Lecture 23:
Sorting with Lambda

Guest Lecture by Elyse Cornwall
Anonymous 7h

glory be to god. thank you Elyse

❤️ 2 Reply Edit Delete ⋯
Recap: Lambda
Recall Lambda - Super Powerful!

```python
>>> list(map(lambda n: n * 2, [1, 2, 3, 4, 5]))
[2, 4, 6, 8, 10]
```
1, 2, 3... Lambda

```python
>>> list(map(lambda n: n * 2, [1, 2, 3, 4, 5]))
```
1, 2, 3... Lambda

```python
>>> list(map(lambda n: n * 2, [1, 2, 3, 4, 5]))
```

1. The word “lambda”
1, 2, 3... Lambda

```python
>>> list(map(lambda n: n * 2, [1, 2, 3, 4, 5]))
```

1. The word “lambda”
2. What type of element? Choose a good parameter name.
1, 2, 3... Lambda

>>> list(map(lambda n: n * 2, [1, 2, 3, 4, 5]))

1. The word “lambda”
2. What type of element? Choose a good parameter name.
3. Expression to produce - no return needed.
Def Functions

Gives a name to some code

def double(n):
    return n * 2
Def vs. Lambda

Function def

def double(n):
    return n * 2

Equivalent lambda

lambda n: n * 2
Def vs. Lambda

Function def

def double(n):
    return n * 2

Equivalent lambda

lambda n: n * 2

Can we just use lambda for everything now?
Def vs. Lambda

Function def

def double(n):
    return n * 2

Equivalent lambda

lambda n: n * 2

Can we just use lambda for everything now?

No.
Features of Def

Def has room for real code features:

- Multiple lines
- If statements
- Variables
- Loops
- Doctests
- Inline comments

Lambda is best for short, one-off lines of code
Practice: \texttt{map\_parens(strs)}

\[
[\text{'xx(hi)xx'}, \text{'abc(there)xyz'}, \text{'fish'}]
\]

\[
[\text{'hi'}, \text{'there'}, \text{'fish'}]
\]
Recap: Sorting
Sorting So Far

Use the `sorted()` function to sort lists:

```python
g>>> nums = [5, 7, 3, 4]
g>>> sorted(nums)
[3, 4, 5, 7]

g>>> strs = ['hi', 'bye', 'greetings', 'good day']
g>>> sorted(strs)
['bye', 'good day', 'greetings', 'hi']
```
Sorting So Far

Use the `sorted()` function to sort lists:

```python
>>> cities = [('tx', 'houston'), ('ca', 'palo alto'), ('ca', 'san jose'), ('tx', 'austin'), ('ca', 'aardvark')]

>>> sorted(cities)
[('ca', 'aardvark'), ('ca', 'palo alto'), ('ca', 'san jose'), ('tx', 'austin'), ('tx', 'houston')]
```
Sorting So Far

Use the `sorted()` function to sort lists:

```python
>>> cities = [('tx', 'houston'), ('ca', 'palo alto'), ('ca', 'san jose'), ('tx', 'austin'), ('ca', 'aardvark')]

>>> sorted(cities)
[('ca', 'aardvark'), ('ca', 'palo alto'), ('ca', 'san jose'), ('tx', 'austin'), ('tx', 'houston')]
```

What if I want to sort by city name? Or length of city name?
Custom Sorting
Custom Sorting Foods

```python
>>> foods = [('radish', 2, 8), ('donut', 10, 1), ('apple', 7, 9), ('broccoli', 6, 10)]
```

Each food is a length-3 food tuple: (name, tastiness 1-10, healthiness 1-10)

- food[0] = its name
- food[1] = how tasty it is 1-10
- food[2] = how healthy it is 1-10
Custom Sorting Foods

```python
>>> t = ('radish', 2, 8)
>>> t[0]
'radish'

>>> foods = [('radish', 2, 8), ('donut', 10, 1), ('apple', 7, 9), ('broccoli', 6, 10)]

>>> sorted(foods)
[('apple', 7, 9), ('broccoli', 6, 10), ('donut', 10, 1), ('radish', 2, 8)]
```
Custom Sorting Foods

What if I want to sort by tastiness?

- Control how sorted() looks at each food tuple
- It’s like drawing a circle around tasty values - sort by these!

[(‘radish’, 2, 8), (‘donut’, 10, 1), (‘apple’, 7, 9), (‘broccoli’, 6, 10)]
Custom Sorting Foods

● "Project out" a value from each item
  ○ For each food tuple, project out its tastiness value
● Projected value is used for sorting comparisons

[('radish', 2, 8), ('donut', 10, 1), ('apple', 7, 9), ('broccoli', 6, 10)]
Custom Sorting Foods

- "Project out" a value from each item
  - For each food tuple, project out its tastiness value
- Projected value is used for sorting comparisons

```
[('radish', 2, 8), ('donut', 10, 1), ('apple', 7, 9), ('broccoli', 6, 10)]
```

\[
\text{lambda food: food[1]}
\]
1, 2 … Custom Sort

>>> sorted(foods, key=lambda food: food[1])

1. Call sorted with your list
2. Provide key = lambda to project out sorting value
3. Optionally: reverse
1, 2 … Custom Sort

```python
>>> sorted(foods,
          key=lambda food: food[1])
```

