For the following questions, use these variable definitions.

\[
\begin{align*}
  a &= 4444 \\
  b &= 2 \\
  c &= 11.11 \\
  d &= "hello" \\
\end{align*}
\]

What is the value and type of each of the following expressions or, if it won't compile, circle that answer.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
<th>Compile Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (2 pts) ( a \div b \div c )</td>
<td>float</td>
<td>200.0</td>
<td>will not compile</td>
</tr>
<tr>
<td>2. (2 pts) ( b + 6 % b )</td>
<td>int</td>
<td>2</td>
<td>will not compile</td>
</tr>
<tr>
<td>3. (2 pts) ( a \div d )</td>
<td></td>
<td></td>
<td>will not compile</td>
</tr>
<tr>
<td>4. (2 pts) ( \text{math.ceil}(c) + 1 )</td>
<td>float</td>
<td>13.1</td>
<td>will not compile</td>
</tr>
<tr>
<td>5. (2 pts) ( d \times 0 )</td>
<td>str</td>
<td>&quot;</td>
<td>will not compile</td>
</tr>
<tr>
<td>6. (2 pts) ( \text{str}(d) )</td>
<td></td>
<td>'hello'</td>
<td>will not compile</td>
</tr>
<tr>
<td>7. (2 pts) ( a &gt; b ) and ( a \leq c )</td>
<td>bool</td>
<td>False</td>
<td>will not compile</td>
</tr>
<tr>
<td>8. (2 pts) ( b ) in ( \text{range}(a) )</td>
<td>bool</td>
<td>True</td>
<td>will not compile</td>
</tr>
<tr>
<td>9. (2 pts) ( b ) in ( \text{range}(-1,13,-2) )</td>
<td>bool</td>
<td>False</td>
<td>will not compile</td>
</tr>
</tbody>
</table>

10. (5pts) The expressions in 8. and 9. are actually incomplete because the for statement is not shown. Circle your answer:

   True \[ \text{False} \]

   Both are simply bool expressions and could be written as the test in an if or elif or while:

   \[
   \begin{align*}
   \text{if } b \text{ in range}(a): \\
   \text{functionCallONE()} \\
   \text{elif } b \text{ in range}(-1,13,-2): \\
   \text{functionCallTWO} \\
   \text{while } b \text{ in range}(a): \\
   \text{functionCallTHREE} \\
   \end{align*}
   \]

   Either could also be part of a for statement but then the \text{in} would NOT be the inclusion operator.
11. (6 pts) What is the output when these statements are executed?

```python
x = 7
while x in range(1, 12, 3):
    if x // 3 != 0:
        print x,
        x -= 4
print x
```

Write the output here

```
7 3
```

12. (10 pts) Draw arrows to show which values the variables refer to after the statements are executed. Show the values numbers in RAM also. (It doesn't matter where the are in the program.)

1. \( x = 3 \)  
2. \( y = 4 \)  
3. \( y += x \)  
4. \( x = y \)

Draw the arrow showing where \( x \) refers to after line 1.

Write the values that will be in RAM

```
3 4 7
```

Draw the arrow showing where \( y \) refers after line 2.

Draw the arrow showing where \( y \) refers after line 3.

Draw the arrow showing where \( x \) refers after line 4.
None shown but note that there will be only ONE value answer and you must CIRCLE your answer

a) value > 23 and value < 77 and value > -13 and value < -2
b) value >= 23 and value < 77 and value > -13 and value < -2
c) 23 < value < 77 and -13 value < -2
d) 23 < value < 77 or -13 value < -2
e) -13 <= -2 < value 23 < 77
f) value in [-13,-2] or [23,77]
g) none of the above

This will compile:

```python
floatVar = float( "one hundred and two point seven" )
```

True False  Circle the letter of your one answer

(True False)

(it WILL compile but it will raise a runtime exception because the string cannot be cast to a float)
13. (5 pts)
How many programmers does it take to call a function?
2

What are their names?
the caller and the one who designs or builds or creates or writes the function def.

Which can break the subproblem into more subparts? How?
ONLY the one who designer or builds ot creates or write the function def.

Then how many programmers are involved?
The original 2 + however many function calls need to be made to solve the subproblems.

14. (5 pts) We have calculated the type of the quark – it's your job to write code to display it.
We can only test for exactly four kinds of quarks: top quark, bottom quark, up quark and down quark.
Display the type of quark.
The question for this problem is snot to write the code – it is to answer this question:
Which problem form is writing the type of quark on the screen?

one of many, one must happen
the answer is NOT if elif elfi else

15. (5 pts) The run time system does what when it first encounters these lines of code (what happens during the compillation phase)?

```python
def meme(prammy, sammy):
    return 2 ** (prammy + sammy)
```

places the name `meme` in the global namespace (G)
places the names `prammy` and `sammy` in the local namespace (L) associated with the name `meme`.
(It also compiles the code stores it in RAM and makes the name `meme` refer to that code)
16. (15 pts) There is a built-in function you have never used before named oblatt.

Here is a session showing the help for that function.

```
IDLE 2.7.1
>>> help(oblatt)
Help on built-in function oblatt in module __builtin__:
oblatt(...)
oblatt(object, object) -> float, float

Return 1st param raised to the power of the 2nd param and the 2nd param raised to the power of the 1st param (two value are returned)
```

Obviously you cannot define this function because it is built in.

Your job is to complete the following function by writing the needed parameters and python statements that do what the docstring for `zeroIfEqualOrSumOfPowers` says.

Even though it's not the best, your function MUST use the `oblatt` built-in function.

You do not need to show any testing for your function.

Do NOT write a main function (obviously).

Do NOT write a complete program (obviously).

Write ONLY the definition for `zeroIfEqualOrSumOfPowers` (don't forget to the parameters!)

```python
def zeroIfEqualOrSumOfPowers(first, second):
    ''' returns 0 if both parameters are the same value
    otherwise, returns the larger param raised to the power of the smaller plus the smaller param raised to the power of the larger
    '''
    if first == second
        return 0
    else
        pow1, pow2 = oblatt(first, second)
        return pow1 + pow2
```
17. (20 pts) Show that you can break a problem into its parts by writing only the main function.

Write ONLY variable definitions and function calls – we will hire professional python programmers to write the definitions for all the functions you need written. (They have to be able to know what to write based **only** on your good names for the functions.)

Do NOT write any function `def`s except `main`.

Do NOT write a complete program, just define `main`.

Whenever you need to call a function to solve a subproblem, choose a good name for that function and call it. We will hire someone to define it for you.

Yes, you should have local variables and use them correctly to solve the problem.

When the other programmers have finished writing the definitions of the functions you called in `main`, your `main` function should solve the following problem:

Do these things until the user enters the same age as yours.

Display sixteen hexagons using the letter Q.

Ask the user for their age and their middle initial
(assume everyone has a middle initial)

Display pairs of hexagons, the first one of the pair using a character you like and the second of the pair using the user's initial
– one of these hexagons pairs for each year they've been alive.

You may NOT write any calls to `raw_input` in `main` (even if you need to get only one piece of data, assume that the function you call will do more than just that – like input validation or something).

Assume any global constants you need are already defined – just use good names and style for them.

```python
def main( ):
    while True:
        for i in range( 16 ):
            drawnHexagon( 'Q' )
        age, initial = getAgeAndMiddleInitial()
        if age == MY_AGE:
            break
        for n in range( age ):
            drawnHexagon( MY_FAVOURITE_CHARACTER )
            drawnHexagon( initial )
```
Even better would be:

```python
for n in range(age):
    drawnHexagonPair(MY_FAVOURITE_CHARACTER, initial)
```