BÀI TẬP THỰC HÀNH MÔN LẬP TRÌNH PYTHON NÂNG CAO

- ❖ Bài tập được thiết kế theo từng lab, mỗi lab là 3 tiết có sự hướng dẫn của GV.
- ❖ Cuối mỗi buổi thực hành, sinh viên nộp lại phần bài tập mình đã thực hiện cho GV hướng dẫn.
- ❖ Những câu hỏi mở rộng/khó giúp sinh viên trau dồi thêm kiến thức của môn học. Sinh viên phải có trách nhiệm nghiên cứu, tìm câu trả lời nếu chưa thực hiện xong trong giờ thực hành.

Nội dung

<u>**LAB 1:**</u> **Ngôn ngữ lập trình Python và modules**

Nội dung:

1.1 Ngôn ngữ lập trình Python
1.2 Thiết kế chương trình phần mềm theo modules và sử dụng modules
1.3 Tự thiết kế modules và xử lý lỗi
1.4 Modules, Lớp (Classes) và Phương thức (Methods)
1.5 Sử dụng Methods theo hướng đối tượng

Bài tập chương 3:

1. Two of Python's built-in functions are min and max. In the Python shell, execute the following function calls:
   a. min(2, 3, 4)
   b. max(2, -3, 4, 7, -5)
   c. max(2, -3, min(4, 7), -5)
2. For the following function calls, in what order are the subexpressions evaluated?
   a. min(max(3, 4), abs(-5))
   b. abs(min(4, 6, max(2, 8)))
   c. round(max(5.572, 3.258), abs(-2))
3. Following the function design recipe, define a function that has one parameter, a number, and returns that number tripled

```python
def triple(num):
    """ (number) -> number
    Return num tripled.
    >>> triple(3)
    9
    """
    return num * 3
```

4. Following the function design recipe, define a function that has two parameters, both of which are numbers, and returns the absolute value of the difference of the two. Hint: Call built-in function abs.

```python
def absolute_difference(number1, number2):
    """ (number, number) -> number
    Return the absolute value of the difference between number1
    and number2.
    >>> absolute_difference(3, 7)
    4
    """
    return abs(number1 - number2)
```

5. Following the function design recipe, define a function that has one parameter, a distance in kilometers, and returns the distance in miles. (There are 1.6 kilometers per mile.)

```
def km_to_miles(km):
    """ (number) -> float
    Return the distance km in miles.
    >>> km_to_miles(5)
    3.125
    """
    return km / 1.6
```

6. Following the function design recipe, define a function that has three parameters, grades between 0 and 100 inclusive, and returns the average of those grades.

```
def average_grade(grade1, grade2, grade3):
    """ (number, number, number) -> number
    Return the average of the grade1, grade2, and grade3, where
    each grade ranges from 0 to 100, inclusive.
    >>> average_grade(80, 95, 90)
    88.33333333333333
    """
    return (grade1 + grade2 + grade3) / 3
```

7. Following the function design recipe, define a function that has four parameters, all of them grades between 0 and 100 inclusive, and returns the average of the *best 3* of those grades. Hint: Call the function that you defined in the previous exercise.

```
def top_three_avg(grade1, grade2, grade3, grade4):
    """ (number, number, number, number) -> number
    Return the average of the top three of grades grade1, grade2,
    grade3, and grade4.
    >>> top_three_avg(50, 60, 70, 80)
    70
    """
    # Here is one solution that does not use average_grade from Q6.
    total = grade1 + grade2 + grade3 + grade4
    top_three = total - min(grade1, grade2, grade3, grade4)
    return top_three / 3
    # Here is a different solution that does use the function from Q6.
    return max(average_grade(grade1, grade2, grade3),
    average_grade(grade1, grade2, grade4),
    average_grade(grade1, grade3, grade4),
    average_grade(grade2, grade3, grade4))

    return (grade1 + grade2 + grade3) / 3
```

8. Complete the examples in the docstring and then write the body of the following function:

```
def weeks_elapsed(day1, day2):
""" (int, int) -> int
day1 and day2 are days in the same year. Return the number of full weeks
that have elapsed between the two days.
>>> weeks_elapsed(3, 20)
2
>>> weeks_elapsed(20, 3)
2
>>> weeks_elapsed(8, 5)
>>> weeks_elapsed(40, 61)
"""
```

9. Consider this code:

```
def square(num):
""" (number) -> number
Return the square of num.
>>> square(3)
9
"""
```

In the following table, fill in the Example column by writing square, num, square(3), and 3 next to the appropriate description.

Description	**Example**
Parameter	
Argument	
Function name	
Function call	

10. Write the body of the square function from the previous exercise.

Bài tập chương 6

1. Import module math, and use its functions to complete the following exercises. (You can call dir(math) to get a listing of the items in math.)
a. Write an expression that produces the floor of -2.8.
b. Write an expression that rounds the value of -4.3 and then produces the absolute value of that result.
c. Write an expression that produces the ceiling of the sine of 34.5.

```
#câu a
math.floor(-2.8)
#câu b
abs(round(-4.3))
#câu c
```

```
math.ceil(math.sin(34.5))
```

2. In the following exercises, you will work with Python's calendar module:
   a. Visit the Python documentation website at http://docs.python.org/release/3.6.0/py-modindex.html, and look at the documentation on module calendar.
   b. Import module calendar.
   c. Using function help, read the description of function isleap.
   d. Use isleap to determine the next leap year.
   e. Use dir to get a list of what calendar contains.
   f. Find and use a function in module calendar to determine how many leap years there will be between the years 2000 and 2050, inclusive.
   g. Find and use a function in module calendar to determine which day of the week July 29, 2016, will be.

```
#câu a
import calendar
#câu b
help(calendar.isleap)
#câu c
calendar.isleap(2016)
#câu d
dir(calendar)
#câu e
calendar.leapdays(2000, 2050)
#câu f
calendar.weekday(2016, 7, 29)
```

3. Create a file named exercise.py with this code inside it:

```
def average(num1: float, num2: float) -> float:
"""Return the average of num1 and num2.
>>> average(10,20)
15.0
>>> average(2.5, 3.0)
2.75
"""
return num1 + num2 / 2
```

   a. Run exercise.py. Import doctest and run doctest.testmod().
   b. Both of the tests in function average's docstring fail. Fix the code and rerun the tests. Repeat this procedure until the tests pass.

