chapter_0_part1

July 25, 2021

0.1 Version Information

We strongly recommend you to use Python 3 throughout this book. Versions of Python 2.x, although they are deprecated, may work in this chapter since most of the exercises rely on built-in functions. We will show you to check your version number below.

[4]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)
```

[5]:
```
# Alternatively, you can just print the Python version
print(sys.version)
```

```
3.8.8 (default, Feb 26 2021, 23:59:43)
[Clang 12.0.0 (clang-1200.0.32.29)]
```

0.2 Activity 1: Find the Maximum Element

You are given a `list` of numbers, `nums`. How would you write Python code to find the largest element in this list?

Note: in our solution, we initialize the `max_element` to `float("-inf")`. This just represents the lowest numerical value that can be stored in a Python variable.

0.2.1 Example Input

[6]:
```
nums1 = [10, 3, 4, 20, 12]
```

0.2.2 Solution 1: Using a `for` loop

[7]:
```
def find_max_1(nums):
    max_element = float("-inf")
    for element in nums:
        if element > max_element:
            max_element = element
    return max_element
```

Test Out Solution 1

[8]:
```
print(find_max_1(nums1))
```

20

0.2.3 Solution 2: Using `max()`

Say hello to our first built-in Python function!

`max()` will return the largest value in finds in any input `list`. Alternatively, you could also pass in several values to the function all at once, and achieve the same result.

i.e. `max([x, y, z])` gives the same result as calling `max(x, y, z)`.

Read more about `max()` in the Python 3 documentation.

Test Out Solution 2

[9]:
```
# does this output the same value as above?
print(max(nums1))
```

20

0.3 Activity 2: Index of Maximum Element

You are given a `list` of numbers, `nums`. How would you write Python code to find the index of the largest element in the list?

`Note`: if there are no elements in `nums`, then just return -1. Also, if the largest value appears more than once in `nums`, then return the index of where it first appears.

0.3.1 Example Input

The largest element here is 20, located at index position 3.

[10]:
```
nums2 = [10, 3, 4, 20, 12]
```

0.3.2 Solution 1: `for` loop using `range()`

[11]:
```
def find_max_index_1(nums):

    max_element = float("-inf")
    max_element_index = -1
    for index in range(len(nums)):
        element = nums[index]
        if element > max_element:
            max_element = element
            max_element_index = index
    return max_element_index
```

Test Out Solution 1

[12]:
```
print(find_max_index_1(nums2))
```

3

0.3.3 Solution 2: `for` loop using `enumerate()`

[13]:
```
def find_max_index_2(nums):
    max_element_index = -1
    max_element = float("-inf")
    for index, element in enumerate(nums):
        if element > max_element:
            max_element = element
            max_element_index = index
    return max_element_index
```

[14]:
```
print(find_max_index_2(nums2))
```

```
3
```

0.3.4 Solution 3: Using `max()` and `index()`

[15]:
```
def find_max_index_3(nums):
    # A: empty list
    if nums is None or len(nums) == 0:
        return -1
    # B: all other lists
    max_element = max(nums)
    max_element_index = nums.index(max_element)
    return max_element_index
```

Test Solution 3

[16]:
```
print(find_max_index_3(nums2))
```

```
3
```

0.4 Activity 3: Product of Array

You are given a `list` of multiple numbers, `nums`. How would you write Python code to compute the product of all the elements in `nums`?

0.4.1 Example Input

[17]:
```
nums3 = [10, 3, 4, 20, 12]
```

0.4.2 Solution 1: Using a `for` loop

[18]:
```
def product_of(nums):
    prod = 1
    for num in nums:
        prod = prod * num
    return prod
```

Test Out Solution 1

[19]:
```
print(product_of(nums3))
```

28800

0.4.3 Solution 2: Using `math.prod`

The `math` module is one of the most useful in the Python Standard Library, specifically for mathematical operations.

The `math.prod()` was introduced in Python 3.8, and it can also be used to accomplish this task. You may read more of the details about this function as you wish in the documentation.

[20]:
```
# A: first, import the math module
import math

# B: then ensure Python version is >=3.8
assert sys.version_info >= (3, 8), "Please use Python 3.8 or higher."

# C: if all is well, return the product
result = math.prod(nums3)
```

Test Out Solution 2

[21]:
```
print(result)
```

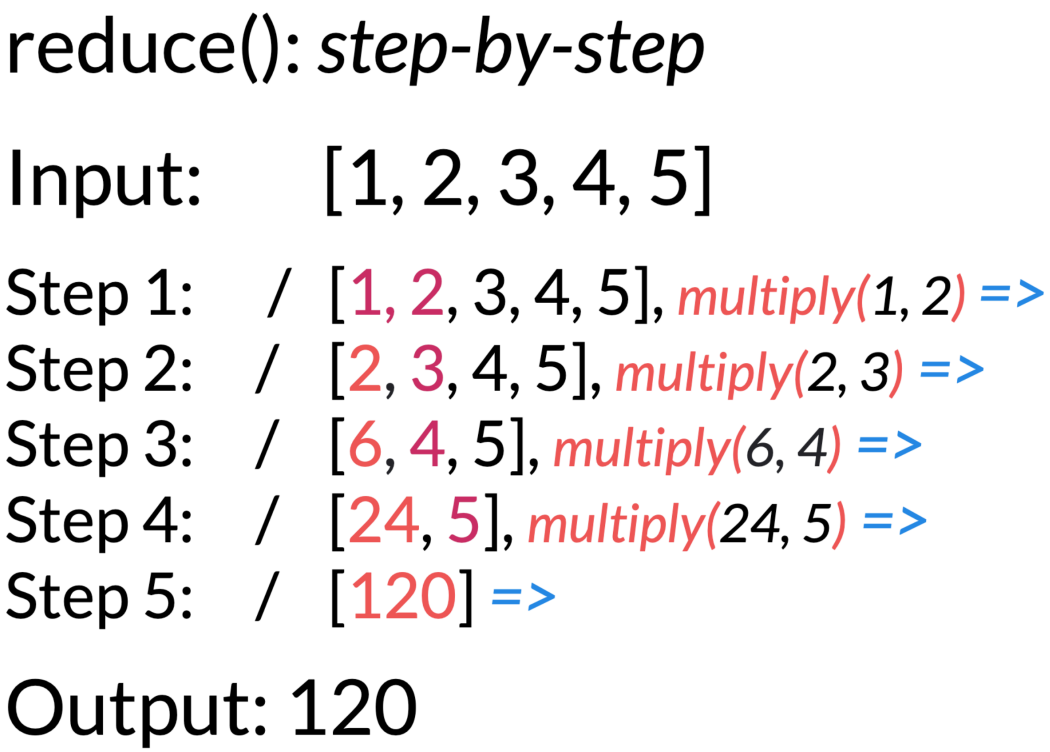
28800

0.4.4 Solution 3: Using `reduce()`

The `reduce()` function is another handy tool. This function is typically used to take a given array of numbers and decompose them down into a singular value. This is done by applying a given helper function repeatedly to the values in the array, over and over again, until they are *reduced* to a single value.

But how does `reduce()` do this, you may ask? Let's take an example to make it clearer: consider you have a Python `list` that contains `[1, 2, 3, 4, 5]`, and a function called `multiply()` that computes the product of two values. If we passed both of these to the `reduce()` function, it would return the product of all the elements, `120`. Essentially, it computes the result of `((((1*2)*3)*4)*5)`.

For your understanding, this process is visualized in the following diagram:



The Python code for this is implemented below:

[22]:
```
# A: first, import reduce()
from functools import reduce

