



Corso di Apprendistato Python

Mod. Python
Docente:
Tonino Petrulli



Introduzione al modulo os

In questa sezione, imparerete a conoscere un modulo chiamato *os*, che vi permette di **interagire con il sistema operativo utilizzando Python**.

Fornisce funzioni disponibili sui sistemi Unix e/o Windows. Se avete familiarità con la console dei comandi, vedrete che alcune funzioni danno gli stessi risultati dei comandi disponibili sui sistemi operativi.

Un buon esempio è la funzione `mkdir`, che consente di creare una directory proprio come il comando *mkdir* di Unix e Windows. Se non conoscete questo comando, non preoccupatevi.

Presto avrete l'opportunità di imparare le funzioni del modulo *os*, per eseguire operazioni su file e directory insieme ai comandi corrispondenti.

Oltre alle operazioni su file e directory, il modulo *os* consente di:

- ottenere informazioni sul sistema operativo;
- gestire i processi;
- operano sui flussi di I/O usando i descrittori di file.

Tra poco vedrete come ottenere informazioni di base sul vostro sistema operativo, anche se la gestione dei processi e il lavoro con i descrittori di file non saranno trattati in questa sede, perché si tratta di argomenti più avanzati che richiedono la conoscenza dei meccanismi del sistema operativo.



Ottenere informazioni sul sistema operativo

Prima di creare la prima struttura di directory, vedremo come ottenere informazioni sul sistema operativo corrente. È molto facile perché il modulo *os* fornisce una funzione chiamata *uname*, che restituisce un oggetto contenente i seguenti attributi:

- **systemname** - memorizza il nome del sistema operativo;
- **nodename** - memorizza il nome della macchina sulla rete;
- **release** - memorizza la release del sistema operativo;
- **version** - memorizza la versione del sistema operativo;
- **machine** - memorizza l'identificatore hardware, ad esempio, x86_64.

Vediamo come funziona in pratica:

```
import os  
print(os.uname())
```

Risultato

```
posix.uname_result(sysname='Linux', nodename='192d19f04766', release='4.4.0-164-generic', version='#192-Ubuntu SMP Fri Sep 13 12:02:50 UTC 2019', machine='x86_64')
```

Come si può vedere, la funzione *uname* restituisce un oggetto contenente informazioni sul sistema operativo. Il codice sopra riportato è stato lanciato su Ubuntu 16.04.6 LTS, quindi non stupitevi se otterrete un risultato diverso, perché dipende dal vostro sistema operativo.

Purtroppo, la funzione *uname* funziona solo su alcuni sistemi Unix. Se si utilizza Windows, è possibile utilizzare la funzione *uname* del modulo *platform*, che restituisce un risultato simile.

Il modulo *os* consente di distinguere rapidamente il sistema operativo utilizzando l'attributo *name*, che supporta uno dei seguenti nomi:

- **posix** - questo nome si usa se si utilizza Unix;
- **nt** - questo nome si ottiene se si utilizza Windows;
- **java** - si ottiene questo nome se il codice è scritto in Jython.

Per Ubuntu 16.04.6 LTS, l'attributo *name* restituisce il nome *posix*:

```
import os  
print(os.name)
```

Result:

posix

NOTA: Sui sistemi Unix, esiste un comando chiamato *uname* che restituisce le stesse informazioni (se eseguito con l'opzione -a) della funzione *uname*.

Creare directory in Python

Il modulo `os` fornisce una funzione chiamata `mkdir` che, come il comando `mkdir` di Unix e Windows, consente di creare una directory. La funzione `mkdir` richiede un percorso che può essere relativo o assoluto. Ricordiamo come si presentano in pratica entrambi i percorsi:

- **`my_first_directory`** - è un percorso relativo che crea la directory `my_first_directory` nella directory di lavoro corrente;
- **`./my_first_directory`** - è un percorso relativo che punta esplicitamente alla directory di lavoro corrente. Ha lo stesso effetto del percorso precedente;
- **`../mia_prima_directory`** - è un percorso relativo che crea la directory `mia_prima_directory` nella directory madre della directory di lavoro corrente;
- **`/python/my_first_directory`** - è il percorso assoluto che creerà la cartella `my_first_directory`, che a sua volta si trova nella cartella `python` della directory principale.

Osservare il codice

```
import os
os.mkdir("my_first_directory")
print(os.listdir())
```

Mostra un esempio di come creare la cartella *my_first_directory* utilizzando un percorso relativo. Questa è la variante più semplice del percorso relativo, che consiste nel passare solo il nome della directory.

Se si testa il codice qui, verrà visualizzata la cartella ['my_first_directory'] appena creata (e l'intero contenuto del catalogo di lavoro corrente).

La funzione *mkdir* crea una directory nel percorso specificato. Si noti che l'esecuzione del programma due volte solleverà un *FileExistsError*.

Ciò significa che non è possibile creare una directory se esiste già. Oltre all'argomento percorso, la funzione *mkdir* può accettare l'argomento *modalità*, che specifica i permessi della directory. Tuttavia, in alcuni sistemi, l'argomento *mode* viene ignorato.

Per modificare i permessi delle directory, si consiglia la funzione *chmod*, che funziona in modo simile al comando *chmod* dei sistemi Unix. Per ulteriori informazioni, consultare la documentazione.

Nell'esempio precedente, viene utilizzata un'altra funzione fornita dal modulo *os*, denominata *listdir*. La funzione *listdir* restituisce un elenco contenente i nomi dei file e delle directory presenti nel percorso passato come argomento.

Se non viene passato alcun argomento, verrà utilizzata la directory di lavoro corrente (come nell'esempio precedente). È importante che il risultato della funzione *listdir* ometta le voci '.' e '..', che vengono visualizzate, ad esempio, quando si usa il comando *ls -a* sui sistemi Unix.

NOTA: Sia in Windows che in Unix, esiste un comando chiamato *mkdir*, che richiede un percorso di directory.

L'equivalente del codice sopra riportato che crea la directory *my_first_directory* è il comando *mkdir my_first_directory*.

Creazione ricorsiva di directory

La funzione `mkdir` è molto utile, ma cosa succede se si ha bisogno di creare un'altra directory nella directory appena creata. Naturalmente si può andare nella directory creata e creare un'altra directory al suo interno, ma fortunatamente il modulo `os` fornisce una funzione chiamata `makedirs`, che facilita questo compito.

La funzione *makedirs* consente la creazione ricorsiva di directory, il che significa che tutte le directory presenti nel percorso saranno create. Diamo un'occhiata al codice

```
import os
os.makedirs("my_first_directory/my_second_directory")
os.chdir("my_first_directory")
print(os.listdir())
```

Vediamo come funziona in pratica.

Il codice dovrebbe produrre il seguente risultato:

```
['my_second_directory']
```

Il codice crea due directory. La prima viene creata nella directory di lavoro corrente, mentre la seconda nella directory *my_first_directory*.

Non è necessario andare nella directory *my_first_directory* per creare la directory *my_second_directory*, perché la funzione *makedirs* lo fa per voi. Nell'esempio precedente, si va alla directory *my_first_directory* per mostrare che il comando *makedirs* crea la sottodirectory *my_second_directory*.

Per spostarsi tra le directory, si può usare una funzione chiamata *chdir*, che cambia la directory di lavoro corrente nel percorso specificato. Come argomento, accetta qualsiasi percorso relativo o assoluto. Nel nostro esempio, gli passiamo il nome della prima directory.

NOTA: L'equivalente della funzione *makedirs* nei sistemi Unix è il comando *mkdir* con il flag *-p*, mentre in Windows è sufficiente il comando *mkdir* con il percorso:

- Unix-like systems:

```
mkdir -p my_first_directory/my_second_directory
```

- Windows:

```
mkdir my_first_directory/my_second_directory
```


Dove mi trovo ora?

Sapete già come creare le directory e come spostarvi tra di esse. A volte, quando si naviga in una struttura di directory molto grande, si può non sapere in quale directory si sta lavorando.



Come avrete capito, il modulo `os` fornisce una funzione che restituisce informazioni sulla directory di lavoro corrente. Si chiama `getcwd`. Guardate il codice per vedere come usarla in pratica.

```
import os
```

```
os.makedirs("my_first_directory/my_second_directory")
os.chdir("my_first_directory")
print(os.getcwd())
os.chdir("my_second_directory")
print(os.getcwd())
```

Risultato:

```
.../my_first_directory
.../my_first_directory/my_second_directory
```

Nell'esempio, si crea la directory *my_first_directory* e la directory *my_second_directory* al suo interno. Nel passo successivo, si cambia la directory di lavoro corrente in *my_first_directory* e si visualizza la directory di lavoro corrente (prima riga del risultato).

Successivamente, passiamo alla directory *my_second_directory* e visualizziamo nuovamente la directory di lavoro corrente (seconda riga del risultato). Come si può notare, la funzione *getcwd* restituisce il percorso assoluto delle directory.

NOTA: Nei sistemi Unix-like, l'equivalente della funzione *getcwd* è il comando *pwd*, che stampa il nome della directory di lavoro corrente

Eliminare le directory in Python

Il modulo `os` consente anche di eliminare le directory. Si può scegliere se cancellare una singola directory o una directory con le sue sottodirectory. Per cancellare una singola directory, si può usare una funzione chiamata `rmdir`, che prende come argomento il percorso. Osservare il codice

```
import os

os.mkdir("my_first_directory")
print(os.listdir())
os.rmdir("my_first_directory")
print(os.listdir())
```

L'esempio precedente è molto semplice. Innanzitutto, viene creata la directory *my_first_directory* e poi viene rimossa con la funzione *rmdir*. La funzione *listdir* viene utilizzata come prova che la directory è stata rimossa con successo. In questo caso, restituisce un elenco vuoto. Quando si elimina una directory, bisogna assicurarsi che esista e che sia vuota, altrimenti verrà sollevata un'eccezione.

Per rimuovere una directory e le sue sottodirectory, si può usare la funzione `removedirs`, che richiede di specificare un percorso contenente tutte le directory da rimuovere:

```
import os
os.makedirs("my_first_directory/my_second_directory")
os.removedirs("my_first_directory/my_second_directory")
print(os.listdir())
```

Come per la funzione *rmdir*, se una delle directory non esiste o non è vuota, verrà sollevata un'eccezione.

NOTA: Sia in Windows che in Unix esiste un comando chiamato *rmdir* che, proprio come la funzione *rmdir*, rimuove le directory. Inoltre, entrambi i sistemi dispongono di comandi per eliminare una directory e il suo contenuto. In Unix, si tratta del comando *rm* con il flag *-r*.

La funzione `system()`

Tutte le funzioni presentate in questa parte del corso possono essere sostituite da una funzione chiamata *system*, che esegue un comando passato come stringa.

La funzione `system` è disponibile sia in Windows che in Unix. A seconda del sistema, restituisce un risultato diverso.

In Windows, restituisce il valore restituito dalla shell dopo l'esecuzione del comando dato, mentre in Unix restituisce lo stato di uscita del processo.

Diamo un'occhiata al codice

```
import os
returned_value = os.system("mkdir my_first_directory")
print(returned_value)
```

L'esempio precedente funziona sia in Windows che in Unix. Nel nostro caso, riceviamo lo stato di uscita 0, che indica il successo nei sistemi Unix.

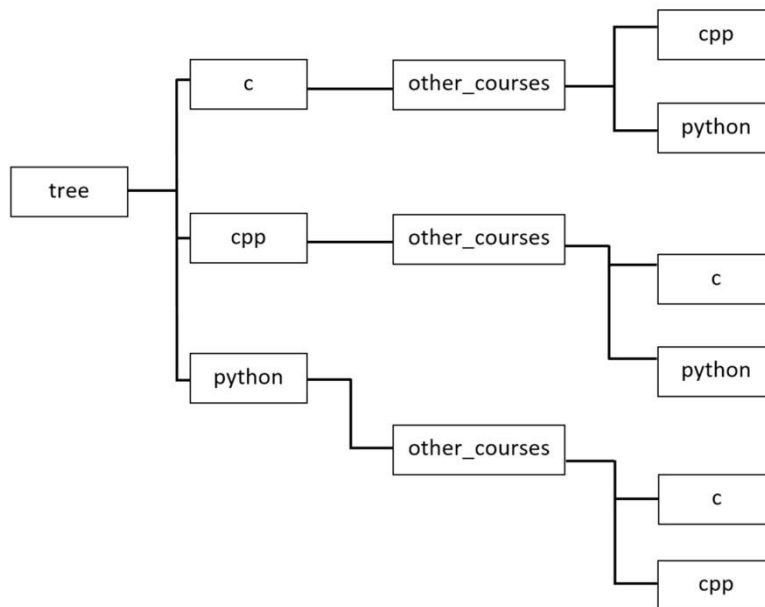
Ciò significa che è stata creata la directory *my_first_directory*. Come parte dell'esercizio, provate a elencare il contenuto della directory in cui è stata creata la directory *my_first_directory*.

LAB-46

Scenario

È ovvio che i sistemi operativi consentono di cercare file e directory. Durante lo studio di questa parte del corso, avete imparato a conoscere le funzioni del modulo `os`, che contiene tutto ciò che serve per scrivere un programma che cerchi le directory in una determinata posizione.

Per facilitare il vostro compito, abbiamo preparato per voi una struttura di directory di prova:



Il programma deve soddisfare i seguenti requisiti:

- Scrivete una funzione o un metodo chiamato *find* che accetta due argomenti chiamati *path* e *dir*. L'argomento *path* deve accettare un percorso relativo o assoluto a una directory da cui iniziare la ricerca, mentre l'argomento *dir* deve essere il nome di una directory che si desidera trovare nel percorso dato. Il programma deve visualizzare il percorso assoluto se trova una directory con il nome dato.
- La ricerca delle directory deve essere effettuata in modo ricorsivo. Ciò significa che la ricerca deve includere anche tutte le sottodirectory del percorso indicato.

Example input:

path="./tree", dir="python"

Example output:

.../tree/python
.../tree/cpp/other_courses/python
.../tree/c/other_courses/python

Punti di forza

1. La funzione `uname` restituisce un oggetto che contiene informazioni sul sistema operativo corrente. L'oggetto ha i seguenti attributi:

- *systemname* (memorizza il nome del sistema operativo)
- *nodename* (memorizza il nome della macchina sulla rete)
- *release* (memorizza la release del sistema operativo)
- *version* (memorizza la versione del sistema operativo)
- *macchina* (memorizza l'identificatore hardware, ad esempio `x86_64`).

2. L'attributo *name* disponibile nel modulo `os` consente di distinguere il sistema operativo. Restituisce uno dei tre valori seguenti:

- *posix* (questo nome si ottiene se si usa Unix)
- *nt* (questo nome si ottiene se si usa Windows)
- *java* (si ottiene questo nome se il codice è scritto in qualcosa come Jython)

3. La funzione `mkdir` crea una directory nel percorso passato come argomento. Il percorso può essere sia relativo che assoluto, ad esempio

```
import os
```

```
os.mkdir("hello") # the relative path
```

```
os.mkdir("/home/python/hello") # the absolute path
```


Nota: se la directory esiste, verrà lanciata un'eccezione `FileExistsError`. Oltre alla funzione `mkdir`, il modulo `os` fornisce la funzione `makedirs`, che consente di creare ricorsivamente tutte le directory in un percorso.

4. Il risultato della funzione `listdir()` è un elenco contenente i nomi dei file e delle directory presenti nel percorso passato come argomento.

È importante ricordare che la funzione `listdir` omette le voci `'.'` e `'..'`, che vengono visualizzate, ad esempio, quando si utilizza il comando `ls -a` sui sistemi Unix. Se il percorso non viene passato, il risultato verrà restituito per la directory di lavoro corrente.

5. Per spostarsi tra le directory, si può usare una funzione chiamata `chdir()`, che cambia la directory di lavoro corrente nel percorso specificato. Come argomento, accetta qualsiasi percorso relativo o assoluto.

Se si vuole sapere qual è la directory di lavoro corrente, si può usare la funzione `getcwd()`, che restituisce il percorso della directory.

6. Per rimuovere una directory si può usare la funzione `rmdir()`, ma per rimuovere una directory e le sue sottodirectory si deve usare la funzione `removedirs()`.

7. Sia su Unix che su Windows, è possibile utilizzare la funzione di sistema, che esegue un comando passato come stringa, ad es:

```
returned_value = os.system("mkdir hello")
```

La funzione `system` su Windows restituisce il valore restituito da shell dopo l'esecuzione del comando dato, mentre su Unix restituisce lo stato di uscita del processo

Esercizio 1

Qual è l'output del seguente snippet se lo si esegue su Unix?

```
import os  
print(os.name)
```

Soluzione

posix

Esercizio 2

Qual è l'output del seguente snippet?

```
import os  
os.mkdir("hello")  
print(os.listdir())
```

Soluzione

['hello']

Introduzione al modulo `datetime`

In questa sezione, imparerete a conoscere un modulo Python chiamato *datetime*.

Come si può intuire, fornisce **classi per lavorare con la data e l'ora**. Se pensate che non sia necessario approfondire questo argomento, parliamo di esempi di utilizzo di data e ora nella programmazione.

La data e l'ora hanno innumerevoli usi e probabilmente è difficile trovare un'applicazione di produzione che non li utilizzi. Ecco alcuni esempi:

- **registrazione degli eventi** - grazie alla conoscenza della data e dell'ora, siamo in grado di determinare quando esattamente si verifica un errore critico nella nostra applicazione. Quando si creano i log, è possibile specificare il formato della data e dell'ora;
- **tracciare le modifiche nel database** - a volte è necessario memorizzare informazioni su quando un record è stato creato o modificato. Il modulo *datetime* è perfetto per questo caso;
- **Convalida dei dati** - imparerete presto a leggere la data e l'ora corrente in Python. Conoscendo la data e l'ora corrente, siamo in grado di convalidare vari tipi di dati, ad esempio se un buono sconto inserito da un utente nella nostra applicazione è ancora valido;
- **memorizzazione di informazioni importanti** - riuscite a immaginare i bonifici bancari senza memorizzare le informazioni su quando sono stati effettuati? La data e l'ora di certe azioni devono essere conservate e noi dobbiamo occuparcene.

La data e l'ora sono utilizzate in quasi tutti i settori della nostra vita, quindi è importante familiarizzare con il modulo *datetime* di Python. Siete pronti per una nuova dose di conoscenza?



Ottenere la data locale corrente e creare oggetti data

Una delle classi fornite dal modulo datetime è una classe chiamata date. Gli oggetti di questa classe rappresentano una data composta da anno, mese e giorno. Guardate il codice

```
from datetime import date
today = date.today()
print("Today:", today)
print("Year:", today.year)
print("Month:", today.month)
print("Day:", today.day)
```

Eseguite il codice per vedere cosa succede.

Il metodo today restituisce un oggetto date che rappresenta la data locale corrente. Si noti che l'oggetto date ha tre attributi: *anno*, *mese* e *giorno*.

Fare attenzione, perché questi attributi sono di sola lettura. Per creare un oggetto data, occorre passare i parametri *anno*, *mese* e *giorno* come segue:

```
from datetime import date
my_date = date(2019, 11, 4)
print(my_date)
```

Eseguite l'esempio per vedere cosa succede.

Quando si crea un oggetto *data*, tenere presente le seguenti restrizioni:

Parametro Limitazioni

`year` Il parametro *anno* deve essere maggiore o uguale a 1 (costante MINYEAR) e minore o uguale a 9999 (costante MAXYEAR).

`month` Il parametro del *mese* deve essere maggiore o uguale a 1 e minore o uguale a 12.

`day` Il parametro *giorno* deve essere maggiore o uguale a 1 e minore o uguale all'ultimo giorno del mese e dell'anno indicati.

Nota: più avanti in questo corso si apprenderà come modificare il formato predefinito della data.

Creare un oggetto data da un timestamp

La classe `date` ci dà la possibilità di creare un oggetto *data* da un *timestamp*.

In Unix, il timestamp esprime il numero di secondi dal 1° gennaio 1970, 00:00:00 (UTC). Questa data è chiamata **Unix epoch**, perché è il momento in cui è iniziato il conteggio del tempo nei sistemi Unix.

Il timestamp è in realtà la differenza tra una data particolare (inclusa l'ora) e il 1° gennaio 1970, 00:00:00 (UTC), espressa in secondi.

Per creare un oggetto `date` da un timestamp, occorre passare un timestamp Unix al metodo

`fromtimestamp` method.

A questo scopo, possiamo utilizzare il modulo `time`, che fornisce funzioni relative al tempo. Una di queste è una funzione chiamata `time()` che restituisce il numero di secondi dal 1° gennaio 1970 al momento attuale sotto forma di numero float. Guardate l'esempio

```
from datetime import date
import time
timestamp = time.time()
print("Timestamp:", timestamp)
d = date.fromtimestamp(timestamp)
print("Date:", d)
```

Eseguire il codice per vedere l'output.

Se si esegue il codice di esempio più volte, si potrà vedere come il timestamp si incrementa da solo. Vale la pena di aggiungere che il risultato della funzione `time` dipende dalla piattaforma, perché **nei sistemi Unix e Windows i secondi bisestili non vengono conteggiati**.

Nota: in questa parte del corso si parlerà anche del modulo *temporale*.

Creare un oggetto data utilizzando il formato ISO

Il modulo datetime fornisce diversi metodi per creare un oggetto data. Uno di questi è il metodo fromisoformat, che accetta una data nel formato **YYYY-MM-DD**, conforme allo standard ISO 8601.

Lo standard ISO 8601 definisce il modo in cui vengono rappresentate la data e l'ora. Viene utilizzato spesso, quindi vale la pena di familiarizzare con esso. Date un'occhiata all'immagine che descrive i valori richiesti dal formato:



YYYY - year (e.g., **1990**)

MM - month (e.g., **11**)

DD - day (e.g., **18**)

Ora guardate il codice

```
from datetime import date
```

```
d = date.fromisoformat('2019-11-04')  
print(d)
```

Nel nostro esempio, YYYY è 2019, MM è 11 (novembre) e DD è 04 (quarto giorno di novembre).

Quando si sostituisce la data, assicurarsi di aggiungere 0 prima di un mese o di un giorno espresso da un numero inferiore a 10.

Nota: Il metodo fromisoformat è disponibile in Python dalla versione 3.7.

Il metodo `replace()`

A volte può essere necessario sostituire l'anno, il mese o il giorno con un valore diverso. Non è possibile farlo con gli attributi `anno`, `mese` e `giorno`, perché sono di sola lettura. In questo caso, si può usare il metodo `replace`.

Eseguire il codice

```
from datetime import date
d = date(1991, 2, 5)
print(d)
d = d.replace(year=1992, month=1, day=16)
print(d)
```

Risultato:

```
1991-02-05
1992-01-16
```

I parametri *anno*, *mese* e *giorno* sono facoltativi. È possibile passare al metodo `replace` solo un parametro, ad esempio l'*anno*, oppure tutti e tre, come nell'esempio.

Il metodo `replace` restituisce un oggetto *data* modificato, quindi bisogna ricordarsi di assegnarlo a qualche variabile.

Che giorno della settimana è?

Uno dei metodi più utili che facilita il lavoro con le date è il metodo chiamato giorno della settimana. Esso restituisce il giorno della settimana come numero intero, dove 0 è lunedì e 6 è domenica. Eseguire il codice

```
from datetime import date  
d = date(2019, 11, 4)  
print(d.weekday())
```

Risultato:

0



La classe `date` ha un metodo simile, chiamato `isoweekday`, che restituisce anche il giorno della settimana come numero intero, ma 1 è lunedì e 7 è domenica:

```
from datetime import date
```

```
d = date(2019, 11, 4)  
print(d.isoweekday())
```

Result:

1

Come si può vedere, per la stessa data si ottiene un numero intero diverso, ma che esprime lo stesso giorno della settimana. L'intero restituito dal metodo `isodayweek` segue le specifiche ISO 85601.

Creazione di oggetti time

Si sa già come presentare una data utilizzando l'oggetto `date`. Il modulo `datetime` ha anche una classe che consente di presentare il tempo. Riuscite a indovinare il suo nome? Sì, si chiama `time`:

`time(hour, minute, second, microsecond, tzinfo, fold)`

Il costruttore della classe `Time` accetta i seguenti parametri opzionali:

Parameter	Restrictions
<code>hour</code>	The <i>hour</i> parameter must be greater than or equal to 0 and less than 23.
<code>minute</code>	The <i>minute</i> parameter must be greater than or equal to 0 and less than 59.
<code>second</code>	The <i>second</i> parameter must be greater than or equal to 0 and less than 59.
<code>microsecond</code>	The <i>microsecond</i> parameter must be greater than or equal to 0 and less than 1000000.
<code>tzinfo</code>	The <i>tzinfo</i> parameter must be a <code>tzinfo</code> subclass object or <code>None</code> (default).
<code>fold</code>	The <i>fold</i> parameter must be 0 or 1 (default 0).

Il parametro *tzinfo* è associato ai fusi orari, mentre *fold* all'ora solare. In questo corso non li utilizzeremo, ma vi invitiamo a familiarizzare con essi.

Vediamo come creare un oggetto tempo nella pratica. Eseguire il codice

```
from datetime import time
t = time(14, 53, 20, 1)
print("Time:", t)
print("Hour:", t.hour)
print("Minute:", t.minute)
print("Second:", t.second)
print("Microsecond:", t.microsecond)
```

Risultato:

Time: 14:53:20.000001

Hour: 14

Minute: 53

Second: 20

Microsecond: 1

Nell'esempio, abbiamo passato quattro parametri al costruttore della classe: *ora*, *minuto*, *secondo* e *microsecondo*. A ciascuno di essi si può accedere utilizzando gli attributi della classe.

