COMP S265F Design and Analysis of Algorithms
Lab 1: Python Programming Fundamentals

In this course, we will write Python programs with Anaconda/Miniconda and Visual Studio Code (VS Code). This lab covers the fundamentals of Python programming.

Task 1: Getting started with Python

In this task, we will set up our Python programming environment and write the "Hello World" program.

1. We need a Python interpreter and an editor in our working environment.

   a. Install the Python 3.8 interpreter at Anaconda or Miniconda official website.

   b. Install VS Code from its the official website.

   - For Windows user, during installation, check (1) Create a desktop icon, (2) Add "Open with Code" action to Windows Explorer file context menu, (3) Add to PATH (requires shell restart).
   - For Mac or Linux user,
     i. After installation, launch VS Code.
     ii. Open the Command Palette (Ctrl+Shift+P) and type `shell command` to find and run the command `Shell Command: Install 'code' command in PATH`.

2. Open the Anaconda prompt (Windows) or Terminal (Mac/Linux).

   a. - For Windows user, type `Anaconda Prompt` in the search bar.
   - For Mac or Linux user,
     i. Launch Terminal.
     ii. You need to activate Anaconda for the first time: `source ~/opt/anaconda3/bin/activate`. If you installed Miniconda in Step 1, change `anaconda3` to `miniconda3`.

   b. Change your current working directory using the command `cd working_directory_path`.

3. Open a new file `hello.py` in VS Code using `code hello.py`. (Step 1b enables the `code` command.)

4. Write the following statement in `hello.py` and save the file.

```python
print("Hello World!")
```

   The function `print()` prints something in the standard output. A newline character is printed at the end by default. This is similar to `System.out.println()` in Java.

5. Run your Python program using the command `python hello.py` and observe the output.

Your task: Write a Python program to print your name and student ID on two lines. The following is the expected output for the student "Lee Lap Kei" with student ID "12345678".

Sample Output

```
Lee Lap Kei
12345678
```

Task 2: Primitive types and standard input

There are four primitive types in Python:

- **Integer:** Python integer does not have an upper bound and a lower bound value. Note that unary increment (`x++`) and decrement (`x--`) are **not** supported.

```
x = 2
print(x, x + 1, x - 1, x * 3, x ** 3)       # x**3 means x^3
print(7 / 3, 7 // 3, 7 % 3)                 # x//y & x%y are quotient & remainder of x/y
```

- **Float:** Python float values are represented as 64-bit double-precision values. Any number greater than the maximum value (approximately 1.8×10^{308}) will be indicated by the string `inf` in Python.

```
y = 2.5
print(y, y + 1, y * 2, y ** 2)
```

- **Boolean:** English words are used (`True, False, and, or, not`). Note that `&, |, ^, ~` are bitwise operators.

```
handsome = False
rich = True
print(handsome and rich)
print(handsome or rich)
print(not handsome)
```

- **String:** Strings in python are surrounded by either single or double quotes. There is a strong support for strings in Python. You can find many built-in methods for string in the official document.

```
name = "Keith"
print(name.lower())                         # Convert all characters to lowercase
print(name[0], name[-1])                    # Print the first and the last character
print(name[1:4])                            # Print substring from index 1 to index 4-1=3
age = 30
print(f"{name} is more than {age}.")        # Adding f before the string makes a f-string
name += "Lee       "
print(name.strip())                         # Remove the leading and trailing whitespaces
```

You can use the function `type` to check the data type.

```
print(type(3), type(3.0), type("True"), type(True))
```

Unlike Java, you cannot use concatenate a number and a string using the operator `+`. You need to cast the number to a string using the built-in function `str()`.

```
number = 3
string = "Three"
print(number + string)                  # Run-time error
print(str(number) + string)             # Prints "3Three"
```

To read from the standard input, you need the `stdin` file object from the `sys` module. The method `readline()` reads one line of input from the standard input. However, a newline character `\n` will be appended at the end. You may use the `replace()` method from string to remove it.

