Factor Count

Factors that Count

Teacher Notes:

This activity is part of a complete module that includes:

- Factor Count
- Euclid's Algorithm
- Euler Totient Function
- Highly Composite Numbers

This module is part of a series that can be combined as a complete unit under the umbrella of a "Mathematics Elective" designed to extend, enrich and prepare students for studies in senior mathematics courses.

The complete "Code by Numbers" module can be downloaded from the STEM pages:

https://education.ti.com/en-au/resources/stem

A PowerPoint slide show is provided with this activity as an introductory presentation for students to watch. The slides work through the algorithmic process for the determination of the factors for the number: 36. Slides reference mathematical terminology such as: divisor, quotient and remainder, the animations are designed to help students understand these terms.

The presentation does not cover all possible divisors, instead, it leaves students pondering the most efficient algorithm by stopping at a divisor of 9 and posing the comment: "The factors are now starting to repeat themselves".

A perfect square has been used on purpose in the slide set, it serves as a subtle hint that it may only be necessary to search up to the square-root of the corresponding number. If this efficiency is incorporated, students would need to incorporate a check routine for perfect squares to ensure a double count of the square-root is not erroneously included in the factor count.

Instructions for the simplest program (not the quickest) are provided here so that students may also be assessed on their ability to independently arrive at a more efficient factor searching routine and conditions.

More advanced students may check if the input is odd and therefore start the loop counter with an odd number and use a step size two, therefore skipping divisibility for all subsequent even quantities.

Author: P. Fox

TEXAS INSTRUMENTS

TI-Codes Lessons:

Unit 1 – Skill Builder 1
⇩
Unit 4 – Skill Builder 1

Commands:

- input
- for (range)
- if
- print
- int (number types)

- [ ] (create a list)
- Append (add elements to a list)
- len (length of a list)
- % (module

Finding and Counting 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 all applicable numbers. For example, suppose we want to determine the quantity of factors for 18. We can determine the quotient and remainder for all the numbers from 1 to 18.

Table 1A – Finding the factors of 18

Divisor	1	2	3	4	5	6	7	8	9
Quotient	18	9	6	4	3	3	2	2	2
Remainder	0	0	0	2	3	0	4	2	0

Table 1B – Finding the factors of 18

Divisor	10	11	12	13	14	15	16	17	18
Quotient	1	1	1	1	1	1	1	1	1
Remainder	8	7	6	5	4	3	2	1	0

Our conclusion is that 18 has six factors since there are six occasions whereby the remainder is equal to zero.

This divisibility check for all numbers is exhaustive. You may have ideas about how this process can be made more efficient, however, this method will provide a basis for an algorithm on which to write a simple program to count the quantity of factors for a given number. You can make the necessary improvements and checks once your initial program is complete and functioning.

Question: 1.

Write a description of the steps required to determine the quantity of factors for any whole number: n.
Answer: Student answers will vary, the pseudo-code provides the basis on which the program will be written.

Sample:

〉 Input number: n
〉 Set factor count to 0
〉 Loop from 1 to n
〉 If n ÷ (loop counter) has no remainder Then increase (factor count)
〉 End Loop
〉 Display (factor count)

Author: P. Fox

TEXAS INSTRUMENTS

Instructions:

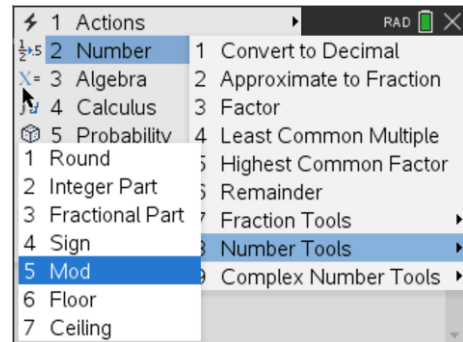Start a new document and insert a calculator application.

Locate the **mod** command using: **Number** > **Number Tools** > **Mod**

Determine the result of the following calculations:

    Mod(18,6)

    Mod(18,5)

    Mod(18,12)

Question: 2.

    Based on your experimentation, what value does the MOD command return?

    The mod command returns the remainder upon division.

Question: 3.

    If MOD(a, b) = 0, what does this say about the relationship between a and b?

