@IT
atmarkIT

解決！ Python
（日付データ編）

かわさきしんじ，Deep Insider 編集部［著］

［解決！Python］今日の日付や時刻を取得するには

Python に標準で付属する datetime モジュールと time モジュールを使って、現在の日付と時刻を取得する方法を紹介する。

かわさきしんじ，Deep Insider 編集部（2024 年 04 月 12 日）

* 本稿は 2021 年 11 月 2 日に公開された記事を Python 3.12.3 で動作確認し、Python 3.12 において datetime.datetime.utcnow クラスメソッドが非推奨になったことを追記したものです（確認日：2024 年 4 月 12 日）。

```
# datetime モジュールを使った現在の日付と時刻の取得
import datetime

dt = datetime.datetime.today()  # ローカルな現在の日付と時刻を取得
print(dt)  # 2021-10-29 15:58:08.356501

# 日付と時刻を構成する要素の取り出し
print(f'year: {dt.year}, month: {dt.month}, day: {dt.day}')
print(f'hour: {dt.hour}, minute: {dt.minute}, second: {dt.second}')
print(f'micro second: {dt.microsecond}')
# 出力例
#year: 2021, month: 10, day: 29
#hour: 15, minute: 58, second: 8
#micro second: 356501

# datetime オブジェクトから日付または時刻を取り出す
d = dt.date()
print(d)  # 2021-10-29

t = dt.time()
print(t)  # 15:58:08.356501

tdy = datetime.date.today()  # 今日の日付の取得
print(tdy)  # 2021-10-29

# タイムゾーンを考慮する
dt = datetime.datetime.now()  # タイムゾーンなしで現在の日付と時刻を取得
print(dt)  # 2021-10-29 16:16:23.202059

dt = datetime.datetime.now(datetime.timezone.utc)  # タイムゾーン付きで UTC を取得
print(dt)  # 2021-10-29 07:23:14.464801+00:00
print(dt.tzinfo)  # UTC

dt = datetime.datetime.utcnow()  # タイムゾーンなしで UTC を取得
print(dt)  # 2021-10-29 07:31:41.503042
```

```
t_delta = datetime.timedelta(hours=9)  # 9時間
JST = datetime.timezone(t_delta, 'JST')  # UTCから9時間差の「JST」タイムゾーン
dt = datetime.datetime.now(JST)  # タイムゾーン付きでローカルな日付と時刻を取得
print(dt)

# timeモジュールを使った現在の日付と時刻の取得
import time

t = time.time()  # UNIX時間（1970/01/01 00:00:00からの経過時刻）を取得
dt_from_timestamp = datetime.datetime.fromtimestamp(t)
print(dt_from_timestamp)  # 2021-10-29 16:59:04.741680

c_time = time.ctime(t)  # UNIX時刻を文字列表現に変換
print(c_time)  # Fri Oct 29 16:59:04 2021

local_time = time.localtime(t)  # ローカル時刻をtime.struct_time型として取得
print(local_time)
gm_time = time.gmtime(t)  # UTC時刻をtime.struct_time型として取得
print(gm_time)
# 出力例
#time.struct_time(tm_year=2021, tm_mon=10, tm_mday=29, tm_hour=16, tm_min=59,
# tm_sec=4, tm_wday=4, tm_yday=302, tm_isdst=0)
#time.struct_time(tm_year=2021, tm_mon=10, tm_mday=29, tm_hour=7, tm_min=59,
#tm_sec=4, tm_wday=4, tm_yday=302, tm_isdst=0)

asc_time = time.asctime(local_time)  # 上のlocal_timeを文字列表現に変換
print(asc_time)  # Fri Oct 29 16:59:04 2021

dt = datetime.datetime.strptime(asc_time, '%a %b %d %H:%M:%S %Y')
print(dt)  # 2021-10-29 16:59:04
```

datetime モジュールを使った現在の日付と時刻の取得

Python で現在の日付と時刻を取得するには幾つかの方法がある。代表的なものとしては Python に標準で付属する datetime モジュールまたは time モジュールを使う方法が挙げられる。まずは datetime モジュールを使う方法を見てみよう。

```
import datetime

dt = datetime.datetime.today()  # ローカルな現在の日付と時刻を取得
print(dt)  # 2021-10-29 15:58:08.356501
```

一番簡単な方法は上に示した、datetime モジュールに含まれる datetime クラスの today クラスメソッドを呼び出すことだ。today クラスメソッドが返送するのは、datetime.datetime 型のオブジェクトであり、以下のように year ／ month ／ day ／ hour ／ minute ／ second ／ microsecond 属性によって日付や時刻を構成する各要素にアクセスできる。

```
print(f'year: {dt.year}, month: {dt.month}, day: {dt.day}')
print(f'hour: {dt.hour}, minute: {dt.minute}, second: {dt.second}')
print(f'micro second: {dt.microsecond}')
# 出力例
#year: 2021, month: 10, day: 29
#hour: 15, minute: 58, second: 8
#micro second: 356501
```

あるいは strftime メソッドを呼び出して、日付と時刻を独自の書式でフォーマットすることも可能だが、これについては「日付や時刻を YYMMDDhhmmss などの形式に書式化するには」で紹介する。

取得した datetime クラスのオブジェクトには、その日付を取り出す date メソッドと時刻を取り出す time メソッドもある。使用例を以下に示す。

```
d = dt.date()
print(d)  # 2021-10-29

t = dt.time()
print(t)  # 15:58:08.356501
```

なお、今日の日付を取得するだけなら、以下のように datetime モジュールに含まれる date クラスの today クラスメソッドを呼び出してもよい。

```
tdy = datetime.date.today()
print(tdy)  # 2021-10-29
```

date メソッドの戻り値は datetime.date クラスのオブジェクトであり、time メソッドの戻り値は datetime.time クラスのオブジェクトだ。これらには datetime クラスのオブジェクトと同様、日付を構成する要素を表す属性と時刻を構成する要素を表す属性がある（例は省略）。

today メソッドと同様に、datetime クラスの now クラスメソッドを呼び出しても現在の日付と時刻を取得できる。

```
dt = datetime.datetime.now()  # タイムゾーンなしで現在の日付と時刻を取得
print(dt)  # 2021-10-29 16:16:23.202059
```

today メソッドを呼び出したり、now メソッドを引数なしで呼び出したりした場合、得られる値はローカルな日付と時刻となる。この場合、datetime オブジェクトにはタイムゾーン情報が含まれない（こうしたオブジェクトのことを「naive」と呼ぶ。逆にタイムゾーンなどの情報を含むオブジェクトのことを「aware」と呼ぶ）。

now メソッドはタイムゾーンを指定して呼び出せる。その場合は、取得する日付と時刻は指定したタイムゾーンのものに変換される。以下は UTC（協定世界時）をタイムゾーンに指定して now メソッドを呼び出す例だ。

```
dt = datetime.datetime.now(datetime.timezone.utc)  # タイムゾーン付きで UTC を取得
print(dt)  # 2021-10-29 07:23:14.464801+00:00
print(dt.tzinfo)  # UTC
```

