YOUR NAME PLEASE:	
NETID:	

Computer Science 200b
Final Exam
May 5, 2019

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	12	Crypto
2	12	Python
3	12	**Python: OK**
4	12	**Happy number: OK**
5	12	Btop
6	12	P to b:
7	12	**UNIX: OK**
8	12	decorator
9	12	**Regex: OK**
10	12	**shell scripts OK**
Total	120	

1.a. (6 points)

Short answer.

1.a. Describe Kerckhoff's Principle and the rationale behind it.

The inner workings of the cryptosystem are completely known to the attacker, and the only secret is a key. The rationale is that while having a secret cryptosystem may be advantageous it will not stay secret for long, and once exposed these secret systems are often insecure.

The bad guys can try to steal it or one of your own people might be persuaded or bribed to reveal it.

Also, by making it public, as with open source software, it is more likely that a bug or vulnerability might be detected and even corrected.

1.b. What is the difference between integrity and confidentiality? Give examples.

Confidentiality - detecting and preventing unauthorized reading
Integrity - preventing and detecting unauthorized writing

1.c (6 points) Write cencode(s,n) which encodes string s by shifting n positions, mod 26. The program converts s to lowercase and removes all non-alphabetic characters. Here are some examples:

```
>>> cencode('abcde',5)
'fghij'
>>> cencode('ABCDE',5)
'fghij'
>>> cencode('FGHIJ',-5)
'abcde'
>>> cencode('abcde',53)
'bcdef'
>>> cencode("what's going on?",1)
'xibuthpjohpo'
>>> cencode("what's going on?",0)
'whatsgoingon'
>>>
```

```
def cencode(s, n):
    n = n % 26
    result = []
    #s2 = [x for x in s.lower() if x != ' ']
    s2 = [x for x in s.lower() if x.isalpha()]
    for c in s2:
        x = ord(c) - ord('a') + n
        x = (x % 26) + ord('a')
        result.append(chr(x))
    return ''.join(result)
```

2. (12 points)

Write the values of **ONLY 6** of the following underlined Python expressions. No errors occur.

```
>>> len("  hello  ".strip())
5


>>> list(reversed([1,2,3,4])).pop()
1


>>> sorted(['ccc', 'aaaa', 'd', 'bb'], key=len)
['d', 'bb', 'ccc', 'aaaa']



>>> set([4,3,2,1,1,2,3,4,5])
{1, 2, 3, 4, 5}



>>> [int(x)* int(x) for x in str(12345)]
[1, 4, 9, 16, 25]



>>> list(filter (lambda x: x % 2 == 0, range(10)))
[0, 2, 4, 6, 8]


>>> (lambda x: (x << 5) + x)(2)
66


>>> {x for x in range(10) if x % 2 == 0}
{0, 2, 4, 6, 8}



>>> {x1:x2 for (x1,x2) in enumerate("abcde")}
{0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e'}
```

3. (12 points)

Write a Python function, f(n), which prints the sides of all right triangles with integer sides less than n, as demonstrated below. Remember, $a^2 + b^2 = c^2$. To get full credit, write it as a list comprehension.

```
>>> f(10)
[(3, 4, 5)]
>> f(15)
[(3, 4, 5), (5, 12, 13), (6, 8, 10)]
>>> f(30)
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9,
12, 15), (10, 24, 26), (12, 16, 20), (15, 20, 25), (20, 21, 29)]
>>> f(50)
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9,
12, 15), (9, 40, 41), (10, 24, 26), (12, 16, 20), (12, 35, 37), (15,
20, 25), (15, 36, 39), (16, 30, 34), (18, 24, 30), (20, 21, 29), (21,
28, 35), (24, 32, 40), (27, 36, 45)]
```

```
def f(n):
   return [(x,y,z) for x in range(1,n) for y in range(x,n) for z in
range(y,n) if x**2 + y**2 == z**2]
```

4. (12 points) A **happy number** is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number either equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are **happy numbers**, while those that do not end in 1 are **unhappy numbers** (or **sad numbers**).

