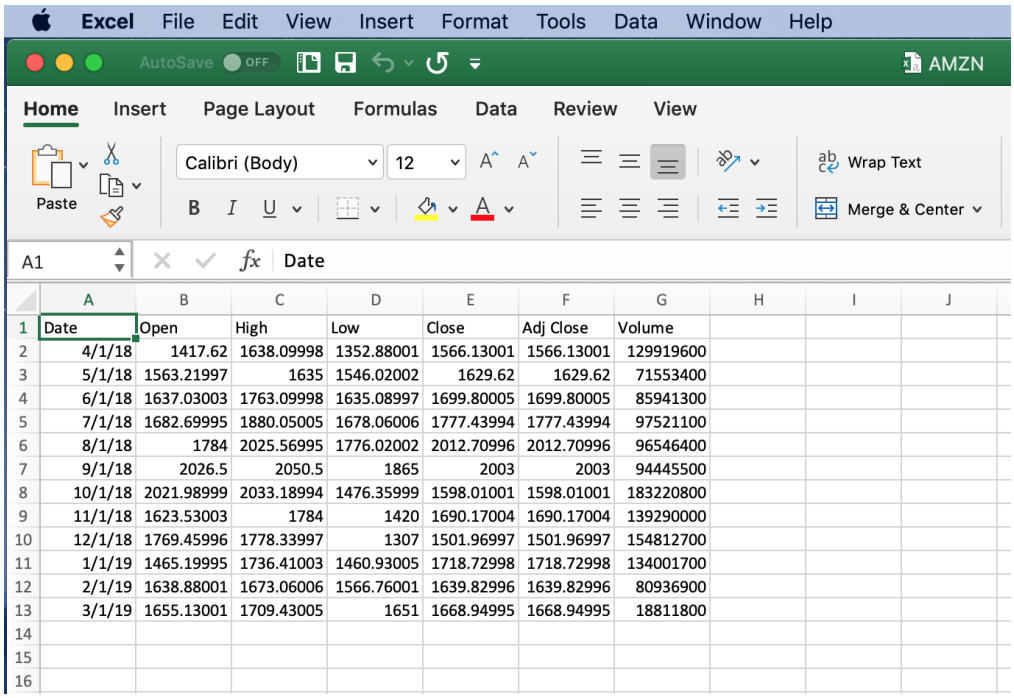
Lecture 17: CSV files for data sharing

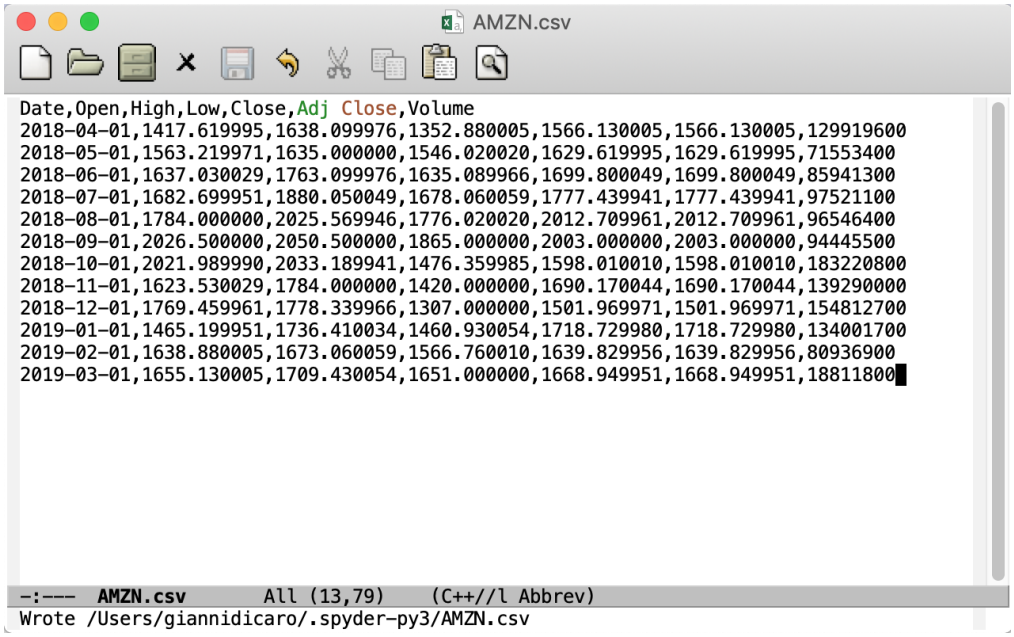
Today we will talk about using the CSV data format for reading/writing/sharing text files. CSV stands for Comma Separated Values. It's quite flexible and compact, it's around since long time, and it's the main format used by popular spreadsheet programs such as Excel. Many data repositories make use of CSV as one their standard formats for data.



	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	4/1/18	1417.62	1638.09998	1352.88001	1566.13001	1566.13001	129919600
3	5/1/18	1563.21997	1635	1546.02002	1629.62	1629.62	71553400
4	6/1/18	1637.03003	1763.09998	1635.08997	1699.80005	1699.80005	85941300
5	7/1/18	1682.69995	1880.05005	1678.06006	1777.43994	1777.43994	97521100
6	8/1/18	1784	2025.56995	1776.02002	2012.70996	2012.70996	96546400
7	9/1/18	2026.5	2050.5	1865	2003	2003	94445500
8	10/1/18	2021.98999	2033.18994	1476.35999	1598.01001	1598.01001	183220800
9	11/1/18	1623.53003	1784	1420	1690.17004	1690.17004	139290000
10	12/1/18	1769.45996	1778.33997	1307	1501.96997	1501.96997	154812700
11	1/1/19	1465.19995	1736.41003	1460.93005	1718.72998	1718.72998	134001700
12	2/1/19	1638.88001	1673.06006	1566.76001	1639.82996	1639.82996	80936900
13	3/1/19	1655.13001	1709.43005	1651	1668.94995	1668.94995	18811800
14							
15							
16							

The file reports the monthly evolution of the Amazon's stock market prices at Nasdaq. Data are dowloaded from Yahoo! Finance:
https://finance.yahoo.com/quote/AMZN/history?period1=1521362028&period2=1552898028&interval=1mo&filter=history&frequency=1mo

How the file looks like? Let's open it with a regular text editor:



CSV (comma separated values) is a format commonly used to hold in a file data that can be naturally represented in tabular form (e.g., excel-like): M data records/rows, each consisting of (at most) N ordered fields/columns:

row 1: column 1, column 2, column 3, ... , column N

row 2: column 1, column 2, column 3, ... , column N

row 3: column 1, column 2, column 3, ... , column N

.....

row M: column 1, column 2, column 3, ... , column N

In practice, data is represented as matrix where each column refers to a common object and each row is a different data entry.

