C: Basic grouping and aggregation

Computing results for groups of records

- Known as **grouping** and **aggregation**
- Very common analytical operations

Examples:

Group **counties by state** and find **total population**
Group **individuals by characteristic** and find **median income**

Example 1: grouping states by different time zone

Input data:

name	timezone
New York	ET
New Jersey	ET
Texas	CT
Illinois	CT
Colorado	MT
Utah	MT
California	PT
Washington	PT

Want to group **by time zone** and **list**:

Timezone	States
CT	Illinois, Texas
ET	New Jersey, New York

MT	Colorado, Utah
PT	California, Washington

Approach:

1. Set up a dictionary for grouping:
    - Keys are time zones and values are empty lists

```
{
   'CT': [],
   'ET': [],
   'MT': [],
   'PT': []
}
```

2. Loop through the data and **append each name** to **its time zone's list**
    - After first two records:

```
{
   'CT': [],
   'ET': ['New York','New Jersey'],
   'MT': [],
   'PT': []
}
```

3. Sort and print the results

Look at code shortly

Example 2: computing median population density

Input data:

state	county	ALAND	pop
Alabama	Autauga Count	1539602123	55200.0
Alabama	Baldwin County	4117546676	208107.0
Alabama	Barbour County	2292144655	25782.0
Alabama	Bibb County	1612167481	22527.0
…	…	…	…

Want to compute density then group **by state** and **find median**, and then sort:

Density in people per square km

State	Density
…	…
Alabama	18.54
Wisconsin	18.91
…	…

Approach:

1. Loop through the data computing the density of each county
2. Set up a dictionary for grouping:
    - Keys are state names and values are empty lists

```
{
   'Alabama': [],
```

```
  'Alaska': [],
  'Arizona': [],
  ...
  'Wyoming': []
}
```

3. Loop through the data and **append each density** to **its state's list**

   density = pop / (ALAND/1e6)

state	county	ALAND	pop	density
Alabama	Autauga Count	1539602123	55200.0	35.85
Alabama	Baldwin County	4117546676	208107.0	50.54
Alabama	Barbour County	2292144655	25782.0	11.25
Alabama	Bibb County	1612167481	22527.0	13.97
…	…	…	…	

   State lists after first four records:

```
{
  'Alabama': [35.85, 50.54, 11.25, 13.97],
  'Alaska': [],
  'Arizona': [],
  ...
  'Wyoming': []
}
```

4. Compute the median of each state's list

5. Sort and print the results

Look at code shortly

Useful Python tools

defaultdict():

- Creates dictionaries with default entries for keys that aren't set

Using a dictionary of lists without defaultdict():

```
state_list = {}                                  1. Create empty dictionary
…
if 'Alabama' not in state_list:                  2. Is key in dictionary?
    state_list['Alabama'] = []                       3. No, create a blank list
state_list['Alabama'].append( 35.85 )            4. Append the data
```

Using a dictionary of lists with defaultdict:

```
state_list = defaultdict(list)                   1. Create empty dictionary of lists
…
state_list['Alabama'].append( 12.34 )            2. Append the data
```

First reference to state_list['Alabama'] creates empty list automatically

tuples:

- Two or more values enclosed in parentheses
- Similar to a list but can't be changed

```
person1 = ( "Jones", "Alice" )
person2 = ( "Jones", "Bob" )
```

Many uses, including as dictionary keys:

```
salary = {
   ("Smith","Clara"): 72015,
   ("Jones","Beth"):  80145,
   ("Jones","Allan"): 65234
}
```

Lists of tuples are automatically sorted hierarchically:

```
keys = salary.keys()

keys = [
   ("Smith","Clara"),
   ("Jones","Beth"),
   ("Jones","Allan")
]

by_name = sorted( keys )

by_name = [
   ("Jones","Allan"),
   ("Jones","Beth"),
   ("Smith","Clara")
]
```

f-strings:

- Streamlined formatting for strings
- First quote is preceeded by f
- Embedded variable names enclosed in {} will be replace by values

```
last = "Jones"
```

```
first = "Beth"
value = 80145
print( f'The salary of {first} {last} is {value}' )
```

Prints: The salary of Beth Jones is 80145