Ever so clearly print your email id:

Ever so clearly print your name:

Pledge:

Notices
- Based on your past educational achievements, I expect you to do well on this test.
- Answer the questions in any order that you want.

Test rules
- Before you leave the room, check that you uploaded all six of your solutions. Do not ask afterwards whether you can submit a forgotten solution.
- This pledged exam is closed notes. The only device you may access during the test is your laptop.
- Any cheating can result in failing the class and the incident being referred to the Honor Committee.
- Do not access class examples artifacts, web solutions, or your own past assignments during the test; that is, the only code you may access or view are ones that you develop for this test.
- The only windows allowed on your laptop are PyCharm and a single browser with tabs reachable from class website.

PyCharm
- PyCharm can be used for developing the modules to be submitted. It cannot be used for the short answer questions of Part 1.

Programming
- Modules should follow class programming practices; e.g., whitespace, identifier naming, and commenting if you think it is needed, etc.
- Whether a module is testable is important.
- Comment out or delete all debugging print() statements before submitting.
Part I (22 points)

1. TRUE FALSE The Python looping statements are the if, for, and while.

2. TRUE FALSE Function parameters are named in the function definition.

3. TRUE FALSE Function parameters are named in a function invocation.

4. TRUE FALSE A function parameter can be used as an argument in a function invocation.

5. TRUE FALSE Function invocations require the use of parentheses.

6. TRUE FALSE No matter what the unknown function f() does, when the below code segment completes, it outputs 123.
   
   ```
   x = 123
   f( x )
   print( x )
   ```

7. TRUE FALSE No matter what the unknown function f() does, when the below code segment completes it outputs 123.
   
   ```
   x = 123
   x = f( x )
   print( x )
   ```

Suppose the following code segment is in effect.

```python
def f( x ) :
    y = 10 * x

a = 2
b = f( a )
```

8. TRUE FALSE An invocation of function f() produces a return value.

9. TRUE FALSE The expression f( a ) is an invocation of function f().

10. TRUE FALSE a is a local variable of function f().

11. TRUE FALSE y is a local variable of function f().
Suppose the following function definitions are in effect.

```python
def s(a):
    a = 1112
    return a
def t(a):
    a = 1112
    return a

def u(a):
    a[0] = 1112
```

12. What is the output of the following code segment?
```python
x = 1
s(x)
print(x)
```

13. What is the output of the following code segment?
```python
a = 1
s(a)
print(a)
```

14. What is the output of the following code segment?
```python
x = 1
t(x)
print(x)
```

15. What is the output of the following code segment?
```python
a = 1
t(a)
print(a)
```

16. What is the output of the following code segment?
```python
x = 1
x = t(x)
print(x)
```

17. What is the output of the following code segment?
```python
x = [3, 1, 4]
u(x)
print(x)
```

Suppose the following function definition is in effect.

```python
def f(x):
    x.append(100)
```

18. What does the following code segment output?
```python
a = [1]
f(a)
print(a)
```
Suppose the following function definition is in effect.

```python
def f(x):
    x = [100]
```

19. What does the following code segment output?

```python
a = [1]
f(a)
print(a)
```

Suppose the following function definition is in effect.

```python
def f(x):
    for v in x:
        if (v <= 0):
            return False
        else:
            return True
```

20. **TRUE**          **FALSE**  Function f() correctly determines whether list x consists of all positive values.

Suppose the following function definition is in effect.

```python
def f(x, y):
    remember = x
    x = y
    y = remember
    return x, y
```

21. **TRUE**          **FALSE**  The below code segment correctly swaps the values of a and b.

```python
a = 11; b = 12
a, b = f(a, b)
```

22. What does the following code segment output?

```python
a = [3, 1, 4, 1]
i = 1
while (i in a):
    print(i)
    i = i + 1
```
Part 2: Programming (78 points)

23. Implement module `me.py`. The module defines a function `id()`. The function has no parameters. The function does not print anything.

The function returns a lowercase alphanumeric string. The string is to be your University of Virginia email id. For example, if your email id was mst3k, the tester should produce the following output.

```
me.id(): mst3k
```

24. Implement module `lin.py`. The module defines a function `ear()`. The function has three numeric parameters \( m, b, \) and \( x \). The function does not print anything. The function returns the value of the linear equation \( mx + b \).

The tester should produce the following output.

```
lin.ear( 3 , 5 , 4 ): 17
lin.ear( 2 , 4 , 3 ): 10
lin.ear( 10 , 15 , 2 ): 35
```

25. Implement module `ph.py`. The module defines a function `one()`. The function has a numeric string parameter `ns`. The function does not print anything.

Parameter `ns` represents a phone number. The first three digits in the string are the area code; the next three digits are the prefix; and the last four digits are the line number.

