CIS 1100

Functional Programming in Python

Python
Fall 2024
University of Pennsylvania

Review:

We covered three general purpose higher-order functions:

- `filter`
- `map`
- `reduce`

and a new language feature: `lambda`

Today we are just going to do a bunch of practice with it and apply it to more complicated scenarios

Why?

Why are we talking about Higher Order Functions (HOF)?

It turns out that a LOT of problems we want to solve in computer science can reduce down to one of the three funnctions we have shown

- `filter`
- `map` (sometimes called `transform`)
- `reduce` (sometimes called `fold`, `accumulate`, `aggregate` or other terms)

These are sort of "fundamental" patterns in computer science, showing up in many programming languages.

If you want to take more CIS courses (e.g. CIS 1200) then this is a core topic.

`filter`

`filter` is a higher order function that takes in a function and sequence and returns a new sequence containing only those elements for which the provided function evaluates to `True`.

`filter(f, seq)` is equivalent to:

```
[elem for elem in seq if f(elem)]
```

or

```
res = []
for elem in seq:
    if f(elem):
        res.append(elem)
```

`map`

`map` is a higher order function that takes in a function and sequence and returns a new sequence containing elements of the input sequence after having `f` applied to them.

`map(f, seq)` is equivalent to:

```
[f(elem) for elem in seq]
```

or

```
res = []
for elem in seq:
    res.append(f(elem))
```

`reduce`

Aggregation is done using a HOF called `reduce` imported from `functools`.

`reduce` is a function that takes in an *accumulator function* and a sequence. It repeatedly accumulates elements from the sequence using the accumulator function.

`reduce(f, seq)` is roughly equivalent to:

```
result = seq[0]
for elem in seq[1:]:
    result = f(result, elem)
```

Lambdas

Lambdas (or anonymous functions) are functions that are defined without names.

```
lambda <parameter_list> : expression
```

would become

```
def no_name(<parameter_list>):
    return expression
```

lore: lambda comes from lambda calculus, a formal system of mathematics.

Lecture Activity

What do each of these evaluate to?

(S7)

```
def perfect_name(name):
    return len(name) == 5 and 'a' in name

names = ["harRy", "Joel", "SUKya", "aDi", "Molly", "ZWEEEE", "Jared", "thEO", "mario"]
result = list(filter(perfect_name, names))
```

(S8)

```
names = ["harRy", "Joel", "SUKya", "aDi", "Molly", "ZWEEEE", "Jared", "thEO", "mario"]
result = list(map(lambda name : name[0] + name[-1], names))
```

(S9)

```
mexican_states = ["Veracruz", "Ciudad de Mexico", "Nuevo Leon", "Zacatecas", "Quintana Roo"]
mexican_states = list(filter(lambda state : len(state.split()) == 1, mexican_states))
mexican_states = reduce(lambda a, b: a + b[0], mexican_states)
```

Practice! Identifying the pattern

As said before, many problems can boil down to a combination of filter/map/reduce.

For these problems what functions do you think would be needed?

(**Select all that apply for each question**, A for filter, B for map, C for reduce)

- (M1) convert a list of strings/characters into a single string e.g. ["h", "i", "!"] -> "hi!"
- (M2) get a list of all the capital letters in a string
- (M3) given a list of numbers, get another list of numbers that has the square (square of x is x^2), and then sum all those numbers together
- (M4) find the maximum value in a list without using `max()`
- (M5) given a list of strings, get a single string that is made up of the last character of each string. (e.g. ["Hello!" , "I, "am", "tired"] -> "HIat")

Lecture Activity: Let's try to write one from scratch.

Let's take a closer look at **"convert a list of strings/characters into a single string e.g."**

Try implementing it with a higher order function (L11):

```
def strlist_to_str( strings : list[str]) -> list[str]:
  single_string = ____________________________
  return single_string
print(strlist_to_str(["h", "i", "!"])) #Should print "hi!"
```

Applying Higher Order Functions to past problems

Remember the `caesar` homework?

We wrote a function called `string_to_symbol_list()` that took a string and returned a list but all characters were converted to symbols. `ord(charater) - 65`

What does this sound like?

Try implementing it with a higher order function (L13):

```
def string_to_symbol_list(string):
    # TODO
```

Applying HOF to Caesar

```
def shift(symbol, n):
    return (symbol + n) % 26 if 0 <= symbol <= 25 else symbol

# assume `n` is the amount we want to shift each symbol by
def encrypt(to_encode, n):
    symbols = string_to_symbol_list(to_encode)
    # TODO: put something here
    return symbol_list_to_string(symbols)
```

Which of these would work for encrypt? (M6)

- (A) `symbols = list(map(shift, symbols))`
- (B) `symbols = list(map(lambda char : shift(char, n), symbols))`
- (C) `symbols = list(map(shift(n), symbols))`
- (D) `symbols = list(reduce(shift, symbols, []))`

(Bonus: When we implement `symbol_list_to_string`, which HOF will we need?)

Creating Dictionaries from `map`

Sometimes, we want to transform a tuple of key-value pairs into a dictionary.

This is where `map` is incredidly useful!

We can use `map` to process a list of data and turn it into a dictionary with `dict()`.

Example: Mapping People to Word Counts

```
statements = [
    ("Dmitri", "Passion rules all!"),
    ("Ivan", "If God does not exist, everything is permitted."),
    ("Alyosha", "Love is the only way.")
]
mapped = map(lambda s: ( s[0], len(s[1].split())), statements)
                    # (  __ ,  _____________  ) is a (key, value) pair!
word_counts = dict(mapped)

print(word_counts)
# Output: {'Dmitri': 3, 'Ivan': 8, 'Alyosha': 5}
```

Activity: Mapping Students to Grades

Use `map` to convert a list of student scores into a dictionary of **letter grades**. C12:

Grading Scale:

- **90+ → 'A', 80-89 → 'B', 70-79 → 'C', 60-69 → 'D', Below 60 → 'F'**

```
students = [ ("Molly", 92), ("Theo", 85), ("Zara", 77), ("Joel", 59), ("Sofia", 88) ]

def get_letter_grade(score):
    #TODO

# TODO: Use map to create a dictionary
mapped = map(_____________________________________)
student_grades = dict(mapped)

print(student_grades)
```

Expected Output:

{'Alice': 'A', 'Bob': 'B', 'Charlie': 'C', 'David': 'F', 'Eve': 'B'}

Extending The Previous Problem

Let's transform this into a dictionary so that:

- Computes the average score for each student
- Filters out students who failed (average < 60)
- Maps remaining students to a letter grade

```
students = {
    "Molly": [92, 85, 78], "Theo": [55, 60, 58], "Zara": [88, 90, 85],  "Joel": [59, 50, 45],
    "Sofia": [75, 80, 72]
}
# Step 1: Create a dictionary of Name : Average Grade (hint use .items())
averages = dict(_________________________)
# Step 2: Remove out failing students < 60
passing_students = dict(_______________________)
# Step 3: Map remaining students to letters using get_letter_grade(score) like we did before
graded_students = dict(__________________________)
print(graded_students)
```

Output: `{'Molly': 'B', 'Zara': 'A', 'Sofia': 'C'}`

Next time!

Searching :)