Chapter 11

How to work with dates and times

Objectives

Applied

1. Code, test, and debug programs that work with dates and times. That includes:

   creating date, time, and datetime objects
   formatting dates and times
   working with spans of time
   comparing datetime objects

Objectives (cont.)

Knowledge

1. Describe the three ways to create date, time, and datetime objects: with methods, with constructors, and by parsing.
2. Distinguish between aware and naïve date, time, and datetime objects.
3. Describe the way spans of times are used when working with dates and times.
4. Describe the way you compare date and time objects.

Two methods of the date and datetime classes

```
date.today()
datetime.now()
```

The constructors for creating date/time objects

```
date(year, month, day)
time([hour][, min][, sec][, microsec])
datetime(year, month, day [, hour][, min][, sec][, microsec])
```

Code that imports the date, time, and datetime classes

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

Another way to import the date, time, and datetime classes

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

Code that uses methods to create date and datetime objects

```
invoice_date = date.today()          # Current date
invoice_date = datetime.now()        # Current date and time
```

Code that uses constructors to create naïve date/time objects

```
halloween = date(1988, 10, 31)       # 10/31/1988
meeting = time(14, 30)               # 2:30 PM
appointment = datetime(2016, 10, 28, 14, 30)
                                     # 10/28/2016 2:30 PM
entry_time = datetime(2017, 10, 28, 14, 32, 48)
                                     # 10/28/2017 2:32:48 PM
```

The strptime() method of the datetime class

```
datetime.strptime(datetime_str, format_str)
```

Common format string codes

Code	Description
`%d`	Day of month as a number
`%m`	Month as a number
`%y`	2-digit year
`%Y`	4-digit year
`%H`	Hour of day in 24 hour format
`%M`	Minute as number
`%S`	Second as number

Code that creates datetime objects using format strings

```
halloween = datetime.strptime("10/31/1988", "%m/%d/%Y")
halloween = datetime.strptime("31-10-1988", "%d-%m-%Y")
halloween = datetime.strptime("1988-10-31", "%Y-%m-%d")
halloween = datetime.strptime("10/31/1988 22:30",
                              "%m/%d/%Y %H:%M")
```

Code that gets a date from the user and prints it to the console

```
date_str = input("Enter date of birth (MM/DD/YYYY): ")
birth_date = datetime.strptime(date_str, "%m/%d/%Y")
print("Date of birth:", birth_date)
```

The console

```
Enter date of birth (MM/DD/YYYY): 2/4/1968
Date of birth: 1968-02-04 00:00:00
```

The strftime() method of all date/time objects

```
strftime(format_str)
```

Some commonly used formatting codes

Code	Description	Example
`%a`	Abbreviated weekday name	`Sat`
`%A`	Full weekday name	`Saturday`
`%b`	Abbreviated month name	`Oct`
`%B`	Full month name	`October`
`%d`	Zero-padded day of month as a number	`01`
`%m`	Zero-padded month as a number	`01`
`%Y`	4-digit year	`1977`
`%y`	2-digit year	`77`
`%H`	Hour of day in 24-hour format	`13`
`%I`	Hour of day in 12-hour format	`01`
`%M`	Minute as number	`59`
`%S`	Second as number	`59`
`%p`	AM/PM specifier	`AM`
`%f`	Microsecond	`0153219`

Code that creates a datetime object that has a date and time

```
halloween = datetime(1988, 10, 31, 22, 48)
```

Code that uses format codes to specify format

```
halloween.strftime("%Y-%m-%d")         # 1988-10-31
halloween.strftime("%m/%d/%Y")         # 10/31/1988
halloween.strftime("%m/%d/%y")         # 10/31/88
halloween.strftime("%B %d, %Y (%A)")   # October 31, 1988 (Monday)
halloween.strftime("%B %d, %H:%M")     # October 31, 22:48
halloween.strftime("%B %d, %I:%M %p") # October 31, 10:48 PM
```

Code that formats for locale

```
halloween.strftime("%c")               # Mon Oct 31 22:48:00 1988
halloween.strftime("%x")               # 10/31/88
```

The constructor for a timedelta object

```
timedelta([days][, seconds][, microseconds][, milliseconds]
          [, minutes][, hours][, weeks])
```

Code that imports the timedelta class

```
from datetime import timedelta
```

Code that creates time spans

```
three_weeks = timedelta(weeks=3)
two_hours_thirty_minues = timedelta(hours=2, minutes=30)
time_span = timedelta(weeks=2, days=3, hours=8, minutes=14)
```

