**convert_size.py**

Write a convenience tool that can handle the translation of memory and disk defined into an integer number of bytes with the following units:

byte
kilobyte
megabyte
gigabyte
terabyte
petabyte
exabyte
zettabyte
yottabyte
kibibyte
mebibyte
gibibyte
tebibyte
pebibyte
exbibyte
zebibyte
yobibyte

So, for example:
Given 400B, the program should return 400.
Given 40KB, the program should return 40000.
Given 50MiB, the program should return 52428800.
Capitalization should not matter, so the following should be True: 50MiB == 50mib == 50MIB.

Your program should take in a string value from the command line and return the number of bytes as standard output. For example:

```
convert_size.py 50MiB

52428800
```

**convert_tasks.py**

Given multiple files filled with tasks, write a program to convert all disk and memory requirements to bytes.

Example task files are attached. These files are in json format. Please see part_four_input_00.json, part_four_input_01.json, and part_four_input_02.json.

Example input contents:

```
{
'TASK01': {
'cpu':  '4',
'mem':  '5GB',
'disk': '4Gib',
'time': amount_of_time_to_complete_task,
'relies_on': 'TASK03'}
}
```

And after running the program, the output should look like:

```
{
'TASK01': {
'cpu':  '4',
'mem':  '50000000000',
'disk': '4294967296',
'time': amount_of_time_to_complete_task,
'relies_on': 'TASK03'}
}
```

Your program should be able to take in multiple file paths as an argument, for example:

convert_tasks.py --files part_four_input_00.json part_four_input_01.json part_four_input_02.json

Your program should create an output file for each input file, for example: part_four_output_00.json, part_four_output_01.json, and part_four_output_02.json

**find_shortest_runtime.py**

You've been given a large number of interdependent software operations to run, but you only have one 16 core computer to run them on.

Your job is to order the packing of these tasks onto your computer so that they run in the shortest amount of time possible.

For this task, try to explain your thinking process (include a readme file and/or ensure the code is well commented and well explained).

Example task files are attached. These files are in json format. Please see part_five_input_00.json, part_five_input_01.json, and part_five_input_02.json.

Each "task" will be the same as in part 4, with 5 features:
'cpu': '#_of_CPUs_needed_by_task'
'time': 'amount_of_time_to_complete_task'
'relies_on': 'TASK##'
Notes:
- Tasks can be run simultaneously so long as the total task resource usage does not exceed the total resources of your computer.
Example:
For a [2cpu] computer.
And two tasks with the requirements:
TASK1:
cpu: 1
time: 30 minutes
relies_on: None
TASK2:
cpu: 1
time: 20 minutes
relies_on: None
It should take 30 minutes total to finish running both tasks.
- Tasks with the "relies_on" tag cannot be run before those tasks have been completed. For example, in the two tasks above, if either relied on the other, they would run in serial and take 50 minutes total.

Your program should be able to take in a single input file of tasks, for example:

```
find_shortest_runtime.py --file part_five_input_00.json
```

And return a single number to standard output representing the shortest total time that the tasks can be completed in:

```
30
```

*Note:* This problem is more difficult. Focus on prioritizing a working program (does not throw an error when run), and generating a practical solution (it does not have to be the best solution). For example, a trivial minimal working example should be simply adding all run times together and not thinking about optimizing at all. Once that program is running without error, from there, one can attempt to improve the run time.