Bài tập chương 7

1. In the Python shell, execute the following method calls:
   a. 'hello'.upper()
   b. 'Happy Birthday!'.lower()
   c. 'WeeeEEEEeeeEEEEeee'.swapcase()

d. 'ABC123'.isupper()
e. 'aeiouAEIOU'.count('a')
f. 'hello'.endswith('o')
g. 'hello'.startswith('H')
h. 'Hello {0}'.format('Python')
i. 'Hello {0}! Hello {1}!'.format('Python', 'World')

2. Using string method count, write an expression that produces the number of o's in 'tomato'.
3. Using string method find, write an expression that produces the index of the first occurrence of o in 'tomato'.
4. Using string method find, write a *single* expression that produces the index of the *second* occurrence of o in 'tomato'. Hint: Call find twice.
5. Using your expression from the previous exercise, find the second o in 'avocado'. If you don't get the result you expect, revise the expression and try again.
6. Using string method replace, write an expression that produces a string based on 'runner' with the n's replaced by b's.
7. Variable s refers to ' yes '. When a string method is called with s as its argument, the string 'yes' is produced. Which string method was called?
8. Variable fruit refers to 'pineapple'. For the following function calls, in what order are the subexpressions evaluated?
   a. fruit.find('p', fruit.count('p'))
   b. fruit.count(fruit.upper().swapcase())
   c. fruit.replace(fruit.swapcase(), fruit.lower())
9. Variable season refers to 'summer'. Using string method format and variable season, write an expression that produces 'I love summer!'
10. Variables side1, side2, and side3 refer to 3, 4, and 5, respectively. Using string
    method format and those three variables, write an expression that produces 'The sides have lengths 3, 4, and 5.'
11. Using string methods, write expressions that produce the following:
    a. A copy of 'boolean' capitalized
    b. The first occurrence of '2' in 'CO2 H2O'
    c. The second occurrence of '2' in 'CO2 H2O'
    d. True if and only if 'Boolean' begins lowercase
    e. A copy of "MoNDaY" converted to lowercase and then capitalized
    f. A copy of " Monday" with the leading whitespace removed
12. Complete the examples in the docstring and then write the body of the following function:

```
def total_occurrences(s1: str, s2: str, ch: str) -> int:
"""Precondition: len(ch) == 1
Return the total number of times that ch occurs in s1 and s2.
>>> total_occurrences('color', 'yellow', 'l')
3
>>> total_occurrences('red', 'blue', 'l')
>>> total_occurrences('green', 'purple', 'b')
"""
```

```
def total_occurrences(s1, s2, ch):
    """ (str, str, str) -> int
    Precondition: len(ch) == 1
    Return the total number of times that ch occurs in s1 and s2.
    >>> total_occurrences('color', 'yellow', 'l')
    3
    >>> total_occurrences('red', 'blue', 'l')
    1
    >>> total_occurrences('green', 'purple', 'b')
    0
    """
    return s1.count(ch) + s2.count(ch)
```

LAB 2: Lưu trữ các bộ dữ liệu với List

Nội dung:

Bài tập chương 8

1. Variable kingdoms refers to the list ['Bacteria', 'Protozoa', 'Chromista', 'Plantae', 'Fungi',
   'Animalia']. Using kingdoms and either slicing or indexing with positive indices,
   write expressions that produce the following:
   a. The first item of kingdoms
   b. The last item of kingdoms
   c. The list ['Bacteria', 'Protozoa', 'Chromista']
   d. The list ['Chromista', 'Plantae', 'Fungi']
   e. The list ['Fungi', 'Animalia']
   f. The empty list

```
#câu a
kingdoms[0]
#câu b
kingdoms[5]
#câu c
kingdoms[:3]
#câu d
kingdoms[2:5]
#câu e
kingdoms[4:]
#câu f
kingdoms[1:0]
```

2. Repeat the previous exercise using negative indices.

```
#câu a
kingdoms[-6]
#câu b
kingdoms[-1]
#câu c
kingdoms[-6:-3]
#câu d
kingdoms[-4:-1]
```

```
#câu e
kingdoms[-2:]
#câu f
kingdoms[-1:-2]
```

3. Variable appointments refers to the list ['9:00', '10:30', '14:00', '15:00', '15:30']. An appointment is scheduled for 16:30, so '16:30' needs to be added to the list.
   a. Using list method append, add '16:30' to the end of the list that appointments refers to.
   b. Instead of using append, use the + operator to add '16:30' to the end of the list that appointments refers to.
   c. You used two approaches to add '16:30' to the list. Which approach modified the list and which approach created a new list?

```
#câu a
appointments.append('16:30')
#câu b
appointments += ['16:30']
#câu c
The approach in (a) modifies the list. The one in (b) creates a new
list.
```

4. Variable ids refers to the list [4353, 2314, 2956, 3382, 9362, 3900]. Using list methods, do the following:
   a. Remove 3382 from the list.
   b. Get the index of 9362.
   c. Insert 4499 in the list after 9362.
   d. Extend the list by adding [5566, 1830] to it.
   e. Reverse the list.
   f. Sort the list.
5. In this exercise, you'll create a list and then answer questions about that list.
   a. Assign a list that contains the atomic numbers of the six alkaline earth metals—beryllium (4), magnesium (12), calcium (20), strontium (38), barium (56), and radium (88)—to a variable called alkaline_earth_metals.
   b. Which index contains radium's atomic number? Write the answer in two ways, one using a positive index and one using a negative index.
   c. Which function tells you how many items there are in alkaline_earth_metals?
   d. Write code that returns the highest atomic number in alkaline_earth_metals.
6. In this exercise, you'll create a list and then answer questions about that list.
   a. Create a list of temperatures in degrees Celsius with the values 25.2,

16.8, 31.4, 23.9, 28, 22.5, and 19.6, and assign it to a variable called temps.
b. Using one of the list methods, sort temps in ascending order.
c. Using slicing, create two new lists, cool_temps and warm_temps, which contain the temperatures below and above 20 degrees Celsius, respectively.
d. Using list arithmetic, recombine cool_temps and warm_temps into a new list called temps_in_celsius.