# B: next, define the helper function
def multiply(num1, num2):
    return num1 * num2

# C: then, compute the product
product = reduce(multiply, nums3)
```

Test Out Solution 3

[23]:
```
print(product)
```

```
28800
```

0.5 Activity 4: Is in Range

Given a number `num`, how would you write Python code to check whether it falls in between a `lower_bound` number and an `upper_bound` number?

Note: for this activity, the range of values between `lower_bound` and `upper_bound` is **inclusive** of the `lower_bound`, and **non-inclusive** of the `upper_bound`.

0.5.1 Example Inputs

For example, if we are given `5` as our `num` and ask our function whether or not it falls in between a `lower_bound` of `3` and an `upper_bound` of `9`, then we would expect it to return `True`.

However, if we use the same `num` and `upper_bound`, but change the `lower_bound` from `3` to `7`, we would now expect the function to return `False`.

[24]:
```
# these tuples order the args according to (lower_bound, upper_bound, num)
true_example = (3, 9, 5)
false_example = (7, 9, 5)


def test_output(function, inputs):
    '''Displays whether the function works as expected for the given input.'''
    # A: unpack the input tuple into the corresponding args
    lower_bound, upper_bound, num = inputs
    # B: compute the result, use whatever function is passed in
    output = function(lower_bound, upper_bound, num)
    # C: display the result (Python 3.x required for format strings)
    print(f"{num} is in between {lower_bound} and {upper_bound}: {output}")
```

0.5.2 Solution 1: Using the built-in `range()` function

[25]:
```
def is_in_range_1(lower_bound, upper_bound, num):
    for element in range(lower_bound, upper_bound):
        if element == num:
            return True
    return False
```

Test Out Solution 1

[26]:
```
test_output(is_in_range_1, true_example)
test_output(is_in_range_1, false_example)
```

```
5 is in between 3 and 9: True
5 is in between 7 and 9: False
```

0.5.3 Solution 2: Using `lambda`

While solving this problem, you may have realized we don't necessarily need to iterate through all the numbers in the range `[lower_bound, upper_bound)`. Rather, we can merely use the comparision operators found in Python. With that in mind, your solution to this activity may have looked something like the following:

```
def is_in_range(lower_bound, upper_bound, num):
    # in the range
    if lower_bound <= num < upper_bound:
        return True
    # outside of the range
    return False
```

This solution works fantastically. Nonetheless, the question you may ask now is: why do we have to dedicate an entire function to doing something with such simple logic?

The creators of Python thought the same way, and that is part of why we have the `lambda` keyword. The `lambda` keyword is an especially useful tool for simplifying our code. It essentially allows us to define a function in a single line, and omit the `return` keyword as well. You may read more in the Python documentation as you wish.

[27]:
```
is_in_range_2 = lambda lb, ub, num: lb <= num < ub
```

Test Out Solution 2

[28]:
```
test_output(is_in_range_2, true_example)
test_output(is_in_range_2, false_example)
```

```
5 is in between 3 and 9: True
5 is in between 7 and 9: False
```

0.6 Activity 5: Element-Wise Sum

You are given two lists of numbers, `nums1` and `nums2`. How would you write Python code to compute their *element-wise sum*?

You may assume that both `nums1` and `nums2` are not empty, and contain the same number of elements.

0.6.1 Example Input

For example, if we are asked to compute the element-wise sum of `[1, 2, 3]` and `[10, 15, 20]`, then we would expect to eventually come up with `[11, 17, 23]`.

This is because in an element-wise sum, we add the elements from both arrays that are at corresponding indices, and place their sum in that same index in a new array.

For your understanding, this process is diagrammed below:

```
Index:        0   1   2
nums1:      [ 1,  2,  3]
nums2:  +   [10, 15, 20]

-----------------------
output:     [11, 17, 23]
```

[29]:
```
nums1 = [1, 2, 3]
nums2 = [10, 15, 20]
```

0.6.2 Solution 1: Using a `for` loop

Notice how we keep track of the `index` as we iterate over both arrays:

[30]:
```
def element_wise_sum_1(nums1, nums2):
  # A: init the output list
  summed_elems = [0 for _ in range(len(nums1))]
  # B: iterate over both input lists
```

```
  for index in range(len(nums1)):
    # C: place the next sum needed in the output list
    elem_sum = nums1[index] + nums2[index]
    summed_elems[index] = elem_sum
  # D: return the output
  return summed_elems
```

Test Out Solution 1

[31]:
```
print(element_wise_sum_1(nums1, nums2))
```

```
[11, 17, 23]
```

0.6.3 Solution 2: Using `zip()`

While the solution above works, it is also somewhat *too* straightforward.

To elaborate, what do you suppose might happen if we were instead asked to compute the element-wise sum of more than just 2 lists? How about 3? Or 10? Or 50?

In particular, line 7 of the `element_wise_sum_1` function may in fact end up looking something like the following:

