# Week 09 Weekly Test Sample Answers

## Test Conditions

These questions must be completed under self-administered exam-like conditions. You must time the test yourself and ensure you comply with the conditions below.

- You may complete this test in CSE labs or elsewhere using your own machine.
- You may complete this test at any time before **Week 10 Thursday 18:00:00**.
- Weekly tests are designed to act like a past paper - to give you an idea of how well you are progressing in the course, and what you need to work on. Many of the questions in weekly tests are from past final exams.
- Once the first hour has finished, you must submit all questions you've worked on.
- You should then take note of how far you got, which parts you didn't understand.
- You may choose then to keep working and submit test question anytime up to Week 10 Thursday 18:00:00
- However the maximum mark for any question you submit after the first hour will be 50%

You may access this **language documentation** while attempting this test:

- manual entries, via the *man* command.
- Texinfo pages, via the *info* command.
- Bash documentation, via the `help` build-in.
- Python documentation, via the `python3 -c 'help()'` command.
- Shell/Regex quick reference
- Python quick reference
- full Python 3.9 documentation

**Any violation of the test conditions will results in a mark of zero for the entire weekly test component.**

## Getting Started

Set up for the test by creating a new directory called `test09` and changing to this directory.

```
$ mkdir test09
$ cd test09
```

There are some provided files for this test which you can fetch with this command:

```
$ 2041 fetch test09
```

If you're not working at CSE, you can download the provided files as a zip file or a tar file.

WEEKLY TEST QUESTION:

## Statistical Analysis of Command Line Arguments

Write a Python program **describe_numbers.py** that numbers as command line arguments prints:

- The count of how many numbers
- The count of how many unique numbers
- The minimum number
- The maximum number
- The mean of the numbers
- The median of the numbers
- The mode of the numbers
- The sum of the numbers
- The product of the numbers

  For example:

```
$ ./describe_numbers.py 1 333 42
count=3
unique=3
minimum=1
maximum=333
mean=125.33333333333333
median=42
mode=1
sum=376
product=13986
$ ./describe_numbers.py 3 4 2 -1 7 -6 5
count=7
unique=7
minimum=-6
maximum=7
mean=2
median=3
mode=3
sum=14
product=5040
$ ./describe_numbers.py 15 -15 8 -11 8
```

NOTE:

Your program can assume is given at least one argument.

Your program can assume is given an odd number of arguments.

You should leave all numbers unrounded.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

No error checking is necessary.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 2041 autotest describe_numbers
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 test09_describe_numbers describe_numbers.py
```

SOLUTION:

Sample solution for `describe_numbers.py`

```python
#!/usr/bin/env python3


from sys import argv
from textwrap import dedent
from functools import reduce
from builtins import sum as builtin_sum
from operator import mul
from statistics import mean as stat_mean, median as stat_median, mode as stat_mode


def main():
    argv.pop(0)

    args = list(map(int, argv))

    count     = len(args)
    unique    = len(set(args))
    sum       = builtin_sum(args)
    product   = reduce(mul, args, 1)
    minimum   = min(args)
    maximum   = max(args)
    mean      = stat_mean(args)
    median    = stat_median(args)
    mode      = stat_mode(args)

    print(dedent(f"""
        {count=}
        {unique=}
        {minimum=}
        {maximum=}
        {mean=}
        {median=}
        {mode=}
        {sum=}
        {product=}
    """).strip())


if __name__ == '__main__':
    main()
```

---

WEEKLY TEST QUESTION:

## N Distinct lines

Write a Python program **distinct_lines.py** which given a single argument **N** as a command-line argument, reads lines from standard input until **N** different lines have been read.

It should then print a message (exactly as below) indicating how many lines were read. It should then stop, and not read any futher input.

If end-of-input is reached before **n** different lines it should print a message indicating how many lines were read.

Your program should ignore case and white-space when comparing lines.

```
$ ./distinct_lines.py 3
hi
hello world
hi
hello world
hello world
bye
3 distinct lines seen after 6 lines read.
```

```
$ ./distinct_lines.py 3
hi
hello world
   hi
hello        world
   HELLO  world
bye
3 distinct lines seen after 6 lines read.
```

```
$ ./distinct_lines.py 4
how
are
you
are
how
are
well
4 distinct lines seen after 7 lines read.
```

```
$ ./distinct_lines.py 3
how
are
you
3 distinct lines seen after 3 lines read.
```

```
$ ./distinct_lines.py 7
hello
how
are
you
Ctrl-D
End of input reached after 4 lines read - 7 different lines not seen.
```

NOTE:

You can assume your program is given a single positive integer as argument.

You can assume STDIN contains only ASCII bytes.

You can assume linesare terminated with a '\n' byte, and STDIN contains no un-terminated lines.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

No error checking is necessary.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 2041 autotest distinct_lines
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 test09_distinct_lines distinct_lines.py
```

SOLUTION:

Sample solution for `distinct_lines.py`

```python
#!/usr/bin/env python3

from sys import argv, stdin
from re import sub
from collections import Counter

def main():
    assert len(argv) == 2, f"Usage: {argv[0]} <N>"

    N = int(argv[1])
    lines = Counter()

    for index, line in enumerate(stdin, 1):
        line = sub(r"\s", "", line) # remove all whitespace
        line = line.lower() # convert to lowercase
        lines[line] += 1

        if len(lines) == N:
            print(f"{N} distinct lines seen after {index} lines read.")
            break
    else:
        print(f"End of input reached after {index} lines read - {N} different lines not seen.")

if __name__ == '__main__':
    main()
```

WEEKLY TEST QUESTION:

## Simple Grep in Python

Write a Python program, **python_grep.py** implements the basic functionallity of the grep command.

Your program will be given two command line arguments: a regular expression and a file.

Your program should print out all the lines in the file that match the regular expression.

```
$ seq 1 1000 > seq.txt
$ ./python_grep.py '[^2468]5[2468]' seq.txt
152
154
156
158
352
354
356
358
552
554
556
558
752
754
756
758
952
954
956
958
$ ./python_grep.py '^(..).+\1.+\1$' dictionary.txt
alkaloidal
antihumanitarian
astasias
endenizen
essentialnesses
esthesises
estimablenesses
```

NOTE:

You can assume that you are always given two command line arguments.

You can assume that the first argument is a valid regular expression.

You can assume that the second argument is a path to a file that exists and is readable.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

No error checking is necessary.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 2041 autotest python_grep
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 test09_python_grep python_grep.py
```

SOLUTION:

Sample solution for `python_grep.py`

```
#!/usr/bin/env python3

from sys import argv
from re import search

def main():
    assert len(argv) == 3, f"Usage: {argv[0]} <regex> <file>"

    regex = argv[1]
    file = argv[2]

    with open(file, 'r') as f:
        for line in f:
            line = line.rstrip('\n')

            if search(regex, line):
                print(line)


if __name__ == '__main__':
    main()
```

# Submission

When you are finished each exercise make sure you submit your work by running **give**.

You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Week 10 Thursday 18:00:00** to complete this test.

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

Hint: do your own testing as well as running `autotest`

## Test Marks

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

The test exercises for each week are worth in total 1 marks.

The best 6 of your 8 test marks for weeks 3-10 will be summed to give you a mark out of 9.

---

**COMP(2041|9044) 22T2: Software Construction** is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au
CRICOS Provider 00098G