# Day 4: Programming with Python

# Working with Time and Dates

It is common in our data to have dates and times especially when we deal with temporal data. The dates and time in Python need to be handled with the appropriate libraries.

Python provides the `datetime` module that supplies classes for manipulating dates and times.

Pandas also provides a lot of functionality to work with times and dates. The most common classes in pandas that can be used to handle time/date data are `Timestamp` and `DatetimeIndex` .

## 0. Importing datetime and pandas

The first step is to import datetime and pandas. `datetime` is both a module and a class within that module, so we will import the datetime class from the datetime module.

In [1]:

```
from datetime import datetime
import pandas as pd
```

## 1. Datetime

Key points:

- **datetime(year=, month=, day=)** - Creates a datetime object
- **d.day/hour/year** - Prints the day/hour/year of a datetime object
- **datetime(year=, month=, day=, hour=, minute=, second=)** - Creates a datetime object
- **d.hour/minute/second** - Prints the hour/minute/second of a datetime object
- **datetime.now()** - Returns the current date and time
- **datetime.strptime("", "%Y-%m-%d")** - Converts a string to datetime

### 1.1 Create a date with datetime

Let's create our first date with the `datetime()` . As arguments we pass the year, month and day of the date and we pass it to the variable `d`

In [2]:

```
d = datetime(year=2021, month=8, day=25)
print(d)
```

```
2021-08-25 00:00:00
```

From the datetime variable it is very easy to extract the day, month and year

In [3]:

```
print(d.day)
print(d.month)
print(d.year)
```

```
25
8
2021
```

Let's say we now want to create a date with time.

In [4]:

```
d = datetime(2021, 8, 25, 10, 5, 10)
print(d)
```

```
2021-08-25 10:05:10
```

Let's print the hour, minute and second

In [5]:

```
print(d.hour)
print(d.minute)
print(d.second)
```

```
10
5
10
```

The `now()` function returns the current date and time.

In [6]:

```
datetime.now()
```

Out[6]:

```
datetime.datetime(2021, 8, 26, 8, 23, 43, 313511)
```

## 1.2 Pass a string into a datetime

We can parse a string into a datetime object with the `strptime()` functon. This function takes as imput the string and the date format of the string. For example, `%Y-%m-%d` means that I will expect a date in the format of year-month-day. The string `2021-8-26` will be parsed but the `2021/8/26` will return an error. Check all the codes here: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes

Let's create a datetime object `d` from the string `2021-8-26`. The format for this date is represented as `%Y-%m-%d`.

In [7]:

```
dateString = "2021-8-25"
d = datetime.strptime(dateString, "%Y-%m-%d")
print(type(d))
print(d)
```

```
<class 'datetime.datetime'>
2021-08-25 00:00:00
```

Now let's try to create a datetime object `d` from the string `2021/8/26`. We need the right format for that.

In [8]:

```
dateString = "2021/8/25"
d = datetime.strptime("2021/8/25", "%Y/%m/%d")
print(d)
```

```
2021-08-25 00:00:00
```

# 2. Handling dates with Pandas

- **pd.Timestamp()** - Creates a Timestamp object
- **ts.day/month/year/hour/minute/second** - Prints the day/hour/year/hour/minute/second of a timestamp object
- **pd.DatetimeIndex()** - Creates a DateTimeIndex
- **pd.date_range(start = , end = , freq = )** - Returns a fixed frequency DatetimeIndex

## 2.1 Timestamp

`Timestamp` is the pandas equivalent of python's datetime and is interchangeable with it in most cases. The `timestamp` takes as input either the string date or integers.

