Dictionaries

- Consider this function
  - A mini English-Spanish dictionary

```
def translate(color):
    if color == "red":
        return "rojo"
    elif color == "blue":
        return "aloz"
    elif color == "green":
        return "verdi"
    elif color == "white":
        return "blanco"
```

- A function of this type is called a *mapping*
- Python has an efficient and flexible mapping device, called a *dictionary*

```
translate = {"red":"rojo", "blue":"aloz", "green":"verdi", "white":"blanco"}
```

  - The value of *translate["blue"]* is *"aloz"*

Dictionaries

- A dictionary organizes information by **association**, not position
  - Example: When you use a dictionary to look up the definition of “mammal,” you don’t start at page 1; instead, you turn to the words beginning with “M”
- Data structures organized by association are also called **tables** or **association lists**
- In Python, a **dictionary** associates a set of **keys** with data values

Dictionaries

- A Python dictionary is written as a sequence of key/value pairs separated by commas
  - Pairs are sometimes called **entries**
  - Enclosed in curly braces (**{** and **}**)
  - A colon (:) separates a key and its value
- Examples:

```
{'Sarah':'476-3321', 'Nathan':'351-7743'}    #A Phone book
{'Name':'Molly', 'Age':18}                   #Personal information
{}                                           #An empty dictionary
```

- Value associated with *key1* is given by the expression *dictionaryName*[*key1*].

Dictionaries

- Keys can be data of any immutable types, including other data structures
- Add a new key/value pair to a dictionary using `[]`: `<a dictionary>[<a key>] = <a value>`
- Example:

```
>>> info = {}
>>> info["name"] = "Sandy"
>>> info["occupation"] = "hacker"
>>> info
{'name': 'Sandy', 'occupation': 'hacker'}
>>>
```

- Use `[]` also to replace a value at an existing key:

```
>>> info["occupation"] = "manager"
>>> info
{'name': 'Sandy', 'occupation': 'manager'}
>>>
```

Dictionaries

- Use `[]` to obtain the value associated with a key
  - If key is not present in dictionary, an error is raised

```
>>> info["name"]
'Sandy'
>>> info["job"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'job'
>>>
```

- If the existence of a key is uncertain, test for it using the dictionary method **`has_key`**
  - Easier strategy is to use the method **`get`**

```
>>> print(info.get("job", None))
None
>>>
```

Dictionaries

- To delete an entry from a dictionary, remove its key using the method **`pop`**
  - **`pop`** expects a key and an optional default value as arguments

```
>>> print(info.pop("job", None))
None
>>> print(info.pop("occupation"))
manager
>>> info
{'name': 'Sandy'}
>>>
```

Dictionaries

- To print all of the keys and their values:

```
for key in info:
   print(key, info[key])
```

- Alternative: Use the dictionary method **`items()`**

```
>>> grades = {90:"A", 80:"B", 70:"C"}
>>> grades.items()
[(80, 'B'), (90, 'A'), (70, 'C')]
```

