

UNIVERSIDAD DE COSTA RICA
Escuela de Ingeniería Eléctrica
IE0117 – Programación bajo plataformas abiertas

Laboratorio 9

Yeison Rodríguez Pacheco, B56074

18/06/18

Índice

1. Introducción	4
2. Nota teórica	4
3. Análisis de resultados	5
3.1. Clase Contador	5
3.1.1. Función <code>__init__(c,m)</code>	5
3.1.2. Función <code>getCuenta()</code>	6
3.1.3. Función <code>getMax()</code>	6
3.1.4. Función <code>getRebase()</code>	6
3.1.5. Función <code>setCuenta(c)</code>	6
3.1.6. Función <code>setMax(m)</code>	6
3.1.7. Función <code>setRebase(r)</code>	6
3.1.8. Función <code>contar()</code>	6
3.1.9. Función <code>borrarRebase()</code>	6
3.2. Clase <code>Reloj(Contador)</code>	6
3.2.1. Función <code>__init__(s,m,h)</code>	7
3.2.2. Función <code>set(s,m,h)</code>	7
3.2.3. Función <code>tic()</code>	7
3.2.4. Función <code>display()</code>	7
3.2.5. Función <code>__str__()</code>	7
3.3. Clase <code>Mes</code>	7
3.3.1. Función <code>__init__(n,c,m)</code>	7
3.3.2. Función <code>getNombre()</code>	8
3.3.3. Función <code>setNombre(n)</code>	8
3.4. Clase <code>calendario(Mes)</code>	8
3.4.1. Función <code>__init__(d,m,a)</code>	8
3.4.2. Función <code>get()</code>	8
3.4.3. Función <code>set(d,m,a)</code>	8
3.4.4. Función <code>avanzar</code>	8
3.4.5. Función <code>__str__()</code>	8
3.4.6. Función <code>__añoBisiesto()</code>	8
3.5. Clase <code>Fecha(Calendario, Reloj)</code>	9
3.5.1. Función <code>__init__(d,m,a,h,minuto,s)</code>	9
3.5.2. Función <code>avanzar()</code>	9
3.5.3. Función <code>__str__()</code>	9
3.6. Comprobación del funcionamiento	9
4. Conclusiones y recomendaciones	11

Índice de figuras

1. Jerarquía de las clases para el Calendario	5
---	---

2.	Comprobación del funcionamiento del programa con el ejemplo dado en el enunciado .	9
3.	Avanzando de un año a otro	10
4.	Año no bisiesto, comportamiento de febrero	10
5.	Año bisiesto, comportamiento de febrero	10
6.	Avanzando un año y un día, desde un año no bisiesto a uno bisiesto	10
7.	Avanzando sobre todos los meses del año	11

1. Introducción

Existen diversos lenguajes de programación, y cada uno tiene enfoques diferentes, ya vimos que C es utilizado para programas que se ejecutan en tiempo real y necesitan estar optimizados en la gestión de los recursos, pero esto hace que su entorno sea menos rápido de aprender y utilizar, pero Python tiene una filosofía totalmente distinta en donde las prioridades son la facilidad y rapidez con la que el programador puede realizar los códigos. Parte de esta ayuda que brinda Python es la programación orientada a objetos (POO), la cuál consiste en la posibilidad de poder crear objetos que tengan estructuras similares y poder utilizar funciones específicas sobre los mismos. Un ejemplo para entender la POO podría ser la creación de una clase carro, la cual tenga funciones como cambiar una llanta, llenar el tanque de gasolina, pintar, etc. Notamos que estas funciones serían generales para todos los autos, aunque los autos tengan características propias distintas como su marca, altura y distancia que pueden recorrer. Entonces notamos que el potencial de la Programación orientada a objetos radica en tratar conjuntos de elementos con funciones programadas solo una vez, lo que facilita el desarrollo de muchas aplicaciones.

