Week 08 Tutorial Sample Answers

1. What types are avalible as inbuilt types in Python?

ANSWER:

- int()
- float()
- str()
- [] - list()

```
[1, 2, 3]
[] # empty list
```

- {} - dict() - store data values in key:value pairs.

```
{"key1": "value1", "key2": "value2"}
{} # empty dict.
```

- (,) - tuple() - immutable list

```
(1, 2, 3, 4, 5)
("hello",) # single elements must end with a comma or python will think it's just a grouping operator
() # NOT an empty tuple
tuple() # is an empty tuple
```

- {,} - set() - unordered collection of unique elements

```
{1, 2, 3, 4, 5}
{"hello",} # single elements must end with a comma or python will complain it looks like a dict
{} # NOT an empty set (is a dict)
set() # is an empty set
```

- frozenset - like a set, but immutable (like a tuple is to a list)

2. What other useful types are available with Python's standard library?

ANSWER:

- collections.Counter

  works like a dictionary, but with some extra methods, plus has a default value of 0.

```
from collections import Counter
c = Counter()
c['a'] += 1   # index error with a normal dictionary
print(c['a'])
```

- collections.defaultdict

  works like a dictionary, but you can set a default value for missing keys.

```
from collections import defaultdict
c = defaultdict(list)
c['a'].append("hello")   # index error with a normal dictionary
print(c['a'])
```

- collections.OrderedDict

  works like a dictionary, but keeps the order of the keys.

  ie. OrderedDict.keys() is the same order as the keys were added.

  This is *currnetly* true for normal dictionaries, **but not garenteed**.

  Always use OrderedDict if you need to preserve the order of the keys.

```
from collections import OrderedDict
c = OrderedDict()
c['z'] = "COMP1511"
c['a'] = "COMP1521"
c['k'] = "COMP1531"
c['b'] = "COMP2041"
print(c.keys())
```

3. Write a Python function `print_dict()` that displays the contents of a dict in the format below:

```
[key] => value
[Andrew] => green
[Anne] => red
[John] => blue
```

ANSWER:

```
def print_dict(d):
    for key, val in d.items():
        print(f"[{key}] => {val}")
```

4. Write a Python program, `times.py` which prints a table of multiplications.

   Your program will be given the dimension of the table and the width of the columns to be printed. For example:

```
$ ./times.py 5 5 5
    1    2    3    4    5
    2    4    6    8   10
    3    6    9   12   15
    4    8   12   16   20
    5   10   15   20   25
```

```
$ ./times.py 10 10 3
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

ANSWER:

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

def main():
    if len(sys.argv) != 4:
        print(f"Usage: {sys.argv[0]} <n> <m> <column-width>")
        sys.exit(1)

    n = int(sys.argv[1])
    m = int(sys.argv[2])
    width = int(sys.argv[3])

    for x in range(1, n + 1):
        print(f"{x: >{width}}", end="")
        for y in range(2, m + 1):
            print(f" {x * y: >{width}}", end="")
        print()


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

5. Write a Python program `missing_words.py` which given two files as arguments prints, in sorted order, all the words found in file1 but not file2.

   You can assume words occur one per line in each file.

ANSWER:

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

"""
print words in file 1 but not file 2
"""

import sys

def main():
    if len(sys.argv) != 3:
        print(f"Usage: {sys.argv[0]} <file1> <file2>")
        sys.exit(1)

    words1 = set()

    with open(sys.argv[1]) as f1:
        for word in f1:
            word = word.strip()
            words1.add(word)

    words2 = set()

    with open(sys.argv[2]) as f2:
        for word in f2:
            word = word.strip()
            words2.add(word)

    for word in words1 - words2:
        print(word)

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

6. Write a Python program `source_count.py` which prints the number of lines of C source code in the current directory. In other words, this Python program should behave similarly to `wc -l *.[ch]`. (Note: you are not allowed to use `wc` or other Unix programs from within the Perl script). For example:

```
$ ./source_count.py
    383 cyclorana.c
    280 cyclorana.h
     15 enum.c
    194 frequency.c
    624 model.c
    293 parse.c
    115 random.c
     51 smooth.c
    132 util.c
     16 util.h
    410 waveform.c
   2513 total
```

ANSWER:

```
#!/usr/bin/env python3
# written by andrewt@cse.unsw.edu.au for COMP2041
# count lines of C source code
from glob import glob

def main():
    total = 0
    for filename in glob("*.[ch]"):
        with open(filename) as f:
            lines = f.readlines()
            n_lines = len(lines)
            print(f"{n_lines:7} {filename}")
            total += n_lines
    print(f"{total:7} total")

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

7. Write a Python program, `phone_numbers.py` which given the URL of a web page fetches it by running *wget* and prints any strings that might be phone numbers in the web page.

   Assume the digits of phone numbers may be separated by zero or more spaces or hyphens ('-') and can contain between 8 and 15 digits inclusive.

   You should print the phone numbers one per line with spaces & hyphens removed.

```
$ ./phone_numbers.py https://www.unsw.edu.au
20151028
11187777
841430912571345
413200225
61293851000
57195873179
```

ANSWER:

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

import sys, re, subprocess

def main():
    for url in sys.argv[1:]:
        process = subprocess.run(f"wget -q -O- {url}", shell=True, capture_output=True,
text=True)
        webpage = process.stdout
        for number in re.findall(r'[\d \-]+', webpage):
            number = re.sub(r'\D', '', number)
            if len(number) >= 8 and len(number) <= 15:
                print(number)

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