Python - Data e Hora

Um programa Python pode lidar com data e hora de várias maneiras. A conversão entre formatos de data é uma tarefa comum para computadores. Os seguintes módulos na biblioteca padrão do Python tratam do processamento relacionado a data e hora -

- Módulo DataHora
- Módulo de tempo
- Módulo de calendário

O que são intervalos de tick

Os intervalos de tempo são números de ponto flutuante em unidades de segundos. Instantes específicos no tempo são expressos em segundos desde 12h00 de 1º de janeiro de 1970 (época).

Existe um módulo **de tempo** popular disponível em Python, que fornece funções para trabalhar com tempos e para converter entre representações. A função **time.time()** retorna a hora atual do sistema em ticks desde 12h00 de 1º de janeiro de 1970 (época).

Exemplo

```
import time # This is required to include time module.
ticks = time.time()
print ("Number of ticks since 12:00am, January 1, 1970:", ticks)
```

Isso produziria um resultado semelhante ao seguinte -

```
Number of ticks since 12:00am, January 1, 1970: 1681928297.5316436
```

A aritmética de datas é fácil de fazer com ticks. No entanto, datas anteriores à época não podem ser representadas desta forma. As datas num futuro distante também não podem ser representadas desta forma – o ponto de corte é em algum momento de 2038 para UNIX e Windows.

O que é Time Tuple?

Muitas das funções de tempo do Python tratam o tempo como uma tupla de 9 números, conforme mostrado abaixo -

Índice	Campo	Valores
0	Ano de 4 dígitos	2016
1	Mês	1 a 12
2	Dia	1 a 31
3	Hora	0 a 23
4	Minuto	0 a 59
5	Segundo	0 a 61 (60 ou 61 são segundos bissextos)
6	Dia da semana	0 a 6 (0 é segunda-feira)
7	Dia do ano	1 a 366 (dia juliano)
8	Horário de verão	-1, 0, 1, -1 significa que a biblioteca determina o horário de verão

Por exemplo,

```
>>>import time
>>> print (time.localtime())
```

Isso produziria uma **saída** como segue -

```
time.struct_time(tm_year=2023, tm_mon=4, tm_mday=19, tm_hour=23, tm_min=49, tm_sec=8
```

A tupla acima é equivalente à estrutura struct_time. Esta estrutura possui os seguintes atributos -

Índice	Atributos	Valores
0	tm_ano	2016
1	tm_mon	1 a 12
2	tm_mday	1 a 31
3	tm_hora	0 a 23
4	tm_min	0 a 59
5	tm_sec	0 a 61 (60 ou 61 são segundos bissextos)

6	tm_wday	0 a 6 (0 é segunda-feira)
7	tm_yday	1 a 366 (dia juliano)
8	tm_isdst	-1, 0, 1, -1 significa que a biblioteca determina o horário de verão

Obtendo a hora atual

Para converter um instante de tempo de **segundos** desde o valor de ponto flutuante da época em uma tupla de tempo, passe o valor de ponto flutuante para uma função (por exemplo, hora local) que retorne uma tupla de tempo com todos os nove itens válidos.

```
import time
localtime = time.localtime(time.time())
print ("Local current time :", localtime)
```

Isso produziria o seguinte resultado, que poderia ser formatado em qualquer outra forma apresentável -

```
Local current time : time.struct_time(tm_year=2023, tm_mon=4, tm_mday=19, tm_hour=23, tm
```

Obtendo a hora formatada

Você pode formatar a qualquer hora conforme sua necessidade, mas um método simples para obter a hora em um formato legível é **asctime()** -

```
import time

localtime = time.asctime( time.localtime(time.time()) )
print ("Local current time :", localtime)
```

Isso produziria a seguinte **saída** -

```
Local current time : Wed Apr 19 23:45:27 2023
```


Obtendo o calendário de um mês

O módulo de calendário oferece uma ampla variedade de métodos para brincar com calendários anuais e mensais. Aqui, imprimimos um calendário para um determinado mês (janeiro de 2008).

```
import calendar
cal = calendar.month(2023, 4)
print ("Here is the calendar:")
print (cal)
```

Isso produziria a seguinte **saída** -

```
Here is the calendar:
    April 2023
Mo Tu We Th Fr Sa Su
           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 29 30
```


O módulo de tempo

Existe um módulo **de tempo** popular disponível em Python, que fornece funções para trabalhar com tempos e para converter entre representações. Aqui está a lista de todos os métodos disponíveis.

