Bloque 3

Fecha y horas
Parámetros formales y reales, por defecto y con nombre.
Formatos de salida

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2024/2025

Tipos fecha y hora

Python implementa otros *tres* tipos predefinidos (built-in) para trabajar con *fechas* y *hora*.

- *date:* permite definir una fecha y, por tanto, trabajar con *día*, *mes* y *año*.
- *time:* permite definir una hora y, por tanto, trabajar con *horas, minutos* y *segundos*. También permite fracciones de tiempo más pequeñas.
- *datetime:* permite definir conjuntamente una fecha y una hora

Para su uso es necesario importar dichos tipos → *from datetime import date, time, datetime*

Ejemplo de construcción del día y hora actual

```
ahora=datetime.now()
hoy=datetime.now().date()
la_hora_de_este_instante= datetime.now().time()
```

Ejemplo de construcción de fechas

```
fecha=date(2024,11,8)
```

Ejemplo de construcción de horas

```
hora=time(15,32,45)
```

Conversiones de tipos fecha y hora

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

Método strptime(): permite convertir una cadena (str) 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 str a fecha, a hora o a fecha y hora

- datetime.*strptime*(*cadena con la fecha*, *mascara de formato*).*date*()
  - `"1/8/2024","%d/%m/%Y"` → obtiene: *date*(2024,8,1)
  - `"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)

Que no se olvide

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 str

```
fecha=date(2024,10,9)
hora=time(14,20,30)
fecha_hora=datetime(2024,10,9,14,20,10,5)
```

Por defecto -*directamente print(…)*-

- `fecha` → `2024-10-09`
- `hora` → `14:20:30`
- `fecha_hora` → `2024-10-09 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")` → `09/10/2024 <-> 14:20:10`

Propiedades de fechas y horas

Se puede acceder a las *propiedades de los objetos fechas y horas* mediante sus respectivos nombres (*observar que para obtener las propiedades no se añaden paréntesis*):

```
fecha=date(2024,10,9)
hora=time(14,20,30,10)
```

```
fecha.year        → 2024
fecha.month       → 10
fecha.day         → 9
hora.hour         → 14
hora.minute       → 20
hora.second       → 30
hora.microsecond  → 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 año, 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:

- *suma o resta* una cantidad de tiempo a una fecha (*date*), una hora (*time*) o 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()                          → 2024-10-09 19:27:48.676132
ayer=ahora-timedelta(days=1)                  → 2024-10-08 19:27:48.676132
mañana=ahora+timedelta(days=1)                → 2024-10-10 19:27:48.676132
otra=datetime(2024,10,11,14,20,10,5)          → 2024-10-11 14:20:10.000005
tiempo=ahora-otra                             → -2 days, 5:07:38.676127
```

Prueba esto:

```
nací=date(a,m,d)  (donde a, m y d son los de tu fecha de nacimiento)
hoy=datetime.now().date()
print("Como nací",nací,"y hoy es",hoy,"he vivido",(hoy-nací))
```

Parámetros de funciones: *formales* y *reales*

Hemos aprendido que las funciones utilizan los parámetros para darles versatilidad y poderlas usar con distintos valores. Un caso común ha sido invocar desde un módulo "test" a una misma función para hacer más de una prueba, por ejemplo, hemos puesto en una prueba como marca de vacunas "JANSSEN" y en otra prueba "MODERNA".

Formalmente los parámetros se denominan de forma distinta según donde se escriben:

- *Parámetros formales: son los que se escriben entre paréntesis en la cabecera de la definición de la función.*

  def nombre_función (nombre_parámetro 1: tipo, nombre_parámetro 2: tipo, …)->tipo:

  Por ejemplo:

  ```
  def filtra_país(poblaciones:List[Población], código:str)->List[Población]:
  ```

- *Parámetros reales: son los que se escriben entre paréntesis cuando se invoca a la función (pueden ser variables o directamente literales).*

  nombre_función (variable1/literal1, variable2/literal2, …)

  Por ejemplo:

  ```
  filtra_país(lista_pob, 'ESP')
  ```

Parámetros por defectos

Podemos hacer que los parámetros formales tomen un valor por defecto, de forma que, si se omite su correspondiente parámetro real, tome dicho valor. Estos últimos (los que se pueden omitir) les podemos denominar parámetros formales opcionales y los demás parámetros formales obligatorios.

Los parámetros opcionales que pueden tomar valor por defecto deben estar escritos en las últimas posiciones. Un parámetro se convierte en opcional añadiendo detrás de su nombre y tipo el signo "=" junto con el valor por defecto.

*def nombre_función(obligatorio1:*tipo,*obligatorio2:*tipo,*…opcional1:*tipo=*valor1*,*opcional2:*tipo =*valor2*,..)
->tipo*:*

Por ejemplo

- *en la definición:*

```
def filtra_país_y_habitantes(poblaciones:List[Población],código:str,
habitantes:int=50000000)->List[Población]:
```

- *en la invocación:*

```
filtra_país_y_habitantes(lista_pob,'ESP',75000000)
filtra_país_y_habitantes(lista_pob,'ESP')
```

(En este último caso el parámetro formal habitantes tomará el valor 50000000)

Invocación de parámetros por el nombre

Sabemos que cuando se invoca a una función los parámetros reales se escriben en el mismo orden en los esperan los parámetros formales. No obstante, se puede alterar el orden si se antepone a los parámetros reales el nombre del parámetro formal seguido del signo "="

Por ejemplo

Normalmente hubiésemos puesto en la invocación los *parámetros reales* en el orden en que aparecen como *parámetros formales*

```
filtra_país_y_habitantes(lista_pob,'ESP',75000000)
```

Pero podemos invocar en otro orden usando los nombres de los parámetros formales:

```
filtra_país_y_habitantes(código='ESP',habitantes=75000000,poblaciones=lista_pob)
```

***Ojo**! Esto sólo se puede hacer si se conoce como se llama cada parámetro formal*

Estrategia de filtrado con parámetros por defecto

Suele ser habitual realizar ejercicios en los que el filtro no solo depende del valor que toman los parámetros, sino que, si toman un valor por defecto, entonces el filtro no tiene efecto.

La estrategia consiste en preguntar primero por si el parámetro correspondiente toma el valor por defecto o que cumpla la condición que se busca en el filtro.

Por ejemplo

Realizar la función *vacunados_entre_fechas* que reciba como parámetros una lista de tuplas de tipo Vacuna y dos fechas con valores por defecto *None*, y devuelva el número de personas vacunadas entre dichas fechas. Si alguna de las fechas, toman el valor por defecto no se debe filtrar por ella.

```
def vacunados_entre_fechas (vacunas:List[Vacuna],f1:date=None,f2:date=None)->int:
  res=0
  for v in vacunas:
    if (f1==None or f1<=v.fecha_admón) and \
       (f2==None or v.fecha_admón<=f2):
         res+=1
  return res
```

Estrategia de filtrado con parámetros por defecto (un test)

```
def test_vacunados_entre_fechas(datos:List[Vacuna])->None:
  print("\ntest_vacunados_entre_fechas")
  f1=None
  f2=None
  print("El número de vacunados entre ",f1,"y",f2,"es:",vacunados_entre_fechas(datos,f1,f2))
  f1=date(2021,7,1)
  f2=None
  print("El número de vacunados entre ",f1,"y",f2,"es:",vacunados_entre_fechas(datos,f1,f2))
  f1=None
  f2=date(2021,9,30)
  print("El número de vacunados entre ",f1,"y",f2,"es:",vacunados_entre_fechas(datos,f1,f2))
  f1=date(2021,7,1)
  f2=date(2021,9,30)
  print("El número de vacunados entre ",f1,"y",f2,"es:",vacunados_entre_fechas(datos,f1,f2))
```

```
test_vacunados_entre_fechas
El número de vacunados entre  None y None es: 1000
El número de vacunados entre  2021-07-01 y None es: 493
El número de vacunados entre  None y 2021-09-30 es: 752
El número de vacunados entre  2021-07-01 y 2021-09-30 es: 245
```

Formateando las salidas (F-String)

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.

No obstante, se dispone de un formato más cómodo, que consiste en escribir una cadena precedida de una f con la información que se quiere visualizar y, en su caso, encerando entre { } las variables o expresiones cuyo valor también se pretende visualizar

```
print (f"texto …{variable/expresión-1} …{variable/expresión-2}…")
```

Ejemplo:

Antes hemos escrito (con a, m y d el año, mes y día de nuestra fecha de nacimiento) :

```
nací=date(a,m,d)
hoy=datetime.now().date()
print("Como nací",nací,"y hoy es",hoy,"he vivido",(hoy-nací).days)
```

Podemos escribir la última línea como:

```
print(f'Como nací el {nací} y hoy es {hoy} he vivido {(hoy-nací).days}')
```

Que visualiza exactamente lo mismo. ¡*Vamos a Notebooks, y lo probamos!*

Formateando las salidas (format)

Los dos formatos vistos hasta ahora

- `print("texto",variable/expresión,"texto", variable/expresión,…)`
- `print (f"texto …{variable/expresión-1}…{variable/expresión-2}…")`

No permiten, por ejemplo, *visualizar con un número de decimales, si separador de miles y otros*

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

```
print ("texto …{:formato} …{:formato} . ".format(var/exp1, var/exp2,…))
```

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

Ejemplo:

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

```
Soy Ana, Feliz Navidad!
Soy Ana       , Feliz Navidad!
Soy        Ana, Feliz Navidad!
```

10 espacios

Formateando las salidas (format)

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

Ejemplo:

```
año = 2025
print ("Feliz {:d}!".format(año))
print ("Feliz {:10d}!".format(año))
print ("Feliz {:<10d}!".format(año))
print ("Feliz {:010d}!".format(año))
```

```
Feliz 2025!
Feliz       2025!
Feliz 2025      !
Feliz 0000002025!
```

10 espacios

10 espacios

Sintaxis para reales: {:n.df} donde *n* es el número de espacios mínimos para visualizar la parte 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
```

10 espacios

Formateando las salidas (format)

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=2025
saldo = 234.678
print("Soy {:>5s} mi saldo para {:d} es de {:10.2f}€".format(nombre,año,saldo))
```

```
Soy   Ana mi saldo para 2025 es de     234.68€
```

5 espacios

Utiliza 4 espacios

10 espacios

Redondea los decimales