7. Complete the examples in the docstring and then write the body of the following function:
def same_first_last(L: list) -> bool:
"""Precondition: len(L) >= 2
Return True if and only if first item of the list is the same as the last.
>>> same_first_last([3, 4, 2, 8, 3])
True
>>> same_first_last(['apple', 'banana', 'pear'])
>>> same_first_last([4.0, 4.5])
"""

```
def same_first_last(L):
    """ (list) -> bool
    Precondition: len(L) >= 2
    Return True if and only if first item of the list is the same as
    the last.
    >>> same_first_last([3, 4, 2, 8, 3])
    True
    >>> same_first_last(['apple', 'banana', 'pear'])
    False
    >>> same_first_last([4.0, 4.5])
    False
    """
    return L[0] == L[-1]
```

8. Complete the examples in the docstring and then write the body of the following function:
def is_longer(L1: list, L2: list) -> bool:
"""Return True if and only if the length of L1 is longer than the length of L2.
>>> is_longer([1, 2, 3], [4, 5])
True
>>> is_longer(['abcdef'], ['ab', 'cd', 'ef'])
>>> is_longer(['a', 'b', 'c'], [1, 2, 3]
"""

```
def is_longer(L1, L2):
    """ (list, list) -> bool
    Return True if and only if the length of L1 is longer than the
    length of L2.
    >>> is_longer([1, 2, 3], [4, 5])
    True
    >>> is_longer(['abcdef'], ['ab', 'cd', 'ef'])
    False
    >>> is_longer(['a', 'b', 'c'], [1, 2, 3])
    False
    """
    return len(L1) > len(L2)
```

9. Draw a memory model showing the effect of the following statements:
   values = [0, 1, 2]
   values[1] = values
10. Variable units refers to the nested list [['km', 'miles', 'league'], ['kg', 'pound', 'stone']].
    Using units and either slicing or indexing with positive indices, write expressions that produce the following:
    a. The first item of units (the first inner list)
    b. The last item of units (the last inner list)
    c. The string 'km'
    d. The string 'kg'
    e. The list ['miles', 'league']
    f. The list ['kg', 'pound']
11. Repeat the previous exercise using negative indices.

Bài tập chương 9

1. Write a for loop to print all the values in the celegans_phenotypes list, one per line. celegans_phenotypes refers to ['Emb','Him', 'Unc', 'Lon', 'Dpy', 'Sma'].

```
for phenotype in celegans_phenotypes:
    print(phenotype)
```

2. Write a for loop to print all the values in the half_lives list, all on a single line. half_lives refers to [87.74, 24110.0,
   6537.0, 14.4, 376000.0].

```
for value in half_lives:
    print(value, end=' ')
```

3. Write a for loop to add 1 to all the values from whales, and store the converted values in a new list called more_whales. The whales list shouldn’t be modified. whales refers to [5, 4, 7, 3, 2, 3, 2, 6, 4, 2, 1, 7, 1, 3].

```
more_whales = []
for count in whales:
    more_whales.append(count + 1)
```

4. In this exercise, you’ll create a nested list and then write code that performs operations on that list.
   a. Create a nested list where each element of the outer list contains the atomic number and atomic weight for an alkaline earth metal. The values are beryllium (4 and 9.012), magnesium (12 and 24.305), calcium (20 and 40.078), strontium (38 and 87.62), barium (56 and 137.327), and radium (88 and 226). Assign the list to variable alkaline_earth_metals.
   b. Write a for loop to print all the values in alkaline_earth_metals, with the atomic number and atomic weight for each alkaline earth metal on a different line.
   c. Write a for loop to create a new list called number_and_weight that contains the elements of alkaline_earth_metals in the same order but not nested.
5. The following function doesn’t have a docstring, type annotations, or comments. Write enough of all three to make it easy for another programmer to understand what the function does and how, and then compare your solution with those of at least two other people. How similar are they? Why do they differ?
   ```
   def mystery_function(values):
   result = []
   for sublist in values:
   result.append([sublist[0]])
   for i in sublist[1:]:
   result[-1].insert(0, i)
   return result
   ```
6. Repetition Based on User Input, you saw a loop that prompted users until they typed quit. This code won’t work if users type Quit, or QUIT, or any other version that isn’t exactly quit. Modify that loop so that it terminates if a user types that word with any capitalization.
7. Consider the following statement, which creates a list of populations of countries in eastern Asia (China, DPR Korea, Hong Kong, Mongolia, Republic of Korea, and Taiwan) in millions: country_populations = [1295, 23, 7, 3, 47, 21]. Write a for loop that adds up all the values and stores them in variable total. (Hint: Give total an initial value of zero, and, inside the loop body, add the population of the current country to total.)
8. You are given two lists, rat_1 and rat_2, that contain the daily weights of two rats over a period of ten days. Assume the rats never have exactly the same weight. Write statements to do the following:

a. If the weight of rat 1 is greater than that of rat 2 on day 1, print "Rat 1 weighed more than rat 2 on day 1."; otherwise, print "Rat 1 weighed less than rat 2 on day 1.".

b. If rat 1 weighed more than rat 2 on day 1 and if rat 1 weighs more than rat 2 on the last day, print "Rat 1 remained heavier than Rat 2."; otherwise, print "Rat 2 became heavier than Rat 1."

c. If your solution to the previous exercise used nested if statements, then do it without nesting, or vice versa.

9. Print the numbers in the range 33 to 49 (inclusive).
10. Print the numbers from 1 to 10 (inclusive) in descending order, all on one line.
11. Using a loop, sum the numbers in the range 2 to 22 (inclusive), and then calculate the average.
12. Consider this code:

```
from typing import List
def remove_neg(num_list: List[float]) -> None:
"""Remove the negative numbers from the list num_list.
>>> numbers = [-5, 1, -3, 2]
>>> remove_neg(numbers)
>>> numbers
[1, 2]
"""
for item in num_list:
if item < 0:
num_list.remove(item)
```

When remove_neg([1, 2, 3, -3, 6, -1, -3, 1]) is executed, it produces [1, 2, 3, 6, -3, 1].

The for loop traverses the elements of the list, and when a negative value (like -3 at position 3) is reached, it is removed, shifting the subsequent values one position earlier in the list (so 6 moves into position 3). The loop then continues on to process the next item, skipping over the value that moved into the removed item's position. If there are two negative numbers in a row (like -1 and -3), then the second one won't be removed. Rewrite the code to avoid this problem.