1. Call `sorted` with your list
1, 2 … Custom Sort

```python
>>> sorted(foods, key=lambda food: food[1])
```

1. Call `sorted` with your list
2. Provide `key = lambda` to project out the sorting value
1, 2 … Custom Sort

```python
>>> sorted(foods, key=lambda food: food[1], reverse=True)
```

1. Call `sorted` with your list
2. Provide `key = lambda` to project out the sorting value
3. Optionally, `reverse`
# ascending tastiness
```
>>> sorted(foods, key=lambda food: food[1])
[('radish', 2, 8), ('broccoli', 6, 10), ('apple', 7, 9), ('donut', 10, 1)]
```

# descending tastiness
```
>>> sorted(foods, key=lambda food: food[1], reverse=True)
[('donut', 10, 1), ('apple', 7, 9), ('broccoli', 6, 10), ('radish', 2, 8)]
```

# descending healthiness
```
>>> sorted(foods, key=lambda food: food[2], reverse=True)
[('broccoli', 6, 10), ('apple', 7, 9), ('radish', 2, 8), ('donut', 10, 1)]
```

# descending composite tastiness-healthiness score
```
>>> sorted(foods, key=lambda food: food[1] * food[2], reverse=True)
[('apple', 7, 9), ('broccoli', 6, 10), ('radish', 2, 8), ('donut', 10, 1)]
```
Sorted, Min, and Max

*What if I just want the most tasty food? Or the least tasty?*

- Sorting n things is kind of expensive
- Use `max()` or `min()` - takes a key=lambda just like `sorted()`
  - All we have to do is change "sorted" to "max" or "min"
Sorted, Min, and Max

# uses index 0 (name) by default - tragic!

```python
>>> max(foods)
('radish', 2, 8)
```

# most tasty

```python
>>> max(foods, key=lambda food: food[1])
('donut', 10, 1)
```

# least tasty

```python
>>> min(foods, key=lambda food: food[1])
('radish', 2, 8)
```
Movie Sorting

movies = [('alien', 8, 1), ('titanic', 6, 9), ('parasite', 10, 6), ('caddyshack', 4, 5)]

Each movie is a length-3 tuple: (name, score, date-score)

- movie[0] = its name
- movie[1] = how good it is 1-10
- movie[2] = how appropriate for a date it is 1-10
Practice: \texttt{sort\_score(movies)}

\[
[(\text{\textquoteleft alien\textquoteleft}, 8, 1), (\text{\textquoteleft titanic\textquoteleft}, 6, 9), (\text{\textquoteleft parasite\textquoteleft}, 10, 6), (\text{\textquoteleft caddyshack\textquoteleft}, 4, 5)]
\]
Practice: \texttt{\textbf{sort\_date(movies)}}

\[
[('alien', 8, 1), ('titanic', 6, 9), ('parasite', 10, 6), ('caddyshack', 4, 5)]
\]
Sort 21

List of integers, want to sort by distance from 21

\[ \text{nums} = [10, 20, 0, 51] \]
List of integers, want to sort by distance from 21

[nums = [10, 20, 0, 51]}

11 1 21 30

*How do we “project out” this distance?*
Sort 21

List of integers, want to sort by distance from 21

\[
\text{nums} = [10, 20, 0, 51]
\]

\[
\lambda n: \text{???}
\]

11 1 21 30
Practice: $\text{sort21}(\text{nums})$

$[10, 20, 0, 51]$

11 1 21 30
Practice: **best21(nums)**

\[ [10, 20, 0, 51] \]

11, 1, 21, 30
Remember wordcount.py?

- Reads in a text file
- Builds a counts dictionary for all words in the file
- Here’s the zip file

$ python3 wordcount.py tale-of-two-cities.txt
a 2866
a-a-a-business 1
a-a-matter 1
a-buzz 1
a-tiptoe 1
aback 1
...
Let’s implement print_top()

$ python3 wordcount.py -top 5 tale-of-two-cities.txt
the 7838
and 4833
of 3933
to 3397
a 2866
Let’s implement print_top()

```python
>>> counts = {'a': 2866, 'tale': 2, 'of': 3933, 'two': 206, 'cities': 2}
```
Let's implement print_top()

```python
>>> counts = {'a': 2866, 'tale': 2, 'of': 3933, 'two': 206, 'cities': 2}

>>> counts.keys()  # list of keys
dict_keys(['a', 'tale', 'of', 'two', 'cities'])

>>> counts.values()  # list of values
dict_values([2866, 2, 3933, 206, 2])

>>> counts.items()  # list of key, value tuples
dict_items([('a', 2866), ('tale', 2), ('of', 3933), ('two', 206), ('cities', 2)])
```
Let’s implement print_top()

[('a', 2866), ('tale', 2), ('of', 3933), ('two', 206), ('cities', 2)]
Let’s implement print_top()

$ python3 wordcount.py -top 5 tale-of-two-cities.txt
the 7838
and 4833
of 3933
to 3397
a 2866
Solution

# 1. Sort largest count first
items = sorted(items, key=lambda pair: pair[1], reverse=True)

# 2. Print first N
for word, count in items[:n]:
    print(word, count)
Thank you!