Ever so clearly print your email id:

Ever so clearly print your name:

TRUE or FALSE: I verified my solutions have been uploaded.

Pledge:

Test rules

- Before you leave the room, check that you uploaded all seven 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 exam is your laptop.
- Please be honorable as you pledged to be at the start of the semester, because any violations can result both in failing the class and the incident being referred to the Honor Committee.
- You may not access class examples, artifacts, solutions on the web, 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.

Modules

- 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.

Short answer questions answers

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____
9. _____
10. _____
11. _____

THIS PAGE IS ALMOST BLANK

Part I Short answers (22 points)

1. TRUE or FALSE: The return value of the following function `f()` is the string `"ABC"`.

```
def f() :
    print( "ABC" )
```

2. TRUE or FALSE: The return value of the following function `f()` is 1.

```
def f() :
    return 1
    return 2
```

3. TRUE or FALSE: In the body of the following function `g()`, a function `f()` is invoked.

```
def g() :
    x = 2 + f
    return x
```

4. TRUE or FALSE: In the following function `f()`, variable `x` is a function parameter.

```
def f( x ) :
    return 2 * x
```

5. TRUE or FALSE: The following function `f()`, returns whether `x` is odd.

```
def f( x ) :
    if ( x % 2 == 0 ) :
        return True
    else :
        return False
```

Suppose the following function definition is in effect.

```
def f( a ) :
    a = "xyz"
```

6. What is the output of the following code segment?

```
x = "abc"
f( x )
print( x )
```

7. What is the output of the following code segment?

```
a = "abc"
f( a )
print( a )
```

Suppose the following function definition is in effect.

```
def f( a ) :
    a = "xyz"
    return a
```

8. What is the output of the following code segment?

```
x = "abc"
f( x )
print( x )
```

9. What is the output of the following code segment?

```
a = "abc"
f( a )
print( a )
```

10. What is the output of the following code segment?

```
x = "abc"
x = f( x )
print( x )
```

Suppose the following function definition is in effect.

```
def f( a ) :
    a[ 0 ] = "xyz"
```

11. What is the output of the following code segment?

```
x = [ "abc", "def" ]
f( x )
print( x )
```

Part II Implementation (78 pts)

12. Implement module *khony.py*. The module defines a function ind(). The function has two parameters x and y. The function does not print anything or get any input.

    The function characterizes how x and y compare to each other. It returns either -1, 0, or 1 depending respectively whether x < y, or x == y, or x > y. The built-in tester should produce the following output.

```
khony.ind( aunt , tia ): -1
khony.ind( 314 , 272 ): 1
khony.ind( 0.0 , 0.0 ): 0
```

13. Implement module *mina.py* . The module defines a function don(). The function has two numeric parameters d and h. The function does not print anything or get any input.

    The function returns the volume of a donut whose width is d and whose hole width is h. The formula for the volume of a donut is

$$\pi^2 (d^2 - h^2) (d - h) / 32$$

Your implementation of the formula should make use of the `math` module variable for Π (`pi`). The built-in tester should produce the following output.

```
mina.don( 4.5 , 1.5 ): 16.654957426838294
mina.don( 4 , 2 ): 7.4022033008170185
```

14. Implement module *amake.py* . The module defines a function `med()`. The function has five integer parameters v, w, x, y, and z. The function does not print anything or get any input.

    The function returns the median of v, w, x, y, and z; that is, the middle value of a sorted version of the five parameter values. The built-in tester should produce the following output.

```
amake.med( 17 87 21 46 84 ): 46
amake.med( 40 90 12 24 95 ): 40
amake.med( 30 34 14 65 49 ): 34
```

15. Implement module *fi.py*. The module defines a function `ser()`. The function has one integer parameter n. The function does not print anything or get any input.

    The function accumulates and returns the sum of the *n+1* values in the series

    $1/2^0, 1/2^1, 1/2^2, 1/2^3, \ldots, 1/2^n$

    The built-in tester should produce the following output.

```
fi.ser( 0 ): 1.0
fi.ser( 1 ): 1.5
fi.ser( 25 ): 1.9999999701976776
fi.ser( 75 ): 2.0
```

16. Implement module *kuv.py* . The module defines a function `cro()`. The function has two integer list parameters x and y (*you may assume x and y have the same length*). The function does not print anything or get any input. The function ***does not alter*** the elements of x and y.

    The function returns a ***new*** list. The return list is the *element-product* of x and y; that is, the list

    $[\, x[0]\cdot y[0],\ x[1]\cdot y[1],\ x[2]\cdot y[2],\ \ldots,\ x[n-1]\cdot y[n-1]\,]$

    For example, if

    x = [3, 1, 4, 1, 5, 9]

    y = [2, 7, 1, 8, 2, 8]

    then the element-product is,

    [3·2, 1·7, 4·1, 1·8 , 5·2, 9·8] = [6, 7, 4, 8, 10, 72]

    The built-in tester should produce the following output

```
kuv.cro( [3, 1, 4, 1, 5, 9] , [2, 7, 1, 8, 2, 8] ): [6, 7, 4, 8, 10, 72]
kuv.cro( [1, 2, 3, 4] , [5, 6, 7, 8] ): [5, 12, 21, 32]
kuv.cro( [3, 5, 7, 9, 11] , [2, 4, 6, 8, 10] ): [6, 20, 42, 72, 110]
```

17. Implement module *mich.py*. The module defines a function dec(). The function has two parameters, a dictionary parameter d and a string s. The function does not print anything or get any input. The function *does not alter* the mappings of d.

    The function uses d to help translate (decode) the characters in s and then return that translation. For each character in s, if the character is a key in d, then it translates to value of that character according to d. If instead, the character is not a key in d, the character translates to the underscore character "_".

    For example, if d equals the below dictionary d1, and s equals "a b c", then the return string substitutes "x" for "a", "y" for "b", and "z" for "c". Because the space character " ", is not a key in d, then the return string substitutes the underscore character "_" for a space. So, the return string would be "x_y_z".

```
d1 = { "a": "x", "b": "y", "c": "z" }
d2 = { "o" : "u", "p": "s", "s": "n", "u": "t" }
d3 = { "a": "a", "c" : "p", "h" : "e", "o": "c", "s": "e" }
```

    The built-in tester should produce the following output.

```
mich.dec( d1 , a b c ): x_y_z
mich.dec( d2 , soup ): nuts
mich.dec( d3 , &#chaos[] ): __peace__
```

18. Implement module *iau.py*. The module defines a function pro(). The function has an integer dataset parameter d. The function does not print anything or get any input. The function ***does not alter*** elements d.

    The function returns the product of the dataset values in d. The built-in tester runs three tests using datasets d1 and d2.

```
d1 = [ [5], [2, 3], [2, 2, 1] ]
d2 = [ [5], [9, 2, 6, 8], [9, 5, 4, 9, 1] ]
d3 = [ [7, 5, 4], [3, 4, 1, 9, 6, 1], [3, 3], [5], [5, 8] ]
```

    The built-in tester should produce the following output.

```
iau.pro( d1 ): 120
iau.pro( d2 ): 6998400
iau.pro( d3 ): 163296000
```