

# Need help understanding function passing in Python

Asked 15 years, 10 months ago   Modified 15 years, 10 months ago   Viewed 3k times



I am trying to teach myself Python by working through some problems I came up with, and I need some help understanding how to pass functions.

4



Let's say I am trying to predict tomorrow's temperature based on today's and yesterday's temperature, and I have written the following function:



```
def predict_temp(temp_today, temp_yest, k1, k2):  
    return k1*temp_today + k2*temp_yest
```

And I have also written an error function to compare a list of predicted temperatures with actual temperatures and return the mean absolute error:

```
def mean_abs_error(predictions, expected):  
    return sum([abs(x - y) for (x,y) in zip(predictions,expected)]) /  
    float(len(predictions))
```

Now if I have a list of daily temperatures for some interval in the past, I can see how my prediction function would have done **with specific k1 and k2 parameters** like this:

```
>>> past_temps = [41, 35, 37, 42, 48, 30, 39, 42, 33]  
>>> pred_temps = [predict_temp(past_temps[i-1],past_temps[i-2],0.5,0.5) for i  
in xrange(2,len(past_temps))]  
>>> print pred_temps  
[38.0, 36.0, 39.5, 45.0, 39.0, 34.5, 40.5]  
>>> print mean_abs_error(pred_temps, past_temps[2:])  
6.5
```

**But how do I design a function to minimize my parameters k1 and k2 of my predict\_temp function given an error function and my past\_temps data?**

Specifically I would like to write a function `minimize(args*)` that takes a prediction function, an error function, some training data, and that uses some search/optimization method (gradient descent for example) to estimate and return the values of k1 and k2 that minimize my error given the data?

I am not asking how to implement the optimization method. Assume I can do that. Rather, I would just like to know **how to pass my predict and error functions** (and my data) to my minimize function, and **how to tell my minimize function that it should optimize the parameters k1 and k2**, so that my minimize function can

automatically search a bunch of different settings of  $k_1$  and  $k_2$ , applying my prediction function with those parameters each time to the data and computing error (like I did manually for  $k_1=0.5$  and  $k_2=0.5$  above) and then return the best results.

I would like to be able to pass these functions so I can easily swap in different prediction and error functions (differing by more than just parameter settings that is). Each prediction function might have a different number of free parameters.

My minimize function should look something like this, but I don't know how to proceed:

```
def minimize(prediction_function, which_args_to_optimize, error_function, data):  
    # 1: guess initial parameters  
    # 2: apply prediction function with current parameters to data to compute predictions  
    # 3: use error function to compute error between predictions and data  
    # 4: if stopping criterion is met, return parameters  
    # 5: update parameters  
    # 6: GOTO 2
```

Edit: It's that easy?? This is no fun. I am going back to Java.

On a more serious note, I think I was also getting hung up on how to use different prediction functions with different numbers of parameters to tune. If I just take all the free parameters in as one tuple I can keep the form of the function the same so it easy to pass and use.

python

Share

edited Feb 23, 2009 at 19:16

Improve this question

Follow

asked Feb 23, 2009 at 18:47



Imran

13.4k ● 8 ● 68 ● 82

---

if you are just asking how you can pass a function as an argument to another function, this question is entirely too long ;) – user3850 Feb 23, 2009 at 19:14

---

Yes I was trying to find a nice balance between asking a general question and giving a concrete example in case some other solution to my problem in context would be better. At that I failed miserably ;) – Imran Feb 23, 2009 at 19:20

---

BTW, I think you mean *args*, not *args*. – Nikhil Feb 24, 2009 at 2:21

---

3 Answers

Sorted by: Highest score (default)





13



Here is an example of how to pass a function into another function. `apply_func_to` will take a function `f` and a number `num` as parameters and `return f(num)`.

```
def my_func(x):
    return x*x

def apply_func_to(f, num):
    return f(num)

>>>apply_func_to(my_func, 2)
4
```

If you wanna be clever you can use lambda (anonymous functions too). These allow you to pass functions "on the fly" without having to define them separately

```
>>>apply_func_to(lambda x:x*x, 3)
9
```

Hope this helps.