Let's try passing to timestamp different versions of dates. We can try the current day in different formats

In [9]:

```
pd.Timestamp("2021/8/25")
```

Out[9]:

```
Timestamp('2021-08-25 00:00:00')
```

In [10]:

```
pd.Timestamp("2021-8-25 10:00:00 PM")
```

Out[10]:

```
Timestamp('2021-08-25 22:00:00')
```

In [11]:

```
pd.Timestamp(2021, 8, 25, 10, 0, 0)
```

Out[11]:

```
Timestamp('2021-08-25 10:00:00')
```

Similar to datetime we can extract the year, month, day, hour, minute, seconds from a timestamp

In [12]:

```
d = pd.Timestamp(2021, 8, 25, 10, 0, 0)
d.hour
```

Out[12]:

```
10
```

## 2.2 DatetimeIndex

The index of a timestamp is the `DatetimeIndex`. `DatetimeIndex` can be used like a regular index and offers all of its intelligent functionality like selection, slicing.

Let's create a list dates and convert it into a series.

In [13]:

```
dates = ['8-10-2020','4-10-2020','5-10-2020','6-10-2020',
         '7-10-2020', '1-10-2020', '2-10-2020']
s = pd.Series([10, 12, 9, 11, 13, 14, 15], index = dates)

print(s)
```

```
8-10-2020    10
4-10-2020    12
5-10-2020     9
6-10-2020    11
7-10-2020    13
1-10-2020    14
```

```
2-10-2020    15
dtype: int64
```

In the above case the index is the dates but they are stored as objects. Let's try to get data until '5-10-2020'

In [14]:

```
print(s.loc[:'5-10-2020'])
```

```
8-10-2020    10
4-10-2020    12
5-10-2020     9
dtype: int64
```

Let's create the series again and now sotre index as DatetimeIndex

In [15]:

```
dates = ['8-10-2020','4-10-2020','5-10-2020','6-10-2020',
         '7-10-2020', '1-10-2020', '2-10-2020']
s = pd.Series([10, 12, 9, 11, 13, 14, 15], index = pd.DatetimeIndex(dates))
print(s)
```

```
2020-08-10    10
2020-04-10    12
2020-05-10     9
2020-06-10    11
2020-07-10    13
2020-01-10    14
2020-02-10    15
dtype: int64
```

Get the data until the 2020-05-05

In [16]:

```
print(s.loc[:'5-10-2020'])
```

```
2020-04-10    12
2020-05-10     9
2020-01-10    14
2020-02-10    15
dtype: int64
```

## 2.3 Date_range

If we want to create a range of dates, we can use the `date_range()` which returns the range of equally spaced time points such that they all occur between start and end. The parameter `freq` defines the frequency (e.g., D for day, W for week, A for year, H for hour)

Let's create dates from 2021-8-1 to 2021-8-31 with a frequency of a day and a week

In [17]:

```
dates = pd.date_range(start = '2021-8-1', end = '2021-8-31', freq = 'D')
dates
```

Out[17]:

```
DatetimeIndex(['2021-08-01', '2021-08-02', '2021-08-03', '2021-08-04',
               '2021-08-05', '2021-08-06', '2021-08-07', '2021-08-08',
               '2021-08-09', '2021-08-10', '2021-08-11', '2021-08-12',
               '2021-08-13', '2021-08-14', '2021-08-15', '2021-08-16',
               '2021-08-17', '2021-08-18', '2021-08-19', '2021-08-20',
               '2021-08-21', '2021-08-22', '2021-08-23', '2021-08-24',
               '2021-08-25', '2021-08-26', '2021-08-27', '2021-08-28',
               '2021-08-29', '2021-08-30', '2021-08-31'],
              dtype='datetime64[ns]', freq='D')
```

In [18]:

```
dates = pd.date_range(start = '2021-8-1', end = '2021-8-31', freq = 'W')
dates
```

Out[18]:
```
DatetimeIndex(['2021-08-01', '2021-08-08', '2021-08-15', '2021-08-22',
               '2021-08-29'],
              dtype='datetime64[ns]', freq='W-SUN')
```

# 3. Timedelta

- **timedelta** - Difference in time of two datetime/timestamp objects

A timedelta object represents a duration, the difference between two dates or times.