The function returns a three-element integer list. The first element of the list is the `ns` area code in integer form; the middle element of the list is the `ns` prefix in integer form; the last element of the list is the `ns` line number in integer form.

The tester should produce the following output.

```
ph.one( '2024561111' ): [202, 456, 1111]
ph.one( '8602941986' ): [860, 294, 1986]
ph.one( '2125552368' ): [212, 555, 2368]
```

26. Implement module `tab.py`. The module defines a function `le()`. The function has one dataset parameter \( d \) and one integer column index \( c \). The function does not print anything. The function does not make any changes to its parameters.

The function returns a list. The elements of that list are column \( c \) values for the rows of dataset \( d \).

Suppose the dataset parameter of interest is the three-row list \([[5, 6, 5], [7, 3, 5, 5], [4, 7, 9, 8, 2, 3]]\) and the column parameter of interest is 1, the return value is \([6, 3, 7]\), because the 1th elements of \([5, 6, 5], [7, 3, 5, 5], \) and \([4, 7, 9, 8, 2, 3]\) are respectively 6, 3, and 7.

The built-in tester runs four tests using the following datasets to initialize parameter \( d \) respectively.

\[
\begin{align*}
d0 &= [[1], [2, 4], [5, 3, 7, 7, 3, 3]] \\
d1 &= [[5, 6, 5], [7, 3, 5, 5], [4, 7, 9, 8, 2, 3]] \\
d2 &= [[1, 4, 6], [4, 7, 2, 7], [3, 8, 4, 5, 8, 5]] \\
d3 &= [[3, 1, 4, 1, 5, 9]]
\end{align*}
\]
The tester should produce the following output.

<table>
<thead>
<tr>
<th>table( d0, 0 ):</th>
<th>[1, 2, 5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>table( d1, 1 ):</td>
<td>[6, 3, 7]</td>
</tr>
<tr>
<td>table( d2, 2 ):</td>
<td>[6, 2, 4]</td>
</tr>
<tr>
<td>table( d3, 3 ):</td>
<td>[1]</td>
</tr>
</tbody>
</table>

27. Implement module *di.py*. The module defines a function `ction()`. The function has one list parameter `x`. The function does not print anything. The function does not make any changes to its parameter.

The function returns a dictionary. The keys to that dictionary are the values of `x`. For each element in `x` there is an entry in the dictionary that maps that element to the number of times it appears in `x`.

For example, suppose `x` equals `['m', 'i', 'm', 'i', 'c']`, then the dictionary maps 'c' to 1, 'i' to 2, and 'm' to 2.

The built-in tester runs four tests using the following datasets to initialize parameter `x` respectively.

- `x0 = ['m', 'i', 'm', 'i', 'c']`
- `x1 = [3, 1, 2, 2, 1, 2]`
- `x2 = [True, False, True, True]`

The tester should produce the following output.

<table>
<thead>
<tr>
<th>di.ction( x0 ):</th>
<th>{ c: 1, i: 2, m: 2 }</th>
</tr>
</thead>
<tbody>
<tr>
<td>di.ction( x1 ):</td>
<td>{ 1: 2, 2: 3, 3: 1 }</td>
</tr>
<tr>
<td>di.ction( x2 ):</td>
<td>{ False: 1, True: 3 }</td>
</tr>
</tbody>
</table>

28. Implement module *al.py*. The module defines a function `ike()`. The function has two list parameters `x` and `y`. The function does not print anything. The function does not make any changes to its parameters.

The function returns whether `x` and `y` are alike; that is, whether the list values are permutations of each other. To be permutations, `x` and `y` and must have the same length and the same element counts.

For example, suppose `x` is `[1, 1]` and `y` is `[1, 1, 1]`, then the function returns `False` because the lists do not have the same number of elements. As a second example, suppose `x` is `[1, 2, True, 2]` and `y` is `[True, 2, 2, 1]`, then the function returns `True` because while their orderings are different, their values are the same. As a third example, suppose `x` is `[1, 2, 'a']` and `y` is `[2, 1, 2]`, then the function returns `False` because `y` does not have an 'a' like `x` does.

The built-in tester runs four tests using the following datasets to initialize parameters `x` and `y` respectively.

- `x0 = [1, 1]`
- `x1 = [1, 2, True, 2]`
- `x2 = [1, 2, 'a']`
- `x3 = []`
- `y0 = [1, 1, 1]`
- `y1 = [True, 2, 2, 1]`
- `y2 = [2, 1, 2]`
- `y3 = []`

The tester should produce the following output.

<table>
<thead>
<tr>
<th>al.ike( x0, y0 ):</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>al.ike( x1, y1 ):</td>
<td>True</td>
</tr>
<tr>
<td>al.ike( x2, y2 ):</td>
<td>False</td>
</tr>
<tr>
<td>al.ike( x3, y3 ):</td>
<td>True</td>
</tr>
</tbody>
</table>