Nota: presto vi spiegheremo come modificare la formattazione dell'ora predefinita.

Il modulo time

Oltre alla classe time, la libreria standard di Python offre un modulo chiamato time, che fornisce una funzione relativa al tempo. Abbiamo già avuto modo di conoscere la funzione time quando abbiamo parlato della classe date. Ora vedremo un'altra utile funzione disponibile in questo modulo.

Durante questo corso dovrete trascorrere molte ore davanti al computer. A volte potreste sentire il bisogno di fare un pisolino. Perché no? Scriviamo un programma che simuli un breve pisolino di uno studente. Guardate il codice

```
import time
class Student:
    def take_nap(self, seconds):
        print("I'm very tired. I have to take a nap. See you later.")
        time.sleep(seconds)
        print("I slept well! I feel great!")

student = Student()
student.take_nap(5)
```

Result:

```
I'm very tired. I have to take a nap. See you later.
I slept well! I feel great!
```

La parte più importante del codice di esempio è l'uso della funzione `sleep` (sì, forse la ricorderete da uno dei laboratori precedenti), che sospende l'esecuzione del programma per un determinato numero di secondi. Nel nostro esempio sono 5 secondi. Hai ragione, è un pisolino molto breve. Estendere il sonno dello studente cambiando il numero di secondi. Si noti che la funzione `sleep` accetta solo un numero intero o un numero in virgola mobile.

La funzione ctime()

Il modulo time fornisce una funzione chiamata ctime, che **converte il tempo in secondi dal 1° gennaio 1970 (epoca Unix) in una stringa**.

Ricordate il risultato della funzione time? È quello che dovete passare a ctime. Guardate l'esempio

```
import time
timestamp = 1572879180
print(time.ctime(timestamp))
```

Result:

Mon Nov 4 14:53:00 2019

La funzione ctime restituisce una stringa per il timestamp passato. Nel nostro esempio, il timestamp esprime il 4 novembre 2019 alle 14:53:00.

È anche possibile chiamare la funzione ctime senza specificare l'ora in secondi. In questo caso, verrà restituita l'ora corrente:

```
import time
print(time.ctime())
```

Le funzioni `gmtime()` e `localtime()`

Alcune delle funzioni disponibili nel modulo `time` richiedono la conoscenza della classe *struct_time*, ma prima di conoscerle vediamo come si presenta la classe:

`time.struct_time`:

- `tm_year` # specifica l'anno
- `tm_mon` # specifica il mese (valore da 1 a 12)
- `tm_mday` # specifica il giorno del mese (valore da 1 a 31)
- `tm_hour` # specifica l'ora (valore da 0 a 23)
- `tm_min` # specifica il minuto (valore da 0 a 59)
- `tm_sec` # specifica il secondo (valore da 0 a 61)
- `tm_wday` # specifica il giorno della settimana (valore da 0 a 6)
- `tm_yday` # specifica il giorno dell'anno (valore da 1 a 366)
- `tm_isdst` # specifica se si applica l'ora legale (1 - sì, 0 - no, -1 - non è noto)
- `tm_zone` # specifica il nome del fuso orario (valore in forma abbreviata)
- `tm_gmtoff` # specifica l'offset a est di UTC (valore in secondi)

La classe *struct_time* consente anche di accedere ai valori tramite indici. L'indice 0 restituisce il valore in *tm_year*, mentre 8 restituisce il valore in *tm_isdst*.

Le eccezioni sono *tm_zone* e *tm_gmtoff*, che non possono essere accessibili tramite indici. Vediamo come utilizzare in pratica la classe *struct_time*. Eseguite il codice

```
import time
timestamp = 1572879180
print(time.gmtime(timestamp))
print(time.localtime(timestamp))
```

Risultato:

```
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53, tm_sec=0, tm_wday=0,
tm_yday=308, tm_isdst=0)
```

```
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53, tm_sec=0, tm_wday=0,
tm_yday=308, tm_isdst=0)
```

L'esempio mostra due funzioni che convertono il tempo trascorso dall'epoca Unix all'oggetto *struct_time*. La differenza è che la funzione *gmtime* restituisce l'oggetto *struct_time* in UTC, mentre la funzione *localtime* restituisce l'ora locale. Per la funzione *gmtime*, l'attributo *tm_isdst* è sempre 0.

Le funzioni `asctime()` e `mktime()`

Il modulo `time` ha funzioni che si aspettano un oggetto *struct_time* o una tupla che memorizza valori secondo gli indici presentati quando si è parlato della classe *struct_time*. Eseguire l'esempio

```
import time
timestamp = 1572879180
st = time.gmtime(timestamp)
print(time.asctime(st))
print(time.mktime((2019, 11, 4, 14, 53, 0, 0, 308, 0)))
```

Risultato:

```
Mon Nov 4 14:53:00 2019
1572879180.0
```

La prima delle funzioni, chiamata `asctime`, converte un oggetto *struct_time* o una tupla in una stringa. Si noti che per ottenere l'oggetto *struct_time* viene utilizzata la nota funzione `gmtime`. Se non si fornisce un argomento alla funzione `asctime`, verrà utilizzato l'orario restituito dalla funzione `localtime`.

La seconda funzione, chiamata `mktime`, converte un oggetto *struct_time* o una tupla che esprime l'ora locale nel numero di secondi dall'epoca Unix. Nel nostro esempio, le abbiamo passato una tupla, composta dai seguenti valori:

2019 => `tm_year`

11 => `tm_mon`

4 => `tm_mday`

14 => `tm_hour`

53 => `tm_min`

0 => `tm_sec`

0 => `tm_wday`

308 => `tm_yday`

0 => `tm_isdst`

Creare oggetti datetime

Nel modulo datetime, la data e l'ora possono essere rappresentate come oggetti separati o come un unico oggetto. La classe che combina data e ora si chiama datetime.

datetime(year, month, day, hour, minute, second, microsecond, *tzinfo*, *fold*)

Il suo costruttore accetta i seguenti parametri:

Parameter	Restrictions
year	The <i>year</i> parameter must be greater than or equal to 1 (MINYEAR constant) and less than or equal to 9999 (MAXYEAR constant).
month	The <i>month</i> parameter must be greater than or equal to 1 and less than or equal to 12.
day	The <i>day</i> parameter must be greater than or equal to 1 and less than or equal to the last day of the given month and year.
hour	The <i>hour</i> parameter must be greater than or equal to 0 and less than 23.
minute	The <i>minute</i> parameter must be greater than or equal to 0 and less than 59.
second	The <i>second</i> parameter must be greater than or equal to 0 and less than 59.
microsecond	The <i>microsecond</i> parameter must be greater than or equal to 0 and less than 1000000.
tzinfo	The <i>tzinfo</i> parameter must be a <i>tzinfo</i> subclass object or None (default).
fold	The <i>fold</i> parameter must be 0 or 1 (default 0).

Diamo ora un'occhiata al codice per vedere come si crea un oggetto `datetime`.

```
from datetime import datetime
dt = datetime(2019, 11, 4, 14, 53)
print("Datetime:", dt)
print("Date:", dt.date())
print("Time:", dt.time())
```

Result:

Datetime: 2019-11-04 14:53:00

Date: 2019-11-04

Time: 14:53:00

L'esempio crea un oggetto `datetime` che rappresenta il 4 novembre 2019 alle ore 14:53:00. Tutti i parametri passati al costruttore sono attributi di sola lettura della classe. Si tratta di *anno*, *mese*, *giorno*, *ora*, *minuto*, *secondo*, *microsecondo*, *tzinfo* e *fold*.

L'esempio mostra due metodi che restituiscono due oggetti diversi. Il metodo chiamato `date` restituisce l'oggetto *date* con l'anno, il mese e il giorno indicati, mentre il metodo chiamato `time` restituisce l'oggetto *time* con l'ora e i minuti indicati.

Metodi che restituiscono la data e l'ora corrente

La classe `datetime` ha diversi metodi che restituiscono la data e l'ora corrente. Questi metodi sono:

- `today()` - restituisce la data e l'ora locali correnti con l'attributo *tzinfo* impostato su *None*;
- `now()` - restituisce la data e l'ora locali correnti, come il metodo *today*, a meno che non gli si passi l'argomento opzionale *tz*. L'argomento di questo metodo deve essere un oggetto della sottoclasse *tzinfo*;
- `utcnow()` - restituisce la data e l'ora UTC corrente con l'attributo *tzinfo* impostato su *None*.