Column data are separated by a given delimiter. The default delimiter is a comma, but other characters can be used as a delimiter.

E.g.: columns are metereological measurements an N different metereological stations, where each row reports the measures for a different day.

E.g.: columns are personal data, such as name, address, and ID, where each row of data refers to a different person.

E.g., each column is the student grade for a specific course, where each row reports the set of grades for a different student.

It is common, but not strictly required, that the first row/record in a csv file contains strings with the names/meanings of the columns (the legend for the file).

E.g., name, address, id, age, sex

J. Smith, Falcon Tower West-Bay, 532720 , 38, M

A. White, Tower 99 The Pearl, 33145, 29, F

In [3]:
```
import csv
# The csv module provides a number of methods to effectively and efficiently deal with the
# basic reading and writing operations on CSV files
```

In [4]:
```
file_path = '/Users/giannidicaro/.spyder-py3/csv/Mall_Customers.csv'
file_name = file_path.split('/')[-1]
#print(file_name)
f_csv = open(file_path)
csv_data = csv.reader(f_csv, delimiter=',')
#
# csv_data is an iterator: at each call will return the next line in the file
# data are read into lists of strings, where each list element is a string with
# a filed value, identified based on the given delimiter
```

In [5]:
```
csv_data
```

Out[5]:
```
<_csv.reader at 0x1055856d8>
```

In [6]:
```
#f = open(file_path)
#cnt = 0
#for ff in f:
#    print(ff)
#    cnt += 1
#    if cnt > 10:
#        break

f_csv.seek(0)
# Let's print out what's in the file
line_count = 0
for row in csv_data:
    print('Row {:d}: {} (length: {})'.format(line_count, row, len(row)))
    next(csv_data)
    # another way to make the same print
    #print('Line: {}'.format(' - '.join(row)))
    line_count += 1
# it looks like most of the column fields are nicely separated by commas, but som fields
# have extra spaces: should we worry about it? Let's re-read the file and let's use the data
```

```
Row 0: ['CustomerID', 'Gender', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)'] (length: 5)
Row 1: ['2', '  Male', ' 21', ' 15', ' 81'] (length: 5)
Row 2: ['4', 'Female', '23  ', '16  ', '77'] (length: 5)
Row 3: ['6', 'Female', '22', '17', '76'] (length: 5)
Row 4: ['8', 'Female', '23', '18', '94'] (length: 5)
Row 5: ['10', 'Female', '30', '19', '72'] (length: 5)
Row 6: ['12', 'Female', '35', '19', '99'] (length: 5)
Row 7: ['14', 'Female', '24', '20', '77'] (length: 5)
Row 8: ['16', 'Male', '22', '20', '79'] (length: 5)
Row 9: ['18', 'Male', '20', '21', '66'] (length: 5)
Row 10: ['20', 'Female', '35', '23', '98'] (length: 5)
Row 11: ['22', 'Male', '25', '24', '73'] (length: 5)
Row 12: ['24', 'Male', '31', '25', '73'] (length: 5)
Row 13: ['26', 'Male', '29', '28', '82'] (length: 5)
Row 14: ['28', 'Male', '35', '28', '61'] (length: 5)
Row 15: ['30', 'Female', '23', '29', '87'] (length: 5)
Row 16: ['32', 'Female', '21', '30', '73'] (length: 5)
Row 17: ['34', 'Male', '18', '33', '92'] (length: 5)
Row 18: ['36', 'Female', '21', '33', '81'] (length: 5)
Row 19: ['38', 'Female', '30', '34', '73'] (length: 5)
```

```
Row 20: ['40', 'Female', '20', '37', '75'] (length: 5)
Row 21: ['42', 'Male', '24', '38', '92'] (length: 5)
Row 22: ['44', 'Female', '31', '39', '61'] (length: 5)
Row 23: ['46', 'Female', '24', '39', '65'] (length: 5)
Row 24: ['48', 'Female', '27', '40', '47'] (length: 5)
Row 25: ['50', 'Female', '31', '40', '42'] (length: 5)
Row 26: ['52', 'Male', '33', '42', '60'] (length: 5)
Row 27: ['54', 'Male', '59', '43', '60'] (length: 5)
Row 28: ['56', 'Male', '47', '43', '41'] (length: 5)
Row 29: ['58', 'Male', '69', '44', '46'] (length: 5)
Row 30: ['60', 'Male', '53', '46', '46'] (length: 5)
Row 31: ['62', 'Male', '19', '46', '55'] (length: 5)
Row 32: ['64', 'Female', '54', '47', '59'] (length: 5)
Row 33: ['66', 'Male', '18', '48', '59'] (length: 5)
Row 34: ['68', 'Female', '68', '48', '48'] (length: 5)
Row 35: ['70', 'Female', '32', '48', '47'] (length: 5)
Row 36: ['72', 'Female', '47', '49', '42'] (length: 5)
Row 37: ['74', 'Female', '60', '50', '56'] (length: 5)
Row 38: ['76', 'Male', '26', '54', '54'] (length: 5)
Row 39: ['78', 'Male', '40', '54', '48'] (length: 5)
Row 40: ['80', 'Female', '49', '54', '42'] (length: 5)
Row 41: ['82', 'Male', '38', '54', '55'] (length: 5)
Row 42: ['84', 'Female', '46', '54', '44'] (length: 5)
Row 43: ['86', 'Male', '48', '54', '46'] (length: 5)
Row 44: ['88', 'Female', '22', '57', '55'] (length: 5)
Row 45: ['90', 'Female', '50', '58', '46'] (length: 5)
Row 46: ['92', 'Male', '18', '59', '41'] (length: 5)
Row 47: ['94', 'Female', '40', '60', '40'] (length: 5)
Row 48: ['96', 'Male', '24', '60', '52'] (length: 5)
Row 49: ['98', 'Female', '27', '60', '50'] (length: 5)
Row 50: ['100', 'Male', '20', '61', '49'] (length: 5)
Row 51: ['102', 'Female', '49', '62', '48'] (length: 5)
Row 52: ['104', 'Male', '26', '62', '55'] (length: 5)
Row 53: ['106', 'Female', '21', '62', '42'] (length: 5)
Row 54: ['108', 'Male', '54', '63', '46'] (length: 5)
Row 55: ['110', 'Male', '66', '63', '48'] (length: 5)
Row 56: ['112', 'Female', '19', '63', '54'] (length: 5)
Row 57: ['114', 'Male', '19', '64', '46'] (length: 5)
Row 58: ['116', 'Female', '19', '65', '50'] (length: 5)
Row 59: ['118', 'Female', '49', '65', '59'] (length: 5)
Row 60: ['120', 'Female', '50', '67', '57'] (length: 5)
Row 61: ['122', 'Female', '38', '67', '40'] (length: 5)
Row 62: ['124', 'Male', '39', '69', '91'] (length: 5)
Row 63: ['126', 'Female', '31', '70', '77'] (length: 5)
Row 64: ['128', 'Male', '40', '71', '95'] (length: 5)
Row 65: ['130', 'Male', '38', '71', '75'] (length: 5)
Row 66: ['132', 'Male', '39', '71', '75'] (length: 5)
Row 67: ['134', 'Female', '31', '72', '71'] (length: 5)
Row 68: ['136', 'Female', '29', '73', '88'] (length: 5)
Row 69: ['138', 'Male', '32', '73', '73'] (length: 5)
Row 70: ['140', 'Female', '35', '74', '72'] (length: 5)
Row 71: ['142', 'Male', '32', '75', '93'] (length: 5)
Row 72: ['144', 'Female', '32', '76', '87'] (length: 5)
Row 73: ['146', 'Male', '28', '77', '97'] (length: 5)
Row 74: ['148', 'Female', '32', '77', '74'] (length: 5)
Row 75: ['150', 'Male', '34', '78', '90'] (length: 5)
Row 76: ['152', 'Male', '39', '78', '88'] (length: 5)
```

```
Row 77: ['154', 'Female', '38', '78', '76'] (length: 5)
Row 78: ['156', 'Female', '27', '78', '89'] (length: 5)
Row 79: ['158', 'Female', '30', '78', '78'] (length: 5)
Row 80: ['160', 'Female', '30', '78', '73'] (length: 5)
Row 81: ['162', 'Female', '29', '79', '83'] (length: 5)
Row 82: ['164', 'Female', '31', '81', '93'] (length: 5)
Row 83: ['166', 'Female', '36', '85', '75'] (length: 5)
Row 84: ['168', 'Female', '33', '86', '95'] (length: 5)
Row 85: ['170', 'Male', '32', '87', '63'] (length: 5)
Row 86: ['172', 'Male', '28', '87', '75'] (length: 5)
Row 87: ['174', 'Male', '36', '87', '92'] (length: 5)
Row 88: ['176', 'Female', '30', '88', '86'] (length: 5)
Row 89: ['178', 'Male', '27', '88', '69'] (length: 5)
Row 90: ['180', 'Male', '35', '93', '90'] (length: 5)
Row 91: ['182', 'Female', '32', '97', '86'] (length: 5)
Row 92: ['184', 'Female', '29', '98', '88'] (length: 5)
Row 93: ['186', 'Male', '30', '99', '97'] (length: 5)
Row 94: ['188', 'Male', '28', '101', '68'] (length: 5)
Row 95: ['190', 'Female', '36', '103', '85'] (length: 5)
Row 96: ['192', 'Female', '32', '103', '69'] (length: 5)
Row 97: ['194', 'Female', '38', '113', '91'] (length: 5)
Row 98: ['196', 'Female', '35', '120', '79'] (length: 5)
Row 99: ['198', 'Male', '32', '126', '74'] (length: 5)
Row 100: ['200', 'Male', '30', '137', '83'] (length: 5)
```

```
---------------------------------------------------------------------------

StopIteration                             Traceback (most recent call last)

<ipython-input-6-f141f8c17dc7> in <module>()
 12 for row in csv_data:
 13     print('Row {:d}: {} (length: {})'.format(line_count, row, len(row)))
---> 14     next(csv_data)
 15     # another way to make the same print
 16     #print('Line: {}'.format(' - '.join(row)))


StopIteration:
```