```
elem_sum = nums1[index] + nums2[index] + nums3[index] + nums4[index] + nums5[index] + nums6[inc
```

Why is this so bad? In Data Science, it is not uncommon to have to manipulate many different arrays all at once. Therefore, if we have to keep track of an `index` variable ourselves, it can often create more room for errors in our code.

This is where Python's built-in `zip()` function can become immensely useful. For our purposes, it will basically combine our arrays into a single matrix - forming an array that is the same length as `nums1` or `nums2` before; and which contains an array for all the elements found at that index in the aforementioned `nums1` and `nums2`. Observe how this works in the diagram below:

```
How zip(nums1, nums2) Works:
index  |   nums1  |     num2   |        out
  0    |     1 -------> 10 --------->  (1, 10)
  1    |     2 -------> 15 --------->  (2, 15)
  2    |     3 -------> 20 --------->  (3, 20)
       |          |            |
```

Essentially, by using `zip()` we take out the need for tracking the `index` ourselves, and can thereby compute the element-wise sum using a more straightforward `for` loop, nested in a *list comprehension*.

[32]:
```
def element_wise_sum_2(nums1, nums2):
    '''Sums 2 different arrays element-wise, by first creating a 2D matrix.'''
    summed_elems = [
        sum(index_elems)  # summing two elements from the same index
        for index_elems in zip(nums1, nums2)  # iterating over the matrix rows
    ]
    return summed_elems
```

You may read more about `zip()` on the Python documentation as you please.

Test Out Solution 2

[33]:
```
print(element_wise_sum_2(nums1, nums2))
```

```
[11, 17, 23]
```

0.6.4 Solution 3: Using `map()`

As a final alternative, we can use another built-in function in Python called `map()`. It is yet another effective tool to make iterative problems easy - its function signature goes as follows:

`map(function, iterable, ...)`

In a nutshell, what `map()` does is apply the given `function` to every element in the given `iterable` (for our purposes, you can suppose this is our array of numbers), and produces the transformed elements in a new array.

But what does this have to do with our solution, you may ask?

The `map()` function is can also accept *multiple arrays* as inputs. It does this through *parallel iteration*, a process by which it will now apply the given `function` to *every element located at a given index* across all the arrays. Just like before, the `function` will then output its result into our output array.

Note: to make `map()` work, the input arrays **must** be of the same length; as well, the number of parameters for the `function` must be equal to the number of arrays passed in, so that all the inputs are, well... *mapped* properly to the outputs.

You may read more about the `map()` function in the Python documentation as you wish.

[34]:
```
# A: first, import the function we can use to add 2 elems
from operator import add

# B: iterate over the two arrays in parallel, to compute the element-wise sum
element_wise_sum_3 = list(map(add, nums1, nums2))
```

Test Out Solution 3

[35]:
```
print(element_wise_sum_3)
```

```
[11, 17, 23]
```

0.7 Activity 6: Squared Array

You are given an array of numbers, `nums`. How would implement a function in Python to compute the square of each number in `nums`?

Please return the ouput in a new array.

0.7.1 Example Input

[36]:
```
nums = [1, 2, 3]
```

0.7.2 Solution 1: Using a List Comprehension

[37]:
```
def square_elements(nums):
    return [num**2 for num in nums]
```

Test Out Solution 1

[38]:
```
print(square_elements(nums))
```

```
[1, 4, 9]
```

0.8 Activity 7: Modelling a Fair Die

Background:

In this exercise, we will ask you to compute the expected value of a fair, six-sided die when it is rolled multiple times over.

The *expected value of a random event*, in this case rolling a die, can be modelled using the following equation:

$E(X) = \sum_{i=1}^{6} i * P(dice = i)$

Where: - X = our random event (aka the *random variable*) - i = the value of the dice on a given roll - $P(dice = i)$ = the probability of the die landing with that value on top

Activities:

7.1: For a fair dice, can you

1. Compute the probability that when the die is rolled, its face value is 1 ($P(dice = 1)$)?
2. Compute the probability that when the die is rolled, its face value is 2 ($P(dice = 2)$)?
3. And so and so forth, for all the faces of the die?

7.2: Now, how would you compute $E(X)$ of the die, using the answer from the questions above?

0.8.1 Solution:

The expected value for a fair dice is:

$E(X) = (1 * 1/6) + (2 * 1/6) + (3 * 1/6) + (4 * 1/6) + (5 * 1/6) + (6 * 1/6)$

$ = E(X) = (21 / 6)$

$ = E(X) = 3.5$

[39]:
```
# We can show that E(X) is the mean of the following dice random variable
import numpy as np
# lets roll the dice 1000 times
dice = np.random.randint(low=1.0, high=7.0, size=1000)
print(dice)
# Compute the mean of dice list
print(np.mean(dice))
print(sum(dice)/len(dice))
```

```
[6 5 6 5 3 6 1 5 2 6 4 4 6 1 3 6 4 2 3 4 1 2 4 1 5 5 5 5 6 1 2 5 2 3 5 6 1
 1 6 6 4 3 4 6 5 4 3 4 4 2 6 4 1 2 3 6 5 3 2 3 1 5 4 1 4 3 4 4 4 3 3 6 4 5
 1 4 3 1 6 2 5 5 5 1 6 4 3 5 3 4 3 3 5 6 2 6 6 1 3 4 1 3 2 5 1 6 5 3 5 2 5
 2 6 5 1 4 2 5 6 2 2 1 2 5 6 4 3 1 6 4 5 1 3 6 6 3 5 3 1 4 5 5 4 4 3 3 1 6
 5 3 6 6 6 3 4 2 6 1 4 4 2 6 4 4 1 3 6 5 4 1 1 5 3 4 3 5 3 1 3 6 3 1 1 6 5
 2 6 3 4 5 2 3 5 2 5 3 2 4 4 2 1 6 6 6 4 2 6 1 5 6 4 2 3 5 6 4 1 1 1 6 3 5
 3 1 2 4 5 1 4 6 3 5 6 4 4 1 5 3 3 3 5 4 4 1 5 5 3 6 4 6 3 5 2 4 3 2 4 6 2
 2 1 6 6 6 4 3 4 2 2 1 5 5 3 4 6 3 3 3 4 4 5 6 5 6 1 5 3 2 5 6 4 6 5 4 4 6
 1 4 2 5 6 6 3 3 2 4 2 6 5 3 4 1 2 5 3 4 2 1 3 2 3 5 1 2 4 2 2 2 4 2 5 5 4
 6 2 5 1 2 1 6 6 5 5 2 5 5 1 3 3 5 4 4 5 6 1 6 2 6 1 6 2 3 2 5 5 5 1 1 1 2
 1 1 2 5 5 2 6 6 6 1 5 5 2 2 5 3 4 4 3 2 6 3 4 2 5 3 2 6 4 3 1 6 1 4 2 1 4
 3 4 1 1 6 5 3 4 3 1 4 3 6 2 3 2 1 6 4 4 5 1 3 4 4 5 5 5 1 2 1 6 6 4 6 1 4
```