For example, 19 is happy, as the associated sequence is:

$1^2 + 9^2 = 82$

$8^2 + 2^2 = 68$

$6^2 + 8^2 = 100$

$1^2 + 0^2 + 0^2 = 1.$

Write the Python procedures happy(n) and is_happy(n) which have the following behavior. You may also write auxiliary functions. (Note that 4 is not happy.)

```
>>> happy(19)
82
>>> happy(82)
68
>>> happy(68)
100
>>> happy(100)
1
>>> happy(4)
16
>>> happy(16)
37
>>> happy(37)
58
>>> happy(58)
89
>>> happy(89)
145
>>> happy(145)
42
>>> happy(42)
20
>>> happy(20)
4
>>> is_happy(19)
True
>>> is_happy(100)
True
>>> is_happy(4)
False
>>> is_happy(2)
False
```

4. (continued)

```
def square(x):
    return int(x) * int(x)

def happy(number):
    return sum(map(square, list(str(number))))

def is_happy(number):
    seen_numbers = set()
    while number > 1 and (number not in seen_numbers):
        seen_numbers.add(number)
        number = happy(number)
    return number == 1
```

5. (12 points) Define a Python procedure bytetopython() that generates the following bytecode

```
>>> dis.dis(bytetopython)
 35           0 LOAD_CONST               1 (10)
              3 STORE_FAST               0 (a)

 36           6 LOAD_CONST               2 (20)
              9 STORE_FAST               1 (b)

 37          12 LOAD_CONST               3 (30)
             15 STORE_FAST               2 (c)

 38          18 LOAD_FAST                0 (a)
             21 LOAD_FAST                1 (b)
             24 BINARY_ADD
             25 LOAD_FAST                2 (c)
             28 BINARY_ADD
             29 STORE_FAST               3 (d)

 39          32 LOAD_FAST                3 (d)
             35 LOAD_CONST               4 (3)
             38 BINARY_TRUE_DIVIDE
             39 RETURN_VALUE
```

```
def bytetopython():
    a = 10
    b = 20
    c = 30
    d = a + b + c
    return d / 3
```

6. (12 points)

Provide the bytecode generated for the following Python function, pythontobyte(). Use dis.dis() format, but without the source code line numbers from the first column.

```
def pythontobyte():
    x0 = 3
    v0 = 2
    t = 10
    a = 9.8
    return x0 + v0*t + .5*a*t*t
```

```
>>> dis.dis(pythontobyte)
 42           0 LOAD_CONST               1 (3)
              3 STORE_FAST               0 (x0)

 43           6 LOAD_CONST               2 (2)
              9 STORE_FAST               1 (v0)

 44          12 LOAD_CONST               3 (10)
             15 STORE_FAST               2 (t)

 45          18 LOAD_CONST               4 (9.8)
             21 STORE_FAST               3 (a)

 46          24 LOAD_FAST                0 (x0)
             27 LOAD_FAST                1 (v0)
             30 LOAD_FAST                2 (t)
             33 BINARY_MULTIPLY
             34 BINARY_ADD
             35 LOAD_CONST               5 (0.5)
             38 LOAD_FAST                3 (a)
             41 BINARY_MULTIPLY
             42 LOAD_FAST                2 (t)
             45 BINARY_MULTIPLY
             46 LOAD_FAST                2 (t)
             49 BINARY_MULTIPLY
             50 BINARY_ADD
             51 RETURN_VALUE
```