    If the 'remainder' of $a \div b$ = 0 then b must be a factor of a.
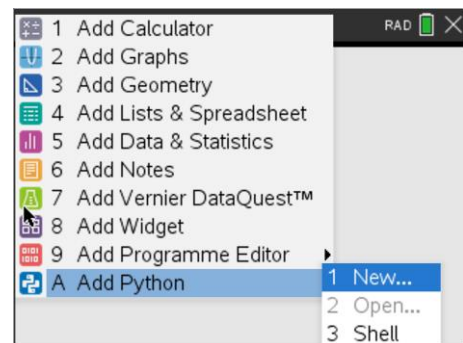
Writing a Program

Create a new Python program by pressing:

    [ctrl] + [doc▾] , **Add Python** > **New…**

Call the program: FactorCount

Note that 'FactorCount' is one word as program names cannot contain spaces.

🎯 If the programming application is launched on the same page as the Calculator Application. The page-layout in the document menu can be used to give each application its own page.
**Short-cut**: [**Ctrl**] + [**6**]. Page 1.1 = Calculator Application. Page 1.2 = Program Application.

The first task is to request a number from the program user and store it as a variable. Type in:

    n =

Python input defaults to text, so the next step is to restrict the input to a whole number (integer). The int() command is located in the "**type**" menu, alternatively it can be typed in directly from the keyboard:

Press:

    [menu] > **Built-ins** > **Type** > **int()**

Author: P. Fox

TEXAS INSTRUMENTS
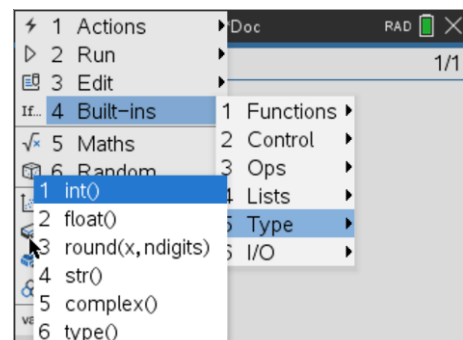
The next step is to enter the "input" command:

menu > **Built-ins** > **I/O** > **input()**

Finish the instruction by adding a text prompt.

When this command is executed, the user's numerical input will be stored in a variable "n"

```
n=int(input("Enter your number: "))
```

- Quotation marks: **" "** can be entered by pressing [**Ctrl**] + [**x**] (multiplication sign)
- Colour is added automatically to help locate the various parts of the syntax.

We could just count factors, however, it is easy record and store them in a list which also helps with checking the results.

Factors = [ ]

This creates an empty list called factors.

A "FOR" loop can be used to check for factors. We use a FOR loop because we can pre-determine the quantity of iterations the loop must perform.

menu > **Built-ins** > **Control** > **For index in Range(***size***)**

Python loop execution ceases when the counter reaches *size*, therefore the counter (*size*) needs to be one more than n.

```
n=int(input("Enter your number: "))
facts=[]
for i in range(1,n+1):
```

An **IF** statement will be used to check if the user's number has a factor each time the program executes the loop.

The IF command can be selected by:

menu > **Built-ins** > **Control** > **If**

The % operation in Python is for modular arithmetic.

The percentage sign can be obtained from the punctuation fly-out menu.

```
n=int(input("Enter your number: "))
facts=[]
for i in range(1,n+1):
    if n%i==0:
```

The 'append' command adds the latest factor to the current list of factors. This command can be typed in directly or accessed from the variable menu (facts) followed by:

menu > **Built-ins** > **Lists** > **.append()**

The value to be added to the list, given that the division has not generated any remainder, is "*i*", the loop counter.

That's all for the algorithmic part of the program! The next step is to display the results. Start by deleting the indentions, the end of the "IF" condition and the For loop.

```
n=int(input("Enter your number: "))
facts=[]
for i in range(1,n+1):
    if n%i==0:
        facts.append(i)
```

Author: P. Fox

TEXAS INSTRUMENTS

The list ("facts") contains all the factors, len(facts) therefore returns the size of the list. This quantity can be stored in 'd'.
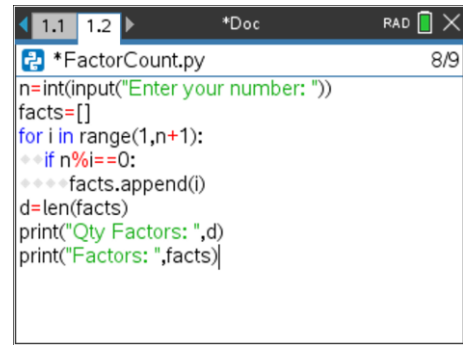
Now the quantity of factors (d) and the actual factors can be printed to the screen.