```
 4 5 6 1 6 2 5 3 3 6 4 5 6 6 3 5 3 5 4 5 4 2 4 2 2 1 3 4 5 1 6 2 6 6 4 3 2
 4 1 2 5 3 2 5 4 6 3 5 1 1 2 1 4 1 6 2 2 5 2 2 6 3 2 5 1 1 4 4 2 4 5 4 1 2
 4 2 2 4 1 5 6 6 4 5 1 1 6 3 6 6 2 4 2 4 2 5 1 2 5 4 1 1 2 2 4 3 3 5 6 1 1
 2 4 4 2 3 1 6 1 5 4 1 2 4 4 4 1 4 5 1 1 3 4 2 6 5 4 4 1 6 4 1 4 3 4 2 4 2
 5 4 5 2 1 1 3 3 5 3 6 6 1 2 3 1 5 3 1 4 6 4 2 6 3 4 2 3 1 2 4 4 1 2 2 1 2
 1 3 1 5 3 5 3 4 6 2 3 6 4 3 2 5 4 6 3 2 3 3 4 2 1 5 2 4 5 3 6 4 1 6 3 4 5
 1 3 2 4 2 4 6 3 4 4 5 6 2 3 4 2 6 6 1 6 5 5 6 4 4 4 2 3 3 4 6 6 3 6 6 2 6
 2 2 3 5 5 5 3 1 1 6 3 6 2 1 5 6 1 3 1 2 3 6 1 4 3 5 5 4 4 4 6 4 1 4 6 2 4
 6 5 3 1 5 1 1 2 2 2 5 6 2 1 5 1 3 3 3 1 1 3 3 5 3 1 6 1 6 4 4 2 3 5 5 2 4
 4 4 3 2 6 2 2 4 3 3 5 3 5 5 5 2 4 1 2 5 3 3 3 4 4 2 3 5 1 5 3 4 2 4 6 3 6
 3 2 3 3 1 5 4 6 1 2 2 5 4 4 6 2 5 6 2 4 2 1 6 5 2 2 3 2 1 1 4 1 2 4 1 5 6
 1 1 6 3 5 6 2 5 4 4 4 6 6 1 4 3 3 3 6 5 1 5 4 1 5 4 4 3 6 3 5 4 4 4 3 1 3
 4 2 1 2 6 1 6 6 5 1 4 1 6 2 4 6 5 6 4 1 6 1 4 1 5 4 6 3 5 6 5 6 1 6 4 4 3
 2 5 6 1 4 3 5 2 2 1 1 1 3 3 2 4 1 3 6 6 6 3 1 1 2 2 2 2 3 5 5 3 1 6 3 5 5
 2 4 1 6 6 1 1 6 5 1 4 5 5 2 1 1 3 4 1 4 4 2 5 6 4 1 6 6 1 4 3 5 1 4 4 3 5
 6]
3.53
3.53
```

0.9 Activity 8: Compute the Mode

You are given a list of numbers of arbitrary length, `data`. How would you write a Python function to compute the *mode* of `data`?

The definition of the mode is the *most commonly occurring value* or feature across our data.

Hint: think about how you would solve this problem without code. You will probably need to keep track of the number of times each unique element appears in `data`. How might you use a *histogram* to help you achieve this?

0.9.1 Example Input

[40]:
```
data = [1, 3, 5, 2, 3, 7, 8, 4, 10, 0, 6, 7, 3, 0, 3, 0, 5, 7, 10, 1, 4, 9, 3]
```

0.9.2 Solution 1: Histogram via `dict()`

In this solution, we use Python's built-in `dict()`, a data type that allows us to map each unique value in the `data`, to its corresponding number of occurences. Thus, creating a histogram!

[41]:
```
def compute_mode_1(dataset):
  # A: init the dictionary
  histogram = {}
  # B: count the occurrences of each unique value
  for num in dataset:
      if num not in histogram:
          histogram[num] = 1
      else:
          histogram[num] += 1
  return (
```

```
    # C: find the highest count, then return the corresponding value
    max(histogram, key=histogram.get)
  )
```

Test Out Solution 1 In case you have any doubts about the answer, you can verify by modifying the `compute_mode_1` function to return the entire histogram itself, rather than the mode.

If everything is working correctly, then the `histogram` should show the following key-value pairs:

```
{
  1: 2,
  3: 5,  # <--- 3 is the mode!
  5: 2,
  2: 1,
  7: 3,
  8: 1,
  4: 2,
  10: 2,
  0: 3,
  6: 1,
  9: 1
}
```

Test Out Solution 1

[42]:
```
print(compute_mode_1(data))
```

```
3
```

0.9.3 Solution 2: Using a `Counter`

Using the `dict` type is to compute histograms in Python - so often in fact, that the creators of the language decided to also create the `Counter` type.

Simply put, `Counter` is just a child of the `dict` type with some extra properties. These properties simplify several operations that we typically perform on histograms.

For example, let's see how we can find the mode of `dataset` by instantiating a `Counter` object, and then using the `most_commmon()` method:

[43]:
```
from collections import Counter

def compute_mode_2(dataset):
  # A: compute the histogram
  histogram = Counter(dataset)
  # B: get the most frequently occurring value
  mode_value, mode_count = histogram.most_common(1)[0]
  return mode_value
```

For more on the properties of the `Counter`, please visit the Python documentation.

Test Out Solution 2

[44]:
```
print(compute_mode_2(data))
```