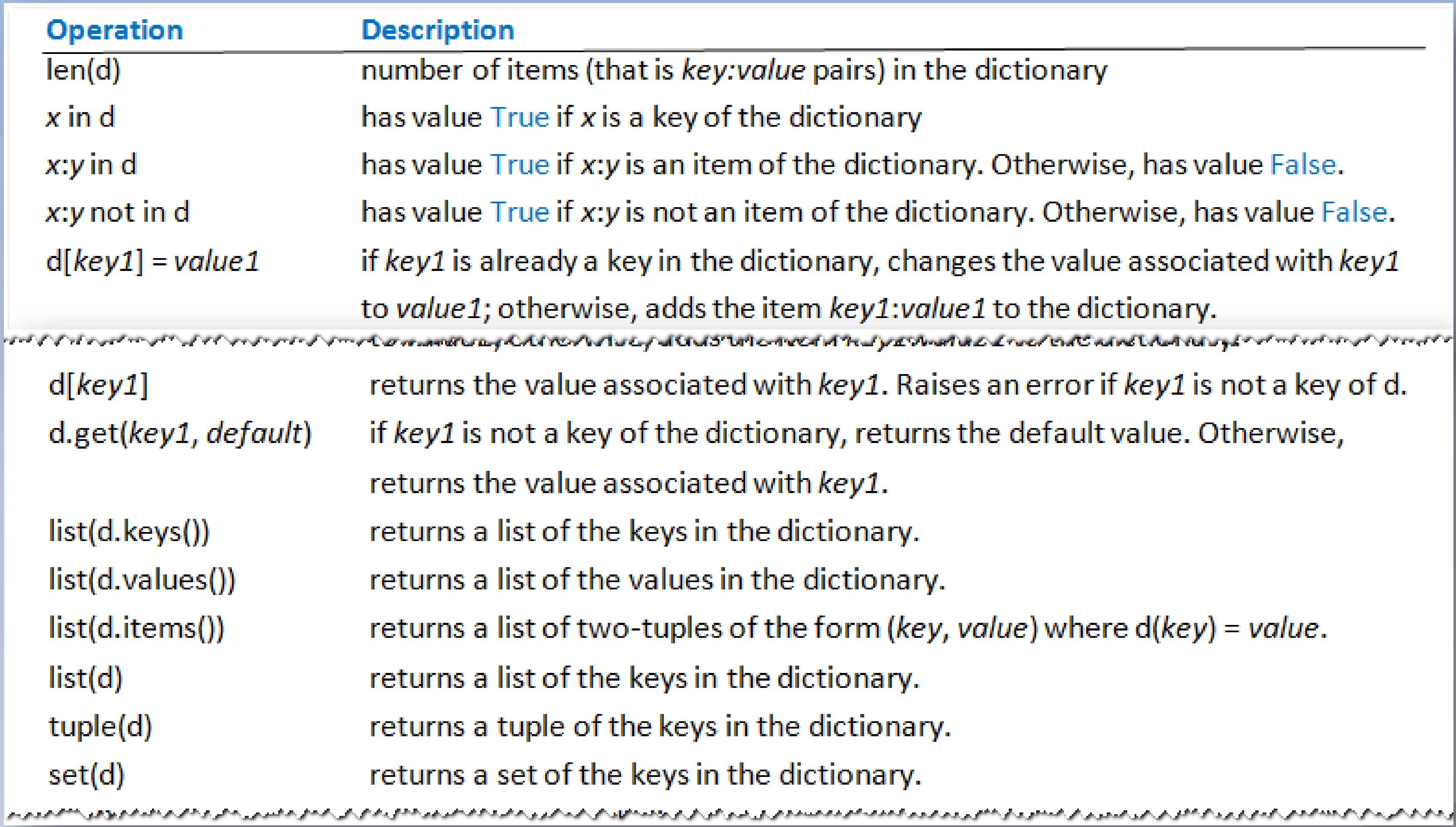
Entries are represented as tuples within the list

```
for (key, value) in grades.items():
   print(key, value)
```

