# Practicum 6 - Functions!

**DS2001 - Computer Science Practicum**
**Fall 2021**
**October 12-13, 2021**
**Practicum Deadline:** October 26th, 2021 at 12:00pm Boston time

**What we're practicing today:**

- functions
- functions
- functions

**Handouts:**

- Writing Functions (from DS 2000)
- Functions + Scope (from DS 2000)
- Functions + Testing (from DS 2000)

**What to do if you miss practicum this week:**

- Fill out the "I'm missing class" form ASAP
- Follow up in office hours with Felix (find office hours links on the course website). **Note that this is required.**
- Come to practicum next week if you are well again. **Do not come to class if you are not feeling well.**

**How far we expect you to get this week:**

- We expect you to complete **Task 2 and attempt Task 3**, which means writing the `pretty_print` function with the correct parameters and attempting to loop through the passed in lists.

Everyone should create **one** file for the whole practicum called **p6.py**.

We have a few problems to tackle today; get through as many as you can, and submit whatever you have done at the end of your section. Each question in this write-up will describe a problem to work on, then might ask you a question that you'll answer with your group.

At the end, everyone should make sure to turn in your code to gradescope **with your group's answers to the questions written as comments in your file!**

**Set-up: background of your data**

In this week's practicum we will be asking Python to process administrative data for us. Specifically, we will be looking at some demographic data collected in 1920. We'll be working with a manageable-sized subset of data, but keep an eye out for pieces of code that you could use on larger datasets!

**Demographics and Tax Collection in Pahrump, Nevada 1920**

Many of the basic tasks of public administration, such as counting people and calculating taxes, involve collecting and summarizing data. Collecting accurate administrative data is an important component of conducting accurate analyses, which allow the powers that be to make informed decisions (like funding public education and other important public services). Censuses are a specific kind of administrative data collection, where every effort is made to collect the same basic information from every resident of a region. Censuses have been conducted since ancient times, notably in Babylon, Persia, and Rome.

The U.S., in fact, conducted a census last year. Talk to your family, friends, and roommates about whether or not they filled out the census. Were you counted? Why or why not? Even though college students make up a large portion of the population, they often prove difficult to count during a census.

By law, US Census individual records (ie. "microdata") are kept behind lock and key for 72 years. After this they become available for demographic, genealogical, and other study. We will use 1920 US Census data to characterize the population of Pahrump, Nevada (2010 population: 36,441) at that point in time.

**Task 0: Write your file comment, the skeleton of your `main()` and import the `p6_utils` module**
First, write down your name in a comment at the top of your file and a small description of what the file is, like we did last time. This is a good habit to get into for all your programs!

Next, import the `p6_utils` module so that you can access the functions in it and write a `main()` function that just prints out "p6".

**Task 1: Read in the example data**
Using the `p6_utils.file_to_list()` function, read in the data from `names.txt` and `jobs.txt` to new lists. Once you've read them in, print out the first three elements of each list. Go ahead and do this in your `main()` function.

Example output:
```
['Frank L. Cate', 'David P. Buchanan', 'Orson T. Johnson']
['MINER', 'MINER', 'MINER']
```

**Task 2: Verify list length**
Write a function with the following signature.
```
"""
name: same_length
This function takes two lists and verifies that they have the same number of elements.
parameters:
lst1 - first list
lst2 - second list
return:
True if both lists have the same length, False otherwise
"""
```

Use this function to verify that your two lists have the same number of elements. In your main, print out a message if they do have the same length, otherwise print out an error message and tell the user the length of the two lists. If the two lists have different lengths, no other code in your program should run.

Example output:
```
congrats, they match in length!
```

or
```
:( they don't match! List 1 has 94 elements and list 2 has 100 elements.
```

**Group Question:** What might the lists passed to this function contain? (string? ints? floats? lists?) Can you think of another datatype (other than list) that you could pass as parameters to this function and have it still work?

**Task 3: Pretty print the lists**
Write a function with the following signature.

```
"""
name: pretty_print
This function takes two lists and prints out the elements in the following format:
lst1_value1: lst2_value1
lst1_value2: lst2_value2
lst1_value3: lst2_value3
...
For all items in the list
parameters:
lst1 - first list
lst2 - second list
return:
nothing
"""
```

Use this function to print out the names paired with the jobs that these people held in 1920.

Example output:

```
Frank L. Cate: MINER
David P. Buchanan: MINER
Orson T. Johnson: MINER
Edna Johnson: NONE
...
```

**Group Question:** What value does a function return if it has no return statement? Test this out by running the following code. What sort of errors do you think it might lead to in your if you save the return value of a function that has no return?

Code to run:

```
mystery = print("print functions don't return a value")
print(mystery)
print(type(mystery)
```

**Task 4: Look-up by name**
Write a function with the following signature.

```
"""
name: lookup_by_name
This function takes a list of names and a string and returns a new list of all the names
that begin with the supplied string. Case insensitive.
parameters:
names - list of names
begins_with - string to search for
return:
a new list of names from the original names list that begin with the begins_with string
"""
```

Hint: you may find the python string functions `upper()`, `lower()` and/or `startswith()` useful!

Test out your function by writing appropriate code in your main to look up people by name. Make sure to ensure that your function is case insensitive!

Example output:
```
Names beginning with 'A'
['Armando Santa Cruz', 'Alfred Russell', 'Annie Pat', 'Annie Morise', 'Ada R. Andrews',
'Annie Lee', 'Anna T. Chuble', 'Annie Cowboy']
Names beginning with 'ANN'
['Annie Pat', 'Annie Morise', 'Annie Lee', 'Anna T. Chuble', 'Annie Cowboy']
```

**Group Question:** What would happen if you ran this code, but passed in the list of jobs instead of a list of names? Test out your hypothesis and tell us if you were right or not!

**Task 5: Count matching elements**
Write a function with the following signature.
```
"""
name: count_matches
This function takes a list of values and a target value and counts the number of times
the target value appears in the list of values.
parameters:
ls - list of elements
to_count - value to search the list for
return:
integer number of times to_count appears in ls
"""
```

Test out your function by writing appropriate code in your main. Test your code on both the names list and the jobs list.

Example output:
```
jobs that are MINER: 8
```

**Finished early? Do the task below....**

**Task 6: Compare your implementation of count_matches to the python built-in list count function**
This task takes a number of steps! Our goal is to test how efficient your count_matches function is versus python's built-in list `count()` function.

You can call the list count() function like:
```
ls = [1, 1, 1, 2, 4, 1]
num_matches = ls.count(1)
print(num_matches)  # 4
```

First, we'll need to implement the following function, since we'll probably only be able to perceive differences between these functions if we have **REALLY BIG LISTS**.

```
"""
name: generate_list
This function takes an integer n and generates a list with n integers in it that have
values 1 and 100.
parameters:
n - int number of values requested
return:
list containing n integers between 1 and 100
"""
```

Next, you'll need to call your function to generate a really big list. How about one with 10000 elements? Do this.

Now, call your `count_matches` function, print out the result, then call the built-in list count function and print out the result. Do they match? (they should) Did you notice one taking longer than the other?

Finally, since we like things to be exact, we're going to measure exactly how much time each function takes to run. To do this, start by importing the time module at the top of your file. To measure how long any piece of code takes to run, we can use the following pattern:

```
import time  # with the other imports, at the top of your file


begin = time.time()  # get current time
result = ls.count(1)  # code that we're measuring
end = time.time()  # get the current time
print("time elapsed:", end - begin)  # see how many seconds elapsed
```

Play around with your code! At what point does your implementation start taking longer than the built-in version?