Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

1. `global` keyword is required to access a global variable from within a function. **T** **F**

2. In Python, an object that is hashable is also immutable. **T** **F**

3. In Python, the *as* operator creates a new name binding. **T** **F**

4. The `__name__` special variable of a module is always the name of the file, without the .py extension. **T** **F**

5. In Python, if an exception is not handled by the end of a function, the program will crash. **T** **F**

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following statements are true about dynamic scoping?

i. Dynamic scoping can only be done at runtime.

ii. You can see which variables are in scope simply by looking at the structure of the code.

iii. Each time a new function is executed, a new scope is created.

iv. Python uses dynamic scoping.

v. Depending on the caller of the function, a name may be resolved to different binding.

b) Which of the following is true about the import statement in Python? e.g. `import foo`, but not `from foo import bar`.

i. You can have multiple instances of a module by importing it multiple times.

ii. In a Python script, you must write all your import statements first before any other statements.

iii. If you import a module that has `__all__ = []`, then nothing can be imported from it.

iv. The import statement does not import any variable whose name starts with an underscore.

v. Runnable code in a module is also executed when it is being imported.

Question 3. Short Questions

a) Explain why the built-in type `set` does not support the subscript operator.

Since set is an unordered type, it cannot guarantee that an index will map to the same object within the container throughout its lifetime. Therefore, would be incorrect to support the subscript operator.

b) Explain the difference between `None` in Python and `NULL` in C++.

None is a *value* that represents "no value". NULL is an *address* that signifies invalid address.

Incorrectly using None will always result in well-defined behaviour (e.g. program crash) whereas dereferencing a null pointer can result in undefined behaviour (e.g. memory corruption instead of program crash).

c) In Python, what is the difference between using `try` … `except` versus the `with` statement?

Try-except statements are used for error handling. The with statement is usually used to perform automatic clean-up (more accurately, context management, not covered in class).

d) What error is shown when you attempt to run this script?

```
MAXLEN = 4

def process(input):
    size = len(input)
    if size < MAXLEN:
        return input + [0] * (MAXLEN - size)
    else:
        MAXLEN *= 2
        return process(input)

print(process(list(range(1, 6))))
```

`UnboundedLocalError`

e) Make a one line fix to the above script. What is the output of this script after you have fixed it?

Add this line at the beginning of the function `process`.

```
global MAXLEN
```

The output is `[1, 2, 3, 4, 5, 0, 0, 0]`

Question 4. Programming Questions

a) Write a function, reverse_dict(d), that will reverse keys and values such that the original values become the new keys to lists of one or more values that were the original keys. For example:

```
{ "bob" : 2, "greg" : 3,  "joe" : 2, "tom" : 1,
     "dave" : 2,  "stu" : 3, "mike" : 5 }
```

becomes

```
{ 1 : ["tom"], 2 : ["bob", "joe", "dave"],
   3 : ["greg", "stu"], 5 : ["mike"] }
```

The function should return a new dictionary and not modify the existing one.

```
def reverse_dict(d):
    out = dict()
    for k, v in d.items():
        if v in out:
            out[v].append(k)
        else:
            out[v] = [k]
    return out
```

b) Write a function `word_count(filename)`, that will keep count of how many times each word appears in a text file, and return the information in a dictionary, like this:

```
{ "the" : 12, "great" : 2, "a" : 21, "hello" : 1, … }
```

To make the count more accurate, you must first sanitize the file content by removing all non-alphanumeric characters, so “done.” will become “done”, and “Joe’s” will become “Joes”.

If you cannot open the file, return an empty dictionary (instead of crashing).

```
#
# Note that even though this solution works, it is much better to use
# io.StringIO so you won’t need to create a possibly gigantic list.
# Look it up if you are interested, but it’s not covered in class.
#
def word_count(filename):
    try:
        f = open(filename, "rt")
    except OSError:
        return {}

    cleaned = []
    for ch in f.read():
        if ch.isspace():
            cleaned.append(" ")
        elif ch.isalnum():
            cleaned.append(ch)
    f.close()

    out = {}
    for word in ''.join(cleaned).split(" "):
        out[word] = out.get(word, 0) + 1

    return out
```