13. Using nested for loops, print a right triangle of the character *T* on the screen where the triangle is one character wide at its narrowest point and seven characters wide at its widest point:

```
T
TT
```

```
TTT
TTTT
TTTTT
TTTTTT
TTTTTTT
```

14. Using nested for loops, print the triangle described in the previous exercise with its hypotenuse on the left side:

    ```
    T
    TT
    TTT
    TTTT
    TTTTT
    TTTTTT
    TTTTTTT
    ```
15. Redo the previous two exercises using while loops instead of for loops.
16. Variables rat_1_weight and rat_2_weight contain the weights of two rats at the beginning of an experiment. Variables rat_1_rate and rat_2_rate are the rate that the rats' weights are expected to increase each week (for example, 4 percent per week).
    a. Using a while loop, calculate how many weeks it would take for the weight of the first rat to become 25 percent heavier than it was originally.
    b. Assume that the two rats have the same initial weight, but rat 1 is expected to gain weight at a faster rate than rat 2. Using a while loop, calculate how many weeks it would take for rat 1 to be 10 percent heavier than rat 2.

LAB 3: Đọc và ghi tập tin

Nội dung:

3.1 Các kiểu tập tin
3.2 Các phép toán với tập tin
3.3 Các kiểu đọc tập tin
3.4 Đọc tập tin từ mạng
3.5 Ghi dữ liệu ra tập tin; thư viện StringIO
3.6 Các phép toán tập tin đa dòng

Bài tập chương 10

1. Write a program that makes a backup of a file. Your program should prompt the user for the name of the file to copy and then write a new file with the same contents but with .bak as the file extension.

```
filename = input('Which file would you like to back-up? ')
new_filename = filename + '.bak'
backup = open(new_filename, 'w')
for line in open(filename):
    backup.write(line)
    backup.close()
```

2. Suppose the file alkaline_metals.txt contains the name, atomic number, and atomic weight of the alkaline earth metals:
beryllium 4 9.012
magnesium 12 24.305
calcium 20 20.078
strontium 38 87.62
barium 56 137.327
radium 88 226
Write a for loop to read the contents of alkaline_metals.txt and store it in a list
of lists, with each inner list containing the name, atomic number, and atomic weight for an element. (Hint: Use string.split.)

```
alkaline_metals = []
for line in open('alkaline_metals.txt'):
    alkaline_metals.append(line.strip().split(' '))
```

3. All of the file-reading functions we have seen in this chapter read forward through the file from the first character or line to the last. How could you write a function that would read backward through a file?
4. In Processing Whitespace-Delimited Data, we used the “For Line in File” technique to process data line by line, breaking it into pieces using string

method split. Rewrite function process_file to skip the header as normal but then use the Read technique to read all the data at once.

```
def process_file(reader):
    """ (file open for reading) -> NoneType
    Read and print the data from reader, which must start with a single
    description line, then a sequence of lines beginning with '#',
    then a sequence of data.
    """
    # Find and print the first piece of data.
    line = skip_header(reader).strip()
    print(line)
    # Read the rest of the data.
    print(reader.read())
```

5. Modify the file reader in read_smallest_skip.py of *Skipping the Header*, on page 188 so that it can handle files with no data after the header.

```
import time_series
def smallest_value_skip(reader):
    """ (file open for reading) -> number or NoneType
    Read and process reader, which must start with a time_series
header.
    Return the smallest value after the header. Skip missing values,
which
    are indicated with a hyphen."""
    line = time_series.skip_header(reader).strip()
    # Only execute this code, if there is data following the header.
    if line != '':
        smallest = int(line)
    for line in reader:
        line = line.strip()
        if line != '-':
            value = int(line)
            smallest = min(smallest, value)
    return smallest
if __name__ == '__main__':
    with open('hebron.txt', 'r') as input_file:
        print(smallest_value_skip(input_file))
```

6. Modify the file reader in read_smallest_skip.py of *Skipping the Header*, on page 188, so that it uses a continue inside the loop instead of an if. Which form do you find easier to read?
7. Modify the PDB file reader of *Multiline Records*, on page 195, so that it ignores blank lines and comment lines in PDB files. A blank line is one that contains only space and tab characters (that is, one that looks empty when viewed). A comment is any line beginning with the keyword CMNT.

8. Modify the PDB file reader to check that the serial numbers on atoms start at 1 and increase by 1. What should the modified function do if it finds a file that doesn't obey this rule?

LAB 4: Phép toán Set, Tuples, Dict

Nội dung:

4.1 Lưu trữ dữ liệu với Set và các phép toán
4.2 Lưu trữ dữ liệu với Tuples và các phép toán
4.3 Lưu trữ dữ liệu với Dict và các phép toán
4.4 Một số thí dụ lập trình nâng cao

Bài tập chương 11

1. Write a function called find_dups that takes a list of integers as its input argument and returns a set of those integers occurring two or more times in the list

```
def find_dups(L):
    """ (list) -> set
    Return the number of duplicates numbers from L.
    >>> find_dups([1, 1, 2, 3, 4, 2])
    {1, 2}
    >>> find_dups([1, 2, 3, 4])
    set()
    """
    elem_set = set()
    dups_set = set()
    for entry in L:
        len_initial = len(elem_set)
        elem_set.add(entry)
        len_after = len(elem_set)
        if len_initial == len_after:
            dups_set.add(entry)
    return(dups_set)
```

2. Write the bodies of the new versions of functions read_molecule and read_all_molecules from *Creating New Type Annotations*, on page 224.

```
def mating_pairs(males, females):
    """ (set, set) -> set of tuple
    Return a set of tuples where each tuple contains a male from males
and a
    female from females.
    >>> mating_pairs({'Anne', 'Beatrice', 'Cari'}, {'Ali', 'Bob',
'Chen'})
    {('Cari', 'Chen'), ('Beatrice', 'Bob'), ('Anne', 'Ali')}
    """
    pairs = set()
    num_gerbils = len(males)
    for i in range(num_gerbils):
        male = males.pop()
        female = females.pop()
        pairs.add((male, female),)
    return pairs
```

