Fast Facts

Teachers Teaching with Technology™
Professional Development from Texas Instruments

Student

7 **8** 9 10 11 12

TI-Nspire | Coding | Student | 60 min

Finding Factors

There are many ways to determine the quantity of factors for a specified number. The most common method is to test the divisibility for every number up to the specified number, however this is a slow process and many of the numbers being tested are not necessary.

Example: Determine the quantity of factors for the number 45.

45 ÷ 1 = 45	Factor ✓	45 ÷ 2 = 24 rem 1	Not a factor	45 ÷ 3 = 15	Factor ✓

Testing from these first three numbers provides some possible short cuts.

- 45 is not divisible by 2, therefore it cannot be even. If the number is not even there is no purpose checking divisibility by 4, 6, 8 … An efficient algorithm (program) should therefore check first to see if the number being tested is even, potentially removing 50% of future testing.
- 45 is divisible by 3 since: 3 x 15 = 45. With the exception of perfect squares, factors occur in pairs so once one number in the pair has been identified the other can be found by division rather than additional searching. This approach means that a much smaller testing threshold can be established.

45 ÷ 1 = 45	Factor ✓	45 ÷ 2 = 24 rem 1	Not a factor	45 ÷ 3 = 15	Factor ✓
45 ÷ 4 = 11 rem 1	Not a factor	45 ÷ 5 = 9	Factor ✓	45 ÷ 6 = 7 rem 3	Not a factor
45 ÷ 7 = 6 rem 2	Not a factor				

- In the example above no further testing is required as the divisor (7) is now greater than the quotient (6). Further increases in the divisor will only continue to reduce the quotient locating factor partners that have already been established. The threshold at which this occurs is the square-root of the original number. $\sqrt{45} \approx 6.71$, so searching for factors of 45 can cease at 6.

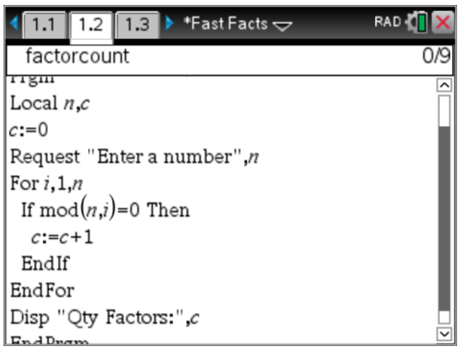
Instructions

Open the TI-Nspire document: Fast Facts.

Navigate to page 1.2: "factorcount".

This is the program listing for the "factorcount" program that 'requests' a number and returns the total number of factors for that number.

Navigate to page 1.3, use the [**Var**] key to access the factorcount program, select and run the program. When prompted enter the number 360. The program should return: 24, signifying there are 24 factors for the number 360.



```
Local n,c
c:=0
Request "Enter a number",n
For i,1,n
 If mod(n,i)=0 Then
  c:=c+1
 EndIf
EndFor
Disp "Qty Factors:",c
```

Author: P. Fox

Question: 1.

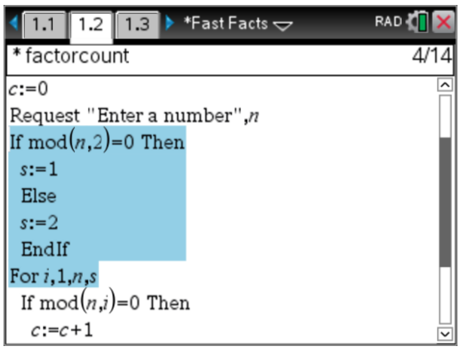
Use a stop watch to record how long the calculator takes to count the quantity of factors for each of the following numbers:

Number	100	101	1,000	1,001	10,000	10,001	100,000	100,001
Time								

Return to page 1.2 and edit the program. The required edits and changes have been highlighted.

When you have finished editing press **Ctrl** + **B** to save and compile the programming code.

Return to page 1.3 so the program can be run again and tested.



```
c:=0
Request "Enter a number",n
If mod(n,2)=0 Then
 s:=1
 Else
 s:=2
 EndIf
For i,1,n,s
 If mod(n,i)=0 Then
  c:=c+1
```

Question: 2.

