

LECTURE 24 — ADVANCED PYTHON TOPICS AND FUNCTIONAL PROGRAMMING

45.1 Problems We'd Like to Solve

Some of these are toy problems, but they illustrate use of tools we'd like to develop and use:

1. How many values are in a list of lists?
2. What is the maximum distance from the origin of the points in a list?
3. What is the sum of squares of the first n integers?
4. Can you sum the positive values in a list?
5. Can you sort a list of points by y value (2nd coordinate) and then by x value?

45.2 Solution Techniques

- We can solve most of these with a `for` loop, but they can be solved even more effectively / efficiently / compactly using advanced Python methods.
- Leads to notions of:
 - `map` and `filter`
 - functions as parameters
 - `lambda` functions
 - stable sort
 - list comprehensions
- Most are examples of *functional programming*

45.3 Map: Apply a function to each element of a list

- Suppose we want to count the number of values in a list of lists. We can use `map` to apply the `len` function to each sublist.

```
>>> v = [ [2, 3, 5, 7], [11,13,17,19], [23, 29], [31,37] ]
>>> print( list(map( len, v)) )
[4, 4, 2, 2]
```

- `map` is an *iterator* class:
 - It produces values in a sequence, one after another, by applying the function (1st argument) to the values of the second argument.
 - Technically, an iterator class is one that has the `__next__` method implemented (correctly).
 - Using `list` gives us the list of lengths of the sublists explicitly.
- To complete the solution we need to just apply `sum`:

```
>>> print sum(map(len,v))
12
```

Notice that this does not explicitly form an intermediate list.

45.4 Passing Functions as Parameters

- The above example passes the `len` function as an argument!
 - We also passed functions as arguments to our callbacks in our GUI programs
- This illustrates the concept that Python treats function as “first-class” objects - in other words functions can be used just like variables and other data.
 - What’s passed as an argument to `map()` is the location of the function code.
- Now suppose we want to find the maximum distance of a list of points from the origin. Here we’ll have to write a function

```
def dist2D( p ):
    return (p[0]**2 + p[1]**2)**0.5

pts = [ (4.5, 3), (2.1,-1), (6.8,-3), (1.4, 2.9) ]
print(max( map(dist2D,pts) ))
```

45.5 Lambda functions: Anonymous functions

- We can avoid the need to write a separate function here by writing an *anonymous* function called a *lambda* function.
- Our first example is just squaring the values of a list

```
>>> list(map( lambda x: x**2, [ 1, 2, 3, 4 ] ))
[ 1, 4, 9, 16 ]
```

- Now, we can sum the squares from 1 to n

```
>>> n = 100
>>> sum( map( lambda x: x**2, range(1,n+1)))
```

- We can also implement the `dist2D` function anonymously:

```
>>> max( map( lambda p: (p[0]**2 + p[1]**2)**0.5, pts) )
7.432361670424818
```

- Notice that we did not need to explicitly form a list in each of the preceding examples. This leads to substantial savings when the list is large!
- Aside: the notion of a lambda function goes all the way back to the origin of computer science

45.6 In-Class Practice Problem:

1. Starting with the following list of x,y point coordinate types, we will use `map()`, a lambda function, and `max()` to find the maximum x coordinate (the 0-th coordinate) in a list of points.

```
pts = [ (6,-1), (8,4), (7.5,-3), (4.4,12), (7,2) ]
```

45.7 Filter: Extract / eliminate values from a list

- Consider a different problem: how to eliminate all of the negative values from a list. Based on what we know so far, this requires a for loop with append.
- We can simplify this using the built-in Python construct called `filter`

```
>>> v = [ 1, 9, -4, -8, 10, -3 ]
>>> list(filter( lambda x: x>0, v))
[1, 9, 10]
```

- Here,
 - The lambda function must produce a boolean value and if that value is `True` the list item is kept; otherwise it is eliminated.
 - The result of `filter` is an iterator object, just like the result of `map` is. We convert to a list in order to display the answer.
- If we want to sum up the non-negative values, then we don't need to explicitly generate a list:

```
>>> sum(filter( lambda x: x>0, v))
20
```

45.8 Lecture Exercises, Problems 1 and 2:

- At this point students will be given the chance to work on the first two lecture exercises.

45.9 Passing Functions to Sort

- Consider the problem of sorting a list of (x,y) points by their y values first and their x values for tied y values, both in decreasing order. For example, given

```
pts = [ (2,5), (12,3), (12,1), (6,5), (14, 10), (12, 10), \
        (8,12), (5,3) ]
```

we'd like the sorted order to be

```
[(8, 12), (14, 10), (12, 10), (6, 5), (2, 5), (12, 3), \
 (5, 3), (12, 1)]
```

- The Python sort function

```
>>> sorted( pts, reverse=True )
[(14, 10), (12, 10), (12, 3), (12, 1), (8, 12), (6, 5), \
 (5, 3), (2, 5)]
```

gives the ordering by x value and then by y value. This is not what we want.

- The first step to a solution is to provide a *key* function to `sorted()` to pull out the information (the y value in this case) from each tuple to use as the basis for sorting:

```
>>> sorted( pts, key = lambda p: p[1], reverse=True)
[(8, 12), (14, 10), (12, 10), (2, 5), (6, 5), (12, 3), \
 (5, 3), (12, 1)]
```

This is close but not quite right because the two points with y=5 are out of order.

- The trick is to sort by x first and then sort by y!

```
>>> by_x = sorted(pts,reverse=True)
>>> by_x
[(14, 10), (12, 10), (12, 3), (12, 1), (8, 12), (6, 5), \
 (5, 3), (2, 5)]
>>> sorted( by_x, key = lambda p: p[1], reverse=True)
[(8, 12), (14, 10), (12, 10), (6, 5), (2, 5), (12, 3), \
 (5, 3), (12, 1)]
```

- This works because `sorted()` uses what's known as a *stable sort*: when two values are “tied” according to the sorting criteria (y value in the second sort) their relative ordering (by x value from the first sort) in the final list is preserved.
 - Therefore, (6,5) comes earlier than (2,5), while (12,3) comes earlier than (5,3)
- A number of variations on sorting use this “stable sort” property, but not all fast sorting algorithms are stable.
- Of course, we can also extend our lambda to reverse the tuple provided to `sort()`

```
>> sorted( pts, key = lambda p: (p[1], p[0]), reverse=True)
[(8, 12), (14, 10), (12, 10), (6, 5), (2, 5), (12, 3), \
 (5, 3), (12, 1)]
```

45.10 Practice Problem

1. Use `filter` to eliminate all words that are shorter than 4 letters from a list of words

45.11 List Comprehensions

- Instead of `map` and `filter` some people prefer another example of functional programming in Python called *list comprehensions*
- Here is an example to generate a list of the squares of the first n integers:

```
>>> n = 8
>>> [ i*i for i in range(1,n+1) ]
[1, 4, 9, 16, 25, 36, 49, 64]
```

- The form of this is an expression followed by a for loop statement.
- We can get the effect of `filter` by adding a conditional at the end:

```
>>> v = [ 1, 9, -4, -8, 10, -3 ]
>>> [ x for x in v if x>0 ]
[1, 9, 10]
```

- Here, the values are only generated in the resultant list when the `if` condition passes.
- We can combine these as well. As a slightly silly example, we can eliminate the negative values and square the positive values

```
>>> v = [ 1, 9, -4, -8, 10, -3 ]
>>> [ x*x for x in v if x>0 ]
[1, 81, 100]
```

- We can get even more sophisticated by nesting for loops. Here is an example where we generate all pairs of numbers between 1 and 4, except for the pairs where the numbers are equal

```
>>> [ (i,j) for i in range(1,5) for j in range(1,5) if i != j ]
[(1, 2), (1, 3), (1, 4), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2),
 (3, 4), (4, 1), (4, 2), (4, 3)]
```

45.12 Exercises

1. Write a list comprehension statement to generate a list of all pairs of odd positive integer values less than 10 where the first value is less than the second value.

45.13 Summary and Discussion

- We've explored programming that is more compact and at a higher level of abstraction. It can be used to effectively interact with data.
- `map` and `filter` each take a function and a sequence (an "iterable") as arguments and produce an iterator as a result:
 - `map` produces the result of applying the function to each element of the iterable
 - `filter` produces each element of the iterable for which the function returns `True`
- Both `map` and `filter` are made more compact by using `lambda` functions
- `lambda` functions can also be used to change the result of sorting
- A *stable sort* preserves the relative order of "tied" values
- *List comprehensions* can be used in place of `map` and `filter`:
 - Some people prefer list comprehensions because they often do not require `lambda` functions, but...
 - List comprehensions explicitly construct the list of results rather than generating them one-by-one, which is what `map` and `filter` do. This makes them less efficient for large data sets.

- These are all examples of *functional programming*.
- We've also used the other major programming paradigms this semester
 - *imperative programming*
 - *object oriented programming*
- Many modern languages like Python provide tools that allow programming using a combination of paradigms