3. Python’s set objects have a method called pop that removes and returns an arbitrary element from the set. If the set gerbils contains five cuddly little animals, for example, calling gerbils.pop() five times will return those animals one by one, leaving the set empty at the end. Use this to write a function called mating_pairs that takes two equal-sized sets called males and females as input and returns a set of pairs; each pair must be a tuple containing one male and one female. (The elements of males and females may be strings containing gerbil names or gerbil ID numbers—your function must work with both.)
4. The PDB file format is often used to store information about molecules. A PDB file may contain zero or more lines that begin with the word AUTHOR (which may be in uppercase, lowercase, or mixed case), followed by spaces or tabs, followed by the name of the person who created the file. Write a function that takes a list of filenames as an input argument and returns the set of all author names found in those files.
5. The keys in a dictionary are guaranteed to be unique, but the values are not. Write a function called count_values that takes a single dictionary as an argument and returns the number of distinct values it contains. Given the input {'red': 1, 'green': 1, 'blue': 2}, for example, it should return 2.
6. After doing a series of experiments, you have compiled a dictionary showing the probability of detecting certain kinds of subatomic particles. The particles’ names are the dictionary’s keys, and the probabilities are the values: {'neutron': 0.55, 'proton': 0.21, 'meson': 0.03, 'muon': 0.07, 'neutrino': 0.14}. Write a function that takes a single dictionary of this kind as input and returns the particle that is least likely to be observed. Given the dictionary shown earlier, for example, the function would return 'meson'.
7. Write a function called count_duplicates that takes a dictionary as an argument and returns the number of values that appear two or more times.
8. A *balanced color* is one whose red, green, and blue values add up to 1.0. Write a function called is_balanced that takes a dictionary whose keys are 'R', 'G', and 'B' and whose values are between 0 and 1 as input and that returns True if they represent a balanced color.

9. Write a function called dict_intersect that takes two dictionaries as arguments and returns a dictionary that contains only the key/value pairs found in both of the original dictionaries.
10. Programmers sometimes use a dictionary of dictionaries as a simple database. For example, to keep track of information about famous scientists, you might have a dictionary where the keys are strings and the values are dictionaries, like this:
```
{'jgoodall' : {'surname' : 'Goodall',
'forename' : 'Jane',
'born' : 1934,
'died' : None,
'notes' : 'primate researcher',
'author' : ['In the Shadow of Man',
'The Chimpanzees of Gombe']},
'rfranklin' : {'surname' : 'Franklin',
'forename' : 'Rosalind',
'born' : 1920,
'died' : 1957,
'notes' : 'contributed to discovery of DNA'},
'rcarson' : {'surname' : 'Carson',
'forename' : 'Rachel',
'born' : 1907,
'died' : 1964,
'notes' : 'raised awareness of effects of DDT',
'author' : ['Silent Spring']}
}
```
    Write a function called db_headings that returns the set of keys used in *any* of the inner dictionaries. In this example, the function should return set('author', 'forename', 'surname', 'notes', 'born', 'died').
11. Write another function called db_consistent that takes a dictionary of dictionaries in the format described in the previous question and returns True if and only if every one of the inner dictionaries has exactly the same keys. (This function would return False for the previous example, since Rosalind Franklin's entry doesn't contain the 'author' key.)
12. A *sparse vector* is a vector whose entries are almost all zero, like [1, 0, 0, 0, 0, 0, 3, 0, 0, 0]. Storing all those zeros in a list wastes memory, so programmers often use dictionaries instead to keep track of just the nonzero entries. For example, the vector shown earlier would be represented as {0:1, 6:3}, because the vector it is meant to represent has the value 1 at

index 0 and the value 3 at index 6.
a. The sum of two vectors is just the element-wise sum of their elements. For example, the sum of [1, 2, 3] and [4, 5, 6] is [5, 7, 9]. Write a function called sparse_add that takes two sparse vectors stored as dictionaries and returns a new dictionary representing their sum.
b. The dot product of two vectors is the sum of the products of corresponding elements. For example, the dot product of [1, 2, 3] and [4, 5, 6] is 4+10+18, or 32. Write another function called sparse_dot that calculates the dot product of two sparse vectors.
c. Your boss has asked you to write a function called sparse_len that will return the length of a sparse vector (just as Python's len returns the length of a list). What do you need to ask her before you can start writing it?

Bài tập chương 12 (sinh viên làm thêm)

1. A DNA sequence is a string made up of the letters *A*, *T*, *G*, and *C*. To find the complement of a DNA sequence, *A*s are replaced by *T*s, *T*s by *A*s, *G*s by *C*s, and *C*s by *G*s. For example, the complement of AATTGCCGT is TTAACGGCA.
   a. Write an outline in English of the algorithm you would use to find the complement.
   b. Review your algorithm. Will any characters be changed to their complement and then changed back to their original value? If so, rewrite your outline. Hint: Convert one character at a time, rather than all of the *A*s, *T*s, *G*s, or *C*s at once.
   Using the algorithm that you have developed, write a function named complement that takes a DNA sequence (a str) and returns the complement of it.
2. In this exercise, you'll develop a function that finds the minimum or maximum value in a list, depending on the caller's request.
   a. Write a loop (including initialization) to find both the minimum value in a list and that value's index in one pass through the list.
   b. Write a function named min_index that takes one parameter (a list) and returns a tuple containing the minimum value in the list and that value's index in the list.
   c. You might also want to find the maximum value and its index. Write a function named min_or_max_index that has two parameters: a list and a bool. If the Boolean parameter refers to True, the function returns a

tuple containing the minimum and its index; if it refers to False, it returns a tuple containing the maximum and its index.

3. In *The Readline Technique,* on page 181, you learned how to read some files from the Time Series Data Library. In particular, you learned about the Hopedale data set, which describes the number of colored fox fur pelts produced from 1834 to 1842. This file contains one value per year per line.
   a. Write an outline in English of the algorithm you would use to read the values from this data set to compute the average number of pelts produced per year.
   b. Translate your algorithm into Python by writing a function named hopedale_average that takes a filename as a parameter and returns the average number of pelts produced per year.
4. Write a set of doctests for the find-two-smallest functions. Think about what kinds of data are interesting, long lists or short lists, and what order the items are in. Here is one list to test with: [1, 2]. What other interesting ones are there?
5. What happens if the functions to find the two smallest values in a list are passed a list of length one? What should happen, and why? How about length zero? Modify one of the docstrings to describe what happens.
6. One or more of the three functions to find the two smallest values don't work if there are duplicate values, and particularly if the two smallest values are the same. Write doctests to demonstrate the problem, run them, and fix the algorithms that exhibit this bug.
7. This one is a fun challenge.
   Edsgar Dijkstra is known for his work on programming languages. He came up with a neat problem that he called the Dutch National Flag problem: given a list of strings, each of which is either 'red', 'green', or 'blue' (each is repeated several times in the list), rearrange the list so that the strings are in the order of the Dutch national flag—all the 'red' strings first, then all the 'green' strings, then all the 'blue' strings.
   Write a function called dutch_flag that takes a list and solves this problem.