```
3
```

0.10 Activity 9: The Largest Elements

You are given a list of numbers, `nums`, and a nonnegative integer, `k`.

How would you write Python code to obtain the largest `k` elements from `nums`?

Note: your function does NOT have to return the largest `k` elements in sorted order.

0.10.1 Example Input

To wrap your head around technical questions like this, it often helps to just ignore all the verbiage - and create an analogy that helps you understand it clearly.

For example, consider the values in `nums` like the scores of different players on an old, arcade video game machine. And let `k` represent the top highest scores in this game. Now, how would you go about building a scoreboard for this machine (such as shown below) with the top `k` scores listed at the top?

Use this example input to guide your thinking:

```
Inputs:
nums = [5, 1, 3, 6, 8, 2, 4, 7]
k = 3

Output: [8, 7, 6]

Explanation:
The output comes from the first 3 values we find in the sorted version of the nums list: [8, 7
```

[45]:
```
nums = [5, 1, 3, 6, 8, 2, 4, 7]
k = 3
```

0.10.2 Solution 1: Using nested `for` loops

[46]:
```
def k_largest_1(nums, k):
    for first_index in range(len(nums)):
        # locate the (k + 1)-th largest element
        is_less_than = []
        for other_index in range(len(nums)):
          # if the first num < other num, then add a 1
          if nums[first_index] < nums[other_index]:
                is_less_than.append(1)
          # otherwise, add a 0
          else:
                is_less_than.append(0)
        # once found: return the num at every index that's > first num
        if sum(is_less_than) == k:
```

```
            return [
                nums[index] for index in range(len(is_less_than))
                if is_less_than[index] == 1
            ]
```

Test Out Solution 1

[47]:
```
print(k_largest_1(nums, k))
```

```
[6, 8, 7]
```

0.10.3 Solutions 2 and 3: Using Built-in Functions

[48]:
```
def k_largest_2(nums, k):
  '''This function mutates the nums list.'''
  nums.sort()
  return nums[-k:]


def k_largest_3(nums, k):
  '''This allocates a new list, so the original nums list in untouched.'''
  sorted_nums = sorted(nums)
  return sorted_nums[-k:]
```

Test Out Solutions 2 and 3 Try uncommenting the second line, to verify it prints the same output as the one above!

[49]:
```
print(k_largest_2(nums, k))
# print(k_largest_3(nums, k))
```

```
[6, 7, 8]
```

0.10.4 Solution 4: Using a `heapq`

Note that for larger values of `k`, it is more efficient to use the built-in sorting function.

Python documentation

[50]:
```
import heapq

def k_largest_4(nums, k):
  return heapq.nlargest(k, nums)
```

Test Out Solution 4

[51]:
```
print(k_largest_4(nums, k))
```

```
[8, 7, 6]
```

0.11 Activity 10: Identical Distributions

Given two arrays, determine if both arrays contain exactly the same elements in exactly the same amounts.

0.11.1 Example Inputs

1. Input `[1,2,5,4,0]` and `[4,2,5,0,1]`: Yes
2. Input `[1,7,1]` and `[7,7,1]`: No

0.11.2 Solution 1: Using `dict`

[52]:
```
def histogram(nums):
    dictionary = {}  # alias for calling dict()
    for num in nums:
        if num not in dictionary:
            dictionary[num] = 1
        else:
            dictionary[num] += 1
    return dictionary

print(histogram([3, 8, 4, 8]))
```

```
{3: 1, 8: 2, 4: 1}
```

[53]:
```
def same_distributions_1(nums1, nums2):
    # A: form histograms for both arrays
    dist1, dist2 = histogram(nums1), histogram(nums2)
    # B: count how many number TYPES are found in both histograms
    num_shared_types = 0
    for key in dist1.keys():
      if key in dist2:
        num_shared_types += 1
    # C: check if all the number types between the 2 histograms are equal
    if len(dist1) == len(dist2) == num_shared_types:
      # D: check if each number type appears in the same amount in both lists
      num_common_counts = 0
      for key in dist1.keys():
        if dist1[key] == dist2[key]:
          num_common_counts += 1
      # E: distributions are equal
      if num_shared_types == num_common_counts:
        return True
    # F: distributions are unequal
    return False
```

Test Out Solution 1

[54]:
```
print(same_distributions_1([1,2,5,4,0], [4,2,5,0,1]))
```

```
True
```

[55]:
```
print(same_distributions_1([1,7,1], [7,7,1]))
```

```
False
```

0.11.3 Solution 2: Using `Counter`

[56]:
```
from collections import Counter

def same_distribution_2(nums1, nums2):
  # A: make the distribution
  dist1 = Counter(nums1)
  dist2 = Counter(nums2)
  # B: see how much they overlap
  dist1.subtract(dist2)
  # C: return the answer
  remaining = list(dist1.elements())
  return remaining == []
```

Test Out Solution 2

[57]:
```
print(same_distribution_2([1,2,5,4,0], [4,2,5,0,1]))
```

```
True
```

[58]:
```
print(same_distribution_2([1,7,1], [7,7,1]))
```

```
False
```

0.12 Activity 11: Reverse a List

You are given a list, `ls`. How would you write Python code to reverse the elements in `ls`?

0.12.1 Example Input

[59]:
```
ls = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

0.12.2 Solution 1: Traversing the List in Reverse

[60]:
```
def reverse_list_1(ls):
    N = len(ls)
    reverse_ls = []
    for i in range(N):
        reverse_ls.append(ls[N - (i+1)])
    return reverse_ls
```

Test Out Solution 1

[61]:
```
print(reverse_list_1(ls))
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

0.12.3 Solution 2: Using List Splicing

List splicing in Python is a shorthand way of traversing a list, to retrieve a sublist. The syntax is to place `[start:stop:step_size]` after the name of the list you want to splice, where:

- `start` = the index where you want the traversal to begin. It defaults to index 0.
- `stop` = the index where you want the traversal to end. It defaults to the length of that list (because this argument is non-inclusive).
- `step` = is the number of index positions you would like the traversal to move by, before it lands on the next list element. It defaults to 1.

In the following solution, note how we are able to traverse the list by simply leaving the `start` and `stop` parameters use their default values, and we traverse the list backwards by setting `step` equal to -1. As with all list splices, this will create a new list object:

[62]:
```
def reverse_list_2(ls):
  return ls[::-1]
```

Test Out Solution 2

[63]:
```
print(reverse_list_2(ls))
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

0.12.4 Solutions 3 and 4: Using Built-in Functions

Just like with `list.sort()` and `sorted()`, the Python language comes with built-in functions that can reverse a list, either by mutating the original or creating a new one:

[64]:
```
def reverse_list_3(ls):
  ls.reverse()
  return ls


def reverse_list_4(ls):
  reverse_list = [elem for elem in reversed(ls)]
  return reverse_list
```

Test Out Solutions 3 and 4 Try uncommenting the second line, to verify it prints the same output as the one above!