Sr. Não.	Função com Descrição
1	**horário.altzone** O deslocamento do fuso horário local do horário de verão, em segundos a oeste do UTC, se houver algum definido. Isto é negativo se o fuso horário local do horário de verão estiver a leste do UTC (como na Europa Ocidental, incluindo o Reino Unido). Use isso apenas se a luz do dia for diferente de zero.
2	**time.asctime([tupletime])** Aceita uma tupla de tempo e retorna uma string legível de 24 caracteres, como 'Tue Dec 11 18:07:14 2008'.
3	**Rélogio de ponto( )**

	Retorna o tempo atual da CPU como um número de segundos em ponto flutuante. Para medir os custos computacionais de diferentes abordagens, o valor de time.clock é mais útil que o de time.time().
4	**tempo.ctime([seg])** Como asctime(localtime(secs)) e sem argumentos é como asctime()
5	**time.gmtime([seg])** Aceita um instante expresso em segundos desde a época e retorna uma tupla de tempo t com a hora UTC. Nota: t.tm_isdst é sempre 0
6	**hora.localtime([seg])** Aceita um instante expresso em segundos desde a época e retorna uma tupla de tempo t com a hora local (t.tm_isdst é 0 ou 1, dependendo se o horário de verão se aplica a segundos instantâneos pelas regras locais).
7	**time.mktime(tupletime)** Aceita um instante expresso como uma tupla de tempo na hora local e retorna um valor de ponto flutuante com o instante expresso em segundos desde a época.
8	**tempo.sleep(s)** Suspende o thread de chamada por segundos.
9	**time.strftime(fmt[,tupletime])** Aceita um instante expresso como uma tupla de tempo na hora local e retorna uma string representando o instante conforme especificado pela string fmt.
10	**time.strptime(str,fmt='%a %b %d %H:%M:%S %Y')** Analisa str de acordo com a string de formato fmt e retorna o instante no formato de tupla de tempo.
11	**tempo.tempo()** Retorna o instante de tempo atual, um número de segundos em ponto flutuante desde a época.
12	**hora.tzset()** Redefine as regras de conversão de tempo utilizadas pelas rotinas da biblioteca. A variável de ambiente TZ especifica como isso é feito.

Vamos examinar brevemente as funções.

Existem dois atributos importantes disponíveis no módulo de tempo. Eles são -

Sr. Não.	Atributo com Descrição
1	**hora.fuso horário** O atributo time.timezone é o deslocamento em segundos do fuso horário local (sem horário de verão) em relação ao UTC (>0 nas Américas; <=0 na maior parte

	da Europa, Ásia, África).
2	**hora.tzname** O atributo time.tzname é um par de strings dependentes de localidade, que são os nomes do fuso horário local sem e com horário de verão, respectivamente.

O módulo de calendário

O módulo de calendário fornece funções relacionadas ao calendário, incluindo funções para imprimir um calendário de texto para um determinado mês ou ano.

Por padrão, o calendário considera segunda-feira como o primeiro dia da semana e domingo como o último. Para mudar isso, chame a função **calendar.setfirstweekday()** .

Aqui está uma lista de funções disponíveis com o módulo **de calendário** -

Sr. Não.	Função com Descrição
1	**calendário.calendar(ano,w=2,l=1,c=6)** Retorna uma string multilinha com um calendário para o ano formatado em três colunas separadas por c espaços. w é a largura em caracteres de cada data; cada linha tem comprimento 21*w+18+2*c. l é o número de linhas de cada semana.
2	**calendário.primeirodia da semana()** Retorna a configuração atual do dia da semana que começa cada semana. Por padrão, quando o calendário é importado pela primeira vez, é 0, ou seja, segunda-feira.
3	**calendário.isleap(ano)** Retorna True se o ano for um ano bissexto; caso contrário, Falso.
4	**calendário.dias bissextos(y1,y2)** Retorna o número total de dias bissextos nos anos dentro do intervalo(y1,y2).
5	**calendário.mês(ano,mês,w=2,l=1)** Retorna uma string multilinha com um calendário para mês mês do ano ano, uma linha por semana mais duas linhas de cabeçalho. w é a largura em caracteres de cada data; cada linha tem comprimento 7*w+6. l é o número de linhas de cada semana.
6	**calendário.mêscalendar(ano,mês)** Retorna uma lista de listas de inteiros. Cada sublista denota uma semana. Os dias fora do mês, mês do ano, são definidos como 0; os dias do mês são definidos como

	o dia do mês, 1 e acima.
7	**calendário.mêsintervalo(ano,mês)** Retorna dois inteiros. O primeiro é o código do dia da semana para o primeiro dia do mês mês do ano ano; o segundo é o número de dias do mês. Os códigos dos dias da semana vão de 0 (segunda-feira) a 6 (domingo); os números dos meses são de 1 a 12.
8	**calendário.prcal(ano,w=2,l=1,c=6)** Como imprimir calendário.calendar(ano,w,l,c).
9	**calendário.prmês(ano,mês,w=2,l=1)** Como imprimir calendário.mês(ano,mês,w,l).
10	**calendário.setfirstweekday(dia da semana)** Define o primeiro dia de cada semana como código de dia da semana. Os códigos dos dias da semana vão de 0 (segunda-feira) a 6 (domingo).
11	**calendário.timegm(tupletime)** O inverso de time.gmtime: aceita um instante de tempo na forma de tupla de tempo e retorna o mesmo instante como um número de segundos em ponto flutuante desde a época.
12	**calendário.dia da semana (ano, mês, dia)** Retorna o código do dia da semana para a data especificada. Os códigos dos dias da semana vão de 0 (segunda-feira) a 6 (domingo); os números dos meses vão de 1 (janeiro) a 12 (dezembro).

módulo datahora

O módulo datetime do Python está incluído na biblioteca padrão. Consiste em classes que ajudam a manipular dados e dados de hora e realizar aritmética de data e hora.

Objetos de classes datetime são conscientes ou ingênuos. Se o objeto incluir informações de fuso horário, ele estará ciente e, caso contrário, será classificado como ingênuo. Um objeto da classe de data é ingênuo, enquanto os objetos de hora e data e hora estão cientes.

data

Um objeto de data representa uma data com ano, mês e dia. O atual calendário gregoriano é estendido indefinidamente em ambas as direções.

Sintaxe

```
datetime.date(year, month, day)
```

Os argumentos devem ser inteiros, nos seguintes intervalos -

- **ano** – MINYEAR <= ano <= MAXYEAR
- **mês** – 1 <= mês <= 12
- **dia** - 1 <= dia <= número de dias em um determinado mês e ano

Se o valor de qualquer argumento fora desses intervalos for fornecido, ValueError será gerado.

Exemplo

```
from datetime import date
date1 = date(2023, 4, 19)
print("Date:", date1)
date2 = date(2023, 4, 31)
```

Ele produzirá a seguinte **saída** -

```
Date: 2023-04-19
Traceback (most recent call last):
 File "C:\Python311\hello.py", line 8, in <module>
  date2 = date(2023, 4, 31)
ValueError: day is out of range for month
```

atributos de classe de data

- **date.min** - A data representável mais antiga, date(MINYEAR, 1, 1).
- **date.max** - A última data representável, date(MAXYEAR, 12, 31).
- **date.resolution** - A menor diferença possível entre objetos de data não iguais.
- **date.year** - Entre MINYEAR e MAXYEAR inclusive.
- **date.month** - Entre 1 e 12 inclusive.
- **date.day** - Entre 1 e o número de dias de um determinado mês de um determinado ano.

Exemplo

```
from datetime import date

# Getting min date
mindate = date.min
print("Minimum Date:", mindate)
```

```
# Getting max date
maxdate = date.max
print("Maximum Date:", maxdate)

Date1 = date(2023, 4, 20)
print("Year:", Date1.year)
print("Month:", Date1.month)
print("Day:", Date1.day)
```

Ele produzirá a seguinte **saída** -

```
Minimum Date: 0001-01-01
Maximum Date: 9999-12-31
Year: 2023
Month: 4
Day: 20
```

Métodos de classe na classe de data

- **today()** - Retorna a data local atual.
- **fromtimestamp(timestamp)** - Retorna a data local correspondente ao timestamp POSIX, como é retornado por time.time().
- **fromordinal(ordinal)** - Retorna a data correspondente ao proléptico ordinal gregoriano, onde 1º de janeiro do ano 1 possui ordinal 1.
- **fromisoformat(date_string)** - Retorna uma data correspondente a uma date_string fornecida em qualquer formato ISO 8601 válido, exceto datas ordinais

Exemplo

```
from datetime import date

print (date.today())
d1=date.fromisoformat('2023-04-20')
print (d1)
d2=date.fromisoformat('20230420')
print (d2)
d3=date.fromisoformat('2023-W16-4')
print (d3)
```

Ele produzirá a seguinte **saída** -

```
2023-04-20
2023-04-20
```

```
2023-04-20
2023-04-20
```

Métodos de instância na classe de data

- **replace()** - Retorna uma data substituindo atributos especificados por novos valores por meio de argumentos de palavra-chave especificados.
- **timetuple()** - Retorna um time.struct_time como o retornado por time.localtime().
- **toordinal()** - Retorna o ordinal gregoriano proléptico da data, onde 1º de janeiro do ano 1 tem o ordinal 1. Para qualquer objeto de data d, date.fromordinal(d.toordinal()) == d.
- **weekday()** - Retorna o dia da semana como um número inteiro, onde segunda-feira é 0 e domingo é 6.
- **isoweekday()** - Retorna o dia da semana como um número inteiro, onde segunda-feira é 1 e domingo é 7.
- **isocalendar()** - Retorna um objeto tupla nomeado com três componentes: ano, semana e dia da semana.
- **isoformat()** - Retorna uma string representando a data no formato ISO 8601, AAAA-MM-DD:
- **__str__()** - Para uma data d, str(d) é equivalente a d.isoformat()
- **ctime()** - Retorna uma string representando a data:
- **strftime(format)** - Retorna uma string representando a data, controlada por uma string de formato explícito.
- **__format__(format)** - O mesmo que date.strftime().

Exemplo

```
from datetime import date
d = date.fromordinal(738630) # 738630th day after 1. 1. 0001
print (d)
print (d.timetuple())
# Methods related to formatting string output
print (d.isoformat())
print (d.strftime("%d/%m/%y"))
print (d.strftime("%A %d. %B %Y"))
print (d.ctime())

print ('The {1} is {0:%d}, the {2} is {0:%B}.'.format(d, "day", "month"))

# Methods for to extracting 'components' under different calendars
t = d.timetuple()
```

```
for i in t:
   print(i)

ic = d.isocalendar()
for i in ic:
   print(i)

# A date object is immutable; all operations produce a new object
print (d.replace(month=5))
```

Ele produzirá a seguinte **saída** -

```
2023-04-20
time.struct_time(tm_year=2023, tm_mon=4, tm_mday=20, tm_hour=0, tm_min=0, tm_sec=0, t
2023-04-20
20/04/23
Thursday 20. April 2023
Thu Apr 20 00:00:00 2023
The day is 20, the month is April.
2023
4
20
0
0
0
3
110
-1
2023
16
4
2023-05-20
```

tempo

Uma classe de hora do objeto representa a hora local do dia. É independente de qualquer dia específico. Se o objeto contém os detalhes do tzinfo, é o objeto consciente. Se for None, então o objeto de tempo é o objeto ingênuo.

Sintaxe

```
datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None)
```

Todos os argumentos são opcionais. tzinfo pode ser None ou uma instância de uma subclasse tzinfo. Os argumentos restantes devem ser inteiros nos seguintes intervalos -

- **hora** - 0 <= hora <24,
- **minuto** - 0 <= minuto <60,
- **segundo** - 0 <= segundo <60,
- **microssegundo** - 0 <= microssegundo <1000000

Se algum dos argumentos estiver fora desses intervalos, ValueError será gerado.

Exemplo

```
from datetime import time

time1 = time(8, 14, 36)
print("Time:", time1)

time2 = time(minute = 12)
print("time", time2)

time3 = time()
print("time", time3)

time4 = time(hour = 26)
```

Ele produzirá a seguinte **saída** -

```
Time: 08:14:36
time 00:12:00
time 00:00:00
Traceback (most recent call last):
  File "/home/cg/root/64b912f27faef/main.py", line 12, in
    time4 = time(hour = 26)
ValueError: hour must be in 0..23
```

Atributos de classe

- **time.min** - O primeiro horário representável, time(0, 0, 0, 0).
- **time.max** - O último horário representável, time(23, 59, 59, 999999).
- **time.resolution** - A menor diferença possível entre objetos de tempo não iguais.

Exemplo

```
from datetime import time
print(time.min)
print(time.max)
print (time.resolution)
```

Ele produzirá a seguinte **saída** -

```
00:00:00
23:59:59.999999
0:00:00.000001
```

Atributos de instância

- **time.hour** - No intervalo (24)
- **time.minuto** - Dentro do intervalo (60)
- **time.second** - No intervalo (60)
- **time.microsecond** - No intervalo (1000000)
- **time.tzinfo** - o argumento tzinfo para o construtor de tempo, ou None.

Exemplo

```
from datetime import time
t = time(8,23,45,5000)
print(t.hour)
print(t.minute)
print (t.second)
print (t.microsecond)
```

Ele produzirá a seguinte **saída** -

```
8
23
455000
```

Métodos de instância

- **replace()** - Retorna um horário com o mesmo valor, exceto para aqueles atributos que recebem novos valores por quaisquer argumentos de palavra-chave especificados.

- **isoformat()** - Retorna uma string representando a hora no formato ISO 8601
- **__str__()** - Por um tempo t, str(t) é equivalente a t.isoformat().
- **strftime(format)** - Retorna uma string representando a hora, controlada por uma string de formato explícito.
- **__format__(format)** - O mesmo que time.strftime().
- **utcoffset()** - Se tzinfo for None, retorna None, caso contrário retorna self.tzinfo.utcoffset(None),
- **dst()** - Se tzinfo for None, retorna None, caso contrário retorna self.tzinfo.dst(None),
- **tzname()** - Se tzinfo for None, retorna None, caso contrário retorna self.tzinfo.tzname(None) ou gera uma exceção

data hora

Um objeto da classe datetime contém as informações de data e hora juntas. Assume o atual calendário gregoriano estendido em ambas as direções; como um objeto de tempo, e há exatamente 3600*24 segundos em cada dia.

Sintaxe

```
datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo
```

Os argumentos ano, mês e dia são obrigatórios.

- **ano** – MINYEAR <= ano <= MAXYEAR,
- **mês** – 1 <= mês <= 12,
- **dia** - 1 <= dia <= número de dias em um determinado mês e ano,
- **hora** - 0 <= hora <24,
- **minuto** - 0 <= minuto <60,
- **segundo** –0 <= segundo <60,
- **microssegundo** - 0 <= microssegundo <1000000,
- **dobrar** - em [0, 1].

Se qualquer argumento fora dos intervalos for fornecido, ValueError será gerado.

Exemplo

```
from datetime import datetime
dt = datetime(2023, 4, 20)
print(dt)
```

```
dt = datetime(2023, 4, 20, 11, 6, 32, 5000)
print(dt)
```

Ele produzirá a seguinte **saída** -

```
2023-04-20 00:00:00
2023-04-20 11:06:32.005000
```

Atributos de classe

- **datetime.min** - A data e hora representável mais antiga, datetime(MINYEAR, 1, 1, tzinfo=None).
- **datetime.max** - A última data e hora representável, datetime(MAXYEAR, 12, 31, 23, 59, 59, 999999, tzinfo=None).
- **datetime.resolution** - A menor diferença possível entre objetos de data e hora diferentes, timedelta (microssegundos = 1).

Exemplo

```
from datetime import datetime
min = datetime.min
print("Min DateTime ", min)

max = datetime.max
print("Max DateTime ", max)
```

Ele produzirá a seguinte **saída** -

```
Min DateTime 0001-01-01 00:00:00
Max DateTime 9999-12-31 23:59:59.999999
```

Atributos da instância

- **datetime.year** - Entre MINYEAR e MAXYEAR inclusive.
- **datetime.month** - Entre 1 e 12 inclusive.
- **datetime.day** - Entre 1 e o número de dias de um determinado mês de um determinado ano.
- **datetime.hour** - No intervalo (24)
- **datetime.minuto** - No intervalo (60)

- **datetime.second** - No intervalo (60)
- **datetime.microsecond** - No intervalo (1000000).
- **datetime.tzinfo** - O objeto passado como argumento tzinfo para o construtor datetime, ou None se nenhum foi passado.
- **datetime.fold** - Em [0, 1]. Usado para eliminar a ambiguidade dos tempos de parede durante um intervalo repetido.

Exemplo

```
from datetime import datetime
dt = datetime.now()

print("Day: ", dt.day)
print("Month: ", dt.month)
print("Year: ", dt.year)
print("Hour: ", dt.hour)
print("Minute: ", dt.minute)
print("Second: ", dt.second)
```

Ele produzirá a seguinte **saída** -

```
Day: 20
Month: 4
Year: 2023
Hour: 15
Minute: 5
Second: 52
```

Métodos de classe

- **today()** - Retorna a data e hora local atual, com tzinfo None.
- **now(tz=None)** - Retorna a data e hora local atual.
- **utcnow()** - Retorna a data e hora UTC atuais, com tzinfo None.
- **utcfromtimestamp(timestamp)** - Retorna o datetime UTC correspondente ao timestamp POSIX, com tzinfo None
- **fromtimestamp(timestamp, timezone.utc)** - Nas plataformas compatíveis com POSIX, é equivalente adatetime(1970, 1, 1, tzinfo=timezone.utc) + timedelta(seconds=timestamp)
- **fromordinal(ordinal)** - Retorna a data e hora correspondente ao proléptico ordinal gregoriano, onde 1º de janeiro do ano 1 tem ordinal 1.

- **fromisoformat(date_string)** - Retorna uma data e hora correspondente a uma date_string em qualquer formato ISO 8601 válido.

Métodos de instância

- **date()** - Retorna o objeto de data com o mesmo ano, mês e dia.
- **time()** - Retorna o objeto de tempo com a mesma hora, minuto, segundo, microssegundo e dobra.
- **timetz()** - Retorna o objeto de tempo com os mesmos atributos de hora, minuto, segundo, microssegundo, dobra e tzinfo. Veja também o método time().
- **replace()** - Retorna uma data e hora com os mesmos atributos, exceto para aqueles atributos que recebem novos valores por quaisquer argumentos de palavra-chave especificados.
- **astimezone(tz=None)** - Retorna um objeto datetime com o novo atributo tzinfo tz
- **utcoffset()** - Se tzinfo for None, retorna None, caso contrário retorna self.tzinfo.utcoffset(self)
- **dst()** - Se tzinfo for None, retorna None, caso contrário retorna self.tzinfo.dst(self)
- **tzname()** - Se tzinfo for None, retorna None, caso contrário retorna self.tzinfo.tzname(self)
- **timetuple()** - Retorna um time.struct_time como o retornado por time.localtime().
- **atetime.toordinal()** - Retorna o ordinal gregoriano proléptico da data.
- **timestamp()** - Retorna o carimbo de data / hora POSIX correspondente à instância de data e hora.
- **isoweekday()** - Retorna o dia da semana como um número inteiro, onde segunda-feira é 1, domingo é 7.
- **isocalendar()** - Retorna uma tupla nomeada com três componentes: ano, semana e dia da semana.
- **isoformat(sep='T', timespec='auto')** - Retorna uma string representando a data e hora no formato ISO 8601
- **__str__()** - Para uma instância de data e hora d, str(d) é equivalente a d.isoformat(' ').
- **ctime()** - Retorna uma string representando a data e hora:
- **strftime(format)** - Retorna uma string representando a data e hora, controlada por uma string de formato explícito.
- **__format__(format)** - O mesmo que strftime().

Exemplo

```
from datetime import datetime, date, time, timezone

# Using datetime.combine()
d = date(2022, 4, 20)
t = time(12, 30)
datetime.combine(d, t)

# Using datetime.now()
d = datetime.now()
print (d)

# Using datetime.strptime()
dt = datetime.strptime("23/04/20 16:30", "%d/%m/%y %H:%M")

# Using datetime.timetuple() to get tuple of all attributes
tt = dt.timetuple()
for it in tt:
   print(it)

# Date in ISO format
ic = dt.isocalendar()
for it in ic:
   print(it)
```

Ele produzirá a seguinte **saída** -

```
2023-04-20 15:12:49.816343
2020
4
23
16
30
0
3
114
-1
2020
17
4
```

delta do tempo

O objeto timedelta representa a duração entre duas datas ou dois objetos de tempo.

Sintaxe

```
datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hour
```

Internamente, os atributos são armazenados em dias, segundos e microssegundos. Outros argumentos são convertidos para essas unidades -

- Um milissegundo é convertido em 1000 microssegundos.
- Um minuto é convertido em 60 segundos.
- Uma hora é convertida em 3600 segundos.
- Uma semana é convertida em 7 dias.

Enquanto dias, segundos e microssegundos são normalizados para que a representação seja única.

Exemplo

O exemplo a seguir mostra que o Python armazena internamente apenas dias, segundos e microssegundos.

```
from datetime import timedelta
delta = timedelta(
   days=100,
   seconds=27,
   microseconds=10,
   milliseconds=29000,
   minutes=5,
   hours=12,
   weeks=2
)
# Only days, seconds, and microseconds remain
print (delta)
```

Ele produzirá a seguinte **saída** -

```
114 days, 12:05:56.000010
```

Exemplo

O exemplo a seguir mostra como adicionar o objeto timedelta a um objeto datetime.

```
from datetime import datetime, timedelta

date1 = datetime.now()
```

```
date2= date1+timedelta(days = 4)
print("Date after 4 days:", date2)

date3 = date1-timedelta(15)
print("Date before 15 days:", date3)
```

Ele produzirá a seguinte **saída** -

```
Date after 4 days: 2023-04-24 18:05:39.509905
Date before 15 days: 2023-04-05 18:05:39.509905
```

Atributos de classe

- **timedelta.min** - O objeto timedelta mais negativo, timedelta(-999999999).
- **timedelta.max** - O objeto timedelta mais positivo, timedelta(dias=999999999, horas=23, minutos=59, segundos=59, microssegundos=999999).
- **timedelta.resolution** - A menor diferença possível entre objetos timedelta não iguais, timedelta(microssegundos=1)

Exemplo

```
from datetime import timedelta

# Getting minimum value
min = timedelta.min
print("Minimum value:", min)

max = timedelta.max
print("Maximum value", max)
```

Ele produzirá a seguinte **saída** -

```
Minimum value: -999999999 days, 0:00:00
Maximum value 999999999 days, 23:59:59.999999
```

Atributos da instância

Como apenas dia, segundo e microssegundos são armazenados internamente, esses são os únicos atributos de instância para um objeto **timedelta** .

- **dias** - Entre -999999999 e 999999999 inclusive
- **segundos** - Entre 0 e 86399 inclusive

- **microssegundos** - Entre 0 e 999999 inclusive

Métodos de instância

timedelta.total_seconds() - Retorna o número total de segundos contidos na duração.

Exemplo

```
from datetime import timedelta
year = timedelta(days=365)
years = 5 * year
print (years)
print (years.days // 365)
646
year_1 = years // 5
print(year_1.days)
```

Ele produzirá a seguinte **saída** -

```
1825 days, 0:00:00
5
365
```