Bài tập chương 13 (sinh viên làm thêm)

1. All three versions of linear search start at index 0. Rewrite all to search from the end of the list instead of from the beginning. Make sure you test them.
2. For the new versions of linear search: if there are duplicate values, which do they find?
3. Binary search is significantly faster than the built-in search but requires that the list is sorted. As you know, the running time for the best sorting algorithm is on the order of N log2 N, where N is the length of the list. If

we search a lot of times on the same list of data, it makes sense to sort it once before doing the searching. Roughly how many times do we need to search in order to make sorting and then searching faster than using the built-in search?

4. Given the unsorted list [6, 5, 4, 3, 7, 1, 2], show what the contents of the list would be after each iteration of the loop as it is sorted using the following:
   a. Selection sort
   b. Insertion sort
5. Another sorting algorithm is *bubble sort*. Bubble sort involves keeping a sorted section at the end of the list. The list is traversed, pairs of elements are compared, and larger elements are swapped into the higher position. This is repeated until all elements are sorted.
   a. Using the English description of bubble sort, write an outline of the bubble sort algorithm in English.
   b. Continue using top-down design until you have a Python algorithm.
   c. Turn it into a function called bubble_sort(L).
   d. Try it out on the test cases from selection_sort.
6. In the description of bubble sort in the previous exercise, the sorted section of the list was at the end of the list. In this exercise, bubble sort will maintain the sorted section at the beginning of the list. Make sure that you are still implementing bubble sort!
   a. Rewrite the English description of bubble sort from the previous exercise with the necessary changes so that the sorted elements are at the beginning of the list instead of at the end.
   b. Using your English description of bubble sort, write an outline of the bubble sort algorithm in English.
   c. Write function bubble_sort_2(L).
   d. Try it out on the test cases from selection_sort
7. Modify the timing program to compare bubble sort with insertion and selection sort. Explain the results.
8. The analysis of bin_sort said, "Since *N* values have to be inserted, the overall running time is *N* log2 *N*." Point out a flaw in this reasoning, and explain whether it affects the overall conclusion.
9. There are at least two ways to come up with loop conditions. One of them is to answer the question, "When is the work done?" and then negate it. In function merge in *Merging Two Sorted Lists*, on page 267, the answer is, "When we run out of items in one of the two lists," which is described by this expression: i1 == len(L1) or i2 == len(L2). Negating this leads to our condition i1 != len(L1) and i2 != len(L2).

Another way to come up with a loop condition is to ask, "What are the valid values of the loop index?" In function merge, the answer to this is 0 <= i1 < len(L1) and 0 <= i2 < len(L2); since i1 and i2 start at zero, we can drop the comparisons with zero, giving us i1 < len(L1) and i2 < len(L2). Is there another way to do it? Have you tried both approaches? Which do you prefer?

10. In function mergesort in *Merge Sort*, on page 267, there are two calls to extend.
    They are there because when the preceding loop ends, one of the two lists still has items in it that haven't been processed. Rewrite that loop so that these extend calls aren't needed.

LAB 5: Lập trình hướng đối tượng với Python

Nội dung:

5.1 Function isinstance, Class object và Class Book
5.2 Thiết kế Methods trong Class Book và nguyên lý OO

Bài tập chương 14

1. In this exercise, you will implement class Country, which represents a country with a name, a population, and an area.
   a. Here is a sample interaction from the Python shell:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> canada.name
'Canada'
>>> canada.population
34482779
>>> canada.area
9984670
```

   This code cannot be executed yet because class Country does not exist. Define Country with a constructor (method __init__) that has four parameters:
   a country, its name, its population, and its area.
   b. Consider this code:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> usa = Country('United States of America', 313914040, 9826675)
>>> canada.is_larger(usa)
```

True
In class Country, define a method named is_larger that takes two Country objects and returns True if and only if the first has a larger area than the second.
c. Consider this code:

```
>>> canada.population_density()
3.4535722262227995
```

In class Country, define a method named population_density that returns the population density of the country (people per square kilometer).
d. Consider this code:

```
>>> usa = Country('United States of America', 313914040, 9826675)
>>> print(usa)
United States of America has a population of 313914040 and is 9826675 square km.
```

In class Country, define a method named __str__ that returns a string representation of the country in the format shown here.
After you have written __str__, this session shows that a __repr__ method would be useful:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> canada
<exercise_country.Country object at 0x7f2aba30b550>
>>> print(canada)
Canada has population 34482779 and is 9984670 square km.
>>> [canada]
[<exercise_country.Country object at 0x7f2aba30b550>]
>>> print([canada])
[<exercise_country.Country object at 0x7f2aba30b550>]
```

Define the __repr__ method in Country to produce a string that behaves like this:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> canada
Country('Canada', 34482779, 9984670)
>>> [canada]
[Country('Canada', 34482779, 9984670)]
```

```python
#câu a
class Country:
    def __init__(self, name, population, area):
        """ (Country, str, int, int)
        A new Country named name with population people and area area.
        >>> canada = Country('Canada', 34482779, 9984670)
        >>> canada.name
        'Canada'
        >>> canada.population
        34482779
        >>> canada.area
        9984670
        """
        self.name = name
        self.population = population
        self.area = area
#câu b
def is_larger(self, other):
    """ (Country, Country) -> bool
    Return whether this country is larger than other.
    >>> canada = Country('Canada', 34482779, 9984670)
    >>> usa = Country('United States of America', 313914040, 9826675)
    >>> canada.is_larger(usa)
    True
    >>> usa.is_larger(canada)
    False
    """
    return self.area > other.area
#câu c
def population_density(self):
    """ (Country) -> float
    Return the population density of this country.
    >>> canada = Country('Canada', 34482779, 9984670)
    >>> canada.population_density()
    3.4535722262227995
    """
    return self.population / self.area
#câu d
def __str__(self):
    """ (Country) -> str
    Return a printable representation of this country.
    >>> usa = Country('United States of America', 313914040, 9826675)
    >>> print(usa)
    United States of America has a population of 313914040 and is
    9826675 square km.
    """
    return '{} has a population of {} and is {} square
km.'.format(self.name, self.population, self.area)
#câu e
def __repr__(self):
    """ (Country) -> str
    Return a concise representation of this country.
    >>> canada = Country('Canada', 34482779, 9984670)
    >>> canada
    Country('Canada', 34482779, 9984670)
    >>> [canada]
    "34482779,
    9984670)":http://pragprog.com/wikis/wiki/Country('Canada',
    """
    return "Country('{0}', {1}, {2})".format(self.name,
self.population, self.area)
```

