Discussion Section # 3 : Functions

Mitchell Valdes

Introduction

files needed = None

This week we are working on

1. Keyword and default arguments in functions
2. General Python practiced

More on functions

- Writing functions is an important part of programming. You write the code once, debug it and test it, and then use it whenever you need to.
- We will also see situations later where we can extend the use of a pandas function by giving it functions of our own to apply to DataFrames. [More on that in a few weeks.]
- Let's dig a bit deeper into functions.

Argument types

A function can take inputs, which are called *arguments* in python and produce outputs. In class, we covered positional arguments. *Positional arguments* have to be passed to the function in the correct order.

```
def divider(numerator, denominator):
    """
    Divides numerator by denominator.
    """
    return numerator/denominator
```

Here numerator = 5 and denominator = 10

```
divider(5, 10)
```

```
0.5
```

Keyword arguments

A disadvantage of positional arguments is that you have to remember what order to pass the values to the function. For simple functions, this is not much to ask.

We can also call a function using keyword arguments. *Keyword arguments* pass the name of the argument and the argument's values.

We can call the function divider using keyword arguments.

```
divider(numerator = 5, denominator = 10)
```

```
0.5
```

is equivalent to

```
divider(denominator = 10, numerator = 5)
```

```
0.5
```

Keyword arguments

When using keyword arguments, the ordering of the arguments does not matter.

```
# print() has a keyword argument 'sep' that sets the separating character.
print('The econ department phone number is:')
```

```
The econ department phone number is:
```

```
print('608', '263', '2989', sep='-')
```

```
608-263-2989
```

- In the last line, the print function took 3 positional arguments and one keyword argument. We can set the `sep` argument to be any character.
- If you wanted each argument printed on one line, you would set it to `\n`.

Keyword arguments

Worth noting is that Python prefers that positional arguments come before any keyword arguments.

```
print(divider(numerator = 5, 10))
```

```
positional argument follows keyword argument (<string>, line 1)
```

```
print('The econ department phone number is:')
print('608', '263',sep='-', '2989')
```

```
positional argument follows keyword argument (<string>, line 2)
```

Default arguments

When writing a function, you may define default values for your arguments.

```
def weight_converter(weight, how='kgtolb'):
    """
    Converts weights to different units. Currently supports pounds to kilograms
    and kilograms to pounds.
    """
    if how == 'kgtolb':
        return weight*2.205
    elif how == 'lbtokg':
        return weight/2.205
    else:
        print('I do not know how to convert that.')
```

```
# If I do not pass the `how` argument, I get kilos to lbs.
weight_converter(1.0)
```

```
2.205
```

```
weight_converter(1.0, how='lbtokg')
```

```
0.4535147392290249
```

Practice

Question 1

What is the default value of `sep` in the `print()` function? What does print's `end` argument do?

Question 2

Does the following code work?

```
print(sep='-', '608', '263', '2989')
```

What does this tell us about how we can mix positional and keyword arguments?

Question 3

Write a function named `employee_summary` that takes three arguments: a dictionary of data, a name, and an argument called `extended`. Make the default value of `extended` be False. The other two arguments do not need default values.

If the name is not in the dict, then the function should return a message stating so.

If the name is in the dict and `extended==False`, then the function should print the employee's name and department. Example: 'Employee Kim is in economics.'

If the name is in the dict and `extended==True`, then the function should print the employee's name, position, and department. Example: 'Employee Kim is a prof in economics.'

Notice that the first letter of the person's name in the output has been capitalized.

Here is the data dictionary:

```
data = {'kim':['prof', 'economics'], 'greg':['TA', 'economics'],
        'bucky':['mascot', 'sports'], 'barry':['coach', 'the football team'],
        'abe':['president', 'USA'], 'hector':['prof', 'biochemistry']}
```

```
# Question 3

def employee_summary(data_dict, name, extended=False):
    # Your code here
    pass
```

Question 4

Test your function with:

```
employee_summary(data, 'greg', extended=True)
employee_summary(data, 'goldy')
employee_summary(data, 'barry', extended=False)
```

```
# Question 4

data = {'kim':['prof', 'economics'], 'greg':['TA', 'economics'],
        'bucky':['mascot', 'sports'], 'barry':['coach', 'the football team'],
        'abe':['president', 'USA'], 'hector':['prof', 'biochemistry']}

employee_summary(data, 'greg', extended=True)
employee_summary(data, 'goldy')
employee_summary(data, 'barry', extended=False)
```