



2. **if** statements come in three flavors: **if..**, **if.. else..**, and **if.. elif.. else..**. They are all found on **<Fns...> Ctl**. Note that there is no “then” in Python.

(They are on the **Ctl** menu because these statements **Control** the flow of your program.)

- if ..**                      *Use when there is no “otherwise” action.*
  
- if .. else ..**            *Use when there are exactly two alternative actions for **True** and **False** (either do this or do that). (You will use this one soon.)*
  
- if .. elif .. else ..**    *Use these when there are three or more actions to be taken based on several conditions. **elif** is short for “else if...” and **requires** a condition just like **if**. You can add as many **elifs** as your algorithm requires. (This structure is used in the application for this unit.)*

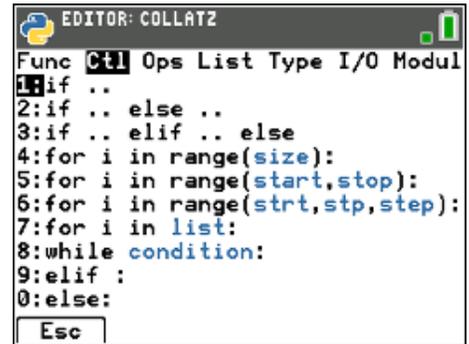
There *must* be a colon ( : ) at the end of *each* **if**, **elif** and **else**. These colons are *required* and indicate that what follows are the actions to be taken when the *condition* is **True** or **False**.

**Teacher Tip:** Selecting the structure from the menu ensures that all proper syntax and characters are included. When typing in the commands by hand, beginners often forget about proper indentation and colons.

Conditions are called *Boolean Expressions*, and each indented “block” will be filled with your actions.

**True**, **False** and **None** are the only Python keywords that are **Capitalized**.

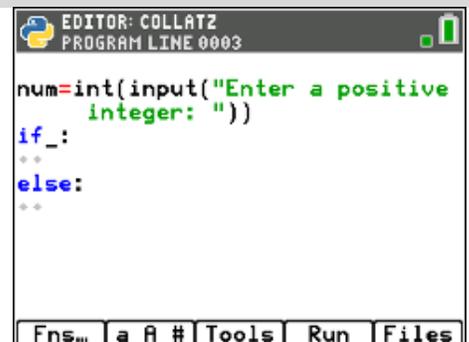
Suppose you write: **thing=(x<0)    #(yes, this is valid!)**  
 Never write **if thing == True:** ... that’s a sure sign of a beginner!  
                   write **if thing:**                      instead, since **thing is** either **True** or **False!**



```

EDITOR: COLLATZ
Func Ctl Ops List Type I/O Modul
1:if ..
2:if .. else ..
3:if .. elif .. else
4:for i in range(size):
5:for i in range(start,stop):
6:for i in range(strt,stp,step):
7:for i in list:
8:while condition:
9:elif :
0:else:
Esc
    
```

3. Insert the **if.. else** statement from **<Fns...> Ctl**.



```

EDITOR: COLLATZ
PROGRAM LINE 0003
num=int(input("Enter a positive
integer: "))
if_:
..
else:
..
Fns... a A # Tools Run Files
    
```

4. The condition (*written between if and the colon*) is ...

```
if num % 2 == 0:
```

% is called “mod” and is the mathematical operator (like +, -, \*, and /) that gives the *remainder* when the first number is divided by the second.

“mod” is short for “modulus.”

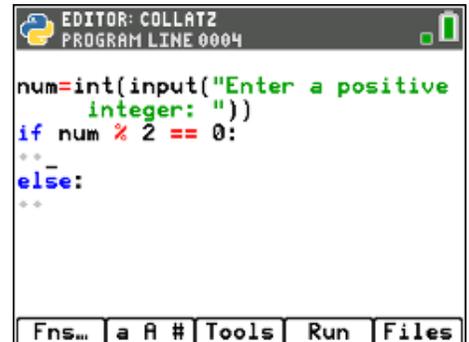
% is found on the <a A #> screen or in the **catalog [2<sup>nd</sup>] [0]**.

The statement now reads: “If the *remainder* when **num** is divided by 2 is zero,” which means “if **num** is *even*...”

Note the two equal signs!

For == just use any of the following:

- press the [sto ->] key twice or
- select it from <Fns...> Ops or
- select from <a A #> or
- press [test]



```
EDITOR: COLLATZ
PROGRAM LINE 0004

num=int(input("Enter a positive
integer: "))
if num % 2 == 0:
**
**
else:
**
**

Fns... a A # Tools Run Files
```

**Teacher Tip:** Notice that there is no “then” in Python. Waste of space!

5. **if** (*the number is even*) :

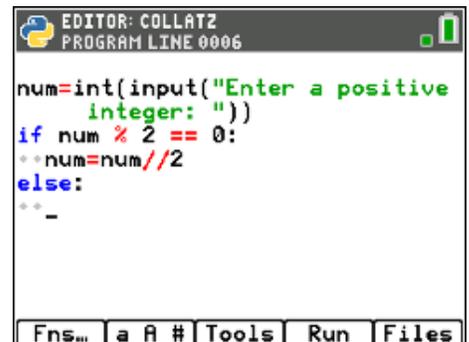
◆◆ (*the “True” block*)

is:

```
◆◆ num = num // 2
```

Just use two / signs (the [ ÷ ] key).

// (two division signs) is called *floor division* (no decimal and truncates to the integer just *below* the decimal value). If you use / you will see a decimal point even if all the numbers are integers.



```
EDITOR: COLLATZ
PROGRAM LINE 0006

num=int(input("Enter a positive
integer: "))
if num % 2 == 0:
** num=num//2
else:
**
**

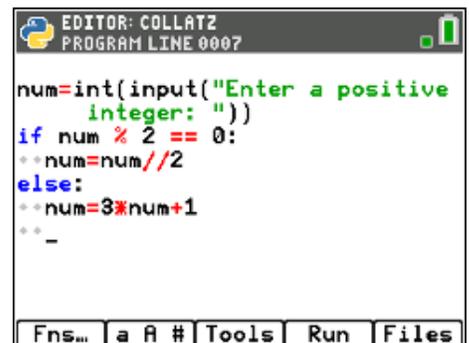
Fns... a A # Tools Run Files
```

6. **else:** (*when the number is odd — no condition here*) :

(*the “False” block*)

is:

```
◆◆ num = 3 * num + 1
```



```
EDITOR: COLLATZ
PROGRAM LINE 0007

num=int(input("Enter a positive
integer: "))
if num % 2 == 0:
** num=num//2
else:
** num=3*num+1
**
**

Fns... a A # Tools Run Files
```

7. After the **if.. else:** structure, backspace to the beginning of a line (erase the indent characters) and write the **print** statement:

**print(num)**

8. **Running the program:**

Select **<Run>** to run the program. Enter a positive integer. An answer appears. Remember it!

Select **<Tools>** **[enter]** to run the program again and this time enter *the last answer*.

Repeat running the program, each time entering the previous answer. Eventually....

*But wait! Let's add a loop to the program so that the process runs repeatedly by itself instead of having to run the program over and over.*

9. Place your cursor right below the **input** statement and above the **if** statement as shown in this image.

10. On this *blank* line add the **while** statement found on

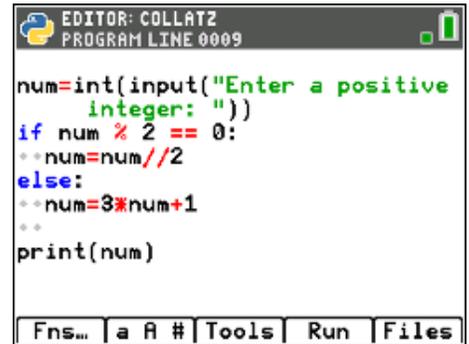
**<Fns...> Ctl**

You will see:

**while :**

◆◆

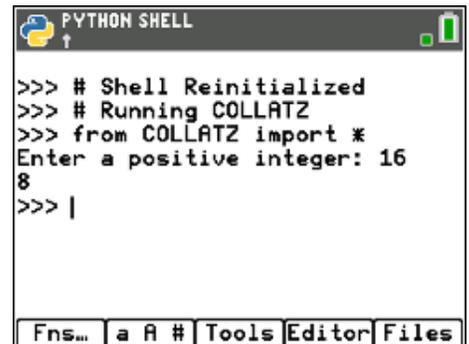
pasted into your program and your cursor is blinking on the colon.



```

EDITOR: COLLATZ
PROGRAM LINE 0009

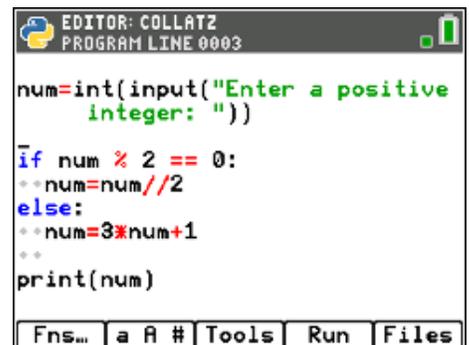
num=int(input("Enter a positive
integer: "))
if num % 2 == 0:
    num=num//2
else:
    num=3*num+1
print(num)
    
```



```

PYTHON SHELL
↑

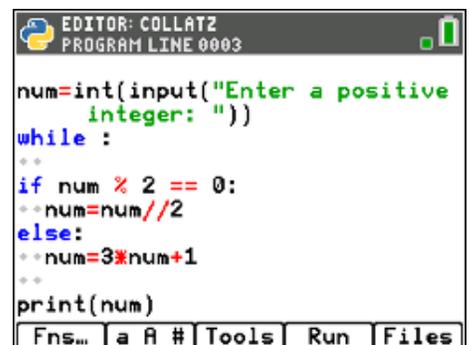
>>> # Shell Reinitialized
>>> # Running COLLATZ
>>> from COLLATZ import *
Enter a positive integer: 16
8
>>> |
    
```



```

EDITOR: COLLATZ
PROGRAM LINE 0003

num=int(input("Enter a positive
integer: "))
if num % 2 == 0:
    num=num//2
else:
    num=3*num+1
print(num)
    
```



```

EDITOR: COLLATZ
PROGRAM LINE 0003

num=int(input("Enter a positive
integer: "))
while :
    if num % 2 == 0:
        num=num//2
    else:
        num=3*num+1
print(num)
    
```

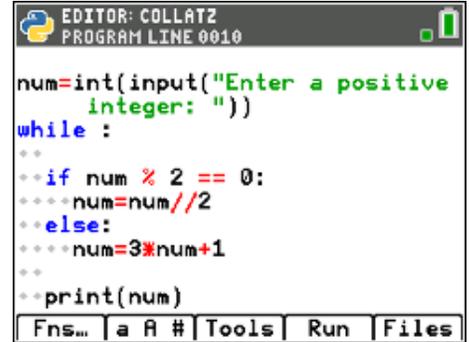
**Teacher Tip:** The While “block” is inserted (the two diamonds) but the actual block is the `if .. else` structure and `print` statement already coded. The next step shows how to indent that section of text to become the `while` block.

11. Indent each line below `while` : (the `if.. else` structure and the `print` statement) by:

- Place your cursor anywhere on a line
- Select **<Tools> Indent▶**

*Blank lines and #comments do not need to be indented since Python ignores both.*

*Indenting causes the statements to become the block of code inside the `while` structure. You still need to write the `while` condition.*



```

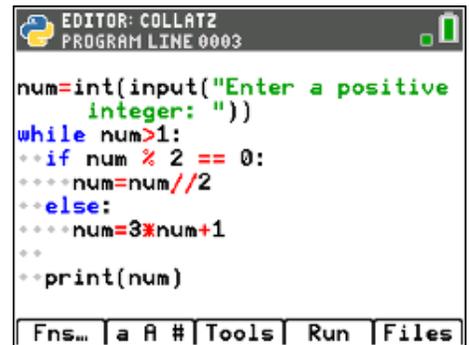
EDITOR: COLLATZ
PROGRAM LINE 0010

num=int(input("Enter a positive
integer: "))
while :
..
..if num % 2 == 0:
...num=num//2
..else:
...num=3*num+1
..
..print(num)
    
```

12. Now write the `condition` after the word `while` (leave a space after `while` and do not erase the colon).

The **Collatz Conjecture** states that all sequences will eventually become 1. As long as the number is greater than 1, continue processing so write:

`while num > 1 :`



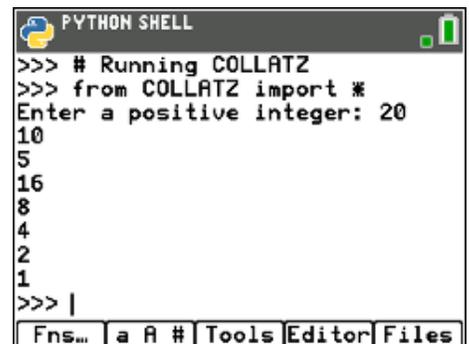
```

EDITOR: COLLATZ
PROGRAM LINE 0003

num=int(input("Enter a positive
integer: "))
while num>1:
..if num % 2 == 0:
...num=num//2
..else:
...num=3*num+1
..
..print(num)
    
```

13. Run the program now. Enter **20** as the number. Follow the logic of the program. Odd numbers get larger and even numbers get smaller:

- 20 is even → 10
- 10 is even → 5
- 5 is odd → 16
- 16 is even → 8
- and so on ...
- ... and the program ends when the number reaches 1.



```

PYTHON SHELL

>>> # Running COLLATZ
>>> from COLLATZ import *
Enter a positive integer: 20
10
5
16
8
4
2
1
>>> |
    
```

Note that it only took **one line of code** to create a loop!

*Can you find a number that causes the program to NEVER end? Try a large number. Notice how fast the numbers fly by! When the program ends you can scroll upward through the Shell history ([2<sup>nd</sup>] [uparrow]) to examine the numbers.*

**Teacher Tip:** The Shell history is plain text and is not preserved. To scroll upwards through the history, press [2<sup>nd</sup>] [up arrow] at the Shell prompt. Each press goes up one line, even into previous runs.

The **Collatz Conjecture** (first presented in 1937) is still unproven. Is there something special about the numbers 3, 1 and 2 in the algorithm? Try using different numbers. One more option that students can add to the program is to count the number of steps to reach 1.