In [7]:
```
# csv.reader() is an iterator: we have already reached the end, therefore, if we want
# to read it again, we have to restart from the beginning
# The function line = next(f_csv) can be used for to go to the next line,
# it returns the current line
f_csv.seek(0)
```

Out[7]: 0

In [8]:
```
# this time let's get more info about the file and let's print output in a more structured way
line_count = 0
for row in csv_data:
    if line_count == 0:
        columns = len(row)
        print('File {} contains {:d} columns: {:s}'.format(file_name, columns, ' - '.join(row)))
    else:
        print('ID {} is a {:>6s} of {:2d} years making {:3d}$/year and has \
```

```
        a spending score of {:3d}'.format(int(row[0]), row[1], int(row[2]),
                    int(row[3]), int(row[4])))
            line_count += 1
        f_csv.close()
        # output is correct: the int() function does a good job getting rid of all extra spaces
        # however, extra spaces in string fields stay there, because a space is a valid character!
```

```
File Mall_Customers.csv contains 5 columns: CustomerID - Gender - Age - Annual Income (k$) - Spending Score (
ID 1 is a   Male of 19 years making  15$/year and has a spending score of  39
ID 2 is a   Male of 21 years making  15$/year and has a spending score of  81
ID 3 is a    Female    of 20 years making  16$/year and has a spending score of   6
ID 4 is a Female of 23 years making  16$/year and has a spending score of  77
ID 5 is a Female of 31 years making  17$/year and has a spending score of  40
ID 6 is a Female of 22 years making  17$/year and has a spending score of  76
ID 7 is a Female of 35 years making  18$/year and has a spending score of   6
ID 8 is a Female of 23 years making  18$/year and has a spending score of  94
ID 9 is a   Male of 64 years making  19$/year and has a spending score of   3
ID 10 is a Female of 30 years making  19$/year and has a spending score of  72
ID 11 is a   Male of 67 years making  19$/year and has a spending score of  14
ID 12 is a Female of 35 years making  19$/year and has a spending score of  99
ID 13 is a Female of 58 years making  20$/year and has a spending score of  15
ID 14 is a Female of 24 years making  20$/year and has a spending score of  77
ID 15 is a   Male of 37 years making  20$/year and has a spending score of  13
ID 16 is a   Male of 22 years making  20$/year and has a spending score of  79
ID 17 is a Female of 35 years making  21$/year and has a spending score of  35
ID 18 is a   Male of 20 years making  21$/year and has a spending score of  66
ID 19 is a   Male of 52 years making  23$/year and has a spending score of  29
ID 20 is a Female of 35 years making  23$/year and has a spending score of  98
ID 21 is a   Male of 35 years making  24$/year and has a spending score of  35
ID 22 is a   Male of 25 years making  24$/year and has a spending score of  73
ID 23 is a Female of 46 years making  25$/year and has a spending score of   5
ID 24 is a   Male of 31 years making  25$/year and has a spending score of  73
ID 25 is a Female of 54 years making  28$/year and has a spending score of  14
ID 26 is a   Male of 29 years making  28$/year and has a spending score of  82
ID 27 is a Female of 45 years making  28$/year and has a spending score of  32
ID 28 is a   Male of 35 years making  28$/year and has a spending score of  61
ID 29 is a Female of 40 years making  29$/year and has a spending score of  31
ID 30 is a Female of 23 years making  29$/year and has a spending score of  87
ID 31 is a   Male of 60 years making  30$/year and has a spending score of   4
ID 32 is a Female of 21 years making  30$/year and has a spending score of  73
ID 33 is a   Male of 53 years making  33$/year and has a spending score of   4
ID 34 is a   Male of 18 years making  33$/year and has a spending score of  92
ID 35 is a Female of 49 years making  33$/year and has a spending score of  14
ID 36 is a Female of 21 years making  33$/year and has a spending score of  81
ID 37 is a Female of 42 years making  34$/year and has a spending score of  17
ID 38 is a Female of 30 years making  34$/year and has a spending score of  73
ID 39 is a Female of 36 years making  37$/year and has a spending score of  26
ID 40 is a Female of 20 years making  37$/year and has a spending score of  75
ID 41 is a Female of 65 years making  38$/year and has a spending score of  35
ID 42 is a   Male of 24 years making  38$/year and has a spending score of  92
ID 43 is a   Male of 48 years making  39$/year and has a spending score of  36
ID 44 is a Female of 31 years making  39$/year and has a spending score of  61
ID 45 is a Female of 49 years making  39$/year and has a spending score of  28
ID 46 is a Female of 24 years making  39$/year and has a spending score of  65
ID 47 is a Female of 50 years making  40$/year and has a spending score of  55
ID 48 is a Female of 27 years making  40$/year and has a spending score of  47
ID 49 is a Female of 29 years making  40$/year and has a spending score of  42
```

```
ID 50 is a Female of 31 years making  40$/year and has a spending score of  42
ID 51 is a Female of 49 years making  42$/year and has a spending score of  52
ID 52 is a   Male of 33 years making  42$/year and has a spending score of  60
ID 53 is a Female of 31 years making  43$/year and has a spending score of  54
ID 54 is a   Male of 59 years making  43$/year and has a spending score of  60
ID 55 is a Female of 50 years making  43$/year and has a spending score of  45
ID 56 is a   Male of 47 years making  43$/year and has a spending score of  41
ID 57 is a Female of 51 years making  44$/year and has a spending score of  50
ID 58 is a   Male of 69 years making  44$/year and has a spending score of  46
ID 59 is a Female of 27 years making  46$/year and has a spending score of  51
ID 60 is a   Male of 53 years making  46$/year and has a spending score of  46
ID 61 is a   Male of 70 years making  46$/year and has a spending score of  56
ID 62 is a   Male of 19 years making  46$/year and has a spending score of  55
ID 63 is a Female of 67 years making  47$/year and has a spending score of  52
ID 64 is a Female of 54 years making  47$/year and has a spending score of  59
ID 65 is a   Male of 63 years making  48$/year and has a spending score of  51
ID 66 is a   Male of 18 years making  48$/year and has a spending score of  59
ID 67 is a Female of 43 years making  48$/year and has a spending score of  50
ID 68 is a Female of 68 years making  48$/year and has a spending score of  48
ID 69 is a   Male of 19 years making  48$/year and has a spending score of  59
ID 70 is a Female of 32 years making  48$/year and has a spending score of  47
ID 71 is a   Male of 70 years making  49$/year and has a spending score of  55
ID 72 is a Female of 47 years making  49$/year and has a spending score of  42
ID 73 is a Female of 60 years making  50$/year and has a spending score of  49
ID 74 is a Female of 60 years making  50$/year and has a spending score of  56
ID 75 is a   Male of 59 years making  54$/year and has a spending score of  47
ID 76 is a   Male of 26 years making  54$/year and has a spending score of  54
ID 77 is a Female of 45 years making  54$/year and has a spending score of  53
ID 78 is a   Male of 40 years making  54$/year and has a spending score of  48
ID 79 is a Female of 23 years making  54$/year and has a spending score of  52
ID 80 is a Female of 49 years making  54$/year and has a spending score of  42
ID 81 is a   Male of 57 years making  54$/year and has a spending score of  51
ID 82 is a   Male of 38 years making  54$/year and has a spending score of  55
ID 83 is a   Male of 67 years making  54$/year and has a spending score of  41
ID 84 is a Female of 46 years making  54$/year and has a spending score of  44
ID 85 is a Female of 21 years making  54$/year and has a spending score of  57
ID 86 is a   Male of 48 years making  54$/year and has a spending score of  46
ID 87 is a Female of 55 years making  57$/year and has a spending score of  58
ID 88 is a Female of 22 years making  57$/year and has a spending score of  55
ID 89 is a Female of 34 years making  58$/year and has a spending score of  60
ID 90 is a Female of 50 years making  58$/year and has a spending score of  46
ID 91 is a Female of 68 years making  59$/year and has a spending score of  55
ID 92 is a   Male of 18 years making  59$/year and has a spending score of  41
ID 93 is a   Male of 48 years making  60$/year and has a spending score of  49
