Lab 7 – Decisions, Decisions...

The Way of the Program - Fall 2023

---

1. Download and unzip **lab07_files.zip**, then open the file **startcode.py** (already provided for you in the lab07_files folder) in IDLE. Examine the code for the **bounce()** program, and then try it out. What happens?

2. Your goal is to make the ball bounce around inside the window, instead of escaping. Inside the for-loop of **bounce()**, add some code to determine the ball's current *x, y* location in the window. Then, using an appropriate combination of if/else or if/elif/else statements, add some code to reverse the direction of the ball whenever it hits the window border. Next, change the window's height from 400 to 300, so that the window is square. Does your code still work?

3. Python's `string` module includes the string constants `string.punctuation` and `string.digits`, which are string constants containing all of Python's punctuation symbols or digit symbols, respectively. They can be used with the `in` operator to check whether or not a character is of a particular type. For example, after doing `import string`, you could check whether a character `char` is a punctuation symbol by writing:

```
if char in string.punctuation:
   ...
```

   Using an if/elif/else construct, complete the function **charType(*char*)** so that it takes a single character as input and returns one of the following string descriptions: `"punctuation mark"`, `"digit"`, `"vowel"`, or `"consonant"`, as appropriate. For our purposes, we will consider vowels to be just *a*, *e*, *i*, *o*, *u* and *A*, *E*, *I*, *O*, *U* (not *y* or *Y*). Test charType by calling the **classify** function at the Python prompt with different kinds of input characters, as shown below. You should also make sure it passes the autotester by typing `test(charType)`.

```
>>> classify("a")
a is a vowel
>>> classify("b")
b is a consonant
>>> classify("3")
3 is a digit
>>> classify("?")
? is a punctuation mark
>>> classify("y")
y is a consonant
```

4. Below is the function **firstVowel(*word*)**, which takes a string of letters as an input parameter, and returns the leftmost vowel that occurs in the string. As soon as the if-statement becomes true, the return statement immediately terminates the loop and returns the current letter as the final result. If the string has no vowels, the if-statement is never true, so the loop runs to completion and the empty string is returned as the final result:

```
def firstVowel(word):
   for i in range(len(word)):
      letter = word[i]
      if charType(letter) == "vowel":
         return letter
      else:
         pass
   return ""
```

   Add this definition to your code and test it out on several different words. For example, typing `firstVowel("crystalline")` should return `'a'` and `firstVowel("rhythm")` should return `''`.

5. Next, write a function similar to firstVowel called **firstVowelPosition(*word*)** that takes a string of letters as an input parameter and returns the *position number* (counting from 0) of the leftmost vowel in the string, rather than the vowel itself. If the string contains no vowels, `-1` should be returned instead. Some examples are shown below. Make sure your function passes the autotester as well, by typing `test(firstVowelPosition)`.

```
firstVowelPosition("crystalline")  should return 5
firstVowelPosition("apple")  should return 0
firstVowelPosition("syrup")  should return 3
firstVowelPosition("rhythm")  should return -1
```

6. Pig Latin is an invented language formed by transforming each English word according to the following rules:

   - If the word begins with a consonant, you form its Pig Latin equivalent by moving the initial consonant string (that is, all letters up to the first vowel) from the beginning of the word to the end and then adding the suffix *ay*.
   - If the word begins with a vowel, you just add the suffix *way*.
   - If the word contains no vowels, just add *ay*.

   For example, the word *scram* becomes *amscray*, and *apple* becomes *appleway*. Otgay itway? Oodgay!
   Let's develop a program to translate English phrases into their Pig Latin equivalents.

   Using firstVowelPosition as a helper, write a function called **translateWord(*word*)** that takes a single English word as an input parameter and returns its Pig Latin equivalent. Remember that if *w* is a string, then the Python expression *w*[*start* : *end*] will return the letters from position *start* up to but not including position *end*. Be sure to test your function on a variety of different words. Examples:

   `translateWord("scram")` should return `'amscray'`
   `translateWord("apple")` should return `'appleway'`
   `translateWord("rhythm")` should return `'rhythmay'`

7. Next, write a function called **translateLine(*line*)** that takes an entire line of text (*i.e.,* a string containing several words separated by spaces) as an input parameter and returns an equivalent line of Pig Latin as a string. Your function should split the input phrase into a list of the words, then call **translateWord** on each individual word, concatenating the results into the final Pig Latin phrase. Test your function thoroughly. For example:

```
>>> translateLine("the rain in Spain")
'ethay ainray inway ainSpay'

>>> translateLine("i scram with apple pancakes")
'iway amscray ithway appleway ancakespay'
```

8. Finally, write a program called **translateFile(*filename*)** that takes the name of a text file containing English sentences as an input parameter and prints out, line by line, a Pig Latin translation of the entire file, using your **translateLine** function as a helper. You can use the file **haunting.txt** to test your program.

9. (Optional) If you feel up to the challenge, write a new version of translateWord called **translateWord2** that correctly handles capitalized words or words with attached punctuation (warning: this can be a little tricky). Have translateWord2 temporarily uncapitalize and remove punctuation from the word, if necessary, before translating it to Pig Latin, then re-capitalize and re-punctuate the Pig Latin version of the word appropriately. For example, calling `translateWord2("Spain!")` should return `'Ainspay!'`.

10. Write a spell checker program called **spell()** that asks the user for the name of a text file and prints out all of the words in the file that are misspelled. To determine whether a word is misspelled, you can simply check whether it appears in a master list of dictionary words. The file **webster2.txt** (available in the lab07_files folder) contains a list of about 235,000 words that appear in Webster's Second International Dictionary. Your spell checker should also correctly handle capitalized words, such as "House", as well as plural nouns ending in -s and past-tense verbs ending in -ed. For example, "bricks" and "walked" are not in the dictionary, but "brick" and "walk" are. So if a word ends in "s" or "ed" and its root form is in the dictionary, it should not be flagged as misspelled. To test your program, try it on **haunting2.txt** (in lab07_files), which contains a few misspelled words. For example:

```
>>> spell()
Enter filename: haunting2.txt
wals unrecognized
agianst unrecognized
Howse unrecognized
wallked unrecognized
```