cartas
36    random.shuffle(baraja)
37
38    # Se muestran las cartas mezcladas:
39    print("\n\n" "Baraja mezclada")
40    for i in range(CANT_CARTAS):
41      mostrar_carta(baraja[i])
42
43  main()
```

En el ejemplo anterior, se "llena" el arreglo `baraja` invocando, en la *línea 28* a la función `inicializar_baraja()`, cargando ordenadamente por palo y número, y cuya implementación se encuentra entre las *líneas 14 y 19*. Luego, es mostrada de esta forma, recorriendo las 48 cartas de la baraja española (líneas 32 a 33). Se utiliza la función `shuffle()` para mezclar la baraja (*línea 36*) y luego mostrar cómo quedó (*líneas 40 y 41*).

Puede ver cómo es posible acceder a los miembros de la estructura dentro del arreglo utilizando primero `'[]'` para seleccionar el elemento del arreglo y luego el operador punto `'.'` para seleccionar el miembro en las *líneas 18 y 19*.

3.1. Arreglos dentro de registros

Una estructura podría tener como miembro un arreglo, incluso un arreglo de registros:

```
class Cancion:
  titulo   : str
  artista  : str
  duracion : float

class Album:
  titulo : str
  track  : list[Cancion]
  precio : float
```

Debe observarse que Album tiene una lista (arreglo), que contiene datos del tipo `Cancion`, pero este no está inicializado, debemos hacerlo luego.

En este ejemplo a continuación, observamos que un álbum musical, efectivamente contiene un arreglo de registros del tipo canción. Y su utilización sería de la siguiente manera:

```
CANT_CANCIONES = 9

mas_vendido = Album()
# Instanciar la lista de canciones:
mas_vendido.track = [Cancion() for i in range(CANT_CANCIONES)]
mas_vendido.precio = 125.59
mas_vendido.titulo = "Thriller" # nombre del album
```


Matías S. Ávalos© CC BY NC SA 4.0

```
mas_vendido.track[3].duracion = 5.96
mas_vendido.track[3].artista = "Michael Jackson"
mas_vendido.track[3].titulo = "Thriller" # nombre canción
```

4. Ejercitación

Codifique los siguientes programas utilizando arreglos de clases y funciones en `Python`.

1. Pida ingresar nombre, apellido y DNI de 10 personas. Muestre las 10 personas con sus datos completos ordenados por su apellido en el siguiente formato:

   ```
   <Apellido>, <Nombre>
   DNI: <XXXXXXXX>
   ```

2. Cree un tipo de dato para almacenar números complejos que contenga parte real e imaginaria. Desarrolle dos funciones que puedan mostrarlos en su forma binómica $(a + bi)$ y polar $(|m| \pm A^o)$.

3. Cree el tipo de dato para almacenar punto en el plano (x,y). Luego, ingrese los puntos `A`, `B`, `C` y `D`. Muestre la suma de los lados de la figura `ABCD`.

4. Cree un tipo de dato para almacenar punto en el espacio (x,y,z) e ingrese 3 puntos. Calcule el perímetro del triangulo formado por los mismos.

5. Cree un tipo dato para una bicicleta que contenga marca, modelo, rodado, precio y cantidad en *stock* (puede ser `0`). Ingrese 10 bicicletas y liste solo las que tengan stock.

6. Ingrese 10 tipos de yerba mate, constituidos por marca, intensidad (suave, mediana y fuerte) y precio. Muéstrelas agrupadas por intensidad.

7. Cree el tipo de dato "libro" que contenga, nombre, isbn y autor. Este último, a su vez contiene nombre, apellido, fecha de nacimiento y nacionalidad. Ingrese los datos de 5 libros y muéstrelos agrupados por nombre del autor.

8. Cree el tipo para jugadores de fútbol que contenga apellido, valor en el mercado (en millones de euros), edad, cantidad de partidos jugados, promedio de goles por partido. Ingrese 10 jugadores y muéstrelos ordenados por valor en el mercado.

9. Ingrese 5 capitales del mundo con su nombre, país, cantidad de habitantes y metros cuadrados. Muéstrelas ordenadas de forma descendente por $habitantes/m^2$

10. Cree un tipo de dato celular constituido por marca, modelo, pulgadas de la pantalla, cantidad de núcleos, RAM y precio. Ingrese 10 celulares y muéstrelos agrupados por marca y ordenados por precio.

Matías S. Ávalos© CC BY NC SA 4.0