Python Programming

More Functions Homework

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Consider

- For simplicity in all this homework assume the following:
  - Iterables are list, tuple, set or dict
  - Their values aren’t None
- Constraints:
  - Don’t use len function
  - Pass functions as lambda functions when convenient

Problem #1: Filter (easy)

- This function takes a function (filter) and an iterable. It returns a **list** of the filtered list
- Below, we try it with **is_even** function to remove odd numbers
- Replace is_even with lambda

```
4   def myfilter(func, iterable):...
11
12
13  def is_even(n):
14      return n % 2 == 0
15
16
17  res = myfilter(is_even, [1, 2, 3, 4, 5, 6, 10, 13])
18
19  print(res)  # [2, 4, 6, 10]
20
```

Problem #2: Reduce v1

- Reduce function reduces a complete iterable to a single value by applying a given function. So with add function it adds all of them. With multiply, it multiplies all of them. The function takes 2 arguments always
- E.g. for sum(a, b) and [2, 5, 6, 7, 8]
  - Sum 2 + 5 ⇒ 7
  - Sum 7 + 6 = 13
  - Sum 13 + 7 = 20
  - Sum 20 + 8 = 20
- Try it with some lambda for:
  - sum, multiply, max, min

```
 3    def myreduce(func, iterable, init = None):...
15
16    print(myreduce(max, {7, 20, 10}))   # 20
```

Problem #2: Reduce v1

- Init value: the default value is None
- If init is given, think of it as a first element in your iterable
- In case of **empty** iterable:
  - If init is not None, return it
  - Otherwise raise type error with msg **reduce of empty sequence with no initial value**

Problem #3: Reduce v2

- **def** myreduce(func1_overall, func2_consecutive, iterable)
- In this version we are given 2 functions:
  - Func2_consecutive is applied on every 2 consecutive numbers
  - Func1_overall is applied on their results (similar logic to last function)
  - Both of them takes 2 arguments
- Assume Func1 is multiplication and Func2 is addition
  - Input: [2, 5, 3, 4, 5, 10] ⇒ (2 + 5) * (3 + 4) * (5 + 10) = 735
  - So divide to pairs, apply func2, and apply func1 over all of them
- Don’t use len functions
- If len < 2: raise error: *The length of the sequence must be at least 2*
- If len is not even: *The length of the sequence must be even*

Problem #4: Map

- This function receives a function and a variable number of iterables
  - Say 5 iterables, then the passed function must receive 5 arguments
- map picks the top element from each iterable, pass them to the function and append the result to a list. It then picks the next top elements. It stops at the shortest length among iterables
- Note: Its code should be a **single short line**

```
4   def mymap(func, *iterables):...
6
7   def multi_abs(a, b, c):
8       return abs(a) * abs(b) * abs(c)
9
10  res = mymap(multi_abs, [1, -2, 3, 2], [-4, 5, 6, 7], [4, -5, -10, 9, 11])
11
12  print(res)  # [16, 50, 180, 126]
13
```

Problem #5: Nested Lambda

- Nested *if, loop, function, class and even lambda* are no new syntax rather than utilizing what we learned
- First, develop a function that takes values for a range:
  - **def** ff(st, en, step)
  - The function returns a closure that receives argument function f to apply on all the range and return result as list. For example for a square function and range(2, 6, 1)
  - processor = ff(2, 6, 1)
  - print(processor(sq)) ⇒ [4, 9, 16, 25]
- Second, rewrite the above function as a nested lambda. Same usage

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."