ID 94 is a Female of 40 years making  60$/year and has a spending score of  40
ID 95 is a Female of 32 years making  60$/year and has a spending score of  42
ID 96 is a   Male of 24 years making  60$/year and has a spending score of  52
ID 97 is a Female of 47 years making  60$/year and has a spending score of  47
ID 98 is a Female of 27 years making  60$/year and has a spending score of  50
ID 99 is a   Male of 48 years making  61$/year and has a spending score of  42
ID 100 is a   Male of 20 years making  61$/year and has a spending score of  49
ID 101 is a Female of 23 years making  62$/year and has a spending score of  41
ID 102 is a Female of 49 years making  62$/year and has a spending score of  48
ID 103 is a   Male of 67 years making  62$/year and has a spending score of  59
ID 104 is a   Male of 26 years making  62$/year and has a spending score of  55
ID 105 is a   Male of 49 years making  62$/year and has a spending score of  56
ID 106 is a Female of 21 years making  62$/year and has a spending score of  42
```

```
ID 107 is a Female of 66 years making  63$/year and has a spending score of  50
ID 108 is a   Male of 54 years making  63$/year and has a spending score of  46
ID 109 is a   Male of 68 years making  63$/year and has a spending score of  43
ID 110 is a   Male of 66 years making  63$/year and has a spending score of  48
ID 111 is a   Male of 65 years making  63$/year and has a spending score of  52
ID 112 is a Female of 19 years making  63$/year and has a spending score of  54
ID 113 is a Female of 38 years making  64$/year and has a spending score of  42
ID 114 is a   Male of 19 years making  64$/year and has a spending score of  46
ID 115 is a Female of 18 years making  65$/year and has a spending score of  48
ID 116 is a Female of 19 years making  65$/year and has a spending score of  50
ID 117 is a Female of 63 years making  65$/year and has a spending score of  43
ID 118 is a Female of 49 years making  65$/year and has a spending score of  59
ID 119 is a Female of 51 years making  67$/year and has a spending score of  43
ID 120 is a Female of 50 years making  67$/year and has a spending score of  57
ID 121 is a   Male of 27 years making  67$/year and has a spending score of  56
ID 122 is a Female of 38 years making  67$/year and has a spending score of  40
ID 123 is a Female of 40 years making  69$/year and has a spending score of  58
ID 124 is a   Male of 39 years making  69$/year and has a spending score of  91
ID 125 is a Female of 23 years making  70$/year and has a spending score of  29
ID 126 is a Female of 31 years making  70$/year and has a spending score of  77
ID 127 is a   Male of 43 years making  71$/year and has a spending score of  35
ID 128 is a   Male of 40 years making  71$/year and has a spending score of  95
ID 129 is a   Male of 59 years making  71$/year and has a spending score of  11
ID 130 is a   Male of 38 years making  71$/year and has a spending score of  75
ID 131 is a   Male of 47 years making  71$/year and has a spending score of   9
ID 132 is a   Male of 39 years making  71$/year and has a spending score of  75
ID 133 is a Female of 25 years making  72$/year and has a spending score of  34
ID 134 is a Female of 31 years making  72$/year and has a spending score of  71
ID 135 is a   Male of 20 years making  73$/year and has a spending score of   5
ID 136 is a Female of 29 years making  73$/year and has a spending score of  88
ID 137 is a Female of 44 years making  73$/year and has a spending score of   7
ID 138 is a   Male of 32 years making  73$/year and has a spending score of  73
ID 139 is a   Male of 19 years making  74$/year and has a spending score of  10
ID 140 is a Female of 35 years making  74$/year and has a spending score of  72
ID 141 is a Female of 57 years making  75$/year and has a spending score of   5
ID 142 is a   Male of 32 years making  75$/year and has a spending score of  93
ID 143 is a Female of 28 years making  76$/year and has a spending score of  40
ID 144 is a Female of 32 years making  76$/year and has a spending score of  87
ID 145 is a   Male of 25 years making  77$/year and has a spending score of  12
ID 146 is a   Male of 28 years making  77$/year and has a spending score of  97
ID 147 is a   Male of 48 years making  77$/year and has a spending score of  36
ID 148 is a Female of 32 years making  77$/year and has a spending score of  74
ID 149 is a Female of 34 years making  78$/year and has a spending score of  22
ID 150 is a   Male of 34 years making  78$/year and has a spending score of  90
ID 151 is a   Male of 43 years making  78$/year and has a spending score of  17
ID 152 is a   Male of 39 years making  78$/year and has a spending score of  88
ID 153 is a Female of 44 years making  78$/year and has a spending score of  20
ID 154 is a Female of 38 years making  78$/year and has a spending score of  76
ID 155 is a Female of 47 years making  78$/year and has a spending score of  16
ID 156 is a Female of 27 years making  78$/year and has a spending score of  89
ID 157 is a   Male of 37 years making  78$/year and has a spending score of   1
ID 158 is a Female of 30 years making  78$/year and has a spending score of  78
ID 159 is a   Male of 34 years making  78$/year and has a spending score of   1
ID 160 is a Female of 30 years making  78$/year and has a spending score of  73
ID 161 is a Female of 56 years making  79$/year and has a spending score of  35
ID 162 is a Female of 29 years making  79$/year and has a spending score of  83
ID 163 is a   Male of 19 years making  81$/year and has a spending score of   5
```

```
ID 164 is a Female of 31 years making  81$/year and has a spending score of  93
ID 165 is a   Male of 50 years making  85$/year and has a spending score of  26
ID 166 is a Female of 36 years making  85$/year and has a spending score of  75
ID 167 is a   Male of 42 years making  86$/year and has a spending score of  20
ID 168 is a Female of 33 years making  86$/year and has a spending score of  95
ID 169 is a Female of 36 years making  87$/year and has a spending score of  27
ID 170 is a   Male of 32 years making  87$/year and has a spending score of  63
ID 171 is a   Male of 40 years making  87$/year and has a spending score of  13
ID 172 is a   Male of 28 years making  87$/year and has a spending score of  75
ID 173 is a   Male of 36 years making  87$/year and has a spending score of  10
ID 174 is a   Male of 36 years making  87$/year and has a spending score of  92
ID 175 is a Female of 52 years making  88$/year and has a spending score of  13
ID 176 is a Female of 30 years making  88$/year and has a spending score of  86
ID 177 is a   Male of 58 years making  88$/year and has a spending score of  15
ID 178 is a   Male of 27 years making  88$/year and has a spending score of  69
ID 179 is a   Male of 59 years making  93$/year and has a spending score of  14
ID 180 is a   Male of 35 years making  93$/year and has a spending score of  90
ID 181 is a Female of 37 years making  97$/year and has a spending score of  32
ID 182 is a Female of 32 years making  97$/year and has a spending score of  86
ID 183 is a   Male of 46 years making  98$/year and has a spending score of  15
ID 184 is a Female of 29 years making  98$/year and has a spending score of  88
ID 185 is a Female of 41 years making  99$/year and has a spending score of  39
ID 186 is a   Male of 30 years making  99$/year and has a spending score of  97
ID 187 is a Female of 54 years making 101$/year and has a spending score of  24
ID 188 is a   Male of 28 years making 101$/year and has a spending score of  68
ID 189 is a Female of 41 years making 103$/year and has a spending score of  17
ID 190 is a Female of 36 years making 103$/year and has a spending score of  85
ID 191 is a Female of 34 years making 103$/year and has a spending score of  23
ID 192 is a Female of 32 years making 103$/year and has a spending score of  69
ID 193 is a   Male of 33 years making 113$/year and has a spending score of   8
ID 194 is a Female of 38 years making 113$/year and has a spending score of  91
ID 195 is a Female of 47 years making 120$/year and has a spending score of  16
ID 196 is a Female of 35 years making 120$/year and has a spending score of  79
ID 197 is a Female of 45 years making 126$/year and has a spending score of  28
ID 198 is a   Male of 32 years making 126$/year and has a spending score of  74
ID 199 is a   Male of 32 years making 137$/year and has a spending score of  18
ID 200 is a   Male of 30 years making 137$/year and has a spending score of  83
```

