Clearly print your computing id:

Clearly print your name:

Pledge:

Test rules

- ***Check that you uploaded your solutions. Do not ask afterwards whether you can submit a forgotten solution.***
- ***The only device you may access during the exam is your laptop.***
- ***You may not access class notes, epistles, examples, artifacts, solutions on the web, or your own past assignments during the test.***
- ***The only code you may access are ones that you develop for this test.***
- ***The only open windows allowed are PyCharm and a browser with tabs linked from the class website.***
- ***PyCharm cannot be used for the short answer questions of Part I.***
- ***Code should follow class programming practices; e.g., whitespace, identifier naming, etc.***
- ***Whether code is testable is important.***
- ***Make sure each function has at least one statement in its body or it will not compile.***
- ***Comment out or delete all debugging `print()` statements before submitting.***
- ***None of your functions should print anything or get input.***

Short answer questions answers

1. ________________________________

2. ________________________________

3. ________________________________

4. ________________________________

5. ________________________________

6. ________________________________

7. ________________________________

8. ________________________________

9. ________________________________

10. ________________________________

11. ________________________________

12. ________________________________

13. ________________________________

14. ________________________________

15. ________________________________

16. ________________________________

17. ________________________________

THIS PAGE ALMOST BLANK

Part I Short answers (34 points)

When indicating values: If the value is integer, do not use a decimal point. If the value is decimal, use a decimal point. If the value is a string, enclose it within quotes. If the value is logical, do not use quotes.

Suppose the following definitions are in effect for the five questions.

```
a = 10;                        d = 0.75
b = 4;                         e = 1 // 10
c = 2;                         f = "1"
```

1. What is the value of g after its assignment statement is executed?

```
g = a / b
```

2. What is the value of h after its assignment statement is executed?

```
h = a % b
```

3. What is the value of i after its assignment statement is executed?

```
i = c * d
```

4. What is the value of j after its assignment statement is executed?

```
j = a * e
```

5. What is the value of k after its assignment statement is executed?

```
k = f + f
```

6. True or False: The following equation correctly swaps the values of x and y.

```
y, x = x, y
```

7. Consider the following code segment. What does it output?

```
x = 1
y = 1
z = ( id( x ) == id( y ) )
print( z )
```

8. True or False: In CS 1112 class terminology, argument and parameter are synonyms.

9. True or False: In CS 1112 class terminology, function definition and function invocation are synonyms.

10. What should be the if test expression, for function z() to return True if x and y have the same type; and to return False, otherwise.

```
def z( x, y ) :
    if ( ... ) :
        return True
    else :
        return False
```

11. What should be the initialization for n, for function z() to return the number of rows in dataset x?

```
def z( x ) :
    n = ...
    return n
```

12. True or False: The following code segment prints the list `[2, 4, 6]`.

```
s = "2 4 6"
t = int( s )
print( t )
```

13. True or False: The following code segment prints the list `[2, 4, 6]`.

```
s = "2 4 6"
t = s.split()
u = int( s )
print( u )
```

14. True or False: The following code segment prints the string `"ABC"`.

```
s = "abc"
s.upper()
print( s )
```

Consider the following statement, where `lib` is Python module with a function definition `f()`.

```
r = lib.f( x )
```

15. True or False: For the code to run correctly, module `lib` must be previously imported.

16. True or False: For the code to run correctly, variable `x` must have been previously assigned a value.

17. True or False: If `x` is an integer, the return value of `f()` must be integer.

Part II Implementation (105 points)

18. Implement module *truth.py*. The module defines a function `dare()`. The function has no parameters. The function returns the logical value `True`.

    The built-in tester should produce the following output.

    ```
    dare(): True
    ```

19. Implement program *eval.py*. The program has a single prompt that expects a formula composed of an integer n1, a character c, and another integer n2.

    - If character c is a "+", then the output is the sum of n1 + n2.
    - If character c is a "−", then the output is the difference n1 – n2.
    - If character c is a "∗", then the output is the product n1 ∗ n2.
    - If character c is anything else, then the output is "Bad input".

    The program does not print anything other than the requested value. Sample runs could be.

    ```
    Enter formula: 2 + 3
    5
    ```

    ```
    Enter formula : 12 – 5
    7
    ```

    ```
    Enter formula : 3 ∗ 15
    45
    ```

    ```
    Enter formula : 16 / 2
    Bad input
    ```

20. Implement module *grow.py*. The module defines a function `up()` with a single string parameter s. The function returns a new string identical to s except each character in s occurs twice in the return string.

    The built-in tester for the module should produce the following output.

    ```
    up( abc ): aabbcc
    up( woah! ): wwooaahh!!
    ```

21. Implement module *case.py*. The module defines a function `swap()` with a single string parameter s. The function returns a new string equal to s except the case of the characters is reversed; that is, if a character is lowercase in s, then it is uppercase in the return string, and vice-versa.

    The built-in tester for the module should produce the following output.

    ```
    swap( ee cummings ): EE CUMMINGS
    swap( CS 1112 ): cs 1112
    swap( aBcD ): AbCd
    ```

22. Implement module *stray.py*. The module defines a function `conv()` with a single string parameter `s`. The function returns a new list where each element in the list is an individual characters of `s`.

    The built-in tester for the module should produce the following output.

```
conv( apple ): ['a', 'p', 'p', 'l', 'e']
conv( banana ): ['b', 'a', 'n', 'a', 'n', 'a']
```

23. Implement module *`geo.py`* . The module defines a function `series()` with `int` parameters x and n. The function returns a new list with n elements whose values are

    $(-x)^1$, $(-x)^2$, $(-x)^3$, …, $(-x)^n$.

    The built-in tester for the module should produce the following output.

```
series( 2 , 5 ): [-2, 4, -8, 16, -32]
series( 3 , 4 ): [-3, 9, -27, 81]
```

24. Implement module *ring.py*. The module defines a function `chomp()`,with three parameters `s`, `i`, and `j`, where `s` is a string, and `i` and `j` are indices into `s`. The function returns a new string that equals the characters of `s` starting at index 0 and up to but not including the character at index `i`, followed by the character in `s` from index `j` onward.

    The built-in tester should produce the following output.

```
chomp( ABCDEFGHIJ , 2 , 6 ): ABGHIJ
```

25. Implement module *tab.py*. The module defines a function `square()`, with a single dataset parameter `x`. The function returns True if the number of columns in each row of dataset `x` is the same as the number of rows in dataset `x`; otherwise, the function returns `False`.

    The built-in tester for the module should produce the following output.

```
square( [[2]] ): True
square( [[2, 4, 6]] ): False
square( [[2], [4], [6]] ): False
square( [[2, 4, 6], [1, 3, 5], [7, 8, 9]] ): True
```

26. Implement module *date.py*. The module defines a function `avg()` with a single dataset parameter `x`. The function returns the decimal average of the values in dataset `x`.

    The built-in tester for the module should produce the following output.

```
avg( [[1, 2]] ): 1.5
avg( [[2, 4, 6], [8], [10, 12]] ): 7.0
```

27. Implement module *riches.py*. The module defines a function `look()` with a single dictionary parameter `x`, where the values in the *set* `x.values()` are all integers. The function returns the key `k` in `x` whose *dictionary value* `x[k]` is maximum. Note: you can only assume `x.values()` are all integers; i.e., it is possible that the key with the maximum dictionary value has a negative dictionary value.

The built-in tester for the module should produce the following output.

```
look( {'A': -100, 'B': 500, 'C': -300, 'D': 400} ): B
look( {'A': -100, 'B': -1000, 'C': -50, 'D': -100000000} ): C
```

28. Implement module *icto.py*. The module defines a function `uniq()` with a single dictionary parameter `x` and a dictionary key parameter `k`. The function returns `True` if no other key in `x` has the same dictionary value as `k`; otherwise, the function returns `False`. Be aware that `x.values()` is not a list. As such, there is no `count()` function for `x.values()`.

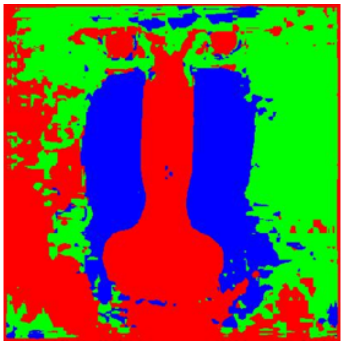
    The built-in tester for the module should produce the following output.

```
uniq( {'A': 1, 'B': 2, 'C': 2, 'D': 4} , A ): True
uniq( {'A': 1, 'B': 2, 'C': 2, 'D': 4} , C ): False
```

29. Implement module *picmax.py*. The module defines a function `rgb()` with a single pixel parameter `p`. The function returns

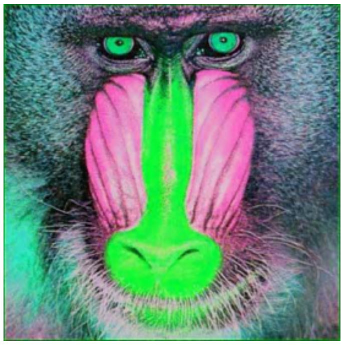
    - (255, 0, 0), if the R value is the maximum of p's RGB values.
    - (0, 255, 0), if the G value is the maximum of p's RGB values.
    - (0, 0, 255), if the B value is the maximum of p's RGB values.

    The built-in tester for the module should produce the following image.



30. Implement module *picspin.py*. The module defines a function `rotate()` with a single pixel parameter `p`. The function returns a new pixel whose R value is p's B value, whose G value is p's R value, and whose B value is p's G value.

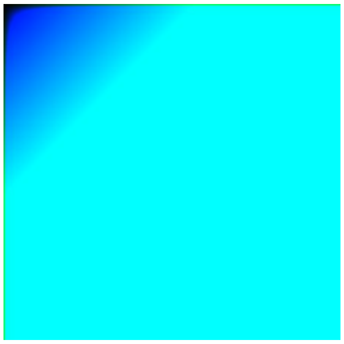
    The built-in tester for the module should produce the following image.



31. Implement module *picroll.py*. The module defines a function `gradient()` with two integer parameters x and y. The function neither prints anything or gets any input.

    Function `gradient()` returns a new pixel whose R value is 0, whose G value is the min of 255 and x + y, and whose B value is the min of 255 and x • y.

    The built-in tester for the module should produce the following image.



32. Implement module *accts.py*. The module defines a function `audit()` with a dictionary parameter x, and a character c. The values in `x.values()` are all integers.

    - If character c is a "-", the function returns the list of keys in x, whose dictionary entries are negative.
    - If character c is a "0", the function returns the list of keys in x, whose dictionary entries are zero.
    - If character c is a "+", the function returns the list of keys in x, whose dictionary entries are positive.
    - If character c is anything else, the function returns an empty list.

    The built-in tester for the module uses the following dictionary

```
d = { 'A': 1, 'B': 0, 'C': -3, 'D': 4, 'E': 0, 'F': 0, 'G': 1, 'H': 5 }
```

When testing `accts()`, the tester should produce the following output.

```
accts( d, - ): ['C']
accts( d, + ): ['A', 'D', 'G', 'H']
accts( d, 0 ): ['B', 'E', 'F']
accts( d, * ): []
```