Save the program and launch it by pressing [**Ctrl**] + [**R**]. A new Python shell will be created and the program name automatically pasted. Start by checking the factor count for 18.

The table at the start of this activity identifies 6 factors, compare this with the output from your program.

To check another number, press [**Ctrl**] + [**R**]

```
1.1  1.2          *Doc          RAD  ☐ ✕
*FactorCount.py                        8/9
n=int(input("Enter your number: "))
facts=[]
for i in range(1,n+1):
    if n%i==0:
        facts.append(i)
d=len(facts)
print("Qty Factors: ",d)
print("Factors: ",facts)
```

Question: 4.

Determine the quantity of factors for each of the following numbers:

a. 24   8 factors … {1, 2, 3, 4, 6, 8, 12, 24}

b. 36   9 factors … {1, 2, 3, 4, 6, 9, 12, 18, 36}

c. 37   2 factors (prime numbers have exactly 2 factors) … {1, 37}

d. 144   15 factors … {1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 36, 48, 72, 144}

Check each of your answers by writing down all the factors. (Factors listed above)

Question: 5.

Determine the quantity of factors for each of the following numbers. Identify a specific characteristic about the quantity of factors and use this to classify the numbers into two groups, explain your classification.

29 (2), 84 (12), 104 (8), 87 (4), 22 (4), 37 (2), 101 (2), 97 (2), 45 (6), 43 (2), 133 (4), 153 (6), 173 (2), 107 (2). The numbers: 29, 37, 101, 97, 43 and 173 all have exactly two factors and form the group of 'prime' numbers. **Teacher Notes**: Referring to prime numbers as having exactly two factors removes any potential ambiguity with regards to whether or not 1 is prime.

Question: 6.

Determine the quantity of factors for each of the following numbers. Identify a specific characteristic about the quantity of factors and use this to classify the numbers into two groups, explain your classification.

28 (6), 30 (8), 90 (12), 45 (6), 50 (6), 60 (12), 120 (16), 72 (12), 25 (3), 49 (3), 81 (5), 144 (15), 441 (9), 82 (4), 24 (8), 720 (30)
This classification is harder than the previous one … students need to pick the 'odd' ones out. If students can see that 25, 49, 81, 144 and 441 all have an odd number of factors they should also identify that these numbers are perfect squares. Perfect squares are the only numbers that have an odd number of factors since each contains a 'repeated' factor.

Question: 7.

The FactorCount program works, but it could be more efficient. Use a stop watch to time how long the program takes to count the number of factors for: 100,000; 200,000 and 300,000. Use these times to predict how long the program will take to count the factors for 500,000. Test your answer!
If you are satisfied with your prediction, how long would it take to find factors for the following number:

2,140,324,650,240,744,961,264,423,072,839,333,563,008,614,715,144,755,017,797,754,920,881,418,023,447,140,136,643,345,519,095,804,679,610,992,851,872,470,914,587,687,396,261,921,557,363,047,454,770,520,805,119,056,493,106,687,691,590,019,759,405,693,457,452,230,589,325,976,697,471,681,738,069,364,894,699,871,578,494,975,937,497,937 [250 digits!] **Note**: This number is associated with RSA Encryption.

Author: P. Fox

TEXAS INSTRUMENTS

Counting factors for 100,000 takes approximately 1 second. [TI-Nspire CX II series]
Counting factors for 200,000 takes approximately 1.5 seconds.
Counting factors for 300,000 takes approximately 2 seconds.
Students should see that each 100,000 numbers take approximately 0.5 seconds. Assuming that the larger numbers do not take any longer, 500,000 should take approximately 2.5 to 3.0 seconds.

Timed result: ≈ 3.1 seconds.

A simple improvement to the algorithm (working to square-root of n), however to include the square-root of a number the maths module needs to be imported first.

Even using the relatively simple improvement to the algorithm, the really big number containing 250 digits, would take 10^{238} years.
This time estimation doesn't allow for additional routines required to handle the quantity of digits in the number.
This time factor is why super-computers are required to work on such large numbers and helps explain why these numbers are used in the public key encryption process.

