takehome-midterm-solution

October 29, 2019

1 COMP 205: Midterm Examination

1.1 Fall 2019

1.1.1 Take Home Version

The issues we encountered during the mid-term on 10/21 are attributable to *one or more* of the following:

1. The network connection in Sci Tech 134 was flaky,
2. The JupyterHub server was overwhelmed by the number of students trying to use it simultaneously,
3. Zoom software that we were using for staying connected taxed everyone's individual machines.
4. Problems with ok.py setup which were corrected later in the hour.

There are reasons to suspect any of the above as possible causes though we don't know precisely which ones to what degree. Meanwhile, it would be difficult (if at all possible) to reconstruct everyone's work from what was turned in and *grade it fairly.* I'd like to follow a multi-part path out of this.

1. Use the midterm as a **take-home exam**. I hope most of you have partial copies of what you submitted. Feel free to fill in and/or correct what you submitted and resubmit as a PDF (*or as a Python Notebook*).
2. Switch away from using the central JupyterHub server, instead encouraging the use of locally-installed Jupyter notebooks on your laptops in accordance with 01-01-getting-started and submitting PDFs into Canvas.
3. Switch to a different method of submitting your solution for the remainder of the semester: PDF files submitted into Canvas rather than through ok.py. **The last two cells of the midterm notebook have been changed accordingly.** *`ok.py` will remain a part of the mix, however, as a way to provide feedback at various stages of progress through the notebook.*
4. We will use Zoom sparingly in the future, using it during lectures as we have been doing in the last few weeks but relying on *Canvas Discussions* to keep a communication channel open during exams.

This exam is administered electronically. Please enter your answers into this workbook in the places provided. This is an open-Internet examination. You may use any books and materials, and the entire internet -- as well as all of the class workbooks and examples -- in answering these questions. You may not use direct communication with other *people* in completing this workbook.

Please do not share this workbook with others until after the exam window is closed. ok.py is set to *allow multiple submissions* and except in special circumstances, **the end time is 11:45;** the last submission before the end time will be considered to be the "official one".

There are 10 questions. Each is worth 10 points. Please note that in general, I will be testing your code against different data of the same form as in the examples.

1. Consider the structure:

[ ]:
```
foo = [ ('M-W', '10:30-11:45'), ( 'comp-205', 'python'), ('T-Th', '18:00-19:
↪15'), ('comp-119', 'big-data') ]
```

Write an expression of `foo` that equals the string`'python'`, only using `foo` and use of `[...]` to locate values.

[ ]:
```
# your answer:
foo[1][1]
```

2. Consider the structure:

[ ]:
```
bar = { 'a': {'protein', 'rna', 'dna'}, 'b': {'wood', 'metal', 'glass'}}
```

What type is `bar['b']`?

Your answer:

`bar['b']` is of type `set`.

3. Write a function `get_tuples` that takes one argument that is a dictionary and converts it to a list of tuples `(key, value)` corresponding to each entry in the dictionary. It should work on the example problem below.

[1]:
```
# fill in details ...
def get_tuples(dictionary):
    tup_list = [(key,value) for key,value in dictionary.items()]
    return tup_list
```

[2]:
```
# this should return [('a', 1), ('b', 2), ('c', 3)]
get_tuples({'a': 1, 'b': 2, 'c': 3})
```

[2]:
```
[('a', 1), ('b', 2), ('c', 3)]
```

4. Write a class `Film` that represents an address book entry and contains attributes 'name', 'actor', 'location'. `Film(name, actor, location)` should create a Film. This should work on the example below.

[3]:
```
# fill in details ...
class Film():
    def __init__(self, actor, name, location):
        self.actor = actor
        self.name = name
        self.location = location
```

```
    def __str__(self):
        return "'{}' starring {} is set in {}.".format(self.name, self.actor,␣
↪self.location)
```

[4]:
```
# This example should print
#                           'A Fish called Wanda' starring John Cleese is set in␣
↪London.
wanda = Film("John Cleese", "A Fish called Wanda", "London")
print(wanda)
```

```
'A Fish called Wanda' starring John Cleese is set in London.
```

5. Write a function `mult_numbers` that prints the product of numbers in a list of numbers and other items. It should convert strings to numbers where appropriate and work on the example below. Hint: use the `try: ... except: ...` pattern.

[12]:
```
# fill in details ...
def mult_numbers(things):
    prod = 1.0
    for thing in things:
        try:
            num = float(thing)
            prod *= num
        except:
            continue # pass would work equally well
    return prod
```

[13]:
```
# this should print 2.0, (being 1   16   0.5   0.25)
prod = mult_numbers([1, 'foo', '16', 'bar', 0.5, '00.2500'])
prod
```

[13]:
```
2.0
```

6. An *index* for a list is a dict that allows each item position in the list to be looked up based upon its value. The index maps strings to list element numbers. Write a function `generate_index` that generates an index from a list of strings. Hint: use `range(len(ind))`.

[14]:
```
# fill in details ...
def generate_index(ind):
    my_dict= {ind[i]:i for i in range(len(ind))}
    return my_dict
```

[15]:
```
# test on this example
generate_index(['Hrothgar', 'Beowulf', 'Grendel'])  # This should return␣
↪{'Hrothgar': 0, 'Beowulf': 1, 'Grendel': 2}
```

[15]:
```
{'Hrothgar': 0, 'Beowulf': 1, 'Grendel': 2}
```

7. A "list of lists of data" is a common data type in data science. Given a list containing lists of equal length, write a function `get_row` that takes this "list of lists" and a number, and generates the tuple of items at position n in each list. The example below should work:

[16]:
```
# fill in details ...
def get_row(lofl, n):
    values = []
    for elem in lofl:
        values.append(elem[n])
    return tuple(values)
```

[17]:
```
# Test on this example
get_row([[1,2,3], ['Hrothgar', 'Beowulf', 'Grendel'], [29,40,1]],1)  # should␣
↪print (2, 'Beowulf', 40)
```

[17]:
```
(2, 'Beowulf', 40)
```

8. write a function `modify_tuple` that changes element n of a tuple t to a new value x and returns the new tuple. Hint: *tuples aren't mutable.* You need to construct a new one!

[18]:
```
# fill in details ...
def modify_tuple(t, i, x):
    retval = []
    j = 0
    for val in t:
        if i == j:
            retval.append(x)
        else:
            retval.append(val)
        j += 1
    return tuple(retval)
```

[19]:
```
modify_tuple((1,2,3), 1, 'foo')  # should return (1, 'foo', 3)
```

[19]:
```
(1, 'foo', 3)
```

9. Consider the class Debt that we considered before:

[20]:
```
class Debt():
    debtor = None
    lender = None
    amount = None
    def __init__(self, p1, p2, m):
        self.debtor = p1
        self.lender = p2
        self.amount = m
    def __str__(self):
        return "{} owes {} ${}".format(self.debtor, self.lender, self.amount)
```

```
    def t(self):
        return (self.debtor, self.lender, self.amount)
```

Write a method of `Debt` called `t` that returns a tuple corresponding to the Debt. It should work as documented below:

[21]:
```
# use this to test your code
d = Debt("Alva", "George", 20)
d.t()  # should output ('Alva', 'George', 20)
```

[21]:
```
('Alva', 'George', 20)
```

10. In a short paragraph, explain when to use a tuple versus a list.

Your answer:

- Lists are mutable whereas tuples are not. This fact drives some other points:
  - The elements of a tuple often have positional relevance -- being at a particular location gives it meaning. For example, the tuple `(mother, father)` might represent `person` and the first element of the tuple will always represent the mother and we might embed that knowledge into the code. Once a tuple has been created, there is generally no need to change it. The elements of a tuple are typically known when the software is designed.
  - We use lists when the number of elements is unknown apriori, as when modeling `children` of a person.

2 When done,

- Save and checkpoint the notebook.
- If your Jupyter installation can download the notebook as a PDF,
  - (File >> Download as >> PDF via LaTeX (.pdf)),
  - Rename this file to `<loginid>-midterm.pdf`. In other words, my filename would be `jsingh11-midterm.pdf`.
  - Submit the file `<loginid>-midterm.pdf` to Canvas.
- Otherwise
  - (File >> Download as >> Notebook (.ipynb)). In other words, my filename would be `jsingh11-midterm.ipynb`.
  - Rename to `<loginid>-midterm.ipynb`,
  - Submit the file `<loginid>-midterm.ipynb` to Canvas.