Eseguite il codice per vederli tutti in pratica. Che cosa si può dire dell'output?

```
from datetime import datetime
```

```
print("today:", datetime.today())  
print("now:", datetime.now())  
print("utcnow:", datetime.utcnow())
```

Come si può vedere, il risultato di tutti e tre i metodi è lo stesso. Le piccole differenze sono dovute al tempo trascorso tra le chiamate successive.

Nota: Per ulteriori informazioni sugli oggetti *tzinfo*, consultare la documentazione.

Ottenere un timestamp

In Internet sono disponibili molti convertitori che possono calcolare un timestamp in base a una data e a un'ora determinate, ma come possiamo farlo nel modulo `datetime`?

Questo è possibile grazie al metodo `timestamp` fornito dalla classe `datetime`. Guardate il codice

```
from datetime import datetime
dt = datetime(2020, 10, 4, 14, 55)
print("Timestamp:", dt.timestamp())
```

Risultato:

Timestamp: 1601823300.0

Il metodo `timestamp` restituisce un valore float che esprime il numero di secondi trascorsi tra la data e l'ora indicate dall'oggetto *datetime* e il 1° gennaio 1970, 00:00:00 (UTC).

Formattazione di data e ora (parte 1)

Tutte le classi del modulo `datetime` presentate finora hanno un metodo chiamato `strftime`. Si tratta di un metodo molto importante, perché ci permette di restituire la data e l'ora nel formato da noi specificato.

Il metodo `strftime` accetta un solo argomento sotto forma di stringa che specifica il formato, che può essere costituito da direttive.

Una direttiva è una stringa composta dal carattere `%` (percentuale) e da una lettera minuscola o maiuscola, ad esempio la direttiva `%Y` indica l'anno con il secolo come numero decimale. Vediamo un esempio. Eseguire il codice

```
from datetime import date
d = date(2020, 1, 4)
print(d.strftime('%Y/%m/%d'))
```

Risultato:

2020/01/04

Nell'esempio, abbiamo passato al metodo `strftime` un formato composto da tre direttive separate da `/` (slash).

Naturalmente, il carattere separatore può essere sostituito da un altro carattere o da una stringa.

È possibile inserire qualsiasi carattere nel formato, ma solo le direttive riconoscibili saranno sostituite con i valori appropriati. Nel nostro formato abbiamo usato le seguenti direttive:

- `%Y` - restituisce l'anno con il secolo come numero decimale. Nel nostro esempio, questo è il 2020.
- `%m` - restituisce il mese come numero decimale riempito di zero. Nel nostro esempio, è 01.
- `%d` - restituisce il giorno come numero decimale riempito di zero. Nel nostro esempio, è 04.

Formattazione di data e ora (parte 2)

La formattazione dell'ora funziona come la formattazione della data, ma richiede l'uso di direttive appropriate. Vediamo di seguito alcune di esse

```
from datetime import time  
from datetime import datetime
```

```
t = time(14, 53)  
print(t.strftime("%H:%M:%S"))
```

```
dt = datetime(2020, 11, 4, 14, 53)  
print(dt.strftime("%y/%B/%d %H:%M:%S"))
```

Result:

14:53:00

20/November/04 14:53:00

Il primo dei formati utilizzati riguarda solo l'ora. Come si può intuire, %H restituisce l'ora come numero decimale con riempimento a zero, %M restituisce i minuti come numero decimale con riempimento a zero e %S restituisce i secondi come numero decimale con riempimento a zero. Nel nostro esempio, %H è sostituito da 14, %M da 53 e %S da 00.

Il secondo formato utilizzato combina le direttive di data e ora. Ci sono due nuove direttive, %Y e %B. La direttiva %Y restituisce l'anno senza il secolo come numero decimale riempito di zero (nel nostro esempio è 20). La direttiva %B restituisce il mese come nome completo del locale (nel nostro esempio, è novembre).

In generale, si ha molta libertà nella creazione dei formati, ma bisogna ricordarsi di usare le direttive in modo corretto. Come esercizio, si può verificare cosa succede se, per esempio, si prova a usare la direttiva %Y nel formato passato al metodo *strftime* dell'oggetto *time*. Cercate di capire perché avete ottenuto questo risultato. Buona fortuna!

La funzione `strftime()` nel modulo `time`

Probabilmente non vi sorprenderà sapere che la funzione `strftime` è disponibile nel modulo `time`. Si differenzia leggermente dai metodi `strftime` delle classi fornite dal modulo `datetime` perché, oltre all'argomento `format`, può accettare (facoltativamente) un oggetto tupla o `struct_time`.

Se non si passa una tupla o un oggetto `struct_time`, la formattazione sarà effettuata utilizzando l'ora locale corrente. Guardate l'esempio

```
import time
timestamp = 1572879180
st = time.gmtime(timestamp)
print(time.strftime("%Y/%m/%d %H:%M:%S", st))
print(time.strftime("%Y/%m/%d %H:%M:%S"))
```

Il risultato è il seguente:

2019/11/04 14:53:00

2020/10/12 12:19:40

La creazione di un formato è simile a quella dei metodi `strftime` del modulo `datetime`. Nel nostro esempio, utilizziamo le direttive `%Y`, `%m`, `%d`, `%H`, `%M` e `%S` che già conosciamo.

Nella prima chiamata alla funzione, si formatta l'oggetto `struct_time`, mentre nella seconda chiamata (senza l'argomento opzionale), si formatta l'ora locale.

Il metodo `strptime()`

Sapere come creare un formato può essere utile quando si usa un metodo chiamato `strptime` nella classe `datetime`. A differenza del metodo `strftime`, questo metodo crea un oggetto `datetime` da una stringa che rappresenta una data e un'ora.

Il metodo `strptime` richiede di specificare il formato in cui sono state salvate la data e l'ora. Vediamo un esempio. Osservate il codice

```
from datetime import datetime
print(datetime.strptime("2019/11/04 14:53:00", "%Y/%m/%d %H:%M:%S"))
```

Risultato:

2019-11-04 14:53:00

Nell'esempio, abbiamo specificato due argomenti obbligatori. Il primo è una data e un'ora come stringa: "2019/11/04 14:53:00", mentre il secondo è un formato che facilita l'analisi di un oggetto `datetime`. Attenzione, perché se il formato specificato non corrisponde alla data e all'ora nella stringa, verrà sollevato un *ValueError*.

Nota: nel modulo `time` si trova una funzione chiamata `strptime`, che analizza una stringa che rappresenta un tempo in un oggetto *struct_time*. Il suo utilizzo è analogo al metodo `strptime` della classe `datetime`:

```
import time
print(time.strptime("2019/11/04 14:53:00", "%Y/%m/%d %H:%M:%S"))
```

risultato:

```
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53, tm_sec=0, tm_wday=0,
tm_yday=308, tm_isdst=-1)
```

Operazioni di data e ora

Prima o poi dovrete eseguire dei calcoli sulla data e sull'ora. Fortunatamente, nel modulo `datetime` esiste una classe chiamata `timedelta`, creata proprio per questo scopo.

Per creare un oggetto `timedelta`, basta eseguire una sottrazione sugli oggetti `date` o `datetime`, proprio come abbiamo fatto nell'esempio

```
from datetime import date
from datetime import datetime
```

```
d1 = date(2020, 11, 4)
d2 = date(2019, 11, 4)
```

```
print(d1 - d2)
```

```
dt1 = datetime(2020, 11, 4, 0, 0, 0)
dt2 = datetime(2019, 11, 4, 14, 53, 0)
```

```
print(dt1 - dt2)
```

Result:

366 days, 0:00:00

365 days, 9:07:00

L'esempio mostra la sottrazione per entrambi gli oggetti `date` e `datetime`. Nel primo caso, si ottiene la differenza in giorni, pari a 366 giorni. Si noti che viene visualizzata anche la differenza in ore, minuti e secondi. Nel secondo caso, il risultato è diverso, perché è stato specificato l'orario che è stato incluso nei calcoli. Il risultato è 365 giorni, 9 ore e 7 minuti.

Tra poco imparerete di più sulla creazione di oggetti `timedelta` e sulle operazioni che si possono fare con essi.

Creazione di oggetti timedelta

Si è già appreso che un oggetto timedelta può essere restituito come risultato della sottrazione di due oggetti date o datetime.

Naturalmente, si può anche creare un oggetto da soli. A tale scopo, facciamo conoscenza con gli argomenti accettati dal costruttore della classe, che sono: giorni, secondi, microsecondi, millisecondi, minuti, ore e settimane. Ognuno di essi è facoltativo e ha come valore predefinito 0.

Gli argomenti devono essere numeri interi o in virgola mobile e possono essere positivi o negativi. Vediamo un semplice esempio

```
from datetime import timedelta  
delta = timedelta(weeks=2, days=2, hours=3)  
print(delta)
```

Result:

16 days, 3:00:00

Il risultato di 16 giorni si ottiene convertendo l'argomento settimane in giorni (2 settimane = 14 giorni) e aggiungendo l'argomento giorni (2 giorni). Questo è un comportamento normale, perché l'oggetto `timedelta` memorizza internamente solo giorni, secondi e microsecondi. Allo stesso modo, l'argomento dell'ora viene convertito in minuti. Si veda l'esempio seguente:

```
from datetime import timedelta
```

```
delta = timedelta(weeks=2, days=2, hours=3)
print("Days:", delta.days)
print("Seconds:", delta.seconds)
print("Microseconds:", delta.microseconds)
```

Result:

Days: 16

Seconds: 10800

Microseconds: 0

Il risultato di 10800 si ottiene convertendo 3 ore in secondi. In questo modo l'oggetto `timedelta` memorizza gli argomenti passati durante la sua creazione. Le settimane vengono convertite in giorni, le ore e i minuti in secondi e i millisecondi in microsecondi.

Creazione di oggetti timedelta: continua

Si sa già come l'oggetto timedelta memorizzi internamente gli argomenti passati. È ora di vedere come può essere utilizzato in pratica.

Vediamo alcune operazioni supportate dalle classi del modulo datetime. Eseguire il codice

```
from datetime import timedelta
from datetime import date
from datetime import datetime
delta = timedelta(weeks=2, days=2, hours=2)
print(delta)
delta2 = delta * 2
print(delta2)
d = date(2019, 10, 4) + delta2
print(d)
dt = datetime(2019, 10, 4, 14, 53) + delta2
print(dt)
```

Result:

16 days, 2:00:00

32 days, 4:00:00

2019-11-05

2019-11-05 18:53:00

L'oggetto `timedelta` può essere moltiplicato per un numero intero. Nel nostro esempio, moltiplichiamo l'oggetto che rappresenta 16 giorni e 2 ore per 2. Il risultato è un oggetto `timedelta` che rappresenta 32 giorni e 4 ore. Si noti che sia i giorni che le ore sono stati moltiplicati per 2. Un'altra operazione interessante che utilizza l'oggetto `timedelta` è l'aggiunta. Nell'esempio, abbiamo aggiunto l'oggetto `timedelta` agli oggetti `date` e `datetime`. Come risultato di queste operazioni, riceviamo oggetti `date` e `datetime` aumentati dei giorni e delle ore memorizzati nell'oggetto `timedelta`.

L'operazione di moltiplicazione presentata consente di aumentare rapidamente il valore dell'oggetto `timedelta`, mentre la moltiplicazione può anche aiutare a ottenere una data dal futuro.

Naturalmente, le classi `timedelta`, `date` e `datetime` supportano molte altre operazioni. Vi invitiamo a familiarizzare con esse nella documentazione.

LAB-47

Scenario

Durante questo corso si è appreso il metodo `strftime`, che richiede la conoscenza delle direttive per creare un formato. È ora di mettere in pratica le direttive conosciute.

Inoltre, avrete l'opportunità di esercitarvi a lavorare con la documentazione, perché dovrete trovare le direttive che non conoscete ancora.

Ecco il vostro compito:

Scrivere un programma che crei un oggetto `datetime` per il 4 novembre 2020 , 14:53:00. L'oggetto creato deve chiamare il metodo `strftime` con il formato appropriato per visualizzare il seguente risultato:

2020/11/04 14:53:00

20/November/04 14:53:00 PM

Wed, 2020 Nov 04

Wednesday, 2020 November 04

Weekday: 3

Day of the year: 309

Week number of the year: 44

Nota: Ogni riga di risultato deve essere creata chiamando il metodo `strftime` con almeno una direttiva nell'argomento `format`.

Punti di forza

1. Per creare un oggetto data, è necessario passare gli argomenti anno, mese e giorno come segue:

```
from datetime import date
```

```
my_date = date(2020, 9, 29)
```

```
print("Year:", my_date.year) # Year: 2020
```

```
print("Month:", my_date.month) # Month: 9
```

```
print("Day:", my_date.day) # Day: 29
```

L'oggetto date ha tre attributi (di sola lettura): anno, mese e giorno.

2. Il metodo today restituisce un oggetto data che rappresenta la data locale corrente:

```
from datetime import date
```

```
print("Today:", date.today()) # Displays: Today: 2020-09-29
```

3. In Unix, il timestamp esprime il numero di secondi dal 1° gennaio 1970, 00:00:00 (UTC). Questa data è chiamata "Unix epoch", perché ha dato inizio al conteggio del tempo sui sistemi Unix. Il timestamp è in realtà la differenza tra una data particolare (inclusa l'ora) e il 1° gennaio 1970, 00:00:00 (UTC), espressa in secondi. Per creare un oggetto data da un timestamp, dobbiamo passare un timestamp Unix al metodo fromtimestamp:

```
from datetime import date
```

```
import date
```

```
import time
```

```
timestamp = time.time()
```

```
d = date.fromtimestamp(timestamp)
```

Nota: la funzione time restituisce il numero di secondi dal 1° gennaio 1970 al momento attuale sotto forma di numero float.

4. Il costruttore della classe `time` accetta sei argomenti (*ora*, *minuto*, *secondo*, *microsecondo*, *tzinfo* e *fold*). Ciascuno di questi argomenti è opzionale.

```
from datetime import time
```

```
t = time(13, 22, 20)
```

```
print("Hour:", t.hour) # Hour: 13
```

```
print("Minute:", t.minute) # Minute: 22
```

```
print("Second:", t.second) # Second: 20
```

5. Il modulo `time` contiene la funzione `sleep`, che sospende l'esecuzione del programma per un determinato numero di secondi, ad esempio:

```
time.sleep(10)
```

```
print("Hello world!") # This text will be displayed after 10 seconds.
```

6. Nel modulo `datetime`, la data e l'ora possono essere rappresentate come oggetti separati o come un unico oggetto. La classe che combina data e ora si chiama *datetime*. Tutti gli argomenti passati al costruttore sono attributi di sola lettura della classe. Essi sono *anno*, *mese*, *giorno*, *ora*, *minuto*, *secondo*, *microsecondo*, *tzinfo* e *fold*:

```
from datetime import datetime
```

```
dt = datetime(2020, 9, 29, 13, 51)
print("Datetime:", dt) # Displays: Datetime: 2020-09-29 13:51:00
```

7. Il metodo `strftime` accetta un solo argomento sotto forma di stringa che specifica un formato che può essere composto da direttive. Una direttiva è una stringa composta dal carattere `%` (percentuale) e da una lettera minuscola o maiuscola. Di seguito sono riportate alcune direttive utili:

- `%Y` - restituisce l'anno con il secolo come numero decimale;
- `%m` - restituisce il mese come numero decimale riempito di zero;
- `%d` - restituisce il giorno come numero decimale riempito di zero;
- `%H` - restituisce l'ora come numero decimale riempito di zero;
- `%M` - restituisce il minuto come numero decimale riempito di zero;
- `%S` - restituisce il secondo come numero decimale riempito di zero.

Esempio:

```
from datetime import date
d = date(2020, 9, 29)
print(d.strftime('%Y/%m/%d')) # Displays: 2020/09/29
```

8. È possibile eseguire calcoli su oggetti data e datetime, ad esempio:

```
from datetime import date
d1 = date(2020, 11, 4)
d2 = date(2019, 11, 4)
d = d1 - d2
print(d) # Visualizza: 366 giorni, 0:00:00.
print(d * 2) # Visualizza: 732 giorni, 0:00:00.
```

Il risultato della sottrazione viene restituito come oggetto timedelta che esprime la differenza in giorni tra le due date dell'esempio precedente.

Si noti che viene visualizzata anche la differenza in ore, minuti e secondi. L'oggetto timedelta può essere utilizzato per ulteriori calcoli (ad esempio, è possibile moltiplicarlo per 2).

Esercizio 1

Qual è l'output del seguente snippet?

```
from datetime import time  
t = time(14, 53)  
print(t.strftime("%H:%M:%S"))
```

Soluzione

14:39:00

Esercizio 2

Qual è l'output del seguente snippet?

```
from datetime import datetime  
dt1 = datetime(2020, 9, 29, 14, 41, 0)  
dt2 = datetime(2020, 9, 28, 14, 41, 0)  
print(dt1 - dt2)
```

Soluzione

1 day, 0:00:00

Introduzione al modulo calendar

Oltre ai moduli datetime e time, la libreria standard di Python fornisce un modulo chiamato calendar che, come suggerisce il nome, offre **funzioni relative al calendario**.

