1. Analyze the code below and identify any problems or issues that might exist. Offer suggestions on how to re-engineer the code to prevent these errors from occurring and/or rewrite the code so that it functions correctly.

```python
rate = str(input("How much do you make per hour? '"))
hours = input("How many hours did you work this week? ")

if hours < 40:
    pay = rate * hours
if hours > 40
    pay = rate * 40
    ot_pay = (hours-40) ** (rate*1.5)

print ("Your total pay is, pay + ot_pay")
```

2. Given the following program:
a = 5
b = 6
c = 20
d = 24

if a < b and b * 2 < c:
    print ("Python Case 1")
    print ("A", \t, "B", \t, "C")
    if a * 2 == c:
        print (a*2, \t, a*2, \t, a*2)
    elif a * 3 == c:
        print (a*3, \t, a*3, \t, a*3)
    elif a * 4 == c:
        print (a*4, \t, a*4, \t, a*4)
    else:
        print ("?", \t, "?", \t, "?")
else:
    print ("Python Case 2")
    print ("a", \t, "b", \t, "c")
    if b * 2 == d:
        print (b*2, \t, b*2, \t, b*2)
    elif b * 3 == d:
        print (b*3, \t, b*3, \t, b*3)
    elif b * 4 == d:
        print (b*4, \t, b*4, \t, b*4)
    else:
        print ("?", \t, "?", \t, "?")

What will be printed to the screen when the program is run?

3. Trace the output of the following programs:
```python
x = 0

while x < 10:
    print (x)
    x += 2

x = 0
y = 10

while x < y:
    print (x, y)
    x += 1
    y -= 1

x = 5
y = 10

while x > y:
    print ("Looping!")
    x += 1

name = "Python"

while len(name) < 10:
    print (name)
```
4. Part 1: Ask the user to enter a positive integer. Only accept positive integers – if the user supplies a negative integer you should re-prompt them.

Part 2: Next, extend your program to print all numbers between 0 and the number the user supplied as follows:

Enter a positive integer: 5

0 1 2 3 4 5
5. Write a “calculator” program that asks the user for two numbers as well as an “operation code” (“a” for add, “s” for subtract, “d” for divide or “m” for multiply). Using the information provided perform the specified operation and print the result. Here is a sample running of the program:

Number 1: 2.0
Number 2: 3.0
Operation (a/s/d/m): add
Invalid operation! Try again.
Operation (a/s/d/m): a
2.0 + 3.0 = 5.0

Note that you cannot assume that the user will enter a valid operation code (i.e. they could type in the string “multiply” instead of the string “m”). *In this case you will need to present the user with some kind of error* (i.e. “Sorry, that’s not a valid operation code”) *and re-prompt them*. However, you can assume that the user will input valid floating-point numbers when prompted.

Also note that dividing a number by 0 will result in a runtime error. Prevent this from happening in your program by providing special output in this case (i.e. 5.0 / 0.0 = undefined)
6. A small college has asked you to write a program for their admissions department to help them determine if a student should be accepted into their school. Write a program that uses the following criteria to determine whether a given applicant should be admitted:

- Combined SAT score of 1600 or more
- A high school GPA of 3.0 or higher
- At least 3 extracurricular activities

However, this particular school places a heavy emphasis on extracurricular activities, so students with 5 or more activities only need a 1400 combined score on their SAT and a GPA of 2.8. Comment your code as necessary. You can assume that the user will enter floating-point values.

Here is a sample running of your program. Note that you should ask the user if they want to repeat the process for additional students when you are finished.

Student name: Craig
Combined SAT Score: 1800
High school GPA: 3.2
# of extracurricular activities: 3
Craig should be admitted!
Another student? yes

Student name: John
Combined SAT Score: 1500
High school GPA: 3.1
# of extracurricular activities: 7
John should be admitted!
Another student? yes

Student name: Chris
Combined SAT Score: 1300
High school GPA: 2.9
# of extracurricular activities: 8
Chris should not be admitted.
Another student? no
7. Write a program that lets the user figure out how many items they can purchase at a local coffee shop. Begin by asking the user to enter in amount of money as a float. Then ask the user to select a product from a pre-determined list. Figure out how many items the user can purchase, noting that the coffee shop does not sell fractional amounts (i.e. you can’t buy half a donut)

How much money do you have?: 10.00

What would you like to buy?
Donut (d) – 1.50
Coffee (c) – 1.00
Bagel (b) – 2.50
Scone (s) – 2.75

Enter your choice (d/c/b/s): d

You can purchase 6 donuts with $ 10.0

Note that you cannot assume that the user will enter a valid product (i.e. they could type in the string “donut” instead of the string “d”). In this case you will need to present the user with some kind of error (i.e. “Sorry, that’s not a valid product”) – you do not need to re-prompt them (you can just end the program). Also, you can assume that the user will input valid floating-point numbers when prompted.
8. Write a “bird watcher” program that prompts the user to enter in the number of birds that they spotted on a given day. The user can enter in values for an unlimited # of days. The user can elect to stop entering data by indicating that they saw 0 birds on a particular day. The user will always supply positive integers for the purpose of this program.

When you’re finished gathering data you should display the number of birds seen, the number of days that the user watched birds as well as the average number of birds seen per day. Here’s a sample running of the program:

```
How many birds did you see today? Enter 0 to end: 5
How many birds did you see today? Enter 0 to end: 10
How many birds did you see today? Enter 0 to end: 15
How many birds did you see today? Enter 0 to end: 0

You watched birds for 3 days
In total you saw 30 birds
That's an average of 10.0 birds per day
```