Week 10 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 11 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 11 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 `test10` and changing to this directory.

```
$ mkdir test10
$ cd test10
```

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

```
$ 2041 fetch test10
```

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

WEEKLY TEST QUESTION:
Replace Those References

Write a Python program **reference.py** that reads lines of text from its standard input and prints them to its standard output. Except for lines which contain with a '#' character followed by a positive integer.

Lines of the form **#n** (where **n** is an integer value), should be replaced this by the **n**'th line of input.

This transformation only applies to lines which start with a # character, followed by the digits of a positive integer and then the newline character. No other characters appear on such lines.

All **n** values will be valid input line numbers,

No **n** values will refer to other #**n** lines.

For example:

```
$ cat reference_input.txt
line A
line B
line C
#7
line D
#2
line E
$ ./reference.py < reference_input.txt
line A
line B
line C
line E
line D
line B
line E
```

NOTE:

You can assume lines are no more than 256 characters long.

You can assume there are no more than 1024 lines in the input,

You can assume lines are no more than 256 characters long.

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 reference
```

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

```
$ give cs2041 test10_reference reference.py
```

SOLUTION:

Sample solution for `reference.py`

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

import sys
import re

lines = sys.stdin.readlines()

for line in lines:
    if line[-1] == '\n':
        line = line[:-1]
    if m := re.fullmatch(r'#(\d+)', line):
        line_number = int(m.group(1))
        print(lines[line_number - 1], end='')
    else:
        print(line)
```

WEEKLY TEST QUESTION:

Well-Rounded Text

Write a Python program **text_round.py**that copies its standard input to standard output but maps all numbers to their nearest whole number equivalent. For example, **0.667** would be mapped to **1**, **99.5** would be mapped to **100**, **16.35** would be mapped to **16**, and so on. All other text in the input should be transferred to the output unchanged.

A *number* is defined as a string containing some digit characters with an optional decimal point ('**.**') followed by zero or more additional digit characters.

For example **0**, **100**, **3.14159**, **1000.0**, **0.999** and **12345.** are all valid numbers.

For example, given this input:

```
I spent $15.50 for 3.3kg of apples yesterday.
Pi is approximately 3.141592653589793
2000 is a leap year, 2001 is not.
```

your program should produce this output:

```
I spent $16 for 3kg of apples yesterday.
Pi is approximately 3
2000 is a leap year, 2001 is not.
```

For example:

```
$ cat text_round_input.txt
I spent $15.50 for 3.3kg of apples yesterday.
Pi is approximately 3.141592653589793
2000 is a leap year, 2001 is not.
$ text_round.py < text_round_input.txt
I spent $16 for 3kg of apples yesterday.
Pi is approximately 3
2000 is a leap year, 2001 is not.
```

NOTE:

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 text_round
```

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

```
$ give cs2041 test10_text_round text_round.py
```

SOLUTION:

Sample solution for `text_round.py`

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

import sys
import re

for line in sys.stdin:
    numbers = re.findall(r'(\d+\.\d+)', line)
    for number in numbers:
        line = line.replace(number, str(int(float(number) + 0.5)))
    print(line, end='')
```

WEEKLY TEST QUESTION:

Caesar Cypher

Write a Python script `caesar_cypher.py` which given an integer *N* prints stdin encrypted using a Caesar cypher with the shift *N*.

That is: for each character in stdin, print the corresponding character in the alphabet shifted by *N* places.

N can be any integer.

Your input will be ASCII text.

Letters should wrap around at the beginning and end of the alphabet.

Uppercase letters and lowercase letters should be shifted separately.

Digits and Symbols should not be shifted.

```
$ ./caesar_cypher.py 13
Hello World
Uryyb Jbeyq
How was COMP2041/9044
Ubj jnf PBZC2041/9044
Ctrl-D
$ ./caesar_cypher.py 13
Uryyb Jbeyq
Hello World
Ubj jnf PBZC2041/9044
How was COMP2041/9044
Ctrl-D
$ ./caesar_cypher.py 4
Now is the winter of our discontent
Rsa mw xli amrxiv sj syv hmwgsrxirx
Ctrl-D
$ ./caesar_cypher.py -4
Rsa mw xli amrxiv sj syv hmwgsrxirx
Now is the winter of our discontent
Ctrl-D
$ ./caesar_cypher.py 443 | ./caesar_cypher.py -443
This is the way the world ends
This is the way the world ends
This is the way the world ends
Not with a bang but a whimper
Ctrl-D
This is the way the world ends
This is the way the world ends
This is the way the world ends
Not with a bang but a whimper
```

NOTE:

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_caesar
```

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

```
$ give cs2041 test10_python_caesar caesar_cypher.py
```

SOLUTION:

Sample solution for `caesar_cypher.py`

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

import sys
from string import ascii_lowercase, ascii_uppercase

assert len(sys.argv) == 2, "Usage: ./caesar_cypher.py <offset>"

offset = int(sys.argv[1])

cypher_lowercase = ''.join(map(lambda c: chr((ord(c) - ord(ascii_lowercase[0]) + offset) %
(ord(ascii_lowercase[-1]) - ord(ascii_lowercase[0]) + 1) + ord(ascii_lowercase[0])),
ascii_lowercase))
cypher_uppercase = ''.join(map(lambda c: chr((ord(c) - ord(ascii_uppercase[0]) + offset) %
(ord(ascii_uppercase[-1]) - ord(ascii_uppercase[0]) + 1) + ord(ascii_uppercase[0])),
ascii_uppercase))

cypher = str.maketrans(ascii_lowercase + ascii_uppercase, cypher_lowercase + cypher_uppercase)

for line in sys.stdin:
    print(line.translate(cypher), end="")
```

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 11 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.