- You can sort the list first:

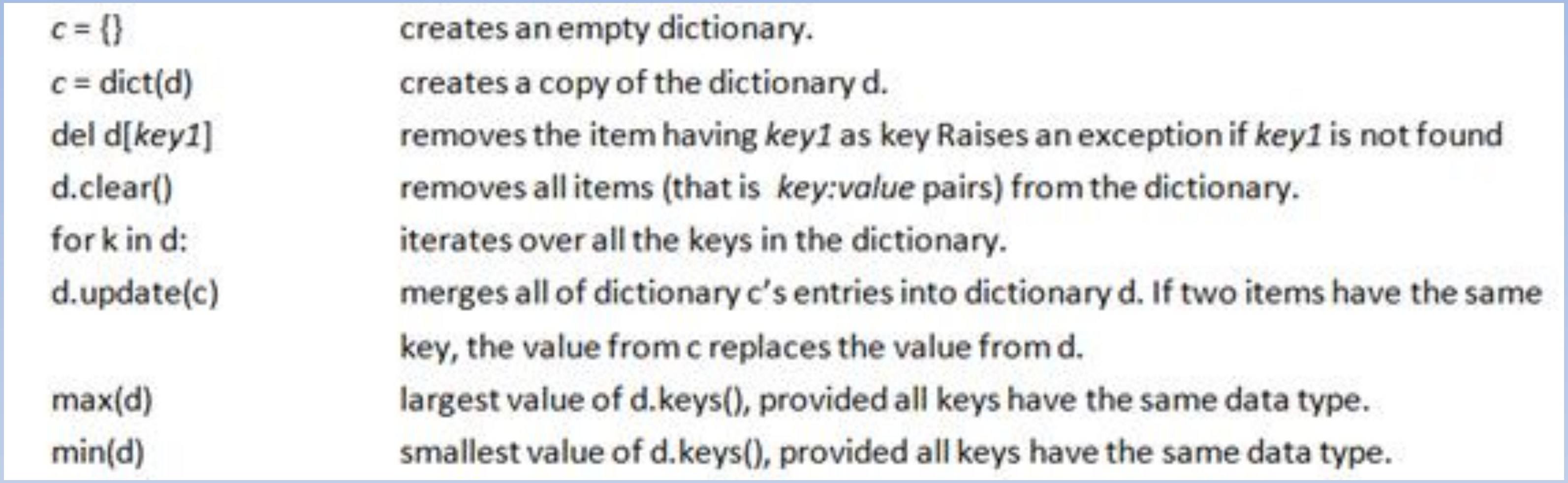
```
theKeys = list(info.keys())
theKeys.sort()
for key in theKeys:
   print(key, info[key])
```

