Practice Exercises for Logic and Conditionals

Solve each of the practice exercises below. Each problem includes two CodeSkulptor3 links: one for a template that you should use as a starting point for your solution and one to our solution to the exercise,.

1. Write a Python function is\_even that takes as input the parameter number (an integer) and returns if number is even and False if number is odd. Hint: Apply the remainder operator to n (i.e.,% 2) and compare to zero. [Even template](http://py3.codeskulptor.org/#exercises3_cond_even_template.py) --- [Even solution](http://py3.codeskulptor.org/#exercises3_cond_even_solution.py)
2. Write a Python function \color{red}{\verb|is\_cool|}is\_cool that takes as input the string \color{red}{\verb|name|}name and returns \color{red}{\verb|True|}True if \color{red}{\verb|name|}name is either \color{red}{\verb|"Joe"|}"Joe", \color{red}{\verb|"John"|}"John" or \color{red}{\verb|"Stephen"|}"Stephen" and returns \color{red}{\verb|False|}False otherwise. (Let's see if Scott manages to catch this. ☺ ) [Cool template](http://py3.codeskulptor.org/#exercises3_cond_cool_template.py) --- [Cool solution](http://py3.codeskulptor.org/#exercises3_cond_cool_solution.py)
3. Write a Python function \color{red}{\verb|is\_lunchtime|}is\_lunchtime that takes as input the parameters \color{red}{\verb|hour|}hour (an integer in the range [1, 12][1,12]) and \color{red}{\verb|is\_am|}is\_am (a Boolean “flag” that represents whether the hour is before noon). The function should return \color{red}{\verb|True|}True when the input corresponds to 11am or 12pm (noon) and \color{red}{\verb|False|}False otherwise. If the problem specification is unclear, look at the test cases in the provided template. Our solution does not use conditional statements. [Lunchtime template](http://py3.codeskulptor.org/#exercises3_cond_lunchtime_template.py) --- [Lunchtime solution](http://py3.codeskulptor.org/#exercises3_cond_lunchtime_solution.py)
4. Write a Python function \color{red}{\verb|is\_leap\_year|}is\_leap\_year that take as input the parameter \color{red}{\verb|year|}year and returns \color{red}{\verb|True|}True if \color{red}{\verb|year|}year (an integer) is a leap year according to the Gregorian calendar and \color{red}{\verb|False|}False otherwise. The Wikipedia entry for [leap years](http://en.wikipedia.org/wiki/Leap_year) contains a simple algorithmic rule for determining whether a year is a leap year. Your main task will be to translate this rule into Python. [Leap year template](http://py3.codeskulptor.org/#exercises3_cond_leapyear_template.py) --- [Leap year solution](http://py3.codeskulptor.org/#exercises3_cond_leapyear_solution.py)
5. Write a Python function \color{red}{\verb|interval\_intersect|}interval\_intersect that takes parameters \color{red}{\verb|a|}a, \color{red}{\verb|b|}b, \color{red}{\verb|c|}c, and \color{red}{\verb|d|}d and returns \color{red}{\verb|True|}True if the intervals [a, b][*a*,*b*] and [c, d][*c*,*d*] intersect and \color{red}{\verb|False|}False otherwise. While this test may seem tricky, the solution is actually very simple and consists of one line of Python code. (You may assume that a \leq b *a*≤*b* and c \leq d*c*≤*d*.) [Interval intersect template](http://py3.codeskulptor.org/#exercises3_cond_interval_intersect_template.py) --- [Interval intersect solution](http://py3.codeskulptor.org/#exercises3_cond_interval_intersect_solution.py)
6. Write a Python function \color{red}{\verb|name\_and\_age|}name\_and\_age that take as input the parameters \color{red}{\verb|name|}name (a string) and \color{red}{\verb|age|}age (a number) and returns a string of the form \color{red}{\verb|"% is % years old."|}"% is % years old." where the percents are the string forms of \color{red}{\verb|name|}name and \color{red}{\verb|age|}age. The function should include an error check for the case when \color{red}{\verb|age|}age is less than zero. In this case, the function should return the string \color{red}{\verb|"Error: Invalid age"|}"Error: Invalid age". [Name and age template](http://py3.codeskulptor.org/#exercises3_cond_nameage_template.py) --- [Name and age solution](http://py3.codeskulptor.org/#exercises3_cond_nameage_solution.py)
