3.3 Nested Branches

When a decision statement is contained inside the branch of another decision statement, the statements are *nested*.

It is often necessary to include an `if` statement inside another. Such an arrangement is called a *nested* set of statements.

Here is a typical example: In the United States, different tax rates are used depending on the taxpayer's marital status. There are different tax schedules for single and for married taxpayers. Married taxpayers add their income together and pay taxes on the total. Table 3 gives the tax rate computations, using a simplification of the schedules in effect for the 2008 tax year. A different tax rate applies to each "bracket". In this schedule, the income in the first bracket is taxed at 10 percent, and the income in the second bracket is taxed at 25 percent. The income limits for each bracket depend on the marital status.

Table 3 Federal Tax Rate Schedule

If your status is Single and if the taxable income is	the tax is	of the amount over
at most $32,000	10%	$0
over $32,000	$3,200 + 25%	$32,000
If your status is Married and if the taxable income is	**the tax is**	**of the amount over**
at most $64,000	10%	$0
over $64,000	$6,400 + 25%	$64,000

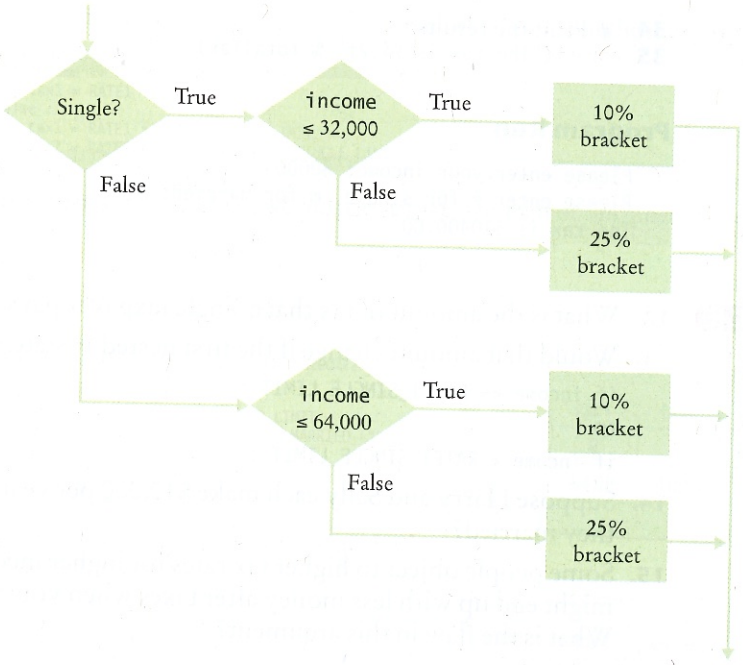
Nested decisions are required for problems that have multiple levels of decision making.

Now compute the taxes due, given a marital status and an income figure. The key point is that there are two *levels* of decision making. First, you must branch on the marital status. Then, for each marital status, you must have another branch on income level.

The two-level decision process is reflected in two levels of `if` statements in the program at the end of this section. (See Figure 3 for a flowchart.) In theory, nesting can go deeper than two levels. A three-level decision process (first by state, then by marital status, then by income level) requires three nesting levels.

Computing income taxes requires multiple levels of decisions.

Figure 3
Income Tax Computation



ch03/taxes.py

```
1  ##
2  #  This program computes income taxes, using a simplified tax schedule.
3  #
4
5  # Initialize constant variables for the tax rates and rate limits.
6  RATE1 = 0.10
7  RATE2 = 0.25
8  RATE1_SINGLE_LIMIT = 32000.0
9  RATE1_MARRIED_LIMIT = 64000.0
10
11 # Read income and marital status.
12 income = float(input("Please enter your income: "))
13 maritalStatus = input("Please enter s for single, m for married: ")
14
15 # Compute taxes due.
16 tax1 = 0.0
17 tax2 = 0.0
18
19 if maritalStatus == "s" :
20    if income <= RATE1_SINGLE_LIMIT :
21       tax1 = RATE1 * income
22    else :
23       tax1 = RATE1 * RATE1_SINGLE_LIMIT
24       tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT)
25 else :
26    if income <= RATE1_MARRIED_LIMIT :
27       tax1 = RATE1 * income
28    else :
29       tax1 = RATE1 * RATE1_MARRIED_LIMIT
30       tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT)
31
32 totalTax = tax1 + tax2
33
```

```
34 # Print the results.
35 print("The tax is $%.2f" % totalTax)
```

