Igpay Atinlay

This week's project shows how we can use Python to mess around with words. We will use a *list* to hold a word, and use something new called a *slice* as a way of getting at the individual letters that make up the word.

Pig Latin is a made-up language that has nothing to do with either pigs or with latin.

Igpay atinlay isay aay ademay-upay anguagelay atthay ashay othingnay otay oday ithway eitheray igspay oray ithway atinlay.

Do you get the idea?

It's intended as a spoken language: with a little practise you can translate it as you speak, and "decode" it as you listen.

We don't need a computer to help us, but we're going to use one anyway.

First, we need a clear set of rules to work with. Here's the rules I learned from http://en.wikipedia.org/wiki/Pig_Latin

1. take first consonant or consonant cluster, move it to the end and then add "ay"
2. for words that begin with a vowel, just add "ay" to the end

As usual, we will build up slowly to our final program. Let's get started.

As usual, start by opening the python shell (IDLE) and use `file->new window` and then, in that new window, `file->save as`. Navigate to the code club drive and set the name of your program. Mine is called neal_igpay. Use your own name, so that everyone's program has a different name.

Type in this piece of code.

```
# igpay - accept a line of text from the user and "translate" it into pig latin

text = input("Type something in and press ENTER: ")

words = text.split()

for word in words:
    print(word)
```

Now click `run->run module` (or press F5). When the program runs it should print out a message, inviting you to type something in. Type in a few words and press the ENTER ("Return") key. You should see what you typed in repeated back to you, one word on each line. If you get an error when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again.

After you have run the program, go into the Python shell and look at the values of each of the variables that we created (don't type the ">>>" that is Python prompting you to

type something in).

```
>>> text
>>> words
>>> words[0]
>>> words[1]
>>> words[34]
>>> words[0][0]
>>> words[0][1]
>>> words[0][2]
>>> text[0]
>>> words[0][1:]
```

From this you should start to see what the program is doing. First, it gets a *string* (a sequence of characters, remember; the computer doesn't know or care what they mean) from the user (you). Second, it uses `split()` to break the sequence up into a list of separate words (`split()` treats one or more spaces as a separator between words). In the program, the variable that holds the list of words is called `words`. It doesn't have to be. You can call it anything you like – provided that you always refer to it by the same name. Finally, the program runs a `for` loop and print out each part (each `element`, remember) of the list.

When you typed in those variables to the Python shell, the computer responds by printing out the value of each variable. You should have seen:

- `text` is exactly the line that you typed in.
- `words` is a *list* of strings. Each one is in quotes, they are separated by commas and the whole thing is in square brackets.
- `words[0]` is the first element in the list – the first word
- `words[34]` is the 35th word in your list. As you probably didn't type in as many words as that, Python probably reported an error.
- `words[0][0]` is interesting. It's the first letter of the first word. It shows that each *element* of our *list* is also itself a list. You can do the same with `text`.
- `words[0][1:]` is called a `slice`. The `1` says start from the second letter (the first letter is `words[0][0]`, remember) and the `:` says *go on to the end*.

Go back to the Python shell and get it to print the values for `words[0][:4]` and `words[0][1:2]` and `words[0][99]` and some other combinations to get a feel for how slices work.

Run the program again and type in a sentence with 5 words. Now go back to the Python shell and try `words[1:]` and `words[2:3]` and some other combinations to get a feel for how slices work.

If you are unclear about any of this, ask for help.

Useless fact: two pig latin words are actually in the dictionary as proper real words: ixnay and amscray.

Modify your program to look like the one below. You don't need to type it all in; just the parts that have changed. The new code is shown in red.

When you type the code in, your code will not be coloured like this!

```
# igpay - accept a line of text from the user and "translate" it into pig latin

text = input("Type something in and press ENTER: ")

words = text.split()

for word in words:
    lc_word = word.lower()
    if lc_word.startswith( ('a','e','i','o','u') ):
        print (lc_word + "ay ")
    else:
        print (lc_word + " hmm..")
```

As before, click `run->run module` (or press F5). When the program runs it should ask you (as it did before) to type something in. Type in the words *the rain in Spain* (it's a bit hard for me to explain what's going on in this part if we are all testing the program with *different* words). As before, if you get an error when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again.

The only part of the program that has changed is the part inside the loop: the piece of the program that runs on each word in turn.

First, `word.lower()` makes a copy of the word, with any capital letters changed to small letters (also called *lower-case* letters). The copy is put into a new variable called `lc_word`.

Second, `if` does a *test* on `lc_word` to see whether it starts with a vowel. It if does, the conversion to pig latin is easy: we just print the word and add the letters "ay". Easy. Otherwise (the else: part) we're a bit stuck; we have solved pig latin rule #2 (the easy one) but we haven't done rule #1 yet.

