Math 1MP3, midterm test 2017

17 February 2017

- Please write your name, student number, and MacID on this test and on your answer booklet. Please *circle your family name*, e.g.

Fred Q. (Student), 00045234, studentfq

- You have 50 minutes
- No external aids (calculator, textbook, notes)
- Please number your answers clearly in the test booklet

1. (1 point each) Given that the variable `s` contains an arbitrary string (I will use the example `"Python is awful"` as an illustration, but **do not assume you know the length of the string in advance**), write a Python expression to extract the following substrings. (All character positions referred to below are counted from 1, **not using Python indexing**: e.g. "first character" means `"P"`, "fourth character" means `"h"`.)

a. the first character (`"P"`) **answer**: `s[0]`
b. the last three characters (`"ful"`) **answer**: `s[-3:]` (or `s[len(s)-3:len(s)]`)
c. the fourth through sixth characters inclusive (`"hon"`) **answer**: `s[3:6]`
d. the odd (first, third, fifth ...) characters (`"Pto safl"`) **answer**: `s[::2]` or `s[0:len(s):2]` (or ... ?)

rubric: Only -1 point overall for a single consistent error type (e.g. not 0-indexing, or forgetting that `a:b` doesn't include `b`; that is, you don't lose points twice for making the same kind of error repeatedly.

2. What do the following Python commands print? (1 point each; assume that these commands are *not* run sequentially, or that `L1` is reset to its original value between each command)

```
L1 = (1,[1,2],[3,4])
L2 = {"A":3,"B":2,"C":3}
```

a. `print(len(L1))` **answer**: 3
b. `print(L1[len(L1)-1])` **answer**: `[3,4]`
c. `print(L1[-1])` **answer**: `[3,4]`
d. `print(L1 + (6,7))` **answer**: `(1,[1,2],[3,4],6,7)`
e. `L1 += (8,); print(L1)` **answer**: `(1,[1,2],[3,4],8)`
f. `print(3 in L2)` **answer**: `False`
g. `print("A" in L2)` **answer**: `True`
h. `print(3 in L2.values())` **answer**: `True`

rubric straightforward

3. (4 points) You should wear your raincoat if it is raining, unless the temperature is above 30 degrees (in which case you don't mind getting wet). Precipitation (rain, snow, hail, etc.) falls as rain if (and only if) the temperature is above freezing (0 degrees). Assume that `is_precip` is a boolean variable that says whether it is precipitating and `temp` is a numeric variable giving the temperature in degrees. Write a function `wear_raincoat(is_precip,temp)` that returns a boolean variable that says whether you should wear your raincoat.

```
## possibility 1
def wear_raincoat(is_precip,temp):
    return(is_precip and temp>0 and temp<30)
## possibility 2
def wear_raincoat(is_precip,temp):
    if not is_precip:
       return(False)
    if temp<0 or temp>30:
       return(False)
    return(True)
```

rubric: there are lots of ways to do this one - one big logical expression (e.g. `return(is_precip and temp>0 and temp<30)` or using `if/elif/else` or ...). -1 point for printing rather than returning. -1 point for minor logical errors.

4. (6 points)

a. What is the outcome of the following Python code?

```
L1 = [1,2,3]
def modify(x,pos,val):
   x[pos] = val
```

```
   return(val)
L2 = modify(L1,1,"Q")
print(L1,L2)
```

answer: `[1,'Q',3], Q` Because lists are mutable, the element 1 (the middle element) of `L1` changes when the function is called. The function returns just the value that was to be used, so `L2` is equal to `Q`.

b. How about this variation?

```
L3 = tuple(L1)
L4 = modify(L3,1,"Q")
print(L3,L4)
```

answer: because tuples are not mutable, the attempt to set element 1 of the tuple fails with an error.

rubric: 3 points for each section. -1 for forgetting mutability. -1 for getting confused about what the function returns. Unfortunately, the second part is all-or-nothing (but any answer referring to "error" or "nothing happens" is acceptable; don't have to explicitly say anything about mutability).

5. (5 points) Write a function `odd_even_fun` that takes two strings, which you can assume are of equal lengths, and creates a string composed of the first, third, ... characters of the first string (i.e. the even *positions*: 0, 2, ...) and the second, fourth, ... characters of the second string (the odd *positions*: 1, 3, ...) For example, `odd_even_fun("abc","def")` should return `"aec"`; `odd_even_fun("hello","world")` should return `"hollo"`.

```
def odd_even_fun(s1,s2):
    ret = ""
    for i in range(len(s1)):
        if i % 2 == 0:
           ret += s1[i]
        else:
           ret += s2[i]
    return(ret)

## testing
if not odd_even_fun("abc","def")=="aec":
    raise(BaseException)
if not odd_even_fun("hello","world")=="hollo":
    raise(BaseException)
```

rubric: As usual, there are many ways to do this. -1 for forgetting about zero-indexing. -0.5 for printing rather than returning. 3 points for a reasonable but wrong attempt, 1 point for writing down anything.

6. (3 points for each part) Each of the following code chunks has a problem. Explain what problem/error each will produce (assume that this code is run in a clean Python session, i.e. no variables have been previously defined and no modules have been loaded):

a.

```
x = 3
y = x**2
print(math.cos(y))
```

answer: error (`NameError: name 'math' is not defined`) because math module isn't loaded. Credit for any answer that says anything about an error because the module isn't loaded (`NameError` not necessary, but just "error" without mentioning the module gets only 2 points).

b.

```
x = 0
while i<5:
   i = 0
   x += 1
   i += 1
```

answer: infinite loop (because `i` is set to zero every time through the loop) (**unless** `i` happens to be set to a value `>=5` before this code occurs, but I don't expect anyone to notice that). Any answer along the lines of "doesn't stop" or "keeps going" or "infinite loop" is acceptable.

c.

```
i = 0
res = []
while i<5:
   res += i
   i += 1
```

answer: error (`i` is an integer, and you can't add an integer to a list: would need `res.append(i)` or `res += [i]`)

7. (5 points) Write a function `all_combs(L1,L2)` that takes two lists `L1` and `L2` and returns a list containing all of their combinations. For example:

```
L1 = [1,2,3]
L2 = ["a","b","c"]
def all_combs(L1,L2):
   res = []
   for i in L1:
     for j in L2:
        res.append([i,j])
   return(res)
## this part is just to get nice print format
from pprint import pprint
pprint(all_combs(L1,L2),compact=True)
```

```
## [[1, 'a'], [1, 'b'], [1, 'c'], [2, 'a'], [2, 'b'], [2, 'c'], [3, 'a'], [3, 'b'],
##  [3, 'c']]
```

rubric: no points off for assuming that the lists are of equal length (because this wasn't absolutely unambiguously stated in both exam rooms). At least 2 points for any solution involving nested `for` loops. -0.5 for each class of logical error (0-indexing, range ends, `+= [i,j]` rather than `+= [[i,j]]` ...)

8. (4 points for each part) Consider a file where each line contains the name of a predator followed by the name of one of its prey species. For example:

```
wolf mouse
lion gazelle
aardvark ant
cheetah gazelle
```

---

a. Write a function `predprey_dict(fn)` that takes a file name `fn` as its argument; opens the file for reading; and constructs a dictionary with the first word in each line (predator) as the keys and the second word (prey associated with each predator) as the values. (Use a `for` loop to go through the lines in the file one at a time, and `x.strip().split(" ")` to take a string `x`, remove line-break (`\n`) characters, and split it into words.)

For the file above, the following dictionary would be returned:

```
## file setup:
x = """wolf mouse
lion gazelle
aardvark ant
cheetah gazelle
"""
f = open("test.txt","w")
print(x,file=f,end="")
f.close()

## the actual answer:
def predprey_dict0(fn):
   d = {}
   f = open(fn)
   for L in f:
       w = L.strip().split(" ")
       d[w[0]] = w[1]
```

```
   return(d)

## pretty output
from pprint import pprint
pprint(predprey_dict0("test.txt"))

## {'aardvark': 'ant', 'cheetah': 'gazelle', 'lion': 'gazelle', 'wolf': 'mouse'}
```

b. Now suppose that each predator can have more than one prey:

```
wolf mouse
wolf duck
lion gazelle
aardvark ant
cheetah gazelle
lion wildebeest
```

Each item of the dictionary should now contain a *list* of one or more prey eaten by each predator. (You should only need to change the code inside your `for` loop ...) The answer should now be:

```
## file setup:
x = """wolf mouse
wolf duck
lion gazelle
aardvark ant
cheetah gazelle
lion wildebeest
"""
f = open("test.txt","w")
print(x,file=f,end="")
f.close()

## the answer:
def predprey_dict(fn):
   d = {}
   f = open(fn)
   for L in f:
       w = L.strip().split(" ")
       if w[0] in d:
           d[w[0]].append(w[1])
       else:
           d[w[0]] = [w[1]]
   return(d)

## pretty output:
from pprint import pprint
pprint(predprey_dict("test.txt"))

## {'aardvark': ['ant'],
##  'cheetah': ['gazelle'],
##  'lion': ['gazelle', 'wildebeest'],
##  'wolf': ['mouse', 'duck']}
```

rubric: this might be the hardest one to mark. -1 point for bad dictionary-setting syntax; -1 point in part b for improper appending to the dictionary key.