What is the "**IF**" statement checking when it tests **mod(n,2)=0**?

Question: 3.

The **FOR** syntax is: **For** *variable, start, finish, step*. What possible values can the step size take according to the previous **IF** statement?

Question: 4.

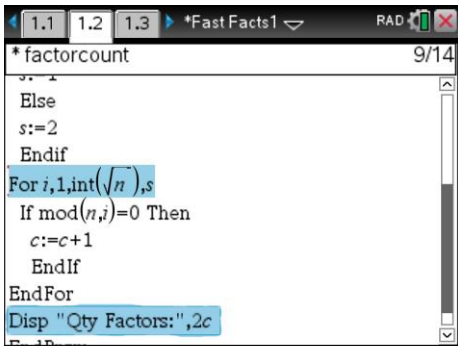
Run the program and check how long it takes to count the factors for the numbers in Table 1 of Question 1, compare your results with those obtained previously.

Return to page 1.2 and edit the highlighted section once again.

The "INT" statement removes the decimal values from a calculation guaranteeing that the FOR loop has a whole number to count up to.

To understand the inclusion of the Square-root calculation, read back through the introduction "Finding Factors"

Notice also that the factor count result is doubled.



```
 Else
 s:=2
 Endif
For i,1,int(√n),s
 If mod(n,i)=0 Then
  c:=c+1
  EndIf
EndFor
Disp "Qty Factors:",2c
```

Question: 5.

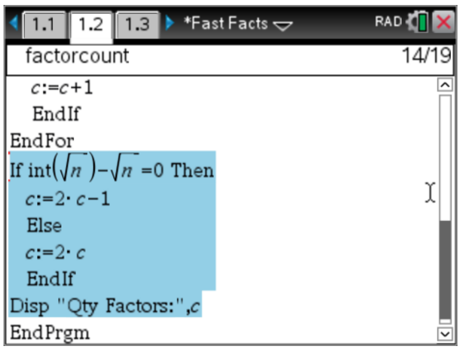
Run the program and check how long it takes to count the factors of 10,001 and 10,000. Compare your results to those obtained in Question 1.

Question: 6.

An important aspect of writing code is to test it, to make sure it is working properly. Write down the factors for each of the following numbers: 18, 32, 37, 45, 50, 100 and 144. Check if your program returns the correct quantity of factors for each number. If there are any discrepancies, suggest a possible cause.

Return to page 1.2 and edit the highlighted section once again.

This final check corrects a problem that occurs for a special group of numbers, the inclusion of this final step however does not add much computational time to the program because it is not contained within the search loop and is therefore executed just one time at the end of the program.



```
  c:=c+1
  EndIf
EndFor
If int(√n)−√n =0 Then
  c:=2· c−1
  Else
  c:=2· c
  EndIf
Disp "Qty Factors:",c
EndPrgm
```

Question: 7.

Write down at least 10 different numbers. Write down the factors for each of the numbers and use your program to check the factor count.

Question: 8.

The number 21 is the product of exactly two prime numbers: 7 and 3.

a. Write down the factors of 21.

b. The number: 6,397 is prime, so too 9,397. The product of these two prime numbers is equal to: 60112609. (60 million, 112 thousand, 609) Predict how many factors there are for 60112609 then use the FactorCount program to check the quantity of factors.

c. The number 67,629,137 is prime, so too is 73,939,133. The product of these two prime numbers is: 5,000,439,755,318,221. (5 Quadrillion, 439 billion, 755 million, 318 thousand and 221). Predict the quantity of factors for this very large number.
Do **NOT** test this with your calculator program!

Encryption techniques used by banks and other organisations that transmit private or secure data over the internet rely on the fact that it is very time consuming to find the factors of a very large number, particularly where that large number is the product of very large prime numbers. As factorising techniques are improved and computer processing power increases, the prime numbers being used for secure transactions must get bigger and bigger in order to keep your data safe.