1. Create a function named**calculate\_discount(price, discount\_percent)** that calculates the final price after applying a discount. The function should take the original price**(price)**and the discount percentage **(discount\_percent)** as parameters. If the discount is **20%** or higher, apply the discount; otherwise, return the original price.
2. Using the**calculate\_discount** function, prompt the user to enter the original price of an item and the discount percentage. Print the final price after applying the discount, or if no discount was applied, print the original price.

Possible answers:

|  |
| --- |
| # module calculate\_store\_discount() is where I am having issues. I also tried linking original\_price() |
|  | # to calculate\_store\_discount() so I wouldn't have to have the user re-enter. The in\_store\_discount() |
|  | # and calculate\_store\_discount() are going to be 'duplicated' with variables relating to the |
|  | # coupon inputs and calculation ( calculate\_coupon(), coupon\_discount() ). Another module is going to |
|  | # be created to calculate the final price after I figure out what my hang up is with the math. |
|  |  |
|  |  |
|  | def main(): |
|  |  |
|  | welcome = welcome\_message() |
|  | store\_discount = in\_store\_discount() |
|  |  |
|  |  |
|  | def welcome\_message(): |
|  |  |
|  | print "Hello! Calculate the final price of an item with an in-store sale and " |
|  | print "a coupon if they are available. First input the original price (ex. 1.00) " |
|  | print " follwed by the in-store discount (ex. .20), and the coupon amount. " |
|  |  |
|  |  |
|  |  |
|  |  |
|  | def original\_item\_price(): |
|  | original\_price = 0.0 |
|  | original\_price = float(input("What is the original price of the item?")) |
|  |  |
|  | return original\_price |
|  |  |
|  |  |
|  |  |
|  |  |
|  | def in\_store\_discount(): |
|  | store\_discount\_choice = 0.0 |
|  | original\_price = 0.0 |
|  | store\_discount\_percentage = 0.0 |
|  |  |
|  | orig\_price = original\_item\_price() |
|  |  |
|  | store\_discount\_choice = int(input("Is there an in-store discount? Enter 1 for Yes, 2 for No.")) |
|  |  |
|  | if store\_discount\_choice == 1: |
|  | calculate\_store\_discount() |
|  |  |
|  | elif store\_discount\_choice == 2: |
|  | original\_price\_remains() |
|  |  |
|  |  |
|  |  |
|  |  |
|  | def calculate\_store\_discount(): |
|  | original\_price = 0.0 |
|  | in\_store\_discount = 0.0 |
|  | discount\_amount = 0.0 |
|  | final\_store\_discount = 0.0 |
|  |  |
|  | original\_price = int(input("Re-Enter the original price.")) |
|  |  |
|  | in\_store\_discount = int(input("What is the in\_store discount? (ex. .20)")) |
|  |  |
|  | discount\_amount = original\_price \* in\_store\_discount |
|  | final\_store\_discount = original\_price - discount\_amount |
|  |  |
|  | print "The in\_store savings amount is ", discount\_amount |
|  | print "Your new price is ", final\_store\_discount |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | main() |

