Functions As Objects

Function object

In Python, **everything** is an object. Including functions.

```
>>> def f(n): return n+2
...
>>> id(f)
4314937816
>>> g = f
>>> print(g(3))
5
>>> id(g)
4314937816
```

You can use this fact to do **amazing things**.

Example: maximum

Suppose you have a list of numbers as strings:

```
nums = ["12", "7", "30", "14", "3"]
```

How can you find the element with biggest numeric value? The `max` builtin does not help:

```
>>> max(nums)
'7'
```

`max` is evaluating the element by a *different criteria* than what we want.

max int value

```
>>> def max_by_int_value(items):
...     # For simplicity, assume len(items) > 0
...     biggest = items[0]
...     for item in items[1:]:
...         if int(item) > int(biggest):
...             biggest = item
...     return biggest
...
>>> max_by_int_value(nums)
'30'
```

max by absolute value

Different data, different criteria:

```
integers = [3, -2, 7, -1, -20]
```

Suppose we want to the element with maximum *absolute value*. That would be -20 here, but standard `max` won't do that:

```
>>> max(integers)
7
```

max by absolute value

Again, let's roll our own, using the built-in `abs` function:

```
>>> def max_by_abs(items):
...     biggest = items[0]
...     for item in items[1:]:
...         if abs(item) > abs(biggest):
...             biggest = item
...     return biggest
...
>>> max_by_abs(integers)
-20
```

list of dictionaries

One more example - a list of dictionary objects:

```
student_joe = {'gpa': 3.7, 'major': 'physics',
               'name': 'Joe Smith'}
student_jane = {'gpa': 3.8, 'major': 'chemistry',
                'name': 'Jane Jones'}
student_zoe = {'gpa': 3.4, 'major': 'literature',
               'name': 'Zoe Grimwald'}
students = [student_joe, student_jane, student_zoe]
```

Now, what if we want the record of the student with the highest GPA?

max gpa

Here's a suitable max function:

```
>>> def max_by_gpa(items):
...     biggest = items[0]
...     for item in items[1:]:
...         if item["gpa"] > biggest["gpa"]:
...             biggest = item
...     return biggest
...
>>> max_by_gpa(students)
{'name': 'Jane Jones', 'gpa': 3.8, 'major': 'chemistry'}
```

What's the difference?

Just one line of code is different between `max_by_int_value`, `max_by_abs`, and `max_by_gpa`:

```
# for max_by_int_value
if int(item) > int(biggest):

# for max_by_abs
if abs(item) > abs(biggest):

# for max_by_gpa
if item["gpa"] > biggest["gpa"]:
```

Other than that, the `max` functions are **identical**.

Comparison key function

Let's define a *key function* for each of them, which extracts the relevant value:

```
# for max_by_int_value
int

# for max_by_abs
abs

# for max_by_gpa
def get_gpa(student): return student["gpa"]
```

Max by key

This lets us define a very generic max function:

```
>>> def max_by_key(items, key):
...     biggest = items[0]
...     for item in items[1:]:
...         if key(item) > key(biggest):
...             biggest = item
...     return biggest
...
>>> # Old way:
... max_by_int_value(nums)
'30'
>>> # New way:
... max_by_key(nums, int)
'30'
>>> # Old way:
... max_by_abs(integers)
-20
>>> # New way:
... max_by_key(integers, abs)
-20
```

Using the key function

This is the important line:

```
        # key is actually int, abs, etc.
        if key(item) > key(biggest):
```

IMPORTANT: You never invoke the key function yourself. You pass the function object to `max_by_key`, which then invokes it for you.

GPA

For the student GPA, there's no built-in... so we make our own.

```
>>> # Old way:
... max_by_gpa(students)
{'gpa': 3.8, 'name': 'Jane Jones', 'major': 'chemistry'}

>>> # New way:
... def get_gpa(who):
...     return who["gpa"]
...
>>> max_by_key(students, get_gpa)
{'gpa': 3.8, 'name': 'Jane Jones', 'major': 'chemistry'}
```

Quick exercise: passing functions to functions

In your labs folder, open `functions/maxbykey.py`. Type in the following for `max_by_key`, so that the tests pass:

```
def max_by_key(items, key):
    biggest = items[0]
    for item in items[1:]:
        if key(item) > key(biggest):
            biggest = item
    return biggest
```

