Balchunas

AP CSP: Problem Set 1

Specification

Submit a .py file named `your_name_pset1.py`. Your file should **NOT** execute any functions. You are just filling them in. When I run your program there should be no errors and no output. You may choose to test your functions as you go, but you may delete the function calls after you test or comment them out. Comment your problems with the problem number above the function header. Remember to import the random module to use the library's functions.

Problem 1

Write a function that returns the number of "e" characters present in a given string. You should count both "E" and "e".

```
count_e(string)
count_e("abced")  # 1
count_e("aeeeEe") # 5
```

Problem 2

Python provides a string method (procedure) called `endswith()` for checking if a string ends with a particular substring. Given a particular string and substring return `True` if the string ends exactly with the substring, else return `False`. Do not use the built-in `endswith()` method. The goal of this problem is to use slicing and see how the Python writers actually created the method.

```
ends_with(string, substring)
ends_with("hello.txt", ".txt") # True
ends_with("andrew", "and")     # False
```

Problem 3

Write a function that given a string returns a reversed version of that string. Do **NOT** use slicing to do this and do not use any built in functions to reverse the string. Begin with an empty string and build the reversed string by traversing the original string. You must use a loop to recieve full credit.

```
reverse_string(string)
reverse_string("abcd")      # "dcba"
reverse_string("eeab beea") # "aeeb baee"
```

Problem 4

Use a loop to calculate the sum of all numbers between two integers, inclusive. Do **NOT** use the `sum()` function. You must use a loop to recieve full credit.

```
sum_between(a, b)
sum_between(3, 1000) # 500497
sum_between(4, 98)   # 4845
```

Problem 5

Write a function that searches a string for the occurrence of a substring. Return the lowest index where the first instance of the substring begins, or -1 if the substring is not found. Do **NOT** use the `find()` method or any built-in Python functions that do this for you. You must use a loop and slicing to recieve full credit.

```
index_of(string, substring)
index_of("hello", "el")      # 1
index_of("hellohello", "lo") # 3
index_of("abc", "xx")        # -1
```

Problem 6

Write a function that returns the length of the longest consecutive substring of similar characters in a given string. Note that there will be no capital letters. You may assume the input has no spaces.

```
count_long_char(string)
count_long_char("abbbdeef") # 3 (bbb)
count_long_char("abcbdefg") # 1
```

Problem 7

We learned in CSP that a good password is long, contains capitals, lowercases, numbers, and special characters. Write a function `check_password()` that takes a string and checks to see if the password is at least 10 characters long, if it contains a mixture of capitals and lowercases, as well as at least one special character. You may not use built-in string methods for this problem. Hint: use the `ord()` function to get the numerical ASCII representation of a character. Return `True` if the password is a good password according to these rules, and `False` otherwise.

```
is_good_password(password)
is_good_password("C$PiSfun!32")          # True
is_good_password("hunter2")              # False
is_good_password("longpasswordbutnotok") # False
```

Problem 8

In mathematics, a harshad number is an integer that is evenly divisible by the sum of its digits. For example, 18 is a harshad number because $1 + 8 = 9$ and 18 is divisible by 9. Write a function that returns the number of harshad numbers between 1 and some input (inclusive).

```
harshad_numbers(n)
harshad_numbers(100)     # 33
harshad_numbers(1000000) # 95428
```

Problem 9

Given a number, return the number of digits the number has. Do **NOT** convert the number to a string or use any `len()` functions to achieve this. Use a loop and a counting variable.

```
count_digits(n)
count_digits(444)  # 3
count_digits(1234) # 4
```

Problem 10

Write a function that takes a string and returns the same string in all uppercase characters. To get full credit you must use the `ord()` and `chr()` functions and may not use any string methods. You must also use a loop and a string building algorithm.

```
upper_string(string)
upper_string("the brown fox") # THE BROWN FOX
upper_string("Go pIraTes")    # GO PIRATES
```

Problem 11

A Caesar cipher is a type of symmetric encryption that offsets each character by a specific number. For example, if each character is offset by 23 then `THE QUICK BROWN FOX` becomes `QEB NRFZH YOLTK CLU`. Given an input string and an offset number, return the encrypted string as all uppercase. You may have to convert the input string to be all uppercase.

```
caesar_encrypt(string, n)
caesar_encrypt("the quick brown fox", 23) # QEB NRFZH YOLTK CLU
```

Problem 12

Given a string encrypted with a caesar ciper, and a key, return the decrypted string. Hint: The `ord()` function converts a string to an ascii code, and the `chr()` function converts an ascii code to a string. Think about how you might use this function combined with a loop to decrypt ANY message encrpyted with a caesar cipher.

```
caesar_decrypt(string, n)
caesar_decrypt("QEB NRFZH YOLTK CLU", 23) # THE QUICK BROWN FOX
caesar_decrypt("QEB NRFZH YOLTK CLU", 13) # DRO AESMU LBYGX PYH
```

Problem 13

Write a function that takes two parameters and returns the value of x^a. Do **NOT** use the built in power operator from Python. Use a loop to find this value.

```
my_pow(x, a)
my_pow(2, 3) # 8
my_pow(4, 3) # 64
```

Problem 14

A factorial of a number `n` is represented as $n!$ is useful in calculating probabilities and combinations. The formula for $n!$ is $n! = (n)(n-1)(n-2)\dots(1)$. Also, $0! = 1$ by definition. Write a function that returns the factorial of a number. Do **NOT** use any built in Python functions that accomplish this. To recieve full credit you must use looping.

```
my_factorial(n)
my_factorial(5) # 120
my_factorial(3) # 6
```

Problem 15

A \textit{double factorial} is symbolized by two exclamation marks. The double factorial of an integer n is represented as $n!!$ and is equal to $(n)(n-2)(n-4)\dots$ ending with a multiplication of 1 for odd values of n or 2 for even values of n. The value of $0!!$ is 1 and $-1!!$ is also 1. It is undefined for $n < -1$, but you need not consider this. Write a function that takes a parameter and returns $n!!$. You must use looping to recieve full credit.

```
double_factorial(n)
double_factorial(5)  # 15
double_factorial(10) # 3840
```

Problem 16

A palindrome is a string that is the same forwards and backwards. Given a string, write a function that returns `True` if the string is a palindrome and `False` otherwise. Do not use any built-in Python procedures that reverse a string.

```
is_palindrome(string)
is_palindrome("tacocat") # True
is_palindrome("anna")    # True
is_palindrome("APCSP")   # False
```

Problem 17

In the first assignment you used the built in square root function provided by the programming language. Not only did Newton discover a majority of classical mechanics, and arguably calculus, he also discovered a way to find the square root of a number. *Newton's method* as it's called works for positive values of x in the following way. Start with an estimate t equal to x. If t^2 is close enough (for your purposes) to x, then t is a square root of x. If it is not within the threshold, we replace t with the average of t and x/t (e.g. $\frac{t+\frac{x}{t}}{2}$). Each time we perform the update, we get closer to the desired answer. Write a function that calculates the square root of a number x. Use a threshold of $\pm 10^{-15}$.

```
my_sqrt(x)
my_sqrt(2)  # 1.414213562373095
my_sqrt(5)  # 2.23606797749979
my_sqrt(16) # 4.0
```

Problem 18

Most text editors come with a find and replace feature where you can replace all instances of a found string with a new string. Write a function that takes three parameters, a string, a target, and a replacement, that will return a new string where all instances of the target in the string have been replaced by the replacement. Do **NOT** use built in Python functions that do this for you.

```
find_and_replace(string, target, replacement)
find_and_replace("The dog", "dog", "cat") # "The cat"
find_and_replace("Firefox", "fox", "cow") # "Firecow"
find_and_replace("The pirate and pirate", "pirate", "astronaut") # The astronaut
and astronaut
```