In [9]:
```
# is , the only allowed delimiter? It is the most common one, but we are not restricted to it
# let's deal with a file with the same content but different delimiter
file_path = '/Users/giannidicaro/.spyder-py3/csv/Mall_Customers-d2.csv'
f2_csv = open(file_path)
csv_data = csv.reader(f2_csv, delimiter=';')
line_count = 0
for row in csv_data:
    print('Line: {}'.format(' '.join(row)))
    line_count += 1
f2_csv.close()
# no problems at all, we get the same output!
```

```
Line: CustomerID Gender Age Annual Income (k$) Spending Score (1-100)
Line: 1  Male   19  15  39
Line: 2   Male  21  15  81
Line: 3    Female    20 16 6
Line: 4 Female 23   16   77
Line: 5 Female 31 17 40
Line: 6 Female 22 17 76
```

```
Line: 7 Female 35 18 6
Line: 8 Female 23 18 94
Line: 9 Male 64 19 3
Line: 10 Female 30 19 72
Line: 11 Male 67 19 14
Line: 12 Female 35 19 99
Line: 13 Female 58 20 15
Line: 14 Female 24 20 77
Line: 15 Male 37 20 13
Line: 16 Male 22 20 79
Line: 17 Female 35 21 35
Line: 18 Male 20 21 66
Line: 19 Male 52 23 29
Line: 20 Female 35 23 98
Line: 21 Male 35 24 35
Line: 22 Male 25 24 73
Line: 23 Female 46 25 5
Line: 24 Male 31 25 73
Line: 25 Female 54 28 14
Line: 26 Male 29 28 82
Line: 27 Female 45 28 32
Line: 28 Male 35 28 61
Line: 29 Female 40 29 31
Line: 30 Female 23 29 87
Line: 31 Male 60 30 4
Line: 32 Female 21 30 73
Line: 33 Male 53 33 4
Line: 34 Male 18 33 92
Line: 35 Female 49 33 14
Line: 36 Female 21 33 81
Line: 37 Female 42 34 17
Line: 38 Female 30 34 73
Line: 39 Female 36 37 26
Line: 40 Female 20 37 75
Line: 41 Female 65 38 35
Line: 42 Male 24 38 92
Line: 43 Male 48 39 36
Line: 44 Female 31 39 61
Line: 45 Female 49 39 28
Line: 46 Female 24 39 65
Line: 47 Female 50 40 55
Line: 48 Female 27 40 47
Line: 49 Female 29 40 42
Line: 50 Female 31 40 42
Line: 51 Female 49 42 52
Line: 52 Male 33 42 60
Line: 53 Female 31 43 54
Line: 54 Male 59 43 60
Line: 55 Female 50 43 45
Line: 56 Male 47 43 41
Line: 57 Female 51 44 50
Line: 58 Male 69 44 46
Line: 59 Female 27 46 51
Line: 60 Male 53 46 46
Line: 61 Male 70 46 56
Line: 62 Male 19 46 55
Line: 63 Female 67 47 52
```

```
Line: 64 Female 54 47 59
Line: 65 Male 63 48 51
Line: 66 Male 18 48 59
Line: 67 Female 43 48 50
Line: 68 Female 68 48 48
Line: 69 Male 19 48 59
Line: 70 Female 32 48 47
Line: 71 Male 70 49 55
Line: 72 Female 47 49 42
Line: 73 Female 60 50 49
Line: 74 Female 60 50 56
Line: 75 Male 59 54 47
Line: 76 Male 26 54 54
Line: 77 Female 45 54 53
Line: 78 Male 40 54 48
Line: 79 Female 23 54 52
Line: 80 Female 49 54 42
Line: 81 Male 57 54 51
Line: 82 Male 38 54 55
Line: 83 Male 67 54 41
Line: 84 Female 46 54 44
Line: 85 Female 21 54 57
Line: 86 Male 48 54 46
Line: 87 Female 55 57 58
Line: 88 Female 22 57 55
Line: 89 Female 34 58 60
Line: 90 Female 50 58 46
Line: 91 Female 68 59 55
Line: 92 Male 18 59 41
Line: 93 Male 48 60 49
Line: 94 Female 40 60 40
Line: 95 Female 32 60 42
Line: 96 Male 24 60 52
Line: 97 Female 47 60 47
Line: 98 Female 27 60 50
Line: 99 Male 48 61 42
Line: 100 Male 20 61 49
Line: 101 Female 23 62 41
Line: 102 Female 49 62 48
Line: 103 Male 67 62 59
Line: 104 Male 26 62 55
Line: 105 Male 49 62 56
Line: 106 Female 21 62 42
Line: 107 Female 66 63 50
Line: 108 Male 54 63 46
Line: 109 Male 68 63 43
Line: 110 Male 66 63 48
Line: 111 Male 65 63 52
Line: 112 Female 19 63 54
Line: 113 Female 38 64 42
Line: 114 Male 19 64 46
Line: 115 Female 18 65 48
Line: 116 Female 19 65 50
Line: 117 Female 63 65 43
Line: 118 Female 49 65 59
Line: 119 Female 51 67 43
Line: 120 Female 50 67 57
```

```
Line: 121 Male 27 67 56
Line: 122 Female 38 67 40
Line: 123 Female 40 69 58
Line: 124 Male 39 69 91
Line: 125 Female 23 70 29
Line: 126 Female 31 70 77
Line: 127 Male 43 71 35
Line: 128 Male 40 71 95
Line: 129 Male 59 71 11
Line: 130 Male 38 71 75
Line: 131 Male 47 71 9
Line: 132 Male 39 71 75
Line: 133 Female 25 72 34
Line: 134 Female 31 72 71
Line: 135 Male 20 73 5
Line: 136 Female 29 73 88
Line: 137 Female 44 73 7
Line: 138 Male 32 73 73
Line: 139 Male 19 74 10
Line: 140 Female 35 74 72
Line: 141 Female 57 75 5
Line: 142 Male 32 75 93
Line: 143 Female 28 76 40
Line: 144 Female 32 76 87
Line: 145 Male 25 77 12
Line: 146 Male 28 77 97
Line: 147 Male 48 77 36
Line: 148 Female 32 77 74
Line: 149 Female 34 78 22
Line: 150 Male 34 78 90
Line: 151 Male 43 78 17
Line: 152 Male 39 78 88
Line: 153 Female 44 78 20
Line: 154 Female 38 78 76
Line: 155 Female 47 78 16
Line: 156 Female 27 78 89
Line: 157 Male 37 78 1
Line: 158 Female 30 78 78
Line: 159 Male 34 78 1
Line: 160 Female 30 78 73
Line: 161 Female 56 79 35
Line: 162 Female 29 79 83
Line: 163 Male 19 81 5
Line: 164 Female 31 81 93
Line: 165 Male 50 85 26
Line: 166 Female 36 85 75
Line: 167 Male 42 86 20
Line: 168 Female 33 86 95
Line: 169 Female 36 87 27
Line: 170 Male 32 87 63
Line: 171 Male 40 87 13
Line: 172 Male 28 87 75
Line: 173 Male 36 87 10
Line: 174 Male 36 87 92
Line: 175 Female 52 88 13
Line: 176 Female 30 88 86
Line: 177 Male 58 88 15
```

```
Line: 178 Male 27 88 69
Line: 179 Male 59 93 14
Line: 180 Male 35 93 90
Line: 181 Female 37 97 32
Line: 182 Female 32 97 86
Line: 183 Male 46 98 15
Line: 184 Female 29 98 88
Line: 185 Female 41 99 39
Line: 186 Male 30 99 97
Line: 187 Female 54 101 24
Line: 188 Male 28 101 68
Line: 189 Female 41 103 17
