PythonTip 01 - Functions (post-class-version)

February 1, 2023

1 Python Tip #1: Functions

Functions are separately defined code snippets that you can then use in your main code.

[1]:
```
def double(number): # arguments = input
    new_number = 2*number
    return new_number
```

[2]:
```
double(5)
```

[2]: 10

[3]:
```
def print_hello():
    print("hello")
```

[5]:
```
print_hello()
```

```
hello
```

[ ]:
```

```

[6]:
```
def double(number=7): # number will default to 7 if you don't specify it
    new_number = 2*number
    return new_number
```

[7]:
```
double()
```

[7]: 14

[8]:
```
double(5)
```

[8]: 10

[9]:
```
double(number=5)
```

[9]: 10

[10]:
```
def double(number): # arguments = input
    new_number = 2*number
```

```
    return new_number
```

"Lambda Functions" sound very fancy, but they are just a quicker way to define very simple functions.

```
double = lambda x : 2*x
```

```
[name] = lambda [inputs] : [outputs]
```

[11]:
```
new_double = lambda number : 2*number
new_double(5)
```

[11]: 10

[13]:
```
# to do a to the power of b in python, the
# syntax is a ** b, not a ^ b
combine = lambda x, y: 2*x + 3 * y**2
```

[14]:
```
combine(5,2)
```

[14]: 22

[26]:
```
# default argument with a lambda function
new_combine = lambda x=1, y=1 : 2*x + 3*y**2
```

[30]:
```
new_combine(5,2)
```

[30]: 22

[28]:
```
new_combine(5)
```

[28]: 13

[20]:
```
def example(x, y=3):
    return x + y
```

[22]:
```
example(y=1, x=2)
```

[22]: 3

[ ]:

[ ]:

They are often useful (as we'll see later) for extracting one component of a tuple or list.

[31]:
```
second_component = lambda r : r[1]
```

[32]:
```
second_component([5, -8, 1])
```

[32]:
```
-8
```

This is totally equivalent to:

```
def second_component(r):
    return r[1]
```

This is mostly useful when you just want to use the function in one spot, and not define it forever.

When sorting a list, you can give it a "key" function to tell it what to sort by.

[33]:
```
L = [-5, 1, 0, 7, -10]
print(L)
L.sort()
print(L)
```

```
[-5, 1, 0, 7, -10]
[-10, -5, 0, 1, 7]
```

[34]:
```
# this sorts numbers according to their absolute
# value, not the number itself
L.sort(key=lambda x : abs(x))
print(L)
```

```
[0, 1, -5, 7, -10]
```

[ ]:
```
L
```

[ ]:
```

```

[36]:
```
L = [(0, 3), (-1, 7), (2, 5)]
print(L)
```

```
[(0, 3), (-1, 7), (2, 5)]
```

[37]:
```
sorted(L)
```

[37]:
```
[(-1, 7), (0, 3), (2, 5)]
```

[ ]:
```
L
```

[38]:
```
sorted(L, key=lambda x : x[1])
```

[38]:
```
[(0, 3), (2, 5), (-1, 7)]
```

[40]:
```
# other version
def second_component(x):
    return x[1]
sorted(L, key=second_component)
```

[40]:
```
[(0, 3), (2, 5), (-1, 7)]
```

[41]:
```
# when you do L.sort(), it sorts the variable in place
# when you do sorted(L), it leaves L alone but returns
#   a new list that has been sorted
L = [-3, 4, 0]
R = sorted(L)
print(L)
print(R)
```

```
[-3, 4, 0]
[-3, 0, 4]
```

[42]:
```
L = [-3, 4, 0]
L.sort()
print(L)
```

```
[-3, 0, 4]
```

[ ]:
```

```