P

|  |
| --- |
| # module calculate\_store\_discount() is where I am having issues. I also tried linking original\_price() |
|  | # to calculate\_store\_discount() so I wouldn't have to have the user re-enter. The in\_store\_discount() |
|  | # and calculate\_store\_discount() are going to be 'duplicated' with variables relating to the |
|  | # coupon inputs and calculation ( calculate\_coupon(), coupon\_discount() ). Another module is going to |
|  | # be created to calculate the final price after I figure out what my hang up is with the math. |
|  |  |
|  | # Okay, so this is your `main` function. There's nothing special about it being |
|  | # called `main`, it's just done that way by convention since the beginning of |
|  | # time. It is your main "driver" for the program as a whole, which you can |
|  | # usually just assume, but you can be sure of it by seeing that at the bottom |
|  | # you have a line that consists only of `main()`. This is syntax for a function |
|  | # call. You'll also see something like the following in a lot of Python |
|  | # scripts: |
|  | # if \_\_name\_\_ == '\_\_main\_\_': |
|  | # main() |
|  | # You can find out more here: |
|  | # http://stackoverflow.com/questions/419163/what-does-if-name-main-do |
|  |  |
|  | # Anyway, the Python interpreter (which is called "Python", but it's distinct |
|  | # from the language "Python" itself) will read your file and execute everything |
|  | # line by line, in order. This line defines a function called "main" and |
|  | # then actually skips over the rest of the definition, which is denoted by |
|  | # the indentation. So now Python knows there is a function called "main" |
|  | # accessible later in your program that does something (it doesn't know what |
|  | # just yet). |
|  | # Skip over to the next definitions until you get to the last line of the |
|  | # program, which calls `main`. |
|  | def main(): |
|  | # So now we're back here from the last line of the file. We're creating a |
|  | # local variable called `welcome`, calling `welcome\_message`, and assigning |
|  | # the return value of `welcome\_message` to the local variable `welcome`. |
|  | welcome = welcome\_message() # Once Python reads `welcome\_message()`, it |
|  | # jumps to the line immediately after |
|  | # `def welcome\_message():`. Go there now. |
|  | # After we get back from calling `welcome\_message`, we'll be here. We're |
|  | # doing something similar - creating a local variable `store\_discount`, |
|  | # calling the `in\_store\_discount` function, and assigning the return value |
|  | # of it to `store\_discount`. |
|  | store\_discount = in\_store\_discount() # Now jump to the line immediately |
|  | # after `def in\_store\_discount()`. |
|  |  |
|  | # We've made it all the way back here! Now we're done in this function |
|  | # and go back to the caller, which is the last line of the file. |
|  |  |
|  | # Immediately after line 28 (`def main():`), we just skip all the way down here |
|  | # to the next definition. It defines a function called `welcome\_message`. |
|  | def welcome\_message(): |
|  | # Now we're here. We're going to call `print "Hello! ..." and print out the |
|  | # desired messages line by line. Great! |
|  | print "Hello! Calculate the final price of an item with an in-store sale and " |
|  | print "a coupon if they are available. First input the original price (ex. 1.00) " |
|  | print " follwed by the in-store discount (ex. .20), and the coupon amount. " |
|  | # Now we're at the end of `welcome\_message`. We will now jump back to line |
|  | # 32 and assign the return value of this function to `main`s local |
|  | # `welcome` variable (declared on line 32). However, this function doesn't |
|  | # actually have a return value (technically all functions do - if one isn't |
|  | # specified then functions return a special value called `None`). This |
|  | # basically means that line 32 can just be `welcome\_message()` rather than |
|  | # `welcome = welcome\_message()`. The variable `welcome` is assigned `None` |
|  | # and never used, so we don't need it. |
|  | # Now jump back to line 35. |
|  |  |
|  | # Next we skip here, define `original\_item\_price`. |
|  | def original\_item\_price(): |
|  | # Now we're here from `in\_store\_discount`. We're creating a new local |
|  | # variable called `original\_price` and assigning the value `0.0` to it. |
|  | original\_price = 0.0 |
|  | # Now we're calling the function `input` which takes a string, prints it |
|  | # to the screen, and accepts input from the user. Then we pass whatever the |
|  | # user input to the function `float` which takes whatever and tries to |
|  | # convert it to a floating point number. Then whatever `float` returns will |
|  | # be assigned to `original\_price`. |
|  | original\_price = float(input("What is the original price of the item?")) |
|  |  |
|  | # Now we return `original\_price`. This means that the value of |
|  | # `original\_price` will be available to whatever called |
|  | # `original\_item\_price`. Great! |
|  | return original\_price |
|  | # Now we return back to the caller in `in\_store\_discount`. |
|  |  |
|  | # Ditto, `in\_store\_discount`. |
|  | def in\_store\_discount(): |
|  | # Now we're here from `main`. We're creating three local variables, |
|  | # `store\_discount\_choice`, `original\_price`, and |
|  | # `store\_discount\_percentage`, and assigning `0.0` to each of them. |
|  | store\_discount\_choice = 0.0 |
|  | original\_price = 0.0 |
|  | store\_discount\_percentage = 0.0 |
|  |  |
|  | # Now we're going to call `original\_item\_price()` and assign the value |
|  | # returned by that function to `orig\_price`. |
|  | # NOTE: The variable here is called `orig\_price`, which is a new local |
|  | # variable being created right now. Python will just create a new variable |