Let's create two datetimes and get their difference

In [19]:
```python
from datetime import timedelta

dateA = datetime(2020, 8, 25, 5, 20, 30)
dateB = datetime(2021, 8, 25, 5, 20, 30)
print(dateB - dateA)
```

```
365 days, 0:00:00
```

We can also add specific amount of time to a date. Let's add 2 weeks, 10 days from '2021-8-25 10:00:00'

In [20]:
```python
originalDate = datetime(2021, 8, 25, 10, 0, 0)
timeDelta = timedelta(weeks = 2, days = 10)
newDate = originalDate + timeDelta
print(newDate)
```

```
2021-09-18 10:00:00
```

# 4. Working with dates and DataFrames

Let's see in practice how to work with dates on a DataFrame.

Let's create a DataFrame with some random prices.

In [21]:
```python
import random

random.seed(0)
prices = []
dates = pd.date_range(start = '2010-1-1', end = '2019-12-31', freq = 'D')

for i in range(len(dates)):
    prices.append(random.randint(0, 20))

df = pd.DataFrame({"prices": prices}, index= dates)

df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3652 entries, 2010-01-01 to 2019-12-31
Freq: D
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   prices  3652 non-null   int64
dtypes: int64(1)
memory usage: 57.1 KB
```

```
Out[21]:
                prices
    2010-01-01      12
    2010-01-02      13
    2010-01-03       1
    2010-01-04       8
    2010-01-05      16
```

We can slice the data to view only data from 10 to 15 of January 2010

```
In [22]:  df.loc['10-1-2010':'15-10-2010']
```

```
Out[22]:
                prices
    2010-10-01       2
    2010-10-02       4
    2010-10-03       0
    2010-10-04      12
    2010-10-05      13
    2010-10-06      10
    2010-10-07       0
    2010-10-08       6
    2010-10-09       0
    2010-10-10       0
    2010-10-11      16
    2010-10-12      19
    2010-10-13       3
    2010-10-14       6
    2010-10-15       3
```

Resampling is for frequency conversion and resampling of time series. So, if one needs to change the data instead of daily to monthly or weekly etc. or vice versa.

Let's calculate the mean for every month and then for every year.

```
In [23]:  df.resample('M').mean()
```

```
Out[23]:
                   prices
    2010-01-31  10.387097
    2010-02-28  10.678571
    2010-03-31  10.354839
    2010-04-30  10.766667
    2010-05-31   8.870968
           ...        ...
```

|  | prices |
|---|---|
| **2019-08-31** | 10.064516 |
| **2019-09-30** | 10.000000 |
| **2019-10-31** | 12.806452 |
| **2019-11-30** | 9.266667 |
| **2019-12-31** | 9.548387 |

120 rows × 1 columns

In [24]:

```
df.resample('Y').mean()
```

Out[24]:

|  | prices |
|---|---|
| **2010-12-31** | 9.824658 |
| **2011-12-31** | 9.632877 |
| **2012-12-31** | 10.002732 |
| **2013-12-31** | 9.890411 |
| **2014-12-31** | 10.002740 |
| **2015-12-31** | 10.060274 |
| **2016-12-31** | 9.778689 |
| **2017-12-31** | 10.441096 |
| **2018-12-31** | 10.169863 |
| **2019-12-31** | 10.197260 |

# Exercises

```
1. Get the datetime of the current date and time and print the
month and the hour
```

In [25]:

```
from datetime import timedelta, datetime

now = datetime.now()
print(now.month)
print(now.hour)
```

```
8
8
```

```
2. You received the following date in string format. Convert it
into Python's datetime object.

**Hint:** Check the abbreviations here:
https://docs.python.org/3/library/datetime.html#strftime-and-
strptime-format-codes
```

In [26]:

```
date_string = "Apr 28 2018 5:20PM"
d = datetime.strptime(date_string, '%b %d %Y %I:%M%p')
```

```
print(d)
```

```
2018-04-28 17:20:00
```

3. Print a range of dates from 1-12-2019 until 31-12-2019 with a range of
a) 2 days and
b) 8 hours

In [27]:

```
dates = pd.date_range(start = '2019-12-1', end = '2019-12-31', freq = '2D')
dates
```