Line: 190 Female 36 103 85
Line: 191 Female 34 103 23
Line: 192 Female 32 103 69
Line: 193 Male 33 113 8
Line: 194 Female 38 113 91
Line: 195 Female 47 120 16
Line: 196 Female 35 120 79
Line: 197 Female 45 126 28
Line: 198 Male 32 126 74
Line: 199 Male 32 137 18
Line: 200 Male 30 137 83
```

```
In [10]: # what about a delimiter with more than one single character?
         file_path = '/Users/giannidicaro/.spyder-py3/csv/Mall_Customers-d3.csv'
         f3_csv = open(file_path)
         try:
             csv_data = csv.reader(f3_csv, delimiter='--')
         except:
             print("Delimiter must be 1-charater string!")
         else:
             line_count = 0
             for row in csv_data:
                 print('Line: {}'.format(' '.join(row)))
                 line_count += 1
         finally:
             f3_csv.close()
         # TypeError! delimiter must be 1-character string!
```

```
Delimiter must be 1-charater string!
```

```
In [11]: # What if I want to use commas but the fields contain commas in their data?
         # Let's look at file employee_adresses.csv
         # each record contains three fields:
         # name, adress, date joined
         # Unfortunately, the field address contains commas, as it is common defining addresses
         # What happens if we try to read the file?
         file_path = '/Users/giannidicaro/.spyder-py3/csv/employee_addresses.csv'
         f_csv = open(file_path)
         csv_data = csv.reader(f_csv, delimiter=',')
         line_count = 0
         for row in csv_data:
             print('Line: {} (#fields: {})'.format(' - '.join(row), len(row)))
             line_count += 1
         f_csv.close()
```

```
        # as expected, the number of fields in each row is 4 instead of being 3, since every comma
        # in the row is interpreted as a field separator
```

```
Line: name - address - date joined (#fields: 3)
Line: john smith - 1132 Anywhere Lane Hoboken NJ -  07030 - Jan 4 (#fields: 4)
Line: erica meyers - 1234 Smith Lane Hoboken NJ -  07030 -   March 2 (#fields: 4)
Line: ann mcdonald -  9223 Yoda Lane Pythonopolis CA -  90001 - April 1 (#fields: 4)
```

```
In [12]: # How do we deal with this issue?
         # Three possible strategies, all requiring modifying the original csv file:
         # 1. Use a different delimiter in the csv file (e.g., ';')
         # 2. Wrap the data containing commas in quotes: the string between the quotes is not
         #      evaluated for the delimiter. The character used for quoting needs to be specified by
         #      the quotechar optional parameter if different from " which is the default
         # 3. Escape the delimiter character in the data: adding \ "protects" the character from
         #          being evaluated as a delimiter. If an escape character is used, it must be
         #          specified using the escapechar optional parameter.
         # Strategy 1:
         file_path = '/Users/giannidicaro/.spyder-py3/csv/employee_addresses-d2.csv'
         f_csv = open(file_path)
         csv_data = csv.reader(f_csv, delimiter=';')
         line_count = 0
         for row in csv_data:
             print('Line: {} (#fields: {})'.format(' - '.join(row), len(row)))
             line_count += 1
         f_csv.close()
         # it works as expected!
```

```
Line: name - address - date joined (#fields: 3)
Line: john smith - 1132 Anywhere Lane Hoboken NJ, 07030 - Jan 4 (#fields: 3)
Line: erica meyers - 1234 Smith Lane Hoboken NJ, 07030 -   March 2 (#fields: 3)
Line: ann mcdonald -  9223 Yoda Lane Pythonopolis CA, 90001 - April 1 (#fields: 3)
```

```
In [13]: # Strategy 2:
         file_path = '/Users/giannidicaro/.spyder-py3/csv/employee_addresses-quotes.csv'
         f_csv = open(file_path)
         csv_data = csv.reader(f_csv, delimiter=',', quotechar='"')
         line_count = 0
         for row in csv_data:
             print('Line: {} (#fields: {})'.format(' - '.join(row), len(row)))
             line_count += 1
         f_csv.close()
         # it works! however, some attention needs to be devoted to the presence of spaces
         # before or after the quoting character, that would prevent from letting the
         # character being properly interpreted
```

```
Line: name - address - date joined (#fields: 3)
Line: john smith - 1132 Anywhere Lane Hoboken NJ, 07030 - Jan 4 (#fields: 3)
Line: erica meyers - 1234 Smith Lane Hoboken NJ, 07030 -   March 2 (#fields: 3)
Line: ann mcdonald - 9223 Yoda Lane Pythonopolis CA, 90001 - April 1 (#fields: 3)
```

```
In [14]: # Strategy 3:
         file_path = '/Users/giannidicaro/.spyder-py3/csv/employee_addresses-escape.csv'
         f_csv = open(file_path)
         csv_data = csv.reader(f_csv, delimiter=',', escapechar='\\')
```

```
        line_count = 0
        for row in csv_data:
            print('Line: {} (#fields: {})'.format(' - '.join(row), len(row)))
            line_count += 1
        f_csv.close()
```

```
Line: name - address - date joined (#fields: 3)
Line: john smith -  1132 Anywhere Lane Hoboken NJ, 07030 - Jan 4 (#fields: 3)
Line: erica meyers -  1234 Smith Lane Hoboken NJ, 07030 -   March 2 (#fields: 3)
Line: ann mcdonald - 9223 Yoda Lane Pythonopolis CA, 90001 - April 1 (#fields: 3)
```