2 つ目の出力ではタイムゾーンが「UTC」となっていること、その上の出力の最後にはそのことを裏付ける「+00:00」があることに注目してほしい。

なお、タイムゾーン情報なしで UTC を取得する utcnow メソッドもある（このメソッドは Python 3.12 で非推奨となり、「datetime.datetime.now(datetime.UTC)」のようにして UTC を取得することが推奨されている）。

```
dt = datetime.datetime.utcnow()  # タイムゾーンなしで UTC を取得
print(dt)  # 2021-10-29 07:31:41.503042
```

タイムゾーンを指定して now メソッドを呼び出すには、datetime.timezone クラスまたは datetime.tzinfo クラスの派生クラスのインスタンスを now メソッドに渡す。以下は、JST（日本標準時。UTC+9）を表す timezone クラスのインスタンスを作成して、それを now メソッドに渡すことで JST をタイムゾーンとするローカルな時刻を取得する例だ。

```
t_delta = datetime.timedelta(hours=9)  # 9 時間
JST = datetime.timezone(t_delta, 'JST')  # UTC から 9 時間差の「JST」タイムゾーン
dt = datetime.datetime.now(JST)  # タイムゾーン付きでローカルな日付と時刻を取得
print(dt)  # 2021-10-29 16:44:02.888144+09:00
```

datetime クラスのオブジェクトにタイムゾーンを設定するには、astimezone メソッドを使えるが、これについては Python のドキュメントを参照されたい。

time モジュールを使った現在の日付と時刻の取得

次に time モジュールを使っても現在の日付と時刻を取得する方法を見る。

time モジュールの time 関数は、UNIX 時間（1970 年 1 月 1 日 0 時 0 分 0 秒）からの経過時間を浮動小数点数で取得する。これを datetime モジュールの datetime クラスが持つ fromtimestamp クラスメソッドに渡すと、これに対応するローカルな日付と時刻を表す datetime オブジェクトが得られる。

```
import time

t = time.time()  # UNIX時間（1970/01/01 00:00:00からの経過時刻）を取得
dt_from_timestamp = datetime.datetime.fromtimestamp(t)
print(dt_from_timestamp)  # 2021-10-29 16:59:04.741680
```

time.time関数の戻り値をtime.ctime関数に渡すと、その値に対応する日付と時刻を文字列で表現したものが得られる。以下に例を示す。

```
c_time = time.ctime(t)  # UNIX時刻を文字列表現に変換
print(c_time)  # Fri Oct 29 16:59:04 2021
```

time.time関数の戻り値をtime.localtime関数に渡すと、その値に対応するローカルな日付と時刻を表すtime.struct_time型の値が得られる。また、time.gmtime関数に渡すと、その値に対応するUTCを表すtime.struct_time型の値が得られる（以下のコード例から分かるように、こちらはローカルな時刻からUTCへ変換が行われる）。なお、引数なしでlocaltime関数とgmtime関数を呼び出した場合は、呼び出し時点の時刻が使われる。

```
local_time = time.localtime(t)  # ローカル時刻をtime.struct_time型として取得
print(local_time)
gm_time = time.gmtime(t)  # UTC時刻をtime.struct_time型として取得
print(gm_time)
# 出力例
#time.struct_time(tm_year=2021, tm_mon=10, tm_mday=29, tm_hour=16, tm_min=59,
# tm_sec=4, tm_wday=4, tm_yday=302, tm_isdst=0)
#time.struct_time(tm_year=2021, tm_mon=10, tm_mday=29, tm_hour=7, tm_min=59,
#tm_sec=4, tm_wday=4, tm_yday=302, tm_isdst=0)
```

localtime関数やgmtime関数が返すstruct_time型の値は、time.asctime関数に渡すことで、その文字列表現が得られる。

```
asc_time = time.asctime(local_time)  # 上のlocal_timeを文字列表現に変換
print(asc_time)  # Fri Oct 29 16:59:04 2021
```

最後に、time.ctime関数やtime.asctime関数が返す文字列表現を、適切な表現と共にdatetimeモジュールのdatetimeクラスが持つstrptime関数に渡すと、datetimeクラスのオブジェクトへとパースできる。

```
dt = datetime.datetime.strptime(asc_time, '%a %b %d %H:%M:%S %Y')
print(dt)  # 2021-10-29 16:59:04
```

［解決！Python］日付や時刻を YYMMDDhhmmss などの形式に書式化するには

datetime モジュールと time モジュールには、それらが提供する日付／時刻のデータを書式化するために使える strftime 関数／メソッドがある。それらの使い方を紹介する。

かわさきしんじ，Deep Insider 編集部（2023 年 09 月 11 日）

* 本稿は 2021 年 11 月 9 日に公開された記事を Python 3.11.5 で動作確認したものです（確認日：2023 年 9 月 11 日）。

```
# datetime モジュールを使用
import datetime

t_delta = datetime.timedelta(hours=9)
JST = datetime.timezone(t_delta, 'JST')
now = datetime.datetime.now(JST)
print(repr(now))
# 出力例
#datetime.datetime(2021, 11, 4, 17, 37, 28, 114417, tzinfo=datetime.timezone
#(datetime.timedelta(seconds=32400), 'JST'))
print(now)  # 2021-11-04 17:37:28.114417+09:00

# YYYYMMDDhhmmss 形式に書式化
d = now.strftime('%Y%m%d%H%M%S')
print(d)  # 20211104173728

d = f'{now:%Y%m%d%H%M%S}'  # f 文字列
d = format(now, '%Y%m%d%H%M%S')  # format 関数
d = '{:%Y%m%d%H%M%S}'.format(now)  # 文字列の format メソッド
print(d)  # 20211104173728

# YYYY/MM/DD hh:mm:ss 形式に書式化
d = now.strftime('%Y/%m/%d %H:%M:%S')
print(d)  # 2021/11/04 17:37:28

# MM/DD/YY hh:mm:ss 形式に書式化
d = now.strftime('%x %X')
print(d)  # 11/04/21 17:37:28

# 日付のみを書式化
d = now.date().strftime('%Y/%m/%d')
print(d)  # 2021/11/04

# 時刻のみを書式化
t = now.time().strftime('%X')
print(t)  # 17:37:28

# 西暦を 2 桁に
d = now.strftime('%Y/%m/%d %H:%M:%S')
```


→目次に戻る

```
print(d)  # 21/11/04 17:37:28

# 12時間制+AM／PM表示
d = now.strftime('%Y/%m/%d %I:%M(%p)')
print(d)  # 2021/11/04 05:37(PM)

# 曜日を含む日付
d = now.strftime('%Y年%m月%d日（%a）')
print(d)  # 2021年11月04日（Thu）

d_week = {'Sun': '日', 'Mon': '月', 'Tue': '火', 'Wed': '水',
          'Thu': '木', 'Fri': '金', 'Sat': '土'}
key = now.strftime('%a')
w = d_week[key]
d = now.strftime('%Y年%m月%d日') + f'（{w}）' #f'{now:%Y年%m月%d日}（{w}）'
print(d)  # 2021年11月04日（木）

d_week = '日月火水木金土日'  # インデックス0の'日'は使用されない
idx = now.strftime('%u')  # '%u'では月曜日がインデックス'1'となる
w = d_week[int(idx)]
d = now.strftime('%Y年%m月%d日') + f'（{w}）'
print(d)  # 2021年11月04日（木）

# タイムゾーン
d = now.strftime('%X%z(%Z)')
print(d)  # 17:37:28+0900(JST)

# timeモジュールを使用
import time

now = time.localtime()
print(now)
# 出力例
#time.struct_time(tm_year=2021, tm_mon=11, tm_mday=4, tm_hour=19, tm_min=40,
#tm_sec=1, tm_wday=3, tm_yday=308, tm_isdst=0)

d = time.strftime('%Y%m%d%H%M%S', now)  # YYYYMMDDhhmmssに書式化
d = time.strftime('%Y/%m/%d %H:%M:%S', now)  # YYYY/MM/DD hh:mm:ssに書式化
d = time.strftime('%x %X', now)  # MM/DD/YY hh:mm:ssに書式化
d = time.strftime('%Y/%m/%d %H:%M:%S', now)  # YY/MM/DD hh:mm:ssに書式化
d = time.strftime('%Y/%m/%d %I:%M(%p)', now)  # 12時間制+AM／PM表示
d = time.strftime('%Y年%m月%d日（%a）', now)  # 曜日を含む日付

d_week = {'Sun': '日', 'Mon': '月', 'Tue': '火', 'Wed': '水',
          'Thu': '木', 'Fri': '金', 'Sat': '土'}
key = time.strftime('%a', now)
w = d_week[key]
d = time.strftime('%Y年%m月%d日', now) + f'（{w}）'
print(d)  # 2021年11月04日（木）
```


→目次に戻る

datetime モジュールを使用している場合

datetime モジュールが提供する datetime クラス、date クラス、time クラスには strftime メソッドがあり、これらを使うことでそれぞれのインスタンスが表す日付や時刻を簡単に書式化できる。

簡単な例を以下に示す。

```
import datetime

t_delta = datetime.timedelta(hours=9)
JST = datetime.timezone(t_delta, 'JST')
now = datetime.datetime.now(JST)
print(repr(now))
# 出力例
#datetime.datetime(2021, 11, 4, 17, 37, 28, 114417, tzinfo=datetime.timezone
#(datetime.timedelta(seconds=32400), 'JST'))
print(now)  # 2021-11-04 17:37:28.114417+09:00

# YYYYMMDDhhmmss 形式に書式化
d = now.strftime('%Y%m%d%H%M%S')
print(d)  # 20211104173728
```

最初にタイムゾーン付きで datetime クラスのオブジェクトを作成して、それを表示した後に、YYYYMMDDhhmmss 形式に書式化している。strftime メソッドには、「now.strftime('%Y%m%d%H%M%S')」のように「書式コード」を含んだフォーマット文字列を指定する。なお、print 関数の出力にある「2021-11-05 08:08:48.050345+09:00」のような表現が必要なら、「str(now)」のように str 関数で文字列化するだけでよい。

上の例では「%Y」「%m」「%d」「%H」「%M」「%S」という書式コードを使用している。これらと共に、本稿で紹介する書式コードを以下に示す（全ての書式コードについては Python のドキュメント「strftime() と strptime() の書式コード」を参照のこと）。

書式コード	説明	例
%Y	西暦（4桁表記。0埋め）	2021
%m	月（2桁表記。0埋め）	11
%d	日（2桁表記。0埋め）	04
%H	時（24時間制。2桁表記。0埋め）	17
%M	分（2桁表記。0埋め）	37
%S	秒（2桁表記。0埋め）	28
%y	西暦の下2桁（0埋め）	21
%I	AM／PMを表す文字列	PM
%x	日付をMM/DD/YY形式にしたもの	11/04/21
%X	時刻をhh:mm:ss形式にしたもの	17:37:28
%a	曜日の短縮形	Thu
%A	曜日	Thursday
%z	現在のタイムゾーンとUTC（協定世界時）とのオフセット	+0900
%Z	現在のタイムゾーン	JST

datetime モジュールの datetime ／ date クラス／ time クラスの strftime メソッドで使える書式コード（一部）

なお、これらの書式コードを使って、f 文字列／ format 関数／文字列の format メソッドで書式化を行うことも可能だ。以下に例を示す。

```
d = f'{now:%Y%m%d%H%M%S}'  # f 文字列
d = format(now, '%Y%m%d%H%M%S')  # format 関数
d = '{:%Y%m%d%H%M%S}'.format(now)  # 文字列の format メソッド
print(d)  # 20211104173728
```

日付と時刻の書式化

書式コードを含むフォーマット文字列には、書式コード以外も含めてよい。例えば、以下は先ほど取得した時刻を「YYYY/MM/DD hh:mm:ss」形式に書式化する例だ。

```
d = now.strftime('%Y/%m/%d %H:%M:%S')
print(d)  # 2021/11/04 17:37:28
```

日付を構成する要素の間にスラッシュを、時刻を構成する要素の間にコロンを入れて、それらを半角空白文字でつなぐようにしている。

なお、日付が「MM/DD/YY」という形式でよければ、今見たように複数の書式コードを使わずに「%x」という書式コードを書くだけでもよい。同様に、時刻が「hh:mm:ss」という形式であれば「%X」という書式コードを使える。

```
d = now.strftime('%x %X')
print(d)  # 11/04/21 17:37:28
```

この他にも、日付を書式化するコードとしては次のようなものが考えられる。

```
d = now.date().strftime('%Y/%m/%d')
print(d)  # 2021/11/04
```

この例では datetime クラスのインスタンスを基に date クラスのインスタンスを作成して、その strftime メソッドを呼び出している。このときには「%H」などの書式コードを使用すると例外を発生せずに「00」という値が得られることには注意されたい。

「MM/DD/YY」以外の形式に書式化したいのであれば、datetime.datetime.strftime メソッドと datetime.date.strftime メソッドのいずれを使うにせよ、「%Y」「%m」「%d」か、以下で紹介する「%y」を組み合わせて使うようにしよう。

同様に、時刻だけを書式化するのであれば、先ほど紹介した「%X」に加えて、datetime モジュールの time クラスが持つ strftime メソッドも使える。time クラスには日付情報が含まれないので、「%Y」「%m」「%d」などの書式コードを使うと、1900 年 01 月 01 日となるように書式化が行われることも覚えておこう。

```
t = now.time().strftime('%X')
print(t)  # 17:37:28
```

西暦を 2 桁表示

「%Y」は西暦を 4 桁で書式化するが、「%y」は西暦の下 2 桁を書式化する。

```
d = now.strftime('%Y/%m/%d %H:%M:%S')
print(d)  # 21/11/04 17:37:28
```

