CS051A
INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

24: Higher order functions

Alexandra Papoutsaki
she/her/hers
Lectures

Zilong Ye
he/him/his
Labs

04-26-2023
Lecture 24: Higher order functions

- Higher order functions
Higher order functions

- Have you ever typed a function into the shell, but forgot the parentheses?
  ```python
def my_function(x):
    return x+1
>>> my_function(2)
3
>>> my_function
<function my_function at 0x108e962f0>
>>> abs
<built-in function abs>
```

- Notice that it does NOT give an error.
  ```python
>>> type(my_function)
<class 'function'>
```
- Instead, it echoes the value, just like any other expression, in this case, the value is a function!
Higher order functions

- Functions in python are values, just like everything else!
  ```python
  >>> y = my_function
  >>> y
  <function my_function at 0x108e962f0>
  >>> y(2)
  3
  >>> my_abs = abs
  >>> my_abs(-10)
  10
  ```

- we can pass them as parameters
- we can return them from functions
- we can even create them on the fly!
What do the first four function in higher_order_functions.py do?
- Take two arguments and do standard mathematical calculations

What does add2 do in higher_order_functions.py?
- Takes one parameter, a tuple of two items
- Unpacks the tuple, adds and returns its items.

What does double do in higher_order_functions.py?
- Takes one parameter.
- Multiplies by 2 and returns it.

What does is_even do in higher_order_functions.py?
- Takes one parameter, a number.
- Returns whether this number is even.
What does `apply_function` do in `higher_order_functions.py`?
- Takes three parameters
  - the first is a function!
  - applies the function passed as the first argument to the second and third argument and returns the result.

We can call our `apply_function` function:
```python
>>> apply_function(add, 2, 3)
5
>>> apply_function(subtract, 2, 3)
-1
```
To pass a function as a parameter you just give the name of the function as the argument.

`def`. What the keyword `def` actually does is:
- create a new function
- assign that function to a variable with the name of the function.
higher_order_functions.py

```python
Python 3.6
known limitations

1. def my_function(x):
   return x + 1

2. print(my_function(3))

https://pythontutor.com/visualize.html#mode=display
```
What does the `apply_function_to_list` function do in `higher_order_functions.py`?

- Takes a function and a list as parameters
- You can tell that the parameter `f` is a function because we apply it in the line with the `append` in it
- Iterates through each value in the list
- Applies the function `f`
- Appends the result of the function `f` to a list that is returned at the end.

High-level: Applies the function to each element in the list and returns a new list containing the result from each of those applications.

For example:
```python
>>> apply_function_to_list(double, [1, 2, 3, 4])
[2, 4, 6, 8]
>>> apply_function_to_list(add2, [(1, 2), (3, 4)])
[3, 7]
```
What does the `apply_function_to_tuple` function do in `higher_order_functions.py`?

- takes a function and a list of two 2-tuples as parameters
- The function should take two parameters
- iterates through each 2-tuple in the list and unpacks it
- applies the function $f$ on the two items
- appends the result of the function $f$ to a list that is returned at the end.

For example:

```python
>>> apply_function_to_tuple(add, [(1, 2), (3, 4)])
[3, 7]
```
apply_function_to_list is actually built in to python and is called map:

```python
>>> help(map)
Help on class map in module builtins:
    class map(object)
      map(func, *iterables) --> map object
      Make an iterator that computes the function using arguments from
      each of the iterables. Stops when the shortest iterable is exhausted.

Takes as input a function and something that is iterable

- only difference from apply_function_to_list is that it returns a map object (not a list), which is also iterable.