In [15]:
```
# A csv file can be seen as a dictionary: each column has a label,
# hence, we can read the csv data file (or, more generically, tabular data)
# into an 'ordered dictionary', an dictionary that preserves/remembers the order for entering
# the keys. The keays are sorted by the order associated to their entrance in the dictionary.
# Each row is an ordered dictionary with respect to the keys/columns
# An ordered dictionary is a data type from the module 'collections' that can be constructed
# with od = collections.OrderedDict()
#
file_path = '/Users/giannidicaro/.spyder-py3/csv/biometric_simple.csv'
f_csv = open(file_path)
csv_data = csv.DictReader(f_csv)
print("Type of object csv_data: {}\n".format(type(csv_data)))
```

```
Type of object csv_data: <class 'csv.DictReader'>
```

In [16]:
```
#
# The csv dictionary reader object csv_data is constructed from the first row
# of the csv file, that specifies the names of the fields, that is, the common label/key
# of each field / column.
#
# Based on the definition of the keys, csv data are read into an ordered dictionary
# where each row is stored in an ordered dictionary of strings: the keys are the strings
# defined in the header row and the values are strings representing the column values
#
# The number and names of the fields/keys can be retrieved by accessing the list .fieldnames
# of the dictionary reader
#
import os

num_of_keys = len(csv_data.fieldnames)
keys = csv_data.fieldnames
stat = os.stat(file_path)
size = stat.st_size
```

