YOUR NAME PLEASE:

NETID:

Computer Science 200b
Practice Final Exam
April 2016

Enter your netid at the bottom of each page - NOW.

Closed book and closed notes, EXCEPT for one 8.5 x 11 page of notes, which you must hand in with your exam. No electronic devices. Show ALL work you want graded on the test itself. You may not hand in a Blue Book.

For problems that do not ask you to justify the answer, an answer alone is sufficient. However, if the answer is wrong and no derivation or supporting reasoning is given, there will be no partial credit.

GOOD LUCK!

Problem	Points	Actual
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
9	10	
10	10	
11	10	
12	10	
Total	120	

1. (10 points)

Write a plausible value of **ONLY 5** of the following underlined R expressions. No errors occur.

```
> seq(from=1.0, to=2.0, length.out=5)
[1] 1.00 1.25 1.50 1.75 2.00

> c(1:5,5:1,1,5)
[1] 1 2 3 4 5 5 4 3 2 1 1 5

> lst <- 10:20
> lst[c(2,4,6)]
[1] 11 13 15


> as.list(lst[3:4])
[[1]]
[1] 12

[[2]]
[1] 13

> sapply(lst, length)
[1] 1 1 1 1 1 1 1 1 1 1 1

> do.call(paste, c(as.list(lst), sep=""))
[1] "10111213141516171819 20"

> choose(5,3)
[1] 10    # (5!)/(3!(5-3)!) == (5*4*3)/(3*2*1)

> runif(1)
[1] 0.9875961  # 0 < n < 1
```

2. (10 points)

Write the values of **ONLY 5** of the following underlined Python expressions. No errors occur.

```
items = [0,1,2,3,4,5,6]
print(items[slice(2,4)])
```

```
[2, 3]
```

```
def x(i):
    y = set()
    for item in i:
        if item not in y:
            yield item
            y.add(item)

print(list(x([1,5,2,1,9,1,10])))
```

```
[1, 5, 2, 9, 10]
```

```
print('\''.join(['a','b','c']))
```

```
a'b'c
```

```
print(round(2015,-1))
```

```
2020
```

```
a = 1,2,3
print(a)
```

```
(1, 2, 3)
```

```
x = 10
f = lambda y: x + y
x = 20
g = lambda y: x + y
print (f(10), g(10))
```

```
30 30
```

3. (10 points)

Write an R function that replicates the behavior of the following Python function.

```
# input parameter list is a list of strings
def process(list):
    for word in list:
        if (word.lower() == word[::-1].lower()):
            print (word)


>>> x = ['aba', 'abc', 'god', 'Bob']
>>> process(x)
aba
Bob

## R version
library("stringi")

process <- function(lst){
  for (word in lst){
    lword <- tolower(word)
    if (lword == stri_reverse(lword)) {
       print(word)
    }
  }
}

x = c('aba', 'abc', 'god', 'Bob')
> process(x)
[1] "aba"
[1] "Bob"
```

4. (10 points)

Write a Python procedure solve(s), given a string, e.g., “axm”, converts it to a number in any base such that that number is the smallest possible. Each digit must be unique, e.g., a and x could not both be 1. You cannot have a leading 0.

```
>>> solve('cats')
'75 : [1, 0, 2, 3] base: 4'
>>> solve('zig')
'11 : [1, 0, 2] base: 3'
>>> solve('11001001')
'201 : [1, 1, 0, 0, 1, 0, 0, 1] base: 2'
>>> solve('22002002')
'201 : [1, 1, 0, 0, 1, 0, 0, 1] base: 2'
>>> solve('765')
'11 : [1, 0, 2] base: 3'
>>> solve('aab')
'6 : [1, 1, 0] base: 2'
```

```
def solve(s):
    chars = set(list(s))
    base = len(chars)
    digits = list(range(base))
    digits[0] = 1
    digits[1] = 0
    bindings = {}
    for c in s:
        if (c not in bindings):
            bindings[c] = digits.pop(0)
    ans = []
    for c in s:
        ans.append(bindings[c])
    num = 0
    for i, n in enumerate(reversed(ans)):
        num += n * pow(base, i)
    return ("{} : {} base: {}".format(num, ans, base))
```

5. (10 points) Define a python procedure btop() that generates the following bytecode

```
>>> dis.dis(btop)
  6           0 LOAD_CONST               1 (10)
              3 STORE_FAST               0 (i)

  7           6 LOAD_CONST               2 (20)
              9 STORE_FAST               1 (j)

  8          12 LOAD_FAST                0 (i)
             15 LOAD_FAST                1 (j)
             18 BINARY_MULTIPLY
             19 STORE_FAST               2 (k)

  9          22 LOAD_FAST                0 (i)
             25 LOAD_FAST                1 (j)
             28 BINARY_ADD
             29 LOAD_FAST                2 (k)
             32 BINARY_ADD
             33 RETURN_VALUE
```

```
def btop():
    i = 10
    j = 20
    k = i * j
    return (i + j + k)
```

6. (10 points)

Provide the bytecode generated for the following Python function. Use dis.dis() format, but without the source code line numbers from the first column. We have provided the array of comparison operators for reference.

```
def ptob():
   a =  1
   b =  2
   if (a > b):
         return a
   else:
         return b

COMPARE_OPERATORS_SYMBOLS = [
   '<', '<=', '==', '!=', '>', '>=', 'in', 'not in', 'is', 'is not', 'sublclass']
```

```
 12           0 LOAD_CONST               1 (1)
              3 STORE_FAST               0 (a)

 13           6 LOAD_CONST               2 (2)
              9 STORE_FAST               1 (b)

 14          12 LOAD_FAST                0 (a)
             15 LOAD_FAST                1 (b)
             18 COMPARE_OP               4 (>)
             21 POP_JUMP_IF_FALSE       28

 15          24 LOAD_FAST                0 (a)
             27 RETURN_VALUE

 17     >>   28 LOAD_FAST                1 (b)
             31 RETURN_VALUE
             32 LOAD_CONST               0 (None)  ## dead code
             35 RETURN_VALUE
```