7. (12 points) Write the UNIX command(s) corresponding to **XXXX** in the transcript below. You may use **echo** once.

```
bash-4.4$ ls
final.py
bash-4.4$ ls . > a
bash-4.4$ cp a b
bash-4.4$ ls -l
total 4
-rw-rw-r-- 1 sbs5 sbs5  11 Apr 26 13:42 a
-rw-rw-r-- 1 sbs5 sbs5  11 Apr 26 13:42 b
-rwxr-x--- 1 sbs5 sbs5 205 Apr 26 13:36 final.py
bash-4.4$ mkdir c
bash-4.4$ ls
a  b  c  final.py
bash-4.4$ file c
c: directory
bash-4.4$ sleep 30 &
[1] 26538
bash-4.4$ ps
  PID TTY          TIME CMD
26047 pts/0    00:00:00 bash
26538 pts/0    00:00:00 sleep
26539 pts/0    00:00:00 ps
bash-4.4$ jobs
[1]+  Running                 sleep 30 &
bash-4.4$ chmod 700 b
[1]+  Done                    sleep 30
bash-4.4$ ls -l
total 8
-rw-rw-r-- 1 sbs5 sbs5   11 Apr 26 13:42 a
-rwx------ 1 sbs5 sbs5   11 Apr 26 13:42 b
drwxrwxr-x 2 sbs5 sbs5 4096 Apr 26 13:42 c
-rwxr-x--- 1 sbs5 sbs5  205 Apr 26 13:36 final.py
bash-4.4$ rmdir c
bash-4.4$ ls
a  b  final.py
bash-4.4$ HELLO=$(ls)
bash-4.4$ echo $HELLO
a b final.py
bash-4.4$ alias whatsup=date
bash-4.4$ whatsup
Thu Apr 26 13:57:26 EDT 2018
```

```
bash-4.4$ whatsup
Thu Apr 26 13:57:38 EDT 2018
bash-4.4$ echo $PATH
/usr/libexec/python3-sphinx:/usr/lib64/qt-3.3/bin:/opt/pgi/linux86-64
/17.4/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
bash-4.4$ cat a
a
final.py
bash-4.4$ ls
a  b  final.py
bash-4.4$ ls >> a
bash-4.4$ diff a b
3,5d2
< a
< b
< final.py
bash-4.4$ ls -l
total 4
-rw-rw-r-- 1 sbs5 sbs5  24 Apr 26 14:00 a
-rwx------ 1 sbs5 sbs5  11 Apr 26 13:42 b
-rwxr-x--- 1 sbs5 sbs5 205 Apr 26 13:36 final.py
```

8. (12 points) Python exceptions and decorators

Write a Python decorator function `safe(func)` that lets a function raise an exception without halting execution. If no exception is raised, the decorated function returns the appropriate value. If an exception is raised, the message “Something bad happened.” is printed. In both cases, the message “All done.” appears. Here are some examples.

```
@safe
def f(d):
    return 100//d

@safe
def lookup(key):
    d = {'one': 1}
    return d[key]

>>> f(10)
All done.
10
>>> f(0)
Something bad happened.
All done.
>>> lookup('one')
All done.
1
>>> lookup('two')
Something bad happened.
All done.
```

```
def safe(func):
    def f(*args):
        try:
            result = func(*args)
        except:
            print ("Something bad happened.")
        else:
            return result
        finally:
            print ("All done.")
    return f
```

9. (12 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, 'xxx', is filled in.

Patterns	1	2	3	4	5	6	7	8	9	10	Strings:	
'xx'	X										**1**	'xxx'
'^x+'	X										**2**	''
'...',	X		X	X	X	X	X	X	X	X	**3**	'345'
'^\\...'				X				X			**4**	'.A4'
'^\w\w\w'	X		x		x		x			x	**5**	'789'
'^[a-z]'	X									x	**6**	' 456 '
'^[^a-z]'			x	X	x	x	X	x	x		**7**	'ABC'
'\w\W\w’											**8**	'...'
'^[0-7]+$'			X								**9**	' '
'^[0-9A-Fa-f]+$'			X		X		X			X	**10**	'abcdef'
'^[A-Z]*$'		X					x					
'^\s.+\s$'						X			X			
'^\d.\d$'			X		X							

10. (12 points)

For each of the following three shell scripts, write their output when invoked as indicated. (3 points each)

10.a.

```
#! /usr/bin/bash
# s1.sh

dkdkdkd &> /dev/null
R1=$?
date > /dev/null
R2=$?
echo $((R1 < R2))
```

```
bash-4.4$ ./s1.sh
0
```

10.b.

```
#! /usr/bin/bash
# s2.sh

X=$(($1 + $2 - $3))
echo $X
```

```
bash-4.4$ ./s2.sh 1 2 3 4 5
0
bash-4.4$ ./s2.sh 5 4 3 2 1
6
```

10.c.

```
#! /usr/bin/bash
# s3.sh

s=0
for f in $*; do
   if (( $f > $s )); then
     s=$f
   fi
done
echo $s
```

```
bash-4.4$ ./s3.sh
0
bash-4.4$ ./s3.sh 1 2 3 4 5 6
6
```

10.d. Write a shell script which has the following behavior. It prints information about the last time the user logged into this machine. (3 points)

```
bash-4.4$ ./s4.sh
sbs5     pts/2        172.28.20.216    Fri Apr 13 10:30 - 11:21  (00:50)
```

```
#! /usr/bin/bash
# sh4.sh

last | grep $USER | tail -1
```