In [17]:
```
# What is the number of records? the reader doesn't know at this stage,
# we must read the data first! But we can print out the number of records
# and maybe the size of the entire file, to get an idea of how big it will be
#
print('File {} has size {} bytes and contains {:d} keys: {:s}\n'.format(file_name,
                                                    size, num_of_keys, ' - '.join(keys)))
```

```
File Mall_Customers.csv has size 100 bytes and contains 5 keys: id - name - age - height - weight
```

In [18]:
```
# After the creation of the ordered dictionary object, the iterator is positioned at
# the first row with actual data
# Now we can read / print each line, which is a ordered dictionary with N keys
# Let's use the (known) names of columns to make a nice printing
line_count = 1
for row in csv_data:
    print('Row {} has type {} and {} keys'.format(line_count, type(row), len(row)))
    print('{:<8s} has ID {:5d}, is {:2d} years old, {:4.2f}m tall, and weights {:5.2f}kg\n'.
          format(row['name'], int(row['id']), int(row['age']),
                 float(row['height'])/100, float(row['weight'])))
    line_count += 1
#f_csv.close()
```

```
Row 1 has type <class 'collections.OrderedDict'> and 5 keys
Alice    has ID   768, is 20 years old, 1.62m tall, and weights 54.60kg

Row 2 has type <class 'collections.OrderedDict'> and 5 keys
Freddie  has ID   562, is 21 years old, 1.74m tall, and weights 78.60kg

Row 3 has type <class 'collections.OrderedDict'> and 5 keys
Bob      has ID   523, is 17 years old, 1.68m tall, and weights 82.00kg
```

In [19]:
```
# Can we get the same nice printing without first opening the file and reading the labels?
#
# First, let's notice that since we have read the entire file, we need to rewind it,
# to go back to the first record. In fact, the instance csv_data is an iterator.
# An iterator emits a unit of data on each explicit/implicit invocation of next() on it,
# and with the above instructions we have performed an implicit next() call at each
# step of the for loop. Therefore, now the iterator is at the end of the file
# and it is necessary to rewind the file, and skip the header
f_csv.seek(0)
next(csv_data)
```

Out[19]:
```
OrderedDict([('id', 'id'),
             ('name', 'name'),
             ('age', 'age'),
             ('height', 'height'),
             ('weight', 'weight')])
```

In [20]:
```
# variable keys above is a list with all dictionary keys in order of column insertion,
# and we know that in each row keys always keep the same order
# print(keys)
#
line_count = 1
for row in csv_data:
    print('{:<8s} has ID {:5d}, is {:2d} years old, {:4.2f}m tall, and weights {:5.2f}kg\n'.
          format(row[keys[1]], int(row[keys[0]]), int(row[keys[2]]),
                 float(row[keys[3]])/100, float(row[keys[4]])))
    line_count += 1
#f_csv.close()
```

```
Alice    has ID   768, is 20 years old, 1.62m tall, and weights 54.60kg

Freddie  has ID   562, is 21 years old, 1.74m tall, and weights 78.60kg

Bob      has ID   523, is 17 years old, 1.68m tall, and weights 82.00kg
```

In [21]:
```
# Let's rewind the file again and skip the header
f_csv.seek(0)
next(csv_data)
```

Out[21]:
```
OrderedDict([('id', 'id'),
             ('name', 'name'),
             ('age', 'age'),
             ('height', 'height'),
             ('weight', 'weight')])
```

In [22]:
```
# csv data are tabular data, therefore a 'natural' way to address the data could be:
# my_data[row][label/column] which is what we used for instance for matrices, that are tables
# We can get this representation by reading all data into a list of ordered dictionaries
# and then address individua data based on record/row and label/column
#
tabular_csv = list(csv_data)
# tabular_csv is a list of records in the form of ordered dictionaries
# How do we access the value of field 'name' in record 1?
print(tabular_csv[1]['name'])

# Using the keys we can also make adopt a label agnostic, more "pure" matrix representation:
print(tabular_csv[1][keys[1]])
```

```
Freddie
Freddie
```

In [23]:
```
# Let's print out all data in the file using this notation, and let's print everything
# as it is in the dictionary, that is, as strings. Setting a predefined length in the
# format specifier it allows to have a decently nice formatting
#
for i in range(len(tabular_csv)):
    for k in range(num_of_keys):
        print('{:9s} '.format(tabular_csv[i][keys[k]]), end='')
    print('\n')

f_csv.close()
```

```
768       Alice     20        162       54.6

562       Freddie   21        174       78.6

523       Bob       17        168       82.0
```