En algún momento el programador podría sentir que si se pudiesen crear varias clases ligadas entre sí se facilitaría la solución de algunos problemas, y esto es posible y recibe el nombre de herencia. La herencia consiste en la capacidad de crear una clase que contenga funciones de otra clase, así que si tuviéramos una clase de vehículos terrestres con funciones como detenerse, acelerar o hacer un cambio de marcha, podríamos hacer que nuestra clase carro también herede esas funciones, por lo que no sería necesario programarlas nuevamente, y se podrían crear funciones dentro de la clase carro que sean solo para la misma.

En este laboratorio se procederá a crear un calendario, el cuál pueda dar la hora exacta a partir de distintos tipos de clases (en total 5), las cuales están ligadas unas con otras. Este calendario estará hecho en base a un diagrama UML, estos diagramas permiten entender programas complejos con facilidad, por lo que son de gran utilidad. El manejo de errores será a la hora de realizar las clases, ya que es común que el usuario no sepa utilizar las mismas y cometa un error al ingresar los datos, por lo que es necesario que el programa sea sólido ante estos inconvenientes.

2. Nota teórica

Las clases creadas en Python tienen funciones especiales, las cuales se indican con el formato `__función__`, una de estas funciones es `__init__()`, la cual es de vital importancia al declarar una clase ya que cuando se crea un objeto esta función es la que se ejecuta de primero, en otras palabras, es la que inicia el objeto con los valores que el usuario desea ingresar. Existen otras funciones especiales como `__str__()` la cual se ejecuta cuando el usuario hace una conversión a tipo string del objeto, por lo que generalmente se utiliza para retornar alguna cadena de caracteres de importancia. También existen funciones privadas y estas son declaradas utilizando el formato `__función()`. Estas funciones solo se ejecutan dentro de la clase y no pueden ser accedidas fuera de la misma. Las funciones públicas se declaran simplemente de la forma `función()`, estas sí pueden ser utilizadas fuera de la clase. [1]

En python es posible utilizar la herencia entre las clases, para esto al declarar la clase de la forma `clase_que_hereda(clase_principal)`, por lo que esta clase heredaría todas las funciones de la clase padre. En este laboratorio se utilizará mucho el término de herencia, ya que se tendrán cinco clases enlazadas entre sí, en la figura 1 se deja un esquema con la jerarquía de las clases a utilizar.

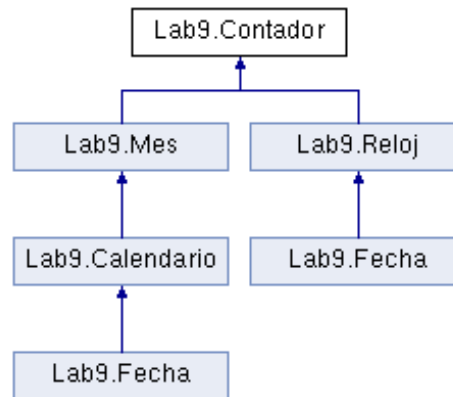


Figura 1: Jerarquía de las clases para el Calendario

El diagrama mostrado anteriormente es útil, pero no refleja mucha información acerca de las clases, ya sean sus parámetros o sus métodos, por lo que es conveniente utilizar diagramas UML (Lenguaje unificado modelado) para representar las clases, los cuales sí especifican toda esta información de forma resumida y clara, por lo que son de gran utilidad tanto para documentar código, como para crearlo a base de los mismos. Este tipo de diagrama posee una gran cantidad de sintaxis para su creación, pero a grandes rasgos se tienen bloques, cada bloque está dividido en tres secciones, en la primera indica el nombre de la clase, en la segunda sus variables y en la última sus funciones. Si las funciones o variables son privadas se utiliza un signo - antes de su declaración, y un + si son públicas. Si una clase hereda de otra esta tendrá una flecha que apunta hacia la que hereda. En caso de necesitar más información se recomienda visitar [2]

3. Análisis de resultados

En esta sección se presentará un análisis del programa realizado, el cual consiste en un conjunto de clases que al juntar sus funciones logran hacer las tareas necesarias para funcionar como un calendario. Se dividirá el análisis de cada clase en una sección, y al final se realizará una comprobación general del funcionamiento.

3.1. Clase Contador

La clase contador se crea para que cumpla las funciones de un contador que cuando llega a un número máximo, se resetea y pone su variable rebase en uno, esto indica que el contador cumplió un ciclo, se utilizará más adelante en otras clases para contar distintos tipos de datos. Se divide en las siguientes funciones

3.1.1. Función `__init__(c,m)`

Esta función es la primera que se ejecuta cuando se inicia un objeto tipo Contador, recibe dos parámetros los cuales son `c` y `m`, con estos valores se inicializan las variables de la clase. `c` es un contador que indica por donde va el contador, y la variable `m` es el máximo valor al que llega la variable `c`. La variable `m` debe ser mayor o igual que `c`.

3.1.2. Función `getCuenta()`

Esta función retorna un entero que indica el valor de `c` que tiene el objeto en ese momento.

3.1.3. Función `getMax()`

Esta función retorna un entero que indica el valor de `m` que tiene el objeto.

3.1.4. Función `getRebase()`

Esta función retorna un entero que indica el valor del rebase que tiene el objeto.

3.1.5. Función `setCuenta(c)`

Esta función recibe un parámetro `c`, el cuál es colocado como el nuevo valor de cuenta del objeto, descartando el anterior.

3.1.6. Función `setMax(m)`

Esta función recibe un parámetro `m`, el cuál es colocado como el nuevo valor máximo al que llega la cuenta del objeto, descartando el anterior.

3.1.7. Función `setRebase(r)`

Esta función recibe un parámetro `r`, el cuál es colocado como el nuevo valor del rebase que posee el objeto, descartando el rebase anterior.

3.1.8. Función `contar()`

Esta función se encarga de aumentar la cuenta de la variable `c` en una unidad, si la variable `c` alcanza el valor máximo (`m`) entonces la variable `c` vuelve a ser cero y se aumenta el valor de la variable de rebase `r` en una unidad.

3.1.9. Función `borrarRebase()`

Esta función se encarga de poner el valor del rebase `r` en cero.

3.2. Clase `Reloj(Contador)`

La clase `reloj` hereda de la clase `contador`, esta clase se encarga de realizar el funcionamiento de un reloj contando en segundos, minutos y horas. A continuación se describirán las funciones que realiza dicha clase.

3.2.1. Función `__init__(s,m,h)`

Función que se ejecuta al iniciar un objeto tipo Reloj, recibe como parametros los segundos, minutos y horas que el usuario desea elegir como parámetros de inicio. Los valores de s y m deben ser menores a 60 mientras que las horas no deben ser mayores a 23. Las variables de la clase se inicializan con estos valores y son objetos del tipo contador. A estos contadores se les coloca como valores máximos los mencionados anteriormente y su valor de contador es el indicado por el usuario en los parámetros s, m y h respectivamente.

3.2.2. Función `set(s,m,h)`

Esta función se encarga de colocar el reloj en los minutos, segundos y horas que el usuario le ingrese, descartando los valores anteriores. Para esto se utiliza la función `setCuenta()` de la clase contador sobre los objetos declarados en la función `init`.

3.2.3. Función `tic()`

Esta función se encarga de avanzar un segundo en el reloj, para esto utiliza la función de la clase contador llamada `contar()` sobre el objeto que contiene los datos respectivos a los segundos, si el valor de rebase de los segundos se coloca en 1 entonces se aumenta un minuto con la misma función y se utiliza la función `borrarRebase()` para reiniciar el rebase de los segundos y que vuelvan a su cuenta desde cero. En caso de que los minutos lleguen a su valor máximo se reinicia su cuenta y se aumenta el valor de las horas. Cuando las horas llegan a su máximo se reinician todos los contadores y el rebase de las horas se coloca en uno.

3.2.4. Función `display()`

Esta función se encarga de retornar un string que contiene los segundos, minutos y horas actuales en un formato tipo: horas:minutos:segundos.

3.2.5. Función `__str__()`

Esta función realiza lo mismo que la anterior pero se activa al hacer la función `str()` sobre un objeto tipo Reloj.

3.3. Clase Mes

Esta clase se encarga de crear objetos del tipo Mes, los cuales contienen el nombre del mes, así como sus días y su día máximo. Esta clase hereda las funciones de la clase Contador. A continuación se presentan las funciones que realiza esta clase.

3.3.1. Función `__init__(n,c,m)`

Esta función es la primera en ejecutarse al crear un objeto, inicializa un objeto contador con los valores de c y m, mientras que el valor de n corresponde al nombre del mes.

3.3.2. Función `getNombre()`

Esta función se encarga de retornar un string con el nombre del mes.

3.3.3. Función `setNombre(n)`

Esta función se encarga de cambiar el nombre del objeto al que el usuario indica en el parámetro `n`

3.4. Clase `calendario(Mes)`

Esta clase hereda los métodos de la clase `mes` y se encarga de tener un calendario anual. A continuación se mostrarán sus funciones

3.4.1. Función `__init__(d,m,a)`

Esta función se ejecuta al crear 12 objetos de tipo `Mes`, los cuales van a contar con los días respectivos de cada uno, y se inicializan con el valor de día que el usuario eligió. El mes de febrero es un caso especial ya que depende del año cambia, más adelante se explicará la función utilizada para controlar esto. La variable `m` guarda el mes actual y la variable `a` el año actual.

3.4.2. Función `get()`

Esta función se encarga de retornar un string con la fecha actual, en formato: día de mes del año.

3.4.3. Función `set(d,m,a)`

Esta función se encarga de poner la fecha ingresada por el usuario como la fecha actual, para esto también revisa si el año es bisiesto.

3.4.4. Función `avanzar`

Esta función se encarga de avanzar un día en el calendario. Si se llega al valor máximo del mes entonces se avanza al siguiente mes y el contador de día se coloca en 1 nuevamente, y su valor máximo es el respectivo de cada mes. En caso de que se llegue al último mes el contador de mes se reinicia y se aumenta el valor del año en una unidad. Al cambiar de año se revisa si el mismo es bisiesto, para corregir si es necesario los valores del mes de febrero.

3.4.5. Función `__str__()`

Esta función se ejecuta al intentar realizar la conversión de un objeto tipo `calendario` a string, retorna un string con la fecha actual en el mismo formato que lo hace la función `get`.

3.4.6. Función `__añoBisiesto()`

Esta es una función privada que se encarga de verificar si el año actual es bisiesto, para esto revisa si el año es divisible entre 4, si es así y no es divisible entre 100 entonces el año es bisiesto. También revisa si el año es bisiesto revisando si el mismo es divisible entre 4, 100 y 400, si cumple todo esto es bisiesto, en caso contrario retorna un valor falso.

3.5. Clase Fecha(Calendario, Reloj)

Esta clase hereda de las clases Calendario y Reloj, y se encarga de dar la fecha exacta con segundos, días, minutos, horas, mes, y año. A continuación se presentan sus funciones.

3.5.1. Función `__init__(d,m,a,h,minuto,s)`

Esta función se ejecuta inicialmente al crear un objeto de tipo Fecha, le pide al usuario los días, minutos, segundos, hora, mes y año.

3.5.2. Función `avanzar()`

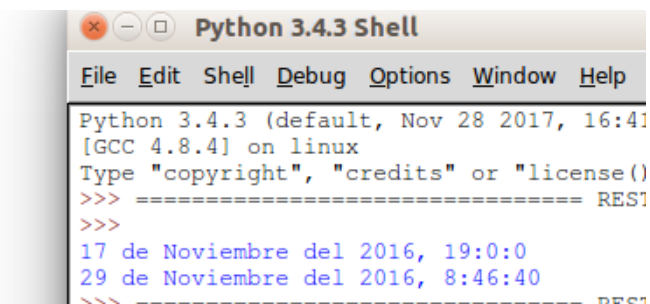
Esta función se encarga de avanzar un segundo en la fecha. Para lograrlo utiliza la función `tic()` de la clase reloj, en caso de que el rebase de las horas se de, se llama a la función `avanzar` de la clase calendario, para que aumente el día en una unidad, y se reinicia el valor de rebase de las horas.

3.5.3. Función `__str__()`

Esta función retorna un string con la fecha actual, en el formato 'día de mes del ano, hh:mm:ss'

3.6. Comprobación del funcionamiento

En esta sección se presentará el funcionamiento del programa entero, no se presentará el uso de cada función individual ya que no es necesario, el programa en sus diferentes clases usa todas las funciones declaradas, por lo que si alguna fallara el programa no funcionaría. En la figura 2 vemos el caso presentado en el enunciado, notamos en la parte izquierda que al llamar la función `avanzar` del objeto tipo Fecha (seteado en la fecha 17 de Noviembre del 2016, 19:00:00) un millón de veces, se obtiene que la fecha actual es 29 de Noviembre del 2016, 8:46:40, que es el resultado que se esperaba. Vamos a ver más adelante varios casos para corroborar la robustez del programa.



```

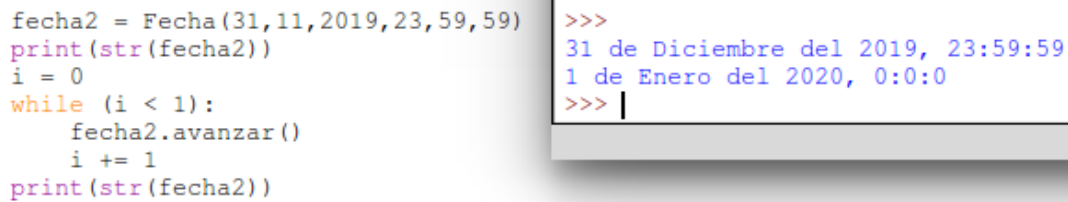
fecha = Fecha(17,10,2016,19,0,0)
print(str(fecha))
i = 0
while (i < 1000000):
    fecha.avanzar()
    i += 1
print(str(fecha))
  
```

```

Python 3.4.3 (default, Nov 28 2017, 16:41
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()"
>>> ----- RESI
>>>
17 de Noviembre del 2016, 19:00:00
29 de Noviembre del 2016, 8:46:40
>>> ----- RESI
  
```

Figura 2: Comprobación del funcionamiento del programa con el ejemplo dado en el enunciado

Vemos ahora en la figura 3 el caso en el que el programa se encuentra en el último momento de un año, y vemos que al llamar a la función `avanzar()` se pasa tanto de año, como de mes y de día, y las horas se reinician, por lo que tiene un buen funcionamiento.



```

fecha2 = Fecha(31,11,2019,23,59,59)
print(str(fecha2))
i = 0
while (i < 1):
    fecha2.avanzar()
    i += 1
print(str(fecha2))

```

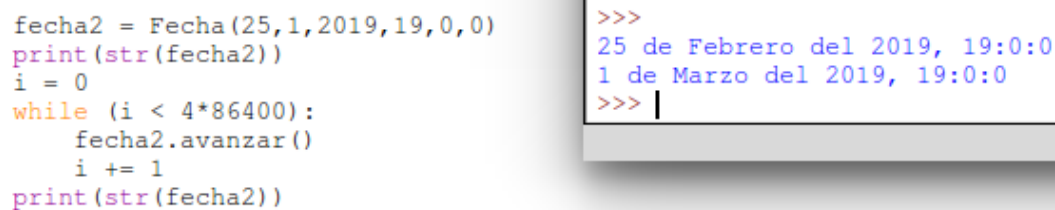
```

>>>
31 de Diciembre del 2019, 23:59:59
1 de Enero del 2020, 0:0:0
>>> |