12 時間制+ AM ／ PM 表示

「%H」は 24 時間制として時の初期化を行うが、「%I」は 12 時間制として書式化する。このとき、AM ／ PM は「%p」で取り出せる。

```
d = now.strftime('%Y/%m/%d %I:%M(%p)')
print(d)  # 2021/11/04 05:37(PM)
```

曜日を含む日付

曜日は「%a」か「%A」の書式コードで取り出せる。前者は短縮形を、後者はフルスペルで取り出す。以下はその例だ。

```
d = now.strftime('%Y 年 %m 月 %d 日（%a）')
print(d)  # 2021 年 11 月 04 日（Thu）
```

筆者が Windows と macOS の Python 処理系で試したところ、「木」ではなく「Thu」と表示された。「木」と表示したいのであれば、以下のような変換テーブル（辞書）を作って、それを使うことが考えられる。

```
d_week = {'Sun': '日', 'Mon': '月', 'Tue': '火', 'Wed': '水',
          'Thu': '木', 'Fri': '金', 'Sat': '土'}
key = now.strftime('%a')
w = d_week[key]
d = now.strftime('%Y 年 %m 月 %d 日') + f'（{w}）' #f'{now:%Y 年 %m 月 %d 日}（{w}）'
print(d)  # 2021 年 11 月 04 日（木）
```

これは strftime メソッドにフォーマット文字列として「%a」のみを指定して、曜日（短縮形）を取得し、それを辞書のキーとして、漢字表記の曜日を取得するものだ。

あるいは、フォーマット文字列に「%u」を指定すると、月曜日を 1、日曜日を 7 とするインデックス（文字列）が得られるので、これを使って以下のようなコードとすることも考えられる（こちらは Python の処理系によっては使えない可能性がある）。

```
d_week = '日月火水木金土日'  # インデックス 0 の '日' は使用されない
idx = now.strftime('%u')  # '%u' では月曜日がインデックス '1' となる
w = d_week[int(idx)]
d = now.strftime('%Y 年 %m 月 %d 日') + f'（{w}）'
print(d)  # 2021 年 11 月 04 日（木）
```

タイムゾーン

「%z」は現在のタイムゾーンの UTC（協定世界時）に対するオフセットを書式化するのに使用する。「%Z」はタイムゾーンの名前を取得する。

```
d = now.strftime('%X%z(%Z)')
print(d)  # 17:37:28+0900(JST)
```

time モジュールを使用している場合

time モジュールには strftime 関数があり、この関数にこれまでに見てきたのと同様なフォーマット文字列と変換したい値（time.localtime 関数や time.gmtime 関数が返す struct_time クラスのインスタンス）の 2 つを渡すと、書式化した結果が返される。以下に例を示す。

strftime がメソッドではなく関数であり、引数に書式化対象の値を渡す以外の使い方は同じなので、コードの解説は省略する。ただし、struct_time クラスのインスタンスに関しては f 文字列や format 関数、文字列の format メソッドによる書式化はできないことには注意されたい。

```
import time

now = time.localtime()
print(now)
# 出力例
#time.struct_time(tm_year=2021, tm_mon=11, tm_mday=4, tm_hour=19, tm_min=40,
#tm_sec=1, tm_wday=3, tm_yday=308, tm_isdst=0)

d = time.strftime('%Y%m%d%H%M%S', now)  # YYYYMMDDhhmmss に書式化
d = time.strftime('%Y/%m/%d %H:%M:%S', now)  # YYYY/MM/DD hh:mm:ss に書式化
d = time.strftime('%x %X', now)  # YY/MM/DD hh:mm:ss に書式化
d = time.strftime('%Y/%m/%d %H:%M:%S', now)  # MM/DD/YY hh:mm:ss に書式化
d = time.strftime('%Y/%m/%d %I:%M(%p)', now)  # 12 時間制＋ AM ／ PM 表示
d = time.strftime('%Y 年 %m 月 %d 日（%a）', now)  # 曜日を含む日付

d_week = {'Sun': '日', 'Mon': '月', 'Tue': '火', 'Wed': '水',
          'Thu': '木', 'Fri': '金', 'Sat': '土'}
key = time.strftime('%a', now)
w = d_week[key]
d = time.strftime('%Y 年 %m 月 %d 日', now) + f'（{w}）'
print(d)  # 2021 年 11 月 04 日（木）
```

［解決！Python］datetime 型の日付や時刻と ISO 8601 形式の文字列とを相互変換するには

datetime クラスの isoformat メソッドと fromisoformat クラスメソッドを使って、datetime オブジェクトを ISO 8601 形式の文字列表現に変換する方法と、その逆を行う方法を紹介する。

かわさきしんじ，Deep Insider 編集部（2021 年 11 月 16 日）

```
import datetime

t_delta = datetime.timedelta(hours=9)
JST = datetime.timezone(t_delta, 'JST')
jst_datetime = datetime.datetime(2021, 11, 16, 12, 34, 56, tzinfo=JST)
naive_datetime = datetime.datetime.now()
print(repr(jst_datetime))
# 出力例：
#datetime.datetime(2021, 11, 16, 12, 34, 56, tzinfo=datetime.timezone(datetime.
#timedelta(seconds=32400), 'JST'))

print(f'jst: {jst_datetime}, timezone: {jst_datetime.tzname()}')  #
print(f'naive: {naive_datetime}, timezone: {naive_datetime.tzname()}')
# 出力例
#jst: 2021-11-16 12:34:56+09:00, timezone: JST
#naive: 2021-11-12 13:36:27.275913, timezone: None

# 日付と時刻を ISO フォーマットに書式化
jst_date = jst_datetime.isoformat()
naive_date = naive_datetime.isoformat()
print(jst_date)  # 2021-11-16T12:34:56+09:00
print(naive_date)  # 2021-11-12T13:36:27.275913

# 日付と時刻を ISO フォーマットで書式化（時刻の要素として含めるものを指定）
jst_dt = jst_datetime.isoformat(timespec='minutes')
print(jst_dt)
naive_dt = naive_datetime.isoformat(timespec='seconds')
print(naive_dt)

# ISO フォーマットから datetime.datetime インスタンスを生成
mydatetime = datetime.datetime.fromisoformat(jst_dt)
print(repr(mydatetime))
# 出力結果：
#datetime.datetime(2021, 11, 16, 12, 34, tzinfo=datetime.timezone(datetime.
#timedelta(seconds=32400)))

mydatetime = datetime.datetime.fromisoformat(naive_dt)
print(repr(mydatetime))
# 出力結果：
#datetime.datetime(2021, 11, 12, 13, 36, 27)
```

ISO 8601 で定められている「YYYY-MM-DDThh:mm:ss」形式の文字列表現への変換

Python に標準で付属の datetime モジュールには日付や時刻を扱う datetime クラスが含まれている。このクラスの isoformat（インスタンス）メソッドと fromisoformat クラスメソッドを使うと、ISO 8601 で定められている「YYYY-MM-DDThh:mm:ss」形式の文字列表現に変換したり、この形式の文字列から datetime クラスのオブジェクトを生成したりできる。

