WELCOME BACK TO WEEK 6 OF INTRO TO PYTHON

LET’S DO A REFRESHER

Create a program that asks the user to guess a number that is randomly generated. You will prompt the user that their guess is either too high, too low, or correct. You must also use a flag to control a while loop that runs while the user guess is incorrect and ends the program when they guess the correct value.

NOTE* a flag is a Boolean variable set to false until a specific circumstance triggers it. For this exercise, the flag is triggered by the user guessing the correct value.

WAYS TO MANIPULATE STRINGS

- Once we have a string stored in a variable, there are a number of ways to access parts of the string, check the string for letters, or manipulate the string
- To check if a character or set of characters is in a string, we use the form:

  <character(s)> in <string>

- This is a Boolean expression that will evaluate to true if the character(s) are in the string and false otherwise

NOTE* This expression is case sensitive! It will not evaluate to true if you are checking for a capital letter that is lowercase in the string.

TRY IT OUT YOURSELF!

Create a program that asks the user for their first name and birthdate and asks them to create a password. If their name or birthday exists within the password, it asks them to create a new one until their password is valid.

NOTE* to turn all letter into lowercase use *variableName.lower()*

Sample Input:

What is your name (all lowercase)? kevin

What is your birthdate? 05/14/01

Create password: keVInSUCKS68!!

Sample Output:

Your password shouldn't contain your name OR birthdate! Please try again!

INDEXING THROUGH A STRING

- Every string stored is considered a list of characters, thus we can also access specific indexes. This can be done using the form:

  variable[#]

- Indexing is 0-based which means that the first character is found at index 0

index	0	1	2	3	4
string	H	E	L	L	O

NOTE* If there is *n* characters in a string, there exists $n - 1$ indexes.

INDEXING THROUGH A STRING CON'T.

- You can also access the string from end to beginning using negative numbers
- This is useful when you want to access a character a certain number of positions from the end of the string

NOTE* This kind of indexing is 1-based, thus the last character of the string is found at -1

index	-5	-4	-3	-2	-1
string	H	E	L	L	O

LET’S PRACTICE INDEXING

Create a program that asks the user to enter a string with only uppercase letters and determine whether or not the string is a palindrome. A palindrome is a string that reads the same forwards and backwards. The key strategy here will be to maintain two indexes: one from the front, counting from 0, and one from the back, counting from -1. We want to see if corresponding characters from the front and back match. If we find a mismatch, we immediately know our string is NOT a palindrome. Using this strategy we can stop halfway through the string.

NOTE* Integer division will be useful in accessing half the string as indexes are only integers.

Sample Input:

Please enter a string, uppercase only: RACECAR

Sample Output:

RACECAR is a palindrome.

SLICING A STRING

- We can access a substring within a string using slicing using the form:

Variable[startingIndex : endingIndex]

NOTE* The starting index is inclusive while the ending index is exclusive!

COMMAND:

print(string[3 : 8])

index	0	1	2	3	4	5	6	7	8
string	U	C	F	R	O	C	K	S	!

OUTPUT:

ROCKS

CREATING LISTS

- Similar to strings, we can have an ordered collection of items stored within a variable with the form:

  variable = [ ]

- While their contents are typically the same type, they don't have to be (i.e. one list can store a string, int, and double)

  randList = ["String", 42 , 3. 5]

- You can access an item in a list using indexing just like with strings!

IMPORTANT FUNCTIONS FOR A STRING

- listN*ame.append(item)* : adds a specified item to the end of the list
- list*Name.remove(item)* : removes a specified item from the list
- *del listName[#]* or *del listName[# : #]* : delete an item or a slice of the list using indexing
- *listName.sort()* : sorts the list according to the natural order or the items (i.e. in numerical or alphabetical order)
- *listName.reverse()* : reverses the list

LET’S PRACTICE!

Create a program to store and update a shopping list for the user. You will give the user a list of options to choose from that includes adding items to the list, removing items from the list, checking if an item is in the list, and ending the program. If the user tries to add an item that already exists in the list or tries to delete an item that does not exist, output an error and do not complete the command.

NOTE* Remember to create functions when necessary (i.e. one function for each command)

LIST FREQUENCIES

- You can also use lists to store the frequency of items

Example:

Imagine we want to store the number of times each letter appears in a message. We will have to create a frequency list of size 26 (since there are 26 letters in the alphabet) and initialize each counter to 0. Then, for each letter in the message, we can simply update the appropriate counter. Since indexing is 0-based, ’A’ will be index 0, ’B’ will be index 1, and so on…

NOTE* Given a letter, the *ord()* function converts the letter to it’s corresponding ASCII value & given a number, the *chr()* function converts an ASCII value to it’s corresponding character

CREATING SETS

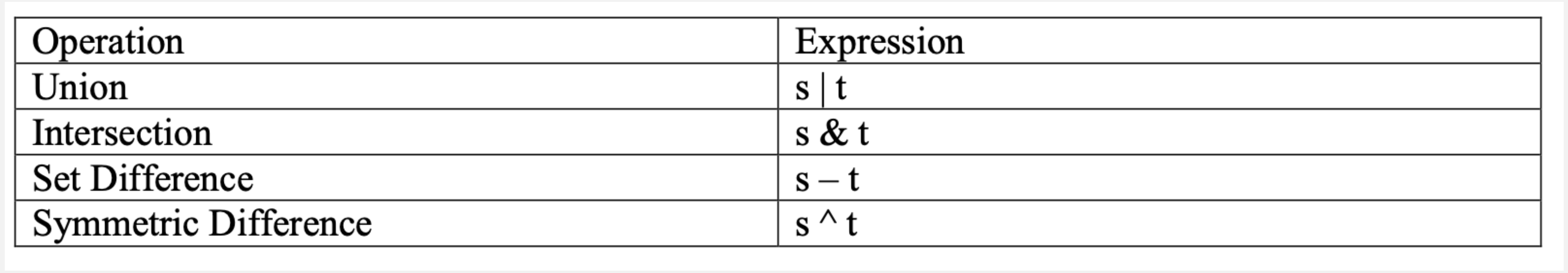
- Sets are similar to lists, except they can store only one copy of each item, they are created using the form:

mySet = set([item1, item2])

- You can add an item to a set using the form:

mySet.add(item)

- These can be useful when merging 2 lists where you want only a single copy of each item



Operation	Expression
Union	s | t
Intersection	s & t
Set Difference	s – t
Symmetric Difference	s ^ t

CREATING DICTIONARIES

- This is a python specific type that simplifies looking up a pair of lists or a list of pairs
- This type is indexed with a key that can be anything type you want, instead of being simply 0-based indexing
- To retrieve an item, you must index it with it’s corresponding key in the form:

variable = { *key* : *value* , *key* : *value* }

- We can add and remove an entry by:

variable[*key*] = *value* and *del variable*[*key*]

- To access a value, you simply call the variable using call it by its key like you would any other indexing:

variable[*key*]

PRACTICE USING DICTIONARIES

Create a simple Phone Book program that allows users to add, remove, update, and retrieve contacts' phone numbers using a dictionary. For this exercise, you will use names as the keys and phone numbers as the corresponding values. Remember to make good use of functions within your program.

LET’S DO SOME MORE PRACTICE

- https://open.kattis.com/problems/classfieldtrip

- Create an account and write a program that is accepted by the problem. You can work in pairs or by yourself to get the accepted status.