Lambda and Built-In Functions

When it comes to Lambdas, some people call them anonymous functions in other languages like JavaScript.

The idea of Lambdas is that it is a procedure that can be run with a short, one line only.

Here is a normal function created based on what we have already learned so far:

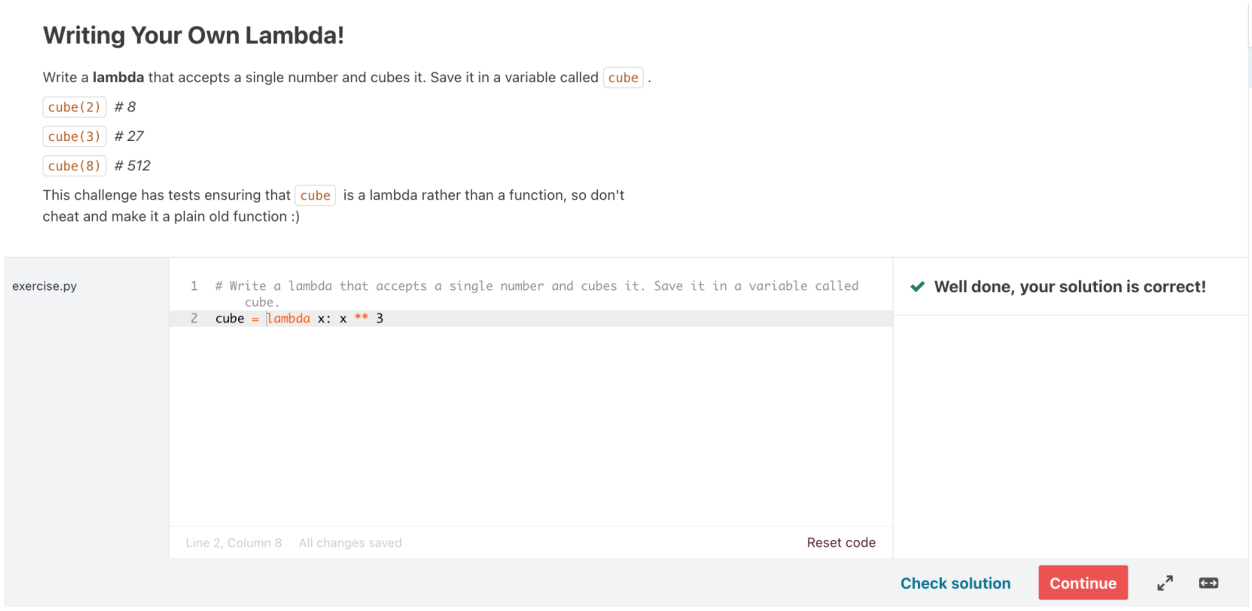
```
def square(num):
        return num * num
```

The lambda version of the function above would look like this:

```
square2 = lambda num: num * num
```

- Lambda functions begin with the word *lambda*.
- The word colored in orange indicates the parameter of the function. There can be multiple parameters as well within a lambda function, but for the simplicity of this example, there is only one.
- Following the parameters, there is a single expression which is indicated by the red colored words.

