Problem Set 1: Intro to Python

This problem set is meant to create a good foundation of programming.

Expectations

We recognize that this may be your first experience with programming. As such, we'll be grading primarily on whether your code works as expected. We'll generally emphasize code functionality and readability over performance. However, we expect your code to be non-repetitive, relatively clean, and efficient enough to deal with medium-to-large datasets. As we move forward with the course, you will be introduced to programming techniques, iterators, and objects that will allow to work with data in relatively efficient ways.

Give credit where credit is due

Throughout the course you will often find yourself taking to the Internet for help. There's no shame in Googling! In fact, any honest programmer will tell you that, no matter your skill level, you'll still find yourself consulting Stack Overflow on a regular basis.

However, if you are using other people's code, make sure to credit the source. Of course, this doesn't necessarily hold true if you're searching for help on something like constructing a basic for loop. But when you're taking large chunks of someone else's code and modifying them for your own purposes, you should always credit the source. You can think of this like citation in an academic paper; it's not ethical to take credit for other's work!

For example, say you're adopting someone else's function. It would be appropriate to write something like:

```
# Code adopted from A Cool Project, authored by a cool person, available at https://github.
def function_i_didn't_write(arg1, arg2):
  result = do_something(arg1, arg2)
  return result
```

Lists

1. Create a list containing any 4 strings.

```
string_list = ["item1", "item2", "item3", "item4"]
```

2. Print the 3rd item in the list - remember how Python indexes lists!

```
string_list[2]
```

3. Print the 1st and 2nd item in the list using [:] index slicing.

```
string_list[0:2]
```

4. Add a new string with text "last" to the end of the list and print the list.

```
string_list.append("last")
print(string_list)
```

5. Get the list length and print it.

```
list_length = len(string_list)
print(list_length)
```

6. Replace the last item in the list with the string "new" and print

```
# Using reverse indexing...
string_list[-1] = "new"
# Slightly less useful---what if the list length changes?---but still correct.
string_list[4] = "new"
print(string_list)
```

Strings

Given the following list of words stored as strings, complete the following tasks:

```
sentence_words = ['I', 'am', 'learning', 'Python', 'to', 'munge', 'large', 'datasets', 'and'
```

1. Convert the list into a normal sentence with `join()`, then print.

```
# Solution
sentence = ' '.join(sentence_words)
print(sentence)
```

2. Reverse the order of this list using the `.reverse()` method, then print. Your output should begin with `["them", "visualize", ... ]`.

```
# Solution
sentence_words.reverse()
print(sentence_words)
```

3. Now user the `.sort()` method to sort the list using the default sort order.

```
# Solution
sentence_words.sort()
print(sentence_words)
```

4. Perform the same operation using the `sorted()` function. Provide a brief description of how the `sorted()` function differs from the `.sort()` method.

```
# Solution
sentence_sorted = sorted(sentence_words)
print(sentence_sorted)
```

5. Extra Credit: Modify the sort to do a case case-insensitive alphabetical sort.

```
# Solution
sentence_words.sort(key = lambda s: s.lower())
print(sentence_words)
```

Random Function

Here is a Python snippet that generates a random integer:

```
from random import randint
# this returns random integer: 100 <= number <= 1000
num = randint(100, 1000)
```

Implement your own random function that builds on this python function, returning an integer between a low and a high number supplied by the user, but that can be called with the low number optional (default to 0).

Test your function by adding the following 2 assert statements to your file (replace myrandom and low with the names you used). The `assert` statement helps you debug, by testing if a statement is true.

- Inputs

1. Two `integers` that will be used as lower and upper bounds of the function. If the user does not pass a lower bound, the default value should be zero.