2. In this exercise, you will implement a Continent class, which represents a continent with a name and a list of countries. Class Continent will use class Country from the previous exercise. If Country is defined in another module, you'll need to import it.

   a. Here is a sample interaction from the Python shell:

```
>>> canada = country.Country('Canada', 34482779, 9984670)
>>> usa = country.Country('United States of America', 313914040,
... 9826675)
>>> mexico = country.Country('Mexico', 112336538, 1943950)
>>> countries = [canada, usa, mexico]
>>> north_america = Continent('North America', countries)
>>> north_america.name
'North America'
>>> for country in north_america.countries:
print(country)
Canada has a population of 34482779 and is 9984670 square km.
United States of America has a population of 313914040 and is 9826675
square km.
Mexico has a population of 112336538 and is 1943950 square km.
>>>
```

   The code cannot be executed yet, because class Continent does not exist. Define Continent with a constructor (method __init__) that has three parameters: a continent, its name, and its list of Country objects.

   b. Consider this code:

```
>>> north_america.total_population()
460733357
```

   In class Continent, define a method named total_population that returns the sum of the populations of the countries on this continent.

   c. Consider this code:

```
>>> print(north_america)
North America
Canada has a population of 34482779 and is 9984670 square km.
United States of America has a population of 313914040 and is 9826675
square km.
Mexico has a population of 112336538 and is 1943950 square km.
```

   In class Continent, define a method named __str__ that returns a string representation of the continent in the format shown here.

3. In this exercise, you'll write __str__ and __repr__ methods for several classes.
   a. In class Student, write a __str__ method that includes all the Member information and in addition includes the student number, the list of courses taken, and the list of current courses.
   b. Write __repr__ methods in classes Member, Student, and Faculty. Create a few Student and Faculty objects and call str and repr on them to verify that your code does what you want it to.
4. Write a class called Nematode to keep track of information about *C. elegans*, including a variable for the body length (in millimeters; they are about 1 mm in length), gender (either hermaphrodite or male), and age (in days). Include methods __init__, __repr__, and __str__.
5. Consider this code:
   ```
   >>> segment = LineSegment(Point(1, 1), Point(3, 2))
   >>> segment.slope()
   0.5
   >>> segment.length()
   2.23606797749979
   ```
   In this exercise, you will write two classes, Point and LineSegment, so that you can run this code and get the same results.
   a. Write a Point class with an __init__ method that takes two numbers as parameters.
   b. In the same file, write a LineSegment class whose constructor takes two Points as parameters. The first Point should be the start of the segment.
   c. Write a slope method in the class LineSegment that computes the slope of the segment. (Hint: The slope of a line is rise over run.)
   d. Write a length method in class LineSegment that computes the length of the segment. (Hint: Use x ** n to raise x to the nth power. To compute the square root, raise a number to the (1/2) power or use math.sqrt.)

<u>LAB 6:</u> Lập trình giao diện đồ họa GUI với Python

Nội dung:

6.1 Giới thiệu về Tkinter, Qt, WxWidgets.
6.2 Quản lý Layout.
6.3 Widget.
6.4 Menu.
6.5 Hội thoại.
6.6 Đồ họa.

Bài tập chương 16

1. Write a GUI application with a button labeled “Goodbye.” When the button is clicked, the window closes.

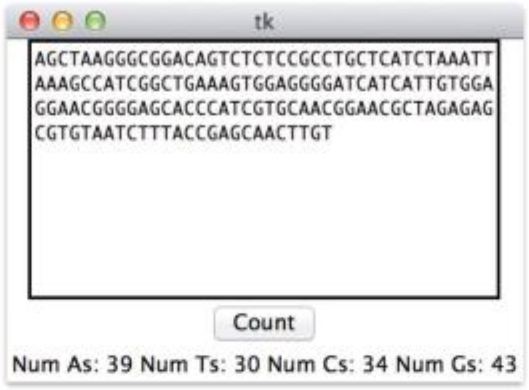
```
import tkinter
window = tkinter.Tk()
frame = tkinter.Frame(window)
frame.pack()
button = tkinter.Button(frame, text='Goodbye', command=lambda:
window.destroy())
button.pack()
window.mainloop()
```

2. Write a GUI application with a single button. Initially the button is labeled 0, but each time it is clicked, the value on the button increases by 1.

```
import tkinter
def increment(text):
    """ Increment number represented by the contents of text by 1 and
update
    text with the result."""
    count = int(text.get())
    text.set(str(count + 1))
window = tkinter.Tk()
frame = tkinter.Frame(window)
frame.pack()
text = tkinter.StringVar()
text.set('0')
button = tkinter.Button(frame, textvariable=text,
command=lambda: increment(text))
button.pack()
window.mainloop()
```

3. What is a more readable way to write the following?
   x = **lambda**: y
4. A DNA sequence is a string made up of *A*s, *T*s, *C*s, and *G*s. Write a GUI application in which a DNA sequence is entered, and when the Count

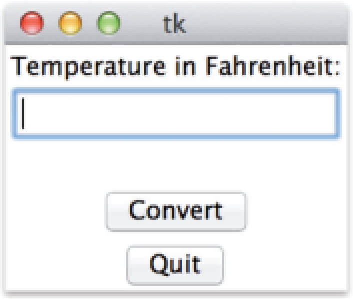
button is clicked, the number of *A*s, *T*s, *C*s, and *G*s are counted and displayed in the window (see the following image).



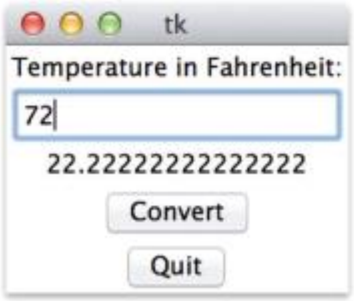
```
AGCTAAGGGCGGACAGTCTCTCCGCCTGCTCATCTAAATT
AAAGCCATCGGCTGAAAGTGGAGGGGATCATCATTGTGGA
GGAACGGGGAGCACCCATCGTGCAACGGAACGCTAGAGAG
CGTGTAATCTTTACCGAGCAACTTGT
```

```
Num As: 39 Num Ts: 30 Num Cs: 34 Num Gs: 43
```

5. In *Defining Our Own Functions*, on page 35, we wrote a function to convert degrees Fahrenheit to degrees Celsius. Write a GUI application that looks like the following image.