以下に簡単な例を示す。ここでは以下のようにして、datetime 型のオブジェクトを 2 つ作ったものとしよう。

```
import datetime

t_delta = datetime.timedelta(hours=9)
JST = datetime.timezone(t_delta, 'JST')
jst_datetime = datetime.datetime(2021, 11, 16, 12, 34, 56, tzinfo=JST)
naive_datetime = datetime.datetime.now()
print(repr(jst_datetime))
# 出力例:
#datetime.datetime(2021, 11, 16, 12, 34, 56, tzinfo=datetime.timezone(datetime.
#timedelta(seconds=32400), 'JST'))

print(f'jst: {jst_datetime}, timezone: {jst_datetime.tzname()}')  #
print(f'naive: {naive_datetime}, timezone: {naive_datetime.tzname()}')
# 出力例
#jst: 2021-11-16 12:34:56+09:00, timezone: JST
#naive: 2021-11-12 13:36:27.275913, timezone: None
```

1 つのタイムゾーンは JST で、もう 1 つはタイムゾーン情報を持たないナイーブなオブジェクトになっている。また、後者は datetime クラスの now クラスメソッドで現在時刻を取得しているので、マイクロ秒単位でデータが得られている。

ISO 8601（の拡張形式）では、日付と時刻はセパレーター「T」を挟んで「YYYY-MM-DDThh:mm:ss」のように表現される。isoformat メソッドを呼び出すことで、上記の 2 つの datetime オブジェクトをこの形式の文字列表現に変換できる。

```
jst_dt = jst_datetime.isoformat()
naive_dt = naive_datetime.isoformat()
print(jst_dt)  # 2021-11-16T12:34:56+09:00
print(naive_dt)  # 2021-11-12T13:36:27.275913
```

タイムゾーン情報を持つ jst_datetime オブジェクトを変換した結果では最後に「+09:00」が表示されている点と、マイクロ秒単位の値を保持している naive_datetime オブジェクトを変換した結果では秒の後にピリオドとマイクロ秒が追加されている点に注目しよう。マイクロ秒が 0 の場合には、1 つ目の例のようにその部分の表示は省略される。

isoformat メソッドでは、時刻をどの単位まで有効にするかを timespec 引数に指定できる。以下に例を示す。

```
jst_dt = jst_datetime.isoformat(timespec='minutes')
print(jst_dt)  # 2021-11-16T12:34+09:00
naive_dt = naive_datetime.isoformat(timespec='seconds')
print(naive_dt)  # 2021-11-12T13:36:27
```

1 つ目の例では、timespec 引数に 'minutes' を指定しているので、分までの時間（12:34）が表示されている。2 つ目の例では、timespec 引数に 'seconds' を指定しているので、秒までの時間（13:36:27）が得られている、つまりマイクロ秒単位のデータが切り捨てられているのが分かる。

datetime モジュールの date クラスと time クラスにも isoformat メソッドがある。date.isoformat メソッドは日付のみを変換し、time.isoformat メソッドは時刻のみを変換することを除けば datetime.isoformat メソッドと同様である（timespec 引数があるのは time.isoformat メソッドのみとなる）。

「YYYY-MM-DDThh:mm:ss」形式の文字列から datetime クラスのオブジェクトを生成する

上とは逆に ISO 8601 で定められている「YYYY-MM-DDThh:mm:ss」形式の文字列表現から、datetime クラスのオブジェクトを生成するには、datetime クラスの fromisoformat メソッドにその文字列表現を与えるとよい。

以下に例を示す。

```
mydatetime = datetime.datetime.fromisoformat(jst_dt)
print(repr(mydatetime))
# 出力結果：
#datetime.datetime(2021, 11, 16, 12, 34, tzinfo=datetime.timezone(datetime.
#timedelta(seconds=32400)))
print(mydatetime)  # 2021-11-16 12:34:00+09:00

mydatetime = datetime.datetime.fromisoformat(naive_dt)
print(repr(mydatetime))
# 出力結果：
#datetime.datetime(2021, 11, 12, 13, 36, 27)
```

1 つ目の例ではタイムゾーン情報付きの文字列（‘2021-11-16T12:34+09:00’）を与えているので、これを考慮した datetime オブジェクトが得られている。この文字列を得る際には、timespec 引数に ‘minutes’ を指定して分単位の時間を取得したので、これを基にして生成した datetime オブジェクトでは秒以下の単位はデフォルト値（1990 年 01 月 01 日 00:00:00）の日付／時刻の構成要素が流用される点にも注意されたい。

2 つ目の例ではタイムゾーン情報がないことと、秒単位の時間が得られるが、マイクロ秒単位については上と同様に 0 となっている。

このとき、時刻やセパレーターは省略できるが、日付を省略して時刻のみを渡すと例外となる点には注意しよう（上と同じく、足りない要素はデフォルトの値が流用される）。

```
mydatetime = datetime.datetime.fromisoformat('2022-01-01')  # OK
print(mydatetime)  # 2022-01-01 00:00:00
mydatetime = datetime.datetime.fromisoformat('12:34:56')  # ValueError
```

なお、このクラスメソッドはあくまで「YYYY-MM-DDThh:mm:ss」（とマイクロ秒やタイムゾーン情報）という文字列表現から、datetime クラスのインスタンスを生成するためのものであり、ISO 8601 が定めているその他の表現（経過時間など）に対応するものではない。

最後に、isoformat メソッドと同じく、date クラスと time クラスにも fromisoformat クラスメソッドがある。date.fromisoformat クラスメソッドは日付のみを受け取り、date クラスのインスタンスを生成する。一方、time.fromisoformat クラスメソッドは時刻のみを受け取り、time クラスのインスタンスを生成する。

```
mydate = datetime.date.fromisoformat('2021-11-22')
print(mydate)  # 2021-11-22
mydate = datetime.date.fromisoformat('2021-11-22T00:11:22')  # ValueError

mytime = datetime.time.fromisoformat('12:34:56')
print(mytime)  # 12:34:56
mytime = datetime.time.fromisoformat('2021-11-22T01:23:45')  # ValueError
```

［解決！ Python］日付や時刻の文字列表現を strptime メソッドで datetime 型のオブジェクトに変換するには

datetime クラスの strptime クラスメソッドを使って、文字列として表現されている日付や時刻から datetime 型のオブジェクトを作成する方法を紹介する。