7. Write a Python function \color{red}{\verb|print\_digits|}print\_digits that takes an integer \color{red}{\verb|number|}number in the range [0,100)[0,100) and prints the message \color{red}{\verb|"The tens digit is %, and the ones digit is %."|}"The tens digit is %, and the ones digit is %." where the percents should be replaced with the appropriate values. The function should include an error check for the case when \color{red}{\verb|number|}number is negative or greater than or equal to 100100. In those cases, the function should instead print \color{red}{\verb|"Error: Input is not a two-digit number."|}"Error: Input is not a two-digit number.". [Print digits template](http://py3.codeskulptor.org/#exercises3_cond_print_digits_template.py) --- [Print digits solution](http://py3.codeskulptor.org/#exercises3_cond_print_digits_solution.py)
8. Write a Python function \color{red}{\verb|name\_lookup|}name\_lookup that takes a string \color{red}{\verb|first\_name|}first\_name that corresponds to one of (\color{red}{\verb|"Joe"|}"Joe", \color{red}{\verb|"Scott"|}"Scott", \color{red}{\verb|"John"|}"John" or \color{red}{\verb|"Stephen"|}"Stephen") and then returns their corresponding last name (\color{red}{\verb|"Warren"|}"Warren", \color{red}{\verb|"Rixner"|}"Rixner", \color{red}{\verb|"Greiner"|}"Greiner" or \color{red}{\verb|"Wong"|}"Wong"). If \color{red}{\verb|first\_name|}first\_name doesn't match any of those strings, return the string \color{red}{\verb|"Error: Not an instructor"|}"Error: Not an instructor". [Name lookup template](http://py3.codeskulptor.org/#exercises3_cond_name_lookup_template.py) --- [Name lookup solution](http://py3.codeskulptor.org/#exercises3_cond_name_lookup_solution.py)
9. [Pig Latin](http://en.wikipedia.org/wiki/Pig_Latin) is a language game that involves altering words via a simple set of rules. Write a Python function \color{red}{\verb|pig\_latin|}pig\_latin that takes a string \color{red}{\verb|word|}word and applies the following rules to generate a new word in Pig Latin. If the first letter in \color{red}{\verb|word|}word is a consonant, append the consonant plus \color{red}{\verb|"ay"|}"ay" to the end of the remainder of the word. For example, \color{red}{\verb|pig\_latin("pig")|}pig\_latin("pig") would return \color{red}{\verb|"igpay"|}"igpay". If the first letter in \color{red}{\verb|word|}word is a vowel, append \color{red}{\verb|"way"|}"way" to the end of the word. For example, \color{red}{\verb|pig\_latin("owl")|}pig\_latin("owl") returns \color{red}{\verb|"owlway"|}"owlway". You can assume that \color{red}{\verb|word|}word is in lower case. The provided template includes code to extract the first letter and the rest of \color{red}{\verb|word|}word in Python. Note that, in full Pig Latin, the leading consonant cluster is moved to the end of the word. However, we don't know enough Python to implement full Pig Latin just yet. [Pig Latin template](http://py3.codeskulptor.org/#exercises3_cond_pig_latin_template.py) --- [Pig Latin solution](http://py3.codeskulptor.org/#exercises3_cond_pig_latin_solution.py)
10. **Challenge:** Given numbers a*a*, b*b*, and c*c*, the [quadratic equation](http://en.wikipedia.org/wiki/Quadratic_equation) a x^2 + b x + c = 0*ax*2+*bx*+*c*=0 can have zero, one or two real solutions (i.e; values for x*x* that satisfy the equation). The quadratic formula x = \frac{-b \pm \sqrt{b^2 - 4 a c}}{2 a}*x*=2*a*−*b*±*b*2−4*ac*​​can be used to compute these solutions. The expression b^2 - 4 a c*b*2−4*ac* is the *discriminant* associated with the equation. If the discriminant is positive, the equation has two solutions. If the discriminant is zero, the equation has one solution. Finally, if the discriminant is negative, the equation has no solutions. Write a Python function \color{red}{\verb|smaller\_root|}smaller\_root that takes an input the numbers \color{red}{\verb|a|}a, \color{red}{\verb|b|}b and \color{red}{\verb|c|}c and returns the smaller solution to this equation if one exists. If the equation has no real solution, print the message \color{red}{\verb|"Error: No real solutions"|}"Error: No real solutions" and simply return. Note that, in this case, the function will actually return the special Python value \color{red}{\verb|None|}None. [Quadratic root template](http://py3.codeskulptor.org/#exercises3_cond_quadratic_root_template.py) --- [Quadratic root solution](http://py3.codeskulptor.org/#exercises3_cond_quadratic_root_solution.py)