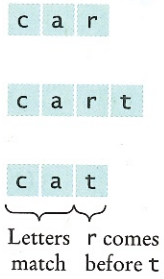
In Python:

- All uppercase letters come before the lowercase letters. For example, "Z" comes before "a".
- The space character comes before all printable characters.
- Numbers come before letters.
- For the ordering of punctuation marks, see Appendix A.

When comparing two strings, you compare the first letters of each word, then the second letters, and so on, until one of the strings ends or you find the first letter pair that doesn't match.

If one of the strings ends, the longer string is considered the "larger" one. For example, compare "car" with "cart". The first three letters match, and we reach the end of the first string. Therefore "car" comes before "cart" in the lexicographic ordering.

When you reach a mismatch, the string containing the "larger" character is considered "larger". For example, let's compare "cat" with "cart". The first two letters match. Because t comes after r, the string "cat" comes after "cart" in the lexicographic ordering.



Lexicographic Ordering

HOW TO 3.1 Implementing an `if` Statement

This How To walks you through the process of implementing an `if` statement.

Problem Statement The university bookstore has a Kilobyte Day sale every October 24, giving an 8 percent discount on all computer accessory purchases if the price is less than $128, and a 16 percent discount if the price is at least $128. Write a program that asks the cashier for the original price and then prints the discounted price.

Step 1 Decide upon the branching condition.

In our sample problem, the obvious choice for the condition is:

```
original price < 128?
```

That is just fine, and we will use that condition in our solution.

But you could equally well come up with a correct solution if you choose the opposite condition: Is the original price at least $128? You might choose this condition if you put yourself into the position of a shopper who wants to know when the bigger discount applies.

Sales discounts are often higher for expensive products. Use the `if` statement to implement such a decision.

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

In this step, you list the action or actions that are taken in the "positive" branch. The details depend on your problem. You may want to print a message, compute values, or even exit the program.

In our example, we need to apply an 8 percent discount:

```
discounted price = 0.92 x original price
```

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

What do you want to do in the case that the condition of Step 1 is not satisfied? Sometimes, you want to do nothing at all. In that case, use an `if` statement without an `else` branch.

In our example, the condition tested whether the price was less than $128. If that condition is *not* true, the price is at least $128, so the higher discount of 16 percent applies to the sale:

```
discounted price = 0.84 x original price
```

Step 4 Double-check relational operators.

First, be sure that the test goes in the right *direction*. It is a common error to confuse > and <. Next, consider whether you should use the < operator or its close cousin, the <= operator.

What should happen if the original price is exactly $128? Reading the problem carefully, we find that the lower discount applies if the original price is *less than* $128, and the higher discount applies when it is *at least* $128. A price of $128 should therefore *not* fulfill our condition, and we must use <, not <=.

Step 5 Remove duplication.

Check which actions are common to both branches, and move them outside.

In our example, we have two statements of the form

```
discounted price = ___ x original price
```

They only differ in the discount rate. It is best to just set the rate in the branches, and to do the computation afterwards:

```
If original price < 128
   discount rate = 0.92
Else
   discount rate = 0.84
discounted price = discount rate x original price
```

Step 6 Test both branches.

Formulate two test cases, one that fulfills the condition of the `if` statement, and one that does not. Ask yourself what should happen in each case. Then follow the pseudocode and act each of them out.

In our example, let us consider two scenarios for the original price: $100 and $200. We expect that the first price is discounted by $8, the second by $32.

When the original price is 100, then the condition 100 < 128 is true, and we get

```
discount rate = 0.92
discounted price = 0.92 x 100 = 92
```

When the original price is 200, then the condition 200 < 128 is false, and

```
discount rate = 0.84
discounted price = 0.84 x 200 = 168
```

In both cases, we get the expected answer.

Step 7 Assemble the `if` statement in Python.

Type the skeleton

```
if :
else :
```

and fill it in, as shown in Syntax 3.1 on page 94. Omit the `else` branch if it is not needed.

In our example, the completed statement is

```
if originalPrice < 128 :
   discountRate = 0.92
else :
   discountRate = 0.84
discountedPrice = discountRate * originalPrice
```

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