<u>Lambda Syntax:</u>
Lambda parameters: body of function

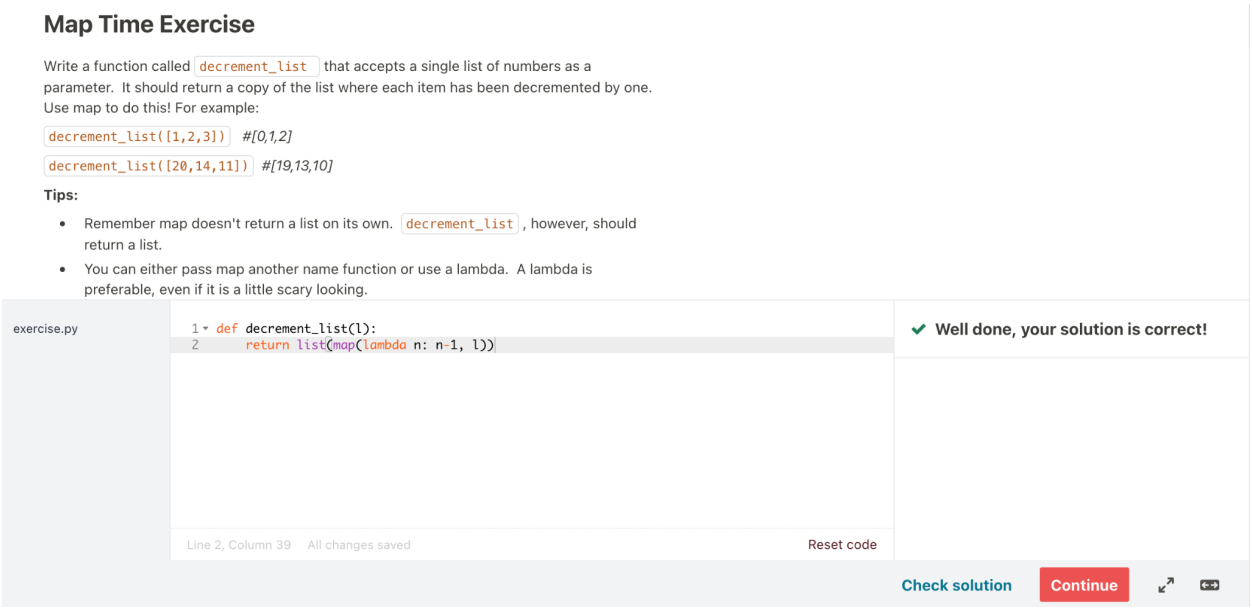


Map:

- A standard function that accepts at least two arguments, a function and an "iterable"
- iterable—something that can be iterated over (lists, strings, dictionaries, sets, tuples)
- runs the lambda for each value in the iterable and returns a map object which can be converted into another data structure

i.e.
nums = [2, 4, 6, 8, 10]
doubles = map(lambda x: x * 2, nums)

Within that example, the map takes the lambda function and the nums list. Then, it'll take every item within nums and run it within the lambda function.



Filter:

- There is a lambda for each value in the iterable
- Returns filter object which can be converted into other iterables
- The object contains only the values that return true to the lambda

l = [1, 2, 3, 4]
evens = list(filter(lambda x: x % 2 == 0, l))
print(evens) #[2, 4]

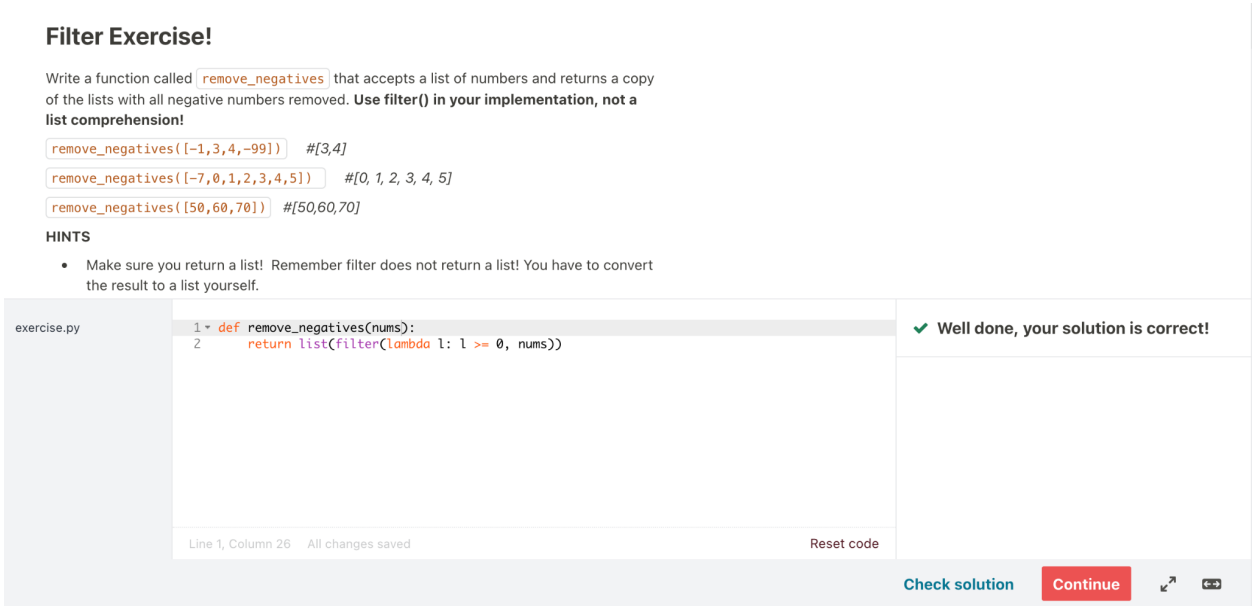
Combining *filter* and *map*:
Given this list of names:

names = [‘Lassie’, ‘Colt’, ‘Rusty’]

Return a new list with the string “Your instructor is “ + each value in the array, but only if the value is less than 5 characters.

list(map(lambda name: f”Your instructor is {name}”, filter(lambda value: len(value) < 5, names)))

What about list comprehension?



Built-in Functions:

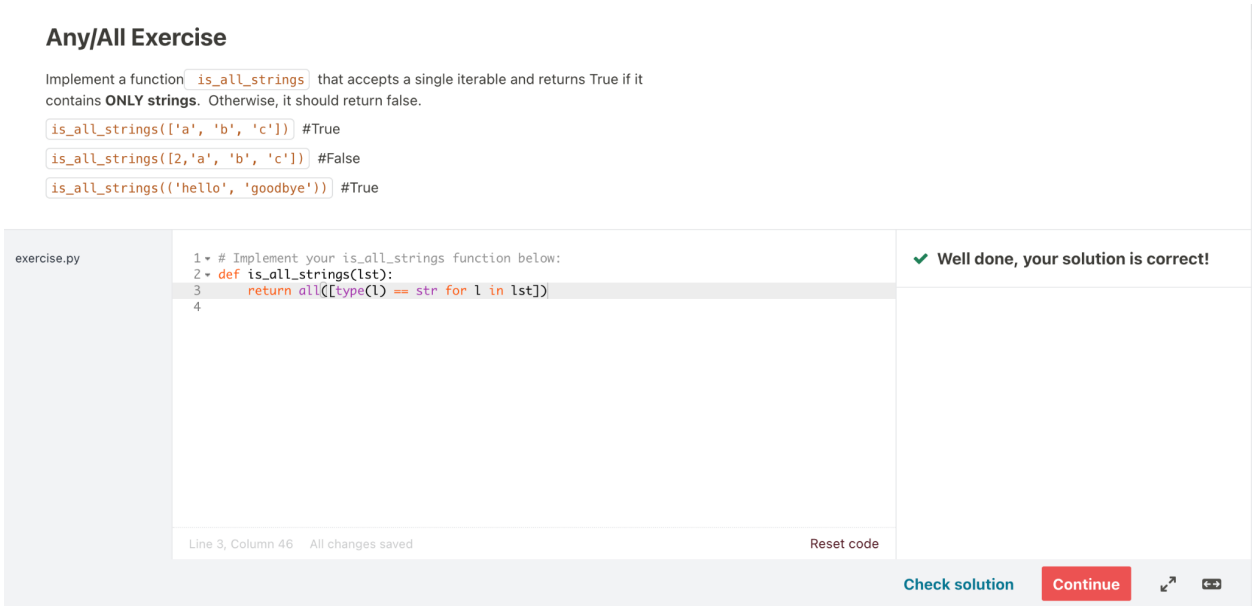
- **all** — returns True if all elements of the iterable are truthy (or if the iterable is empty)

i.e. nums = [2, 60, 26, 18]
all([num % 2 == 0 for num in nums]) # returns True

- **any** — return True if any element of the iterable is truthy. If the iterable is empty, return False.

i.e. nums = [2, 60, 26, 28, 21]
any([num % 2 == 1 for num in nums]) # returns True

Generator Expression and Using sys.getsizeof:
https://stackoverflow.com/questions/47789/generator-expressions-vs-list-comprehensions



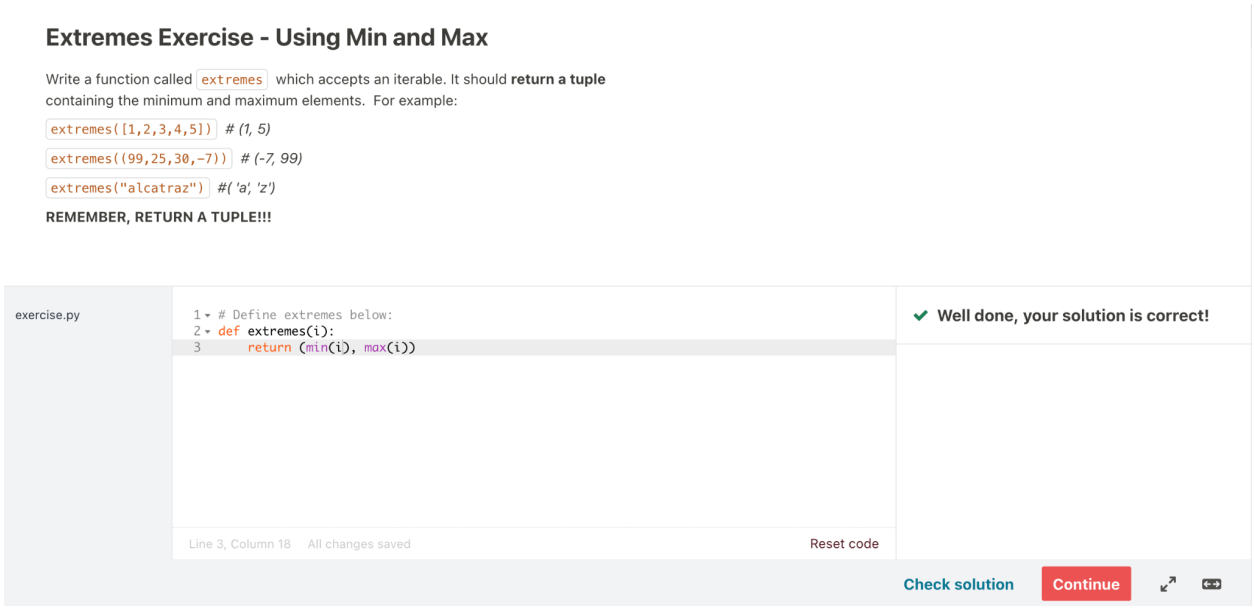
- Sorted — returns a new sorted list from the items in iterable

i.e. nums = [4, 6, 1, 30, 55, 23]
sorted(nums) # returns [1, 4, 6, 23, 30, 55]
sorted(nums, reverse=True) #returns [55, 30, 23, 6, 4, 1]

- Max — return the largest item in an iterable or the largest of two or more arguments
- Min — returns the smallest item in an iterable or the smallest of two or more arguments

i.e. names = [‘Arya’, ‘Samson’, ‘Dora’, ‘Tim’, ‘Ollivander’]
min(len(name) for name in names) # returns 3

max(names,key=lambda n: len(n)) # returns ‘Ollivander’
min(names,key=lambda n: len(n)) # returns ‘Tim’



- Reversed — return a reverse iterator
    - The difference with reversed from list.reverse() is that the list method will reverse the list in place and it works only with lists, whereas the reversed built-in function will return a reverse iterator
- Len — return the length (number of items of an object. The argument may be a sequence (such as string, tuple, list, or range) or a collection (such as a dictionary, set)
    - In a sense, it is calling the __len__() dunder method and acting like an adapter to call the specific method on whatever the user is trying to find the length of.
- Abs — return the absolute value of a number. The argument may be an integer or a floating point number.
    - The official definition: the magnitude of a real number without regard to its sign
- Sum — takes an iterable and an optional start
    - Returns the sum of start and the items of an iterable from left to right and returns the total
    - Starts defaults to 0
- Round — return number rounded to ndigits precision after the decimal point. If ndigits is omitted or is None, it returns the nearest integer to its input

Greatest Magnitude Exercise

Write a function `max_magnitude` that accepts a single list full of numbers. It should return the magnitude of the number with the largest magnitude (the number that is furthest away from zero).

`max_magnitude([300, 20, -900])` *#900*

`max_magnitude([10, 11, 12])` *#12*

`max_magnitude([-5, -1, -89])` *#89*

Hint: use max and abs!

exercise.py

```
1 def max_magnitude(nums):
2     return max(abs(num) for num in nums)
```

Line 2, Column 41 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

sum_even_values

Write a function called **sum_even_values**. This function should accept a variable number of arguments and return the sum of all the arguments that are divisible by 2. If there are no numbers divisible by 2, the function should return 0. To be clear, it accepts all the numbers as individual arguments!

```
1 sum_even_values(1,2,3,4,5,6) # 12
2 sum_even_values(4,2,1,10) # 16
3 sum_even_values(1) # 0
```

exercise.py

```
1 '''
2 sum_even_values(1,2,3,4,5,6) # 12
3 sum_even_values(4,2,1,10) # 16
4 sum_even_values(1) # 0
5 '''
6
7 # define sum_even_values
8 def sum_even_values(*nums):
9     return sum(num for num in nums if num % 2 == 0)
```

Line 9, Column 51 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

sum_floats

Write a function called **sum_floats.** This function should accept a variable number of arguments. The function should return the sum of all the parameters that are floats. If there are no floats the function should return 0

exercise.py

```
'''
sum_floats(1.5, 2.4, 'awesome', [], 1) # 3.9
sum_floats(1,2,3,4,5) # 0
'''

def sum_floats(*args):
    return sum(arg for arg in args if type(arg) == float)
```

Line 7, Column 57 All changes saved Reset code

✔ **Well done, your solution is correct!**

Check solution Continue