# US Social Security data on given names

Eric Martin, CSE, UNSW

COMP9021 Principles of Programming, trimester 1, 2019

```
In [1]: from pathlib import Path
  输入  import os
        import csv                  违约；不付欠款
        from collections import defaultdict
```

Downloaded from https://www.ssa.gov/OACT/babynames/limits.html, the `names` directory contains, besides `NationalReadMe.pdf`, files whose names are of the form `yob****.txt` with "yob" standing for "year of birth" and `****` ranging from `1880` to `2017`. These are csv files, with "csv" standing for "comma separated values": each line consists of 3 fields: a first name, `F` or `M` for female or male, respectively, and a strictly positive integer for the count of newborns who have been given that name in the year whose value is embedded in the file name. All female names are listed before all male names. For a given gender, data are listed in decreasing order of count. For a given gender and count, names are listed in alphabetical order. For instance, for the oldest year, here are the first 10 lines:

```
In [2]: !head names/yob1880.txt
```

用严格正整数表示在值嵌入到文件名中的年份。
csv：代表“逗号分隔值”
‘yob’ standing for “year of birth

```
Mary,F,7065
Anna,F,2604
Emma,F,2003
Elizabeth,F,1939
Minnie,F,1746
Margaret,F,1578
Ida,F,1472
Alice,F,1414
Bertha,F,1320
Sarah,F,1288
```

And here are the last 10 lines:

```
In [3]: !tail names/yob1880.txt
```

```
Unknown,M,5
Vann,M,5
Wes,M,5
Winston,M,5
Wood,M,5
Woodie,M,5
Worthy,M,5
```

```
Wright,M,5
York,M,5
Zachariah,M,5
```

我们的第一个任务是重新组织数据：创建一个目录names_per_gender，创建两个子目录，女性和男性的 names_per_gender，并在每个子目录和名称中的每个.txt fileF中创建一个F的副本，以便

目录 子目录

Our first task is to reorganise the data: create a directory `names_per_gender`, create two subdirectories, `female` and `male`, of `names_per_gender`, and in each of both subdirectories and for each `.txt` file $F$ in `names`, create a copy of $F$ such that:

女性子目录中的F副本将由F中所有女性姓名的行组成，只有2个字段，即名字和计数，所以没有F，第二个字段；

计算

男性子目录中的F副本将由F中所有男性名称的行组成，只有2个字段，即名字和计数，因此没有M，第二个字段。

- the copy of $F$ in the `female` subdirectory will consist of the lines for all female names in $F$ with only 2 fields, namely, first name and count, so without `F`, the second field;
- the copy of $F$ in the `male` subdirectory will consist of the lines for all male names in $F$ with only 2 fields, namely, first name and count, so without `M`, the second field.

To work with directories and files in a platform independent manner, the `Path` class from the `pathlib` module is appropriate. One can create `Path` objects from directory and file names and check whether they exist with `Path`'s `exists()` method. Given a `Path` object $P$ for a directory $D$, `Path` objects for subdirectories of $D$ or for files in $D$ can be created with the `/` operator, with as first and second operands, $P$ and the subdirectory or file name, respectively; `/` will produce path names with a separator that is appropriate for the operating system on which code is executed:

```
In [4]: Path('names'), Path('names').exists()
        Path('names') / 'yob1880.txt', (Path('names') / 'yob1880.txt').exists()
        Path('nonexisting'), Path('nonexisting').exists()
        Path('names') / 'yob1800.txt', (Path('names') / 'yob1800.txt').exists()

Out[4]: (PosixPath('names'), True)

Out[4]: (PosixPath('names/yob1880.txt'), True)

Out[4]: (PosixPath('nonexisting'), False)

Out[4]: (PosixPath('names/yob1800.txt'), False)
```

要以独立于平台的方式处理目录和文件，可以使用pathlib中的Path类模块是适当的。可以从目录和文件名创建Path对象，并使用Path的exists()方法检查它们是否存在。给定目录D的路径对象P，可以用/操作符创建D的子目录或D中的文件的路径对象，分别作为第一和第二操作数，P和子目录或文件名;/会产生路径名称，并使用适合执行程式码的操作系统的分隔符：

We first create a `Path` object for the existing `names` directory, for the to be created `names_per_gender` directory, and for the to be created `female` and `male` subdirectories of `names_per_gender`:

```
In [5]: names_dirname = Path('names')
        names_per_gender_dirname = Path('names_per_gender')
        female_subdirname = names_per_gender_dirname / 'female'
        male_subdirname = names_per_gender_dirname / 'male'
```

我们首先为现有的names目录、将要创建的names_per_gender目录以及将要创建的names_per_gender的女性和男性子目录创建一个Path对象。这样做的目的是创建路径。

os模块的path模块中的exists()函数也允许检查a目录或文件存在。该模块还有其他有用的功能，特别是：

The `exists()` function from the `path` module of the `os` module also allows one to check whether a directory or file exists. That module has other useful functions, in particular:

- `removedirs()`, to remove an empty directory;
- `mkdir()`, to create (make) a directory that does not already exist. 创建(制作)一个不存在的目录

For instance, if the `names_per_gender` directory existed, contained `female` and `male` and no other subdirectories, and both `female` and `male` were empty directories, then the following code fragment would successfully

目录

子目录；[计] 分目录

os.mkdir() 方法用于以数字权限模式创建目录。默认的模式为 0777 (八进制)。
os.mkdir(path[, mode]) path -- 要创建的目录mode -- 要为目录设置的权限数字模式

- remove the `female` directory,
- remove the `male` directory, and
- remove the then empty `names_per_gender` directory.

这将允许接下来的三个对os.mkdir()的调用成功执行，而不需要a
将引发FileExistsError错误

That would allow the next three calls to `os.mkdir()` to execute successfully, without a `FileExistsError` error to be raised:

```
In [6]: if os.path.exists(names_per_gender_dirname):
            os.removedirs(female_subdirname)
            os.removedirs(male_subdirname)
            os.removedirs(names_per_gender_dirname)
        os.mkdir(names_per_gender_dirname)
        os.mkdir(female_subdirname)
        os.mkdir(male_subdirname)
```

We need to process all files in `names` except for `NationalReadMe.pdf`. We could use the `listdir()` function from the `os` module to list all files in `names` and ignore files not ending in `.txt`:

```
In [7]: for file in os.listdir(names_dirname):
            if not file.endswith('.txt'):
                print(file)
```

os.listdir() 方法用于返回指定的文件夹包含的文件或文件夹的名字的列表。这个列表以字母顺序。 它不包括 '.' 和'..' 即使它在文件夹中。

只支持在 Unix, Windows 下使用。

```
NationalReadMe.pdf
```

由于Path类的glob()方法，我们只能生成感兴趣的文件名。

Thanks to the `glob()` method of the `Path` class, we can instead generate only the file names of interest. This method uses Unix syntax to create patterns and match file and directory names:

- `*` to match a (possibly empty) sequence of characters
- `?` to mach a single character
- square brackets to enclose the characters to match.

The following statements illustrate:

```
In [8]: list(names_dirname.glob('*17*'))
        list(names_dirname.glob('*2???7*'))
        list(names_dirname.glob('*2???[357]*'))

Out[8]: [PosixPath('names/yob2017.txt'), PosixPath('names/yob1917.txt')]

Out[8]: [PosixPath('names/yob2017.txt'), PosixPath('names/yob2007.txt')]

Out[8]: [PosixPath('names/yob2015.txt'),
         PosixPath('names/yob2017.txt'),
         PosixPath('names/yob2003.txt'),
         PosixPath('names/yob2007.txt'),
         PosixPath('names/yob2013.txt'),
         PosixPath('names/yob2005.txt')]
```

要提取csv文件的值，当然可以打开该文件，逐行读取，然后将其拆分
行中使用逗号作为分隔符，但是让对象返回是更干净和更健壮的
open()为csv模块的reader()函数的参数;该函数返回一个迭代器
为文件中的每一行生成该行上的值的元组。下面的代码片段说明了，
在yob1880.txt中打印出所有女性或男性姓名大于2000的行数:

提取

To extract the values of a csv file, one can of course open the file, read it line by line, and split each line using the comma as separator, but it is cleaner and more robust to instead, let the object returned by `open()` be the argument of the `reader()` function of the `csv` module; that function returns an iterator to generate for each line in the file, the tuple of values on that line. The following code fragment illustrates, printing out all lines in `yob1880.txt` for counts of female or male name greater than 2000:

```
In [9]: with open(names_dirname / 'yob1880.txt') as file:
            csv_file = csv.reader(file)
            for name, gender, tally in csv_file:
                if int(tally) > 2_000:
                    print(name, gender, tally)
```

```
Mary F 7065
Anna F 2604
Emma F 2003
John M 9655
William M 9532
James M 5927
Charles M 5348
George M 5126
Frank M 3242
Joseph M 2632
Thomas M 2534
Henry M 2444
Robert M 2415
Edward M 2364
Harry M 2152
```

像yob188 .txt这样的文件将作为名称中的文件之一处理，其路径由 glob()应用于names_dirname。从yob1880.txt中的行中提取的名称和计数 的子目录中，以yob188 .txt的名称写入这两个文件之一
names_per_gender。由于Path对象的name属性，这两个文件的路径都很方便 从路径创建到yob188 .txt的名称:

A file such as `yob1880.txt` is to be processed as one of the files in `names` whose paths are generated by `glob()` applied to `names_dirname`. Names and counts extracted from the rows in `yob1880.txt` are to be written to one of both files with the name `yob1880.txt` located in the `female` and `male` subdirectories of `names_per_gender`. Thanks to the `name` attribute of a `Path` object, the paths to both files are conveniently created from the path to `yob1880.txt` in `names`:

```
In [10]: filename = next(names_dirname.glob('*1880*'))

         filename
         filename.parent
         filename.name
         female_subdirname / filename.name
         male_subdirname / filename.name

Out[10]: PosixPath('names/yob1880.txt')

Out[10]: PosixPath('names')

Out[10]: 'yob1880.txt'

Out[10]: PosixPath('names_per_gender/female/yob1880.txt')
```

对于名称中的每个.txt文件F，我们打开一个with语句F(用于阅读)和两个在names_per_gender的子目录female和male中，文件FF和FM的名称与F相同，分别创建FF和FM路径，用于编写目的。在平行于使用csv.reader()，我们使用csv.writer()在csv文件中写入数据行用逗号适当分隔的连续值。字典csv_file_per_gender允许这样做选择要写入FF或FM中的哪一个。在下面代码片段的最后一行中 赋值给_的唯一目的是抑制木星的输出：

```
Out[10]: PosixPath('names_per_gender/male/yob1880.txt')
```

For each `.txt` file $F$ in `names`, we open, with a single `with` statement, $F$ for reading purposes, and two files $F_F$ and $F_M$ with the same name as $F$ in the subdirectories `female` and `male` of `names_per_gender`, respectively, for writing purposes, with the paths to $F_F$ and $F_M$ created as just described. In parallel to making use of `csv.reader()`, we make use of `csv.writer()` to write rows of data in a csv file, with successive values properly separated with commas. The dictionary `csv_file_per_gender` allows one to choose which one of $F_F$ or $F_M$ should be written to. In the last line of the following code fragment, the only purpose of the assignment to _ is to suppress Jupyter output:

```
In [11]: for filename in names_dirname.glob('*.txt'):
             with open(filename) as file,\
                  open(female_subdirname / filename.name, 'w') as female_file,\
                  open(male_subdirname / filename.name, 'w') as male_file:
                 csv_file = csv.reader(file)
                 female_csv_file = csv.writer(female_file)
                 male_csv_file = csv.writer(male_file)
                 csv_file_per_gender = {'F': female_csv_file, 'M': male_csv_file}
                 for name, gender, tally in csv_file:
                     _ = csv_file_per_gender[gender].writerow((name, tally))
```

Our second task is to find out the longest intervals of time that separate the years $Y_1$ and $Y_2$ when a name was given (as a male or female name) in both $Y_1$ and $Y_2$, but not in-between. We would like to output the top 10 longest intervals together with the years that start and end the interval, and together with the name that was "forgotten and revived" in that time interval.

To this aim, it is convenient to create a dictionary whose keys are names, with for a given key $N$, the list of years, from oldest to most recent, when $N$ was given once at least. For instance, here are the years when `Franc` was given as a name:

我们的第二个任务是找出Y1和Y2年之间最长的时间间隔，在这段时间里，一个名字同时以Y1和Y2的形式(以男性或女性的名字)出现，而不是在Y1和Y2之间。我们希望输出前10个最长的时间间隔，以及开始和结束时间间隔的年份，以及在该时间间隔中被“遗忘和恢复”的名称。
为了达到这个目的，创建一个键是名称的字典是很方便的，对于给定的键N，它是年份的列表，从最老的到最近的，当N至少被给定一次时。例如，以下是法郎作为名字的年代：

```
In [12]: !grep ^Franc, names/*

names/yob1882.txt:Franc,F,5
names/yob1883.txt:Franc,F,5
names/yob2001.txt:Franc,M,5
names/yob2002.txt:Franc,M,6
names/yob2013.txt:Franc,M,5
```

So `'Franc'` should be one of the keys, with as value `[1882, 1883, 2001, 2002, 2013]`.

Years will be added one by one to the lists of values as files are processed one by one. Using a simple dictionary, one has to distinguish between creating a key and a value, that should be a list with a single year, and adding a new year to the list that is the value of an existing key:

当文件被一个一个地处理时，年数将一个一个地添加到值列表中。使用一个简单的字典，一个人必须区分创建一个键和一个值，这应该是一个列表与一个单一年，并将new year添加到列表中，该列表是现有键的值

```
In [13]: name = 'Franc'
         years_per_name = {}
         for year in 1882, 1883, 2001, 2002, 2003:
             if not name in years_per_name:
                 years_per_name[name] = [year]
                 print(f'Processing year {year}: '
```

```
                    f'creating key "{name}" and value [{year}]'
                   )
            else:
                years_per_name[name].append(year)
                print(f'Processing year {year}: '
                      f'appending {year} to value for key "{name}"'
                     )

        years_per_name
```

```
Processing year 1882: creating key "Franc" and value [1882]
Processing year 1883: appending 1883 to value for key "Franc"
Processing year 2001: appending 2001 to value for key "Franc"
Processing year 2002: appending 2002 to value for key "Franc"
Processing year 2003: appending 2003 to value for key "Franc"
```

```
Out[13]: {'Franc': [1882, 1883, 2001, 2002, 2003]}
```

A `KeyError` error is generated when trying to access a nonexisting key:

当试图访问一个不存在的密钥时，会生成一个KeyError错误：

```
In [14]: name = 'Franc'
         years_per_name = {}
         years_per_name[name]
```

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

        KeyError                                  Traceback (most recent call last)

        <ipython-input-13-dce68830c349> in <module>()
          1 name = 'Franc'
          2 years_per_name = {}
    ----> 3 years_per_name[name]


        KeyError: 'Franc'
```

When using a `defaultdict` from the `collections` module, trying to access a nonexisting key creates the key, together with the default value for the class provided as argument to `defaultdict`:

当使用集合模块中的defaultdict时，尝试访问一个不存在的键创建键，以及作为defaultdict参数提供的类的默认值

```
In [15]: name = 'Franc'

         years_per_name = defaultdict(int)
         print('Creating a key with 0 as default value:')
         years_per_name[name];
         years_per_name
         print('Creating a key with 0 as default value, immediately modified:')
```

```
        years_per_name = defaultdict(int)
        years_per_name[name] += 2; years_per_name

        years_per_name = defaultdict(list)
        print('Creating a key with [] as default value:')
        years_per_name[name]
        years_per_name
        print('Creating a key with [] as default value, immediately modified:')
        years_per_name[name].append(1882); years_per_name
```

```
Creating a key with 0 as default value:


Out[15]: 0

Out[15]: defaultdict(int, {'Franc': 0})
```

缺省值；省略补充　　立即修改

```
Creating a key with 0 as default value, immediately modified:


Out[15]: defaultdict(int, {'Franc': 2})

Creating a key with [] as default value:


Out[15]: []

Out[15]: defaultdict(list, {'Franc': []})

Creating a key with [] as default value, immediately modified:


Out[15]: defaultdict(list, {'Franc': [1882]})
```

Thanks to default dictionaries, the key `'Franc'` can be created and years incrementally added to the value list as follows:

由于默认字典，键'Franc'可以创建，并逐年添加到值列表如下

```
In [16]: name = 'Franc'
         years_per_name = defaultdict(list)
         for year in 1882, 1883, 2001, 2002, 2003:
             years_per_name[name].append(year)

         years_per_name

Out[16]: defaultdict(list, {'Franc': [1882, 1883, 2001, 2002, 2003]})
```

Extracting years from filenames is easy:

```
In [17]: int('yob1880.txt'[3: 7])

Out[17]: 1880
```

因此，创建完整的字典可以如下所示;我们只需注意glob()不会
返回按字母顺序排列的文件名，因此我们使用ordered()，因为组成years_per_name给定键值的年份必须从最老的年份排序到最近的
年份

So creating the full dictionary can be done as follows; we only have to beware that `glob()` does not return the file names in alphabetical order, so we use `sorted()` as it is essential that the years that make up the value of a given key of `years_per_name` are sorted from oldest to most recent:

```
In [18]: years_per_name = defaultdict(list)
         for filename in sorted(names_dirname.glob('*.txt')):
             year = int(filename.name[3: 7])
             with open(filename) as file:
                 csv_file = csv.reader(file)
                 for name, _, _ in csv_file:
                     years_per_name[name].append(year)

         years_per_name['Franc']
```

```
Out[18]: [1882, 1883, 2001, 2002, 2013]
```

从years_per_name,我们可以创建一个列表的三元组的形式(D, Y, N)，D是一年不同,Y是一年开始每年不同的D（D可以添加和产量今年结束今年的区别),和N是一个名称是Y,只有D年年后

From `years_per_name`, we can create a list of triples of the form $(D, Y, N)$ where $D$ is a year difference, $Y$ is a year that starts a year difference of $D$ (to which $D$ can be added and yield the year that ends the year difference), and $N$ is a name that was given in year $Y$ and only $D$ years later:

```
In [19]: revivals = [[years_per_name[name][i + 1] - years_per_name[name][i],
                      years_per_name[name][i],
                      name
                     ] for name in years_per_name
                       for i in range(len(years_per_name[name]) - 1)
                    ]

         [revival for revival in revivals if revival[0] == 2001 - 1883]
```

```
Out[19]: [[118, 1883, 'Franc']]
```

按相反的顺序对恢复进行排序，将得到一个列表，其中：
• 年份差异由大到小排序;
• 对于给定的年份差异，开始年份差异的年份顺序是从最近年份到最老年份;
• 对于给定的年份差异和开始年份差异的年份，名称按反词典顺序排列：

Sorting `revivals` in reversed order results in a list where:

- year differences are ordered from largest to smallest;
- for a given year difference, years that start the year difference are ordered from most recent to oldest;
- for a given year difference and year that starts the year difference, names are ordered in anti-lexicographic order:

```
In [20]: revivals.sort(reverse = True)
         for i in range(10):
             print(f'{revivals[i][2]} was last used in {revivals[i][1]} '
                   f'and then again in {revivals[i][1] + revivals[i][0]}, '
                   f'{revivals[i][0]} years later.'
                  )
```

```
Franc was last used in 1883 and then again in 2001, 118 years later.
Izzie was last used in 1891 and then again in 2006, 115 years later.
Rasmus was last used in 1888 and then again in 2003, 115 years later.
Izma was last used in 1899 and then again in 2007, 108 years later.
```

Leannah was last used in 1889 and then again in 1996, 107 years later.
Almar was last used in 1915 and then again in 2017, 102 years later.
Addiemae was last used in 1915 and then again in 2017, 102 years later.
Saidee was last used in 1893 and then again in 1995, 102 years later.
Onah was last used in 1916 and then again in 2017, 101 years later.
Tabea was last used in 1915 and then again in 2016, 101 years later.