02hello_name.py

```
from sys import stdin

name = stdin.readline()
name = name.replace("\n", "")
print("Hello {}!".format(name))
```

The method `readline()` always reads in a string. You can cast it to a number using `int()` or `float()`.

`02area_square.py`

```python
from sys import stdin

size = stdin.readline()
size = float(size)
area = size * size

print(area)
```

It is tedious to type your input many times for testing. You can redirect the standard input to a file using `<` in your command. Similarly, `>` will redirect your standard output to a file. For example, you can read input from the file `name.txt` and output to the file `output.txt` using `python 02hello_name.py < name.txt > output.txt`.

Your task: Write a program to read the height and base of a triangle from standard input, then output its area.

Sample Input

```
3.5
4
```

Sample Output

```
7.0
```

Task 3: Control structure

Python provides if-elif-else structure, while loop, and for-each loop as the control structure. They all work in a similar way as their counterparts in other programming languages. Note that Python uses indentation (rather than brackets) to indicate a block of code; you have to use the same number of spaces for indentation. VS Code automatically converts a tab to four spaces.

- **If-elif-else structure:**

```python
if condition:
    do someting
elif condition2:          # elif (not else-if)
    do other thing
else:
    do other other thing
```

- **While loop:**

```python
while condition:
    do something
```

- **For-each loop:**

```python
for item in iterable:
    do something with each item
```

Python does not have the `for(begin; condition; step)` structure, there are two ways to rewrite it.

- **Rewrite as a while loop:** Split the begin, condition checking and step into 3 standalone statements.

```python
i = 0                     # begin
while i < n:              # condition checking
    print(i)
    i += 1                # step
```

- **Using the range() function:** The built-in `range(stop)` or `range(start, stop[, step])` function returns an *iterable* of numbers. It starts with `start` (or 0) and ends up to (but not including) `stop`.

```python
for i in range(n):
    print(i)              # print from 0 to n-1
```

```python
for i in range(x, n):
    print(i)              # print from x to n-1
```

```python
for i in range(x, n, s):
    print(i)              # print from x, x+s, x+s+s...
```

Discussion: Can you point out a limitation of using `range()` but not while-loop as shown above?

Defining a function by `def`: In Python, a function is defined using the `def` keyword. You can add as many arguments as you want by just separating them with commas. In the `main` function of the following program, `stdin` is used with for-each loop to read all the lines from the standard input. It demonstrates how to read a list of integers and print the number of positive integers.

03count_positive.py

```python
from sys import stdin

def main():
    positive = 0
    for num in stdin:
        num = int(num)
        if num > 0:
            positive += 1
    print(f"There are {positive} positive numbers.")

main()
```

When typing the input by keyboard, you need to enter EOF (end-of-file) to stop the above program. EOF can be typed using `Ctrl-Z` (Windows) and `Ctrl-D` (Mac and Linux).

Your task: Write a program to read a list of integers, and print the average of only the *positive* integers.

Sample Input

```
3
4
-1
1
0
2
```

Sample Output

```
The mean is 2.5.
```

Task 4: Data container

You will use mainly the following four types of data containers.

- **List:** A list consists of comma-separated values (items) in a pair of square brackets. The items can be of different types. You can find the built-in methods for list in the official document. *List comprehension* offers a shorter syntax when you want to create a new list based on the values of an existing list:
  [*expression* `for` *item* `in` *iterable*]
  [*expression* `for` *item* `in` *iterable* `if` *condition*]

```python
nums = [x for x in range(10)]    # generate a list of integer from 0 upto 10 (exclusive)
even = [x for x in nums if x % 2 == 0]
print(even)

# append an item (number 10) to the end of list
even.append(10)
print(even)

# check if an item (num) is in the list; if yes, prints its index
for num in [3, 4, 5, 6]:
    if num in even:
        print(even.index(num), end=" ")  # end with a space instead of \n
    else:
        print("No", end=" ")
print()

# remove an item (number 4) from the list
even.remove(4)
print(even)

# remove the item at index 2
del even[2]
print(even)

# insert the number 3 at index 2
even.insert(2, 3)
print(even)

# print the first three items of even
print(even[:3])

# extend a list using another list
prime = even[1:3]
prime.extend([5, 7])
print(prime)

# print the list in reverse order
print(prime[::-1])    # list slicing operator [start:stop:step]
```

- **Tuple:** A tuple consists of a number of values separated by commas in a pair of brackets. Different to list, items in a tuple are immutable (i.e., unchangeable) and thus can be accessed more efficiently.

```python
t = (12345, 54321, 'hello!')
print(t)          # Prints (12345, 54321, 'hello!')
print(t[1])       # Prints 54321
for item in t:
    print(item)
a, b, c = t
print(a, b, c)    # Prints 12345 54321 hello!
```

- **Set:** A set is an unordered collection without duplicates. A set object also supports mathematical operations like union, intersection, difference, and symmetric difference. You can find the built-in methods for set in the official document.

```
my_set = {1,3,5,1}
print(my_set)         # Prints {1, 3, 5}
print(2 in my_set)    # Prints False
print(3 in my_set)    # Prints True
my_set.add(2)
print(my_set)         # Prints {1, 2, 3, 5}
my_set.remove(1)
print(my_set)         # Prints {2, 3, 5}
for item in my_set:
    print(item)
```

- **Dictionary:** Dictionaries are sometimes found in other languages as "associative memories" or "associative arrays". Lists are indexed by a number, while dictionaries are indexed by keys, which can be any immutable type (e.g., strings and numbers). It is best to think of a dictionary as a set of `key:value` pairs, with all keys being unique. You can find the built-in methods for dictionary in the official document.

```
phonebook = {'Keith': '27686024', 'Vanessa': '27686814'}

# Andrew is not in the phone book
if 'Andrew' in phonebook:
    print("Andrew's number is", phonebook['Andrew'])
else:
    print("Andrew's number is not in the phone book.")

# We add Andrew's contact to the phone book
print("Let's add Andrew's phone")
phonebook['Andrew'] = '27686846'

# Now we should be able to get Andrew's phone
if 'Andrew' in phonebook:
    print("Andrew's number is", phonebook['Andrew'])
else:
    print("Andrew's number is not in the phone book.")

# change Keith's number
print("Keith's original number is", phonebook['Keith'])
phonebook['Keith'] = '21800000'
print("Keith's new number is", phonebook['Keith'])

del phonebook['Keith']
if 'Keith' in phonebook:
    print("Keith's number is", phonebook['Keith'])
else:
    print("Keith's number is not in the phone book.")

names = phonebook.keys()
print(names)
for name in names:
    print(name, phonebook[name])
```

Your task: The program 04gen_dictionary.py reads a text file with only English words (e.g., hello.py). Rewrite it such that it prints the number of occurrences of each word in alphabetic order. Case is ignored.

Sample Input

```
Adventures in Disneyland
Two blondes were going to Disneyland when they came to a fork in the road
The sign read Disneyland Left
So they went home
```

Sample Output

```
a 1
adventures 1
blondes 1
came 1
disneyland 3
fork 1
going 1
home 1
in 2
left 1
read 1
road 1
sign 1
so 1
the 2
they 2
to 2
two 1
went 1
were 1
when 1
```

Task 5: Third-party modules

There are many third-party modules available on the Internet. To use these modules, you need to install them in your environment.

The following program uses the third party module – NumPy. If your environment does not have this module, you will get the `ModuleNotFoundError`. Anaconda/Miniconda provides the module manager `conda`. It helps you to find and install many modules. You can run the command `conda search numpy` to search the availability of different versions of the NumPy module. Running `conda install numpy` installs the latest version of the NumPy module. If you want to uninstall it, you can run `conda uninstall numpy`.

05try_numpy.py

```python
import numpy as np

x = np.array([1.0, 2.0, 3.0])
print(x)
```

Your task: The program 05sin.py uses the module `matplotlib` to plot a sine wave. Rewrite it to plot a cosine wave.

–End of Lab 1–