|  | # if you use one that doesn't exist, which can lead to confusing |
|  | # situations. |
|  | orig\_price = original\_item\_price() # Jump to the line immediately after |
|  | # `def original\_item\_price():`. |
|  | # We've just come back from `original\_item\_price` and assign the return |
|  | # value to `orig\_price`. What this means in human-terms is that the user |
|  | # has been asked what the original price of the item was, they put in a |
|  | # number (hopefully), and now the local variable `orig\_price` knows that |
|  | # number. |
|  |  |
|  | # Now we're prompting the user for input again in the same way as in |
|  | # `original\_item\_price()`. The only difference is what we do with the value |
|  | # that `input` returns - in this case we convert it to an integer with the |
|  | # `int` function rather than converting it to a floating point number with |
|  | # `float`. Great. Then we assign that value to `store\_discount\_choice`. |
|  | store\_discount\_choice = int(input("Is there an in-store discount? Enter 1 for Yes, 2 for No.")) |
|  |  |
|  | # Now we compare `store\_discount\_choice` with the integer `1`. If they are |
|  | # equal, we call `calculate\_store\_discount()`. |
|  | # This is a "branch", which means that there are now two different code |
|  | # paths that can be followed, depending upon the value of |
|  | # `store\_discount\_choice`. You can go to either function definition below. |
|  | # I'll assume both happen (which is impossible in a single run, but we'll |
|  | # assume the user runs the program twice). |
|  | if store\_discount\_choice == 1: |
|  | calculate\_store\_discount() # If they entered "1", jump to the line |
|  | # immediately after |
|  | # `def calculate\_store\_discount():` |
|  | # We're now done with the call, which will end up being the last thing this |
|  | # function does. We would now jump back to the caller of this function, |
|  | # which is line 39 in the function `main`. |
|  |  |
|  | # Otherwise, if `store\_discount\_choice` equals `2`, we call |
|  | # `original\_price\_remains` (which doesn't actually exist). |
|  | elif store\_discount\_choice == 2: |
|  | original\_price\_remains() # If this function existed, you would jump to |
|  | # it's definition. |
|  |  |
|  | # Finally, define `calculate\_store\_discount`. |
|  | def calculate\_store\_discount(): |
|  | # If the user entered "1", we'll be here. We create four local variables, |
|  | # `original\_price`, `in\_store\_discount`, `discount\_amount`, and |
|  | # `final\_store\_discount`. We assign the value `0.0` to all of them. |
|  | original\_price = 0.0 |
|  | in\_store\_discount = 0.0 |
|  | discount\_amount = 0.0 |
|  | final\_store\_discount = 0.0 |
|  |  |
|  | # Here we are asking the user to re-enter the original price. We actually |
|  | # don't need to do this, because we already asked! In `in\_store\_discount`, |
|  | # we have a variable with the value they input. We can use this for |
|  | # whatever we want. The most common (and best) way to have functions work |
|  | # with each other is to pass values to functions using parameters. You'll |
|  | # need to do this if you want to have multiple functions doing the work |
|  | # (which you do!). You're already using this functionality in your calls to |
|  | # `input()`. You're passing a value (in all these cases, a string) to |
|  | # `input()`, which returns a value (which is whatever the user entered), |
|  | # which is then passed to the function `int()` which THEN returns a value |
|  | # which is assigned to the variable `original\_price`, in this case. You can |
|  | # use this for whatever, like passing it to functions. |
|  | original\_price = int(input("Re-Enter the original price.")) |
|  |  |
|  | # Now we do a similar thing to what you did above with `original\_price`. |
|  | # One important thing to note is that, based on the usage, you're expecting |
|  | # a floating point number (e.g., .2), but you're passing whatever the user |
|  | # inputs to the `int` function, which converts things to integers (whole |
|  | # numbers). What this means is that it will take something that looks like |
|  | # a number and convert it to a whole number based on a few rules. The most |
|  | # relevant one for your situation right now is that floating point to |
|  | # integer conversions truncate. So 0.2 turns to 0, 0.8 turns to 0, 1.4 |
|  | # turns to 1, 1234.56789 turns to 1234 - it will simply cut off everything |
|  | # after the decimal point and doesn't bother rounding. 0.999999 -> 0. |
|  | # This is the main issue you're experiencing. By changing the call from |
|  | # `int` to `float` (i.e., `float(input("..."))`), you'll get the output you |
|  | # expect. |
|  | in\_store\_discount = int(input("What is the in\_store discount? (ex. .20)")) |
|  |  |
|  | # We now calculate some values and assign them to variables. These work as |
|  | # expected. |
|  | discount\_amount = original\_price \* in\_store\_discount |
|  | final\_store\_discount = original\_price - discount\_amount |
|  |  |
|  | # Finally, print out the information to the user. |
|  | print "The in\_store savings amount is ", discount\_amount |
|  | print "Your new price is ", final\_store\_discount |
|  |  |
|  | # Again, here we implicitly return `None` and would jump back to the caller |
|  | # (line 119). |
|  |  |
|  | # Now we're here. We've defined five functions, `main`, `welcome\_message`, |
|  | # `original\_item\_price`, `in\_store\_discount`, and `calculate\_store\_discount`. |
|  | # The next line, `main()` will call the function `main()`. So execution now |
|  | # jumps back up to the line immediately after `def main():`. |
|  | main() |
|  |  |
|  | # Finally we've finished main, we reach the end of the file, and the Python |
|  | # intepreter says "I'm done!" and the program is over. |

ossible answer 2: