

Python's `range` expression

Recall that a *range* expression generates integers that can be used in a FOR loop, like this:

```
for k in range(n):  
    ... k ...
```

In that example, *k* takes on the values **0**, **1**, **2**, ... **n-1**, as the loop runs. That is:

`range(n)` generates *n* integers starting at **0** (and hence ending at **n-1**).

Python allows two other forms of the *range* expression, for your convenience. You never have to use these forms (the single-argument form is sufficient), but they are often handy.

`range(m, n)` generates integers starting at *m*, ending at **n-1** (and hence generates *n-m* integers if $n \geq m$).

For example, the loop shown below to the left generates the output shown below to the right.

```
for k in range(5, 9):  
    print(k)
```

5
6
7
8



Caution: `range(m, n)` generates NO integers if $n \leq m$. For example, the loop `for k in range(9, 5):` runs NO times:

```
for k in range(9, 5):  
    print(k)
```

(no output)

The third form of the *range* expression works like this:

`range(m, n, j)` generates integers starting at *m*, in "steps" of *j*, stopping when the generated integer would be greater than or equal to *n* (if *j* is positive) or when the generated integer would be less than or equal to *n* (if *j* is negative).

For example, the loops below generate the output shown to the right.

```
for k in range(5, 11, 2):  
    print(k)
```

5
7
9

```
for k in range(8, 3, -1):  
    print(k)
```

8
7
6
5
4



Caution: In all three forms, the generated numbers *start* at the *first* number and stop just *before* reaching the "stop" number:

`for k in range(30):` does *not* include 30

`for k in range(3, 56):` does *not* include 56

`for k in range(10, 40, 5):` does *not* include 40

`for k in range(40, 10, -5):` does *not* include 10

So the example to the right does NOT generate 5, 4, 3, 2, 1, 0, as the student hoped. (Figure out why and then look at the next page.)

```
for k in range(5, 0, -1):  
    print(k)
```



Answer: the loop to the right stops just **before** it reaches 0, so it generates 5, 4, 3, 2, 1 (which may or may not be what you intend).

```
for k in range(5, 0, -1):  
    print(k)
```



Another common error is to write *this*, which runs NO times, when you meant to write *this*:

```
for k in range(0, 5, -1):  
    ... k ...
```



```
for k in range(5, 0, -1):  
    ... k ...
```



Or, similarly, to write *this* when you meant the 3-argument expression (with a negative step), as above.

```
for k in range(5, 0):  
    ... k ...
```



The arguments in a *range* expression must be *integers*. So yet another common mistake is to write *this* when the following is necessary even if the argument *n* is even.

```
for k in range(n / 2):  
    ... k ...
```



```
for k in range(n // 2):  
    ... k ...
```



Summary:

range(n) generates *n* integers starting at 0 (and hence ending at *n-1*).

range(m, n) generates integers starting at *m*, ending at *n-1* (and hence generates *n-m* integers if *n ≥ m*).

range(m, n, j) generates integers starting at *m*, in "steps" of *j*, stopping when the generated integer would be greater than or equal to *n* (if *j* is positive) or when the generated integer would be less than or equal to *n* (if *j* is negative).

Don't hesitate to use the two and three-argument forms when they clarify the code, but be aware of the pitfalls that may arise.