かわさきしんじ，Deep Insider 編集部（2024 年 03 月 01 日）

* 本稿は 2021 年 11 月 24 日に公開された記事を Python 3.12.2 で動作確認したものです（確認日：2024 年 3 月 1 日）。

```
from datetime import datetime, timedelta, timezone

str_date = '2021/11/24 12:34:56'
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S')
print(mydate)  # 2021-11-24 12:34:56

str_date = '2021年11月24日 12時34分56秒'
mydate = datetime.strptime(str_date, '%Y年%m月%d日 %H時%M分%S秒')
print(mydate)  # 2021-11-24 12:34:56

# 西暦2桁
str_date = '21/11/24 12:34:56'
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S')
print(mydate)  # 2021-11-24 12:34:56

# 12時間制
str_date = '2021/11/24 01:23:45(PM)'
mydate = datetime.strptime(str_date, '%Y/%m/%d %I:%M:%S(%p)')
print(mydate)  # 021-11-24 13:23:45

# 曜日付き
str_date = '2021/11/24(Wed) 12:34:56'
mydate = datetime.strptime(str_date, '%Y/%m/%d(%a) %H:%M:%S')
idx = mydate.weekday()
day_of_week = '月火水木金土日'
print(f'{mydate} ({day_of_week[idx]})')  # 2021-11-24 12:34:56 (水)

# タイムゾーン付き
str_date = '2021/11/24 12:34:56+0900(JST)'
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S%z(%Z)')
print(f'{mydate} ({mydate.tzname()})')  # 2021-11-24 12:34:56+09:00 (JST)

# Windowsでは上のコードでは例外が発生するので自前で処理
str_date = str_date.split('+')[0]
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S')
print(f'{mydate} ({mydate.tzname()})')  # 2021-11-24 12:34:56 (None)
```

```
jst_tdelta = timedelta(hours=+9)
JST = timezone(jst_tdelta, 'JST')
mydate = mydate.astimezone(JST)
print(f'{mydate} ({mydate.tzname()})')  # 2021-11-24 12:34:56+09:00 (JST)
```

datetime.strptime クラスメソッドによる日付／時刻の文字列表現のパース

Python に標準で添付される datetime モジュールが提供する datetime クラスには strptime メソッドがある。このメソッドを使うと文字列として表現された日付や時刻から datetime 型のオブジェクトを作成できる。なお、ISO 8601 で規定されている「YYYY-MM-DDThh:mm:ss」形式で表現された文字列から datetime オブジェクトを作成する方法については「datetime 型の日付や時刻と、ISO 8601 形式の文字列とを相互変換するには」を参照されたい。

以下に単純な例を示す。

```
from datetime import datetime, timedelta, timezone

str_date = '2021/11/24 12:34:56'
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S')
print(mydate)  # 2021-11-24 12:34:56

str_date = '2021年11月24日 12時34分56秒'
mydate = datetime.strptime(str_date, '%Y年%m月%d日 %H時%M分%S秒')
print(mydate)  # 2021-11-24 12:34:56
```

strptime クラスメソッドには、datetime 型のオブジェクトに変換したい文字列表現と、それをパースする際の書式指定文字列を渡す。上の例では「2021/11/24 12:34:56」または「2021 年 11 月 24 日 12 時 34 分 56 秒」という日付と時刻の文字列表現をそれぞれ「%Y/%m/%d %H:%M:%S」および「%Y 年 %m 月 %d 日 %H 時 %M 分 %S 秒」という書式指定文字列を使ってパースして、datetime オブジェクトを取得している。

書式指定文字列には以下のような書式コードを記述できる（一部）。

書式コード	説明	例
%Y	西暦（4桁表記。0埋め）	2021
%m	月（2桁表記。0埋め）	11
%d	日（2桁表記。0埋め）	04
%H	時（24時間制。2桁表記。0埋め）	17
%M	分（2桁表記。0埋め）	37
%S	秒（2桁表記。0埋め）	28
%y	西暦の下2桁（0埋め）	21
%I	AM／PMを表す文字列	PM
%x	日付をMM/DD/YY形式にしたもの	11/04/21
%X	時刻をhh:mm:ss形式にしたもの	17:37:28
%a	曜日の短縮形	Thu
%A	曜日	Thursday

datetime モジュールの datetime クラスの strptime メソッドで使える書式コード（一部）

以下ではこれらを使って日付や時刻をパースする例を幾つか紹介する。

西暦が 2 桁で記述されている場合

「21/11/24 12:34:56」のように、西暦が 2 桁で表記されているときには、書式コードに「%Y」ではなく「%y」を使用する。

```
str_date = '21/11/24 12:34:56'
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S')
print(mydate)  # 2021-11-24 12:34:56
```

12 時間制で時刻が記述されている

時刻が 12 時間制で記述されている場合には、書式コードに「%H」ではなく「%I」を使用する。また、「%p」で午前／午後を示す文字列をパースできる。以下に例を示す。

```
str_date = '2021/11/24 01:23:45(PM)'
mydate = datetime.strptime(str_date, '%Y/%m/%d %I:%M:%S(%p)')
print(mydate)  # 021-11-24 13:23:45
```

この例では、時刻の部分が「01:23:45(PM)」となっている。「PM」とあるから、これは「午後 1 時 23 分 45 秒」のことだ。そして、これを「%I:%M:%S(%p)」でパースした結果を見ると、確かに「13:23:45」となっている。

曜日を含んだ文字列表現

文字列に曜日が含まれているときには、「%a」または「%A」でパースできる（前者は省略形、後者はフルスペル）。以下に例を示す。

```
str_date = '2021/11/24(Wed) 12:34:56'
mydate = datetime.strptime(str_date, '%Y/%m/%d(%a) %H:%M:%S')
idx = mydate.weekday()
day_of_week = '月火水木金土日'
print(f'{mydate} ({day_of_week[idx]})')  # 2021-11-24 12:34:56 (水)
```

この例では、生成された datetime オブジェクトの weekday メソッドを呼び出して、その戻り値（月曜日を 0 とする整数値）をインデックスに使い、月曜日から日曜日のいずれかに当てはまるかを確認している。

なお、これはパースをするだけで、実際の曜日は日付から自動的に設定されると思われる。例えば、以下は上の例で日付はそのまま、曜日だけを「Thu」に変更したものだ。

```
str_date = '2021/11/24(Thu) 12:34:56'
mydate = datetime.strptime(str_date, '%Y/%m/%d(%a) %H:%M:%S')
idx = mydate.weekday()
day_of_week = '月火水木金土日'
print(f'{mydate} ({day_of_week[idx]})')  # 2021-11-24 12:34:56 (水)
```