- Outputs

1. A random number within the established bounds

```
assert(0 <= my_random(100) <= 100)
assert(50 <= my_random(100, low = 50) <= 100)
```

```
# Solution
def rand_number(max, min = 0):
  return randint(min, max)

assert(0 <= rand_number(100) <= 100)
assert(50 <= rand_number(100, min = 50) <= 100)
```

String Formatting Function

Write a function that expects two inputs. The first is a title that may be multiple words, the second is a number. Given these inputs, print the following string (replacing `n` and `title` with dynamic values passed to the script).

```
The number n bestseller today is: title
```

- Inputs

1. An integer representing the position on the bestseller list and a string representing the a title. If not already titlecased, the function should titlecase the string.

- Outputs

1. Print a string of the format: `The number n bestseller today is: title`. You should use an `f` string or the `.format()` method to format the printed string.

```
def bestseller(t, rank):
  t = t.title()
  # f string
  msg = f"The number {rank} bestseller today is: {t}"
  # .format() method
  # msg = "The number {} bestseller today is: {}".format(rank, t)
  print(msg)

bestseller("title is a title", 4)
```

Password Validation Function

Write a password validation function. Ask the user to input a password that meets the criteria listed below. You can either use the Python `input` built-in function, or just pass the password as a function argument. Validate that the user's password matches this criteria. If password is valid, print a helpful success message.

Test that the user's password...

- is 8-14 characters long
- includes at least 2 digits (i.e., numbers)
- includes at least 1 uppercase letter
- includes at least 1 special character from this set: `['!', '?', '@', '#', '$', '%', '^', '&', '*', '(', ')', '-', '_', '+', '=']`

- Inputs

1. A `string` that will be tested for the password requirements. The string can be passed as an argument to the function, or as an input through the `input` function.

- Outputs

1. A success message if the password works, an error message if it fails.

```
# Solution
def pass_valid():
  attempt = input('password: ')
  sp_list = ['!', '?', '@', '#', '$', '%', '^', '&', '*', '(', ')', '-', '_', '+', '=']

  length = 8 <= len(attempt) <= 14
  digit = sum(i.isdigit() for i in attempt) >= 2
  upper = sum(i.isupper() for i in attempt) >= 1
  sp = sum(attempt.count(i) for i in sp_list) >= 1

  if length & digit & upper & sp:
    print("Success!")
  else:
    print("Your password does not meet our RIGOROUS standards.")
```

Exponentiation Function

Create a function called `exp` that accepts two integers and then `return`s an exponentiation, **without using the exponentiation operator** (`**`). You may assume these are positive integers. Use at least one custom-defined function.

For example, some outputs of this function could be:

```
exp(2, 3) # output: 8
exp(5, 4) # output: 625
```

- Inputs

1. An `integer` that will be recursively multiplied
2. An `integer` that will define the number of times to multiply the number to get the exponentiation.

- Outputs

1. An `integer` that is the result of the exponentiation.

Hint: You can recursively multiply a number. The second function parameter defines the number of times the recursive loop happens. Every time the loop happens, you can redefine the variable that gets multiplied.

```
# Solution (for loop)
def exp(x, y):
  ex = x
  for _ in range(1, y):
    ex *= x
  return ex

# Solution (while loop)
def exp(x, y):
```

```
  ex = x
  i = 1
  while i < y:
    ex *= x
    i += 1
  return ex
```

Extra Credit: Min and Max Functions

Write your own versions of the Python built-in functions `min()` and `max()`. They should take a list as an argument and return the minimum or maximum element. Assume lists contain numeric items only.

- Inputs:

1. A `list` of `numbers` to be tested.

- Outputs:

1. A `number` of the list that is the maximum or minimum.

Hint: Pick the first element as the minimum/maximum and then loop through the elements. Each time you find a smaller/larger element, update your minimum/maximum.

```
# Solution
def min(num_list):
  m = num_list[0]
  for i in num_list:
    if i < m:
      m = i
    else:
      continue
  return m

def max(num_list):
  m = num_list[0]
  for in num_list:
    if i > m:
      m = i
    else:
      continue
  return m
```