Uno di questi è naturalmente la visualizzazione del calendario. È importante che i giorni della settimana siano visualizzati dal lunedì alla domenica e che ogni giorno della settimana sia rappresentato da un numero intero:

Day of the week	Integer value	Constant
Monday	0	<code>calendar.MONDAY</code>
Tuesday	1	<code>calendar.TUESDAY</code>
Wednesday	2	<code>calendar.WEDNESDAY</code>
Thursday	3	<code>calendar.THURSDAY</code>
Friday	4	<code>calendar.FRIDAY</code>
Saturday	5	<code>calendar.SATURDAY</code>
Sunday	6	<code>calendar.SUNDAY</code>

La tabella precedente mostra la rappresentazione dei giorni della settimana nel modulo calendario. Il primo giorno della settimana (lunedì) è rappresentato dal valore 0 e dalla costante *calendar.MONDAY*, mentre l'ultimo giorno della settimana (domenica) è rappresentato dal valore 6 e dalla costante *calendar.SUNDAY*.



Per i mesi, i valori interi sono indicizzati a partire da 1, cioè gennaio è rappresentato da 1 e dicembre da 12. Purtroppo non esistono costanti che esprimano i mesi. Purtroppo non esistono costanti che esprimano i mesi. Le informazioni di cui sopra vi saranno utili quando lavorerete con il modulo calendario in questa parte del corso, ma prima iniziamo con alcuni semplici esempi di calendario.

Il vostro primo calendar

Inizierete la vostra avventura con il modulo calendario con una semplice funzione chiamata `calendar`, che consente di **visualizzare il calendario dell'intero anno**. Vediamo come utilizzarla per visualizzare il calendario del 2020. Eseguire il codice

```
import calendar
print(calendar.calendar(2020))
```

Il risultato visualizzato è simile a quello del comando *cal* disponibile in Unix. Se si desidera modificare la formattazione predefinita del calendario, è possibile utilizzare i seguenti parametri:

- `w` - larghezza della colonna della data (valore predefinito 2)
- `l` - numero di linee per settimana (valore predefinito 1)
- `c` - numero di spazi tra le colonne del mese (default 6)
- `m` - numero di colonne (predefinito 3)

La funzione `calendar` richiede di specificare l'anno, mentre gli altri parametri responsabili della formattazione sono opzionali. Vi invitiamo a provare personalmente questi parametri.

Una buona alternativa alla funzione precedente è la funzione *prcal*, che accetta gli stessi parametri della funzione `calendar`, ma non richiede l'uso della funzione `print` per visualizzare il calendario. Il suo utilizzo si presenta come segue:

```
import calendar
calendar.prcal(2020)
```

Calendario per un mese specifico

Il modulo calendar ha una funzione chiamata month, che consente di visualizzare un calendario per un mese specifico. Il suo utilizzo è molto semplice, basta specificare l'anno e il mese - si veda il codice

```
import calendar  
print(calendar.month(2020, 11))
```

L'esempio mostra il calendario di novembre 2020. Come per la funzione calendario, è possibile modificare la formattazione predefinita utilizzando i seguenti parametri:

- w - larghezza della colonna della data (valore predefinito 2)
- l - numero di linee per settimana (valore predefinito 1)

Nota: è possibile utilizzare anche la funzione prmonth, che ha gli stessi parametri della funzione month, ma non richiede l'uso della funzione print per visualizzare il calendario.

La funzione `setfirstweekday()`

Come si sa, per impostazione predefinita nel modulo `calendar`, il primo giorno della settimana è il lunedì. Tuttavia, è possibile modificare questo comportamento utilizzando una funzione chiamata `setfirstweekday`. Ricordate la tabella che mostra i giorni della settimana e la loro rappresentazione sotto forma di valori interi? È ora di usarla, perché il metodo `setfirstweekday` richiede un parametro che esprima il giorno della settimana sotto forma di valore intero. Guardate l'esempio

```
import calendar
calendar.setfirstweekday(calendar.SUNDAY)
calendar.prmonth(2020, 12)
```

L'esempio utilizza la costante `calendar.SUNDAY`, che contiene il valore 6. Naturalmente, si potrebbe passare questo valore direttamente alla funzione `setfirstweekday`, ma la versione con una costante è più elegante. Il risultato è un calendario che mostra il mese di dicembre 2020, in cui il primo giorno di tutte le settimane è la domenica.

La funzione weekday()

Un'altra funzione utile fornita dal modulo calendario è la funzione chiamata weekday, che restituisce il giorno della settimana come valore intero per l'anno, il mese e il giorno dati. Vediamola in pratica.

Eseguite il codice per verificare il giorno della settimana che cade il 24 dicembre 2020.

```
import calendar  
print(calendar.weekday(2020, 12, 24))
```

Risultato:

3

La funzione weekday restituisce 3, il che significa che il 24 dicembre 2020 è un giovedì.

La funzione `weekheader()`

Probabilmente si è notato che il calendario contiene intestazioni settimanali in forma abbreviata. Se necessario, è possibile ottenere nomi di giorni della settimana abbreviati utilizzando il metodo `weekheader`.

Il metodo `weekheader` richiede di specificare la larghezza in caratteri per un giorno della settimana. Se la larghezza fornita è maggiore di 3, si otterranno comunque i nomi abbreviati dei giorni della settimana, composti da tre caratteri.

Vediamo quindi come ottenere un'intestazione più piccola. Eseguire il codice

```
import calendar  
print(calendar.weekheader(2))
```

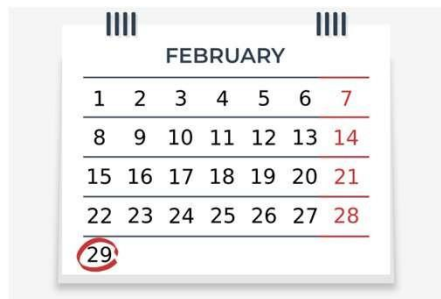
Result:

Mo Tu We Th Fr Sa Su

Nota: se si modifica il primo giorno della settimana, ad esempio utilizzando la funzione `setfirstweekday`, il risultato della funzione `weekheader` ne risentirà.

Come si controlla se un anno è bisestile?

Il modulo del calendario offre due funzioni utili per verificare se gli anni sono bisestili.



Il primo, chiamato `isleap`, restituisce *True* se l'anno passato è bisestile, altrimenti *False*. Il secondo, chiamato `leapdays`, restituisce il numero di anni bisestili nell'intervallo di anni indicato.

Eseguire il codice

```
import calendar
print(calendar.isleap(2020))
print(calendar.leapdays(2010, 2021)) # Up to but not including 2021.
```

Result:

True

3

Nell'esempio, otteniamo il risultato 3, perché nel periodo dal 2010 al 2020 ci sono solo tre anni bisestili (nota: il 2021 non è incluso). Si tratta degli anni 2012, 2016 e 2020.

Classi per la creazione di calendari

Le funzioni presentate non sono tutto ciò che il modulo calendario offre. Oltre ad esse, possiamo utilizzare le seguenti classi:

`calendar.Calendar` - fornisce metodi per preparare i dati del calendario per la formattazione;

`calendar.TextCalendar` - è usato per creare calendari di testo regolari;

`calendar.HTMLCalendar` - è usato per creare calendari HTML;

`calendar.LocalTextCalendar` - è una sottoclasse della classe `calendar.TextCalendar`. Il costruttore di questa classe prende il parametro *locale*, che viene utilizzato per restituire i nomi dei mesi e dei giorni della settimana appropriati.

`calendar.LocalHTMLCalendar` - è una sottoclasse della classe `calendar.HTMLCalendar`. Il costruttore di questa classe prende il parametro *locale*, che viene utilizzato per restituire i nomi dei mesi e dei giorni della settimana appropriati.

Durante questo corso, avete già avuto modo di creare calendari di testo quando abbiamo parlato delle funzioni del modulo calendario.

È ora di provare qualcosa di nuovo. Diamo un'occhiata più da vicino ai metodi della classe `Calendar`

Calendar

FEBRUARY						
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

TextCalendar

FEBRUARY						
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

HTMLCalendar

<div>

FEBRUARY						
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

</div>

LocalTextCalendar

FEBRUARY						
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

LocalHTMLCalendar

<div>

FEBRUARY						
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

</div>

Creare un oggetto Calendario

Il costruttore della classe Calendario accetta un parametro opzionale chiamato primo giorno della settimana, per impostazione predefinita uguale a 0 (lunedì).