Out[27]:

```
DatetimeIndex(['2019-12-01', '2019-12-03', '2019-12-05', '2019-12-07',
               '2019-12-09', '2019-12-11', '2019-12-13', '2019-12-15',
               '2019-12-17', '2019-12-19', '2019-12-21', '2019-12-23',
               '2019-12-25', '2019-12-27', '2019-12-29', '2019-12-31'],
              dtype='datetime64[ns]', freq='2D')
```

In [28]:

```
dates = pd.date_range(start = '2019-12-1', end = '2019-12-31', freq = '8H')
dates
```

Out[28]:

```
DatetimeIndex(['2019-12-01 00:00:00', '2019-12-01 08:00:00',
               '2019-12-01 16:00:00', '2019-12-02 00:00:00',
               '2019-12-02 08:00:00', '2019-12-02 16:00:00',
               '2019-12-03 00:00:00', '2019-12-03 08:00:00',
               '2019-12-03 16:00:00', '2019-12-04 00:00:00',
               '2019-12-04 08:00:00', '2019-12-04 16:00:00',
               '2019-12-05 00:00:00', '2019-12-05 08:00:00',
               '2019-12-05 16:00:00', '2019-12-06 00:00:00',
               '2019-12-06 08:00:00', '2019-12-06 16:00:00',
               '2019-12-07 00:00:00', '2019-12-07 08:00:00',
               '2019-12-07 16:00:00', '2019-12-08 00:00:00',
               '2019-12-08 08:00:00', '2019-12-08 16:00:00',
               '2019-12-09 00:00:00', '2019-12-09 08:00:00',
               '2019-12-09 16:00:00', '2019-12-10 00:00:00',
               '2019-12-10 08:00:00', '2019-12-10 16:00:00',
               '2019-12-11 00:00:00', '2019-12-11 08:00:00',
               '2019-12-11 16:00:00', '2019-12-12 00:00:00',
               '2019-12-12 08:00:00', '2019-12-12 16:00:00',
               '2019-12-13 00:00:00', '2019-12-13 08:00:00',
               '2019-12-13 16:00:00', '2019-12-14 00:00:00',
               '2019-12-14 08:00:00', '2019-12-14 16:00:00',
               '2019-12-15 00:00:00', '2019-12-15 08:00:00',
               '2019-12-15 16:00:00', '2019-12-16 00:00:00',
               '2019-12-16 08:00:00', '2019-12-16 16:00:00',
               '2019-12-17 00:00:00', '2019-12-17 08:00:00',
               '2019-12-17 16:00:00', '2019-12-18 00:00:00',
               '2019-12-18 08:00:00', '2019-12-18 16:00:00',
               '2019-12-19 00:00:00', '2019-12-19 08:00:00',
               '2019-12-19 16:00:00', '2019-12-20 00:00:00',
               '2019-12-20 08:00:00', '2019-12-20 16:00:00',
               '2019-12-21 00:00:00', '2019-12-21 08:00:00',
               '2019-12-21 16:00:00', '2019-12-22 00:00:00',
               '2019-12-22 08:00:00', '2019-12-22 16:00:00',
               '2019-12-23 00:00:00', '2019-12-23 08:00:00',
               '2019-12-23 16:00:00', '2019-12-24 00:00:00',
               '2019-12-24 08:00:00', '2019-12-24 16:00:00',
               '2019-12-25 00:00:00', '2019-12-25 08:00:00',
               '2019-12-25 16:00:00', '2019-12-26 00:00:00',
               '2019-12-26 08:00:00', '2019-12-26 16:00:00',
               '2019-12-27 00:00:00', '2019-12-27 08:00:00',
               '2019-12-27 16:00:00', '2019-12-28 00:00:00',
               '2019-12-28 08:00:00', '2019-12-28 16:00:00',
               '2019-12-29 00:00:00', '2019-12-29 08:00:00',
               '2019-12-29 16:00:00', '2019-12-30 00:00:00',
               '2019-12-30 08:00:00', '2019-12-30 16:00:00',
               '2019-12-31 00:00:00'],
              dtype='datetime64[ns]', freq='8H')
```

4. Add 6 days and 12 hours to the following date: 25/2/2020 10.00 am

In [29]:
```
sampleDate = datetime(2020, 2, 25, 10, 00, 00)
print(sampleDate)

newDate = sampleDate + timedelta(days = 6, hours = 12)
print(newDate)
```

```
2020-02-25 10:00:00
2020-03-02 22:00:00
```

5. Write a function that takes as an argument the birthday of someone and outputs the age of the person. Call the function to calculate the age of someone born on 12/10/1990. Call the function again to calculate your own age.

In [30]:
```
def age_from_birthday(birthday):
    today = datetime.now()
    return (today.year - birthday.year)

print()
print(age_from_birthday(datetime(1990,10,12)))
print(age_from_birthday(datetime(1965,1,12)))
print()
```

```
31
56
```

## Bonus 6.

a) Read the file daily_natural_gas.csv that is stored in the data folder.
b) Calculate the mean for the range from 01-01-1997 until 25-08-1997
c) Print the max value per year
d) Find the dates that are not in the index for the year 1997 (check the DateTimeIndex difference in https://pandas.pydata.org/pandas-docs/version/0.18/generated/pandas.DatetimeIndex.difference.html)

**Hint** Parse the date as dates and make it index

In [31]:
```
df = pd.read_csv('../data/daily_natural_gas.csv',
                 parse_dates = ['Date'], index_col = 'Date')
df.head()
```

Out[31]:

|  | Price |
|---|---|
| **Date** |  |
| **1997-01-07** | 3.82 |

|  | Price |
|---|---|
| Date |  |
| 1997-01-08 | 3.80 |
| 1997-01-09 | 3.61 |
| 1997-01-10 | 3.92 |
| 1997-01-13 | 4.00 |

In [32]:

```
df['1-1-1997':'25-8-1997'].mean()
```

Out[32]:

```
Price    2.314161
dtype: float64
```

In [33]:

```
df.resample('Y').max()
```

Out[33]:

|  | Price |
|---|---|
| Date |  |
| 1997-12-31 | 4.71 |
| 1998-12-31 | 2.65 |
| 1999-12-31 | 3.10 |
| 2000-12-31 | 10.49 |
| 2001-12-31 | 10.31 |
| 2002-12-31 | 5.31 |
| 2003-12-31 | 18.48 |
| 2004-12-31 | 8.12 |
| 2005-12-31 | 15.39 |
| 2006-12-31 | 9.90 |
| 2007-12-31 | 9.14 |
| 2008-12-31 | 13.31 |
| 2009-12-31 | 6.10 |
| 2010-12-31 | 7.51 |
| 2011-12-31 | 4.92 |
| 2012-12-31 | 3.77 |
| 2013-12-31 | 4.52 |
| 2014-12-31 | 8.15 |
| 2015-12-31 | 3.32 |
| 2016-12-31 | 3.80 |
| 2017-12-31 | 3.71 |
| 2018-12-31 | 6.24 |
| 2019-12-31 | 4.25 |
| 2020-12-31 | 2.57 |

In [34]:

```
rangeOfDates = pd.date_range(start="1997-01-01", end="1997-12-01")
print(rangeOfDates.difference(df.index))
```

```
DatetimeIndex(['1997-01-01', '1997-01-02', '1997-01-03', '1997-01-04',
               '1997-01-05', '1997-01-06', '1997-01-11', '1997-01-12',
               '1997-01-18', '1997-01-19',
               ...
               '1997-11-08', '1997-11-09', '1997-11-15', '1997-11-16',
               '1997-11-22', '1997-11-23', '1997-11-27', '1997-11-28',
               '1997-11-29', '1997-11-30'],
              dtype='datetime64[ns]', length=107, freq=None)
```

In [ ]:

In [ ]: