Función “ponme_en:ruta”

La estructura de nuestros proyectos es la siguiente:

nombre_del_proyecto

- *data*
  - *nombre_archivo.csv*
- *doc*
  - *enunciado.pdf*
- *src*
  - *nombre_módulo.py*
  - *test_nombre_módulo. py*

Estos proyectos suelen estar dentro de otra carpeta “ProyectosPython”:

Por ejemplo:

- Si VSC está abierto en “*ProyectosPython*” y dentro están nuestros proyectos la ruta será:
  “nombre_del_proyecto/data/nombre_archivo.csv”
- *Si VSC está abierto directamente en la carpeta del nombre_del_proyecto la ruta será:*
  “data/nombre_archivo.csv”

Función “ponme_en:ruta”

La siguiente función se cambia automáticamente de carpeta para independizar la lectura de la carpeta donde está abierto VSC

```
def ponme_en_ruta():
    import os
    ruta=os.path.abspath(__file__)  #obtiene la ruta donde está el test
    posición=ruta.index("src")      #busca la cadena “src” dentro de la ruta
    os.chdir(ruta[:posición]+"data\\") #Se cambia a la carpeta data

def lee_________(ruta:str)->list[_____]:
    ponme_en_ruta()  #antes de abrir el archivo
    res=list()
    with open(ruta,'rt',encoding='utf-8') as f:
…
```

En el test solo se escribe el nombre_del_archivo.csv

Conversiones de tipos fecha y hora

Parseo (desde string a fecha, a hora o a fecha y hora)

Método strptime(): permite convertir una string que representa un tipo *fecha (date), hora (time)* o *fecha_hora* (*datetime*).

Sintaxis: la *máscara de formato* depende de cómo estén los datos en la *cadena*.

Ejemplos de conversión de string a date/time/datetime

- datetime.*strptime*(*cadena con la fecha*, *mascara de formato*).*date*()
  - `"1/8/2023","%d/%m/%Y"` → obtiene: *date*(2023,8,1)
  - `"1/08/23","%d/%m/%y"` → obtiene lo mismo: *date*(2023,8,1)
  - `"28-november-2021","%d-%B-%Y"` → obtiene: *date*(2021,11,28)
  - `"28 nov 21","%d %b %y"` →obtiene lo mismo: *date*(2021,11,28)
  - `"28#nov#21","%d#%b#%y"` →obtiene lo mismo: *date*(2021,11,28)
- datetime.*strptime*(*cadena con la hora*, *mascara de formato*).*time*()
  - `"15:30:02","%H:%M:%S"` → obtiene: *time*(15,30,2)
  - `"15#30-2","%H#%M-%S"` → obtiene lo mismo: *time*(15,30,2)
- datetime.*strptime*(*cadena con la fecha y hora*, *mascara de formato*)
  - `"5/9/2023 - 15:31:9","%d/%m/%Y - %H:%M:%S"`→ obtiene: *datetime*(2023,9,1,15,31,9)

Construcción de fechas y horas

Ejemplo de construcción del día y hora actual

```
from datetime import datetime, date,time

ahora=datetime.now()                    → 2023-10-09 19:04:26.719387
hoy=datetime.now().date()               → 2023-10-09
en_este_momento= datetime.now().time()  → 19:04:26.719387
```

Ejemplo de construcción de fechas

```
from datetime import datetime, date

fecha1=date(2023,11,8)                  ← Suele ser la más habitual
fecha2=datetime.strptime("8-november-2023","%d-%B-%Y").date()
fecha3=datetime.strptime("8-11-2023","%d-%m-%Y").date()
```

Ejemplo de construcción de horas

```
from datetime import datetime, time

hora1=time(15,32,45)                    ← Suele ser la más habitual
hora2=datetime.strptime("15:32:45","%H:%M:%S").time()
```

Dando formato de salida a las fechas y horas

Método strftime(): permite convertir fechas (*date*), horas (*time*) y fechas_horas (*datetime*) a formato cadena (*str*).

Sintaxis: la *máscara de formato* depende de cómo se quiera la representación como *cadena*.

variable_fecha.*strftime*(*mascara de formato*)

Ejemplo de conversión de fechas a string

```
from datetime import date, time, datetime
fecha=date(2023,10,9)
hora=time(14,20,30)
fecha_hora=datetime(2023,10,9,14,20,10,5)
```

Por defecto *-directamente print(…)-*

- `fecha` → `2023-10-09`
- `hora` → `14:20:30`
- `fecha_hora` → `2023-10-09 14:20:10.000005`

Con *strftime*

- `fecha.strftime("%d/%m/%y")` → `09/10/23`
- `hora.strftime("%H#%M--%S")` → `14#20—30`
- `fecha_hora.strftime("%d/%m/%Y <-> %H:%M:%S")` → `09/10/2023 <-> 14:20:10`

Operadores de Relación de fecha y hora

Operadores de relación:

Las fechas (*date*), las horas (*time*) y las fechas_horas (*datetime*), tienen predefinidos los *operadores de relación* ==, !=, <, <=, > y >= que permiten establece *un criterio de ordenación* entre ellas, con la lógica ordenación natural de estas:

- Una fecha *f1* será menor que otra *f2* si *f1* representa una fecha “anterior en el tiempo” a *f2*.
- Dos fechas serán iguales si tienen el mismo años, mes y día
- Una hora *h1* será menor que otra *h2* si *h1* representa una hora “más temprana” que *h2*.
- Dos horas serán iguales si tienen la misma hora, minutos, segundos y, en su caso, fracciones de este
- De igual forma ocurre para el tipo *datetime*.

Operadores “aritméticos” sobre fechas y horas

En la librería *datetime* existe un método *timedelta* que permite:

- *sumar o restar* una cantidad de tiempo a una fecha (*date*), a una hora (*time*) o a una fecha_hora (*datetime*).
- Calcular *el intervalo de tiempo* que transcurre entre dos fechas (*date*), horas (*time*) o fechas_horas (*datetime*), restando una fecha/hora/fechahora de otra fecha/hora/fechahora

Sintaxis:

- *timedelta(weeks: float = ..., days: float = ..., hours: float = ..., minutes: float = ..., seconds: float = ..., milliseconds: float = ..., microseconds: float …)*

```
from datetime import date, time, datetime, timedelta

ahora=datetime.now()                    → 2023-10-09 19:27:48.676132
ayer=ahora-timedelta(days=1)            → 2023-10-08 19:27:48.676132
mañana=ahora+timedelta(days=1)          → 2023-10-10 19:27:48.676132
otra=datetime(2023,10,11,14,20,10,5)    → 2023-10-11 14:20:10.000005
tiempo=ahora-otra                       → -2 days, 5:07:38.676127
```

Dando formato a las salidas

Hemos aprendido desde el primer día que la función *print()* tiene un número indeterminado de parámetros, de tipo cadena o numérico, separados por coma "," que permite visualizarlos por la consola.

El método *format* da un paso más para la presentación de los datos con unas características determinadas según la siguiente sintaxis:

print ("texto …{:expresión de formato} …. ".*format*(variable a visualizar))

Sintaxis para cadena: {:*ns*} donde n es el número de espacios mínimos para visualizar la cadena

Ejemplo:

```
nombre="Ana"
print ("Soy {:s},  Feliz Navidad!".format(nombre))  → Soy Ana, Feliz Navidad!
print ("Soy {:10s}, Feliz Navidad!".format(nombre)) → Soy Ana       , Feliz Navidad!
print ("Soy {:>10s}, Feliz Navidad!".format(nombre))→ Soy        Ana, Feliz Navidad!
```

Dando formato a las salidas (II)

Sintaxis para enteros: {*:nd*} donde *n* es el número de espacios mínimos para visualizar el número (en su caso también el signo)

Ejemplo:

```
año = 2023
print ("Feliz {:d}!".format(año))
print ("Feliz {:10d}!".format(año))
print ("Feliz {:<10d}!".format(año))
print ("Feliz {:010d}!".format(año))
```

```
→ Feliz 2023!
→ Feliz       2023!
→ Feliz 2023      !
→ Feliz 0000002023!
```

Sintaxis para reales: {*:n.df*} donde *n* es el número de espacios mínimos para visualizar la perte entera, la coma y los decimales y *d* es el número de decimales

Ejemplo:

```
saldo = 234.678
print ("Mi saldo es {:10.2f} euros".format(saldo))
print ("Mi saldo es {:010.4f} euros".format(saldo))
```

```
→ Mi saldo es     234.68 euros
→ Mi saldo es 00234.6780 euros
```

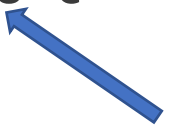
Dando formato a las salidas

Se pueden combinar diversos valores y formatos en la misma expresión y en este caso los especificadores de formato se asignan de izquierda a derecha con las variables del método format

Ejemplo:

```
nombre="Ana"
año=2023
saldo = 234.678
print("Soy {:>5s} mi saldo para {:d} es de {:10.2f} €".format(nombre,año,saldo))
```

```
Soy Ana mi saldo para 2023 es de 234.68 €
```



Redondea los decimales

Ejercicio:

Modifique el proyecto *T09_Datos_Personales*

- En la carpeta "*data*" copie el fichero "*datos_personales4.csv*". Este fichero tiene los siguientes campos por cada línea :

dni;nombre;apellidos;edad;estatura;peso;localidad;provincia;esmujer;hobbies;fecha-hora de entrada

Todas string salvo: edad de tipo int, estatura y peso de tipo float, esmujer de tipo bool, hobbies de tipo list[str] y fecha-hora de entrada que responde al patrón "día/mes/año#hora:minutos:segundos"

- En el módulo *datos_personales.py*:
  - Cree una nueva namedtuple *Persona4* a partir de una copia de *Persona3*. *Tenga en cuenta que la "fecha-hora de entrada" se guardarán por separado en dos campos: fecha y hora*
  - Añada una función: *lee_datos_personales4* que, recibiendo el nombre de un fichero, devuelva una lista los registros leídos.
- En el módulo *test_datos_personales.py*
  - Añada un test que permita probar la función *lee_datos_personales4* visualizando:
    a) El número de registros leídos
    b) El tercer registro (sin contar el registro de cabecera)
    c) Los tres primeros registros leídos
    d) Los tres últimos registros leídos