Il parametro firstweekday deve essere un numero intero compreso tra 0-6. A tale scopo, si possono utilizzare le costanti già note - si veda il codice

```
import calendar
c = calendar.Calendar(calendar.SUNDAY)
for weekday in c.iterweekdays():
    print(weekday, end=" ")
```

Il programma produce il seguente risultato:

6 0 1 2 3 4 5

L'esempio di codice utilizza il metodo della classe Calendar denominato iterweekdays, che restituisce un iteratore per i numeri dei giorni della settimana.

Il primo valore restituito è sempre uguale al valore della proprietà firstweekday. Poiché nel nostro esempio il primo valore restituito è 6, significa che la settimana inizia di domenica.

Il metodo itermonthdates()

La classe Calendar ha diversi metodi che restituiscono un iteratore. Uno di questi è il metodo itermonthdates, che richiede di specificare l'anno e il mese.

Di conseguenza, vengono restituiti tutti i giorni del mese e dell'anno specificati, nonché tutti i giorni prima dell'inizio o della fine del mese necessari per ottenere una settimana completa.

Ogni giorno è rappresentato da un oggetto datetime.date. Si veda l'esempio

```
import calendar
c = calendar.Calendar()
for date in c.itermonthdates(2019, 11):
    print(date, end=" ")
```

Il codice visualizza tutti i giorni di novembre 2019. Poiché il primo giorno di novembre 2019 è un venerdì, vengono restituiti anche i giorni successivi per ottenere la settimana completa: 10/28/2019 (lunedì) 10/29/2019 (martedì) 10/30/2019 (mercoledì) 10/31/2019 (giovedì).

L'ultimo giorno di novembre 2019 è stato un sabato, quindi per mantenere la settimana completa, un giorno in più viene restituito il 12/01/2019 (domenica).

Altri metodi che restituiscono iteratori

Un altro metodo utile della classe Calendar è quello chiamato `itermonthdays`, che prende come parametri l'anno e il mese e poi restituisce l'iteratore dei giorni della settimana rappresentati da numeri.

Guardate l'esempio

```
import calendar
c = calendar.Calendar()
for iter in c.itermonthdays(2019, 11):
    print(iter, end=" ")
```

Avrete sicuramente notato il gran numero di 0 restituiti come risultato del codice di esempio. Si tratta di giorni al di fuori dell'intervallo di mesi specificato che vengono aggiunti per mantenere la settimana completa.

I primi quattro zeri rappresentano il 28/10/2019 (lunedì) il 29/10/2019 (martedì) il 30/10/2019 (mercoledì) il 31/10/2019 (giovedì). I numeri rimanenti rappresentano i giorni del mese, tranne l'ultimo valore di 0, che sostituisce la data 12/01/2019 (domenica).

Esistono altri quattro metodi simili nella classe Calendar che differiscono per i dati restituiti:

- `itermonthdates2` - restituisce i giorni sotto forma di tuple composte da un numero di giorno del mese e un numero di giorno della settimana;
- `itermonthdates3` - restituisce i giorni sotto forma di tuple composte da anno, mese e giorno del mese. Questo metodo è disponibile dalla versione 3.7;
- `itermonthdates4` - restituisce i giorni sotto forma di tuple composte da anno, mese, giorno del mese e giorno della settimana. Questo metodo è disponibile dalla versione 3.7 di Python.

A scopo di test, utilizzare l'esempio precedente e vedere come appaiono in pratica i valori di ritorno dei metodi descritti.

Il metodo monthdays2calendar()

La classe Calendario ha diversi altri metodi utili che possono essere approfonditi nella documentazione (<https://docs.python.org/3/library/calendar.html>).

Uno di questi è il metodo monthdays2calendar, che prende l'anno e il mese e restituisce un elenco di settimane in un mese specifico. Ogni settimana è una tupla composta da numeri di giorni e giorni della settimana. Guardate il codice

```
import calendar
c = calendar.Calendar()
for data in c.monthdays2calendar(2020, 12):
    print(data)
```

Si noti che i numeri dei giorni al di fuori del mese sono rappresentati da 0, mentre i numeri dei giorni della settimana sono un numero da 0 a 6, dove 0 è il lunedì e 6 la domenica.

Tra poco, questo metodo potrà esservi utile per completare un compito di laboratorio. Siete pronti?

LAB-48

Scenario

In questo corso abbiamo analizzato un po' la classe `Calendar`. Il vostro compito è quello di estendere la sua funzionalità con un nuovo metodo chiamato `count_weekday_in_year`, che prende come parametri un anno e un giorno della settimana e restituisce il numero di occorrenze di uno specifico giorno della settimana nell'anno.

Utilizzate i seguenti suggerimenti:

Creare una classe chiamata `MyCalendar` che estende la classe `Calendar`;

creare il metodo `count_weekday_in_year` con i parametri `year` e `weekday`. Il parametro giorno della settimana deve essere un valore compreso tra 0 e 6, dove 0 è lunedì e 6 è domenica. Il metodo deve restituire il numero di giorni come numero intero;

nella vostra implementazione, utilizzate il metodo `monthdays2calendar` della classe `Calendar`.

I risultati attesi sono i seguenti:

Argomenti di esempio

anno=2019, giorno della settimana=0

Risultato previsto

52

Argomenti di esempio

anno=2000, giorno della settimana=6

Risultato previsto

53

Punti di forza

1. Nel modulo calendario, i giorni della settimana sono visualizzati dal lunedì alla domenica. Ogni giorno della settimana viene rappresentato sotto forma di numero intero, dove il primo giorno della settimana (lunedì) è rappresentato dal valore 0, mentre l'ultimo giorno della settimana (domenica) è rappresentato dal valore 6.
2. Per visualizzare un calendario per qualsiasi anno, richiamare la funzione `calendario` con l'anno passato come argomento, ad es:

```
import calendar  
print(calendar.calendar(2020))
```

Nota: una buona alternativa alla funzione precedente è la funzione `prcal`, che accetta gli stessi parametri della funzione `calendario`, ma non richiede l'uso della funzione `print` per visualizzare il calendario.

3. Per visualizzare un calendario per qualsiasi mese dell'anno, richiamare la funzione `month`, passandole anno e mese. Ad esempio:

```
import calendar  
print(calendar.month(2020, 9))
```

Nota: è possibile utilizzare anche la funzione `prmonth`, che ha gli stessi parametri della funzione `month`, ma non richiede l'uso della funzione `print` per visualizzare il calendario.

4. La funzione `setfirstweekday` consente di modificare il primo giorno della settimana. Assume un valore da 0 a 6, dove 0 è domenica e 6 è sabato.

5. Il risultato della funzione `giorno della settimana` è un giorno della settimana come valore intero per un dato anno, mese e giorno:

```
import calendar  
print(calendar.weekday(2020, 9, 29)) # This displays 1, which means Tuesday.
```

6. La funzione `weekheader` restituisce i nomi dei giorni della settimana in forma abbreviata. Il metodo `weekheader` richiede di specificare la larghezza in caratteri per un giorno della settimana. Se la larghezza fornita è maggiore di 3, si otterranno comunque i nomi abbreviati dei giorni della settimana, composti da soli tre caratteri. Per esempio: `import calendar`

```
print(calendar.weekheader(2)) # This display: Mo Tu We Th Fr Sa Su
```

7. Una funzione molto utile disponibile nel modulo calendario è quella chiamata `isleap`, che, come suggerisce il nome, consente di verificare se l'anno è bisestile o meno: `import calendar`

```
print(calendar.isleap(2020)) # This displays: True
```

8. È possibile creare un oggetto calendario utilizzando la classe `Calendar` che, al momento della creazione del suo oggetto, consente di modificare il primo giorno della settimana con il parametro opzionale `firstweekday`, ad es:

```
import calendar
```

```
c = calendar.Calendar(2)
```

```
for weekday in c.iterweekdays():
```

```
    print(weekday, end=" ")
```

```
# Result: 2 3 4 5 6 0 1
```

L'`iterweekdays` restituisce un iteratore per i numeri dei giorni della settimana. Il primo valore restituito è sempre uguale al valore della proprietà `firstweekday`.

Esercizio 1

Qual è l'output del seguente snippet?

```
import calendar  
print(calendar.weekheader(1))
```

Soluzione

M T W T F S S

Esercizio 2

Qual è l'output del seguente snippet?

```
import calendar  
c = calendar.Calendar()  
for weekday in c.iterweekdays():  
    print(weekday, end=" ")
```

Soluzione

0 1 2 3 4 5 6