7.6. The Accumulator Pattern

One common programming "pattern" is to traverse a sequence, **accumulating** a value as we go, such as the sum-so-far or the maximum-so-far. That way, at the end of the traversal we have accumulated a single value, such as the sum total of all the items or the largest item.

The anatomy of the accumulation pattern includes:

- **initializing** an "accumulator" variable to an initial value (such as 0 if accumulating a sum)
- **iterating** (e.g., traversing the items in a sequence)
- **updating** the accumulator variable on each iteration (i.e., when processing each item in the sequence)

For example, consider the following code, which computes the sum of the numbers in a list.

Save & Run 4/30/2021, 8:48:54 PM - 2 of 2 Show in CodeLens

```
1 nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2 accum = 0
3 for w in nums:
4     accum = accum + w
5 print(accum)
6
```

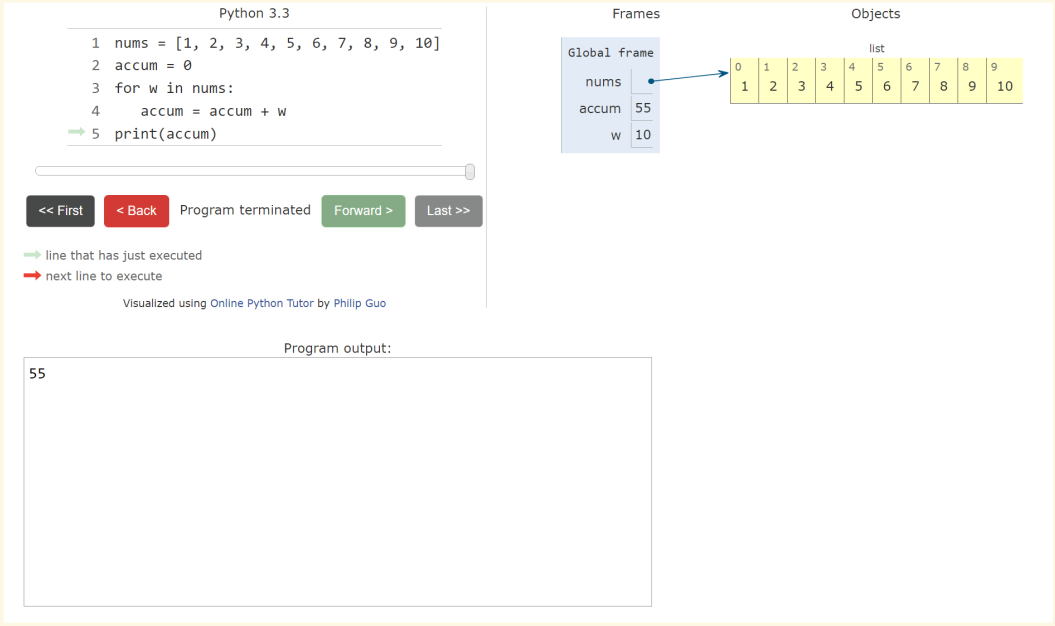
```
55
```

Activity: 1 -- ActiveCode (ac6_6_1)

In the program above, notice that the variable `accum` starts out with a value of 0. Next, the iteration is performed 10 times. Inside the for loop, the update occurs. `w` has the value of current item (1 the first time, then 2, then 3, etc.). `accum` is reassigned a new value which is the old value plus the current value of `w` .

This pattern of iterating the updating of a variable is commonly referred to as the **accumulator pattern**. We refer to the variable as the **accumulator**. This pattern will come up over and over again. Remember that the key to making it work successfully is to be sure to initialize the variable before you start the iteration. Once inside the iteration, it is required that you update the accumulator.

Here is the same program in codelens. Step through the function and watch the "running total" accumulate the result.



Activity: 2 -- CodeLens: (clens6_6_1)

Note

What would happen if we indented the print accum statement? Not sure? Make a prediction, then try it and find out.

We can utilize the range function in this situation as well. Previously, you've seen it used when we wanted to draw in turtle. There we used it to iterate a certain number of times. We can do more than that though. The `range` function takes at least one input - which should be an integer - and returns a list as long as your input. While you can provide two inputs, we will focus on using range with just one input. With one input, range will start at zero and go up to - but not include - the input. Here are the examples:

Save & Run 4/30/2021, 8:49:01 PM - 2 of 2 Show in CodeLens

```
print("range(5): ")
for i in range(5):
    print(i)

print("range(0,5): ")
for i in range(0, 5):
    print(i)

# Notice the casting of `range` to the `list`
print(list(range(5)))
print(list(range(0,5)))

# Note: `range` function is already casted as `list` in the textbook
print(range(5))

```

```
range(5):
0
1
2
3
4
range(0,5):
0
1
2
3
4
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
```

Activity: 3 -- ActiveCode (ac6_8_10)

One important thing to know about the range function in python3 is that if we want to use it outside of iteration, we have to cast it as a list using `list()`. Inside the textbook you'll notice that `range` works with or without casting it as a list but it is best for you to try and get into the habit of casting it as a list. Here's how you could use the range function in the previous problem.

Save & Run 4/30/2021, 8:49:04 PM - 2 of 2 Show in CodeLens

```
accum = 0
for w in range(11):
    accum = accum + w
print(accum)

# or, if you use two inputs for the range function

sec_accum = 0
for w in range(1,11):
    sec_accum = sec_accum + w
print(sec_accum)

```

```
55
55
```

Activity: 4 -- ActiveCode (ac6_6_2)

Because the range function is exclusive of the ending number, we have to use 11 as the function input.

We can use the accumulation pattern is count the number of something or to sum up a total. The above examples only covered how to get the sum for a list, but we can also count how many items are in the list if we wanted to.

Save & Run 4/30/2021, 8:49:06 PM - 2 of 2 Show in CodeLens

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
count = 0
for w in nums:
    count = count + 1
print(count)

```

```
10
```

Activity: 5 -- ActiveCode (ac6_6_3)

In this example we don't make use of `w` even though the iterator variable (loop variable) is a necessary part of constructing a for loop. Instead of adding the value of `w` to `count` we add a 1 to it, because we're incrementing the value of count when we iterate each time through the loop. Though in this scenario we could have used the `len` function, there are other cases later on where len won't be useful but we will still need to count.

Check your understanding

iter-6-1: Consider the following code:

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for w in nums:
    accum = 0
    accum = accum + w
print(accum)
```

What happens if you put the initialization of accum inside the for loop as the first instruction in the loop?

- A. It will print out 10 instead of 55
- B. It will cause a run-time error
- C. It will print out 0 instead of 55

Check me | Compare me

✔ The variable accum will be reset to 0 each time through the loop. Then it will add the current item. Only the last item will count.

Activity: 6 -- Multiple Choice (question6_6_1)

iter-6-2: Rearrange the code statements so that the program will add up the first n odd numbers where n is provided by the user.

Drag from here *Drop blocks here*

```
n = int(input('How many odd numbers would you like to add together?'))
thesum = 0
oddnumber = 1
for counter in range(n):
    thesum = thesum + oddnumber
    oddnumber = oddnumber + 2
print(thesum)
```

Check | Reset

Perfect! It took you only one try to solve this. Great job!

Activity: 7 -- Parsons (pp6_6_1)

Write code to create a list of integers from 0 through 52 and assign that list to the variable `numbers`. You should use a special Python function – do not type out the whole list yourself. HINT: You can do this in one line of code!

Save & Run 4/30/2021, 8:51:29 PM - 4 of 4 Show in CodeLens

```
1 numbers = range(53)
2
```

Activity: 8 -- ActiveCode (ac6_6_4)

Result	Actual Value	Expected Value	Notes
Pass	[0, 1..., 52]	[0, 1..., 52]	Testing that numbers is a list that contains the correct elements.

Expand Differences

You passed: 100.0% of the tests

Count the number of characters in string `str1`. Do not use `len()`. Save the number in variable `numbs`.

Save & Run 5/13/2021, 6:42:31 PM - 22 of 22 Show in CodeLens

```
str1 = "I like nonsense, it wakes up the brain cells. Fantasy is a necessary ingred
numbs=0
for i in str1:
    numbs+=1
print(numbs)
```

```
90
```

Activity: 9 -- ActiveCode (ac6_6_10)

Result	Actual Value	Expected Value	Notes
Pass	90	90	Testing that numbs is assigned to correct values.
Pass	'len('	'str1 ...umbs)'	Testing your code (Don't worry about actual and expected values).

Expand Differences

You passed: 100.0% of the tests

Create a list of numbers 0 through 40 and assign this list to the variable `numbers`. Then, accumulate the total of the list's values and assign that sum to the variable `sum1`.

Save & Run 5/13/2021, 6:44:28 PM - 6 of 6 Show in CodeLens

```
numbers = range(41)
sum1 = 0
for w in numbers:
    sum1 = sum1 + w
print(sum1)

```

```
820
```

Activity: 10 -- ActiveCode (ac6_8_9)

Result	Actual Value	Expected Value	Notes
Pass	[0, 1..., 40]	[0, 1..., 40]	Testing that numbers is assigned to correct values.
Pass	820	820	Testing that sum1 has the correct value.

Expand Differences

You passed: 100.0% of the tests

You have attempted 11 of 10 activities on this page

7.5. Lists and for loops">
7.5. Lists and for loops">
7.7. Traversal and the for Loop: By Index">
Completed. Well D
7.7. Traversal and the for Loop: By Index">Next Section - 7.7. Traversal and the for Loop: By Index

| Back to top