```

Figura 3: Avanzando de un año a otro

La figura 4 muestra un caso para el cuál el año no es bisiesto, vemos que si el programa se encuentra en el 25 de febrero y se avanzan 4 días exactos, la fecha final es el primero de mayo. Pero veamos ahora el caso de que el año sea bisiesto como en la figura 5, si se avanzan nuevamente 4 días exactos desde el 25 de febrero, el programa finaliza en la fecha 29 de febrero, por lo que queda demostrado que detecta correctamente si un año es bisiesto o no.



```

fecha2 = Fecha(25,1,2019,19,0,0)
print(str(fecha2))
i = 0
while (i < 4*86400):
    fecha2.avanzar()
    i += 1
print(str(fecha2))

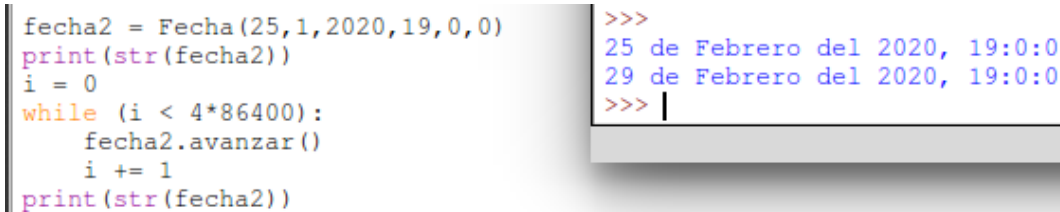
```

```

>>>
25 de Febrero del 2019, 19:0:0
1 de Marzo del 2019, 19:0:0
>>> |

```

Figura 4: Año no bisiesto, comportamiento de febrero



```

fecha2 = Fecha(25,1,2020,19,0,0)
print(str(fecha2))
i = 0
while (i < 4*86400):
    fecha2.avanzar()
    i += 1
print(str(fecha2))

```

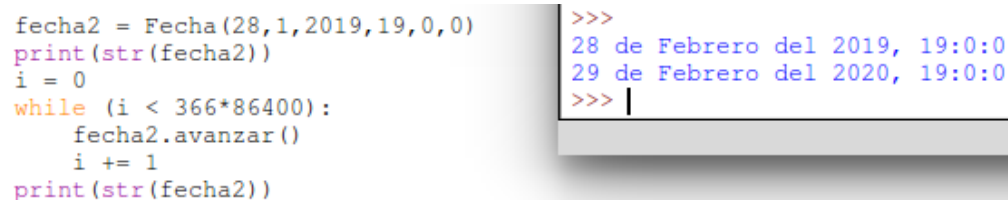
```

>>>
25 de Febrero del 2020, 19:0:0
29 de Febrero del 2020, 19:0:0
>>> |

```

Figura 5: Año bisiesto, comportamiento de febrero

En la figura 6 se muestra otro ejemplo de que el programa detecta si el año es bisiesto o no, vemos que si colocamos la fecha actual el 28 de febrero del 2019, y avanzamos 366 días en el calendario, el programa nos indica que la fecha actual es el 29 de febrero de 2020, ya que es un año bisiesto.



```

fecha2 = Fecha(28,1,2019,19,0,0)
print(str(fecha2))
i = 0
while (i < 366*86400):
    fecha2.avanzar()
    i += 1
print(str(fecha2))