Investigation

Why are factors important? To answer this question, consider the opposite situation, the *absence* of factors. Billions of dollars are moved around electronically every day, to do this securely, the electronic transfers must be encrypted. The most common encryption method (RSA) is built around very large prime numbers, numbers with an *absence* of factors! All encryption methods are essentially built on numbers, so being able to 'assemble' and 'disassemble' numbers is extremely important.

Teacher Notes

To help build relevance to this investigation, consider engaging students in a discussion about the Enigma Machine, a powerful encryption tool created and used by the Germans during World War II. The Imitation Game [movie] is a wonderful way to show students just how important mathematics and mathematicians are to the world. The movie looks at Alan Turing and a team of mathematicians as they build a computing device to help solve the enigma code. While the Enigma machine was not based on prime numbers, it helps illustrate a long history, pre-dating computers, pertaining to the importance of encryption.

Imitation Game Move Trailer

In the 21st century, encryption requirements have become ubiquitous. RSA encryption, invented in 1977 by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman is based on prime numbers. The story reads like a Hollywood movie! Initially the encryption process was supposed to be reserved only for military communications, however Ron, Adi and Leonard were so confident their system could not be hacked, they released it to the world, even explaining how it works! Their encryption system is still in use today!

Interview with Ron Rivest

Author: P. Fox

Texas Instruments

Your task is to find a rule that determines the quantity of factors for any whole number, given the prime factorisation of that number.

A few clues are provided along the way to help you on your factor sleuth journey. Document your search findings and conclusions using the clues and your constructed program to help expedite your investigation.

Clue 1:

Determine the prime factorisation and corresponding quantity of factors for the following numbers: 36; 100; 441; 3025 & 48841.

Clue 2:

Determine the prime factorisation and corresponding quantity of factors for the following numbers: 24; 250; 1029; 6655 and 198911.

Clue 3:

Based on the first two clues, generate some numbers that you believe have exactly 8 factors. Test your numbers and comment on the results.

Clue 4:

Determine the prime factorisation and corresponding quantity of factors for the following numbers: 2000; 64827; 107811; 668168 and 1585615607. [Note: For this last number you will need a fast algorithm!]

Clue 5:

Create some numbers of the form: $m^2 \times n^5$ where m and n are both prime. Determine the quantity of factors for each of your numbers. [Note: You may want to choose relatively small prime numbers for m and n.]

Continue the exploration, tabulate your results and record your thoughts, hypotheses, tests and reflections as you go. Documenting findings is an important part of the investigative process. Detectives may have many suspects in their initial investigations, however as more clues surface they develop hypotheses. Detectives test each hypothesis, review what they already know or go in search of more clues. Some investigations end up as Cold Cases, however it is critical that detailed documentation of all aspects of their investigation are retained in the event the investigation is re-opened. Some crimes remain unsolved despite having significant suspects, in mathematics these are often called 'conjectures', a theory that seems to work but has never been proven.

Answers: Students should record their findings in a table. The clues prompt students to focus on the exponents rather than the bases. For example: $2^2 \times 3^2$ = 36 and $2^2 \times 5^2$ = 100 both have the same quantity of factors. They have the same indices but different bases, similarly with the other examples of 441, 3025 and 48841.

If students tabulate the indices and quantity of factors they should start to see the connection: {2, 2, 9}; {2, 3, 12}; {1, 2, 6} … adding one to each exponent and then calculating the product results in the quantity of factors.

Students should go beyond numbers with two prime factors and also check that the algorithm works for prime numbers.

Teacher Notes: A sample program loops up to the square-root of the input number.

Python does not have a square-root command so it must be imported first.

Author: P. Fox

TEXAS INSTRUMENTS

```
◄ 1.1  1.2  1.3 ►    *Doc        RAD ☐ ✕
📄 FactorCount.py                    6/18
  if n%i==0:
      facts.append(i)
m=len(facts)−1
if sqrt(n)−int(sqrt(n))==0:
  m=m−1
for i in range(m,−1,−1):
  p=int(n/facts[i])
  facts.append(p)
d=len(facts)
print("Qty Factors: ",d)
print("Factors: ",facts)
```

Author: P. Fox

TEXAS INSTRUMENTS