Contents

Tips, Tricks, and Pitfalls **1**

Level 3 ..... **1**

I: Advanced Functions **1**

II: Exception Handling **2**

Tips, Tricks, and Pitfalls

Level 3

I: Advanced Functions

1. Functions (and other callables) are themselves Python objects. In fact, if you call **dir** on the function or callable name, you can see all its internal functions and attributes. Most of them are beyond the scope of this course, but one thing worth understanding is the concept of 'static variables'. This term is borrowed from other languages (such as C++), but it essentially means to have a variable in a function that retains its state between calls to the function. In Python, you can achieve this by creating and modifying an attribute on the function object. For example:

```
def MyFunction():
    # We need to check if the _numCalls attribute exists yet on the function object.
    # This is necessary in this case, as we can only increment a variable that already exists.
    # The first time the function is called, _numCalls does not yet exist,
    # so we need to define and initialize is to 1.
    # See the below commented-out code for an alternative approach.
    if not hasattr(MyFunction, '_numCalls'):
        # Create a _numCalls attribute on the function object.
        MyFunction._numCalls = 1
    else:
        # Increment the existing _numCalls attribute.
        MyFunction._numCalls += 1

    print('Do work here...')
    print('MyFunction was called ' + str(MyFunction._numCalls) + ' times.')


# The below code is an alternative approach to initializing the
# _numCalls attribute o nthe MyFunction object. If you take this approach,
# there is no need for the if hasattr... logic;
# can just have the MyFunction._numCalls += 1 inside the function.
# However, the above approach is still better,
# as it encapsulates everything within the function itself.
# MyFunction._numCalls = 0

MyFunction()
MyFunction()
MyFunction()

# etc...
```

II: Exception Handling

1. Can catch multiple exception types in one clause by using a tuple. For example:

```
try:
    CallFunction()
except (ValueError,TypeError):
    print('Exception was either a ValueError or a TypeError')
except:
    print('Exception was any error besides a ValueError or TypeError')
```

2. A common beginners' mistake is to put the **try/except** in the same function that an error was raised. This leads to bad code design, in most cases. In general, **try/except** should be done by the *caller* of a function that raises an exception.

3. Can use an **else** clause in an exception handling block. Additionally, there is a **finally** clause for exception blocks. The **else** clause will execute if there were no exceptions (it must appear after all **except** blocks) and the **finally** clause will execute whether or not there was an exception (it must be after all **except** and **else** blocks). See example below:

```
try:
    result = numer/denom
except ZeroDivisionError:
    print('Denom must not be zero!')
except Exception as ex:
    print(ex) # Other, unanticipated exception type.
else:
    # We only want to print result if there was no exception.
    # Attempting to print result after the try block would throw an error
    # if there was an exception, since result does not exist.
    print(result)
finally:
    # This clause is useful for any cleanup,
    # such as closing file handles or releasing external resources.
    print('Put any cleanup code here')

# print(result) # This would throw an error if the above had an exception.
```