If you are unclear about any of this, ask for help.

Modify your program to look like the one below. You don't need to type it all in; just the parts that have changed. The new code is shown in red.

When you type the code in, your code will not be coloured like this!

```
# igpay - accept a line of text from the user and "translate" it into pig latin

text = input("Type something in and press ENTER: ")

words = text.split()

# if this had [] it would be a LIST. With () it is a TUPLE
vowels = ('a','e','i','o','u')

for word in words:
    lc_word = word.lower()
    if lc_word.startswith( vowels ):
        print (lc_word + "ay ")
    else:
        # simple (works for some words)
        pig_word = lc_word[1:] + lc_word[0] + "ay"
        print (pig_word)
```

As before, click `run->run module` (or press F5). When the program runs it should ask you (as it did before) to type something in. Type in the words *the rain in Spain*. As before, if you get an error when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again.

The correct translation is *ethay ainray inay ainspay*, but we have *hetay ainray inay painsay*. For words that start with a consonant, we generated `pig_word` by moving the first letter to the end and adding "ay". That works for words that have a vowel as the second letter, but it's not good enough as a general-purpose solution. We need a more complex test.

If you are unclear about any of this, ask for help.

Modify your program to look like the one below. You don't need to type it all in; just the parts that have changed. The new code is shown in red.

When you type the code in, your code will not be coloured like this!

```
# igpay - accept a line of text from the user and "translate" it into pig latin

text = input("Type something in and press ENTER: ")

words = text.split()

# if this had [] it would be a LIST. With () it is a TUPLE
vowels = ('a','e','i','o','u')

for word in words:
    lc_word = word.lower()
    if lc_word.startswith( vowels ):
        pig_word = word + "ay "
    else:
        # better. Look for first vowel
        first_vowel = 0
        for letter in lc_word:
            if letter.startswith(vowels):
                # done!
                break
            first_vowel = first_vowel + 1
        pig_word = lc_word[first_vowel:] + lc_word[0:first_vowel] + "ay"
    print(pig_word)
```

As before, click `run->run module` (or press F5). When the program runs it should ask you (as it did before) to type something in. Type in the words *the rain in Spain*. As before, if you get an error when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again. In particular, make sure you have your *indentation* (spaces at the start of the line) correct: each time a line ends with a : some of the following lines are grouped together by being *indented* by an additional 4 spaces. When the group ends, we *outdent* (have 4 fewer spaces).

The change here is that the program uses a loop to step through the word, letter by letter, looking for the first vowel. When it finds one, it uses `break` to stop the loop early, and leaves the variable `first_vowel` set to the `index` of the first vowel within the word. Using `first_vowel` it's easy to form our pig word: we start with all the letters from the first vowel to the end, then add all the letters from the beginning up to (but not including) the first vowel, and finally we add "ay" on the end.

Look back at the correct translation for our sentence: you should see that the latest version of the program is working correctly.

Finally, a bit of tidying up. Lets get the whole thing printed on a single line. Do you remember how to do that from before? Just change one line (add the part in red).

```
    print(pig_word, end="")
```

As before, click `run->run module` (or press F5) and try it out.

If you are unclear about any of this, ask for help. Otherwise, read on for some extras...

Extras

1. Modify the program to get rid of `lc_word`

Hint: words = text.lower().split()

You can test this out in the Python shell:

```
>>> fred = "Hello friends and WELCOME"
>>> foo = fred.lower().split()
>>> foo
```

What happens if you swap the `lower` and the `split`?

To go with `lower()` there is also `upper()` and `swapcase()` and `capitalize()` (Although Python was written by a Dutch man, he chose to use American spellings rather than those of his nearer neighbour..)

2. Now get rid of `text`, too.

Hint: go straight to the list `words` all in one line.

3. Move the main part of the program into a procedure (defined with `def`) so that the program looks like this:

```
# igpay - accept a line of text from the user and "translate" it into pig latin

text = input('Type something in and press ENTER: ')

# if this had [] it would be a LIST. With () it is a TUPLE
vowels = ('a','e','i','o','u')

def to_pig(text):
    # your code goes here

exttay = to_pig(text)
print(exttay)
```

4. From the Python shell try the Help->Python Docs menu item. If it works, use the "Quick search" box to search for `swapcase`. In the search results, click on `str.swapcase` to see a description of what it does. Scroll up and down to see some of the other things that you can do with strings.

5. Can you write a procedure `from_pig` to translate a sentence from pig latin back to English?

Hint: have a think about some of the examples before you start coding!