Problem 19

The Fibonacci numbers are defined by the sequence: $f_1 = 1, f_2 = 1, f_n = f_{n-1} + f_{n-2}$. The first few are $1, 1, 2, 3, 5, 8, 13, \ldots$ Write a function that returns the n-th Fibonacci number. Do **NOT** use a built in Python function to accomplish this. Use a loop in your solution. Your code may not finish in a reasonable amount of time for large Fibonacci numbers. This is okay.

```
fib(n)
fib(5) # 5
fib(9) # 34
```

Problem 20

Write a function that takes a string and returns a compressed version of the string using run-length encoding. The compressed string will consist of each character followed by the number of conecutive times it occurs.

```
run_length_encode(string)
run_length_encode("aaabbcddddd") # a3b2c1d5
run_length_encode("12111122233") # 1121142332
```

Problem 21

Write a function that takes a run-length encoded string and returns the original uncompressed string. Run-length encoding is described in Problem 20.

```
run_length_decode(string)
run_length_decode("a3b2c1d5")   # aaabbcddddd
run_length_decode("1121142332") # 12111122233
```

Problem 22

Write a function that returns `True` if both inputs are anagrams and `False` otherwise. An anagram is a word or phrase formed by rearranging the letters of a different word or phrase.

```
are_anagram(string1, string2)
are_anagrams("listen", "silent") # True
are_anagrams("listen", "hear")   # False
```

Problem 23

Write a function that removes all duplicate characters from a given string and returns the result. The order of characters should be maintained.

```
remove_duplicates(string)
remove_duplicates("aabccdee") # abcde
remove_duplicates("abcdecad") # abcde
```

Problem 24

The exponential function e^x is packaged nicely inside of the math module for you to use. But have you thought about how it is actually represented inside of the computer? It uses an approximation of e^x as an infinite sum of x to various powers. The approximation for e^x is as follows:
$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$ forever and ever. The neat math way of writing this is $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ where Σ represents a "summation" in math. It works very much like a loop in computer science where all the terms are added together.

Write a function that takes a parameter x and returns the value of e^x. The exit condition for the while loop you'll use here is not obvious so we must think about how numbers are represented inside of a computer. You may stop adding when the term $\frac{x^n}{n!}$ is *small enough*. Let's say 10^{-15}.

```
my_exp(x)
my_exp(5)  # 148.41...
my_exp(10) # 22026.47...
```

Problem 25

Write a function that takes an integer n and returns a filled and hollow square (using asterisks) placed next to each other of length n and width n.

```
make_squares(5)

***** *****
***** *   *
***** *   *
***** *   *
***** *****
```

Problem 26

Write a function that takes an integer n and returns string representing a hollow diamond with side length n.

```
make_hollow_diamond(3)

  *
 * *
*   *
 * *
  *
```

Problem 27

Write a function that takes an integer n and returns a string representing a diamond with side length n.

```
make_diamond(3)

  *
 ***
*****
 ***
  *
```

Monte Carlo Problems

A Monte Carlo problem is one that simulates a probabilistic event. We can get statistics by repeating experiments and seeing what happens.

Problem MC1

Given a coin with p proability of getting heads, and n flips, find the probability there are k consecutive heads or tails.

```
k_consecutive(p, n, k) # returns a number on [0, 1]
```

Problem MC2

A bag contains n coins. g of the n coins are fair coins, and $n - g$ of n coins are weighted to produce heads p percent of the time. You pick a coin at random and perform n_flips flips. Exactly n_heads of the n_flips are heads. Given the parameters n, g, p, n_heads, and n_flips determine the probability that you picked an unfair coin.

```
p_unfair_coin(n, g, p, n_heads, n_flips) # returns a number on [0, 1]
```

Problem MC3

In a deck of n cards, there are g good ones. Write a simulation that determines the probability that by $turn$ random draws without replacement that you have selected all g of them.

```
draw_all_good_cards(n, g, turn) # returns a number on [0, 1]
```