Dictionary Operations



Operation	Description
len(d)	number of items (that is *key:value* pairs) in the dictionary
x in d	has value True if *x* is a key of the dictionary
x:*y* in d	has value True if *x*:*y* is an item of the dictionary. Otherwise, has value False.
x:*y* not in d	has value True if *x*:*y* is not an item of the dictionary. Otherwise, has value False.
d[*key1*] = *value1*	if *key1* is already a key in the dictionary, changes the value associated with *key1* to *value1*; otherwise, adds the item *key1*:*value1* to the dictionary.
d[*key1*]	returns the value associated with *key1*. Raises an error if *key1* is not a key of d.
d.get(*key1*, *default*)	if *key1* is not a key of the dictionary, returns the default value. Otherwise, returns the value associated with *key1*.
list(d.keys())	returns a list of the keys in the dictionary.
list(d.values())	returns a list of the values in the dictionary.
list(d.items())	returns a list of two-tuples of the form (*key*, *value*) where d(*key*) = *value*.
list(d)	returns a list of the keys in the dictionary.
tuple(d)	returns a tuple of the keys in the dictionary.
set(d)	returns a set of the keys in the dictionary.

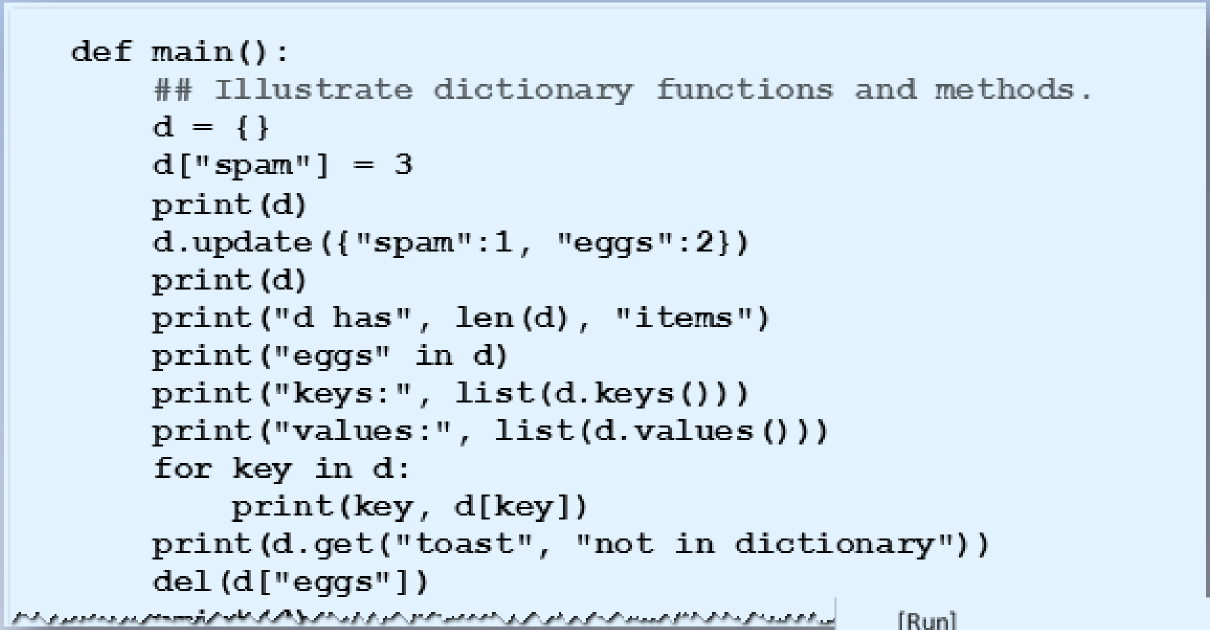
Dictionary Operations



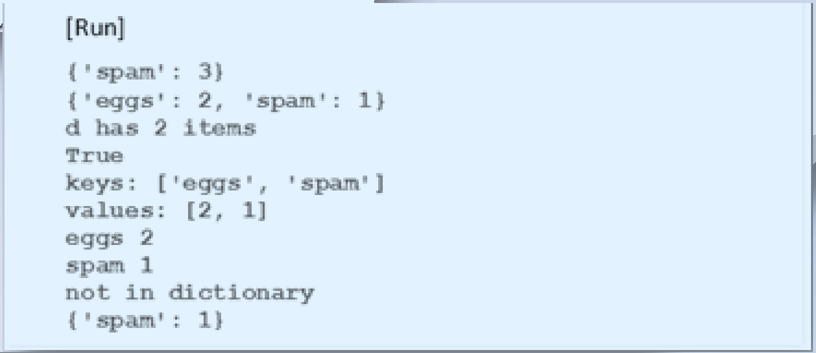
c = {}	creates an empty dictionary.
c = dict(d)	creates a copy of the dictionary d.
del d[*key1*]	removes the item having *key1* as key Raises an exception if *key1* is not found
d.clear()	removes all items (that is *key:value* pairs) from the dictionary.
for k in d:	iterates over all the keys in the dictionary.
d.update(c)	merges all of dictionary c's entries into dictionary d. If two items have the same key, the value from c replaces the value from d.
max(d)	largest value of d.keys(), provided all keys have the same data type.
min(d)	smallest value of d.keys(), provided all keys have the same data type.

Dictionaries

- Example: Program illustrates many of the functions and methods for dictionaries.



```
def main():
    ## Illustrate dictionary functions and methods.
    d = {}
    d["spam"] = 3
    print(d)
    d.update({"spam":1, "eggs":2})
    print(d)
    print("d has", len(d), "items")
    print("eggs" in d)
    print("keys:", list(d.keys()))
    print("values:", list(d.values()))
    for key in d:
        print(key, d[key])
    print(d.get("toast", "not in dictionary"))
    del(d["eggs"])
```



[Run]

```
{'spam': 3}
{'eggs': 2, 'spam': 1}
d has 2 items
True
keys: ['eggs', 'spam']
values: [2, 1]
eggs 2
spam 1
not in dictionary
{'spam': 1}
```