```

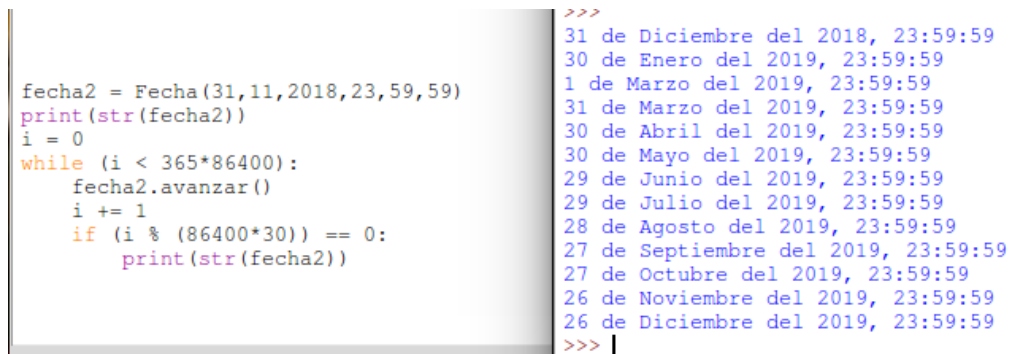
```

>>>
28 de Febrero del 2019, 19:0:0
29 de Febrero del 2020, 19:0:0
>>> |

```

Figura 6: Avanzando un año y un día, desde un año no bisiesto a uno bisiesto

Por último en la figura 7 se procede a imprimir la fecha cada 30 días comenzando desde el 31 de Diciembre de 2018, vemos que el programa va avanzando e imprimiendo cada mes de manera correcta, por lo que se demuestra que puede manejar cualquier lugar del calendario.



```
fecha2 = Fecha(31,11,2018,23,59,59)
print(str(fecha2))
i = 0
while (i < 365*86400):
    fecha2.avanzar()
    i += 1
    if (i % (86400*30)) == 0:
        print(str(fecha2))
```

```
>>>
31 de Diciembre del 2018, 23:59:59
30 de Enero del 2019, 23:59:59
1 de Marzo del 2019, 23:59:59
31 de Marzo del 2019, 23:59:59
30 de Abril del 2019, 23:59:59
30 de Mayo del 2019, 23:59:59
29 de Junio del 2019, 23:59:59
29 de Julio del 2019, 23:59:59
28 de Agosto del 2019, 23:59:59
27 de Septiembre del 2019, 23:59:59
27 de Octubre del 2019, 23:59:59
26 de Noviembre del 2019, 23:59:59
26 de Diciembre del 2019, 23:59:59
>>> |
```

Figura 7: Avanzando sobre todos los meses del año

4. Conclusiones y recomendaciones

En el laboratorio se pudo observar la orientación que tiene Python hacia la programación orientada a objetos, ya que presenta muchas facilidades para su implementación como las funciones de inicialización en las clases, así como otras funciones especiales. También se pudo observar el potencial que presentan las clases a la hora de trabajar con objetos, ya que permite resolver problemas complejos de manera sencilla.

Los conceptos de herencia y polimorfismo se ejemplifican claramente en este laboratorio, ya que todas las clases están conectadas entre sí para ejecutar distintas tareas. Por otro lado, se pudo verificar el potencial que presentan los diagramas UML para ejemplificar códigos complejos de manera simple, por lo que esto es de gran ayuda a la hora de enseñar el código a otras personas, o de crear código a partir de estos diagramas.

Se recomienda tener especial cuidado con el manejo de errores en las clases, ya que es sencillo que el usuario ingrese un dato que no es permitido y el programa deje de funcionar, por lo que se debe tener robustez en el código. También se recomienda que si se debe programar en base a un diagrama UML, se comience por la clase superior, la que no depende de las demás, ya que si se hiciera de otra forma, no tendría mucho sentido o no se podría realizar.

Referencias

- [1] Martelli, A. (2006). *Python in a Nutshell: A Desktop Quick Reference*. O'REILLY. 104-107.
- [2] Bezivin, J. (1998). *The Unified Modeling Language. "UML" '98: Beyond the Notation: First*. Springer. 173.