ch03/sale.py

```
1  ##
2  #  Compute the discount for a given purchase.
3  #
4
5  # Obtain the original price.
6  originalPrice = float(input("Original price before discount: "))
7
8  # Determine the discount rate.
9  if originalPrice < 128 :
10    discountRate = 0.92
11 else :
12    discountRate = 0.84
13
14 # Compute and print the discount.
15 discountedPrice = discountRate * originalPrice
16 print("Discounted price: %.2f" % discountedPrice)
```

WORKED EXAMPLE 3.1 Extracting the Middle

Problem Statement Your task is to extract a string containing the middle character from a given string. For example, if the string is `"crate"`, the result is the string `"a"`. However, if the string has an even number of letters, extract the middle two characters. If the string is `"crates"`, the result is `"at"`.

Step 1 Decide on the branching condition.

We need to take different actions for strings of odd and even length. Therefore, the condition is

Is the length of the string odd?

In Python, you use the remainder of division by 2 to find out whether a value is even or odd. Then the test becomes

len(string) % 2 == 1?

Step 2 Give pseudocode for the work that needs to be done when the condition is true.

We need to find the position of the middle character. If the length is 5, the position is 2.

c	r	a	t	e
0	1	2	3	4

In general,

position = len(string) / 2 (with the remainder discarded)
result = string[position]

Step 3 Give pseudocode for the work (if any) that needs to be done when the condition is *not* true.

Again, we need to find the position of the middle characters. If the length is 6, the starting position is 2, and the ending position is 3. That is, we would call

result = string[2] + string[3]

c	r	a	t	e	s
0	1	2	3	4	5

In general,

```
position = len(string) / 2 - 1 (with the remainder discarded)
result = string[position] + string[position + 1]
```

Step 4 Double-check relational operators.

Do we really want `len(string) % 2 == 1`? For example, when the length is 5, `5 % 2` is the remainder of the division 5 / 2, which is 1. In general, dividing an odd number by 2 leaves a remainder of 1. Therefore, our condition is correct.

Step 5 Remove duplication.

Here is the statement that we have developed:

```
If len(string) % 2 == 1
   position = len(string) / 2 (with remainder discarded)
   result = string[position]
Else
   position = len(string) / 2 - 1 (with remainder discarded)
   result = string[position] + string[position + 1]
```

The first statement in each branch is almost identical. Could we make them the same? We can, if we adjust the position in the second branch:

```
If len(string) % 2 == 1
   position = len(string) / 2 (with remainder discarded)
   result = string[position]
Else
   position = len(string) / 2 (with remainder discarded)
   result = string[position - 1] + string[position]
```

Now we can move the duplicated computation outside the `if` statement:

```
position = len(string) / 2 (with remainder discarded)
If len(string) % 2 == 1
   result = string[position]
Else
   result = string[position - 1] + string[position]
```

Step 6 Test both branches.

We will use a different set of strings for testing. For an odd-length string, consider `"monitor"`. We get

```
position = len(string) / 2 = 7 / 2 = 3 (with remainder discarded)
result = string[3] = "i"
```

For the even-length string `"monitors"`, we get

```
position = len(string) / 2 = 4
result = string[3] + string[4] = "it"
```

Step 7 Assemble the `if` statement in Python.

Here's the completed code segment.

```
position = len(string) // 2
if len(string) % 2 == 1 :
   result = string[position]
else :
   result = string[position - 1] + string[position]
```