Code that adds and subtracts a span of time

```
three_weeks_from_today = date.today() + timedelta(weeks=3)
three_weeks_ago = date.today() - timedelta(weeks=3)
three_hours_from_now = datetime.now() + timedelta(hours=3)
```

Three attributes and a method of a timedelta object

```
days
seconds
microseconds
total_seconds()
```

Code that gets the time span between two dates

```
halloween = datetime(2017, 10, 31)
time_span = halloween - datetime.now()     # Span between dates

days = time_span.days                      # Days
seconds = time_span.seconds                # Seconds
microseconds = time_span.microseconds      # Microseconds
total_seconds = time_span.total_seconds() # Seconds.microseconds

print(days, "days",
      seconds, "seconds, and",
      microseconds, "microseconds.")
print(total_seconds, "seconds and microseconds.")
```

The console

```
405 days, 34951 seconds, and 289282 microseconds.
35026951.289282 seconds and microseconds.
```

Another way to get the span between two dates

```
days = (halloween - datetime.now()).days     # Number of days
```

The user interface for the Invoice Due Date program

```
The Invoice Due Date program

Enter the invoice date (MM/DD/YY): 7/14/17

Invoice Date: July 14, 2017
Due Date:     August 13, 2017
Current Date: September 12, 2017

This invoice is 30 day(s) overdue.

Continue? (y/n):
```

The code

```
from datetime import datetime, timedelta

def get_invoice_date():
    invoice_date_str = input(
        "Enter the invoice date (MM/DD/YY): ")
    invoice_date = datetime.strptime(invoice_date_str,
                                     "%m/%d/%y")
    return invoice_date

def main():
    print("The Invoice Due Date program")
    print()

    while True:
        invoice_date = get_invoice_date()
        print()

        # calculate due date and days overdue
        due_date = invoice_date + timedelta(days=30)
        current_date = datetime.now()
        days_overdue = (current_date - due_date).days
```

The code (cont.)

```
        # display results
        print("Invoice Date: " +
              invoice_date.strftime("%B %d, %Y"))
        print("Due Date:     " +
              due_date.strftime("%B %d, %Y"))
        print("Current Date: " +
               current_date.strftime("%B %d, %Y"))
        if days_overdue > 0:
            print("This invoice is", days_overdue,
                  "day(s) overdue.")
        else:
            days_due = days_overdue * -1
            print("This invoice is due in", days_due, "day(s).")
        print()

        # ask if user wants to continue
        result = input("Continue? (y/n): ")
        print()
        if result.lower() != "y":
            print("Bye!")
            break

if __name__ == "__main__":
    main()
```

The user interface for the Timer program

```
The Timer program

Press Enter to start...
Start time: 2016-09-16 15:24:08.633025

Press Enter to stop...
Stop time:  2016-09-16 15:24:25.320605

ELAPSED TIME
Time: 00:00:16.687580
```

The code

```
from datetime import datetime, time

def main():
    print("The Timer program")
    print()

    # start timer
    input("Press Enter to start...")
    start_time = datetime.now()
    print("Start time:", start_time)
    print()

    # stop timer
    input("Press Enter to stop...")
    stop_time = datetime.now()
    print("Stop time: ", stop_time)
    print()

    # calculate elapsed time
    elapsed_time = stop_time - start_time
    days = elapsed_time.days
    minutes = elapsed_time.seconds // 60
    seconds = elapsed_time.seconds % 60
    microseconds = elapsed_time.microseconds
```

The code (cont.)

```
        # calculate hours and minutes
        hours = minutes // 60
        minutes = minutes % 60

        # create time object
        time_object = time(hours, minutes, seconds, microseconds)

        # display results
        print("ELAPSED TIME")
        if days > 0:
            print("Days:", days)
        print("Time:", time_object)

if __name__ == "__main__":
    main()
```

Attributes that return the parts of a date/time object

```
year
month
day
hour
minute
second
microsecond
```

Code that gets the parts of a datetime object

```
halloween = datetime(1988, 10, 31, 14, 32, 30)

year = halloween.year                   # 1988
month = halloween.month                 # 10
day = halloween.day                     # 31
hour = halloween.hour                   # 14
minute = halloween.minute               # 32
second = halloween.second               # 30
microsecond = halloween.microsecond     # 0
```

Code that checks parts of a date object

```
today = date.today()
if (today.month == 10 and today.day == 31):
    print("Happy Halloween!")
else:
    print("Dang, it's not Halloween today.")
```

Code that creates a new date based on the current date

```
today = date.today()
last_year = today.year - 1
one_year_ago_today = date(last_year, today.month,
                          today.day)
```