Creating a Dictionary from a Text File

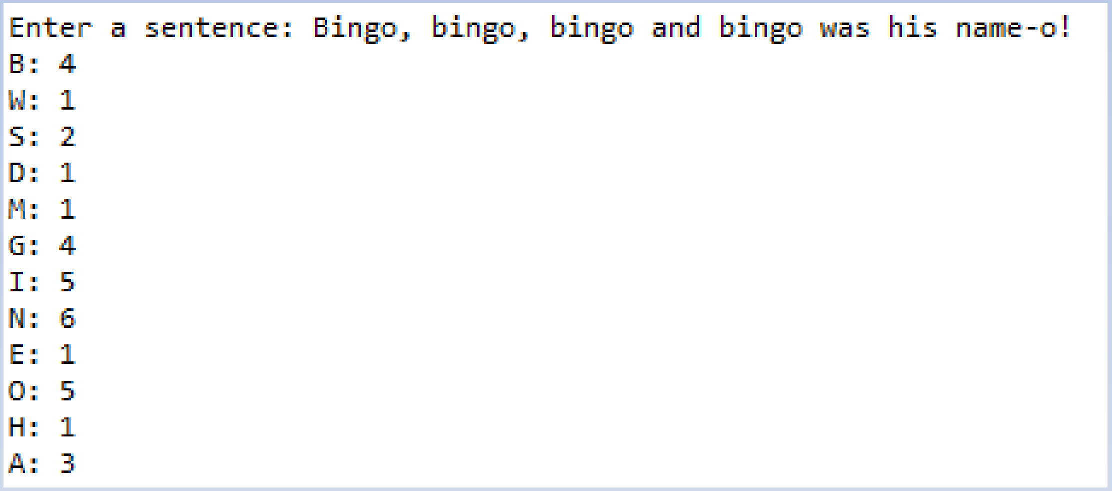
- Example: Program with long *if-elif* statement can be simplified with use of a dictionary.

```
def main():
    ## Determine an admission fee based on age group.
    print("Enter the person's age group ", end="")
    ageGroup = input("(child, minor, adult, or senior): ")
    print("The admission fee is", determineAdmissionFee(ageGroup), "dollars." )

def determineAdmissionFee(ageGroup):
    if ageGroup == "child":      # age < 6
        return 0                 # free
    elif ageGroup == "minor":    # age 6 to 17
        return 5                 # $5
    elif ageGroup == "adult":    # age 18 to 64
        return 10
    elif ageGroup == "senior":   # age >= 65
        return 8

     ...
```

```
def determineAdmissionFee(ageGroup):
    dict = {"child":0, "minor":5, "adult":10, "senior":8}
    return dict[ageGroup]
```

Case Study: Letter Frequency

- Use the dictionary object as a way to count frequency of letters in a sentence
- Skip all non-alphabetic characters

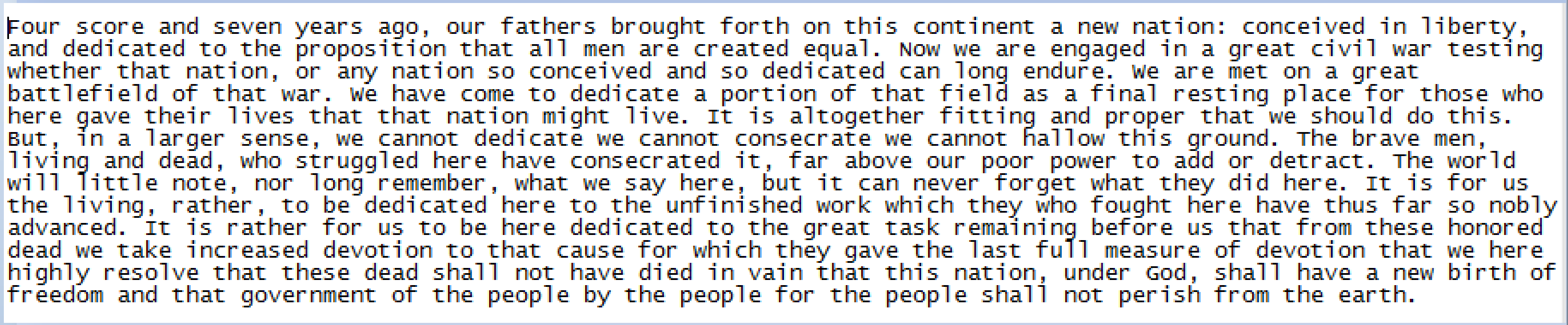


```
Enter a sentence: Bingo, bingo, bingo and bingo was his name-o!
B: 4
W: 1
S: 2
D: 1
M: 1
G: 4
I: 5
N: 6
E: 1
O: 5
H: 1
A: 3
```

- Can you sort the frequency list?