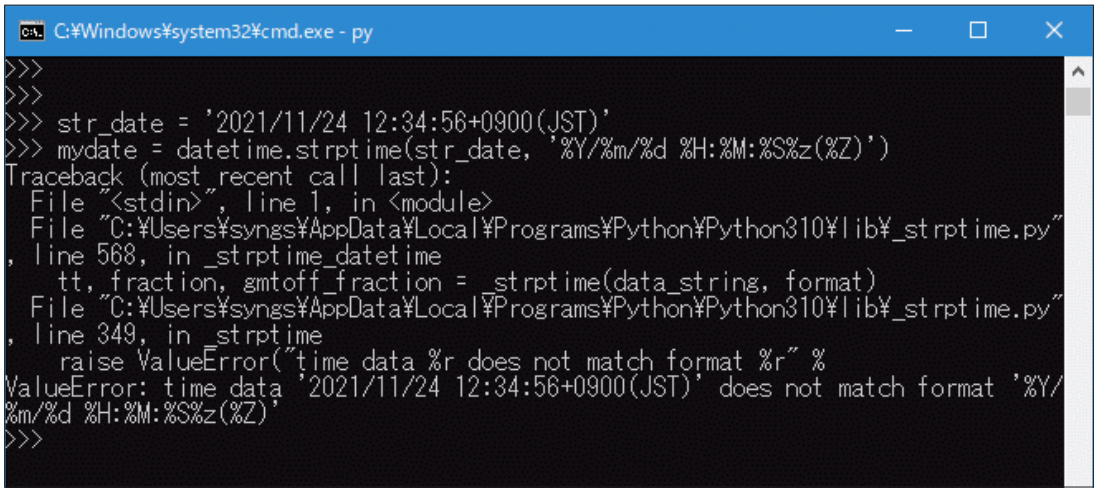
この場合でも出力（つまり作成された曜日情報を保持する datetime オブジェクトの値）は「2021-11-24 12:34:56 (水)」となる。このことからは「2021 年 11 月 24 日（水）」のような文字列を変換するとしたら、正規表現を利用するなどして、文字列から曜日部分を削除してしまうのが簡単な処理の方法と考えられる（コードは省略）。

タイムゾーン付き

文字列表現の中に UTC とのオフセット、タイムゾーン情報が含まれているときには「%z」（オフセット）と「%Z」（タイムゾーン名）が使用できる。以下に例を示す。

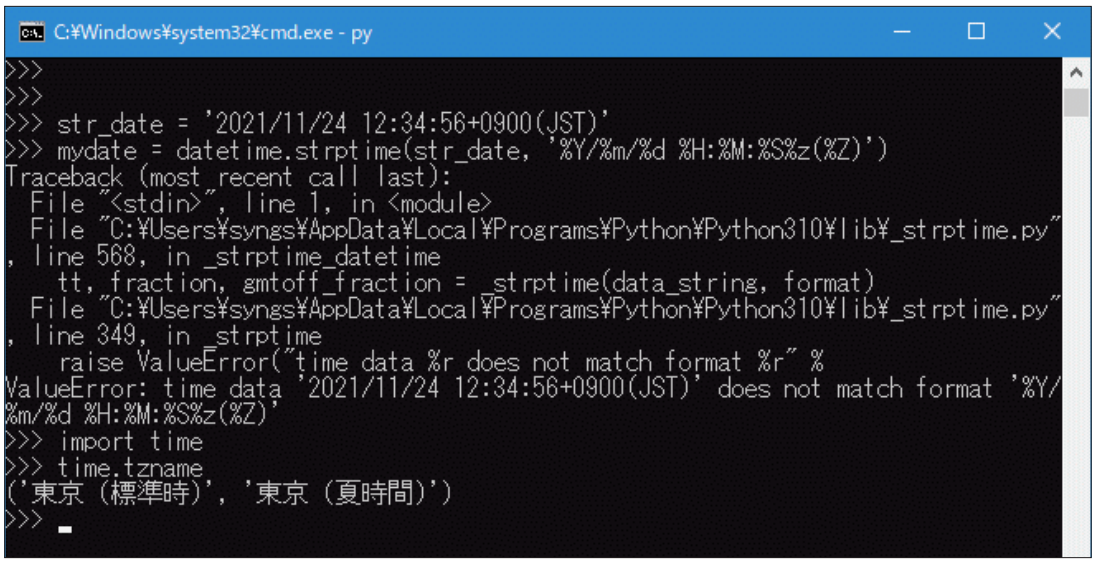
```
str_date = '2021/11/24 12:34:56+0900(JST)'
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S%z(%Z)')
print(f'{mydate} ({mydate.tzname()})')  # 2021-11-24 12:34:56+09:00 (JST)
```

ただし、このコードを筆者が Windows 環境にインストールした Python 処理系で実行すると次のように例外が発生する。



「2021/11/24 12:34:56+0900(JST)」が「%Y/%m/%d %H:%M:%S%z(%Z)」にマッチしないため例外となった

Python のドキュメントによれば、strptime クラスメソッドで文字列をパースする際には、「%Z」は「使用しているマシンのロケールによる time.tzname の任意の値」か「ハードコードされた値 UTC または GMT」のいずれかだけを受け入れるとある。そこで、Windows 環境で time.tzname の値を調べると次のようになった。



time.tzname の値

ここにJSTが含まれていないことから、筆者がWindowsにインストールしているPython処理系ではうまくパースができなかったということだ（実際、上で示された「東京 (標準時)」を文字列に含めるとパースに成功した）。

そのため、Windows で同等な処理を行うには、例えば、以下のように文字列から一度オフセット情報やタイムゾーン名を削除した状態で naive な datetime オブジェクトを作成し、その後でそれらを補うように自前で処理をする必要があるだろう。

```
str_date = str_date.split('+')[0]
mydate = datetime.strptime(str_date, '%Y/%m/%d %H:%M:%S')
print(f'{mydate} ({mydate.tzname()})')  # 2021-11-24 12:34:56 (None)

jst_tdelta = timedelta(hours=+9)
JST = timezone(jst_tdelta, 'JST')
mydate = mydate.astimezone(JST)
print(f'{mydate} ({mydate.tzname()})')  # 2021-11-24 12:34:56+09:00 (JST)
```

これは例なのでシンプルに書けているが、汎用性を持たせようとすると大変になるかもしれない。

［解決！Python］日付から曜日を求めるには

datetime モジュールの datetime クラスや date クラスの weekday ／ isoweekday ／ strftime メソッド、calendar モジュールの weekday 関数を使って日付から曜日を取得する方法を紹介する。

かわさきしんじ，Deep Insider 編集部（2024 年 06 月 18 日）

```
from datetime import datetime, date

day = date(2024, 6, 18)  # 2024年6月18日

# weekayメソッドは月曜日を0として、日曜日を6として返す
dow = day.weekday()  # day of the week
print(dow)  # 1

# 曜日名に変換する
day_name = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
            'Friday', 'Saturday', 'Sunday']
print(day_name[dow])  # Tuesday

day_name = '月火水木金土日'  # day_name = list('月火水木金土日')でも可
print(day_name[dow])  # 火

# isoweekdayメソッドは月曜日を1として、日曜日を7として返す
dowiso = day.isoweekday()  # day of the week (ISO形式)
print(dowiso)  # 2
print(day_name[dowiso - 1])  # 火

# datetime型のオブジェクトでも同様に曜日を求めることができる
dt = datetime(2024, 6, 18, 12, 34, 56)
dow = dt.weekday()
print(dow)  # 1
print(day_name[dow])  # 火

dowiso = dt.isoweekday()
print(dowiso)  # 2
print(day_name[dowiso - 1])  # 火

# strftimeメソッドを使って曜日名を取得する
dow = day.strftime('%a')  # 完全な曜日名の取得
print(dow)  # Tuesday

dow = dt.strftime('%a')  # 短縮形の曜日名の取得
print(dow)  # Tue

# ロケールを設定すれば日本語表記も取得できる
import locale

setting = locale.getlocale(locale.LC_TIME)
```


→目次に戻る

```
print(setting)  # (None, None)
locale.setlocale(locale.LC_TIME, 'ja_JP.UTF-8')  # 日付と時刻のロケールを設定

dow = day.strftime('%a')
print(dow)  # 火曜日

dow = day.strftime('%a')
print(dow)  # 火

locale.setlocale(locale.LC_TIME, setting)  # ロケール設定を元に戻す

# 環境変数に従ったロケール設定
locale.setlocale(locale.LC_ALL, '')

dow = day.strftime('%a')
print(dow)  # 火曜日

# calendar モジュールを使って曜日を求める
from calendar import weekday, day_name

dow = weekday(2024, 6, 18)
print(dow)  # 1
print(day_name[dow])  # 火曜日

# format 属性の値を変更して短縮形の曜日名を取得
day_name.format = '%a'
print(day_name[dow])  # 火
```

datetime モジュールを使って日付から曜日を取得する

Python に標準で付属する datetime モジュールは日付や時刻に関する機能を提供する。このうち、datetime.datetime クラスと datetime.date クラスには weekday メソッドと isoweekday メソッドが備わっている。これらのメソッドを使って、datetime オブジェクトや date オブジェクトが表している日付から対応する曜日を取得する方法を見ていこう。加えて、strftime メソッドを使用する方法や calendar モジュールの weekday 関数を使用する方法も紹介する。

以下は date クラスのインスタンスを作成して、その曜日を取得する例だ。

```
from datetime import datetime, date

day = date(2024, 6, 18)  # 2024 年 6 月 18 日

# weekay メソッドは月曜日を 0 として、日曜日を 6 として返す
dow = day.weekday()  # day of the week
print(dow)  # 1
```

weekday メソッドは呼び出しに使用した date オブジェクト（または datetime オブジェクト）が表す日付から対応する曜日を返す。戻り値は月曜日を 0、日曜日を 6 とする昇順の値である。上の例では date オブジェクトが表す日付である 2024 年 6 月 18 日は火曜日であることから、戻り値は 1 となる。

得られた数値から文字列表現の曜日を得るのであれば、次のように曜日を要素としたリストなどを作ればよい。

```
day_name = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
            'Friday', 'Saturday', 'Sunday']
print(day_name[dow])  # Tuesday

day_name = '月火水木金土日'  # day_name = list('月火水木金土日') でも可
print(day_name[dow])  # 火
```

最初の 2 行は英語表記の曜日を要素とするリストを作成して、取得した値から文字列表記の曜日を得ている。次の 2 行では、得られた数値を文字列の要素に対するインデックスとして使用している。

weekday メソッドは月曜日を 0 とするが、isoweekday メソッドは月曜日を 1 として日曜日を 7 とする昇順の値を返す。

```
dowiso = day.isoweekday()  # day of the week (ISO 形式)
print(dowiso)  # 2
print(day_name[dowiso - 1])  # 火
```

そのため、isoweekday メソッドの戻り値をリストや文字列のインデックスとして使うときには調整が必要になる点には注意しよう。

datetime クラスのインスタンスでも同じことをする例を以下に示しておく。

```
dt = datetime(2024, 6, 18, 12, 34, 56)
dow = dt.weekday()
print(dow)  # 1
print(day_name[dow])  # 火

dowiso = dt.isoweekday()
print(dowiso)  # 2
print(day_name[dowiso - 1])  # 火
```

strftime メソッドによる曜日名の取得

datetime クラス／ date クラス（加えて time クラス）には日付や時刻を書式化する strftime メソッドがあり、その書式を指定するディレクティブ「%A」と「%a」を使うことで日付から曜日を取得できる。「%A」は完全な曜日名を、「%a」は短縮形の曜日名を得るのに使用する。以下に例を示す。

```
dow = day.strftime('%a')  # 完全な曜日名の取得
print(dow)  # Tuesday

dow = dt.strftime('%a')  # 短縮形の曜日名の取得
print(dow)  # Tue
```

strftime メソッドに「%A」を渡すと「Tuesday」と完全な曜日名が得られ、「%a」を渡すと「Tue」と短縮形の曜日名が得られている点に注目されたい。

このとき、locale モジュールの setlocale 関数を使って日付／時刻のロケールを日本語（ja_JP.UTF-8）に設定することで日本語表記の曜日も取得可能だ。

```
import locale

setting = locale.getlocale(locale.LC_TIME)
print(setting)  # (None, None)
locale.setlocale(locale.LC_TIME, 'ja_JP.UTF-8')  # 日付と時刻のロケールを設定

dow = day.strftime('%a')
print(dow)  # 火曜日

dow = day.strftime('%a')
print(dow)  # 火

locale.setlocale(locale.LC_TIME, setting)  # ロケール設定を元に戻す
```

こちらでは「%A」により「火曜日」が、「%a」により「火」が得られている。

上の例では LC_TIME カテゴリを「ja_JP.UTF-8」に設定したが、「setlocale(locale.LC_ALL, '')」にこの関数を呼び出すと現在の環境変数の設定に従ってロケールが設定される（setlocale 関数はスレッドセーフではないので、マルチスレッドで何かをしようとするときには注意が必要だ）。

```
locale.setlocale(locale.LC_ALL, '')

dow = day.strftime('%a')
print(dow)  # 火曜日
```

calendar モジュールを使って曜日を求める

calendar モジュールの weekday 関数を使うことで日付から曜日を取得できる。この関数に年、月、日の 3 つの引数を渡すと曜日が返される（上で紹介した weekday メソッドと同様に月曜日が 0、日曜日が 6 となる）。以下に例を示す。

```
from calendar import weekday, day_name

dow = weekday(2024, 6, 18)
print(dow)  # 1
```

この関数は datetime クラスや date クラスのインスタンスは受け付けないが、単に一時的に特定の日が何曜日かを知りたいだけならこの関数を使うのが簡単だ。

なお、calendar モジュールには day_name 属性がある。これは現在のロケールに応じた曜日名を格納する配列として使える。「calendar.day_name[1]」のようにすれば火曜日の曜日名が得られるということだ。以下に例を示す（環境変数に対応したロケール設定を使用）。

```
locale.setlocale(locale.LC_ALL, '')

print(day_name[dow])  # 火曜日
```

day_name 属性には format 属性がある。この値を「%a」に変更すれば、短縮形の曜日名が得られる。

```
day_name.format = '%a'
print(day_name[dow])  # 火
```

この例では、format 属性の値を「%a」にしたので短縮形の「火」が得られた。

編集：@IT 編集部
発行：アイティメディア株式会社