Highly Composite Numbers

Student Activity

TI-Nspire™

Activity

Student

180 min

7 8 **9** 10 11 12

TI-Codes Lessons:

Unit 1 – Skill Builder 1

Unit 4 – Skill Builder 1

Commands:

- input
- for (range)
- if
- print
- int (number types)
- def function
- [ ] (create a list)
- Append (add elements to a list)
- % (modular arithmetic)
- Import module

Introduction

A highly composite number has more factors than any of its predecessors. Think of it as competition along the number line. The difficulty in locating highly composite numbers is that you must already know the previous highly composite number in order to identify how many factors the next number must have in order to qualify. Any search for highly composite number therefore generally starts at 1.

Whilst 1 only has one factor, there are no predecessors, so by default, 1 is the first highly composite number. Naturally 2 is the next highly composite number having two factors. The next is 4 with three factors then 6 with four factors. With one, two, three and four factors already checked, it would be easy to assume that the next highly composite number would have five factors, however 12 is the next highly composite number with six factors.

Question: 1.

Write a description of a program that will determine the Highly Composite number up to some value n.
Note: The quantity of factors for any number can be referenced as 'factor_count'.

Writing a Program

Instructions:

Start a new document; insert a new Python program.

Add Python > **New**

Call the program: HCN

To make the program efficient, it is desirable to have access to the 'square-root' function. Import the 'math' module.

Math > **from math import**

To access results outside the Python shell, import the TI-System module.

More Modules > **TI-System** > **from ti-system import**

```
1.1        *Doc        RAD
*HCN.py                3/4
from math import *
from ti_system import *
```

© Texas Instruments 2023. You may copy, communicate and modify this material for non-commercial educational purposes provided all acknowledgements associated with this material are maintained.

Author: P. Fox

Creating a function to efficiently determine the quantity of factors will make the main program much easier.

Define a function called "factors" with input 'n':

Built-ins > **functions** > **def function()**

A counter (c) will be used to count each factor with a loop to search for the factors. The loop only needs to go to the square-root of the chosen number, but a final check will be necessary in the event that the original number is a perfect square.

```
1.1  *Doc  RAD
*HCN.py  1/8
from ti_system import *
from math import *
def factors(n):
  c=0
  for i in range(1,int(sqrt(n))+1):
```

The loop checks if the current number (n) is divisible using modular arithmetic (%), if there is no remainder, then 'i' must be a factor of 'n', so the counter is increased by one.

```
1.1  *Doc  RAD
*HCN.py  8/8
from math import *
def factors(n):
  c=0
  for i in range(1,int(sqrt(n))+1)):
    if n%i==0:
      c=c+1
```

Once the loop has finished, a check must be performed to see if the original number was a perfect square. If the original number was a perfect square, doubling the quantity of factors would count the square-root twice.

If the original number was not a perfect square, then the quantity of factors is doubled as all the factors counted to date have a 'partner'.

Finally, the quantity of factors (c) is returned to the program.

```
1.1  *Doc  RAD
*HCN.py  12/14
from math import *
def factors(n):
  c=0
  for i in range(1,int(sqrt(n))+1):
    if n%i==0:
      c=c+1
  if sqrt(n)==int(sqrt(n)):
    c=2*c-1
  else:
    c=2*c
  return(c)
```

Several variables need to be initialised at the start of the program.

- QTY = The quantity of factors for the highly composite number
- HCNS = Highly Composite Numbers
- Record = Quantity of factors for the current HCN.

The first highly composite number '1' is seeded into the variables as it is the only 'odd' highly composite number.

Note: "qty' and 'hcns' will hold a list of numbers that will be continually updated.

```
1.1  *Doc  RAD
*HCN.py  17/18
  if sqrt(n)==int(sqrt(n)):
    c=2*c-1
  else:
    c=2*c
  return(c)

qty=[1]
hcns=[1]
record=1
p=int(input("Number: "))
```

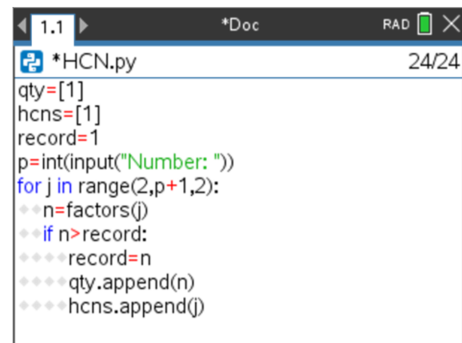
© Texas Instruments 2023. You may copy, communicate and modify this material for non-commercial educational purposes provided all acknowledgements associated with this material are maintained.

Author: P. Fox

The loop can start at 2 since the first highly composite number (1) has already been stored. As all subsequent HCN's are even, the step counter can be set at 2.

The first instruction in the loop is to store the quantity of factors in variable 'n'; if this quantity is larger than the current record, the current record is updated and the 'qty' and 'hcns' lists are updated.

Note: The append command adds the specified value to the end of the specified list.



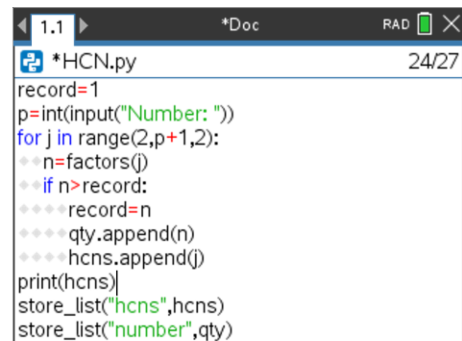
Once the loop is finished, all the highly composite numbers have been stored and can therefore be displayed and transferred to variables that can be accessed by the current document.

More Modules > TI System > store_list("name",list)

"name" represents the name of the variable in the current document.

"list" refers to the list in the current program (Python shell).

The program is now complete and ready to run.



Question: 2.

Run your program and check that the first five highly composite numbers are: 1, 2, 4, 6, 12; then determine all the highly composite number from 1 to 100.

Question: 3.

Determine all the highly composite numbers from 1 to 1000 and their corresponding quantity of factors.

Question: 4.

Express each of the Highly Composite Number in the previous question as a product of its prime factors.

Question: 5.

Study the prime factorisations closely. Suggest a possible prime factorisation for the next highly composite number, the corresponding number and quantity of factors.
Note: You may have more than one educated guess.

Investigation

To continue exploring Highly Composite Numbers, a more efficient program (or new program) is required, one that no longer starts at 1, rather one that starts at some previously identified Highly Composite Number and uses information gleaned from the first sixteen highly composite numbers.

- Re-write your HCN program so that it can start at any HCN.
- Continue recording HCNs and the corresponding prime factorisations. When and what will be the next prime factor to be included in the prime factorisation?
- Identify any patterns you can find in the prime factorisation that would help in locating subsequent prime factorisations.
- What prior learning are you using to identify the quantity of factors, make predictions and search?

© Texas Instruments 2023. You may copy, communicate and modify this material for non-commercial educational purposes provided all acknowledgements associated with this material are maintained.

Author: P. Fox