Code that creates a new date and time based on the current date and time

```
now = datetime.now()
this_time_next_year = datetime(now.year + 1, now.month,
                               now.day, now.hour,
                               now.minute)
```

Code that compares two date objects

```
today = date.today()
halloween = date(2017, 10, 31)

if today > halloween:
    print("Halloween 2017 has come and gone.")
elif today < halloween:
    print("Halloween 2017 is coming soon.")
elif today == halloween:
    print("Happy Halloween 2017!")
```

Code that prints the number of days until Halloween

```
today = date.today()
halloween = date(today.year, 10, 31)

if today > halloween:
    next_year = today.year + 1
    halloween = date(next_year, 10, 31)
days_until = (halloween - today).days
print(days_until, "day(s) until Halloween.")
```

Code that compares two datetime objects

```
meeting_start = datetime(2017, 12, 2, 9, 30)
meeting_end = meeting_start + timedelta(hours=1)
now = datetime.now()

if now > meeting_start and now < meeting_end:
    print("This meeting is happening now.")
elif now < meeting_start:
    print("This meeting is coming up.")
elif now > meeting_end:
    print("This meeting already took place.")
```

Code that automatically adjusts a two-digit year to be correct

```
# get input and convert to a datetime object
birth_date = input("Enter birth date (MM/DD/YY): ")
birth_date = datetime.strptime(birth_date, "%m/%d/%y")

# if necessary, subtract 100 to fix birth year
if birth_date > datetime.now():
    fixed_birth_year = birth_date.year-100
    birth_date = datetime(fixed_birth_year,
                          birth_date.month,
                          birth_date.day)

print("Birth date: " + birth_date.strftime("%m/%d/%Y"))
```

The console

```
Enter date of birth (MM/DD/YY): 2/4/68
Date of birth: 02/04/1968
```

The Hotel Reservation program (user interface)

```
The Hotel Reservation program

Enter arrival date (YYYY-MM-DD): 2017-8-15
Enter departure date (YYYY-MM-DD): 2017-8-19

Arrival Date:   August 15, 2017
Departure Date: August 19, 2017
Nightly rate:   $105.00 (High season)
Total nights:   4
Total price:    $420.00

Continue? (y/n): y

Enter arrival date (YYYY-MM-DD): 2017-9-15
Enter departure date (YYYY-MM-DD): 2017-9-19

Arrival Date:   September 15, 2017
Departure Date: September 19, 2017
Nightly rate:   $85.00
Total nights:   4
Total price:    $340.00

Continue? (y/n): n

Bye!
```

The code

```
from datetime import datetime
import locale

def get_arrival_date():
    while True:
        date_str = input("Enter arrival date (YYYY-MM-DD): ")
        try:
            arrival_date = datetime.strptime(date_str, "%Y-%m-%d")
        except ValueError:
            print("Invalid date format. Try again.")
            continue

        # strip non-zero time values from datetime object
        now = datetime.now()
        today = datetime(now.year, now.month, now.day)
        if arrival_date < today:
            print("Arrival date must be today or later. Try again.")
            continue
        else:
            return arrival_date
```

The code (cont.)

```
def get_departure_date(arrival_date):
    while True:
        date_str = input("Enter departure date (YYYY-MM-DD): ")
        try:
            departure_date = datetime.strptime(date_str, "%Y-%m-%d")
        except ValueError:
            print("Invalid date format. Try again.")
            continue

        if departure_date <= arrival_date:
            print("Departure date must be after arrival date. " +
                  "Try again.")
            continue
        else:
            return departure_date
```

The code (cont.)

```
def main():
    print("The Hotel Reservation program\n")
    while True:
        # get datetime objects from user
        arrival_date = get_arrival_date()
        departure_date = get_departure_date(arrival_date)
        print()

        # calculate nights and cost
        rate = 85.0
        rate_message = ""
        if arrival_date.month == 8:    # August is high season
            rate = 105.0
            rate_message = "(High season)"
        total_nights = (departure_date - arrival_date).days
        total_cost = rate * total_nights
```

The code (cont.)

```
        # format results
        date_format = "%B %d, %Y"
        locale.setlocale(locale.LC_ALL, '')
        print("Arrival Date:  ",
              arrival_date.strftime(date_format))
        print("Departure Date:",
              departure_date.strftime(date_format))
        print("Nightly rate:  ",
              locale.currency(rate), rate_message)
        print("Total nights:  ", total_nights)
        print("Total price:   ", locale.currency(total_cost))
        print()

        # check whether the user wants to continue
        result = input("Continue? (y/n): ")
        print()
        if result.lower() != "y":
            print("Bye!")
            break

if __name__ == "__main__":
    main()
```