Share

edited Feb 24, 2009 at 8:39

answered Feb 23, 2009 at 18:59

Improve this answer



Il-Bhima

10.9k ● 1 ● 49 ● 51

Follow



2



Function passing in Python is easy, you just use the name of the function as a variable which contains the function itself.

```
def predict(...):
    ...

minimize(predict, ..., mean_abs_error, ...)
```

As for the rest of the question: I'd suggest looking at the way [SciPy implements this](#) as a model. Basically, they have a function `leastsq` which minimizes the sum of the squares of the residuals (I presume you know what least-squares minimization is ;-). What you pass to `leastsq` is a function to compute the residuals, initial guesses for the parameters, and an arbitrary parameter which gets passed on to your residual-computing function (the closure), which includes the data:

```
# params will be an array of your k's, i.e. [k1, k2]
def residuals(params, measurements, times):
    return predict(params, times) - measurements

leastsq(residuals, initial_parameters, args = (measurements, times))
```

Note that SciPy doesn't actually concern itself with how you come up with the residuals. The `measurements` array is just passed unaltered to your `residuals` function.

I can look up an example I did recently if you want more information - or you can find examples online, of course, but in my experience they're not quite as clear. The particular bit of code I wrote would relate well to your scenario.

Share

edited Feb 23, 2009 at 19:16

answered Feb 23, 2009 at 18:55

Improve this answer

Follow



David Z

131k ● 29 ● 261 ● 281

---

Thanks for the SciPy info, I will definitely check that out. – [Imran](#) Feb 23, 2009 at 19:38

---



1



As [David](#) and [Il-Bhima](#) note, functions can be passed into other functions just like any other type of object. When you pass a function in, you simply call it like you ordinarily would. People sometimes refer to this ability by saying that functions are *first class* in Python. At a slightly greater level of detail, you should think of functions in Python as being one type of *callable object*. Another important type of callable object in Python is class objects; in this case, calling a class object creates an instance of that object. This concept is discussed in detail [here](#).

Generically, you will probably want to leverage the positional and/or keyword argument feature of Python, as described [here](#). This will allow you to write a generic minimizer that can minimize prediction functions taking different sets of parameters. I've written an example---it's more complicated than I'd like (uses generators!) but it works for prediction functions with arbitrary parameters. I've glossed over a few details, but this should get you started:

```
def predict(data, k1=None, k2=None):
    """Make the prediction."""
    pass

def expected(data):
    """Expected results from data."""
    pass

def mean_abs_err(pred, exp):
    """Compute mean absolute error."""
    pass

def gen_args(pred_args, args_to_opt):
    """Update prediction function parameters.

    pred_args : a dict to update
    args_to_opt : a dict of arguments/iterables to apply to pred_args

    This is a generator that updates a number of variables
```

```

over a given numerical range. Equivalent to itertools.product.

"""

base_args = pred_args.copy() #don't modify input

argnames = args_to_opt.keys()
argvals = args_to_opt.values()
result = [[]]
# Generate the results
for argv in argvals:
    result = [x+[y] for x in result for y in argv]
for prod in result:
    base_args.update(zip(argnames, prod))
    yield base_args

def minimize(pred_fn, pred_args, args_to_opt, err_fn, data):
    """Minimize pred_fn(data) over a set of parameters.

    pred_fn : function used to make predictions
    pred_args : dict of keyword arguments to pass to pred_fn
    args_to_opt : a dict of arguments/iterables to apply to pred_args
    err_fn : function used to compute error
    data : data to use in the optimization

    Returns a tuple (error, parameters) of the best set of input parameters.
    """
    results = []
    for new_args in gen_args(pred_args, args_to_opt):
        pred = pred_fn(data, **new_args) # Unpack dictionary
        err = err_fn(pred, expected(data))
        results.append((err, new_args))
    return sorted(results)[0]

const_args = {k1: 1}
opt_args = {k2: range(10)}
data = [] # Whatever data you like.
minimize(predict, const_args, opt_args, mean_abs_err, data)

```

Share

Improve this answer

Follow

edited May 23, 2017 at 12:13



Community Bot

1 • 1

answered Feb 23, 2009 at 20:10



zweiterlinde

14.8k • 2 • 29 • 33