Using Dictionary as Frequency Table

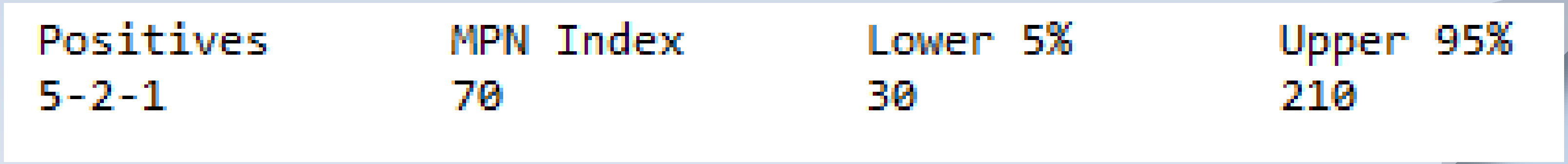
Gettysburg Address



Four score and seven years ago, our fathers brought forth on this continent a new nation: conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war testing whether that nation, or any nation so conceived and so dedicated can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate we cannot consecrate we cannot hallow this ground. The brave men, living and dead, who struggled here have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember, what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion that we here highly resolve that these dead shall not have died in vain that this nation, under God, shall have a new birth of freedom and that government of the people by the people for the people shall not perish from the earth.

Case Study – MPN Data

- Microbiologists estimating the number of bacteria in a sample that contains bacteria that do not grow well on solid media may use a statistical technique called the most probable number (MPN) method.
- Five tubes of nutrient medium receives 10 ml of the sample
- A second set of five tubes receives 1 ml of sample per tube
- A third set of five tubes, only 0.1 ml of sample is placed
- Each tube in which bacterial growth is observed is recorded as a positive, and the numbers for the three groups are combined to create a triplet such as 5-2-1, which means that all five tubes receiving 10 ml of sample show bacterial growth, only two tubes in the 1-ml group show growth, and only one of the 0.1-ml group is positive
- A microbiologist would use this combination –of-positive triplet as an index to a table to determine that the most probable number of bacteria as well as lower and upper bounds in a majority (95%) of the samples



```
Positives       MPN Index       Lower 5%        Upper 95%
5-2-1           70              30              210
```

Case Study – MPN Data

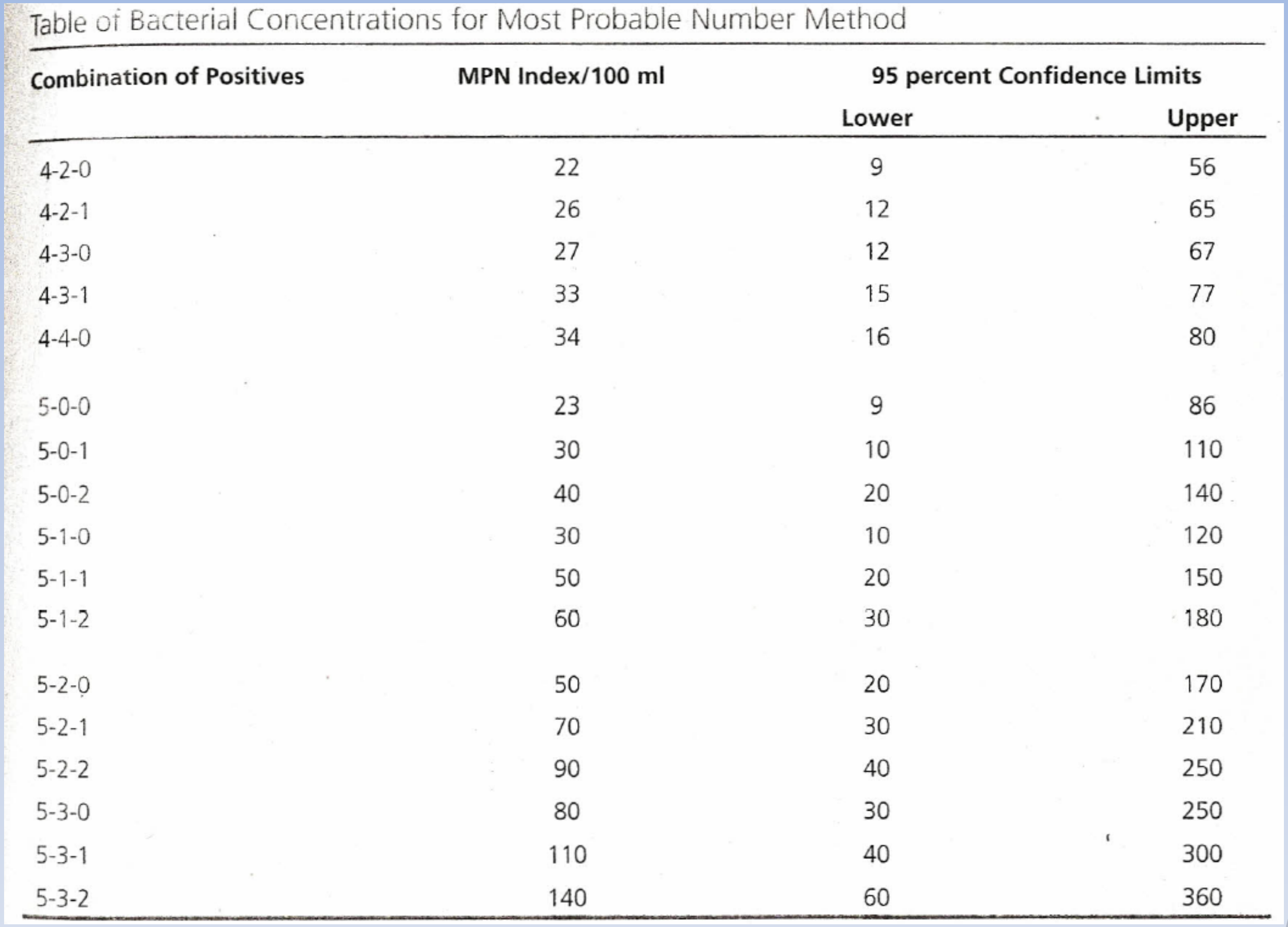
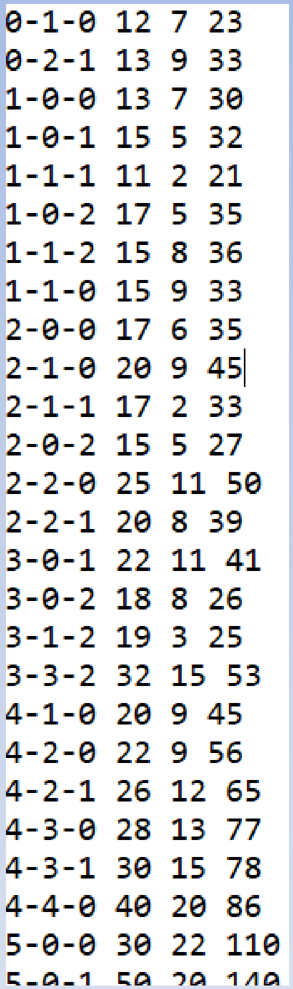


Table of Bacterial Concentrations for Most Probable Number Method

Combination of Positives	MPN Index/100 ml	95 percent Confidence Limits	
		Lower	Upper
4-2-0	22	9	56
4-2-1	26	12	65
4-3-0	27	12	67
4-3-1	33	15	77
4-4-0	34	16	80
5-0-0	23	9	86
5-0-1	30	10	110
5-0-2	40	20	140
5-1-0	30	10	120
5-1-1	50	20	150
5-1-2	60	30	180
5-2-0	50	20	170
5-2-1	70	30	210
5-2-2	90	40	250
5-3-0	80	30	250
5-3-1	110	40	300
5-3-2	140	60	360

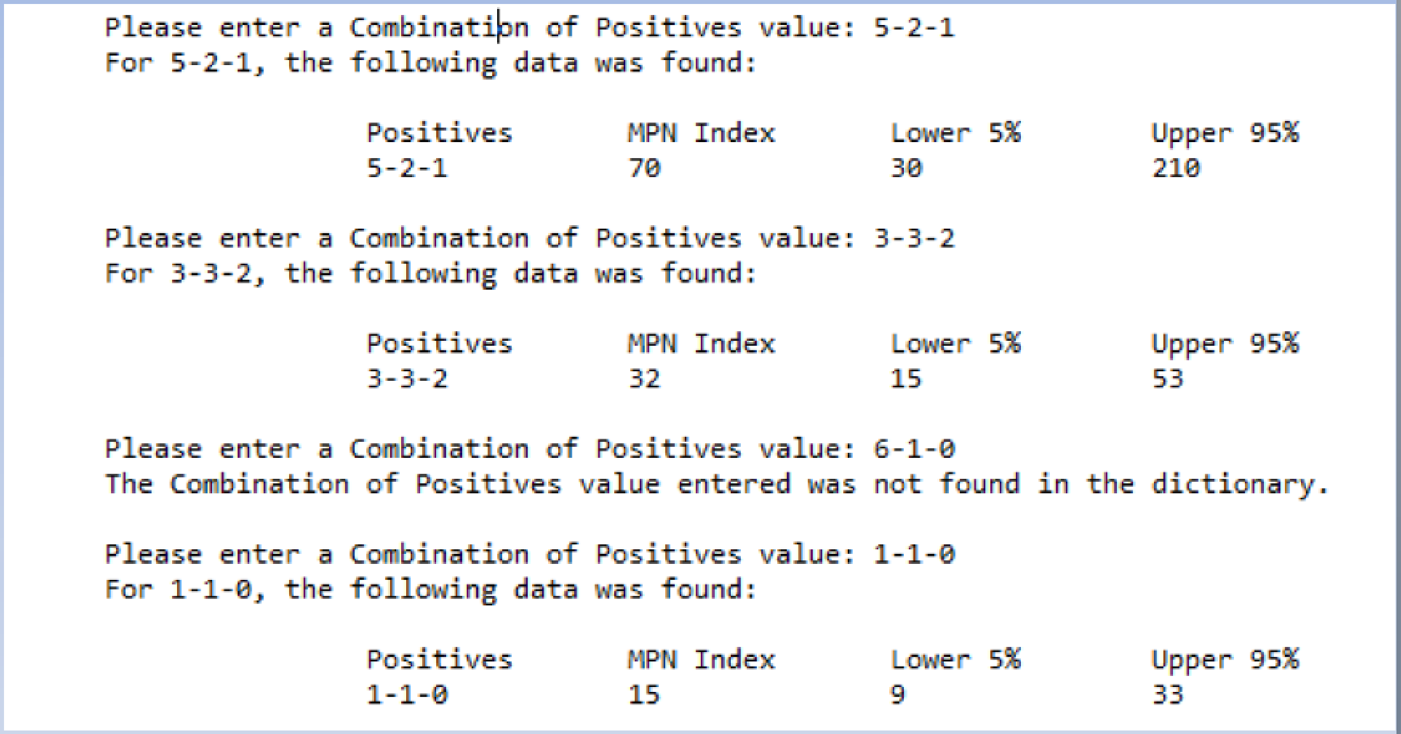
Case Study – MPN Data

Input Sample



```
0-1-0 12 7 23
0-2-1 13 9 33
1-0-0 13 7 30
1-0-1 15 5 32
1-1-1 11 2 21
1-0-2 17 5 35
1-1-2 15 8 36
1-1-0 15 9 33
2-0-0 17 6 35
2-1-0 20 9 45
2-1-1 17 2 33
2-0-2 15 5 27
2-2-0 25 11 50
2-2-1 20 8 39
3-0-1 22 11 41
3-0-2 18 8 26
3-1-2 19 3 25
3-3-2 32 15 53
4-1-0 20 9 45
4-2-0 22 9 56
4-2-1 26 12 65
4-3-0 28 13 77
4-3-1 30 15 78
4-4-0 40 20 86
5-0-0 30 22 110
5-0-1 50 20 140
```

Program Execution



```
Please enter a Combination of Positives value: 5-2-1
For 5-2-1, the following data was found:

                Positives       MPN Index       Lower 5%        Upper 95%
                5-2-1           70              30              210

Please enter a Combination of Positives value: 3-3-2
For 3-3-2, the following data was found:

                Positives       MPN Index       Lower 5%        Upper 95%
                3-3-2           32              15              53

Please enter a Combination of Positives value: 6-1-0
The Combination of Positives value entered was not found in the dictionary.

Please enter a Combination of Positives value: 1-1-0
For 1-1-0, the following data was found:

                Positives       MPN Index       Lower 5%        Upper 95%
                1-1-0           15              9               33
```