Key Functions in Python

Too bad the built-in `max()` doesn't do this, huh?

```
>>> nums = ["12", "7", "30", "14", "3"]
>>> max(nums)
'7'
```

Oh wait! It DOES!!

```
>>> max(nums, key=int)
'30'
```

max, min, sorted

It also works with `min()` and `sorted()`.

```
>>> # Default behavior...
... min(nums)
'12'
>>> sorted(nums)
['12', '14', '3', '30', '7']
>>>
>>> # And with a key function:
... min(nums, key=int)
'3'
>>> sorted(nums, key=int)
['3', '7', '12', '14', '30']
```

Many algorithms can be cleanly expressed using `min`, `max`, or `sorted`, along with an appropriate key function.

Warning: Use "key="

```
>>> nums = ["12", "7", "30", "14", "3"]
>>> # This works...
... max(nums, key=int)
'30'
>>> # And this does not.
... max(nums, int)
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: unorderable types: type() > list()
```

Student data

Sorting by GPA:

```
>>> student_joe = {'gpa': 3.7, 'major': 'physics', 'name': 'Joe Smith'}
>>> student_jane = {'gpa': 3.8, 'major': 'chemistry', 'name': 'Jane Jones'}
>>> student_zoe = {'gpa': 3.4, 'major': 'literature', 'name': 'Zoe Grimwald'}
>>> students = [student_joe, student_jane, student_zoe]
>>>
>>> def get_gpa(who):
...     return who["gpa"]
...
>>> sorted(students, key=get_gpa)
[{'gpa': 3.4, 'major': 'literature', 'name': 'Zoe Grimwald'},
 {'gpa': 3.7, 'major': 'physics', 'name': 'Joe Smith'},
 {'gpa': 3.8, 'major': 'chemistry', 'name': 'Jane Jones'}]
```

operator module tools

Instead of defining `get_gpa`, we can use `operator.itemgetter`.

```
>>> from operator import itemgetter
>>>
>>> # Sort by GPA...
... sorted(students, key=itemgetter("gpa"))
[{'gpa': 3.4, 'major': 'literature', 'name': 'Zoe Grimwald'},
 {'gpa': 3.7, 'major': 'physics', 'name': 'Joe Smith'},
 {'gpa': 3.8, 'major': 'chemistry', 'name': 'Jane Jones'}]
>>>
>>> # Now sort by major:
... sorted(students, key=itemgetter("major"))
[{'gpa': 3.8, 'major': 'chemistry', 'name': 'Jane Jones'},
 {'gpa': 3.4, 'major': 'literature', 'name': 'Zoe Grimwald'},
 {'gpa': 3.7, 'major': 'physics', 'name': 'Joe Smith'}]
```

Functions returning functions

Note: `itemgetter` is a function that creates and returns a function. The following two key functions are completely equivalent:

```
# What we did above:
def get_gpa(who):
    return who["gpa"]

# Using itemgetter instead:
from operator import itemgetter
get_gpa = itemgetter("gpa")
```

A powerful example of using function objects!

Getting attributes

`operator.attrgetter` does something similar for classes.

```
>>> class Student:
...     def __init__(self, name, major, gpa):
...         self.name = name
...         self.major = major
...         self.gpa = gpa
...     def __repr__(self):
...         return "{}: {}".format(self.name, self.gpa)
...
>>> student_objs = [
...     Student("Joe Smith", "physics", 3.7),
...     Student("Jane Jones", "chemistry", 3.8),
...     Student("Zoe Grimwald", "literature", 3.4),
...     ]
>>> from operator import attrgetter
>>> sorted(student_objs, key=attrgetter("gpa"))
[Zoe Grimwald: 3.4, Joe Smith: 3.7, Jane Jones: 3.8]
```

Lab: Key Functions

This lab challenges you to use key functions along with `max`, `min`, and `sorted`.

Lab file: `functions/keyfunc.py`

- In labs/py3 for 3.x; labs/py2 for 2.7
- When you are done, give a thumbs up...
- and then start the extra lab.

Extra Lab: Once you get the lab to pass, read about lambda expressions in the Python docs: https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions

Modify your functions to use lambdas as much as possible. What are the trade-offs for readability?