23.2. Map

You previously were introduced to accumulating a list by transforming each of the elements. Here we revisit that pattern.

The following function produces a new list with each item in the original list doubled. It is an example of a mapping, from the original list to a new list of the same length, where each element is doubled.

Save & Run Original - 1 of 1 Show in CodeLens

```
1 def doubleStuff(a_list):
2     """ Return a new list in which contains doubles of the elements in a_list. """
3     new_list = []
4     for value in a_list:
5         new_elem = 2 * value
6         new_list.append(new_elem)
7     return new_list
8
9 things = [2, 5, 9]
10 print(things)
11 things = doubleStuff(things)
12 print(things)
13
```

```
[2, 5, 9]
[4, 10, 18]
```

Activity: 1 -- ActiveCode (ac21_2_1)

The doubleStuff function is an example of the accumulator pattern, in particular the mapping pattern. On line 3, `new_list` is initialized. On line 5, the doubled value for the current item is produced and on line 6 it is appended to the list we're accumulating. Line 7 executes after we've processed all the items in the original list: it returns the `new_list`. Once again, codelens helps us to see the actual references and objects as they are passed and returned.

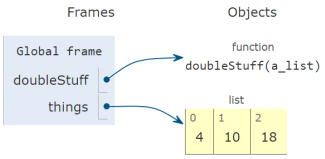
Python 3.3

```
1  def doubleStuff(a_list):
2      """ Return a new list in which contains doubles of the elements in a_list. """
3      new_list = []
4      for value in a_list:
5          new_elem = 2 * value
6          new_list.append(new_elem)
7      return new_list
8
9  things = [2, 5, 9]
10 things = doubleStuff(things)
```

<< First < Back Program terminated Forward > Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo



Program output:

Activity: 2 -- CodeLens: (clens21_2_1)

This pattern of computation is so common that python offers a more general way to do mappings, the `map` function, that makes it more clear what the overall structure of the computation is. `map` takes two arguments, a function and a sequence. The function is the mapper that transforms items. It is automatically applied to each item in the sequence. You don't have to initialize an accumulator or iterate with a for loop at all.

Note

Technically, in a proper Python 3 interpreter, the `map` function produces an "iterator", which is like a list but produces the items as they are needed. Most places in Python where you can use a list (e.g., in a for loop) you can use an "iterator" as if it was actually a list. So you probably won't ever notice the difference. If you ever really need a list, you can explicitly turn the output of map into a list: `list(map(...))`. In the runestone environment, `map` actually returns a real list, but to make this code compatible with a full python environment, we always convert it to a list.

As we did when passing a function as a parameter to the `sorted` function, we can specify a function to pass to `map` either by referring to a function by name, or by providing a lambda expression.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
def triple(value):
    return 3*value

def tripleStuff(a_list):
    new_seq = map(triple, a_list)
    return list(new_seq)

def quadrupleStuff(a_list):
    new_seq = map(lambda value: 4*value, a_list)
    return list(new_seq)

things = [2, 5, 9]
things3 = tripleStuff(things)
print(things3)
things4 = quadrupleStuff(things)
```

```
[6, 15, 27]
[8, 20, 36]
```

Activity: 3 -- ActiveCode (ac21_2_2)

Of course, once we get used to using the `map` function, it's no longer necessary to define functions like `tripleStuff` and `quadrupleStuff`.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
things = [2, 5, 9]

things4 = map((lambda value: 4*value), things)
print(list(things4))

# or all on one line
print(list(map((lambda value: 5*value), [1, 2, 3])))

```

```
[8, 20, 36]
[5, 10, 15]
```

Activity: 4 -- ActiveCode (ac21_2_3)

Check Your Understanding

1. Using map, create a list assigned to the variable `greeting_doubled` that doubles each element in the list `lst`.

Save & Run | 5/14/2021, 10:55:57 PM - 18 of 18 | Show in CodeLens

```

lst = [["hi", "bye"], "hello", "goodbye", [9, 2], 4]
greeting_doubled = []
def double(value):
    return 2*value

greeting_doubled = map(double, lst)

print(greeting_doubled)

```

```
[['hi', 'bye', 'hi', 'bye'], 'hellohello', 'goodbyegoodbye', [9, 2, 9, 2], 8]
```

Activity: 5 -- ActiveCode (ac21_2_4)

Result	Actual Value	Expected Value	Notes	
Pass	[['hi...], 8]	[['hi...], 8]	Testing that greeting_doubled is assigned to correct values	Expand Differences
Pass	'map('	'\nlst ...led)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'filter('	'\nlst ...led)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'sum('	'\nlst ...led)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'zip('	'\nlst ...led)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences

You passed: 100.0% of the tests

2. Below, we have provided a list of strings called `abbrevs`. Use map to produce a new list called `abbrevs_upper` that contains all the same strings in upper case.

Save & Run 5/14/2021, 10:40:40 PM - 2 of 2 Show in CodeLens

```
1
2 abbrevs = ["usa", "esp", "chn", "jpn", "mex", "can", "rus", "rsa", "jam"]
3
4 def f(st):
5     return st.upper()
6 abbrevs_upper = map(f, abbrevs)
7 print(abbrevs_upper)
8
```

```
['USA', 'ESP', 'CHN', 'JPN', 'MEX', 'CAN', 'RUS', 'RSA', 'JAM']
```

Activity: 6 -- ActiveCode (ac21_2_5)

Result	Actual Value	Expected Value	Notes	
Pass	['USA....JAM']	['USA....JAM']	Testing that abbrevs_upper is correct.	Expand Differences
Pass	'map('	'\nabbr...per)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'filter('	'\nabbr...per)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'sum('	'\nabbr...per)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'zip('	'\nabbr...per)\n'	Testing your code (Don't worry about actual and expected values).	Expand Differences

You passed: 100.0% of the tests

You have attempted 7 of 6 activities on this page

23.1. Introduction: Map, Filter, List Comprehensions, and Zip">
troduction: Map, Filter, List Comprehensions, and Zip">
✔ Completed. Well Done!
23.3. Filter">
23.3. Filter">Next Section - 23.3. Filter

| Back to top