7. (10 points) Write the UNIX command(s) corresponding to **XXXX** in the transcript below. You may not use **echo**.

```
bash-4.2$ pwd
/home/accts/sbs5/python/final
bash-4.2$ ls
bash-4.2$ ls -al > pierson
bash-4.2$ ls -l
total 4
-rw-rw-r-- 1 sbs5 sbs5 144 Apr 22 09:25 pierson
bash-4.2$ cat pierson
total 8
drwxrwxr-x 2 sbs5 sbs5 4096 Apr 22 09:25 .
drwxrwxr-x 5 sbs5 sbs5 4096 Apr 22 09:25 ..
-rw-rw-r-- 1 sbs5 sbs5    0 Apr 22 09:25 pierson
bash-4.2$ cp pierson davenport
bash-4.2$ ls -l
total 8
-rw-rw-r-- 1 sbs5 sbs5 144 Apr 22 09:26 davenport
-rw-rw-r-- 1 sbs5 sbs5 144 Apr 22 09:25 pierson
bash-4.2$ diff pierson davenport
bash-4.2$ wc pierson
  4  29 144 pierson
bash-4.2$ id
uid=37645(sbs5) gid=26038(sbs5)
groups=26038(sbs5),11760(cs458),31955(zookeep),49258(cs4\
58ta),63505(cs201ta),63533(cs200ta)
bash-4.2$ chmod 600 pierson
bash-4.2$ ls -l
total 8
-rw-rw-r-- 1 sbs5 sbs5 144 Apr 22 09:26 davenport
-rw------- 1 sbs5 sbs5 144 Apr 22 09:25 pierson
bash-4.2$ date
Fri Apr 22 09:28:42 EDT 2016
bash-4.2$ NOW=$(date)
bash-4.2$ echo $NOW
Fri Apr 22 09:28:44 EDT 2016
bash-4.2$ which which
/usr/bin/which
```

8. (10 points) Python exceptions

Write a Python function f(a,b) which exhibits the following behavior.

```
>>> x = f(1,2)
result is 0.5
done
>>> x
0.5
>>> x = f(1,0)
division by zero!
done
>>> x
```

```
def f(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
        result = None
    else:
        print("result is", result)
    finally:
        print("done")
        return result
```

9. (10 points) Python decorators

Define the decorator href() that has the following behavior.

```
@href
def link(url):
    print (str(url), end="")


>>> link('http://cnn.com')
<a href="http://cnn.com">link</a>
>>>


def href(func):
    def g(*args):
        print ('<a href="', end="")
        func(*args)
        print ('">link</a>\n', end="")
    return g
```

10. (10 points)

Regular expressions. Fill in the following grid, marking an X in each square in which a pattern matches the indexed string. Example: the column for the first string, ‘aaa’, is filled in.

Regular expressions. Fill in the following grid, marking an X in each square in which a pattern matches the indexed string. Example: the column for the first string, 'aaa', is filled in.

Patterns	1	2	3	4	5	6	7	8	9	10	Strings:	
'aaa'	X										1	'aaa'
'...'	X		X	X	X	X	X	X	X	X	2	''
'^...$'	X		X		X		X	X	X		3	'333'
'\.\.\.'								X			4	'4444'
'^[aeiou]*'	X	X	X	X	X	X	X	X	X	X	5	'789'
'^[^aeiou]+'			X	X	X	X	X	X	X		6	'123---456'
'\w\w\w'	X		X	X	X	X	X			X	7	'ABC'
'^\d+$'			X	X	X						8	'...'
'^[0-7]+$'			X	X							9	' '
'^[0-9A-Fa-f]+$'	X		X	X	X		X			X	10	'abcdef'
'^[a-z]*$'	X	X								X		
'^\s+'									X			
'^\d\s?'			X	X	X	X						

See http://zoo.cs.yale.edu/classes/cs200/lectures/f0217a.py

11. (10 points)

Run a simulation in which you generate two random samples: one from a uniform distribution and one from a normal distribution. There are four possible outcomes:

- If the uniform sample is less than .5 and the normal sample is negative: return 1.
- If the uniform sample is less than .5 and the normal sample is positive: return 2.
- If the uniform sample is greater than or equal to .5 and the normal sample is negative: return 3.
- If the uniform sample is greater than or equal to .5 and the normal sample is positive: return 4.

```
> s()  ## uniform sample, normal sample, value
[1] 0.5803783 0.1729185 4.0000000
> s()
[1]  0.8529958 -1.2997085  3.0000000
> s()
[1]  0.6895439 -0.3631031  3.0000000
> s()
[1]  0.0837248 -0.9171370  1.0000000
> s()
[1] 0.2787119 0.6397318 2.0000000
```

```
s <- function(){
  ru = runif(1)
  rn = rnorm(1)
  val = 0
  if (ru < .5) {
     if (rn < 0) { val = 1 }
     else { val = 2 }
  } else {
     if (rn < 0) { val = 3 }
     else {val = 4}
  }

  return ( c(ru, rn, val))
}
```

Other questions

- Python list comprehension
  - See http://zoo.cs.yale.edu/classes/cs200/lectures/f0208.py
- Idempotence
  - See http://zoo.cs.yale.edu/classes/cs200/lectures/0307.html
- Python object oriented programming
  - See http://zoo.cs.yale.edu/classes/cs200/lectures/oop.html