Program Run

```
Please enter your income: 80000
Please enter s for single, m for married: m
The tax is $10400.00
```

12. What is the amount of tax that a single taxpayer pays on an income of $32,000?
13. Would that amount change if the first nested `if` statement changed from
    `if income <= RATE1_SINGLE_LIMIT :`
    to
    `if income < RATE1_SINGLE_LIMIT :`
14. Suppose Harry and Sally each make $40,000 per year. Would they save taxes if they married?
15. Some people object to higher tax rates for higher incomes, claiming that you might end up with less money after taxes when you get a raise for working hard. What is the flaw in this argument?

Practice It Now you can try these exercises at the end of the chapter: R3.9, P3.20, P3.23.

Programming Tip 3.2

Hand-Tracing

A very useful technique for understanding whether a program works correctly is called *hand-tracing*. You simulate the program's activity on a sheet of paper. You can use this method with pseudocode or Python code.

Get an index card, a cocktail napkin, or whatever sheet of paper is within reach. Make a column for each variable. Have the program code ready. Use a marker, such as a paper clip, to mark the current statement. In your mind, execute statements one at a time. Every time the value of a variable changes, cross out the old value and write the new value below the old one.

Let's trace the `taxes.py` program on page 107 with the inputs from the program run that follows it. In lines 12 and 13, `income` and `maritalStatus` are initialized by input statements.

Hand-tracing helps you understand whether a program works correctly.

```
5 # Initialize constant variables for the tax rates and rate limits.
6 RATE1 = 0.10
7 RATE2 = 0.25
8 RATE1_SINGLE_LIMIT = 32000.0
9 RATE1_MARRIED_LIMIT = 64000.0
10
11 # Read income and marital status.
12 income = float(input("Please enter your income: "))
13 maritalStatus = input("Please enter s for single, m for married: ")
```

tax1	tax2	income	marital status
		80000	m

In lines 16 and 17, `tax1` and `tax2` are initialized to 0.0.

```
16 tax1 = 0.0
17 tax2 = 0.0
```

tax1	tax2	income	marital status
0	0	80000	m

Because maritalStatus is not "s", we move to the else branch of the outer if statement (line 25).

```
19 if maritalStatus == "s" :
20    if income <= RATE1_SINGLE_LIMIT :
21       tax1 = RATE1 * income
22    else :
23       tax1 = RATE1 * RATE1_SINGLE_LIMIT
24       tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT)
25 else :
```

Because income is not <= 64000, we move to the else branch of the inner if statement (line 28).

```
26    if income <= RATE1_MARRIED_LIMIT :
27       tax1 = RATE1 * income
28    else :
29       tax1 = RATE1 * RATE1_MARRIED_LIMIT
30       tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT)
```

The values of tax1 and tax2 are updated.

```
28    else :
29       tax1 = RATE1 * RATE1_MARRIED_LIMIT
30       tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT)
```

tax1	tax2	income	marital status
~~0~~	~~0~~	80000	m
6400	4000		

The sum totalTax is computed and printed. Then the program ends.

```
32 totalTax = tax1 + tax2
   . . .
35 print("The tax is $%.2f" % totalTax)
```

Because the program trace shows the expected output ($10,400), it successfully demonstrated that this test case works correctly.

tax1	tax2	income	marital status	total tax
~~0~~	~~0~~	80000	m	
6400	4000			10400

3.4 Multiple Alternatives

Multiple if statements can be combined to evaluate complex decisions.

In Section 3.1, you saw how to program a two-way branch with an if statement. In many situations, there are more than two cases. In this section, you will see how to implement a decision with multiple alternatives.

For example, consider a program that displays the effect of an earthquake, as measured by the Richter scale (see Table 4).

The 1989 Loma Prieta earthquake that damaged the Bay Bridge in San Francisco and destroyed many buildings measured 7.1 on the Richter scale.

Table 4 Richter Scale

Value	Effect
8	Most structures fall
7	Many buildings destroyed
6	Many buildings considerably damaged, some collapse
4.5	Damage to poorly constructed buildings