Lab 6

Questions

Unless otherwise indicated, the code samples below do not contain any errors. If you experience an error when you run the code, this is due to a copy/paste error.

1. Predict what the following code snippet will print (explain your reasoning). Then run the code and test your prediction. If your prediction was wrong, explain why. Ask your instructor if you are unsure.

```
def function():
  number = 5
  print(number)

number = 10
function()
print(number)
```

Download this code here
If you get stuck, check out slides 28-39 of this lecture

2. The following code produces an error. Explain what the error is and how to resolve it.

```
PI = 3.14

def calc_area(radius):
  result = PI * radius**2
  return result

r = float(input("Enter the radius:  "))
calc_area(r)

print("The area is",result)
```

Download this code here
If you get stuck, check out slides 50-62 of this lecture

Writing Programs

Don't worry about the style and readability guidelines for these programs.

1. Use the functions and variables defined in the `math` module to write the following program: Have the user enter a vector in the form of a magnitude and an angle (in degrees) ($< r, \phi >$), and then convert this polar vector to cartesian coordinates $< x, y >$ via: $x = r\cos\phi$, $y = r\sin\phi$. Remember that the trig functions in the `math` module expect the argument to be in radians. You can use other functions in this module to perform that conversion.

   Tip: To view a summary of the math module, import the math module and then run help(math)

2. Write three functions: `pow1(x,y)`, `pow2(x,y)`, and `pow3(x,y)` to perform the operation x^y three different ways:

   (a) `pow1(x,y)`: Simply return `x**y`

   (b) `pow2(x,y)`: Return `math.pow(x,y)`

   (c) `pow3(x,y)`: Write your own algorithm which doesn't use the `**` operator or any Python modules. *Hint: A number raised to a power is nothing more than repeated multiplication.* 2^4 *is just* $2\cdot2\cdot2\cdot2$, $5^6 = 5\cdot5\cdot5\cdot5\cdot5\cdot5$, *etc.*

   Now use the `time` module to rank the three functions from fastest to slowest. *To get a reliable estimate for the time of a process, it is best to measure the time it takes for a process to repeat many thousands of times, and then divide the time by the number of repetitions.*

3. Write a program which prompts the user to enter the length and width of two rectangles (so four numbers in total) and then tell the user if their areas are equal.

4. In this problem we will write a program to investigate the Collatz problem. Write a function named `collatz(number)` that has one parameter (`number`). The function should do the following:

   (a) If `number` is *even*: calculate the integer quotient* of `number/2`, print this value, and return it.

   (b) If `number` is *odd*: print and return the value 3 * number + 1.

   Once you have written this function, write a program which asks the user to enter any number (make sure you convert it to an integer). Pass that number as an argument to `collatz()`, then send the result of this operation back again as an argument to `collatz()`, then call `collatz()` again with *this* result. If you continue in this way long enough, then `collatz()` will eventually evaluate to 1. Amazingly, this seems to be true for any starting integer number, and no one is sure why! Write a loop which continues this process until you reach the number 1.

*By integer quotient, I simply mean the integral part of the result of the division operation (rounding down). For example, if `number` is 23, then $23/2 = 11.5$, so we would return 11.