Math 1MP3: midterm test 1, fall 2019
22 October 2019

You have one hour to complete the test. Please answer the questions on the same page as they are listed. No calculators or other test aids are allowed. There are 9 regular questions worth a total of 120 points and one extra credit question (5 points). Good luck!

1. **(8 points)** String slicing and indexing: what are the results of the following Python commands?

```
S = "Hello, python!"
```

   a. `S[:3]`
   b. `S[-1]`
   c. `S[:len(S)+1]`
   d. `"a" in S`

solutions

```
a. "Hel"
b. "!"
c. "Hello, python!"
d. False
```

rubric

- -1 points for bad indexing (e.g. “He” instead of “Hel” in a.)
- -2 points for saying that c. gives an out-of-range-index

2. **(12 points)** List slicing, indexing, and manipulation: what are the results of the following Python commands?

```
L = [[1,2,3],[4,5,6],[7,8]]
```

a. `len(L)`
b. `L[2][1]`
c. `L+[2]`
d. `L.append([2]); print(L)`
e. `L = [[1,2,3],[4,5,6],[7,8]]; L.extend([2]); print(L)`
f. `L = [[1,2,3],[4,5,6],[7,8]]; L[1].sort(reverse = True); print(L[1])`

```
a. 3
b. 8
c. [[1,2,3],[4,5,6],[7,8],2]
d. [[1,2,3],[4,5,6],[7,8],[2]]
e. [[1,2,3],[4,5,6],[7,8],2]
f. [6,5,4]
```

rubric

- -1 for indexing mistake
- -1 for each confusion of extend and append (e.g. `[2]` for `2` or vice versa

3. (**6 points**) What is the outcome of the following Python code?

```
x = 3
def add_x(x):
    print(x+3)
    return(x+2)
print(add_x(2))
print(x)
```

solution

```
5
4
3
```

rubric

- -2 points for not realizing that `x` is unchanged by use inside the function
- -1 points for misc errors

4. (**20 points**) 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 = 4+5
y += 1
print(x)
print(y)
```

b.

```
def func(x)
    return(x+1)
```

c.

```
x = 4
while x > 0:
    x += 1
```

d.

```
x = (1,2,3)
x.insert(3,2)
```

```
a. undefined symbol (name 'y' is not defined)
b. missing colon/syntax error
c. infinite loop
d. tuples can't be modified
```

rubric

Any reasonably interpretable explanation is OK. "syntax error" is OK for b (since we didn't say how specific to be)

5. **(20 points)** Write a function `ave_no_max_min(DataList)` that returns the average of a given list of numbers (no repeated values in the given list), *excluding* the maximum and minimum values. For example, `ave_no_max_min([2,4,3,5,11])` should return (3+4+5)/3= 4; `ave_no_max_min([11,2,7,9])` should return (7+9)/2 = 8.

solution
Lots of clever possibilities, e.g.

```
def ave_no_max_min(DataList):
   return sum(DataList.sort()[1:-1])/(len(DataList)-2)
```

rubric

- -1 for each minor mistake
- -2 for not enclosing in a function
- -4 for no return, or printing instead of returning

6. **(16 points)** Suppose `time` is a numeric value between 0 and 24 (inclusive) and the day of the week `day` is encoded as Sunday=0, Monday=1, Tuesday=2, ... Saturday=6. You work between 9 AM (`time`=9) and noon (`time=12`) and then from 1 PM (`time`=13) to 5 PM (`time`=17) on weekdays. You don't work on the weekend (Saturday and Sunday), unless you have a deadline (`has_deadline` is a logical (`bool`) value). If you have a deadline then you work between 1 PM and 5 PM on the weekend. (All time intervals are *inclusive*, i.e. including the endpoints.) Write a function `work(time,has_deadline,day)` that returns a `bool` describing whether you are working or not. For example, `work(14, True, 0)` should return `True`; `work(1,True, 2)` should return `False`.

solution

```
def work(time,has_deadline,day):
   if day>0 and day<6:  ## weekday
      if (time>=9 and time<=12) or (time>=13 and time<=17):
       return True
      else:
       return False
   if not has_deadline:
      return False
   if time>=13 and time<=17:
      return True
   return False
```

(this could probably be more compact, but I wrote it following the question).

- -2 points for wrong time endpoints (exclusive vs inclusive)
- -2 points for not enclosing in a function
- -2 points for each `and`/`or` mixup
- -1 points for forgetting about 24-hour time

7. **(12 points)** Given a tuple of integers `T` and a string `s` with the same length (i.e. `len(T)==len(s)`), write a function `dup_letters(s,T)` that returns a new string with each of the letters in `s` duplicated the number of times specified by the matching element in `T`. For example, if `s="hello"` and `T=(1,1,2,3,2)`, the function should return `"hellllloo"`.

```
def dup_letters(s,T):
    ret = ""
    for i in range(len(s)):
        ret += T[i]*s[i]
    return ret
```

8. **(12 points)** Write a function `get_first(L,v)` that returns the index of the first occurrence of a value `v` in a list `L`, **without using the `.index()` method.** For example, `get_first(["a","b","b","c"],"b")` should return 1. (You can assume that `v in L` is `True`.)

```
def get_first(L,v):
   for i in range(len(L)):
      if L[i]==v:
         return i
```

9. **(14 points)** Write a function `collatz(x)` where `x` is an integer. If `x` is even (remember the `%` operator!), divide `x` by 2. If `x` is odd, multiply `x` by 3 and add 1. Keep repeating this rule until `x==1`. The function should return the number of steps that have been executed.

For example, if x is initially 3 (odd), the next value is (step 1) $3x + 1 = 10$. Since 10 is even the next value is (step 2) 10/2=5. The next steps are (3) 5*3+1=16; (4) 16/2=8; (5) 8/2=4; (6) 4/2=2; (7) 2/2=1. The function should return the value 7.

```
def collatz(x):
   i = 0
   while x!=1:
      if x%2 == 0:
           x /= 2
      else:
           x = x*3 + 1
      i+=1
   return i
```

10. **(5 points, EXTRA CREDIT)** Write a function `sort_copy_list(L)` that takes a list `L` and returns a sorted copy of the list, **without modifying the original list**. For example, `L=["a","c","b"]; sort_copy_list(L)` should return the list `["a","b","c"]`, and the value of `L` should be unchanged.

```
def sort_copy_list(L):
   L2 = L.copy()
   L2.sort()
   return(L2)
```

or use `L2=list(tuple(L))` or `L2 = L[::]` or `L2=[]; for v in L: L2.append(v)` or ...

This page is intentionally (mostly) blank.