```
map

- By itself, this may not seem useful, but we can do more complicated things. What would this print?

```python
>>> for val in map(double, map(double, [1, 2, 3, 4])):
    print(val)
```

- The first map doubles it and then we iterate on this result and double it again!
filter

- What does the filter_list function do in higher_order_functions.py code?
  - Also takes a function some_function and a list some_list as parameters
- Are there any expectations on what some_list should do/return?
  - it's used in an if statement
  - it should return a bool, i.e. True or False
- Similarly to map, Python has a built-in function for this behavior called filter.
  - The filter function returns a list of all elements of some_list that would return True when passed to some_function. Note how it differs from map.
- For example,
  ```python
  >>> list(map(is_even, [1, 2, 3, 4]))
  [False, True, False, True]
  >>> list(filter(is_even, [1, 2, 3, 4]))
  [2, 4]
  ```
Lambda

- It can be a bit annoying having to write all of these simple functions to simply pass them as an argument to another function.
- Python allows us to create **anonymous functions**, i.e., functions that don't have an explicit name, but are simply code.
- The syntax is:
  ```
  lambda <input>: <expression>
  ```
  - `<input>` is the parameter to the anonymous function.
    - If you need to pass multiple inputs, just pass them as a tuple.
  - `<expression>` is the body of the function that is executed and returned. It can only be a single expression (i.e., something that represents a value).
- An example:
  ```
  >>> lambda x: x+1
  <function <lambda> at 0x7f7ff80981e0>
  ```
  - Notice that it gives the same function type back, but it doesn't have a name!
  ```
  >>> (lambda x: x+1)(2)
  3
  ```
Lambda

- We can also associate it with a variable and call it, e.g.,

  ```python
  f = lambda x: x + 1
  >>> f(2)
  3
  ```

- Makes life easier!

  ```python
  >>> filter_list(lambda num: num % 2 == 0, [1, 2, 3, 4])
  [2, 4]
  ```
Lambda

- Let’s look at this unusual function that returns a... function

```python
def kinda_crazy(num):
    def multiplier(x):
        return num * x
    return multiplier

>>> type(kinda_crazy(3))
<class 'function'>
>>> kinda_crazy(3)(2)
6
```

- We could use an anonymous function to be even more concise!
```python
def crazy(num):
    return lambda x: num * x

>>> crazy(3)(2)
6
```
Monte Carlo sampling

- Monte Carlo methods are a way of determining the answer to numerical problems via random sampling.

- General idea:
  - generate random samples
  - look at the outcome of those random samples
  - use the answer to the outcomes to estimate the answer for the original problem.

- An example: calculating the area of a shape
  - We want to calculate the area of a shape. Specifically, if I draw an arbitrary shape within a 1 by 1 box, can you tell me the area?
    - kind of hard!
  - What if I put a bunch of points uniformly in the box. Could I tell how many are inside the shape?
    - e.g., if I put 1000 points in the box with a triangle shape, how many would you expect in the triangle?
      - about 500
    - what would be the area of the triangle?
      - $500/1000 = 0.5$
  - key idea: use the proportion of points that fall inside the shape to estimate the area.
Assuming $0 \leq x \leq 1$ and $0 \leq y \leq 1$ what does the `in_triangle` function do?

- Returns true if $x$ and $y$ are within the red triangle

Graph for $1-x$
montecarlo.py

- Assuming $0 \leq x \leq 1$ and $0 \leq y \leq 1$ what does the `in_circle` function do?
- Returns true if $x$ and $y$ are inside the quarter circle.
Write a function `monte_carlo` that takes two parameters:
number of trials (samples) and a shape function

- generate "trials" random points (x, y points between 0 and 1)
- count how many are "inside" the shape
- return the proportion, i.e., count/trials.

Hint:

- `import random`
- `random.random() # returns random value between 0 and 1`
Look at the `monte_carlo` function in `montecarlo.py` code

We can use this to estimate the area of different shapes:

```python
>>> monte_carlo(1000, in_triangle)
0.484
>>> monte_carlo(10000, in_triangle)
0.5005
>>> monte_carlo(100000, in_triangle)
0.49756
>>> monte_carlo(100000, in_circle)
0.7854
>>> monte_carlo(100000, in_circle)*4
3.14896
>>> monte_carlo(1000000, in_circle)*4
3.141972
>>> monte_carlo(10000000, in_circle)*4
3.141894
```
Resources

▸ higher-order_functions.py

▸ montecarlo.py

Homework

▸ Assignment 12 (cont’d)