When a value is entered in the text field and the Convert button is clicked, the value should be converted from Fahrenheit to Celsius and displayed in the window, as shown in the following image.



```
72
22.2222222222222
```

6. Rewrite the text editor code from *Using Menu*, on page 337, as an objectoriented GUI

Chương 17 (sinh viên làm thêm)

1. In this exercise, you will create a table to store the population and land area of the Canadian provinces and territories according to the 2001 census. Our data is taken from http://www12.statcan.ca/english/census01/home/index.cfm.

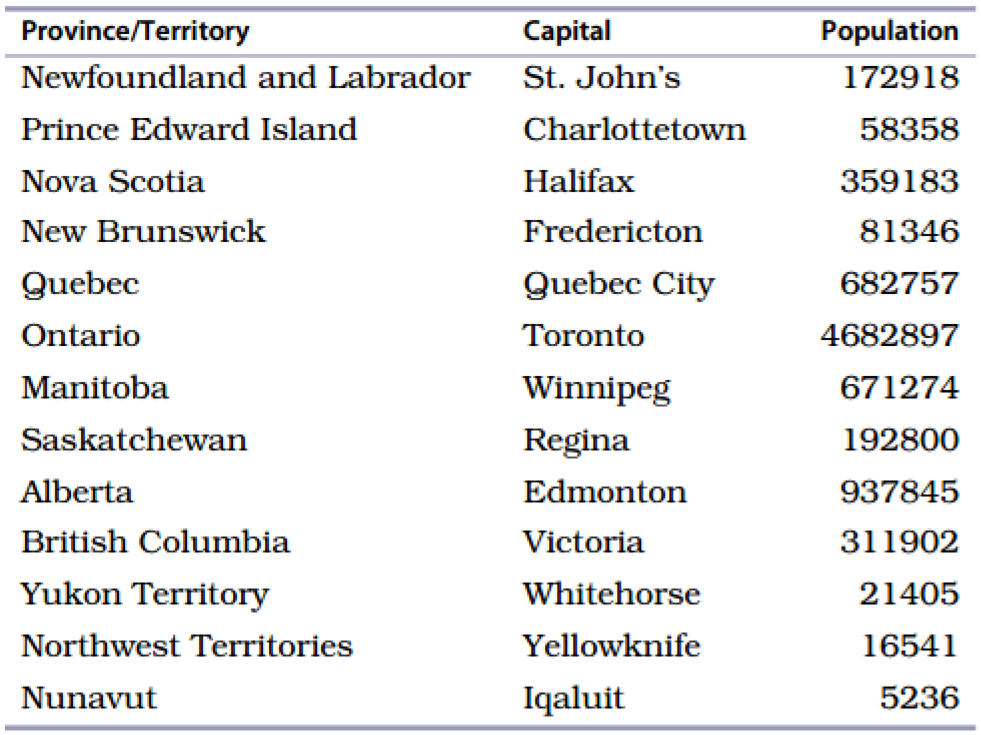
Province/Territory	Population	Land Area
Newfoundland and Labrador	512930	370501.69
Prince Edward Island	135294	5684.39
Nova Scotia	908007	52917.43
New Brunswick	729498	71355.67
Quebec	7237479	1357743.08
Ontario	11410046	907655.59
Manitoba	1119583	551937.87
Saskatchewan	978933	586561.35
Alberta	2974807	639987.12
British Columbia	3907738	926492.48
Yukon Territory	28674	474706.97
Northwest Territories	37360	1141108.37
Nunavut	26745	1925460.18

Table 34—2001 Canadian Census Data

Write Python code that does the following:
a. Creates a new database called census.db
b. Makes a database table called Density that will hold the name of the province or territory (TEXT), the population (INTEGER), and the land area (REAL)
c. Inserts the data from Table 34, *2001 Canadian Census Data*, on page 365
d. Retrieves the contents of the table
e. Retrieves the populations
f. Retrieves the provinces that have populations of less than one million
g. Retrieves the provinces that have populations of less than one million or greater than five million
h. Retrieves the provinces that do not have populations of less than one million or greater than five million
i. Retrieves the populations of provinces that have a land area greater than 200,000 square kilometers
j. Retrieves the provinces along with their population densities (population divided by land area)

2. For this exercise, add a new table called Capitals to the database. Capitals has three columns—province/territory (TEXT), capital (TEXT), and

population
(INTEGER)—and it holds the data shown here:



Province/Territory	Capital	Population
Newfoundland and Labrador	St. John's	172918
Prince Edward Island	Charlottetown	58358
Nova Scotia	Halifax	359183
New Brunswick	Fredericton	81346
Quebec	Quebec City	682757
Ontario	Toronto	4682897
Manitoba	Winnipeg	671274
Saskatchewan	Regina	192800
Alberta	Edmonton	937845
British Columbia	Victoria	311902
Yukon Territory	Whitehorse	21405
Northwest Territories	Yellowknife	16541
Nunavut	Iqaluit	5236

Table 35—2001 Canadian Census Data: Capital City Populations

Write SQL queries that do the following:
a. Retrieve the contents of the table
b. Retrieve the populations of the provinces and capitals (in a list of tuples of the form [province population, capital population])
c. Retrieve the land area of the provinces whose capitals have populations greater than 100,000
d. Retrieve the provinces with land densities less than two people per square kilometer and capital city populations more than 500,000
e. Retrieve the total land area of Canada
f. Retrieve the average capital city population
g. Retrieve the lowest capital city population
h. Retrieve the highest province/territory population
i. Retrieve the provinces that have land densities within 0.5 persons per square kilometer of on another—have each pair of provinces reported only once

3. Write a Python program that creates a new database and executes the following SQL statements. How do the results of the SELECT statements differ from what you would expect Python itself to do? Why?

```
CREATE TABLE Numbers(Val INTEGER)
INSERT INTO Numbers Values(1)
INSERT INTO Numbers Values(2)
SELECT * FROM Numbers WHERE 1/0
SELECT * FROM Numbers WHERE 1/0 AND Val > 0
SELECT * FROM Numbers WHERE Val > 0 AND 1/0
```