[65]:
```
print(reverse_list_3(ls))
# print(reverse_list_4(ls))
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

0.13 Activity 12: Two-Sum

You are given a list of numbers, `nums`, and another integer, `target`. How would you implement Python code to find all the pairs of numbers found in `nums` that sum to `target`?

Please return a 2D list of the pairs you find.

You may assume that:

1. `nums` contains at least 2 integers.
2. `nums` is unsorted.
3. The ordering of the elements in each pair does not matter.

0.13.1 Example Input

[66]:
```
target = 7
nums = [3, 5, 2, -4, 8, 11]
```

0.13.2 Solution 1: Try Every Possible Pair

[67]:
```
def find_sum_pairs_1(target, nums):
    # A: init a list to hold pairs we find
    pairs = []
    for i in range(len(nums)):
        # B: try to combine this number with every other
        for j in range(i+1, len(nums)):
            num1, num2 = nums[i], nums[j]
            # C: form a pair, if possible
            if num1 + num2 == target:
                pairs.append([num1, num2])
    # D: return the pairs
    return pairs
```

Test Out Solution 1

[68]:
```
print(find_sum_pairs_1(target, nums))
```

```
[[5, 2], [-4, 11]]
```

0.13.3 Solution 2: Use a `set()` to Store Elements

[69]:
```
def find_sum_pairs_2(target, nums):
    # A: init a list to hold pairs we find
    pairs = []
    previously_seen = set()
    for num in nums:
      diff = target - num
      # B: form a pair if possible
      if diff in previously_seen:
        pairs.append([num, diff])
      # C: always add this number, and move on
      previously_seen.add(num)
    # D: return the pairs
    return pairs
```

Test Out Solution 2

[70]:
```
print(find_sum_pairs_2(target, nums))
```

```
[[2, 5], [11, -4]]
```

0.14 Activity 13: Largest Number of Patients

You are building a new monitoring system for a local hospital. The director of the hospital wants to know how busy the building gets throughout the day.

In this problem, you are given two lists of numbers, `start` and `end`, which represent the entering time and exit time of patients in the hospital on a certain day.

How would you write Python code to find the **maximum** number of patients present in that hospital on that day?

0.14.1 Example Input

As an example, consider the following lists as entering time and exit time for 4 patients:

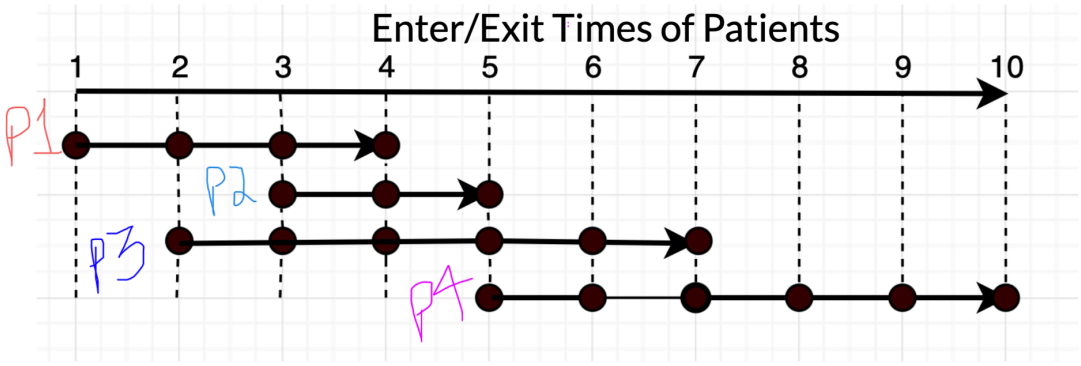
```
start = [1, 3, 2, 5]

end = [4, 5, 7, 10]
```

These two lists are ordered based on patients, i.e. the first elements represent the entering and exit time of Patient 1, the second elements are for Patient 2, and so on.

For simplicity, you may also assume that people only ever enter/exit the hospitals at the top of the hour, e.g. 1 o'clock, 2 o'clock, 3 o'clock, etc.

Although initially confusing, making visuals such as the following timeline can tremendously improve your understanding. As you can see, one way we can tell the maximum number of patients in the hospital is simply by looking for where the most "overlaps" occur between the stays of Patients 1, 2, 3, and 4 (note that their timelines are labeled in the diagram as "P1-P4", for brevity).



Therefore, as you can see above, the maximum number of patients for this day was 3. This happened

between 3-4, when Patients 1, 2, and 3 were all in the hospital.

[71]:
```
start = [1, 3, 2, 5]
end = [4, 5, 7, 10]
```

0.14.2 Solution 1: Naive Approach

[72]:
```
def find_max_num_patients_1(start, end, time_interval=1):
    # A: store the amount of patients in the hospital at all time intervals
    num_patients = []
    # B: find out how many patients were in, at each time interval
    for time in range(min(start), max(end) + 1, time_interval):
        # C: count up the num of patients at this time
        patient_count = 0
        for start_hr, end_hr in zip(start, end):
            # when our time falls within the stay of this patient, increment
            if time in range(start_hr, end_hr + 1, time_interval):
                patient_count += 1
        # D: store the total # of patients in the hospital at this time
        num_patients.append(patient_count)
    # E: return the global maximum for the # of patients, across time intervals
    return max(num_patients)
```

Test Out Solution 1

[73]:
```
print(find_max_num_patients_1(start, end))
```

```
3
```

0.14.3 Solution 2: Using the Net Change in Hospital Population

[74]:
```
from collections import Counter

def find_max_num_patients_2(start, end):
    # A: compute the total change in # of patients at each hr, using a dict
    pos_changes_in_num_patients = Counter(start)
    neg_changes_in_num_patients = Counter(end)
    # B: combine the pos and neg changes into 1 overall tally
    pos_changes_in_num_patients.subtract(neg_changes_in_num_patients)
    net_changes_in_num_patients = sorted(pos_changes_in_num_patients.items())
    # C: track the overall num_patients over the course of time
    max_patient_count = current_patients = 0
    for time, change in net_changes_in_num_patients:
      current_patients += change
      # D: update the max_num of patients as appropiate
      if current_patients > max_patient_count:
        max_patient_count = current_patients
    # E: does the answer come out the same as before
    return max_patient_count
```

Test Out Solution 2

[75]:
```
print(find_max_num_patients_2(start, end))
```

```
3
```

0.14.4 Solution 3: Using a List to Track Population Changes

[76]:
```
def find_max_num_patients_3(start, end, time_interval=1):
    # A: compute all the times when the num_patients can change
    earliest_enter = min(start)
    latest_exit = max(end)
    net_changes_in_num_patients = [
      0 for _ in range(earliest_enter, latest_exit + 1, time_interval)
    ]
    # B: combine the pos changes at each hr
    for pos_change in start:
      index = pos_change - earliest_enter
      net_changes_in_num_patients[index] += 1
    # C: do the same for all the decreases in num_patients
    for neg_change in end:
      index = neg_change - earliest_enter
      net_changes_in_num_patients[index] -= 1
    # D: track the overall num_patients over the course of time
    max_patient_count = current_patients = 0
    for change in net_changes_in_num_patients:
      current_patients += change
      # E: update the max_num of patients as appropiate
      if current_patients > max_patient_count:
        max_patient_count = current_patients
    # F: does the answer come out the same as before
    return max_patient_count
```

Test Out Solution 3

[77]:
```
print(find_max_num_patients_3(start, end))
```

```
3
```

0.15 Activity 14: Partition a List

You are given a list of numbers, `nums`, and are asked to divide it into a certain number of partitions, `n_parts`.

How would you implement Python code to perform this task?

0.15.1 Example Input

You may assume `n_parts` is a positive integer. For simplicity, don't worry about which elements go in which partition.

[78]:
```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
n_parts = 2
```

0.15.2 Solution 1: Using List Splicing

[79]:
```
def partition_1(nums, n_parts):
    # A: compute how long each part will be
    length_part = len(nums) // n_parts
    # B: partition the list
    return [
        nums[start:start+length_part]  # form each part using list splicing
        for start in range(0, len(nums), length_part)
    ]
```

Test Out Solution 1

[80]:
```
print(partition_1(nums, n_parts))
```

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]]
```

0.15.3 Solution 2: Using NumPy

[81]:
```
import numpy as np

def partition_2(nums, n_parts):
    # A: compute the length of each part
    length_part = len(nums) // n_parts
    # B: form the parts - aka create a matrix
    reshaped_array = np.reshape(nums, (n_parts, length_part))
    # C: convert from NumPy array to a list
    return [list(part) for part in reshaped_array]
```

Alternatively, you can also just let NumPy infer how many elements go in each partition of the list, simply by passing `-1` in place of `length_part` in step B above.

Test Out Solution 2

[82]:
```
print(partition_1(nums, n_parts))
```

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]]
```

0.16 Activity 15: Filter a Matrix

You are given a 2D array called `matrix`, and a number called `threshold`.

How would you implement Python code to filter out all the elements in `matrix` that are greater than the `threshold`?

By "filter", here's what we mean: - Return a `tuple` containing both the index of the row and column where the element was found; - Also, include the element itself; - *There's just one exception*: if the row and column indices of the element are the same, then don't return it. This essentially means we are ignoring the diagonal elements in the `matrix`.

Example Input

[83]:
```
matrix = [[1.   , 0.273, 0.195, 0.89 , 0.216, 0.134, 0.064, 0.944, 0.235],
       [0.273, 1.   , 0.236, 0.273, 0.143, 0.321, 0.084, 0.29 , 0.309],
       [0.195, 0.236, 1.   , 0.195, 0.97 , 0.14 , 0.133, 0.206, 0.123],
       [0.89 , 0.273, 0.195, 1.   , 0.272, 0.134, 0.064, 0.828, 0.235],
       [0.216, 0.143, 0.97 , 0.272, 1.   , 0.137, 0.13 , 0.229, 0.06 ],
       [0.134, 0.321, 0.14 , 0.134, 0.137, 1.   , 0.373, 0.142, 0.   ],
       [0.064, 0.084, 0.133, 0.064, 0.13 , 0.373, 1.   , 0.067, 0.305],
       [0.944, 0.29 , 0.206, 0.828, 0.229, 0.142, 0.067, 1.   , 0.249],
       [0.235, 0.309, 0.123, 0.235, 0.06 , 0.   , 0.305, 0.249, 1.   ]]

threshold = 0.5
```

0.16.1 Solution 1

[84]:
```
def index_cosine_matrix(matrix, threshold):
    for index_row, row in enumerate(matrix):
        for index_element, element in enumerate(row):
            if (element > threshold) and (index_element != index_row):
                print((element, index_row, index_element))
```

[85]:
```
index_cosine_matrix(matrix, threshold)
```

```
(0.89, 0, 3)
(0.944, 0, 7)
(0.97, 2, 4)
(0.89, 3, 0)
(0.828, 3, 7)
(0.97, 4, 2)
(0.944, 7, 0)
(0.828, 7, 3)
```

0.17 Activity 16: Compute a Histogram via NumPy

In this activity, we re-introduce you to computing histograms. Except this time, we'll utilize one of the most versatile third-party libraries in Python for mathematical operations: NumPy.

You are given the following two arguments:

1. `nums`: a list of numerical values
2. `number_of_bins`: this is the number of buckets the histogram should have.

Your function would return a Python dictionary (via the `dict` type or any of its subclasses). In the dictionary, please have the ith-interval for the keys, and how many of elements in `nums` are in that interval as the corresponding value.

You may use the following pseudocode to help you as you understand our solution, since we want you to focus on most of your energy at understanding how NumPy aids us in implementing the actual code:

```
# Pseudocode for Histogram:
```

```
# A: Obtain the minimum element of the list
# B: Obtain the maximum element of the list
# C: Obtain the interval length
# D: Count how many elements in the list is in first interval, how many elements in the lis
```

0.17.1 Mini-Lesson: How To Filter NumPy Arrays

4 Quick Takeaways About NumPy: 1. You can use greater than (i.e. `>` or `>=`) or less than (i.e. `<`, `<=`) operators directly on NumPy arrays (i.e. objects of type `np.array`), and it will create new arrays for you. 2. If you need to use more than 1 condition, then join them will operators such as `&` or `|`, rather than the Python keywords `and` or `or`. 3. As a result, the new array will contain `True` or `False` values, depending on whether the element in the corresponding index of the original array met the condition. 4. When using `np.sum` on an array of `boolean` values, NumPy will equate a `True` value as adding 1 to the sum, and `False` will add 0.

[86]:
```
a = np.arange(10)
print(f"Original NumPy array: {a}")
b = a > 5
print(f"The Result of Filtering: {b}")
print(f"The 'sum' value of the filtered array: {np.sum(b)}")
```

```
Original NumPy array: [0 1 2 3 4 5 6 7 8 9]
The Result of Filtering: [False False False False False False  True  True  True
True]
The 'sum' value of the filtered array: 4
```

0.17.2 The Solution, in 4 Steps

Step 1: Import Packages

[87]:
```
import pandas as pd
import numpy as np
```

Step 2: Load the Dataset In this solution, our URL points to a CSV module of the Titanic dataset as an example. This has been uploaded on our companion GitHub repository. Alternatively, you can also load this CSV module from a location on your local file system.

[88]:
```
# A: save the URL address of the dataset
URL = "https://raw.githubusercontent.com/UPstartDeveloper/
 ↪Python-Literacy-Project/\
main/chapter0-exercises/titanic.csv"

# B: load the dataset into a DataFrame
df = pd.read_csv(URL)
```

Step 3: Build the Histogram

[ ]:
```
def custom_histogram_distribution(data, number_of_bins):
    # A: Obtain the minimum element of the list
    min_ls = np.min(data)
```

```
    # B: Obtain the maximum element of the list
    max_ls = np.max(data)
    # C: Obtain the interval length, I
    I = (max_ls - min_ls) / number_of_bins
    # D: Count how many elements in the list belong to each interval
    histogram = dict()
    for ith_interval in range(number_of_bins):
        lower_bound = min_ls + (ith_interval * I)
        upper_bound = min_ls + ((ith_interval + 1) * I)
        # Filters the array + counts the # samples (similar to the above cell)
        num_samples = np.sum((data >= lower_bound) & (data <= upper_bound))
        histogram[ith_interval] = num_samples
    return histogram

print(custom_histogram_distribution(df['Age'].dropna().values, 16))
```

```
{0: 44, 1: 20, 2: 19, 3: 96, 4: 122, 5: 108, 6: 88, 7: 67, 8: 47, 9: 39, 10: 24,
11: 18, 12: 14, 13: 3, 14: 4, 15: 1}
```

[ ]:
```
print(custom_histogram_distribution(df['Age'].dropna().values, 5))
```

```
{0: 100, 1: 346, 2: 188, 3: 69, 4: 11}
```

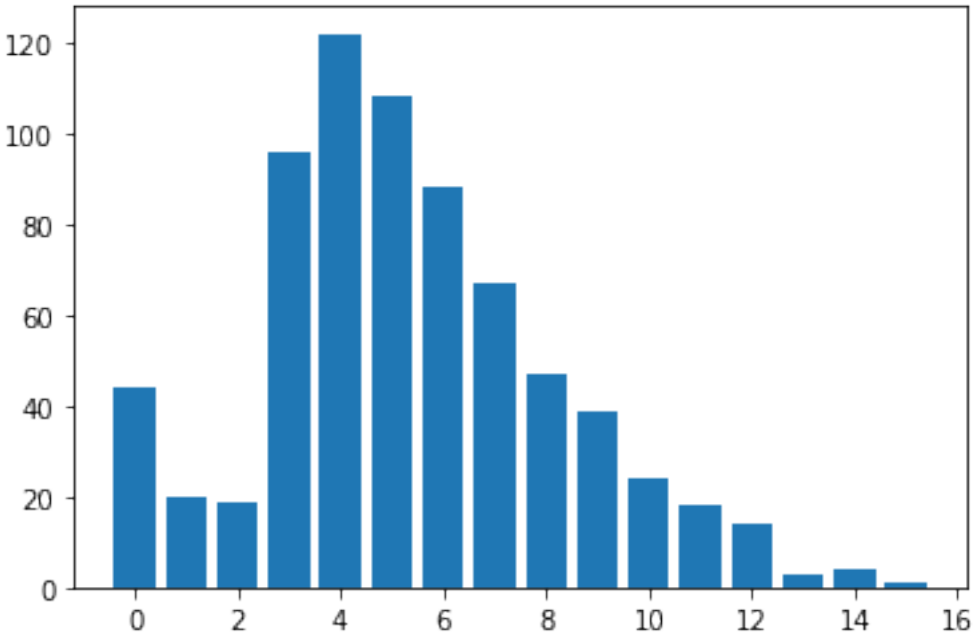
0.18 Challenge: Plot the Histogram!

Using the histogram function you created above, how would you implement Python code to visualize it, so that a friend may easily understand the data?

Hint: use the bar graph function from the `matplotlib` library!

[ ]:
```
# import packages
import matplotlib.pyplot as plt
# this magic function saves your plots directly under the respective cell
%matplotlib inline

# plot the data
hist_dict = custom_histogram_distribution(df['Age'].dropna().values, 16)
plt.bar(hist_dict.keys(), hist_dict.values())
plt.show()
```



To make sure it works, try another number of bins!

[ ]:
```
hist_dict = custom_histogram_distribution(df['Age'].dropna().values, 5)
plt.bar(hist_dict.keys(), hist_dict.values())
plt.show()
```