

Práctica 1

de Estructura de Datos

2014-2015

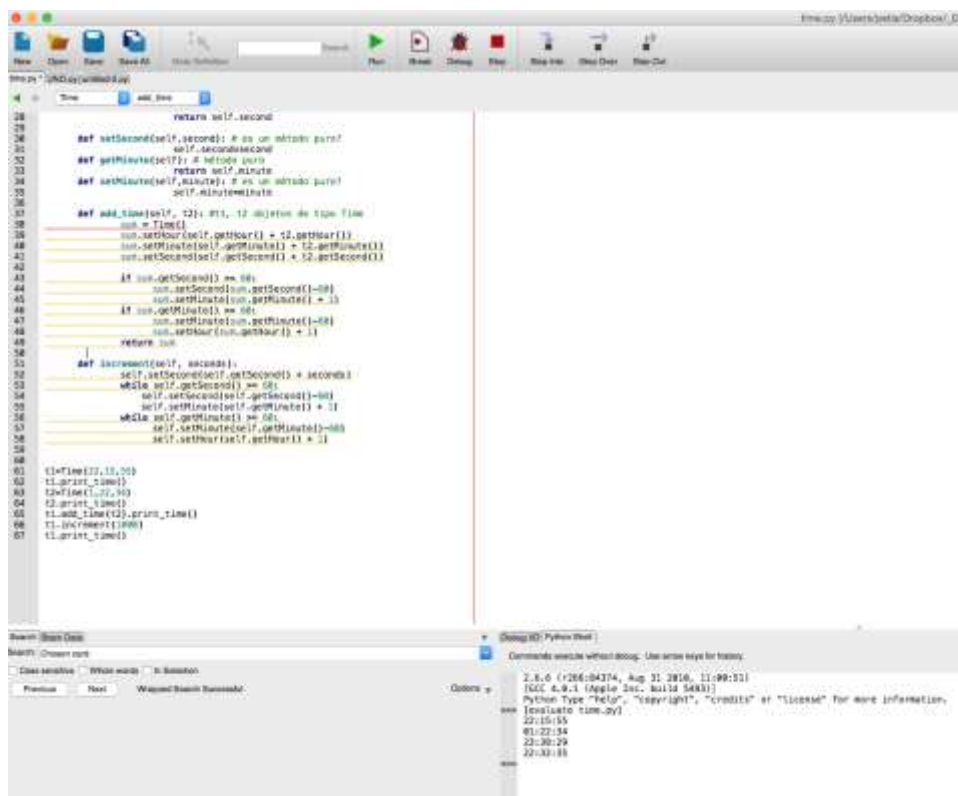
El objetivo de esta práctica es conocer y practicar los tipos de datos abstractos (clases) en Python. Para esto necesitaremos trabajar con Wing IDE como entorno para desarrollar, debugar y completar el código de la práctica.

El Wingware – Wing IDE 101 4.0.1-1 (<https://wingware.com>) ofrece un entorno de desarrollo integrado especialmente diseñado para el lenguaje de programación Python. Con esta herramienta tendremos un editor de código, un compilador y un depurador que ayuda a localizar los errores de código que puedan haber y solucionarlos fácilmente.

Descargar y explorar el entorno. Probar el código de la clase de teoría sobre Clases, en particular, la clase Time.

Ejercicio 1

Escribir la clase Time con sus métodos (constructor, print_time, increment, time_to_int, is_after, getHour, setHour, getSecond, setSecond, getMinute, setMinute) vista en clase de teoría.



```
28         return self._second
29
30     def setSecond(self, second): # es un método para?
31         self._second=second
32
33     def getMinute(self): # método para
34         return self._minute
35
36     def setMinute(self, minute): # es un método para?
37         self._minute=minute
38
39     def add_time(self, t2): # t1, t2 objetos de tipo Time
40
41         t3 = Time()
42         t3.setHour(self.getHour() + t2.getHour())
43         t3.setMinute(self.getMinute() + t2.getMinute())
44         t3.setSecond(self.getSecond() + t2.getSecond())
45
46         if t3.getSecond() >= 60:
47             t3.setSecond(t3.getSecond()-60)
48             t3.setMinute(t3.getMinute() + 1)
49         if t3.getMinute() >= 60:
50             t3.setMinute(t3.getMinute()-60)
51             t3.setHour(t3.getHour() + 1)
52         return t3
53
54     def increment(self, seconds):
55         self.setSecond(self.getSecond() + seconds)
56         while self.getSecond() >= 60:
57             self.setSecond(self.getSecond()-60)
58             self.setMinute(self.getMinute() + 1)
59         while self.getMinute() >= 60:
60             self.setMinute(self.getMinute()-60)
61             self.setHour(self.getHour() + 1)
62
63 t1=Time(12,15,30)
64 t1.print_time()
65 t2=Time(1,22,30)
66 t2.print_time()
67 t3=add_time(t2).print_time()
68 t1.increment(1000)
69 t1.print_time()
```

Search (Basic Data) | [Python Shell] | Commands executed without debug. Use arrow keys for history.

```
Python 2.7.6 (r266:84774, Aug 31 2010, 11:09:31)
GCC 4.6.1 (Apple Inc. build 5483)
Python Type "help()", "copyright()", "credits()" for more information.
>>> from time import *
>>> t1=Time(12,15,30)
>>> t1.print_time()
12:15:30
>>> t2=Time(1,22,30)
>>> t2.print_time()
01:22:30
>>> t3=add_time(t2).print_time()
13:38:00
>>> t1.increment(1000)
>>> t1.print_time()
13:39:00
```

Fig.1 El entorno Wing IDE

Grabar el fichero con el nombre *Time.py*.

Ejecutar el siguiente código:

```
def ej1():  
  
    t1=Time(22,15,55)  
    t1.print_time()  
    t2=Time(1,22,34)  
    t2.print_time()  
    t1.add_time(t2).print_time()  
    t1.increment(1000)  
    t1.print_time()  
  
>>>ej1()
```

Explicar qué hace este código y completar los comentarios del código.

Para ejecutar, utilizar el botón con el símbolo verde triangular en la parte superior. Poner un breakpoint en la línea:

```
if sum.getSecond() >= 60:
```

de la función `add_time()` y debugar el código línea por línea utilizando el botón de debugar:



Observar los valores de las variables en la parte inferior izquierda cuando se está debugando el código.

Ejercicio 2

Al final del documento, en el Apéndice 1, encontrareis una lista con los tiempos del último ensayo de los competidores de Fórmula 1, de Sochi. Escribir una función `estadisticaFormulaUno(lista, h, m, s)` que contesta cuántos competidores han conseguido menor tiempo que un tiempo definido con su valor de horas (h), minutos (m) y segundos (s). Definir la función con valores por defecto. Ejecutar el siguiente código:

```
def ej2(h=1,m=40,s=0):  
    lista=[Time(1,38,26),Time(1,39,16),Time(1,39,7),Time(1,39,55),    Time(1,39,54),    Time(1,40,9),  
    Time(1,40,11),    Time(1,40,51),    Time(1,40,05),    Time(1,40,38),    Time(1,40,55),    Time(1,40,56),  
    Time(1,40,55), Time(1,41,46), Time(1,41,52), Time(1,41,15), Time(1,42,36), Time(1,43, 9), Time(1,43,5),  
    Time(1,44,37)]  
    print "El número de deportistas con tiempo menor es: " + str(estadisticaFormulaUno (lista, h,m,s))  
>>>ej2(1,41)
```

Explicar qué hace el código de las dos funciones y completar los comentarios que faltan.

Ejercicio 3

UNO es un juego popular de cartas estadounidense desarrollado en 1971 por Merle Robbins en [Reading, Ohio](#). El juego cuenta con un total de 108 cartas para el juego UNO original y 112 para, el lanzado en 2005, **UNO SPIN** que a diferencia del original tiene una ruleta.



Fig2. El juego del UNO

En qué consiste el juego del UNO (versión básica):

Cartas

Las cartas se dividen en 4 colores: Azul, Rojo, Verde y Amarillo y a su vez cada color trae cartas numéricas del 0 al 9 (dos de cada una).

Reglas del juego:

1. Inicialmente todas las cartas están en un mazo
2. Al principio del juego se reparten un número predefinido (normalmente 7) de cartas a cada jugador, estas cartas proceden del mazo y serán las cartas que tiene el jugador en la mano.
3. Después, se saca aleatoriamente una carta del mazo que servirá como carta inicial de la pila (lo llamaremos `discard_pile`) y el resto de cartas se dejan en el mazo.
4. Todos los jugadores, por turnos, deben tirar cartas a la pila hasta que puedan, del mismo color o número que la carta visible de la pila. La carta visible será la última que se ha tirado a la pila.
5. Si un jugador no tiene una carta para jugar, debe robar del mazo hasta que obtenga una que le permita jugar. Una vez haya robado una que puede tirar, puede ir tirando mientras pueda.
6. Finalmente, se define un criterio de ganar, por ejemplo, gana el que primer jugador que acabe las sus cartas de la mano.

Vamos a implementar durante las próximas sesiones el juego básico del UNO.

I. Anotaciones:

Utilizaremos la siguiente sintaxis:

- Las clases empiezan con mayúscula y el resto de letras va con minúscula.
- Los objetos empiezan con minúscula.
- Los atributos se definen como privados
- Las funciones empiezan en minúscula y las diferentes palabras se distinguen utilizando la primera letra de las palabras en mayúscula (`chequearCartas`) o están separadas por el guión bajo (`chequear_cartas`).
- Siempre que se pueda, aplicar la sobrecarga de métodos en las clases.

II. Estructuras de datos

Vamos a definir las siguientes 4 clases:

1. La clase carta (la definiremos con el nombre **Card**). La clase contendrá:

- a. Datos:
 - i. Color (`color`)
 - ii. Número (`number`)
- b. Métodos: Utiliza como nombre de los métodos los dados entre paréntesis:
 - i. Constructor (`__init__`)
 - ii. Método para imprimir (`__str__`)
 - iii. Comparar dos cartas (`check_card`) - comprueba si dos cartas son compatibles es decir el color o el número de dos cartas coinciden para poder jugar
 - iv. Función de test (`test`) – define una función según tu criterio para probar si la implementación de la clase es correcta. Por ejemplo, una posibilidad es definir una función que escoge aleatoriamente 2 cartas y comprueba si son compatibles.

2. La clase jugador (**Player**). Contendrá:

- a. Datos:
 - i. Nombre (`name`)
 - ii. Cartas de la mano del jugador (`cards`)

- b. Métodos
 - i. Constructor – inicializa el nombre del jugador y sus cartas
 - ii. Imprimir – imprime el nombre del jugador y sus cartas
 - iii. Comprobar si puede jugar (can_play_card) – dada una carta (p.e. la última de la pila), comprueba retornando un booleano si puede coincidir por color o número con por lo menos una de las suyas.
 - iv. Jugar carta (play_a_card) – una vez seleccionada una carta de la mano, juega la carta enviándola a pila y borra la carta de sus cartas de la mano.
 - v. Seleccionar carta (select_card) –dado un número encontrar en sus cartas la primera carta con este número y devolverla.
 - vi. Imprimir cuántas cartas tiene (len) en la mano.
 - vii. Función de test (test) – define una función de test para comprobar la implementación de la clase según tu criterio.
3. La clase mazo (**Deck**) – es el conjunto de cartas de donde sacamos las cartas (invisibles)
- a. Datos:
 - i. Lista de cartas del mazo (cards)
 - b. Métodos
 - i. Constructor – crea todas las cartas con números de 0 a 9 y colores [blue,'green','red','yellow'].
 - ii. Obtener la carta “i” del mazo (__getitem__) – devuelve la carta en la posición “i” del mazo.
 - iii. Retorna cuántas cartas tiene el mazo (len).
 - iv. Borrar la carta “i” del mazo (remove).
 - v. Repartir una carta a un jugador (deal_one_card) –reparte al jugador una carta aleatoria sacándola del mazo.
 - vi. Repartir las cartas (deal) – reparte N cartas a cada uno de los jugadores (p.e. N=7). El valor de N se pondrá como una constante en la primera línea del código.
 - vii. Función de test (test) – define función de test según tu criterio.
4. La clase pila (**Discard_Pile**) - donde amontonamos las cartas, solo la última carta ha de ser visible
- a. Datos
 - i. Discard_pile (una lista de cartas)
 - b. Métodos
 - i. Constructor (__init__) – crea y saca aleatoriamente del mazo (deck).
 - ii. Mostrar la última carta (show_last_card) - retorna la última carta de la pila.
 - iii. Poner una carta sobre la pila (append) – dada una carta, añadirla a la pila.
 - iv. Comprobar cuántas cartas tiene la pila (len).
 - v. Función de test (test) – define función de test según tu criterio.

Comentar qué métodos de las cuatro clases aplican la sobrecarga.

El apéndice 2 muestra el algoritmo completo del juego del UNO (al que llamaremos ONE a partir de ahora) y que acabaremos de implementar en la segunda práctica.

Entrega: Se ha de entregar en un fichero comprimido con el nombre del alumno que contenga: los ficheros con el código en python, compilados en Wing IDE, correspondientes a ejercicios 1 2, y 3, junto con un documento en formato pdf o Word que explique qué hace el código en los ejercicios 1 y 2, y la implementación de las clases en el ejercicio 3. **La fecha límite de entrega es el día 3 de Marzo a las 23:00h.** No se aceptarán entregas de la práctica S1 después de esta fecha.

Apéndice 1:

La lista de tiempos del último ensayo de los competidores de Fórmula 1, de Sochi fue la siguiente:

1. Lewis Hamilton (GBR/Mercedes) 1:38.26
2. Nico Rosberg (GER/Mercedes) 1:39.16
3. Valtteri Bottas (FIN/Williams) 1:39.07
4. Daniel Ricciardo (AUS/Red Bull) 1:39.55
5. Felipe Massa (BRA/Williams) 1:39.54
6. Daniil Kvyat (RUS/Toro Rosso) 1:40.09
7. Kimi Räikkönen (FIN/Ferrari) 1:40.11
8. Fernando Alonso (ESP/Ferrari) 1:40.51
9. Jean-Eric Vergne (FRA/Toro Rosso) 1:40.05
10. Sebastian Vettel (GER/Red Bull) 1:40.38
11. Jenson Button (GBR/McLaren) 1:40.55
12. Nico Hülkenberg (GER/Force India) 1:40.6
13. Sergio Perez (MEX/Force India) 1:40.9
14. Adrian Sutil (GER/Sauber) 1:41.14
15. Esteban Gutierrez (MEX/Sauber) 1:41.52
16. Romain Grosjean (FRA/Lotus) 1:41.15
17. Kevin Magnussen (DIN/McLaren) 1:42.36
18. Marcus Ericsson (SWE/Caterham) 1:43.10
19. Kamui Kobayashi (JPN/Caterham) 1:43.5
20. Max Chilton (GBR/Marussia) 1:44.37

Apéndice 2:

Para implementar el juego, definiremos la clase **ONE** de la siguiente forma:

5. La clase del juego (**ONE**) – crea los datos, prepara el juego y empieza a jugar hasta que se cumpla el criterio de acabar
 - a. Datos
 - i. Lista de jugadores
 - ii. Baraja o mazo (Deck)
 - iii. Pila (Discard_pile)
 - b. Métodos
 - i. Crear juego – prepara el juego (prepare_game) y lo pone en marcha (run_game)

- ii. Preparar_juego (prepare_game)
 - a. Define los jugadores (p.e. consideramos que hay 4 jugadores). Los datos del número de jugadores se leen desde teclado.
 - b. Crea la baraja para repartir (deck)
 - c. Crea la pila (discard_pile)
 - d. Reparte las cartas (deal) del deck
 - e. Se selecciona aleatoriamente el jugador actual (p.e. el primero)
- iii. Repartir N cartas a cada jugador (deal), p.e. N=7. Éste valor de N se define como una constante al principio del código.
- iv. Aplicar criterio para parar (stop_criterion) – Se pueden definir varios criterios, por ejemplo, cuando el primero de los jugadores haya acabado con todas las cartas de su mano, entre otras opciones.
- v. Comprobar y felicitar al campeón (announce_champion) – comprueba qué jugador ha cumplido el criterio de parar el juego en primer lugar.
- vi. Cambiar de turno (change_turn) – si el jugador actual es p1, pasar a p2, etc.
- vii. Función principal (run_game):
 - 1. Mientras no se cumple el criterio de parar
 - i. Se imprime el estado del juego (visualize_state) - la última carta de la pila y las cartas del jugador actual
 - ii. Se comprueba si el jugador actual puede jugar
 - iii. En caso que no: el juego saca automáticamente cartas del mazo y se las asigna a la mano del jugador actual. Éstas cartas se van imprimiendo por pantalla, hasta que pueda jugar.
 - iv. Mientras el jugador actual pueda jugar:
 - 1. El juego pregunta al jugador qué carta desea jugar y se lee por teclado su número
 - 2. El juego selecciona la carta a jugar según el número introducido (select_card de la clase Player)
 - 3. El jugador actual juega la carta (play_a_card)
 - 4. Se visualiza el estado del juego
 - v. Si el jugador actual cumple el criterio de parar el juego, se le felicita y se acaba el juego
 - vi. Sino, se pasa el turno al siguiente jugador

En la página siguiente tenéis un diagrama que resume el funcionamiento del juego del UNO que se desea implementar en la segunda práctica de Estructura de Datos.

Diagrama del algoritmo

